SURESH GYAN VIHAR UNIVERSITY
Accredited by NAAC with 'A+' Grade

Bachelor of Computer Application (B.C.A.)

Introduction to Operating System

Semester-III

Author- Harshita V. Vachhani

SURESH GYAN VIHAR UNIVERSITY

Centre for Distance and Online Education
Mahal, Jagatpura, Jaipur-302025

Published by:

S. B. Prakashan Pvt. Ltd.

WZ-6, Lajwanti Garden, New Delhi: 110046

Tel.: (011) 28520627 | Ph.: 9205476295

Email: info@sbprakashan.com | Web.: www.sbprakashan.com

Designed & Graphic by : S. B. Prakashan Pvt. Ltd.

Printed at :

Syllabus

Introduction to Operating System

Learning Objective

- To explain main components of OS and their working
- To familiarize the operations performed by OS as a resource Manager
- To impart various scheduling policies of OS To teach the different memory management techniques.

UNIT I

OPERATING SYSTEMS OVERVIEW: Introduction, operating system operations, process management, memory management, storage management, protection and security, distributed systems. OPERATING SYSTEMS STRUCTURES: Operating system services and systems calls, system programs, operating system structure, operating systems generations.

UNIT II

PROCESS MANAGEMENT: Process concepts, process state, process control block, scheduling queues, process scheduling, multithreaded programming, threads in UNIX, comparison of UNIX and windows. CONCURRENCY AND SYNCHRONIZATION: Process synchronization, critical section problem, Peterson's solution, synchronization hardware, semaphores, classic problems of synchronization, readers and writers problem, dining philosophers problem, monitors, synchronization examples (Solaris), atomic transactions. Comparison of UNIX and windows.

UNIT - III

DEADLOCKS: System model, deadlock characterization, deadlock prevention, detection and avoidance, recovery from deadlock banker's algorithm. MEMORY MANAGEMENT: Swapping, contiguous memory allocation, paging, structure of the page table, segmentation, virtual memory, demand paging, page-replacement algorithms, allocation of frames, thrashing, case study - UNIX.

UNIT IV

FILE SYSTEM: Concept of a file, access methods, directory structure, file system mounting, file sharing, protection. File system implementation: file system structure, file system implementation, directory implementation, allocation methods, free-space management, efficiency and performance, comparison of UNIX and windows.

UNIT - V

I/O SYSTEM: Mass storage structure - overview of mass storage structure, disk structure, disk attachment, disk scheduling algorithms, swap space management, stable storage implementation, tertiary storage structure. I/O: Hardware, application I/O interface, kernel I/O subsystem, transforming I/O requests to hardware operations, streams, performance.

References

- Abraham Silberschatz, Peter Baer Galvin, Greg Gagne (2006), Operating System Principles, 7th edition, Wiley India Private Limited, New Delhi.
- Stallings (2006), Operating Systems, Internals and Design Principles, 5th edition, Pearson Education, India.
- Andrew S. Tanenbaum (2007), Modern Operating Systems, 2nd edition, Prentice Hall of India, India.
- Deitel & Deitel (2008), Operating systems, 3rd edition, Pearson Education, India.

Contents

* * *

Chapter 1

INTRODUCTION TO OPERATING SYSTEM

1. Introduction

An operating system is the program that is loaded into the computer and which co-ordinates all the activities among computer hardware devices. It is an interface between user and computer. An operating system makes everything in the computer to work smoothly and efficiently. It controls the hardware in the computer peripherals and manages memory and files. It enables the user to communicate with the computer and other software.

Examples of operating system include Microsoft Windows, Macintosh, Linux, Unix and DOS.

Definition

"An Operating System (OS) is a program that acts as an intermediary between the user of a computer and the computer hardware."

3

Oct.15,11, Apr.12– *2M*
Define Operating System.

1
Apr. 2013 – 4M
What is the purpose of an Operating System?

The purpose of operating system is to provide an environment in which a user can execute programs.

The primary goal of an operating system is thus to make the computer system convenient to use.

The secondary goal is to use the computer hardware in an efficient manner.

The components of a computer system are its hardware, software and data. The operating system provides the means for the proper use of these resources in the operation of the computer system.

An operating system is similar to a government. Like a government, the operating system performs no useful function by itself. It simply provides an environment within which other programs can do useful work.

We can view an operating system as a **resource allocator**. A computer system has many hardware and software resources that may be required to solve a problem. CPU time, memory space, file storage space, input-output devices and so on. The operating system acts as the manager of these resources and allocates them to specific program and users as necessary for tasks.

An operating system also acts as a **control program**. A control program controls the execution of user programs to prevent errors and improper use of the computer. It is especially concerned with the operation and control of input/output devices.

A more common definition of operating system is that it is the "one program running at all times on the computer (usually called the Kernel) with all else being application program".

Components of a Computer System

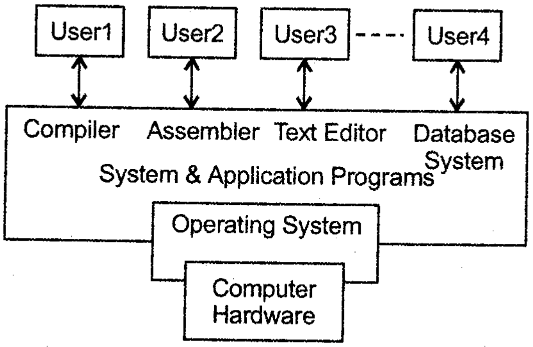


Figure 1.1: Abstract view of the components of a computer system

Main components of a computer system are:

i. Hardware
ii. Operating system
iii. Application programs
iv. Users

The *hardware* i.e., Central Processor Unit (CPU), the memory and the Input/Output (I/O) devices provides the basic computing resources for the system. The *applications program* such as word processors, spreadsheets, computers and the web browsers define the ways in which these resources are used to solve users computing problems. The *operating system* controls the hardware and coordinates its use among the various application programs for the various users.

2. Services Provided by OS

An operating system provides an environment for the execution of programs. The operating system provides certain services to programs and to the users of those programs.

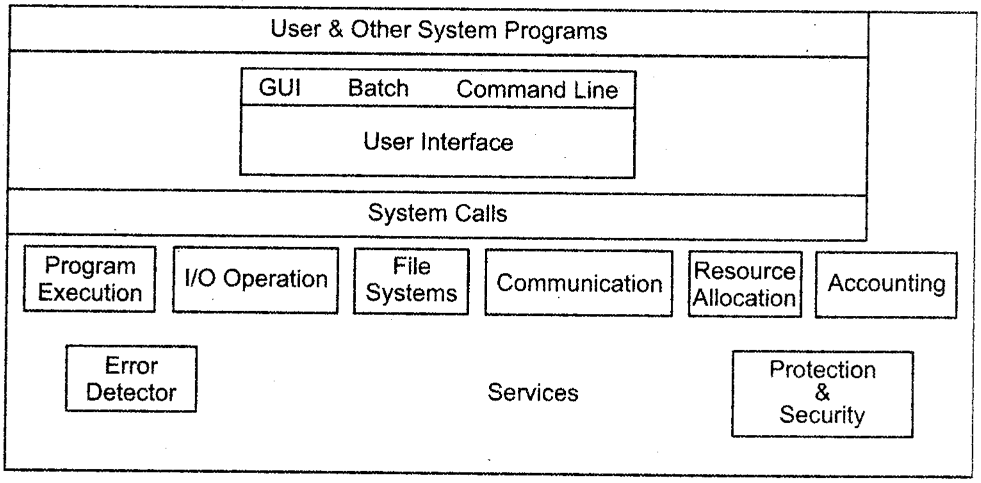


Figure 1.2: A view of operating system services

2

Oct. 2012 – *4M*
Explain different functions performed by an Operating System.
Apr. 2011 – *4M*
Explain basic services provided by an Operating System.

1. **User interface:** All operating systems have a User Interface (UI). This interface can take several forms. e.g., Command Line Interface (CLI), Batch Interface or Graphical user Interface (GUI).
2. **Program execution:** The system must be able to load a program into memory and to run that program.
3. **I/O (Input/Output) operations:** Running program may require I/O. For efficiency and protection users cannot control I/O devices directly. Therefore, the operating system must provide some means to do I/O.
4. **File system manipulation:** Operating system reads and writes into a file. It is the most visible service of an operating system.
5. **Communications:** Communications may be implemented via shared memory or by the technique of message passing. Communication which takes place between the concurrent processes can be divided in two parts:
   a. Take place between the processes that are running on the same computer and the other.
   b. Type of processes are those that are being executed on different computer systems through a computer network.
6. **Error detection:** System must be able to detect CPU or memory failure. There are various types of errors that occur when the process is running. These errors may be caused by CPU, memory hardwork, I/O devices, etc. The job of operating system to keep track of the errors, raise appropriate errors at the users screen.

   Example, memory error, power failure, lack of paper or printer etc.
7. **Resource allocation:** Where there are multiple users or multiple jobs running at the same time, resources must be allocated to each of them.
8. **Accounting:** The operating system keeps track of which users are using how much and what kinds of computer resources. This record keeping can be used for accounting or simply for accumulating usage statistics.
9. **Protection and security:** Protection involves ensuring that all access to the system resources is controlled.

   Security of the system starts with requiring each user to authenticate himself or herself to the system, usually by means of a password to gain access to the system resources.

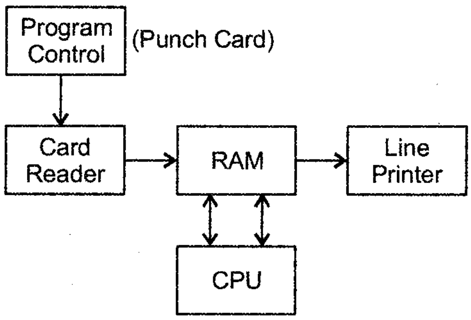
3. Types of Operating System

Various types of operating systems have evolved over time as computer systems and users' expectations of them have developed, i.e., as computing environments have evolved.

1
Oct. 2015 – *5M*
List the different types of operating system. Explain any one.

3.1 Simple Batch Operating System (Long Term Scheduler)

A batch operating system normally reads a stream of separate jobs (from a card reader) each with its own control cards that predefine what the job does. When the job is complete, its output is usually printed (on a line printer). The definitive feature of a batch system is the lack of interaction between the user and the job while that job is executing. The delay between job submission and job completion (called turnaround time) may result from the amount of computing needed or from delays before the operating system starts to process the job. In this execution environment, the CPU is often idle. This idleness occurs because the speeds of the mechanical I/O devices are intrinsically slower than those of electronic devices.



1
Oct. 2015 – *4M*
Explain in detail the long term scheduler.

Figure 1.3

The introduction of disk technology has helped in this regard. Rather than the cards being read from the card reader directly into memory and then the job being processed, cards are read directly from the card reader onto the disk. The location of card images is recorded in a table kept by the operating system. When a job is executed, the operating system satisfies its requests for card reader input by reading from the disk. Similarly, when the job requests the printer to O/P a line, that line is copied into a system buffer and is written to the disk. When the job is completed the O/P is actually printed. This form of processing is called **spooling.**

[S – Simultaneous P – Peripheral
O – Operation O – Online]

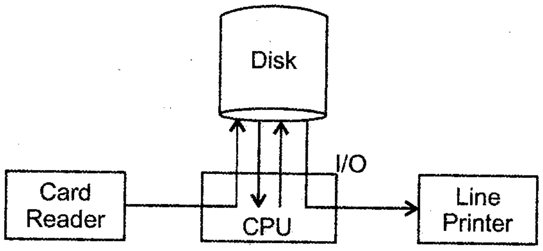


Figure 1.4

Spooling in essence, uses the disk as a huge buffer, for reading as far ahead as possible on input devices and for storing output files until the output devices are able to accept them.

Spooling is also used for processing data at remote sites. Spooling has direct beneficial effect on the performance of the system. For the cost of some disk space and a few tables, the computation of one job can overlap with the I/O of other jobs. Thus, spooling can keep both the CPU and I/O devices working at much higher rates.

3.2 Multiprogram Batch System

Spooling provides an important data structure: a job pool. A pool of jobs on disks allows the operating system to select which job to run next, to increase CPU utilization. Thus, job scheduling is possibly the most important aspect of job scheduling which is the ability to multiprogram.

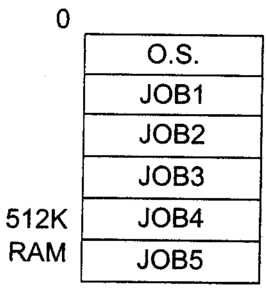


Figure 1.5: Memory layout for a multiprogram batch system

Multiprogramming is a feature of an operating system which allows running multiple program simultaneously on CPU. It is a form of parallel processing in which several programs run at the same time on a uniprocessor.

The idea of multiprogramming is as follows:

2

Oct. 12, Apr.11 – *2M*

What is Multiprogramming?

The operating system keeps several jobs in memory at a time as shown in *figure 1.5*. These set of jobs is a subset of jobs kept in job pool. The operating system picks and begins to execute one of the jobs in the memory eventually, the job may have to wait for some task (*example*, an I/O operation to compute).

In a non-multiprogram system, CPU would sit idle. In a multiprogram system the operating system simply switches to and executes another job. When that job needs to wait CPU will switch to another job and so on.

Eventually, first job finishes waiting and gets CPU back. As long as there is always some jobs to execute CPU will never be idle.

3.3 Time Sharing System (Middle Term Scheduler)

1

Apr. 2015 – *4M*

Explain medium term scheduler.

There are some difficulties with a batch system from the point of view of the user, however since, the user cannot interact with the job when it is executing the user must set up the control cards to handle all possible outcomes and it can be extremely difficult to define completely what to do in all cases.

Another difficulty is that program must be debugged statically, from **snapshot dumps.**

Time sharing or multitasking is a logical extension of multiprogramming. Multiple jobs are executed by the CPU switching between them but the switches occur so frequently that the users may interact with each program while it is running.

An interactive or hands-on computer system provides online communication between the user and the system.

Time sharing system are developed to provide interactive use of a computer system at a reasonable cost. A time shared operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time shared computer. Each user has atleast one separate program in memory.

A program that is loaded into memory and is executing is commonly referred to as a process.

When a process executes, it specially executes for only a short time before it either finishes or needs to perform interactive I/O. Rather than let the CPU sit idle when the interactive I/P takes place, the operating system will rapidly switch the CPU to the program of some other user.

A time shared operating system allows many users to share the computer simultaneously. Since each action/command in a time shared system tends to be short only a little CPU time is needed for each user. As the system switches rapidly from one user to the next, each user is given the impression that he has his own computer whereas actually one computer is being shared among many users.

3.4 Real Time Systems

Real time systems are used when there are rigid time requirements on the operation of a processor for the flow of data and thus is often used as a controlled device in a dedicated application. A real time operating system has well defined fixed time constraints. Processing must be done within the defined constrains, or the system will fail. A real time system is considered to function correctly only if it returns the correct result within any time constraints.

3.5 Clustered System

2

Apr.13, Oct. 12 – *2M*

What is meant by Multiprocessor System?

Multiprocessing system is similar to multiprogramming system, except that there is more than one CPU available.

Like multiprocessor system, clustered systems gather together multiple CPUs to accomplish computational work.

Clustering is usually used to provide high availability service. That is service will continue even if one or more systems in the cluster fail. High availability is generally obtained by adding a level of redundancy in the system. A layer of cluster software runs on the cluster nodes. Each node can monitor one or more of others. If the monitored machine fails, the monitoring machine can take ownership of its storage and restart the applications that were running on the failed machine. The user and clients of the applications can see only a brief interruption of service.

Clustering can be structured asymmetrically or symmetrically. In asymmetrically clustering, one machine is in not-standby mode while the other is running the application and in symmetric mode, two or more hosts are running applications and are monitoring each other.

A cluster consists of several computer systems connected via a network. Cluster may also be used to provide high performance computing environments.

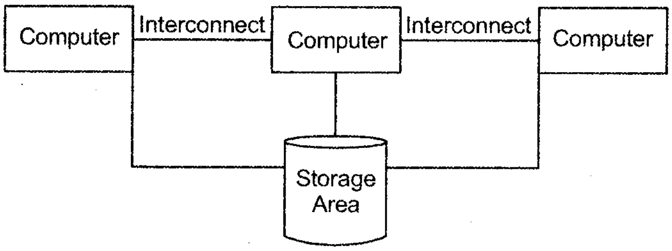


Figure 1.6: General structure of a clustered system

3.6 Distributed Operating Systems

A distributed system is a system consisting of two or more nodes, each node being a computer system with its own memory, some communication hardware and a capability to perform some control functions of an operating system.

1
Oct. 2014 – *2M*
What is Distributed System. List types of distributed system.

Distributed operating system provides the following advantages:

i. Resource sharing
ii. Reliability
iii Computation speed-up
iv. Communication
v. Incremental growth

SUMMARY

- Operating system is a program that acts as on intermediary between the user of a computer and the computer hardware.
- An operating system acts as a resource allocator.
- An operating system acts as an control program.
- Various services are provided by operating system like
  - a. User interface
  - b. Program execution
  - c. I/O operation
  - d. File system Manipulation
  - e. Communications
  - f. Error detection
- There are six types of operating system
  - a. Simple Batch operating system
  - b. Multiprogram operating system
  - c. Time sharing operating system
  - d. Real time operating system
  - e. Clustered operating system
  - f. Distributed operating system

PU Questions

2 Marks

1. Define Operating System **[Oct.15,11 Apr. 12 –*2M*]**
2. What is Distributed System? List types of distributed system. **[Oct. 2014 – *1M*]**
3. What is meant by Multiprocessor System? **[Apr. 13, Oct.11 – *2M*]**
4. What is Multiprogramming? **[Oct. 12,Apr.11 – *2M*]**

4 Marks

1. Explain in detail the long term scheduler. **[Oct. 2015 – *4M*]**
2. List the different types of operating system. Explain any one. **[Oct. 2015 – *4M*]**
3. Explain medium term scheduler. **[Apr. 2015 – *4M*]**
4. What is the purpose of an Operating System? **[Apr. 2013 – *4M*]**
5. Explain different functions performed by an Operating System. **[Oct. 2012 – *4M*]**
6. Explain basic services provided by an Operating System. **[Apr. 2011 – *4M*]**

Chapter 2 SYSTEM STRUCTURE

1. User Operating System Interface

There are two fundamental approaches for users to interface with the operating system. One technique is to provide a command-line interface or command interpreter that allows users to directly enter commands that are to be performed by the operating system. The second approach allows the user to interface with the operating system via a graphical user interface or GUI.

2

Oct.2015 – *2M*
What is command interpreter?
Apr.2015 – *2M*
What is the purpose of command interpreter.

Command Interpreter

Some operating systems include the command interpreter in the kernel. Others, such as Windows XP and UNIX, treat the command interpreter as a special program that is running when a job is initiated or when a user first logs on (an interactive systems).
On systems with multiple command interpreters to choose from, the interpreters are known as shells. *For example*, on UNIX and Linux systems, there are several different shells a user may choose from including the Bourne shell, C shell, Bourne-again shell, the Korn shell, etc. Most shells provide similar functionality with only minor differences; most users choose a shell based upon personal preference.

The main function of the command interpreter is to get and execute the next user-specified command. Many of the commands given at this level manipulate files: create, delete, list, print, copy, execute, and so on. The MS-DOS and UNIX shells operate in this way. There are two general ways in which these commands can be implemented.

In one approach, the command interpreter itself contains the code to execute the command. *For example,* a command to delete a file may cause the command interpreter to jump to a section of its code that sets up the parameters and makes the appropriate system call. In this case, the number of commands that can be given determines the size of the command interpreter, since each command requires its own implementing code.

An alternative approach used by UNIX, among other operating systems implements most commands through system programs. In this case, the command interpreter does not understand the command in any way; it merely uses the command to identify a file to be loaded into memory and executed.

Thus, the UNIX command to delete a file

```
rm file.txt
```

would search for a file called rm, load the file into memory, and execute it with the parameter file.txt. The function associated with the rm command would be defined completely by the code in the file rm. In this way, programmers can add new commands to the system easily by creating new files with the proper names. The command-interpreter program, which can be small, does not have to be changed for new commands to be added.

Graphical User Interfaces

A second strategy for interfacing with the operating system is through a user friendly graphical user interface or GUI. Rather than having users directly enter commands via a command-line interface, a GUI allows a mouse-based window-and-menu system as an interface. A GUI provides a desktop metaphor where the mouse is moved to position its pointer on images, oricons, on the screen (the desktop) that represent programs, files, directories, and system functions. Depending on the mouse pointer's location, clicking a button on the mouse can invoke a program, select a file or directory known as a folder or pull down a menu that contains commands.

Graphical user interfaces first appeared due in part to research taking place in the early 1970s at Xerox PARC research facility. The first GUI appeared on the Xerox Alto computer in 1973. However, graphical interfaces became more widespread with the advent of Apple Macintosh computers in the 1980s. The user interface to the Macintosh Operating System (Mac OS) has undergone various changes over the years, the most significant being the adoption of the Aqua interface that appeared with Mac OS X. Microsoft's first version of Windows version 1.0 was based upon a GUI interface to the MS-DOS operating system. The various versions of Windows systems preceeding this initial version have made cosmetic changes to the appearance of the GUI and several enhancements to its functionality, including the Windows Explorer Commercial versions of UNIX such as Solaris and IBM's AIX system. However there has been significant development in GUI designs from various open source projects such as K Desktop Environment (or KDE) and the GNOME desktop by the GNU project. Both the KDE and GNOME desktops run on Linux and various UNIX systems and are available under open-source licenses, which mean their source code is in the public domain. The choice of whether to use a command-line or GUI interface is mostly one of personal preference. As a very general rule, many UNIX users prefer a command-line interface as they often provide powerful shell interfaces.

Alternatively, most Windows users are pleased to use the Windows GUI environment and almost never use the MS-DOS shell interface.

The various changes undergone by the Macintosh operating systems provide a nice study in contrast. Historically, Mac OS has not provided a command line interface, always requiring its users to interface with the operating system using its GUI. However, with the release of Mac OS X (which is in part implemented using a UNIX kernel), the operating system now provides both a new Aqua interface and command-line interface as well.

The user interface can vary from system to system and even from user to user within a system. It typically is substantially removed from the actual system structure. The design of a useful and friendly user interface is therefore not a direct function of the operating system.

2. System Calls

2

Oct.2015 – *4M*

Define system call. Explain the system calls related to device manipulation.

Oct.2014 – *4M*

List and Explain system calls related to Device Management.

Systems calls provide the interface between a process and the OS. These calls are generally available in the assembly language instructions. Certain systems allow system calls to be made directly from a higher level language program in which the calls normally resemble predefined function.

The request and the release of resources are system calls.

For example, request, release of device, open, close of file, allocate and free memory.

System calls provides the interface between a process and the operating system. These calls are generally available as assembly language instructions.

Some systems may allow system calls to be made directly from a higher level language program, in this case the calls normally reassemble predefined functions or subroutine calls. They may generate a call to a special run-time routine that makes the system call.

System calls can be roughly grouped into following categories:

i. **Process or Job Control**

A running program needs to be able to halt execution either normally (end) or abnormally (abort).

Process Control

a. End abort
b. Load execute
c. Create process, terminate process
d. Allocate a file memory

ii. **File Management**

a. Create file, delete file
b. Open, close file
c. Read, write reposition file.

iii. **Device Management**

A process may need several resources to execute main memory, disk drives access to file, etc. If the resources are available they can be granted and control can be returned to the user process.

a. Request device, release device
b. Logically attach or detach devices

iv. **Information Maintenance**

Operating system keeps information about all its processes and system calls are used to access this information.

a. Get system data, set system data
b. Get/set time date.
c. Get/set process, file or device attributes.

v. **Communication**

There are two common models of interprocess communication: the message passing model and the shared memory model.

In the message passing model, the communicating processes exchange messages with one-another to transfer information.

In the shared memory model, processes use shared memory, create and shared memory attach system calls to create and gain access to regions of memory owned by other processes.

a. Create delete communication

b. Send/receive messages

1
Apr.2015 – *1M*
Define system program.

vi. System Program (SP)

System programs, also known as system utilities, provide a convenient environment for program development and execution. S.P. are divided into following categories:

a. *File Management:* These programs create, delete, copy, rename, print, dump, list the files.

b. *Status information:* These are used to get information about system e.g., the system program that can use to get date, time of system, amount of available memory, amount of free memory etc.

1
Apr.2012 – *4M*
List and explain different types of system program.

c. *File modification:* To create file and edit the content of files we can have text editor.

d. *Programming language support:* Compilers, assembler, debuggers and interpreter for common programming languages are provided.

e. *Program loading and execution:* System program like loaders, links are provided.

3. Architecture of Computer System

Computer architecture is the conceptual design and fundamental operational structure of a computer system. It is a blueprint and functional description of requirements (especially speeds and interconnections) and design implementations for the various parts of a computer focusing largely on the way by which the Central Processing Unit (CPU) performs internally and accesses addresses in memory. It may also be defined as the science and art of selecting and interconnecting hardware components to create computers that meet functional, performance and cost goals. "Architecture" therefore typically refers to the fixed internal structure of the CPU (i.e. electronic switches to represent logic gates) to perform logical operations, and may also include the built-in interface (i.e. opcodes) by which hardware resources (i.e. CPU, memory, and also motherboard, peripherals) may be used by the software. It is frequently confused with computer organization. But computer architecture is the abstract image of a computing system that is seen by a machine language (or assembly language) programmer, including the instruction set, memory address modes, processor registers, and address and data formats; whereas the computer organization is a lower level, more concrete, description of the system that involves how the constituent parts of the system are interconnected and how they interoperate in order to implement the architectural specification.

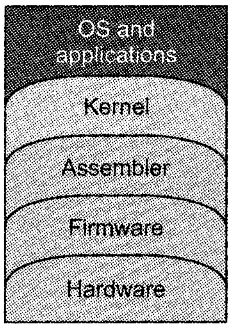


Figure 2.1

Figure 2.1 a typical vision of a computer architecture as a series of abstraction layers: hardware, firmware, assembler, kernel, operating system and applications

Abstraction Layer

An abstraction layer (or abstraction level) is a way of hiding the implementation details of a particular set of functionality. Perhaps the most well known software models which use layers of abstraction are the OSI 7 Layer model for computer protocols, OpenGL graphics drawing library, and the byte stream I/O model originated by Unix and adopted by MSDOS, Linux, and most other modern operating systems. In computer science, an abstraction level is a generalization of a model or algorithm, away from any specific implementation. These generalizations arise from broad similarities that are best encapsulated by models that express similarities present in various specific implementations. The simplification provided by a good abstraction layer allows for easy reuse by distilling a useful concept or metaphor so that situations where it may be accurately applied can be quickly recognized. A good abstraction will generalize that which can be made abstract; while allowing specificity where the abstraction breaks down and its successful application requires customization to each unique requirement or problem.

Firmware

In computing, firmware is software that is embedded in a hardware device. It is often provided on flash ROMs or as a binary image file that can be uploaded onto existing hardware by a user.

Firmware is defined as:

i. The computer program in a Read-Only Memory (ROM) integrated circuit (a hardware part number or other configuration identifier is usually used to represent the software);

ii. The erasable programmable Read-Only Memory (EPROM) chip, whose program may be modified by special external hardware, but not by [a general purpose] application program.

iii. The electrically Erasable Programmable Read-Only Memory (EEPROM) chip, whose program may be modified by special electrical external hardware (not the usual optical light), but not by [a general purpose] application program.

Assembler

An assembly language program is translated into the target computer's machine code by a utility program called an assembler. Typically a modern assembler creates object code by translating assembly instruction mnemonics into opcodes, and by resolving symbolic names for memory locations and other entities. The use of symbolic references is a key feature of assemblers, saving tedious calculations and manual address updates after program modifications.

Kernel

In computing, the kernel is the central component of most computer operating systems (OSs). Its responsibilities include managing the system's resources and the communication between hardware and software components. As a basic component of an operating system, a kernel provides the lowest-level abstraction layer for the resources (especially memory, processor and I/O devices) that applications must control to perform their function. It typically makes these facilities available to application processes through inter-process communication mechanisms and system calls. These tasks are done differently by different kernels, depending on their design and implementation. While monolithic kernels will try to achieve these goals by executing all the code in the same address space to increase the performance of the system, micro kernels run most of their services in user space, aiming to improve maintainability and 9 modularity of the code base. A range of possibilities exists between these two extremes.

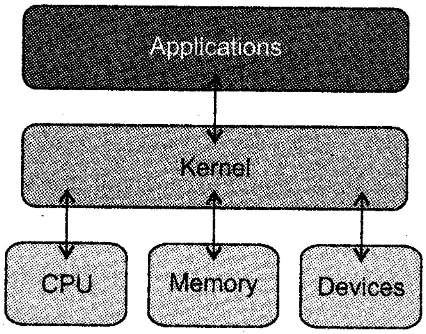


Figure 2.2

Figure 2.2 shows a kernel connects the application software to the hardware of a computer.

3.1 Operating System

An operating system (OS) is a computer program that manages the hardware and software resources of a computer. At the foundation of all system software, an operating system performs basic tasks such as controlling and allocating memory, prioritizing system requests, controlling input and output devices, facilitating networking, and managing files. It also may provide a graphical user interface for higher level functions. It forms a platform for other software.

Application Software

Application software is a subclass of computer software that employs the capabilities of a computer directly to a task that the user wishes to perform. This should be contrasted with system software which is involved in integrating a computer's various capabilities, but typically does not directly apply them in the performance of tasks that benefit the user. In this context the term application refers to both the application software and its implementation

4. Operating System Structure

An operating system provides the environment within which programs are executed. One can view an operating system from various angles. One is by examining the services it provides. Another is by looking at the interface it makes available to users and programmers. Third is by disassembling the system components and their interconnections. Whenever we want to build a large system, we typically break it into smaller components.
Operating system is divided in two parts:

i. **Kernel:** This is the innermost layer of operating system close to the hardware and it controls the actual hardware. It is the heart of the operating system. It consists of routines, which are required very often and almost all the time.

ii. **Special routines:** These are loaded from the disk to the memory as and when required. It saves the usage of memory but extra time is required for loading and unloading the routines.

1
Oct.2011 – *4M*
Describe the Structure of Operating System with the help of a suitable diagram.

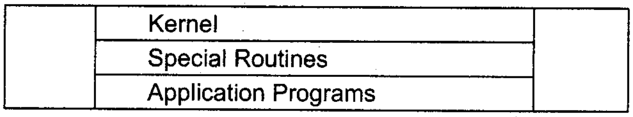


Figure 2.3: Structure of operating system

The operating system structure is a broad framework that unifies many features and services provided by the operating system.

Operating systems are broadly classified into following categories:

i. Simple structure
ii. Layered structure
iii. Microkernel structure
iv. Monolithic structure

4.1 Simple Structure

Some commercial operating systems do not have well-defined structure. Such operating systems are initially designed as small and simple. MS-DOS is an *example* of such operating system. When it was designed and implemented originally by few people, they had no idea about the popularity of this system.

It was designed to provide number of functions using very less memory space. So it was not properly divided into modules. Also, the interfaces and functionality levels are not well separated.

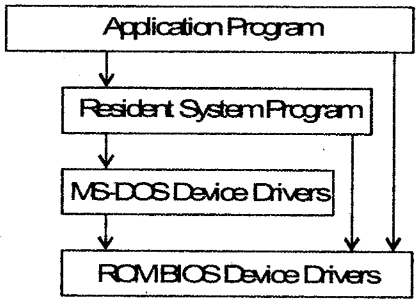


Figure 2.4: MS-DOS layer structure

As shown in *figure 2.4*, the application programs can directly access the device drivers and resident system programs, due to which MS-DOS is vulnerable to errant programs and crashes the entire system. MS-DOS was designed for Intel 8088, and was unable to provide dual mode and hardware protection.

Another example of limited structuring is the original UNIX operating system. UNIX operating system was initially limited by hardware functionality.

4.2 Layered Structure

The components of layered operating system are organized into modules. These modules are layered one on top of the other, i.e., a top down approach is provided. A set of functions is provided by each module that is called by other modules. The layered operating system structure with hierarchical organization of modules is shown in *figure 2.5*.

System verification and debugging is simplified due to such approach. The first layer uses only the basic hardware to implement its functions, so it is easy to debug the first layer without any concern for the rest of the system. The second layer can be debugged only after debugging the first layer and assuming the correct functioning of the first layer and so on.

