Introduction:

Define operating system (OS)

An operating system is a set of program that controls, co-ordinates and supervises the activities of the computer hardware and software.

Role of an os

An OS acts as an interface between the user and the computer. It acts as the manager of the resources of the computer.

Need for an OS

A medium is needed to communicate between the user and the machine. An OS acts as a medium of interface

Characteristics of an OS

(i) User friendly .

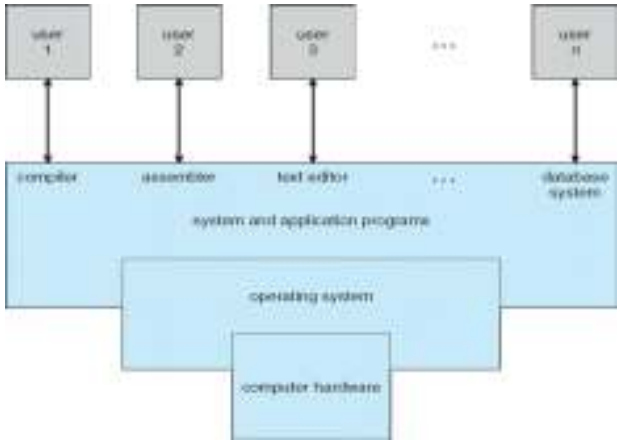
(ii) Keep track of the status of each RESOURCE.

(iii) Allows sharing of resources (H/W and S/W).

(iv) Provides adequate security.

(v) Protection.

Four Components of a Computer System



OS GOALS& FUNCTIONS

- Operating system goals:
    1. Execute user programs and make solving user problems easier
    2. Make the computer system convenient to use
    3. Use the computer hardware in an efficient manner
- **The primary goal** of an operating system is a convenience for the user.

1. Operating systems exit because they are supposed to make it easier to compute with an operating system than without an operating system.
2. This is particularly clear when you look at operating system for small personal computers.

- **A secondary goal** is the efficient operation of an computer system.

1. This goal is particularly important for large, shared multi-user systems.
2. Operating systems can solve this goal.
3. It is known that sometimes these two goals, convenience and efficiency, are contradictory.

FUNCTIONS OF OS

1. Process Management
2. Memory Management
3. I/O Management
4. File Management
5. Security
6. Control over system performance
7. Job accounting
8. Error detecting aids
9. Coordination between other software and users

- **Process Management :** The Operating System also Treats the Process Management means all the Processes those are given by the user or the Process those are System ‘s own Process are Handled by the Operating System .
- **Memory Management:** Operating System also Manages the Memory of the Computer System means Provide the Memory to the Process and Also Deallocate the Memory from the Process. And also defines that if a Process gets completed then this will deallocate the Memory from the Processes.
- **Device Management or IO *Management***

1. An Operating System manages device communication via their respective drivers. It does the following activities for device management −
2. Keeps tracks of all devices. Program responsible for this task is known as the **I/O controller**.
3. Decides which process gets the device when and for how much time.
4. Allocates the device in the efficient way.
5. De-allocates devices.

- **File Management**

1. A file system is normally organized into directories for easy navigation and usage.
2. These directories may contain files and other directions.
3. An Operating System does the following activities for file management −
4. Keeps track of information, location, uses, status etc. The collective facilities are often known as **file system**.
5. Decides who gets the resources.

6. Allocates the resources.
7. De-allocates the resources.

HISTORY OF OPERATING SYSTEMS

- Since operating systems have historically been closely tied to the architecture of the computers on which they run, we will look at successive generations of computers to see what their operating systems were like.
- **This mapping of operating system generations to computer generations is crude, but it does provide some structure where there would otherwise be none.**

Introduction

- The first true digital computer was designed by the English mathematician Charles Babbage (1792-1871).
- Although Babbage spent most of his life and fortune trying to build his **"analytical engine."** he never got it working properly because it was purely mechanical, and the technology of his day could not produce the required wheels, gears, and cogs to the high precision that he needed.
- *Needless to say, the analytical engine did not have an operating system.* As an interesting historical aside, Babbage realized that he would need software for his analytical engine, so he hired a young woman named Ada Lovelace, who was the daughter of the famed British poet Lord Byron, as the world's first programmer. The programming language Ada is named after her.

Generations

- The First Generation (1945-55) Vacuum Tubes and Plugboards
- The Second Generation (1955-65) Transistors and Batch Systems
- The Third Generation (1965-1980) ICs and Multiprogramming
- The Fourth Generation (1980-Present) Personal Computers
- Ontogeny Recapitulates Phytogeny

1940 – 1956: First Generation – Vacuum Tubes

- These early computers used vacuum tubes as circuitry and magnetic drums for memory. As a result they were enormous, literally taking up entire rooms and costing a fortune to run.
- These were inefficient materials which generated a lot of heat, sucked huge electricity and subsequently generated a lot of heat which caused ongoing breakdowns.
- These first generation computers relied on 'machine language' (which is the most basic programming language that can be understood by computers).
- These computers were limited to solving one problem at a time. Input was based on punched cards and paper tape. Output came out on print-outs.
- The two notable machines of this era were the UNIVAC and ENIAC machines – the UNIVAC is the first every commercial computer which was purchased in 1951 by a business – the US Census Bureau.

1956 – 1963: Second Generation – Transistors

- The replacement of vacuum tubes by transistors saw the advent of the second generation of computing. Although first invented in 1947, transistors weren't used significantly in computers until the end of the 1950s.

- They were a big improvement over the vacuum tube, despite still subjecting computers to damaging levels of heat. However they were hugely superior to the vacuum tubes, making computers smaller, faster, cheaper and less heavy on electricity use. They still relied on punched card for input/printouts.
- The language evolved from cryptic binary language to symbolic ('assembly') languages. These meant programmers could create instructions in words.
- About the same time high level programming languages were being developed (early versions of COBOL and FORTRAN). Transistor-driven machines were the first computers to store instructions into their memories – moving from magnetic drum to magnetic core 'technology'. The early versions of these machines were developed for the atomic energy industry.

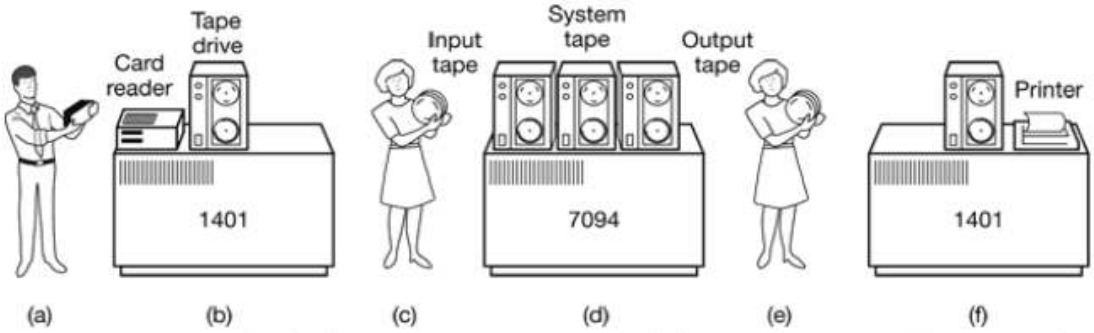


Figure 1-2. An early batch system. (a) Programmers bring cards to 1401. (b) 1401 reads batch of jobs onto tape. (c) Operator carries input tape to 7094. (d) 7094 does computing. (e) Operator carries output tape to 1401. (f) 1401 prints output.

1964 – 1971: Third Generation – Integrated Circuits

- By this phase, transistors were now being miniaturized and put on silicon chips (called semiconductors).
- This led to a massive increase in speed and efficiency of these machines.

- These were the **first computers where users interacted using keyboards and monitors which interfaced with an operating system, a significant leap up from the punch cards and printouts.** This enabled these machines to run several applications at once using a central program which functioned to monitor memory.
- **As a result of these advances which again made machines cheaper and smaller, a new mass market of users emerged during the '60s.**

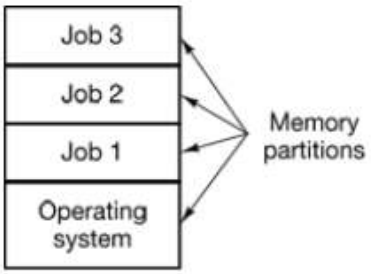


Figure 1-4. A multiprogramming system with three jobs in memory.

1972 – 2010: Fourth Generation – Microprocessors

- This revolution can be summed in one word: Intel.
- The chip-maker developed the Intel 4004 chip in 1971, which positioned all computer components (CPU, memory, input/output controls) onto a single chip.
- What filled a room in the 1940s now fit in the palm of the hand. The Intel chip housed thousands of integrated circuits.
- **The year 1981 saw the first ever computer (IBM) specifically designed for home use** and 1984 saw the MacIntosh introduced by Apple. Microprocessors even moved beyond the realm of computers and into an increasing number of everyday products.

- The increased power of these small computers meant they could be linked, **creating networks. This ultimately led to the development, birth and rapid evolution of the Internet.**
- Other major advances during this period have been the **Graphical user interface (GUI), the mouse and more recently the astounding advances in lap-top capability and hand-held devices.**

2010- : Fifth Generation – Artificial Intelligence

- Computer devices with artificial intelligence are still in development, but some of these technologies are beginning to emerge and be used such as voice recognition.
- AI is a reality made possible by using parallel processing and superconductors. Leaning to the future, computers will be radically transformed again by quantum computation, molecular and nano technology.
- The essence of fifth generation will be using these technologies to ultimately create machines which can process and respond to natural language, and have capability to learn and organise themselves.

Types of Operating System

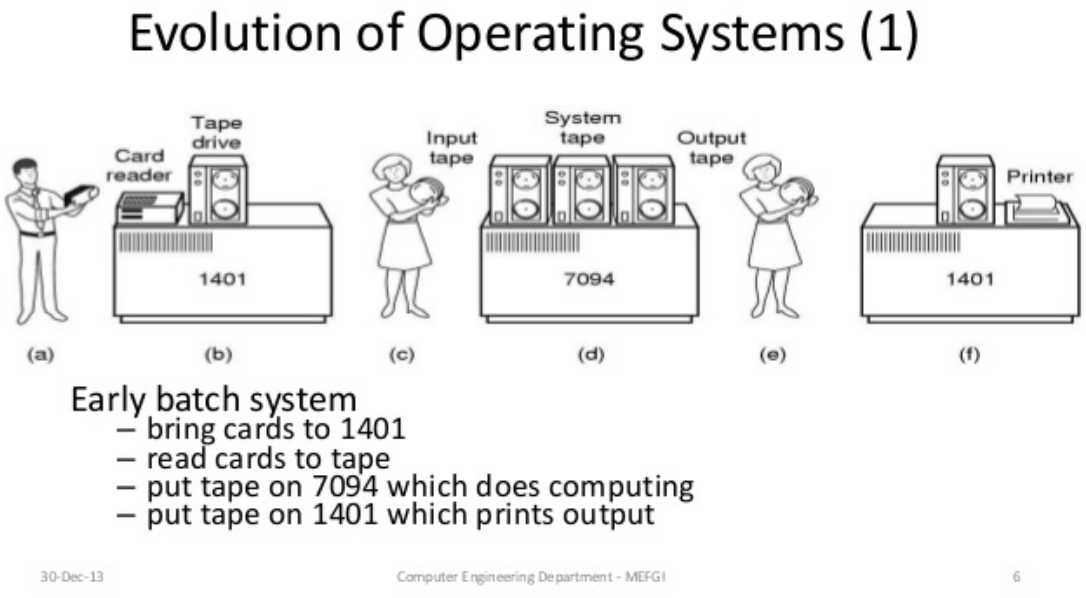
1. Batch operating system
2. Real-time
3. Multi-user vs. Single-user
4. Multi-tasking vs. Single-tasking
5. Single-processor Systems
6. Multi-processor Systems
7. Distributed:
8. Embedded

Batch operating system

- The users of a batch operating system do not interact with the computer directly.
- Each user prepares his job on an off-line device like punch cards and submits it to the computer operator.
- To speed up processing, jobs with similar needs are batched together and run as a group.
- The programmers leave their programs with the operator and the operator then sorts the programs with similar requirements into batches.

The problems with Batch Systems are as follows −

- Lack of interaction between the user and the job.
- CPU is often idle, because the speed of the mechanical
- I/O devices is slower than the CPU.
- Difficult to provide the desired priority.



Real-time Operating System

- A real-time operating system is a multitasking operating system that aims at executing real-time applications.
- Real-time operating systems often use specialized scheduling algorithms so that they can achieve a deterministic nature of behavior.
- *The main object of real-time operating systems is their quick and predictable response to events.*

  a. **Hard real-time system:** It has the most stringent requirements, guaranteeing that real-time tasks be completed within their deadlines. Safety-critical systems are typically hard real-time systems

  b. **Soft real-time system:** It is less restrictive, simply providing that a critical real-time task will receive priority over other tasks and that it will retain that priority until it completes. Many commercial operating systems – as well as Linux – provide soft real-time support

Multi-user vs. Single-user

- **A multi-user operating system** allows multiple users to access a computer system concurrently. Time-sharing system can be classified as multi-user systems as they enable a multiple user access to a computer through the sharing of time.
- **Single-user operating systems**, as opposed to a multi-user operating system, are usable by a single user at a time.

Multi-tasking vs. Single-tasking

- When a single program is allowed to run at a time, the system is grouped under a single-tasking system, while in case the operating system allows the execution of multiple tasks at one time, it is classified as a multi-tasking operating system.

- Multi-tasking can be of two types namely, pre-emptive or co-operative.
  1. In **pre-emptive multitasking**, the operating system slices the CPU time and dedicates one slot to each of the programs. Unix-like operating systems such as Solaris and Linux support pre-emptive multitasking.
  2. **Cooperative multitasking** is achieved by relying on each process to give time to the other processes in a defined manner. MS Windows prior to Windows 95 used to support cooperative multitasking

Single-processor Systems

- On a single-processor system, there is one main CPU capable of executing a general-purpose instruction set, including instructions from user processes

Multi-processor Systems

- A multiprocessing operating system allows a program to run on more than one central processing unit (CPU) at a time.
- This can come in very handy in some work environments, at schools, and even for some home-computing situations.

Distributed

- A distributed operating system manages a group of independent computers and makes them appear to be a single computer.
- The development of networked computers that could be linked and communicate with each other, gave rise to distributed computing.
- Distributed computations are carried out on more than one machine. When computers in a group work in cooperation, they make a distributed system.

Embedded

- Embedded operating systems are designed to be used in embedded computer systems.
- They are designed to operate on small machines like PDAs with less autonomy.
- They are able to operate with a limited number of resources.
- They are very compact and extremely efficient by design. Windows CE and Minix 3 are some examples of embedded operating systems

COMPUTER HARDWARE REVIEW

1. An operating system is intimately tied to the hardware of the computer it runs on.
2. It extends the computer's instruction set and manages its resources.

- Processors
- Memory
- I/O Devices
- Buses

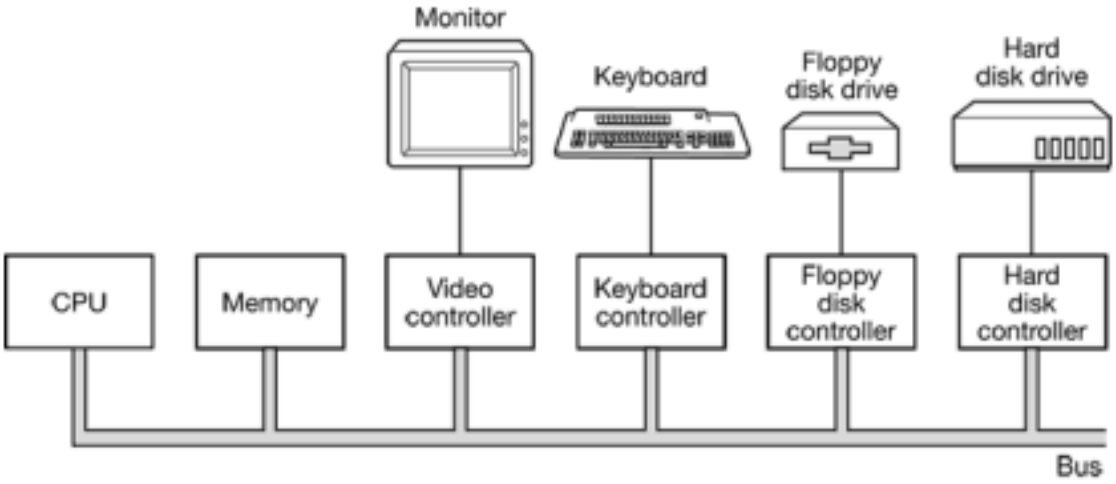


Figure 1-5. Some of the components of a simple personal computer.

Processors

- The "brain" of the computer is the CPU.
- It fetches instructions from memory and executes them.
- The basic cycle of every CPU is to fetch the first instruction from memory, decode it to determine its type and operands, execute it, and then fetch, decode, and execute subsequent instructions. In this way, programs are carried out.

Memory

- The second major component in any computer is the memory.
- Ideally, a memory should be extremely fast (faster than executing an instruction so the CPU is not held up by the memory), abundantly large, and dirt cheap.
- The memory system is constructed as a hierarchy of layers, as shown in Fig. 1-7.

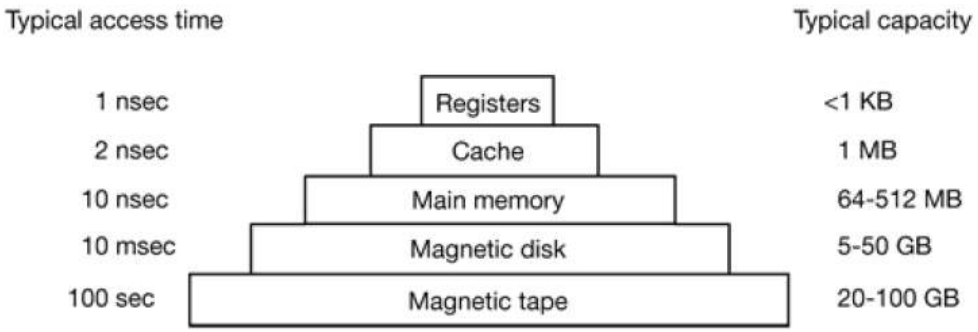


Figure 1-7. A typical memory hierarchy. The numbers are very rough approximations.

I/O Devices

- Memory is not the only resource that the operating system must manage.
- I/O devices also interact heavily with the operating system.

- As we saw in Fig. 1-5, I/O devices generally consist of two parts: a controller and the device itself.
- The controller is a chip or a set of chips on a plug-in board that physically controls the device.
- It accepts commands from the operating system, for example, to read data from the device, and carries them out.

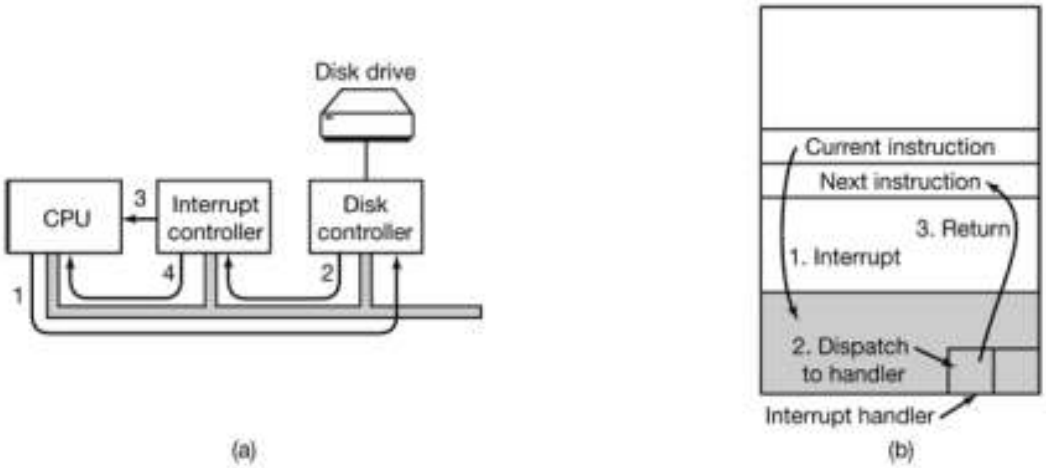


Figure 1-10. (a) The steps in starting an I/O device and getting an interrupt. (b) Interrupt processing involves taking the interrupt, running the interrupt handler, and returning to the user program.

