African Virtual University

Applied Computer Science: CSI 3101

INTRODUCTION TO OPERATION SYSTEMS

William Korir

Foreword

The African Virtual University (AVU) is proud to participate in increasing access to education in African countries through the production of quality learning materials. We are also proud to contribute to global knowledge as our Open Educational Resources are mostly accessed from outside the African continent.

This module was developed as part of a diploma and degree program in Applied Computer Science, in collaboration with 18 African partner institutions from 16 countries. A total of 156 modules were developed or translated to ensure availability in English, French and Portuguese. These modules have also been made available as open education resources (OER) on oer.avu.org.

On behalf of the African Virtual University and our patron, our partner institutions, the African Development Bank, I invite you to use this module in your institution, for your own education, to share it as widely as possible and to participate actively in the AVU communities of practice of your interest. We are committed to be on the frontline of developing and sharing Open Educational Resources.

The African Virtual University (AVU) is a Pan African Intergovernmental Organization established by charter with the mandate of significantly increasing access to quality higher education and training through the innovative use of information communication technologies. A Charter, establishing the AVU as an Intergovernmental Organization, has been signed so far by nineteen (19) African Governments - Kenya, Senegal, Mauritania, Mali, Cote d'Ivoire, Tanzania, Mozambique, Democratic Republic of Congo, Benin, Ghana, Republic of Guinea, Burkina Faso, Niger, South Sudan, Sudan, The Gambia, Guinea-Bissau, Ethiopia and Cape Verde.

The following institutions participated in the Applied Computer Science Program: (1) Université d'Abomey Calavi in Benin; (2) Université de Ougagadougou in Burkina Faso; (3) Université Lumière de Bujumbura in Burundi; (4) Université de Douala in Cameroon; (5) Université de Nouakchott in Mauritania; (6) Université Gaston Berger in Senegal; (7) Université des Sciences, des Techniques et Technologies de Bamako in Mali (8) Ghana Institute of Management and Public Administration; (9) Kwame Nkrumah University of Science and Technology in Ghana; (10) Kenyatta University in Kenya; (11) Egerton University in Kenya; (12) Addis Ababa University in Ethiopia (13) University of Rwanda; (14) University of Dar es Salaam in Tanzania; (15) Universite Abdou Moumouni de Niamey in Niger; (16) Université Cheikh Anta Diop in Senegal; (17) Universidade Pedagógica in Mozambique; and (18) The University of the Gambia in The Gambia.

Bakary Diallo

The Rector

African Virtual University

Production Credits

Author

William Korir

Peer Reviewer

John Kandiri

AVU - Academic Coordination

Dr. Marilena Cabral

Overall Coordinator Applied Computer Science Program

Prof Tim Mwololo Waema

Module Coordinator

Victor Odumuyiwa

Instructional Designers

Elizabeth Mbasu

Benta Ochola

Diana Tuel

Media Team

Sidney McGregor

Barry Savala

Edwin Kiprono

Kelvin Muriithi

Victor Oluoch Otieno

Michal Abigael Koyier

Mercy Tabi Ojwang

Josiah Mutsogu

Kefa Murimi

Gerisson Mulongo

Copyright Notice

Supported By

AVU Multinational Project II funded by the African Development Bank.

Table of Contents

Course Overview

Welcome to Introduction to Operating Systems

Operating systems are central to computing activities. An operating system is a program that acts as a communication interface among the user, application software and the computer's physical component. Two primary aims of an operating systems are to manage resources (e.g. CPU time, memory) and to control users and software.

This course introduces the fundamental principles of operating systems design goals, and gives them hands on experience with operating systems installation and maintenance.

Prerequisites

The course is designed to acquaint students with operating system, which is a resource handler and coordinator of the computer system and relate the operating system to users, applications and hardware. It is believed that students already have a good understanding of a computer organization and architecture as well as the different data structures and programming paradigms used in computer software that have an impact on the structure and performance of an operating system.

Materials

The materials required to complete this course are:

- A personal computer (PC) loaded with operating system software, like Windows or Linux
- Abrowser software
- An Internet connection

Course Goals

Upon completion of this course the learner should be able to ·

1. Demonstrate a good understanding of Operating System concepts, basic structure and functioning of OS;
2. Exhibit the problems related to the process management and synchronization as well as is able to apply learned methods to solve the basic problems;
3. Demonstrate the cause and effect related to deadlocks and is able to analyze them to related common circumstances in OS;

4. Demonstrate the basics of memory management, the use of virtual memory, and structure of most common file systems;
5. Elucidate the basic computer threats and security, embedded system and their requirements, and virtual machine

Units

Unit 0: Overview of Operating Systems

This unit gives a high level review of the computer system and operating systems. It discusses the structure and operation of a computer system emphasizing on issues related to operating system. Moreover, this unit addresses the definition, evolution, structure and services of operating system.

Unit 1: Operating System Principles

This unit emphasizes on the principles of operating system. The objectives of the operating system addressed from different perspectives are discussed.

Unit 2: Process Managment

This unit covers the major service delivered by the operating system, i.e., process. Process associated concepts, like thread management, inter-process communication, process scheduling, and deadlock, are covered.

Unit 3: Memory Management

This unit ellaborates and describes concepts of memory management. How memory is organized in a computer system, swapping, partitioning, and page repalcement algorithms are discussed.

Unit 4: Device Management and File Systems

The major goal of this unit is to enlighten students on how device and file systems are managed by the operating system. Characterstices and principles of input/output devices and softwares are discussed. File systems, directories, implementation of files and directories, file system hierarchy, file sharing, disk management are covered.

Unit 5: Security and Protection

This unit addresses concepts associated with security, specifically, encryption, user authentication, attack typs, and protection mechnisms.

Unit 6: Special Topics

This unit explains topics related to basic OS configuration and maintenance, real time and embedded OS, and basic virtual machines.

Assessment

1	Continuous assessment		Weight (%)
	1.1	Test I	10
	1.2	Test II	15
	1.3	Home take assignment	10
	1.4	Laboratory Project	25
2	Final Examination		40
Total			100

Schedule

Unit	Activities	Estimated time
Overview of Opearting Systems	Activity 0.1 – Structure of a Computer System	½ Hours
Operating System Principles	Activity 1.1 – Definition of Operating System	½ Hours
	Activity 1.2 – Evolution of Operating System	2 Hours
	Activity 1.3 – Structure and Services of Operating System	2 Hours
	Activity 1.4 – Laboratory Activity	4 Hours
	Activity 1.5 – Objectives of Operating System	1½ Hour

Process Management	Activity 2.1 – Overview of Processes	1 Hour
	Activity 2.2 – Process states	6 Hours
	Activity 2.3 – Thread Management	4 Hours
	Activity 2.4 – Inter-Process Communication (IPC)	10 Hours
	Activity 2.5 – Process scheduling	10 Hours
	Activity 2.6 – Deadlock	2 Hours
	Activity 2.7 – Deadlock Handling Mechanisms	10 Hours

Memory Management	Activity 3.1 – Overview of Memory	5 Hours
	Activity 3.2 – Swapping and Partitions	10 Hours
	Activity 3.3 – Page Replacement Algorithms	10 Hours
Device Management and File Syste	Activity 4.1 – Characteristicsof I/O devices	½ Hours
	Activity 4.2 – Overview of file systems	½ Hours
	Activity 4.3 – Directories	1 Hour
	Activity 4.4 – File system hierarchy	4 Hours
	Activity 4.5 – File system implementation	8 Hours
	Activity 4.6 – Directory implementation	8 Hours
	Activity 4.7 – File system reliability	4 Hours

Security and Protection	Activity 5.1 – Overview	½ Hours
	Activity 5.2 – Encryption	1 Hour
	Activity 5.3 – User Authentication	1 Hour
	Activity 5.4 – Types of Attacks	1 Hour
	Activity 5.5 – Protection mechanisms	1½ Hours

Special Topics	Activity 6.1 – Basic OS Configuration and Maintenance	5 Hours
	Activity 6.2 – Introduction to Real Time and Embedded OS	4 Hours
	Activity 6.3 – Basic Virtual Machines	3 Hours

Readings and Other Resources

The readings and other resources in this course are:

Unit 0

Required readings and other resources:

- Andrew S. Tanenbaum (2008). Modern Operating System. 3rd Edition. Prentice-Hall
- William Stallings (2002). Computer Organization and Architecture, 6th Edition. Prentice Hall

Optional readings and other resources for Evolution, Structure and Services of Operating Systems**:**

- William Stallings (2005). Operating Systems Internals and Design Principles. 4th edition. Prentice Hall.
- A Silberschatz, Peter B Galvin, G Gagne (2009). Operating System Concepts. 8th Edition. Wiley.

Unit 1

Required readings and other resources:

- Andrew S. Tanenbaum (2008). Modern Operating System. 3rd Edition. Prentice-Hall.
- A Silberschatz, Peter B Galvin, G Gagne (2009). Operating System Concepts. 8th Edition. Wiley.

Unit 2

Required readings and other resources:

- Andrew S. Tanenbaum (2008). Modern Operating System. 3rd Edition. Prentice-Hall.
- A Silberschatz, Peter B Galvin, G Gagne (2009). Operating System Concepts. 8th Edition. Wiley.

Optional readings and other resources for Process Scheduling:

- William Stallings (2005). Operating Systems Internals and Design Principles. 4th edition. Prentice Hall.

Unit 3

Required readings and other resources:

- Andrew S. Tanenbaum (2008). Modern Operating System. 3rd Edition. Prentice-Hall.

- A Silberschatz, Peter B Galvin, G Gagne (2009). Operating System Concepts. 8th Edition. Wiley.

Optional readings and other resources for Page Replacement Algorithms:

- William Stallings (2005). Operating Systems Internals and Design Principles. 4th edition. Prentice Hall.

Unit 4

Required readings and other resources:

- Andrew S. Tanenbaum (2008). Modern Operating System. 3rd Edition. Prentice-Hall.

Optional readings and other resources for File Systems:

- William Stallings (2005). Operating Systems Internals and Design Principles. 4th edition. Prentice Hall.

Unit 5

Required readings and other resources:

- Andrew S. Tanenbaum (2008). Modern Operating System. 3rd Edition. Prentice-Hall.

Optional readings and other resources for Encryption, User Authentication, and types of attacks:

- William Stallings (2005). Cryptography and Network Security Principles and Practices, 4th edition. Prentice Hall
- A Silberschatz, Peter B Galvin, G Gagne (2009). Operating System Concepts. 8th Edition. Wiley.

Unit 6

Required readings and other resources:

- A Silberschatz, Peter B Galvin, G Gagne (2009). Operating System Concepts. 8th Edition. Wiley.

Unit 1: Operating System Principles

What is operating system? What objectives does an operating system wants to achieve?

Introduction

This unit introduces definition, evolution, structure and services of operating system. As you were introduced in the course "Introduction to Computer Application", the operating system is a system software. What are operating systems functionalities? What were the activities performed to reach to the contemporary operating systems features? Since operating system is a software, what kind of software structure does it have?

This unit addresses the above questions providing the definition, history, services and structure of operating systems as a whole. Focus on terminologies used, functionalities of an operating systems has and types of operating systems.

The very foundations of operating system can be viewed from different perspectives. Mainly, these perspectives focus on user and system level. Additionally, we can consider the operating system's nature as a software that requires to evolve through time depending on change on hardware as well as human needs. The principles that govern operating systems can be viewed from the following:

- System view as resource manager
- Users view as virtual machine
- Ability to evolve as software

Unit Objectives

Upon completion of this unit you should be able to:

- State the principles and objectives of the operating system with the corresponding aims they have
- Demonstrate the task of operating system as being an interface, resource manager, etc
- Describe the correlation that exists between operating systems and computer architecture, the evolution stages of operating systems
- Demonstrate the services provided by the operating systems
- Describe the different software structures of operating systems

Key Terms

Batch system: A system that processes in jobs.

Time sharing system: A system that shares CPU's time between jobs.

Multi-programmed system: A system with uniprocessor executing several jobs at a time.

Learning Activities

Activity 1 Definition of Operating Systems

Introduction

So far, we dealt with the computer system along with its components. The focus is about the operating system software. What is it?

What is Operating System?

Operating systems are system software that run on top of computer hardware. This definition needs to be observed from different perspectives of computer system, namely from application software and user hardware interaction.

Operating system can be defined as, system program that monitors works of application programs. It is responsible for the loading and execution of application programs. It has to make sure the availability of the required hardware and software requirements before executing the application program.

Operating system can also be defined as system software which acts as a bridge between the hardware and its users. The operating system, according to this definition, has a responsibility to hide the complexities of the underlying hardware for the end user. The end user is not supposed to know the details of hardware components like CPU, memory, disk drives, etc.

Conclusion

As we have discussed the definitions of operating system so far, we can conclude that the operating system is found between user, application software, and physical component of the computer system. It facilitates and monitors running of application programs and functions as a middle man between hardware and users.

Instructions

Write short answers for the following questions

What is an operating system?

Feedback

Operating system is a general purpose software that monitors execution of application software. It is responsible for the loading and execution of application programs.

Activity 2 – Evolution of Operating Systems

Introduction

Computers gone through different generations. The generations are classified based on varieties of criterion. One of the major criterion to classify computer generations is the type of operating system used. Evolution of operating systems has a close tie with the generation of computers. What were the basic changes made on operating systems over these generations?

Evolution of Operating Systems

Studying evolution of Operating Systems is used to understand key requirements of an Operating System. Moreover, it helps to understand significance of the major features of modern Operating Systems.

Operating systems and computer architecture are historically tied. The combination of computer architecture with operating system is known as computer platform. Architectural changes affect structure and performance of Operating Systems. While studying the evolution of operating systems, consideration on the architecture is required.

The following shows the major evolution observed as per Operating Systems:

- Wiring-up plug-boards
- Serial Processing
- Simple batch processing
- Spooling batch processing
- Multi-programmed batch processing
- Timesharing
- Real time processing
- Symmetric multi-processing
- Network processing
- Distributed processing

i. Wiring-up Plug-boards

Low level programming language called machine language was used to write computer instructions through wiring up plug-boards Plug-boards control the basic functions of the computer. No Operating Systems software were introduced. No programming languages to develop applications. No operators were required. Complex programming and machine underutilization.

ii. Serial Processing

The Technique

Running a Job

- Developer creates his/her program and punches it on cards
- S/he submits the card deck to an operator in the input room
- S/he reserves machine time on a sign-up sheet
- Operator set-up the job as scheduled
- Operator caries over the returned output to the output room
- The developer will collect the output later

No Operating Systems software were introduced. No programming languages to develop applications.

Features

- Users get access in series
- Program writing was improved

Drawbacks

- Wasted time due to scheduling and setup
- Wasted time while operators walk around the machine room
- Large machine/processor idle time

iii. Simple Batch Processing

The Technique

To run jobs

- Users submit jobs on cards
- Operator batches jobs together into a tape inside the input room
- Operator rewinds the input tape and brings it to machine room
- Monitor, a special system program in the system tape, runs each job sequentially

- Output is written onto output tape and is brought to output room
- Output is printed offline in the output room
- Each job is structured using JCL (Job Control Language)
- To switch control back and forth between job and monitor
- To automate job set-up (loading source code, compilers, object code etc)
- Monitor controls sequence of events
- Relies on some hardware features
- Memory protection
- Timer
- Processor modes
- Interrupts
- Operating Systems introduced for the first time
- FMS – Fortran Monitor System for IBM 701 and later for IBM 704 IBMSYS – for IBM 7090/7094

Features

Much of the works of the operator are shifted to the computer

- Improve machine utilization
- Handle scheduling problem
- Reduce set-up time

Drawbacks

- Large turn-around time
- Large CPU idle time on heavily I/O bound jobs
- Large CPU idle time due to slower speed of I/O devices compared to CPU speed
- CPU idle time between batches of jobs: the computer is required to read the deck of cards to start performing the batched jobs.

iv. Spooling Batch Processing

The Technique

- Adds up Spooling technique to simple batch processing
- Spooling - Simultaneous Peripheral Operation Online
- It is the ability to read jobs/print job outputs while the processor executes jobs
- Read jobs – from cards to disk
- Print job outputs – from disk to printer

Used in input and output operations

With spooling, every time a currently executing task is completed, a new task is shipped from storage to the now-empty panel and be executed by the OS

Spooling is the first attempt of multiprogramming

Features

- Avoids CPU idle time between batches of jobs
- Improve turn-around time: Output of a job was available as soon as the job completed, rather than only after all jobs in the current cycle were finished.

Drawbacks

- Large turn-around time
- Large CPU idle time on heavily I/O bound jobs

v. Multi-programmed Batch Processing

The Technique

- Multiprogramming
- Multiple jobs are loaded in a partitioned memory
- Processor runs the jobs by cycling through them in some order
- Processor works on individual jobs for a specified period of time
- The system has a task of instantiating user tasks
- completing or winding tasks
- Managing I/O for the user jobs
- moving between user jobs
- Ensuring proper protection
- Relies on some hardware features
- Memory protection
- Timer
- Processor modes
- Interrupts
- DMA (Direct Memory Access)

Features

- OS/360 – the first multi-programmed batch system designed for IBM System/360

- Improve machine utilization
- Resemble a modern operating system

Drawbacks

- Long response time: the elapsed time to return back a result for a given job was often several hours
- Poor interactivity between programmer and his program

vi. Timesharing/Time-slicing

The Technique

Another form of multiprogramming where every end user has an online terminal

- Processor time is shared among multiple users
- Its main objective is getting quick response time
- The computer offers quick, cooperative services to several users
- CTSS – the first timesharing system developed for IBM 709 and later to IBM 7094
- CTSS supported 32 users at a time

Features

- Minimizes response time
- Support many users simultaneously
- Difference between multiprogramming and timesharing

Multiprogramming

- Its objective is to maximize processor use
- JCL commands are source of directives to the OS
- Designed to keep the processor & I/O devices simultaneously busy

Timesharing

- Its objective is to minimize response time
- Terminal commands are sources of directives to the OS
- Designed to be reactive to distinct users' needs.

Drawbacks of multiprogramming and timesharing

- Problems related with multiple programs in memory interfering with each other
- Problems related with multiple interactive users accessing the same file system
- Problems related with multiple programs competing for the same resource

vii. Real-time processing

Real time operating systems are operating systems that allow operation of jobs to be completed under a specific time constraints. Programs are guaranteed to have an upper bound on tasks that they carry out. Usually it is desired that the upper bound be very small. Severely constrained by the timing requirements. Usually used in dedicated/special purpose computers.

viii. Symmetric Multiprocessing

Runs on a multiprocessor standalone computer system. Processors share the same primary storage and I/O devices and can perform the same task. Jobs are scheduled across all processors. Several jobs run in parallel

Advantages

Performance: due to multiprocessing

Availability: Failure of a single processor does not halt the machine

Incremental growth: additional processors can be added easily

Scaling: Range of products with different price and performance

ix. Networked Processing

Comprise of several computers that are interconnected to one another. Each of these networked devices has their own local users and executes their own local OS not basically different from single computer OSs. Users also know about the presence of the several computers. Additional features needed by network operating systems

- Network interface controller, NIC
- A low level software to drive it
- software for remote login & remote file access

x. Distributed Processing

Runs on a multi-computer system: set of computers, each having its own memory, storage devices and other I/O modules. This is useful for distributing the task between these different computers. Existence of multiple computers is transparent to the user: It appears as a uniprocessor system. It differs in critical ways from uniprocessor OSs. Examples of distributed operating systems include LOCUS, MICROS, IRIX, Solaris, Mac/OS, and OSF/1.

Conclusion

Evolution of operating systems over the past decades has shown number of progresses from several perspectives. Starting from the point where machine does not have operating systems to a distributed processing capability. The foundation for the contemporary operating system was built over different parameters.

Assessment: Multiple Choice Questions

Instructions: Choose the best answer

1. Monitoring utilities were put to memory when required in simple batch processing systems

    a.True

    b.False

2. In which type of system, the output of a program is sent to the disk and is printed only when the job completes?

    a.Multi-programmed batch systems

    b.Batch systems with spooling

    c.Time sharing systems

    d.Multi-printing systems

3. A system that supports multiple processes per user is called a

    a.Multi-user system

    b.Multi-programming system

    c.Multi-tasking system

    d.None of the above

4. You developed an operating system that allows a computer to work independently as well as provides a means for sharing resources (e.g. files, printers, etc) to other computers connected to it. What type of operating system did you develop?

    a.Real-time operating system

    b.Symmetric operating system

    c.Local area network

    d.Network operating system

Activity 3– Structure and Services of Operating Systems

Introduction

Lots of issues have been discussed so far to understand the external view of an operating system. This section looks at different operating system structures to understand the inside and also the services provided by an operating system to its various clients: users, processes and other systems.

Structure of Operating System

Any system as complex as the Operating System cannot be produced unless a structured approach is used. A number of structures have been used to develop the various operating systems that have been developed for the last five decades. In this module five types of OS structures will be discussed though several other structures exist. Tradeoffs between competing requirements are involved when designing an operating system. Major issues taken into consideration for the design of an OS internal structure are:

Performance criteria

- Most OS's are designed to support user interactions.
- Launching new processes as well as responding to users input need to be fast

Security level

- Different level of security are desired according to the target environment
- Server OS vs workstation OS
- Correctness and maintainability:
- Well modularized code is easier to maintain
- Clean architecture is also an advantage

Compatibility

- Backward compatibility is main requirement for an OS
- Codes difficult to be maintained are inevitable

Available hardware

- Low-level functions of an OS often require hardware support such as protection mechanisms
- Can limit allowed commands

Activity Details

Reading Activity:

Monolithic OS structure

- It is also referred to as "The Big Mess" due to an absence of any kind of organization
- It consists of a bundle of procedures calling one another whenever they need to

- Each procedure has a well-defined interface in order to allow the programmer to use it in the development of other procedures.
- The object program of the OS will be constructed by first compiling each procedure separately and then binding them all together into one object using the system linker

At its worst case

This structure is basically one big lump with no well separated functionality levels, no clearly defined internal interfaces and no level of encapsulation as every procedure is visible to all others.

Example: MS-DOS

At its best case

The monolithic structure includes a little structure. It has clearly defined interfaces in that services of the Operating System are demanded by placing the required arguments in a well-defined area and then executing a trap instruction. The OS will then get the parameters and determines which service is being requested and indexes into a table that contains pointers to the procedures carrying out the services. In this case, the basic structure of the operating system can be summarized as:

- Main program invoking required system call procedure
- Collection of service modules to perform the system calls
- Collection of helpful programs assisting the service procedures

But the compiled code is still one big executable and changes in some part requires re-compiling the whole package.

Example: most Linux Kernels

Layered OS structure

- Hierarchical organization of an OS where each layer implements a function based on the layer beneath.
- The lower most level is the hardware and the uppermost level is the user interface
- High degree of encapsulation
- Easily maintainable and extendable
- With modularity, layers are arranged in such a way that each upper layer uses services of only the lower level layers
- This was the first OS designed with such structure by Dijkstra and his students
- It is a batch operating system with 6 layers

Layer	Name	Function
5	Operator	Interface for the system to operate
4	User programs	Enable users to perform an action
3	I/O Management	Controls I/O devices and buffers information stream to and from them
2	Operator console	Manages communication between processes and operator console
1	Memory management	Allocate space for processes
0	CPU scheduling	Processor allocation and Switching between processes when an interrupt occurs

Virtual Machines

- This approach provides several Virtual Machines identical to the underlying bare hardware without any additional feature such as a file system.
- Considers the hardware and the operating system kernel together as a hardware
- Is used to separate the multiprogramming and extended machine functions found in timesharing operating systems
- The VM Controller that runs on the plain machine manages the multiprogramming and provides multiple virtual machines to the upper layers
- Virtual machines are created by sharing the physical machine's resources
- Exact copy of actual computer is given to every user process.
- The virtual machines can run different operating systems
- As each virtual machine is totally insulated from all other machines, there is a complete protection and thus high encapsulation

- however, resource sharing is difficult among the VMs
- VM in general, is difficult to implement as it requires an effort to create exact copy of the underlying machine

Exokernels

- In such OS design, users are presented with a replica of the actual device but with only a portion of the device's resources
- The exokernel allocates resources to the available virtual machines and controls any attempt of accessing others resources.
- It has a main advantage of avoiding address mapping

Client-Server Model

- Modern operating systems tend to move codes, as much as possible from the operating system up into upper levels to leave a minimal kernel or microkernel
- A so called message passing is used to facilitate communication among user modules
- A user process called client process sends its request for a service to a server process
- The server process responds to the request by performing the required service and sending back the answer to the client process
- This structure splits the operating system into smaller and manageable parts each carrying out only one part of the system
- Easily maintainable and extendable as all server processes run in user mode
- Is more reliable and secure OS structure
- Easy to port the OS to new architectures
- It also has an advantage of being adaptable for distributed systems
- However, there is a performance overhead of user space to kernel space communication

Microkernel

- A microkernel OS is structured as a tiny kernel that provides the minimal services used by cooperating processes. These processes later provide the higher-level functionalities of operating system.
- The microkernel itself do not have file systems and several other functionalities which are required to be offered by an operating system.
- Microkernel provides a mechanism to implement the operating system.

Services of Operating System

Operating system provides several services to users, programs as well as the whole machine. As system software, it acts as a bridge between users and the hardware in accepting user requests and interpreting them into machine executable commands. Processes directly consult the operating system for their various resource requests as well. Some of the basic services provided by an operating system are listed below.

User Interface

Users communicate with a computer system to use its resources through an interface provided by an operating system

The UI provided can be:

1. **Command Line Interface (CLI):** allows direct command entry by fetching commands from user and executing it. it provides text based interaction for the user where by users are forced to remember the label of each command. The commands range from a built-in to simple program names.
2. **Graphical User Interfaces (GUI):** easy to use, user oriented interfaces where users interact with the system through icons. These are user friendly desktop metaphor interfaces. Users pass their commands through keyboard, mouse and monitor.

System and utility function calls

These are programming interfaces to the services provided by the operating system.

Programs access these calls through an Application Program Interface (API)

They are written and used with a high-level programming language like C++.

When implemented, each system call is associated with a number that is used for indexing.

The API invokes the required system call in the OS kernel and sends back the status of the system call along with any return value.

The API hides any detail of the OS from the caller.

System calls can be about:

Process control

Creating, terminating and waiting for a process

File management

Create, read, write and delete files

Communication management

Creating connections, sending message, receiving message, sharing memory

Information maintenance

Getting/setting time, system data, file and process attributes

Device management

Access, read, write devices

Most commonly known APIs are: WIN32 for Windows and POSIX for UNIX, LINUX and MAC OS

The following table shows the system calls for different activities in POSIX and WIN32

Process management		
POSIX	WIN32	Description
Fork	CreateProcess	Clone current process
exec(ve)		Replace current process
waidpid	WaitForSingleObject	Wait for a child to terminate
Exit	ExitProcess	Terminate current process and return status
File Management		
Open	CreateFile	Open a file & return descriptor
Close	CloseHandle	Close an open file
Read	ReadFile	Read from file to buffer
Write	WriteFile	Write from buffer to file
Lseek	SetFilePointer	Move file pointer
Stat	GetFileAttributesEx	Get status info
Directory and File system management		
Mkdir	CreateDirectory	Create new directory
Rmdir	RemoveDirectory	Remove empty directory
Link	(none)	Create a directory entry
unlink	DeleteFile	Remove a directory entry
mount	(none)	Mount a file system
unmount	(none)	Unmount a file system

Table 1 POSIX and WIN32 System Calls [Tanenbaum]

Program Execution

The OS is capable of loading a program onto main memory and run the program

It ends the execution of a program normally, if no errors are found, or terminates abnormally otherwise

Input/output Operations

Running programs mostly require input and output tasks.

The input/output can be either from/to file or I/O devices

It is the task of the OS to facilitate such kinds of requests coming from programs

File Management

The OS handles reading and writing files and directories

It also manages creation, deletion and searching of items in the files and directories

Access Permission of files and directories is also the service of an OS

Error Detection

Errors might occur in a system in different areas including CPU, memory, I/O devices or even user programs

The OS ensures correct and consistent operation of the system despite the errors occurred by taking the appropriate measurement on the detected errors

To do so, the OS requires to continuously inspect the system and identify possible errors

Communication

Processes found on the same computer or over a networked computers might need to communicate among themselves

This communication can be carried out either through message passing or shared memory concept

The OS facilitates such communication among processes through implementing different communication management mechanisms

Resource Allocation

A computer system consists of several kinds of assets

Some of these resources such as the CPU and main memory need unique assignment code while others like the I/O devices might need a general request and release code to be accessed

It is the job of the OS to identify the type of resource required and run the appropriate code to access the resource

Especially, in a multi-tasking and multi-user environment, the OS should be able to assign resources for each of the concurrently running jobs

Accounting

The OS also keeps track of the number of users using the system,

It also maintains which users are using which kinds of resources and their consumption rate of the resource as well

System Protection

The OS controls all access to a systems' resource

It also enforces authentication and different access control mechanisms to keep the system free of undesired access attempts

Conclusion

It is impractical to design and implement an operating system, which is a large and complex software, without having a certain structuring method. There are various models for the structure of operating systems. The main five models are: monolithic structure, layered structure, virtual machines, exokernels and client server structure. The Operating System is the glue that binds users request and systems' response together by facilitating an abstracted and easy to use environment for the users and providing other several services to programs as well.

Operating system provides number of services like user interface, program execution, system protection, accounting, input/output management, file management, error handling, etc.

How do we interact with the operating system? How does the operating system does exposes its services? The next laboratory activity will be focusing on these questions.

Assessment: Essay Type question

Instructions

- Write the answers of the following questions.
- What are the major issues considered when the internal structure of operating system is designed?
- What is the worst case scenario of the monolithic structure of operating system?
- Discuss the different layers of layered operating system structure?
- What is a microkernel?
- List at least four services of the operating system?

Feedback

The following issues are considered: performance criteria, security level, Correctness and maintainability, compatibility, and the available hardware.

Monolithic structure is basically one big lump with no well separated functionality levels, no clearly defined internal interfaces and no level of encapsulation as every procedure is visible to all others.

The has the following layers :

5	Operator
4	User programs
3	I/O Management
2	Operator console
1	Memory management
0	CPU scheduling

A microkernel OS is structured as a tiny kernel that provides the minimal services used by cooperating processes.

The following are services of the operating system: user interface, program execution, system protection, accounting, input/output management, file management, error handling, etc.

Activity 4 – Laboratory Activity

How to Use the Shell

Objectives of the Labortory

- Introduce the Ubuntu shell
- Use the shell to execute system calls
- Write and execute C program to issue operating system calls

Details of the Laboratory Exercise/Activities

Objective of the Exercise

- Introduce the Ubuntu shell or terminal
- Execute the shell or terminal

Resources Required

- Personal computer installed with Ubuntu Operating System

Time Requried

- 1 Hour

Description of the Laboratory Exercise/Activities

Introduction

How do we interact with the operating system? How does the operating system exposes its services? This laboratory activity will focus on these questions.

Use any of the Linux operating systems like Ubuntu, Red Hat, Fedora, CentOS, Kali, or any other to perform the lab activities in this module. The source code are tested on Ubuntu 13.04 in this module. Run your machine and logon. Use the terminal to gain access to the shell.

A shell is a command line interpreter that acts as a main boundary between a user sitting at his/her terminal and the operating system. It is a process (running program) that reads commands. Although it is not made as a component of the operating system, it makes substantial use of several features of the operating system.

How the shell works?

A shell is activated or started during a login activity of any user

The shell then displays a prompt to inform the user that it is ready to geta request from him/her

When the user enters a command, a child process is created by the shell that runs the program for the given command

The shell then waits until the child process finishes its execution and comes to termination

When the child process is done,, the shell displays the message and waits to get the next command

Turn the computer on. After the Ubuntu operating system loads and ready for user, simply search Terminal from the application list to launch the shell.

When running the shell, you will be prompted the following interface:

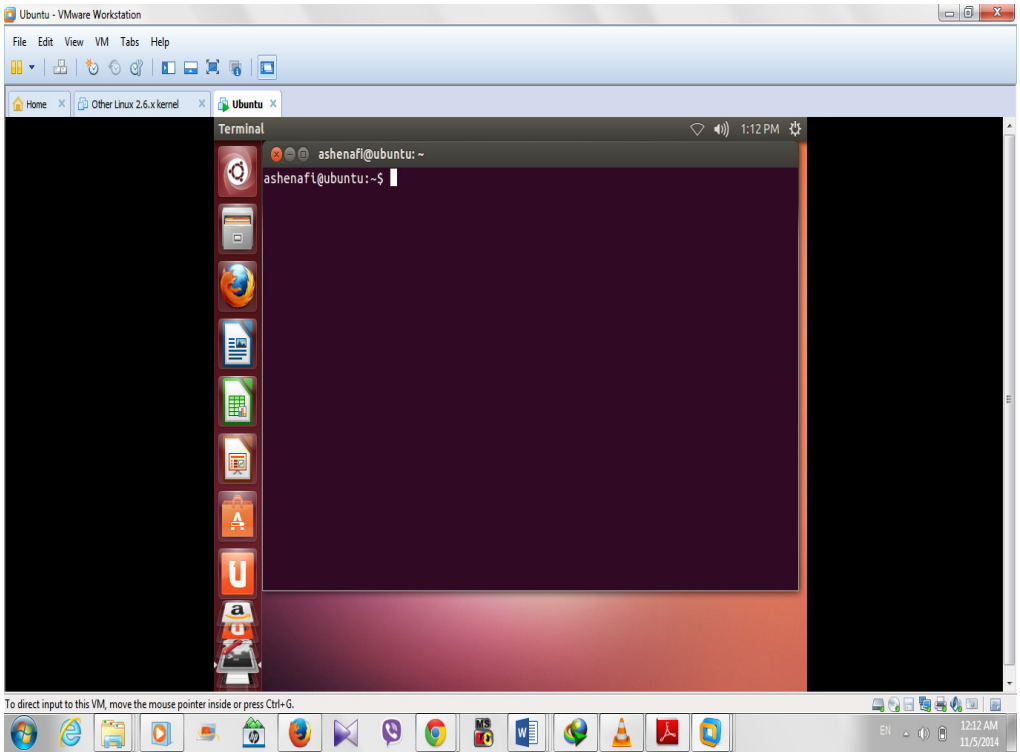
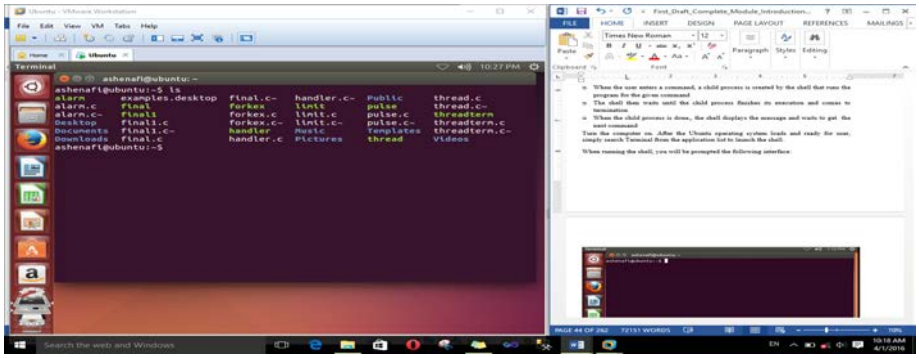


Figure 1.1: Terminal interface of Ubuntu operating system

On the shell prompts with the user name and the server name separted by @ sign. As you can see from the screenshot, ashenafi stands for the user name and ubuntu is the server name.

To execute system calls, simply type the name of a command. For example, if we execute the ls command the result looks like the following.



Results and Submission Requirments

Execute the following system calls and take the screenshot of the output result.

- ls –l
- pwd
- whoami
- finger

Assessment Criteria

The student has to show the outputs of the system calls with the corresponding description to obtain total of 4 marks.

References or Key Links

- http://linuxcommand.org/http://linuxcommand.org/
- https://linuxconfig.org/linux-commands

Details of the Laboratory Exercise/Activities

Objective of the Exercise

- Edit, link, and execute C program
- Work with system calls like fork, getpid, getppid, open, write, close, and create.

Resources Required

- Personal computer installed with Ubuntu Operating System

Time Requried

- 3 Hours

Description of the Laboratory Exercise/Activities

Let's see some examples of the system calls discussed earlier through programming activities.

Linux has many text editors. User either vi, vim, emacs, or gedit to write the following C programs.

The fork() system call gives an output of true in the parent and false in the child.

Type

geditforkex.c (Enter key).

Type the code shown in the following box.

```
#include <stdio.h>

#include <sys/types.h>

int main()

{

pid_t pid;

pid = fork();

if(pid > 0)

{

/* parent process gets executed */

printf("This is the parent process with process id %d\n.", getpid());

}

else

{

/* child process gets executed */

printf("This is the child process with process id %d and parent
process id %d\n.", getpid(), getppid());

}

return 0;

}
```

To save and exit from the editor:

Press ESC key from the keyboard

Type colon(:), w, and q

Press Enter key

To compile and link the source code from the shell type the following:

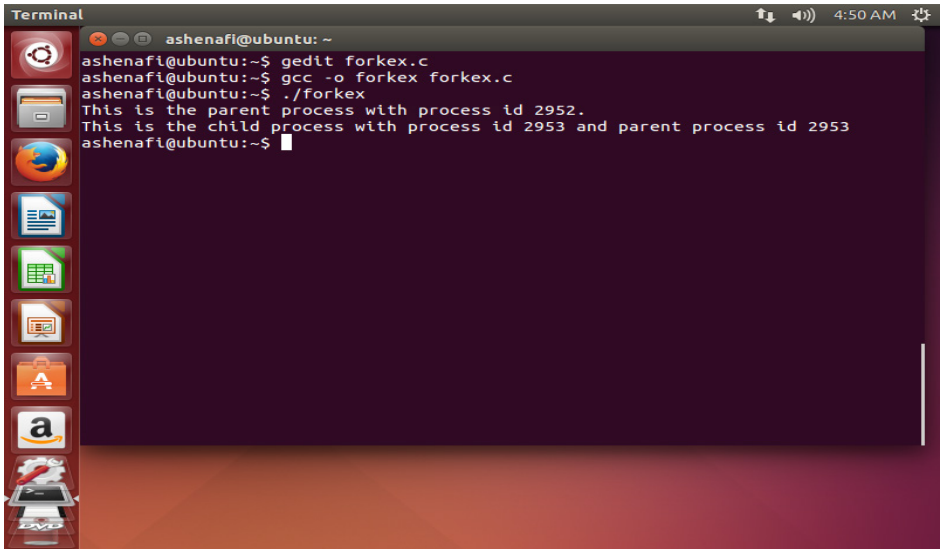
$ gcc –o forkex forkex.c

Note: If bugs are found on the source code, the compiler notifies on the shell interface. If that occurs, go back to the source code as you have done it earlier and correct the errors. You have to recompile the source code after you fix the errors.

To execute the program, write the following on the shell prompt:

$ ./forkex

The output looks like the following:



The following code, opens an existing file called my file twice, write once and read once via two different file descriptors

```
#include < string.h >

#include < unistd.h >

#include < fcntl.h >

int main (void)

{

int fd[2];

char buf1[12] = "just a test";

char buf2[12];

fd[0] = open("myfile",O_RDWR);

fd[1] = open("myfile",O_RDWR);

write(fd[0],buf1,strlen(buf1));

write(1, buf2, read(fd[1],buf2,12));

close(fd[0]);

close(fd[1]);

return 0;

}
```

Following is an example of how to use create():

```
/*  creat.c */

#include <stdio.h>

#include <sys/types.h>   /* defines types used by sys/stat.h */

#include <sys/stat.h>  /* defines S_IREAD & S_IWRITE  */

int main()9     {

int fd;

fd = creat("datafile.dat", S_IREAD | S_IWRITE);

if (fd == -1)

printf("Error in opening datafile.dat\n");

else

           {

printf("datafile.dat opened for read/write access\n");

printf("datafile.dat is currently empty\n");

           }

close(fd);

exit (0);

     }
```

Results and Submission Requirments

Edit, compile, link, and execute the last two programs and show the outputs. What are changes on the files? Show the contents of the files affected by the programs.

Assessment Criteria

The student has to show the outputs of the programs with the corresponding description to obtain total of 4 marks.

References or Key Links

- http://linux.die.net/man/2/forkhttp://linux.die.net/man/2/fork
- http://man7.org/linux/man-pages/man2/

Conclusion

One can interact with the kernel of the operating system using the interface provided by the operating system. Shell is the interface that is not part of the kernel. It is used to execute system calls of the Unix operating system. Here, we also covered how we can write C codes to access the system calls of Unix operating system.

Assessment

Practical Assignment

Write a C program that prints the title of the module you are learning along with your full name on new line. Edit, compile and execute the program you wrote. Print the screen shot of the program output.

Activity 5 – Objectives of Operating System

Introduction

The very foundations of operating system can be viewed from different perspectives. Mainly, these perspectives focus on user and system level. How is the operating system perceived from user view, hardware view, and as a software?

System view as resource manager

For the computer system, an operating system is a software that closely interacts with the hardware. A computer system has a collection of components for moving, processing and storing data. Thus, from the machine point of view, an operating system can be considered as resource manager and allocator. A number of resource requests that may also possibly be conflicting comes to the system and the operating system should choose how to assign them to particular programs and users and ensure the system is performing efficiently and fairly. The operating system should also handle different I/O devices and application software found in a computer system so that bugs and improper use of the system is prevented.

Users view as virtual machine

Most Computer users have a PC with system unit, monitor, keyboard and mouse. In other cases, users might sit at a terminal connected to large computers as mainframe and minicomputer or at workstations connected to other networked workstations. In all these cases, requirements of users totally vary. In the case of a PC, the system is designed only for single user and the user is allowed to monopolize all the resources; the goal is maximizing the service that the user obtains. Thus, the operating system is designed to provide simple use, with some considerations to performance but nothing to resource utilizations.

In the second case where users are sitting at a terminal, there are also other users sharing the same resource with them in their respective terminals. For such cases, the operating system should be designed in such a way that the main concern is resource utilization to guarantee efficient usage of all available resources such as CPU time, main memory and I/O and everyone is sharing the resources fairly. In the latter case that involves a workstation, users have both dedicated resources at their disposal and also share resources like file servers or print servers with others. The operating system for such cases should be designed to compromise between resource utilization and individual usability.

Considering all these and other circumstances of users' requirement, an operating system is a system designed to provide an easy-to-work-in environment for users and manage all the possible needs of its users regardless of the resource scarcity.

Ability to evolve as software

The operating system is a system software that bridges hardware with the rest of the computer components.

The operating system is the software that communicates with the software, i.e., device drivers, of the hardware which are the products of different vendors. If there are any changes on the hardware peripherals, then the vendors change the device drives of the hardware. Thus, these changes needs to be incorporated in the operating system.

The operating system, as discussed in unit 1, is responsible services to the user. If the user is requiring additional services, the operating system has to deliver those services, too.

As a software, the operating system might have a bug. Later on, usage of the operating system will result in error on the operating system. These bugs must be fixed.

The addition of hardware resources, need for new services, and maintenance of bugs has to be done on the next version of the operating system. Every version of the operating system has at least one of the following features:

- Capability to recognize a new hardware
- New service(s)
- Fixed bugs

Conclusion

The objectives of the operating system can be viewed from the following perspectives:

Convenience – Top Down View – Virtual Machine

Extending the real hardware and acting as a Virtual Machine

Hiding the truth about the hardware from the user

Providing the user a convenient interface which is easier to program

Efficiency – Bottom Up View – Resource Manager

Providing an orderly and controlled allocation of resources

Improving resource utilization

Ability to Evolve

Adaptability to new changes

Hardware upgrades/New types of hardware

Need of new services

Easy fixes of latent errors

Unit Assessment

Check your understanding!

Assessment: Essay Type Questions

Instructions

1. Write your answer for the following questions
2. Read about the different extensions of the virtual machine structure.

    a. How can a client-server model be used on a single computer?

    b. Discuss the advantage of virtual machine structure over the layered structure of an Operating System.

    c. What advantage does the accounting service of an Operating system has for users?

    d. What are the objectives of operating system? How do you describe the operating system as a resource manager?

Grading Scheme

This unit's assessments worth a total of 20% from which

- Activity Assessments: 5%
- Unit Assessment I:3 %
- Unit Assessment II (Lab Project): 12%

Feedback

Rubrics for assessing activities

Assessment Criteria	Doesn't attempt both assessment or the answers are all wrong	Attempt both assessments and provide partial answers or partially correct answers	Attempt all assessments with full and correct answer
Grading Scales	Fail	Got half mark for each assessment	Score full mark for each asssessment

Unit Readings and Other Resources

- Andrew S. Tanenbaum (2008). Modern Operating System. 3rd Edition. Prentice-Hall. Chapter 1.

- A Silberschatz, Peter B Galvin, G Gagne (2009). Operating System Concepts. 8th Edition. Wiley. Chapter 1 and 2.

Unit 2: Process Management

Unit Introduction

The most central notion in any operating system is a process. A process is a program currently running and that needs different kinds of resources to be allocated. A process also interacts with other processes more frequently. Thus, an operating system is responsible to manage all the requirements of processes and also coordinate communication among different processes. This unit covers in detail processes, types of processes, the different states a process be in, the difference between process and thread and the interaction among different processes.

Traditionally, a computer system executes processes sequentially. There are cases where we would like the system to divert from this fashion and handle processes execution differently. Scheduling is a means for deciding which process runs at a given time. It is a process of selecting one of the processes currently in main memory that are ready to run, and grant the CPU to one of them. Scheduling is the basis for multi-programmed systems as the system needs to make choices to grant the CPU to the ready state threads and processes. This unit elaborates the need for scheduling and the different scheduling algorithm used to schedule processes effectively. A deadlock situation and its consequences as well as the solution towards it will also be discussed in this unit.

- Overview of Process
- Types of processes
- Process states
- Thread Management
- Inter-Process Communication (IPC)
- Scheduling and Dispatch
- Deadlock

Unit Objectives

Upon completion of this unit you should be able to:

1. Describe how process is managed
2. Demonstrate how CPU scheduling is managed
3. Demonstrate a deadlock situation and how deadlock is managed

Key Terms

Process: Program in execution

Multiprogramming: Several programs are running at the same

time on a uniprocessor

Thread: A light-weight process

Program: Execute code

Concurrency: Processes executing at a time

Scheduling: Ordering of several processes' execution

Scheduler: An algorithm that selects a process to execute next based on the scheduling algorithm

Context switch: Swapping of processes on the CPU

Deadlock: Two or more processes waiting to each other to release a resource

Learning Activities

Activity 1 – Overview of Processes

Introduction

One of the fundamental functionalities of operating system is process management. What is process? What types does it have?

Early computer systems were capable of running only one program at a time and the program had full control of the systems' resources and thus no structuring is required. But today's computer systems are designed to support multiprogramming, running more than one program concurrently, besides the operating system they have. This arrangement requires the system to know which is which so that going back and forth between the programs is possible and thus enforces the need for structuring. Process is an instance of a program that is being executed. It is a program that has been loaded from disk onto memory and is running. While a program is just a passive set of commands, process is the actual execution of these instructions and one program can be partitioned to more than one processes. It is the basic unit of execution in an operating system. A process consumes some amount of assets like CPU time, main memory, and I/O devices to carry out its operations. It is the duty of the operating system to manage all these requirements of processes and bring them to termination.

A process is the basic execution unit in an operating system with unique number assigned as an identifier (PID). Though the CPU is running only one program at a time, it may also work on multiple programs by switching and running them for tens or hundreds of milliseconds. This in turn requires more control and partitioning of programs which resulted in a process. Thus, the operating system will have many processes running at the same time all requiring resources.

A process has three main components namely address space, processor state and OS resources.

1.**Address space** is a memory location that the process can access. It is how a process sees its memory. Every process has a separate address space. An address space encompasses many parts like the instructions, local variables, heap, stack, etc. as can be seen in the following diagram.

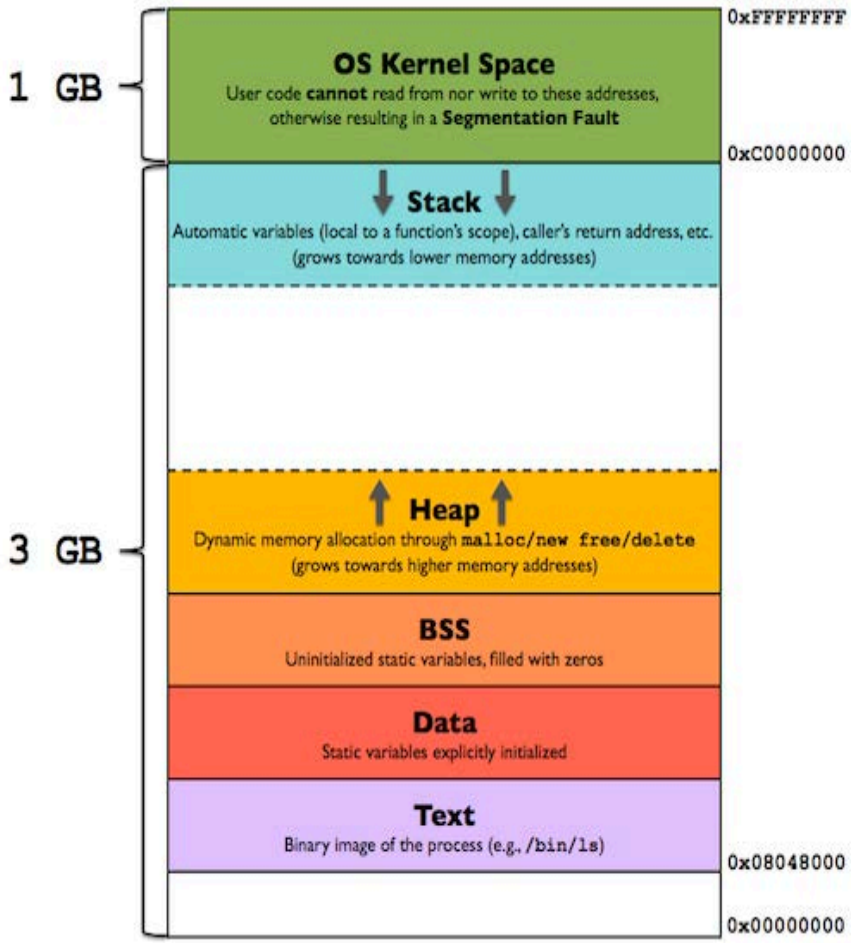


Figure 2.1. A process in memory

From figure 2.1. above:

Text Segment /Code *Section* holds the executable Code of the program which is read from secondary storage when the program is launched

Data Segment maintains static variables and initialized Global variables prior to the execution of main

Heap is used for dynamically allocated memory during run time and it is managed through new, delete, malloc, etc. calls.

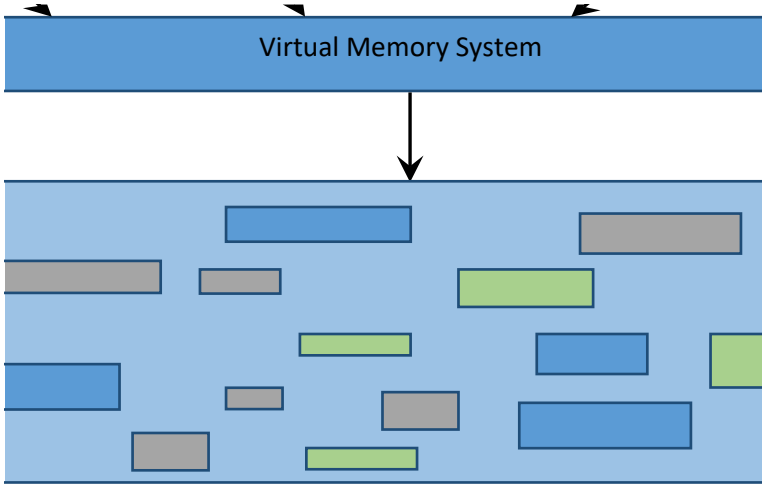
Stack Segment holds all local variables and saved registers for each procedure call. The stack will be freed when the variables go out of scope

2**. Processor state**: these are CPU registers associated with the running process. The registers hold the results so far of running the code of the process. It includes general purpose registers, program counter, stack counter, etc. the information on these registers must be stored and restored as the processes swap out of memory and later restored.

3. OS resources: the different OS state related with the process. Examples: open files, network sockets, etc.

The operating system maintains a process table to store these details of each process.

A conceptual model to simplify the management of several parallel processes is being used while designing an operating system. The design organizes all executable programs on a system into a number of sequential processes. The CPU has a multiprogramming capability where it switches between processes back and forth to manage their execution. Every process has its own virtual program counter that will be placed onto the actual physical program counter of the system when the process runs as shown in figure 3.2. below. And when the process finishes its execution for now, the content of the actual program counter is maintained in the processes' logical counter in memory. Processes shall not be developed with built-in timing assumptions as the amount at which a process carries out its operations may not be consistent due to the back and forth movement of CPU among processes.



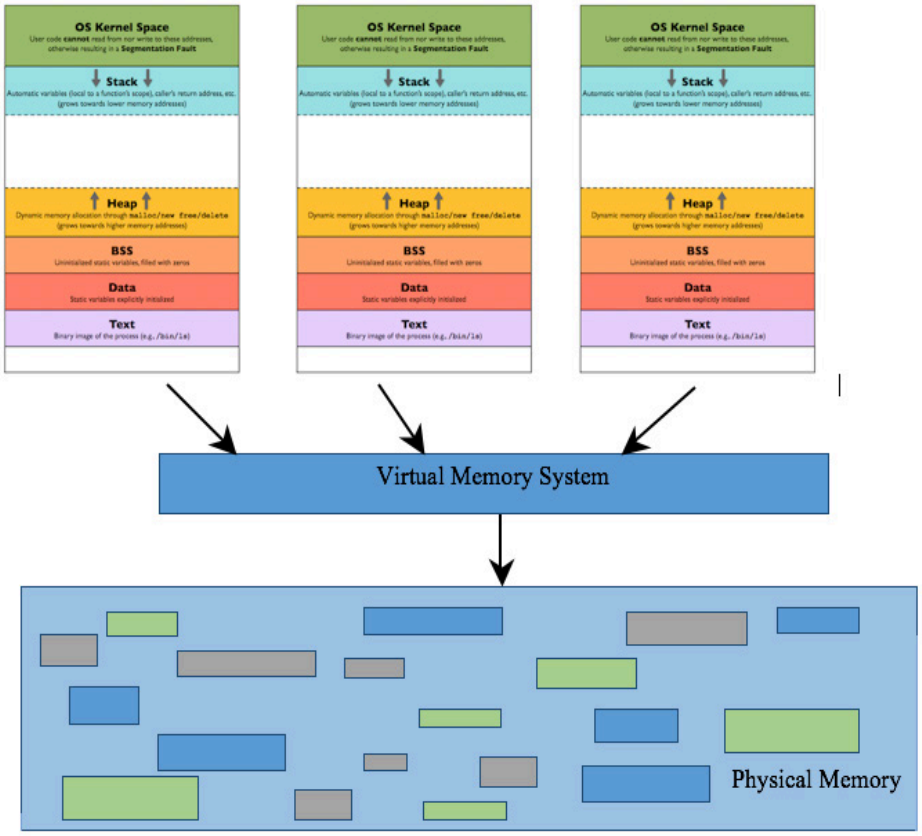


Figure 2. Conceptual model of Multiprocessing

A general purpose operating system allows for processes to execute concurrently and needs a mechanism for creation and termination of processes as needed during operation to assure all necessary processes exist. There are three basic situations for a process creation. These are:

System initialization

Several processes are created at system boot time. These processes can be either foreground processes that allow users to accomplish a task or background processes known as daemons that handles a specific activity for the system such as web page or print handling and are not associated with a specific user. The ps program and task manager are used to list processes in UNIX and Windows operating system respectively.

Running process: executing process creation system calls

Sometimes, a running process requests a system call for creating new processes to assist it on its jobs if the job can easily be formulated into multiple inter-related but independent communicating processes.

User request

New process can also be created when a user types a command or double clicks an icon to start a program. If the operating system has a command based interface, the new process uses the window in which it was created. On the other hand, if the operating system is graphical, a process do not have a window when started so it creates one. In both cases, users can have multiple windows opened simultaneously and they can interact with the processes through a mouse click.

The other way in which processes can be created is through batch systems found on mainframe computers. In such cases, users give bunch of tasks to the computer and if the operating system has the required resources to entertain another task, it produces the new process requested.

In all the process creation cases discussed, a process creation system call is executed from an already existing process to create the new one.. The operating system will be informed by the system call to create the new process and which program to run in it. The system call used to create new processes in UNIX is fork(). Fork makes an exact replica of the calling process which results in parent and child processes. These processes have the same memory image, environment string and open files. The child process can then execute the execve system call to create its own memory image and run a new program.

CreateProcess is the system call used to create new process in windows operating system.

Laboratory Activity

Objectives of the Labortory

- Execute process management commands
- Work with pstree, fork, CreateProcess, WIFEXITED, etc… system calls
- Analyze the outputs of the system calls

Details of the Laboratory Exercise/Activities

Objective of the Exercise

- Use the shell to execute UNIX's ps system call and Windows task manager processes
- Analyze the outputs of UNIX and Windows process management tools
- Resources Required
- Personal computer installed with Ubuntu and Windows Operating System

Time Requried

- 1 Hour

Description of the Laboratory Exercise/Activities

Using the ps command

- Type the ps command on your UNIX terminal:
- root@ubuntu:/home/ubuntu# ps
- What is being displayed ?

PID	TTY	TIME	CMD
3038	pts/0	0:00:00	su
3047	pts/0	0:00:00	bash
3510	pts/0	0:00:00	ps

Table 2.1. ps command output

What does each columns' value indicate?

PID is the process ID, a unique number given to every running process. A 32-bit counter called last_pid is used by the operating system hold the last PID given to a process. This counter value is incremented during process creation making its value the PID for the new process.

TTY is used to show the terminal a process is attached to pts means the shell. /0 shows this is the first shell launched by the user. If you launch another shell it would have pts/1 for its TTY field.

TIME indicates the total time required by the process to reach its exit

CMD (COMMAND), is the name of the running command. Here, we're only looking at the processes which the user root is currently running (in this terminal or shell). These are bash (root's shell program), and the ps command itself. As you can see, bash is running concurrently with the ps command

Exercise

1. Open other terminals and combine ps with e as ps –e
2. What changes do you observe?
3. What about ps –l? What does each of the additional columns represent?
4. Now combine ps with e and l as ps –e –l/ps –el and observe the changes on the output
5. Do the same with task manager in windows operating system

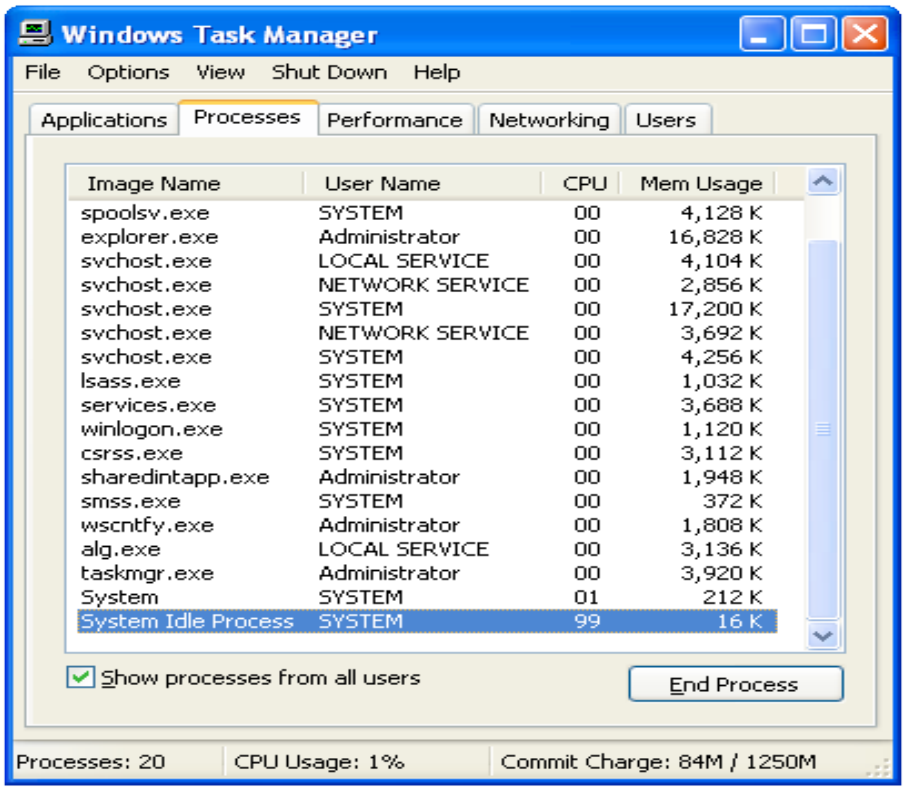


Figure 2.2. Windows task manager

Processes have child-parent relationship. The child process can have several child processes under it forming a hierarchy of processes. Note that a process has only one parent but zero or more children. In UNIX this hierarchy forms a process group and signal sent from user will be received by all. The processes in the group then acts differently for the request received. For instance, when the UNIX operating system is started, it is initialized with init process which is present in the boot image. This process reads a file that holds the number of terminals existing in the system and fork off new child processes per each terminal. The child process wait for login request and when received, shell is executed to get commands. The commands in turn may start other new processes. This shows that all the processes are under same tree having init at the root. This hierarchical arrangement of processes in UNIX can be viewed as tree.

Results and Submission Requirments

Respond to the questions on the exercise.

Assessment Criteria

The student has to properly answer the questions on the exercise to obtain total of 3 marks.

References or Key Links

- http://linux.about.com/od/commands/l/blcmdl1_ps.htmhttp://linux.about.com/od/commands/l/blcmdl1_ps.htm
- http://linuxcommand.org/man_pages/ps1.htmlhttp://linuxcommand.org/man_pages/ps1.html
- http://linux.about.com/od/commands/l/blcmdl.htm

Details of the Laboratory Exercise/Activities

Objective of the Exercise

- Use the shell to execute UNIX's pstree system call

Resources Required

- Personal computer installed with Ubuntu Operating System

Time Requried

- 10 Minutes

Description of the Laboratory Exercise/Activities

using pstree and related commands

The pstree command can be used to display a tree structure of all processes in a system

Uses appropriate versions of ps and pstree commands and list five processes that are currently sleeping

Display a process which has a grandparent along with ID of its parent and grandparent

However, windows operating system does not have such hierarchical arrangement of processes and every process is equal. When a process, the parent, defines a new process, the child, the parent will be given a special token known as handler to control the child process. But this handler can be passed to another process freely which invalidates the hierarchy.