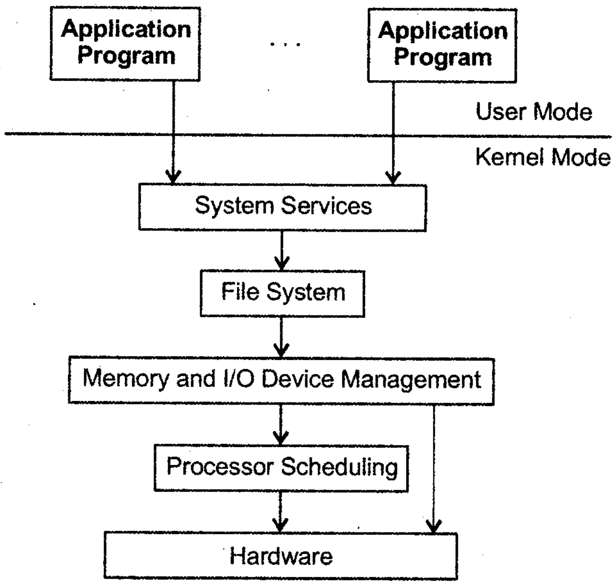


Figure 2.5: Layered operating system

If any error is found during the debugging of a particular layer, the error must be on the layer, because the lower layers are already debugged. That is in this approach, the Nth layer can access services provided by the (N-1)th layer and provide services to the (N+1)th layer. Thus, the operating system is debugged starting from the lowest layer, adding one layer at a time until the whole system works correctly.

The operating system can be enhanced easily by using the layered structure; one entire layer can be replaced without affecting other parts of the system. Layered operating system gives low application performance in comparison to monolithic operating system.

*Example*s of layered operating systems are Multics and UNIX.

4.3 Microkernel Structure

Microkernel does not mean small system. The word 'micro' means the kernel providing minimum functions that allow user-level system processes to perform operating services efficiently. The microkernel implements essential core-operating system functions. The functions include process management, inter-process communication, address space management, and hardware abstraction.

A microkernel is a tiny operating system core to be used in next-generation operating system as it provides foundation for modular and portable extensions of operating systems. A microkernel operating system provides unprecedented modularity, flexibility, and portability.

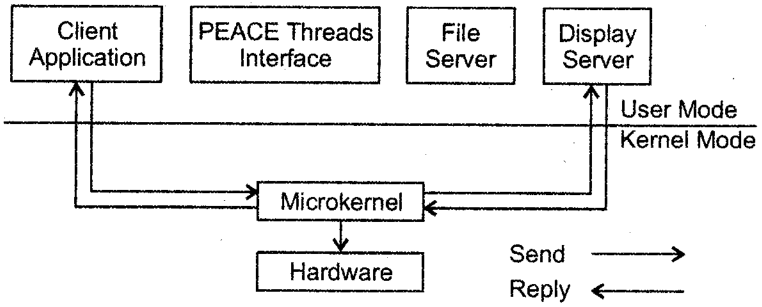


Figure 2.6: Microkernel operating system

In microkernel structure, the operating system is divided into several processes, each of which implements a single set of services, *example*, I/O servers, memory server, process server, threads interface system. Each server runs in user mode, and it provides services to the requested client. The client can be either another operating system or application program. The client requests a service by sending a message to the server. This communication takes place by using message passing method. The microkernel running in kernel mode delivers the message to the appropriate server. The server then performs the operation and microkernel passes the results to the client in another message, as shown in *figure 2.6*. Components above the microkernel communicate directly with one another by passing messages through microkernel itself. The microkernel controls the traffic. It validates messages and passes them between the components and grants access to hardware.

Since most of the services are running as a user processes rather than kernel processes, the microkernel structure provides more security and reliability.

4.4 Monolithic Structure

A Monolithic kernel is one single large program, composed of several logically different program pieces. The components of monolithic operating system are organized haphazardly and any module can call any other module without any reservation. Similar to the other operating systems, applications in monolithic OS are separated from the operating system itself, i.e., operating system code runs in a privileged processor mode (kernel mode), which has access to system data and to the hardware; applications run in a non-privileged processor mode (user mode), with a limited set of interfaces available and with limited access to system data. The monolithic operating structure with separate user and kernel processor mode is shown in *figure 2.7.*

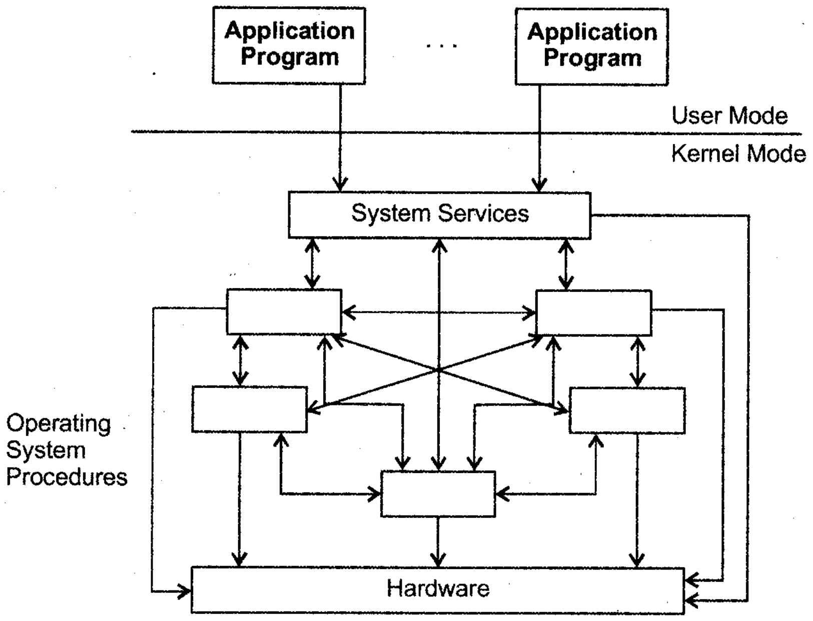


Figure 2.7: Monolithic operating system

When a user mode program calls a system service, the processor traps the call and then switches the calling thread to kernel mode. Completion of system service, switches the thread back to the user mode, by operating system allows the caller to continue.

The monolithic structure does not enforce data hiding in the operating system. It delivers better application performance, but extending such a system can be difficult work because modifying a procedure can introduce bugs in seemingly unrelated parts of the system. *Example* of monolithic OS is MS-DOS.

SUMMARY

- An operating system provides the environment within which programs are executed.
- The design of a new operating system is a major task.
- System calls provide the interface between a process and the operating system.
- An operating system structure is a broad framework that unifies many features and services provided by the operating system.

PU Questions

2 Marks

[Oct.2015 – *2M*]	1.	What is command interpreter?
[Apr.2015 – *2M*]	2.	What is the purpose of command interpreter?
[Apr.2015 – *2M*]	3.	Define system program.
[Oct.2014 – *2M*]	4.	List system calls related to communication.

4 Marks

[Oct.2015 – *4M*]	1.	Define system call. Explain the system calls related to device manipulation.
[Oct.2014 – *4M*]	2.	List and Explain system calls related to Device Management.
[Oct.2014 – *4M*]	3.	Explain architecture of Computer System.
[Oct.2011 – *4M*]	4.	Describe the Structure of Operating System with the help of a suitable diagram.
[Apr.2012 – *4M*]	5.	List and explain different types of system programs.

VISION

Chapter 3
PROCESS MANAGEMENT

1. Process Concept

2

Apr. 15, Oct.12 – *4M*
What is meant by Process?

The concept of process is the heart of operating systems.

Informally, a process is a program in execution.

A process is more than the program code (sometimes known as the text section or code segment). It also includes the current activity as represented by the value of program counter and the contents of the processor register.

A process generally includes the process stack, containing temporary data and a data section containing global variables.

Typically, a batch job is a process and a time shared user program is a process, a system task such as spooling is also a process.

Process execution is a cycle of CPU execution and I/O wait. Processes alternate back and forth between these two states. Process execution begins with a CPU burst.

It is followed by I/O burst which is followed by another CPU burst then another I/O burst and so on. Eventually the last CPU burst will end with a system request to terminate execution.

Functions of Process

1
Apr.2015 – 4M
In normal mode of operation. List and explain the sequence of utilization of resources by process.

i. Creating and removing processes.

ii. Controlling the progress of processes that is, ensuring that each logically enabled processes make progress towards its completion at a positive rate.

iii. Allocating hardware resources among processes.

iv. Providing a means of communicating messages or signals among processes.

v. Acting on exceptional conditions arising during the execution of a process, including interrupts and arithmetic errors.

Once created, a process becomes active and eligible to compete for system resources such as processor and I/O devices. Each active process is an individually schedulable entity.

A process is a dynamic concept that refer to a program in execution which undergoes frequent state and attribute changes.

2. Process State

As a process executes, it changes state. The state of a process is defined in part by the current activity of the process. Each process may be in one of the following states.

1
Oct.2012 – 4M
Explain in detail the various process states with the help of diagram.

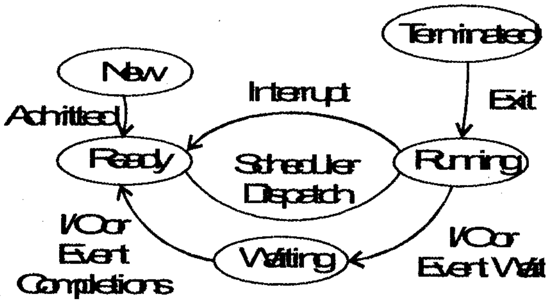


Figure 3.1: Process state diagram

i. **New:** The process is being created.
ii. **Running:** Instructions are being executed.
iii. **Waiting:** The process is waiting for some event to occur.
iv. **Ready:** The process is waiting to be assigned to a processor.
v. **Terminated:** The process has finished execution.

The state of a process is defined in part by the current activity of that process.

Process execution is an alternating sequence of CPU and I/O bursts beginning and ending with the CPU burst.

The process states can be further refined. Since the CPU may be shared among several process, an active process may either be waiting for the CPU or executing on it. A process which is waiting for the CPU is ready. A process, which has been allocated the CPU is running.

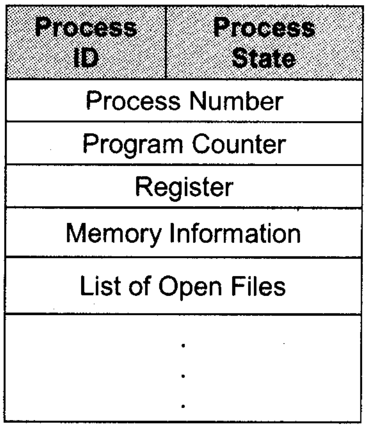
Whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed. The part of the operating system which carries out this selection process is called as **short–term scheduler** or **CPU scheduler**. The scheduler selects from among the process in memory that are ready to execute and allocates the CPU to one of them. The algorithm used by CPU scheduler is called as CPU scheduling algorithm.

3. Process Control Block

1
Oct.2015 – ***4M***
Explain in detail the process control block.

The operating system groups all information that it needs about a particular process into a data structure called a **Process Descriptor or Process Control Block (PCB).**

Whenever, a process is created the operating system creates a corresponding process control block to serve as its run-time description during lifetime of the process. When process terminates, its PCB is released to pool of free cells.



3
Apr.2013 – ***4M***
Explain PCB with proper diagram.
Apr.11, Oct.11 – ***4M***
Explain Process Control Block (PCB) in detail with the help of diagram.

1. **Process state:** It may be ready, running, waiting, halted and so on.
2. **Program counter:** The counter indicates address of next instruction to be executed for this process.
3. **CPU registers:** Along with program counter this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterwards.
4. **CPU scheduling information:** This information includes process, pictures pointer to scheduler queue and other scheduling parameters.
5. **Memory management information:** This information include value of base and limit registers, page tables or segment tables.
6. **Accounting information:** This information includes the amount of CPU and real-time used, time limits, account numbers, job or process number and so on.
7. **I/O states information:** This information includes list of the I/O devices allocated to this process, a list of open files and so on.

4. Context Switch

4

Oct.15,14,11,Apr.12–*2M*
What is meant by Context Switch?

Switching the CPU to another process requires saving the state of the old process and loading the saved state for the new process. This task is known as **Context switch.**

The context of a process is represented in the PCB of a process, it includes the value of the CPU registers, the process state and memory management information. When a context switch occurs, the kernel (operating system) saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run.

Context switch time is pure overhead, because the system does no useful work while switching. Its speed varies from machine to machine depending on the memory speed, the number of registers that must be copied and the existence of special instructions.

5. Operations on Processes

1

Apr.2015 – *4M*
Explain operations on process in detail.

The processes in the system can execute concurrently and they must be created and deleted dynamically. Thus, the operating system must provide a mechanism for process creation and termination.

5.1 Process Creation

A process may create several new process via a create process system call during its course of execution. The creating process is called a parent process whereas the new processes are called as the child of that process. Each of these processes may in turn create other process forming a tree of processes.

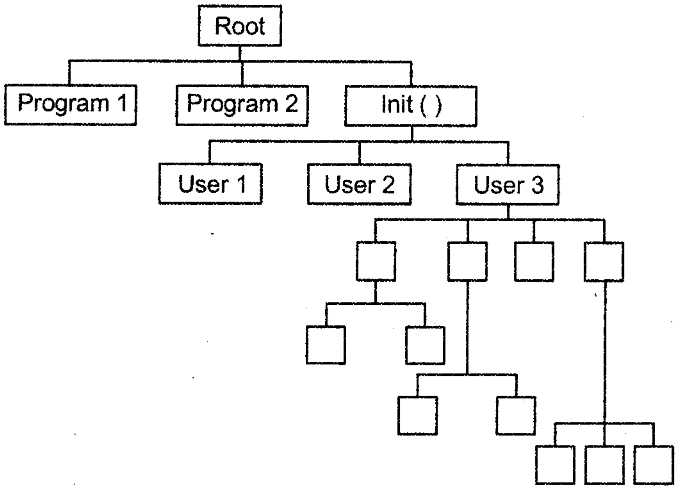


Figure 3.2: Tree of process

1
Apr.2012 – *4M*
Explain the Creation and Termination of Processes.

A process during execution needs various resources like CPU time, memory files, I/O devices etc. When a process creates sub-processes the sub-process will also require some of the resources. The sub-process may be able to obtain its resources directly from the operating system or it may be constrained to a subset of the resources of the parent process.

The parent may have to partition its resources among its children or it may be able to share some resources among several of its children.

Restricting a child process to a subset of the parents resources prevents any process from overloading the system by creating too many sub-processes.

When a process creates a new process, following two possibilities exists in terms of execution:

i. The parent continues to execute concurrently with its children.

ii. The parent waits until some or all of its children have terminated.

There are also two possibilities in terms of the address space of the new process (child process)

a. The child process (new process) is a duplicate of the parent process.

b. The child process has a program loaded into it.

Following are the functions/tasks performed by the operating system (Kernel) when a process is created.

i. It allocates a slot in the process table for new process.

ii. It assigns unique ID number to the child process.

iii. It makes a logical copy of the context of the parent process since, certain portions of a process, such as a text region, may be shared between processes the kernel can sometimes increment a region to a new physical location in memory.

iv. It increments file and inode table counters for files associated with the process.

v. It returns ID number of child to parent process and a value 0 (zero) to child process.

5.2 Process Termination

A process terminates when it finishes executing its final statement and asks the operating system to delete it by using the **'exit'** system call.

At that point the process may return data (Output) to its parent process (via the 'wait' system call).

Process in UNIX operating system terminate by executing the **exit** system call. An exiting process enters zombie state, releases all its resources dismantles its context except for its slot in the process table.

Syntax

```
exit (status)
```

The value of the status is returned to the parent process for its examination. Processes may call exit explicitly or implicitly at the end of the program.

All the resources of the process are deallocated by the operating system. Termination might occur under additional circumstances.

For example,

A process can cause the termination of another process via an appropriate system call (*example*, abort).

Usually only the parent of the process that is to be terminated can invoke such a system call, otherwise users would arbitrarily keep aborting each others job.

A parent therefore needs to know the identities of its children. Thus when one process creates a new process the identity of the newly, created process is passed to the parent.

A parent can terminate the execution of its children for a number of reasons like:

i. If the child has executed its usage of some of the resources that it has been allocated.

ii. The task assigned to the child is no longer required.

iii. The parent is terminating in this case the operating system does not allow a child to continue if its parent terminates. So if a process terminates (either normally or abnormally) then all its children must also be terminated. This is called as **cascading termination** and is normally initiated by the operating system.

6. Types of Processes

Processes can be of three types

1. **User process:** User process is a process associated with the terminal.
2. **Daemon process:** Daemon process do a system wide function such as administration and control of networks, execution of time dependent activities, line printer spooling. These processes are like user processes in that they run at user node and make system calls to access system services.
3. **Kernel process:** The kernel processes execute only in kernel mode. Kernel process are similar to Daemon processes, they provide system wide services but they have greater control over their execution as it is a part of the kernel. They can access kernel algorithms and data structures directly without use of system calls. But, they are less flexible as kernel need to recompile them to change them.

7. Signals

Signals inform processes of the occurrence of a synchronous events. Processes may send signals with the kill system call or kernel may send signals internally.

Signals are used on termination of process, when a process exits or when a process invoke signal system call on the termination of child process.

Signals are caused by an unexpected error condition during a system call, such as making a nonexistent system call.

Signals are used for tracing execution of the process.

SUMMARY

- A process is more than a program code. A process is a program in execution.
- There are various states of a process.
  - a. New b. Waiting
  - c. Running d. Terminated e. Ready
- The operating system groups all information that it needs about a particular process into a data structure called as a process Descriptor or Process Control Block (PCB).
- Switching the CPU to another process requires scoring the state of the old process and loading the saved state for the new process. This task is known as context switch.
- Various operation on the process can be performed like.
  - a. Process creation: The create() system call is used for creating a new process.
  - b. Process termination: The exit() system call is used for exiting from the process.
- There are three types of process in Unix operating system.
  - a. User processes
  - b. Daemon processes
  - c. Kernel processes
- Signals are used to inform the processes for the occurrences of the synchronous events.

PU Questions

2 Marks

1. What do you mean by context switch? **[Oct.15,14,11,Apr.12 – *2M*]**
2. What is meant by Process? **[Apr.15,Oct.12 – *2M*]**

4 Marks

1. Explain in detail the process control block. **[Oct. 2015 – *4M*]**
2. In normal mode of operation. List and explain the sequence of utilization of resources by process. **[Apr. 2015 – *4M*]**
3. Explain operations on process in detail. **[Apr. 2015 – *4M*]**
4. Explain PCB with proper diagram. **[Apr. 2013 – *4M*]**
5. Explain in detail the various process states with the help of diagram. **[Oct.2012 – *4M*]**
6. Explain the Creation and Termination of Processes. **[Apr.2012 – *4M*]**
7. Explain Process Control Block (PCB) in detail with the help of diagram. **[Oct.11,Apr. 2011 – *4M*]**

Chapter 4 CPU SCHEDULING

1. Introduction

CPU scheduling is the basis of multiprogrammed operating systems. By switching the CPU among processes, the operating system can make the computer more productive. In this chapter, we introduce basic CPU-scheduling concepts and present several CPU-scheduling algorithms. We also consider the problem of selecting an algorithm for a particular system.

2. Scheduling Concepts

In multiprogramming environment, there may be the situation where two or more processes are simultaneously in ready state and if only one CPU is available, then a choice has to be made as to which process should execute next. The part of operating system that makes this choice is called *scheduler* and the algorithms it uses are called *scheduling algorithms*.

A major issue related to scheduling is when to make scheduling decisions.

There are many situations when scheduling is required.

i. When a new process is created, then decision regarding whether parent or child process should be executed.

ii. When a process exits, then some other process must be selected from set of ready states. If no process is in ready state, a system supplied idle process is executed.

iii. When a process blocks on I/O, on a semaphore, or for some other reason, another process has to be selected to run.

iv. When an I/O interrupt occurs, a scheduling decision may be made.

Scheduling algorithms can be divided into two categories depending on how they deal with clock interrupts:

A *non-preemptive* scheduling algorithm picks a process to run and then just lets it run until it blocks or until it voluntarily releases its CPU. In other words, no scheduling decisions are made during clock interrupts. After clock interrupt processing has been completed, the process that was running before the interrupt is always resumed.

A *Preemptive* scheduling algorithm picks a process to run and lets it run for a maximum of some fixed time. If it is still running at the end of time interval, it is suspended and the scheduler picks another process to run.

Different scheduling algorithms are needed in different environment and different application areas. Three distinguishable environments are:

a. **Batch:** No users are impatiently waiting for quick response. This approach reduces process switches thus improving performance.

b. **Interactive:** Here preemption is essential to keep one process from hogging the CPU and denying service to others. One process might shut out all others indefinitely. Hence preemption is needed to prevent this behavior.

c. **Real time:** Here preemption is sometimes not needed since the processes know that they may not run for long periods of time and usually do their work quickly.

Goals of scheduling algorithms also differ under different circumstances as can be seen from following chart:

All systems	Fairness: Giving each process a fair share of CPU *Policy enforcement*: Seeing that stated policy is carried out *Balance*: Keeping all parts of system busy
Batch systems	*Throughput*: Maximize jobs per hour *Turnaround time*: Minimize time between submission and termination *CPU utilization*: Keep CPU busy all the time
Interactive systems	*Response time*: Respond to requests quickly *Proportionality*: Meet users expectations
Real time systems	*Meeting deadlines*: Avoid losing data *Predictability*: Avoid quality degradation in multimedia systems

2.1 CPU Scheduler (Short Term Scheduler)

1

Oct.2014 – *4M*
Explain short term scheduler.

An interesting property of processes is that process execution consists of a cycle of CPU execution and I/O wait. Processes alternate back and forth between these two states.

Whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed. The part of the operating system which carries out this selection process is called as **'Short-term scheduler or CPU scheduler.'**

The scheduler select from among the processes in memory that are ready to execute and allocates the CPU to one of them. The algorithm used by CPU scheduler is called as '**CPU scheduling algorithm.**'

2.2 CPU I/O Burst Cycle

The success of CPU scheduling depends on as observed property of processes. Process execution consists of a cycle of CPU execution and I/O wait. Processes alternate between these two states. Process execution begins with a CPU burst that is followed by an I/O burst, which is followed by another CPU burst, then another I/O burst and so on. Eventually, the final CPU burst ends with a system request to terminate execution. An I/O bound program typically has many short CPU bursts.

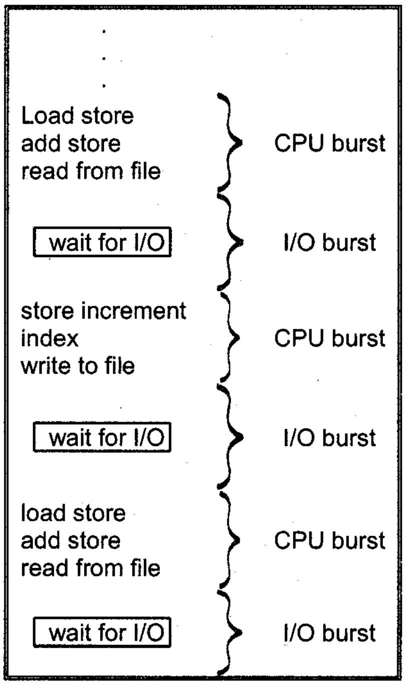


Figure 4.1: Alternating sequence of CPU and I/O bursts

2.3 Preemptive and Non-preemptive Scheduling

1. **Non-preemptive scheduling:** With non-preemptive case, once CPU is given to a particular process, it will not release the CPU till its CPU burst time is over.

   Characteristics of Non-Preemptive Scheduling

   a. In non preemptive system, short jobs are made to wait by longer jobs but the overall treatment of all processes is fair.

   b. In non-preemptive, response times are more predictable because incoming high priority jobs cannot displace waiting jobs.

   c. In non-preemptive scheduling a scheduler executes jobs in the following two situations.

      i. When a process switches from running state to the waiting state.

      ii. When a process terminates.

2. **Preemptive scheduling:** The strategy of allowing processes that are logically, runnable to be temporarily suspended is called **preemptive scheduling** and it is contrast to the **run to completion** method.

A scheduling discipline is preemptive if once a process has been given the CPU can take away.

1 **Oct.2015** – *2M* What do you mean by pre-emptive scheduling?

Differentiate between the Preemptive and Non-preemptive Scheduling

2 **Oct.12, 11** – *4M*

	Non-preemptive	Preemptive
i.	In this, once the CPU is allocated to the process, the process keeps the CPU till the time it terminates or it switches to wait state.	In this, once the process is allocated to CPU, it can be preempted any time as a result of occurrence of higher priority process or an interrupt occurs or process finishes its I/O.
ii.	Context switch is not required.	Requires context switching.
iii.	Always supports single programming.	Supports multiprogramming.
iv.	Suffers from deadlock.	Suffers from starvation.
v.	Process waiting time is less.	Waiting time of low priority processes is more.
vi.	Used in Windows 3.x.	Windows 95 is used till now in all next generations of windows.

2.4 Dispatcher

2 **Oct.2012** – *2M* What is the function of Dispatcher.

Oct.2011 – *2M* Define the term Dispatcher.

The dispatcher is a module that gives control of the CPU to the process selected by the short term scheduler. The time it takes for the dispatcher to stop one process and start another running is known as **'dispatch latencing.'**

1 **Oct.2014** – *2M* Define Dispatch latency.

Functions of Dispatcher

i. Switching context
ii. Switching to user mode
iii. Jumping to the proper location in the user program to restart that program.

3. Scheduling Criteria

Different CPU scheduling algorithms have different properties and the choice of a particular algorithm may favour one class of processes over another.

2
Oct.2014 – *4M*
List and explain scheduling criteria.
Oct.2011 – *4M*
What is CPU Scheduler? State the criteria of CPU Scheduling.

The criteria for good scheduling algorithm are:

1. **CPU Utilization:** We want to keep the CPU as busy as possible. Conceptually, CPU utilization can range, from 0 to 100%. In a real system, it should range from 40% to 90%.

2. **Throughput:** If the CPU is busy executing processes, then work is being done. One measure of work is the number of processes that are completed pre unit time called throughput. For long process, this rate may be one process per hour, for short transactions it may be ten processes per second.

   In short throughput is: *"The number of processes that are completed per time unit is called throughput it should be maximum."*

1
Oct.2012 – *2M*
What is meant by Throughput?

3. **Turnaround time:** The interval from the time of submission of a process to the time of completion of a process is called turnaround time and it should be **minimum**.

   The turnaround time is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU and doing I/O.

3
Apr.15, 13 – *2M*
Define Turnaround Time.
Apr.2011 – *2M*
What do you mean by Turnaround Time?

4. **Waiting time:** Waiting time is the period spent waiting in the ready queue by a process. It should be minimum.

   The CPU scheduling algorithm does not affect the amount of time during which a process executes or does I/O. It affects only the amount of time that a process spends waiting in the ready queue.

2
Apr.2012 – *2M*
Define Waiting Time.
Oct.2011 – *2M*
What do you mean by Waiting Time?

5. **Response time:** Response time is the time from the submission of a request until the first response is produced. This measure called the response time, is the time it takes to start responding, not the time it takes to output the response.

1
Apr.2015 – *1M*
Define Burst Time.

Allocation of the time slice to a process/thread which is waiting for CPU time is referred as burst time. **Burst in computer language is referred when a process/thread** needs CPU time. OS allocates the time slice to each process/thread.

4. Scheduling Algorithm

CPU scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated to the CPU. There are many different CPU scheduling algorithms.

4.1 First Come First Serve Scheduling (FCFS)

It is the simplest CPU scheduling algorithm. In this scheme, the process that requests the CPU first is allocated the CPU first. The average waiting time under the First Come First Serve scheduling (FCFS) policy is often quite long. The FCFS scheduling algorithm is **NON-PREEMPTIVE**. The implementation of the FCFS policy is easily managed with a FIFO queue. When a process enters the ready queue, its PCB (Process Control Block) is linked onto the tail of the queue. When the CPU is free it is allocated to the process at the head of the queue. The running process is then removed from the queue. The code for FCFS scheduling algorithm is simple to write.

Examples

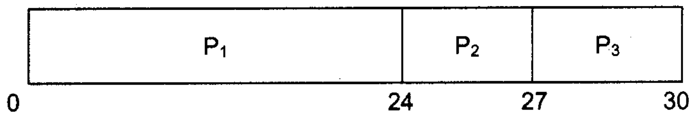
1.

Process	Burst Time
P_1	24
P_2	3
P_3	3

Solution

If the process arrive in the order P_1, P_2, P_3 and are served in FCFS order, we get the result as shown in the following **Gantt Chart**, which is a bar chart that illustrates a particular schedule including the start and finish times of each of the participating process.

Gantt chart



Now the average waiting time for the processes P_1, P_2 and P_3 as follows

$P_1 = 0$ ms (milliseconds)

$P_2 = 24$ ms

$P_3 = 27$ ms

$$\text{Average waiting time (AWT)} = \frac{P_1 + P_2 + P_3}{3}$$

$$= \frac{0 + 24 + 27}{3}$$

$$= 17 \text{ milliseconds}$$

$\therefore$ Average waiting time = 17 ms

Now,

We need to calculate the average turnaround time.

$\therefore$ Average Turnaround Time (ATT) = Waiting Time (WT) + Burst Time (BT)

$\therefore$ $P_1 = 0 + 24 = 24$

$P_2 = 24 + 3 = 27$

$P_3 = 27 + 3 = 30$

$$\text{ATT} = \frac{P_1 + P_2 + P_3}{3}$$

$$= \frac{(24 + 27 + 30)}{3} = \frac{81}{3}$$

$$= 27 \text{ msec.}$$

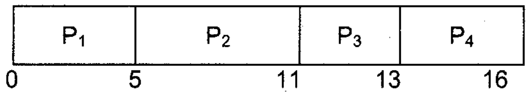
$\therefore$ Average turnaround time = **27 ms**

2.

Process	Burst time
P_1	5
P_2	6
P_3	2
P_4	3

Solution

The Gantt chart for the above processes would be



Average Waiting Time

$P_1 = 0$ ms

$P_2 = 5$ ms

$P_3 = 11$ ms

$P_4 = 13$ ms

$$\text{AWT} = \frac{P_1 + P_2 + P_3 + P_4}{4}$$

$$= \frac{0 + 5 + 11 + 13}{4}$$

$$= \frac{29}{4}$$

AWT = 7.25 ms

Now,

We need to calculate the average turnaround time

Average Turnaround Time =

$P_1 = 0 + 5 = 5$ ms

$P_2 = 5 + 6 = 11$ ms

$P_3 = 11 + 2 = 13$ ms

$P_4 = 13 + 3 = 16$ ms

$$\text{ATT} = \frac{P_1 + P_2 + P_3 + P_4}{4}$$

$$= \frac{5 + 11 + 13 + 16}{4} = \frac{45}{4}$$

ATT = 11.25 ms

∴ Average Turnaround Time = **11.25 ms**

4.2 Shortest-Job First Scheduling Algorithm (SJFS)

This algorithm associates each process with the length of the next CPU burst. When the CPU is available it is assigned to the process that has the smallest next CPU burst. If two process have the same length for next CPU burst, FCFS scheduling is used to break the tie. The shortest job first scheduling algorithm may be either **preemptive or non-preemptive.**

The choice arises when a new process arrives at the ready queue while a previous process is executing. The new process may have a shorter, next CPU burst than what is left of the currently executing process.

A preemptive SJFS algorithm will preempt the currently executing process, whereas a non-preemptive SJFS algorithm will allow the currently running process to finish its CPU burst.

Preemptive SJF scheduling is sometimes called as **Shortest Remaining Time First Scheduling (SRTFS)** algorithm.

Examples

1

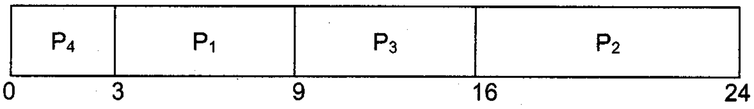
Apr.2011 – *4M*
Calculate Average Turnaround Time and Average Waiting Time for algorithm using Non-preemptive SJF.

1. Non-Preemptive SJF algorithm

Process	Burst time
P_1	6
p_2	8
P_3	7
P_4	3

Solution

Gantt chart



Average Waiting Time for

$P_1 = 3$ ms

$P_2 = 16$ ms

$P_3 = 9$ ms

$P_4 = 0$ ms

$$AWT = \frac{P_1 + P_2 + P_3 + P_4}{4}$$

$$= \frac{3 + 16 + 9 + 0}{4} = \frac{28}{4}$$

AWT = 7 ms

∴ Average Waiting Time = 7 ms

Now,

We need to calculate the average turnaround time.