Buses

- The organization of Fig. 1-5 was used on minicomputers for years and also on the original IBM PC.
- However, as processors and memories got faster, the ability of a single bus (and certainly the IBM PC bus) to handle all the traffic was strained to the breaking point. Something had to give.
- As a result, additional buses were added, both for faster I/O devices and for CPU to memory traffic.

- As a consequence of this evolution, a large Pentium system currently looks something like Fig. 1-11.

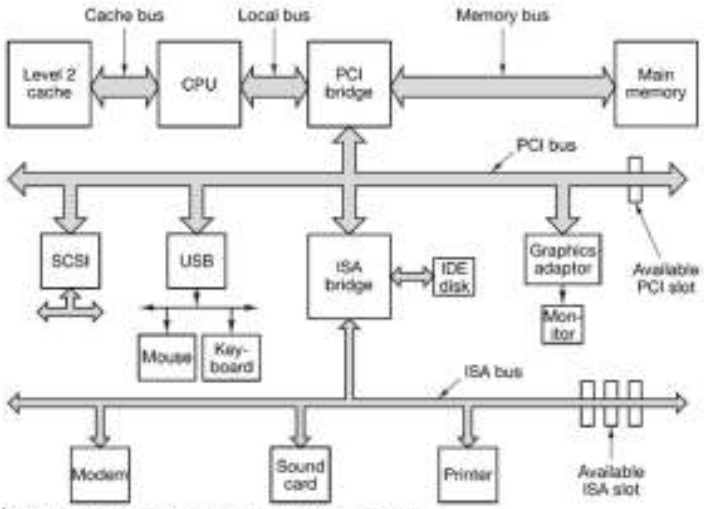


Figure 1-11. The structure of a large Pentium system

OPERATING SYSTEM CONCEPTS

- Processes
- Memory Management
- Input/Output
- Files
- Security

Process Management : The Operating System also Treats the Process Management means all the Processes those are given by the user or the Process those are System ‘s own Process are Handled by the Operating System .

Memory Management: Operating System also Manages the Memory of the Computer System means Provide the Memory to the Process and Also Deallocate the Memory from the Process. And also defines that if a Process gets completed then this will deallocate the Memory from the Processes.

Device Management or IO *Management*

1. An Operating System manages device communication via their respective drivers. It does the following activities for device management
2. Keeps tracks of all devices. Program responsible for this task is known as the **I/O controller**.
3. Decides which process gets the device when and for how much time.
4. Allocates the device in the efficient way.
5. De-allocates devices.

File Management

1. A file system is normally organized into directories for easy navigation and usage.
2. These directories may contain files and other directions.
3. An Operating System does the following activities for file management −
4. Keeps track of information, location, uses, status etc. The collective facilities are often known as **file system**.
5. Decides who gets the resources.
6. Allocates the resources.
7. De-allocates the resources.

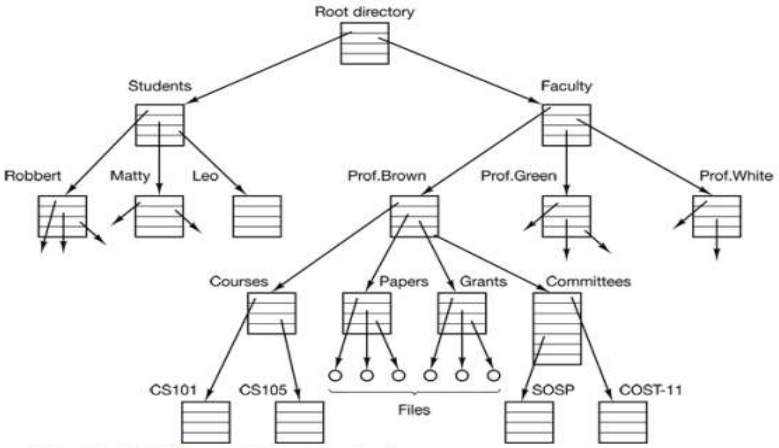


Figure 1-14. A file system for a university department.

Security

- Computers contain large amounts of information that users often want to keep confidential.
- This information may include electronic mail, business plans, tax returns, and much more.
- It is up to the operating system to manage the system security so that

  files, for example, are only accessible to authorized users.
- Files in UNIX are protected by assigning each one a 9-bit binary protection code.
- The protection code consists of three 3-bit fields, one for the owner, one for other members of the owner's group (users are divided into groups by the system administrator), and one for everyone else.
- These 3 bits are known as the **rwx** bits.
- For example, the protection code ***rwxr-x--x*** **means**
- that the owner can read, write, or execute the file,

- other group members can read or execute (but not write) the file, and
- everyone else can execute (but not read or write) the file.
- For a directory, *x* indicates search permission. A dash means that the corresponding permission is absent.

System Call

- As we know that for performing any Operation as user must have to specify the Operation which he wants to Operate on the Computer.
- We can say that For Performing any Operation a user must have to Request for a Service from the System.
- For Making any Request a user will prepare a Special call which is also known as the **System Call**.
- The System Call is the Request for Running any Program and for Performing any Operation on the System. When a user First Time Starts the System then the **System is in the user Mode and When he request For a Service then the User Mode will be Converted into the Kernel Mode** Which just Listen the Request of the user and Process the Request and Display the Results those are Produced after the Processing.
- When a user Request for Opening any Folder or When a Moves his Mouse his Mouse on the Screen, then this is called as the System call which he is using for performing any Operation.

Categories of System Calls

System calls can be grouped into five major categories as follows.

- **Process control**
- **File management.**

- **Device management**
- **Information Maintenance and**
- **Communication.**

Process control

Some system calls under process control are:

- End, abort
- Load, execute
- Create process, terminate process
- Get process, terminate process
- Wait for time
- Allocate and free memory

File management

Some system calls under file management are:

- Create file, delete file
- Open , close
- Read, write, reposition.
- Get file attributes, set fil attributes

Device management

Some system calls under device management are:

- Request Device, release device
- Read, write, reposition.
- Get device attributes and set device attributes
- Logically attach or detach devices

Information Maintenance

Some system calls under information maintenance are:

- Get time or date, Set time of date
- Logically attach or detach devices

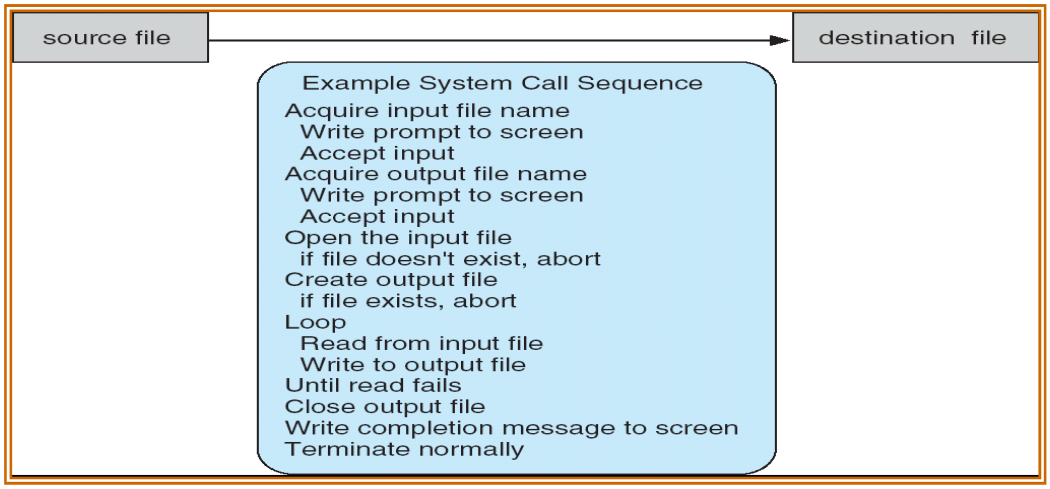
- Information maintenance
- Get system data, Set Systems data
- Get process, file of device attributes
- Set process, file or device attributes

Communication

- Create, delete communication connection.
- Send, receive messages
- Transfer status information
- Attach or detach remote devices.

Example of System Calls

System call sequence to copy the contents of one file to another file



OPERATING SYSTEM STRUCTURE

❖ Now that we have seen what operating systems look like on the outside (i.e., theprogrammer's interface), it is time to take a look inside.

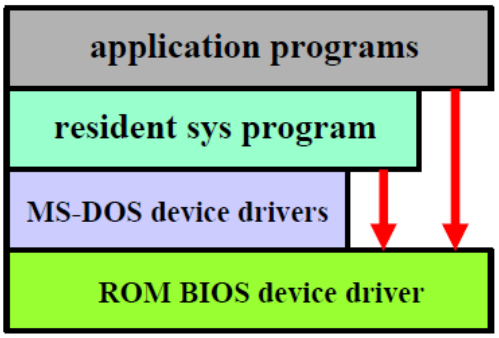
- In the following sections, we will examine five different structures that have been tried, in order to get some idea of the spectrum of possibilities.
- These are by no means exhaustive, but they give an idea of some designs that have been tried in practice.

- The five designs are
  1. monolithic systems,
  2. layered systems,
  3. virtual machines,
  4. exokernels, and
  5. client-server systems.

Simple (i.e. monolithic)

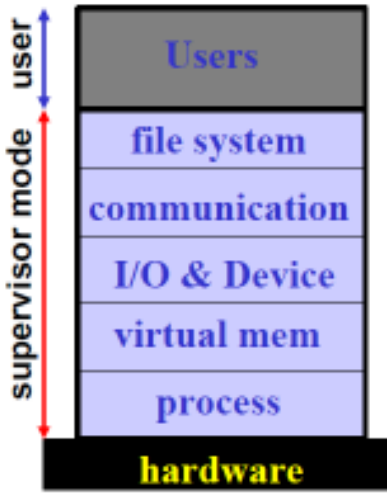
1. Only one or two levels of code
2. Simple structure systems do not have well-defined structures
3. The Unix only had limited structure: kernel and system programs
4. Everything between the system call interface and physical hardware is the kernel.

MS-DOS structure



Layered

1. Lower levels independent of upper levels
2. The operating system is broken up into a number of layers (or levels), each on top of lower layers.
3. Each layer is an implementation of an abstract object that is the encapsulation of data and operations that can manipulate these data.
4. The bottom layer (layer 0) is the hardware.
5. The main advantage of layered approach is *modularity*.



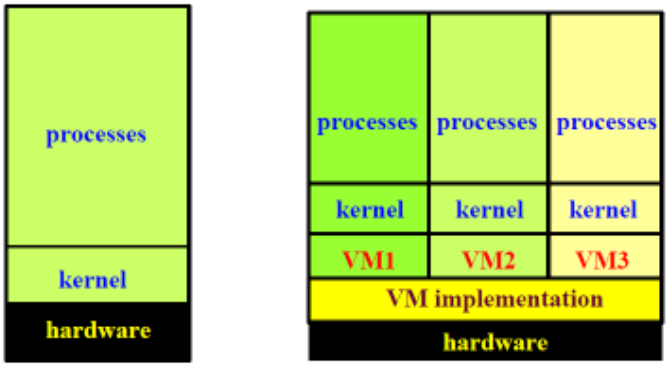
- The lowest layer is *process management*.
- Each layer *only uses the operations provided by lower layers* and does not have to know their implementation.
- Each layer hides the existence of certain data structures, operations and hardware from higher-level layers.

Virtual Machines:

- A virtual machine, VM, is a software between the kernel and hardware.

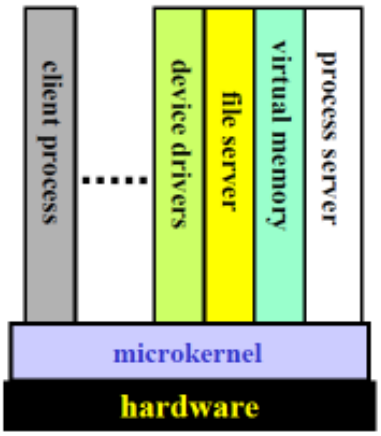
- Thus, a VM provides all functionalities of a CPU with software simulation.
- A user has the illusion that s/he has a real processor that can run a kernel.

Virtual Machines: 2/4



Microkernel

1. OS built from many user-level processes
2. Only *absolutely essential core* OS functions should be in the kernel.
3. Less essential services and applications are built on the kernel and *run in user mode*.
4. Many functions that were in a traditional OS become external subsystems that interact with the kernel and with each other.
5. The main function of the microkernel is to provide communication facility between the client program and various services.
6. Communication is provided by *message passing*.



Client Server Model

1. In this model, shown in Fig. 1-27, all the kernel does is handle the communication between clients and servers.

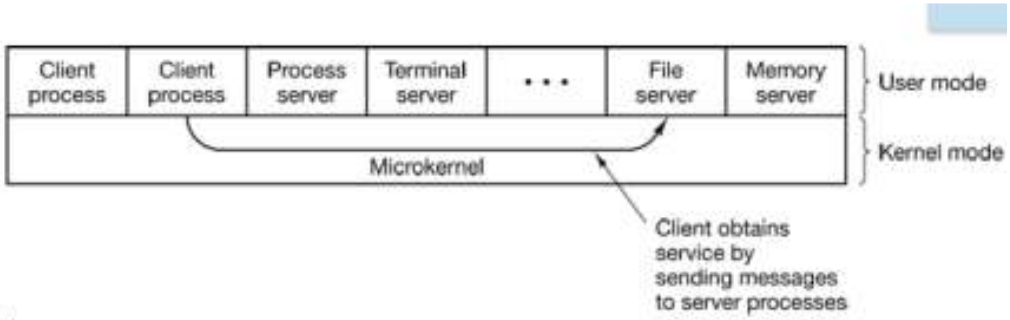


Figure 1-27. The client-server model.

2. By splitting the operating system up into parts, each of which only handles one facet of the system, such as file service, process service, terminal service, or memory service, each part becomes small and manageable.
3. Furthermore, because all the servers run as user-mode processes, and not in kernel mode, they do not have direct access to the hardware.
4. As a consequence, if a bug in the file server is triggered, the file service may crash, but this will not usually bring the whole machine down.

5. Another advantage of the client-server model is its adaptability to use in distributed systems (see Fig. 1-28).

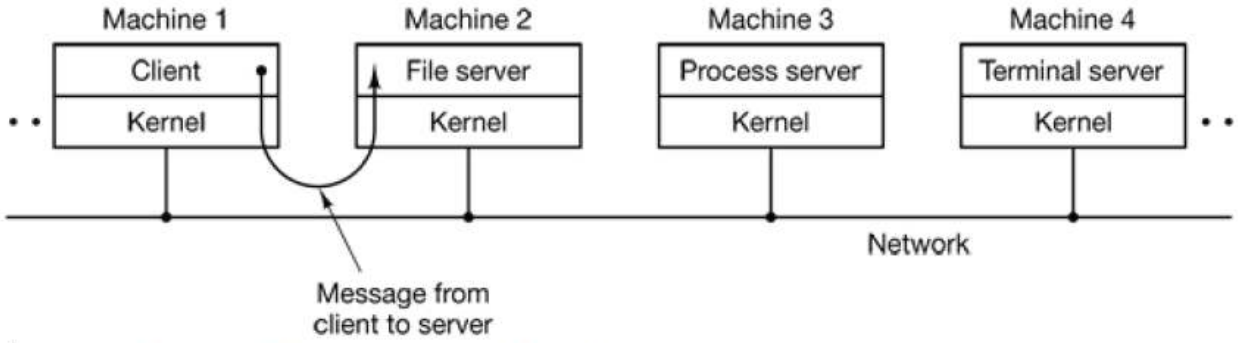


Figure 1-28. The client-server model in a distributed system.

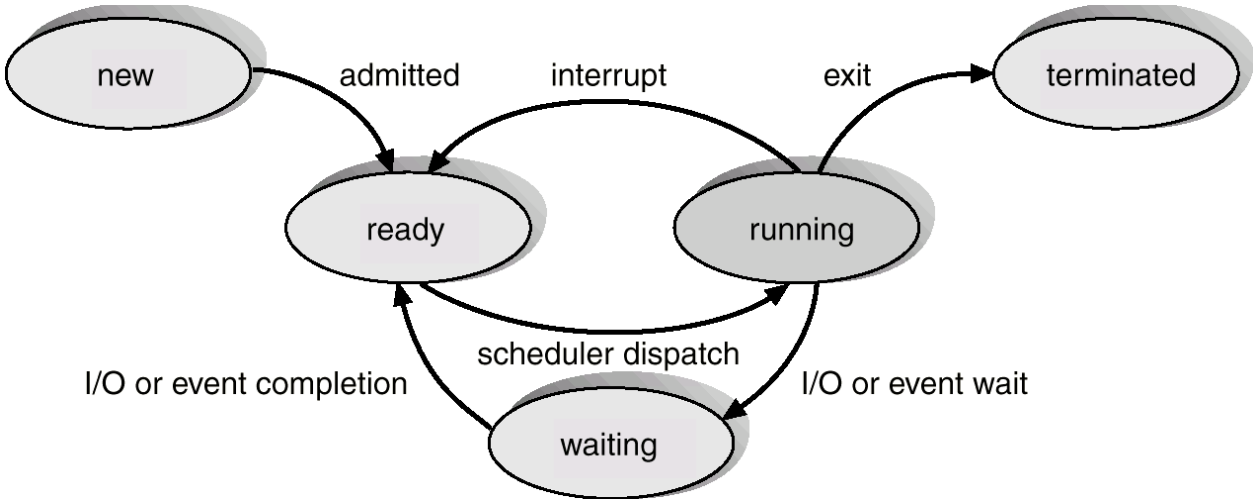
Unit II

What is a Process

A program in the execution is called a Process. Process is not the same as program. A process is more than a program code. A **process is an 'active' entity** as opposed to **program which is considered to be a 'passive' entity.**

Process States

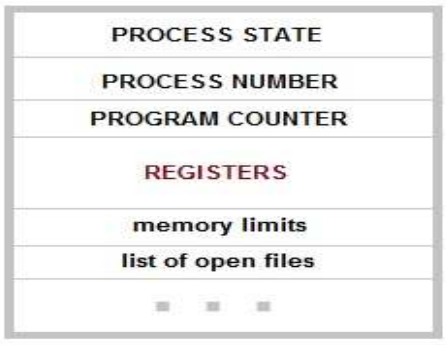
1. **New** The process is just being put together.
2. **Running** Instructions being executed. This running process holds the CPU.
3. **Waiting** For an event (hardware, human, or another process.)
4. **Ready** The process has all needed resources - waiting for CPU only.
5. **Suspended** Another process has explicitly told this process to sleep. It will be awakened when a process explicitly awakens it.
6. **Terminated** The process is being torn apart.



Process Control Block

There is a Process Control Block for each process, enclosing all the information about the process. It is a data structure, which contains the following :

- Process State - It can be running, waiting etc.
- Process ID and parent process ID.
- CPU registers and Program Counter. **Program Counter** holds the address of the next instruction to be executed for that process.
- CPU Scheduling information - Such as priority information and pointers to scheduling queues.
- Memory Management information - Eg. page tables or segment tables.
- Accounting information - user and kernel CPU time consumed, account numbers, limits, etc.
- I/O Status information - Devices allocated, open file tables, etc.



Process Scheduling

The act of determining which process in the ready state should be moved to the running state is known as **Process Scheduling.**

Schedulers fell into one of the two general categories :

- **Non pre-emptive scheduling.** When the currently executing process gives up the CPU voluntarily.
- **Pre-emptive scheduling.** When the operating system decides to favour another process, pre-empting the currently executing process.

Scheduling Queues

- All processes when enters into the system are stored in the **job queue**.
- Processes in the Ready state are placed in the **ready queue**.
- Processes waiting for a device to become available are placed in **device queues**. There are unique device queues for each I/O device available.