Details of the Laboratory Exercise/Activities

Objective of the Exercise

- Use the shell to execute UNIX's fork and CreateProcess system calls

Resources Required

- Personal computer installed with Ubuntu Operating System

Time Requried

- 1 Hour

Description of the Laboratory Exercise/Activities

Using fork() and CreateProcess() system calls

fork() is called by the root process to create another process called the child. The two processes execute independently without sharing any data. The following sample program illustrates the operation of the fork() system call.

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
int  main()
{
pid_t  pid;
}
/*  fork a  child  process  */
pid  =fork();
if  (pid <  0)  {  /*  error  occurred */
fprintf(stderr,  "Fork  Failed");
return  1;
}
else if  (pid ==  0)  {  /*  child  process  */
execlp("lbinlls","ls",NULL);
}
else {  /*  parent process  */
}
/*  parent will wait for  the  child  to  complete  */
wait (NULL)  ;
printf("Child Complete");
return  0;
}
```

Compile and run the above program. How many lines are printed by the program?

CreateProcess() does similar task as fork() in windows operating system. Their difference is fork() has the child process inheriting the address space of its parent while CreateProcess () requires loading a specified program into the address space of the child process at process creation. Furthermore fork() has no parameters it requires, while CreateProcess () expects no fewer than ten parameters including the application name to be executed, command line parameters and other security related attributes.

```
/* sample 2 */
/*using the CreateProcess() system call in windows*/
#include <stdio.h>
#include <windows.h>
int  main(VOID)
{
STARTUPINFO  si;
PROCESS_INFORMATION  pi;
}
// allocate memory
ZeroMemory(&si,  sizeof(si));
si.cb  =  sizeof(si);
ZeroMemory(&pi,  sizeof(pi));
// create  child  process
if  (!CreateProcess(NULL,  //use  command  line
"C:\\WINDOWS\\system32\\mspaint.exe", //command line
NULL, //don't inherit process handle
{
}
NULL,  //don't  inherit  thread handle
FALSE,  //disable  handle inheritance
0,  //no  creation  flags
NULL, //use parent's environment block
NULL,  // use  parent's  existing  directory
&si,
&pi))
fprintf(stderr,  "Create  Process  Failed");
return  -1;
//parent will wait for the child to complete
WaitForSingleObject(pi.hProcess, INFINITE);
```

```
printf("Child Complete");

// close handles

CloseHandle(pi.hProcess);

CloseHandle(pi.hThread);
```

After a process is being created and does what it needs to do, it will be brought to termination. The termination may arose from normal exit, where a process completes running its last instruction and requests the operating system to remove it through the exit() and exitProcess() system calls in UNIX and windows operating system respectively. In such cases, the child process may return a status value (typically an integer) to its parent process (via the wait() system call). The operating system then takes back all the resources allocated to the process such as the physical and virtual memory, open files, and I/O buffers. In some cases, a process may terminate due to error exit, fatal error exit or being killed by another process. If the termination is requested from another process, kill() for UNIX and TerminateProcess() in windows will be used.

Results and Submission Requirments

What are the outputs of the programs?

Assessment Criteria

The student has to properly answer the questions obtain total of 3 marks.

References or Key Links

- http://linux.about.com/od/commands/l/blcmdl.htmhttp://linux.about.com/od/commands/l/blcmdl.htm
- http://www.techytalk.info/linux-system-programming-open-file-read-file-and-write-file/

Details of the Laboratory Exercise/Activities

Objective of the Exercise

- Use the shell to execute UNIX's wait, exit, and waitpid system calls

Resources Required

- Personal computer installed with Ubuntu Operating System

Time Requried

- 30 Minutes

Description of the Laboratory Exercise/Activities

Process termination

A process terminates its execution when the system call exit() is invoked which takes the exit status value 0 or 21 as a parameter as exit(0), or exit(21). To check the termination status of its child processes, a parent process has to call wait() or waitpid()which returns the pid of the child process that terminated (or -1 if the process has no children), and wait() takes an integer pointer as an argument, called status. The "status" parameter of these two system calls encodes the child exit status (the number passed to the exit() system call in the child process). To learn about the exit status of a program we can use the macros from sys/wait.h which check the termination status and return the exit status.

WIFEXITED (status)gives a true output if the child termination was successful, that is, by calling exit(3) or _exit(2), or by returning from main().

WEXITSTATUS(status) returns the exit status of the child which has the least significant 8 bits of the status argument that the child passed while invoking exit(3) or _exit(2) or as the argument for a return statement in main(). This macro should only be employed if WIFEXITED returned true.

Add the correct form of wait() and exit() in the following program, so that the child terminates successfully and the parent waits for the child to finish execution displaying "I am …".

Compile and run the program to see the effects of these calls

```
/*sample 4*/
#include<unistd.h>
#include<sys/wait.h> /* wait is defined here */
#include<stdio.h>
#include<stdlib.h>  /* exit is defined here */
void main()
{
pid_t pid, child;
pid=fork();
if(pid==0){ /*code executed in child */
printf ( " I am child PID %d\n", getpid());
}
else {
printf ( " I am parent PID %d\n", getpid());
} }
```

Which process, the child or the parent, prints the statement "I am …" first? Why?

Results and Submission Requirments

What are the outputs of the programs?

Assessment Criteria

The student has to properly answer the questions obtain total of 3 marks.

Interactive processes

Interactive processes are processes which are started by someone connected to the system. They are initialized and controlled through a terminal session not automatically as part of the system functions. These processes can run in the foreground, occupying the terminal that started the program, and other applications cannot be started as long as this process is running in the foreground. Alternatively, the processes can also run in the background, so that the terminal in which you started the program can accept new commands while the program is running. In such a case, the user is also allowed to do other activities in the terminal from which he/she started the program, while it is running. The terminal provides a service called job control that enables simple management to several processes by moving amongst foreground and background processes. .With this service, it is also possible to start programs in the background immediately.

Automatic processes

Automatic or batch processes are processes that have no linkage to a terminal but are queued into a spooler area, where they wait to be executed on a FIFO (first-in, first-out) basis. Such tasks can be executed using one of two criteria:

At a certain date and time: done using the at command

At times when the total system load is low enough to entertain additional tasks: done using the batch command. By default, tasks are put in a queue where they wait to be executed until the system load is lower than 0.8. In large environments, the system administrator may prefer batch processing when large amounts of data have to be processed or when tasks demanding a lot of system resources have to be executed on an already loaded system. Batch processing is also used for optimizing system performance.

Daemons

Daemons are server processes that run continuously. Most of the time, they are initialized at system startup and then wait in the background until their service is required. A typical example is the networking daemon, xinetd, which is started in almost every boot procedure. After the system is booted, the network daemon just sits and waits until a client program, such as an FTP client, needs to connect.

Summary

A process is a program in execution which requires some resources granted by the operating system for its execution. Every process can be described by its address space, processor state and OS resources. A process is created on different events including user request and system initialization. A process can also create another process when it is simple to pass some jobs to the new process/es. UNIX forms a process group for processes with the same root parent forming a tree like hierarchical arrangement of the processes. Windows has, however, equal status for all processes and no parent-child relationship exists between processes.

There are different types of processes based on the application requirement. It can be interactive, automatic, or daemons.

Assessment

a. What distinguishes process from progam?

b. What three basic reasons cause process creation?

c. How does the system differentiate between processes?

d. How many processes are created when the below C code executes?

```
#include <stdio.h>
#include <unistd.h>
int main()
{
}
/* fork a child process */
fork();
/* fork another child process */
fork();
/* and fork another*/
fork();
return;
}
```

Activity 2 – Process states

Introduction

Starting from the execution of process, it passes through different states till its termination. What are those states? How is the transition from one state to another invoked?

The state of a process changes while the process is running. The state indicates what the process is currently doing. A process can assume any one of the following three states.

Ready: the process is ready to go but is waiting to get granted the CPU as some other process is using it.

Running: the process is actually using the CPU at the moment and is being executed.

Waiting: sleeping or blocked state where the process is waiting for an event. Even if the CPU is free to handle requests, a process at this state will not use it as it is waiting for some external events to occur.

Four possible moves can be made by a process between these states as it executes as shown in figure 2.3 below.

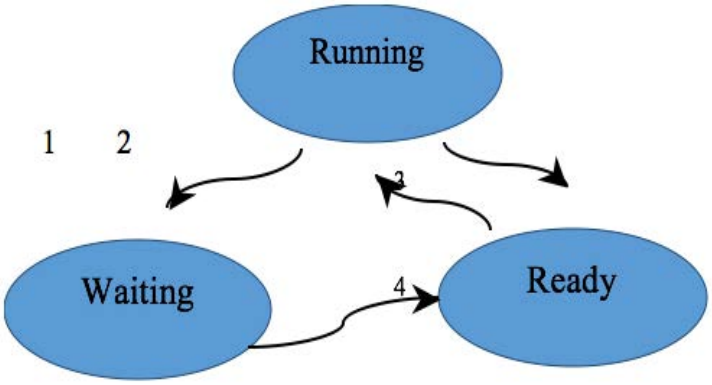


Figure 2.3. Process transitions between states

Transition 1 happens when a process has to wait for some event like I/O and can't continue further. Transition 2 and 3 happens when the process scheduler decides so without the knowledge of the process. In transition 2, the scheduler decides that the process has run long enough and it is time to let others use the CPU or an interrupt has occurred which signals the CPU to stop the current process and do some other activity. In transition 3, the scheduler decides the other processes have enough use of the CPU and it is this process's turn to use the CPU again. The last transition, transition 4, occurs when the process has got what it was waiting for and jumps immediately to ready state where it is moved to the running state, if the CPU is idle, or remains in the ready state until the CPU is idle. It is important to note that several processes can be in ready and waiting state while only one process is running at any instant of time.

A process's life cycle comprise of five basic states as new, ready, running, waiting and terminate phase.

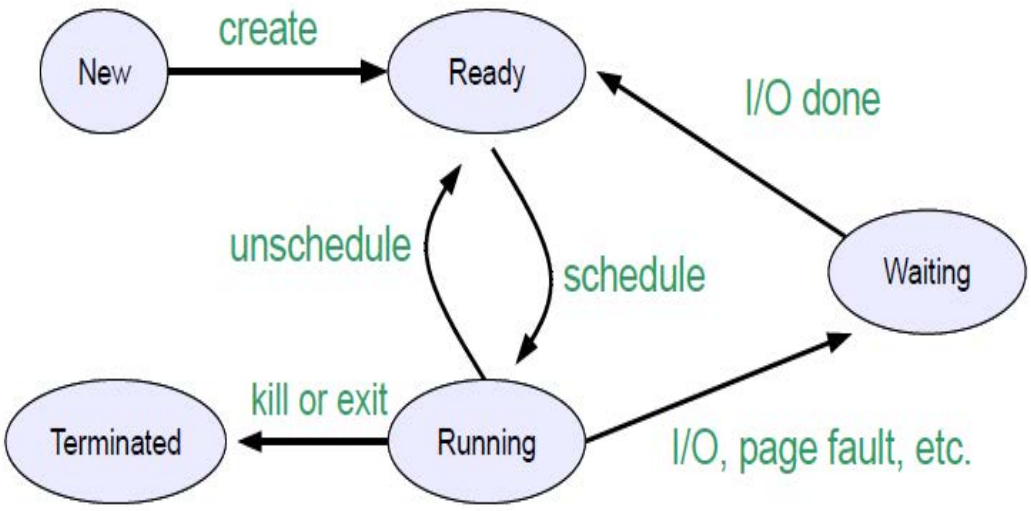


Figure 2.4. Life cycle of a process

The operating system represents each process by a task control block called Process Control Block (PCB). This block is a big data structure holding different important information associated with a process as shown in figure 3.5. below.

Process state
Process number, PID
Program counter
Registers
Memory limit
List of open files

Table 2.2: PCB Entries per process

Process state: indicate the current status of the process as new, running, waiting, ready.

Process number: a unique positive integer value associated with each process during creation and used as an identifier. The getpid() command is used to return the current process's number, PID. In order to read the parent's id, we can use the getppid() command.

Program counter: a value indicating the next instruction to be executed for the process

Registers: including accumulators, stack pointers, and the like maintains the current value produced when the process runs so that the process resumes from where it left off in case of interrupts.

Memory limit: holds the value of the base and limit registers, page table size etc.

Summary

After a process is created and before it is brought to termination, it assumes different states. A process state determines the current status of the process. There are three different states a process can be in. since only one process is allowed to use the CPU at a time, only that process is in the running state while zero or more processes can be either in the ready or waiting state. A process also moves back and forth between these three different states all the time except no transition is allowed from ready to waiting state. Every process has important information associated and these information are stored in a PCB by the operating system.

Assessment

a. What are the different possible states that a process can be in?

b. When does a process move from running to waiting/blocked/sleeping state?

c. Why is it not possible to move a process from ready to waiting state?

d. What is the reason for a process to stay in its ready state?

Activity 3 – Thread Management

Introduction

To perform several jobs at a time multiprogramming is a must. A program by itself cannot address several problems at a time. But, there is a mechanism through which a program can solve this problem. The answer is thread. What is thread? How does it work?

A process is a means through which interrelated resources such as the address space, child processes, opened files, etc. are grouped together. Threads are multiple execution streams within a single process and are used to separate the resource management task of the process from its actual execution. While processes are used to group resources together, threads are the objects scheduled for execution on the CPU. Thread can also be considered as lightweight process in that it does small portion of the entire task of a process. Having a thread adds an advantage to a process by enabling multiple independent executions in the same process environment by sharing the resources grouped together. The CPU moves back and forth among the threads running and creates an illusion that all of the threads are running in parallel. A process may create several threads (multi-threading) that share same address space and work together to accomplish a job in a sense they are not as such independent as two processes are. Threads are very important in today's programming to handle execution of several tasks of a single process independently of the others. This is mainly true when one of the tasks of the process may block, and it is required to let the remaining tasks run without blocking.

For instance, in a word processor, a thread running in the background may validate spelling and grammar while a foreground thread handles user's input, and yet another third thread fetches images from the local disk, and even a fourth thread is doing periodic automatic backups of the file being edited and so on. Like a single thread (one process), threads can be in any of the states a process can be in which we previously discussed.

In a multi-thread model, a process starts by creating only one thread first and the thread creating other threads through a create-thread library procedure call. The newly created thread runs automatically in the address space of the creating thread and the identifier of the newly created thread will be passed to the creating thread. The exit_thread library procedure is used to terminate a thread when it finishes its task. A thread can also wait for another thread with a call to the library procedure thread_wait. The CPU has no interrupt towards threads, as with processes, that forces them to surrender their usage of the CPU. However, threads use the thread_yield procedure call to willingly give up the CPU for other threads to use. This avoids the monopolized usage of the system by a single thread.

Multi-threading brings four main advantages to processes. These are:

- **Responsiveness** –a thread may give results quickly while other threads are waiting or slowed down performing some serious computations.
- Resource sharing - By default threads use common code, data, and other assets, that enables several jobs to be carried out concurrently in one address space.
- **Economy** – Creating, destroying and handling threads is much faster than performing the same operation for processes since no resource is attached to a thread
- **Scalability** – A process with one thread can only run on a single CPU, regardless of the number of CPUs available, while running a process with several threads may require distributing the threads t amongst available processors.

Despite these advantages, threads also bring a complication to processes. One problem is associated with the fork() system call. Assume a parent and child processes with multi-threading capability. What would happen if a thread in the parent is waiting for input? Does a thread in the child also get blocked? Who gets a typed input, only the parent? The child? Or both?

Another problem arose due to the common resources shared among threads. One thread might close or modify a particular resource which is currently being also used by another thread.

In general, since the advantage of multi-threading overweighs its problems, modern computer systems implement the concept of multi-threading but it requires a careful thought and design.

A system can implement and manage threads through two main ways namely user thread and kernel thread.

User space thread: the thread packages are placed in their entirety on the user space. These are the threads defined by application software developers during application creation with the kernel having no knowledge about them and it operates as if a single-threaded process is being managed. This is a good mechanism to implement threads on operating systems that does not support one.

The thread archive also has codes to be used for threads creation and deletion, for transferring messages and data amongst threads, for scheduling thread execution and also for storing and reinstating thread contexts. The application software starts with one thread and its execution begins in that thread. Each process defines a separate thread table which is then controlled by a run-time system to maintain information related to each thread in the process. Whenever a thread would like to move from one state to another, a run-time system procedure is called that checks the validity of the change and stores the current values in the thread's registers into the thread table so that the thread will resume later on from where it has stopped.

Invoking the run-time system procedures is way effective than performing a kernel call as no trap and no context switch is needed and also as it does not require to flash the memory cache making thread scheduling very fast. Such type of thread implementation also allows processes to have their own customized scheduling algorithm. Scalability is again another advantage of user space thread implementation as several threads can be defined and implemented easily. User space thread implementation has also its own drawbacks.

One problem is in relation to page fault. System can be set not to load a program in its entirety onto main memory at once. A page fault occurs when a process looks for instructions not loaded in memory. During such cases, the process is blocked until the operating system finds and reads the required instruction from disk. If a thread makes a page fault to happen, the kernel will block the entire process until the missing request is being processed as it is unaware of the multi-threaded implementation of the process and deprive other runnable threads from running.

The second problem associated with user space thread implementation is that a single thread can monopolize the time slice and make starve the other threads within the process unless it willingly returns the CPU. Since clock interruption is not found within a single process round-robin scheduling is impossible and also making it impossible for the scheduler to interrupt unless a thread enters the runtime system of its own free will. There are cases where absence of clock interrupts is very important like during synchronization. Especially, it is common in distributed systems for one thread to start a job that requires reply of another thread and then waits in a busy loop checking if the response has happened.

This situation is called a spin lock or busy waiting. A spin lock is important especially when the response is needed immediately and the cost of using semaphores is high. If threads are suspended automatically every few milliseconds using a clock interrupts, this approach works fine. However, if threads are allowed to execute until they block, this situation is an input for deadlock.

Figure 2.5 depicts user space thread implementation

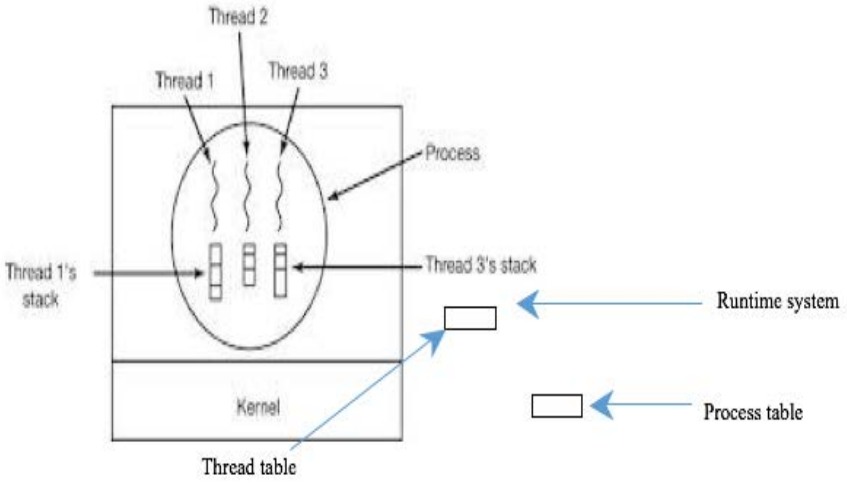


Figure 2.5. User space thread implementation

Kernel space thread: kernel is aware of available threads and manages the threads. All modern operating systems provide kernel level threads, enabling the kernel to carry out several concurrent jobs and/or to run multiple kernel system calls at a time. Rather than defining separate thread tables in every process, the kernel maintains a single thread table with similar attributes as the process's thread table for all the threads found in the system. The kernel also has a process table to maintain the existing processes. Whenever a thread is blocked through system calls, the kernel can run either a ready thread from same process or another process.

Thread creation and destroy has larger costs in this kernel mode. To minimize this cost, some systems implement thread recycling. In this case, a thread destroyed will be deactivated without affecting its data structures and when a new thread is created, the deactivated thread will be brought back to life. In Kernel threads there is no need to define new non-blocking system calls and if a thread has encountered a page fault, the kernel will check for another ready thread in a process and allow this thread to run until the page fault is resolved.

The main difference between a kernel thread and a user thread is the fact that the kernel thread has no user task memory mapping. Like other user processes, it shares the kernel address space but the thread operates only in kernel space with kernel privileges. Because kernel has full knowledge of all threads, Scheduler may allocate more execution time to a process with large number of threads than processes having small number of threads. Kernel-level threads are mainly useful for processes that frequently block. However, kernel-level threads are known to be slow and inefficient. Threads operations in kernel-level are hundreds times slower than that of user-level threads. Since kernel has to control and schedule threads as well as processes, the kernel needs to keep information of each thread through a full thread control block. As a result, there is substantial overhead, substantial cost to system calls and increased complexity in kernel.

Figure 2.6. Depicts the kernel level thread implementation

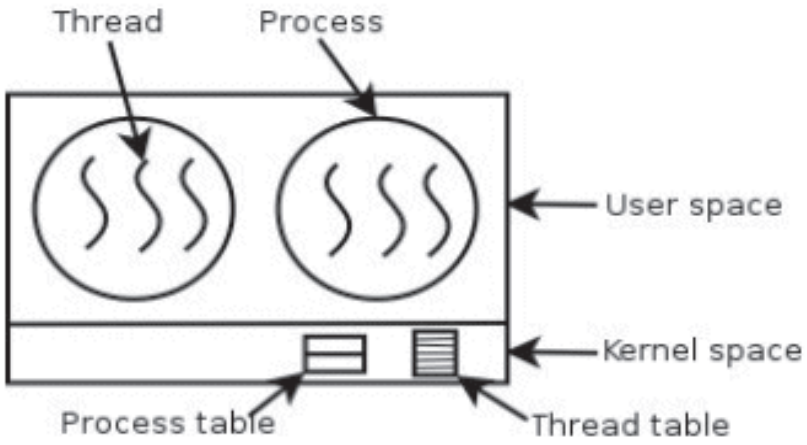


Figure 2.6. Kernel thread implementation

Hybrid thread implementation: combining the advantages of both the user space thread implementation and the kernel level thread implementation can be more effective than using them individually. Different techniques had been explored on how to bring together the benefits of user-level threads to that of kernel-level threads. Using kernel level threads and multiplexing user level threads onto all or some of the kernel threads is one way identified. In this method the kernel knows only about the kernel level threads, which might have zero or more multiplexed user-space threads that take turns using it, and handles scheduling of those threads.

Another approach is the so called scheduler activation that imitates the services of kernel level threads, but with improved efficiency and better flexibility usually related with thread packages implemented in user space. In this technique, the kernel allocates some amount of logical processors to every process and allows the user-space run-time system assign threads to processors. Initially, the kernel will assign a single virtual processor to each process but processes can also demand for additional and can also give back if no longer needed. It is also possible for the kernel to take back an already given virtual processors and re-allocate them to other, more needy, processes.

The kernel activates the user space run time system through an up call in order to notify about an event happening on threads. For instance, when the kernel is aware of a thread just moved to its blocked state, it will immediately notify the run time system by putting the code of the thread identified along with a report of the event happened on to the stack.

With this knowledge, the runtime system tries to reschedule the threads according to their states. A hardware interrupt may occur while a user thread is being executed where the interrupted virtual CPU moves to a kernel mode and manages the interrupt. If the interrupt occurred has no significance to the interrupted thread, it will be returned back in its state before the interruption. If, on the other hand, the thread has some interest towards the interrupt, it will be suspended and the runtime system will be up called with the state of this thread passed onto the stack. The runtime system then decides which thread to grant the CPU for: the interrupted thread, a new thread which is in its ready state, or some other third choice.

Thread Management

Pthreads are Linux implementation of thread management mechanism. Pthreads are defined as a set of C language programming types and procedure calls. Vendors usually provide a Pthreads implementation in the form of a header/include file and a library, which you link with your program. The two names used in thread management frequently are: pthread_t for thread objects and pthread_attr for thread attributes objects.

The pthread_create function is used to create a new thread, and pthread_exit function for terminating a thread by itself. A thread to wait for termination of another thread uses the function pthread_join.

Normally when a program starts up and becomes a process, it starts with a default thread which brings the conclusion that every process has at least one thread of control. A process can create extra threads using the following function :

```
#include <pthread.h>

int pthread_create(pthread_t *restrict tidp, const pthread_attr_t
*restrict attr, void *(*start_rtn)(void), void *restrict arg)
```

Where:

- The first argument is a pthread_t type address. Once the function is called successfully, the variable whose address is passed as first argument will hold the thread ID of the newly created thread.
- The second argument may contain certain attributes which we want the new thread to contain. It could be priority etc.
- The third argument is a function pointer. This is something to keep in mind that each thread starts with a function and that functions address is passed here as the third argument so that the kernel knows which function to start the thread from.
- As the function may accept some arguments also we can pass these arguments in form of a pointer to a void type, which is the fourth argument.

Laboratory Activity

Thread Management

Objectives of the Labortory

- Execute thread management system calls like pthread_create, pthread_join, pthread_exit, etc…
- Analyze the outputs of the system calls

Details of the Laboratory Exercise/Activities

Objective of the Exercise

- Use the shell to execute UNIX's pthread_create and pthread_join system calls

Resources Required

- Personal computer installed with Ubuntu Operating System

Time Requried

- 30 Minutes

Description of the Laboratory Exercise/Activities

Edit the following the source code using your preferable editor.

```
/* thread creation */
#include <stdio.h>
#include <pthread.h>
/* All thread functions and datatypes are defined in pthread.h */
void *kidfunc(void *p)
printf ("Kid ID is ---> %d\n", getpid( ));
}
void main(){
pthread_t kid; /* Declare a variable of type pthread_t :*/
pthread_create (&kid, NULL, kidfunc, NULL);
printf ("Parent ID is ---> %d\n", getpid());
pthread_join (kid, NULL);
printf ("No more kid!\n") ;
}
```

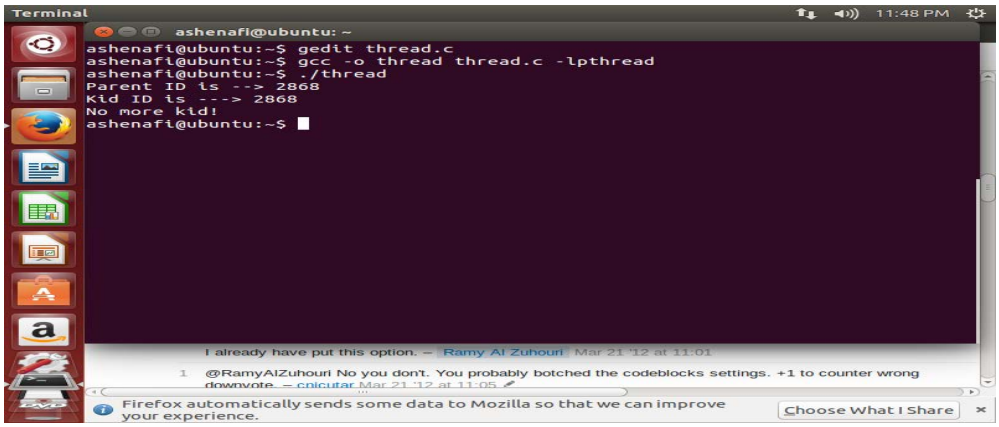
The compiler needs to link this program with pthreads library file. This can be done by adding -lpthread to the linker flags as:

gcc lab1.c –o threads -lpthread

Then run the program as:

. /threads

What is the output of the program?



How many threads are there in the program? Which thread executes which statement?

Are the process id numbers of the threads same or different? Why?

Which thread finishes first its gedit execution? Why?

Results and Submission Requirments

Answer the previous questions with appropriate explanation.

Assessment Criteria

The student has to properly answer the questions obtain total of 2 marks.

References or Key Links

- http://man7.org/linux/man-pages/man7/

Details of the Laboratory Exercise/Activities

Objective of the Exercise

- Use the shell to execute UNIX's pthread_equal and pthread_exit system calls

Resources Required

- Personal computer installed with Ubuntu Operating System

Time Requried

- 30 Minutes

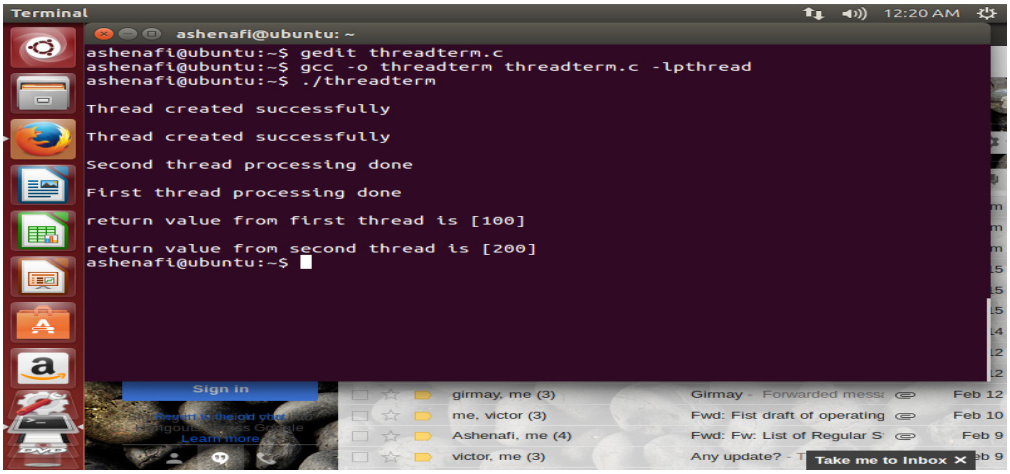
Description of the Laboratory Exercise/Activities

```
/* thread termination */
#include<stdio.h>
#include<string.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>
pthread_t tid[2];
int ret1,ret2;
void* doSomeThing(void *arg)
{
unsigned long i = 0;
pthread_t id = pthread_self();
for(i=0; i<(0xFFFFFFFF);i++);
if(pthread_equal(id,tid[0]))
    {
printf("\n First thread processing done\n");
        ret1  = 100;
        pthread_exit(&ret1);
    }
else
    {
printf("\n Second thread processing done\n");
        ret2  = 200;
        pthread_exit(&ret2);
    }
return NULL;
}
int main(void)
{
```

```
int i = 0;

int err;

int *ptr[2];

while(i < 2)

    {

err = pthread_create(&(tid[i]), NULL, &doSomeThing, NULL);

if (err != 0)

printf("\ncan't create thread :[%s]", strerror(err));

else

printf("\n Thread created successfully\n");

i++;

    }

    pthread_join(tid[0], (void**)&(ptr[0]));

    pthread_join(tid[1], (void**)&(ptr[1]));

printf("\n return value from first thread is [%d]\n", *ptr[0]);

printf("\n return value from second thread is [%d]\n", *ptr[1]);

return 0;

}
```

What activities are handled with this program?

What is the output of this program?



What would happen if the pthread_join function is removed?

Try to create and terminate a thread with WIN32 functions.

Results and Submission Requirments

Answer the previous questions with appropriate explanation.

Assessment Criteria

The student has to properly answer the questions obtain total of 2 marks.

References or Key Links

- http://man7.org/linux/man-pages/man7/http://man7.org/linux/man-pages/man7/
- http://man7.org/linux/man-pages/man7/

Summary

Thread is a basic unit of CPU utilization, consisting of a program counter, a stack, a set of registers, and a thread ID. A process can have one or more threads which share the same resources like the address space and opened files of a process. Threads differ from processes in that dependency exists between threads and also they work collaboratively to accomplish a task and also share same resources. Threads have also similarity with processes as they have a unique thread ID similar with PID, threads can also be in any of the process states, threads can create child threads, etc.

Assessment

1. What is the importance of using threads?
2. What are the main differences between user space and kernel threads?
3. What are the disadvantages of kernel level thread implementation?
4. Which problems of user space thread implementation are solved by using the scheduler activation thread implementation method?

Activity 4 Inter-Process Communication (IPC)

Introduction

In a multiprogramming environment several processes executes at a time. These processes compute to control resources. Processes need to communicate to each other for sharing resources. How does the operating system handle such inter-process communication?

Concurrency and Its Problems

A. Basic Concept

The most fundamental task of modern operating systems is management of multiple processes within uniprocessor, multiprocessor or distributed computer systems. The fundamental design issue in the management of multiple processes is concurrency: simultaneous execution of multiple processes

Concurrency arises in three different contexts

Multiple applications: concurrently running applications

Structured applications: an application structured as a set of concurrent processes (threads)

OS structure: OS implemented as a set of processes

Concurrency provides major benefits in processing efficiency and in program structuring

B. Inter-process Communication

There is frequent interaction among concurrently running processes. There are three ways in which concurrent processes interact with each other:

Competition for Resources

It occurs when independent processes that are not intended to work together compete for the use of the same or shared resource, e.g. printer, memory, or file

No communication or message exchange exists amongst such competing processes

Processes are unaware of the existence of each other

Cooperation by Sharing Resources

It occurs when processes that are not necessarily aware of each other usage and interaction to shared data without reference to other processes but suspects that other processes may have access to the same data

Processes need to collaborate and guarantee that the data they share are properly managed

Processes are aware of the existence of each other indirectly

Cooperation by Communication

It occurs when various processes communicate with each other, for instance with message passing in order to provide a way to synchronize or coordinate their various activities.

There is nothing shared between processes

Processes are aware of the existence of each other directly

C. Concurrency Problems

There are some serious problems associated with the interaction of concurrently running processes:

Race Condition

A situation that occurs when two or more processes are reading or writing into some shared data and the final result depends on who runs precisely when

E.g. **1 Printer Spooler:** a spooler folder is used by processes to maintain name of files the process needs to print. Assume that this spooler directory consists a large number of partitions, numbered 0,1,2,…

There are two globally shared variables

Outfile: points to the next file to be printed

Infile: points to the next free slot in the directory

There were some files in the spooler directory and assume the current value of infile is 7 and that of outfile is 3

Assume that simultaneously process A and process B decide they want to queue a file for printing

Process A reads infile and keepsthe result 7 in its local variable (x=infile)

An interrupt message has been sent to the CPU and it concluded that process A has executed long enough so it gives the priviedge to process B

Process B reads infile and stores the value 7 in its local variable (y=infile)

Process B keeps the label of its file in slot 7 and adjusts infile to be 8

Eventually process A runs again, it checks x and finds 7 there, and writes its file name in slot 7, erasing the name process B just put there, it updates infile to be 8

The printer daemon will now print the file of process A, process B file will never get any output

E.g. 2 Character echo procedure: consider the following globally shared procedure

Void echo

```
 {
chin=getchar(); //read a character from keyboard
chout=chin;
putchar (chout); //display the character on the screen
}
```

Consider two processes (P1 & P2) trying to access the procedure

Process P1 invokes the echo procedure and is interrupted immediately after the conclusion of getchar function (chin =x)

Process P2 is activated and invokes the echo procedure, which runs to conclusion, inputting an displaying a single character, y

Process P1 is resumed. By this time the value x has been overwritten in chin and therefore lost. Instead chin contains y, and is displayed twice

Deadlock

It is the permanent blocking of a set of processes that either compete for system resources or communicate with each other. It involves conflicting needs for resources by two or more processes.

It refers to a situation in which a set of two or more processes are waiting for other members of the set to complete an operation in order to proceed, but none of the members is able to proceed.

E.g. Traffic deadlock: consider a situation in which four cars have arrived at a four-way stop intersection at the same time. The four quadrants of the intersection are the resources over which control is needed. If all four cars proceed into the intersection, then each car controls one resource (one quadrant) but cannot proceed because the required second resource has already been controlled by another car. Hence deadlock will occur. The main reason for this deadlock is because each car needs exclusive use of both resources for certain period of time

It is a difficult phenomenon to anticipate and there are no easy general solutions to this problem.

We shall be discussing deadlock problem and handling mechanisms in the next unit.

Starvation

It referees to the situation in which a process is ready to execute but is continuously denied access to a processor in deference to other processes.

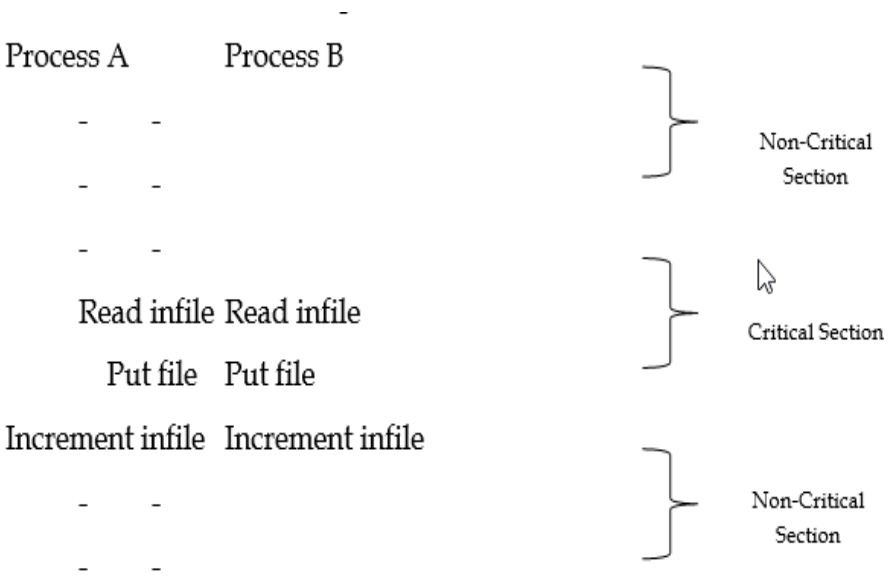
E.g. suppose that there are three processes P1, P2, and P3 and each require periodic access to resource R. If the operating system grants the resource to P1 and P2 alternately, P3 may indefinitely be denied access to the resource, thus starvation may occur.

In large part, it is a scheduling issue that shall be addressed in the next unit

D. Mutual Exclusion

The key to preventing race condition is to enforce mutual exclusion: It is the ability to exclude (prohibit) all other processes from using a shared variable or file while one process is using it.

Part of a program where shared resource (critical resource) is accessed is called critical *region* or *critical* section



The fundamental strategy for implementing mutual exclusion is avoiding the situation where no two processes could ever be in their critical regions at the same time.

There are some basic requirements that should be met while implementing mutual exclusion

i. No two processes may be simultaneously in their critical regions

ii. No assumptions may be made about speed or the number of processors

iii. No process running outside its critical region may block other processes

iv. No process should have to wait forever to enter its critical region

Implementing Mutual Exclusion with Busy Waiting

In this section, various proposals for achieving mutual exclusion with the help of busy waiting are examined:

A. Disabling Interrupts

An interrupt is an event that alters the sequence in which a process executes instruction

In this technique, each process disables all interrupts just after entering its critical section and re-enables them just before leaving it. With interrupts turned off the processor will not be switched to another process

Disadvantages

1. Processes shall not be given the ability to close or shut down interrupts. For instance, a process can turn off interrupts but never turn them on again in which the whole system freezes
2. If the system is multiprocessor, disabling interrupts affects only the processor that executed the disable instruction. The other ones will continue running and can access the shared memory.

3. Disabling interrupts is often a useful technique within the operating system itself but is not appropriate as a general mutual exclusion mechanism for user processes.

B. Lock Variables

Assume we have a single shared (lock) variable initially set to 0

A process enters its critical section if this variable is 0, when it enters it sets it to 1

If the variable has a value 1, the process needs to wait until the value changes to 0

Thus a 0 indicates that no process is in its critical region, and a 1 indicates some process is in its critical region

Disadvantages

Assume a process observes that the lock is set to 0.but just before this process changes the lock to 1, another process is scheduled, runs and changes the lock to 1. The first process then gets a chance to execute again, it will also set the lock to 1 making both processes to be in their critical region simultaneously causing race condition

Continuously testing a variable waiting for some value to appear is called busy waiting. This technique wastes processor time

C. Strict Alternation

Strict alternation is shown in the program fragment below for two processes, process 0 and process 1. This solution requires that the two processes strictly alternate in entering their critical regions.

```
while (TRUE) {

while(turn!=0) /*wait*/;

critical_region();

turn=1;

noncritical_region();

}

while (TRUE) {

while(turn!=1) /*wait*/;

critical_region();

turn=0;

noncritical_region();

}
```

The integer turn, initially 0, keeps track of whose turn it is to enter the critical region and examines or updates the shared memory. Initially, process 0 inspects turn, finds it to be 0, and enters its critical region. Process 1 also finds to be 0 and therefore sits in a tight loop continually testing turn to see when it becomes 1. When the first process, process 0, exits the critical region, it puts turn to 1, so that process 1 can be in its critical region.

Disadvantages

If processes are not of equal efficiency and one is much slower than the other, taking turns is not a good idea as it violates condition 3 of implementing mutual exclusions: process 0 is being blocked by a process not in its critical region.

D. Peterson's Solution

It is a software solution. It combines turn takings with lock variables and warning variables. It does not require strict alternation. Before using the shared variable, i.e. before entering its critical region, each process calls enter_region procedure with its own process number, 0 or 1 as a parameter. This call will cause it to wait, if need be, until it is safe to enter. After it has finished with the shared variables, the process calls leave_region procedure to indicate that it is done and to allow the other process to enter, if it so desires. The code is shown below:

```
#define FALSE 0

#define TRUE 1

#define N 2    /* number of processes*/

int turn; /*whose turn is it?*/

int interested[N]; /*all values initially 0 (FALSE)*/

void enter_region (int process)  /*process is 0 or 1*/

{

int other; /*number of the other process*/

other = 1-process; /*the opposite of the process*/

interested[process] = TRUE; /*show that you are interested*/

turn = process; /* set flag*/

while (turn ==process && interested[other]!=TRUE); /*null statement*/

}

void leave_region (int process) /*leaving process*/

{

interested[process] = FALSE;

}
```

At the start, both processes are not in their critical section. Now process 0 invokes enter_region. It shows its need by setting its array element & sets turn to 0. Since enter_region is not called by any other process (process 1), it responds immediately. If process 1 now invokes enter_region, it will wait there until interested process[0] goes to false, a situation that occurs only when process 0 instantiates leave_region to leave the critical region.

Next assume a situation where both processes envoke enter_region almost simultaneously. Both will put their process number in turn and the store performed last is the one to exist in turn as the first one will be overridden and lost. Suppose that process 1 stores last, so turn is 1. When both processes come to the while statement, process 0 executes it zero times and enters its critical section. Process 1 loops and does not enter its critical section.

E. TSL Instruction

This technique requires a little help from the hardware. It uses the hardware instruction TSL.

TSL (Test and Set Lock) is an indivisible atomic instruction that copies the content of a memory location into a register and stores a non-zero value at the memory location. The operation of reading the word and storing into it are guaranteed to be indivisible, i.e. no other processor can access the memory word until the instruction is finished. The processor executing the TSL instruction locks the memory bus to prohibit other processors from accessing the memory until it is done.

To implement mutual exclusion with TSL instruction, a shared variable, lock, is used and it monitors access to shared memory. When lock is 0 the process may set it to 1 using TSL instruction and executes its critical section. When finished, the process assigns back lock to 0.

```
enter_region:
        TSL register, lock ; copy lock to register and set lock to 1
        CMP register, #0 ; was lock 0?
        JNE enter region; if it was non zero, lock was set, so loop
        RET                ; return to caller, critical region entered
leave_region:
        MOV lock, #0       ; store 0 in lock
        RET                ; return to caller
```

enter_region has busy waiting until the lock is free and needs to be invoked by a process which requires to enter to its critical region, When the lock is free, the enter_region gets the lock and returns this to the caller process making it enter its critical region. After the critical region the process invokes the leave_region method, to set lock to 0.

Implementing Mutual Exclusion without Busy Waiting

Both Peterson's solution and the solution using TSL are correct, but both have the defect of requiring busy waiting which wastes CPU time and which can also have unexpected effects, like the priority inversion problem.

Priority inversion problem: Let's assume a computer system having two processes, H with higher precedence and L with low priority. The scheduling rules are such that H runs whenever it is in the ready state. At a certain moment, with L in its critical region, H becomes ready to run. H now begins busy waiting, but since L is never scheduled while H is running, L never gets the chance to leave its critical region, so H loops forever.

Let's move our discussion to some process communication primitives that blocks when they can't enter their critical region instead of wasting CPU time Sleep: It is a system call that causes the caller to block, i.e. be suspended until another process wakes it up.

Sleep: It is a system call that causes the caller to block, i.e. be suspended until another process wakes it up.

Wakeup: It is a system call that causes the process specified by the parameter to wake up.

As an example of how these primitives can be used let us consider the producer-consumer problem (also known as the bounded buffer problem)

Producer-consumer problem

Two processes share a common fixed-size buffer. One of, the producers, puts information in the buffer, and the other one, the consumer, takes it out.

When the producer wants to put a new item in the buffer, it checks the buffer, if it is full, it goes to sleep, to be awakened when the consumer has removed one or more items.

When the consumer wants to remove an item from the buffer and sees that the buffer is empty, it goes to sleep until the producer puts something in the buffer and wakes it up.

Let us see the producer-consumer problem using c programming

```
#define N 100  /* number of slots in the buffer*/
int count =0;  /*number of items in the buffer*/
void producer ()
{
while (TRUE)
{ /*repeat forever*/
item=producer_item(); /*generate next item*/
if(count==N) sleep(); /*if buffer is full, go to sleep*/
```

```
insert_item(item); /*put item in buffer*/
count=count+1; /*increment count of items in buffer*/
if(count==1) wakeup(consumer);    /*was buffer empty?*/
}
}
void consumer ()
{
while (TRUE)
{
if(count==0) sleep(); /*if buffer is empty, go to sleep*/
item=remove_item(); /*take item out of buffer*/
count=count-1; /*decrement count of items in buffer*/
if(count==N-1) wakeup(producer); /*was buffer full?*/
consume_item(item);  /*print item*/
}
}
```

Problem:

Race condition can occur because access to count is unconstrained. Consider the following situation. The buffer is empty and the consumer has just read count to see if it is 0. At that instant, the scheduler decides to stop running the consumer temporarily and start running the producer. The producer enters an item in the buffer, increments count, and notices that it is now 1. Reasoning the count was just 0, and thus the consumer must be sleeping, and the producer calls wakeup to wake the consumer up. Unfortunately, the consumer is not yet logically asleep, so the wakeup signal is lost. When the consumer next runs, it will test the value of count it previously read, find it to be 0, and go to sleep. Sooner or later the producer will fill up the buffer and also go to sleep. Both will sleep forever.

The problem arises because the wakeup signal is lost. A quick fix is to add to the rules a wakeup-waiting bit which is a piggy store for wakeup signals

The bit will be set during a wakeup signal submission for an already awake process. And later, if the process attempts to go to sleep while the wakeup-waiting bit is on, the bit will be turned off, but the process will stay awake.

The wakeup waiting bit cannot be a general solution, especially for any random number of processes.

A. Semaphores

Semaphores solve the lost-wakeup problem

A semaphore is a new integer variable type that counts the number of wakeups saved for future use. A semaphore could have the value 0, indicating that no wakeups were saved or some positive value if one or more wakeups were pending.

Two operations were proposed to implement semaphores: up and down

DOWN operation

It checks the value of the semaphore to see if the value is greater than 0. If so it decrements the value and just continues. If the value is 0, the process is put to sleep without completing the DOWN operation for the moment

Checking the value, changing it and going to sleep is all done as a single, indivisible atomic operation. after a semaphore has started its operation, other processes can't communicate with it unless it comes to completion or the operation is blocked

UP operation

It increments the value of the semaphore. If one or more processes were sleeping on that semaphore, unable to complete an earlier DOWN operation, one of them is chosen by the system and is allowed to complete its DOWN operation

The process of increasing the semaphore and waking up a process is also indivisible.

Semantics of DOWN and UP operations

Void DOWN(s:semaphore)

```
{
if(s==0) sleep();
s=s-1;
}
Void UP(s:semaphore)
{
s=s+1;
wakeup a sleeping process if any;
}
```

Semaphores can be implemented in either of the two ways:

As system calls. The operating system disables all interrupts and performs semaphore testing, update the semaphore and make the semaphore asleep, if required

With lock variables: in a multi-processor system, lock variables can be used to protect each semaphore and ensure only one CPU is evaluating a semaphore through the TSL instruction.

Solving the Producer – Consumer problem using semaphores

```
#define N 100 /*number of slots in the buffer*/
typedef int semaphore; /*semaphores are a special kind of int*/
semaphore mutex =1; /*controls access to critical section*/
semaphore empty =N; /*counts empty buffer slots*/
semaphore full =0; /*counts full buffer slots*/
void producer ()
{
int item;
while (TRUE)  {
item=produce_item(&item); /*generates something to put in buffer*/
down(&empty); /*decrements empty counts*/
down(&mutex); /*enter critical section*/
insert_item(item); /*enter new item in buffer*/
up(&mutex); /*leave critical section*/
up(&full);  /*increment count of full slots*/ }
}
void consumer ()
{
int item;
while (TRUE)  {
down(&full); /*decrement full count*/
down(&mutex); /*enter critical section*/
item=remove_item(&item); /*take item from buffer*/
up(&mutex); /*leave critical section*/
up(&empty); /*increment count of empty slots*/
consumer_item(item); /*do something with the item*/  }
}
```

The solution uses three semaphores: full, to count the number of full slots, empty, to count the number of empty slots and mutex, to ensure both the producer and the consumer are not accessing the buffer simultaneously

Mutex is used for mutual exclusion, i.e. it is developed in a way that ensures read and write operations can be performed on a buffer and associated variables only by one process at a time Full/empty are used for synchronization, i.e. they are designed to guarantee that certain event sequences do or do not occur. When the buffer is observed to be full, the producer stops running , and the consumer does the same when it observes the buffer is empty

Binary semaphores are semaphores initialized to 1 and used by processes to guarantee only one of them can be in their critical region at the same time.

Problems

Semaphores are too low-level and error prone. If you are not careful when using them, errors like race condition, deadlocks and also other unexpected and irreproducible behaviors can occur.Suppose that the two downs in the producer's code were interchanged or reversed in order and suppose also the buffer was full and mutex is 1

down (&mutex);

down (&empty);

The producer does a down on mutex and mutex becomes 0 and then the producer does a down on empty. The producer would block since the buffer is full. Next time the consumer does a down on mutex and it blocks since mutex is 0. Therefore both processes would block forever and hence deadlock would occur.

B. Monitors

A monitor is a higher level of synchronization primitive proposed by Hoare and Branch Hansen to make writing a correct program easier.

A monitor is a collection of procedures, variables and data structures that are all grouped together in a special kind of module or package.

Rules associated with monitors

While it is possible for processes to invoke procedures of a monitor, direct interaction to the internal data structures of a monitor is not allowed to them.

Only one procedure can be active in a monitor at any instant. Monitors are programming language construct, so the compiler knows that they are special and can handle calls to a monitor procedures differently from other procedure calls

It is the compiler that implements mutual exclusion. it is not important for the monitor programmer to know the mutual exclusion arrangement made by the compiler. It is enough to know that using monitor procedures and replacing critical regions of processes ensures no two processes will ever execute their critical regions at the same time.

Monitors use WAIT and SIGNAL operations on condition variables, to block and wake up a process.

WAIT

When a monitor procedure knows it cannot go on, it does a WAIT on some condition variable that causes the calling procedure to block. It allows another process that had been previously prohibited from entering the monitor to enter now.

SIGNAL

A process can wake up its sleeping partner by doing a SIGNAL on the condition variable that its partner is waiting on. A process doing a SIGNAL statement must exit the monitor immediately, i.e. SIGNAL will be the final statement in a monitor procedure to eliminate the concurrent presence of two active processes in a monitor. Running a SIGNAL operation on condition variable which is being waited by more than one process revives only one of these processes based on the system scheduler's decision.

Condition variables are not counters. They do not accumulate signals for later use, unlike semaphores. If a condition variable is signaled with no one waiting on it, the signal is lost.

The WAIT must always come before the SIGNAL

An outline of the producer-consumer problem with monitors is shown below

Monitor ProducerConsumer

```
Condition full, empty;

Integer count;

Procedure enter;

Begin

If count=N then wait(full);

Enter_item;

Count=count+1;

If count =1 then Signal(empty);

End;

Procedure remove;

Begin

If count=0 then wait(empty);

remove_item;

Count=count-1;

If count =N-1 then Signal(full);
```

```
End;
Count=0;
End monitor;
Procedure producer;
Begin;
While true do
Begin
Produce_item;
ProducerConsumer.enter;
End;
End;
Procedure consumer;
Begin;
While true do
Begin
ProducerConsumer.remove;
Produce_item;
End;
End;
```

WAIT and SIGNAL operations resemble SLEEP and WAKEUP operations respectively but with one critical difference. SLEEP and WAKEUP failed because while one process was trying to go to sleep, the other one was trying to wake it up. With monitors, this cannot happen.

The automatic mutual exclusion on monitor procedures guarantees that if, say, the procedure inside a monitor discovers that the buffer is full, it will be able to complete the WAIT operation without having to worry about the possibility that the scheduler may switch to the consumer just before the WAIT completes.

Advantage of monitors

By making mutual exclusion of critical regions automatic, monitors make parallel programming much less error prone than with semaphores

Drawback of monitors

A computer language with pre-defined monitors is required and languages that have built-in monitors are rare. But adding semaphores in C, C++ and other languages is easy.

Drawback of monitors and semaphores

The initial objective was to avoid mutual exclusion situations in a single or multiprocessor system sharing same main memory

In a distributed system consisting of multiple CPUs, each with its own private memory, connected by a local area network, these primitives become inapplicable

Monitors are unusable in most programming languages due to their requirement and semaphores are too primitive.

And none of them also facilitate information exchange between systems.

B. Message Passing

This technique uses t the SEND and RECEIVE methods

SEND and RECEIVE are system calls and they can be put as library procedures: SEND(dest, &msg), RECEIVE(source, &msg)

If no message is available, the receiver could block until one arrives or could return with an error code

The producer-consumer problem with message passing

An outline for the solution of producer-consumer problem with message passing is shown below

```
#define N 100
void producer()
{
int item;
message m;
while (TRUE)  {
produce_item(&item);
receive(consumer, &m);
build_message(&m, item);
send(consumer, &m);    }
}
void consumer()
{
int item, i;
```