Average Turnaround Time for

$P_1 = 3 + 6 = 9$ ms

$P_2 = 16 + 8 = 24$ ms

$P_3 = 9 + 7 = 16$ ms

$P_4 = 0 + 3 = 3$ ms

$$\text{Average Turnaround Time} = \frac{P_1 + P_2 + P_3 + P_4}{4}$$

$$= \frac{9 + 24 + 16 + 3}{4}$$

$$= \frac{52}{4}$$

ATT = 13 ms

∴ Average Turnaround Time = **13 ms**

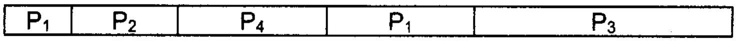
2. Preemptive SJF algorithm

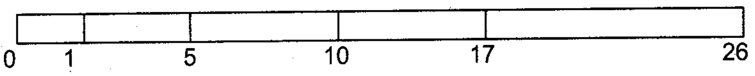
Process	Arrival Time	Burst time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

Solution

If the processes arrive at the ready queue at the times shown and need the indicated burst times then the resulting preemptive SJF schedule is as depicted on the following Gantt Chart.

Gantt chart





Now,

Average Waiting Time for

$P_1 = 10 - 1 = 9$ ms

$P_2 = 1 - 1 = 0$ ms

$P_3 = 17 - 2 = 15$ ms

$P_4 = 5 - 3 = 2$ ms

$$\text{ATT} = \frac{P_1 + P_2 + P_3 + P_4}{4}$$

$$= \frac{9 + 0 + 15 + 2}{4} = \frac{26}{4}$$

AWT = 6.5 ms

$\therefore$ Average Waiting Time = 6.5 ms

Now,

We need to calculate the Average Turnaround Time (ATT)

$$\text{ATT} = \frac{\text{AWT + Burst Time}}{\text{No of processes}}$$

$P_1 = 9 + 8 = 17$ ms

$P_2 = 0 + 4 = 4$ ms

$P_3 = 15 + 9 = 24$ ms

$P_4 = 2 + 5 = 7$ ms

$$\therefore \text{Average Waiting Time} = \frac{P_1 + P_2 + P_3 + P_4}{4}$$

$$= \frac{17 + 4 + 24 + 7}{4} = \frac{52}{4}$$

$= 13$ ms

$\therefore$ Average Turnaround Time = **13 ms**

4.3 Priority Scheduling Algorithm

1

Oct.2015 – *2M*
Explain the working of priority scheduling.

With this scheme, a priority is associated with each process and the CPU is allocated to the process with the highest priority.

Equal priority process are scheduled in FCFS order. Priorities can be defined either internally or externally.

Internally, defined priorities may use time limits, memory requirements and the number of open files etc. in computing priorities.

External priorities are set by criteria that are external to the operating system, such as the importance of the process, the type and amount of funds being paid for computer use, and other often political factors.

Priority scheduling can be either preemptive or non-preemptive.

A major problem with priority scheduling algorithms is **indefinite blocking or starvation.**

A process that is ready to run but lacking the CPU can be considered blocked, waiting for the CPU.

A solution to the problem of indefinite blockage of low priority processes is **aging.**

'Aging is a technique of gradually increasing the priority of processes that wait in the system for long time'.

Examples

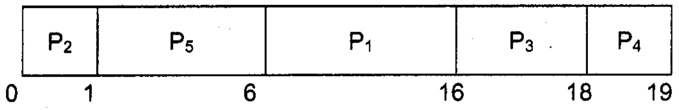
1. **Non- preemptive priority scheduling algorithm**

Process	Burst Time	Priority
P_1	10	3
P_2	1	1 (high)
P_3	2	5
P_4	1	4
P_5	5	2

Solution

Using priority scheduling, we would schedule these processes according to the following Gantt chart

Gantt chart



The Average Waiting Time for

$P_1 = 6$ ms

$P_2 = 0$ ms

$P_3 = 16$ ms

$P_4 = 18$ ms

$P_5 = 1$ ms

$$\text{Average Waiting Time} = \frac{P_1 + P_2 + P_3 + P_4 + P_5}{5}$$

$$= \frac{6 + 0 + 16 + 18 + 1}{5} = \frac{41}{4}$$

AWT = 8.2 ms

∴ Average Waiting Time = 8.2 ms

Now,

∴ Average Turnaround Time for

$P_1 = 6 + 10 = 16$ ms

$P_2 = 0 + 1 = 1$ ms

$P_3 = 16 + 2 = 18$ ms

$P_4 = 18 + 1 = 19$ ms

$P_5 = 1 + 5 = 6$ ms

$$\text{ATT} = \frac{P_1 + P_2 + P_3 + P_4 + P_5}{5}$$

$$= \frac{16 + 1 + 18 + 19 + 6}{5}$$

$$= \frac{60}{5}$$

ATT = 12 ms

∴ Average Turnaround Time = **12 ms**

4.4 Round Robin Scheduling Algorithm

This algorithm is similar to FCFS scheduling, but preemptive is added to switch between processes.

A small unit of time, called a time quantum, or time slice is defined.

The ready queue is treated as a **circular queue**.

New processes are added to the tail of the ready queue.

The CPU scheduler picks the first process from the ready queue sets a timer to interrupt after one time quantum and dispatches the process.

3
Apr.2013 – *4M*
What do you mean by Processor Share in case of Round-Robin Scheduling?
Apr.15,12 – *2M*
"Round Robin Algorithm is Non-preemptive." Comment and Justify.

One of the two things will then happen

i. The process may have a CPU burst time of less than one time quantum. In this case, the process itself will release the CPU voluntarily.

   The scheduler will then proceed to the next process in the ready queue.

ii. Otherwise, if the CPU burst of the currently running process is longer than one 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 CPU scheduler will then select the next process in the ready queue.

The average waiting time under the Round Robin policy is often quite long. This algorithm is **preemptive algorithm**.

Time Shot

In operating system time shot/time slice Or "time quantum", "quantum" is the period of time for which a process is allowed to run uninterrupted in a pre-emptive multitasking operating system.

1
Oct.2015 – *1M*
Define the term time shot.

Examples

1. Preemptive for Round Robin (RR) algorithm

Process	Burst Time
P_1	24
p_2	3
P_3	3

Solution

Consider the above set of processes that arrive at time 0 with the length of the CPU burst given in milliseconds.

If we use time quantum/time slice of 4 ms

i.e. **time slice/quantum** = 4 ms

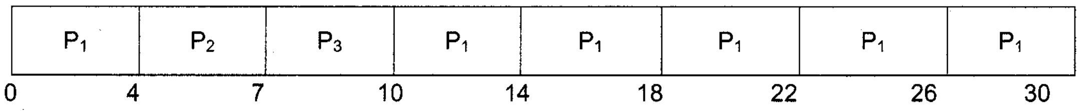
Then process P_1 gets the first 4 ms since it requires another 20 ms, it is preempted after the first time slice and the CPU is given to the next process in the queue process P_2.

Process P_2 does not need 4 ms, so it quits before time slice expires.

The CPU is then given to the next process P_3. Once each process has received 1 time slice, the CPU is returned to the process P_1 for an additional time slice.

The resulting RR (Round Robin) schedule is as follows

Gantt Chart



Lets,

Calculate the average Waiting Time for

$P_1 = 10 - 4 = 6$ ms

$P_2 = 4$ ms

$P_3 = 7$ ms

$$\text{Average Waiting Time} = \frac{P_1 + P_2 + P_3}{3}$$

$$= \frac{6 + 4 + 7}{3} = \frac{17}{3}$$

$$= 5.66 \text{ ms}$$

Average Waiting Time = 5.66 ms

Now, lets calculate the average turnaround time

$P_1 = 6 + 24 = 30$ ms

$P_2 = 4 + 3 = 7$ ms

$P_3 = 7 + 3 = 10$ ms

$$\text{ATT} = \frac{30 + 7 + 10}{3} = \frac{47}{3}$$

$$= 15.6 \text{ ms}$$

$\therefore$ Average Turnaround Time = 15.6 ms

Note: **Time slice\Time quantum –** It is the largest amount of CPU time any program can consume when scheduled to execute on the CPU.

4.5 Multilevel Queues

1

Apr.2011 – *4M*
Explain Multilevel Queue Scheduling in detail.

Another class of scheduling algorithms has been created for situations in which jobs are easily classified into different groups. *Example*: common division is made between foreground (Interactive) jobs and background (Batch) jobs.

A multilevel queue scheduling algorithm partitions the ready queue in separate queues. Jobs are permanently assigned to one queue generally based on some property of the job like memory size or job type, etc. each queue has its own scheduling algorithm.

In addition, there must be scheduling between the queues. This is commonly fixed priority preemptive scheduling. *For example:* Foreground queue may have absolute priority over the background queue.

For example: A multi-queue scheduling can have following queues:

i. System jobs
ii. Interactive programs
iii. Interactive editing
iv. Batch jobs
v. Student jobs

Each queue has absolute priority over lower priority queues. No job in the batch queue, *for example*, can execute unless queues for the system jobs, interactive jobs, and interactive editing are empty (*figure 4.2*).

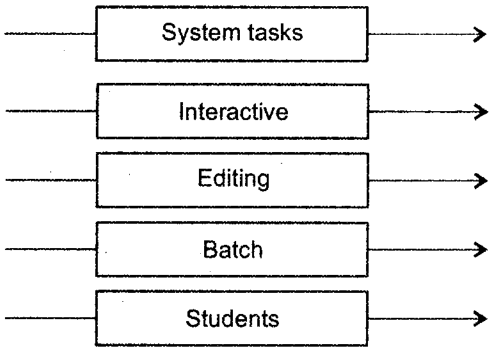


Figure 4.2: Multi level queue scheduling

4.6 Multi Level Feedback Queues

2

Apr.2015 – *4M*
Explain multilevel feedback queue algorithm

Apr.2012 – *4M*
Explain Multilevel Feedback Queue Scheduling Algorithm in detail.

Normally, in a multi queue scheduling algorithm, jobs are permanently assigned to a queue upon entry to the system. Jobs do not move between queues.

Multilevel feedback queue however, allow a job to move between queues. The idea is that if a job uses too much of CPU time, it will be moved to a lower priority queue.

Similarly a job which waits too long in a lower-priority queue it may be moved to a higher priority queue.

For example: Consider a multilevel feedback queue scheduler with 3 queues, numbered from 0 to 2. The scheduler first executes all jobs in queue 0.

A job entering the ready queue is put in queue 0. A job in queue 0 is given a time quantum of 8 milliseconds. If it does not finish within this time, it is moved to the tail of queue 1. If queue 0 is empty, the job at the head of queue 1 is given a quantum of 16 milliseconds. If it does not complete, it is preempted and put into queue 2. Jobs in queues are executed FCFS only when, Queue 0 and 1 are empty (*figure 4.3*).

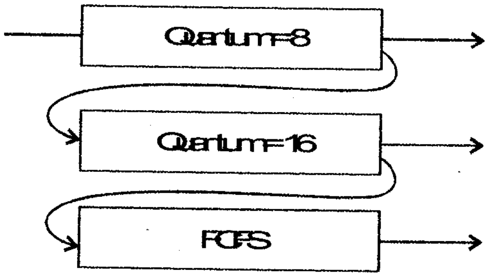


Figure 4.3: Multilevel feedback queue

5. Operation System Examples

The scheduling algorithms (policies) used by the Solaris, Windows XP, Window 2000 and Linux operating system are

1. **Solaris scheduling:** Solaris uses priority based thread scheduling where each thread belongs to one of the six classes
   a. Time sharing (TS)

b. Interactive (IA)

c. Real time (RT)

d. System (Sys)

e. Far Systems Source (FSS)

f. Fixed Priority (FP)

Within each class there are different priorities and different scheduling algorithms.

The default scheduling class for a process is time sharing.

2. **Windows XP scheduling/Windows 2000 scheduling:** Window XP schedules threads using a priority based, preemptive scheduling algorithm.

   The windows XP scheduler ensures that the highest priority thread will always run. The portion of the windows XP Kernel that handles scheduling is called the **dispatcher.**

   A thread selected to run by the dispatcher will run until it is preempted by a higher priority thread, until it terminates, until its time quantum ends, or until it calls a blocking system call, such as for I/O.

3. **Linux scheduling:** Linux scheduler provides two separate process scheduling algorithms. They are

   a. Time sharing algorithm for fair preemptive scheduling among multiple processes and

   b. Other is designed for real time tasks where absolute priorities are more important than fairness. Linux allows only process running in user mode to be preempted.

Solved Examples

1. **Calculate Average Turn Around Time and Average Waiting Time for all set of processes using Non-pre-emptive Priority:**

1 Apr.2013 – 4M

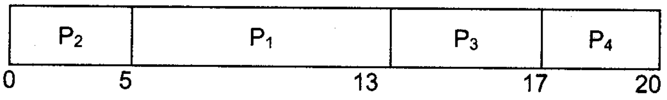
Process	Burst Time	Arrival Time	Priority
P_1	8	2	2
P_2	5	1	1(high)
P_3	4	0	3
P_4	3	3	4

Solution

Given

Process	Burst Time	Arrival Time	Priority
P_1	8	2	2
P_2	5	1	1(high)
P_3	4	0	3
P_4	3	3	4

Gantt chart



Average waiting time

$$\begin{aligned} P_1 &= 5 \\ P_2 &= 0 \\ P_3 &= 13 \\ P_4 &= \underline{17} \\ & 35/4 = 8.75 \text{ m/sec} \end{aligned}$$

Average turn around time = Burst time + Waiting time

$$\begin{aligned} P_1 &= 8 + 5 = 13 \\ P_2 &= 5 + 0 = 5 \\ P_3 &= 4 + 13 = 17 \\ P_4 &= 3 + 17 = \underline{20} \\ & \quad 55/4 = 13.75 \text{ m/s} \end{aligned}$$

∴ Average turnaround time = 13.75 m/sec.

1 Oct.2012 – *4M*

2. Calculate Average Turn Around Time and Average Waiting Time for all set of processes using SJF.

Process	Burst time	Arrival time
P_1	4	1
P_2	3	0
P_3	2	2
P_4	4	3
P_5	1	2

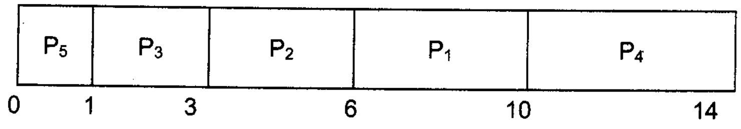
Solution

Process	B. T.	A. T.
P_1	4 (4)	1
P_2	3 (3)	0
P_3	2 (2)	2
P_4	4 (5)	3
P_5	1 (1)	2

Step 1: Shortest jobs which will be executed are:

Process	B. T.
P_5	1
P_3	2
P_2	3
P_1	4
P_4	4

Step 2: Drawing Gantt chart



Step 3: Calculating Average turnaround time and Average waiting time.

Average waiting time

$$\begin{array}{ll} P_1 & = 6 \\ P_2 & = 3 \\ P_3 & = 1 \\ P_4 & = 10 \\ P_5 & = \underline{\ 0\ } \\ & 20/5 \quad = 4 \text{ m/sec} \end{array}$$

Average waiting time= 4 m/sec

Average turn around time = Waiting time + Burst time

$$\begin{array}{lrl} P_1 = & 6 + 4 = & 10 \\ P_2 = & 3 + 3 = & 6 \\ P_3 = & 1 + 2 = & 3 \\ P_4 = & 10 + 4 = & 14 \\ P_5 = & 01 + 0 = & \underline{01} \\ & & 34/5 \quad 6.8 \text{ m/sec} \end{array}$$

∴ Average turnaround time = 6.8 m/sec.

3. Calculate Average Turn Around Time and Average Waiting Time for all set of processes using FCFS:

1 Apr.2012 – *4M*

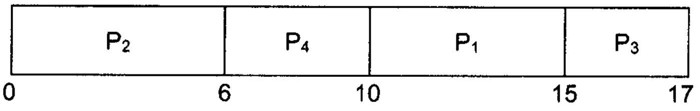
Process	Burst time	Arrival time
P1	5	1
P2	6	0
P3	2	2
P4	4	0

Solution

Given

Process	Burst time	Arrival time
P_1	5	1
P_2	6	0
P_3	2	2
P_4	4	0

Gantt chart by applying FCFS



Waiting Time: P_1 = 10

P_2 = 0

P_3 = 15

P_4 = 06

$$\text{Average Waiting Time} = \frac{P_1 + P_2 + P_3 + P_4}{4} = \frac{10 + 0 + 15 + 06}{4} = \frac{31}{4}$$

$$= 7.75 \text{ units}$$

Turn Around Time: $P_1 = 15$

$P_2 = 6$

$P_3 = 17$

$P_4 = 10$

$$\text{Average Turn Around Time} = \frac{15 + 6 + 17 + 10}{4} = \frac{48}{4} = 12$$

$\therefore$ ATT = 12, AWT = 7.75

1

Oct.2011 – *4M*

4. Consider the following set of processes:

Process	CPU Burst Time (in milliseconds)
P_1	30
P_2	6
P_3	8

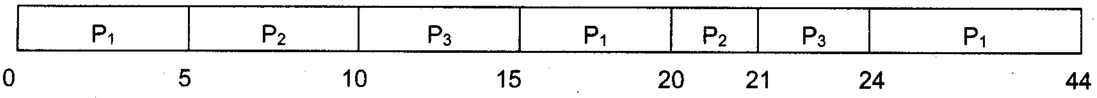
Calculate the Average Waiting Time and Average Turnaround time by using Round Robin CPU Scheduling Algorithm. (The time quantum is of 5 milliseconds)

Solution

Given

Process	CPU Burst Time (in milliseconds)
P1	30
P2	6
P3	8

Round Robin CPU scheduling Algorithm with time quantum of 5 ms.



Let's, calculate AWT for

$P_1 = 0 + (15 - 5) + (24 - 20) = 10 + 4 = 14$ ms

$P_2 = 5 + (20 - 10) = 5 + 10 = 15$ ms

$P_3 = 10 + (21 - 15) = 10 + 6 = 16$ ms

$$\therefore \text{AWT} = \frac{P_1 + P_2 + P_3}{3} = \frac{14 + 15 + 16}{3} = \frac{45}{3} = 15 \text{ ms.}$$

Calculate ATT

$$\text{ATT} = \frac{\text{AWT + Brust time}}{\text{No. of processes}}$$

$$\therefore\ P_1 = 14 + 30 = 44$$

$$P_2 = 15 + 6 = 21$$

$$P_3 = 16 + 8 = 24$$

$$\therefore\ \text{ATT} = \frac{44 + 21 + 24}{3} = \frac{89}{3}$$

$$= 29.6 \text{ ms.}$$

5. Calculate Average Turnaround Time and Average Waiting Time for algorithm using Non-preemptive SJF.

1

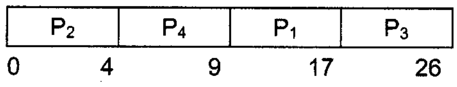
Apr.2011 – *4M*

Process	**Burst Time**	**Arrival Time**
P_1	8	0
P_2	4	1
P_3	9	2
P_4	5	3

Solution

Process	Burst Time	Arrival Time
P_1	8	0
P_2	4	1
P_3	9	2
P_4	5	3

Gantt chart



Average waiting time

$P_1 =$ 9
$P_2 =$ 0
$P_3 =$ 17
$P_4 =$ 4

30 /4 = 75 m/sec

Arrange turnaround time = Arrange time – Burst time

$P_1 = 9 - 8 =$ 1
$P_2 = 0 - 4 =$ 4
$P_3 = 17 - 9 =$ 8
$P_4 = 4 - 5 =$ 1

14 /4 = 3.5 m/sec

SUMMARY

- The main objective of multiprogramming is to have some process running at all times in order to maximize CPU.
- **Short-term scheduler/CPU scheduler**: Whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed. The part of the operating system which carries out this selection process is called as short-term scheduler or CPU scheduler.
- **Non-preemptive scheduling:** With non-preemptive case, once CPU is given to a particular process, it will not release the CPU till to CPU burst time is over.
- **Preemptive scheduling:** The strategy of allowing processes that are logically runnable to be temporarily suspended is called preemptive scheduling and it is contrast to the run to completion method.
- **Dispatcher:** The dispatcher is a module that gives control of the CPU to process selected by CPU scheduler.
- **Dispatch latencing:** The time it takes for the dispatcher to stop one process and start another running is known as the **dispatch latencing**.
- **Throughput:** The number of processes that are completed per time unit is called throughput and it should be maximum.
- **Turnaround time:** The interval from the time is submission of a process to the time of completion of a process is called turnaround time and it should be minimum.
- **Waiting time:** Waiting time is the period spent waiting in the ready queue it should be minimum.
- **Response time:** It is the time from the submission of a request until the first response is produced.
- The CPU scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated to the CPU.
  CPU uses many different CPU scheduling algorithms such as
  a. FCFS
  b. SJFS
  c. RR
  d. Priority based scheduling
  e. Multilevel feedback queue
  f. Multilevel queue
- The examples of operating system when uses different CPU scheduling algorithms are
  a. *Solaris:* It uses priority based scheduling algorithm.
  b. *Windows XP/Windows 2000:* It uses priority based, preemptive scheduling algorithm.
  c. *Linux:* It uses two types of CPU scheduling algorithms
     i. Time sharing algorithm for fair preemptive scheduling among multiple processes.
     ii. Real time tasks where absolute priorities are more important than fairness.
- **Time quantum/ Time slice:** It is the largest amount of CPU time any program can consume when scheduled to execute on the CPU.

PU Questions

2 Marks

1. Define the term time shot. [Oct.2015 – *1M*]
2. What do you mean by pre-emptive scheduling? [Oct.2015 – *1M*]
3. Round Robin algorithm is non-preemptive comment and justify. [Apr.15,12 – *1M*]
4. Define Burst Time. [Apr.2015 – *1M*]
5. Define the term Turn-Around Time. [Oct.14,Apr.13 – *2M*]
6. Define Dispatch latency. [Oct.2014 – *2M*]

7. What is the function of Dispatcher? **[Oct.2012 – *2M*]**
8. What is meant by Throughput? **[Oct.2012 – *2M*]**
9. Define Waiting Time. **[Apr.2012 – *2M*]**
10. Define the term Dispatcher. **[Oct.2011 – *2M*]**
11. What do you mean by Waiting Time? **[Oct.2011 – *2M*]**
12. What do you mean by Turnaround Time? **[Apr.2011 – *2M*]**

4 Marks

1. Explain the working of priority scheduling. **[Oct.2015 – *4M*]**
2. Consider the following set of processes with the length of CPO Burst time and arrival time given in milliseconds: **[Oct.2015 – *4M*]**

Process	Burst time	Arrival time
P_1	4	1
P_2	2	0
P_3	3	3
P_4	5	2
P_5	7	2

Calculate turn around time, waiting time, average waiting time and average turn around time using FCFS algorithm.

3. Explain multilevel feedback queue algorithm. **[Apr2015 – *4M*]**
4. Consider the following set of processes with the length of CPU Burst time and arrival time. **[Apr2015 – *4M*]**

Process	Burst time	Arrival time
P_1	5	1
P_2	3	0
P_3	2	2
P_4	4	3
P_5	2	13

Calculate Turn around time, waiting time, Average waiting time and Average turn around time using Round Robin Algorithm with time quantum = 2.

5. List and explain scheduling criteria. **[Oct.2014 – *4M*]**
6. Calculate Turn Around Time, Average Turn Around time, waiting time and average waiting time for all set of processes using SJF (Shortest Job First).Non Preemptive algorithm. **[Oct.2014 – *4M*]**

Process	Burst time	Arrival time
P_1	3	1
P_2	2	2
P_3	5	0

[Oct.2014 – *4M*]

7. Explain short term scheduler.

[Apr.2013 – *4M*] 8. Calculate Average Turn Around Time and Average Waiting Time for all set of processes using Non-pre-emptive Priority:

Process	Burst Time	Arrival Time	Priority
P_1	8	2	2
P_2	5	1	1(high)
P_3	4	0	3
P_4	3	3	4

[Apr.2013 – *4M*] 9. What do you mean by Processor Share in case of Round-Robin Scheduling?

[Oct.12,11 – *4M*] 10. Differentiate between the Preemptive and Non-preemptive Scheduling.

[Oct.2012 – *4M*] 11. Calculate Average Turn Around Time and Average Waiting Time for all set of processes using SJF.

Process	Burst time	Arrival time
P_1	4	1
P_2	3	0
P_3	2	2
P_4	4	3
P_5	1	2

[Apr.2012 – *4M*] 12. Explain Multilevel Feedback Queue Scheduling Algorithm in detail.

[Apr.2012 – *4M*] 13. Calculate Average Turn Around Time and Average Waiting Time for all set of processes using FCFS:

Process	Burst time	Arrival time
P1	5	1
P2	6	0
P3	2	2
P4	4	0

[Oct.2011 – *4M*] 14. Consider the following set of processes:

Process	CPU Burst Time (in milliseconds)
P_1	30
P_2	6
P_3	8

Calculate the Average Waiting Time and Average Turnaround time by using Round Robin CPU Scheduling Algorithm. (The time quantum is of 5 milliseconds)

[Oct.2011 – *4M*] 15. What is CPU Scheduler? State the criteria of CPU Scheduling

[Oct.2011 – *4M*] 16. A disk drive has 540 cylinders numbered 0-539. The drive is currently serving the request at cylinder 54. The queue is in order:98,183, 47, 125, 10, 126, 380, 200, 79.Starting from the current head position what is the total distance that the disk arm moves to satisfy all the pending request for the following Disk Scheduling Algorithm?

i. FCFS ii. SCAN

Chapter 5 PROCESS SYNCHRONIZATION

1. Introduction

A co-operating process is one that can affect or be affected by other processes executing in the system. The co-operating processes may either directly share code and data or be allowed to share data only through files or messages.

If more than one processes are allowed to access the same set of data concurrently, then the execution of these concurrent processes might leave the data incorrect. Such a situation where several processes access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place is called as a race condition.

To guard against race condition, we must ensure that only one process at a time can be manipulating the data which is commonly used.

For this, some kind of synchronization of the processes is required, such situations occur frequently in operating systems as different parts of the system manipulate resources and we want that changes should not interfere with one another.

2. Interprocess Communication

Very often processes may need to communicate and may need to use a shared resource. This resource can be a Software (S/W) resource such as a file a global variable etc or can be a Hardware (H/W) resource such as printer or tape drive.

Race Condition

Situations where two or more processes are reading or writing some shared data and the final result depends on who runs precisely when, are called race condition.

To avoid race conditions, we have to find some way to prohibit more than one process from reading and writing the shared data at the same time we need mutual exclusion some way of making sure that if one process is using a shared variable or file, the other process will be excluded from doing the same thing.

The part of the program where the shared memory is accessed is called the **critical section** or **critical region.**

To avoid race conditions following four conditions must hold

i. No two processes may be simultaneously inside their critical sections.

ii. No assumptions may be made about speeds or the number of CPU's.

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

iv. No process should have to wait for over to enter its critical section.

3. Critical Section Problem

1
Oct. 2012 – *2M*
Define the term Critical Section.

Consider a system consisting of n cooperating processes $\{P_1, P_2, P_3, \ldots, P_n\}$. Each process has a segment of code called the **critical section** in which the process may be reading common variables, updating a common table, writing a common file and so on.

The important feature of the system is that when one process is executing in its critical section, no other process is to be allowed to execute in its critical section. Thus the execution of critical sections by the process is mutually exclusive in time. The critical section problem is to design a protocol which the processes may use to cooperate. Each process must request permission to enter its critical section. The section of code implementing this request is the entry section. The critical section may be followed by an exit section and the remaining code is the remainder section.

A solution to the critical section problem must satisfy the following requirements:

1. **Mutual exclusion:** If process P_i is executing in its critical section then no other process can be executing in its critical section.

2. **Progress:** If no process is executing in its critical section and there exists some processes that wish to enter in the critical section, then only those processes that are not executing in their remainder section can participate in the decision as to who will enter in the critical section next, and this section cannot be postponed indefinitely.

3. **Bounded waiting:** There must exist a bound on the number of times that other processes are allowed to enter their critical sections, after a process has made a request to enter its critical section and before that request is granted.

We shall try to find solution to the critical section problem that satisfy these three requirements.

2

Oct. 2015 – *4M*
Explain the critical section problem in detail.

Apr. 2013 – *4M*
What is Critical Section Problem? Explain the following term in the context of it:
i. Mutual Exclusion
ii. Progress
iii. Bounded Wait

1

Apr. 2012 – *4M*
What is Critical Section Problem? Which requirement must be satisfied by solution to the Critical Section?

While constructing an algorithm a typical process p_i is considered to be similar as shown in *figure 5.1.*

```
do {
    entry section
        critical section;
    exit section
        remainder section;
 } while(1);
```

Figure 5.1: General structure of a typical process p_i

4. Semaphores

6
Oct. 2015 – *2M*
Define semaphore.
Apr.15.13,12 – *2M*
What is Semaphore?
Oct.11, Apr.11 – *4M*
Write a short note on Semaphores.

In 1965, *E.W. Dijkstra* suggested using an integer variable to count the number of wake ups saved for future use. In his proposal, a new variable type, called a semaphore was introduced.

A semaphore would have the value few indicating that no wake ups were saved, or some positive value of one or more wake ups were pending.

Checking the value changing it, and possibly going to sleep is all done as a single, indivisible atomic action.

It is guaranteed that once a semaphore operation has started, no other process can access the semaphore until the operation has completed or blocked. This atomicity is absolutely essential for solving synchronization problems and avoiding race conditions.

Dijkstra proposed having two operations

1. **Down:** This operation decrements the value of the semaphore addressed if it is greater than zero.
2. **Up:** This operation increments the value of the semaphore addressed.

   A semaphore is a synchronization tool used to deal with the critical section on the mutual exclusion problem.

   In general, a semaphores is an integer variable apart from initialization can be accessed only through two standard atomic operations P and V.

   P is called as 'wait' and 'V' is called as 'signal'.

   The definition of P and V is as follows:

```
wait(s)/P(s):while s ≤ 0 do skip;
          s:= s-1;
signal(s)/ V(s): s:= s+1;
```

   Semaphore can be used in dealing with the n-process critical section problem. The n process share a common semaphore 'mutex' initialized to 1.

   Each process P_l is organized as follows:

```
repeat
     p(mutex);
        critical section
     v(mutex)
        remainder section
while false
```

Types of Semaphore

1

Oct. 2014 – *4M*
Define semaphores. List its types. Explain any one in detail.

There are two types of semaphore

i. **Counting semaphore:** The semaphore described above is known as 'Counting semaphore', since its integer value may range over an unrestricted domain.

ii. **Binary semaphore:** It is a semaphore with an integer value that can range only between 0 and 1. It can be simple to implement than a counting semaphore.

4.1 Usage

Semaphores can be used to tackle the n process critical section problem. The n process share a semaphore called 'mutex' which is initialized to 1. Each process p_i is organized as shown in *figure 5.2.*

```
do {
        wait(mutex);
            critical section;
        signal(mutex);
            remainder section;
    } while(1);
```

Figure 5.2: Mutual exclusion implemented with semaphores

Semaphores can also be used to solve various synchronization problems. *For example:* Consider two concurrently running processes, p_1 with a statement s_1 and p_2 with a statement s_2. Suppose we require that s_2 be executed only after s_1.

We can do this by letting p_1 and p_2 share a common semaphore 'synch', which is initialized to 0 and by inserting the statements.

```
s1;
signal(synch);
In the process p1 and the statements
wait(synch);
s2;
```

In process p_2, because synch is initialized to 0, p_2 will execute s_2 only after p_1 has invoked signal (synch) which is after s_1.

4.2 Implementation

The main disadvantage of above scheme is that they all require busy waiting i.e. while a process is in critical section, any other process which tries to enter the critical section, must continuously loop in its entry code. This wastes CPU's cycles, which otherwise could have been used by some other process for some productive work.

To solve this problem, we must modify the definition of 'wait' and 'signal' semaphore operations. When a process executes the wait operation, it must block itself. The block operation places the process into a waiting queue associated with the semaphore. The process which is blocked, waiting on a semaphore S should be restarted when some other process executes a 'signal' operation. This is done by a 'wake up' operation, which changes the process from the waiting state to the ready state. The process is then placed in the ready queue. Such semaphores are also called as a 'spinlock'.

Here a semaphore is defined as:

```
typedef struct
{
   int value;
   struct process *L;
} semaphore;
```

Each semaphore has an integer value and a list of processes. When a process must wait on a semaphore, it is added to the list of processes. A signal operation removes one process from the list and awakens that process.