Types of Schedulers

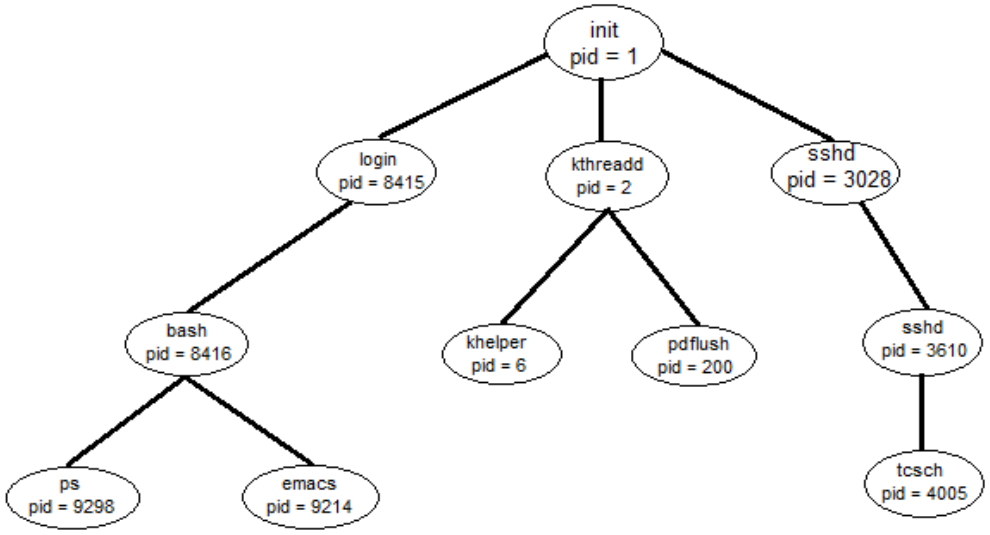
There are three types of schedulers available :

- **Long Term Scheduler** :Long term scheduler runs less frequently**. Long Term Schedulers decide which program must get into the job queue**. From the job queue, the Job Processor, selects processes and loads them into the memory for execution. **Primary aim of the Job Scheduler is to maintain a good degree of Multiprogramming**. An optimal degree of Multiprogramming means the average rate of process creation is equal to the average departure rate of processes from the execution memory.
- **Short Term Scheduler** :**This is also known as CPU Scheduler and runs very frequently. The primary aim of this scheduler is to enhance CPU performance and increase process execution rate.**
- **Medium Term Scheduler** :During extra load, this scheduler picks out big processes from the ready queue for some time, to allow

smaller processes to execute, thereby reducing the number of processes in the ready queue.

Operations on Process

- Through appropriate system calls, such as fork or spawn, processes may create other processes.
- The process which creates other process, is termed the **parent** of the other process, while the created sub-process is termed its **child**.



A Tree of processes on a typical Linux system

Process Termination

After a process has been created, it starts running and does whatever its job is. However, nothing lasts forever, not even processes. Sooner or later the new process will terminate, usually due to one of the following conditions:

1. Normal exit (voluntary).
2. Error exit (voluntary).
3. Fatal error (involuntary).
4. Killed by another process (involuntary).

Threads

- *A thread is a single sequence stream within in a process*. Because threads have some of the properties of processes, they are sometimes called *lightweight processes*

Processes Vs Threads

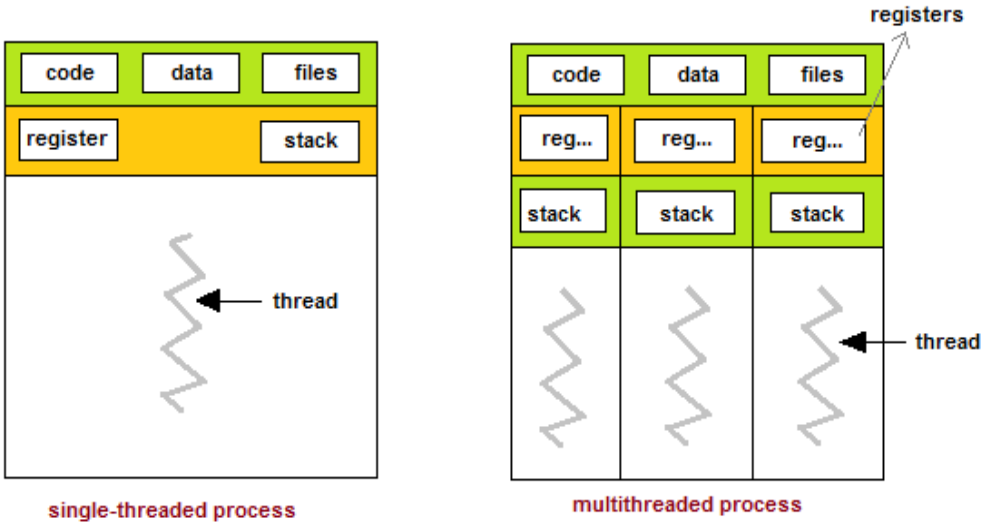
As we mentioned earlier that in many respect threads operate in the same way as that of processes. Some of the similarities and differences are:

Similarities

- Like processes threads share CPU and only one thread active (running) at a time.
- Like processes, threads within a processes, threads within a processes execute sequentially.
- Like processes, thread can create children.
- And like process, if one thread is blocked, another thread can run.

Differences

- Unlike processes, threads are not independent of one another.
- Unlike processes, all threads can access every address in the task .
- Unlike processes, thread are design to assist one other. Note that processes might or might not assist one another because processes may originate from different users.



Types of Thread

There are two types of threads :

1. **User Threads**
2. **Kernel Threads**

- **User threads**, are above the kernel and without kernel support. These are the threads that application programmers use in their programs.
- **Kernel threads** are supported within the kernel of the OS itself. All modern OSs support kernel level threads, allowing the kernel to perform multiple simultaneous tasks and/or to service multiple kernel system calls simultaneously.

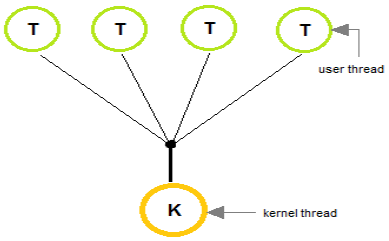
Thread Models

The user threads must be mapped to kernel threads, by one of the following strategies.

1. Many-To-One Model
2. One-To-One Model
3. Many-To-Many Model

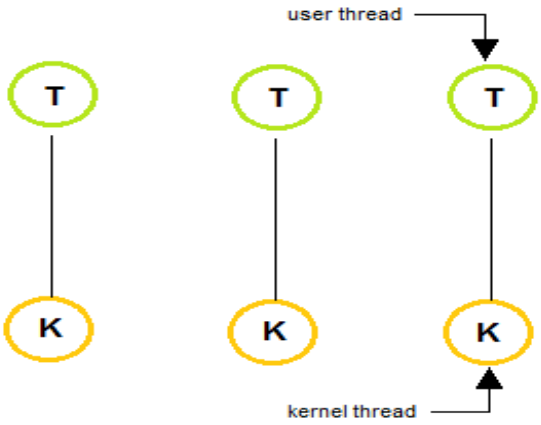
Many-To-One Model

- In the many-to-one model, many user-level threads are all mapped onto a single kernel thread.
- Thread management is handled by the thread library in user space, which is efficient in nature.



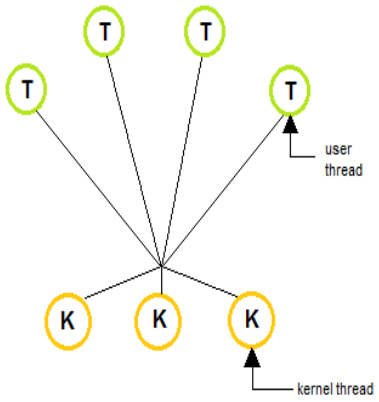
One-To-One Model

- The one-to-one model creates a separate kernel thread to handle each and every user thread.
- Most implementations of this model place a limit on how many threads can be created.
- Linux and Windows from 95 to XP implement the one-to-one model for threads.



Many-To-Many Model

- The many-to-many model multiplexes any number of user threads onto an equal or smaller number of kernel threads, combining the best features of the one-to-one and many-to-one models.
- Users can create any number of the threads.
- Blocking the kernel system calls does not block the entire process.
- Processes can be split across multiple processors.



Thread Usage:

1. Responsiveness
2. Resource sharing, hence allowing better utilization of resources.
3. Economy. Creating and managing threads becomes easier.
4. Scalability. One thread runs on one CPU. In Multithreaded processes, threads can be distributed over a series of processors to scale.
5. Context Switching is smooth. Context switching refers to the procedure followed by CPU to change from one task to another.

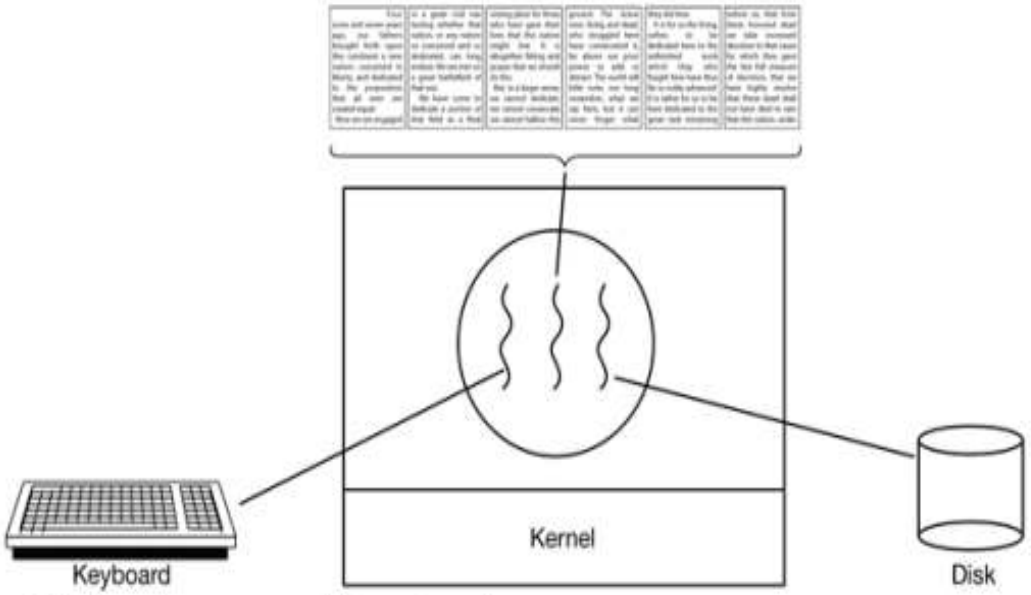


Figure 2-9. A word processor with three threads.

INTERPROCESS COMMUNICATION

There are two types of process

- Cooperating process
- Independent process

Independent process

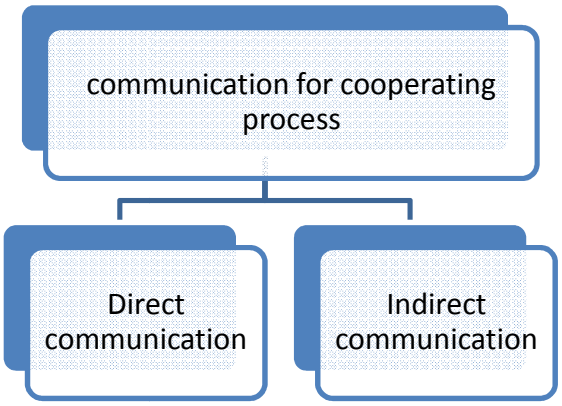
- **one that is independent of the rest of the universe.**
  - Its state is not shared in any way by any other process.
  - Deterministic: input state alone determines results.
  - Reproducible.
  - Can stop and restart with no bad effects (only time varies).
  - Example: program that sums the integers from 1 to i (input).

Co-operating Processes

- A process is Co-operating if it can affect or is affected by the other processes executing in the System.
- That means any process that shares data with other processes is a Co-operating Process.

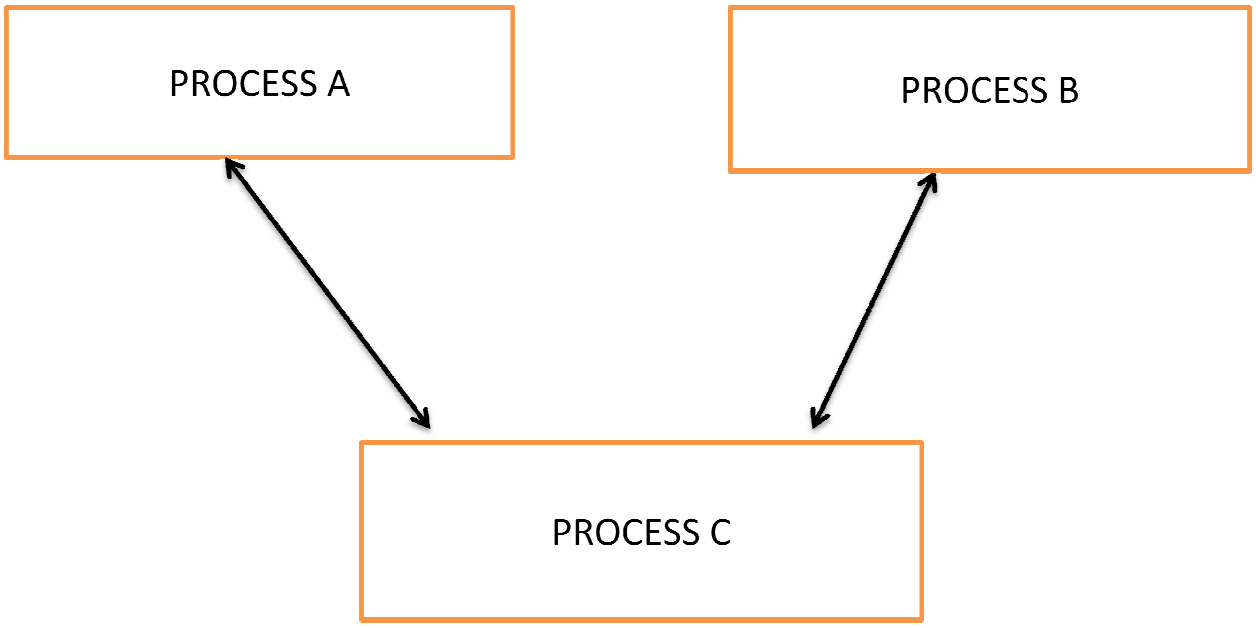
Inter-Process Communication

- Cooperating processes can communicate in a shared-memory environment.
- Cooperating processes communicate with each other via an Inter-Process-Communication (IPC) facility.
- IPC provides a mechanism to allow processes to communicate and to synchronize their actions.
- Inter-Process Communication is best provided by a Message System.
- Message System can be defined in many different ways.
- An IPC facility provides at least the two operations - send(message) and receive(message)



Direct Communication

- In the Direct Communication, each process that wants to communicate must explicitly name the recipient or sender of the communication.
- In this scheme, the send and receive primitives are defined as follows :
- **Send (P, message) - Send a message to process P.**
- **Receive (Q, message) - Receive a message from process Q.**

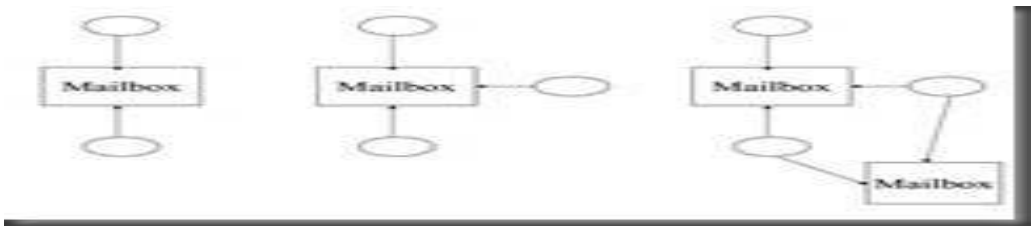


In Direct Communication

- With Indirect Communication, the messages are sent to and received from mailboxes.
- A mailbox can be viewed abstractly as, an object into which messages can be placed by processes and from which messages can be removed.

The send and receive primitives are defined as follows :

- **Send (A, message) - Send a message to mailbox A.**
- **Received (A, message) - Receive a message from mailbox A**



Race condition

A **race condition** is an undesirable situation that occurs when a device or system attempts to perform two or more operations at the same time, but because of the nature of the device or system, the operations must be done in the proper sequence to be done correctly.

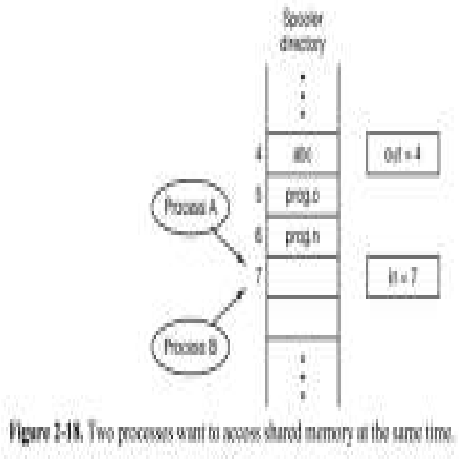
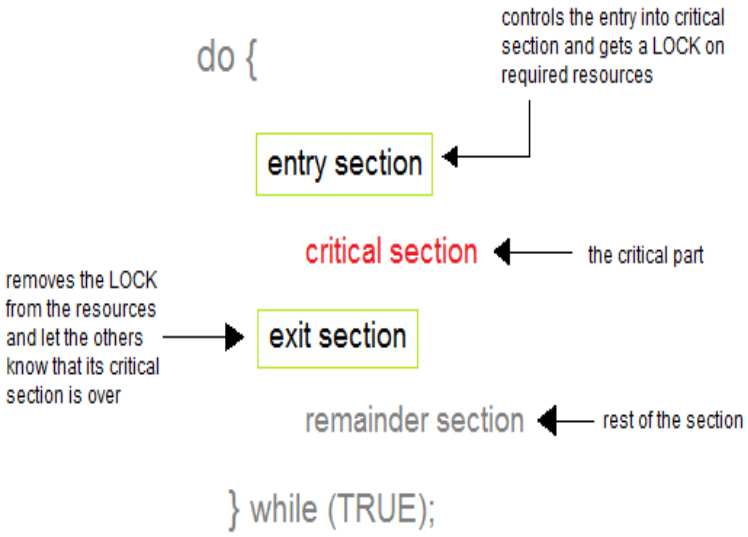


Figure 2-18. Two processes want to access shared memory at the same time.

Critical Section Problem

- A Critical Section is a code segment that accesses shared variables and has to be executed as an atomic action.
- It means that in a group of cooperating processes, at a given point of time, only one process must be executing its critical section.
- If any other process also wants to execute its critical section, it must wait until the first one finishes.



Solution to Critical Section Problem

A solution to the critical section problem must satisfy the following three conditions:

Mutual Exclusion

- Out of a group of cooperating processes, only one process can be in its critical section at a given point of time.

Progress

- If no process is in its critical section, and if one or more threads want to execute their critical section then any one of these threads must be allowed to get into its critical section.

Bounded Waiting

- After a process makes a request for getting into its critical section, there is a limit for how many other processes can get into their critical section, before this process's request is granted. So after the limit is reached, system must grant the process permission to get into its critical section.

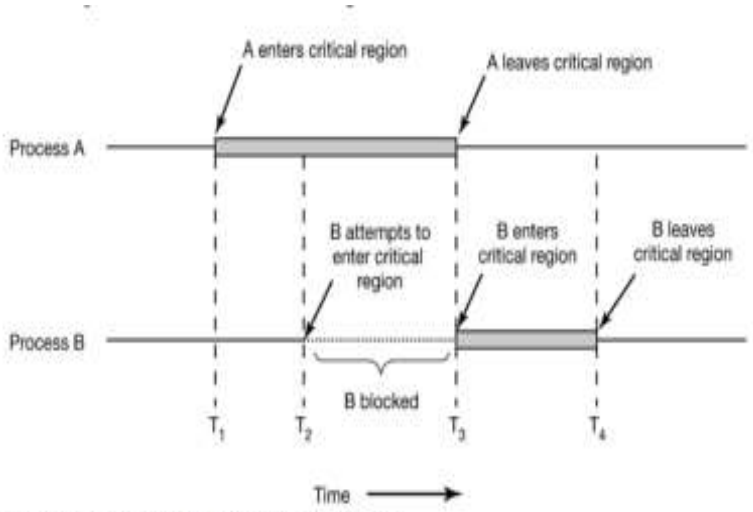


Figure 2-19. Mutual exclusion using critical regions.

Semaphores

- In 1965, Dijkstra proposed a new and very significant technique for managing concurrent processes by using the value of a simple integer variable to synchronize the progress of interacting processes.
- This integer variable is called **semaphore**.
- So it is basically a synchronizing tool and is accessed only through two low standard atomic operations, wait and signal designated by P() and V() respectively.