```
message m;
for(I=0;I<N;I++) send (producer, &m);
while(TRUE)   {
receive(producer, &m);
extract_item(&m, &item);
send(producer, &m);
consume_item(item);
}
}
```

All messages are of the same size

Messages sent are buffered automatically by the operating system

The consumer starts out by sending N empty messages

Whenever the producer has an item to give to the consumer, it takes an empty message and sends back a full one

If the producer works faster

the message buffer will be full and all messages will end up waiting for the consumer

The producer will be blocked, waiting for an empty messages to comeback

If the consumer works faster

the message buffer will be consumed quickly waiting for the producer to fill them up

The consumer will be blocked, waiting for a full message

Message passing variants (How messages are addressed?)

Using process address

Assigning each process a unique address and have messages be addressed to processes

Using mailbox

A mailbox is a space to keep some amount of messages, as indicated during the mailbox definition

The address parameters in SEND and RECEIVE calls are mailboxes not processes

The producer sends packet filled with a data to the consumer's mailbox and the consumer sends empty messages to the producer's mailbox

Rendezvous (Eliminate buffering)

SEND is done before RECEIVE

SEND won't run unless RECEIVE executes enabling direct transfer of message from sender to the receiver and avoiding the need to have an intermediate buffering process

If ,on the other hand, the receive is executed first, the receiver will be blocked until a send happens

It is easier to implement

It has minimal flexibility as sender and receiver are forced to run in lock stop

Design issues for message passing

Messages can be lost. The solution for this problem is using acknowledgement message

Acknowledgement message can be lost. this problem can be overcomed by placing successive sequence numbers within each original message

Ambiguous process names. The solution for this problem to use naming conventions

Authentication

Performance, especially if sender and receiver processes are on the same machine

Conclusion

Concurrently running processes compute for resources. These processes might communicate directly through message or share common variable on the memory. Concurrency have associated problems, namely race condition, starvation and deadlock. To avoid race condition, mutual exclusion is a solution. This unit addressed implementation of mutual exclusion using number of techniques from busy waiting and non-busy waiting approaches.

Then, how do we solve the problem of starvation and deadlock?

Assessment

1. Discuss the problems of concurrency.
2. What kind of solution is test-and-set lock (TSL) provide?
3. What is the difference between semaphores and monitors?
4. How does deadlock happen in a system?

Activity 5 – Process Scheduling

Introduction

Starvation is one of the problems of concurrency. In a multiprogramming environment, processes compute for the uniprocessor. How does the operating system executes these processes in turn with minimal starvation?

Overview

Kernel is one part of an operating system that carries out several activities and operations among which process management is one. The process management task includes handling process creation, termination, inter-process communication, scheduling, switching, dispatching and management of process control blocks. Scheduling is a means through which fair distribution of execution time is granted to processes with similar state for a multiprogramming system. Multiprogramming aims to have some process running at all times and maximize CPU utilization by switching the CPU among processes so frequently that users can interact with each program while it is running.

The two main objectives of the process scheduling are to keep the CPU busy at all times and to deliver "acceptable" response times for all programs, particularly for interactive ones. The process scheduler must meet these objectives by implementing suitable policies for swapping processes in and out of the CPU. The idea behind multiprogramming is to execute a process until it changes its state to waiting, typically for the completion of some I/O request and use this time productively rather than making the CPU idle until the process comes back to its ready state again. To accomplish this, several processes are kept in memory at one time and when one process has to wait, the operating system takes the CPU away from that process and grants to another process. This pattern continues.

However, there is a frequent chance for more than one process to be in a ready state at the same time which forces them to compete for the CPU. The system should then make a wise choice as to which process to run next. Component of the operating system with the responsibility of choosing a process to run is known as scheduler and the steps it follows scheduling algorithm. The task of the scheduler is, thus, to decide on a policy about which process to be dispatched, select a process according to the scheduling algorithm chosen and load the process state or dispatches the process for execution. Scheduling is a means by which process migration between the three different queues maintained by a multiprogramming system can be achieved.

These three different queues of processes are:

- Job queue: set of all processes found in the system.
- Ready queue: set of all processes residing in main memory, ready and waiting to execute.
- Device queue: set of processes waiting for a particular I/O device.

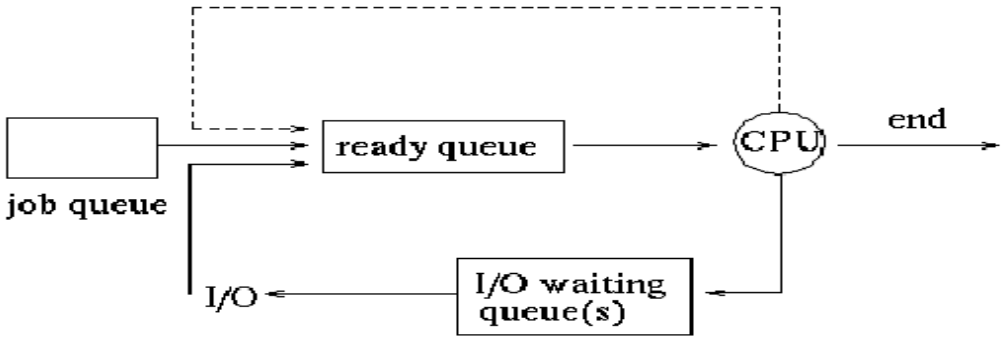


Figure2.7. The three processes queue and CPU interaction among them

There are two kinds of schedulers in a system: Long-term scheduler (job scheduler) and Short-term scheduler (CPU scheduler).

Long-term scheduler: selects processes from job queue and loads the selected process into memory for execution and then updates the ready queue. This scheduler also manages the number of processes in memory that entitles it in control of the multiprogramming degree within a system. A good mix of CPU-bound and I/O bound processes should be maintained in memory by this scheduler to make the system more efficient. This scheduler is invoked very infrequently and thus can be slower when it comes to performance. Such type of schedulers may not be present in some systems like time sharing ones.

Short-term scheduler: Decides which process to execute next and allocates the CPU. It picks processes found in the ready queue and dispatches them for execution by the CPU. This scheduler is also known as the dispatcher and invoked very frequently which makes it to be extremely fast. As the CPU is one of the primary computer resources, its scheduling is central to operating system design.

Scheduling is fundamental to operating system function. Almost all computer resources are scheduled before use. The scheduler will not have that much of a task on simple multi-programmed PCs as only one program is actually running currently and the user also waits for the completion of all programs. When it comes to high-end networked workstations and servers, the scheduler has a real job here and scheduling algorithms used significantly affects the efficient use of the system. When does a system need to schedule is a question that should be addressed before going deep into the scheduling process. Four different events bring the need for scheduling. These are:

State change from running to waiting. The current process goes from the running to the waiting state because it issues an I/O request or some operating system request that cannot be satisfied immediately.

Termination of the current process

State change from running to ready. For instance, a timer interrupt causes the scheduler to run and decide that a process has run for its allotted interval of time and itis time to move it from the running to the ready stateState change from waiting to ready. For instance an I/O operation is complete for a process that requested it and the process now moves from the waiting to the ready state. The scheduler may then decide to preempt the currently-running process and move this ready process into the running state.

Some processes spend more time in a waiting state as they have more I/O requests while some processes spend more time in their running state. The period of computation between I/O requests of a process is called the CPU burst. In general, processes can be described as I/O bound or CPU bound. An I/O-bound process is one that spends more of its time doing and waiting for I/O than it spends doing computations. Such type of processes generally experience short CPU bursts. A CPU-bound process, in contrast, generates I/O requests infrequently, using more of its time doing computations and exhibit long CPU bursts. It is important that the job scheduler selects a good process mix of I/O-bound and CPU-bound processes for best performance.

There are two types of process scheduling policies that can be used by the scheduler. These are: pre-emptive scheduling algorithm and non-pre-emptive scheduling algorithm.

Non Preemptive scheduling

If the scheduler cannot take the CPU away from a process, it is referred as non-pre-emptive or cooperative scheduler. Such types of scheduling run a process to completion and there will not be any scheduling choice to be made. The process itself releases the CPU for other processes willingly by terminating or by moving to a waiting state. In both cases, a software interrupt is issued which in turn switches the CPU over to start running the scheduler. The scheduler can now decide the next process to be run on the CPU. A non-pre-emptive or cooperative scheduling is a way in which processes will not be asked to give up the CPU unless they terminate or change their state to waiting. Older versions of Macintosh operating system and Windows 3.x are examples that had used the non-pre-emptive scheduling method.

Pre-emptive scheduling

A preemptive scheduler has the ability to get invoked by an interrupt and move a process out of a running state and let another process run. a special hardware device called an interval timer is designed to produce a hardware interrupt on a periodical basis (known as a time slice) and when this interrupt is received by the CPU, it will switch over to start running the scheduler, which will in turn decide on a new process to be run on the CPU. Such types of scheduling algorithms make interruption to distribute the CPU fairly among processes. A process is allowed to run only for some fixed amount of time and when the time is over, another process from the ready list will be picked and granted the CPU.

Pre-emptive scheduling has two main drawbacks, the extra cost incurred while accessing shared data and the complexity it adds on the kernel of the operating system.Note that in both cases, the scheduler will be activated upon reception of an interrupt. For non preemptive scheduling, the interrupt is a software interrupt issued by the process, whereas in preemptive scheduling, the interrupt is a hardware interrupt issued by an external hardware device or it can also be a software interrupt (through an I/O operation) issued by a process which will activate the scheduler in the same way as in non-preemptive scheduling.

Context switch and the interrupt handler

Whenever the CPU switches from executing one process to executing another process, a context switch is said to take place. Context switching requires saving of information for the processes so that execution can recommence at the right place. For example, assume you are arguing with your friend about the benefits of learning Operating System instead of Computer Maintenance (or vice versa, of course!) and you are rudely interrupted by another friend who insists on having an urgent discussion about a project. After you have finished your discussion about the project, you return to your first friend. At this stage, in order to be able to resume the conversation at exactly the point you left off at, you will need to remember what you were talking about, the stage of the argument you were at and all the points that you have presented so far. If you had not previously 'saved' this information, you would have to restart the conversation from scratch, hardly an appealing consideration for your friend. This same analogy should be performed by the CPU every time one process' execution is interrupted and another process gets the privilege.

Assume that the CPU is currently executing instruction 4 of process A, and for some reason, a switch is made to process B. If execution is to be resumed at A, then we will need to know the next instruction to be executed (presumably instruction 5). Thus the contents of the PC and other registers as well must be saved in the PCB for process A. After all the saving is done, the CPU is now ready to execute process B. In order to do this, it will need to know the next instruction to execute in process B and store the address of this instruction in the PC which can be found in the PCB of process B. In effect, during a single context switch, we need to first save the context of the current process that is going to be suspended and then restore the context of the new process that the CPU is going to switch to. This entire context saving and restoring is done by a special program called the interrupt handler that is part of the process management component of the OS. Saving and restoring information by the interrupt handler is a time consuming operation and represents wasted time as far as the CPU is concerned. This is because the CPU is only considered to be doing something useful when it is executing instructions that belong to a user process (either process A or B in the previous example). During a context switch, the CPU is actually executing the interrupt handler which handles the saving and restoring of information contained in the individual PCBs. The more context switches there are between processes, the more time is wasted running the interrupt handler (instead of running the processes) and the less efficient the CPU will be. This is an important consideration especially when it comes to preemptive scheduling.

Scheduling criteria

Different schedulers aim to achieve different goals like minimizing latency, maximizing throughput or maximize process utilization, etc. Some properties are needed by all systems while some depend on the implementation environment due to a variant in the specific requirements of the various application areas as well as different operating systems. In a batch operating system, for instance, time is an insignificant attribute to be considered by the scheduler as users are not looking for instant response from the system. As a result non-preemptive or preemptive with long time for each process scheduling algorithms can be used by the scheduler enhancing the system's performance through less process switching. On the other hand, in a real time environment, where processes are aware of giving up the CPU so quickly to one another, light preemptive scheduling algorithm is enough. For interactive systems, however, restricted preemptive scheduling algorithm is needed to avoid depriving CPU among processes. Some of the common criteria different schedulers considering a scheduling algorithm are the following.

Fairness: all processes should be able to use the CPU fairly, especially comparable processes. Fairness is a way in which the scheduler tries to allocate CPU for comparable processes at comparable rate so that equitable access to resources is granted. Sometimes, however, fairness is at odds with other scheduling criteria. For instance, throughput and response time of a system can be enhanced by making it less fair.

System utilization: the CPU as well as I/O devices of the system should always be kept busy. Keeping them execute all the time implies more operation performed per second. The scheduler needs to keep a mixture of CPU and I/O bound processes in memory to avoid idleness in all components of the system. In real systems, this ranges from 40% to 90% utilization

Throughput: is the amount of tasks completed by a system per time unit. Ideally, a system is required to finish as many processes as possible though the actual number depends upon the lengths of the processes. Throughput should be maximized by minimizing overheads and efficiently using systems' resources.

Turnaround time: The time gap between arrivals of a process to its completion. It is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/0. It predicts for how long an average user needs to wait for a response from the system. Generally, a process is expected to take minimal time from its creation to termination.

Response time: is the time elapsed to get the result of a submitted command. It is the amount of time it takes from when a request was submitted until the first response is produced, not output. This is very crucial especially for interactive and real time systems.

Ideally, a scheduling algorithm needs to maximize resource utilization and through put while minimizing turnaround time and response time.

Scheduling Algorithms

First-Come First-Served (FCFS) Scheduling

This is the simplest non- pre-emptive scheduling algorithm whereby the CPU executes processes that arrived earliest. In this algorithm, processes are dispatched according to their arrival time. There is one FIFO queue for ready state processes. When the first process comes to the system, it will be started immediately and run till it finishes all its jobs or till it needs to. If other jobs have arrived during execution of the first process, they will be lined up in the ready queue and will get a chance to run sequentially when the first process terminates or blocks. When a blocked process changes its state to ready, it will be treated like a new one and put at the end of the queue. The main advantage of this algorithm is its simplicity to implement.

A single linked list needs to be created and every time the system is free, the front child will be removed from the list and every time new process is added to the ready state or a blocked process has moved to its ready state, a new node is added at the end of the list. However, FCFS scheduling comes with a severe problem which is poor waiting and turnaround time for processes. Consider three processes P1, P2, and P3 submitted to the ready queue in this sequence and assume the processes have a burst time of 24, 3, and 3 respectively. Process P1 will run automatically as there is no other process ahead of it in the queue which makes the waiting time of it 0. Waiting time of P2 is 24 as it goes after P1 has completed.

Waiting time for P3 will be 27 (P1 run time+ P2 runtime) as shown in figure 4.1 below and the average waiting time becomes 17 [(0+24+27)/3].The average completion time for the three process also become 27 [(24+27+30)/3]. This shows that for processes with long runtime being in front of the queue, short jobs get stuck behind and makes the waiting time very long reducing the performance of these processes. Such types of scheduling algorithms are also not suitable for interactive systems.

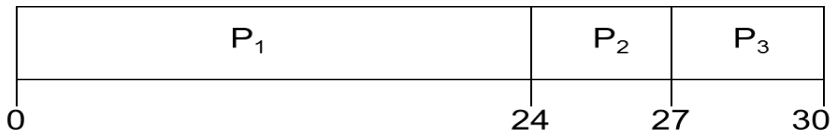


Figure 2.8. FCFS process execution.

Shortest Job First (SJF)Scheduling

This is another non-preemptive scheduling algorithm that handles jobs based on the length of time that the CPU requires to process them. When there are several equally important jobs queued in the ready queue waiting to be started, the scheduler picks the process with the shortest execution time requirement first. Moving a short job before a long one decreases the waiting time for short job more than it increases the waiting time for the longer process. SJF scheduling prefers I/O bound processes because running short CPU-burst jobs first gets them done and moves them out of the way and it also enables to overlap their I/O request which enhances the efficiency of the CPU. This algorithm is actually taken from the "procrastination" behavior of human beings. When faced with too much work, we prefer to do the short tasks first and get them out of the way leaving the big, hard tasks for later.

This algorithm is efficient only if it is possible to predict in advance the amount of CPU time required by each job which is possible for batch environments, but difficult for interactive systems. If two processes have the same length next CPU burst, FCFS scheduling is used to break the tie.

Consider the previous three processes P1, P2 and P3 with same CPU time and similar sequence. Now using SJF, P2 will be executed first followed by P3 and at last P1 if they arrived simultaneously.

Figure 2.9.Sample arrangement of process in SJF scheduling

As you can observe on the figure, waiting time for P2 has become0, for P3 to be3 and for P1 6 reducing the average waiting time from 17 (FCFS) to 3 and the average completion time 13 [(3+6+30)/3] from 27.

However, it is worth to note that SJF is optimal only for simultaneously available jobs.

Shortest Remaining Time Scheduling

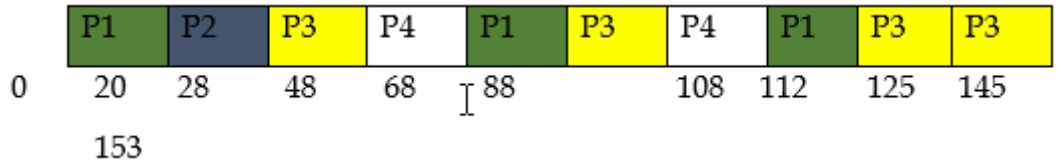
This is a pre-emptive variant of the SJF scheduling algorithm which preempt the currently running job if a newly arriving job has shorter expected CPU burst time. It operates by comparing the time left for completion of the running process with the total execution time of a new process. The scheduler immediately stops the running process and gives higher priority to the newly arriving process if and only if its execution time is smaller than remaining completion time of the running one. Again in this algorithm, the runtime of a process needs to be known in advance. The advantage of SRT scheduling algorithm is the optimal average response time resulted and the reduced waiting time especially for short jobs. However, larger jobs might be deprived of the CPU and end up in starvation if several small jobs arrive.

Round Robin Scheduling (RR)

This scheduling algorithm is known to be the simplest, fair and most widely used scheduling algorithm most suitable for interactive systems. It assigns time slices to each process in equal portions and in circular order, handling all processes without priority. Round-robin scheduling can also be applied to other scheduling problems, such as data packet scheduling in computer networks. The name of the algorithm comes from the round-robin principle known from other fields, where each person takes an equal share of something in turn. It is similar to FCFS scheduling, but preemption is added to switch between processes. A small unit of time, called a time quantum (or time slice), is defined which is generally from 10 – 100 milliseconds. The ready queue is treated as a circular queue. The scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum. No process is allocated the CPU for more than 1 time quantum in a row. If a process' CPU burst exceeds 1 time quantum, that process is preempted and is put back in the ready queue.

To implement the RR scheduling, we keep the ready queue as a FIFO queue of processes. New processes are added to the tail of the queue. The CPU scheduler picks from the head of the queue, sets a timer to interrupt after 1 time quantum, and dispatches the process. One of two things will then happen. The process may have a CPU burst of less than one time quantum in which case the process itself releases the CPU voluntarily and the scheduler will then proceed to the next process in the ready queue. Otherwise, if the CPU burst of the currently running process is longer than 1 time quantum, the timer will go off and will cause an interrupt to the operating system. A context switch will be executed, and the process will be put at the tail of the ready queue. The scheduler then selects the next process in the ready queue. It is important to note that the length of the quantum time determines the efficiency of the algorithm as well as the turnaround time. If the quantum time is too large, RR becomes a mere copy of FCFS scheduling algorithm and results in poor response for small interactive jobs. If too small is the quantum time, it results in several switches between processes increasing the average turnaround time and reduce the efficiency of CPU. A quantum around 20-50msec can be a reasonable compromise.

Example: Consider four processes P1, P2, P3 and P4 arrived at time 0 with CPU-burst time of 53,8,68, and 24msecs respectively. Using the RR scheduling algorithm with 20msec quantum time, the processes will be executed in the sequence shown in the following chart below



Waiting time for P1=(68-20)+(112-88)=72

P2= (20-0) =20

P3= (28-0) + (88-48) + (125-108) =85

P4= (48-0) + (108-68) =88

Average waiting time = (72+20+85+88)/4=66¼

Practice activity: Implement the RR scheduling algorithm with C

Priority Scheduling

In a multiprogramming system, processes can be given different rank taking into consideration behaviors of the processes or other system defined or user defined criteria. For instance, a daemon process transferring email in the background should have less priority than a foreground process outputting video to the visual display in real time. A priority scheduling algorithm works by allocating the CPU for processes with highest priority value. Each process in a job queue is assigned a priority value according to its importance and urgency, the scheduler then allocates CPU to the processes according to their priority level. Again, note that for processes with similar priority value, FCFS scheduling algorithm is used.

Priority is expressed in terms of fixed range number such as 0 to 10. However there is no general agreement on whether 0 is the highest or lowest priority. Some systems use low numbers to represent low priority while others use low numbers for high priority. But in this module, we will use low numbers to represent high priority.

Priority scheduling works as preemptive as well as nonpreemptive policy. The priority of a newly arriving process into a ready queue is compared with that of the currently running process. A preemptive priority-scheduling algorithm will preempt the CPU if the priority of the newly arrived process is higher than that of the currently running process. A nonpreemptive priority-scheduling algorithm, on the other hand, will simply put the new process at the head of the ready queue and waits for the running process to give the CPU voluntarily and run the arrived process then after.

Processes can be given priority statically or dynamically. In a static priority, process is assigned a fixed priority value according to some external criteria such as importance of the process, owner of the process, or any other factor which is not part of the operating system. In such cases, the priority will not be adjusted dynamically. In dynamic allocation of priority, the system uses some measurable quantities or qualities including time limits, memory requirements, file requirements and CPU or I/O requirements to compute priority of a process and accomplish a certain system goals. For example a process with high I/O bound time, high priority will be given and CPU is granted immediately to avoid memory occupation for unnecessarily long time.

Example: Consider list of processes with arrival time, burst time and priority level as given in the following table below

Process	*Arrival*	*CPU-Burst time*	*Priority*
P1	*0*	*3*	*2*
P2	*1*	*2*	*1*
P3	*2*	*1*	*3*

At time, 0sec:

Only P1 has arrived to the ready queue with Burst Time 3 and Priority 2. So CPU will start P1 and do 1 second job of P1.

At time, 1sec:

Now the processes become two; P1 that arrived at 0sec and P2 arrived at 1sec. since priority of P2 is higher than that of P1, P2 will get a chance to use the CPU and will run for 1sec.

At time, 2sec:

Now there are 3 jobs; P1 that arrived at 0sec and has 2sec job to be done with Priority 2, P2 that arrived at 1sec and has 1sec job to be done with Priority 1 and P3 that arrived at 2sec and has 1sec job to be done with Priority 3.Here, P2 is with the highest priority, thus CPU will do 1sec remaining job of P2 which brings the process to completion.

At time, 3sec:

Two processes are left, P1 and P3 with 2 and 3 priority levels respectively. P1 will be thus, executed to completion till 5sec.

At time, 5sec:

There is only 1 job left by now, that is P3 with burst time 1 and priority 3. So CPU will do 1sec job of P3 bringing it to completion. This whole activity can be shown in a time-line as shown below

The average waiting time can be computed as:

(P1's Waiting Time +P2's Waiting Time + P3's Waiting Time)/3= (2+0+3)/3=1.7sec

It is also possible to group processes into a priority class and use the priority scheduling among the groups while RR or other scheduling algorithm can be used for the processes inside a class

A major problem with priority scheduling is indefinite blocking or starvation of processes with the least priority value. Low priority processes indefinitely wait for the CPU because of a steady stream of higher-priority processes. Aging can be used as a solution towards this problem. Aging is a technique of gradually increasing the priority of processes that wait in the system for a long period of time.

Multilevel Queue Scheduling

This scheduling algorithm uses the best of each algorithm by having more than one queue and is used for situations in which processes can easily be classified into different groups. For example, a common division is made between foreground (interactive) processes and background (batch) processes. These two types of processes have different response-time requirements and so may have different scheduling needs. Ready queue is partitioned into several separate queues and processes will be permanently placed in these queues based on some criteria. Each queue then implements its own scheduling algorithm for the processes inside and commonly a static preemptive scheduling algorithm is used among the queues. For example foreground queue may have absolute priority over background queue.

Therefore no process in the background queue could run except the foreground queues are empty. If a process entered the foreground queue while a process from the background queue is running, the background queue process will be preempted and the new process from the foreground queue gets the CPU causing possible starvation for the background queue process. To address this problem, time slice can be used between the queues where each queue gets a certain portion of the CPU time which it can then schedule among the various processes in its queue. On the above example, for instance, the foreground queue can be given 80% of the CPU time for RR scheduling among its processes, whereas the background queue receives 20% of the CPU to give to its processes in FCFS manner.

Multilevel Feedback Scheduling algorithm is another implementation of multilevel queue where processes have the right to move between the several queues of the ready queue instead of being placed permanently into queues at arrival time based on their CPU-burst properties. A process that uses too much CPU time is degraded to a lower-priority queue and a process that waits too long is upgraded to a higher-priority queue.

For example consider a MLFQ scheduler with three queues, Q0 with time quantum 8 milliseconds, Q1 with time quantum 16 milliseconds and Q2 on FCFS basis only when queues Q0 and Q1 are empty. A new job enters queue Q0 is served by FCFS and receives 8 milliseconds. If not finished in 8 milliseconds, it is moved to Q1. At Q1 the job is again served by FCFS and receives 16 milliseconds. If not completed, it is preempted and moved to Q2 where it is served in FCFS order with any CPU cycles left over from queues Q0 and Q1.

Figure 2.10. .Multilevel Feedback Queue

In general, a MLFQ scheduler is defined by the following parameters:

- The number of queues
- The scheduling algorithm for each queue
- The method used to determine when to upgrade a process to a higher-
- priority and a lower-priority queue
- The method used to determine which queue a process will enter when that process needs service
- In conclusion, there are four primary methods to servicing the queues in a multilevel queue scheduling.
- In no movement between queues, all the jobs in the higher priority queue are serviced first before any job in a lower priority queue.

Movement between queues and adjust priorities assigned to jobs. When a time slice interrupt occurs, a job is pre-empted and moved to the end of the next lower priority queue if it is CPU bound. On the other hand a job that is pre-empted because of an I/O operation is moved up to the next higher level queue. In variable time quantum per queue, each queue is given a time quantum as long as the previous queue. If a job does not finish execution in the first time quantum, it is moved down to the next lower priority queue. If it doesn't finish execution in the second time quantum (which will now be longer than the first time quantum), it is moved down to the next lower priority queue and so on.

Aging is used to guard against indefinite postponement. This refers to a situation when a job somehow, due to a combination of certain circumstances, is unable to gain access to the CPU in order to execute. To guard against this possibility, whenever a job has not been serviced for a long period of time, it is moved up to the next higher level queue. If it still is not serviced after a given period of time, it is moved up again to the next higher level queue and so on. From the definition of a multilevel queue scheduler, it is the most general CPU-scheduling algorithm but also the most complex one as it requires some means by which values for all the parameters can be selected.

Thread Scheduling

In a previous section, we have discussed what threads are and the two different implementations of a thread as user-level and kernel-level threads. Now let's see how the scheduler allocates execution time for threads in a process. When several processes are available with several threads, scheduling differs as to whether the threads are user-level or kernel-level. If the threads in a process are user-level, the kernel operates as usual only on processes since it is unaware of the threads and the scheduler schedules the processes by giving them control for a quantum. A thread scheduler inside a process will then use any of the scheduling algorithm discussed previously and picks a thread and grants the execution time.

This thread runs until it finishes its job or until the quantum time for the process ends in which case the system scheduler takes the CPU and grants it for another process. When the first process is granted the CPU again, that same thread resumes execution if it did not finish its execution earlier. In such cases, only the threads within the process are affected and other processes in the system will run without being affected. If the threads in processes are of kernel-level, the kernel operates on the threads and picks them to run by giving a fixed quantum and forcibly suspend the thread if it exceeds the quantum allotted. The scheduler does not consider from which process a thread is picked to run.

The main difference between these two thread implementations scheduling is performance where the kernel-level scheduling requires a full context switch as every information related with a thread from one process must be maintained before a thread from another process gets to execute, making the algorithm too slow. But the kernel can take the cost associated with this context switching and make an optimal decision that can improve this performance problem. On the other hand, scheduling the kernel-level threads will not suspend an entire process when a thread in that process waits for I/O as user-level thread scheduling does.

One additional thing associated with user-level threads scheduling is the use of application-specific thread scheduler where the urgency of the threads will be examined and CPU is granted accordingly. This scheme increases parallelism in processes where several I/O bound threads are available

Multiple Processor Scheduling

So far, our discussion was solely about scheduling processes and threads on a single processor system. However, there are also multi-processor systems with processes to be scheduled where the scheduling problem becomes even more complex. Let's consider a system with homogeneous or identical processors and discuss the different possible scheduling mechanisms. One scheduling possibility is to use asymmetric processing where one of the processors entitled as a master server and handle all scheduling, I/O processing and all other system activities while the remaining other processors handle only user codes.

Another approach in multiprocessor system can be symmetric processing where all processors have similar level and scheduling is done by each independently. Scheduling then proceeds by having the scheduler for each processor examine the ready queue and select a process to execute. The ready queue can be a separately maintained one by each processor or a common ready queue shared by all processors where the first scenario could make one processor be idle, with an empty queue, while another processor too busy, with full ready queue. But if a common ready queue is used, all processes go into one queue and are scheduled onto any available processor. However, it is important to ensure that no two processors choose the same process and that processes are not lost from the queue. Nearly all modern operating system support this symmetric scheduling approach.

Conclusion

Process scheduling is a process of granting CPU for processes in the ready queue. The scheduler decides which process to grant the CPU according to a scheduling policy. Pre-emptive and non-pre-emptive scheduling policies are used by the scheduler where the pre-emptive policy forcibly suspends a process and gives the CPU to other processes in a ready queue while the non-pre-emptive scheduling policy simply waits for one process to finish its execution or voluntarily give-up the CPU and assigns another process to execute. FCFS is a non-pre-emptive scheduling algorithm which schedules processes according to their arrival time and results in starvation for short processes at the tail of long ones. SJF is another non-pre-emptive scheduling algorithm which schedules processes according to their CPU-burst time.

SRT is a pre-emptive scheduling algorithm which compares the remaining CPU-burst time of a running process with the CPU-burst time of a newly coming process and grants the CPU for the smallest CPU time requirement. RR is a pre-emptive scheduling algorithm that allocates a fixed quantum time for processes and allows them to run till that time expires. It is mainly suitable for interactive systems. Priority scheduling is where processes are assigned a value to indicate their priority for execution and get the CPU according to these values. It can be implemented both as pre-emptive and non-pre-emptive policy. Multilevel queue scheduling groups processes into various categories and uses different scheduling algorithms to schedule the groups while implementing another scheduling algorithm within a group.

Thread scheduling also varies according to the thread implementation. Multiprocessor systems have a more complex scheduling requirement that must be handled by the scheduler. Symmetric or asymmetric scheduling can be followed in such system where the asymmetric one is commonly used by all current operating systems.

Assessment

Define the differences between pre-emptive and non-pre-emptive scheduling

Explain the differences in the degree to which the following scheduling algorithms discriminate in favor of short processes:

- FCFS
- RR

Multilevel Feedback queues

What is the importance of distinguishing CPU-bound processes from I/O bound processes for the scheduler?

Suppose that the following processes arrive for execution at the times indicated and each will run for the amount of time listed. Calculate the average turnaround time of these processes when FCFS and SJF scheduling algorithms are used

Process Name	*Arrival Time*	*CPU-burst Time*
P1	*0*	*8*
P2	*0.4*	*4*
P3	*1*	*1*

P is a process that requires exactly 5 seconds of CPU time. The administrator has given P the maximum priority available in the system. Nevertheless, P started over 10 seconds ago and is still not finished. What are two likely explanations for why P hasn't finished yet?

Suppose the following three processes (A, B, C) are submitted to the system at the same time but process D is submitted when process A finishes. Assume these are the only active processes with 20 secs context switch time. None of them ever blocks on I/O. Let the variable T represent the average turnaround time of these processes. Depending on how the jobs are scheduled, T may be higher or lower.

Process	*CPU Time Required*
A	*80 min*
B	*4 min*
C	*1 min*
D	*10 min*

Fill the turnaround time column if processes run in non-pre-emptive way as they ordered in "Run Order" column.

Run Order	*Turnaround time*
A-B-C-D	
B-A-C-D	
C-B-A-D	

1. What is the smallest possible value of T? Which scheduling order leads to this value?
2. Suppose the order from b) above is reversed. Now what will be the value of T?
3. Suppose these jobs are scheduled by a Round Robin scheduler with a small quantum (about 2 seconds). Approximately what will be the resulting value of T? Show clearly how you compute your answer.
4. What are the two scheduling approaches used in a multi-processor system?
5. Discuss the main differences between user-level thread scheduling and kernel-level thread scheduling
6. Discuss the two scheduler types
7. What is an interrupt handler?
8. Discuss what a context switching is. What activities took place during a context switch?

Activity 6 – Deadlock

Introduction

Deadlock is one of the problems of concurrency. In a multiprogramming environment, processes compute for resources. Processes wait for resources which are owned by other processes, in turn these processes wait for that process to release a resource. How does the operating system bring these processes to an agreement?

In a multiprogramming system, processes are always in request of the several finite resources available in the system for their successful execution. The resources are partitioned into several types like Memory space, CPU cycles, files, and I/O devices (such as printers and tape drives). A process must request a resource before using it, and must release the resource after using it. When two or more processes simultaneously request for a resource, only one process will be allowed to use the resource at any instant of time to avoid unintended outcomes or corrupted outputs.

But for most programs, a process requires exclusive use of more than one resource at a time without exceeding the number of available resources in the system. For instance, two processes, A and B, may need to write document being scanned on a CD. Process A asks for the scanner first and is allowed to use it while process B asks for the CD recorder first because of its different programming structure and will also be permitted to use the recorder,. After process A completes the scanning task, it requests for the CD recorder but will be denied as it is already being in use by process B. Process B now requests for the scanner without releasing the CD recorder which is requested by A. such situations causes both processes to be blocked or move to their waiting state and remain there forever as both are in need of same resource simultaneously resulting in a deadlock condition.

Deadlock can be defined as a situation whereby two or more processes get into a state whereby each is holding a resource that the other is requesting and processes are waiting in a loop for the resource to be released. None of these waiting processes will ever wake up and start execution as all the processes are blocked and needs an event to happen by one of these blocked processes to move to a running state

At this point, it is important to distinguish between starvation and a deadlock. In starvation, a process cannot continue simply because it is not allocated the CPU while in deadlock, processes cannot continue because they are blocked and would remain in this state even if the CPU was available to them. Deadlock is a global condition rather than a local one which means that if were to analyze any process involved in a deadlock, we would not be able to find any error in the program of that process. The problem does not lie with any particular process, but with the interaction between a groups of processes that are multitasked together. Deadlock effectively brings a large portion of the system to a halt.

Normally, a process goes through three consecutive steps to utilize a resource. These are:

Request a resource

Use the resource

Release the resource

The operating system keep in a system table the status of each resource as free or allocated and if allocated, to which process. If a requested resource is not available, the operating system can add the requesting process to a waiting queue of processes for this resource or the process itself waits a little while and checks again the resource as is the case in some systems. A resource can be either a preemptable or non-preempt able type. A preemptable resource is a type of resource that can be shared by taking away from a process owning it without negatively affecting it. A good example of such type is a memory. A non-preemptable resource, in contrary, is a non-sharable resource that cannot be taken from a process currently owning it without negatively affecting it. CD recorders are examples of non-preemptable resources. A deadlock happens when resources are non-preemptable types and is most common in multitasking and client/server environments.

For a deadlock to happen between processes, the following four conditions should hold simultaneously in a system.

Mutual Exclusion: resources are non-preemptable and once a process is allocated a resource, it has exclusive access to that resource. This resource cannot be shared with other processes

Hold and Wait: a process must be holding exclusively at least one resource and still make a request for another resource that is currently hold by another process.

No Preemption: Resources can only be released voluntarily by the process itself rather than by the action of an external event

Circular Wait: two or more processes must be in a circular chain where each process waits for a resource that the next process in the chain holds.

All these four conditions are necessary but not sufficient for a deadlock to occur. Only if the right combinations of unfortunate circumstances arise, then will deadlock manifest itself.

Resource-Allocation (R-A) Graph

The four conditions of the deadlock can be described more precisely in terms of a directed graph called a system Resource allocation graph. This graph consists of a set of vertices V and a set of edges E.

V is partitioned into two different types of nodes P = {P1, P2, …, Pn}, the set consisting all the active processes in the system, and R = {R1, R2, …, Rm}, the set consisting all resource types in the system. A directed edge from process Pi, to resource type Rj, known as request edge, is denoted by Pi → Rj and signifies that process Pi requested an instance of resource type Rj and is currently waiting for that resource. A directed edge from resource type Rj to process Pi, known as assignment edge, is denoted by Rj → Pi; it signifies that the process has been allocated an instance of resource Rj.

When a process Pi request an instance of resource type Rj, a request edge is inserted in the resource-allocation graph. The request edge will instantaneously change to an assignment edge when the request is fulfilled. When the process no longer needs access to the resource it releases the resource, and as a result the assignment edge is deleted.

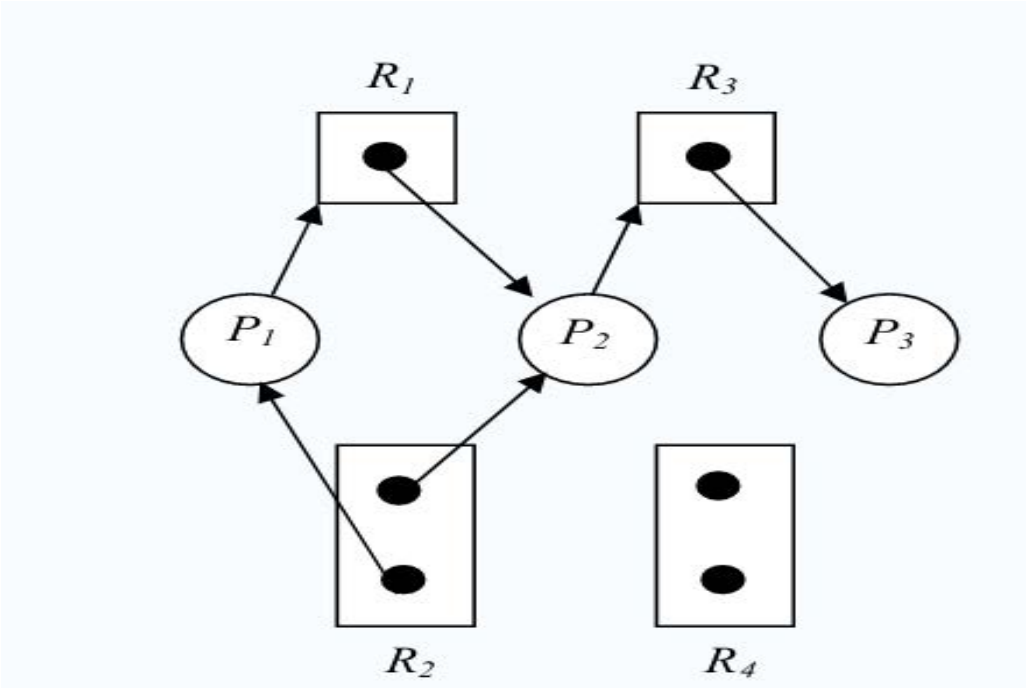


Figure 2.11. Resource allocation graph

Figure 2.11 depicts a resource allocation graph with three processes P1, P2 and P3; four different resources R1, R2, R3 and R4, and request from process to resource. Resource R2 and R4 have two instances while R1 and R3 are only single instance.

- Process P1 is holding an instance of resource type R2, and is waiting for an instance of resource type R1.
- Process P2 is holding an instance of resource type R1 and R2, and is waiting for an instance of resource type R3.
- Process P3 is holding an instance of resource type R3 and has no more requests further.

As can be depicted from the figure and also the description above, the processes possible to encounter deadlock are P1 and P2 as they have a waiting state. In general, if a cycle can be seen on the resource allocation graph and each resource type has exactly one instance, there is an implication for a deadlock and each process involved in that cycle is deadlocked. In this case, a cycle in the graph is both a necessary and a sufficient condition for the existence of deadlock. However, if each resource type has several instances, then a cycle does not necessarily imply that a deadlock has occurred. In this case, a cycle in the graph is a necessary but not sufficient condition for the existence of deadlock.

If no cycle exists in the graph, then all the processes are free of any deadlock situation

In the resource graph diagram above, suppose process P3 requests an instance of resource type R2.Since no resource instance is currently available, a request edge P3 → R2 is added to the graph as in figure 4.6 below.

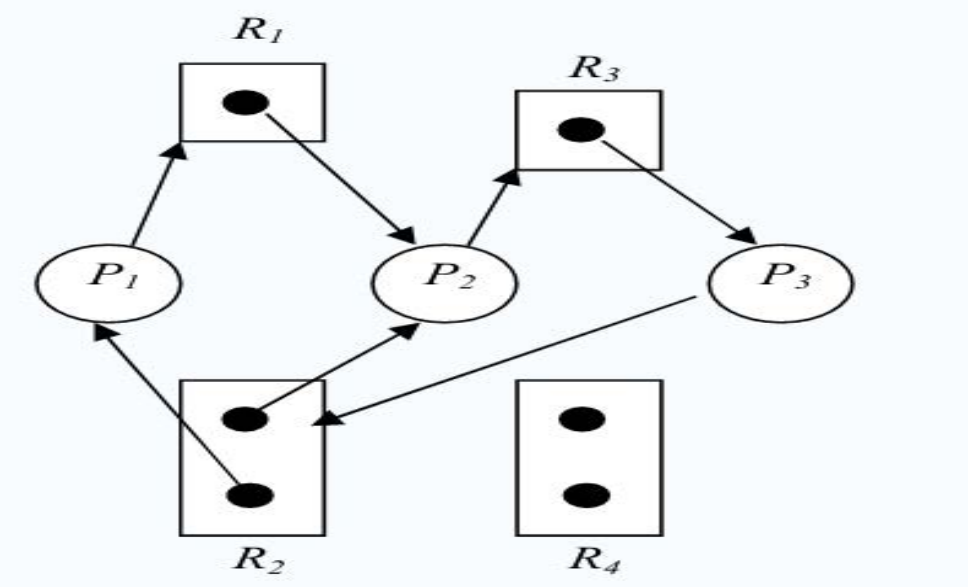


Figure 2.12. Resource allocation graph with deadlock

.At this point, two minimal cycles can be observed:

P 1 → R1 → P2 → R3 → P3 → R2 → P1 and

P2 → R3 → P3 → R2 → P2

Moreover, Process P2 is waiting for resource R3, which is held by process P3. Process P3, on the other hand, is waiting for either process P1 or P2 to release resource R2. In addition, process P1 is waiting for P2 to release resource R1. These two conditions result in a deadlock situation for processes P1, P2 and P3.

To conclude, resource allocation graphs are tools that can be used to identify the occurrence of a deadlock given sequence of request/release. If a cycle can be seen in the graph, there is a possibility of deadlock none otherwise.

Conclusion

Processes may request and hold resources of the computer system. The order of allocating resources to processes is not predefined. A process may hold a resource and wait for another one without releasing the first one. This is a potential cause of deadlock, if another process is doing the same. To demonstrate this effect, one can use resource allocation graph. If cycle is occurred in the resource allocation graph, deadlock occurs.

Assessment

a. Give a practical example how dead lock occurs.

b. Demonstrate the example that you provided on (1) using resource allocation graph.

c. What are the requirement for deadlock to occur?

Activity 7 - Deadlock Handling Mechanisms

Introduction

If deadlock happens how does the operating system has to solve the problem? In this activity, we will discuss the four strategies of handling deadlock. But, different operating systems provide similar kind of solution.

Four different strategies can be used towards overcoming deadlocks but each with their own corresponding advantages and drawbacks. These are:

Ignoring: ignore the problem altogether and pretend that deadlock has never occur in the system.

Detection and recovery : make the system enter into a deadlock, detect the deadlock and devise actions to recover from the situation

Avoidance : carefully allocate resources to make sure no process will encounter a deadlock situation

Prevention: structurally negate one of the four conditions that results a deadlock and prevent the occurrence. Let's discuss each briefly

The OstrichAlgorithm

Ignore and pretend that a deadlock never happened in the system. This method is the simplest and used by most operating systems, including Linux, Windows. These operating systems use this method by assuming infrequent deadlock is appreciated by users than a restricting rule that forces all users to only one of everything including process and open file.

A tradeoff between convenience and correctness has to be made though which is more important and to whom is a serious point to consider. In such systems, application developer must write programs that handle deadlocks.

Deadlock Detection and Recovery

Detection

In this method, the system uses neither deadlock avoidance nor prevention but allows deadlocks to happen. It then tries to detect when the deadlock happens and take actions to overcome the problem after it happened. Such systems employ deadlock detection and recovery algorithms that tracks resource allocation and process states, and rolls back and restarts one or more of the processes in order to remove the deadlock. Detecting a deadlock that has already occurred is easily possible since the resources that each process has locked and/or currently requested are known to the resource scheduler or OS. Let's discuss some of the several deadlock detection as well as deadlock recovery methods re available.

Deadlock detection with one resource of each type

In such cases, systems with two instances of same resource are excluded and the system has only one instance of each resource type: one CD recorder, one Printer, and so on. If there is one or more cycle on the resource allocation graph of such system, it indicates a deadlock and any process within the cycle is deadlocked. Periodically invoking an algorithm to search for a cycle in the resource allocation graph enables the system to identify a deadlock. An algorithm to detect a cycle in a graph requires an order of n2 operations, where n is the number of vertices in the graph.

Several such algorithms exist among which we consider the simplest one. This algorithm defines a single data structure with list of nodes. The algorithm inspects the graph and terminates either when a cycle is found or has shown none exists by marking the arcs already inspected in the graph to avoid repetitive inspection. The algorithm operates in the following manner for each node N, in the graph

- Initialize list of nodes, L, to a blank list and label each of the arcs as unmarked
- Append a new node to the tail of L and examine if this node now appears twice in L. if it does, the algorithm exits as a cycle is identified.
- Inspect for any unmarked outgoing arcs from the given node and if so goto d and if not proceed to e
- Randomly pick one of the outgoing unmarked arc and mark it. Then follow it to the new current node and go to c
- This is a dead end and should be removed. Go to the node that was current node prior to this one and label it current node then jump to step c. if this node happens to be the initial node, then the diagram has no cycle and the algorithm terminates

Let's see how the algorithm operates with example. Let's take a system consisting of seven processes and six resources as shown in figure 2.13 below.

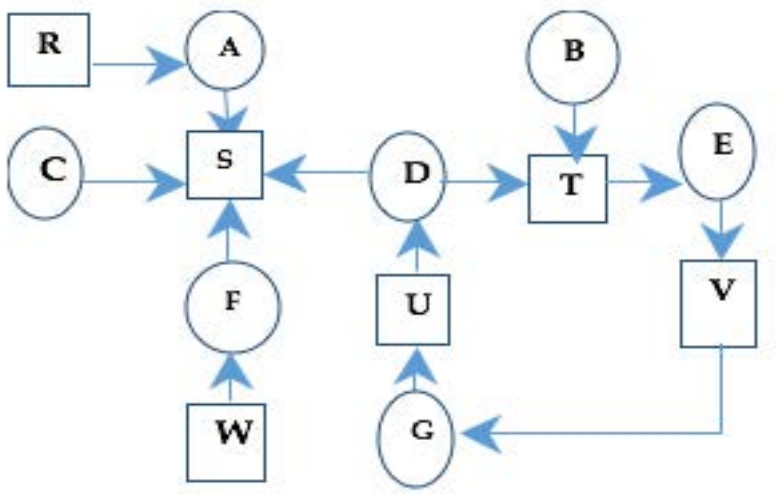


Figure 2.13 (a) resource allocation graph

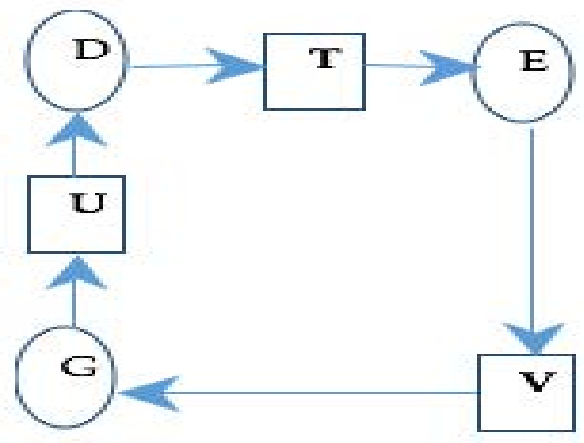


Figure 2.13 (a) Cycle extracted from (a)

Let's start inspecting left to right and top to bottom as any node can be picked arbitrarily. Let's move resource R to the empty list L and initialize it. The only possible move is to A which adds process A to the list making L={R, A}. Now move to S and L is updated to be {R, A, S}. S is a dead end with no outgoing arcs forcing us to move back to process A which in turn has no unmarked arcs that makes us retreat back to R and finish examining node R. now, the algorithm restarts for another node, say A emptying L. since this path has already been inspected, no need to trace it again for us and the algorithm also terminates instantly because of the presence of only a single arc from A to S. let's start from B and follow outgoing arcs till we arrive at D which makes L to be {B,T,E,V,G,U,D}. S or T can be taken randomly afterwards. If S is taken, this will lead to a dead end causing to move back to D. if T is picked, it will cause the algorithm terminate as this will make T to be found in L twice resulting in a cycle.

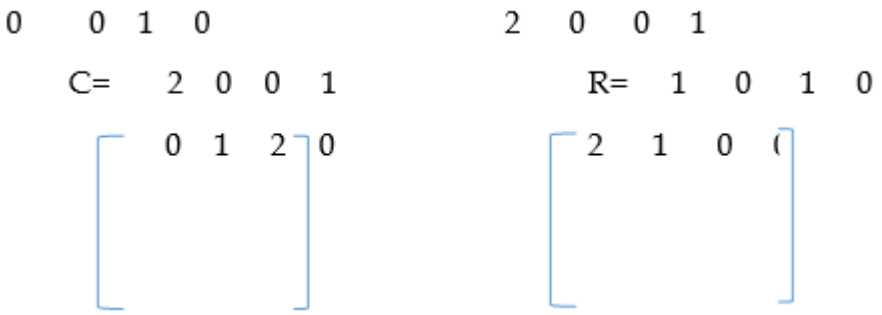
Deadlock detection with several instances of resource type

Different deadlock detection mechanism is required when more than one instance of a resource is found in a system. A matrix-based deadlock detection algorithm can be used for n processes and m resource classes with Ei resources of class i (1<i< m). E is the existing resource vector which gives the total number of instances of each resource type found in the system. Let A be the available resource vector which gives the number of resource instances currently unassigned. Let R and C be request matrix and current allocation matrix respectively. How many instances of a resource class is currently held by process Pi is represented in the ith row of C. thus Cij represents the number of resource instances j held by process i while Rij depicts the number of resource instances j required by process Pi. After defining these four vectors, the algorithm works by comparing the vectors as follows:

- Look for any unmarked process Pi for which the ith row of R is less than or equal to A
- Add the ith row of C to A, if such a process exists, note the process and return back to step a
- If not, exit

Initially, all the processes are unmarked but will be marked as the algorithm proceeds to indicate the processes are completed concluding they are not deadlocked. If any process remains unmarked when the algorithm finishes, such process is said to be deadlocked. The Algorithm requires an order of m x n2 operations to detect whether the system is in a deadlocked state.

To illustrate the algorithm, consider an example system with three processes, P1 to P3 and four resource classes, R1 to R4 with 4, 2, 3 and 1 instance respectively. The number of instances of each resource class E is {4,2,3,1} and the available instances A is {2,1,0,0}. Assume P1 has 1 R3, P2 has 2 R1 and 1 R4, P3 has 1 R2 and 2 R3 and each process also requests for additional resources as can be seen from the below C and R matrices respectively. .



The algorithm looks first for a process whose request can be provided. P1's request can't be fulfilled since there is no more instance of R4 in A. P2's request can't also be granted as no instance of R3 is found in A. P3 is thus the only process that can be given the current request as both of its needed additional resources (2 R1 and 1 R2) are available in A. so P3 completes its execution and returns back all its resources making A be {2, 2, 2, 0}. Now P2 gets a chance to run as its requests can be granted and finally making A to be {4, 2, 2, 1}. Finally P1 gets what it has asked for implying occurrence of no deadlock. However, if we make a little modification on the requests made by P2 asking for 2 R1, 1 R2, and 1 R4, the entire system ends up being deadlocked as none of the requests of all the processes can be satisfied.

After identifying the means through which deadlock can be determined in a system, when and how often to invoke the algorithms should be a question to be answered and the answer to these questions, of course, depends on how often a deadlock is likely to occur and how many processes will need to be rolled back. Generally, one possibility could be to detect the deadlock as early as possible by running the algorithm with each resource request. This, however, costs the system CPU time. Alternatively, the algorithm can also be invoked every k minutes or when the CPU utilization becomes lower than some pre-set threshold value.

Recovery

Once, the detection algorithm has succeeded in identifying a deadlock, several ways exist to overcome the situation. One alternative can be through notifying the operator about the deadlock and let the operator handle it manually. Another possibility can be letting the system recover from the deadlock automatically. Two different approaches are used to recover automatically: pre-emption and abortion.

Recovery through resource pre-emption

This method works by temporarily taking away a resource from its current owner process and gives it to another process and finally returns back to the owner. i.e. successively preempt some resources from processes and give these resources to other processes until the deadlock cycle is broken. Three important issues need to be addressed if to use this method

Victim: which resources and which processes are to be pre-empted? Cost minimization must be considered while ordering the pre-emption. The number of resources a deadlock process is holding and the time thus far used by the deadlocked process during its execution can be taken as cost factors.

Rollback: if a process is pre-empted a resource, what is the fate of this process? Obviously, the normal execution of the process can't proceed as it lacks some required resources. Thus, the process must be rolled back to some safe state and restarted from that state after the deadlock is solved. Since, in general, it is difficult to determine what a safe state is, the simplest solution could be a total rollback.

Starvation: how can we guarantee that resources will not always be pre-empted from the same process? Same process may always be picked as victim which leads to starvation. Including the number of rollback in the cost factor can be a solution to this problem.

This recovery method is difficult, if not impossible, as picking a process to pre-empt is highly dependent on the ease of taking back resources now held.

Recovery through process termination

Killing one or more processes in a system may help to recover from deadlocks. Choose a victim process from the cycle and abort it so that other processes continue running by using the freed resources. This process can be repeated several times in the cycle till breaking the cycle is possible. A process outside a cycle can also be picked as a prey process where the resources hold by it are needed by one or more deadlocked processes. For example two processes are running one holding a printer and requesting a CD reader while the other one using a CD reader and waiting for a printer. The two processes end up being deadlocked. If there is a third process with another identical printer and CD reader resource granted, killing this process makes the resources free which can then be allocated to the two processes solving the deadlock situation. Killing a process may not be easy. If the process was in the midst of updating a file or printing data on a printer, terminating it will leave the file in an incorrect state and the system needs to reset the printer to its correct state before printing the next job. Factors such as the priority of the process, how many more resources are required to complete the process, etc. affect the process chosen as a victim. In general, It is best if the process chosen as a victim is a process that can resume from the beginning with no bad effect or the termination cost is minimal.

Deadlock Avoidance

This is a conservative approach to handling deadlock that requires priori information about the maximum number of resources of each type that a process will ever request for in its lifetime, the maximum claim. This method is the simplest and most useful model which dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition. Resource-allocation state is defined by the number of available and allocated resources, and the maximum demands of the processes.

The idea is to ensure that a system consisting of a number of interacting processes remain in a safe state. A safe state is a situation where even if every single process simultaneously makes its maximum claim for resources at the same time, deadlock will still not occur and there is some scheduling arrangement to allow every process run and finish its execution. More formally, a system is in a safe state only if there exists a safe sequence of processes <P1 , P2, ... , Pn> for the current allocation state. That is, for each Pi, the resource requests that Pi can still make can be satisfied by the currently available resources plus the resources held by all Pj, with j <i. In this situation, if the resources that Pi needs are not immediately available, then Pi can wait until all Pj have finished. When they have finished, Pi can obtain all of its needed resources, complete its designated task, return its allocated resources, and terminate.

When Pi terminates, Pi+l can obtain its needed resources, and so on. If no such sequence exists, then the system is said to be in unsafe state. Let's see with example safe and unsafe states.

Consider a system with 10 instances of one resource type and three different processes P1, P2 and P3. P1 currently is using 3 instances of the resource but requires 6 more instances, P2 has 2 instances being in use and needs 2 more while P3 also has 2 instances now and needs 5 more as shown in table 4.1. below.

Process	*Currently has*	*Maximum requirement*
P1	3	9
P2	2	4
P3	2	7

Table 2.3. Resource allocation and maximum claim

From this, only 3 instances of the resource are available to be given to one of the processes. The system is in a safe state time t0 with <P2, P3, P1> process sequence as the scheduler can immediately allocate the resource to P2 (since it requires only 2 more instances leaving only 1 instance free) and finish it which will then make all the 4 instances available to be used. Then with 5 instances at hand, the system can now schedule P3 to run to completion using all the available instances. Finally P1 gets the chance to run as 7 instances are available and P1 only needs 6 of them. With this careful scheduling, the system is able to avoid deadlock making it safe.

On the other hand, assume at time t1, P1 requests one more instance and is granted making the total resource instances occupied by P1 4 and all the others as described on table 4.1 above. Now the system has only 2 instances free. With these, P2 can run safely using both leftover instances. After P2's completion, the system will have 4 free instances to be given to one of the processes. However, both P1 and P3 require 5 instances to run which makes the scheduler get stuck as there is not enough free resource available. Hence, the two processes cannot get to completion moving the system to unsafe state.

Of course, deadlock is not guaranteed to occur in an unsafe state, but should the worst case scenario arise, then deadlock will definitely occur.

A scheduling algorithm used to avoid such deadlock situation is known as the Banker's Algorithm which dynamically examines the resource-allocation state ,that is defined by the number of available and allocated resources and the maximum demands of the processes, to ensure that a circular-wait condition can never exist. The name was chosen because the algorithm could be used in a banking system to ensure that the bank never allocated its available cash in such a way that it could no longer satisfy the needs of all its customers.

In this algorithm, the goal is not to entertain any request that can change the safe state to unsafe one. Whenever a request comes for a resource, the algorithm checks if fulfilling this request may result to unsafe state or not. If so, the request is postponed until later but otherwise granted.

This means the OS must always identify the process with the smallest number of remaining resources and make sure that the current number of remaining resources is always equal to, or greater than, the number needed for this process to run to completion. The Banker's algorithm works similar for both single and several resource types .For the Banker's algorithm to work, it needs to know three things:

a. How much of each resource each process could possibly request?

b. How much of each resource each process is currently holding?

c. How much of each resource the system has available?

Given n processes and m resource types in a system, the data structures required to implement the Banker's algorithm are as follows:

- A: a vector specifying the available resources
- P: a vector of possessed resources by processes
- E: a vector of the existing instances of resources

The algorithm can now be stated as:

Find a process P in the n by m matrix whose number of resource request rejections are smaller or equal to A. if it is impossible to find one, the system will eventually go to deadlock since no process can be brought to completion. If more than one process is obtained, the choice is arbitrary. Assume this process has requested all the needed resources and finishes its execution. Mark the process terminated and update vector A by adding all the resources returned back by this process

Repeat these two steps until either all the processes in the system comes to termination, which shows safety of the initial state, or until a deadlock happens in which case it was not. Like most algorithms, the Banker's algorithm involves some trade-offs.
The required number of resources by a process needs to be known for the algorithm to operate which is unavailable in most systems, making the Banker's algorithm useless. Moreover, it is also unrealistic to assume static number of processes in a system as this number varies dynamically in most systems. The requirement that a process will eventually release all its resources on termination is insufficient for a practical system as waiting for hours even days for the resources to be released is unacceptable. In conclusion, there are few, if any, currently existing systems using this algorithm to avoid deadlocks.

Deadlock Prevention

Deadlock avoidance is almost impossible in real systems and as discussed previously, for a deadlock to occur, each of the four necessary conditions must hold. By ensuring that at least one of the four necessary conditions cannot hold, a system can prevent the occurrence of a deadlock. Let's elaborate on this approach by examining each of the four conditions separately.

Mutual exclusion

Sharable resources, like Read-only files do not require mutually exclusive access making them out of a deadlock situation. If several processes attempt to open a read-only file at the same time, they can be granted simultaneous access to the file as a process never needs to wait for a sharable resource. The mutual exclusion condition, however, must hold for non-sharable resources like printer that cannot be simultaneously shared by several processes .Algorithms that avoid mutual exclusion are called non-blocking synchronization algorithms. In general, since some resources are inherently non-sharable, deadlocks cannot be avoided by denying the mutual exclusion condition.

Hold and wait

To ensure the hold and wait condition never happens in a system, we must guarantee that whenever a process requests a resource, it does not hold any other resources. This requires process to request and be allocated all its resources before it begins execution which can be implemented by requiring that system calls requesting resources for a process precede all other system calls. Alternatively, this can be avoided by allowing a process to request resources only when the process has none. A process may request some resources and use them. To request for another additional resources, however, it must release all the resources that it is currently allocated. This method has two main drawbacks: Low resource utilization since many of the resources may be allocated but unused for a long period of time and starvation as a process that needs several popular resources may have to wait indefinitely.

No preemption

The third necessary condition for occurrence of a deadlock is that there be no pre-emption of resources that have been allocated. To ensure that this condition does not hold, the following algorithm can be used. If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released. The pre-empted resources are then added to the list of resources for which the process is waiting for and the process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.

This protocol is often applied to resources whose state can be easily saved and restored later such as CPU registers and memory space. It cannot generally be applied to such resources as printer and tape drives. If a process had been given the printer and is printing its output, taking away the printer forcibly because another resource requested by the process could not be granted is a complicated task if not impossible .Attacking this condition is even more difficult than the previous two.

Circular wait condition

The last condition leading to a deadlock is the circular wait. This can be avoided by letting processes to wait but not in a circular fashion. One way to achieve this could be through assigning a global precedence number to resources and force processes to request for resources according to increasing precedence. That is, if a process holds some resources and the highest precedence of these resources is m, then this process cannot request any resource with ordering smaller than m. This forces resource allocation to follow a particular and non-circular ordering, so circular wait cannot occur. Alternatively, a system can set a rule specifying a process is entitled to only a single resource at any time. This allows holding only one resource per process and if a process requests another resource, it must first free the one it's currently holding (or hold-and-wait).

Conclusion

A deadlock is a situation whereby a process holds and requests for another resource which is currently in use by another process that is looking for a resource captured by the first process. It is a potential in any operating system. A deadlock situation may occur if and only if four necessary conditions hold simultaneously in the system: mutual exclusion, hold and wait, no pre-emption, and circular wait. A system can use four different techniques in order to protect itself from a deadlock situation by determining which process sequencing is safe or not. These are: using protocols to prevent or avoid deadlocks, ensuring that the system will never enter a deadlock state, allow the system to enter deadlock state, detect it, and then recover, and just ignore the problem altogether, and pretend that deadlocks never occur in the system.

Assessment

- Using R-A graph, describe deadlocks
- State the necessary conditions for deadlock to occur.
- What are the various methods for handling deadlocks?

Consider a system having two processes and three similar type resources. Assume each process requires a maximum of 2 resources. Is it possible for a deadlock to happen? Explain your answer.

Let there be four processes and five resource types in a system with the following snapshot taken at t0

	Allocated	*Maximum*
P1	1 0 2 1 1	1 1 2 1 3
P2	2 0 1 1 0	2 2 2 1 0
P3	1 1 0 1 0	2 1 3 1 0
P4	1 1 1 1 0	1 2 2 1

What is the smallest value of x which makes this state safe, if Available vector (0 0 x 1 1) is given.

Given A= (1 3 2 0 1), if a request from process P1 arrives for (0, 4, 2, 0, 1), can the request be granted immediately?

Unit Summary

This unit centered on the concept of Processes in OS. A process is a program in execution. As a process executes, it changes state as new, ready, running, waiting, or terminated. The state of a process is defined by that process current activity. Each process is represented in the operating system by its own process-control block (PCB) and has its own address space, PID, and other important resources. Some applications require the process broken down into smaller components and run in parallel for efficient execution. Threads are basic unit of CPU utilization, each consisting of their own program counter, a stack, a set of registers, and a thread ID but share the address space of the process they are in. the three different implementation modes of a thread are user space, kernel or hybrid of these.

This unit also treated two most important concepts of process management: scheduling and deadlock. Scheduling is the means by which the operating system picks processes out of the job queue and put them in a ready queue so that they can be executed. Two types of schedulers are available in a system that performs this operation: short-term scheduler and long-term scheduler.

The short-term scheduler selects from among the processes in memory that are ready to execute and allocate the CPU to one of them.

The long-term scheduler also known as the dispatcher performs context switching and gives control of the CPU to the process picked by the short-term scheduler. Scheduling decisions are required when a process switches from

- Running to waiting state
- Running to ready state
- Waiting to ready state or
- Terminates

Scheduling criteria varies based on the system type but generally the focus is optimization which can be achieved through maximum CPU utilization, maximum throughput, minimum turnaround time, minimum waiting time and minimum response time. Algorithms that implements Pre-emptive and non-pre-emptive scheduling policies are used by the scheduler to guarantee these.

Two or more processes might compete for same resource in a system which creates a situation called deadlock. A deadlock occurs when group of processes in a system are blocked and never run again because each of the processes were allowed an exclusive usage of some resources and yet each also requires additional resource that is hold by a process in the group. Mutual exclusion, hold and wait, no pre-emption and circular wait are the four conditions that must hold true simultaneously for a deadlock to happen. If the system is capable of avoiding any of these conditions from happening, it is free of being deadlocked. Ignoring a deadlock as if it never happened is the commonly used way of handling it in most operating systems including UNIX. Some systems, however, implement deadlock detection and recovery methods while others also use prevention and avoidance algorithms to overcome this deadlock situation and maintain a safe state by ordering the execution of processes carefully.

Unit Assessment

In a system with user space thread implementation, is there one stack per thread or per process? What about if the system was using kernel level thread?

- Describe a process control block (PCB).
- What possible information are kept in a PCB by the system?
- Discuss the advantages and disadvantages of user space kernel implementation.
- When does a process makes transition between states? Discuss each transition briefly
- Discuss process creation and termination.

The Fibonacci sequence is the series of numbers 0, 1, 1, 2, 3, 5, 8, ...Formally, it can be expressed as:

fib0 = 0

fibl = 1

fibn = fibn-l + fibn-2 . Write a C program using the fork() system call that generates the Fibonacci sequence in the child process. The number of the sequence will be provided in the command line. For example, if 5 is provided, the first five numbers in the Fibonacci sequence will be output by the child

Assuming the actual pids of the parent and child processes of the following program 2600 and 2603 respectively, what will be the values of pid at lines A, B, C, and D?