The 'wait' semaphore operation can now be defined as:

```
void wait (semaphore s)
{
   s.value - -;
   if (s.value < 0)
   {
      add this process to S.L;
      block ();
   }
}
```

A signal operation can be defined as:

```
void signal (semaphore s)
{
   s.value ++;
```

```
  if (s.value <=0)
  {
    remove a process p from s.L;
    wakeup(p);
  }
}
```

4.3 Deadlock and Starvation

The implementation of a semaphore with a waiting queue may result in a situation where two or more processes are waiting indefinitely for an event that can be caused only by one of the waiting process. Here we say that the processes are deadlocked.

For example: Consider two processes p_0 and p_1 which are executing simultaneously and using the semaphores S and Q, set to value 1 if the statements in p_0 and p_1 are executed in the following sequence:

p_0	p_1
Wait (S)	
	Wait (Q)
Wait (Q)	
⋮	Wait (S)
	⋮
Signal (S)	Signal (S)
Signal (Q)	Signal (Q)

Here p_0 executes wait (S) then p_1 executes wait (Q), now when p_0 tries to execute wait (Q), it must wait until p_1 executes signal (Q). Similarly, when p_1 executes wait (S), it must wait until p_0 executes signal (S). Thus each process is waiting for the other to proceed and they are deadlocked.

Another problem related to deadlock is indefinite blocking a saturation or starvation, where a process waits indefinitely within a semaphore.

4.4 Binary Semaphores

The semaphore described above is known as counting semaphore, since its integer value may range over an unrestricted domain. A binary semaphore is a semaphore with an integer value that can range only between 0 and 1. A binary semaphore can be simple to implement than a counting semaphore. We now show, how a counting semaphore can be implemented using a binary semaphore.

Let S be a counting semaphore. To implement it in terms of binary semaphores, we need the following data structure:

```
Binary_semaphore s1,s2;
int c;
```

Initially $s_1 = 1$, $s_2 = 0$ and value of c is set to the initial value of counting semaphore S.

The wait operation on the counting semaphore S can be implemented as follows:

```
wait(s1);
c--;
if(c<0){  signal(s1);
          wait(s2);
       }
       signal(s1);
```

The signal operation on the counting semaphore S can be implemented as follows:

```
wait(s1);
c++;
if(c<=0)
  signal(s2);
else
  signal(s1);
```

5. Monitors

To make it easier to write correct programs, *Hoare* (1974) and *Brinch Hansen* (1975) proposed a higher level synchronization primitive called a **Monitor.**

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

Processes may call the procedures in a monitor whenever they want to, but they cannot directly access the monitors internal data structures from procedures declared outside the monitor.

Monitors have an important property that makes them useful for achieving mutual exclusion only one process can be active in a monitor at any instant. typically, when a process calls a monitor procedure the first few instructions of the procedure will check to see of any other process is currently active within the monitor.

If so, the calling process will be suspended until the other process has left the monitor.

If no other process is using the monitor, the calling process may enter.

6. Classical Problems of Synchronization

In this section, a couple of different synchronization problems as an example for a large class of concurrency control problems are presented.

1

Apr. 2011 – *4M*
Explain bounded buffer problem in detail with the help of suitable example.

6.1 The Bounded Buffer Problem

We have a producer process and a consumer process. The producer process produces information that is consumed by a consumer process. The bounded buffer producer consumer problem assumes a fixed buffer size. In this case, the consumer must wait if the buffer is empty and the producer must wait if the buffer is full.

Here, we assume that the pool consists of n buffers, each capable of holding one item. The mutex semaphore provides mutual exclusion for accesses to the buffer pool is initialized to the value 1. The 'empty' and 'full' semaphore count the number of empty and full buffers respectively. The semaphore 'empty' is initialized to the value n, the semaphore 'full' is initialized to the value 0.

The code for the producer process is as below:

```
do
{
  produce an item in next p
  =
  wait(empty);
  wait(mutex);
```

```
    =
    add next p to buffer
    =
    signal(mutex);
    signal(full);
} while (1);
```

The code for the consumer process is as shown below:

```
do
{
    wait(full);
    wait(mutex);
    =
    remove an item from buffer to next c
    =
    signal(mutex);
    signal(empty);
    =
    consume the item in next c
    =
} while (1);
```

We can interpret this code as the producer producing full buffers for the consumer or as the consumer producing empty buffers for the producer.

1
Apr. 2015 – *4M*
Explain the Reader's writer's problem which is a classic problem of synchronization.

6.2 Readers and Writers Problem

An object (*for example:* a file or record) is to be shared among several concurrent processes. Some processes want to only read the data object such processes are called reader process. Some other processes may want to update the shared object. Such processes are called writers. If more than one reader processes access the object simultaneously then there is no problem. But if two writer process want to access the object simultaneously then it might result in a problem. To ensure that these difficulties do not arise, we require that the writers have exclusive access to the shared object. This problem is referred to as 'Readers Writers' problem.

A variation in the readers writers problem is the first readers-writers problem, it requires that no readers will be kept waiting unless a writer has already obtained permission to use the shared object. In other words, no reader should wait for other readers to finish simply because a writer is waiting.

In the solution to the first readers writers problem, the reader process share the following data structures

```
semaphore mutex, wrt;
int readcount;
```

The semaphores mutex and wrt are initialized to 1; readcount is initialized to 0. The semaphore wrt is common to both the reader and writer processes. The mutex semaphore is used to ensure mutual exclusion when variable readcount is updated, it keeps track of how many processes are currently reading the object.

The semaphore wrt functions as a mutual exclusion semaphore for the writers. It is used by 1st or last reader that enters or exits its critical section. It is not used by readers who enter or exits while other readers are in their critical sections.

The code for a writer process is as below:

```
wait(mutex);
readcount ++;
if(readcount == 1)
    wait(wrt);
  signal(mutex);
  =
  reading is performed
  ≡
  wait(mutex);
  readcount --;
  if(readcount == 0)
    signal(wrt);
  signal(mutex);
```

Here if writer is in critical section and n readers are waiting, then one reader is queued on wrt, and n – 1 readers are queued on mutex. When a writer executes signal (wrt), we may resume execution of the waiting readers or a single waiting writer.

6.3 Dining Philosophers Problem

3

Oct. 2015 – *4M*
Explain in detail Dining-Philosopher Problem.

Oct. 2012,11 – *4M*
Describe in detail the "Dining Philosopher Problem" Synchronization Problem.

There are five philosophers who spend their life in thinking and eating. They share a common circular table and 5 chairs, each belonging to one philosopher. In the centre of the table there is a bowl of rice and the table is laid with 5 single chopsticks (*figure 5.3*). When a philosopher thinks, she does not interact with her colleagues, from time to time a philosopher gets hungry and picks up 2 chopsticks that are closer to her. A philosopher may pick up only one chopstick at a time. When hungry and having both the chopsticks, she eats without releasing her chopsticks. When she finishes eating, she puts down both the chopsticks and starts thinking.

One simple solution is to represent each chopstick by a semaphore. A philosopher tries to grab the chopstick, executing a wait operation on that semaphore, she releases her chopsticks by executing the signal operation on the appropriate semaphore. Thus the shared data are

```
semaphore chopstick [5];
```

where all the elements of chopstick are initialized to 1. The structure of philosopher i is as shown below:

```
do
{
  wait(chopstick{i});
  wait(chopstick[(i+1)%5]);
  ≡
    eat;
  ≡
  signal(chopstick[i]);
  signal(chopstick[(i+1)%5]);
  ≡
    think;
  ≡
  } while (1);
```

Although it guarantees that no two neighbours are eating simultaneously. It must be rejected because it has a possibility of creating a deadlock. Suppose that all five philosophers became hungry and grab 1 chopstick each. To ensure that there is no deadlock. We must

i. Allow at most four philosophers to be sitting simultaneously at the table.

ii. Allow a philosopher to pick up her chopstick only if both chopsticks are available.

OR

ii. Use an asymmetric solution i.e. an odd philosopher picks up first her left chopstick and then her right chopstick, an even philosopher pick up her right chopstick and then her left chopstick.

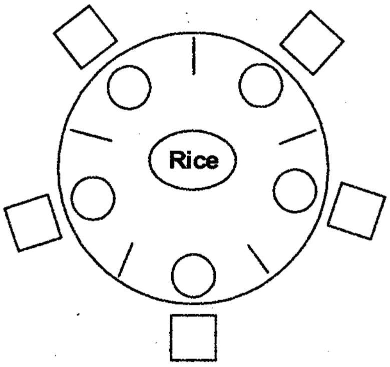


Figure 5.3: The situation of the dinning - philosopher

SUMMARY

- **Race conditions:** Situations where two or more processes are reading or writing some shared data and the final result depends on who runs precisely when, are called race conditions.
- **Critical section/ region**: The part of the program where the shared memory is accessed is called the critical region or critical section.
- A semaphore is a synchronization tool used to deal with the critical section on the mutual exclusion problem.
- To make it easier to write correct program *Moare* and *Brench Husen* proposed a higher level synchronization primitive called a monitor.
- **Monitor:** A monitor is a collection of procedures variables and data structures that are all grouped together in a spiral kind of module or package.
- The classic problems of synchronization can be solved using the following techniques
  a. Bounded buffer problem
  b. Readers writers problem
  c. Dining philosophers problem
- **Binary semaphore:** A binary semaphore is a semaphore with an integer value that can range only between 0 and 1.

PU Questions

2 Marks

[Oct.2015 – 2M] 1. Define semaphore.

[Apr.15,13,12 – 2M] 2. What is Semaphore?

[Oct.2012 – 2M] 3. Define the term Critical Section.

4 Marks

[Oct.2015 – 4M] 1. Explain the critical section problem in detail.

[Oct.2015 – 4M] 2. Explain in detail Dining-Philosopher Problem.

[Apr.2015 – 4M] 3. Explain the Reader's writer's problem which is a classic problem of synchronization.

[Oct.2014 – 4M] 4. Define semaphores. List its types. Explain any one in detail.

[Apr.2013 – 4M] 5. What is Critical Section Problem? Explain the following term in the context of it:

i. Mutual Exclusion
ii. Progress
iii. Bounded Wait

[Oct.2012 – 4M] 6. What is Semaphore? List and explain different types of the Semaphores.

[Oct.12,11 – 4M] 7. Describe in detail the "Dining Philosopher Problem" Synchronization Problem.

[Apr.2012 – 4M] 8. What is Critical Section Problem? Which requirement must be satisfied by solution to the Critical Section?

[Oct.11,Apr.11 – 4M] 9. Write a short note on Semaphores.

[Apr.2011 – 4M] 10. Explain bounded buffer problem in detail with the help of suitable example.

VISION

Chapter 6

DEADLOCKS

1. Introduction

In a multiprogramming environment several processes may compete for a finite number of resources. A process requests resources if the resources are not available at that time, the process enters a waiting state. It may happen that waiting processes will never again change state, because the resources they have requested are held by other waiting processes. This unfortunate situation is called a deadlock.

Deadlock can be defined formally as follows:

'*A set of processes is deadlocked if each process in the set is waiting for an event that only other process in the set can cause*'.

3

Oct.15,Apr.15 – *2M*
What is Deadlock?
Oct.2011 – *2M*
What is meant by Deadlock?

2. System Model

A system consists of a finite number of resources to be distributed among a number of competing processes. The resources are partitioned into several types, each of which consists of some number of identical instances.

Under the normal mode of operation, a process may utilize a resource in only of the following sequence:

1. **Request:** If the request cannot be granted immediately.

   For example, if the resource is a printer, the process can acquire the resource.

2. **Use:** The process can operate on the resource.

   For example, if the resource is a printer, the process can be print on the printer.

3. **Release:** The process releases the resources.

A process must request a resource before using it and release the resource after using it. A process may request as many resources as required to carry out the designated task.

A set of process is in a deadlock state, when every process in the set is waiting for an event that can only be caused by another in the set.

3. Deadlock Characterization

A deadlock situation can arise if and only if the following conditions hold simultaneously in a system.

3.1 Necessary Conditions for Deadlock

Coffman (1971) showed that four conditions must hold for there to be a deadlock.

1. **Mutual exclusion:** At least one resource is held in a non-sharable mode, that is only one process at a time can use the resource.

2. **Hold and wait:** There must exist a process that is holding atleast one resource and is waiting to acquire additional resources that are currently being held by another process.

3. **No preemption:** Resources cannot be preempted, i.e., resource can only be released voluntarily by the person holding it, after the process has completed its task.

4
Oct.15,12,11,Apr.11–*4M*
What are the necessary conditions for Deadlock Occurrence?

4. **Circular wait:** There exists a set (p_0, p_1, . . ., p_n) of waiting processes such that p_0 is held by p_1. p_{n-1} is waiting for resources which are held by p_n and p_n is waiting for a resource held by p_0. Thus, there must be a circular chain of two or more processes, each of which is waiting for a resource held by the next number of the chain.

All of these conditions must be present for a deadlock to occur. If one of them is absent, no deadlock is possible.

3
Oct.2015 – *4M*
Write a short note on resource allocation graph.
Apr.2015 – *4M*
Explain-Resource-Allocation graph in detail.
Oct.2011 – *4M*
Explain the Resource Allocation Graph with suitable example.

4. Resource Allocation Graphs

i. It consists of the sets P, R and E
ii. Resource instances
iii. Process states

V is partitioned into two types:

i. P = {P_1, P_2,….., P_n}, the set consisting of all the processes in the system.
ii. R = {R_1, R_2, ….., R_n}, the set consisting of all the resources types in the system.
iii. **Request edge:** A request edge $P_i \rightarrow R_i$ in the resource allocation graph indicates that process P_i may request resource R_j at some time in the future. A request edge is represented by a dashed line.

1
Oct.2012 – *2M*
What do you mean by Request Edge?

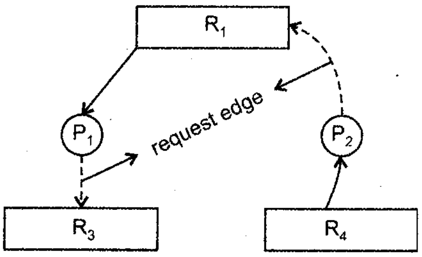


Figure 6.1

3
Oct.2014 – *2M*
Define claim edge in Resource Allocation graph.

iv. **Assignment edge:** directed edge $R_j \rightarrow P_i$

A set of vertices V and a set of edges E.

Process for drawing Resource Graph: Resource type with 4 instances P_i requests instance of R_j.

P_i is holding an instance of R_j

P_i
P_i
R_j
R_j

Examples,

i. Example of a resource allocation graph

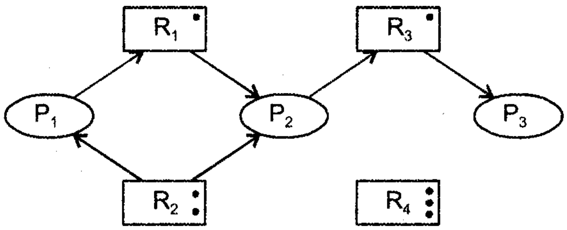


Figure 6.2

ii. Resource allocation graph with a deadlock

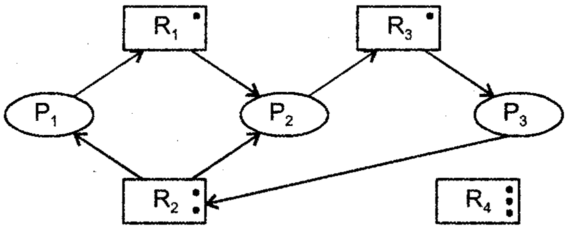


Figure 6.3

Basic Facts

i. If graph contains no cycles → no deadlock

ii. If a graph contains a cycle → deadlock

iii. If only one instance per resource type, then deadlock

iv. If several instances per resource type, possibility of deadlock

5. Safe State

1

Apr.2013 - *2M*
Define Safe Sequence.

A state is safe if the system can allocate resources to each process (up to its maximum) in some order and still avoid deadlock.

Formally, we can say, a system is in a safe state only if there exist a safe sequence.

A sequence of processes $<P_1, P_2, \ldots, P_n>$ is a safe sequence for the current allocation state, if for each P_i, the resources that P_i can still request can be satisfied by the currently available resources and the resources held by all the P_j, with $j < i$.

6. Deadlock Prevention

In general, four strategies are used for dealing with deadlocks

i. Just ignore the problem all together.

ii. Deadlock detection and recovery.

iii. Dynamic avoidance by careful resource allocation.

iv. Deadlock prevention, by structurally negating one of the four necessary conditions.

6.1 Just Ignore the Problem all together

Ostrich Algorithm

This is the simplest algorithm. The idea is as follows:

Stick your head in the sand and pretend there is no problem at all.

(*For example,* Unix operating system uses this algorithm)

6.2 Deadlock Detection

1
Apr.2013 – *2M*
Explain Deadlock Detection in detail.

In this technique, the system does not attempt to prevent deadlocks from occurring, instead it lets them occur, tries to detect when this happens, and then takes some action to recover after the fact.

Deadlock Detection with one resource of each type

Resource graph, In this case, we assume that system might have only one resource of each type.

For such system, we can construct a resource graph. If the graph contains one or more cycles, a deadlock exists. Any process that is part of the cycle is deadlocked. If no cycles exist, the system is not deadlocked.

Consider the system with seven processes. A through G and six resources, R through W.

Resource ownership is as follows:

i. Process A holds R and wants S
ii. Process B wants T
iii. Process C wants S
iv. Process D holds U and wants S and T
v. Process E holds T and wants V
vi. Process F holds W and wants S
vii. Process G holds V and wants U.

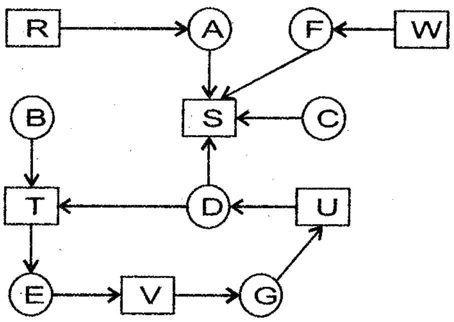


Figure 6.4: Resource graph

After detecting a deadlock some way is needed to recover and get the system going again.

6.3 Recovery from Deadlock

1

Apr.2015 – *4M*
Explain different methods for recovery from a deadlock.

1. **Recovery through preemption:** In some cases it may be possible to temporarily take some resource away from a process and give it to another process.

   In many cases, manual intervention may be required. The ability to take a resource away from a process, have another process use it, and then give it back without the process noticing is highly dependent on the nature of the resource. Recovering this way is frequently difficult or impossible.

2. **Recovery through rollback:** System designers can arrange to have processes check pointed periodically. Check pointing a process means that its state is written to a file, so that it can be restarted later, along with the resource state, that is which processes are currently assigned to the process.

   When a deadlock is detected, it is easy to see which resources are needed. To do the recovery, a process that runs a needed resource is rolled back to a point in time before it acquired some other resource by starting one of its earlier check points. In effect, the process is reset to an earlier moment when it did not have the resource, which is now assigned to one of the deadlocked processes.

   If the restarted process tries to acquire the resource again, it will have to wait until it becomes available.

3. **Recovery through killing processes:** The crudest, but simplest way to break a deadlock is to kill one or more processes. One possibility is to kill a process in the cycle with a little luck, the other processes will be able to continue. If this does not help, it can be repeated until the cycle is broken.

3

Oct.2014 – *4M*
Explain deadlock prevention strategies

Oct.2012 – *4M*
Explain Deadlock Prevention in detail

Apr.2012 – *4M*
What is Deadlock Prevention? Explain Deadlock Prevention Strategies.

6.4 Deadlock Prevention

If we can ensure that at least one of the four necessary conditions for deadlock is never satisfied, then deadlocks will be structurally impossible.

1. **Attacking the mutual exclusion condition:** If no resource was ever assigned exclusively to a single process, we would never have a deadlock.

   By spooling printer output, we can eliminate deadlock for the printer. Unfortunately, this cannot be applicable for some other resources.

2. **Attacking the hold and wait condition:** One way to achieve this goal is to require all processes to request all their resources before starting execution. If everything is available, the process will be allocated whatever it needs and can run to completion. If one or more resources are busy, nothing will be allocated and the process would just wait.

   An immediate problem with this approach is that many processes do not know how many resources there will be until they have started running.

   Another problem is that resources may not be used optimally.

   Some main frame batch systems require the user to list all the resources on the first line of each job.

   The system then acquires all resources immediately, and keeps them until the job finishes while this method puts a burden on the programmer and wastes resources, it does prevent deadlocks.

   A slightly different way is to require a process requesting a resource to first temporarily release all the resources it currently holds and then it tries it get everything it needs all at once.

3. **Attacking the no-preemption condition:** Attacking this condition is not feasible.

4. **Attacking the circular wait condition:** One way is simply to have a rule saying that a process is entitled only to a single resource at any moment if it needs a second resource, it must release the first resource. Another way is to provide a global numbering of all the resources.

   Now the rule is, 'Processes can request resources whenever they want to, but all requests must be made in numerical order'.

   With this rule, the resource allocation graph can never have cycles. A minor variation is to drop the requirement that resources be acquired in strictly increasing sequence and merely insist that no process should request a resource lower than what it is already holding.

6.5 Deadlock Avoidance By Careful Allocation of Resource

The main algorithm for deadlock avoidance are based on the concept of **'safe states'**. A state is said to be safe if it is not deadlocked and there is a way to satisfy all requests currently pending by running the processes in same order.

An unsafe state is not a deadlocked state because the system can run for a while, Infact, one process can even complete. The difference between a safe state and an unsafe state is that from a safe state the system can guarantee that all processes will finish, whereas from an unsafe state, no such guarantee can be given.

7. Banker's Algorithm For a Single Resource

A scheduling algorithm that can avoid deadlocks is due to *Dijkstra* (1965) and is known as the Banker's Algorithm.

The Bankers algorithm is thus to consider each request as it occurs and see if granted to a safe state. If it does, the request is granted otherwise, it is postponed until later.

8. Banker's Algorithm for Multiple Resources

The algorithm for checking to see if a state is safe can now be stated as follows

1. Look for a row R, whose unmet resource needs are all smaller than or equal to A. If no such row exists, the system will eventually deadlock since no process can run to completion.
2. Assume the process of the row chosen requests all the resources it needs (which is guaranteed to be possible) and finishes. Mark that process as terminated and add all its resources to the A vector.
3. Repeat steps 1 and 2 until either all processes are marked terminated, in which case the ended state was safe, or until a deadlock occurs, in which case it was not.

The above algorithm uses three vectors E, P and A to denote existing, possessed and available respectively.
Although in theory the algorithm is wonderful, in practice it is essentially useless because processes rarely know in advance what their maximum resource needs will be. In addition, the number of processes is not fixed, but dynamically varying as new users login and logout. Furthermore, resources that were thought to be available can suddenly vanish.

9. Process Termination

In order to avoid deadlock by killing a process two methods can be used

1. **Kill all deadlocked process:** This type will surely break the deadlock but at a great expense, some of the processes may have computed for a long period of time and will have to be started again.

2. **Kill one process at a time until the deadlock cycle is eliminated:** In this method, it requires considerable overhead, since after each process is killed a deadlock detection algorithm must be invoked to determine if any process is still deadlocked.

   Even when we decide to kill some process, which process should be chosen, can be decided using one of the following criteria.

   i. Priority of the process
   ii. How many processes will be involved in rollback?
   iii. Resources the process needs in order to complete.
   iv. How many and what type of resources the process has used?
   v. How long the process has computed and how much longer the process will compute?

10. Resource Preemption

1
Apr.2012 – *4M*
Explain the term 'Selecting a Victim' in the Context of Deadlock Recovery.

1
Apr.2012 – *2M*
Define Rollback.

In order, to eliminate deadlocks by using resource preemption, we successively preempt some resources from process and give these resources to other process until the deadlock cycle is broken.

Following issues need to be addressed

1. **Selecting a victim:** Which resources and which processes are to be preempted? We must determine the order of preemption in order to minimize the lost.

2. **Roll back:** If we preempt a resource from a process what should be done with that process? It cannot continue with its normal execution if it is missing some needed resource.

   We must rollback the process to some safe state and restart it from that state. The simplest solution is a total rollback, about the process and restart it. However, it is more effective to rollback the process only, as far as necessary to break the deadlock. This method however requires the system to keep more information about the state of all the running processes.

3. **Starvation:** How do we ensure that starvation will not occur? i.e. how can we guarantee that resources will not always be preempted from same process.

Solved Examples

1. Consider the following snapshot of a system:

Process	Allocation	Max	Available
	A B C	A B C	A B C
P_0	2 3 2	9 7 5	3 3 2
P_1	4 0 0	5 2 2	
P_2	5 0 4	11 0 4	
P_3	4 3 3	4 4 4	
P_4	2 2 4	6 5 5	

1 Apr.2013 – 4M

Answer the following questions using Banker's Algorithm:

i. What is the content of Need Matrix?

ii. Is the system in a Safe State? If yes, give the safe sequence.

Solution

Given:

Process	Allocation	Max	Available
	A B C	A B C	A B C
P_0	2 3 2	9 7 5	3 3 2
P_1	4 0 0	5 2 2	
P_2	5 0 4	11 0 4	
P_3	4 3 3	4 4 4	
P_4	2 2 4	6 5 5	

i. Given 5 processes

$P = \{P_0, P_1, P_2, P_3, P_4\}$

Resources type = {A, B, C}

Allocation matrix and max as follows:

	Allocation	Max
	A B C	A B C
P_0	2 3 3	9 7 5
P_1	4 0 0	5 2 2
P_2	5 0 4	11 0 4
P_3	4 3 3	4 4 4
P_4	2 2 4	6 5 5

Total instances of each resource type is given {3, 14, 12}

∴ Available resources = Total instance – Allocation

= {3, 14, 12} – {3, 3, 2}

Available = {0, 11, 10}

Need matrix

Need[i][i] = Max[i] [j] – Allocation [i][j]

	A	B	C
P_0	6	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

ii.

a. Initialize 'Finish' = (F) for all processes

$\therefore$ finish [] = {F, F, F}

b. Work = available = {0, 11, 10}

for i = 0

If(finish i) = false &&(needi $\leq$ work)

$\therefore$ finish = F &&(6, 4, 3 $\leq$ 0,11,10)

= false

{P_0 cannot be granted}

$\therefore$ the system is not in a safe state.

1 Oct.2012 – 4M

2. **Consider the system with 5 process P = {P_0, P_1, P_2, P_3, P_4} and four resources type {A, B, C, D}. There are 3 instances of type A, 10 instances of type B, 15 instances of type C and 7 instances of type D.**

The allocation and Maximum demand matrix are as follows:

	Allocation					MAX			
	A	B	C	D		A	B	C	D
P_0	0	1	2	1	P_0	0	8	4	4
P_1	0	1	2	1	P_1	0	6	5	2
P_2	1	0	0	0	P_2	1	6	4	1
P_3	1	3	5	3	P_3	2	3	7	5
P_4	0	0	4	1	P_4	0	5	5	7

Answer the following question using Bankers Algorithm:

i. Is the system in a Safe State?

ii. If a request from process P_4 arrives for (0, 2, 0, 2) can it be granted.

Solution

Step 1: Total instances of each resource type is given {3, 10, 15, 7}

$\therefore$ Available resources = Total instance – Allocation

= {3, 10, 15, 7} – {2, 5, 13, 6}

= {1, 5, 2, 1}

Step 2: To check safe state

Calculate Need [i] [j] = Max [i][j] – Allocation [i][j]

$\Rightarrow$ Need

	A	B	C	D
P_0	0	7	2	3
P_1	0	5	3	1
P_2	0	6	4	1
P_3	1	0	2	2
P_4	0	5	1	6

Step 3: Initialise 'Finish' = (F) for all processes

$\therefore$ finish [] = {F, F, F, F, F}

Step 4: Work = available = {1, 5, 2, 1}

a. For i = 0

If (Finish [i] = False && need$_i$ $\le$ work)

$\therefore$ Finish = F && (0, 7, 2, 3 $\le$ 1, 5, 2, 1) = True.

$\therefore$ P_0 can be granted.

Releasing resources by P_0, work will update to

Work = Work + Allocation

$\therefore$ Work = {1, 5, 2, 1} + {0, 1, 2, 1}

Work = {1, 6, 4, 2}

Finish [0] = True

Safe Sequence = {P_0}

b. i = 1

If Finish$_i$ = F && need$_i$ $\le$ Work

= F && {0, 5, 3, 1 $\le$ 1, 6, 4, 2}

= True.

$\therefore$ P_1 is granted

Work = Work + Allocation$_1$

= {1, 6, 4, 2} + {0, 1, 2, 1}

Work = {1, 7, 6, 3}

Finish [1] = True

Safe Sequence = {P_0, P_1}

c. $i = 2$

If $Finish_2 = F$ && $need_2 \leq Work$

$Finish_2 = F$ && $\{0, 6, 4, 1 \leq 1, 7, 6, 3\}$ = True.

$\therefore$ P_2 is granted.

$Work = Work + Allocation_2$

$= \{1, 7, 6, 3,\} + \{1, 0, 0, 0\}$

$= \{2, 7, 6, 3\}$

Finish [2] = True

Safe Sequence = $\{P_0, P_1, P_2\}$

d. $i = 3$

If $Finish_3 = F$ && (need $\leq$ Work)

$Finish_3 = F$ && $\{1, 0, 2, 2 \leq 2, 7, 6, 3\} \rightarrow$ True.

$\therefore$ P_3 is granted.

$Work = Work + Allocation_3$

$= \{2, 7, 6, 3\} + \{1, 3, 5, 3\}$

$= \{3, 10, 11, 6\}$

Finish [3] = True

Safe Sequence = $\{P_0, P_1, P_2, P_3\}$

e. $i = 4$

If $Finish_4 = F$ && ($need_4 \leq Work$)

$= F$ && $\{0, 5, 1, 6 \leq 3, 10, 11, 6\}$

= True.

$\therefore$ P_4 is granted

$\therefore$ Work = Work + Allocation

$= \{3, 10, 11, 6\} + \{0, 0, 4, 1\}$

Work = $\{3, 10, 15, 7\}$

Finish [4] = True

Safe Sequence = $\{P_0, P_1, P_2, P_3, P_4\}$

We can conclude that the system is in safe state.

ii. If a request from process P_4 arrives for (0, 2, 0, 2) can the request be granted.

We check

a. $Request_i \le Need_i$

$(0, 2, 0, 2) \le (0, 5, 1, 6)$

= False

That means process P_4 request is not legal. It is asking for more resources it should demand for. So process P_4 with new request will **not be** granted.

3. Consider the system with 5 process $P = \{P_0, P_1, P_2, P_3, P_4\}$ and four resources type {A, B, C, D}. There are 3 instances of type A, 14 instances of type B, 12 instances of type C and 12 instance of type D.

1 Apr.2012 – 4M

The allocation and Maximum demand matrix are as follows:

	Allocation					Max			
	A	B	C	D		A	B	C	D
P_0	0	6	3	2	P_0	0	6	5	2
P_1	0	0	1	2	P_1	0	0	1	2
P_2	1	0	0	0	P_2	1	7	5	0
P_3	1	3	5	4	P_3	2	3	5	6
P_4	0	0	1	4	P_4	0	6	5	6

Answer the following question using Bankers Algorithm:

i. Is the system in a Safe State?

ii. If a request from process P_4 arrives for (0, 0, 4, 1) can be the request be immediately granted.

Solution

Given: 5 Process

$P = \{P_0, P_1, P_2, P_3, P_4\}$

Resources of type {A, B, C, D}

Allocation Matrix and Max as follows:

	Allocation					Max			
	A	B	C	D		A	B	C	D
P_0	0	6	3	2	P_0	0	6	5	2
P_1	0	0	1	2	P_1	0	0	1	2
P_2	1	0	0	0	P_2	1	7	5	0
P_3	1	3	5	4	P_3	2	3	5	6
P_4	0	0	1	4	P_4	0	6	5	6

i. Is system in safe state.