The classical definition of **wait and signal** are :

- **Wait:** decrement the value of its argument S as soon as it would become non-negative.
- **Signal:** increment the value of its argument, S as an individual operation.

Properties of Semaphores

- Simple
- Works with many processes
- Can have many different critical sections with different semaphores
- Each critical section has unique access semaphores

- Can permit multiple processes into the critical section at once, if desirable.

Types of Semaphores

- Semaphores are mainly of two types:

Binary Semaphore

- It is a special form of semaphore used for implementing mutual exclusion, hence it is often called *Mutex*.
- A binary semaphore is initialized to 1 and only takes the value 0 and 1 during execution of a program.

Counting Semaphores

- These are used to implement bounded concurrency.

Limitations of Semaphores

- Priority Inversion is a big limitation os semaphores.
- Their use is not enforced, but is by convention only.
- With improper use, a process may block indefinitely. Such a situation is called Deadlock. We will be studying deadlocks in details in coming lessons.

Mutex

- A **mutex** is a variable that can be in one of two states:
  - unlocked or
  - locked.
- Consequently, only 1 bit is required to represent it, but in practice an integer often is used, with 0 meaning unlocked and all other values meaning locked. Two procedures are used with mutexes. When a thread (or process) needs access to a critical region, it calls *mutex_lock* .

- If the mutex is current unlocked (meaning that the critical region is available), the call succeeds and the calling thread is free to enter the critical region.

Monitors

- A monitor is a collection of procedures, variables, and data structures that are all grouped together in a special kind of module or package.
- Processes may call the procedures in a monitor whenever they want to, but they cannot directly access the monitor's internal data structures from procedures declared outside the monitor.

```
monitor example
integer i ;
condition c ;

procedure producer ( );


end ;

procedure consumer ( );

end ;
end monitor ;
```

Figure 2-26. A 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.

Resources:

(1) Generally, any item that can be used. Devices such as printers and disk drives are resources, as is memory.

(2) In many operating systems, including Microsoft Windows and the Macintosh operating system, the term *resource* refers specifically to data or routines that are available to programs. These are also called *system resources*.

Resources come in two flavors:

- preemptable and
- nonpreemptable.
  - ✓ A preemptable resource is one that can be taken away from the process with no ill effects. Memory is an example of a preemptable resource.
  - ✓ On the other hand, a nonpreemptable resource is one that cannot be taken away from process (without causing ill effect). For example, *CD* resources are not preemptable at an arbitrary moment.

In general, deadlocks involve nonpreemptable resources.

Deadlock:

Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.

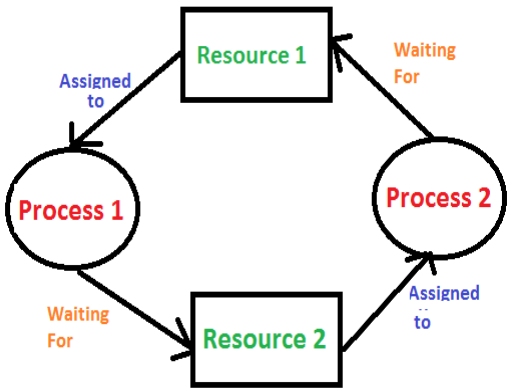
Real time Example:

Consider an example when two trains are coming toward each other on same track and there is only one track, none of the trains can move once they are in front of each other. Similar situation occurs in operating systems

when there are two or more processes hold some resources and wait for resources held by other(s).

Conceptual Example:

For example, in the below diagram, Process 1 is holding Resource 1 and waiting for resource 2 which is acquired by process 2, and process 2 is waiting for resource 1.



Deadlock can arise if following four conditions hold simultaneously (Necessary Conditions)

Mutual Exclusion: One or more than one resource are non-sharable (Only one process can use at a time)

Hold and Wait: A process is holding at least one resource and waiting for resources.

No Preemption: A resource cannot be taken from a process unless the process releases the resource.

Circular Wait*:* A set of processes are waiting for each other in circular form.

Methods for handling deadlock

There are **three ways** to handle deadlock

1) **Deadlock prevention or avoidance:** The idea is to not let the system into deadlock state.

2) **Deadlock detection and recovery:** Let deadlock occur, then do preemption to handle it once occurred.

3) **Ignore the problem all together:** If deadlock is very rare, then let it happen and reboot the system. This is the approach that both Windows and UNIX take.

Deadlock Prevention and Avoidance

Deadlock Characteristics

Mutual Exclusion.

Hold and Wait.

No preemption

Circular wait

Deadlock Prevention

We can prevent Deadlock by eliminating any of the above four condition.

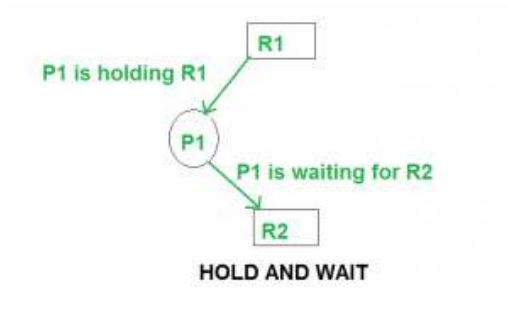
Eliminate Mutual Exclusion

It is not possible to dis-satisfy the mutual exclusion because some resources, such as the tap drive and printer, are inherently non-shareable.

Eliminate Hold and wait

1. Allocate all required resources to the process before start of its execution, this way hold and wait condition is eliminated but it will lead to low device utilization. for example, if a process requires printer at a later time and we have allocated printer before the start of its execution printer will remained blocked till it has completed its execution.

2. Process will make new request for resources after releasing the current set of resources. This solution may lead to starvation.



Eliminate No Preemption

Preempt resources from process when resources required by other high priority process.

Eliminate Circular Wait

Each resource will be assigned with a numerical number. A process can request for the resources only in increasing order of numbering.

For Example, if P1 process is allocated R5 resources, now next time if P1 ask for R4, R3 lesser than R5 such request will not be granted, only request for resources more than R5 will be granted.

Deadlock Avoidance

Deadlock avoidance can be done with Banker's Algorithm.

Banker's Algorithm

Bankers's Algorithm is resource allocation and deadlock avoidance algorithm which test all the request made by processes for resources, it check for safe state, if after granting request system remains in the safe state it allows the request and if their is no safe state it don't allow the request made by the process.

Inputs to Banker's Algorithm

1. Max need of resources by each process.

2. Currently allocated resources by each process.

3. Max free available resources in the system.

Request will only be granted under below condition.

1. If request made by process is less than equal to max need to that process.

2. If request made by process is less than equal to freely availbale resource in the system.

Example for Bankers Algorithm.

Suppose there are four processes in execution with 12 instances of resources are in a system. The maximum need of each process and current allocation are given below.

Process	Maximum. Need	Current allocation
P1	8	3
P2	9	4
P3	5	2
P4	3	1

With reference to current allocation is system safe? If so what is the safe sequence?

Solution

Yes,

p4,p3,p1,p2

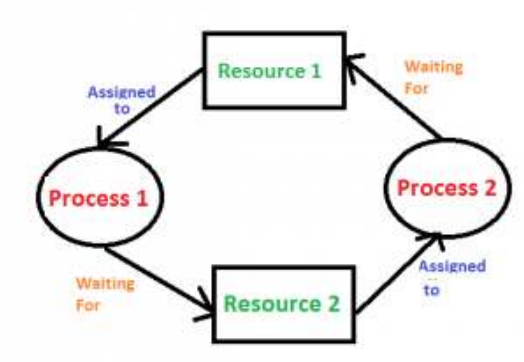
Deadlock Detection And Recovery

3. Deadlock Detection

1. If resources have single instance:

In this case for Deadlock detection we can run an algorithm to check for cycle in the Resource Allocation Graph. Presence of cycle in the graph is

the sufficient condition for deadlock.



In the above diagram, resource 1 and resource 2 have single instances. There is a cycle R1–>P1–>R2–>P2. So Deadlock is Confirmed.

2. If there are multiple instances of resources:

Detection of cycle is necessary but not sufficient condition for deadlock detection, in this case system may or may not be in deadlock varies according to different situations.

4. Deadlock Recovery

Traditional operating system such as Windows doesn't deal with deadlock recovery as it is time and space consuming process. Real time operating systems use Deadlock recovery.

Recovery method

1. Killing the process.
   - Killing all the process involved in deadlock.
   - Killing process one by one. After killing each
   - process check for deadlock again keep repeating
   - Process till system recover from deadlock.

2. Resource Preemption

Resources are preempted from the processes involved in deadlock, preempted resources are allocated to other processes, so that there is a possibility of recovering the system from deadlock. In this case systems go into starvation.

Unit III

CPU Scheduling Introduction:

CPU scheduling is a process which allows one process to use the CPU while the execution of another process is on hold (in waiting state) due to unavailability of any resource like I/O etc, thereby making full use of CPU. **The aim of CPU scheduling is to make the system efficient, fast and fair.**

Objectives of Scheduling Algorithm

- Max CPU utilization [Keep CPU as busy as possible]
- Fair allocation of CPU.
- Max throughput [Number of processes that complete their execution per time unit]
- Min turnaround time [Time taken by a process to finish execution]
- Min waiting time [Time a process waits in ready queue]
- Min response time [Time when a process produces first response]

Scheduling Method:

Schedulers

Schedulers are special system software which handle process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run. Schedulers are of three types −

- Long-Term Scheduler
- Short-Term Scheduler
- Medium-Term Scheduler

Long Term Scheduler

- It is also called a **job scheduler**.
- A long-term scheduler determines which programs are admitted to the system for processing.
- It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling.
- The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound.

Short Term Scheduler

- It is also called as **CPU scheduler**.
- Its main objective is to increase system performance in accordance with the chosen set of criteria. It is the change of ready state to running state of the process.
- CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.
- Short-term schedulers, also known as dispatchers, make the decision of which process to execute next.
- Short-term schedulers are faster than long-term schedulers.

Medium Term Scheduler

- Medium-term scheduling is a part of **swapping**.
- It removes the processes from the memory.
- The medium-term scheduler is in-charge of handling the swapped out-processes.

Scheduling Criteria

There are many different criterias to check when considering the "best" scheduling algorithm:

CPU utilization

To make out the best use of CPU and not to waste any CPU cycle, CPU would be working most of the time (Ideally 100% of the time). Considering a real system, CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded.)

Throughput

It is the total number of processes completed per unit time or rather say total amount of work done in a unit of time. This may range from 10/second to 1/hour depending on the specific processes.

Turnaround time

It is the amount of time taken to execute a particular process, i.e. The interval from time of submission of the process to the time of completion of the process(Wall clock time).

Waiting time

The sum of the periods spent waiting in the ready queue amount of time a process has been waiting in the ready queue to acquire get control on the CPU.

Load average

It is the average number of processes residing in the ready queue waiting for their turn to get into the CPU.

Response time

Amount of time it takes from when a request was submitted until the first response is produced. Remember, it is the time till the first response and not the completion of process execution (final response).

Key Point:

In general CPU utilization and Throughput are maximized and other factors are reduced for proper optimization.

Types of Scheduling:

1. **Preemptive**
2. **Non-preemptive Scheduling**

The Scheduling algorithms can be divided into two categories with respect to how they deal with clock interrupts.

Nonpreemptive Scheduling

A scheduling discipline is nonpreemptive if, once a process has been given the CPU, the CPU cannot be taken away from that process.

Following are some characteristics of nonpreemptive scheduling

1. In nonpreemptive system, short jobs are made to wait by longer jobs but the overall treatment of all processes is fair.
2. In nonpreemptive system, response times are more predictable because incoming high priority jobs can not displace waiting jobs.
3. In nonpreemptive scheduling, a schedular executes jobs in the following two situations.
    a. When a process switches from running state to the waiting state.
    b. When a process terminates.

Preemptive Scheduling

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

The strategy of allowing processes that are logically runable to be temporarily suspended is called Preemptive Scheduling and it is contrast to the "run to completion" method.

Scheduling algorithms

A Process Scheduler schedules different processes to be assigned to the CPU based on particular scheduling algorithms. There are six popular process scheduling algorithms

- First-Come, First-Served (FCFS) Scheduling
- Shortest-Job-Next (SJN) Scheduling
- Priority Scheduling
- Shortest Remaining Time
- Round Robin(RR) Scheduling
- Multiple-Level Queues Scheduling

First Come First Serve (FCFS)

- Jobs are executed on first come, first serve basis.
- It is a non-preemptive scheduling algorithm.
- Easy to understand and implement.
- Its implementation is based on FIFO queue.

- Poor in performance as average wait time is high.

Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16

Wait time of each process is as follows −

Process	**Wait Time : Service Time - Arrival Time**
P0	0 - 0 = 0
P1	5 - 1 = 4
P2	8 - 2 = 6
P3	16 - 3 = 13

Average Wait Time: (0+4+6+13) / 4 = 5.75

Shortest Job Next (SJN)

- This is also known as **shortest job first**, or SJF
- This is a non-preemptive, pre-emptive scheduling algorithm.
- Best approach to minimize waiting time.

- Easy to implement in Batch systems where required CPU time is known in advance.
- Impossible to implement in interactive systems where required CPU time is not known.
- The processer should know in advance how much time process will take.

Process	Arrival Time	Execute Time	Service Time
P0	0	5	3
P1	1	3	0
P2	2	8	16
P3	3	6	8

Wait time of each process is as follows −

Process	Wait Time : Service Time - Arrival Time
P0	3 - 0 = 3
P1	0 - 0 = 0
P2	16 - 2 = 14
P3	8 - 3 = 5

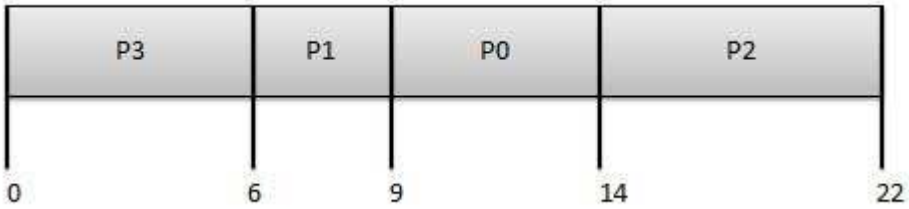
Average Wait Time: (3+0+14+5) / 4 = 5.50

Priority Based Scheduling

- Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems.
- Each process is assigned a priority. Process with highest priority is to be executed first and so on.

- Processes with same priority are executed on first come first served basis.
- Priority can be decided based on memory requirements, time requirements or any other resource requirement.

Process	Arrival Time	Execute Time	Priority	Service Time
P0	0	5	1	9
P1	1	3	2	6
P2	2	8	1	14
P3	3	6	3	0



Wait time of each process is as follows −

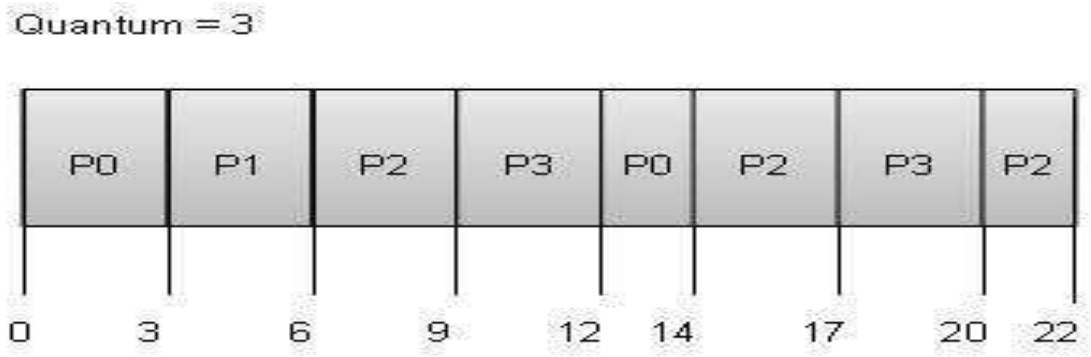
Process	Wait Time : Service Time - Arrival Time
P0	9 - 0 = 9
P1	6 - 1 = 5
P2	14 - 2 = 12
P3	0 - 0 = 0

Average Wait Time: (9+5+12+0) / 4 = 6.5

Round Robin Scheduling

- Round Robin is the preemptive process scheduling algorithm.

- Each process is provided a fix time to execute, it is called a **quantum**.
- Once a process is executed for a given time period, it is preempted and other process executes for a given time period.
- Context switching is used to save states of preempted processes.



Wait time of each process is as follows −

Process	Wait Time : Service Time - Arrival Time
P0	(0 - 0) + (12 - 3) = 9
P1	(3 - 1) = 2
P2	(6 - 2) + (14 - 9) + (20 - 17) = 12
P3	(9 - 3) + (17 - 12) = 11

Average Wait Time: (9+2+12+11) / 4 = 8.5

Multiple-Level Queues Scheduling

Multiple-level queues are not an independent scheduling algorithm. They make use of other existing algorithms to group and schedule jobs with common characteristics.

- Multiple queues are maintained for processes with common characteristics.
- Each queue can have its own scheduling algorithms.
- Priorities are assigned to each queue.

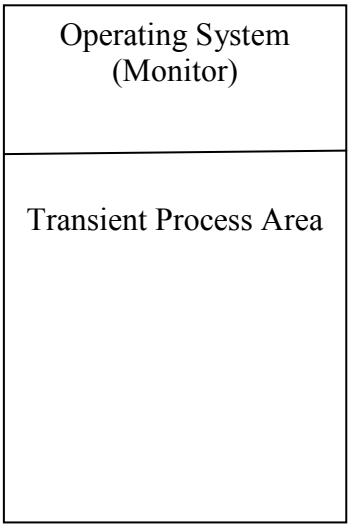
For example, CPU-bound jobs can be scheduled in one queue and all I/O-bound jobs in another queue. The Process Scheduler then alternately selects jobs from each queue and assigns them to the CPU based on the algorithm assigned to the queue.

BASIC MEMORY MANAGEMENT

1. **Single Process Monitor**
2. **Partition Memory Allocation-Static**
3. **Partition Memory Allocation-Dynamic**
4. **Swapping**

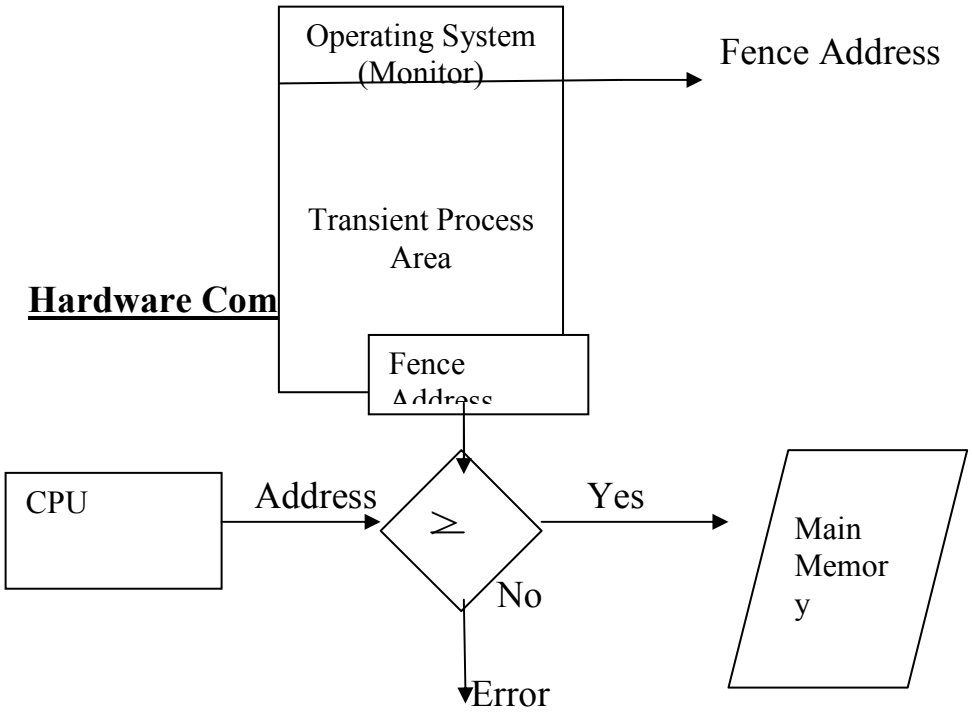
Single Process Monitor:

- The single process monitor is one of the simplest ways of managing memory
- The memory is simply divided into two contiguous areas
- One of them is usually permanently allocated to the resident portion of the operating system
- The remaining memory is allocated to the so called transient processes, which are loaded and executed one at a time, in response to user commands
- When a transient process is completed, the os may load another one for execution
- Both the user processes and non resident portions of the OS may be executed in the transient process area
- This form of memory management is commonly used by single process microcomputer operating systems, such as CP/M and PC-DOS



Memory Protection:

- We must ensure that user program is not permitted to access the operating system.
- Because if user program access the operating system sometimes it leads to some critical situation.
- But OS can access the user program to control the system
- So we need some sort of fencing called as an Fence Address.
- Fence Address or Fence Register is used to draw a boundary between the operating system and the user transient process area.