```
#include <sysltypes.h>
#include <stdio. h>
#include <unistd.h>
void main()
{
pid_t  pid'  pid1;
 /*  fork  a  child  process  */
pid  =  fork();
if  (pid <  0)  {  /*  error  occurred */
fprintf(stderr,  "Fork  Failed");
return  1;
}
else if  (pid ==  0)  {  /*  child  process  */
pid1 =  getpid();
}
printf("child: pid  =  %d",pid);  /*  A  */
printf("child: pid1 =  %d",pid1); /*  B  */
else {  /*  parent process  */
pid1 =  getpid()  ;
}
printf("parent:  pid  =  %d",pid);  /*  C  */
printf("parent:  pid1 =  %d"  ,pid1); /*  D  */
wait(NULL);
}
```

Round Robin scheduler normally maintains a list containing exactly one instance of all runnable processes. What would happen if a process happens twice?

Why is it important for the scheduler to distinguish I /O bound programs from CPU bound programs?

In a multilevel queue scheduling algorithm, what is the advantage of having different time-quantum sizes at different levels?

Explain the differences in how much the following scheduling algorithms discriminate in favor of short processes:

- FCFS
- RR
- Multilevel feedback queues

Discuss how the following pairs of scheduling criteria conflict in certain settings.

CPU utilization and response time

Average turnaround time and maximum waiting time

Assume a disk drive has 100 tracks, from 0 to 99. It has just finished processing an I/O request that leaves its disk head at track 21. New I/O requests immediately arrive at the disk drive in the following order: 28,14,20,30,6

If it takes 2 time unit to move the disk head by 1 track, what is the total seek time to service all the requests if the disk uses:

- First-corne-first served scheduling
- Shortest -seek -time- first scheduling

What is the effect of setting large quantum on response time and turnaround goals of scheduling?

Suppose the following four processes (A, B, C, D) are submitted to the system in the specified time below. Assume these are the only active processes with zero context switch time. None of them ever blocks on I/O. Based on Non-preemptive shortest job first scheduling algorithm fill 'Ends at' and 'turnaround time' columns for each

Process	Submitted at	CPU time required	Ends at	Turnaround time
A	2:00:00	4 sec		
B	2:00:03	0.5 sec		
C	2:00:00	2 sec		
D	2:00:04	10 sec		

What is the optimistic assumption made in the deadlock-detection algorithm? How can this assumption be violated?Is it possible to have a deadlock involving only one process? Explain your answer.

Grading Scheme

This unit's assessments worth a total of 30% from which

1.Activity Assessments: 10%

2.Unit Assessment I: 15 %

3.Unit Assessment II (Lab Project): 5%

Feedback

Rubrics for assessing activities

Assessment Criteria	Doesn't attempt both assessment or the answers are all wrong	Attempt both assessments and provide partial answers or partially correct answers	Attempt all assessments with full and correct answer
Grading Scales	Fail	Got half mark for each assessment	Score full mark for each assessment

Unit Readings and Other Resources

Required readings and other resources:

- Andrew S. Tanenbaum (2008). Modern Operating System. 3rd dition. Prentice-Hall. Chapter 2 and 3
- A Silberschatz, Peter B Galvin, G Gagne (2009). Operating System Concepts. 8th Edition. Wiley. Chapter 3, 4, 5, 6, and 7

Optional readings and other resources for Process Scheduling

- William Stallings (2005). Operating Systems Internals and Design
- Principles. 4th edition. Prentice Hall.

Unit 3: Memory Management

How does the operating system manage memory?

Introduction

Every process needs to be stored in memory. Operating systems keeps track of every information of process, memory, file and I/O, as discussed on Unit 2. Processes, Files, and I/O has to be stored in the memory. One of the fundamental tasks of operating system is to allocate memory for processes, opened files, and I/O devices.

Unit Objectives

1. Upon completion of this unit you should be able to:
2. Elicit the importance of memory
3. Describe the difference and usage of swapping, and partitions
4. Work with the techniques of paging and segmentation
5. Effectively address the different page replacement algorithms and working sets
6. Explain the concept of caching

Key Terms

Swapping: Replacing a process in a memory with another one from a disk

Paging: Bringing and taking of pages a process in and out of a memory

Segmentation: Dividing parts of a process from programmer point of view

Page replacement algorithm: Algorithm used to identify a victim page to be evicted from the memory

Caching: A memory that keeps a subset of a data set in a more accessible but space-limited location

Address space: Sets of memory addresses

Learning Activities

Activity 3 Overview of Memory

Introduction

The other fundamental functionality of the operating system is memory management. Processes, opened files, and input/output devices require memory to operate. How does the operating manage to allocate memory in a multiprogramming environment?

Main memory (RAM) is one of the major component of a computer system. Every program to get executed must be loaded to the memory. Execution of the program happens once part or all of the program is loaded and the program becomes process. Hence, there are possibly number of processes running there will be an input queue of processes on the disk that wait to be brought to the memory. Obviously, programs needs to pass through several steps before they start to execute.

A program file or binary file is different from data or ASCII file in such a way that it has code (text) part, data part, and stack part. These parts are attached to the memory at different stages/times. Attaching these components of a program is known as binding. Address binding of code or text, data, and stack parts of the program to memory addresses occurs at the following stages:

Compile time: Address binding is done at program compilation time. Absolute code is generated if the available memory is known in advance. Stating memory address is changed, the source code must be recompiled.

Load time: If the memory location to bind is not known in advance, a relocate table code has to be generated at the loading time, i.e., the time when the program is brought in to the memory.

Execution time: With the support from hardware, for example, using base and limit registers, address maps can be done at execution time. Address binding can be delayed until execution time if the process has to be moved from segment to segment.

The steps of program execution is illustrated in the Fig 3.1.

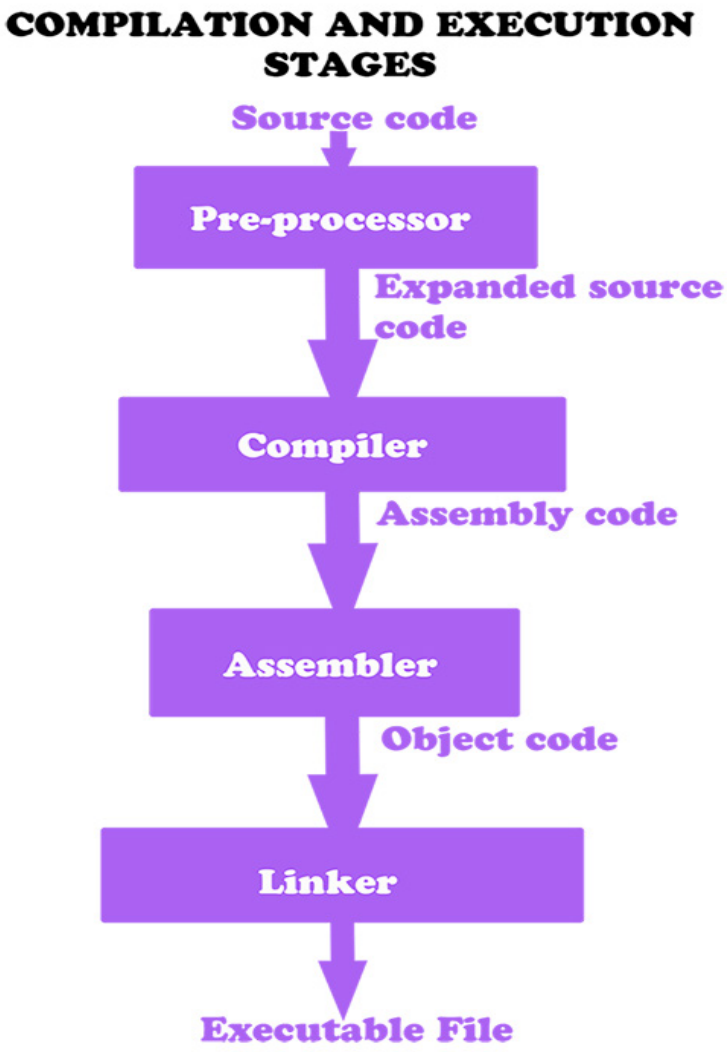


Fig. 3.1. Steps of program execution

Logical (Virtual) Address Space

Processes generate memory addresses to referencing its own memory location or others. These addresses generated by a process is known as logical or virtual addresses. Collection of such addresses generated by the process, is called logical address space. It starts from 0 to the size of the process.

Physical Address Space

The actual address generated by the memory management unit where the instruction and data is located on the physical memory. It can start anywhere a process is located. Collection of such addresses is referred as physical address space.

Logical and physical addresses are entirely similar in the two stages of address binding, i.e., on compile time and load time. They differ at the execution time of the process. Once the process to execute it produces the logical (virtual) address. This address must be translated to physical address to access the instruction and/or data. The hardware device that performs such a translation is known as Memory Management Unit (MMU). MMU maps logical (virtual) address to physical address.

Every process stores its start address of the process in the base/relocation register. When logical address is generated by the CPU, it is checked whether it is less than the content of the limit register. If it is in the MMU, the logical address is added to the value stored in the base/ relocation register to produce the physical address. Otherwise, the process is trying to the memory location of other processes, access is denied and a trap is generated. The illustration is shown in Fig 3.2. The user program doesn't know the value of the physical address.

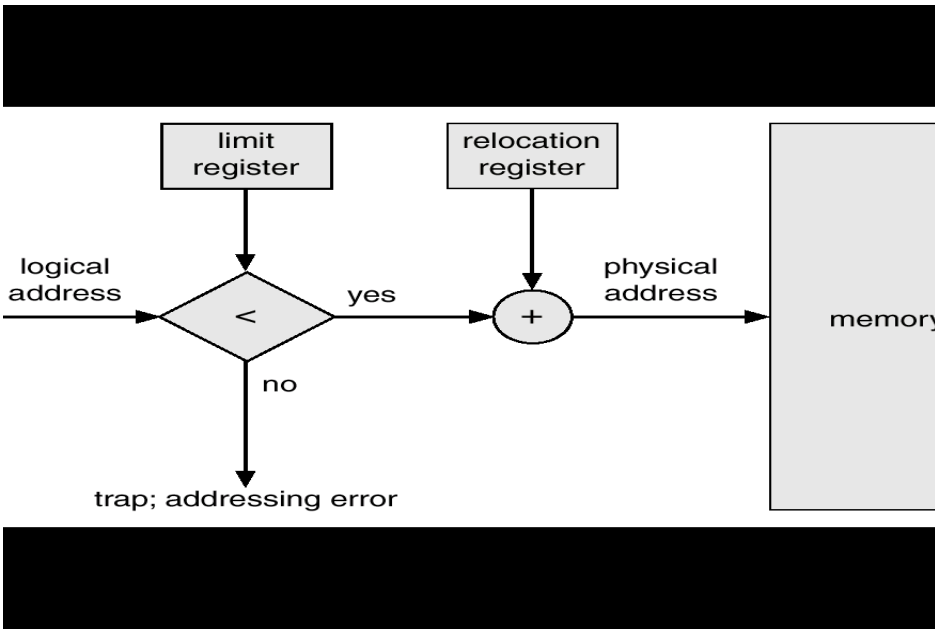


Fig. 3.2. MMU's mapping of virtual address to physical address.

For example, 510 is the logical address generated by the CPU. If 510 is greater than the content of limit register, a trap is generated for addressing error. Otherwise, it is added to the contents of the relocation register, i.e., 16000. The MMU produces 16510 as a physical address.

Conclusion

Programs go through different times before execution, namely compile, load, and execution times. Logical address are generated by the CPU while execution while physical addresses refers to the actual address of the instructions of the process on the memory. Translation from logical address to physical address is mandatory which is accomplished by the Memory management Unit (MMU).

Assessment

1. Describe the different parts of a program.
2. When does the address binding activity is done?
3. What is the purpose of limit register?

Activity 2 Swapping and partitions

Introduction

Concurrently running multiple processes on a uni-processor environment requires larger memory. How large the memory be? If we set the size always we want to run additional process that requires more memory. One cannot set the exact size of the memory that is large enough for all the processes on the system. The solution for this problem is swapping. What is swapping?

Swapping

*I*t is impractical to imagine the size of main memory to accommodate all the processes. Normally, if you check the number of processes Windows and Linux are executing at a time, more than 60 at start up. Maintaining all these processes in the primary memory is costly as large area of memory is needed. A process can be swapped temporarily out of the main memory and stored in the disk is the simplest strategy.

It is the responsibility of the memory manager to swap ideal processes fast enough in the memory. If you consider any CPU-scheduling algorithm, there is a mechanism that identifies a process that finishes its quanta recently, or has least priority, or least waiting time, or long execution time. In a specific scheduling algorithms, there are processes that are scheduled last. So, the memory manager swaps these processes to the disk in order to larger space in the main memory and be able to execute new processes.

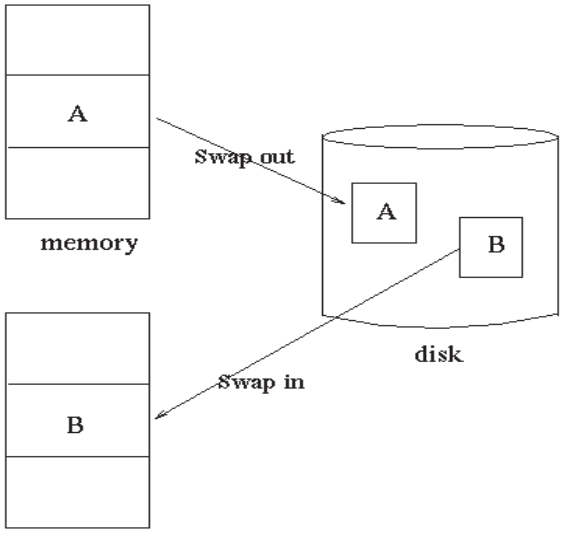


Figure 3.3. Block diagram for swapping of Process A and Process B

At the time when the need arises the swapped out processes will be swapped into the memory in its entirety. Idle processes are stored in a disk unless there is a need to run them, upon which they will be swapped in. Usually, the process replaces another swapped out process from the memory and claims the same address space it used to have previously for the purpose of address binding.

Partitions

Multiple processes reside in the main memory at a time. Main memory is partitioned into two: resident operating system, also known as kernel space, in low memory area and user space where user processes are held in high memory area.

Single Partition Allocation

Base/relocation register is used to hold the starting physical address for the user process. They provide protection for the operating system code and data from being changed by user processes. The logical address generated by user processes should not be greater than the limit register. Otherwise, the process is trying to address a memory location which is not part of its own address space. The partition size is fixed. The early IBM mainframe operating system, OS/MFT (Multiprogramming with Fixed Number of Tasks), was a successful single partition operating system.

Multiple Partition Allocation

The user space is divided into multiple partitions. The free partitions are referred to as holes. Holes have various size, which are scattered throughout the main memory. A process arriving to the memory will be allocated to a hole which is large enough to hold it.

The operating system is responsible to keep track of information about the allocated and free (hole) partitions.

Dynamic Memory Allocation

It is an algorithm that addresses how to allocate a process with n bytes to a list of free holes. The following are the three different algorithms:

First-fit: Starting from the first hole, scan for a hole which is large enough for the process to be allocated.

Best-fit: Scan for the entire holes in the list, allocated the process in a hole that produces a smallest leftover or internal fragmentation. It is best allocation algorithm but poor performance, since it has to scan all the free holes.

Next-fit: Scanning starts from the location of the last placement, and chooses the next available block that is large enough.

Worst-fit: Scan for the entire holes in the list, allocated the process in a hole that produces a largest leftover or internal fragmentation. It is worst allocation algorithm with poor performance, since it has to scan all the free holes.

First-fit and best-fit algorithms are better in terms of storage utilization.

Fragmentation

As a process is not large enough to take the entire space of a hole, there is a space which is unused. Such spaces are referred as fragmented spaces. There are two kinds of fragmentations:

- External fragmentation
- Internal fragmentation

External fragmentation

It refers to the total memory space which is left over when holes are assigned to processes. These spaces are not contiguous.

Internal fragmentation

The assigned process is not large enough to take the entire space of the hole producing an internal fragmentation.

To reduce external fragmentation a technique called compaction is used. Compaction moves all allocated memory spaces to one end of the main memory in order to produce one larger free memory block. It is expensive activity due to the effort it takes to move the content. Moreover, the relocation should be dynamic, since compaction is done at execution time.

Activity 3 Paging and Segmentation

Paging

Both single partition and multiple partitions allocation are inefficient in the use of memory; the former results in internal fragmentation, the latter in external fragmentation.

Suppose, however, that main memory is partitioned into equal fixed-size chunks that are relatively small known as frames or page frames (size is power of 2, between 512 bytes and 8192 bytes). And that each process is also divided into small fixed-size chunks of the same size known as pages. To run a program of size n pages, need to find n free frames and load program. In this approach, internal fragmentation happens only on a fraction of the last page of a process. There is no external fragmentation.

While processes occupy some of the frames, there are also free frames. A list of free frames is maintained by the operating system.

The page table shows a frame location for each page of a process. The logical address generated by the process consists of a page number and an offset within the page. A logical address is the location of a word relative to the beginning of the process.

The CPU translates the logical address into a physical address. Logical-to-physical address translation is done by Memory Management Unit (MMU). The processor uses the page table to convert the logical address logical address (page number, offset) to produce a physical address (frame number, offset).

Page number (p) - used to identify the page's index in a page table which contains base address of each page in physical memory.

Page offset (d) - united with start address to define the physical memory address that is sent to the physical memory.

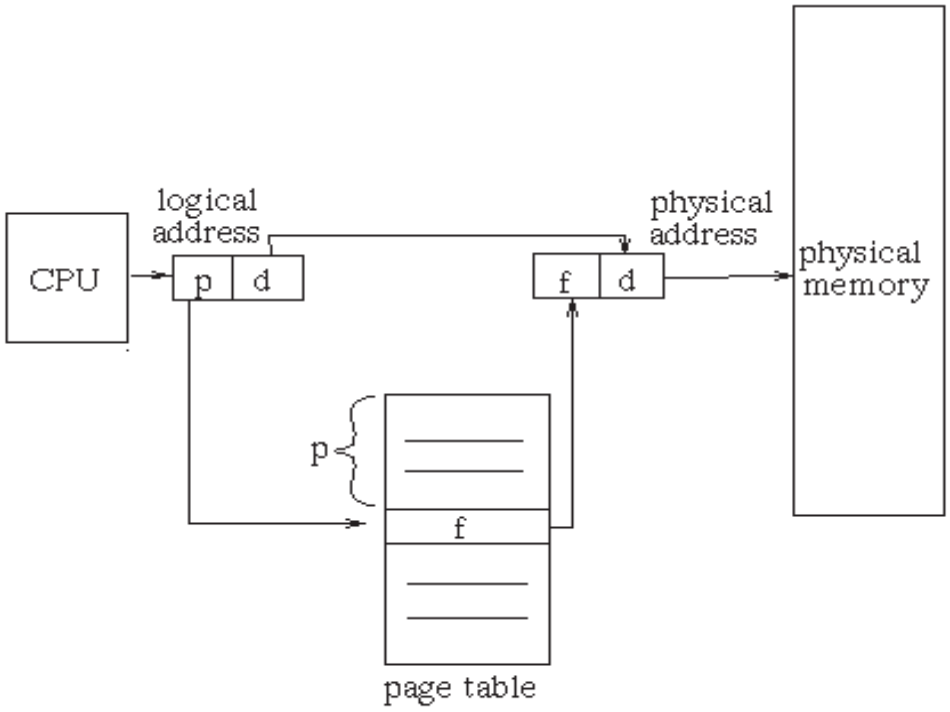


Figure 3.4. Block diagram for mapping logical address to physical address

Assume that the logical memory has four pages (page 0 to page 3). The physical memory has eight frames. The page table entry shows the mapping of the logical pages to the frames. Consider page size of 1KB. Let the logical address generated by the processor is 1300. To determine the frame where this page is located, divide the logical address by the page size, 1024, resulting page 1. Locating the frame that corresponds to page 1 is frame 6. The base address is the frame number times the frame size, which is equal to page size, results 6144.

To find the page offset, compute the remainder of the 1300 by page size. The page offset is 276. The physical address is obtained by adding the base address with the page offset, which is 6420.

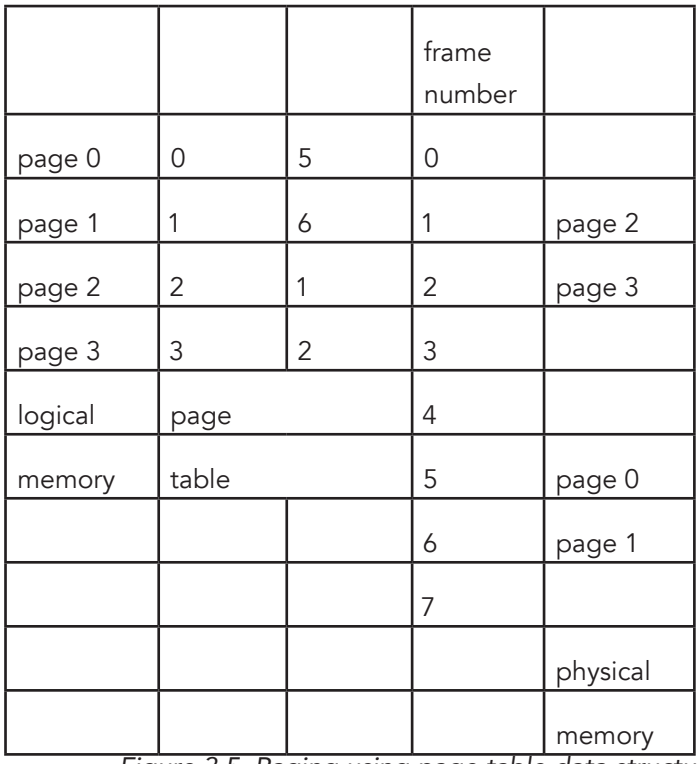


Figure 3.5. Paging using page table data structure

If the page do not have an entry in the page table, you cannot located the corresponding frame number. That shows the page is not loaded to the physical memory. In such cases, a page fault is said to happen and the operating system is notified of a trap. The process which results a page fault is blocked. Operating system loads the page that creates a page fault and brings the process to ready state.

Implementation of Paging

A process executes only in main memory that memory is referred to as real memory. In modern operating systems part of the hard disk can be used as memory. This memory is referred to as virtual memory. Virtual memory facilitates the effective management of multiprogramming.

In the case of simple paging, each process has its own page table, and when all of its pages are loaded into main memory, the page table for a process is created and loaded into main memory. Each page table entry contains the frame number of the corresponding page in main memory. A page table is also needed for a virtual memory scheme based on paging. Again, it is typical to associate a unique page table with each process. Because only some of the pages of a process may be in main memory, a bit is needed in each page table entry to indicate whether the corresponding page is absent/present bit (P) in main memory or not. If the bit indicates that the page is in memory, then the entry also includes the frame number of that page.

The page table entry includes a modify (M) bit, indicating whether the contents of the corresponding page have been altered since the page was last loaded into main memory. If there has been no change, then it is not necessary to write the page out when it comes time to replace the page in the frame that it currently occupies.

Caching disabled	Referenced	Protection	Absent/ Present Bit	Modify Bit	Page Frame Number

Figure Page table entry

Hierarchical Page Table

In systems with larger address space the one-level paging results in too long page tables which exceed the size of a page. Consequently they have to be paged as well as the program code and data. For example the 32-bit addressing system with 4 Kbyte (212) pages would have page tables with max 1 M entries. If each entry takes 4 bytes, the page table becomes 4 Mbytes long, or 4x220/ (212) = 1024 pages. Therefore a two-level paging scheme is needed.

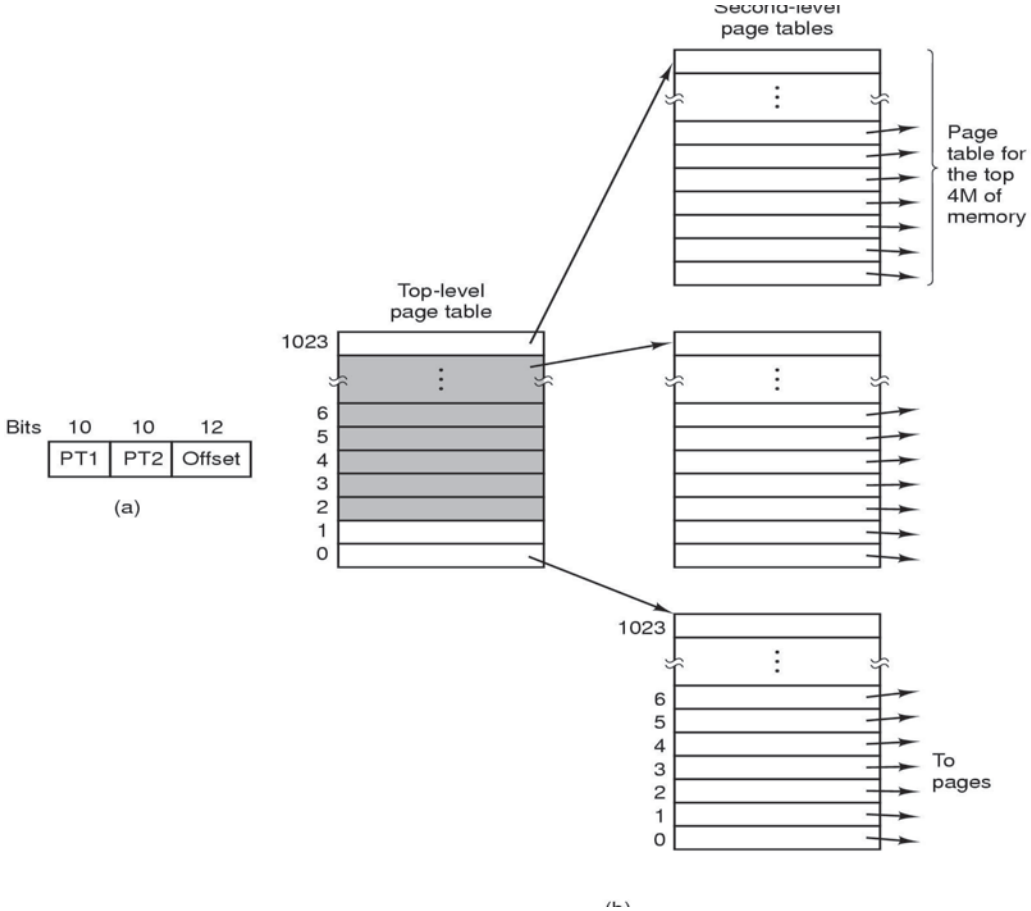


Figure 3.6. Two-level paging

This first level PT is small enough to store in memory, PT1 with 10-bits. It contains one PTE for every page of PTEs in the 2nd level PT, which reduces space by a factor of one or two thousand. But since we still have the 2nd level PT, PT2 also with 10-bits, we have made the world bigger not smaller.

Don't store in memory those 2nd level page tables all of whose PTEs refer to unused memory. That is use demand paging on the (second level) page table. This idea can be extended to three or more levels.

Address Translation with a 2-Level Page Table

For a two level page table the virtual address is divided into three pieces, as in Figure (a). PT1 gives the index into the first level page table. Follow the pointer in the corresponding PTE to reach the frame containing the relevant 2nd level page table.PT2 gives the index into this 2nd level page table. Follow the pointer in the corresponding PTE to reach the frame containing the (originally) requested page. Offset gives the offset in this frame where the originally requested word is located.

Inverted Page Tables

Usually, the logical address space is much bigger than the size of physical memory. In particular, with 64-bit addresses, the range is 264 bytes, which is 16 million terabytes. If the page size is 4KB and a page table entry is 4 bytes, a full page table would be 16 thousand terabytes.

A two level table would still need 16 terabytes for the first level table, which is stored in memory. A three level table reduces this to 16 gigabytes, which is still large and only a 4-level table gives a reasonable memory footprint of 16 megabytes.

An alternative is to instead keep a table indexed by frame number. The content of entry f contains the number of the page currently loaded in frame f. This is often called a frame table as well as an inverted page table.

Now there is one entry per frame. Again using 4KB pages and 4 byte PTEs, we see that the table would be a constant 0.1% of the size of real memory.

But on a Translation Lookaside Buffer (TLB) miss, the system must search the inverted page table, which would be hopelessly slow except that some tricks are employed. Specifically, hashing is used.

A Translation Lookaside Buffer or TLB is an associate memory where the index field is the page number. The other fields include the frame number, dirty bit, valid bit, etc.

Note that, unlike the situation with a page table, the page number is stored in the TLB; indeed it is the index field.

A TLB is small and expensive but at least it is fast. When the page number is in the TLB, the frame number is returned very quickly.

On a miss, a TLB reload is performed. The page number is looked up in the page table. The record found is placed in the TLB and a victim is discarded (not really discarded, dirty and referenced bits are copied back to the page table entry). There is no placement question since all TLB entries are accessed at the same time and hence are equally suitable. But there is a replacement question.

Segmentation

A user program is divided into segments. Rather than dividing the program into unstructured pages, it divides the program based on the different sections, segments. The size of the segments are not necessarily equal. As with paging, a logical address using segmentation consists of two parts, in this case a segment number and an offset.

Since segments are unequal in size, segmentation is similar to dynamic partitioning. The difference with dynamic partitioning is that with segmentation a program may occupy more than one partition. Moreover, these partitions need not be contiguous. Segmentation avoids internal fragmentation but, similar to dynamic partitioning, it suffers from external fragmentation.

Segmentation gives much more sense for programmers, compared to paging, in visualizing and organizing programs and data. Typically, the programmer or compiler will assign programs and data to different segments. For purposes of modular programming, the program or data may be further broken down into multiple segments.

A segment refers to a logical unit of a program which is code/text, stack, local variables, global variables, etc. Figure 5.2 illustrates the association of user space and physical memory allocation for segments of a program.

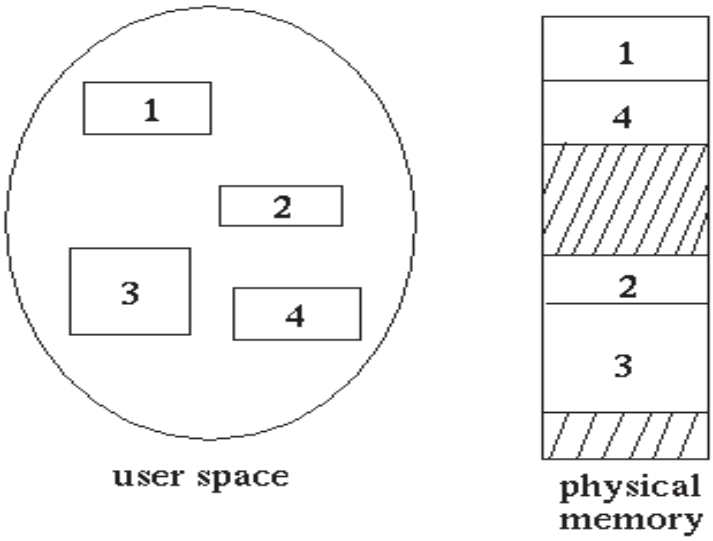


Figure 3.7. Segments in physical memory

In this case, there is no simple relationship between logical addresses and physical addresses. Similar to paging, logical address consists of segment number and offset. A segment table is used to map logical addresses to physical addresses. In the segment table, each segment table entry would have to give the starting address in main memory of the corresponding segment using the base register. The entry should also provide the length of the segment, to assure that invalid addresses are not used using the limit register.

Conclusion

The physical memory is not large enough to run several processes at a time. Different mechanisms were applied and memory allocation for processes is handled by the operating system. The mechanisms were swapping of processes in and out of memory, partitioning of memory, paging of processes, and segmentation. This unit dealt with detailed implementation issue of paging and segmentation techniques.

When a given page is brought to memory, another page should be evicted. How is a victim page selected? What algorithms do we have to select this victim page? The following activity addresses these questions.

Assessment

1. Explain the difference between single versus multiple partition allocation.
2. When does internal fragmentation occurs?
3. Explain what will happen if a page is not found in the physical memory.
4. Why hierarchical pages structure is required?

Activity 4 Page Replacement Algorithms

Introduction

In the previous activity, we demonstrated how pages are managed. If a page is found on the physical memory, the instruction continue to execute. But, if page fault happens, the required page shall be brought to the memory replacing a victim page. How is the victim page selected? We require an algorithm which is efficient enough in such a way that it minimizes subsequent page faults.

A page is brought into the memory when it is referenced. When a process first is started, there will be number of page faults. After a while, the number of page faults should drop to a very low level. The operating system is responsible to identify a victim page to be dispossessed from memory during a page fault.

These are solutions to the replacement a page in a memory question. A better solution take advantage of locality when choosing the victim page to replace.

Temporal locality: If a word is referenced now, it is likely to be referenced in the near future. This argues for caching referenced words, i.e. keeping the referenced word near the processor for a while.

Spatial locality: If a word is referenced now, nearby words are likely to be referenced in the near future. This argues for pre fetching words around the currently referenced word.

Temporal and spatial locality are lumped together into locality: If any word in a page is referenced, each word in the page is likely to be referenced. So it is good to bring in the entire page on a miss and to keep the page in memory for a while.

At the very beginning a program has no record to refer for locality. At this point the paging system is said to be undergoing a cold start.

Pages belonging to processes that have terminated are of course perfect choices for victims. Pages belonging to processes that have been blocked for a long time are good choices as well.

The Optimal Page Replacement Algorithm (PRA)

Replace the page whose next reference will be furthest in the future. Also known as Belady's minimum algorithm. Provably optimal. That is, no algorithm generates fewer page faults. The probability of implementing this algorithm is impractical, because it requires predicting the future. It has good upper bound on performance.

The Not Recently Used (NRU) PRA

Divide the frames into four classes and make a random selection from the lowest nonempty class.

- Not referenced, not modified.
- Not referenced, modified.
- Referenced, not modified.
- Referenced, modified.

Assumes that in each page table entry there are two extra flags R (for referenced) and M (for modified, or D, for dirty).

NRU is based on the belief that a page in a lower priority class is a better victim.

If a page is not referenced, locality suggests that it probably will not referenced again soon and hence is a good candidate for eviction.

If a clean page (i.e., one that is not modified) is chosen to evict, the operating system does not have to write it back to disk and hence the cost of the eviction is lower than for a dirty page.

How it works:

When a page is brought in, the operating system resets R and M (i.e. R=M=0).

On a read, the hardware sets R.

On a write, the hardware sets R and M.

Since every page is brought into the memory when referenced, its R bit is set. It must be cleared frequently. At every k clock ticks, the operating system re***sets all R bits.***

First In, First Out (FIFO) PRA

Belady's Anomaly: Can have more frames yet generate more faults. The natural implementation is to have a queue of nodes each referring to a resident page (i.e., pointing to a frame).When a page is loaded, a node referring to the page is appended to the tail of the queue. When a page needs to be evicted, the head node is removed and the page referenced is chosen as the victim.

This sound good, but only at first. The trouble is that a page referenced say every other memory reference and thus very likely to be referenced soon will be evicted because we only look at the first reference.

Second chance PRA

Similar to the FIFO PRA, but altered so that a page recently referenced is given a second chance.

When a page is loaded, a node referring to the page is appended to the tail of the queue. The R bit of the page is cleared. When a page needs to be evicted, the head node is removed and the page referenced is the potential victim.

If the R bit is unset (the page hasn't been referenced recently), then the page is the victim .If the R bit is set, the page is given a second chance. Specifically, the R bit is cleared, the node referring to this page is appended to the rear of the queue (so it appears to have just been loaded), and the current head node becomes the potential victim.

What if all the R bits are set?

We will move each page from the front to the rear and will arrive at the initial condition but with all the R bits now clear. Hence we will remove the same page as FIFO would have removed, but will have spent more time doing sow might want to periodically clear all the R bits so that a long ago reference is forgotten (but so is a recent reference).

Clock PRA

Same algorithm as second chance, but a better potentially implementation for the nodes: Use a circular list with a single pointer serving as both head and tail.

Let us begin by assuming that the number of pages loaded is constant.

So the size of the node list in second chance is constant.

Use a circular list for the nodes and have a pointer pointing to the head entry. Think of the list as the hours on a clock and the pointer as the hour hand. (Hence the name clock PRA.)

Since the number of nodes is constant, the operation we need to support is replace the oldest, unreferenced page by a new page.

Examine the node pointed to by the (hour) hand. If the R bit of the corresponding page is set, we give the page a second chance: clear the R bit, move the hour hand (now the page looks freshly loaded), and examine the next node

Eventually we will reach a node whose R bit is clear. The corresponding page is the victim.

Replace the victim with the new page (may involve 2 I/Os as always).

Update the node to refer to this new page.

Move the hand forward another hour so that the new page is at the rear.

Thus, when the number of loaded pages (i.e., frames) is constant, the algorithm is just like second chance except that only the one pointer (the clock hand) is updated.

The number of frames can change when we use a so called local algorithm where the victim must come from the frames assigned to the faulting process. In this case we have a different frame list for each process. At times we want to change the number of frames assigned to a given process and hence the number of frames in a given frame list changes with time.

How does this affect 2nd chance?

We now have to support inserting a node right before the hour hand (the rear of the queue) and removing the node pointed to by the hour hand.

The natural solution is to double link the circular list.

In this case insertion and deletion are a little slower than for the primitive second chance (double linked lists have more pointer updates for insert and delete).

So the trade-off is: If there are mostly inserts and deletes, and granting second chances is not too common, use the original 2nd chance implementation. If there are mostly replacements, and you often give nodes a 2nd chance, use clock.

Last In, First Out (LIFO) PRA

All but the last frame are frozen once loaded so you can replace only one frame. This is especially bad after a phase shift in the program as now the program is references mostly new pages but only one frame is available to hold them, which is bad.

Least Recently Used (LRU) PRA

When a page fault occurs, choose as victim that page that has been unused for the longest time, i.e. the one that has been least recently used.

LRU is definitely:

Implementable: The past is knowable.

Good: *Simulation studies have shown this.*

Difficult. Essentially the system needs to either:

Keep a time stamp in each page table entry, updated on each reference and scan all the page table entries when choosing a victim to find the page table entry with the oldest timestamp.

Keep the page table entries in a linked list in usage order, which means on each reference moving the corresponding PTE to the end of the list.

Simulating (Approximating) LRU in Software

The Not Frequently Used (NFU) PRA

Keep a count of how frequently each page is used and evict the one that has been the minimum score. Specifically:

- Include a counter (and reference bit R) in each page table entry.
- Set the counter to zero when the page is brought into memory.

- Every k clocks, perform the following for each page table entry.
- Add R to the counter.
- Clear R.
- Choose as victim the page table entry with minimum count.

The Aging PRA

NFU doesn't distinguish between old references and recent ones. The following modification does distinguish.

- Include a counter (and reference bit, R) in each PTE.
- Set the counter to zero when the page is brought into memory.
- Every k clock ticks, perform the following for each PTE.
- Shift the counter right one bit.
- Insert R as the new high order bit of the counter.
- Clear R.
- Choose as victim the PTE with lowest count.

Aging does indeed give more weight to later references, but an n bit counter maintains data for only n time intervals; whereas NFU maintains data for at least 2n intervals.

Working Sets and Thrashing

Normally, if a process takes a page fault and must wait for the page to be read from disk, the operating system runs a different process while the I/O is occurring. Thus page faults are "free"?

1. **What happens if memory gets overcommitted?**

Suppose the pages being actively used by the current processes don't all fit in physical memory.

Each page fault causes one of the active pages to be moved to disk, so another page fault will occur soon.

The system will spend all its time reading and writing pages, and won't get much work done.

This situation is called thrashing; it was a serious problem in early demand paging systems.

2. **How to deal with thrashing?**

If a single process is too large for memory, there is nothing the operating system can do. That process will simply thrash. If the problem arises because of the sum of several processes:

Figure out how much memory each process needs.

Change scheduling priorities to run processes in groups that fit comfortably in memory: must shed load.

Working Sets: conceptual model proposed by Peter Denning to prevent thrashing.

Informal definition: the collection of pages a process is using actively, and which must thus be memory-resident to prevent this process from thrashing.

If the sum of all working sets of all runnable process's threads exceeds the size of memory, then stop running some of the threads for a while.

Divide processes into two groups: active and inactive:

When a process is active its entire working set must always be in memory: never execute a thread whose working set is not resident. When a process becomes inactive, its working set can migrate to disk. Threads from inactive processes are never scheduled for execution. The collection of active processes is called the balance set. The system must have a mechanism for gradually moving processes into and out of the balance set. As working sets change, the balance set must be adjusted.

5. **How to compute working sets?**

Denning proposed a working set parameter T: all pages referenced in the last T seconds comprise the working set. Can extend the clock algorithm to keep an idle time for each page. Pages with idle times less than T are in the working set.

Difficult questions for the working set approach:

a. How long should T be (typically minutes)?

b. How to handle changes in working sets?

c. How to manage the balance set?

d. How to account for memory shared between processes?

Page Fault Frequency: *a*nother approach to preventing thrashing. Per-process replacement; at any given time, each process is allocated a fixed number of physical page frames. Monitor the rate at which page faults are occurring for each process. If the rate gets too high for a process, assume that its memory is overcommitted; increase the size of its memory pool. If the rate gets too low for a process, assume that its memory pool can be reduced in size. If the sum of all memory pools don't fit in memory, deactivate some processes.

In practice, today's operating systems don't worry much about thrashing: With personal computers, users can notice thrashing and handle it themselves:

Typically, just buy more memory

Or, manage balance set by hand

Thrashing was a bigger issue for timesharing machines with dozens or hundreds of users: Why should I stop my processes just so you can make progress? System had to handle thrashing automatically. Technology changes make it unreasonable to operate machines in a range where memory is even slightly overcommitted; better to just buy more memory.

Caching

A cache keeps a subset of a data set in a more accessible but space-limited location. Caches are everywhere in systems, such as, web proxy servers make downloads faster and cheaper, web browser stores downloaded files, registers are a cache for L1 cache which is cache for L2 cache, etc..

The main goal of caching is minimize cache miss rate. In the context of paging, it is minimizing page fault rate. It requires a good algorithm.

We create caches because:

a. There is not enough fast memory to hold everything we need

b. Memory that is large enough is too slow

c. Performance metric for all caches is EAT (Effective Access Time). Goal is to make overall performance close to cache memory performance.

d. By taking advantage of locality — temporal and spatial

e. By burying a small number of accesses to slow memory under many, many accesses to fast memory

Summary

One of the fundamental and complex tasks of an operating system is memory management. Memory management involves treating main memory as a resource to be allocated to and shared among a number of active processes. To use the processor and the I/O facilities efficiently, it is desirable to maintain as many processes in main memory as possible. In addition, it is desirable to free programmers from size restrictions in program development.

The basic tools of memory management are paging and segmentation. With paging, each process is divided into relatively small, fixed-size pages. Segmentation provides for the use of pieces of varying size. It is also possible to combine segmentation and paging in a single memory management scheme.

To use the processor and the I/O facilities efficiently, it is desirable to maintain as many processes in main memory as possible. In addition, it is desirable to free programmers from size restrictions in program development. The way to address both of these concerns is virtual memory. With virtual memory, all address references are logical references that are translated at run time to real addresses. This allows a process to be located anywhere in main memory and for that location to change over time. Virtual memory also allows a process to be broken up into pieces. These pieces need not be contiguously located in main memory during execution and, indeed, it is not even necessary for all of the pieces of the process to be in main memory during execution.

Two basic approaches to providing virtual memory are paging and segmentation. With paging, each process is divided into relatively small, fixed-size pages. Segmentation provides for the use of pieces of varying size. It is also possible to combine segmentation and paging in a single memory management scheme.

A virtual memory management scheme requires both hardware and software support. The hardware support is provided by the processor. The support includes dynamic translation of virtual addresses to physical addresses and the generation of an interrupt when a referenced page or segment is not in main memory.

Unit Assessment

Check your understanding!

Assessment: Multiple Choice Questions

Instructions

Select the best answer among the choices

1. A page fault occurs when

    a. The deadlock happens

    b. The segmentaion starts

    c. The page is found in the memory

    d. The page is not found in the memory

2. Bringing a page into memory only when it is needed, this mechanism is called

    a.Deadlock

    b.Page fault

    c.Inactive paging

    d.Demand paging

3. Bringing a process from memory to disk to allow space for other processes is called

    a.Swapping

    b.Demand paging

    c.Deadlock

    d.Page fault

4. Which of the following memory allocation scheme suffers from external fragmentaion ?

   a.Segmentation

   b.Demand paging

   c.Swapping

   d.Paging

5. Page fault frequency in an operating system is reduced when the

   a.Processes tend to be of an equal ration of the I/O-bound and CPU-bound

   b.Size of pages is increased

   c.Locality of references is applicable to the process

   d.Processes tend to be CPU-bound

6. What is demand paging?

   a.A policy for determining which page to replace.

   b.Loading a page into memory only on a page-fault.

   c.Starting a process with all of its pages resident in physical memory.

   d.Discarding the least-recently-used (LRU) page in the system.

7. When would you recommend using an inverted page table?

   a.When the maximum amount of physical memory is much less than the maximum logical address space.

   b.When the maximum amount of physical memory is much greater than the maximum logical address space.

   c.When you want to support page sharing across different address spaces.

   d.None of the above

8. Why does the OS want to know the working set for each process? Circle the best answer and briefly explain your reasoning.

   a.To determine the best page to replace.

   b.To know when memory is over-committed.

   c.Assessment: Essay Type Questions

Instructions

1. Write your answers for the following questions
2. What is the job of Memory Management Unit (MMU) ?
3. What is the basic objective of page replacement algorithm ?
4. Consider the page replacement policies of OPRA, FIFO, and LRU. Which of the following statements are true? Be careful to notice when the phrase states "better than or equal to" versus "strictly better than".

    a. OPT always performs better than or equal to LRU.

    b. OPT always performs strictly better than LRU.

    c. LRU always performs better than or equal to FIFO.

    d. LRU always performs strictly better than FIFO.

    e. OPT with n+1 pages of physical memory always performs better than or equal to OPT with n pages.

    f. OPT with n+1 pages of physical memory always performs strictly better than OPT with n pages.

    g. FIFO with n+1 pages of physical memory always performs better than or equal to FIFO with n pages.

    h. FIFO with n+1 pages of physical memory always performs strictly better than FIFO with n pages.

    i. LRU with n+1 pages of physical memory always performs better than or equal to LRU with n pages.

    j. LRU with n+1 pages of physical memory always performs strictly better than LRU with n pages.

Grading Scheme

This unit's assessments worth a total of 20% from which

a. Activity Assessments: 5%

b. Unit Assessment I: 10 %

c. Unit Assessment II (Lab Project): 5%

Feedback

Rubrics for assessing activities

Assessment Criteria	Doesn't attempt both assessment or the answers are all wrong	Attempt both assessments and provide partial answers or partially correct answers	Attempt all assessments with full and correct answer
Grading Scales	Fail	Got half mark for each assessment	Score full mark for each assessment

Unit Readings and Other Resources

Required readings and other resources:

- Andrew S. Tanenbaum (2008). Modern Operating System. 3rd Edition. Prentice-Hall. Chapter 4
- A Silberschatz, Peter B Galvin, G Gagne (2009). Operating System Concepts. 8th Edition. Wiley. Chapter 8 and 9

Optional readings and other resources for Page Replacement Algorithms:

- William Stallings (2005). Operating Systems Internals and Design *Principles. 4th edition. Prentice Hall.*

Unit 4: Device Management and File Systems

What are the objectives of the operating system while it manages devices?

Introduction

I/O device management is a very important activity of the operating system. I/O is important for the communication of users to computer. The following are services of the operating system as I/O manager:

- Monitoring all the I/O devices
- Order the I/O devices, capture interrupts and manage bugs related to I/O
- Avail communication channel between I/O devices and all other hardware components

Accessing and storing information is the task of every computer application. A clear and obvious requirement of an operating system is thus, the provision of a convenient, efficient, and robust information handling system. A process can use its address space to store some amount of information. However, three main problems are associated with this method.

- One is the adequacy of the space to accommodate all information to be stored by the application as the size of the address space is determined by the size of the virtual address space.