⇒ Total instances of each resource type is given {3, 14, 12, 12}

$\therefore$ Available resources = Total instance – Allocation
= {3, 14, 12, 12} – {2, 9, 10, 12}
= {1, 5, 2, 0}

To check the safe state,

Calculate Need [i] [j] = Max [i][j] – Allocation [i][j]

$\Rightarrow$ Need

	A	B	C	D
P_0	0	0	2	0
P_1	0	0	0	0
P_2	0	7	5	0
P_3	1	0	0	2
P_4	0	6	4	2

1. Initialise 'Finish' = (F) for all processes
   $\therefore$ finish [] = {F, F, F, F, F}
2. Work = available = {1, 5, 2, 0}
   a. For i = 0
      If (Finish [i] = False && need$_i$ $\leq$ work)
      $\therefore$ Finish = F && (0, 0, 2, 0 $\leq$ 1, 5, 2, 0) = True
      $\therefore$ P_0 can be granted.
      Releasing resources by P_0, work will update to
      Work = Work + Allocation$_0$
      $\therefore$ Work = {1, 5, 2, 0} + {0, 6, 3, 2} = {1, 11, 5, 2}
      Finish [0] = True
      Safe Sequence = {P_0}
   b. i = 1
      If Finish$_i$ = F && need$_i$ $\leq$ Work
      Finish$_i$ = F && {0, 0, 0, 0} $\leq$ {1, 11, 5, 2} is True.
      $\therefore$ P_1 is granted
      Work = Work + Allocation$_1$
      = {1, 11, 5, 2} + {0, 0, 1, 2} = {1, 11, 6, 4}
      Finish [1] = True
      Safe Sequence = {P_0, P_1}
   c. i = 2
      If Finish$_2$ = F && need$_2$ $\leq$ Work
      Finish$_2$ = F && {0, 7, 5, 0} $\leq$ {1, 11, 6, 4} $\rightarrow$ True.

$\therefore$ P_2 is granted.

Work = Work + Allocation_2

= {1, 11, 6, 4,} + {1, 0, 0, 0} = {2, 11, 6, 4}

Finish_2 = {T}

Safe Sequence = {P_0, P_1, P_2}

d. i = 3

If Finish_3 = F && $\text{need}_3 \leq$ Work

Finish_3 = F && {1, 0, 0, 2} $\leq$ {2, 11, 6, 4} $\rightarrow$True.

$\therefore$ P_3 is granted.

Work = Work + Allocation_3

= {2, 11, 6, 4} + {1, 3, 5, 4} = {3, 14, 11, 8}

$\text{Finish}_{[3]}$ = {T}

Safe Sequence = {P_0, P_1, P_2, P_3}

e. i = 4

If Finish_4 = F && $\text{need}_4 \leq$ Work

Finish_4 = F && {0, 6, 4, 2} $\leq$ {3, 14, 11, 8} $\rightarrow$ True.

$\therefore$ P_4 is granted

Work = Work + Allocation_4

= {3, 14, 11, 8} + {0, 0, 1, 4} = {3, 14, 12, 12}

$\text{Finish}_{[4]}$ = {T}

Safe Sequence = {P_0, P_1, P_2, P_3, P_4}

So Finish [] = {T, T, T, T, T}

And Safe Sequence is {P_0, P_1, P_2, P_3, P_4}

We can conclude that the system is in safe state.

ii. If a request from process P_4 arrives for (0, 0, 4, 1) can the request be immediately granted.

Check

$\text{Request}_i \leq \text{Need}_i$

(0, 0, 4, 1) $\leq$ (0, 6, 4, 2) $\rightarrow$ False

That means process P_4 request is not legal. It is asking for more resources it should demand for. So process P_4 with new request will **not be** granted.

SUMMARY

- A set of processes is deadlocked if each process in the set is waiting for an event that only other process in the set can cause.
- There are four necessary conditions for deadlock:
  a. Mutual exclusion condition b. Hold and wait condition
  c. No preemption condition d. Circular wait condition
- The strategies used for dealing with the deadlock:
  a. Just ignore the problem together
  b. Deadlock detection and recovery
  c. Dynamic avoidance by careful resource allocation
  d. Deadlock prevention by structurally negating one of the four necessary conditions
- A state is said to be safe if it is not deadlocked and there is a way to satisfy all requests currently pending by running the process in same order.
- To eliminate deadlocks by using resource preemption, we successively preempt some resources from process and give these resources to other process until the deadlock cycle is broken. Following issues need to be addressed:
  a. Selecting a victim

- A set of processes is deadlocked if each process in the set is waiting for an event that only other process in the set can cause.
- There are four necessary conditions for deadlock:
  a. Mutual exclusion condition b. Hold and wait condition
  c. No preemption condition d. Circular wait condition
- The strategies used for dealing with the deadlock:
  a. Just ignore the problem together
  b. Deadlock detection and recovery
  c. Dynamic avoidance by careful resource allocation
  d. Deadlock prevention by structurally negating one of the four necessary conditions
- A state is said to be safe if it is not deadlocked and there is a way to satisfy all requests currently pending by running the process in same order.

- To eliminate deadlocks by using resource preemption, we successively preempt some resources from process and give these resources to other process until the deadlock cycle is broken. Following issues need to be addressed:
  a. Selecting a victim b. Rollback c. Starvation.

PU Questions

2 Marks

1. What is Deadlock? **[Oct.15, Apr.15 – *2M*]**
2. Define claim edge in Resource Allocation graph. **[Oct.2014 – *2M*]**
3. Define Safe Sequence. **[Apr.2013 – *2M*]**
4. What do you mean by Request Edge? **[Oct.2012 – *2M*]**
5. Define Rollback. **[Apr.2012 – *2M*]**
6. What is meant by Deadlock? **[Oct.2011 – *2M*]**

4 Marks

1. Write a short note on resource allocation graph. **[Oct.2015 – 4M]**
2. What are the necessary conditions for deadlock occurrence? **[Oct.15,12,11,Apr.11–4M]**
3. Consider the five processes P_0, P_1, P_2, P_3, P_4 and three resources R_1, R_2, R_3 resources. Type R_1 has 8 instances, R_2 has 4 instances and R_3 has 9 instances. Allocation and maximum matrix is given below: **[Oct.2015 – 4M]**

	Allocation			Maximum		
	R_1	R_2	R_3	R_1	R_2	R_3
P_0	1	0	2	5	4	2
P_1	2	1	1	3	2	2
P_2	2	0	3	8	0	4
P_3	1	1	2	2	2	2
P_4	0	1	0	5	2	3

Answer the following questions using Banker's Algorithm:
i. What is the content of need matrix.
ii. Is the system in a safe sequence? If yes, give the safe sequence.

4. Explain different methods for recovery from a deadlock. **[Apr.2015 – 4M]**
5. Explain-Resource-Allocation graph in detail. **[Apr.2015 – 4M]**
6. Consider the five processes P_0, P_1, P_2, P_3, P_4 and three resources R_1, R_2, R_3 resources type R_1 has 10 instances, R_2 has 5 instances and R_3 has 7 instances. Allocation and max matrix is given below: **[Apr.2015 – 4M]**

	Allocation			MAX		
	R_1	R_2	R_3	R_1	R_2	R_3
P_0	0	1	0	7	5	3
P_1	2	0	0	3	2	2
P_2	3	0	2	9	0	2
P_3	2	1	1	2	2	2
P_4	0	0	2	4	3	3

Answer the following questions using Banker's Algorithm:
i. What is the content of Need Matrix.
ii. Is the system in a safe sequence? If yes, give the safe sequence.

7. Explain deadlock prevention strategies. **[Oct.2014 – 4M]**
8. Consider the following snapshot of system. A system has 5 processes p1 through p5 and four resources type A through D. **[Oct.2014 – 4M]**

	Allocation				MAX			
	A	B	C	D	A	B	C	D
P_1	0	0	1	2	0	0	1	2
P_2	1	0	0	0	1	7	5	0
P_3	1	3	5	4	2	3	5	6
P_4	0	6	3	2	0	6	5	2
P_5	0	0	1	4	0	6	5	6

Available			
A	B	C	D
1	5	2	0

Answer the following question using Banker's Algorithm.
i. What are the content of matrix need.
ii. Is the system in safe states? Give the safe sequence

[Apr.2013 – *4M*] 9. Consider the following snapshot of system. A system has 5 processes p1 through p5 and four resources type A through D.

Process	Allocation	Max	Available
	A B C	A B C	A B C
P_0	2 3 2	9 7 5	3 3 2
P_1	4 0 0	5 2 2	
P_2	5 0 4	1 1 0 4	
P_3	4 3 3	4 4 4	
P_4	2 2 4	6 5 5	

Answer the following questions using Banker's Algorithm:

i. What is the content of Need Matrix?

ii. Is the system in a Safe State? If yes, give the safe sequence.

[Apr.2013 – *4M*] 10. Explain Deadlock Detection in detail.

[Oct.2012 – *4M*] 11. Explain Deadlock Prevention in detail.

[Oct.2012 – *4M*] 12. Consider the system with 5 process P = {P_0, P_1, P_2, P_3, P_4} and four resources type {A, B, C, D}. There are 3 instances of type A, 10 instances of type B, 15 instances of type C and 7 instances of type D.

	Allocation			
	A	B	C	D
P_0	0	1	2	1
P_1	0	1	2	1
P_2	1	0	0	0
P_3	1	3	5	3
P_4	0	0	4	1

	MAX			
	A	B	C	D
P_0	0	8	4	4
P_1	0	6	5	2
P_2	1	6	4	1
P_3	2	3	7	5
P_4	0	5	5	7

Answer the following question using Bankers Algorithm:

i. Is the system in a Safe State?

ii. If a request from process P_4 arrives for (0, 2, 0, 2) can it be granted.

[Apr.2012 – *4M*] 13. What is Deadlock Prevention? Explain Deadlock Prevention Strategies.

[Apr.2012 – *4M*] 14. Explain the term 'Selecting a Victim' in the Context of Deadlock Recovery.

Chapter 7

MEMORY MANAGEMENT

1. Introduction

The main purpose of a computer system is to execute programs. These programs, together with the data they access, must be atleast partially in main memory during execution.

The part of the operating system that manages memory is called the memory manager. Its job is to keep track of which parts of memory are in use and which part are not in use, to allocate memory to processes when they need it and deallocate it when they are done, and manage swapping between main memory and disk when main memory is not big enough to hold all the processes.

1.1 Address Binding

The binding of instructions and data to memory address is known as address binding.

A program resides in the disk in binary executable form. When it is brought in main memory, it is called as a process.

At a given time, there can be more than one processes in the main memory waiting to be executed. There processes form a input queue.

A user program has to go through several steps during execution (*figure 7.1*).

1
Oct. 2011 – *2M*
What is meant by Address Binding?

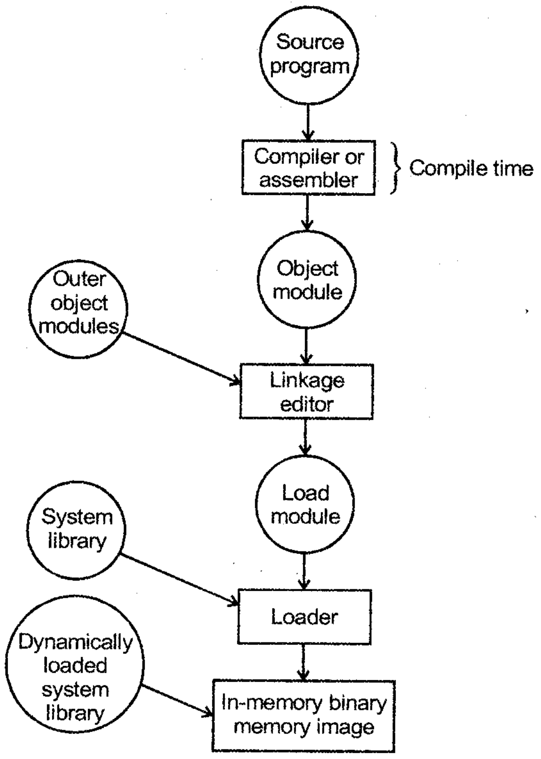


Figure 7.1: Multi-step processing of user program

Classically the binding of instructions and data to memory addresses can be done at any step along the way.

1. **Compile time binding:** If it is known at compile time where the process will reside in memory then absolute code can be generated.

   Example, MS-DOS, .Com format programs are absolute code bound at compile time.

2. **Load time binding:** If it is not known at compile time where the process will reside in memory, then the compiler must generate relocatable code in this case final binding is delayed until load time.

   Example, Relocatable programs have load time binding.

3. **Run-time binding:** If the process can be moved during its execution from one memory segment to another, then binding must be delayed until run-time special hardware must be delayed until run time. Special hardware must be available for scheme to work.

   Example, In paging systems this binding is used. MMU is used to perform run-time mapping.

1.2 Logical Verses Physical Addresses

Oct.15, Apr.13 – *2M*
Define Logical Address.

The address generated by CPU is called **logical address**, whereas the address seen by memory unit i.e. one loaded in the memory register is called **physical address**.

Oct.14,12 – *2M*
Define Physical Address.

The compile time and load time binding methods generate identical, logical and physical addresses but execution time an address binding scheme results in different logical and physical addresses. Logical address are also called as virtual address. The set of all logical addresses of a program is its logical address space and the set of all physical addresses is a physical address space.

The run-time mapping of virtual to physical addresses is done by a hardware device called Memory Management Unit (MMU). It makes use of a register called as relocation or base register.

The value in the base register is added to every address generated by any user process. *For example:* If the Fence Register or Base Register is at 1400, then an attempt by the user to address location 0 is dynamically relocated to 1400. The user never sees the real address (*figure 7.2*).

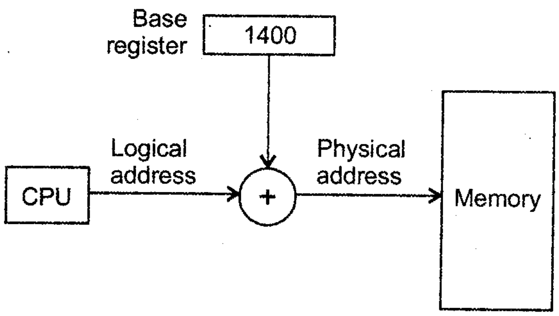


Figure 7.2: Dynamic relocation using base register

1.3 Static Linking

Static linking is the process of copying all library modules used in the program into the final executable image. This is performed by the linker and it is done as the last step of the compilation process. The linker combines library routines with the program code in order to resolve external references, and to generate an executable image suitable for loading into memory.

When the program is loaded, the operating system places into memory a single file that contains the executable code and data.

This statically linked file includes both the calling program and the called program.

Static linking is performed by programs called linkers as the last step in compiling a program. Linkers are also called link editors.

Statically linked files are significantly larger in size because external programs are built into the executable files.

In static linking if any of the external program has changed then they have to be recompiled and re-linked again else the changes won't reflect in existing executable file.

Statically linked program takes constant load time every time it is loaded into the memory for execution.

Programs that use statically-linked libraries are usually faster than those that use shared libraries.

In statically-linked programs, all code is contained in a single executable module. Therefore, they never run into compatibility issues.

1.4 Dynamic Loading

In our discussion so far, the entire program and data must reside in main memory. So the size of the program is restricted by the size of main memory. To obtain better memory utilization, we can use dynamic loading. Here a routine is not loaded until it is called. All routines are kept in disk in a relocatable format. The main program is loaded and executed, when a routine needs to call another routine, it first checks if the needed routine is loaded, if not, it calls the relocatable linking loader to load the required routine.

The advantage here is that an unused routine is never loaded. Dynamic loading does not require special support from the operating system. It is the responsibility of the user to design their programs to take advantage of such a method.

1.5 Dynamic Linking and Shared Libraries

Figure 7.1 also shows dynamic linked libraries. If the operating system supports only static linking, then the system libraries are also linked like all other object modules. The concept of dynamic linking is similar to that of dynamic loading. The linking of a program to system libraries is postponed upto execution. Here a stub is included in the image for each library routine reference. This stub is a small piece of code that indicates how to locate the appropriate memory resident library routine. When executed, the stub checks if the routine is already in memory, if not, it is loaded into memory. It replaces itself with the address of the routine and executes the routine.

Unlike dynamic loading, dynamic linking generally requires help from the operating system.

1.6 Overlays

2

Apr. 2011 – *2M*
What is the use of Overlays in Memory Management?
Oct. 2012 – *4M*
Write a short note on Overlays.

To enable a process to be larger than the amount of memory allocated to it, we can use overlays. The idea of overlays is to keep in memory only those instructions and data, that are needed at any given time. When other instructions are needed they are loaded into space occupied previously by instructions that are no longer needed.

For example: Consider a two pass assembler. This program can be roughly divided into following parts:

Pass 1	70 KB
Pass 2	80 KB
Symbol table	20 KB
Common routine	30 KB

If loaded completely the program would occupy 200 KB of memory, but using an overlay manager (size 10 KB), we could utilize the same memory space for pass 1 and pass 2 as shown in *figure 7.3*.

We can now run the assembler in 150 KB of memory.

1

Apr. 2015 – *4M*
Explain overlay's in detail with diagram.

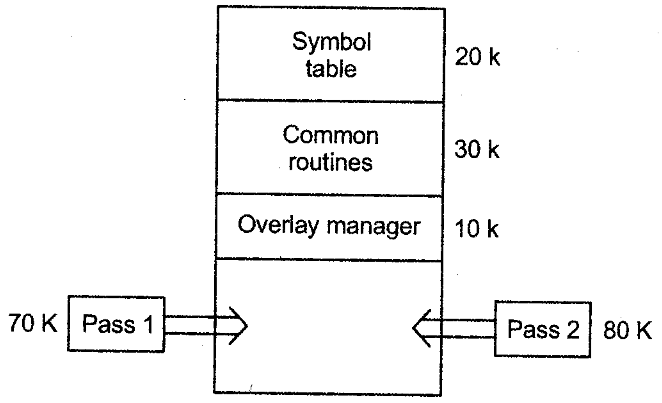


Figure 7.3: Overlays for 2 pass assembler

2. Swapping

2
Apr. 12,11 – *2M*
What is Swapping?

Moving processes from main memory to disk and back is called swapping. It is a simple memory / process management technique used by the operating system to increase the utilization of the process or by moving some blocked process from the main memory to the secondary memory (hard disk).

A process needs to be in memory for execution. However, it is possible that a process may be swapped temporarily out of memory to a backing store and then brought back into memory for continued execution.

For example: In a multiprogramming, environment if round robin CPU scheduling is used, then when the time slot of a process is over and the CPU jumps to the next process, then this process can be swapped to the backing store and brought back when again it will be allotted a time slot by the CPU (*figure 7.4*).

1
Apr. 2013 – *4M*
Explain Swapping of two processes using disk as a backing store with diagram.

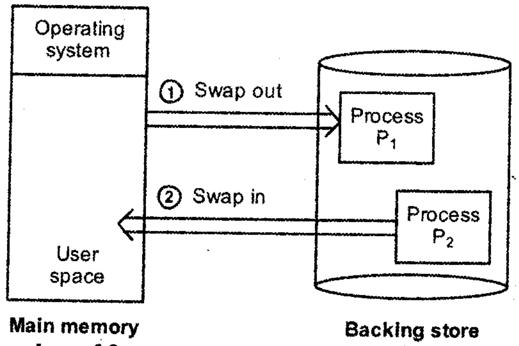


Figure 7.4: Swapping of 2 processes using a disk as a backing store

A variant of this swapping is used for priority based scheduling. If a higher priority process arrives and is to be executed, the scheduler swaps out the lower priority process and swaps in the higher priority process. This variant is often called roll-out, roll-in.

If for a process which is swapped out binding is done at assembly time or load time, then the process must be swapped in, in the same address space, but if the binding is in execution time, then the process can be swapped in, in any different addresses also.

The context switch time of swapping is fairly high. Swapping is also constrained by various factors like pending I/O. If a process has pending I/O and it is swapped out then it might create problems.

Currently, standard swapping is used in very few systems because it requires too much swapping time and provides too little execution time to be a good memory management solution.

A modification of swapping is used in many versions of UNIX. Swapping is normally kept disabled, but it would be started if many of the process are in main memory and all using threshold of memory.

The time taken by the swapper to swap a process that can be swapped temporarily out of main memory to a backing store, and then brought back into memory for continued execution is called swap time.

3. Contiguous Memory Allocation

3.1 Single Partition Allocation (Monoprogramming without Swapping or Paging)

The simplest possible memory management scheme is to have just 1 process in memory at a time and to allow that process to use cell of the memory. When the system is organized in this way, only process at a time can be

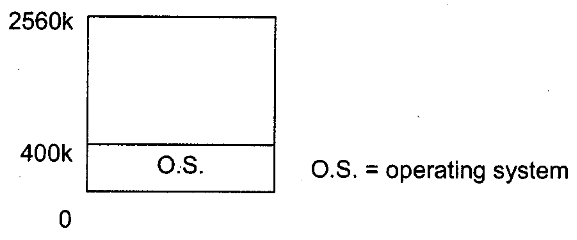


Figure 7.5: Memory layout for above system

3.2 Multiprogramming without Swapping or Paging i.e. Multiprogramming with Fixed Partition (MFT)

In this technique memory is divided up into n (possibly unequal) partition. When a job arrives it can be put into the job queue for the smallest partition, large enough to hold it. Since the partitions are fixed in this scheme any space in a partition not used by a job is lost. The disadvantage of sorting the incoming jobs into separate queues becomes apparent when the queue for large partition is empty and queue for small partition is full.

1
Oct. 2014 – *4M*
Explain MFT (Multiple Contiguous Fixed Partition allocation).

1
Apr. 2013 – *4M*
Explain in detail M.F.T. Job Scheduling.

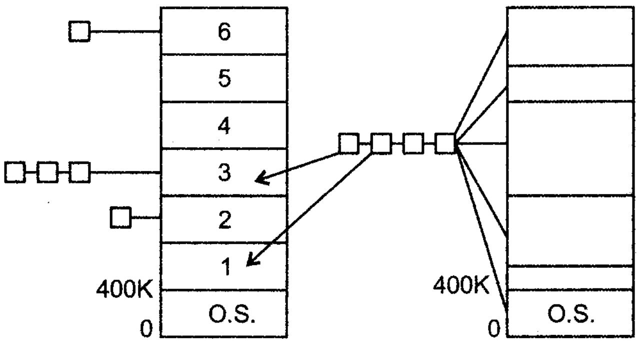
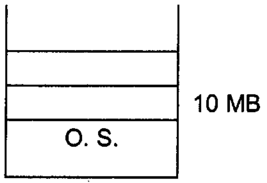


Figure 7.6: I/P queue for every partition

An alternate organization is to maintain a single queue.

Whenever partition becomes free the job closest to the front of queue that fits in it could be loaded into the empty partition and run. Since it is undesirable to waste a large partition on a small job a different strategy is to search the whole i/p queue, whenever the partition becomes free and pick the largest job that fits. Such scheme is used by OS/360 on large IBM mainframes called as MFT or OS/MFT.



3.3 Multiple Partition Allocation (Multiprogramming with Variable Partitions) (MVT)

In this scheme, the operating system keeps a table indicating which parts of memory are available and which are occupied.

Initially all memory is available for user processes and is considered as one large block of available memory, called a **Hole**. When a process arrives and needs memory, we search for a hole large enough for this process. If we find one, we allocate only as much memory as is needed, keeping the rest available to satisfy future requests.

1
Oct. 2012 – *4M*
Explain in detail M.V.T. Job Scheduling.

It is possible to combine all the Holes into one big Hole by moving all the processes downward as far as possible, this technique is known as **Memory Compaction**.

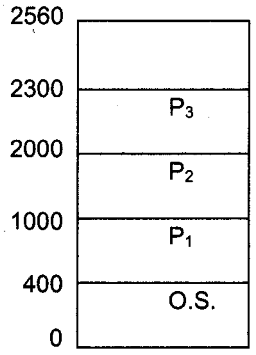
1
Apr. 2015 – *4M*
Write a note on memory compaction.

Example,

Process	Memory	Time
P_1	600 k	10
P_2	1000 k	5
P_3	300 k	20
P_4	700 k	8
P_5	500 k	15

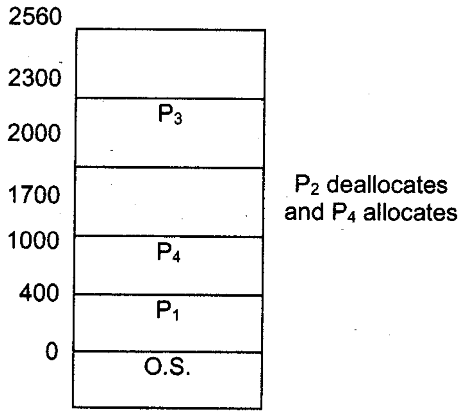
Show the execution (allocator) of these processes using MVT Scheduling alg (2560 k RAM).

Solution

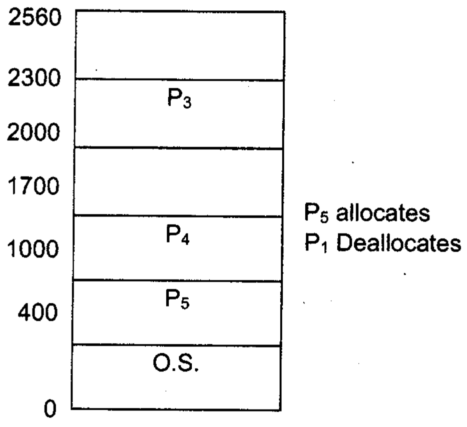


(At start)

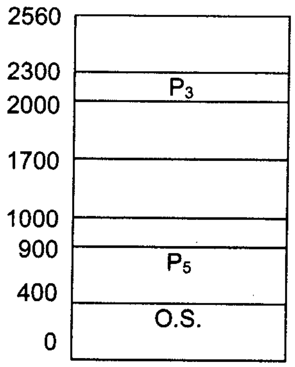
After 5 ms



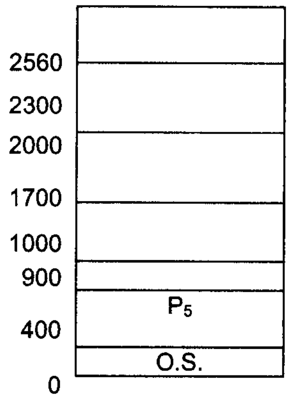
After 10 ms



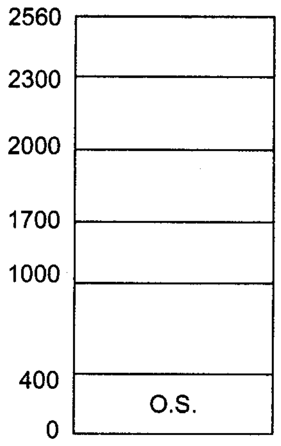
At 13 ms P4 deallocates



At 20 ms P_3 deallocates



Then,
At 25 ms P_5 deallocates



Apr. 15,12 – *4M*
Differentiate between MVT and MFT Job Scheduling.

Difference between MVT and MFT

	MVT	MFT
1.	MVT was considerably larger and more complex than MFT and therefore was used on the most powerful System 360 CPUs.	MFT was intended to serve a stop-gap until Multiprogramming with a Variable number of Tasks (MVT), the intended 'target' configuration of OS/360, became available in 1967.
2.	It treated all memory not used by the operating system as a single pool from which contiguous 'regions' could be allocate as required by an indefinite number of simultaneous application programs.	Early versions of MVT had many problems, so the simpler MFT continued to be used for many years. After introducing new system 370 machines with virtual memory, in 1972 MFT was developed into OS/VS$_1$, the last system of this particular line.

3.	This scheme was more flexible than MFT's and in principle used memory more efficiently, but was liable to fragmentation-after a while one could find that, although there was enough spare memory in total to run a program, it was divided into separate chunks none of which was large enough.	After introducing new System/370 machines with virtual memory, in 1972 MFT was developed into OS/VS$_1$, the last system of this particular line.
4.	In 1971 the Time Sharing Option (TSO) for use with MVT was added as part of release 2001. TSO became widely used for program development because it provided an editor. The ability to submit batch jobs, be notified of their completion, and view the results without waiting for printed reports, and debuggers for some of the programming languages used on System/360.	The first version of MFT shared much of the code and architecture with PCP, and was limited to four partitions. It was very cumbersome to run multiple partitions. Many installations used Houston Automatic Spooling Priority (HASP) to mitigate the complexity. MFT Version II (MFT-II) shared much more of the Control Program and Scheduler code with MVT, and was much more flexible to run. The maximum number of partitions increased to 52.
5.	Multiprocessing with a variable number of tasks.	Multiprocessing with a fixed number of tasks.
6.	Both the number and size of the partitions change with time.	Both the number and size of the partitions are fixed.
7.	Introduces external fragmentation, i.e., holes outside any region.	Introduces Internal fragmentation.
8.	There is dynamic address translation (during run time).	No dynamic address translation.

3.4 Fragmentation

1

Apr. 2011 – *2M*
What is meant by Fragmentation?

The segments of a program can be stored anywhere in the memory, but each segment has to be stored in a continuous memory.

Segmentation may suffer from external fragmentation as it is possible that there is free memory but is not contiguous to be allocated to the next segment.

1

Oct. 2011 – *4M*
What is Fragmentation? Compare Internal and External Fragmentation.

Internal and External Fragmentation

As processes are loaded and removed from memory the free memory space is broken into little pieces called **Holes**.

External fragmentation exists when enough total memory space exists to satisfy a request, but it is not contiguous, storage is fragmented into a large number of small holes.

Memory that is internal to a partition, but is not being used is called internal fragmentation.

2

Apr. 2015 – *2M*
What is External fragmentation?
Oct. 2014 – *2M*
What is Internal fragmentation?

MFT: Internal fragmentation

MVT: External fragmentation

Paging: Internal fragmentation

Segmentation: External fragmentation

One solution to the problem of external fragmentation is compaction.

The goal is to shuffle the memory contents to place all free memory together in one large block.

1

Oct. 2015 – *4M*
Differentiate between internal fragmentation and external fragmentation.

4. Free Space Management Techniques

The set of holes is searched to determine which hole is best to allocate.

1

Apr.2013 – *2M*
List various techniques of free Space Management in File System.

1. **First fit algorithm:** The memory manager finds a 1st hole that is big enough, the hole is then broken up into 2 pieces, one for the process and one for the unused memory. It is the fastest algorithm.

2. **Next fit algorithm:** It is similar to 1st fit, except that it keeps track of where it finds a suitable Hole. The next time it is called, it starts from where it left off. It has slightly worse performance than 1st fit.

3. **Best fit algorithm:** It searches the entire list and takes the smallest hole that is adequate.

   It is slower than above two algorithm. It also results in more wasted memory than above two algorithms.

4. **Worst fit algorithm:** It always takes the largest available hole so it is best.

5. **Yet another allocation algorithm:** It is Quick fit, which maintains separate lists for some of the common sizes requested.

5. Allocation Swap of Space

In some systems, when a process is in memory, no disk space is allocated to it. When it must be swapped out, space must be allocated in the disk swap area for it.

On each swap, it may be placed somewhere else on the disk. The algorithms for managing swap space are the same. Ones used for managing main memory.

In other systems, when a process is created, swap space is allocated for it on the disk using above algorithms. Whenever a process is swapped out, it is always swapped to its allocated space, rather than going to a different place each time.

When the process exits, the swap space is deallocated.

6. Virtual Memory (Overlays)

In early days of computing, IT industry was facing a problem that some programs were too big to fit in the available memory.

The solution usually adopted was to split the program into pieces called overlays. Overlays would start running first.

When it was done it would call another overlay.

Although the actual work of swapping overlays in and out was done by the system, the work of splitting the program into pieces had to be done by the programmer.

Splitting up large programs into small, modular pieces was time consuming and boring.

The alternate method that was deviced has come to be known as virtual memory (*Father Ingham*, 1961).

The basic idea behind it is that the combined size of the program, data and stack (i.e. the three segments code, data and stack) may exceed the amount of physical memory available for it.

The operating system keeps those parts of the program currently in use in main memory and rest on disk.

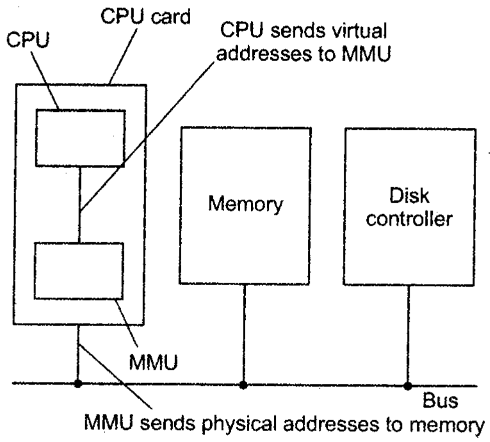
Virtual memory and multiprogramming fit together very well. While a program is waiting for part of itself to be swapped in, it is waiting for I/O and cannot run, so the CPU is given to another process.

7. Paging

When a program uses an instruction like

MOVE REG 1000

On a computer which uses virtual memory, these virtual addresses do not go directly to the memory bus. Instead, they go to a Memory Management Unit (MMU), a chip or collection of chips that maps the virtual addresses onto the physical memory addresses.