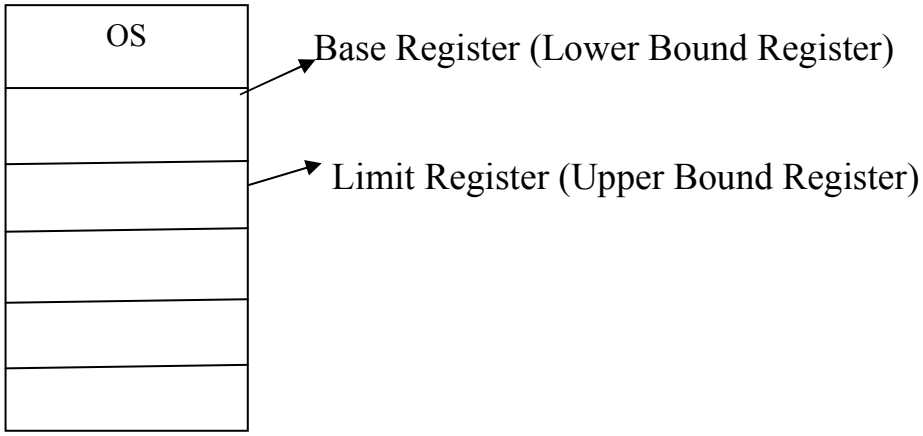


Hardware Com

- CPU generated the address it is greater than or equal to the fence address then only the user program is to allow to access the main memory
- If the CPU generated address is less than the fence address then the job will be terminated

Partition Memory allocation –Static:

Introduction:



- Main memory is divided into number of partitions
- One partitioned is obviously allocated to the OS
- Remaining user area are divided into different partitions
- These partitions are different size or equal size
- Each of the partition can contain one user job
- We can change instruction pointer to switch over from one job to another
- Each partition can contain exactly one job no other jobs are permitted

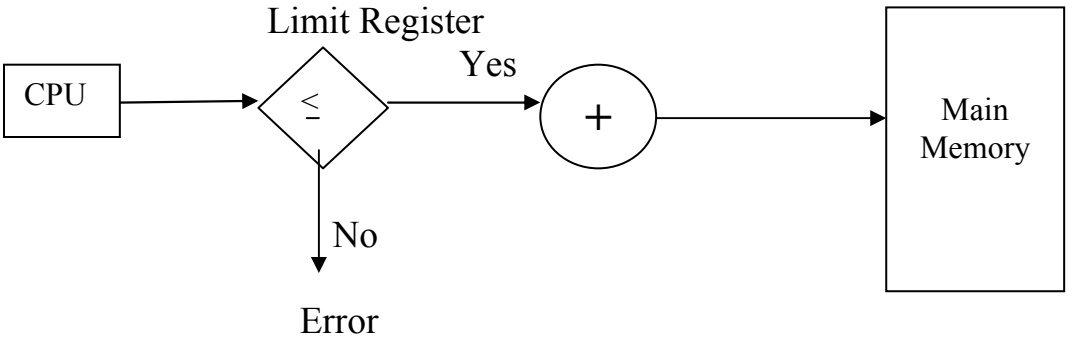
Lower Bound Register (Base Register)

The address generated by the CPU is less than Base Register, is invalid because is trying to access the memory which is not allocated to this job.

Upper Bound Register (Limit Register)

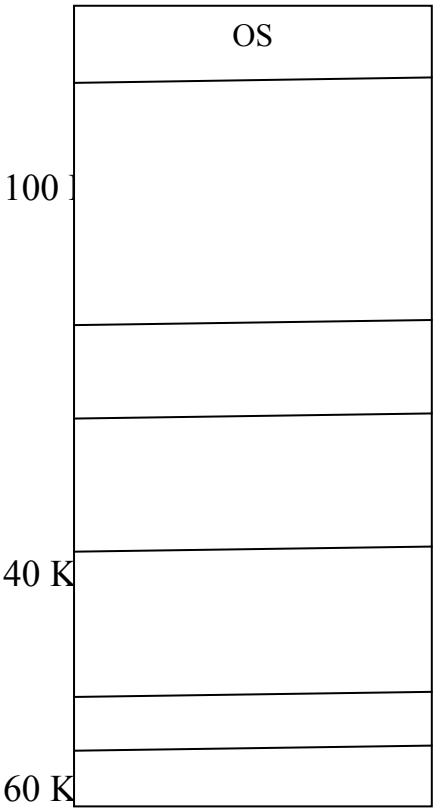
The address generated by the CPU is Greater than Limit Register, is invalid because is trying to access the memory which is not allocated to this job.

Diagrammatic Representation :



- Whenever CPU generated a Address compare with Limit Register is less than or equal to Limit register
- If not less than or equal to limit register its trying to access another location not allocated to them
- If address generated by the CPU is less than the limit register then add base address with limit register to get physical address in the main memory.

Example :



100 KB

20 KB

50 KB

- In this example memory is divided in to number different types of partitions
- If I have a job is size requirement is 50 kb that is **Job1=50 kb**
- Then the job will loaded in to the main memory if the size of the partition is greater than or equal to the job size.
- We have so many option, because we have lot of different size of partition. We must choose any one of the partition in the main memory.
- For choosing the partition we use two different kinds of algorithm that is
    1. First Fit
    2. Best Fit

First Fit:

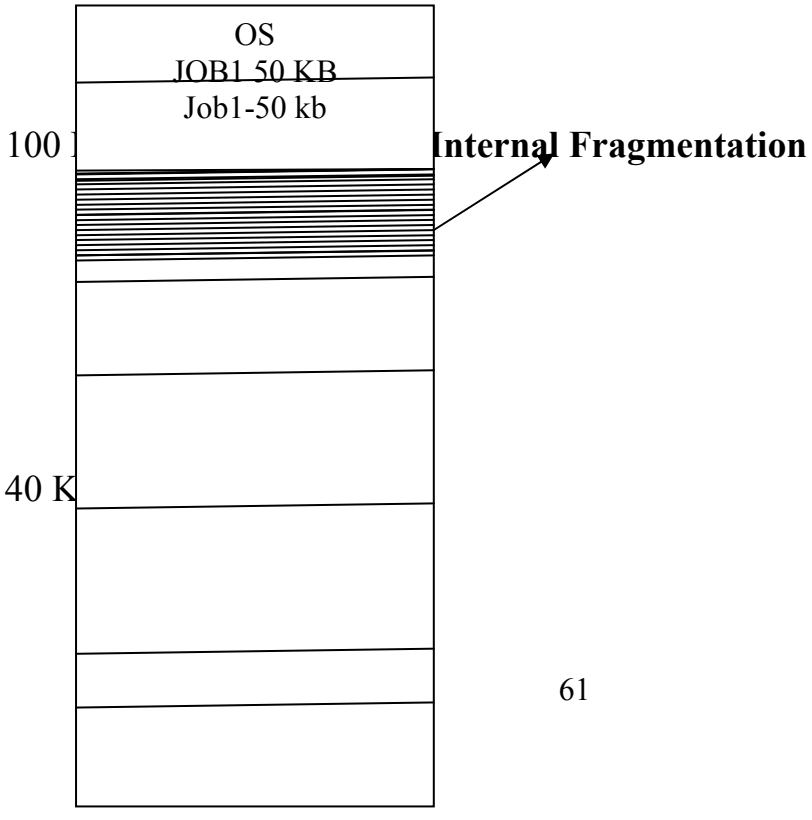
It simply checks the partition sequentially the moment it gets the partition its size is greater than or equal to the required size of the job. Immediately that partitioned is allocated.

Best Fit:

It will check all the free partition and it will allocate that free partition to this job which will more than or nearly to the job size.

The following example uses the First Fit algorithm

Example :



60 KB

100 KB

20 KB
50 KB

Note: *Internal Fragmentation means Memory block assigned to process is bigger. Some portion of memory is left unused as it cannot be used by another process.*

- The job1(50 kb) is allocated to the partition that contain 100 kb our job size is 50 kb remaining 50 kb are wasted, it cannot be used by another process this problem is known as internal fragmentation.

Problems in static memory allocation:

- Internal Fragmentation

Swapping

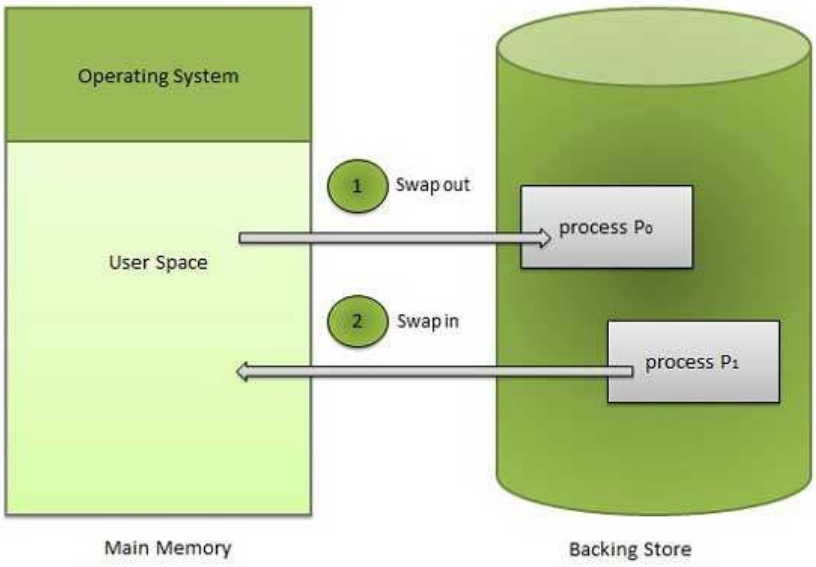
- Swapping is a mechanism in which a process can be swapped temporarily out of main memory to a backing store , and then brought back into memory for continued execution.
- Backing store is a usually a hard disk drive or any other secondary storage which fast in access and large enough to accommodate copies of all memory images for all users.
- It must be capable of providing direct access to these memory images.
- Major time consuming part of swapping is transfer time. Total transfer time is directly proportional to the amount of memory swapped.
- Let us assume that the user process is of size 100KB and the backing store is a standard hard disk with transfer rate of 1 MB per second.

- The actual transfer of the 100K process to or from memory will take

100KB / 1000KB per second

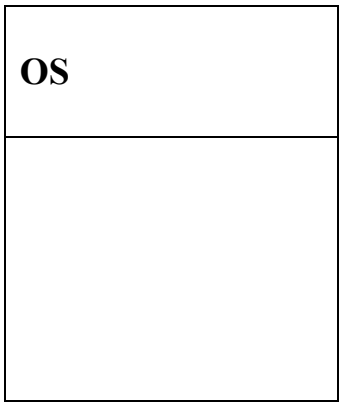
= 1/10 second

= 100 milliseconds



Partition memory allocation – dynamic:

- This technique we don't have pre-defined partition. It is also called multi programming with variable number of task.



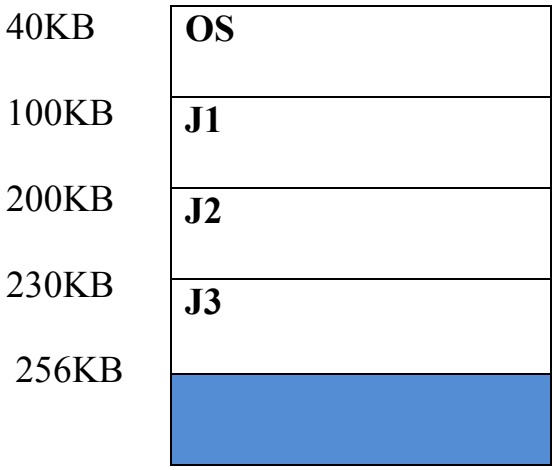
- Memory is initially divided into two partitions.
- One of the partitions is allocated to OS.
- Initially there is no job in the main memory.
- Let as assume our memory size is 256 KB
- Out of which the 40KB are allocated to operation system.

- We have following members of jobs,memory requirement and burst time.

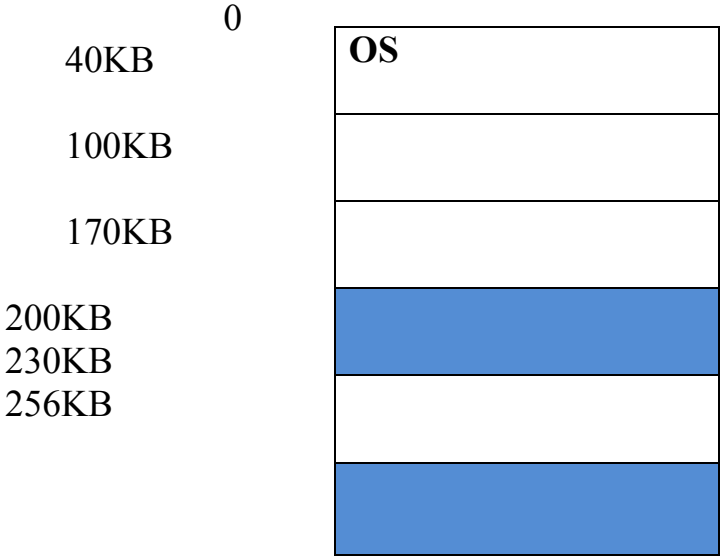
JOBS	MEMORY REQUIREMENT	CPU BURST
J1	60KB	10
J2	100KB	5
J3	30KB	20
J4	70KB	8
J5	50KB	15
J6	60KB	9

- Let's as see how dynamic partition memory allocation allocate the space for above mentioned jobs.
- Now come to the job queue. We have 216KB of memory remaining in the user area.
- Job 1 it requires 60KB of memory and allocates 60KB to the job 1.
- Job 2 it requires 100KB and it is allocated and it is allocated.
- After this job 3 it requires 30KB of memory and it's given to job 3.
- We are totally allocated 230KB to job 1, job 2,job3 and we have remaining 26KBof memory
- But we cannot put job 4 in 26KB of memory because it's memory requirement is 70KB
- So that 26KB of memory is wasted it is called as an external fragmentation.
- That is memory size is less than job size it is called as an external fragmentation

.



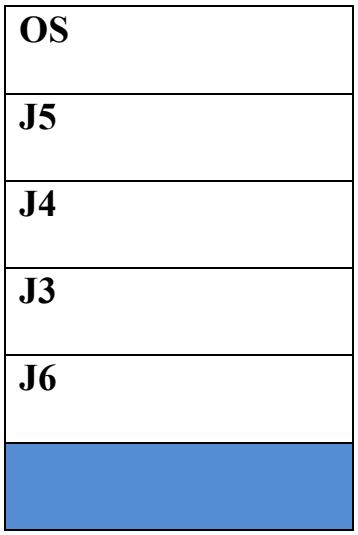
- After 5 times units job J2 will complete its execution so that we got free memory of 100KB
- This 100KB it's allocated to job J4 which require 70KB.
- But we have 100KB of memory, remaining 30KB are wasted, totally the wasted memory is 56KB.



- After 10 time units job 1 will complete its execution so that we got free memory of 60KB.
- In this 60KB will allocate job 5 which requires 50KB.

- Remaining 10 KB are wasted totally we got 66KB

- We want to allocate memory for each job it's require 60KB of memory.
- But we have 66KB of memory we cannot use these memory space job 6 because it's continuous.
- To solve this problem we must use a technique called compaction.
-



- The compaction technique gives non-contiguous memory.

- After his we got 66KB of memory which is continuous.
- We can allocate J6 that requires 60KB into 66KB of free memory.
- At lost only 6 KB of memory are wasted.

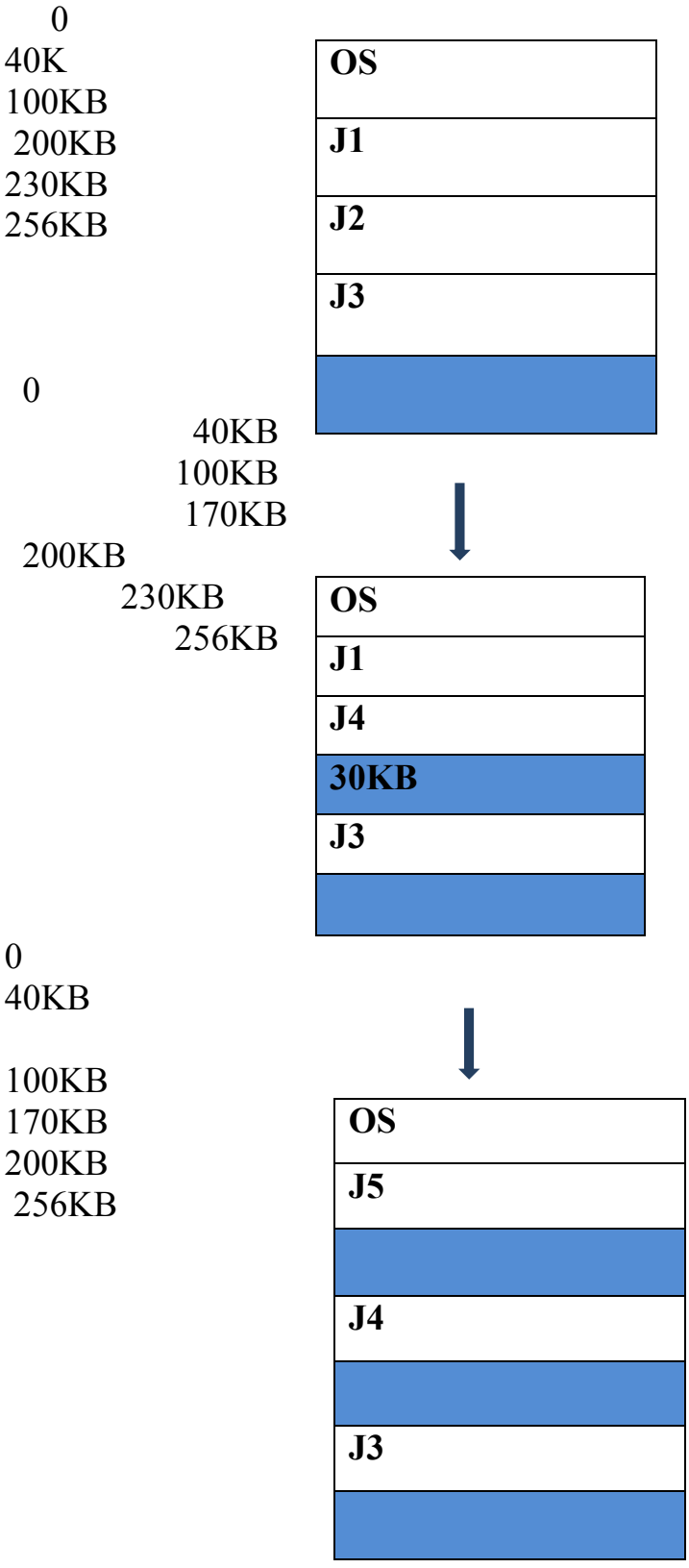
Conclusion:

- We cannot remove fragmentation but we can reduce it.