- The second problem is the data loss as the process terminates the information kept on its address space is also lost though the information is required to be retained for long period of time.
- The third problem is concurrent accessibility of the information by other processes as information saved in one process's address space is accessible only to that process and sometimes there is a need to make this information as whole or part of it available to other processes as well. Solving these problems by separately managing information resulted by a process from its process is a concern of any operating system which is usually done by storing the information on external media in units called files.

The operating system manages naming, structure, access, use, protection and implementation of these files. Thus, component of an operating system and monitors activities related with files is known as a file system which is to be addressed in this section. File management system consists of system utility programs that run as privileged applications concerned with secondary storages.

The module is organized as follows. We start by discussing what a file system is followed by discussion about the file system hierarchy. The concept of directories and the file system implementation will be explained at last.

Unit Objectives

*U*pon completion of this unit you should be able to:

1. State the characteristics of input/output devices
2. Address the principles of input/output hardware and software
3. Define and express the concept of file systems and how a file can be logically organized
4. Explain the organization of the directory, the FAT and the data area on a single partition
5. Discuss details of file system and directory implementations

Key Terms

Input: raw data that is to be given to a computer for processing

Output: processed information delivered to users or other processes

File: collection of data created by users

Directory: A strutured organization of files

File System: core component of an operating system that handles data organization as files and defines collection of functions that can be performed on files

Learning Activities

Activity 1 Characteristics of I/O Devices

Introduction

The I/O devices are varied that operating systems devotes a subsystem to handle the variety. The range of devices on a modern computer system include from mice, keyboards, disk drives, display adapters, USB devices, network connections, audio I/O, printers, special devices for the handicapped, and many special-purpose peripherals. These devices can be roughly categorized as storage, communications, and user-interface. The peripheral devices can communicate with the computer via signals sent over wires or through the air and connect with the computer via ports, e.g. a serial or parallel port. A common set of wires connecting multiple devices is termed as a bus.

Device drivers are modules that can be plugged into an OS to handle a particular device or category of similar devices.

Principles of I/O hardware

I/O devices have three sections:

- I/O devices: - concerned with the way data are handled by the I/O device. There are two types of I/O devices known as blocked and character devices.
- Blocked devices (such as disks) - are devices with fixed-size slots having unique addresses and stores data accordingly.. Byte ranges of 512 to 32,768 are the general block size ranges. The ability to read and write each blocks regardless of the other blocks is an important feature of these devices. A disk is the commonly known block device which allows moving the read/write arm any time to any required cylinder position and awaits for the needed block to spin under the head
- Character devices (such as printers, mouse and NIC) - The other type of I/O device is the character device. A character device delivers or accepts stream of character, without regard to any block structure. It is not addressable and does not have any seek operation.

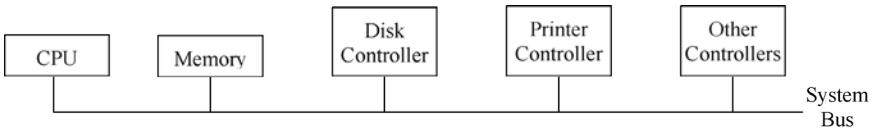


Figure 4. 1 1. Device

I/O Unit:

indicates the hardware components. There are two major components – Electronic Component (Device controller/Adapter) and the Mechanical Component

Memory-mapped I/O is a technique for communicating with I/O devices.

In this case a certain portion of the processor's address space is mapped to the device, and communications occur by reading and writing directly to/from those memory areas.

Memory-mapped I/O is suitable for devices which must move large quantities of data quickly, such as graphics cards.

Memory-mapped I/O can be used either instead of or more often in combination with traditional registers. For example, graphics cards still use registers for control information such as setting the video mode.

A potential problem exists with memory-mapped I/O, if a process is allowed to write directly to the address space used by a memory-mapped I/O device.

Direct Memory Access (DMA): is a technique for moving a data directly between main memory and I/O devices without the CPU's intervention. In the absence of DMA reading from a disk is done with the following steps:

the controller serially fetches every bit from the block which may have one or more sectors and maintains the whole data in its internal buffer

The controller then checks for a read error by computing the checksum value of the data

The controller then sends an interrupt signal to the system

The OS reads the information found in the controller's buffer, which is the disk block byte by byte or word by word and put it on memory

Problem: Since the OS controls the loop of reading, it wastes the CPU time. On the other hand, when DMA is used the Device Controller will do the counting and address tracking activities.

The following diagram shows the steps to use DMA:

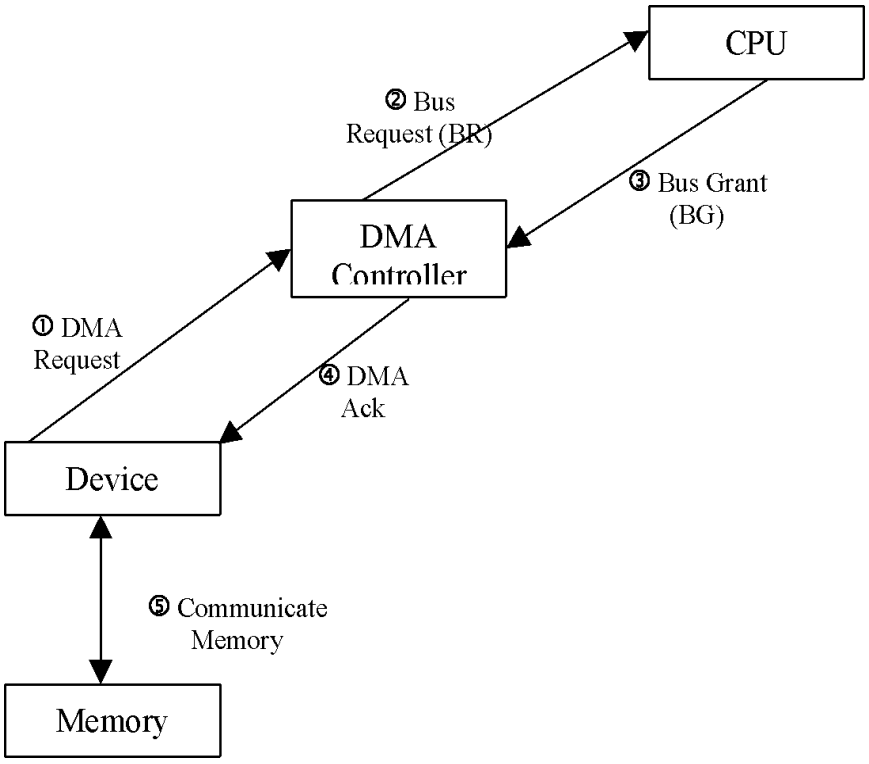


Figure 4.2. Steps of DMA between main memory and I/O devices

Principles of I/O Software

Layered technique is used

Goals and issues of I/O software:

Device Independence: It should be possible to write programs that can read files on a floppy disk, on hard disk, or on a CD-ROM, without having to modify the program for each different device types. It is up to the operating system to take care of the problems caused by the fact that these devices are really different.

Uniform Naming: file or device names can be any string or number identifier which has no dependency to the file or device, what so ever.

Error handling: Broadly speaking, errors need to be dealt with at the device controller level. Most of the errors to be observed are temporary like for instance read errors due to dust spots on the read head and can be solved by performing the needed operation repetitively.

Transfer: There are two types of transfer modes – Synchronous (Blocking) and Asynchronous (interrupt –driven). In the case of synchronous transfer the program requesting I/O transfer will be suspended until the transfer is completed. In the case of Asynchronous transfer the CPU starts the transfer and goes off to do something until the interrupt that shows the completion of the transfer arrives.

Device types: There are two device types –

Sharable (such as disk) devices- accessed by multiple users simultaneously. Nothing is wrong if several users try to open a file from one disk concurrently.

Dedicated (tape drives) have to be dedicated to a single user until that user is finished. Having two or more users writing blocks intermixed at random to the same tape will definitely not work

Layers of I/O software

The following are the I/O software layers

- Interrupt handler (bottom)
- Device driver
- Device independent OS software
- User-level software (top)

Interrupt Handler

Interrupts are undesirable and inevitable situations in life but can be hidden away. One of the methods used to hide interrupts is through blocking all processes with an I/O operation till the I/O is finished and an interrupt happens. The interrupt method will then need to perform various actions and unblock the process which started it.

Device Driver

All device – dependent code goes in the device driver

Only one device type, or at most, one group of closely linked devices is handled through each device driver.

Each device driver handles one device type, or at most, one class of closely related devices.

Each controller has one or more device registers used to give it command

The device driver is responsible to give these commands and ensure their proper execution

Thus, the disk driver is the only part of the OS that knows how many registers that disk controller has and what they are used for.

Generally, we can say that a device driver is responsible to get requests from the software, which is device independent, and issue orders for the execution of the requests.

Steps in carrying out I/O requests:

- Translate the requests from abstract to concrete terms
- Write the interpreted requests into registers of the device controller's
- The device driver blocks itself until an interrupt comes which awakens or unblocks the driver
- The Device driver starts its operations by first testing for any error on the I/O device
- If no error is identified and everything is properly functioning, data is passed from the driver to the requester software
- If the driver couldn't get any request in the queue, it will move back to block state and waits until a request comes in.

Device Independent I/O Software

It is large fraction of I/O software

Services it provides are:

Uniform interfacing for device drivers – perform I/O function common to all drives

Device Naming – responsible for mapping symbolic devices names onto the proper driver

Device protection – secures devices by blocking illegitimate or non-allowed access requests of users

Providing device-independent block size – provide uniform block size to higher layers hiding differences in block sizes

Buffering: if a user process write half a block, the OS will normally keep the data in buffer until the rest of the data are written. Keyboard inputs that arrive before it is needed also require buffering.

Storage allocation on block devices: when a file is created and filled with data, new disk blocks have to be allocated to the file. To perform this allocation, the OS needs a list of free blocks and used some algorithms for allocation

Allocating and releasing dedicated devices: The OS is solely responsible to validate any device request and act accordingly by either granting or denying the service.

Error reporting: Errors handling, by and large, is done by drivers. Most errors are device dependent. When a device request comes to the driver, it attempts to communicate the requested block certain number of times.

If it is unable to read the block during these times, it stops trying and communicate the requester software the status and finally reports to the caller

User Space I/O Software

A situation where a small portion of the I/O software is outside the OS

In this case, library procedures are used to perform system calls including I/O system calls. For instance, if a C program has a function call count=write(fd, buffer, n bytes); the write procedure from the library will be linked with it and kept in the binary program which will be loaded to memory during execution

The library procedures are also responsible to format the I/O requests

The following is an example how the I/O system works during reading by an application

Step 1: system call to do file reading is passed from the user program.

Step 2: the device-independent software checks the block cache; if the requested block is found there, the device driver will be called.

Step 3: The device driver issue the request to the hardware and the requesting user process will be moved to a block state until the disk operation is finished

Step 4: the disk then generates an interrupt once it finishes the operation

Step 5: The interrupt handler immediately takes over and investigate the interrupt i.e., it first checks for the device currently requiring attention. It then reads the output of the device and unblocks the sleeping process indicating the I/O process is being completed and let the user process continue

The following table 4.1. shows the I/O system layers along with the major responsibilities of each layer.

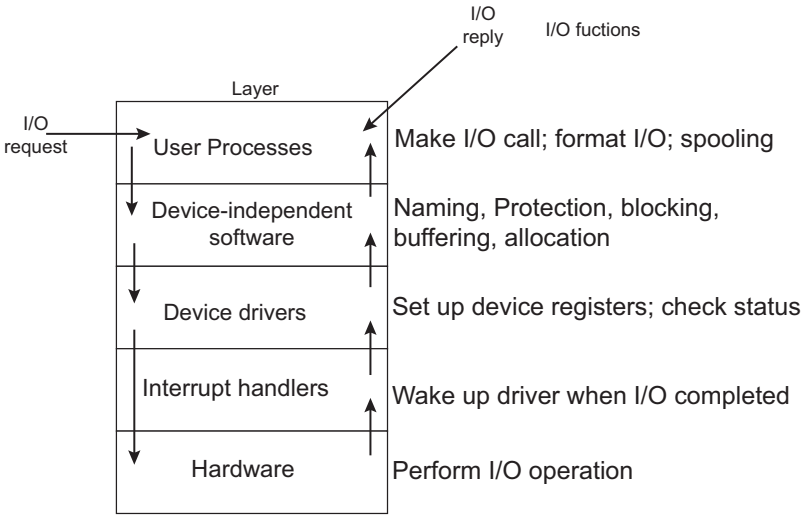


Table 4.1. Input Output layers

Disk

All real disks are organized into cylinders, each one containing many tracks. Each of the tracks then will be divided into sectors (equal number of sectors or different number of sectors)

In the case of equal number of sectors

The data density as closer to the center (hub) is high

The speed increases as the read/write moves to the outer tracks

Modern large hard drives have more sectors per track on outer tracks e.g. IDE drives

Many controllers, except floppy disk controllers, are capable of doing a read or write operation on one drive and also seek operation on one or more other drives simultaneously.

Disk Access Time

Three factors determine the time required to read or write a disk block :

The seek time (the time to move the arm to the proper cylinder)

The rotational delay (the time for the proper sector to rotate under the head)

The actual data transfer time

For most disks, the seek time dominates the other two times, so reducing the mean seek time can improve system performance substantially.

Disk requests can come from processes while the arm is doing a seek operation of another process. A table of waiting disk requests is kept by disk drivers. The table is indexed by cylinder number and pending requests of each cylinder is linked through a linked list that is headed by the table entries.

Disk Arm Scheduling Algorithm

The OS maintains queue of requests for each I/O operation and it uses various disk scheduling algorithms. To mention some:

First Come First Served (FCFS) : Accept single request at a time and perform the requests in same order

E.g. Track initial position: 11

Track request: 1,36,16,34,9,12

Service order: 1,36,16,34,9,12

Arm motion required: 10, 35, 20, 18, 25, 3,

Total= 111 tracks

The simplest and the fairest of all, but it doesn't improve performance

Shortest Seek First (SSF): It handles the closest (the least disk arm movement) request next, to minimize seek time.

E.g. Track initial position: 11

Track request: 1,36,16,34,9,12

Service order: 12,9,16, 1, 34, 36

Arm motion required: 1, 3, 7, 15, 33, 2

Total= 61 tracks

Advantage: Performance (efficiency), provides better performance

Disadvantage: Possibility of starvation (it lacks fairness)

SCAN (Elevator) Algorithm : The disk arm moves in one direction till it couldn't find any more request in that direction, then switches to another direction

Direction bit 1= up, 0=down

E.g. Track initial position: 11

Track request: 1,36,16,34,9,12

Direction bit: 1

Service order: 12, 16, 34, 36, 9, 1

Disk arm motion: 1, 4, 18, 2, 27, 8

Total= 60 tracks

It provides better service distribution

C-SCAN (Modified Elevator) Algorithm

It restricts scanning to one direction only. This is a bit modified version of an elevator algorithm and exhibits smaller variance in response times as it always scan in the same direction. When the highest numbered cylinder with a pending request has been serviced, the arm goes to the lowest-numbered cylinder with a pending request and then continues moving in an upward direction. In effect, the lowest-numbered cylinder is thought of as being just above the highest-numbered cylinder. It reduces the maximum delay experienced by new request.

RAM Disk

A RAM disk has the advantage of having instant access

Unix support mounted file system but DOS and Windows do not support

The RAM disk is split up into n blocks each with a size equal to the real disk

Finally the transfer will be done

A RAM disk driver may support several areas of memory used as RAM disk

Disk Cache

Memory cache – To narrow the distance between the processor and memory.

Disk cache – To narrow the distance between the processor/ memory and I/O

It uses a buffer kept in main memory that functions as a cache of disk memory and the rest of the main memory.

It contains a copy of some of the sectors on the disk.

It improves performance (by minimizing block transfer. Disk memory

Design issue:

Data transfer - Memory-to-memory

- using shared memory (pointer)

Replacement algorithm - Least Recently used

- Least Frequently Used

Conclusion

- I/O devices are interfaces that communicate users with the computer system. To manage the communication effectively, the operating system uses the I/O subsystem, which has a complete layer of hardware and software.
- The I/O function is generally broken up into a number of layers, with lower layers dealing with details that are closer to the physical functions to be performed and higher layers dealing with I/O in a logical and generic fashion. The layering is done in such a way that changes that a specific layer won't affect the other layers.
- The aspect of I/O that has the greatest impact on overall system performance is disk I/O. Two of the most widely used approaches to improve disk I/O performance are disk scheduling and the disk cache.
- At any time, there may be a queue of requests for I/O on the same disk. It is the object of disk scheduling to satisfy these requests in a way that minimizes the mechanical seek time of the disk and hence improves performance. The physical layout of pending requests plus considerations of locality come into play.
- A disk cache is a buffer, usually kept in main memory that functions as a cache of disk blocks between disk memory and the rest of main memory. Because of the principle of locality, the use of a disk cache should substantially reduce the number of block I/O transfers between main memory and disk.

Assessment

1. Explain the I/O software goals

2. Discuss the two most widely used approaches to improve disk I/O performance

3. What is DMA? What advantages does it provide to the operating system?

Activity 2 Overview of file systems

Introduction

A file is a named collection of related information that are treated as a single entity defined by its creator and kept on secondary storage devices that can be accessed by users or applications through file management systems. In this activity, we will discuss how files are handled by the operating system, the naming scheme, access methods, and operations carried out on a file.

A file is a contiguous logical storage unit defined by the operating system by abstracting the physical properties of storage devices so that the computer system will be convenient to use by providing a uniform logical view of information storage. File is the most visible aspect of an OS which is used to store and retrieve information from the disk

Files are mapped by the operating system onto physical devices. These storage devices are usually non-volatile making the contents persistent through power failures and system reboots. Files represent both programs and data where data files may be numeric, alphabetic, alphanumeric, or binary. Files can also be either in free form or rigidly formatted. Every file has a name associated with it which is used to interact with. Operating systems provide a layer of system-level software, using system-calls to provide services relating to the provision of files which avoids the need for each application program to manage its own disk allocation and access. File manager component of the operating system is responsible for the maintenance of files on secondary storages as memory manager is responsible for the maintenance of primary memory. Let's discuss some properties of files from user's point of view.

File naming

Files are abstraction mechanism use by a computer system and naming is an important aspect of a good abstraction mechanism. Name is assigned to a file by the creating process at the time of creation which is then used by other processes to communicate with the file after the creating process terminates. A file is named, for the convenience of its human users, and is referred to by its name. A string of eight or more characters is used to label a file in most operating system though digits and some special symbols are allowed in some situations. Some operating systems, like UNIX, make distinctions between uppercase and lowercase file names where as others like MS-DOS do not. A file name has two parts separated by a period.

The first part is the label for the file while the last part is the extension indicating the type of the file and with which the operating system identifies the owner program for that file. Thus, opening the file will start the program assigned to its file extension using the file as a parameter.

File structure

A file is a sequence of bits, bytes, lines, or records, the meaning of which is defined by the file's creator and user. A file can be structured in several ways among which these three types are common.

Byte sequence

The file is organized in unstructured sequence of bytes where the operating system is unaware of the file content. Meaning of the bytes is imposed by user programs providing maximum flexibility but minimal support. This structure is advantageous for users who want to define their own semantics on files. UNIX and MS-DOS operating systems use this structure

Record sequence

The file is treated as a sequence of internally structured fixed length records. In such a structure, reading from or writing into a file interacts with one record. No systems currently use this method though they were very popular in early times in mainframe computers.

Tree

The file consists of a tree of records that may differ in length and each having a key parameter in a fixed position in the record which is used to interact with the records. The operating system can add new records to the file deciding the place for the record. The records on the tree are sorted based on the key field which makes searching faster. Large mainframe computers use this structure.

File types

Several types of files exist that are supported by an operating system.

Regular files: are the most common types of files consisting user information. Regular files may contain ASCII characters which are texts, binary data, non-texts and not readily readable, executable program binaries, program input or output. The contents of such files is structured with no kernel level support

Directory: is a binary file consisting of list of files contained in it. These are system files used to manage file system structure which may contain any kind of files, in any combination. . and .. refer to directory itself and its parent directory. The two commands used to manage directories are mkdir, to create a directory, and rmdir, to remove a directory

Character-special files: are I/O related files that allows the device drivers to perform their own I/O buffering. These files are used for unbuffered data transfer to and from a device. The files generally have names beginning with r(for raw), such as/dev/rsd0a

Block-special files: *are files used for modeling disks and other devices that handle I/O in large chunks, known as blocks. These files expect the kernel to perform buffering for them. They generally have names without the r, such as /dev/sd0a.*

File access

Files can be accessed either sequentially or randomly. In a sequential file access, a system reads the bytes in order starting from the beginning without skipping any in between. Rewinding or backing up is possible but with limited accuracy and slow performance. Such files were available during the magnetic tape era. In a random file access, a system can traverse the file directly in any order. Bytes or records can be read out of order and access is based on key rather than position. Disks made possible this random accessibility of files. These are essential for database systems.

File attributes

Attributes of a file are extra information associated with files in addition to name and data. Some of these attributes are listed in table7.1 below

Attribute	Description
Protection	Who and how a file can be accessed
Password	The key to access a file
Creator	ID of the user who created the file
Owner	The current owner of a file
Read-only flag	0 for read/write, 1 for read only
Hidden-file flag	0 for normal, 1 for not in the display list
System file flag	0 for normal, 1 for system files
Archive flag	0 for backed up, 1 for a need to backup
ASCII/Binary flag	0 for ASCII, 1 for binary
Random access flag	0 for sequential, 1 for random
Temporary flag	0 for normal, 1 for delete on process exit
Lock flag	Non-zero for locked, 0 for unlocked
Record length	Number of bytes in a record
Key position	Offsets of the key within each record
Key length	Number of bytes in a key field
Creation time	Date and time of file creation
Time of last access	Date and time of the file's last access

Time of last change	Date and time of the file's last update
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

Table 4.2. Attributes of files and their description

File operations

Storage and retrieval of files is handled with different types of operations provided by a system. The operating system can provide system calls to create, write, read, reposition, delete, and truncate files. Some of the operations defined on files are:

Create file: Create a new file of size zero or with no data. The attributes are set by the environment in which the file is created. The task of this system call is to inform about the file and define the attributes it has

Open file: used to establish a logical connection between process and file. It fetches the attributes and list of disk addresses into main memory for rapid access during subsequent calls

Write: Transfer the memory into a logical record starting at current position in file. This would increase the size of the file if current position is the end of file, or may overwrite and cause content loss if current position is found at the center of the file

Read: Transfer the logical record starting at current position in file to memory starting at buffer known as input buffer. It is the task of the caller to indicate the amount of data to be read and where to read

Close: Disconnects file from the current process. The file will not be accessible to the process after close

Delete: removing a file when no more needed so that some disk space is utilized.

Append: used to perform restricted write operation by adding data at the end of an existing file

Seek: used to specify from where a data should be accessed in a randomly accessible file by repositioning the current-file-position pointer to a given value. In addition to these basic operations, there are also common operations defined on files such as rename, copy, set and get attributes, etc. most of these operations require searching for a named file in the directory of files causing continuous search.

To overcome this continuous searching, most systems perform a call to open operation before the file is first brought active.

Conclusion

When a process has information to be maintained, it keeps it in its address space which causes three main problems. Storage capacity of a system is restricted to the size of available virtual memory which may not be enough for applications involving large data; the virtual memory is volatile which may not be good for long term storage and information need not be dependent upon process as there might be a need to modify that data by different processes. To overcome these limitations, long term information storage, file, is essential. A file is a named collection of related information defined by its creator. It is an abstraction used by the kernel to represent and organize the system's non-volatile storage resources, including hard disks, CD-ROMs, and optical disks. File system is part of the operating system that manages files. Every file has a name attribute associated with it which is used to communicate with it. file names have a different extension they end with according to the file type. A file can represent a program or data and can support free form or rigid form. A file can be structured in byte, record or tree form and can be accessed either sequentially or randomly. The operating system uses several types of operations, through system calls, to interact with files.

Assessment

Short answer

a. What is file?

b. Discuss in detail three file attributes.

c. What is the difference between random access and sequential access?

d. Discuss the problems encountered by a process when keeping information in its address space

e. What are the extensions found in file name used for?

Activity 3 Directories

Introduction

Directory is collection of nodes that contains information about files. Directories shall be logically organized to achieve efficiency, facilitate convenient naming, and allow file grouping. This activity covers overview of file directory, organization of directories and operations that can be carried out on directories by users or applications.A general purpose computer maintains thousands and millions of files on secondary storages.

An entire space of a secondary storage device may be used to store a file or only its portion. Directory is a means provided by the file system to keep track of files. In most systems, directories, also known as folders, are files themselves owned by the operating system and have their own organization, properties and operations. They provide mapping between file names and the files themselves. Directory contains a number of entries about files such as attributes, location, ownership etc.. The directory may keep the attributes of a file within itself, like a table, or may keep them elsewhere and access them through a pointer. In opening a file, the OS puts all the attributes in main memory for subsequent usage

Files can be organized in a single-level directory structure or multi-level directory structure. In a single level directory structure, all the files in a system are kept in one directory, which may also be referred as root directory. This organization scheme is known to be simple and results in fast file search. However, it encounters a problem when the number of files to be maintained increases or when used in a multi-user systems as names of each file is required to be unique. If two or more users create files with same name, this uniqueness requirement is overridden and in such cases the last file created overwrites the previous one causing the first file being replaced with another file of same name.

In a two-level directory structure, a private user level directory is assigned for each user which elevates the naming conflicts encountered in a single-level directory structure. The basic implementation of this organization requires all users to access only their own directory. But with little modification, it can extend to allow users to also access other user's directories through some notification mechanism. In this organization, it is possible to provide same names for files of different user. The system implicitly knows where to search for a file when asked to open a file since each user is associated with a private directory. This organization also has its own drawbacks. It creates total isolation between users which is not required in systems where processes share and exchange data. It may also not be satisfactory if users have many files. Figure 4.3 and figure4.4 below shows the single-level and two-level directory organizations respectively.

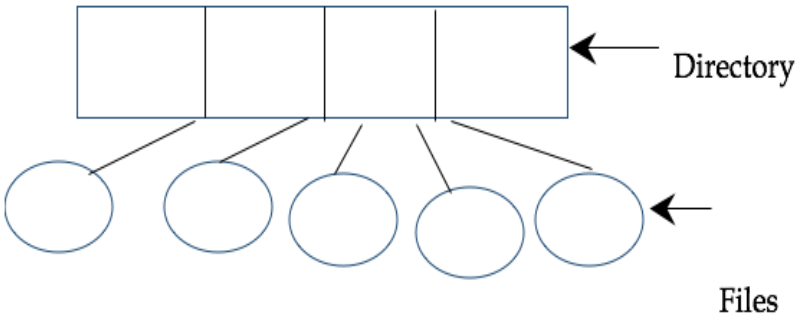


Figure 4.3. single level directory organization

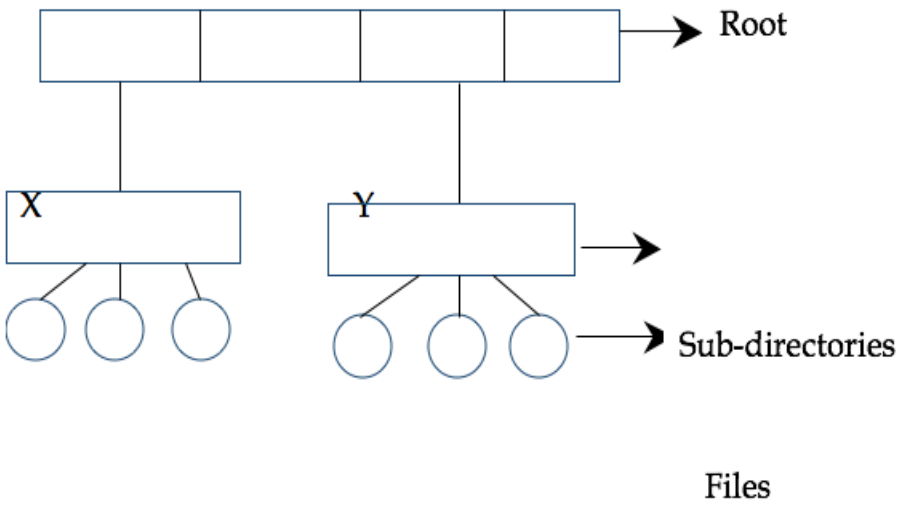


Figure 4.4. Two level directory organization

Organizing files in a two level directory system does not suffice for systems with several files and where a logical organization is required. A general hierarchical structure of several directories needs to be allowed for each user where files are organized in their natural categorical manner by extending the two level directory organizations. This hierarchical organization of directories is called tree structure where the top directory is the root sitting at the top of the tree and all directories spring out of the root allowing logical grouping of files. Every process can have its own working directory also referred as current directory to avoid affecting other processes. A directory or sub-directory has a set of files or other directories inside.

When a process references a file, it will be searched in the current directory which contains files which are currently active. Otherwise, if user needs to access a file not residing in the current directory, file name should be specified through path names or the current directory should be set to the directory holding the desired file through a system call which takes the name of the directory as parameter. Path names are conventions used to specify a file in the tree hierarchy of directories. A hierarchy starts at the directory/, known as the root directory.

A path name is then made up of a list of directories crossed to reach the desired file followed by the file name itself. Two kinds of path name specifications exist. An absolute path name is a unique name that always start from the root directory and extends to a file. The part separator in directories (/ for UNIX and \ for Windows) is the first character of any absolute path name. example /usr/books/os in UNIX and \usr\books\os in windows. A relative path name is a name which does not begin with the root directory name. This path name is specified relatively to the current directory of a process. for instance if the working directory for a process is /usr/books, then the previous file with absolute path name of /usr/books/os can be communicated as os. A relative path name is more convenient than the absolute form and achieves the same effect.

The advantage of a tree directory structure is its efficient searching and grouping capability where users are allowed to define their own subdirectories by enforcing structures on their files.

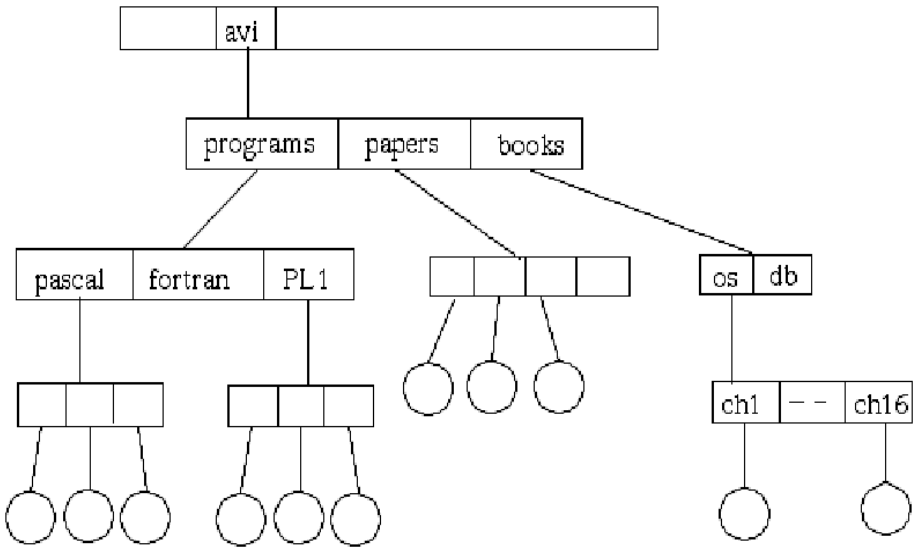


Figure 4.5. Tree structure of directories

Directory Operations

There are various system calls associated with directory management. Let's discuss some of these operations defined on directories.

Create: is an operation used to create a new and empty directory except for the two special components that are automatically included by a hierarchical directory structure supporting systems. These are the dot (.) and dot dot (..) that refers to the current directory and its parent respectively. The mkdir command is used in both UNIX and MS-DOS to create a directory.

Delete: a call used to remove an existing directory from a system. A directory with no component (except the dot and dot dot) can be deleted from a system. The rmdir command does the magic in UNIX and MS-DOS.

Open directory: used to open a directory for reading through an opendir system call.

Close directory: calls the closedir operation and is used to exit a directory that was opened for reading to free space on internal table

Read directory: done by the readdir system call to access an opened directory by returning the next entry of the directory

Rename: change the previous directory name

Link: is creation of pointers to other files or directories so that same file can appear in multiple directories enabling file sharing. The system call generates a link from an existing file to a given path taking an existing file name and a path name. A file may have any number of links but all will not affect the attributes of the file. A link can be either a hard link done by the link() system call, where links are made only to existing files in one file system not across file systems and requires all links to the file must be removed before the file itself is removed , or a symbolic link done by the symlink() system call, which points to another named file spanning across file systems. The original file can be removed without affecting the links.

Unlink: used to remove a directory entry. If the file being removed is present in one directory, it is removed from the file system. Whereas, if the file being removed is present in multiple directories, only the path name specified is removed; others remain. The rm/rmdir user commands and the unlink() system call can be used to unlink a file.

Conclusion

A directory is a means through which files are organized and managed by a file system. A single directory may contain all available files in a system or a two level directory organization can be used where by each user is assigned its own directory. A logical structuring of files in a system can be achieved through a tree like hierarchical organization of directories where users can create their own subdirectories and impose different structures to their files. A tree structure has an efficient search and grouping advantages to users. Referencing of files in such organizations is done through path names which can be specified either absolutely or relatively in reference to user's current directory. A number of operations can be carried out on directories either through system calls or user commands.

Assessment

a. What is a directory?

b. What are the limitations of a two-level directory organization?

c. What does a link operation define on directories?

d. What is a path?

e. What is the difference between absolute path and relative path of a file?

Activity 4 File System Hierarchy

Introduction

So far, our discussion has focused mainly on file systems as viewed by end users. The kernel of an operating system has also its own perception of files that will be addressed in this activity. Here the issue is about file system implementation dealing with storage of files and directories, disk space management and making everything work effectively and reliably.

Using disks to efficiently store, search and read data on disks is made possible through file systems. Two quite different design issues are faced by a file system: defining the file system interface to the user through defining a file with its attributes, the operations allowed on a file, and the directory structure for organizing files and defining algorithms and data structures to plot the logical file system onto the physical secondary-storage devices. The file system is generally structured with several levels making a layered design used to abstract the lower level details associated with it from the higher level components but each high level layer making use of the lower level layers functions to produce new features. Figure 7.4 below shows this hierarchy of a file system design.

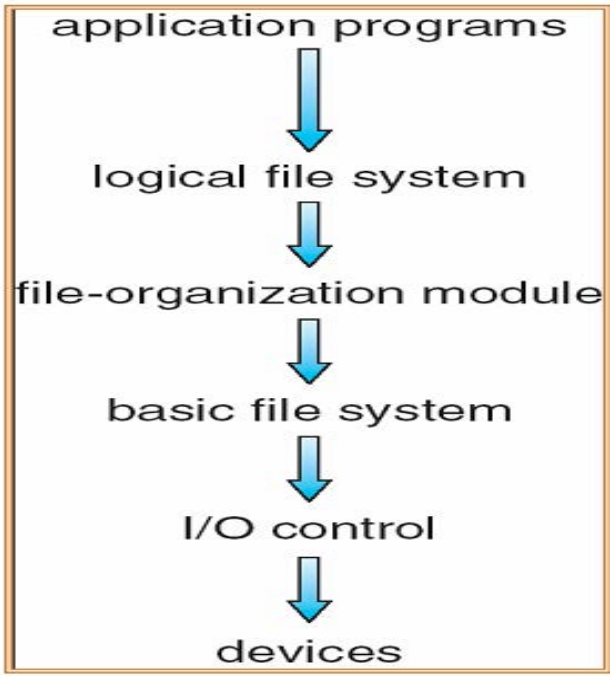


Figure 4.6. File system Hierarchy

The top level, the application programs, are codes making file requests and handle the content of the files as they have knowledge of the file's internal structure. The logical file system layer is responsible for managing the metadata of the file system, not the data in the files. It also manages directory structures and maintains file structures through a file control block that contains information about the file. The file-organization module layer translates logical block addresses to physical ones as logical addresses of files do not match to physical addresses. This layer also has free space managers that tracks and inform it about free blocks.

The basic file system layer makes generic requests to appropriate device drivers to read or write the physical blocks on the disk. Memory buffers and caches where file systems and data blocks reside are also managed by this layer.

The I/O control layer presents device drivers and interrupts handlers used to transfer information between the device and main memory. It generally interfaces with the hardware devices and handles different interrupts associated with the I/O devices.

The last layer is the device layer which presents the actual devices like disks, tapes, etc. a new file is created through a call made by application program to logical file system which, as been mentioned, knows about directory structures and defines a new file control block to the file. The system then uploads the indicated directory into memory and update its content with the new file name and file control block and finally return it back to the disk.

The file organization module is then called by the logical file system so that the directory I/O is mapped to a disk block numbers which is then used by basic file system and I/O control components concluding a file creation call. I/O can then be performed on this file once it is opened for such operations. All modern operating systems have adopted the hierarchical directory model to represent collections of files as they support more than one file system both removable-media based and disk based file systems.

In a layered file system structure, code reusability is enhanced and duplication of codes highly reduced as the I/O control and also the basic file system modules can be used by more than one file system each then defining their own logical file system and file organization module. However, layering also has its own drawback which is performance overhead on the operating system. Thus, decisions about hierarchical file system structure are a serious design issue that needs to be addressed when new systems are brought.

Conclusion

In this activity, files are discussed from the operating system's kernel perspective. The operating system has a file management system by which secondary storage devices are utilized to maintain data permanently through files. The file system has a hierarchical arrangement so that lower level technical details of how files are handled by the system is abstracted from the higher level users or applications. This hierarchical arrangement of file defines six different components with a distinct operation to handle. Though hierarchical arrangement of these components reduces code duplication by making some of the components used in several file systems, performance overhead resulted because of it should not also be neglected which requires a trade-off to be made between these two choices while designing a system.

Assessment

- Discuss what a file system is
- What are the different layers found in a file system hierarchy?
- What advantage has the hierarchical arrangement of a file system?
- What are the two design issues related with file systems?

Activity 5 File System Implementation

Introduction

In the previous activities, we tried to point out file system from the user's view as well as kernel's view. Let's now move our discussion to the structures and operations used to implement file systems. In this activity, files and directory storages, disk space management, as well as efficient and reliable file management system implementations will be examined.

File system layout

File systems are stored permanently on secondary storage devices such as disks. *These* disks can be used in their entirety or partitioned into partitions whose layout differs between file systems and each maintains different file systems. On disk, the file system contains information about how an operating system is to be booted, the total number of blocks, the number and location of free blocks, the directory structure, and individual files.

The Master Boot Record (MBR) which is found at sector 0 of a disk contains information used to boot an operating system from that disk. The end of the MBR is the partition table which gives the start and ending address of each partition. When the MBR executes, the first thing it does is locating an active partition among the available partitions by reading it to its boot block in UNIX and partition boot sector in windows , the first block, which contains a bootable operating system that is loaded when programs on this block execute. A super block, also termed as volume control block is another component of a file system which consists key parameters of the file system such as available blocks in a partition, size of each block, free block count, etc… and is loaded to memory either on system boot or on first use of the file system. A free space management attribute, nodes describing the files, root directory of the system and all the other directories found in the file system are also some other components of a file system. The general file system layout is shown in figure 4.7 below

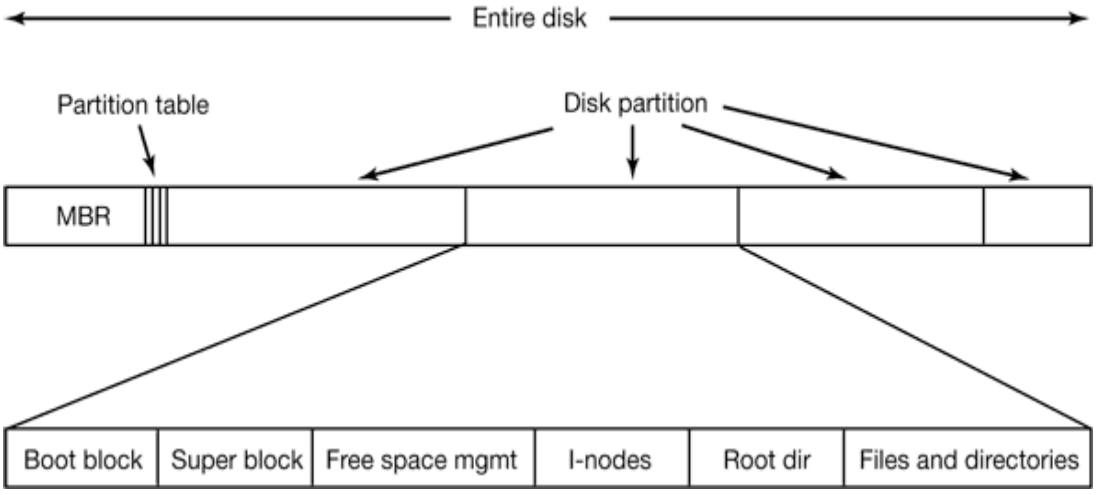


Figure 4.7. A file system layout

The most important thing to consider during file implementation is making associations between files and blocks associated with each file system. To achieve this, various allocation methods are used that differs among different operating systems. Three general methods of allocating storage on hard disks are available.

Contiguous allocation: In contiguous space allocation, each file occupies a set of contiguous blocks on the disk by laying down the entire file on contiguous sectors of the disk. Disk addresses (or sector addresses) are defined linearly on the disk. Therefore disk address 0 would map to cylinder 0, head 0, sector 0, disk address 1 would map to cylinder 0, head 0, sector 1, disk address 2 would map to cylinder 0, head 0, sector 2 and so on. Then logical block 0 is stored in disk address 0, logical block 1 is stored in disk address 1, and so on. This method has a simple implementation as keeping track of files only requires memorizing of two numbers: first block's disk address and number of blocks in a file with which any other block can be found through addition. Moreover, only a single seek operation is used to read the whole file from the disk which enhances system's read performance. Thus when accessing files that have been stored contiguously, the seek time and search time is greatly minimized. The main disadvantage of such method is the difficulty to obtain such contiguous locations especially when the file to be stored is large in size. Such files cannot be expanded unless there is empty space available immediately following it. If there is not enough room, the entire file must be recopied to a larger section of the disk every time records are added. There is also the problem of external fragmentation as disks are not compacted the moment a file is removed leaving holes in between files and causing this fragmentation problem.