CPU card
CPU
CPU sends virtual addresses to MMU
Memory
Disk controller
MMU
Bus
MMU sends physical addresses to memory

Figure 7.7

The virtual address space is divided up into units called pages.

The corresponding units in the physical memory are called **page frames**. The pages and page frames are always of the same size.

Page sizes from 512 bytes to 8 k are commonly used with 64 k of virtual address space and 32 k of physical memory, we have 16 virtual pages and 8 page frames. Transfer unit is page virtual address.

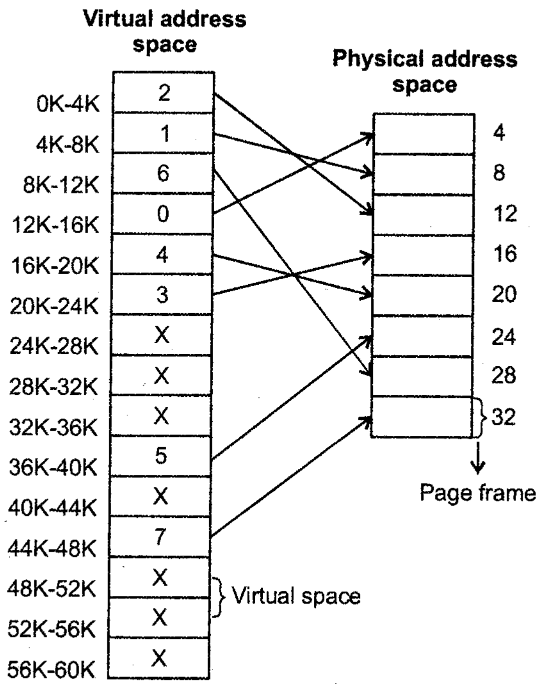


Figure 7.8

Unmapped virtual pages are shown by 'X' in the above *figure*. In the actual hardware, a present/absent bit in each entry of virtual page table keeps track of whether the page is mapped or not.

If the program tries to use an unmapped page, the MMU notices that the page is unmapped and causes the CPU to trap to the operating system. This trap is called a page fault. The operating system picks a little used page frame and writes its contents back to the disk. It then fetches the page just referenced into the page frame just freed, changes the map, and restarts the trapped instruction.

2
Apr. 15,12 – *4M*
What is Page Fault? Explain the different steps in Handling a Page Fault.

Page Fault

In virtual Memory management, when we use demand paging concept, i.e., instead of loading entire program into memory to execute only needed pages are loaded whenever any page is demanded then only that page is loaded into memory. When programs try to access a page which is **not** in the memory, **page fault** occur. For every page loaded at least once there will be at least one page fault.

Actions taken by O.S. in handling a page fault:

i. O.S. checks reference bit of that page if it is legal or not.

7. Paging

When a program uses an instruction like

MOVE REG 1000

On a computer which uses virtual memory, these virtual addresses do not go directly to the memory bus. Instead, they go to a Memory Management Unit (MMU), a chip or collection of chips that maps the virtual addresses onto the physical memory addresses.

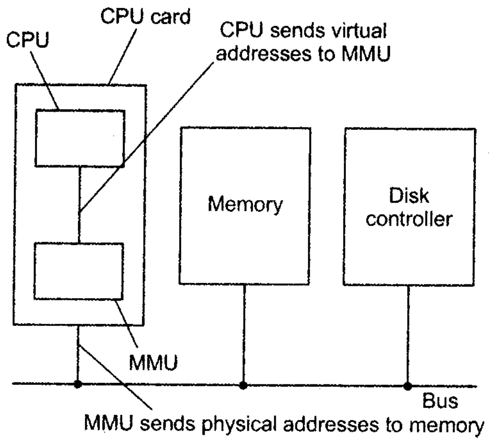

CPU card
CPU
CPU sends virtual addresses to MMU
Memory
Disk controller
MMU
Bus
MMU sends physical addresses to memory

Figure 7.7

The virtual address space is divided up into units called pages.

The corresponding units in the physical memory are called **page frames**. The pages and page frames are always of the same size.

Page sizes from 512 bytes to 8 k are commonly used with 64 k of virtual address space and 32 k of physical memory, we have 16 virtual pages and 8 page frames. Transfer unit is page virtual address.

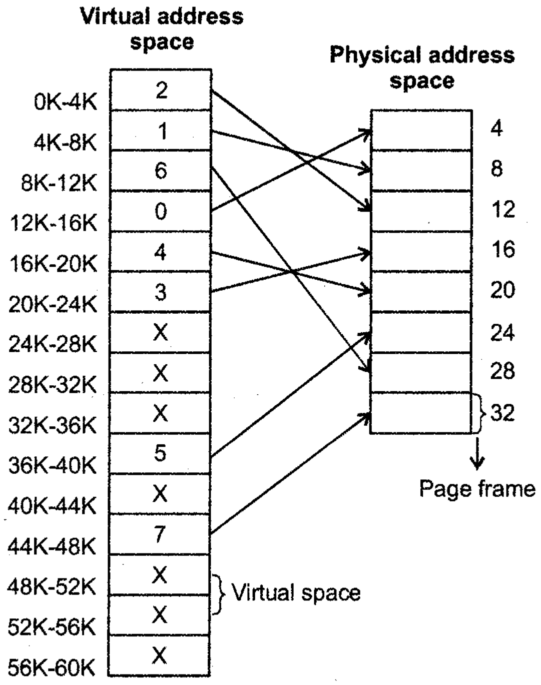


Figure 7.8

Unmapped virtual pages are shown by 'X' in the above *figure*. In the actual hardware, a present/ absent bit in each entry of virtual page table keeps track of whether the page is mapped or not.

If the program tries to use an unmapped page, the MMU notices that the page is unmapped and causes the CPU to trap to the operating system. This trap is called a page fault. The operating system picks a little used page frame and writes its contents back to the disk. It then fetches the page just referenced into the page frame just freed, changes the map, and restarts the trapped instruction.

2
Apr. 15,12 – *4M*
What is Page Fault? Explain the different steps in Handling a Page Fault.

Page Fault

In virtual Memory management, when we use demand paging concept, i.e., instead of loading entire program into memory to execute only needed pages are loaded whenever any page is demanded then only that page is loaded into memory. When programs try to access a page which is **not** in the memory, **page fault** occur. For every page loaded at least once there will be at least one page fault.

Actions taken by O.S. in handling a page fault:

i. O.S. checks reference bit of that page if it is legal or not.

ii. If bit is invalid process is terminated.

iii. If bit is valid then we now have to bring page in.

iv. Check if there is any frame free to swap page in. If frame is available page is swapped in.

v. If frame is not available, make frame free with available page-replacement algorithm and then swap require page in.

7.1 Internal Operation of MMU

The incoming 16-bit vertical address is split up into a 4- bit page number and a 12-bit offset. A page number is used as an index into the page table, yielding the number of the page frame corresponding to that virtual page.

If the present/ absent bit is zero, a trap to the operating system is caused. If the bit is one, the page frame number is found in the page table.

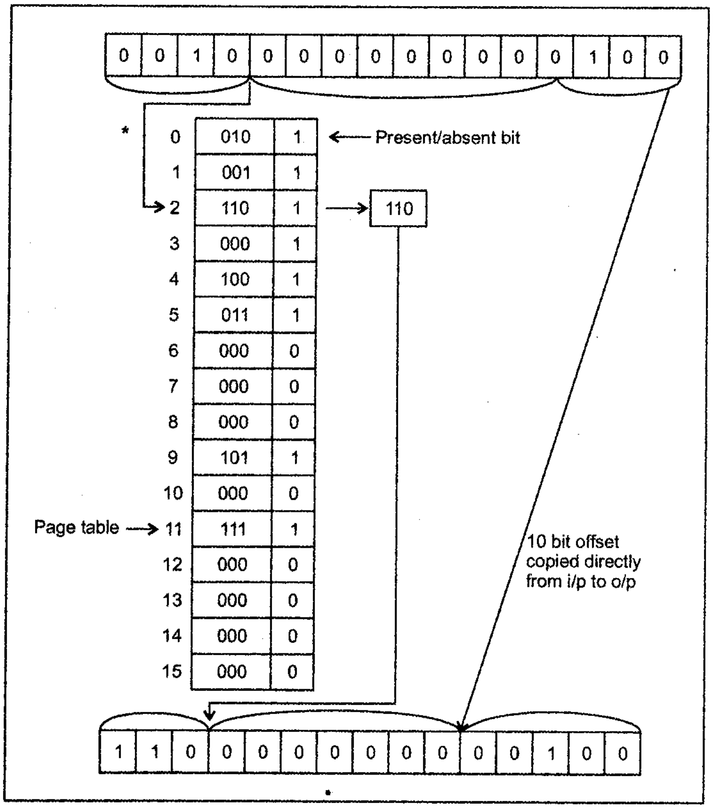


Figure 7.9: Internal Operation of the MMU with 16 4k pages

* Virtual page = 2 is used as an index into the page table.
▲ Incoming virtual address 8196
● Outgoing physical address 24580

Is copied to the high order 3 bits of the O/P register, along with the 12 bit offset, which is copied unmodified from the incoming virtual address together they form a 15- bit physical address. The O/P register is then put onto the memory bus as the physical memory address.

1- mapped 0- unmapped

8. Page Replacement Algorithms

To illustrate the page replacement algorithms we shall use the reference strings

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 1, 2, 0, 1, 7, 0, 1

For a memory with three page frames.

8.1 FIFO Algorithm

This algorithm associates with each page the time when that page was brought into memory. When a page must be replaced, the oldest page is chosen.

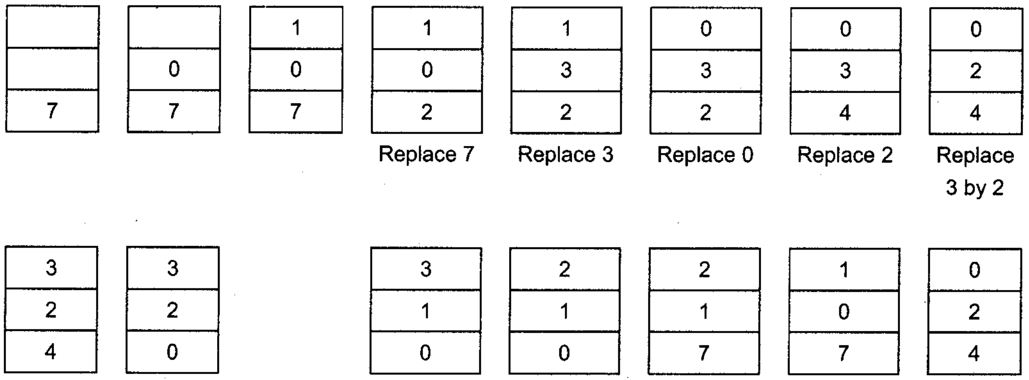


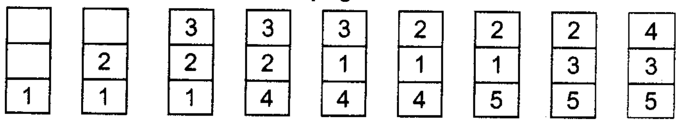
Figure 7.10: The FIFO page replacement

Algorithm is easy to understand and program. However, the performance is not always good.

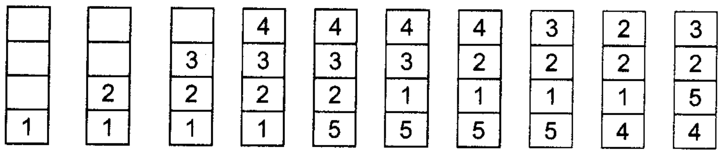
To illustrate the problems that are possible with a FIFO page replacement algorithm we consider the reference string-

1, 2, 3, 4, 1, 2, 5,
1, 2, 3, 4, 5

With 3 page frames



With 4 page frames



We noticed that the number of faults for 4 frames (10) is greater than the number of faults for three frames (9). This result is most unexpected and is known as *"Belady's Anomoly"*.

1
Apr.2013 – *2M*
Define Belady's Anomaly.

According to Belady, for some page –replacement algorithm, the page-fault rate may **increase** as the number of allocated frames **increases**. It is also called as FIFO anomaly, as this situation can occur only in FIFO page replacement algorithm.

8.2 Optimal Algorithm

This algorithm has the lowest page fault rate of all algorithms. It will never suffer from *"Belady's Anomoly"*. It is simply

"Replace the page that will not be used for the longest period of time."

Reference string

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

		1	1	3	3	3	1	1
	0	0	0	0	4	0	0	0
7	7	7	2	2	2	2	2	7

Page faults = 9

Unfortunately, the optimal page replacement algorithm, is difficult to implement, because it requires future knowledge of the reference string. As a result, this algorithm is used mainly for compare comparative studies.

8.3 Algorithm (Least Recently Used)

LRU replacement associates with each page the time of that pages last use. When a page must be replaced, LRU chooses that page that has not been used for the longest period of time.

This strategy is the optimal page replacement algorithm looking backward in time rather than forward.

Reference string

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

		1	1	3	3	2	2	2	2	2	7
	0	0	0	0	0	0	3	3	3	0	0
7	7	7	2	2	4	4	4	0	1	1	1

This policy is often used as a page replacement algorithm and is considered to be quite good.

Neither optimal replacement nor LRU replacement suffers from Belady's Anomoly. There is a class of page replacement algorithm called stack algorithms, that can never exhibit Belady's Anomoly. A stack algorithms is an algorithm for which it can be shown that the set of pages in memory for n frame and it would be in memory with n + 1 frames.

8.4 Second Chance Algorithm (MRU with Reference bit)

2

Oct. 2014 – *4M*
Write a note on second chance page replacement Algorithm

Apr. 2013 – *4M*
Write a short note on Second Chance Algorithm.

The basic algorithm of 2nd chance replacement is a FIFO replacement algorithm. When a page has been selected however, we inspect its reference bit. If a value is zero, we proceed to replace this page. If the reference bit is 1, however, we give that page a second chance and move on to select the next FIFO page. When a page gets a second page its reference bit is cleared and its arrival time is reset to the current time.

Reference string

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

		1	1	1	0	0	0	0	1	1	1
	0	0	0	3	3	3	2	2	2	2	7
7	7	7	2	2	2	4	4	3	3	0	0

1

Apr. 2012 – *4M*
Write a short note on Demand Paging.

9. Demand Paging

It is similar to a paging system with swapping. Processes reside on secondary memory. When we want to execute a process, we swap it into memory. Rather than swapping the entire process into memory, however, we use a lazy swapper. A lazy swapper never swaps a page into memory unless that page wide be needed. Since we are now viewing a process as a sequence of pages, the use of term swap is technically incorrect. A swapper manipulates entire processes, whereas a pager is concerned with the individual pages of a process.

In the purest form of paging, processes are started up with none of their pages in memory. As soon as the CPU tries to fetch the first instruction, it gets a page fault, causing the O.S. to bring in the page containing the first instruction. Other page faults usually follow quickly. After a while, the process has most of the pages it needs and settle down to run with relatively few page faults. This strategy, is called demand paging because pages are loaded on demand, not in advance.

Problem with this technique is that every time process is loaded into memory it will cause many page faults, causing wastage of considerable CPU time, since it takes the O.S. a few milliseconds of the CPU time to process a page fault. This problem becomes more significant in case of time shared and system.

Definition

The set of pages that a process is currently using is called its working set. If the available memory is too small to hold the entire working set, the process will cause many page faults and run very slowly, since executing an instruction typically takes a fraction of microsecond and reading in a page from the disk typically takes tens of milliseconds. A program causing page faults every few instructions is said to be ***Thrashing (Denning, 1968 b)***

Solution is as follows:

Many paging systems try to keep track of each process working set, and make sure that it is in memory before letting the process run. This approach is called the **working sit model (Denning 1970).** It is designed greatly **to reduce the pagefault rate.** Loading the pages before letting processes run is called pre-paging.

10. Segmentation

The users view of virtual memory is actually not as a linear array of bytes, but user views the memory as a collection of variable sized segments with no necessary ordering among segments.

1

Oct. 2015 – *4M*
Write a short note on segmentation.

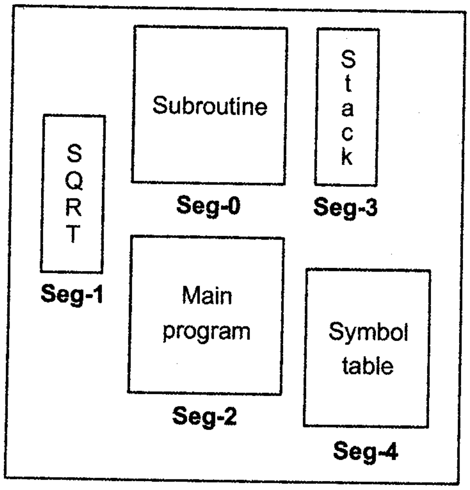


Figure 7.11

Segmentation is a memory management scheme that supports this user view of memory. A logical address space is a collection of segments. Each segment has a name and a length. The user therefore specifies each logical address by two quantities: a segment number and an offset within the segment (contrast this scheme with the paging scheme, where the user specified only a single address, which was partitioned by the hardware into a page number and an offset, all invisible to the program).

Thus a logical address consists of a two tuple

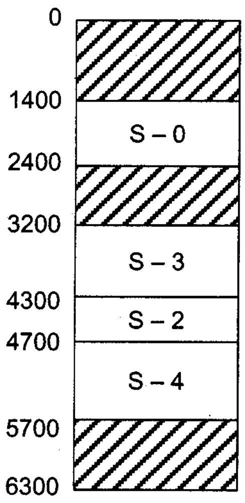
<segment no, offset>

Although logical address is referred by a tuple, the actual physical memory is still, of course, a 1 – D sequence of bytes.

The mapping of 2 – D user defined addresses into 1 – D physical addresses is effected by a segment table.

Each entry of the segment table has a segment base and a segment limit. The segment base contains the starting physical address where the segment resides in memory, whereas, the segment limit specifies the length of the segment.

Physical address space



	End	Start
	limit	Base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

3

Oct. 2012 – *4M*
What do you mean by Segmentation? Explain the advantages of Segmentation.
Oct. 11, Apr.11 – *4M*
What is meant by segmentation? List advantages and disadvantages of segmentation.

Advantages of segmentation

i. It simplifies the handling of data structures that are growing or shrinking.

ii. It is usually visible, unlike paging which is invisible to the programmer.

iii. It provides a convenient way of organizing programs and data to the programmer.

Disadvantages of Segmentation

i. It suffers from external fragmentation.

ii. Address translation, i.e., conversion from logical address to physical address is not a simple function, as compared to paging.

10.1 Segmentation With Paging

It is very common for the size of program modules to change dynamically. For instance, the programmer may have no knowledge of the size of a growing data structure. If a single address space is used, as in the paging form of virtual memory, once the memory is allocated for modules they cannot vary in size. This restriction results in either wastage or shortage of memory. To avoid the above problem, some computer systems are provided with many independent address spaces. Each of these address spaces is called a segment. The address of each segment begins with 0 and segments may be compiled separately. In addition, segments may be protected individually or shared between processes. However, segmentation is not transparent to the programmer like paging. The programmer is involved in establishing and maintaining the segments.

Some operating systems allow for the combination of segmentation with paging. If the size of a segment exceeds the size of main memory, the segment may be divided into equal size pages.

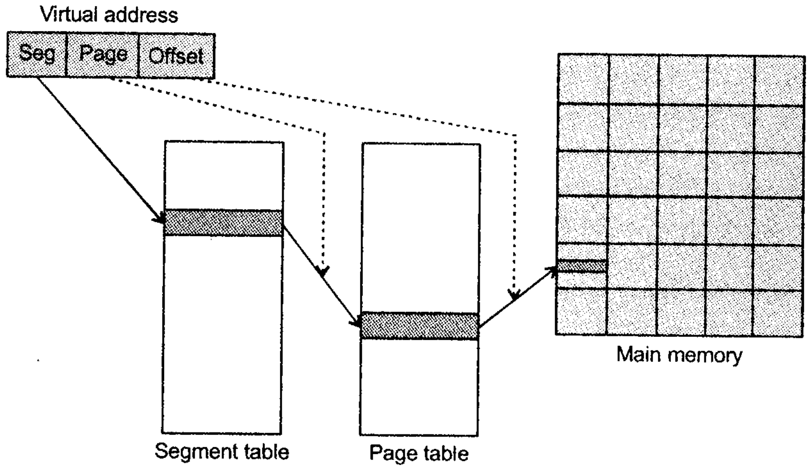


Figure 7.12

The address consists of three parts:

i. Segment number

ii. The page within the segment

iii. The offset within the page.

The segment number is used to find the segment descriptor and the address within the segment is used to find the page frame and the offset within that page.

Difference between Paging and Segmentation

Paging: Computer memory is divided into small partitions that are all the same size and referred to as, page frames. Then when a process is loaded it gets divided into pages which are the same size as those previous frames. The process pages are then loaded into the frames.

Segmentation: Computer memory is allocated in various sizes (segments) depending on the need for address space by the process. These segments may be individually protected or shared between processes. Commonly you will see what are called 'Segmentation Faults' in programs, this is because the data that is about to be read or written is outside the permitted address space of that process.

So now we can distinguish the differences and look at a comparison between the two:

Comparison between Paging and Segmentation

	Segmentation	Paging
1.	Involves programmer.	Transparent to programmer.
2.	Separate compiling.	No separate compiling.
3.	Separate protection.	No separate protection.
4.	Shared code.	No shared code.

In short, Segments can be of different lengths, so it is harder to find a place for a segment in memory than a page. With segmented virtual memory, we get the benefits of virtual memory but we still have to do dynamic storage allocation of physical memory. In order to avoid this, it is possible to combine segmentation and paging into a two-level virtual memory system. Each segment descriptor points to page table for that segment. This give some of the advantages of paging (easy placement) with some of the advantages of segments (logical division of the program).

Segmentation: Rectifies internal fragmentation.

Paging scheme

i. Rectifies external fragmentation.

ii. Combine both schemes and eliminates both problems in managing the memory.

Paged segmentation

i. Divide program's logical address space into one or more segments.

ii. Maintain a page table for each segment.

iii. Each segment pointer points to page table.

iv. Segment paging is opposite to above explanation.

Belady's Anamoly

1 Apr.2012 – *2M* Explain Belady's Anomaly.

The Belady's anomaly occurs in case of the FIFO page replacement policy in the operating system. When this FIFO is used and the number of page frames are increased in number, then the frames that are required by the program varies in a large range(due to large no of pages) as a result of this the number of page faults increases with the number of frames. This anomoly doesn't occur in the LRU (Least Recently Used) scheduling algorithm.

It is also called FIFO anomaly. Usually, on increasing the number of frames allocated to a process' virtual memory, the process execution is faster, because fewer page faults occur. Sometimes, the reverse happens, i.e., the execution time increases even when more frames are allocated to the process. This is Belady's Anomaly. This is true for certain page reference patterns.

Solved Examples

1 Apr.2013 – *4M*

1. **Consider the following page reference string:**
6, 4, 5, 1, 2, 6, 5, 4, 5, 3, 4,
The number of frames is 3. Show page trace and calculate page faults for the following page replacement schemes:
i. LRU ii. FIFO

Solution

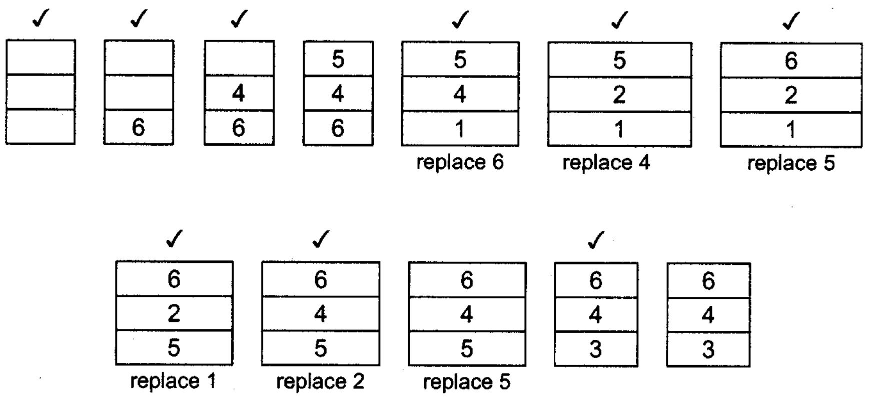
Given:

Reference String

6,4,5,1,2,6,5,4,5,3,4

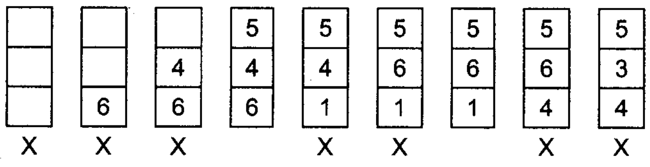
page frames: 3

i. FIFO



Total page faults = 9

ii. LRU



Page faults: 03 + 04 = 07

2. Consider the following page reference string:

4, 3, 2, 1, 4, 3, 5, 4, 3, 2, 1, 5

The number of frames are 3. Show page trace and calculate page faults for the following page replacement schemes: a. FIFO b. MFU

1 **Oct.2012 – *4M***

Solution

Page Reference String

4, 3, 2, 1, 4, 3, 5, 4, 3, 2, 1, 5

Number of page frames: 3

a. **FIFO**

			2	2	2	3	3	3	3	5
		3	3	3	4	4	4	2	1	1
	4	4	4	1	1	1	5	5	5	5
X	X	X		X	X	X	X	X	X	

Total number of page faults ∴ 6 + 3 = 9

b. **MFU**

Reference String

4, 3, 2, 1, 4, 3, 5, 4, 3, 2, 1, 5

			2	2	3	3	3	3	1
		3	3	1	1	1	5	5	5
	4	4	4	4	4	4	4	2	2
X	X	X		X	X		X	X	X

Number of page faults ∴ 05 + 03 = 08

3. **Consider the following page reference string:**
1, 3, 2, 1, 4, 3, 5, 4, 3, 2, 1, 5
The number of frames are 4. Show page trace and calculate page faults for the following page replacement schemes:

i. **FIFO** **ii. LRU**

1 Apr.2012 – 4M

Solution

Given: Reference string is 1, 3, 2, 1, 4, 3, 5, 4, 3, 2, 1, 5

Number of Frames **4**

i. **FIFO**

	1	3	2	1	4	3	5	4	3	2	1	5
	1	1	1	1	1	1	5	5	5	5	5	5
		3	3	3	3	3	3	3	3	3	1	1
			2	2	2	2	2	2	2	2	2	2
					4	4	4	4	4	4	4	4
P.F=	✓	✓	✓	X	✓	X	✓	X	X	X	✓	x

Total page faults = 06

ii. **LRU**

1	3	2	1	4	3	5	4	3	2	1	5
1	1	1	1	1	1	1	1	1	2	2	2
	3	3	3	3	3	3	3	3	3	3	3
		2	2	2	2	5	5	5	5	1	1
				4	4	4	4	4	4	4	5
✓	✓	✓	X	✓	X	✓	X	X	✓	✓	✓

Total page faults = 08

4. Consider the following page reference string:
7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1.
The number of frames are 3. Show the page trace and calculate the page faults for the following page replacement schemes.
i. LRU ii. Optimal Page Replacement

1 **Oct.2011 – *4M***

Solution

i. LRU

Ref. String	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	7	0	1
Frame 1	**7**	7	7	**2**	2	2	2	**4**	4	4	**0**	0	0	**1**	1	1	**7**	7	7
Frame 2		**0**	0	0	0	0	0	0	0	**3**	3	3	3	3	3	**0**	0	0	0
Frame 3			**1**	1	**1**	**3**	3	3	2	2	2	2	2	2	2	2	2	2	**1**
Page Fault Occurred?	Y	Y	Y	Y	N	Y	N	Y	Y	N	Y	N	N	Y	N	Y	Y	N	Y

Total number of page faults = 12

ii. Optimal Replacement

Ref. String	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	7	0	1
Frame 1	**7**	7	7	**2**	2	2	2	2	2	2	2	2	2	2	2	2	**7**	7	7
Frame 2		**0**	0	0	0	0	0	**4**	4	4	**0**	0	0	0	0	0	0	0	0
Frame 3			**1**	1	1	**3**	3	3	3	3	3	3	3	**1**	1	1	1	1	1
Page Fault Occurred?	Y	Y	Y	Y	N	Y	N	Y	N	N	Y	N	N	Y	N	N	N	N	N

Total number of page faults = 8

5. Consider the following page reference string:
2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2
The number of frame are 3. Show page trace and calculate page faults for the following page replacement schemes:
i. FIFO ii. MFU

1 **Apr.2011 – *4M***

Solution

i. FIFO

2 3 2 1 5 2 4 5 3 2 5 2

Page frames = 3

			1	1	1	4	4	2	2
	3	3	3	3	2	2	3	3	2
2	2	2	2	5	5	5	5	5	5

Page faults = 4

ii. MFU

Page frames = 3

2 3 2 1 5 2 4 5 3 2 5 2

		1	1	4	3
	3	3	5	5	5
2	2	2	2	2	2

Page faults = 3

SUMMARY

- The main purpose of a computer system is to execute programs. These programs, together with the data they access, must be at least partially in main memory during execution.
- There are three types of bindings

  a. Compile time binding b. Load time binding c. Run-time binding
- As process are loaded and removed from memory, the free memory space is broken into little pieces called **Holes.**
- External fragmentation exists when enough total memory space exists to satisfy a request but it is not contiguous storage is fragmented into a large number of small holes.
- Memory management techniques are
  - a. *Single partition allocation:* It keeps only one process in memory at a time and allow that process to use all of the memory.
  - b. *Multiprogramming with in fixed partition:* In this memory is divided into n partition when a job arrives it can be put into the I/O queue for the smallest partition large enough to hold it.
  - c. *Multiple partition allocation:* In this scheme, the O.S. keeps a label indicating which parts of memory are available which are occupied.
- There are various free space management techniques like

  a. First fit algorithm b. Next fit algorithm

  c. Best fit algorithm d. Worst fit algorithm e. Quick fit algorithm
- **Overlays:** It is to split the program into pieces called overlays, overlay zero would start running first.
- **Page Frames:** The corresponding units in a physical memory are called page frames.
- **Pages:** The virtual address space is divided up into units called page.
- **Working set:** The set of pages that a process is currently using is called as a working set.

PU Questions

2 Marks

[Oct.15, Apr.13 – *2M*]	1.	Define the term logical address.
[Apr.2015 – *2M*]	2.	What is External fragmentation?
[Apr.2015 – *2M*]	3.	Define Swap time.
[Oct.2014 – *2M*]	4.	Define static linking.
[Oct.2014 – *2M*]	5.	What is Internal fragmentation?
[Oct.14,12 – *2M*]	6.	Define Physical Address.
[Apr.2013 – *2M*]	7.	List various techniques of free Space Management in File system.
[Apr.2013 – *2M*]	8.	Define Belady's Anomaly.
[Apr.12,11 – *2M*]	9.	What is Swapping?

10. Explain Belady's Anomaly. **[Apr.2012 – *2M*]**
11. What is meant by Address Binding? **[Oct.2011 – *2M*]**

4 Marks

1. Differentiate between internal fragmentation and external fragmentation. **[Oct.2015 – *4M*]**
2. Write a short note on segmentation **[Oct.2015 – *4M*]**
3. Consider the following page reference string: 1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 3. Find the number of page fault for the following algorithm with 3 frames: i. FIFO ii. LRO **[Oct.2015 – *4M*]**
4. Consider the following job queue. **[Oct.2015 – *4M*]**

Job	Memory	Time
P_1	400 K	10
P_2	1200 K	5
P_3	300 K	20
P_4	700 K	8
P_5	500 K	15

Show the memory map of various stages by using MVT scheduling.Assume the total memory is of 3560 K and monitor is of 400 K and all jobs are arrived at same time.

5. What is page fault? Explain the different steps in handling a page fault. **[Apr.15,12 – *4M*]**
6. Differentiate between MVT and MFT job scheduler. **[Apr.15,12 – *4M*]**
7. Write a note on memory compaction. **[Apr.2015 – *4M*]**
8. Explain overlay's in detail with diagram. **[Apr.2015 – *4M*]**
9. Explain the overlapped swapping in detail. **[Apr.2015 – *4M*]**
10. Consider the following page reference string: 7, 5, 4, 9, 4, 7, 8, 5, 2, 3, 4, 7, 9, 7, 4.Find the number of page fault for the following algorithm with 3 frames: i. LFV ii. FIFO **[Apr.2015 – *4M*]**
11. Consider the following job queue. **[Apr.2015 – *4M*]**

Job	Memory	Time
1	80K	9
2	110K	4
3	20K	18
4	60K	5
5	40K	10

Show the memory map of various stages by using MVT scheduling. Assumption total memory is of 400K and monitor of 100K and all jobs are arrived at same time.