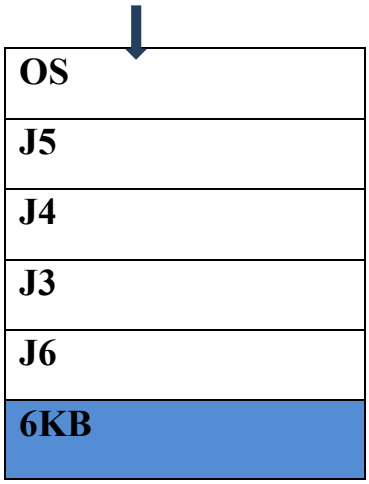
Example:

JOBS	MEMORY REQUIREMENT	CPU BURST
J1	60K	10
J2	100K	5
J3	30K	20
J4	70K	8
J5	50K	15
J6	60K	9

- In dynamic partition memory allocation we cannot have pre-decided partition memory allocation.
- For example, we have 6 kinds of jobs
- Jobs 1 to job 6


0
40K
100KB
200KB
230KB
256KB
OS
J1
J2
J3
0
40KB
100KB
170KB
200KB
230KB
256KB
OS
J1
J4
30KB
J3
0
40KB
100KB
170KB
200KB
256KB
OS
J5
J4
J3

COMPACTION



Virtual memory

- A computer can address more memory than the amount physically installed on the system.
- This extra memory is actually called **virtual memory** and it is a section of a hard disk that's set up to emulate the computer's RAM.
- The main visible advantage of this scheme is that programs can be larger than physical memory.

Virtual memory serves two purposes.

- First, it allows us to extend the use of physical memory by using disk.
- Second, it allows us to have memory protection, because each virtual address is translated to a physical address.

In real scenarios, most processes never need all their pages at once, for following reasons:

- ✓ Error handling code is not needed unless that specific error occurs, some of which are quite rare.

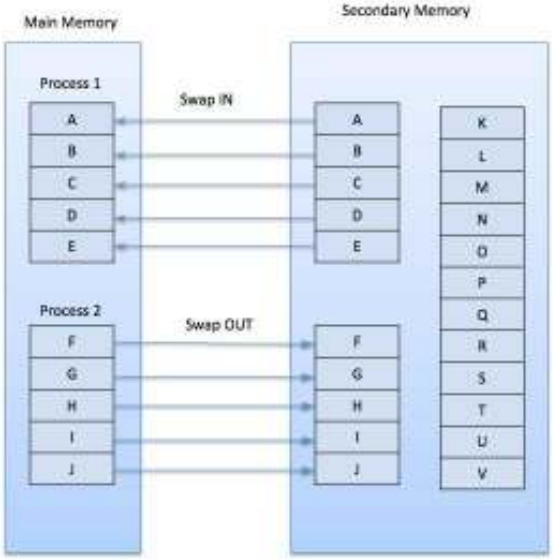
- ✓ Arrays are often over-sized for worst-case scenarios, and only a small fraction of the arrays are actually used in practice.
- ✓ Certain features of certain programs are rarely used.

Benefits of having Virtual Memory:

1. Large programs can be written, as virtual space available is huge compared to physical memory.
2. Less I/O required, leads to faster and easy swapping of processes.
3. More physical memory available, as programs are stored on virtual memory, so they occupy very less space on actual physical memory.

Demand Paging

- A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance.
- When a context switch occurs, the operating system does not copy any of the old program's pages out to the disk or any of the new program's pages into the main memory
- Instead, it just begins executing the new program after loading the first page and fetches that program's pages as they are referenced.



While executing a program, if the program references a page which is not available in the main memory because it was swapped out a little ago, the processor treats this invalid memory reference as a **page fault** and transfers control from the program to the operating system to demand the page back into the memory.

Advantages

Following are the advantages of Demand Paging −

- Large virtual memory.
- More efficient use of memory.
- There is no limit on degree of multiprogramming.

Disadvantages

- Number of tables and the amount of processor overhead for handling page interrupts are greater than in the case of the simple paged management techniques.

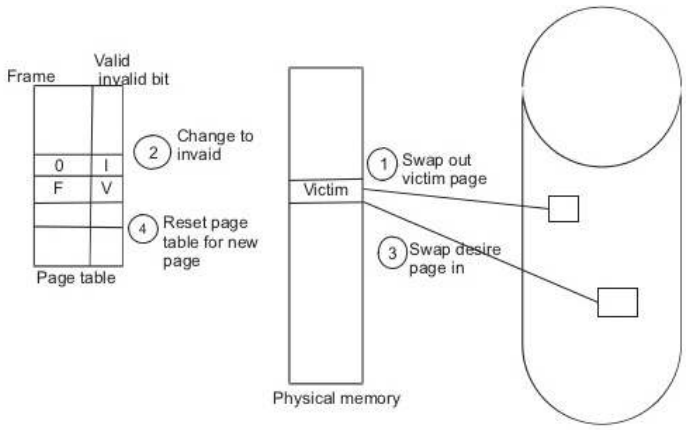
PAGE REPLACEMENT ALGORITHMS

What is paging

- The OS divides virtual memory and the main memory into units, called pages.
- Each used page can be either in secondary memory or in a page frame in main memory.
- A frame does not have to comprise a single physically contiguous region in secondary storage.

Page replacement:

- When memory located in secondary memory is needed, it can be retrieved back to main memory.
- Process of storing data from main memory to secondary memory ->**swapping out**
- Retrieving data back to main memory ->**swapping in.**

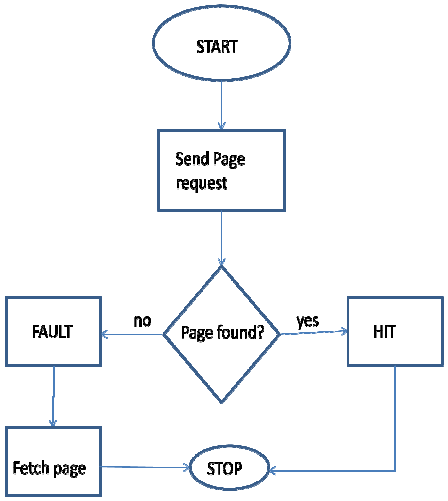


Page replacement algorithms:

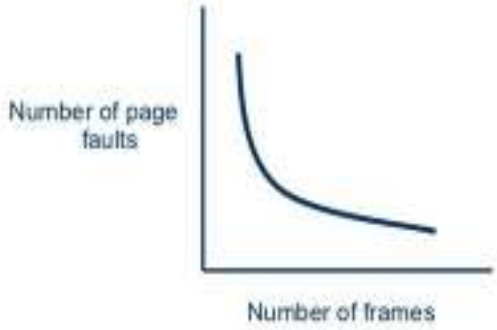
- Deals with which pages need to be swapped out and which are the ones that need to be swapped in
- The efficiency lies in the least time that is wasted for a page to be paged in

Why we need a page replacement algorithm?

- The main goal of page replacement algorithms is to provide lowest page fault rate.



No. of Page Faults vs. No. of Frames:

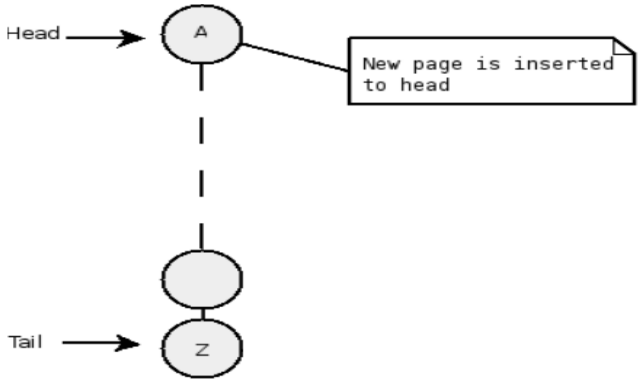


Algorithms

- First In First Out
- Optimal Replacement
- Least Recently Used

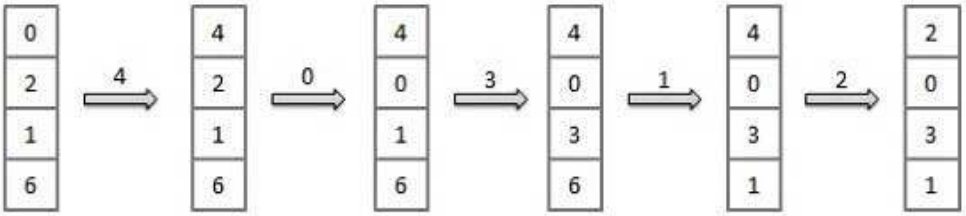
First-In First-Out (FIFO)

- Oldest page in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages from the tail and add new pages at the head.



Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x x x x



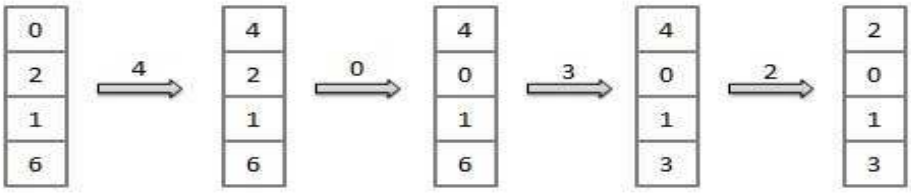
Fault Rate = 9 / 12 = 0.75

Least Recently Used (LRU) algorithm

- Page which has not been used for the longest time in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages by looking back into time.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x x x x



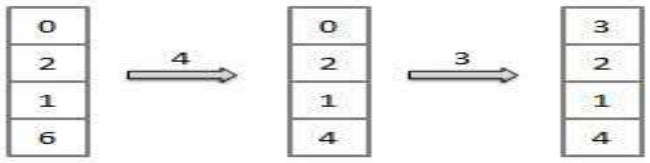
Fault Rate = 8 / 12 = 0.67

Optimal Replacement (OPT)

- An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms. An optimal page-replacement algorithm exists, and has been called OPT or MIN.
- Replace the page that will not be used for the longest period of time. Use the time when a page is to be used.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x



Fault Rate = 6 / 12 = 0.50

Conclusion:

FIFO → Might throw out important pages.

OPTIMAL → Not implementable.

LRU → Excellent but difficult to implement.

PRINCIPLES OF I/O HARDWARE

- ❖ I/O Devices
- ❖ Device Controllers
- ❖ Memory-Mapped I/O
- ❖ Direct Memory Access (DMA)
- ❖ Interrupts Revisited

I/O Devices:

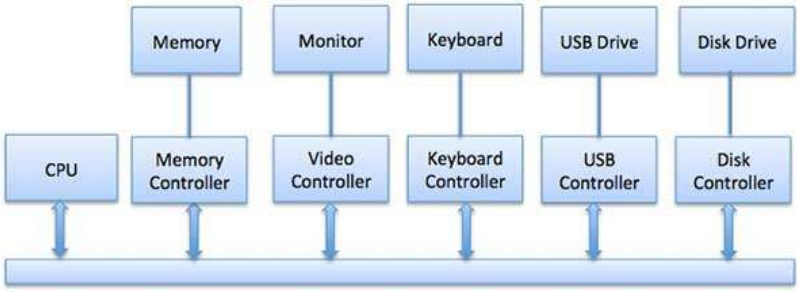
One of the important jobs of an Operating System is to manage various I/O devices including mouse, keyboards, touch pad, disk drives, display adapters, USB devices, Bit-mapped screen, LED, Analog-to-digital converter, On/off switch, network connections, audio I/O, printers etc.

I/O devices can be divided into two categories −

- **Block devices** − A block device is one with which the driver communicates by sending entire blocks of data. For example, Hard disks, USB cameras, Disk-On-Key etc.
- **Character devices** − A character device is one with which the driver communicates by sending and receiving single characters (bytes, octets). For example, serial ports, parallel ports, sounds cards etc

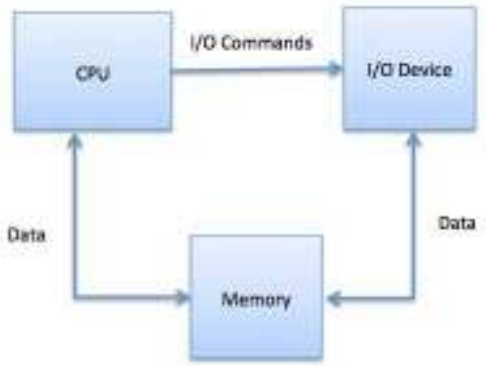
Device Controllers

- Device drivers are software modules that can be plugged into an OS to handle a particular device.
- Operating System takes help from device drivers to handle all I/O devices.
- The Device Controller works like an interface between a device and a device driver.
- I/O units (Keyboard, mouse, printer, etc.) typically consist of a mechanical component and an electronic component where electronic component is called the device controller.
- There is always a device controller and a device driver for each device to communicate with the Operating Systems.



Memory-mapped I/O

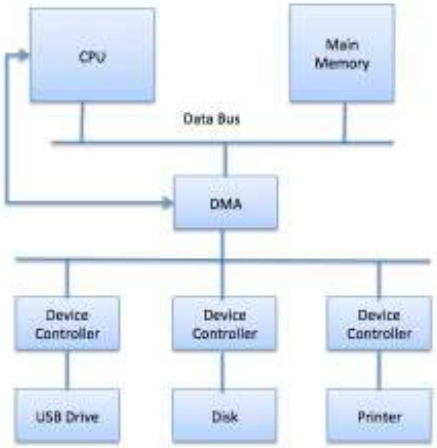
- When using memory-mapped I/O, the same address space is shared by memory and I/O devices.
- The device is connected directly to certain main memory locations so that I/O device can transfer block of data to/from memory without going through CPU.



- Memory mapped IO is used for most high-speed I/O devices like disks, communication interfaces.

Direct Memory Access (DMA)

- Direct Memory Access (DMA) means CPU grants I/O module authority to read from or write to memory without involvement.
- DMA module itself controls exchange of data between main memory and the I/O device.
- CPU is only involved at the beginning and end of the transfer and interrupted only after entire block has been transferred.
- Direct Memory Access needs a special hardware called DMA controller (DMAC) that manages the data transfers and arbitrates access to the system bus.



Interrupts I/O

- An interrupt is a signal to the microprocessor from a device that requires attention.
- A device controller puts an interrupt signal on the bus when it needs CPU's attention when CPU receives an interrupt; It saves its current state and invokes the appropriate interrupt handler using the interrupt vector (addresses of OS routines to handle various events).
- When the interrupting device has been dealt with, the CPU continues with its original task as if it had never been interrupted.

Principles of I/O Software

1. Goals of the I/O Software
2. Programmed I/O
3. Interrupt-Driven I/O
4. Using DMA

Goals of the I/O Software

- Device independence
- Uniform naming
- Error handling

Programmed I/O

- As mentioned just above with programmed I/O the processor moves the data between memory and the device.
- How does the process know when the device is ready to accept or supply new data?
- In the simplest implementation, the processor loops continually asking the device. This is called **polling** or **busy waiting**.
- If the device is slow, polling is clearly wasteful, which leads us to.

Interrupt-Driven I/O

- The device interrupts the processor when it is ready.
- An interrupt service routine then initiates transfer of the next datum

- Normally better than polling, but not always. Interrupts are expensive on modern machines.
- To minimize interrupts, better controllers often employ ...

I/O Using DMA

- We discussed DMA above.
- An additional advantage of dma, not mentioned above, is that the processor is interrupted only at the end of a command not after each datum is transferred.
- Many devices receive a character at a time, but with a dma controller, an interrupt occurs only after a buffer has been transferred.

Unit V

6 FILE SYSTEMS

6.1 FILES

6.1.1 File Naming

6.1.2 File Structure

6.1.3 File Types

6.1.4 File Access

6.1.5 File Attributes

6.1.6 File Operations

6.2 DIRECTORIES

6.2.1 Single-Level Directory Systems

6.2.2 Two-level Directory Systems

6.2.3 Hierarchical Directory Systems

6.2.4 Path Names

6.2.5 Directory Operations

6.3 FILE SYSTEM IMPLEMENTATION

File

- A file is a named collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tapes and optical disks.
- ***In general, a file is a sequence of bits, bytes, lines or records whose meaning is defined by the files creator and user.***

File Naming

- The exact rules for file naming vary somewhat from system to system, but all current operating systems allow strings of one to eight letters as legal file names.
- Thus *andrea* , *bruce* , and *cathy* are possible file names. Frequently digits and special characters are also permitted, so names like *2* , *urgent!* , and *Fig.2-14* are often valid as well.

Extension	Meaning
file.bak	Backup file
file.c	C source program
file.gif	Compuserve Graphical Interchange Format image
file.hlp	Help file
file.html	World Wide Web HyperText Markup Language document
file.jpg	Still picture encoded with the JPEG standard
file.mp3	Music encoded in MPEG layer 3 audio format

file.mpg
Movie encoded with the MPEG standard
file.o
Object file (compiler output, not yet linked)
file.pdf
Portable Document Format file
file.ps
PostScript file
file.tex
Input for the TEX formatting program
file.txt
General text file
file.zip
Compressed archive

Figure 6-1. Some typical file extensions.

File Structure

- Files can be structured in any of several ways.
- Three common possibilities are depicted in Fig. 6-2.

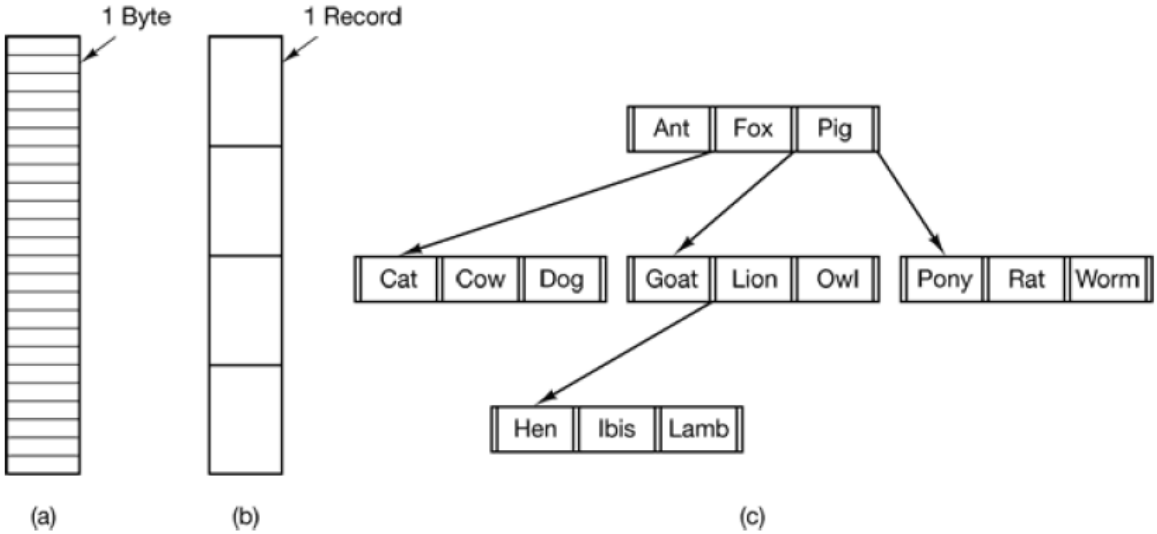


Figure 6-2. Three kinds of files. (a) Byte sequence. (b) Record sequence. (c) Tree.

- The file in Fig. 6-2(a) is an unstructured sequence of bytes. In effect, the operating system does not know or care what is in the file. All it sees are bytes.
- The first step up in structure is shown in Fig. 6-2(b). In this model, a file is a sequence of fixed-length records, each with some internal structure.
- The third kind of file structure is shown in Fig. 6-2(c). In this organization, a file consists of a tree of records, not necessarily all the same length, each containing a **key** field in a fixed position in the record. The tree is sorted on the key field, to allow rapid searching for a particular key.

File Type

- File type refers to the ability of the operating system to distinguish different types of file such as text files source files and binary files etc.
- Many operating systems support many types of files. Operating system like MS-DOS and UNIX have the following types of files −

Ordinary files

- These are the files that contain user information.
- These may have text, databases or executable program.
- The user can apply various operations on such files like add, modify, delete or even remove the entire file.

Directory files

- These files contain list of file names and other information related to these files.

Special files

- These files are also known as device files.
- These files represent physical device like disks, terminals, printers, networks, tape drive etc.

These files are of two types −

- **Character special files** − data is handled character by character as in case of terminals or printers.
- **Block special files** − data is handled in blocks as in the case of disks and tapes.

File Access Mechanisms

File access mechanism refers to the manner in which the records of a file may be accessed. There are several ways to access files −

- Sequential access
- Direct/Random access
- Indexed sequential access

Sequential access

- ❖ A sequential access is that in which the records are accessed in some sequence,

- i.e., the information in the file is processed in order, one record after the other.
- This access method is the most primitive one. Example: Compilers usually access files in this fashion.