Activity 1

Can you see why seek time and search time is reduced?

Linked list allocation: solves all the problems of contiguous allocation by storing each file as a linked list of disk blocks which may be stored at any particular disk address and using the first word of each block as a pointer to the next one,. The entry in the directory for a file will consist of a pointer to the disk address for the first block (or record) in the file and the last block (or record) in the file. The first block will contain a pointer to the second block, which in turn will contain a pointer to the third block and so on. There's no external fragmentation since each request is for one block. This method can only be effectively used for sequential files. Linked allocation however does not permit direct access since it is necessary to follow each block one at a time in order to locate a required block causing extremely slow access. Pointers also use up space in each block and reliability is not high because any loss of a pointer results in loses the rest of the file. These problems can be solved through placing the pointer words of each block in a table known as the File Allocation Table (FAT) in memory. Using a separate disk area to hold the links frees the entire block for data and also solves the slow access in random access though this table needs be available in memory all the time. A FAT file system is used by MS-DOS

I-nodes*:* a data structure called an index node (i-node) listing the attributes and disk addresses of the file's blocks is associated with each file in this method. The I-node is created at the time of file system creation (disk partition) and remains always in the same position on the file system. I-node table size determines the maximum number of files, including directories, special files, and links that can be stored into the files system.

Each file uses an index block on disk to contain addresses of other disk blocks used by the file. When the ith block is written, the address of a free block is placed at the ith position in the index block. Such method requires the i-node to be in memory only when the file is open which solves the limitations of the FAT system by reducing the space requirement. An I-node creates an array of size proportional to the maximum number of concurrently accessed files which differs it from the FATof linked list allocation system which has a proportional size to the disk size growing linearly as the disk grows. The i-node also has its limitation if each node is of fixed number of disk addresses and can't address when files exceed this limitation.

Conclusion

From a top level perspective, a file system is perceived as set of files, directories and operations defined to manipulate them which is quite different from its internal arrangements. File system implementers' or designers need to deal with storage allocation of files and also maintain the block file associations in order to place inline an efficient and reliable file manipulation schemes.

Assessment

Discussion

- Discuss in pair cons and pros of the three file storage allocation methods on hard disks
- Discuss about the MBR along with its contents.

Activity 6 Directory implementation

Introduction

In this activity, the focus is on how directories are implemented within a system and what the concerns are in relation with directories. In order to read a file, it needs be opened. During file opening, a user provides a path name which is an ASCII value to be mapped and used by the operating system to locate the directory entry which gives information used to track disk blocks associated with that file. The information may be the whole file's disk address, for adjacent allocation, the number of the first block, for linked list allocation or the number of the i-node. Every file system also needs to maintain file attributes. File attributes are properties of files such as its creation time, owner, etc that needs to be stored. It is possible to maintain these attributes in a directory entry which is how most systems handle the issue.

A directory is defined with fixed size entries per each file and holds the file name, structure of the file attributes and disk addresses specifying locations of disk blocks. It is also possible to store this information in the i-nodes for systems with i-node space allocation.

A file name can be of fixed length with a pre-defined number of characters constituting the name or variable length which doesn't restrict the character numbers in a file name. These variant of file names can be implemented by setting a limit on file names length, commonly 255 characters. This approach has simple implementations but wastes directory space as few files have such lengthy names.

Another possibility can be considering a variant entry size in directories containing the length of the entry itself, some file attributes and the file name. Every file name is expected to fill out an integral number of words so that the next directory entry is started on a word boundary. Figure7.6 depicts this method of handling variable length file names. During file removal process,, a variable size break is introduced in the directory which may not be enough for a newly coming file. A page fault can also occur while reading a file name as directories may cover multiple pages of memory.

Making constant size directory entries and keeping all the file names together in a heap found at the end of the directory can also be another solution towards handling the variant long file names as. This method solves the fitting into a freed space problem encountered in the previous scheme. But heap management should be performed and there is no guarantee of page fault avoidance.

In all these directory implementations, a linear list of directory entries with pointers to the data blocks is used which makes searching for files linear. This method is simple to program but time-consuming to execute. Creating a new file, for instance, requires searching the directory to be sure that no existing file has the same name and the new entry is added at the end of the directory. File deletion also requires searching the directory for the named file and then release the space allocated to it. This linear arrangement of entries results in a slow search operation especially when the directory is too long. Caching results of a search or using a hash table of each directory can be used to speed up the search. The cache is first checked for the file name before starting the search avoiding the long lookup if it is found in the cache.

Through hash table method, file name is entered by representing it with value between 0 and n-1 by performing division by n and taking the remainder of the division or by adding the words in the file name and dividing it by n where n is the size of the hash table. This hash code is inspected to determine if it is used or not. If the slot is occupied, a linked list headed at that table entry is created that threads through all entries with same hash table. But if the slot is unexploited, a pointer to the file entry that follows the hash table will be kept there. Searching for a file is done by hashing the file name and select a hash table entry and checking if the file is present in all the chain entries. If the file is not found, the file is not available in the directory. While searching is improved in hash table implementation, administration complexity is also inevitable.

File sharing

In a multiuser system, there is a frequent need to share files among users which demands the need to present common files within different directories of multiple users at a time. Two important issues needs to be addressed in association with file sharing. These are access rights and simultaneous access of files. One thing to note is the difference between shared file (or directory) and two copies of the file. With two copies of a file, the copy is accessible by each user not the original, and if one user updates the file, the changes will not appear in the other's copy. With a shared file, however, only one actual file exists, making immediately visible any changes made by one person to the other. Sharing is particularly important for subdirectories as a newly created file by one user will automatically appear in all the shared subdirectories. Several implementation possibilities are there for shared files.

A common way, exemplified by many of the UNIX systems, is to create a new directory entry called a link which is effectively a pointer to another file or subdirectory. When a shared file or directory exists, the system creates a new file of type link specifying the path name of the file it is linked to. Links are easily identified by their format in the directory entry (or by having a special type on systems that support types) and are effectively indirect pointers which are neglected by the operating system when traversing directory trees and thus referred as symbolic links. The advantage of symbolic link implementation is their use to link to files over networks through the network address specification of the machine that holds the shared file as link. But the problem with this implementation is the resulting of two or more paths of same file. Another problem associated with the symbolic link implementation is the extra overhead incurred as a result of parsing the path component wise through extra disc accesses.

Another shared file implementation approach is through simple duplication of all information about the shared files in both directories resulting in identical and non-distinguishable entries which is different from the link implementation discussed earlier. Consistency maintenance is a major problem in this implementation during file modifications as the changed content is visible only to the user making the changes which brings down the sharing concept.

Disk space management

In order to maintain files on disks, a space should be allocated and the system needs to keep track of free spaces to be allocated. An n byte file can be stored on disks using either of the two possibilities. These are allocating n successive bytes of a disk space or breaking down the file into several fixed size non-successive blocks. As contiguous allocation has its own limitation which we had seen earlier, most systems prefer the non-adjacent fixed sixe block partitioning of files. The question, however, is the size of the block as making it too large wastes the total disk space while too small results files to have several number of blocks. In general, a block can be allocated sector, track or cylinder sizes. Having blocks of cylinder size for every file lack resource utilization as small files don't consume all the space. However, access time which is directly dependent on the seek time and rotational delay of the r/w head, is enhanced with larger block sizes as the data rate is directly proportional to block size. If a small size block is defined, which makes the file to be sliced up into several blocks, a space utilization is increased in a block but also causes a longer access time. Overall, blocks of larger size tend to have less space utilization but better performance while blocks of small size have better space utilization but poor performance. Keeping track of free blocks is another concern in disk space management after the disk size is being defined. Since disk space is limited, we need to reuse the space from deleted files for new files, if possible. A free space list is maintained by the system to keep track of free disk space which records all free disk blocks which are not allocated to some file or directory. During file creation, the free space list is looked up for the needed amount of space and allocate this block for the new file which will then remove it from the list and when a file is removed, its disk space is added to the free space list. This free block space management is carried out by bitmap and linked list of blocks.

Bitmap: an n bit bitmap is defined for an n block disk where allocated blocks indicated with a bit value 0 and free blocks with bit value 1 in the bitmap. A bitmap requires less space as the blocks are represented by bits. This method is relatively simple and is efficient in finding free blocks though different bit-manipulation instructions that can be used effectively for that purpose. One technique for finding the first free block on a system that uses a bit-vector to allocate disk space is to sequentially check each word in the bit map to see whether that value is not 0. The first non-0 word is scanned for the first 1 bit, which is the location of the first free block.

Linked list: all free disk space blocks are linked together and a pointer to the first free block is kept in a special location on the disk and caching it in memory. During file creation, the blocks needed are captured from the blocks of pointers that are read from the disk when file creation ends.. Deleting a file adds the freed space to the block of pointers in memory which is written to disk. There are cases where this method is not efficient due to unnecessary disk I/O. traversing the list requires reading each block which requires a substantial I/O time though it is done less frequently as the first free block is always allocated to a file that needs a space. The FAT method's allocation data structure has this free-block accounting which avoids the need for a separate method of free block management.

Disk quotas

*I*n multiuser operating systems, there is a need to limit the maximum allotment of files and blocks for each user to prevent users from monopolizing the disk space and the system assures no user exceeds the limit set. Among the attributes of a file is the owner entry specifying by whom the file is owned and this will be reviewed when a file is opened to charge the owner for any increase in the file's size. Another table defining the quota of each user with a currently open file is also maintained by the system whose records are written back to the quota file once the opened files are closed. This table has a pointer to the quota file of the user which is used to check the limits of the user every time a block is added to the opened file resulting in error if the limit is exceeded.

Conclusion

In this activity, we tried to discuss issues related with directory implementation. File naming conventions, file searching efficiency, file sharing methods among users and applications, disk space management schemes, and disk quotas were explored all of which are concern of system designers for the betterment of structured file arrangement.

Assessment

- What are the file sharing schemes used by a file management system?
- What advantages and disadvantages do you observe on disk quota specification?
- What is a free block space management? Why is it necessary for the ystem to identify free blocks? What schemes are used by the system to manage these free blocks?

Activity 7 File system reliability

Introduction

If your computer is damaged by an accident, it is possible to replace the damaged component with minimal cost. However, if a file in a computer is damaged or lost, recovering from the loss would be expensive, if not impossible, which is the case most of the time. File damage can happen due to physical or logical reasons and we need to have an efficient and optimal file system that can protect the information from logical damages. In this activity, issues involved in protecting the file system and reliability concerns will be discussed.

Backups

Taking backups is an important task which mostly is overlooked by most users. Backing up is a process of saving files on external devices, usually tapes so that a system can use it to recover from a disaster when encountered. Backing up takes a long time and large storage space which requires to be done efficiently. Considerations need to be made as to whether the entire file system or part of it should be backed up. Mostly, it is desirable to take backups of directories and files that can't be found elsewhere. For instance, program files need not be backed up as it is possible to reinstall them from the manufacturer provided devices. Similarly, temporary files shouldn't also be included in the backup contents. Moreover, taking a backup of unchanged files and directories since the last backup is also a waste of space and time. An incremental dump is a means by which a backup of whole file is taken periodically and makes a daily backup of only the changed components since the last backup and even more effective is to take backups of those files changed since they were last dumped. Recovering is made complex in this incremental dump though it reduces recovery time.

When a system tries to recover using this dump file, the most current full backup is restored first after which comes all incremental backups but in reverse order. If the backup file is too much, compression algorithms can be used to compress the backup before saving it onto the tape which will then be decompressed by decompression algorithms at the recovery. If files and directories are active while taking backups, inconsistent results might be obtained. So making systems offline while backup or using algorithms to take rapid snapshots of file systems can be used to avoid such situations. Two dumping mechanisms are used to backup files. A physical dump writes all the disk blocks starting at block 0 of the disk till the end. It will not leave out empty disk blocks and bad blocks which may result in an infinite disk read errors during the backup process. This backup is known for its simplicity and great speed while making unselected backup, incrementally dumping and individual file restores upon request are its disadvantages.

A logical dump is the most commonly used backup taking system used which starts writing from selected files and directories and recursively backs up all files that had been changed since the last backup or system installation to a tape making recovery of a selected file or directory simpler. During recovery, an empty file system is created onto which the most recent full dump will be stored and the system makes use of this file to restore the directories and files in it. Then, if there exists a dump performed incrementally after the full one, the system restores them by performing the same task as with the full dump recovery.

A logical dump has some critical issues that we need to be aware of. One, it doesn't save the free disk block list and the system needs to reconstruct this block after restore is performed. Second, for a linked file, the system should make sure this fie is restored only once. Thirdly, files with holes inside, the hole should neither be dumped nor restored so system should carefully inspect such types of files before restoring them from the dump taken.

File system consistency

A file system is always dealing with blocks through Reading, modifying and writing blocks. Inconsistency may arise if a file system updates a block but the system is interrupted before the changes are written. This inconsistency becomes even worse when the unwritten block is free list, i-node or directory blocks. To overcome this problem, most systems implement a utility program to check for file system consistency that is run after each system boot like the scandisk for windows and fsck for UNIX. The consistency check can be performed on blocks or files. Two tables are constructed by the utility program during block consistency checking each table with counters for each block that is initialized with 0. The first table's counters count the number of times each block exist in that file while the second table's counters keep track of the number of times each block is found in the free list. List of all block numbers used in a file is constructed by the program from the i-node. The counter at the first table is then incremented for every block read. The program also checks the bitmap to see blocks not in use and increments the counter of the second table for every occurrence of a block in the bitmap. The consistency of the blocks will then be checked by checking the counters from the two tables.

If each block has a 1 value either in the first or second table, the program returns fine indicating block consistency while a value 0 in both tables corresponding to a block number indicates file inconsistency.

The utility program then fixes the inconsistency and informs the user. File consistency checking is also done in same fashion in a directory system. The inspection starts from the root directory and recursively descends through the tree. The file's usage counter is incremented for every file in each directory which will be checked against a list sorted by i-node numbers that indicates the number of directories where each file is found in. When a file is created, the counter starts at 1 and increments every time a file is linked. If the counters value match, the file is determined to be consistent. The inconsistency happens if the values of the counters do not match in which case the program should take measures to correct the values.

File system performance

The time taken to access a disk is way longer than it takes to access main memory. But this disk access time can also be improved in several ways. As a result several file systems come with optimization techniques that would enhance the performance of disk access.

Caching is one method through which disk access time is optimized. Cache is a collection of storage blocks which are kept in memory but are logical part of a disk in order to enhance performance. Some systems maintain a separate section of main memory for a cache where blocks are kept under the assumption that they will be used again shortly. Other systems cache file data using a page cache. The page cache uses virtual memory techniques to cache file data as pages rather than as file system oriented blocks which is more efficient than caching through physical disk blocks, as accesses interface with virtual memory rather than the file system. All read requests first check the cache for the presence of the block in the cache. If the block is found, the demand will be replied without disk communication. On the other hand, if the block can't be found in the cache, the block will be fetched from the disk on to the cache first and then the request is answered. Due to the large number of blocks found in cache, the system needs to respond quickly for a block's request. One way to do this can be through hashing the device and disk addresses and then look for the disk in the hash table. If a block is not found in the cache and needs to be brought and if the cache is already full, some blocks must be removed and re-written back to the disk and make some space for the new comer. This situation is similar with paging and thus the algorithms discussed for page replacement can also be used for block replacement as well.

Block read ahead is another technique used by file systems to optimize performance. This method brings blocks into the cache ahead before they are requested so that a hit rate is increased. With this method, a requested block and several subsequent blocks which are likely to be requested after current block is processed are read and cached. Retrieving these data from the disk in one transfer and caching them saves a considerable amount of time. This works well for sequentially accessed files where *the* file system checks if block a+1 exists in the cache while reading block and makes a schedule to read block a+1 to the cache.

However, if the file is randomly accessible, block read ahead has a disadvantage of tying up the disk bandwidth with writing unwanted blocks into the cache and even dropping possibly needed blocks from the cache.

Arranging blocks which more likely are accessed in sequence together also minimizes the amount of disk arm motion which in turn enhances disk access performance.

Conclusion

A file has different organization when seen from the system's perspective. File system designers are thus required to focus on internal structures as storage allocation, disk space management, sharing of files, reliability and performance. Three methods are used to allocate space to file systems among which the i-node is known to be the optimal way. Directory-management routines must consider efficiency, performance, and reliability. A hash table is a commonly used method, as it is fast and efficient. Unfortunately, damage to the table or a system crash can result in inconsistency between the directory information and the disk's contents. A consistency checker can be used to repair the damage. Operating-system backup tools allow disk data to be copied to tape, enabling the user to recover from data or even disk loss due to hardware failure, operating system bug, or user error.

Assessment

- What are the mechanisms used by a file system to check for consistency?
- What is backup? Discuss the methods of backup?
- What are the main issues need to be addressed in disk space management in relation to file systems?
- How can a disk access time be enhanced?
- What are the three space allocation methods used in file systems?
- What is the difference between file and block?

Unit Summary

A file is a named collection of related information that are treated as a single entity defined by its creator and kept on secondary storage devices. File names can be of fixed length or variable length and are used to interact with it. A file can be structured in bytes, records or trees. Several user programs or system calls can be carried out to interact with files. Each device in a file system keeps a volume table of contents or a device directory listing the location of the files on the device. In addition, it is useful to create directories to allow files to be organized.

Three directory organization methods are available: a single-level, two-level and tree directory organization. A tree-structured directory allows a user to create subdirectories to organize files.

The file system resides permanently on secondary storage devices, mostly on disks which is designed to hold a large amount of data permanently. Physical disks may be segmented into partitions to control media use and to allow multiple, possibly varying, file systems on a single spindle. These file systems are mounted onto a layered logical file system architecture to make use of them. The lower levels deal with the physical properties of storage devices while the upper levels deal with symbolic file names and logical properties of files. The Intermediate levels map the logical file concepts into physical device properties. Disk space is allocated to files through three different mechanisms. Directory-management routines must consider efficiency, performance, and reliability of file systems.

Unit assessment

Instruction. Listed below are 8 questions taken out from this unit. Attempt all questions and submit your answer in written format.

- What are the two file access methods?
- Do we need a rewind operation in a random access files?
- What are the problems with contiguous file allocation?
- What advantages does the FAT linked list file allocation have?
- What are the open() and close() operations used for?
- What are the merits and demerits of the two file sharing methods?
- What is caching?
- Discuss about bitmap and linked list free space management implementations.

Grading Scheme

- This unit's assessments worth a total of 30% based on
- The assessments provided in each activity are to be marked by the tutor among which three will be taken for grading rewarding a total of 18 Points
- Each question in the unit assessment worth 1.5 points.

Feedback

Rubrics for the unit Assessment

Assessment Criteria	Doesn't attempt all questions or answers are all wrong	Provide partial answers or partially correct answers	Attempt all questions with full and correct answer
Grading Scales	0	Got half mark for each question	Full point for each question

Unit Readings and Other Resources

- Tanenbaum, Modern Operating System, second edition, 2001, Chapter 6
- William Stalling, Operating Systems: Internals and Design Principles, seventh edition, 2011, Chapter 12.

Unit 5: Security and Protection

Unit Introduction

Operating system consists of a collection of objects, hardware or software each object with unique name that can be accessed through a well-defined set of operations. The system should allow different users to access different resources in a shared way and needs to ensure that each object is accessed correctly and only by those processes that are allowed to do so. Security ensures the authentication of system users to protect the integrity of the information stored in the system as well as the physical resources of the computer system. The security system prevents unauthorized access, malicious destruction or alteration of data, and accidental introduction of inconsistency. Protection mechanisms control access to a system by limiting the types of file access permitted to users. In addition, protection must ensure that only processes that have gained proper authorization from the operating system can operate on memory segments, the CPU, and other resources. This unit covers in detail the concept of operating system security and the mechanisms used to provide system protection. We start our discussion with different security threats, intruders, and accidental data losses followed by the protection mechanisms enforced to overcome the threats.

Unit Objective

At the end of this session you should be able to

Explain what a computer security entails

Identify different threats to a system security

Discuss the goals and principles of protection in a modern computer system

Discuss the different security protection mechanisms enforced by a system

Key Terms

Computer Security: The prevention and protection of computer resources from unauthorized access, use, alteration, degradation, destruction, or any other problem.

Threat: Any person, act, or object that poses a danger to computer security

Intruder: Any person, or program that carries out malicious or unauthorized activity on a system and jeopardize its safety.

Attack: A threat that is carried out or practiced on a system

Cryptography: A means used to hide communicated message content

Authentication: is a process of validating one's identity

Access Control: A method used to define users' interaction with a system

Activity 1- Overview

Introduction

"The most secure computers are those not connected to the Internet and shielded from any interference". However, in this era of information access anywhere and at anytime, the phrase violates the goal of communication and resource sharing.

- How can one then keep itself connected and accessible anytime without jeopardizing its safety?
- What are the concerns out there that threatens one's safety?
- What measures one shall take to overcome these threats?

This activity builds your knowledge about computer security concerns as well as counter measures set by the operating system to be taken to secure all resources of a system.

Current computer systems are designed in a way to handle several resources and bulk information. As the number of information stored grows, the need to protect this information from damage, unwelcomed change as well as unintended access is becoming more important. However, several sophisticated systems are also being used that opposes the objectives of the operating system and abuse the information causing catastrophic disasters. Security refers to the wellbeing of a system that requires adequate protection as well as consideration of the external environment within which the system operates. A secured system is thus, a system whose resources are used as intended in all situations. There are several situations which violates security and causes a system unsecured.

These problems can be external or internal to the system as well as intended or unintended problems. Computer Security aims in safeguarding a system from external threats as well as from legitimate users as possible attackers. System security has three main objectives. These are:

Confidentiality: preventing, detecting and deterring improper disclosure of information, Integrity: preventing, detecting and deterring improper modification of information and Availability: preventing, detecting and deterring improper denial of access to services. Security has many aspects among which the behavior of the threat, the attacker and inadvertent data loss are the most important ones which will be discussed in detail in this unit.

1. Threats

A threat is a potential occurrence of an event which results in unpleasant effect on the data and resources of a computer system. These are basically situations which attacks the three goals of a system security Confidentiality is basically about secrecy of information which requires an information should not be exposed to unauthorized users. Integrity focuses on protecting information from alteration and deletion by unauthorized users. Availability deals with avoiding disturbances that cause a system to be unavailable or unusable. Exposure of information, altering information, and denial of service are the threats faced by the security objectives mentioned respectively. The disclosure threat exposes an asset to someone who is illegitimate to have that. This may happen while the resource is still maintained in the computer or during communication over a communication channel. An integrity attack alters data found on a computer system or in transit over the network. Denial of service threat deprives legitimate users from getting access to computer system resources.

2. Intruders

Intruders are people or malicious programs breaching security. These are people causing security threats usually by trying to gain access to a system, or to increased privileges to which they are not entitled, often by obtaining the password for a legitimate account or programs written by highly skilled programmers for unhealthy intentions of penetrating a system and exploiting its resources illegitimately. Intruders can act in two different ways. An active intruder tries to make unintended changes to data and is more dangerous. A passive intruder listens or eavesdrops a data that is not allowed to it. Intruders use several standard methods in their attempts to breach security each requiring different protection means.

- **Casual prying by naive users:** these are people with no technical knowledge but read other users information
- **Snooping:** highly experienced users trying to break the security of a system to test their capability.
- **Making money:** determined attempts to change mostly financial data and get money as a result.
- **Commercial espionage:** spying information for a particular content without the knowledge of the owner

The malicious program considered as intruder is known as a virus. Virus is a snippet of code capable of replicating itself by attaching itself to normal programs in the system and damage the system. The damage caused by a virus is general in that it attacks all resources of a system.

3. **Accidental data loss**

There are also several situations causing unintentional important data losses in a system which results more damage than intruders. Some of the causes resulting accidental data loss are:

- **Natural phenomenon :** accidents like earthquake, fire, flood, etc
- **Hardware or software error** *: malfunctioning of CPU, disk failures, errors in programs*
- **User errors** *:* invalid input, wrong program run, disk loss, wrong disk mount, etc

These problems can be solved mostly by taking appropriate and adequate backups on devices furthest away from the original location

Conclusion

Security in a computer system focuses on safeguarding the resources (hardware, software) of a system. Several conditions threaten the proper functioning of a system through exploitations of the possibilities for attacking the system. A system's resource may be damaged by intention or non-intentionally. Intruders are people and programs that intentionally damage a system while accidental data loss is non-intended system damage as a result of different phenomenon like natural accidents, hardware/software errors etc…

Assessment

1. What is a threat?
2. What differences are there between active and passive intruder?
3. What makes intruders attack a system?
4. What are the three goals of computer security?

Activit 2- System Security Attacks

Introduction

In activity 5.1., we have talked about security threats which attempt to break system's security by standing against any of the security goals. In this activity, we will try to explore the different attacks exercised on a system security along with design principles to be considered while developing an operating system so that a system can have a resistance to these attacks.

A system can be attacked by different attacking situations that test its vulnerability. The attacks can be physical which involves stealing or physically damaging system's resources or logical which involves using malicious programs and damage system's resources. In general computer security attacks are classified as inside or outside attacks.

Inside attacks

These kind of attacks are carried out by an entity which has an access privilege. The attacker can steal or physically damage resources or after successful penetration to a system, the intruder can perform its desired operation and attack the system. Several inside logical attacks can be imposed on systems. To mention some:

Trojan horses: is an apparently innocent program but with codes inside that causes unintended operation such as modifying, deleting or even transferring the user's information onto the attacker's location. These programs trick the user by using an appealing and convincing presentation which makes the user believe and run the program.

Login spoofing: the attacker presents a fake login interface which looks exactly like the true login interface. The user then gives his/her authentication data to the fake interface which collects and sends the information to the attacker and exits. The original login interface is displayed then and the user believes of making some error previously on the information and thinks the system is displaying the interface for the second time.

Logic bomb: is a code embedded in a program that "explodes" when certain conditions are met, e.g. a certain date or the presence of certain files or users. Logic bombs also often originate with the developers of the software.

Trap doors: is a secret entry point into a program, often left by the program's developers, or sometimes delivered via a software update. These are codes included into a program to help the attacker bypass some normal checks.

Buffer overflow: is an attack that happens on a system by exploiting bugs in a program and using these bugs to damage the system. The bug can be a simple case of poor programming, in which the programmer neglected to code bound checking on an input field. The attacker after examining the nature and holes of the program, writes a program that sends more data than the program was expecting causing memory overwriting by those outbound values.

Outside attacks

These are attacks caused by an attacker from outside the system mostly on a network. These attacks make use of codes being transmitted to the targeted machine over a network and runs on it causing the damage. These codes are known as malicious codes and can be virus, worm, mobile codes or java applets.

Virus: is a program that replicates by attaching itself with another program and causing harm on the system's resources. The attacker distributes the virus by attaching it to an appealing program which is more likely to be used by users (games, free apps, etc). After penetrating the system, the virus remains inactive until the victim application is launched and when initiated, it starts affecting other programs in the system. Viruses are of different types according to their properties and activation methods.

Preventing a virus from penetrating into a system on the first place is an ideal solution to protect a system from a virus attack which is very difficult, if not impossible, to do in the era of networked systems. So the best alternative towards virus protection could be detection of a virus, once detected identifying its type, and then trying to remove it from the system or if not possible remove the entire affected program. These are carried out by special programs known as antivirus. Antivirus programs are used to protect system from being affected by viruses. An antivirus maintains definition for viruses which is the pure code of the viruses. The antivirus then inspects executable files of the system to check if there is a code matching the virus definition. If so, they try to fix the program by removing the virus code.

Worm: is similar with virus except it has self-replicating capability and does not need to attach itself on other programs to duplicate itself. It is an independent program that spreads via network connections, typically using either email, remote execution, or remote login to deliver or execute a copy of itself to or on another system, as well as causing local damage.

Conclusion

An attacker is always in an attempt to damage a system. System attacks can be either internal which happens after a user successfully logs in or external from a distant place especially in networked systems. Trojan horse, logic bomb, spoofing, buffer overflow are some of the attacks from inside the system while viruses, worms, mobile codes are attacks from outside the system.

Assessment

1. How does the buffer overflow attack a system?
2. What are the solution to protect system from worm attacks?
3. Explain the difference among a virus and worm
4. What is a Trojan horse ? How does it operate and attack a system?
5. What is the difference between inside attack and outside attack?

Activity 3- Protection mechanisms

Introduction

An operating system has the responsibility of enforcing security protection mechanisms to avoid possible threats from happening and protect the various objects of a system. Each asset of a system has a name and well defined set of operations through which access is made. In this section various countermeasures set by a computer system to protect itself against security attacks will be addressed along with the design principles that shall be followed to strengthen system's security.

Design principles of an operating system

A core set of principles to operating system security design are:

Least privilege: Every object (users and their processes) should work within a minimal set of privileges; access rights should be obtained by explicit request, and the default level of access should be "none".

Economy of mechanisms: security mechanisms should be as small and simple as possible, aiding in their verification. This implies that they should be integral to an operating system's design, and not an afterthought.

Acceptability: security mechanisms must at the same time be robust yet non-intrusive. An intrusive mechanism is likely to be counter-productive and avoided by users, if possible.

Complete: Mechanisms must be pervasive and access control checked during all operations including the tasks of backup and maintenance.

Open design: An operating system's security should not remain secret, nor be provided by stealth. Open mechanisms are subject to scrutiny, review, and continued refinement.

With this in mind, let's move our discussion to some major security guarding mechanisms implemented by a system to assure secured resource communication and access

Cryptography

Introduction

Have you ever heard the word cryptography? Or have you seen people communicating with some weird, non-understandable writings where only the communicating parties know what it means? Well, in this activity we are going to address what a cryptography is, which is one of the most popular security provision ways, its importance in a system security and discuss some cryptography schemes .Encryption is a process of transforming a message with an intention of hiding its meaning to avoid possible attacks on it. Encryption is achieved with a concept of cryptography which converts the original data, termed as plaintext, into a secreted one, termed as cyphertext that can only be understood and reverted back by legitimate users of the data. Reverting back from the cyphertext to the plaintext is known as decryption.

Most of the time encryption and decryption works well through security by obscurity where the encryption and decryption algorithms are made publicly accessible while the contents are private.

Encryption and decryption algorithms have a secret parameter known as key which needs to be known by only authorized users and with which the text is converted to its required representation (cypher for encryption and plain for decryption). An encryption function can be stated as C= E(P, Ke) where a cyphertext C is obtained by an encryption algorithm E implemented on a plaintext P with an encryption key Ke.

The decryption can also be stated as P=D(C,Kd),where a plaintext P of an encrypted data is obtained by implementing a general decryption algorithm D with the cyphertext C and decryption key Kd as parameters.

Figure 5.1 shows the pictorial representations for encryption and decryption

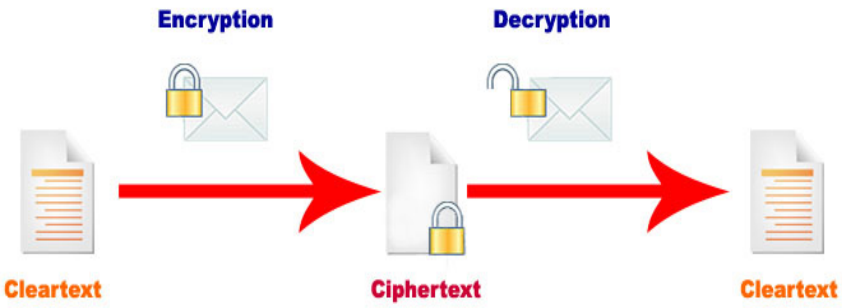


Figure 5. 1. Encryption and decryption of a message

Properties of good encryption technique are:

a. Relatively simple for authorized users to encrypt and decrypt data.

b. Encryption scheme depends not on the secrecy of the algorithm but on a parameter of the algorithm called the encryption key.

c. Extremely difficult for an intruder to determine the encryption key.