11. Explain Inverted page table with diagram. **[Oct.2014 – *4M*]**
12. Consider the following reference string 5, 4, 3, 2, 5, 4, 6, 5, 4, 3, 2, 6. How many page fault occurs for the following algorithm with 3 page frames. i. Optimal ii. FIFO **[Oct.2014 – *4M*]**
13. Explain resident monitor in detail. **[Oct.2014 – *4M*]**

[Oct.2014 – 4M] 14. Write a note on second chance page replacement Algorithm.

[Oct.2014 – 4M] 15. Explain MFT (Multiple Contiguous Fixed Partition allocation).

[Oct.2014 – 4M] 16. Consider the segment table:

Segment	Base	Length
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

What are Physical Addresses for following logical addresses?
i. 0,430 ii. 3,400
iii. 4,112 iv. 1,10

[Apr.2013 – 4M] 17. Consider the following page reference string:
6, 4, 5, 1, 2, 6, 5, 4, 5, 3, 4,
The number of frames is 3. Show page trace and calculate page faults for the following page replacement schemes:
i. LRU ii. FIFO

[Apr.2013 – 4M] 18. Explain in detail M.F.T. Job Scheduling.

[Apr.2013 – 4M] 19. Explain Swapping of two processes using disk as a backing store with diagram.

[Apr.2013 – 4M] 20. Write a short note on Second Chance Algorithm

[Oct.2012 – 4M] 21. Consider the following page reference string:
4, 3, 2, 1, 4, 3, 5, 4, 3, 2, 1, 5
The number of frames are 3. Show page trace and calculate page faults for the following page replacement schemes:
a. FIFO b. MFU

[Oct.2012 – 4M] 22. Explain in detail M.V.T. Job Scheduling.

[Oct.2012 – 4M] 23. What do you mean by Segmentation? Explain the advantages of Segmentation.

[Oct.2012 – 4M] 24. Write a short note on Overlays.

[Apr.2012 – 4M] 25. Consider the following page reference string:
1, 3, 2, 1, 4, 3, 5, 4, 3, 2, 1, 5
The number of frames are 4. Show page trace and calculate page faults for the following page replacement schemes:
i. FIFO ii LRU

[Apr.2012 – 4M] 26. Write a short note on Demand Paging

[Oct.2011 – 4M] 27. What is Fragmentation? Compare Internal and External Fragmentation.

Chapter 8
FILE SYSTEM

1. Introduction and File Concepts

File

A file is a named collection of related information that is recorded on secondary storage. File represents programs and data.

2
Apr.15,Oct.14 – *2M*
Define file.

File Attributes

A file is named and is referred to by its name. It has certain attributes such as:

1. **Name:** It is a symbolic name kept in a readable form.
2. **Identifier:** It is a unique tag. Usually a number used to uniquely identify its file. It is in non-human readable form.
3. **Type:** This information is needed for those systems that support different types.
4. **Location:** This information is a pointer to a device and points to the location of the file on that device.

3
Oct.2014 – *4M*
List and explain different attributes related to file
Oct.2012 – *2M*
List any four file Attributes.
Apr.2011 – *2M*
What is a File? List any two attributes of a file.

5. **Size:** The current size of the file (in bytes, words or blocks) and possibly the maximum allowed size.
6. **Protection:** Access control information determines who can do reading, writing, execution and so on.
7. **Time, data and user identification:** The information may be kept for creation, last modification and last use.

1.1 File Operations

A file is a logical entity, which represents a named piece of information. A file is a mapped onto a physical device.

1
Oct.2015 – *4M*
List and explain any two operations that can be performed on file.

1
Apr.2015 – *2M*
List Basic operations on file.

3
Oct.2011 – *2M*
List various Operations on Files.
Oct.2012 – *4M*
Define File. Explain the different operations of File.
Apr.2011 – *4M*
Write a short note on Operations on File.

There are many ways to map, or to implement files.

The operations on files are broadly grouped into as follows

1. **Creating a file:** To create a file, first space in the file system must be found for the file.
2. **Writing a file:** For writing a file a system call is made specifying the name of the file and the information to be written to the file. The operating system searches for the file entry in the directory. The directory entry must store a pointer to the current end of the file. Using this pointer, address of the next block can be computed and the information can be written.
3. **Reading a file:** To read from a file a system call specifies the name of the file, again the operating system searches the directory for the named file and the directory also needs to have a read pointer pointing to the next block to be read.

   In general, a file is either being read or written. Thus instead of having 2 pointer read pointer and write pointer, most systems have only one pointer called the current file pointer.
4. **Rewind a file:** Rewinding a file need not involve any actual I/O. Rather the directory is searched for appropriate entry and the current file position is simply reset to the beginning of the file.
5. **Delete the file:** To delete a file, we search the directory for the named file. If found the directory entry, it releases all file space and invalidate the directory entry.

6. **Truncating a file:** The user may want to erase the contents of a file but keep its attributes rather than facing the user to delete the file and then recreating it, we can truncate the file which maintains the attributes of the file but resets the length to zero.

7. **File Pointer:** The system must track the last read write location as a current file portion pointer. Thus pointer is unique to each process operating on the file and therefore must be kept separate from the disk file attributes.

8. **Access rights:** Each process opens a file in the access mode. This information is stored on the pre-process table so the operating system can allow or deny subsequent I/O requests.

9. **Desk location of the File:** Most file operation requires the system to modify data within the file. The information needed to locate the file on disk is kept in memory to avoid having to read it from disk for each operation.

2. Access Methods

There are several ways in which information stored in a file can be accessed, which are as follows:

2.1 Sequential Access

Most of the operations on a file are reads and writes. A read operation reads the next portion of the file and automatically advances a file pointer. Similarly a write operation appends to the end of the file and advances a file pointer to the end of the newly written data. Such a file can be rewound, and on some systems, a program may be able to skip forward or backward n records. This scheme is known as sequential access to a file. Sequential access is based upon a tape model of a file.

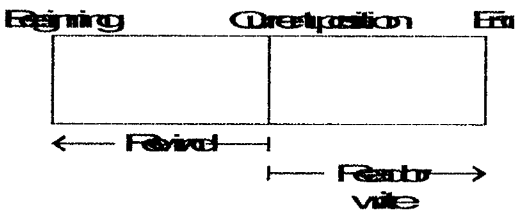


Figure 8.1: Sequential access files

2.2 Direct Access

Direct access is based on disk model of a file. For a direct access, the file is viewed as a numbered sequence of block or records. A block is generally a fixed-length quantity defined by the operating system.

A direct access file allows arbitrary blocks to be read or written. Thus we may read block 54 then read block 20 then write to block 8. There is no restriction on the order of reading or writing for a direct access file.

Direct access files are of great use for immediate access to large amounts of information.

The file operations must be modified to include the block number as the parameter. Thus we have 'read n' where n is the block number rather than 'read next' and 'write n' rather than 'write next'.

The block number provided by the user to the operating system is normally a relative block number. A relative block number is an index relative to the beginning of the file. Thus the first relative block of a file is 0. The next is 1 and so on even though the actual absolute addresses of the block may be 1078 etc.

The use of relative block numbers allows the operating system to decide where the file should be placed in the memory.

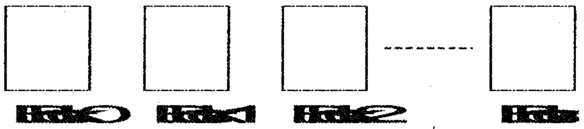


Figure 8.2: Direct access file

2

Oct.11, Apr.11 – *4M*

Differentiate between Sequential Access and Direct Access

Sequential access	Direct access
In this type of file a fixed format is used for records.	In this type of file, it is viewed as a number of sequential blocks of records.
All records are of same length, consisting of the same number of fixed length fields in a particular order.	A block is generally of fixed length quantity defined by an operating system.
Usually, first field in each record is referred to as the key field. The key field uniquely identifies the record.	A direct access file allows arbitrary blocks to be read or written.
In this method information in the file is processed in order, one record after the other.	There is no restrictions on the order of reading or writing for a direct access file.
In this type of file, a fixed format is used for records.	The block number provided by the user to the operating system is normally a relative block number and is of valuable length.
Absence of data structure.	Data structure is required for storing the data.
Automatic backup copy is created.	Automatic backup copy is not created.

2.3 Other Access Methods

Other access methods can be built on top of a direct access method. These methods generally involve the construction of an index for the file. The index contains pointers to the various blocks. To find a record in a file, we first search the index and then use the pointer to access the file directly and to find the desired record.

With large files, the index file itself may become too large to be kept in memory. One solution is to create an index for the index file. The primary index file would contain pointers to the secondary index files which would point to the actual data items.

3. File/Directory Structure

1

Oct.2011 – *4M*
Write a short note on File Directories.

The file systems of a computer can be huge, to manage all the data, we need to organize it. This organization is usually done in two parts. First, the disks are split into one or more partitions also called volumes.

A single disk can be partitioned and each partition is treated as a separate storage area or sometimes the partitions are larger than the disk size i.e. more than one disks comprise a single partition. The user needs to be concerned only with the logical directory and file structure and can completely ignore the problems of physically allocating space for files. So partitions can be thought of as virtual disks. Partitions can also store multiple operating system, allowing a system to boot and run more than one operating systems.

Second, each partition contains the information about the file within it. This information is kept in a device directory or volume table of contents which is normally called only 'directory' (*figure 8.3*).

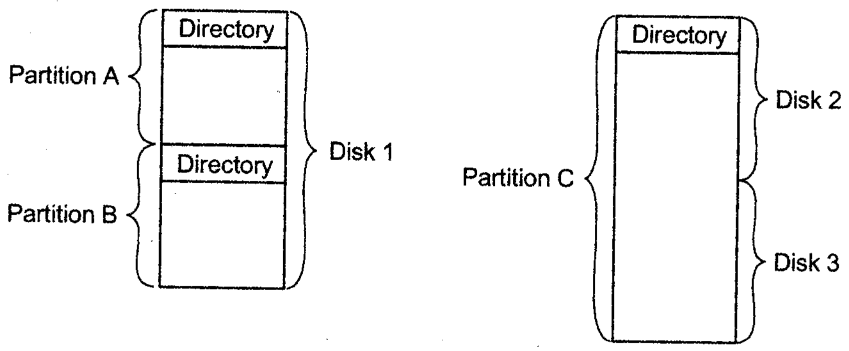


Figure 8.3: A typical file system organization

The directory records information such as name, location, size and type for all files on the device.

Many different types of directory structures have been proposed. But before seeing for structures we have to keep in mind the operations that are to be performed on a directory.

- **Search:** We need to be able to search the directory to find the entry for a particular file.
- **Create file:** New files need to be created and added to the directory.
- **Delete file:** When a file is no longer needed, we want to remove it from the directory.
- **List directory:** We need to be able to list the files in a directory and the contents of the directory entry for each file in the list.
- **Backup:** For reliability generally backup is taken at regular intervals. This often consists of copying all files to magnetic tape.

3.1 Single Level Directory

This is the simplest directory structure. All files are contained in the same directory.

A single level directory structure results in a file system tree with two levels: the single root directory and (all) the files in this directory. That is, there is one level of directories and another level of files so the full file system tree has two levels.

On early personal computers, this system was common, in part because there was only one user. The world's first supercomputer, the CDC 6600, also had only a single directory for all files, even though it was used by many users at once.

A single-level directory has significant limitations, when the number of files increases or when the system has more than one user.

Since all files are in the same directory, they must have unique names. If two users call their data file test, then the unique-name rule is violated.

Even a single user on a single-level directory may find it difficult to remember the names of all the files as the number of files increases.

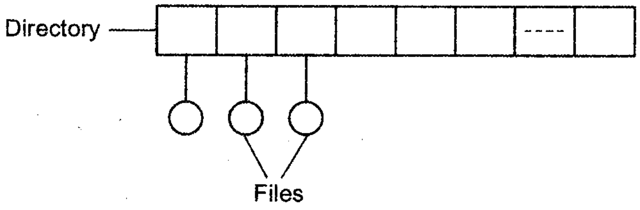


Figure 8.4: Single level directory

1

Apr.2013 – *4M*
Give the diagrammatic representation of Single Level Directory. Also list out the disadvantages of Single Level Directory Structure.

Disadvantages

i. It has significant limitations as the number of files increases.

ii. It becomes more and more difficult to store all the files in the same directory.

iii. Since each file should have a unique name and if number of users of the system is more, many users tend to create files with the same names.

3.2 Two Level Directory

The major disadvantage of a single level directory is the confusion of file names between different users. The solution is to create separate directory for each user.

In a two level directory, each user has his own User File Directory (UFD). Each user directory has a similar structure but lists the files of a single user.

When a user logs in, the systems Master File Directory (MFD) is searched.

When a user refers to a file, only their own directory is searched. Thus different users may have files with the same name, as long as all files within each user, file directories are unique.

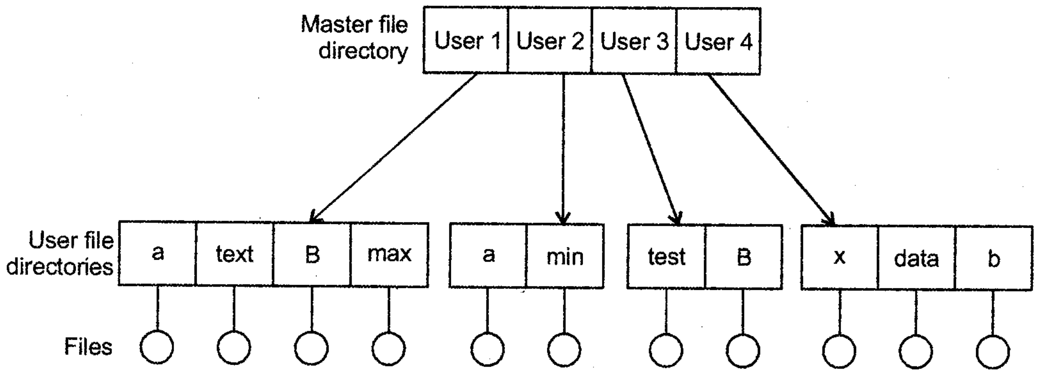


Figure 8.5: Two level directory structure

There are still problems with the two level directories. The structure isolates one user from another. This is advantageous if the users are completely independent but a disadvantage is that if the users want to cooperate and access files of other users.

Some systems allow such access but the complete path of the file has to be specified.

For example: If user 1 wants to access the test file of user 3 he has to specify the path as /user3/test.

3.3 Tree Structure Directories

This generalization allows users to create their own sub-directory and to organize their files accordingly.

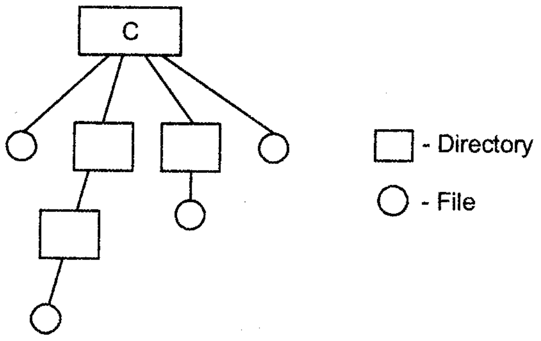


Figure 8.6

Path name can be of two types

i. **R ⟶ D:** An absolute path name begins at the root and follows a path down to the specified file, giving the directory names on the path.

ii. **D:** Relative path name defines a path from the current directory.

3.4 Acyclic Graph Directories

1

Apr.2012 – ***4M***
Write short note on Acyclic Graph Directory.

Suppose the users of the operating system needs to share a **common subdirectory or files,** tree structured approach wont work.

An **acyclic graph allows directories to have shared subdirectories and files.** The same file or sub-directory may be in two different directories. An acyclic graph is a natural generalization of the tree structured directory scheme. In UNIX this directory structure is used.

Advantage of this directory structure is that **it is more flexible and allows sharing of files efficiently**.

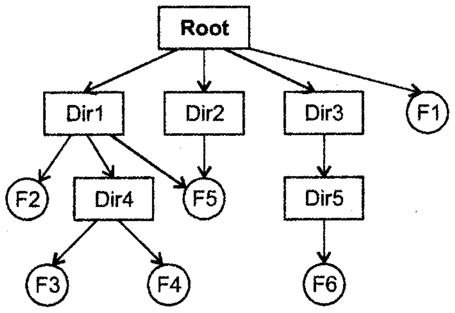


Figure 8.7

Drawbacks of this structure can be

i. Entry file system traversal is difficult.

ii. Deletion of file requires extra care such as link count taken in Unix operating system.

4. File Allocation Methods

The direct access nature of disks allows us flexibility in the implementation of files. In most cases, many files will be stored on the same disk. The main problem is how to allocate space to these files. So that, disk space is utilized effectively and files can be accessed quickly.

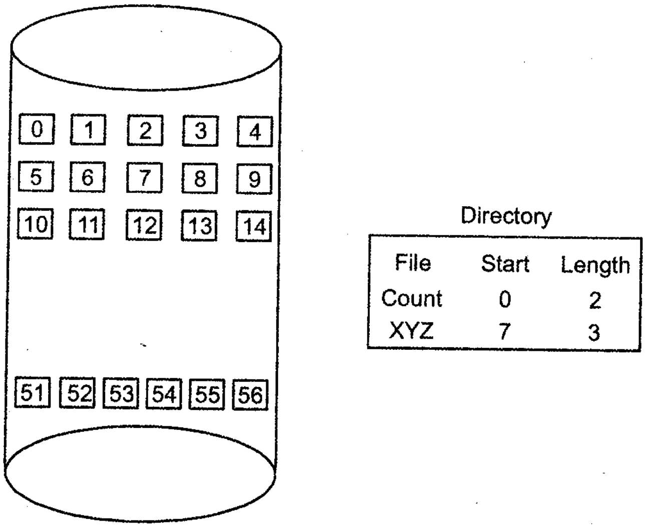
Three major methods of allocating disk space are:

1. Contiguous Allocation
2. Linked
3. Indexed

4.1 Contiguous Allocation

The contiguous allocation method requires each file to occupy a set of contiguous blocks on the disk. Disk addresses define linear ordering on the disk.

Accessing a file that has been allocated contiguously is easy. One difficulty with contiguous allocation is finding space for a new file. However it suffers from the problem of external fragmentation.



File	Start	Length
Count	0	2
XYZ	7	3

Figure 8.8

4.2 Linked Allocation

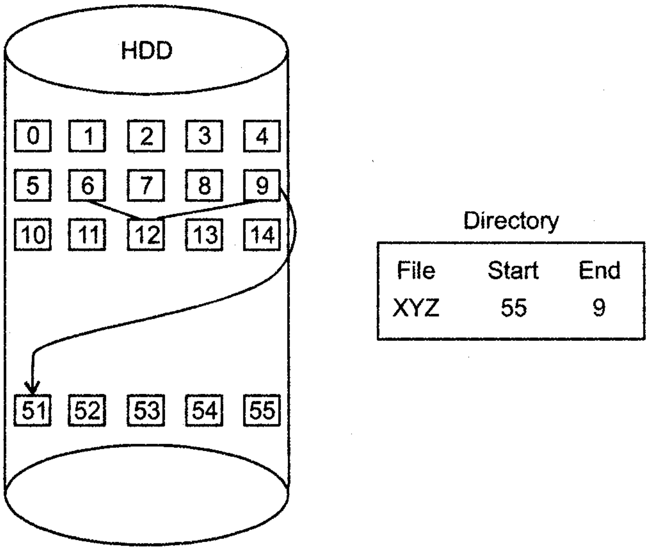


Figure 8.9

Linked allocation solves all problems of contiguous allocation. With linked allocation each file is a linked list of disk blocks, the disk blocks may be scattered anywhere on the disk. The directory contains the pointer to the first and last blocks of the file.

i. The major problem is that it cannot be used effectively for random access files.

ii. Another disadvantage is the space required by the pointer. Yet another problem is reliability.

4.3 Indexed Allocation

Linked allocation solves the external fragmentation and size declaration problem of contiguous allocation. However, in the absence of FAT, linked allocation cannot support efficient direct access. Since the pointer to the blocks are scattered with the blocks themselves all over the disk and need to be retrieved in order indexed allocation solves this problems by bringing all the pointers together in one location: the index block.

1

Apr.2012 – *4M*
Explain Indexed Allocation Method in detail.

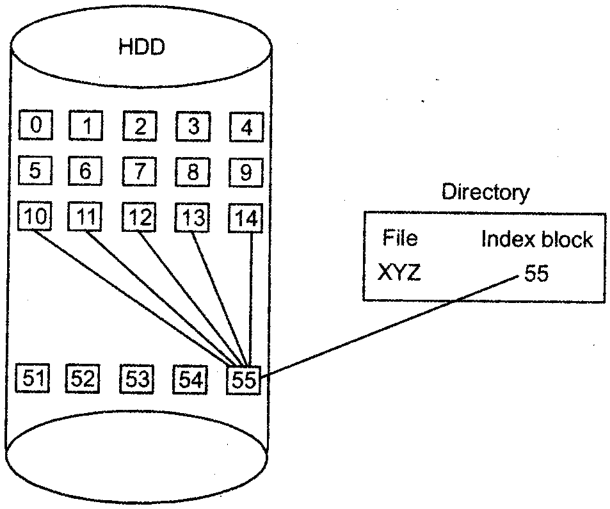


Figure 8.10

Each file has its own index block which is an array of desk block addresses. The i th entry in the index block points to the i th block of file. The directory contains the address of the index block as shown in above *figure*.

5. File System Structure

Disk provide the bulk of secondary storage on which a file system is maintained. They have two characteristics that make them a convenient medium for storing multiple files.

i. They can be rewritten in place, it is possible to read a block from the disk to modify the block and to write it back into the same place.

ii. They can access directly any given block of information on the disk. It is simple to access any file either sequentially or randomly and switching from one file to another requires only moving the read-write heads and waiting for the disk to rotate.

Rather than transferring a byte at a time, to improve I/O efficiency, I/O transfers are performed in units of blocks. Each block is one or more sectors. The operating system imposes one or more file systems to allow data to be stored, located and retrieved easily. The file system itself is generally composed of many different levels. The structure shown in *figure 8.11* is an *example* of layered design. Each level in the design uses the features of lower levels to create new features for use by higher levels.

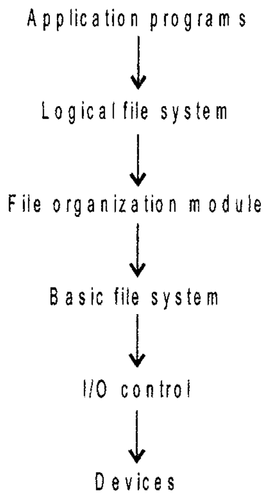


Figure 8.11: Layered file system

The lowest level, the I/O control, consists of device drivers and interrupts handlers to transfer information between the main memory and the disk system.

The basic file system needs only to issue generic commands to the appropriate device driver to read and write physical block on the disk. Each physical block is identified by its numeric disk address (*for example:* drive 1, cylinder 43, etc.)

The file organization module knows about the files and their logical blocks as well as physical blocks. The file organization module translates logical block addresses to physical block addresses.

The logical file system manages Metadata information. Metadata includes all of the file system structure, excluding the actual data or contents of the file. The logical file system manages the directory structure to provide the file organization module with the needed information. It maintains the file structure using file control blocks. The File Control Blocks (FCB) contains information about the file including ownership, permissions and location of the file contents.

6. Free Space Management

Files are created and deleted frequently during the operation of a computer system. We have to reuse the space from deleted files for new files. To keep track of free disk space, the file system maintains a free space list.

The free space list records all disk blocks which are free (i.e. not allowed to a file). To create a file, we search the free list for the required amount of space and allocate into the new file. This space is then removed from the free space list. When a file is deleted, its disk space is added to the free space list.

6.1 Bit Vector

1
Oct.2012 – *4M*
Discuss various techniques of Free Space Management in File System.

The free space list may be complemented in many different ways. One implementation is a bit map or bit vector. Each block is represented by one bit. If the block is free, the bit is 0 if the block is allocated, the bit is 1.

Example of the bit vector is
0 0 1 0 1 1 0 0 1 0 1 1 0

Then the block is 0, 1, 3, 6, 7, 9 are free.

6.2 Linked List

Another approach is to link all the free blocks together keeping a pointer to the first free block. This block contains a pointer to the next free block and so on.

1
Apr.2011 – *4M*
What is meant by Free Space Management? Define Bit Vector and Linked List.

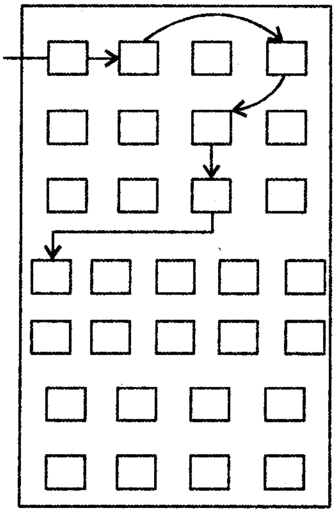


Figure 8.12: Linked free space list on disk

6.3 Grouping

It stores the addresses of n free blocks in the first block. The first n – 1 of these are actually free. The last one is the disk address of another block containing the addresses of another n free blocks. Here the advantage is that addresses of a large number of free blocks can be found quickly.

6.4 Counting

Another approach is to take advantage of the fact that generally several contiguous blocks may be allocated or freed simultaneously. Thus rather than keeping a list of a free disk addresses we can keep the address of the first free block and the number or of the free contiguous blocks which follow it. Each entry requires more space than a simple disk address as it consists of a disk address and a count, but the overall list will be sheltered.

SUMMARY

- File is a named collection of related information defined by its creator.
- **File attributes are:** name identifier, type, location, size, protection, time, date and user identification.
- **File operations:** Create, unite, read, reposition (rewind), delete, truncate, open, close.
- There are three access methods:
  a. Sequential access method
  b. Direct access method
  c. Other access methods
- There are five types of directory structures viz:
  a. Single level directory b. Two level directory
  c. Tree structure directory d. Acyclic graph directory
- Desk provides the block of secondary storage on which a file system is maintained.
- An important issue for an operating system while creating files is how to allocate space to these files.
  There are different file allocation methods such as:
  a. Contiguous allocation b. Linked allocation
  c. Indexed allocation
- The free space list records of all disk blocks which are free.
  The various free space management techniques are:
  a. Bit vector b. Linked list
  c. Grouping d. Counting

PU Questions

2 Marks

[Apr.15,Oct.14 – *2M*]	1.	Define file.
[Apr.2015 – *2M*]	2.	List Basic operations on file.
[Oct.2012 – *2M*]	3.	List any four file Attributes.
[Oct.2011 – *2M*]	4.	List various Operations on Files.
[Apr.2011 – *2M*]	5.	What is a File? List any two attributes of a file.

4 Marks

[Oct.2015 – *4M*]	1.	List and explain any two operations that can be performed on file.
[Oct.2014 – *4M*]	2.	List and explain different attributes related to file.
[Apr.2013 – *4M*]	3.	Give the diagrammatic representation of Single Level Directory. Also list out the disadvantages of Single Level Directory Structure.
[Oct.2012 – *4M*]	4.	Discuss various techniques of Free Space Management in File System.
[Oct.2012 – *4M*]	5.	Define File. Explain the different operations of File.
[Apr.2012 – *4M*]	6.	Explain Indexed Allocation Method in detail.
[Apr.2012 – *4M*]	7.	Write short note on Acyclic Graph Directory.
[Oct.2011 – *4M*]	8.	Write a short note on File Directories.
[Oct.11, Apr.11 – *4M*]	9.	Differentiate between Sequential Access and Direct Access.
[Oct.11, Apr. 11 – *4M*]	10.	What is meant by Free Space Management? Define Bit Vector and Linked List.
[Apr.2011 – *4M*]	11.	Write a short note on Operations on File.

VISION

Chapter 9
I/O SYSTEM

1. Introduction

The two main jobs of a computer are I/O and processing. The role of the computer I/O is to manage and control I/O operations and I/O devices.

The control of devices connected to the computer is a major concern of operating system designers. The I/O devices vary so widely in their function and speed, a variety of methods are needed to control them. These methods form the I/O subsystem of the kernel which separates the rest of the kernel from the complexity of managing I/O devices.

The device drivers present a uniform device access interface to the I/O subsystem, same like the system calls provide a standard interface between the applications and the Operating System.

2. I/O Hardware

Many errors can occur in the programs written by the programmer, such as an attempt to execute illegal instruction, or to access memory that is not in the users address space-then hardware will trap to the Operating System. Whenever a program error occurs, the Operating System must abnormally terminate the program.

An appropriate error message is given and the memory of the program is dumped. The memory dumped is usually written to a file so that the user can examine it and perhaps can correct and restart the program.

3. Application of I/O Interface

2

Oct., Apr.11 – *4M*
Describe the application of I/O interfaces in details

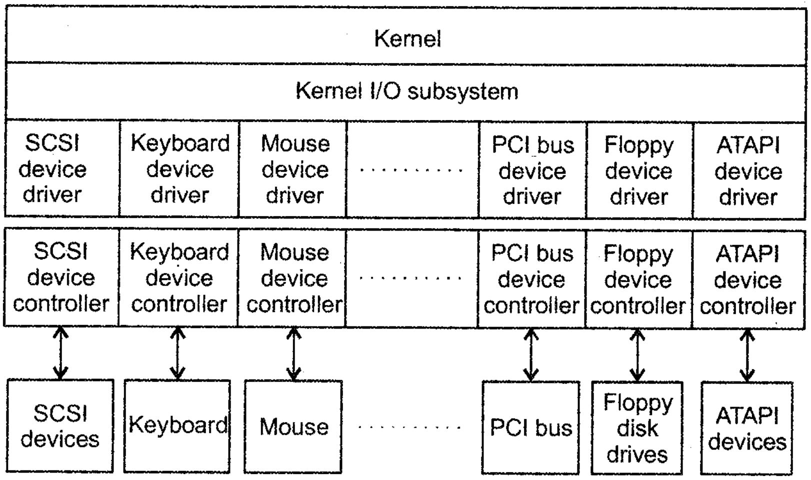


Figure 9.1: Kernel I/O structure

The device driver layer is used to hide the differences among device controllers from the I/O subsystem of the kernel making the I/O subsystem independent of the hardware, which simplifies the job of operating system developers. It also benefits the hardware manufacturers. They either design new devices to be compatible with an existing host controller interface or they write devices to drivers to interface the new hardware to the operating system.

There may be variations with the devices in

1. **Character stream or block:** A character stream device transfers bytes one by one whereas a block device transfers a block by bytes as a unit.
2. **Sharable or dedicated:** A sharable device can be used concurrently by several processes or threads. A dedicated device cannot be used by more than one processes.
3. **Speed of operation:** Device speeds range from a few bytes per second to a few giga bytes per second.

4. **Read, write, read only or write only:** Some devices perform both input and output operations, but others support only one data direction.

5. **Synchronous or asynchronous:** A synchronous device is one that performs data transfers with predictable response times. An asynchronous device exhibits irregular or unpredictable response time.

6. **Sequential or random access:** A sequential device transfers data in a fixed order determined by the device, whereas the user of a random access device can instruct the device to seek to any of the available data storage location.

4. Direct Memory Access (DMA)

Direct memory access (DMA) is a method that allows an input/output (I/O) device to send or receive data directly to or from the main memory, bypassing the CPU to speed up memory operations. A computer's system resource tools are used for communication between hardware and software. The four types of system resources are:

i. I/O addresses
ii. Memory addresses
iii. Interrupt Request Numbers (IRQ)
iv. Direct Memory Access (DMA) channels

1
Oct.2015 – *4M*
Write a short note on Direct Memory Access (DMA).

DMA channels are used to communicate data between the peripheral device and the system memory. All four system resources rely on certain lines on a bus.

A DMA channel enables a device to transfer data without exposing the CPU to a work overload. Without the DMA channels, the CPU copies every piece of data using a peripheral bus from the I/O device. Using a peripheral bus occupies the CPU during the read/write process and does not allow other work to be performed until the operation is completed.

With DMA, the CPU can process other tasks while data transfer is being performed. The transfer of data is first initiated by the CPU. During the transfer of data between the DMA channel and I/O device, the CPU performs other tasks. When the data transfer is complete, the CPU receives an interrupt request from the DMA controller.

5. Kernel I/O Subsystem

1

Apr.2013 – *4M*
Explain in brief different services provided by Kernel related to I/O.