Direct/Random access

- Random access file organization provides, accessing the records directly.
- Each record has its own address on the file with by the help of which it can be directly accessed for reading or writing.
- The records need not be in any sequence within the file and they need not be in adjacent locations on the storage medium.

Indexed sequential access

- This mechanism is built up on base of sequential access.
- An index is created for each file which contains pointers to various blocks.
- Index is searched sequentially and its pointer is used to access the file directly.

File Attributes

- Every file has a name and its data.
- In addition, all operating systems associate other information with each file, for example, the date and time the file was created and the file's size.
- We will call these extra items the file's **attributes**. The list of attributes varies considerably from system to system.

- The table of Fig. 6-4 shows some of the possibilities.

Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Offset of the key within each record	
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file has last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

Figure 6-4. Some possible file attributes.

File Operations

- Files exist to store information and allow it to be retrieved later.
- Different systems provide different operations to allow storage and retrieval.
- Below is a discussion of the most common system calls relating to files.

Create. The file is created with no data. The purpose of the call is to announce that the file is coming and to set some of the attributes.

Delete. When the file is no longer needed, it has to be deleted to free up disk space. There is always a system call for this purpose.

Open. Before using a file, a process must open it. The purpose of the open call is to allow the system to fetch the attributes and list of disk addresses into main memory for rapid access on later calls.

Close When all the accesses are finished, the attributes and disk addresses are no longer needed, so the file should be closed to free up internal table space.

Read Data are read from file. Usually, the bytes come from the current position. The caller must specify how much data are needed and must also provide a buffer to put them in.

Write Data are written to the file, again, usually at the current position. If the current position is the end of the file, the file's size increases. If the current position is in the middle of the file, existing data are overwritten and lost forever.

Append . This call is a restricted form of write. It can only add data to the end of the file. Systems that provide a minimal set of system calls do not generally have append , but many systems provide multiple ways of doing the same thing, and these systems sometimes have append .

Seek For random access files, a method is needed to specify from where to take the data. One common approach is a system call, seek, that repositions the file pointer to a specific place in the file. After this call has completed, data can be read from, or written to, that position,

Rename. It frequently happens that a user needs to change the name of an existing file. This system call makes that possible. It is not always strictly

necessary, because the file can usually be copied to a new file with the new name, and the old file then deleted.

DIRECTORIES

To keep track of files, file systems normally have **directories** or **folders** , which, in many systems, are themselves files.

Single-Level Directory Systems

- The simplest form of directory system is having one directory containing all the files. Sometimes it is called the **root directory**, but since it is the only one, the name does not matter much.

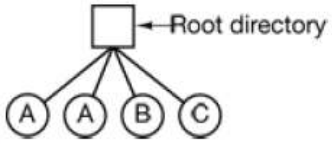


Figure 6-7. A single-level directory system containing four files, owned by three different people, *A* , *B* , and *C* .

Two-level Directory Systems

- To avoid conflicts caused by different users choosing the same file name for their own files, the next step up is giving each user a private directory.
- In that way, names chosen by one user do not interfere with names chosen by a different user and there is no problem caused by the same name occurring in two or more directories.
- This design leads to the system of Fig. 6-8.

open("nancy/x")

might be the call to open a file *x* in the directory of another user, Nancy.

Hierarchical Directory Systems

- The two-level hierarchy eliminates name conflicts among users but is not satisfactory for users with a large number of files.
- Even on a single-user personal computer, it is inconvenient.
- It is quite common for users to want to group their files together in logical ways.

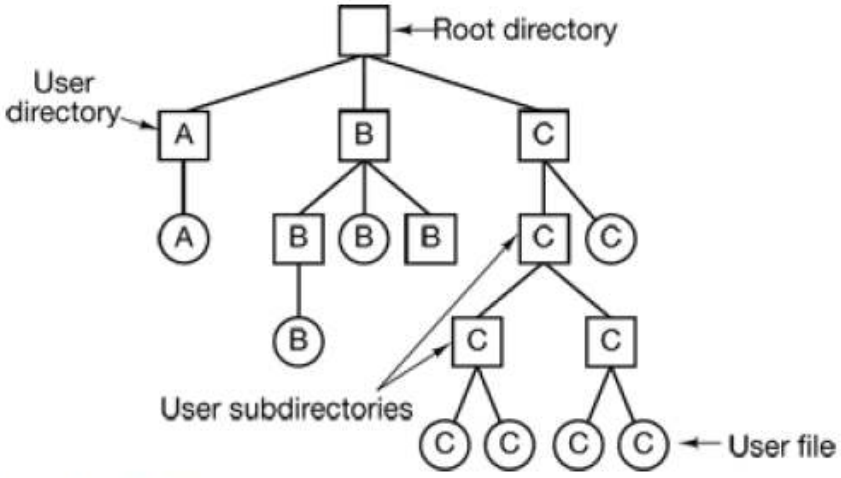


Figure 6-9. A hierarchical directory system.

Path Names

- When the file system is organized as a directory tree, some way is needed for specifying file names.
- Two different methods are commonly used. In the first method, each file is given an **absolute path name** consisting of the path from the root directory to the file.
- As an example, the path */usr/ast/mailbox* means that the root directory contains a subdirectory *usr* , which in turn contains a subdirectory *ast* , which contains the file *mailbox* . Absolute path names always start at the root directory and are unique.
- The relative form is often more convenient, but it does the same thing as the absolute form.

Directory Operations

- The allowed system calls for managing directories exhibit more variation from system to system than system calls for files.
- To give an impression of what they are and how they work, we will give a sample (taken from UNIX).

Create. A directory is created. It is empty except for dot and dotdot, which are put there automatically by the system (or in a few cases, by the *mkdir* program).

Delete. A directory is deleted. Only an empty directory can be deleted. A directory containing only dot and dotdot is considered empty as these cannot usually be deleted.

Opendir: Directories can be read. For example, to list all the files in a directory, a listing program opens the directory to read out the names of all the files it contains. Before a directory can be read, it must be opened, analogous to opening and reading a file.

Closedir . When a directory has been read, it should be closed to free up internal table space.

Rename . In many respects, directories are just like files and can be renamed the same way files can be.

FILE SYSTEM IMPLEMENTATION

Contiguous Allocation

- Each file occupies a contiguous address space on disk.
- Assigned disk address is in linear order.
- Easy to implement.

- External fragmentation is a major issue with this type of allocation technique.

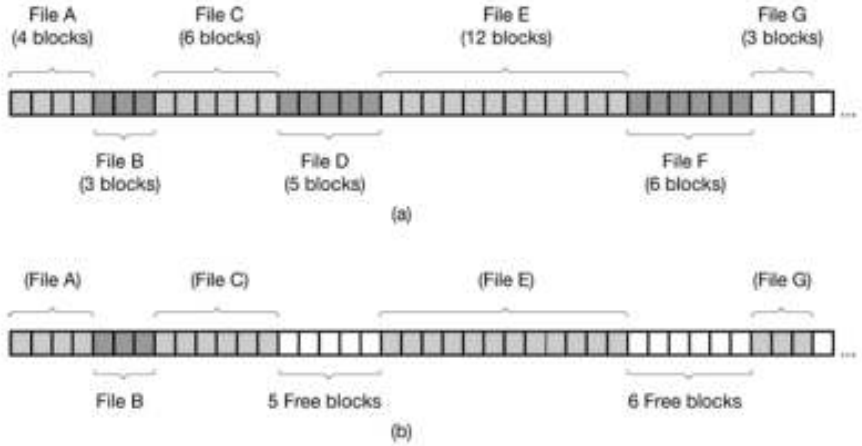


Figure 6-12. (a) Contiguous allocation of disk space for seven files. (b) The state of the disk after files *D* and *F* have been removed.

Linked Allocation

- Each file carries a list of links to disk blocks.
- Directory contains link / pointer to first block of a file.
- No external fragmentation
- Effectively used in sequential access file.
- Inefficient in case of direct access file.

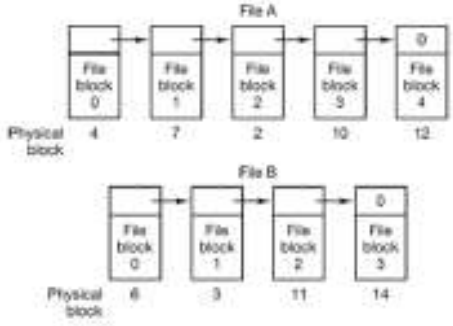


Figure 6-13. Storing a file as a linked list of disk blocks.

I-nodes

Our last method for keeping track of which blocks belong to which file is to associate

with each file a data structure called an **i-node** (**index-node**), which lists the attributes

and disk addresses of the files blocks. A simple example is depicted in Fig. 6-15.

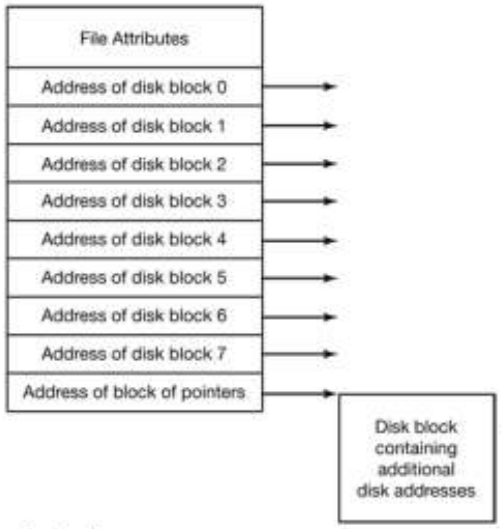
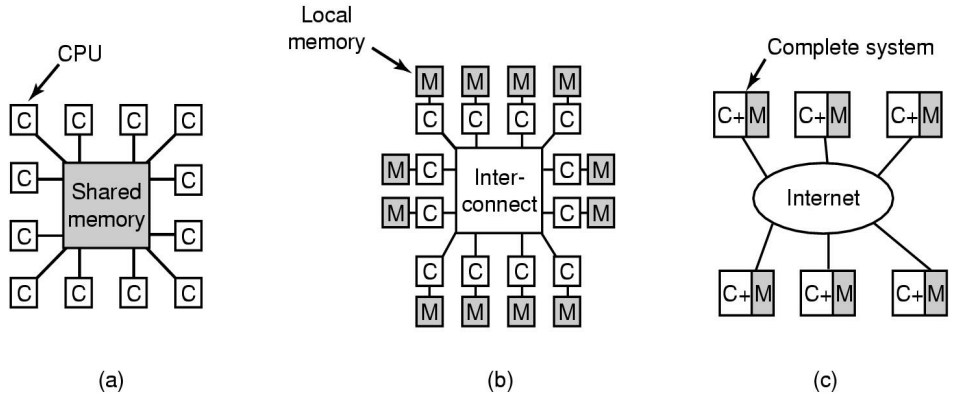


Figure 6-15. An example i-node.

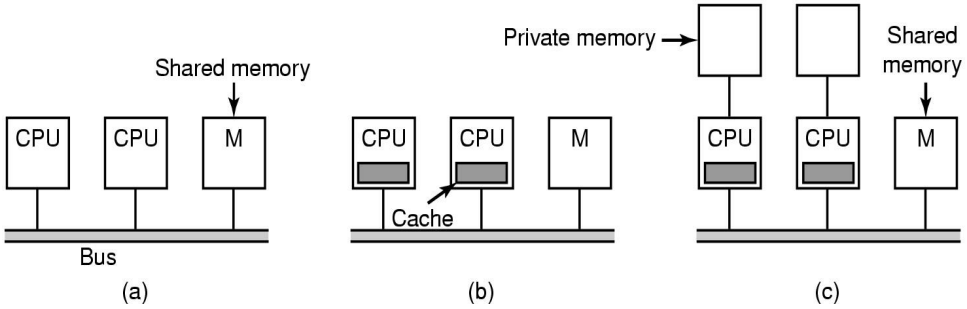
Multiple Processor Systems
8.1 Multiprocessors
8.2 Multicomputers
8.3 Distributed systems

Definition:

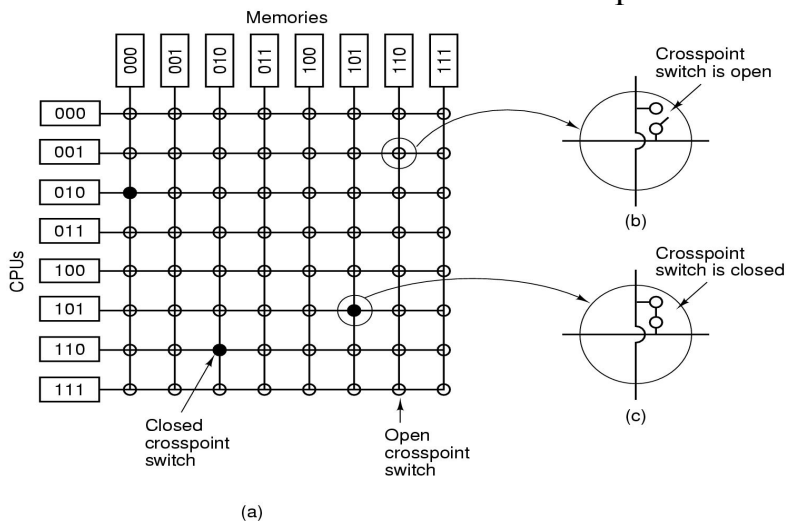
A computer system in which two or more CPUs share full access to a common RAM



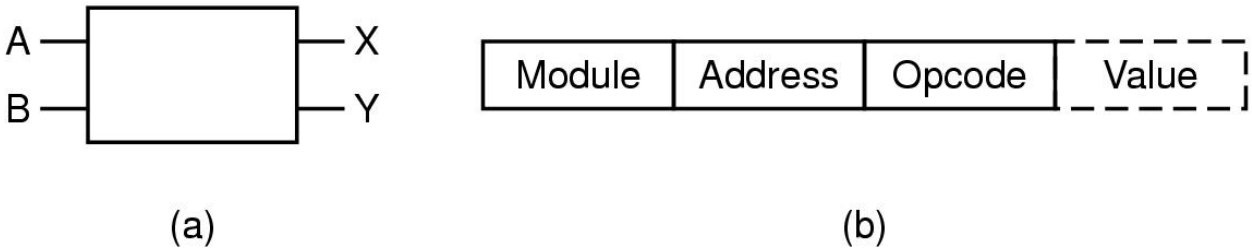
Multiprocessor Hardware



Bus-based multiprocessors



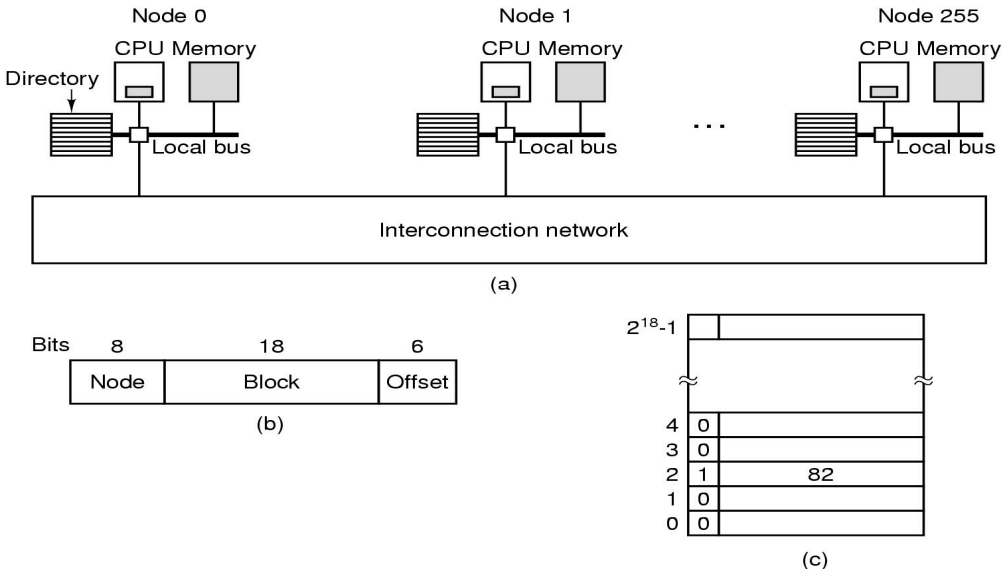
- MA Multiprocessor using a crossbar switch
- UMA multiprocessors using multistage switching networks can be built from 2x2 switches



(a) 2x2 switch (b) Message format

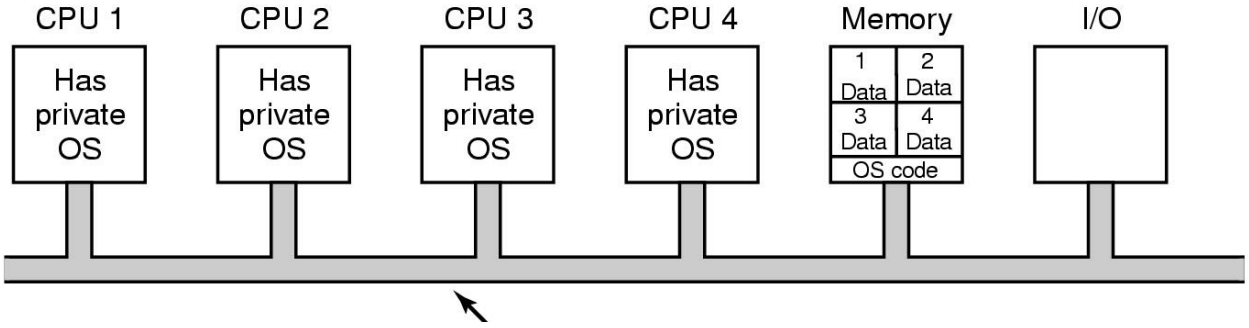
NUMA Multiprocessor Characteristics

1. Single address space visible to all CPUs
2. Access to remote memory via commands
   - LOAD
   - STORE
3. Access to remote memory slower than to local

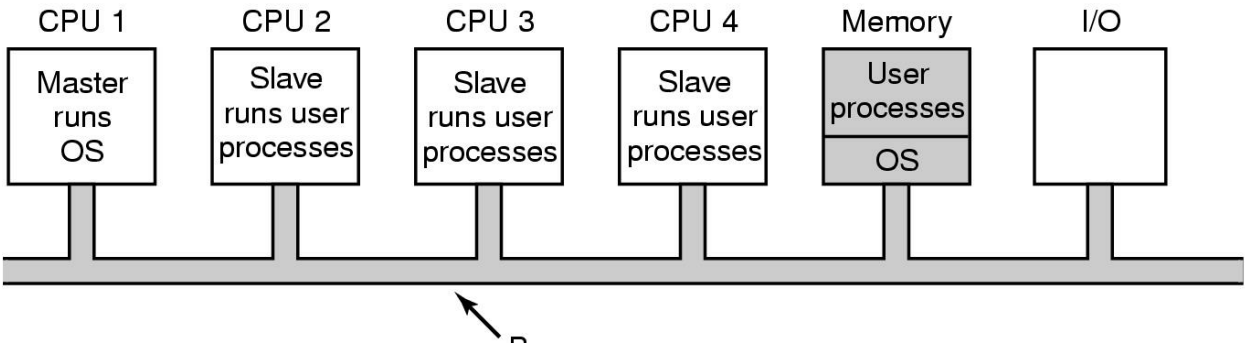


(a) 256-node directory based multiprocessor
(b) Fields of 32-bit memory address
(c) Directory at node 36