There are two methods of cryptography: Secret key cryptography and public key cryptography.

Secret key cryptography is also known as symmetric cryptography which uses same key for encryption and decryption. It also requires on establishing a secured channel to communicate the symmetric key used for both encryption and decryption among the communicating bodies. Three types of algorithms are known to use secret key cryptography: substitution where each letter in the plain text is replaced with another letter to get the cyphered text, transposition where the positions of the characters in the plaintext are transposed to some specified position, and data encryption system (DES) which is combination of the two. The main advantage of this cryptography is its efficiency in computing the cypher as well as plain text but has a disadvantage of requiring secret key communication which is very difficult especially when the communicating parties are apart.

Public key cryptography also known as asymmetric key cryptography is a method used to overcome the limitation of the symmetric key cryptography by using mathematical functions. It uses two different keys for encryption and decryption, one secret and the other public to encrypt and decrypt a message. The encryption key is mostly made public while the decryption key is private, though it depends on the purpose of the cryptography usage. The public key cryptography works by first generating a pair of related keys, private key and public key, and publicizing the public key of all communicating entities. The sender party then encrypts the message using that public key of the receiver while the receiver decrypts it with the associated secret decryption key it only has. RSA is the most widely known public key cryptography.

User Authentication

Introduction

Another security ensuring mechanism used by a computer system is user authentication.This resembles a real world situation where you are asked to bring out your student identification paper to be allowed, for instance, to get into the university compound. Unless and otherwise you come with valid and not outdated student identification paper, you will not be allowed to enter into the compound. A computer system also make use of such kind of scheme to filter between legitimate and non-legitimate users based on which resources are granted or denied. Let's define what user authentication is and the various schemes used to carry out user authentications by a system along with their merits and demerits.

Mostly an operating system allows users to use system's resource based on their identity which is based on authentication. User authentication is a processes of determining the identity of a user based on which his/her permitted activities are also identified. It allows an entity (a user or a system) to prove its identity to another entity. Typically, the entity whose identity is verified reveals knowledge of some secret S to the verifier. The computer system performs authentication when a user attempts to log into a system and is based on four different principles. These are:

- Based on what the user knows : including Password, personal information, PIN, Secret Question
- Based on what the user is : also referred as biometrics which includes Fingerprints, voiceprint, signature dynamics
- Based on what the user has : Physical key, ticket, passport, token, smart card
- Based on something about the user's context: including Location, time
- Each of these principles set their own authentication requirement with different security properties and different level of complexity. Let's see each of the principles separately.
- Authentication with what the user knows (password)

This is probably the oldest authentication mechanism which is also most widely used by computer systems. This method requires the user to type username and password on login. For each user, system stores (ID, F(password)), where F is some transformation (e.g., one-way hash) in a password file. When user enters the password, system computes F(password) to check if a match can be found to proof the identity. It is needless to say that a password should not be visible when typed but different systems use different techniques towards this. For example windows operating system displays boxes in place of each character in a password while LINUX displays nothing at all when the user types the password. Passwords have an advantage in their ease to modify when compromised but comes also with several vulnerabilities to attack. Inherent vulnerabilities are: easy to guess or snoop and no control on sharing. Practical vulnerabilities are visible if unencrypted in distributed and network environment, and susceptible for replay attacks if encrypted naively. Due to these vulnerabilities, an intruder can attempt to attack one's password. The possible attacks on password are:

Guessing attack/dictionary attack

This is a trial and error attack which exploits human nature to use easy to remember passwords. An intruder needs to know the user or have some information about the user to perform a guessing attack. Alternatively, the attacker can do Brute-force attack by trying all possible passwords using exhaustive search until it matches. Mostly, passwords of shorter length are susceptible to this brute force attack as it is possible to get the password after a limited number of guessing attempts.

But it becomes less successful on systems that allow longer passwords that include both uppercase and lowercase letters, numbers and all punctuation characters

Social Engineering

Attacker asks for password by masquerading as somebody else (not necessarily an authenticated user)

Sniffing

Anyone with access to a network on which a computer resides can seamlessly add a network monitor, allowing him/her to watch all data being transferred on the network including user IDs and passwords. Encrypting the data stream containing the password solves this problem. Even such a system could have passwords stolen, however.

Trojan login

Trojan horses are seemingly normal programs but with hidden activities of accessing system's resources by making the user believe it is safe. Trojan login is thus, tricking someone into executing a program that does nasty things. This Trojan program installed on the system captures every keystroke before sending it on to the application.

Improving password security

Different techniques can be used to enhance the security of a password by eliminating the security threats and reducing its vulnerability. Some of the techniques used are:

- Educate users to make better choices. Better passwords can be constructed by combining letters (upper and lower), digits and symbols and making them lengthy, using non-guessable phrases, etc.
- Define rules for good password selection and ask users to follow them
- Ask or force users to change their password periodically such as one time password which forces users to change their password after each login.
- Actively attempt to break user's passwords and force users to change broken ones.
- Screen password choices. Some systems use a password screening module that forbids users from using a vulnerable password.

Challenge-response authentication can be combined with password authentication where the user provides list of questions with answers at password creation and the system picks any of the questions randomly and asks the user during every login attempt to make sure the password is used by the owner. The problem with this method is, however, several question-answer pair might be required. Challenge-response is a variant of this method which lets the user pick an algorithm during registration. The system then gives an argument for the user on an attempt to login where the user returns the computed result of the argument by applying the algorithm picked during registration.

The challenge response authentication method, in general, asks the user questions no one else would know the answer to except the user.

Authentication with what the user is (biometrics)

This authentication method uses physical properties of a user known as biometrics. These physical properties include Fingerprint, Retina scan, Voice pattern, Signature, Typing style. A biometric system is constructed from enrollment and identification part. The enrollment part measures and collects the physical property and associates it with a user defined username. The identification part starts its work when the user attempts to login. It requests the user to provide his/her username and makes the physical property measurement again which will be checked against the previously collected value by the enrollment. If they match, the system allows the user to get access.

Biometrics have been hailed as a way to get rid of the problems with password and token-based authentication though, they have their own problems mostly cost and rare false readings. The physical property considered should be picked in such a way that clear distinguishing can be made between two people and also is durable characteristics that does not change much over time. For instance using a hair color as a physical characteristic would not be appropriate as more than one person can have same color. Moreover, voices, face figures and the like that varies greatly based on current situations, are also not good candidates to be measured and recorded for a physical property of a person.

Authentication with what the user has

This is an authentication method that uses a physical object possessed by a user such as ATM cards, smartcards, etc… inserted into a reader machine which is capable of reading the content associated with the object from the central database kept with the object provider. This method uses a personal identification number (PIN) or even a biometrics data associated with each object owned by a user to authenticate the user. Smart cards are the most commonly used physical objects recently. Smart cards are portable devices with a CPU, I/O ports, and some nonvolatile memory.

Authentication with the user's context

This is also referred as next generation's authentication method. It works by analyzing various user's attributes such as logon attributes based on a set of configurable parameters including geographical location, IP address, time of day, and device recognition and generates a context assurance level. These context assurance level requires a different level of authentication as administrators optimize security for any given logon instance. Context based authentication allows the system to create rules that determine, pre-authentication, whether and how a given authentication process should proceed based on context.

Access Control

System needs to make sure that assets are accessed correctly only by those processes that are allowed to do so. Associated with each user, there can be a profile that specifies permissible operations and accesses and through the user access control procedure (log on), a user can be identified to the system. The operating system can then enforce rules based on the user profile.

Access control is a process of verifying access rights to prevent misuse of resources. Three types of access control models can be used to ensure the right objects are accessed only by the right processes. These are: Discretionary Access model, Mandatory Access model and Role-Based Access model

Discretionary Access Model, DAC

Access control is based on User's identity and Access control rules. DAC has policies that manage subject to object interaction based on identifications of subjects, objects as well as allowed operations assigned for each subject on each object. When a subject shows an interest to communicate with an object, DAC grants the request by first verifying whether the subject has such a right or not on that object.. To do so, DAC uses a matrix of access rights for each subject on each object. Cells of the matrix contain an expression that represents the rights as shown in figure5.2 below

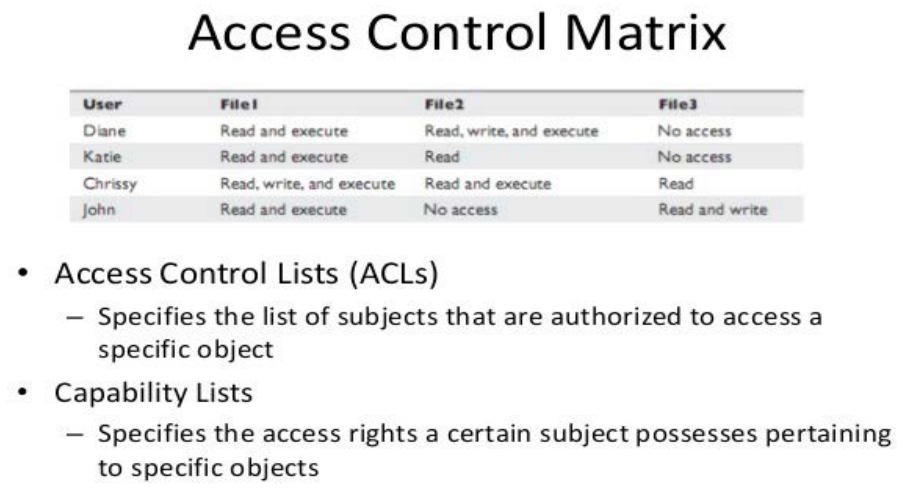


Figure 5. 2 Matrix representation for access control

This matrix can be implemented in two different ways.

Access Control List (ACL): lists users with their access rights to objects. It Stores the access matrix by columns. With each object Oj, a list of pairs <Si, A[Si, Oj]>is stored for each subject Si, such that A[Si, Oj] is not null. DAC consults ACL every time the user submits an access request. Figure 5.3 depicts an ACL organization.

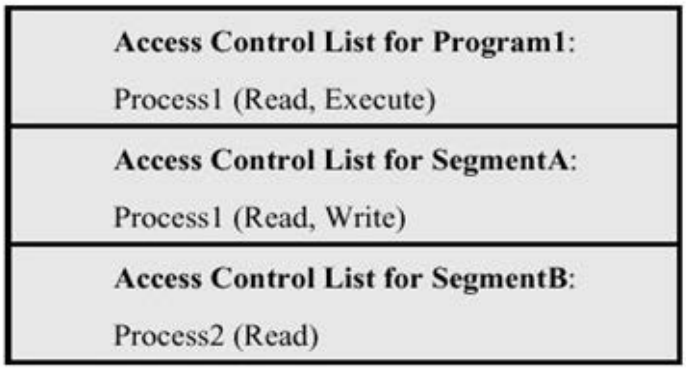


Figure 5. 3. ACL for processes

The advantage of ACL is that it is simplicity to be sustained as it only requires removing of a row when an object is deleted and query the system on an object basis is also easy However identifying all objects a subject can communicate with is difficult on top of the difficult subject based grant and revoke operations.

Capability List, CL

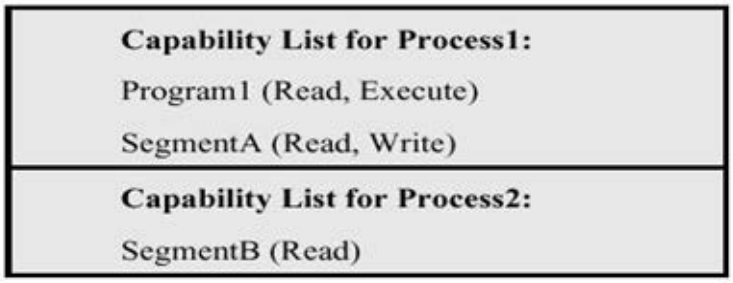


Figure 5.4 depicts CL implementation of an access control.

Each user is given a number of capability tickets specifying authorized objects and operations for a user. This implementation stores the access matrix by rows where each subject Si is the label of the access matrix rows. a list of pairs <Oj, A[Si, Oj]>is associated for each object Oj, such that A[Si, Oj] is not empty. Each row of the CL stores objects and the access rights granted to the subject on the objects. CL is also advantageous due to the easy maintenance and simple query system which is subject based. However, the computational difficulty of identifying the set of subjects that have access right on a given object exists here as well. Moreover, revoking access rights of subjects for removed objects and granting rights to a newly created objects is time taking.

Mandatory Access Control model, MAC

Regulates the flow of information by assuring information goes only to the intended recipients. When a system mechanism controls access to an object and an individual user cannot alter that access, the control is a mandatory access control (MAC). This model is important for environments requiring a much tighter security such as military system. Once access is defined, altering is not possible by users. The Bell-La-Padula model and the Bela model are access control models based on MAC.

Role-Based Access Control model, RBAC

Access rights are defined based on the role(s) a user has instead of his/her identity (DAC) or his/her clearance (MAC). Access rights are defined based on subject roles. It is used to express organizational policies with delegation of authority and separation of duties.

Conclusion

A computer system needs to protect itself from unwanted and illegitimate access. To do so, it sets various system protection mechanisms among which cryptography, user authentication and access control are the major ones.

Assessment

1. How does the public key cryptography overcome the limitations of private key cryptography?
2. How does a receiver convert a cyphertext to a plain text in a substitution private key cryptography?
3. What are the possible vulnerabilities of a password?
4. How does a challenge response authentication works?
5. What are the two implementations of MAC model?
6. How does the DAC model differ from the MAC model?
7. What advantages does the RBAC has over the other access control methods?

Unit Summary

A computer system has different resources that it needs to control to avoid misuse of these resources. As the communication scheme has changed to a networked communication where one system interacts with other several systems, possibilities of resource misuse is growing which needs more jobs to be done to secure a system. A threat is a potential situation that results in unsecured system. Three general security threats are there violating the objectives of system security. The threats are initiated by intruders which can be either active or passive and attack a system through different attacking mechanisms. Attacks happen from inside a system or outside a system and can be intentional or non-intentional. System uses different mechanisms to protect itself from such attacks including cryptography, user authentications, access control mechanisms, other software methods like use of anti-viruses, etc.

Cryptography, one security enforcement mechanism, is a science and art of communication where by the communicated information is changed to a code that can't be perceived by any other entity except the communicating ones. It encompasses both encryption and decryption schemes to achieve its objective.

Encryption is a way used to hide information communicated between a sender and a receiver with an objective of avoiding any harm on the information by illegitimate users. A message encrypted on the sender side needs to be reverted back to its original representation with a process of decryption.

Cryptography can be **symmetric** *or* **Asymmetric** depending on the code generation method it utilizes.

Authentication is another important aspect of a system used to assure security of resources. It is a means by which object's identity is validated for further service provision. The system uses several methods of authentication such as authentication by password, question-response, biometrics, physical objects, etc. The third major security provision method is an access control mechanism through which every object of a system is associated with each subject including people in terms of allowable operations. Access control is thus a way of managing permissions to subjects on objects. Three access control model are DAC, MAC, and RBAC each with their own implementation.

Unit Assessment

Check your understanding!

I. Matching

A	*B*
_____1. Cipher defined	a. Access can't be altered by users, once
_____2. Fingerprint authorized users	b. Assets can only be modified by
_____3. Substitution	c. Coded message
_____4.Logic Bomb	d. Authentication method
_____5. MAC letters	e. Letters of plain text replaced by other
_____6. Integrity and explodes when conditions are made	f. Code embedded in a program certain

II. Practical Exercises

Here are two activities you can work in the computer laboratory. Attempt both activities and submit the source code of the program you have written

Activity1. Implement one of the private key cryptography algorithms using your preferred programming language.

Activity2. Implement the RSA cryptography algorithm using your preferred programming language

Grading Scheme

- This unit's assessments worth a total of 20% from which
- Activity Assessments: 5%
- Unit Assessment I:3 %
- Unit Assessment II (Lab Project): 12%

Feedback

Rubrics for assessing activities

Assessment Criteria	Doesn't attempt both assessment or the answers are all wrong	Attempt both assessments and provide partial answers or partially correct answers	Attempt all assessments with full and correct answer
Grading Scales	Fail	Got half mark for each assessment	Score full mark for each asssessment

Unit Readings and Other Resources

Required readings and other resources:

1. Andrew S. Tanenbaum (2008). Modern Operating System. 3rd
2. Edition. Prentice-Hall. Chapter 5
3. Optional readings and other resources for Encryption, User Authentication, and types of attacks:
4. William Stallings (2005). Cryptography and Network Security
5. Principles and Practices, 4th edition. Prentice Hall
6. A Silberschatz, Peter B Galvin, G Gagne (2009). Operating
7. System Concepts. 8th Edition. Wiley.

Unit 6: Special Topics

Introduction

So far we had seen the different features of a general purpose operating system in detail. However, different operating systems are designed and used according to the requirements and environment in which they are implemented. For instance vital software with great service and performance such as medical applications, space shuttle, etc sets different requirement than the less critical applications with different purposes like PDA, phone, smart card, microwave oven, etc. thus the operating system used in each environment should be able to suit the situation and perform what is being requested with the desired property. A complex software architecture implemented as multiple concurrent communicating tasks and interrupt handlers, with priority requirements necessitates a real-time operating system. a real-time operating system is a special system which has no control over its time domain but the progress of time of the environment which dictates how time has to progress inside the system. The requirements of real-time systems differ from those of many of the systems we have described, mainly because real-time systems must produce results within certain time limits. Inside technical artifacts, for instance, many operations depend on timing, e.g. the control of turbines or combustion engines etc. A real time system by itself is a very wide concept which cannot be covered in a single unit like this. Thus, this chapter discuss a general high level description of what a real time OS is along with the requirements of a real time system and the different classifications of such systems to enable you understand the differences between such operating systems and the ones discussed so far. Another special topic to be discussed in this unit is a virtual machine. A virtual machine, VM, is an isolated replica of a real machine that behaves identical to the real physical machine but with no interference to the real machine or any other VMs. A virtual machine is an operating system OS or application environment that is installed on software which imitates dedicated hardware. For the end user the look and feel of the running an application or operating system on the virtual machine is the same as on a dedicated hardware.

Unit Objectives

1. Upon completion of this unit you should be able to:
2. Discuss installation and configuration of an OS
3. Explain and discuss real time and embedded system
4. Discuss the special requirements of a real time system
5. Differentiate between different virtual machine technologies and how to manage them
6. State how to migrate from physical to virtual machines

Key Terms

Real time system: A system that must process information and produce a response within a specified time

Embedded system: A system specifically designed for a particular operation

Virtual Machine: A software computer that, like a physical computer, runs an operating system and applications

Learning Activities

Activity 1: Basic OS Configuration and Maintenance

Introduction

How do we identify the requirements of OS? How do we conduct installation and configuration of OS?

How can we run different operating systems on a single machine? How does a virtual machine work?

In this activity, we are going to address these question and understand the basic steps followed in system installation and configuration.

Requirements Identification

As a software, operating systems do have hardware and software requirements. Before you perform installation of operating systems. You have to consider the following basic requirements:

First, identify what kind of microprocessor you have. Is it 32-bit? Or 64-bit? Modern operating systems comes with both flavors. Some are only 64-bit OSs. Pick the operating system with the appropriate microprocessor word length.

Second, make sure the main memory satisfies the minimum requirement of the operating system. In order to run several other applications after the operating system is bootstrapped, the memory required for the operating system mentioned by the vendor is the minimum.

Next, identify the hard disk space requirement of the operating system and make sure it is available on the machine. Just like the memory requirement, the operating system states the minimum requirement of the hard disk space.

Last, make sure the device drivers (software) of the peripheral devices are available. These software include device drivers for display, audio, network, etc…for your specific machine. If the operating system fails to identify a peripheral device connected to your machine, you cannot make you of that specific peripheral. To make sure it operates correctly, install the appropriate device driver for that specific machine after installing the operating system.

Usually, modern operating systems have a pack of device drivers for known hardware. But, you have to take a precaution on taking a backup of the device drivers.

Laboratory Activity

Operating System Installation

Objectives of the Labortory

Install Windos 7 and Ubuntu 14.04.1 operatinig sytems

Details of the Laboratory Exercise/Activities

Objective of the Exercise

Install Windows 7 operating system

Resources Required

Personal computer

Windows 7 on DVD or bootable USB drive

Time Requried

30 Minutes

Description of the Laboratory Exercise/Activities

Installation of OS

After going through the identification of requirements of the operating system for installation, you can jump to installing the selected OS. We picked the two operating systems, Windows 7 and Linux's Ubuntu for the demonstration.

On a specific disk drive, you can only install a single OS. Otherwise, conflict will rise. If you do have two partitions on the hard disk, you may install different operating systems on the different partitions. In doing so, make sure that you install the Windows OS first and then the UNIX OS later. If you reverse this installation process, you may end up not accessing the installed UNIX OS. Let's see how to install the two OSs.

Installing Windows 7 OS

Check you machine's compatibility for the following criteria before you start to install:

- 1 gigahertz (GHz) or faster 32-bit (x86) or 64-bit (x64) processor
- 1 gigabyte (GB) RAM (32-bit) or 2 GB RAM (64-bit)
- 16 GB available hard disk space (32-bit) or 20 GB (64-bit)
- DirectX 9 graphics device with WDDM 1.0 or higher driver

If your machine satisfies the above criteria, you can begin the installation process.

Put Windows 7 DVD in your machine's DVD drive and boot it. It will now load the setup files.

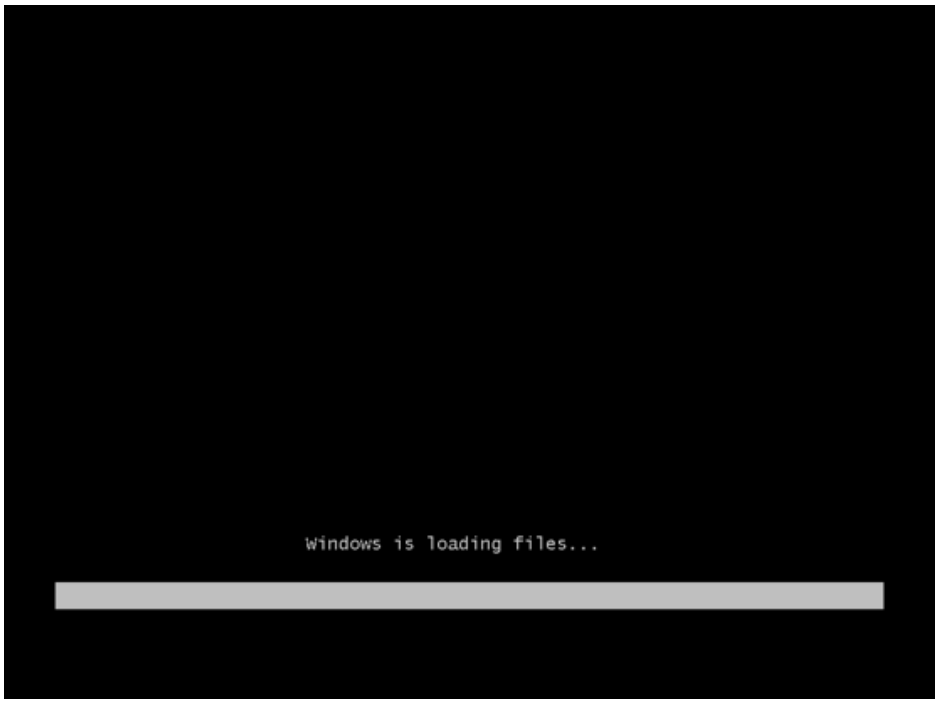


Figure 6.1. Screenshot taken while the Windows 7 OS setup file loads

Figure 6.2. Screenshot while the windows 7 OS starts

Select your language, time & currency format, keyboard or input method and click Next.

Figure 6.3. Screenshot showing preference choice entry menu before installing Windows 7

Click Install now.

Figure 6.4. Screen shot for Windows 7 installation

Check I accept the license terms and click Next.

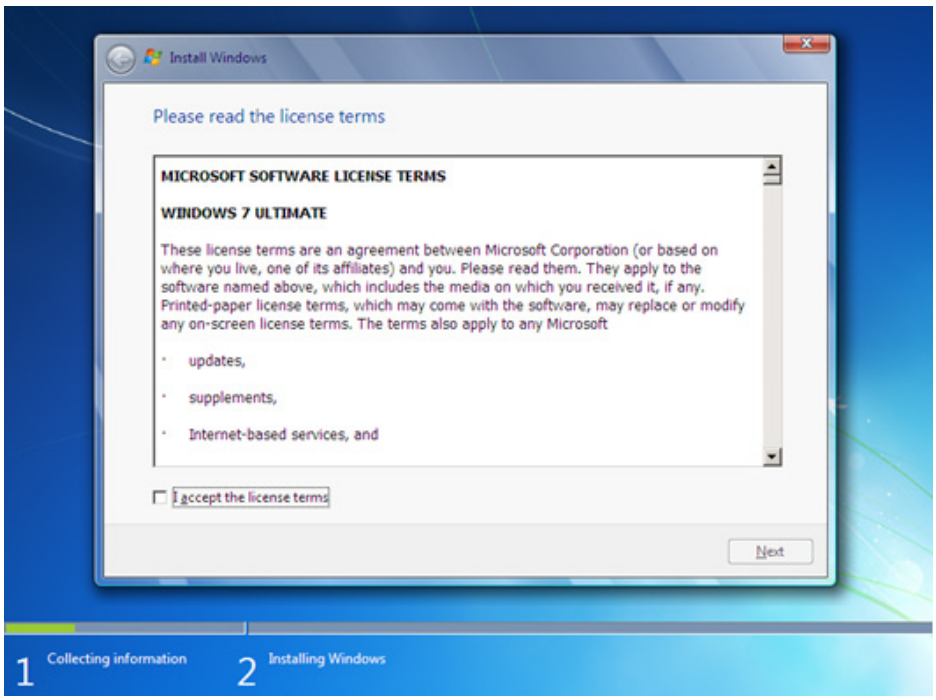


Figure 6.5. Screenshot showing the License Terms and the checkbox to click when accepted

Click Upgrade if you already have a previous Windows version or Custom (advanced) if you don't have a previous Windows version or want to install a fresh copy of Windows 7.

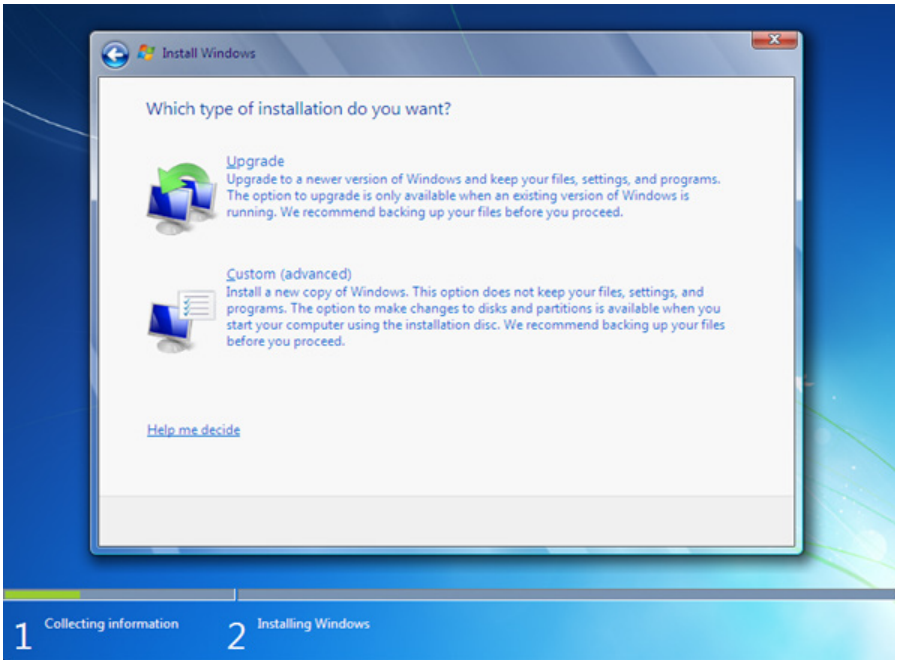


Figure 6.6. Snapshot that shows the two installation options to choose from

(Skip this step if you chose Upgrade and have only one partition) Select the drive where you want to install Windows 7 and click Next. If you want to make any partitions, click Drive options (advanced), make the partitions and then click Next.

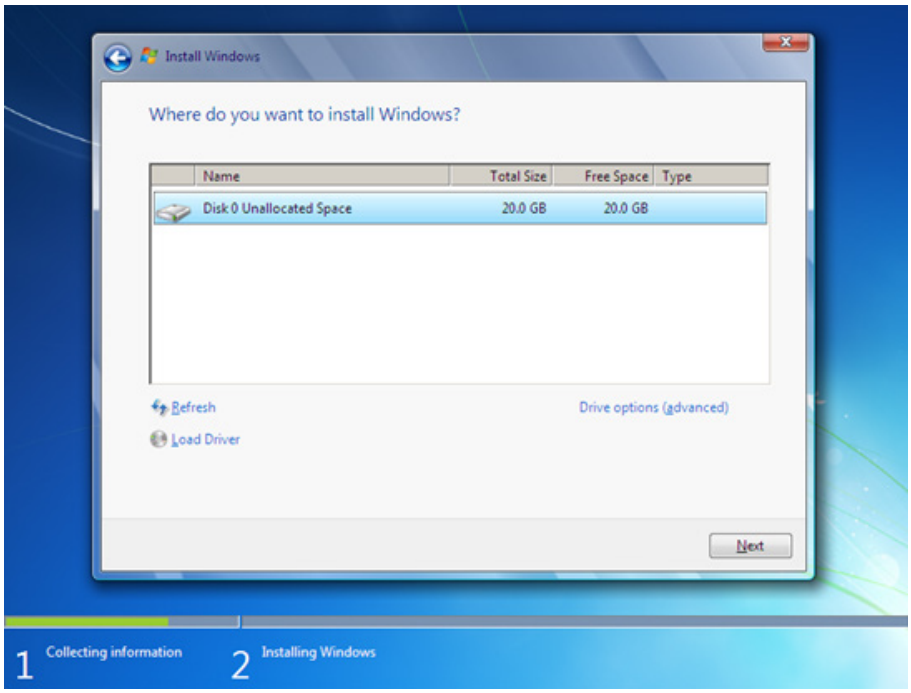


Figure 6.7. Snapshot showing windows 7 installation interface allowing to choose installation location

It will now start installing Windows 7. The first step, (i.e. Copying Windows files) was already done when you booted the Windows 7 DVD so it will complete instantly.

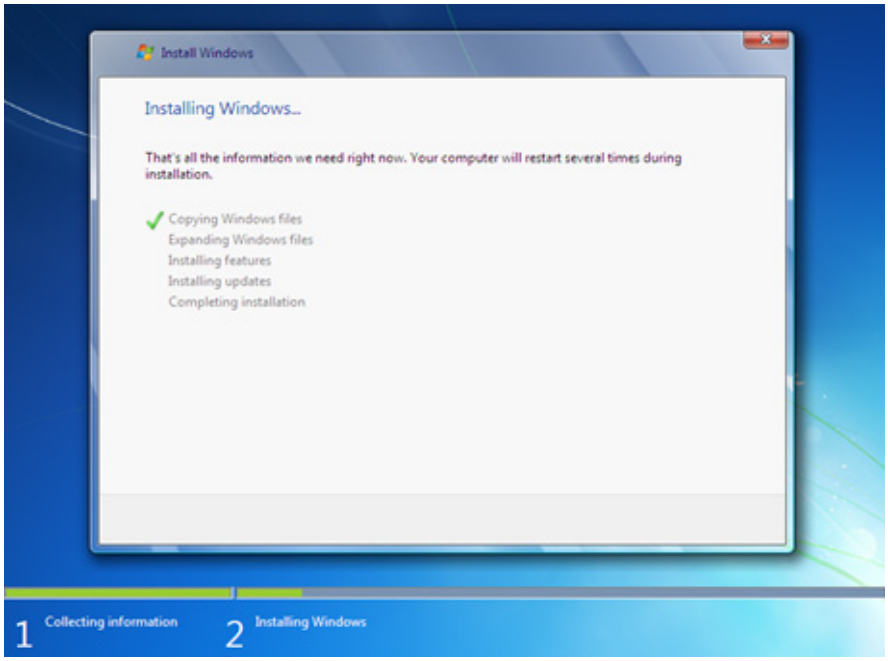


Figure 6.8. Screenshot captured while Windows 7 OS starts installing

After completing the first step, it will expand (decompress) the files that it had copied.

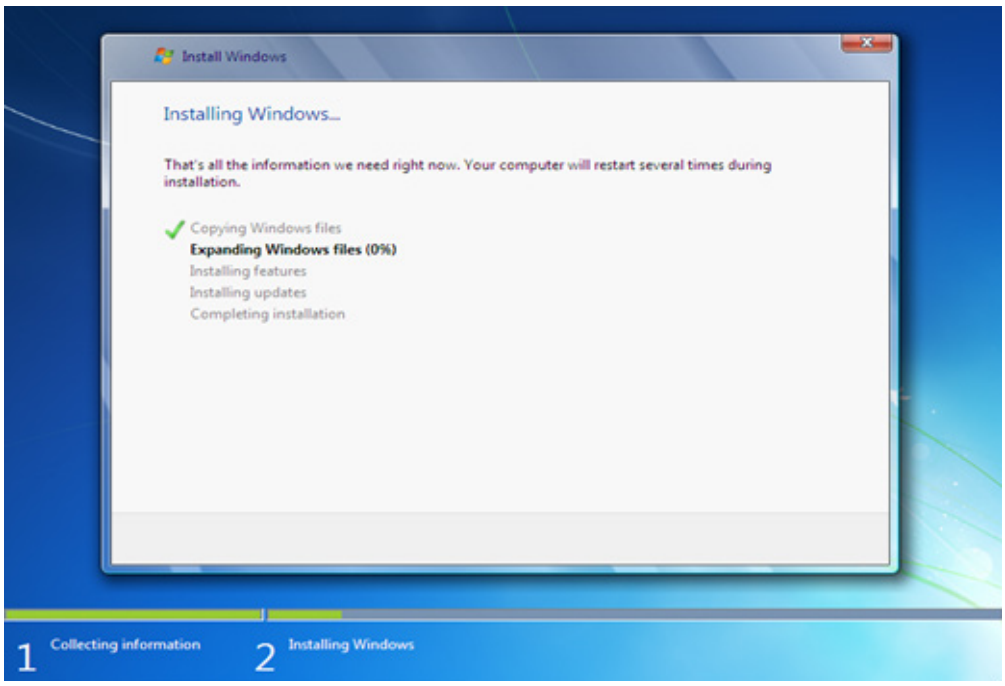


Figure 6.9. screenshot showing the file decompression step during installation

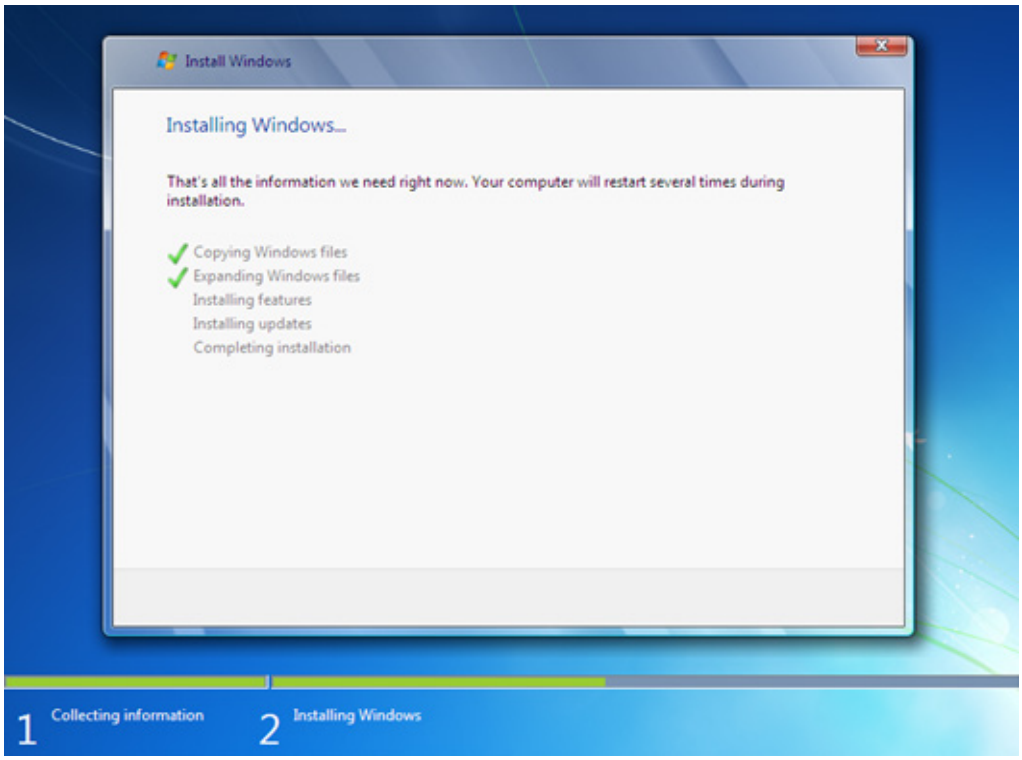


Figure 6.10. Snapshot for the completion of the first two steps during installation

The third and fourth step will also complete instantly like the first step.

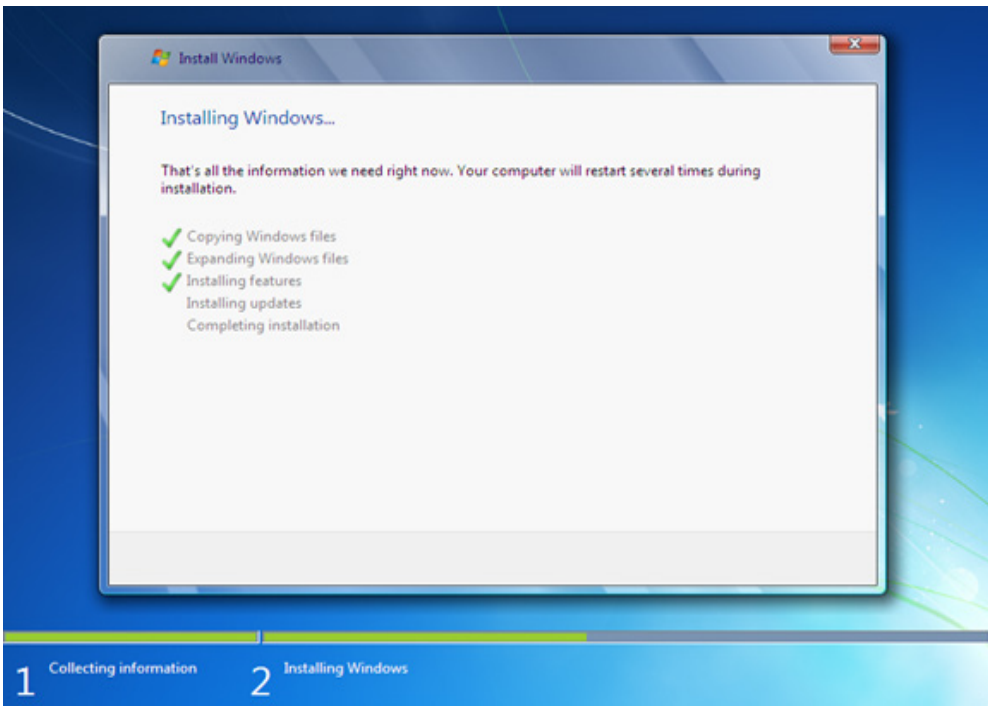


Figure 6.11. Screenshot captured after the first three installation steps are completed

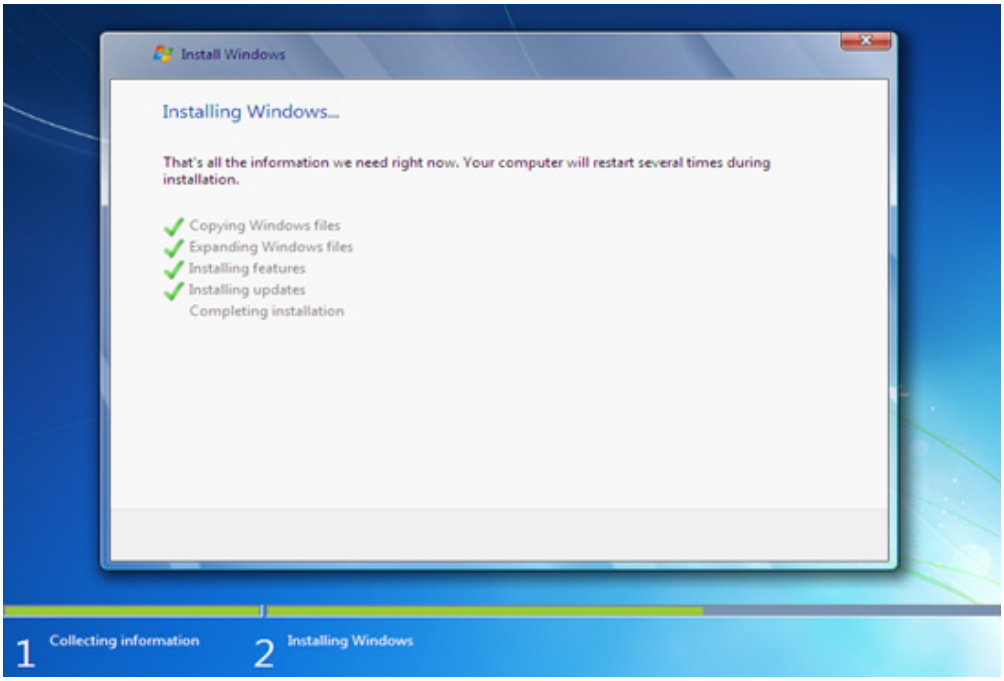


Figure 6.12. Screenshot after successful executionof the first four installation activities

After that it will automatically restart after 15 seconds and continue the setup. You can also click Restart now to restart without any delays.

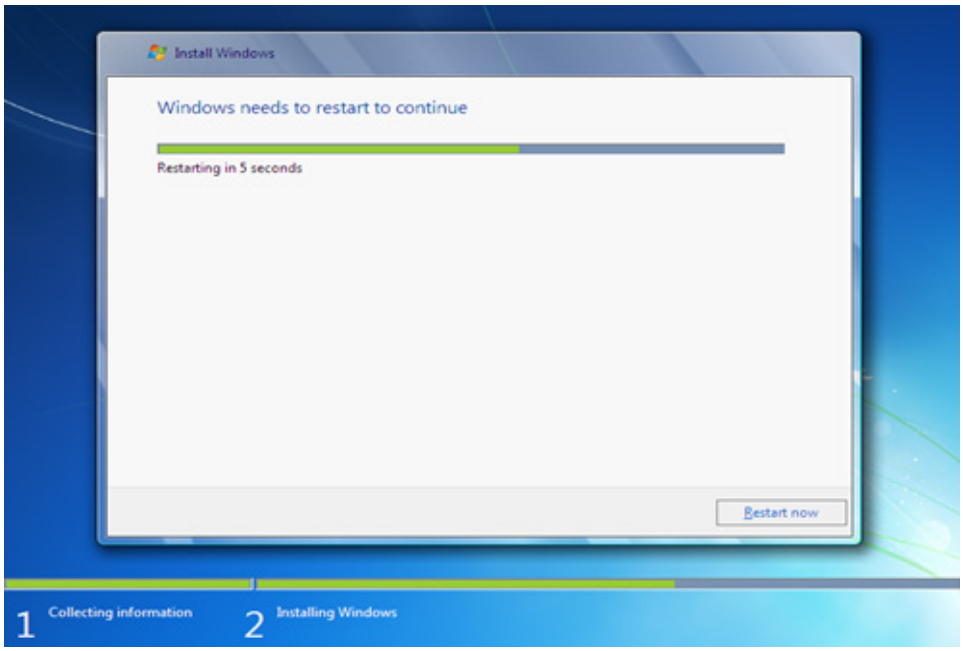


Figure 6.13. Request to restart once installation is completed

Figure 6.14. Screenshot captured while Windows 7 restarts for configuration

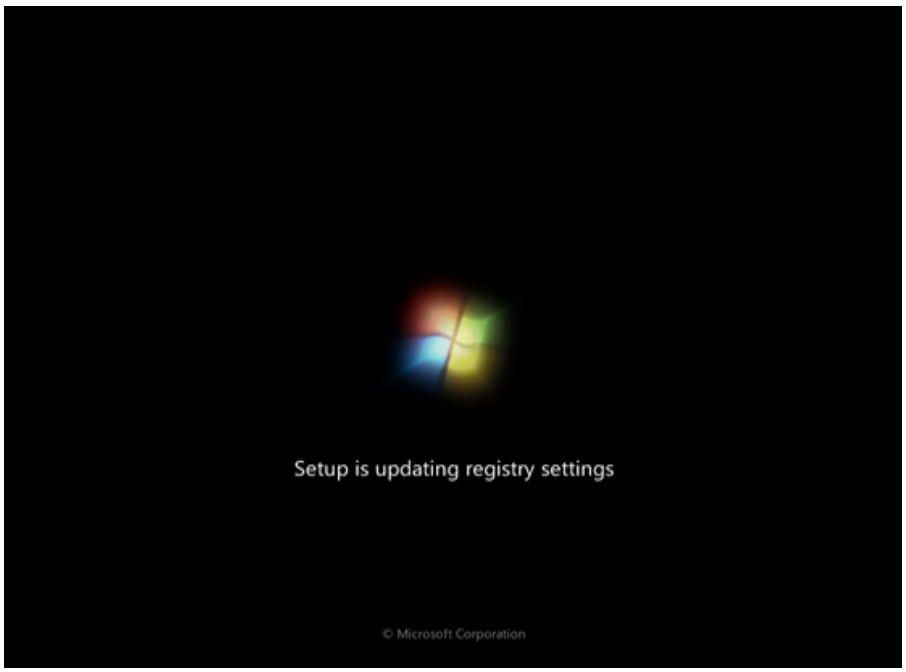


Figure 6.15. Screenshot captured during the Windows 7 OS updates registry settings

After restarting for the first time, it will continue the setup. This is the last step so it will take the most time than the previous steps.

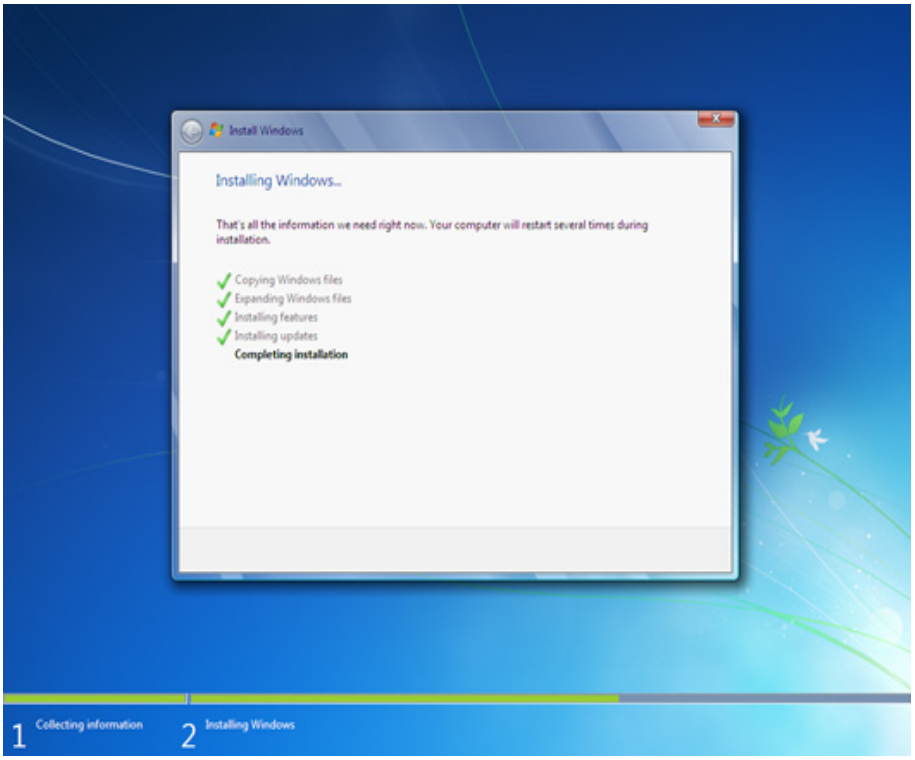


Figure 6.16. Last step completion for Windows 7 setup

It will now automatically restart again and continue the setup. You can click Restart now to restart without any delays.

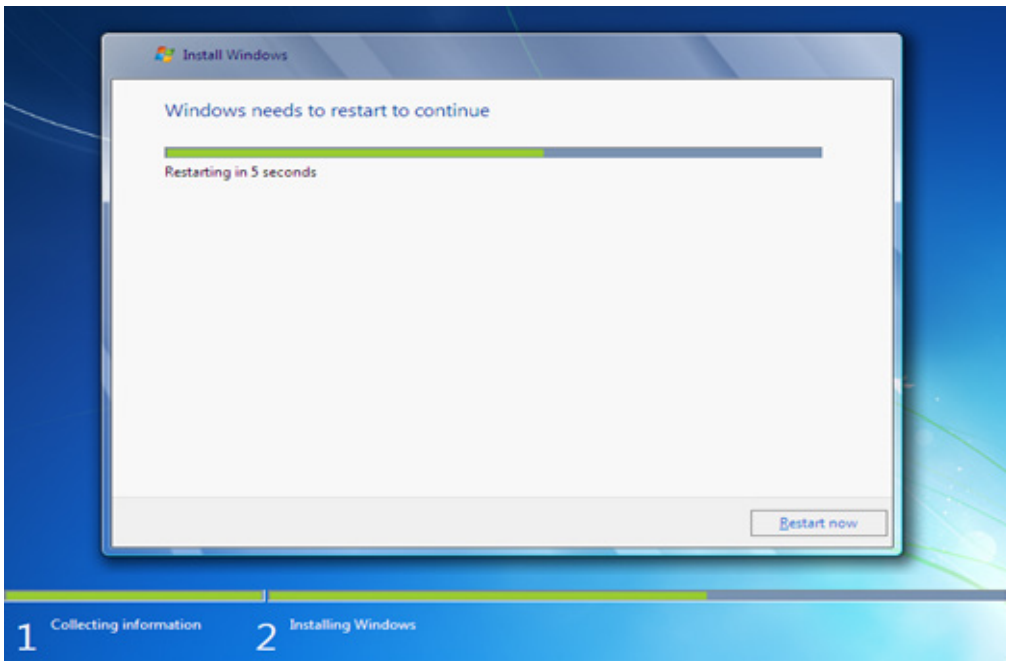


Figure 6.17. Second attempt to restart the system

Figure 6.18. Restarting the Windows 7 OS for the second time

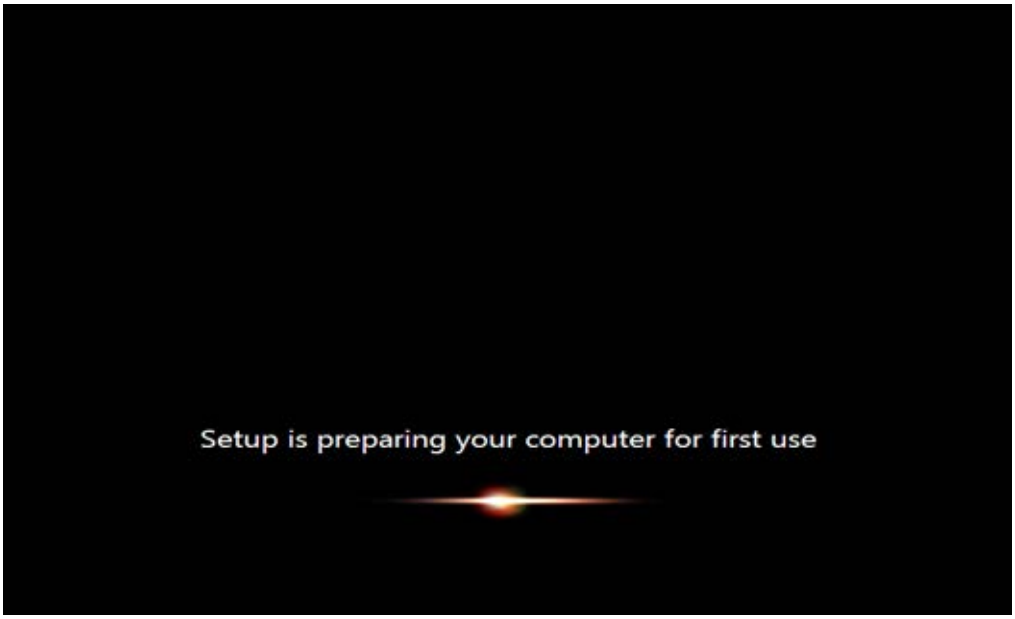


Figure 6.19. Preparing the system for first use

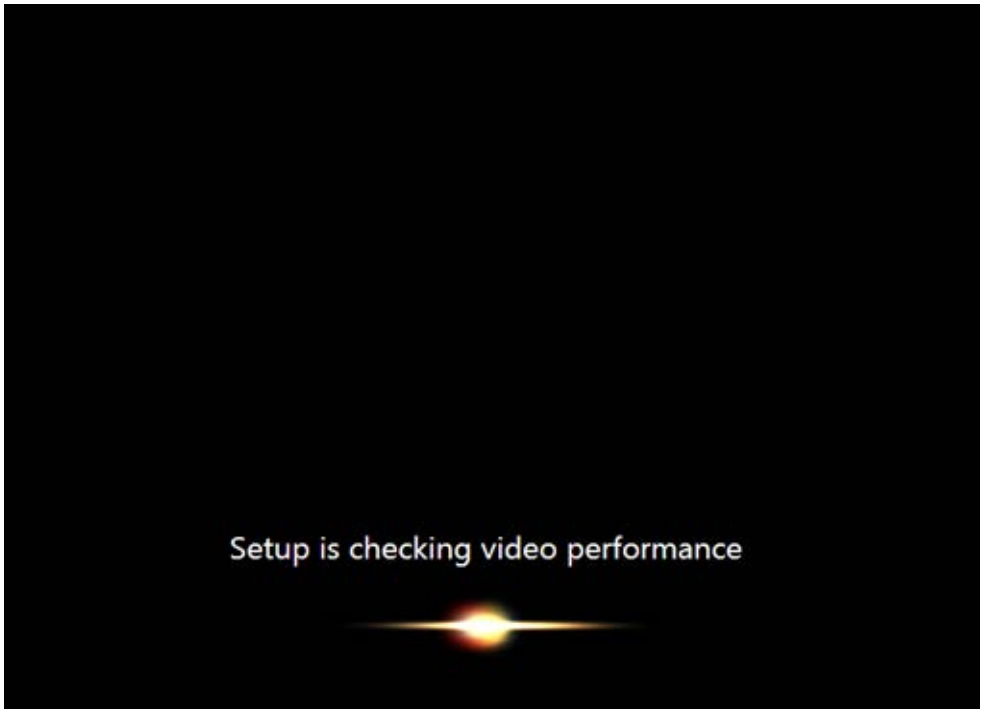


Figure 6.20 Setup Checking system's video performance

Type your desired user name in the text-box and click Next. It will automatically fill up the computer name.

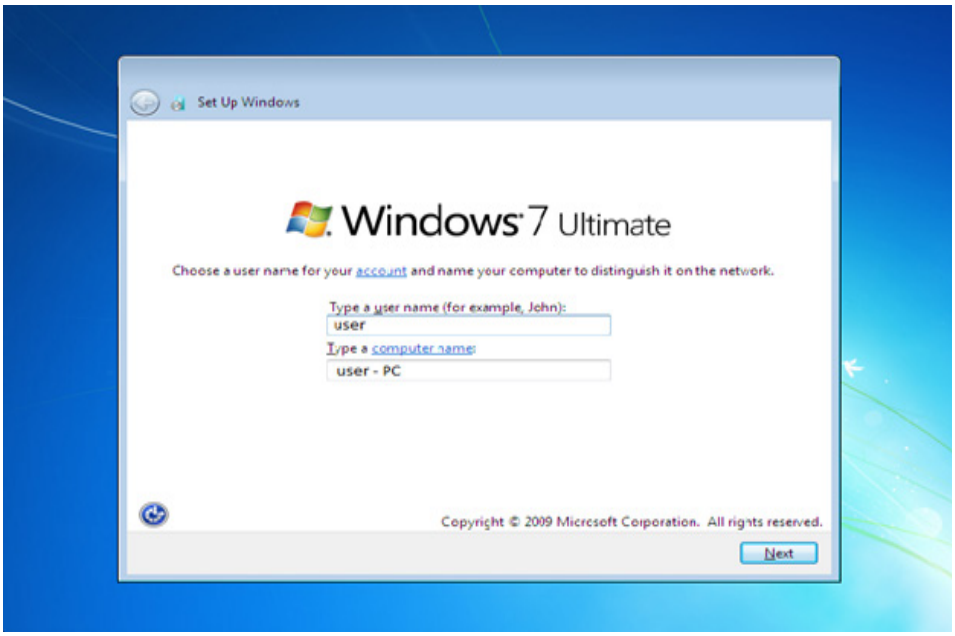


Figure 6.21. User name specification window for the user

If you want to set a password, type it in the text-boxes and click Next.

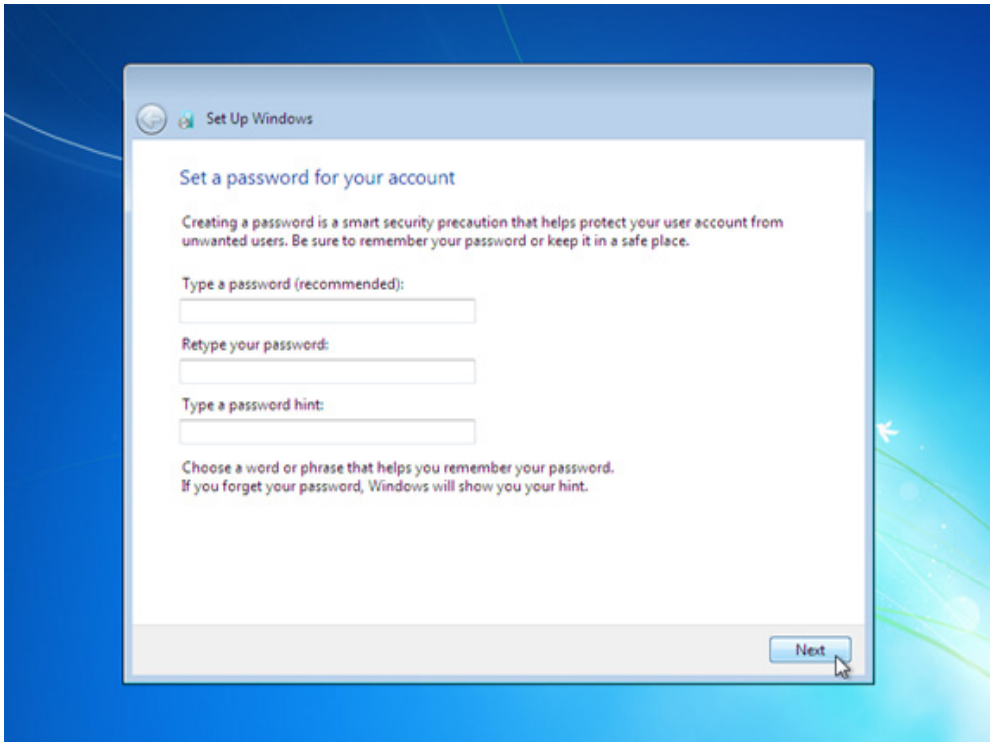


Figure 6.22. Password creation window

Type your product key in the text-box and click Next. You can also skip this step and simply click Next if you want to type the product key later. Windows will run only for 30 days if you do that.

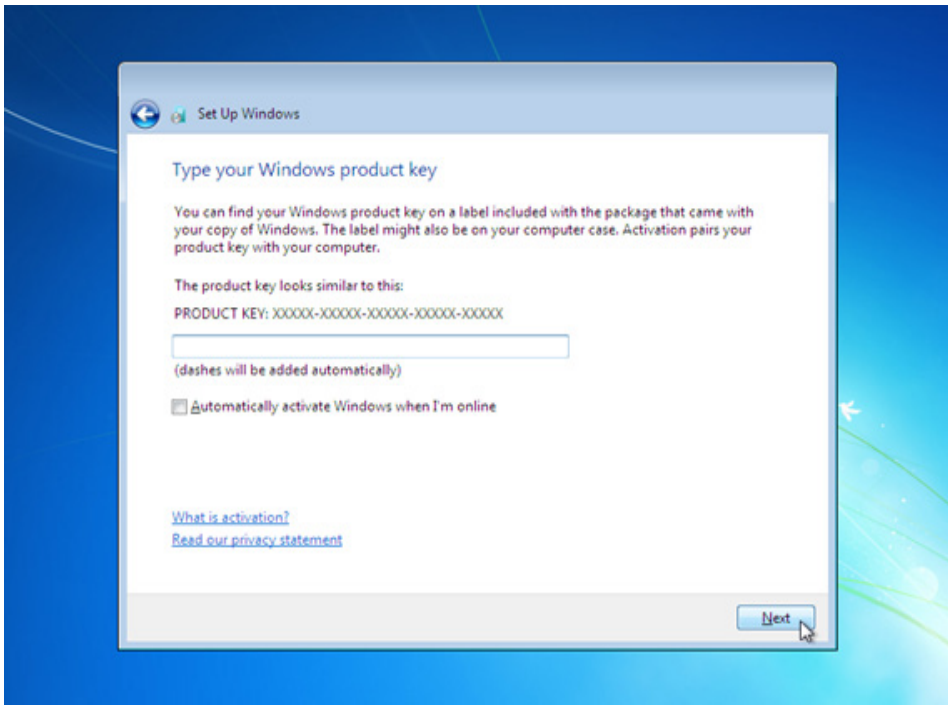


Figure 6.23. Product key typing area

Select your desired option for Windows Updates.

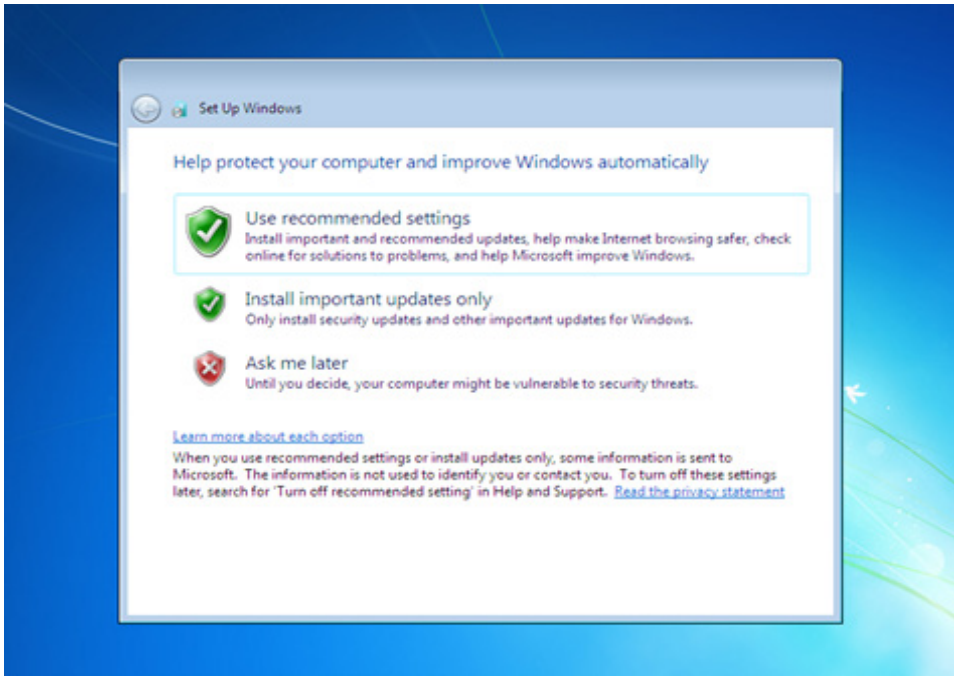


Figure 6.24. windows update preference entry interface

Select your time and click Next.

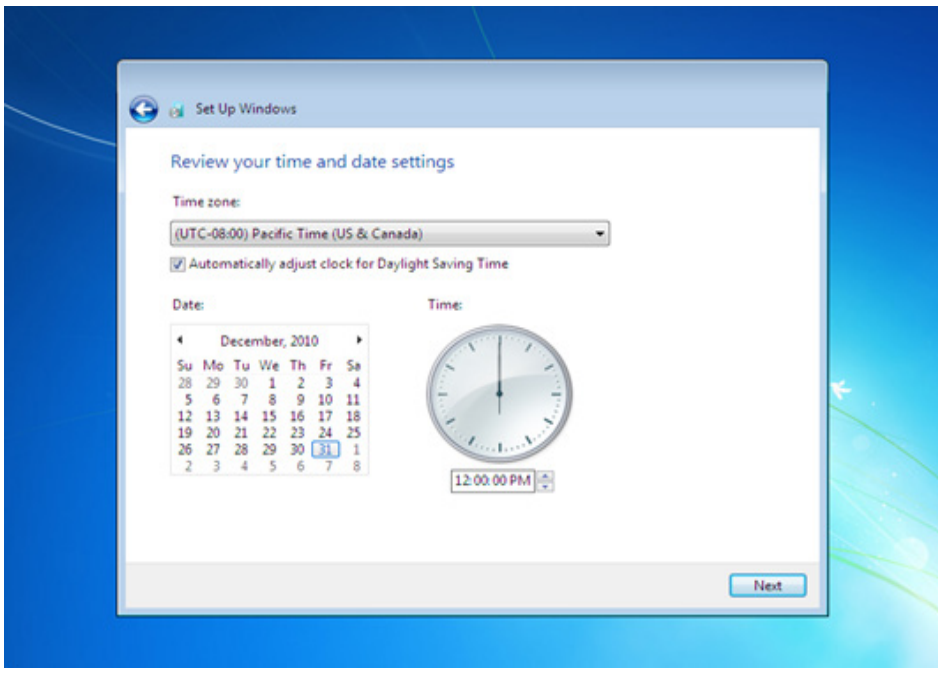


Figure 6.25. Time and Date specification area

If you are connected to any network, it will ask you to set the network's location.

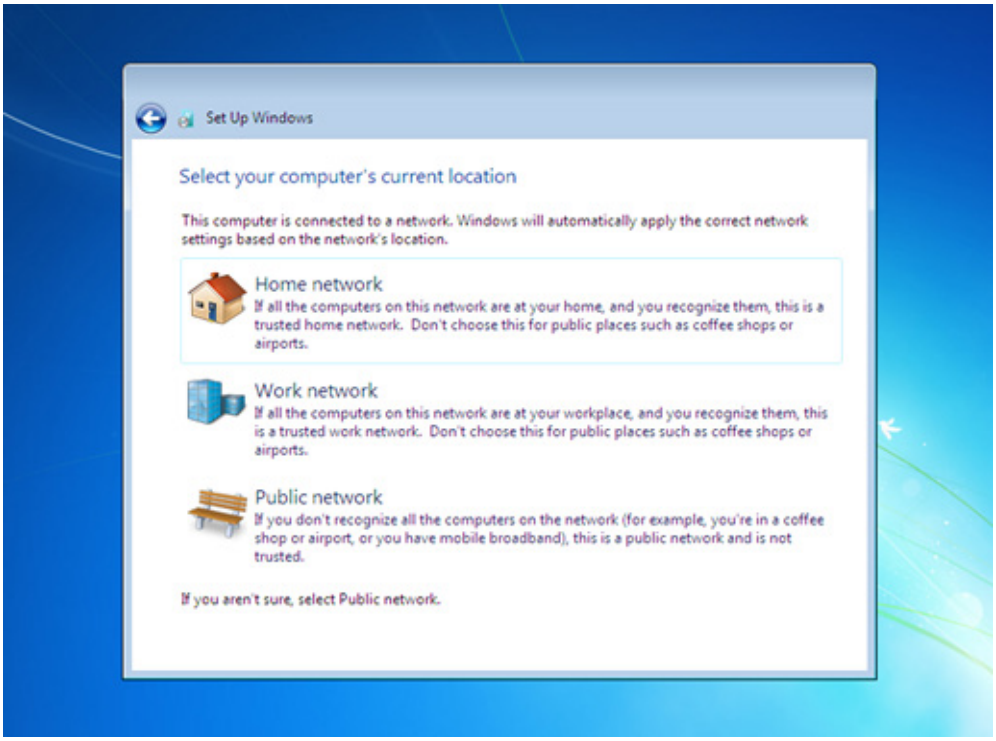


Figure 6.26. Menu to set your network's location

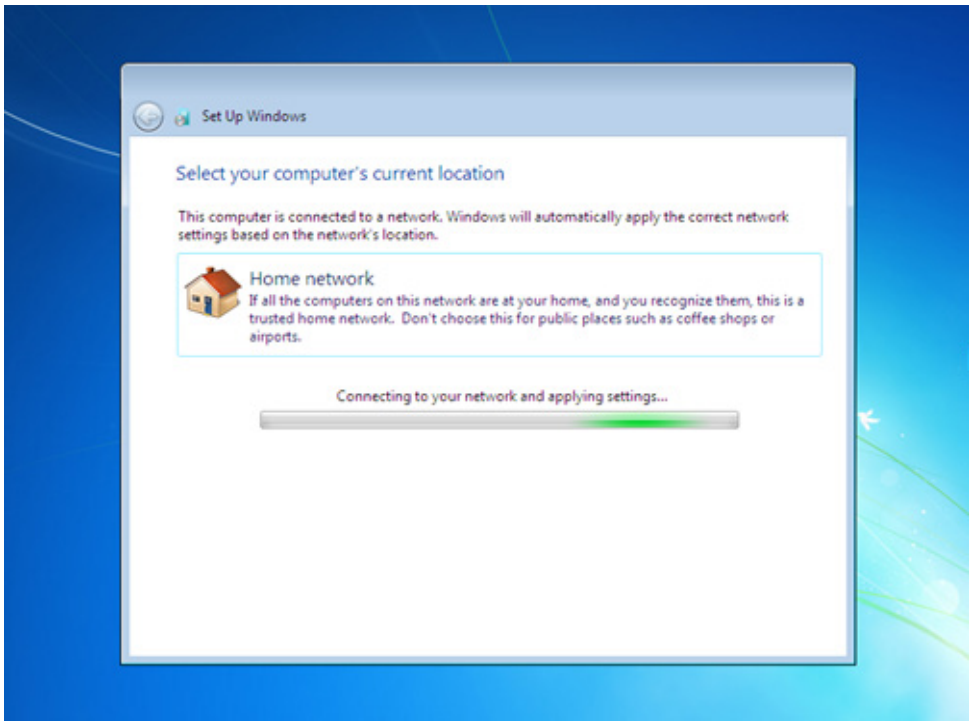


Figure 6.27. windows setup trying to connect and apply settings of the network

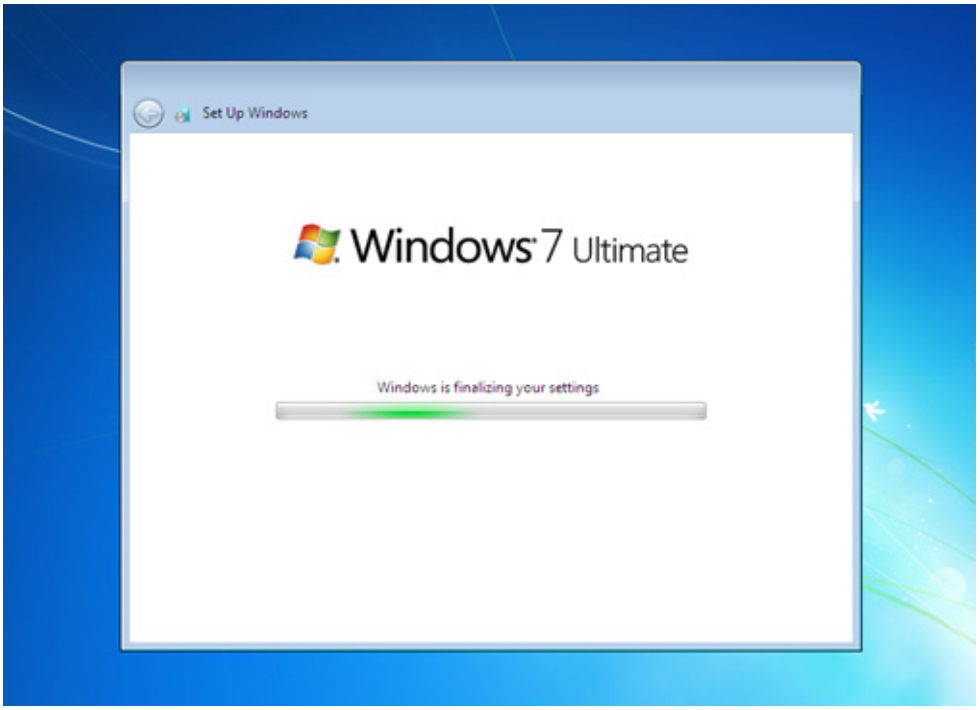


Figure 6.28. Screenshot taken while windows 7 OS setup completes the settings

Figure 6.29.first welcome message displayed by the Windows 7 OS

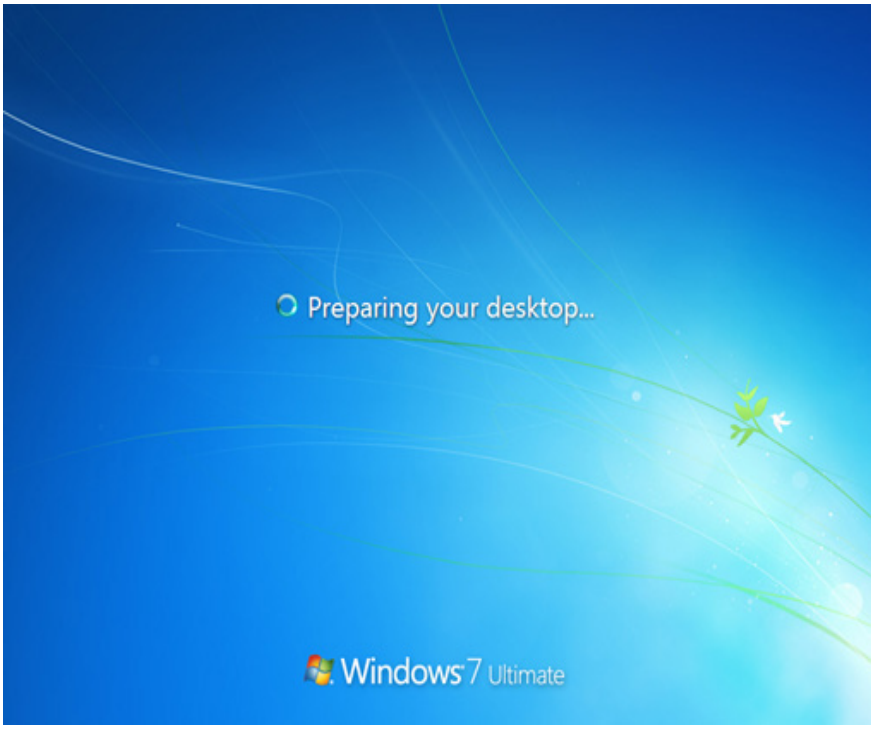


Figure 6.30. Screenshot taken while your desktop is configured for first use

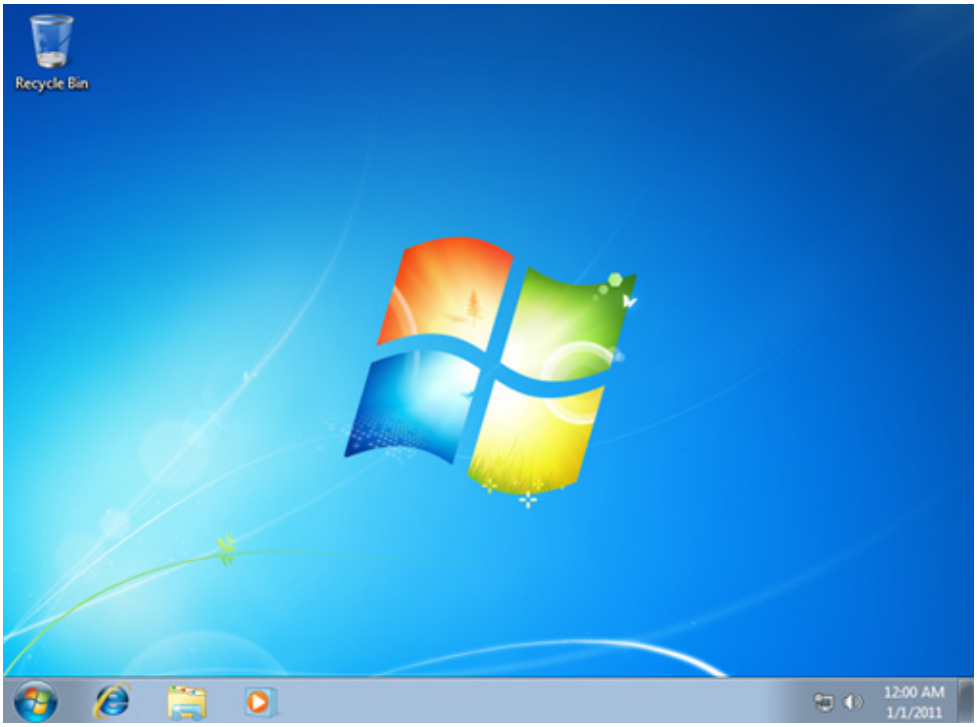


Figure 6.31. Windows 7 OS successfully started

And there you have a fresh copy of Windows 7 installed!

Results and Submission Requirments

The student has to install a fresh copy of Windows 7 operating system.

Assessment Criteria

The student has to conduct installation of the operting system obtain total of 4 marks.

References or Key Links

- http://windows.microsoft.com/en-us/windows/installing-reinstalling-windows#1TC=windows-7http://windows.microsoft.com/en-us/windows/installing-reinstalling-windows - 1TC=windows-7
- http://windows.microsoft.com/en-us/windows/installing-reinstalling-windows - 1TC=windows-7

Details of the Laboratory Exercise/Activities

Objective of the Exercise

Install Ubuntu 14.04.1 LTS operating system

Resources Required

- Personal computer

Ubuntu 14.04.1 on DVD or bootable USB drive

Time Requried

- 30 Minutes

Description of the Laboratory Exercise/Activities

Installing Ubuntu 14.04.1 LTS OS

Using a DVD

The easiest way to install Ubuntu is from a DVD. Here's what you need to do:

Put the Ubuntu DVD into the DVD-drive. Restart your computer. You should see a welcome screen prompting you to choose your language and giving you the option to install Ubuntu or try it from the DVD.

Using a USB drive?

Most modern computers can boot from USB. You should see a welcome screen prompting you to choose your language and giving you the option to install Ubuntu or try it from the CD, i.e., live CD.

If your computer doesn't automatically do so, you might need to press the F12 key to bring up the boot menu, but be careful not to hold it down - that can cause an error message.

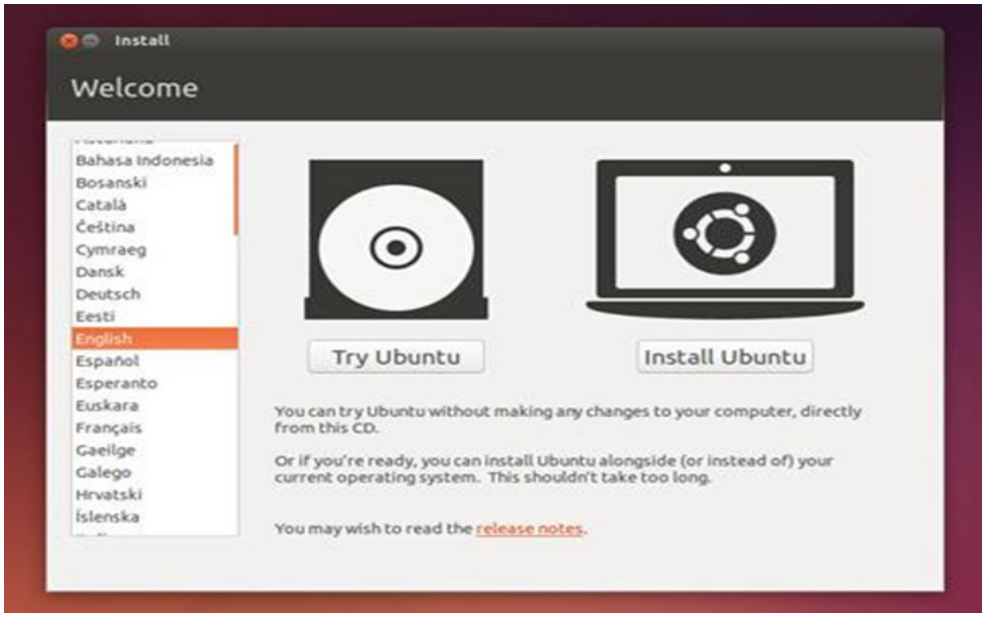


Figure 6.32. Linux OS installation welcome page

Prepare to install Ubuntu

- Plug your computer into a power source –recommended
- You should also make sure you have enough space on your computer to install Ubuntu
- We advise you to select Download updates while installing and Install this third-party software now
- You should also stay connected to the Internet so you can get the latest updates while you install Ubuntu
- If you're not connected to theInternet, we'll help you set up wireless at the next step

Figure 6.33. Screenshot captured during preparation for installation

Set up wireless

If you are not connected to the Internet, you will be asked to select a wireless network, if available. You are advised to connect during the installation so Ubuntu can ensure your machine is up to date. So, if you set up your wireless network at this point, it's worth then clicking the Back button to go back to the last screen (Preparing to install Ubuntu) and ticking the box marked 'Download updates while installing'.

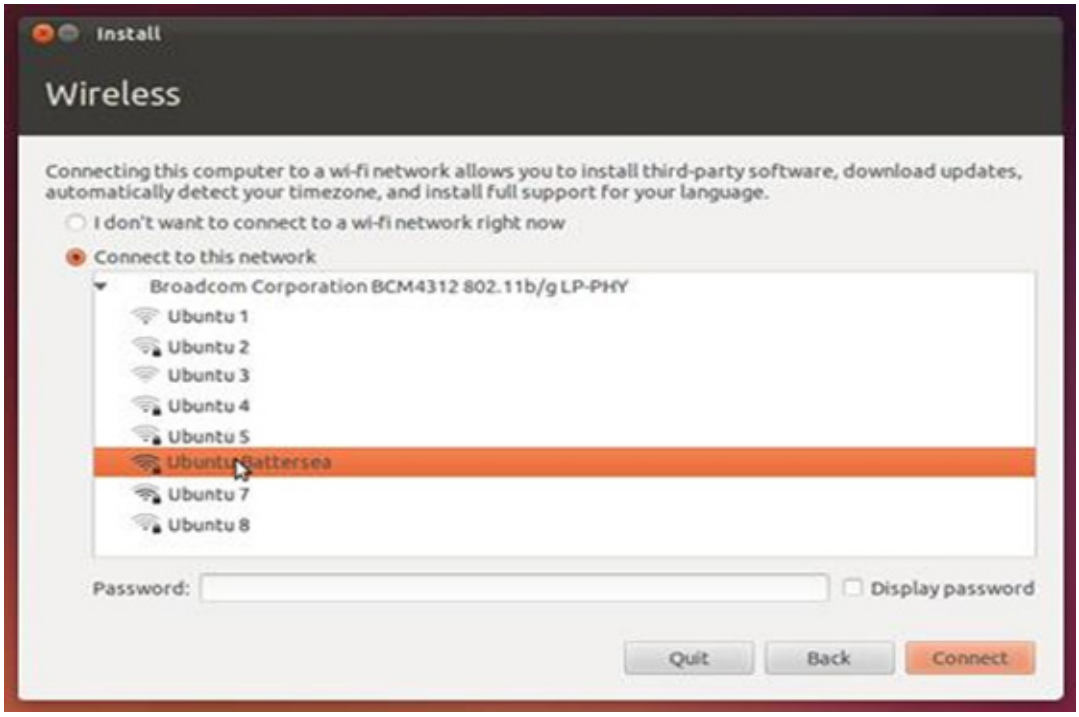


Figure 6.34. Connecting your system to a Wifi during installation

Allocate drive space

Use the checkboxes to choose whether you'd like to Install Ubuntu alongside another operating system, delete your existing operating system and replace it with Ubuntu, or — if you're an advanced user — choose the 'Something else'option

Figure 6.35. Select your preferred installation option for Ubuntu

Begin the installation

Depending on your previous selections, you can now verify that you have chosen the way in which you would like to install Ubuntu. The installation process will begin when you click the Install Now button.

Ubuntu needs about 4.5 GB to install, so add a few extra GB to allow for your files.

Figure 6.36. Allocating drive space for the Operating system to be installed

Select your location

If you are connected to the Internet, this should be done automatically. Check your location is correct and click'Forward' to proceed. If you're unsure of your time zone, type the name of the town you're in or click on the map and we'll help you find it.

TIP: If you're having problems connecting to the Internet, use the menu in the top-right-hand corner to select a network.

Figure 6.37. Screenshot showing your location selection

Select your preferred keyboard layout

Click on the language option you need. If you're not sure, click the 'Detect Keyboard Layout' button for help.

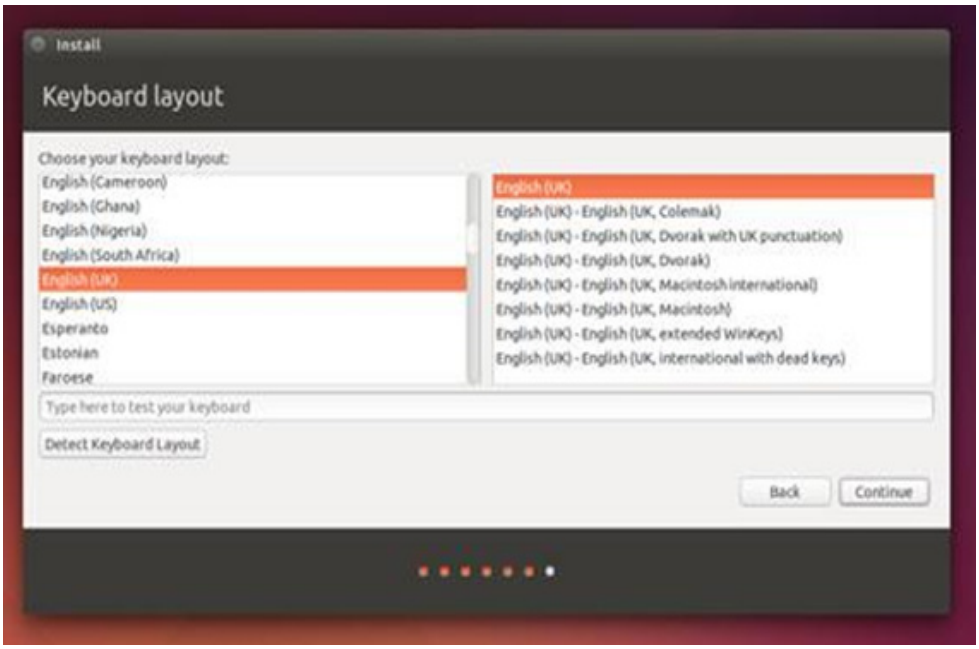


Figure 6.38. Menu to select preferred keyboard layout

Enter your login and password details

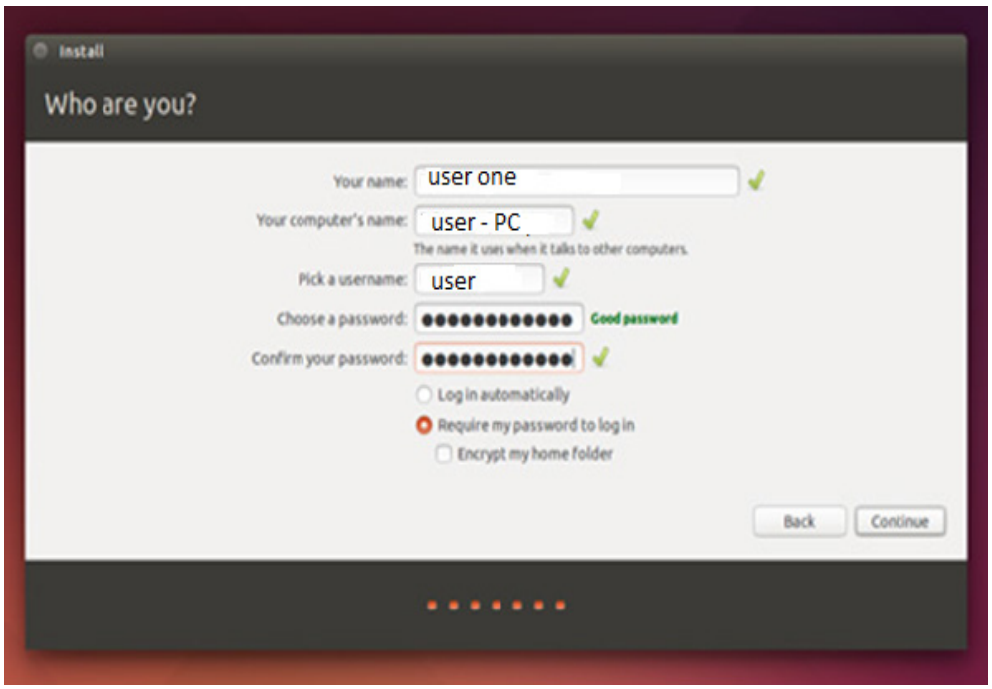


Figure 6.39. Menu to specify your credential information

Learn more about Ubuntu while the system installs...

Read the features of Ubuntu 14.04 while installing. A brief slideshow explaining about Ubuntu while the system installs. The setup will take something like 10-40 minutes, depending on your hardware and Internet connection, especially if you selected updates and third-party software earlier.

Figure 6.40. Screenshot captured while Ubuntu starts t be installed

Finalization.

All that's left is to restart your computer and start using Ubuntu. Once the procedure complete, you can continue using Ubuntu in the live session, but no changes will be persistent. You will have to reboot and select your operating system in the boot menu.

Results and Submission Requirments

The student has to install a fresh copy of Ubuntu 14.04.1 operating system.

Assessment Criteria

The student has to conduct installation of the operting system obtain total of 4 marks.

References or Key Links

- http://www.ubuntu.com/download/desktop/install-ubuntu-desktophttp://www.ubuntu.com/download/desktop/install-ubuntu-desktop
- http://www.ubuntu.com/download/desktop/install-ubuntu-desktop

Basic Configuration of OS

After installing the operating systems, you can conduct configuration to customize the features of the operating system. Different tools are used for configuration of the operating systems.

Laboratory Activity

Operating System Installation

Objectives of the Labortory

Basic configuration of Windos 7 and Ubuntu 14.04.1 operatinig sytems

Details of the Laboratory Exercise/Activities

Objective of the Exercise

Basic configuration of Windows 7 operating system

Resources Required

Personal computer installed with Windows 7 operating system

Time Requried

1 Hour

Description of the Laboratory Exercise/Activities

Configuration of Windows 7

How to Start the System Configuration (msconfig.exe) Tool?

As with most Windows tools, you can start this utility in several ways. In Windows 7, search for "system" or "system configuration" in the Start Menu and click its shortcut.

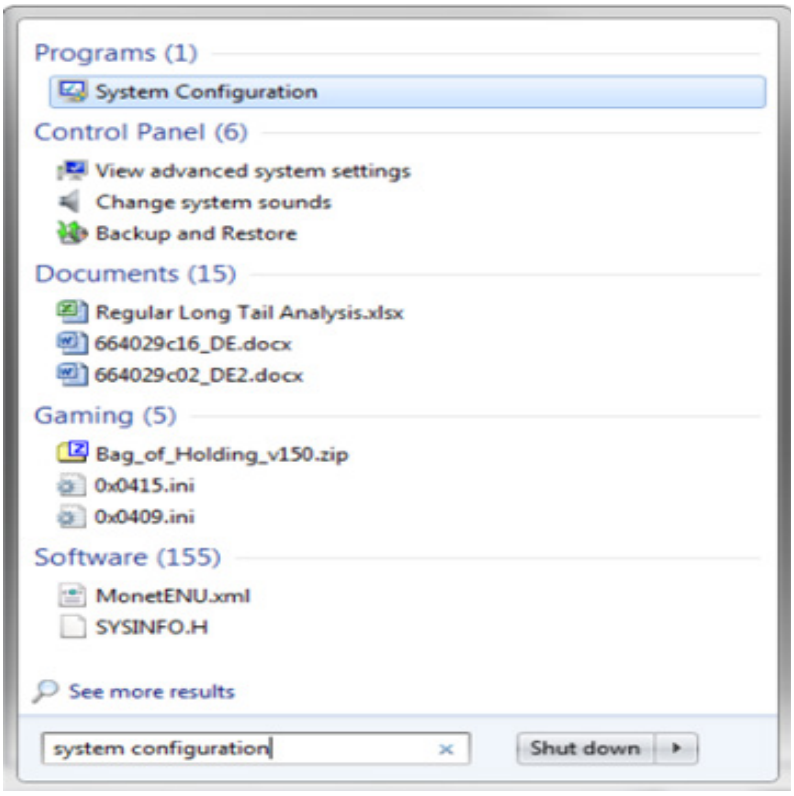


Figure 6.41. System Configuration Utility for Windows 7 from the start menu

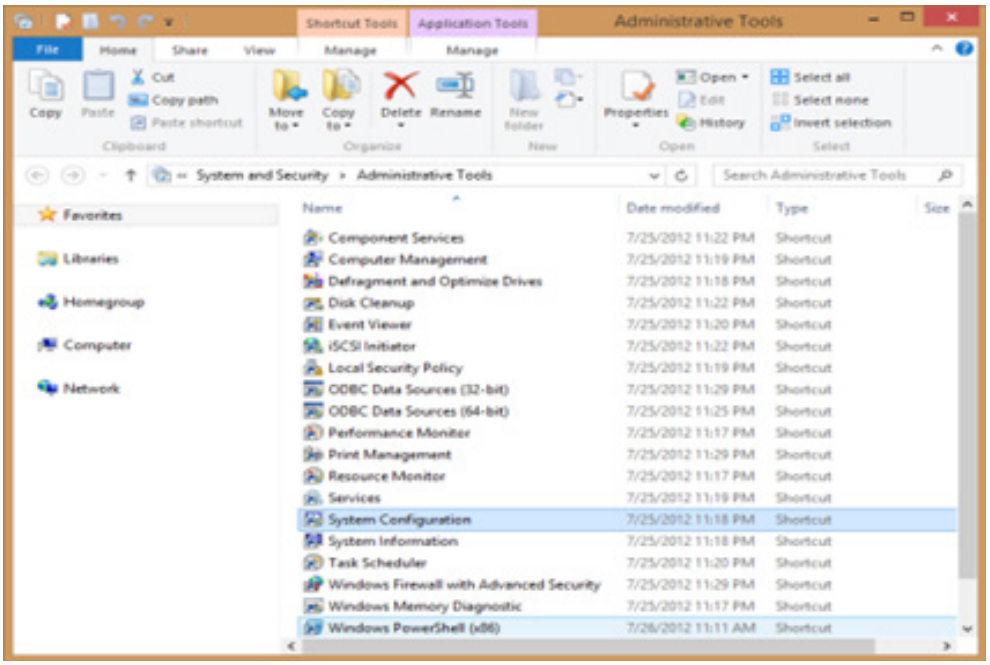


Figure 6.42. ystem Configuration Utility for Windows 7 from the run menu

Another method that works in all operating systems is to type msconfig.exe in the Run window, the Start Menu search box (in Windows 7).

The Windows Startup Selection

In the General tab, there are options available for how you want Windows to start up:

Normal startup - starts Windows as is, with ALL the installed startup items, drivers and services.

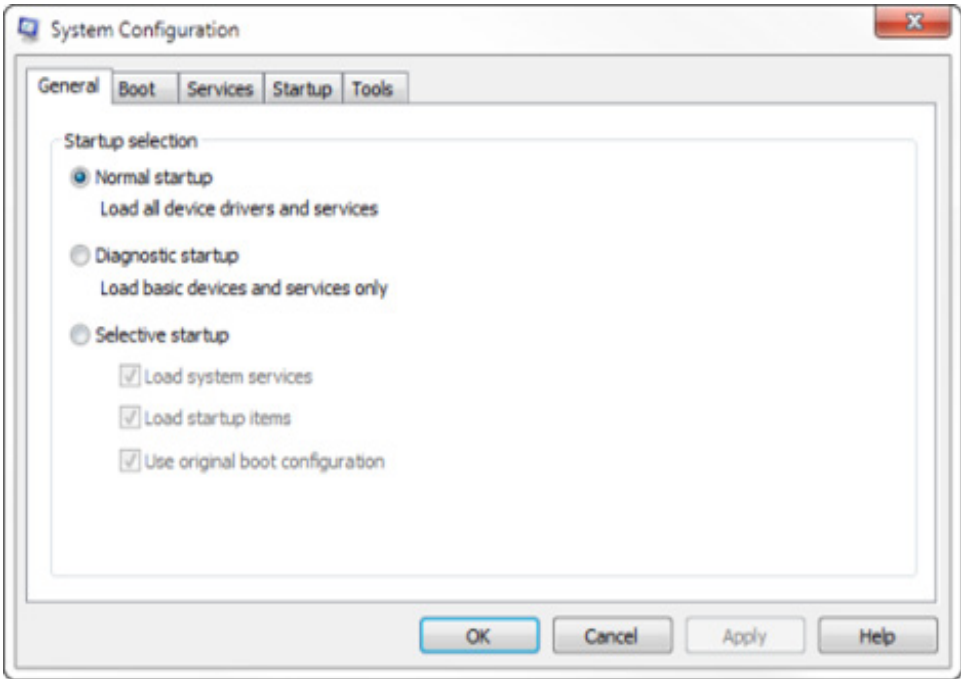


Figure 6.43. Windows startup selection menu

Diagnostic startup- this mode is similar to booting in Safe Mode.Safe Mode runs only Windows services and drivers. This mode might run, on top of them, networking services or important services from third-party applications such as your antivirus, firewall or security suite.

This mode is useful if you want to rule out Windows files and services as being the source of possible system stability problems. If you select it and clickApply, you will notice that Selective startup is then shown as selected.

Selective startup- starts Windows with its basic services and drivers. Also, it allows you to select other services and startup items you want to run, from the Services and Startuptabs.

Configuring the Boot Procedure

The Boot tab is a very important one. Here you can view the operating systems installed on your computer and select the default one – if the machine is a multi-booted.

To select a new default operating system, click on it and then on "Set as default".

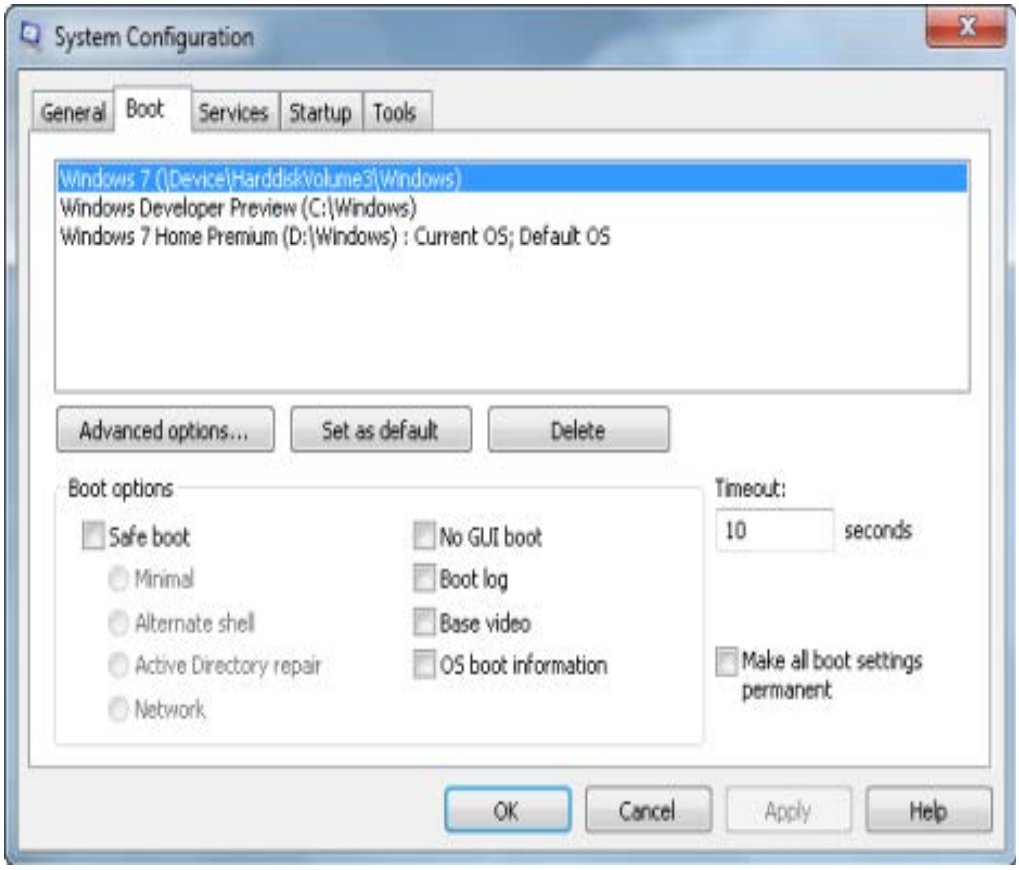


Figure 6.44. Boot tab

For each of the existing operating systems, if you click on Advanced options, you can set things such as the number of processors (cores) allocated to the operating system at boot, or the amount of RAM memory available to it.

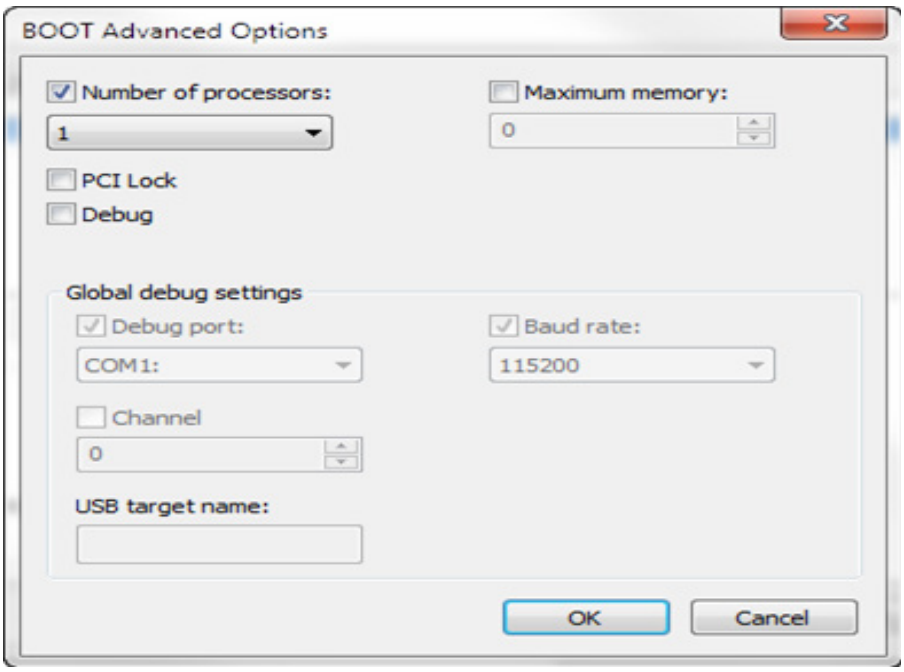
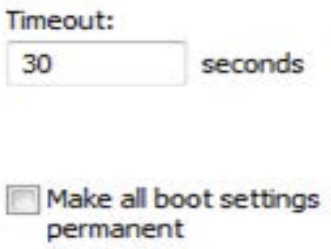


Figure 6.45. Boot Advanced Options

IMPORTANT: If you set a maximum number of cores and RAM memory, Windows will continue to correctly identify the number of cores the processor has and the amount of physical RAM memory. However, it will only use the number of cores (processors) and the maximum memory you set.

Another interesting setting (in case of a multi-boot setup) is the Timeout. The number of seconds you set represents how much your computer will wait for you to select between the available operating systems, when booting. If no choice is made during the set time, the default operating system will start.



By default, this is set to 30 seconds. If you have a multi-boot setup, you might want to set it to a smaller value.

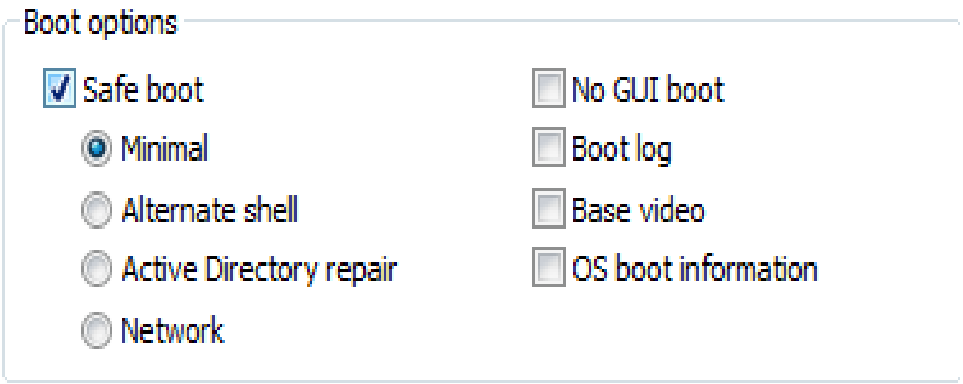


Figure 6.46. Boot type selection

For each operating system, you can also select if you want to make a Safe boot or not, using any of the available options:

Minimal - the normal safe boot, with a user interface and no networking services enabled.

Alternate shell- opens the Command Prompt in safe mode. The networking services and the graphical user interface are disabled.

Active Directory repair- a normal safe boot which runs, additionally, the Active Directory services and features.

Network - the normal safe boot with networking services enabled.

Then you have a set of options which can be applied to both normal and safe mode boot procedures:

No GUI boot - during boot, you are not shown the usual loading screen, only a black screen with no information.

Boot log - during boot, Windows write a complete log with information about the startup process. Usually, it can be found at this location: "C:\Windows\Ntbtlog.txt".

Base video- this option is very useful if you just installed bad video drivers. It makes a normal Windows startup, with the difference that it loads only the standard video drivers that come with Windows, instead of the ones specific to your video card.

OS boot information - this option should be used together with No GUI Boot. The usual Windows loading screen will get replaced with a black screen, displaying complete information about the drivers that are loaded during the startup process. If your Windows crashes during boot, this visualization mode can be useful to identify the driver that causes the crash.

NOTE*:* Using the Boot tab works very well with Microsoft Windows operating systems. If you have a multi-boot setup involving non-Microsoft operating systems, you might need to use other tools for managing the boot procedure.

Managing Startup Services

The Services tab shows a list with all the services that start when Windows starts. For each service you see its name, the manufacturer, the current status and the date when it was disabled (if it was disabled).

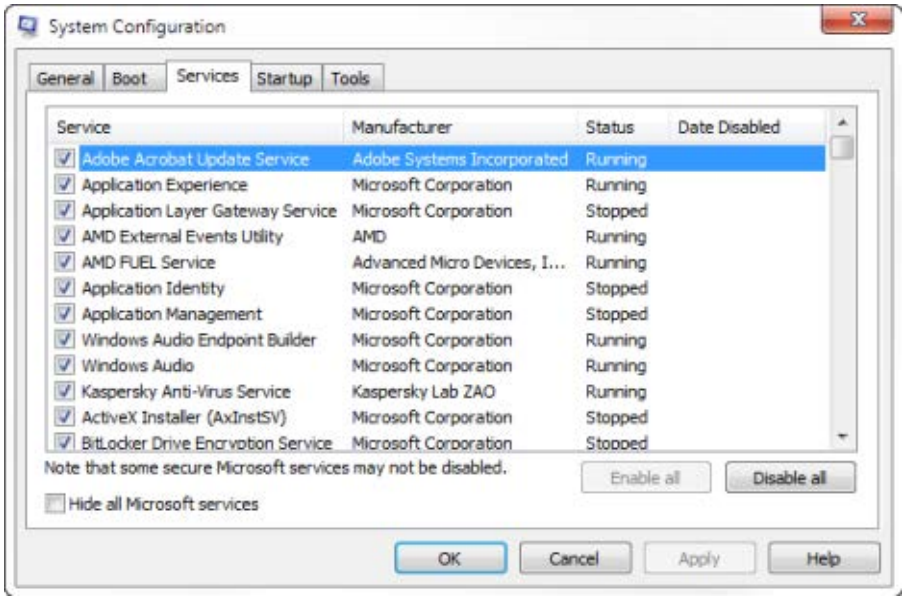


Figure 6.47.Startup service management

You can check the services you want to run at startup and uncheck the ones you don't. If you want to see only third-party services, installed by your applications, check the box which says"Hide all Microsoft services".Also, please note that the selections you make in this tab are applied to your current startup selection, from the General tab.

If you were using a Normal startup and then you disabled some services, the startup selection gets changed automatically to Selective startup.

Managing Startup Programs

The Startup tab shows a list with all the programs and files that start when Windows starts. For each items you see its name, the manufacturer, the "command" used to start it (the path towards the program and additional parameters, if used), the registry startup location where it is stored and the date when it was disabled (if it was disabled).

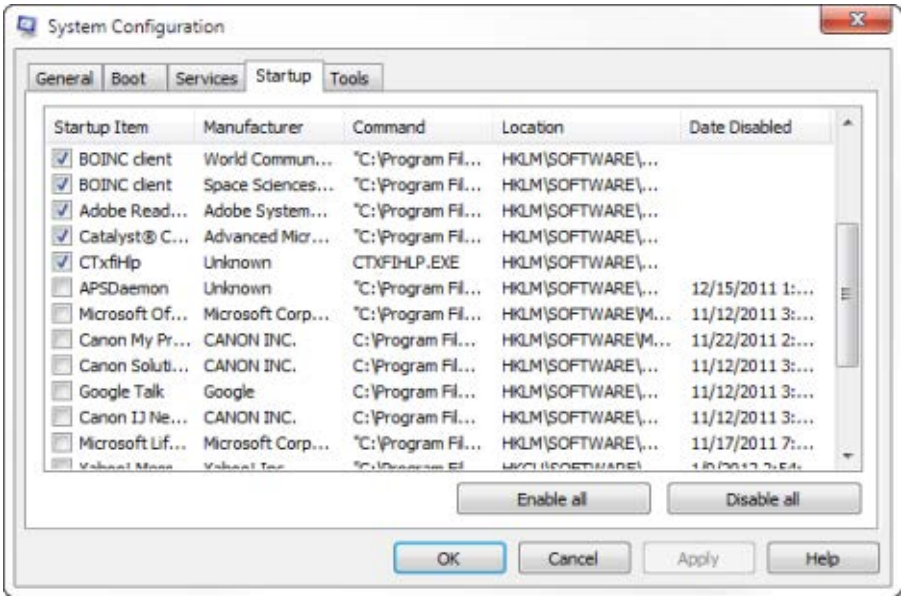


Figure 6.48. Managing startup programs

An interesting aspect to remember about the registry location is that, if you see a location starting with HKLM, it means that the startup item is "global" - applied to all user accounts defined on the active operating system. Disabling them from one user account, means that they get disabled for all user accounts.

The locations starting with HKCU are for startup items active only for the current user account. They might not be starting up for other user accounts. Also, such items need to be disabled individually, for each user account, if you want to prevent them from starting up completely.

Just like the Services tab, the selections you make are applied to your current startup selection, from the General tab.

Launching Administrative Programs and Panels

This section deals with the Tools tab and its functionality. If you click on it, you get a list with Windows administrative tools such as: the Action Center, the Windows Registry, Event Viewer, etc.

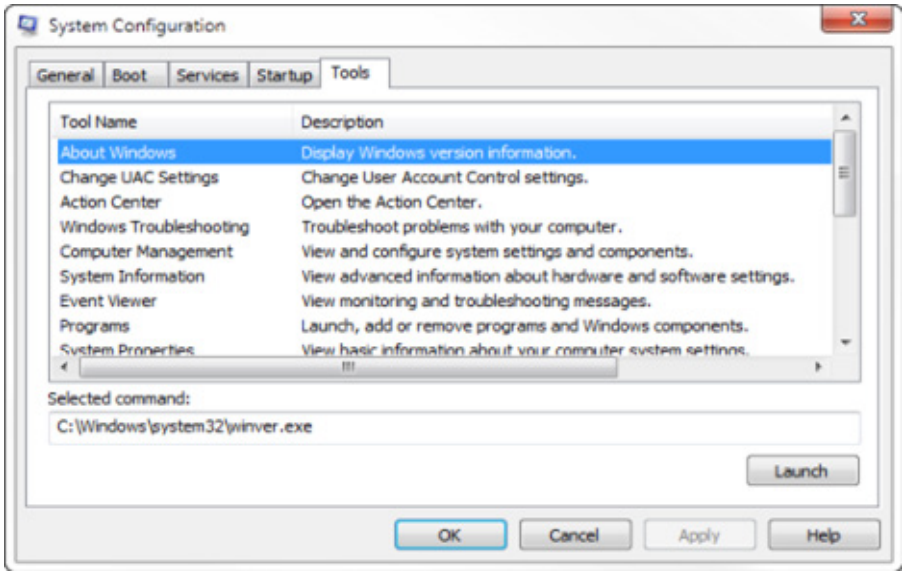


Figure 6.49. Tools Menu of System Configuration

For each tool, you see its name and description. If you click on it, you can see the command used to start it, in the "Selected command" field.

To run any of the available tools, select the one you want and click Launch.

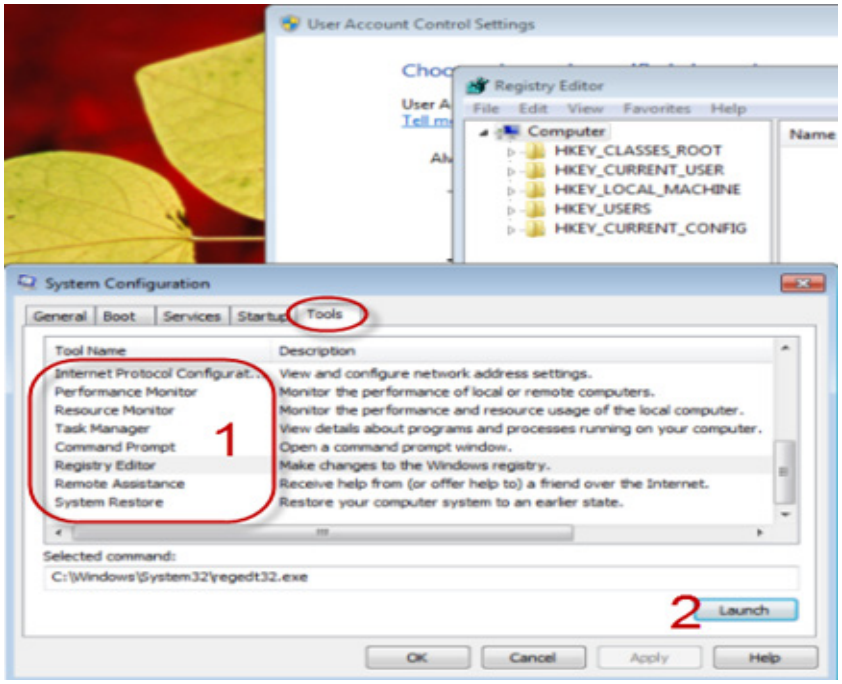


Figure 6.50. Tools name with their description

This tab is pretty handy as it lists administrative tools generally used during troubleshooting system stability or performance problems.

Saving your Settings

After making all the changes you want, don't forget to press Apply or OK, so that they get applied.

Also, if you are using the tool for the first time, when closing it, you will be informed that you need to restart your computer in order for the changes to get applied.

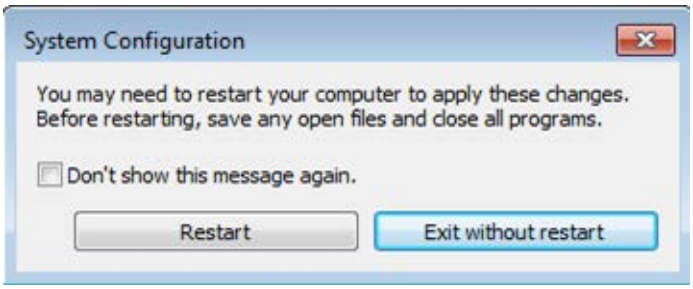


Figure 6.51. Option to restart the system to apply the changes

If you don't want to see this message again, check the box which says "Don't show this message again" and choose the restart option you prefer.

Results and Submission Requirments

The student has to perform the appropriate configuration of Windows 7 operating system.

Assessment Criteria

The student has to conduct configuration of the operting system to obtain total of 3 marks.

References or Key Links

http://windows.microsoft.com/en-us/windows/start-system-configuration#1TC=windows-7

Details of the Laboratory Exercise/Activities

Objective of the Exercise

Basic configuration of Ubuntu 14.04.1 operating system

Resources Required

Personal computer installed with Ubuntu 14.04.1 operating system

Time Requried

- 1 Hour

Description of the Laboratory Exercise/Activities

Configuration of Ubuntu 14.04

The recent versions of Ubuntu have used Unity as a default user interface for the GNOME-3 desktop environment.

Adjust the Launcher Icon Size