Kernel provides many services related to I/O. Several services like buffering, caching, spooling, device reservation and error handling are provided by the kernels I/O subsystem and built on the hardware and device driver infrastructure.

5.1 I/O Scheduling

To schedule I/O means to determine the order in which to execute the I/O. I/O requests are rarely executed in the order in which they are received. Scheduling can improve overall system performance and can reduce the average waiting time for I/O to complete.

A queue is maintained for all the I/O requests of a device. The scheduler then rearranges the order of processes in the queue to improve the overall system efficiency and the average response time experienced by application.

5.2 Buffering

2

Oct.2014 – *4M*
Explain Buffering in detail

Oct.2012 – *2M*
What is Buffering?

1

Oct.2015 – *4M*
Write a short note on Buffering.

A buffer is a memory area that stores data while they are transferred between two devices. There are various reasons why buffering needs to be done. One reason is to cope with a speed mismatch between the producer and consumer of data.

For example: A file is being transferred to a disk via a modem. The hard disk is much faster than the modem so to mask this speed difference, a buffer is created in main memory to accumulate bytes received from the modem. When an entire buffer of data has arrived then the buffer is written to disk in a single operation. The second use of buffering is to adopt between devices that have different data transfer sizes.

For example: In networking, when fragmentation and reassembly of messages is done. At the sending side, a large message is fragmented into small network packets and the receiving size places them in a reassembly buffer to form an image of the source data.

Another application of buffering is to support copy semantics for application I/O. A copy semantics is that when an application generates a write () system call, with a pointer to a buffer that it wishes to write, then the contents of the buffer are immediately copied to kernel buffers and then the data is transferred from kernel buffers to the disk. This is done to ensure that even if changes are made to the buffer by the application, still only those contents that were present in the buffer at the time, when it made a write () call are written to the disk.

5.3 Caching

A cache is a region of fast memory that holds copies of data. Access to the cached copy is more efficient than access to the original. *For example:* The instructions of the currently running process are stored on disk, cached in physical memory and copied again in the CPU's secondary and primary caches. The difference between a buffer and a cache is that a buffer may hold the only existing copy of a data item, where a cache just holds a copy on faster storage of an item that resides elsewhere.

Caching and buffering are distinct functions but sometimes a region of memory can be used for both the purposes. *For example:* To preserve copy semantics and to enable efficient scheduling disk I/O, the operating system uses buffers in main memory to hold disk data. These buffers are also used as a cache to improve the I/O efficiency for files that are shared by applications, that are being written and re-read rapidly.

5.4 Spooling and Device Reservation

A spool is a buffer that holds output for a device such as a printer, that cannot accept interleaved data stream. *For example:* The printer can print only one job at a time, but more than one applications may want to print concurrently at the same time, but the output should not be mixed together. The operating system solves this problem by intercepting all output to the printer. Each applications output is spooled to a separate disk file and the spooling system maintains a queue of such spooled file. When the printer finishes the job it is doing, the spooling system assigns the next job from the queue to the printer. The operating system provides a control interface that enables users and system administrator to view the queue, to remove unwanted jobs before they are printed and to suspend printing while the printer is serviced.

Some devices such as tape drives cannot usefully multiplex the I/O requests of multiple concurrent applications. Here the way to deal with concurrent device access is to provide explicit facilities for co-ordination. Some systems provide support for exclusive device access by enabling a process to allocate an idle device and to deallocate that device when it is no longer needed.

5.5 Error Handling

An operating system that uses protected memory can guard against many kinds of hardware and application errors, so complete system failure is not the usual result of any minor problem. Devices and I/O transfer can fail in many ways, either for transient reasons. Such as network becoming over loaded as for permanent reasons, such as disk controller becoming defective. Operating system can often effectively compensate for transient failures. *For example:* read () failure results in a read () retry and a network send () failure results in a resend (). But if an important component experiences a permanent failure then the Operating System is unlikely to recover.

Generally an I/O system call will return 1 bit information about the status of the call, signifying either success or failure. Some systems use addition bits to give more detailed information of the error occurred, type of error, etc.

5.6 Kernel Data Structures

The kernel needs to keep state information about the use of I/O components. It does so through a variety of in-kernel data structure, such as the open file table structure and other such structures that the kernel uses to track network connections, character device communications and other I/O activities.

Some operating system use object oriented methods. *For example:* Windows NT uses a message passing implementation for I/O. An I/O request is converted into a message that is sent through the kernel to the I/O manager and then to the device drive, each of which may change the message contents. For output the message contains the data to be written, for input the message contains buffer to receive the data. The message passing approach can add overhead, but it simplifies the structure and design of the I/O system and adds flexibility.

The I/O subsystem co-ordinates and supervises a lot of activities like:

- Management of name space for files and devices
- Access control to files and devices
- Operation control
- File system space allocation
- Device allocation
- Buffering, caching and spooling
- I/O scheduling
- Device status monitoring, error handling and failure recovery
- Device driver configuration and initialization

5.7 I/O Protection

To prevent user from performing illegal I/O, we define all I/O instructor to be privileged instructions. Thus, users cannot issue I/O instructions directly, they must do it through Operating System. To do I/O, a user program executes a system call to request that the operating system perform I/O on its behalf.

The operating system executing in monitor mode checks that the request is valid and if it is, does the I/O requested, the operating system then returns to the user.

In addition, any memory mapped and I/O port memory locations must be protected from user access by the memory protection system.

5.8 Memory Protection

Memory protection is provided by using two registers, usually a base and a limit. The base register holds the smallest legal physical memory address the limit register contains size of the range.

Thus protection is accomplished by the CPU hardware comparing every address generated in user mode with the registers. Any attempt by a program executing in user mode to access monitor memory or other users memory results in a trap to the monitor, which treats the attempt as a fatal error.

5.9 CPU Protection

We must prevent a user program from getting stuck into an infinite loop and never return in control to the operating system to achieve this we can use a timer. A timer can be set to interrupt the computer after a specified period.

6. Dual Mode Operation

To ensure proper operation, we must protect the operating system and all other programs and their data from any malfunctioning program.

For this purpose there are two separate modes of operation

i. User mode

ii. Monitor mode (supervision system / privileged mode)

A bit called the mode bit is added to the hardware of the computer to indicate the current mode: Monitor (0) or user (1).

At system boot time the hardware starts in monitor mode. The operating system is then loaded and starts user processes in user mode. Whenever a trap or interrupt occurs, the hardware switches from user mode to the monitor mode.

Some of the instructions are designed to the privileged instructions that may cause harm. The hardware allows the privileged instructions to be executed in only monitor mode.

7. Disk Scheduling

2

Oct.2011 – *2M*
What do you mean by Seek Time in Disk Scheduling?

Apr.2011 – *2M*
What is meant by Disk Scheduling?

File systems must be accessed in an efficient manner as the computer deals with multiple processes over a period of time, a list of request to access the disk builds up. The operating system uses a disk scheduling technique to determine which request to satisfy.

The disk access time has two major components

1. **The seek time:** It is the time for the disk arm to move the heads to the cylinder containing the desired sector.
2. **Rotational latency:** It is the additional time waiting for the disk to rotate the desired sector to the disk head.

Whenever a process needs I/O to / from the disk, it issues a system call to the operating system if the derived disk drives and controllers are available, the request can be serviced immediately. If the drive or controller is busy, any new requests for service will need to be placed on the queue of pending requests for that drive.

For a multiprogramming system with many processes, the disk queue may often have several pending requests. Thus, when one request is completed, the operating system has an opportunity to choose which pending requests to service next. The part of the operating system which makes this decision is called **disk scheduler**.

Disk scheduling techniques are:

i. FCFS

ii. SSTF (Shortest Time Seek First)

iii. Scan

iv. C–Scan

v. C–Look

7.1 FCFS Scheduling

This algorithm is intrinsically fair, but it generally does not provide the fastest service.

Example: Disk queue with requests for I/O to block on cylinders 98, 183, 37, 122, 14, 124, 65, 67

If the disk head is initially at cylinder 53 calculate the total head movement using FCFS scheduling for the cylinder range from 0-199

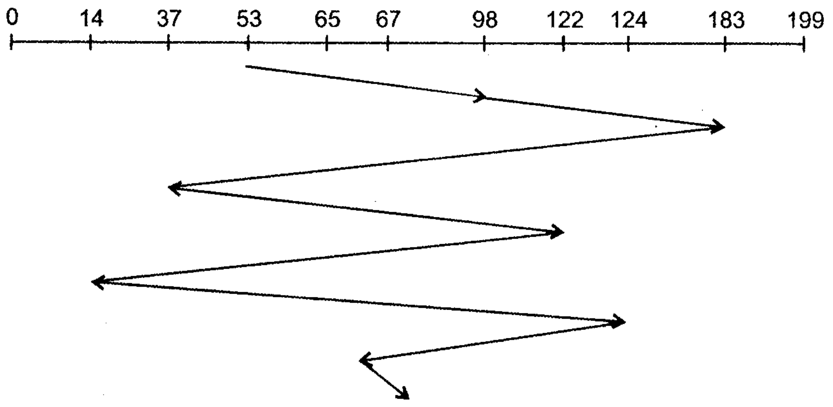


Figure 9.2

Total head movement = |53 – 98| + |98 – 183| + |183 – 37| + |37 -122|

+ |122 – 14| + |14 – 124| + |124 – 65| + |65 – 67|

= 45 + 85 + 146 + 85 + 108 + 110 + 59 + 2

= 640 cylinders

7.2 Shortest Seek Time First (SSTF) Scheduling

This algorithm selects the request with the minimum seek time from the current head position. Since seek time increases with the number of cylinders traversed by the head, SSTF chooses the pending request closest to the current head position. SSTF scheduling like 'SJF', it may cause starvation.

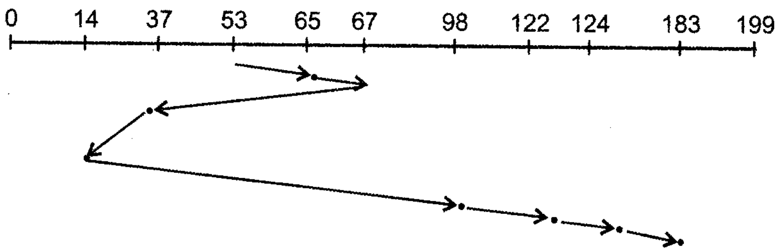


Figure 9.3

Total head movement = |53 – 65| + |65 – 67| + |67 – 37| + |37 – 14|

+ |14 – 98| + |98 – 122| + |122 – 124| + | 124 – 183|

= 12 + 2 + 30 + 23 + 84 + 24 + 2 + 59

= 236

7.3 Scan Scheduling

In this algorithm the disk arm starts at one end of the disk and moves towards the other end of the disk, servicing requests as it reaches each cylinder, until it gets to the other end of the disk. At the other end, the direction of head movement is reversed, and servicing continues. The head continuously scans back and forth across the disk. This algorithm is sometimes called the **Elevator Algorithm**. Since the disk arm behaves first like an elevator in a building, first servicing all the request going up and then reversing to service requests the other way.

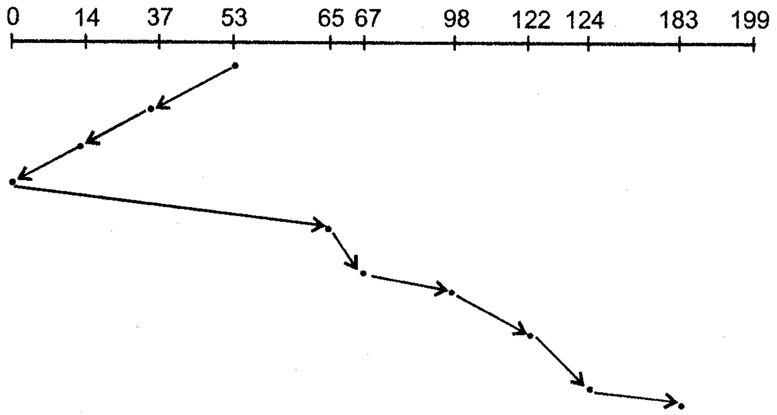


Figure 9.4

Total head movement = |53 – 37| + |37 – 14| + |14 – 0| + |0 – 65| + |65 – 67|

+ |67 – 98| + |98 – 122| + |122 – 124| + |124 – 183|

= 16 + 23 + 14 + 65 + 2 + 31 + 24 + 2 + 59

= 236

7.4 C Scan (Circular Scan)

It is a variant of scan that is designed to provide a more uniform wait time. Like scan, C scan moves the head from one end of the disk to the other, servicing requests along the way. When the head reaches the other end, however, it immediately returns to the beginning of the disks without securing any requests on the return trip.

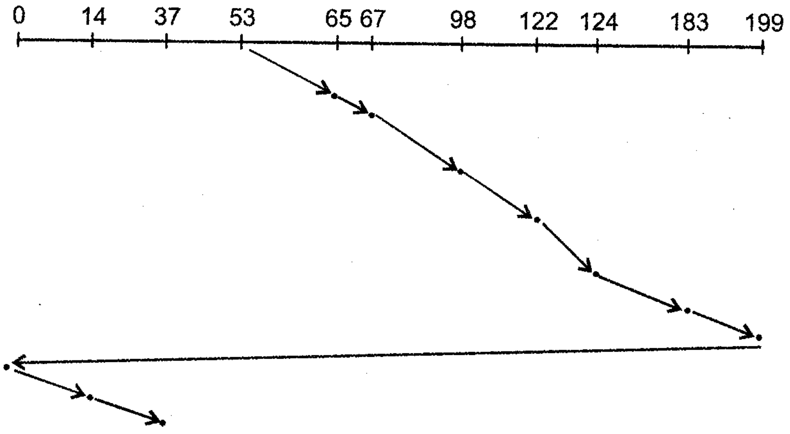


Figure 9.5

Total head movement = |53 – 65| + |65 – 67| + |67 – 98| + |98 – 122| + |122 – 124|

+ |124 – 183| + |183 – 199| + |199 – 0| + |0 – 14| + |14 – 37|

= 12 + 2 + 31 + 24 + 2 + 59 + 16 + 199 + 14 + 23

= 382

7.5 Look and C Look Scheduling

Both scan and C scan move the disk arm across the full width of the disk. In practice, neither algorithm is implemented this way. More commonly, the arm goes only as far as the final request in each direction then, it reverses direction immediately, without first going all the way to the end of the disk. These versions of scan and C – Scan are called **Look and C- Scan**, because the looks for a request before continuing to more in a given direction.

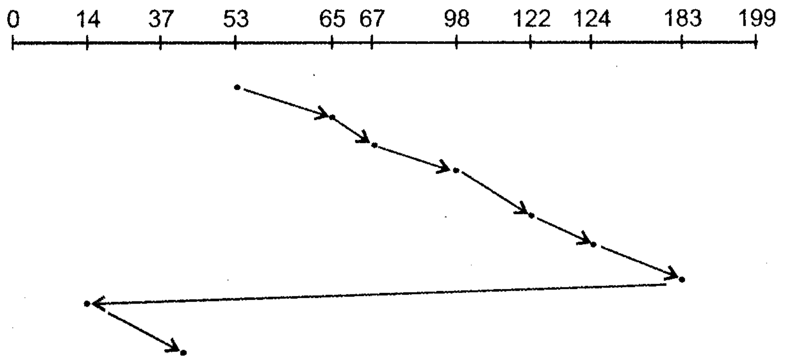


Figure 9.6

Total head movement = |53 – 65| + |65 – 67| + |67 – 98| + |98 – 122|

+ |122 – 124| + |124 – 183| + |183 – 14| + |14 – 37|

= 12 + 2 + 31 + 24 + 2 + 59 + 16 + 169 + 23

= 322 cylinders.

1

Apr.2013 – *4M*

What is polling and how it is achieved to control more than one device.

8. Polling

The protocol for handshaking between the host and controller is simple *for example:* we assume that 2 bits are used to co-ordinate the producer consumer relationship between the controller and the host. The controller indicates its state through the 'busy' bit in the status register to 1. The controller sets the busy bit when it is busy working and clears the busy bit when it is ready to accept the next command.

The host signals its wishes via the command ready bit in the command register. The host sets the command ready bit when a command is available for the controller to execute.

For example: The host writes the O/P through a port, coordinating with the controller by handshaking as follows:

i. The host repeatedly reads the busy bit until that bit becomes clear.
ii. The host sets the write bit in the command register and writes a byte into the data-out register.
iii. The host sets the command ready bit.
iv. When the controller notices that the command ready bit is set, it sets the busy bit.

v. The controller reads the command register and sees the 'write' command. It reads the data-out register to get the byte and does the I/O to the device.

vi. The controller clears the command ready bit, clears the error bit in the status register to indicate that the device. I/O succeeded and clears the busy bit to indicate that it is finished.

This loop is repeated for each byte. In step 1, the host is busy waiting or polling. Here it keeps reading the status register over and over until the busy bit becomes clear. If the wait is short, this method is a reasonable one, but if the wait is long, then the host should probably switch to another task.

But then how does the host know when the controller has become idle? For this we could arrange for the hardware controller to notify the CPU when the device becomes ready for service, rather than to require the CPU to poll repeatedly for an I/O completion. This hardware mechanism that enables a device to notify the CPU is called an interrupt.

9. Interrupts

3

Oct.15, Apr.13 – *2M*
Define the term Interrupt.
Oct.2014 – *4M*
Explain interrupt in detail.

Interrupts are very important in O.S. most peripheral devices generate interrupts inorder to receive service from the operating system. Interrupts are the principle method of initiating servicing actions by the operating system. The interrupt signal causes the CPU to stop what it is doing and turns its attention to something else.

The basic interrupt mechanism works as follows:

The CPU hardware has a wire called the interrupt request line that the CPU senses after executing every instruction. When the CPU detects that a controller has sent a signal on the interrupt request line. The CPU saves its current state and jumps to the interrupt handler routine at a fixed address in memory.

Thus handler determines the cause of the interrupt, performs the necessary processing and executes a return from interrupt instruction to the CPU execution state prior to the interrupt.

We say that the controller **raises** an interrupt, CPU **catches** the interrupt and **dispatches** it to the interrupt handler. The handler **clears** the interrupt by servicing the device (*figure 9.7*).

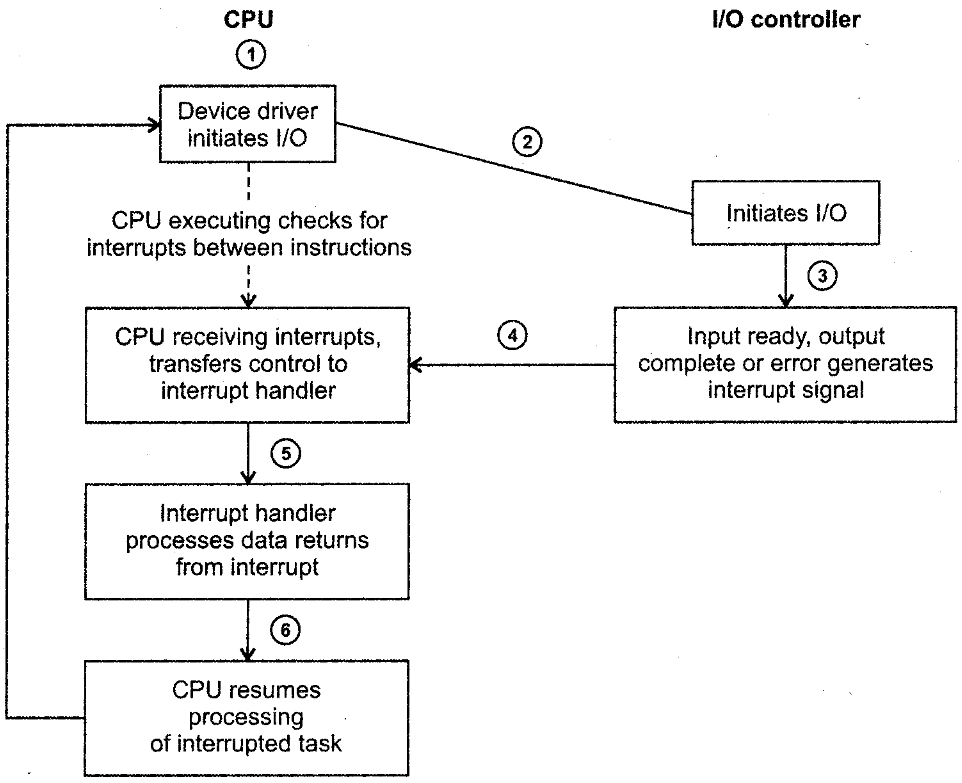

CPU
I/O controller
1
Device driver initiates I/O
2
Initiates I/O
CPU executing checks for interrupts between instructions
3
CPU receiving interrupts, transfers control to interrupt handler
4
Input ready, output complete or error generates interrupt signal
5
Interrupt handler processes data returns from interrupt
6
CPU resumes processing of interrupted task

Figure 9.7: Interrupt driven I/O cycle

In modern operating system, we need more sophisticated interrupt handling features. We need ability to differ interrupt handling during critical processing, also an efficient way to dispatch to the proper interrupt handler. For a device, without first polling all the devices to see which one raised the interrupt and we need multilevel interrupts so that the operating system can distinguish between high and low priority interrupts and can respond with the appropriate degree of urgency.

Most CPU's have two interrupt request lines. One is the **non-maskable** interrupt which is reserved for events such as unrecoverable memory errors. The second interrupt line is **maskable**. It can be turned off by the CPU before the execution of critical instruction sequences that must not be interrupted.

1
Apr.2012 – *4M*
What do you mean by Maskable and Non-maskable Interrupt?

The interrupt mechanism accepts an address which selects the specific interrupt handling routine from a small set. This address is an offset in a table called the interrupt vector.

Table 9.1 shows a typical interrupt vector but in many systems there are more interrupt handler than the address in the interrupt vector so chaining is done where each element in the vector points to the head of a list of interrupt handlers. When an interrupt is raised, the handlers on the list are called one by one until the one is found which can service the request.

The interrupt mechanism also implements a system of interrupt priority levels.

Table 9.1: Event vector table

Vector number	Description
0	Divide error
1	Debug exception
2	Null interrupt
3	Break point
4	INTO detected overflow
5	Bound range exception
6	Invalid opcode
7	Device not available
8	Double fault
9	Co-process segment over run
10	Invalid task state segment
11	Segment not present
12	Stack full
13	Generate protection
14	Page fault
15	(Reserved, do not use)
16	Floating point error
17	Alignment check
18	Machine check
19-31	(Reserved, do not use)
32-255	Maskable interrupts

The interrupt mechanism is also used to handle a wide variety of exceptions, such as dividing by zero, accessing a protected or non-existent memory address or attempting to execute a privileged instruction from user mode.

Solved Examples

1

Apr.2013 – *4M*

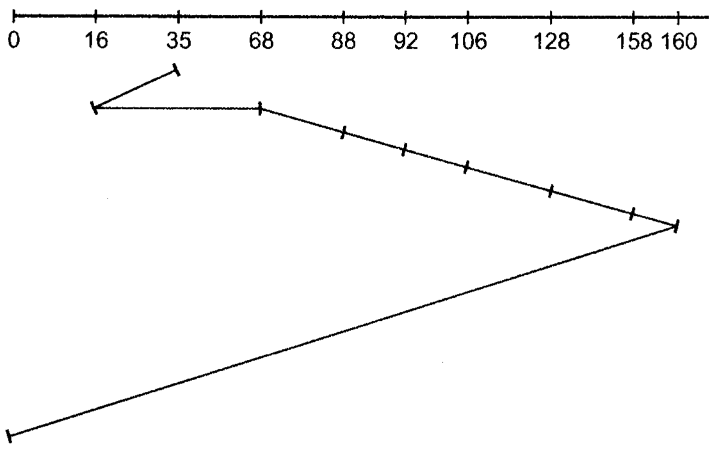
1. **Assume there are total 200 tracks that are present on each surface of the disk. If request queue is 68, 92, 16, 88, 160, 128,158, 106 and initial position of the head is 35. Apply SCAN Disk Scheduling Algorithm and calculate total head movement.**

Solution

Given

Request Queue

68, 92, 16, 88, 160, 128, 158, 106



Arm movements

= (35 – 16) + (68 – 16) + (88 – 68) + (92 – 88) + (106 – 92) + (128 – 106) + (158 – 128) + (160 – 158)

= 19 + 52 + 20 + 04 + 14 + 22 + 30 + 02

= 163

---

2. Assume there are total 200 tracks that are present on each surface of the disk. If request queue is 70, 120, 10 180, 90, 50, 100 and initial position of the head is 105. Apply FCFS Disk Scheduling Algorithm and calculate total head movement.

1 Oct.2012 – *4M*

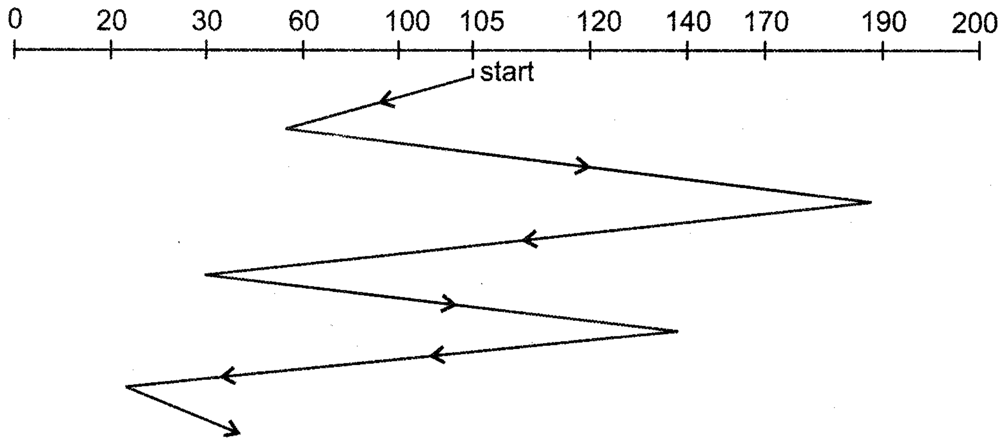
Solution

Given

Total tracks = 200

Request queue: 70, 120, 10 180, 90, 50, 100.

Initial position of head is 105.



Total movement

= (105 – 70) + (120 – 70) + (120 – 10) + (180 – 90) + (100 – 50)

= 35 + 50 + 110 + 90 + 50

= 335

1 Apr.2012 – *4M*

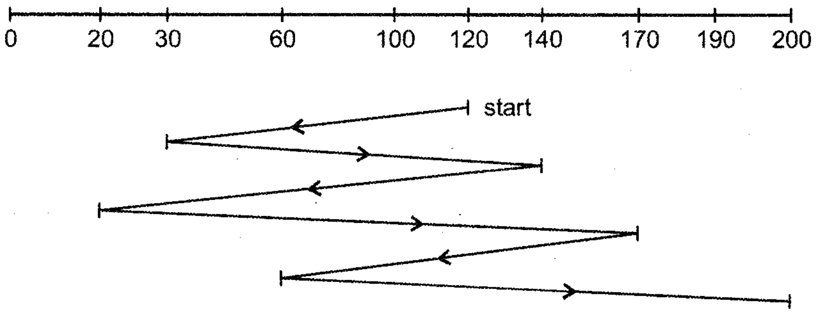
3. Assume there are total 200 tracks that are present on each surface of the disk. If request queue is 30, 140, 20 170, 60, 190 and initial position of the head is 120. Apply FCFS Disk Scheduling and calculate total head movement.

Solution

Given: Total tracks = 200

Request queue: 30, 140, 20, 170, 60, 190. Initial position of head is 120.

FCFs Disk Scheduling



Total movement

= (120 – 30) + (140 – 30) + (140 – 20) + (170 – 20) + (170 – 60) + (190 – 60)

= 90 + 110 + 120 + 150 + 110 + 130

= 710

SUMMARY

- The two main jobs of a computer are I/O and processing.
- To ensure proper operation we must protect the O.S and all other programs and their data from any malfunctioning program. For this purpose there are two separate modes of operation.
  a. User mode b. Monitor mode
- **Seek time:** It is the time for the disk arm to move the heads to the cylinder containing the desired sector.
- **Rotational latency:** It is the additional time waiting for the disk to rotate the desired sector to the disk head.
- **Disk Scheduler:** For a multiprogramming system with many processes, the disk queue may often have several pending requests. Thus when one request is completed the operating system has an opportunity to choose which pending requests to service next. The part of the operating system which makes this decision is called Disk Scheduler.
- There are various disk scheduling algorithm like:
  a. First Come First Serve Scheduling (FCFS)
  b. Shortest Seek Time First Scheduling (SSTF)
  c. Scan scheduling
  d. C-Scan scheduling
  e. Look and C-Look scheduling
- A spool is a buffer that holds output for a device such as a printer, that cannot accept interleaved data stream.
- A cache is a region of fast memory that holds copies of data access to the cached copy is more efficient than access to the original.
- A buffer is a memory area that stores data while they are transferred between two devices.

PU Questions

2 Marks

1. Define the term Interrupt. **[Oct.15,Apr.13 – *2M*]**
2. Explain Block and character devices. **[Oct.2014 – *2M*]**
3. Explain Buffering in detail. **[Oct.2014 – *2M*]**
4. What is Buffering? **[Oct.2012 – *2M*]**
5. What do you mean by Seek Time in Disk Scheduling? **[Oct.2011 – *2M*]**
6. What is meant by Disk Scheduling? **[Apr.2011 – *2M*]**

4 Marks

1. Explain client-server system in detail. [Oct.2015 – 4M]
2. Write a short note on Direct Memory Access (DMA). [Oct.2015 – 4M]
3. Write a short note on Buffering. [Oct.2015 – 4M]
4. The request queue is as follows:28, 136, 15, 185, 50, 197. [Oct.2015 – 4M]
   Number of tracks = 0 to 199.
   Starting position or current head position = 134
   Find total head movement by Applying FCFS (First Come First Serve) disk scheduling algorithm.
5. Explain Direct Access method with advantages and disadvantages [Apr.2015 – 4M]
6. The request queue is as follows: [Apr.2015 – 4M]
   87, 148, 92, 171, 96, 131, 103, 71
   Number of tracks = 0 to 199.
   Starting position or current head position = 125
   Find total head movement by Applying SSTF (Shortest seek time first) disk scheduling Algorithm.
7. Explain interrupt in detail. [Oct.2014 – 4M]
8. Assume there are total 200 tracks that are present on each surface of the disk. If request queue is 86, 147, 91, 170, 95, 130, 102, 70 And initial position of the head is 125 Apply C-look. Disk scheduling Algorithm and calculate total head movement. [Oct.2014 – 4M]
9. Explain Buffering in detail. [Oct.2014 – 4M]
10. Explain in brief different services provided by Kernel related to I/O. [Apr.2013 – 4M]
11. Assume there are total 200 tracks that are present on each surface of the disk. If request queue is 68, 92, 16, 88, 160, 128,158, 106 and initial position of the head is 35. Apply SCAN Disk Scheduling Algorithm and calculate total head movement. [Apr.2013 – 4M]
12. What is polling and how it is achieved to control more than one device? [Apr.2013 – 4M]
13. Assume there are total 200 tracks that are present on each surface of the disk. If request queue is 70, 120, 10 180, 90, 50, 100 and initial position of the head is 105. Apply FCFS Disk Scheduling Algorithm and calculate total head movement. [Oct.2012 – 4M]
14. What do you mean by Maskable and Non-maskable Interrupt? [Apr.2012 – 4M]
15. Assume there are total 200 tracks that are present on each surface of the disk. If request queue is 30, 140, 20 170, 60, 190 and initial position of the head is 120. Apply FCFS Disk Scheduling and calculate total head movement. [Apr.2012 – 4M]

Suggestive Readings:

1. Andrew M. Lister, Fundamentals of Operating Systems,
2. Wiley. Andrew S. Tanenbaum and Albert S. Woodhull, Systems Design and Implementation, Prentice Hall.
3. Andrew S. Tanenbaum, Modern Operating System, Prentice Hall.
4. Colin Ritchie, Operating Systems, BPB Publications.
5. Deitel H.M., “Operating Systems, 2nd Edition, Addison Wesley.
6. I.A. Dhotre, Operating System, Technical Publications.
7. Milankovic, Operating System, Tata MacGraw Hill, New Delhi.
8. Silberschatz, Gagne & Galvin, “Operating System Concepts”, John Wiley & Sons, Seventh Edition.
9. Stalling, W., “Operating Systems”, 2nd Edition, Prentice Hall.
10. File systems: design & implementation by Daniel Grosshans
11. Practical File System Design with the Be File System By Dominic Giampaolo Be, Inc.