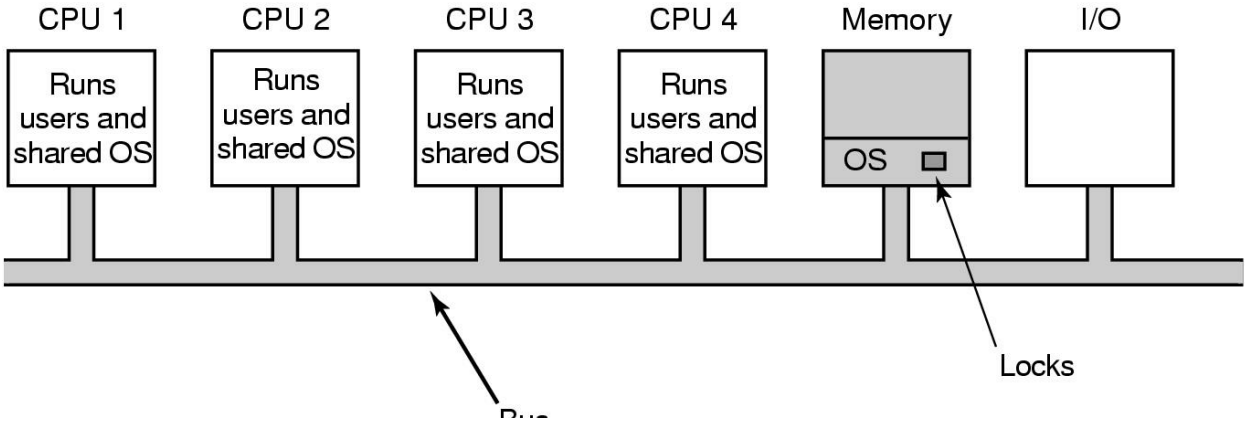
Multiprocessor OS Types



Each CPU has its own operating system

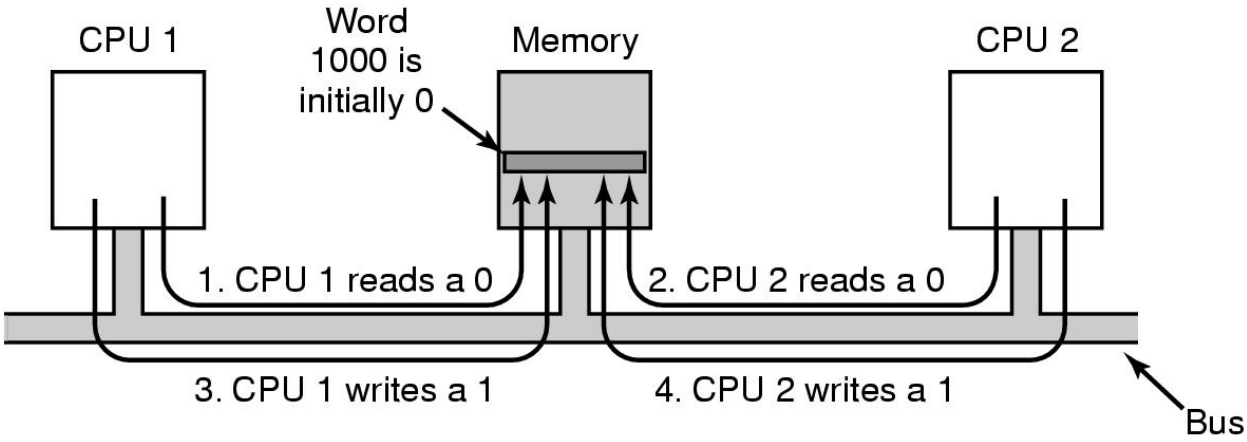


Master-Slave multiprocessors



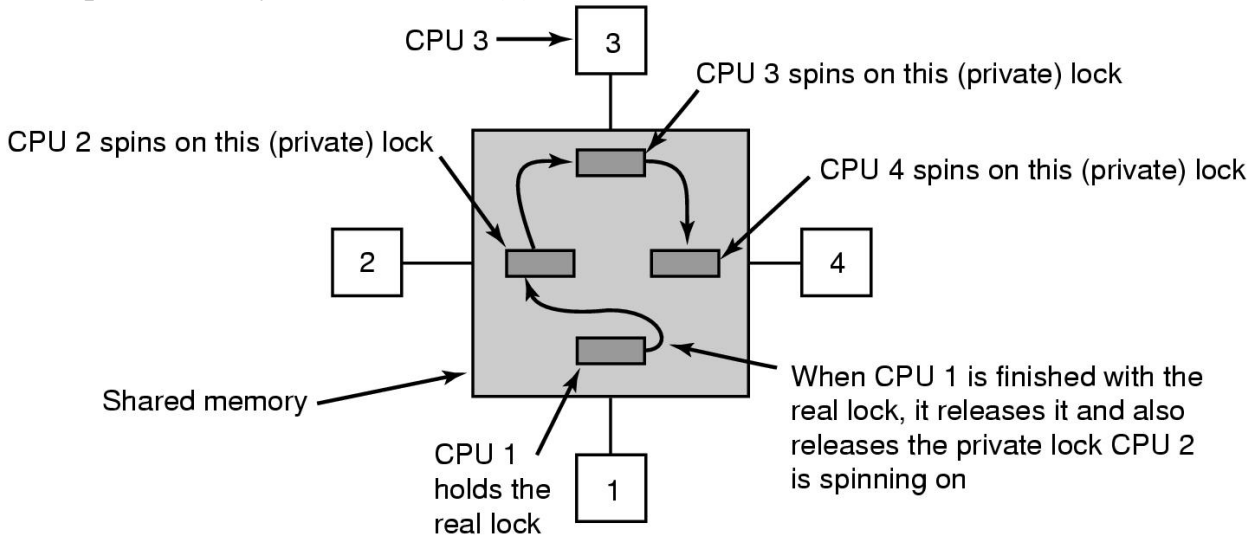
- Symmetric Multiprocessors
  - SMP multiprocessor model

Multiprocessor Synchronization



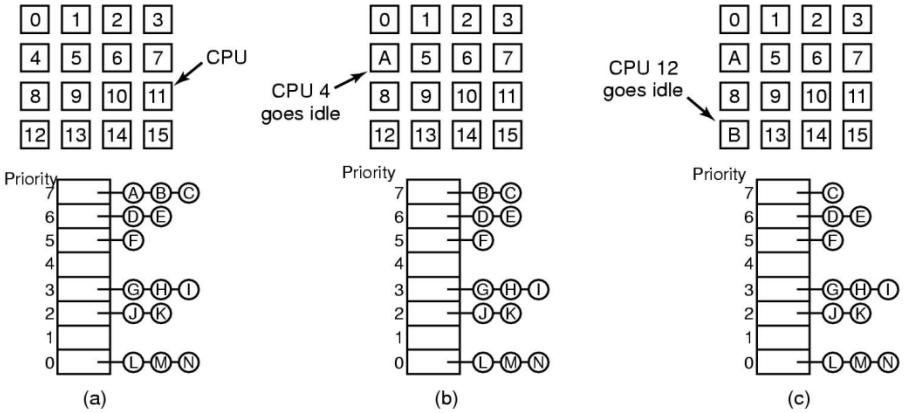
TSL instruction can fail if bus already locked

Multiprocessor Synchronization (2)

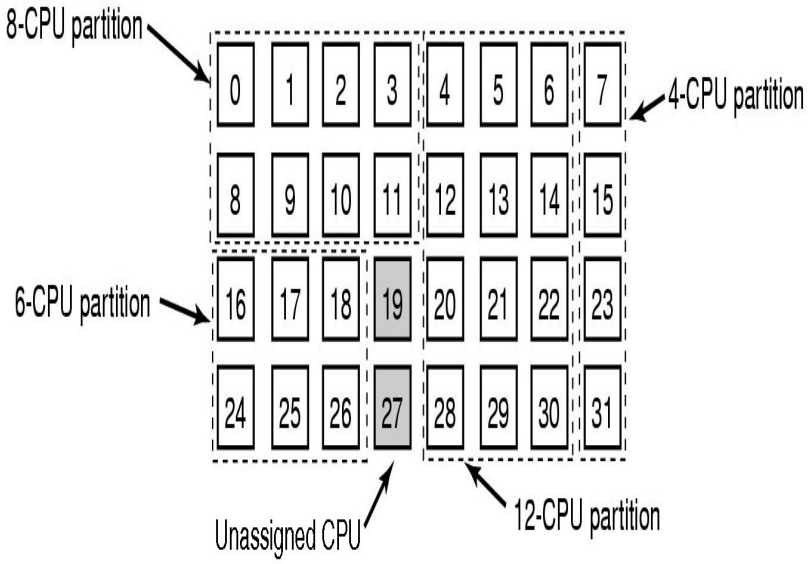


Multiple locks used to avoid cache thrashing

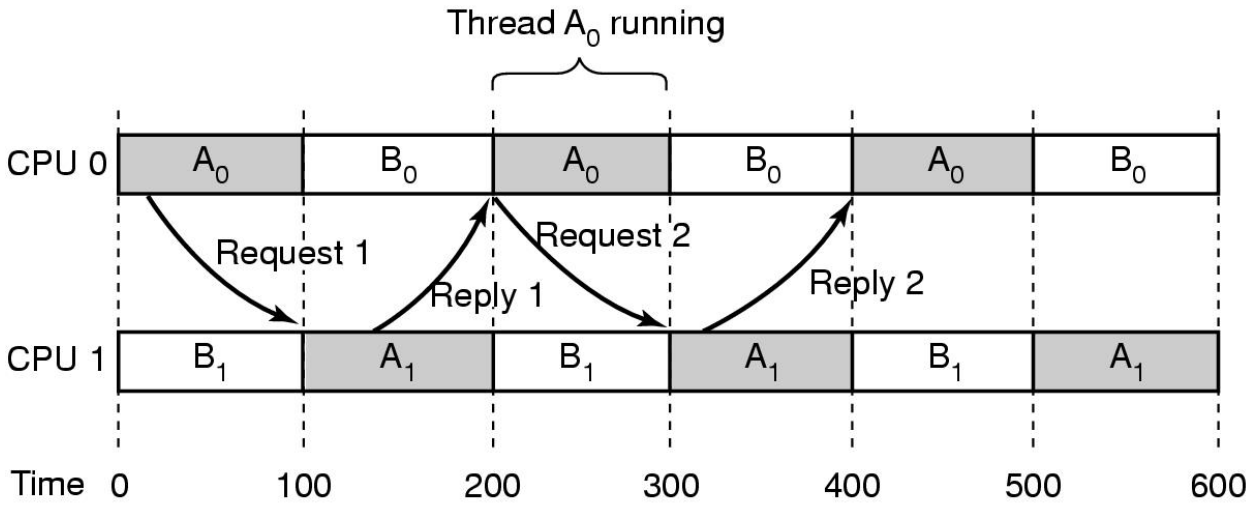
Multiprocessor Scheduling



- Timesharing
  - note use of single data structure for scheduling



- Space sharing
  - multiple threads at same time across multiple CPUs

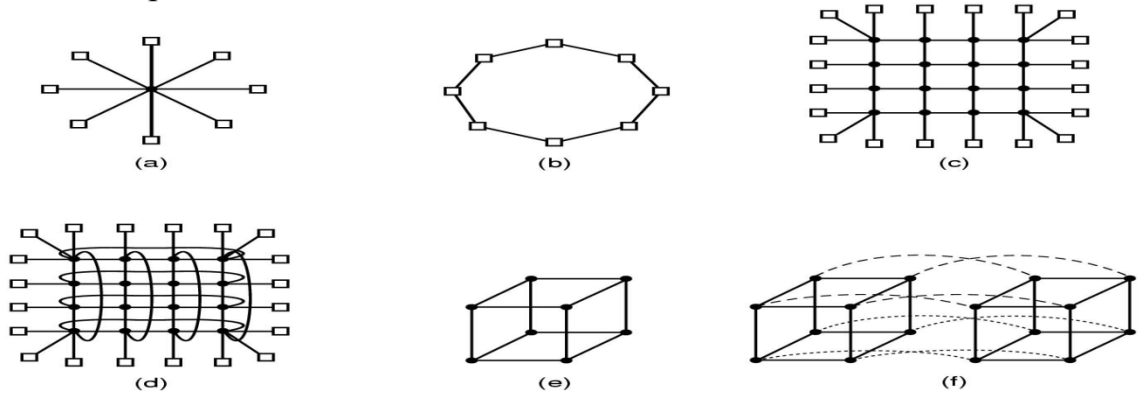


- Problem with communication between two threads
  - both belong to process A
  - both running out of phase

Multicomputers

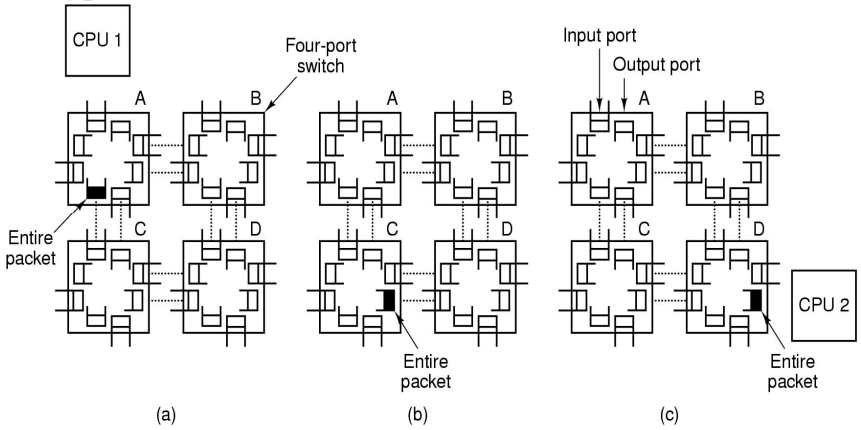
- Definition:
  Tightly-coupled CPUs that do not share memory
  - Also known as
  - cluster computers
  - clusters of workstations (COWs)

Multicomputer Hardware

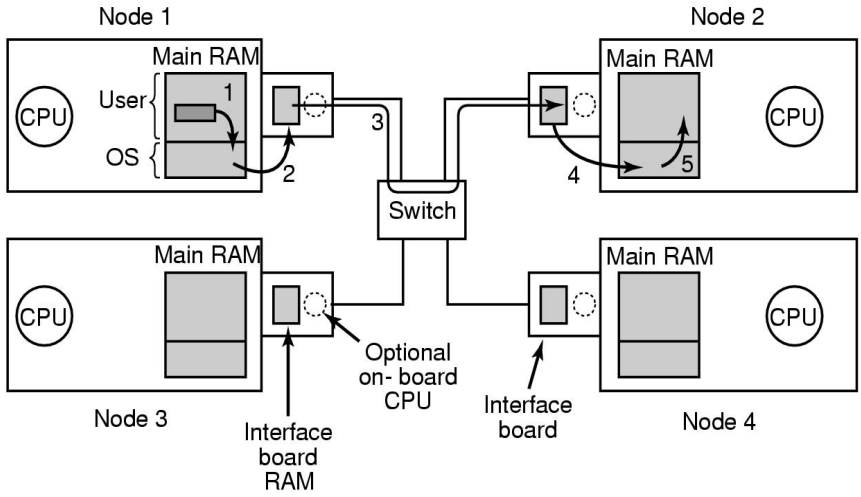


- Interconnection topologies

(a) single switch
(b) ring
(c) grid
(d) double torus
(e) cube
(f) hypercube

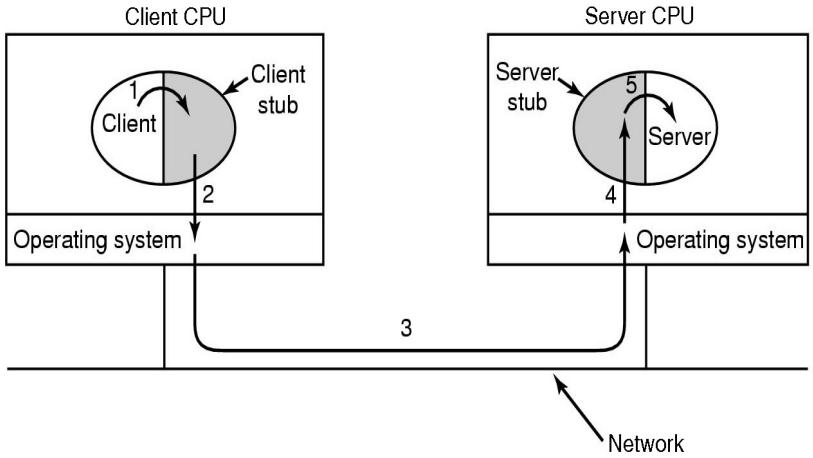


- Switching scheme
  - store-and-forward packet switching



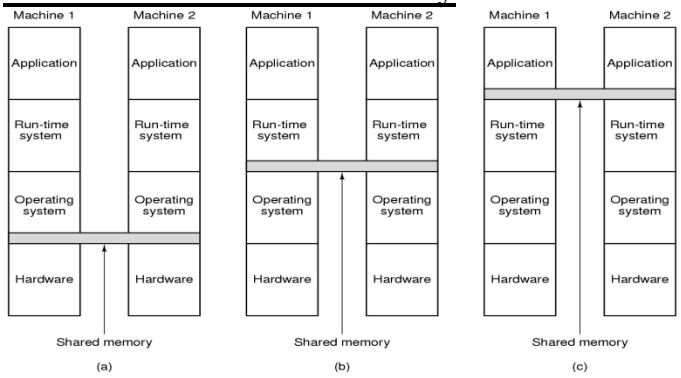
Network interface boards in a multicomputer

Remote Procedure Call



- Steps in making a remote procedure call
  - the stubs are shaded gray

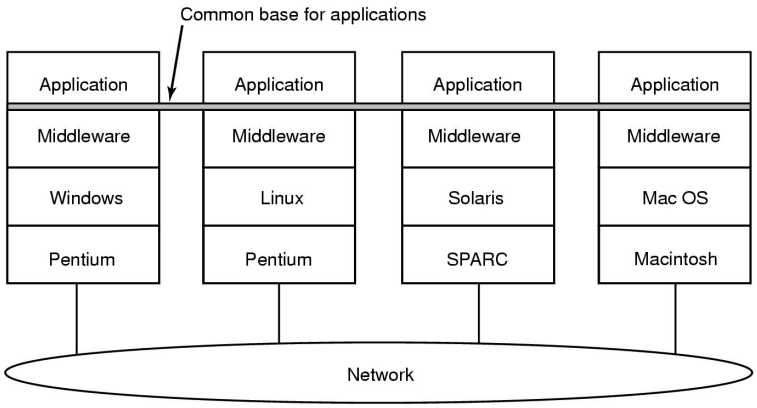
Distributed Shared Memory



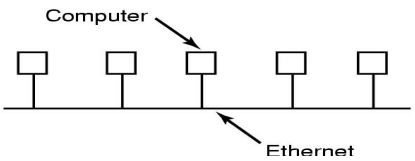
Distributed Systems

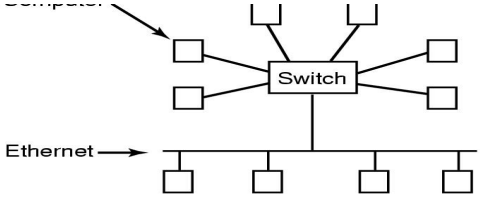
Comparison of three kinds of multiple CPU systems

Item	Multiprocessor	Multicomputer	Distributed System
Node configuration	CPU	CPU, RAM, net interface	Complete computer
Node peripherals	All shared	Shared exc. maybe disk	Full set per node
Location	Same rack	Same room	Possibly worldwide
Internode communication	Shared RAM	Dedicated interconnect	Traditional network
Operating systems	One, shared	Multiple, same	Possibly all different
File systems	One, shared	One, shared	Each node has own
Administration	One organization	One organization	Many organizations



Network Hardware





- Ethernet

(a) classic Ethernet
(b) switched Ethernet

What you the difference between Multiprocessor and Multicomputer?

Multiprocessor	Multicomputer
• A multiprocessor system is simply a computer that has more than one CPU on its motherboard.	• A computer made up of several computers. The term generally refers to an architecture in which each processor has its own memory rather than multiple processors with a shared memory
• Multiprocessing is the use of two or more central processing units (CPUs) within a single computer system.	

MODEL QUESTIONS

PART A (10X2=20)
Answer All the Questions

1. Define OS
2. Expand MS-DOS and GUI
3. Define Deadlock
4. Define Thread
5. Define MLQ
6. Compare FCFS and RR scheduling
7. Define Swapping
8. List the registers in IO port
9. Define Directories
10. Define Distributed systems

PART B (5X5=25)
Answer All the Questions

11.

a) Explain the following :

1. Real time Operating System
2. MultiProcessor Operating system

b) Explain system calls in detail

12. a) Explain dead lock avoidance Or Notes on IPC

13. a) Discuss about FCFS Or Explain Scheduling Criteria

14. a) Explain Partition memory allocation Or Block Diagram of I/O Controller

15. Explain files and Directories Or

Compare Multiprocessors, Multi Computers, and Distributed system

PART C (3X10=30)
Answer Any three Questions

16. Explain the following 1) System Calls b) OS structure
17. Explain Deadlock Prevention in detail
18. Explain SJF scheduling
19. Explain FIFO page replacement algorithm
20. Explain File system implementation