Ubuntu includes a launcher on the left of your screen. If it appears that the size of the icons on the launcher is either too small or too big, you can adjust it to the size you like.

- Click the Control Gear and select "System Settings".
- Click "Appearance" under "Personal".
- Under the Look tab, drag the slider of "Launcher Icon Size" to the left for a smaller size, or right for a bigger size.

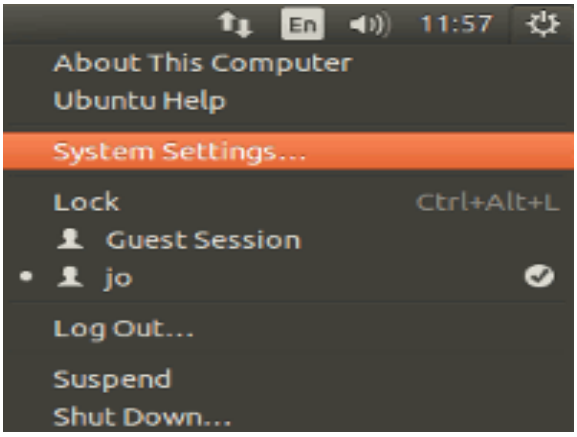


Figure 6.52. System setting menu for Ubuntu

Auto Hide the Launcher

Depending on the version of Ubuntu you use, the Launcher is set to always either appear or hide on the screen by default. You can change this default setting easily.

- Click the Control Gear and select "System Settings".
- Click "Appearance" under "Personal".
- Under the "Behavior" tab, switch on or off the button of "Auto-hide the Launcher".

To reveal the Launcher temporarily, just press and hold the Windows key, or move your mouse cursor to the far left of the screen. You can adjust the reveal sensitivity with the slider under the "Behavior" tab mentioned above.

Disable or Enable the Global Menu

Ubuntu system places the application menu (File/Edit/View) (also known as AppMenu or Global Menu) on the top bar by default, but it also allows for a locally integrated menu (LIM). If you want to opt for an LIM which moves the menu back to the application's window, follow these steps:

- Click the Control Gear and select "System Settings".
- Click "Appearance" under "Personal".
- Under the Behavior tab, check "In the window's title bar" to show the menus for a window.
- The menu appears when you mouse over the window's title bar. To enable the global menu, re-check "In the menu bar" to show the menus for a window.

Customize the Theme

Themes in Ubuntu can be customized to suit your needs. The default theme "Ambiance" has the menu (File, Edit, View, etc) printed in white on a dark background, but you can easily switch to another theme such as "Radiance" to have the menu printed in black on a light background. These are easy steps to customize a theme.

- Click the Control Gear and select "System Settings".
- Click "Appearance" under "Personal".
- Under the Look tab, select a theme from the drop-down menu.

Change Wallpapers Automatically

You can right click your desktop, select "Change Desktop Background" and choose any one of the wallpapers pre-installed, but you need to do it each time you want to change a wallpaper. What about changing a wallpaper automatically within a certain time interval? Try to install the wallpaper changer Variety.

- Press Ctrl-Alt-T to run Terminal.
- Enter sudo add-apt-repository ppa:peterlevi/ppa
- Enter sudo apt-get update
- Enter sudo apt-get install variety

With this wallpaper changer, you can change wallpapers in a fixed time interval from the sources you set and run it with several other settings such as randomly applying filter effects, color and size, customizing the indicator icon and so on.

Set a Default View in Files Manager

Windows Explorer allows for users to set a default view to all folders. In almost the same way, Ubuntu's Files Manager allows for these settings:

- Click the Files icon on the Launcher.
- From the menu, click "Edit" then "Preferences".
- Under Default View, change "Icon View" to "List View", to see more details in columns.
- Tick "Show hidden and backup files" if that's your choice.
- Other various settings, such as single or double click to open items, icon captions, list columns and preview files can be done in the same Files Preferences window as well.

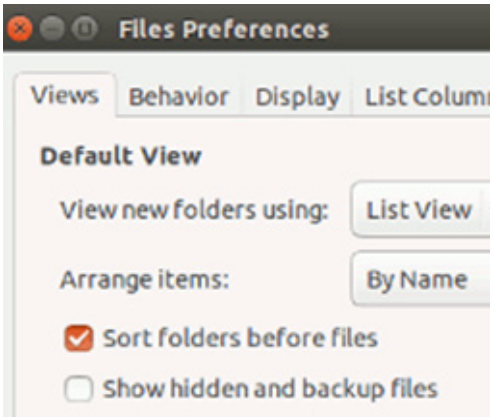


Figure 6.53. File preference menu

Terminate Unresponsive Programs

Xkill is part of the X11 utilities pre-installed in Ubuntu and a tool for terminating misbehaving X clients or unresponsive programs. You can easily add a shortcut key to launch xkill with the steps below.

- Click the Dash Home icon on the Launcher (or tap Super), then typekeyboard into the Search box and press Enter.
- Under the "Shortcuts" tab, select Custom Shortcuts, then click the "+" sign to create a custom shortcut.
- Enter xkill to both the Name and Command boxes and click the Applybutton.
- Click on Disabled at the xkill row in the Keyboard Shortcuts window (Disabled is then changed to New accelerator...).
- Press a new key combination, e.g. Ctrl+Alt+X (New accelerator... is then changed to Ctrl+Alt+X).
- Xkill is ready for use. Press the above key combination to turn the cursor to an X-sign, move the X-sign and drop it into a program interface to terminate the unresponsive program, or cancel the X-sign with a right-click.

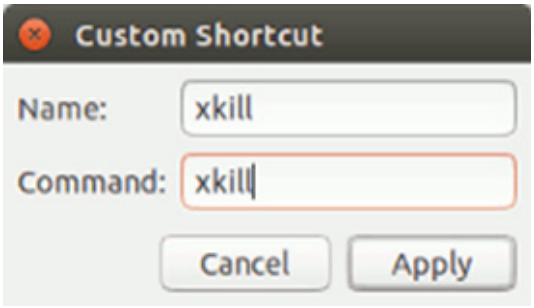


Figure 6.54. Shortcut for the xkill utility

Auto Mount Drives at System Startup

Ubuntu is capable of reading and writing files stored on Windows formatted partitions using NTFS file system, but partitions must be 'mounted' before they can be accessed. With these steps, you can auto mount the drives or partitions without the need to manually mount them for access each time you start up the system. Below is a way of doing it by adding an entry in the fstab file.

- In the Terminal, enter sudo blkid to get the UUID (Universally Unique Identifier) of the partition you want to mount.
- Enter sudo mkdir /media/ntfs to create a mount point
- Enter gksu gedit /etc/fstab and add the following line in the fstab file: UUID=1234567890123456/media/ntfs ntfs rw,nosuid,nodev,noatime,allow_other 0 0
- Replace the above 16-digit number with the UUID you've got from step 1, then click 'Save'.
- Restart the system and check if the partition is mounted.

Manually Mount a USB Drive

A USB storage device plugged into the system usually mounts automatically, but if for some reasons it doesn't automount, it's possible to manually mount it with these steps.

- Press Ctrl-Alt-T to run Terminal.
- Enter sudo mkdir /media/usb to create a mount point called usb.
- Enter sudo fdisk –l to look for the USB drive already plugged in, let's say the drive you want to mount is /dev/sdb1.
- Enter sudo mount -t vfat /dev/sdb1/media/usb-o uid=1000,gid=100,utf8,dmask=027,fmask=137 to mount a USB drive formatted with FAT16 or FAT32 system. OR:
- Enter sudo mount -t ntfs-3g /dev/sdb1/media/usb to mount a USB drive formatted with NTFS system.
- To unmount it, just enter sudo umount /media/usb in the Terminal.

Auto Start Up an Application

In Windows, you can place a program shortcut in a startup folder for running a program automatically when the system starts. In Ubuntu, you can do the same in this way:

- Click the Dash Home icon (or tap Super), type 'Startup Applications' to search for the application and run it.
- Click the "Add" button.
- Name a program.
- Click the "Browse" button and navigate to "Computer" > usr > bin, where programs are usually installed.
- Select a program, click the "Open" button followed by the "Add" button.

The above program will then be listed in additional startup programs. Check if the program runs automatically by logging out and back to the system.

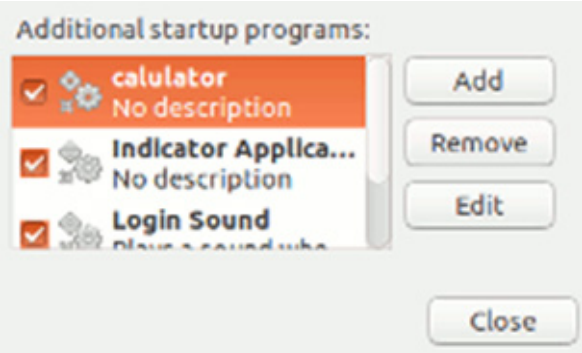


Figure 6.55. Startup Programs

Change Default Boot Options

After full installation, Ubuntu is set to be the default operating system to boot up if no key is pressed within a few seconds on a multi-boot system. You might want to set your preferred operating system to boot up by default. This can be done easily with Grub Customizer.

- Press Ctrl-Alt-T to call up Terminal, copy following codes and paste (Ctrl-Shift-V) them inside Terminal to install Grub Customizer.
- Sudo add-apt-repository ppa:danielrichter2007/grub-customizer
- Sudo apt-get update
- Sudo apt-get install grub-customizer
- After installation, run Grub Customizer to set the default boot options with the following steps.
- Press Alt-F2, type Grub Customizer into the box and press Enter to run it.
- Under the "General Settings" tab, select the default entry you like to boot up from the drop-down menu.

- Adjust the timeout value if needed, then press the Close button and the Save button.
- Avoid changing timeout to 0 seconds if you need to select a system to boot up from a multi-boot menu.

Auto Shutdown the System

A simple command can be entered in the Terminal to schedule a time for the system to shut down.

- Press Ctrl-Alt-T to run Terminal.
- Enter sudo shutdown -h +m (replace m with the number of minutes, e.g. +60).
- OR: enter sudo shutdown -h hh:mm (replace hh:mm with the time on the 24hr clock, e.g. 23:15).

Enter password and minimize the Terminal window.

The system will then shut down within the minutes or at the time specified. To cancel a scheduled time, entersudo shutdown -c in the Terminal.

Results and Submission Requirments

The student has to perform the appropriate configuration of Ubuntu 14.04.1 operating system.

Assessment Criteria

The student has to conduct configuration of the operting system to obtain total of 3 marks.

References or Key Links

http://howtoubuntu.org/things-to-do-after-installing-ubuntu-14-04-trusty-tahr

Conclusion

Identification of requirement of an operating system has paramount importance. Checking the architecture of the microprocessor, memory and storage space requirements for the importing system is crucial. Moreover, we have to take a backup of device drivers.

We choose to install Window 7 and Ubuntu 14.04 OSs. Due to their popularity and functionality. The former being proprietary and the latter being open source.

To configure Windows 7, as you can see, the System Configuration (msconfig.exe) utility is very versatile and offers functionality very few people know about. It can be a great tool for managing the startup process of your computer and for troubleshooting stability and performance problems.

In Ubuntu 14.04, you can use different tools to configure the system according to your preferences.

Assessment

1. Discuss the hardware and software requirements of an operating system
2. Have you observed any different requirements between the Windows 7 and Ubuntu OS covered?
3. What are the importance of configuration once system is installed?

Activity 2: Introduction to Real Time and Embedded OS

Introduction

A real time operating system is category of an operating system that is used to handle real-time applications. Real time applications are those which guarantee both correctness of result and the added constraint of meeting a deadline. So a real-time system is a computer system which needs to produce a result within a specified deadline period. Thus results produced after the deadline has passed, even if correct may be of no real value. To achieve such objective, a real-time system can't use the scheduling and other process management algorithms discussed so far for the traditional, non-time constrained operating systems. Some modifications need to be included in those algorithms so as to be used in a real time environment. In this activity, we will try to address issues related with real-time operating systems and the changes arose to run these software.

Real-time systems can run on traditional computer hardware or specially designed devices such as home appliances, communication devices, etc being embedded on them. An embedded system is thus a computing device that is part of a larger system in which the presence of a computing device is often not obvious to the user. It is a reactive computing situation where communication is through reaction/ response to external real world events. For example, consider an embedded system for controlling a home dish-washer. The embedded system may allow various options for scheduling the operation of the dishwasher-the water temperature, the type of cleaning or even a timer indicating when the dishwasher is to start. Most likely, the user of the dishwasher is unaware that there is in fact a computer embedded in the appliance. Though a real-time system can be implemented as an embedded system, it is good to note that not all embedded systems are real time. For instance, an embedded system controlling a home furnace may have no real-time requirements whatsoever. According to the requirement it has, a real time system can be either hard or soft.

A hard real time system is a one with the most strict time requirements assuring very critical real time tasks completed within the deadline. These systems cause a severe problem when an incorrect operation, usually due to a missed deadline happens. Examples are weapons systems, antilock brake systems, flight-management systems, and health-related embedded systems. In all these cases, unless the system responds for events according to the time schedule, a catastrophic problem arises

A soft real time system is a bit flexible and less restrictive system than the hard one. It gives high priority for critical real-time tasks and maintains that priority until it completes. Many commercial operating systems as well as Linux provide soft real-time support.

Requirements of real time systems

A real time system is generally a system with specific purpose, small size, inexpensively mass-produced and with Specific timing requirements. Real time systems are designed to a single purpose and you can't use it to do other tasks than the specified one during design. Moreover, most real time systems are used in space constrained environments which also places a CPU and memory limitation to these systems. As these systems are also mostly used with home appliances and consumer devices which are mass manufactured in cost aware environments, the processors of such systems also needs to be manufactured by with less cost. Lastly, the time requirement is the defining characteristics of a real time system which makes it to react to events in a timely manner. In general, a real time system has three main requirements

1. **Predictability :** The timing behavior of the operating system needs to be predictable
2. **Management***:* The operating system should manage timing and scheduling of as it possibly has to be aware of task deadlines. Moreover, the OS needs to generate time constrained functionalities of high quality.
3. **Speed***:* The OS need to be quick in all responses and this is practically important.

The traditional operating systems have the capability of supporting several peripheral devices like input, output, etc. , multiple users, protection mechanisms against any possible threat, and virtualization to enhance its memory space where all these create some complexity on the system kernel. In contrast, real time systems are designed simply with small number of codes and have none of these functionalities. The main reasons as to why real time systems fail to do so are:

- Because most real time system are special purpose system with serving only a single task. Furthermore, a typical real-time system does not include the notion of a user. The system simply supports a small number of tasks, which often gets its input from hardware devices such as sensors.
- Because it is impossible to give the features supported by standard desktop operating systems without fast processors and abundant amounts of memory which are both unavailable in real-time systems.
- Because supporting features common in standard desktop systems by real time systems would make them costly and economically impractical.

When it comes to real time systems implementation, the scheduling algorithms discussed for the general purpose operating systems can not be used as they are. Some modification according to the requirements of a real time system should be made to utilize the algorithms. Let's identify and discuss some features necessary for implementing a real-time operating system.

Priority based scheduling: a real time operating system must support a priority with pre-emption based scheduling algorithm as it needs to respond immediately for real time processes to grant their request of CPU. This algorithm works by assigning a priority value to each process according to their importance and executes them based on the value. A soft real time system can more preferably use this method as it doesn't have other serious requirements.

Pre-emptive kernels*:* to meet the strict timing requirement of a hard real time system, a real time system kernel should be designed in a pre-emptive way interrupting tasks that run in the kernel mode. If not, a real-time task might have to wait an arbitrarily long period of time while another task was active in the kernel.

Latency minimization*:* a real time system is an event driven system waiting for a real world event to happen to commence its activities. The events may arise either in software as when a timer expires or in hardware as when a direction sensing device detects that it is approaching to a crossing. When such an event happens, the system needs to immediately respond and service the requests made. Latency is thus the amount of time that elapses from when an event occurs to when it is serviced. Interrupt latency and dispatch latency are the two types of latencies affecting the performance of a real time system. Interrupt latency is the time elapsed by an operating system when an interrupt occurs to first complete the instruction it is currently executing and determine the type of interrupt that occurred, then save the state of the current process before servicing the interrupt. So it is very important for real time systems to have small interrupt latency time as much as possible so as to process real time tasks quickly. Dispatch latency is the time elapsed by scheduling dispatcher to stop one process and start another. The smaller the dispatch latency is, the better the performance of the real time system and one way of reducing it is carried out through implementation of pre-emptive kernel though a situation that can affect dispatch latency may arise when a higher-priority process needs to read or modify kernel data that are currently being accessed by a lower-priority process forcing the higher-priority process wait until the lower-priority one finishes with the resource. This situation is known as priority inversion and can be solved by implementing by using protocols called priority inheritance protocols.

CPU scheduling in Real-time systems

Processes in a hard real time system are considered periodic with requiring the CPU at constant intervals, each with fixed processing time once CPU is granted, has a deadline by which time it must be serviced by the CPU. Scheduling algorithms that address these requirements need to be used by the CPU. Rate monitoring scheduling, earliest deadline first scheduling, proportional share scheduling, and pthread scheduling are some of the algorithms used to perform CPU scheduling in real time systems.

Classification of Real-time and Embedded OS

As mentioned earlier, real-time systems can be classified as:

Hard real time system

A system whose operation is degraded if results are not performed according to specified timing requirements. System response occurs with in a specified deadline. Failure to meet such a timing requirement can have catastrophic consequences where it is absolutely imperative that responses occur with in the required deadline. Example flight control systems, automotive systems, robotics, etc.

Soft real time system :

Is a system whose operation is incorrect if results are not produced according to the timing constraints. Catastrophic results will happen then. The response times are important but not critical to the operation of the system. Failure to meet the timing requirements would not impair the system. These are systems where deadlines are important but which will still function correctly if deadlines are occasionally misused. Example: Banking system, multimedia etc.

Conclusion

Traditional operating systems are general purpose systems managing the resources of the system and satisfying user needs with better performances. Special types of operatig systems, known as real time systems are those which has a time constraint defined on them on top of these services as the output of an activity is expected to be used as an input to another activity and failing to do so results in system damage. A real time system is embedded on special types of devices or it can also be run on traditional hardware. Embedded system is a system developed for such special devices and operates seamlessly on the device. A real time system can be either hard or soft system based on the strict rule it follows to deadlines. A real time system has its own requirement that needs to be addressed to give services to applications and users. It is generally designed in such a way as small size, time constrained, cheap and single task oriented. It is difficult and also useless to make these systems support the different services provided by general purpose operating systems because of the design properties. A real time system scheduler needs to support pre-emptive priority based scheduling, latency minimization scheduling and with kernel pre-emptives scheduling.

Assessment

1. Go through the various CPU scheduling algorithms used in a real time system. Compare and contrast these algorithms with a general purpose operating system scheduling algorithms
2. What is the difference between hard real time Os and soft real time OS?
3. Discuss at least three requirements imposed on real time OS?
4. What is an embedded system?

Activity 3: Basic Virtual Machines

Introduction

Virtualization is a mechanism that came to existence around 1970s by which an existing interface of a system is extended to mimic the behavior of another system. It's used mainly to handle platform diversity by running applications on the Virtual Machines(VM) which simplified portability and flexibility issues. Concerns related with VM are:

- How are VMs possible? what are the technologies out there?
- Is it possible to move between VM and actual/physical device?
- Is managing a VM a complex task?

This activity tries to elaborate these concerns

Overview

A virtual machine program is a computer program that creates a virtual computer system, complete with virtual hardware devices. This virtual computer "machine" runs as a process in a window on your current operating system. You can boot an operating system installer disc (or live CD) inside the virtual machine, and the operating system will be "tricked" into thinking it's running on a real computer. It will install and run just as it would on a real, physical machine. Whenever you want to use the operating system, you can open the virtual machine program and use it in a window on your current desktop.

Virtual machine executes on your current operating system – ,the "host" operating system, – which offers virtual hardware to "guest" operating systems. The guest operating systems run in windows on your host operating system, just like any other program on your computer. The guest operating system runs normally, as if it were running on a physical computer – from the guest operating system's perspective, the virtual machine appears to be a real, physical computer.

Virtual machines provide their own virtual hardware, including a virtual CPU, memory, hard drive, network interface, and other devices. The virtual hardware devices provided by the virtual machine are mapped to real hardware on your physical machine. For example, a virtual machine's virtual hard disk is stored in a file located on your hard drive.

Your virtual machine's operating system is stored on a virtual hard drive — a big, multi-gigabyte file stored on your hard drive. The file is presented to the operating system as a real hard drive. This means you won't have to mess around with partitioning.

A virtual machine also gives you a way to run another operating system's software. So, if you're a Linux user, you can install Windows in a virtual machine and run Windows desktop programs in that virtual machine. Mac users can also use virtual machines to run Windows software.

Virtual machines are also "sandboxed" from the rest of your system, which means that software inside a virtual machine can't escape the virtual machine and tamper with the rest of your system. A virtual machine can be a good place to rest out programs you don't trust and see what they do. Advantages of a virtual machine are:

- Flexibility
- Portability
- Isolation
- Security

Virtual Machine Technologies

There are several different virtual machine programs you can choose from among which:

- VirtualBox (Windows, Linux, Mac OS X): VirtualBox is very popular because it's open-source and completely free. There's no paid version of VirtualBox, so you don't have to deal with the usual "upgrade to get more features" upsells and nags. VirtualBox works very well, particularly on Windows and Linux where there's less competition — it's a good place to start out.
- VMware Player (Windows, Linux): VMware has their own line of virtual machine programs. You can use VMware Player on Windows or Linux as a free, basic virtual machine tool. More advanced features require upgrading to the paid VMware Workstation program. We recommend starting out with VirtualBox, but if it doesn't work properly you may want to try VMware Player.
- VMware Fusion (Mac OS X): Mac users will need to buy VMware Fusion to use a VMware product, as the free VMware Player isn't available on a Mac. However, VMware Fusion is more polished.
- Parallels Desktop (Mac OS X): Macs also have Parallels Desktop available. Both Parallels Desktop and VMware Fusion for Mac are more polished than the virtual machine programs on other platforms — they're marketed to average Mac users who might want to run Windows software.
- While VirtualBox works very well on Windows and Linux, Mac users may want to buy a more polished, integrated Parallels Desktop or VMware Fusion program. Windows and Linux tools like VirtualBox and VMware Player tend to be targeted to a geekier audience.

Migration from Physical to Virtual Machines

Building a virtual machine from scratch might take several hours due to the huge time required to setup the hardware parameters, install the operating system, and then add third-party applications to the OS. Fortunately, it is possible to reduce this time by replicating and importing physical systems to your VMware server.

Migration from Physical to virtual (P2V) is a term that refers to the copying of an operating system (OS), application programs and data from a computer's main hard disk to a virtual machine or a disk partition.

There are really two main methods for running aP2V migration. The first is using the free Microsoft Systinternals Disk2vhd utility. The second is by using System Center Virtual Machine Manager (SCVMM) which is part of the Microsoft System Center suite. While both will allow you to accomplish the same goal ultimately, you will have a very different experience with each. Disk2vhd will only convert your physical drives to virtual disks (in VHDx format). Disk2vhd will not build a VM, nor will it create any configuration. It literally only converts a disk to a virtual hard disk.

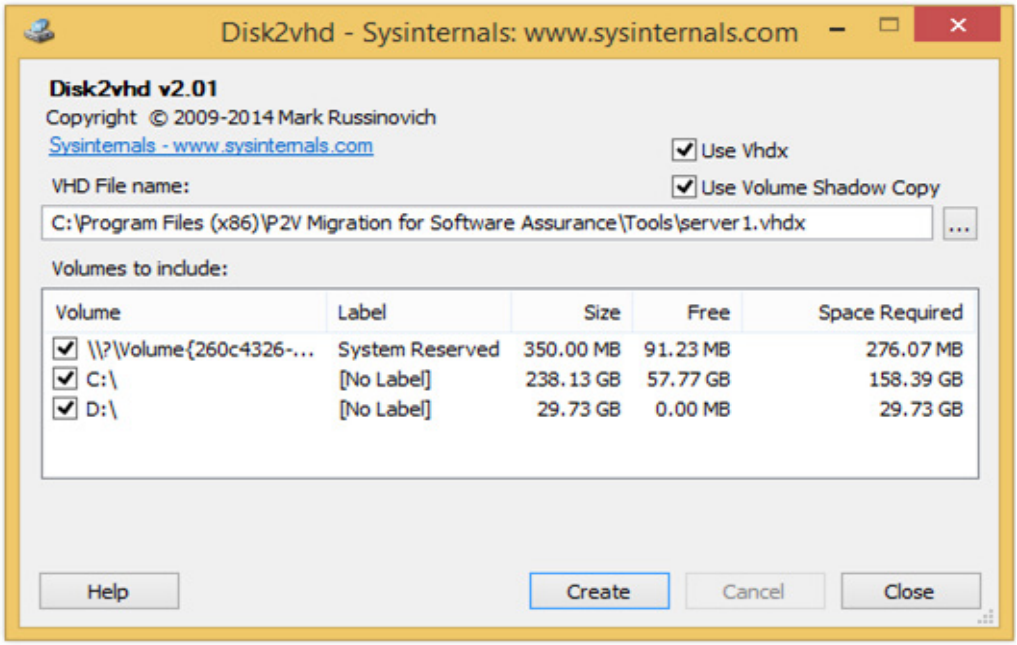


Figure 6.56. P2V using the Microsoft Systinternals Disk2vhd utility

SCVMM, on the other hand, will automate the entire process for you. You will start off with an easy to use wizard which in turn builds a rather large PowerShell script utilizing the functionality and tools built into Virtual Machine Manager. Once your script is created, you can either proceed to execute, or schedule the process to execute at another time.

Laboratory Activity

Managing Virtual Machine

Objectives of the Laboratory

Basic installation and configuration of Oracle Virtual Machine

Details of the Laboratory Exercise/Activities

Objective of the Exercise

Basic installation and configuration of Oracle Virtual Machine

Resources Required

- Personal computer installed with Windows 7 operating system
- For additional requriments, refer to "Minimum Requirments" section of this laboratoy activity

Time Requried

- 1 Hour

Managing Virtual Machines

As virtual mancine technology becomes very much used,physical hosts have been freed from the bounds of single-instance operating systems and users efficiently multiplex their hosts with multiple operating systems as virtual machines (VMs). But, the density of operating systems on a host also increases the management requirements.

Aspects of virtual machine management include:

- Performance monitoring
- Optimizing the mix of virtual machines that should reside on each physical server
- Automating virtual machine provisioning
- Load balancing
- Patch management
- Configuration management and fail-over and enabling policy-based orchestration to automatically trigger the appropriate responses to events

To manage virtual machine, let's consider a specific virtual machine technology – Oracle **VirtualBox and discuss the tasks performed by this VM manager.**

Minimum system requirements:

The Oracle VM Manager template requires approximately 15GB of diskspace on the host filesystem (5GB to host the .ova file download and 10GB for the actual Virtual Machine appliance). The virtual machine is configured with 4GB RAM, to run the software optimally.

Statically allocated IP address and hostname for the Virtual Machine - when configuring the Oracle VM Manager template you are required to provide a static IP address for the VM, along with a fixed hostname, Oracle VM Manager needs to be able to do two-way communication between the Manager instance and the Oracle VM server agents running on different servers. Once an Oracle VM server is discovered by the Manager instance, it expects the same IP address to exist for future communications.

Oracle VM VirtualBox 4.1.8 or later - download and install Oracle VM VirtualBox for your host system prior to starting the installation of the template.

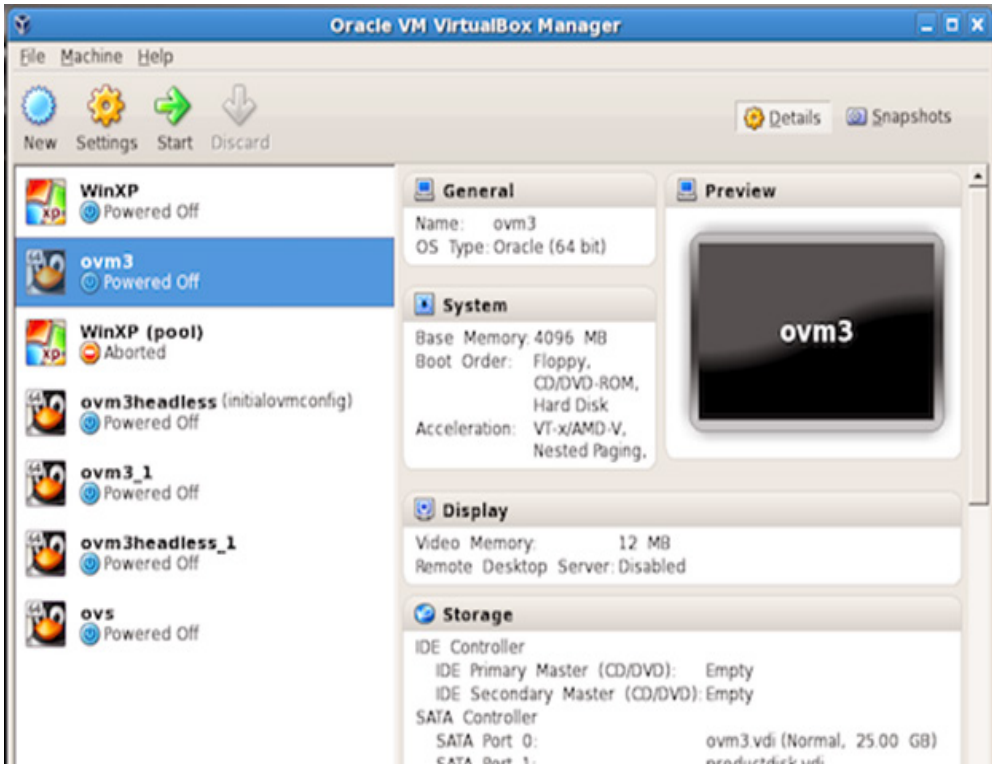


Figure 6.57. Oracle VM VirtualBox Manager Application Interface

Start Oracle VM VirtualBox on your local system. If you already have a number of Virtual Machines created, you will see something similar as in this screenshot. Go to File -> import Appliance to import the Oracle VM Manager template.

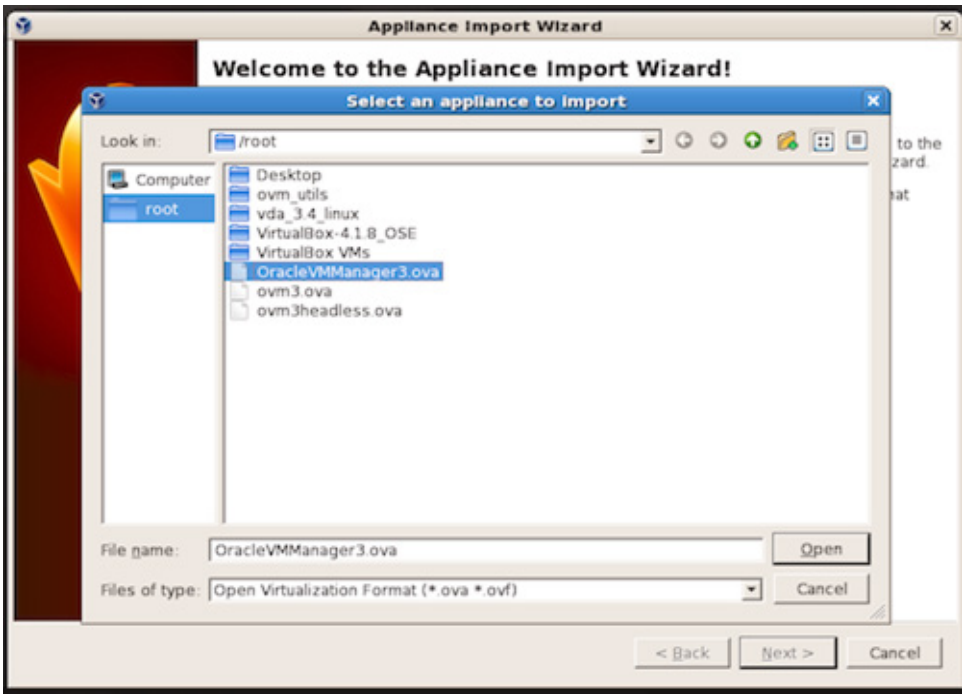


Figure 6.58. Appliance import interface

In the import appliance wizard, click on Choose, and locate the OracleVMManager3.ovafile that you downloaded.

Figure 6.59. The Oracle VM manager Appliance import wizard

Click on Next to move to the import settings screen.

If you want to change the default Name of the Virtual Machine that will be created, double-click onName and enter a different one. Click on Import to start the import process.

Figure 6.60. Appliance Import Setting

The first step of the import process is accepting the license for this template. Read the license and click on Agree or Disagree. If you Agree then the import will continue. Depending on the performance of your host system, the import can take a number of minutes to complete.

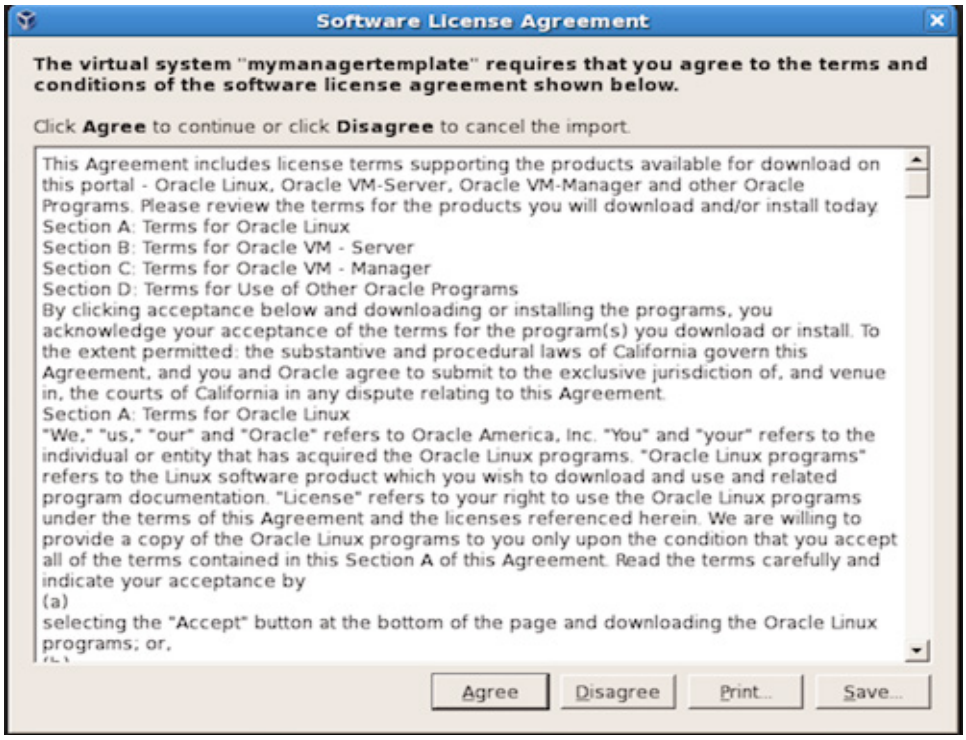


Figure 6.61. Lisence Agreement Interface

When the Import process finishes you will see the VM show up in the list on the left. In this example it is called "mymanagertemplate". Click on the template name and then on the Settings icon at the top to modify the VM settings if needed.

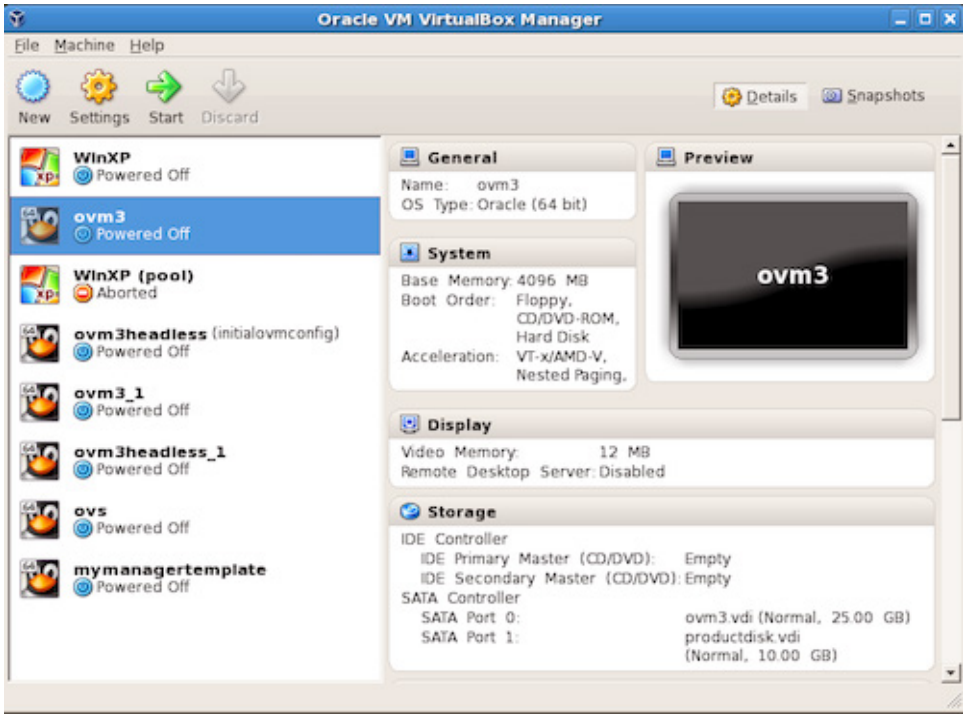


Figure 6.62. Your VM successfully created and displayed at the left pane

You can modify various settings but it is recommended to use the defaults, to avoid any difficulties running the pre-configured environment.

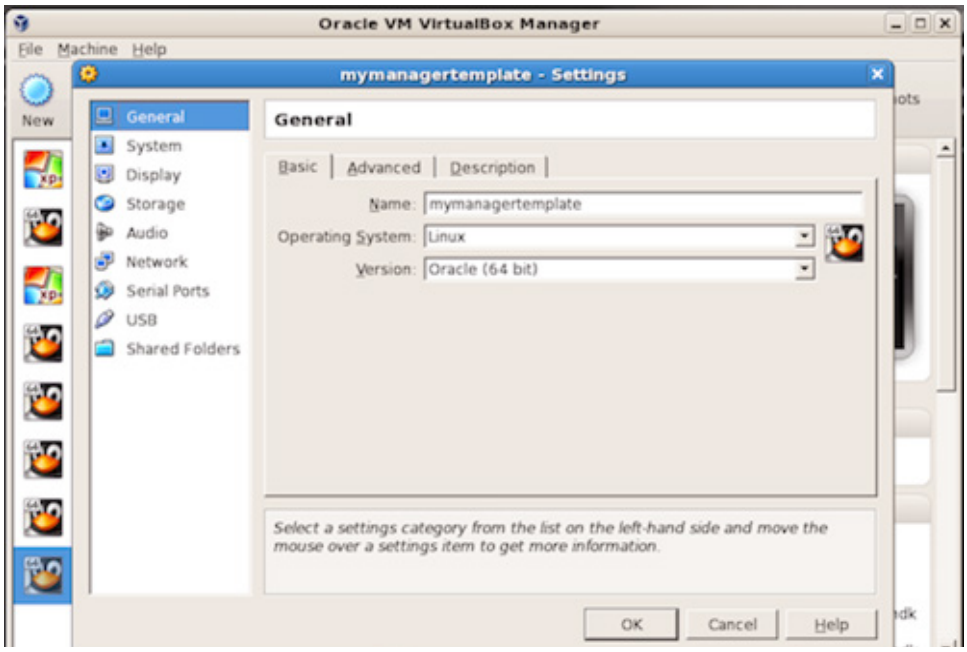


Figure 6.63. Managing the settings of your VM

Before you start, go to the Network settings and make sure that the correct host network is assigned to the Virtual Network Adapter. Click on the Name dropdown to find the networks that Oracle VM VirtualBox detected on your system. Pick the correct network from the list. When finished click OK.

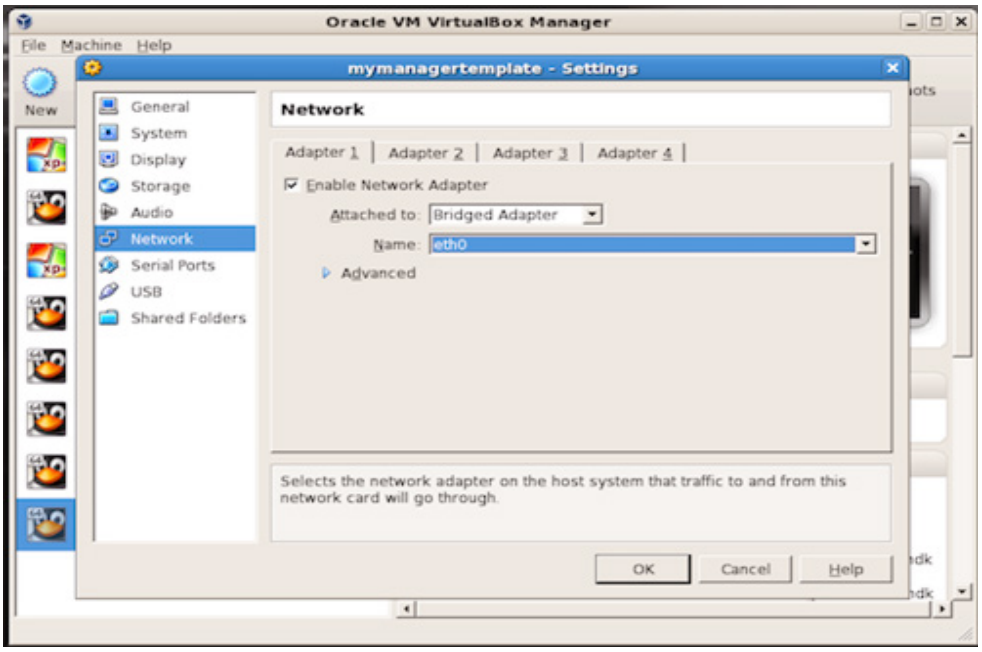


Figure 6.64. Assigning your network host to the Virtual network adapter

To start the Virtual Machine click on the Start Icon and the VM console will pop up showing the Virtual Machine booting into Oracle Linux. This process can take a moment.

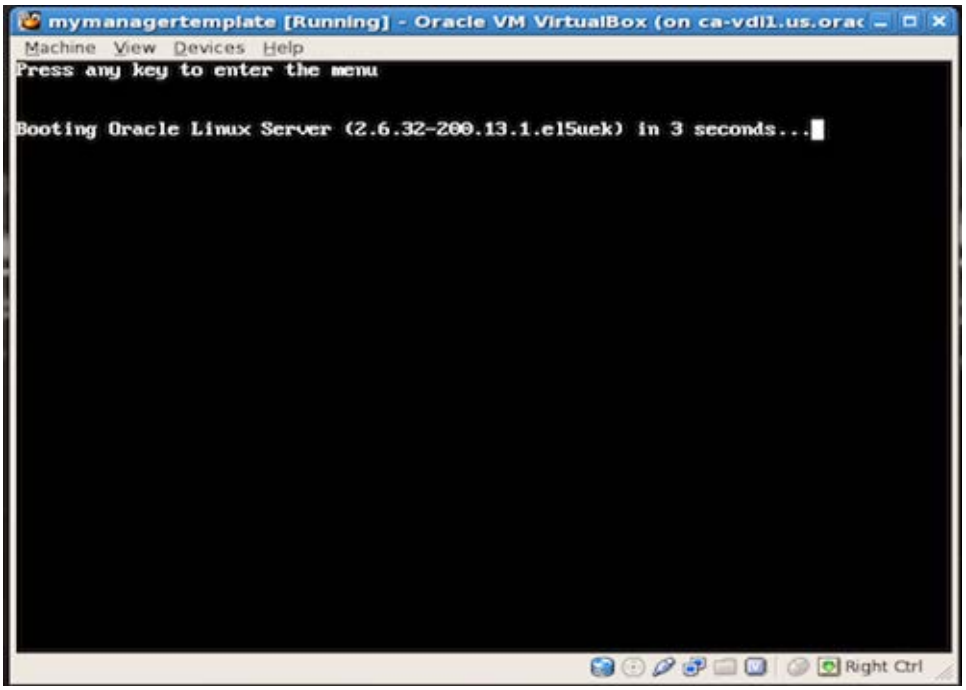


Figure 6.65. Your Virtual Machine booting into Oracle Linux

During the boot process, you will see various OS services start, including start-up of Oracle Database 11g Express Edition.

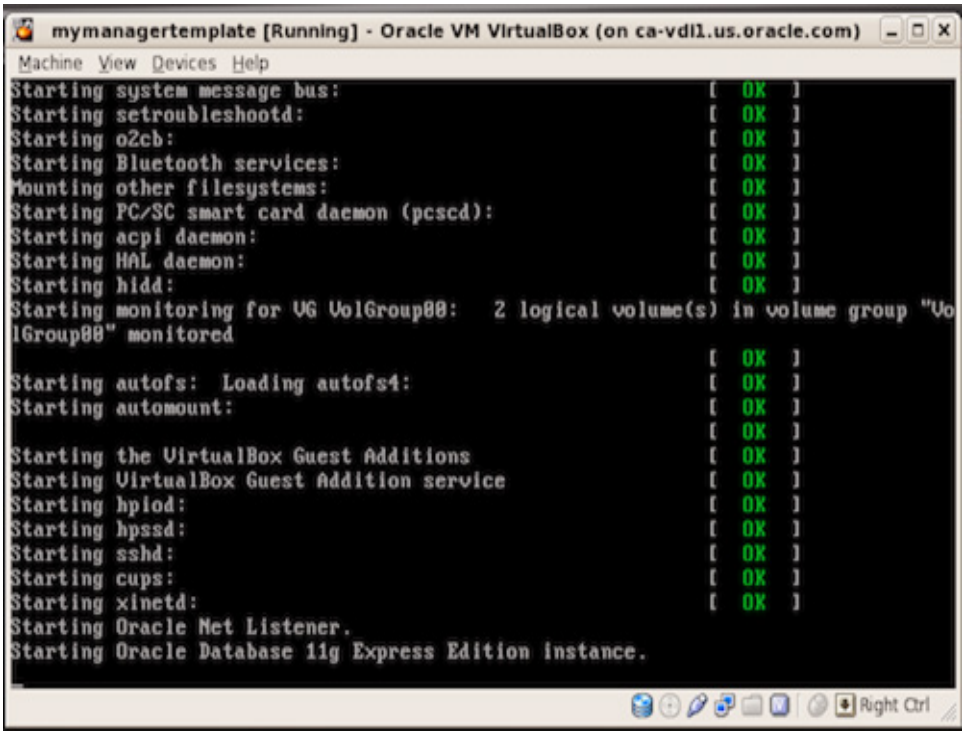


Figure 6.66. OS services being started by the VM

The next step is to provide a new password for the root (administrator) user of the Virtual Machine. Please enter and re-enter a new password when prompted to do so.

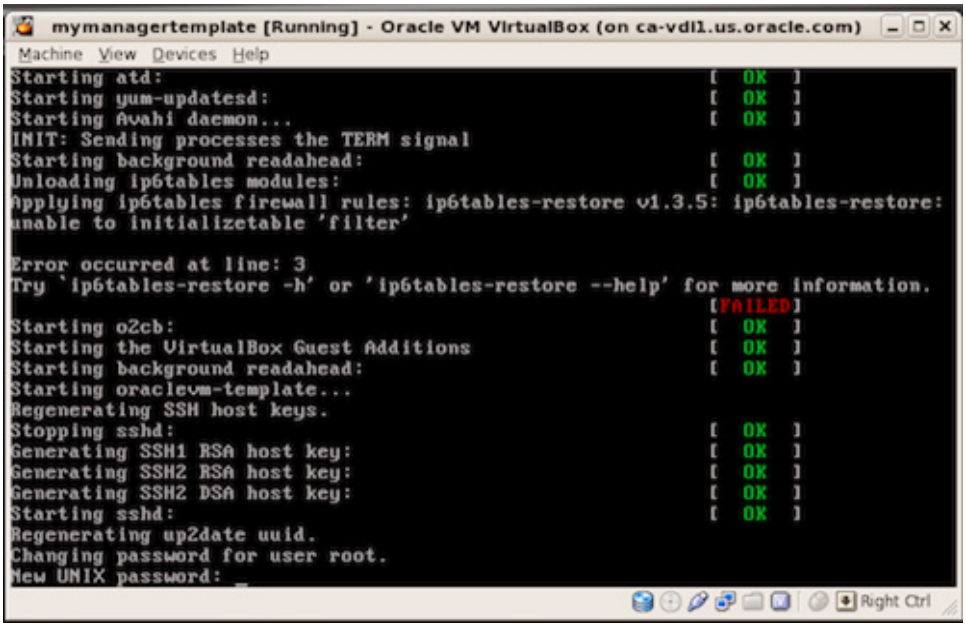


Figure 6.67. Changing the root password

After the root password change, enter the Network settings for this VM template. Please have the static IPaddress, netmask, gateway and DNSserver IP addresses handy during first boot. This information is required to finish the configuration and installation.

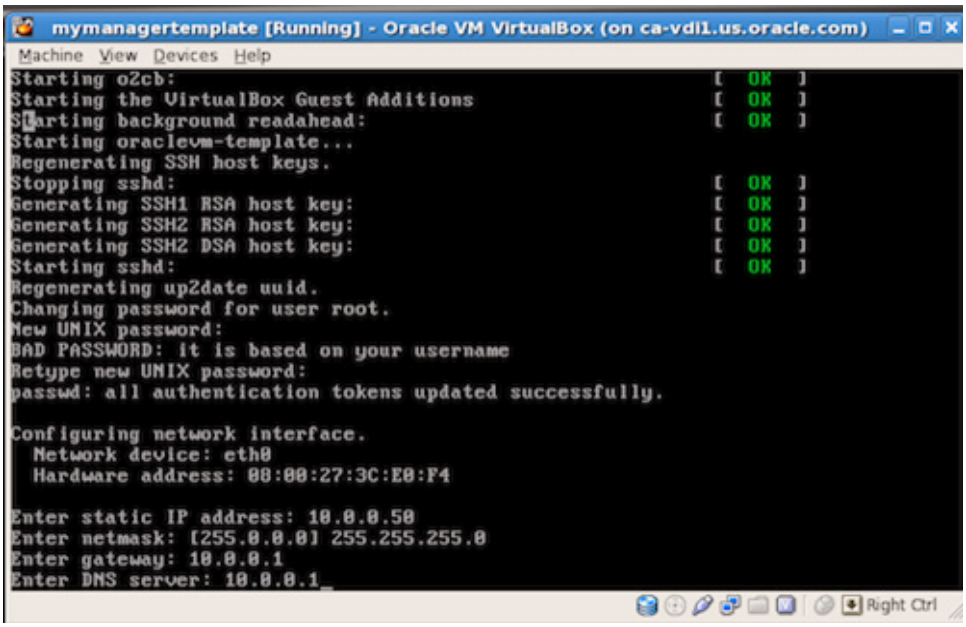


Figure 6.68. Setting up the network parameters for the VM

After the network has re-started, the system will try to determine the host name and domain name. If it fails, it will prompt for a new hostname to be entered. If it finds the hostname, confirm by just hitting Enter.

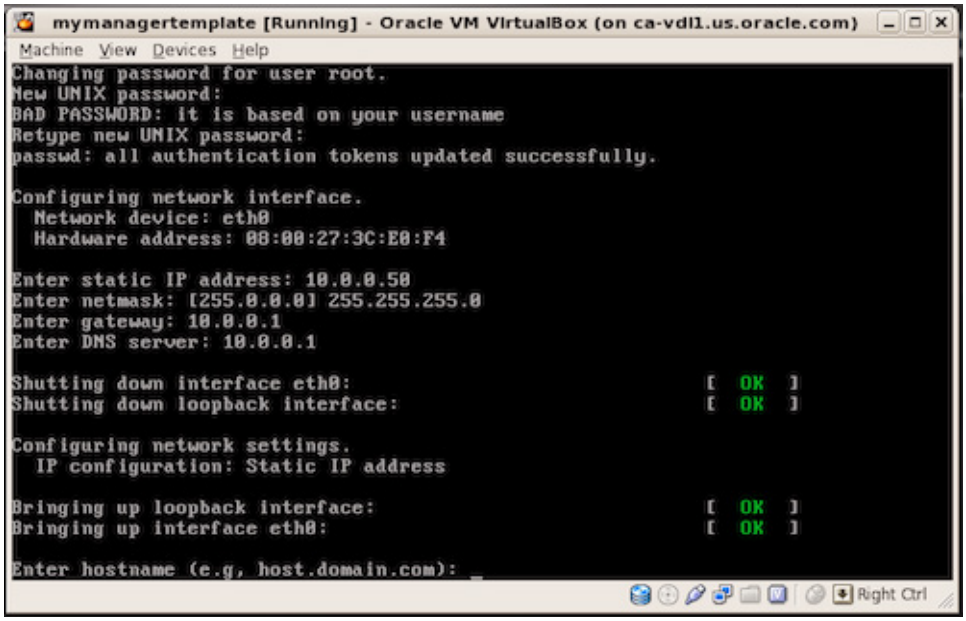


Figure 6.69. Prompting for a host name to be entered

Enter a key or wait 10 seconds until the VM configuration completes and the boot process finishes.

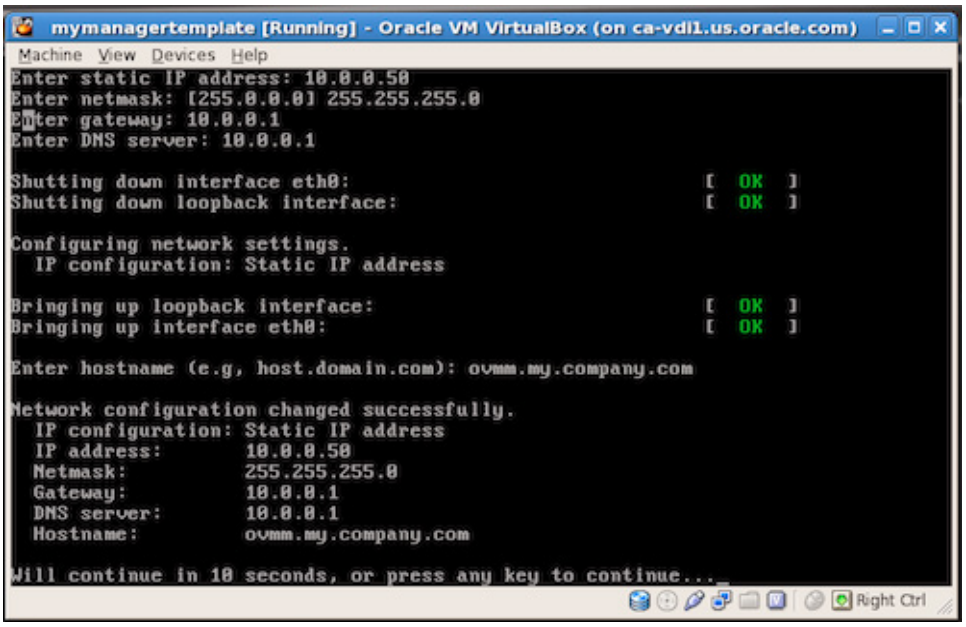


Figure 6.70. Finishing the Boot process

The boot process is complete when the template starts the Gnome desktop and automatically logs in as the user OVM. The desktop contains a number of links to provide easy access to get started. First double-click on the READ ME FIRST icon.

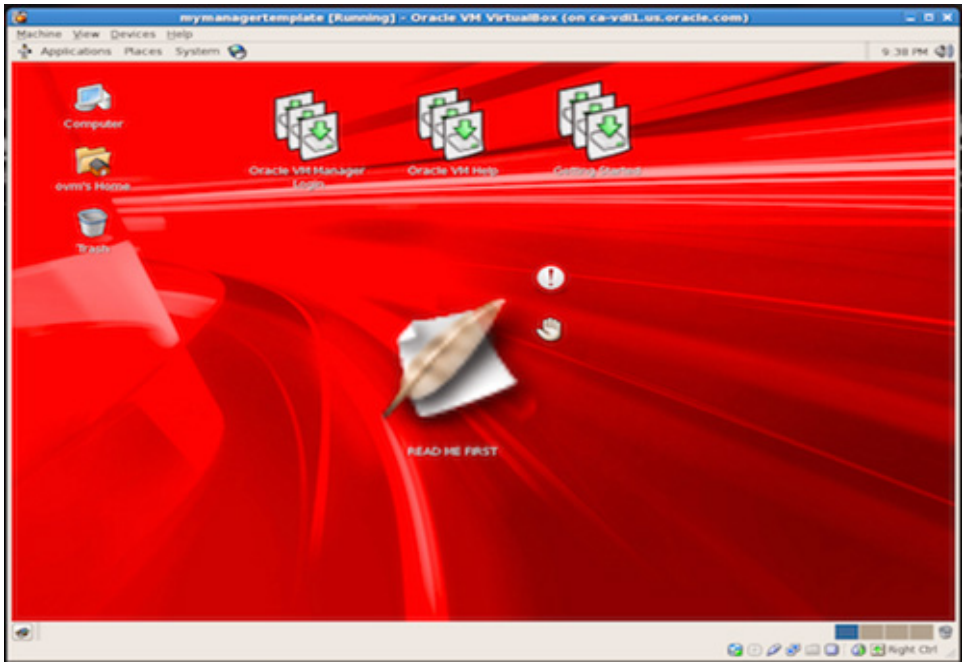


Figure 6.71. Virtual Machines desktop

This icon will start the web browser and display a local document that contains an overview of the content of the Oracle VM Manager template including the pre-created userIDs and passwords.

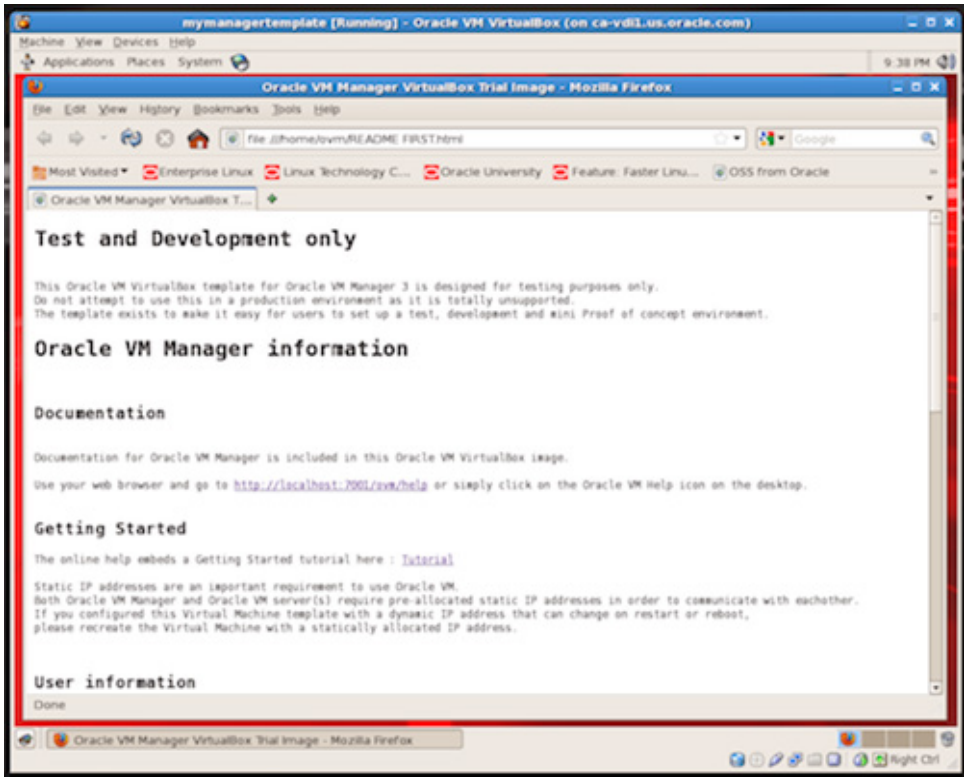


Figure 6.72. Content displayed when ReadMe First icon is clicked

To log into the Oracle VM Manager console, double click on the OracleVM Manager Login icon on the desktop. Enter username admin and default password Welcome1 to log into Oracle VM Manager.

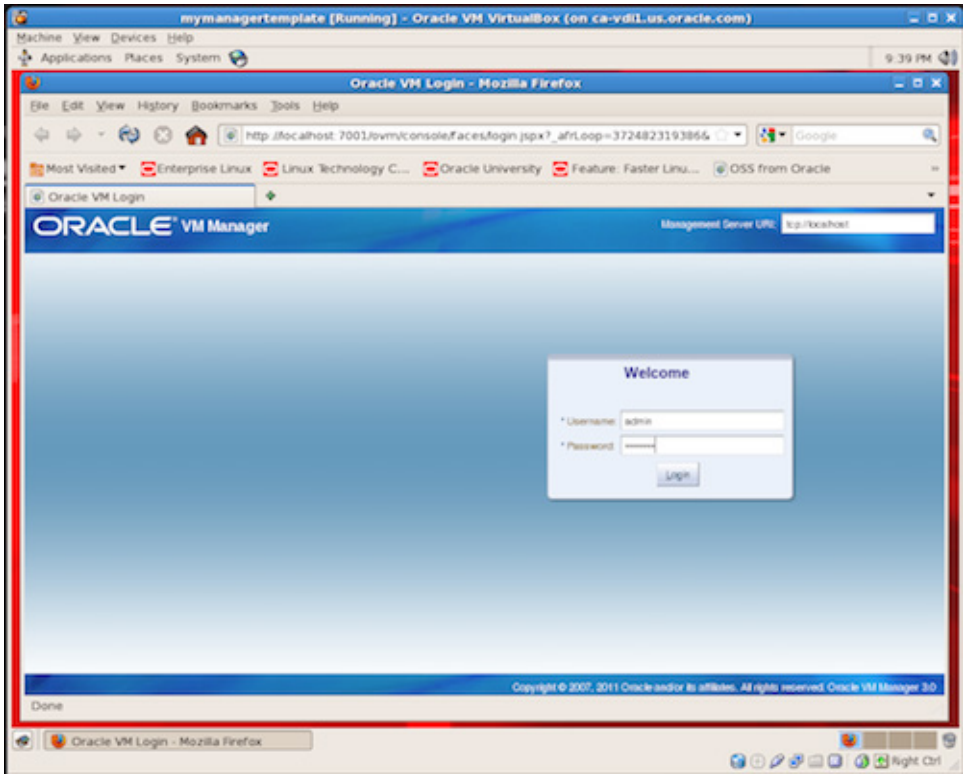


Figure 6.73. OracleVM Manager Login interface

Once logged into the application, you can start discovering Oracle VM servers and learn about the product.

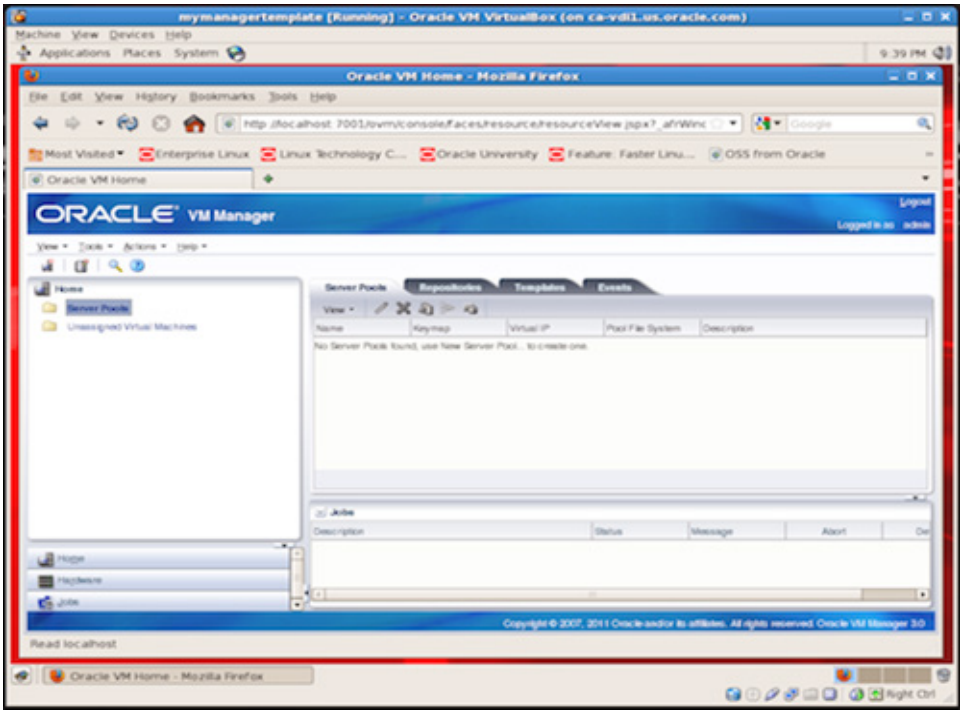


Figure 6.74. OracleVM Manager

The complete Oracle VM documentation is included in the template. Simply click on the Oracle VM Help or Getting Started icons on the desktop to read the documentation. We highly recommend you read the Getting Started manual prior to experimenting with Oracle VM.

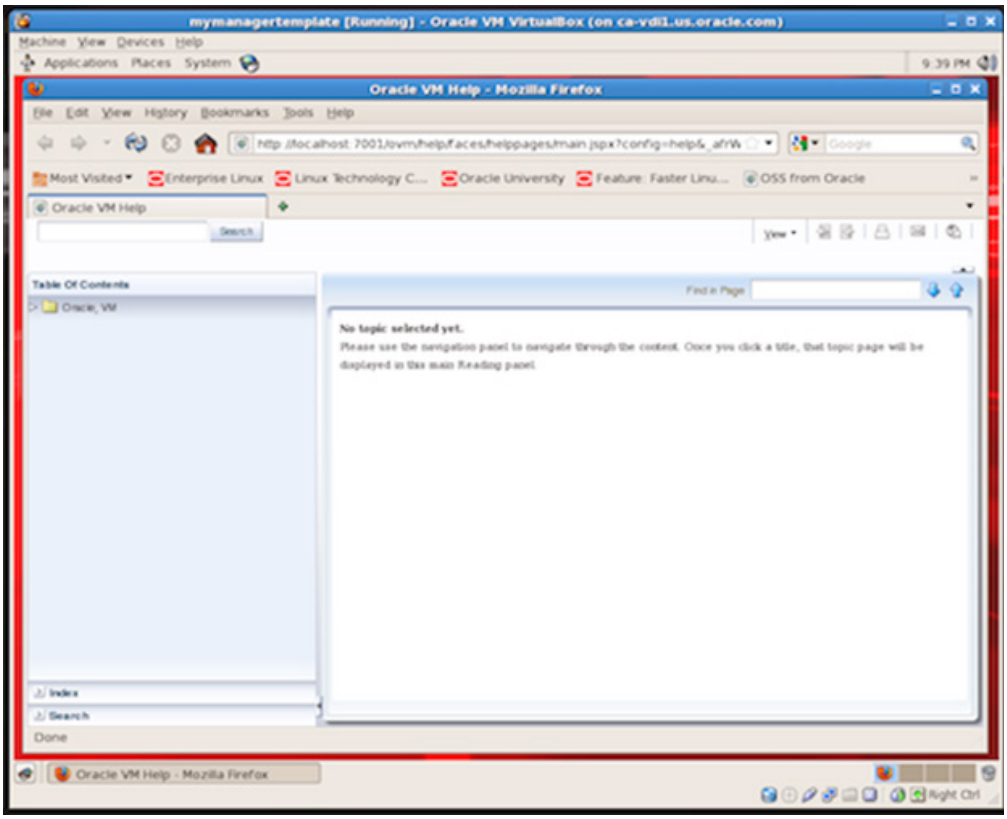


Figure 6.75. OracleVM Manager help menu

Results and Submission Requirments

The student has to perform the appropriate installation configuration of Oracle virtual machine.

Assessment Criteria

9Oracle virtual machine to obtain total of 3 marks.

References or Key Links

- https://docs.oracle.com/cd/E20065_01/doc.30/e18548/server.htmhttps://docs.oracle.com/cd/E20065_01/doc.30/e18548/server.htm
- https://docs.oracle.com/cd/E20065_01/doc.30/e18548/server.htm

Conclusion

Virtual Machines play a vital role in enhancing resource utilization of a system. In this activity, we discussed about virtual machines. What they are and how they work. The technologies used to implement virtual machines were addressed. Management of virtual machines were discussed using Oracle's VirtualBox.

Assessment

1 .Discuss the main difference between a physical and virtual machine

2. What advntages and disadvntages does a VM bring to a system

3. Install and configure a virtual machine using a VMWare player on both Windows and Linux environment

4. Use the Oracle VirtualBox VM managere and perform VM management tasks discussed

Unit Summary

In this unit, we tried to discuss special topics related with an operating system including basic OS Configuration and Maintenance issues, introduction to Real Time and Embedded OS as well as basics of a Virtual Machine.

A well configured and maintained OS is the basis for better functioning and resource management of a system. Before attempting to install an OS, one needs to understand the requirements imposed by that particular operating system and abide to those requirements. Once, installation of the operating system is completed, it is also possible to customize or personalize various features of the OS through configuration management tools.

This unit also presented high level discussion about real time and embedded systems focusing on the main differences between a general purpose system and real time ones along with the requirements of the real time systems. A real-time system is a time constrained system that needs arrival of results within a deadline period and those results arriving after the deadline are useless. Many of such systems are embedded in consumer and industrial devices. Real time systems can be either Soft or Hard systems based on the degree of time constrain they define. In order to achieve the timely execution of activities, real-time operating systems need to employ various techniques. The scheduler for a real-time operating system must differ from the general system scheduler, minimize both interrupt and dispatch latency, etc.

The last special topic presented in this unit talked about virtual machines and the various technologies that are used for VM creation as well as management. It also touched on the migration of physical to VM which is a process of defining a VM from an already existing resource of a system.

Unit Assessment

Check your understanding!

Essay Type Questions

*I*nstruction: Below are 5 essay type questions dealing with topics discussed in this unit. Attempt all questions

- How is a soft real time operating system different from hard real time operating system
- Explain the essential properties of real time operating systems?
- what is physical to VM migration? why is it needed? how is it achieved?
- Is it possible to run more than one VM on a single machine? If so, what are the pros and cons of having these multiple VMs on one machine?

Grading Scheme

This unit's assessments worth a total of 15% from which

a. Activity Assessments: 5%

b. Unit Assessment :10 %

c. Feedback

Rubrics for assessing activities

Assessment Criteria	Doesn't attempt all assessment or the answers are all wrong	Attempt all assessments and provide partial answers or partially correct answers	Attempt all assessments with full and correct answer
Grading Scales	0	Got half point for each assessment	Score full mark for each asssessment

Unit Readings and Other Resources

Required readings and other resources:

- A Silberschatz, Peter B Galvin, G Gagne (2009). Operating
- System Concepts. 8th Edition. Wiley. Chapter 19

Course Assessment

Final Exam

*I*nstructions: This Final exam consists two types of questions: Multiple choice questions and Workout questions. Attempt all questions accordingly.

Part I. 15 Multiple Choice questions are given below. Choose the best answer from the given choices and put the letter in the space provided

1._____ One of the following has a paramount importance in disk access performance

A. Seek time

B. Rotational delay

C. Access time

D. None of the above

2._____ The best disk arm scheduling algorithms is

A. FIFO

B. Elevator

C. Shortest Seek First

D. All of the above

3._____ Authentication of user cannot be done through

A. What the user has

B. What the user knows

C. Who the user is

D. None of the above

4.A malicious code written to give access to the programmer after the system is operational, without the knowledge of the system administrator is

A. Trojan horse

B. Trap door

C. Internet worm

D. Virus

E. All of the above

5.One is not the goal of I/O software

A. Buffering

B. Device independence

C. Uniform naming

D. Error handling

E. None of the above

6.Hardware device that maps physical address to logical address is

A. MMU

B. IRQ

C. DMA

D. Cache

E. Relocation Register

7.Address binding of instructions and data to memory address cannot be done at

A. Compile time

B. Execution time

C. Load time

D. Segmentation time

E. All

8.Which of the following disk arm scheduling algorithm results in starvation?

A. FIFO

B. Elevator

C. Shortest Seek First

D. None of the above

9.Which of the following is an overhead?

A. TSL

B. Context switch

C. Scheduler

D. Busy waiting

11.Pick the wrong statement

A. Logical and physical addresses are the same in compile-time address-binding schemes

B. Logical and physical addresses are the same in load-time address-binding schemes

C. logical and physical addresses differ in execution-time address-binding scheme

D. Logical and physical addresses differ in load-time address-binding schemes

E. All of the above

13.In terms of speed and storage utilization,

A. First fit algorithm is better

B. Best fit algorithm is better

C. Worst fit algorithm is better

D. All of the above

14.Functions of the device-independent I/O software does not include

A. Buffering

B. Device protection

C. Error reporting

D. Setup device registers

E. None of the above

15.One is not in the layers of I/O software layers

A. Interrupt handler

B. Device driver

C. Device independent OS software

D. User-level software

16.One is not a problem caused by concurrency of processes

A.Race condition

B. Starvation

C. Deadlock

D. Mutual exclusion

E. None

17. One of the following is not a solution for implementing mutual exclusion using busy waiting

A. TSL

B.trict alteration

C. Disabling interrupts

D. Semaphore

E. Lock variables

PART II: WORK OUT QUESTIONS. Read each question carefully and provide short and precise answers for each

1.Given the following request orders, Show order of disk arm movement and total number of tracks moved for the corresponding algorithms (3 pts.)

Order of request: 20, 75, 140, 60, 84

Current position: 55

A. FIFO

B. Elevator

Shortest Seek. First Assume that the size of the page frame in a virtual memory management system is 512 Bytes. Consider also the following page table:

5	3	1
4	5	1
3	4	1
2	X	0
1	0	1
0	2	1

Page Page Frame Present/Absent Bit

a. Give the real address corresponding to the virtual address 520, if possible. Explain how you obtained this result. (2 pts)

b. Give the real address corresponding to the virtual address 1030, if possible. Explain how you obtained this result. (2 pts)

2.Given five memory partitions of 100KB, 500KB, 200KB, 300KB, and 600KB (in order). How would each of the first-fit, best-fit, and worst-fit algoritouthms place processes of 212KB, 417KB, 112KB, and 426KB (in order)? Which algorithm makes the most efficient use of memory? (3 pts.)

Consider the following snapshot of a system:

Allocation	Max Required	Available
A B C D	A B C D	A B C D
1 5 2 0	0 0 1 2	0 0 1 2
P0		
P1	1 0 0 0	1 7 5 0
P2	1 3 5 4	2 3 5 6
P3	1 3 5 4	0 6 5 2
P4	0 0 1 4	0 6 5 6

3.Answer the following questions using the banker's algorithm:

a. What is the content of the Request matrix? (1 pt)

b. Is the system in a safe state? If yes, what should be the execution order of the processes (1 pt)

c. If a request from process P1 arrives for (0, 4, 2, 0), can the request be granted immediately? (1 pt)

4.Given the following jobs (processes) and their arrival time, Priority and Quantum = 3. Show the average waiting/response time for each of the following scheduling algorithms. (6 pts.)

a) FCFS

b) Shortest Job First(SJF)

c) Shortest Remaining First(SRT)

d) Round Robin

Process	Burst Time (CPU Time)	Arrival Time
P1	7	0
P2	5	2
P3	2	4

Grading Scheme

- Multiple choice part of the final exam worth 15 points each question worthing 1 point.
- Workout part of the final exam worth a total of 19 points

Feedback

Rubrics of the assessment

Assessment Criteria	Doesn't attempt all assessment or the answers are all wrong	Attempt all assessments and provide partial answers or partially correct answers	Attempt all assessments with full and correct answer
Grading Scales	0	Got half point for each assessment	Score full mark for each asssessment

Essay Type Questions

Instructions

Below are five essay type questions. Attempt all questions and write precise answers for all accordingly

1 Why is so important to have a file system?

2. Describe the goals and treats of computer security.

3. What are the two conditions why a process loses the CPU?

4. down() and up() operations of a semaphore variable should be atomic in order to have cooperating processes. Which condition(s) will be violated if these operations are not atomic? Explain how the condition(s) will be violated

5. What is the reason why the "Disabling Interrupts" technique of avoiding race condition is not considered as a good solution?

Grading Scheme

This assessment worth a total of 15 points each question weighing 3 points

Feedback

Assessment Criteria	Doesn't attempt all questions or the answers are all wrong	Attempt all questions and provide partial answers or partially correct answers	Attempt all questions with full and correct answer
Grading Scales	0	Earn 1.5 points for each answered question	Score full mark for each question

Course References

a. Tanenbaum, "Modern Operating System", second edition, 2001

b. Silberschatz, Galvin, Gagne, "Operating Systems Principles", 8th edition, 2009

c. William Stallings, Computer Organization and Architecture, 5/E, 6/E, Prentice Hall, 2003

The African Virtual University Headquarters

Cape Office Park

Ring Road Kilimani

PO Box 25405-00603

Nairobi, Kenya

Tel: +254 20 25283333

contact@avu.org

oer@avu.org

The African Virtual University Regional Office in Dakar

Université Virtuelle Africaine

Bureau Régional de l'Afrique de l'Ouest

Sicap Liberté VI Extension

Villa No.8 VDN

B.P. 50609 Dakar, Sénégal

Tel: +221 338670324

bureauregional@avu.org