1.1 Operating System Objectives and Functions

SPPU : Aug.-18, Dec.-18, 19

- OS definition : *Operating System is a program that controls the execution of application programs. It is an interface between applications and hardware.*
- OS provides different types of view. For user, it is abstract view because it provides features which are important for users. OS is intermediary between user and the computer system.
- The major design goals/ functions of an operating system are :

1. Efficient use of a computer system 2. User convenience 3. Ability to evolve

- An operating system is software that manages the computer hardware. The hardware must provide appropriate mechanisms to ensure the correct operation of the computer system and to prevent user programs from interfering with the proper operation of the system.
- Efficiency is the one of the parameter for use of computer system. Operating system consumes some resources during its own operation. For example it uses CPU and memory. CPU is busy with scheduling and memory is occupied by instruction and required data.
- This is one type of overhead and because of this lesser resources are available to the user.
- An OS makes a computer more convenient to use. If the operating system can not allocates the free available resources to program or it over allocates the resources then efficiency is poor.

Efficient use

- For efficient use of resources, it must be monitored by operating system. Proper scheduling of resources is also required.
- Computer contains different type's resources like CPU, memory and I/O device etc. Proper monitoring is required on these resources to avoid the overhead. As per the resource, scheduling is required.
- Special attention to be given for CPU and memory. If memory is not free then user can not load new program into the memory. Then CPU will be busy with memory management.

User convenience

- User convenience is affected by computing environment of the computer system. The computing environment is comprised of computer system, its interfaces with other systems and nature of computations performed by its users.
- Computer architecture and use change the computing environment of the system. Following factors are considered while considering user convenience :

1. Good service 2. Ease of use 3. New programming model
4. Evolution 5. User friendly OS

Ability to evolve

- An OS should be constructed in such a way as to permit the effective development, testing and introduction of new system functions without at the same time interfering with service.
- Task performed by operating systems :
  1. Maintain the list of resources in the system
  2. Maintain the list of authorized users
  3. Initiate the execution of programs and process
  4. Maintain resource usage list
  5. Maintain the resource allocated list
  6. Scheduling of resources (CPU, Secondary storage etc.)
  7. Also maintain the protection information.

1.1.1 Operating System as a User Interface

- Computer system consists of software and hardware to solve specific problems. User, application program, operating system and the hardware are the components of the computer systems.
- Application program used to solve specific program. Student attendance monitoring is the example of application program.
- Operating system is a subset of the system software. OS interacts directly with the hardware to provide an interface to other system software.
- System software and hardware exist to support the creation and effective use application software.
- Fig. 1.2.1 shows conceptual view of a computer system.
- Resource sharing and resource abstraction are two key aspects of the operating system.
- Purposes of the operating system :
  1. OS provides an interface between the computer hardware and computer user (programmer). It simplifies the programmer job like editing, coding, creation.
  2. Allocation and use of computer resources among the programmer is controlled by OS.

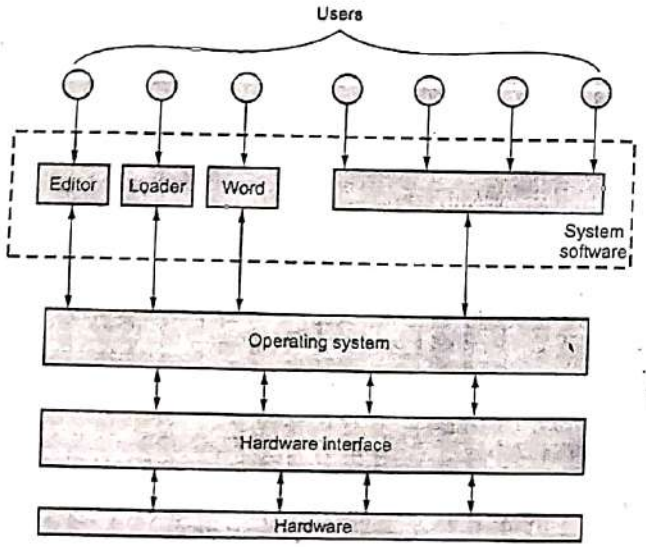


Fig. 1.1.1 Computer system conceptual view

1.1.2 Operating System as Resource Manager

- A computer is a set of resources. These resource provides various functions to the user. Functions like data movement, storing of data and program, operation on data are control by an operating system.
- Fig. 1.1.2 shows OS as a resource manager.

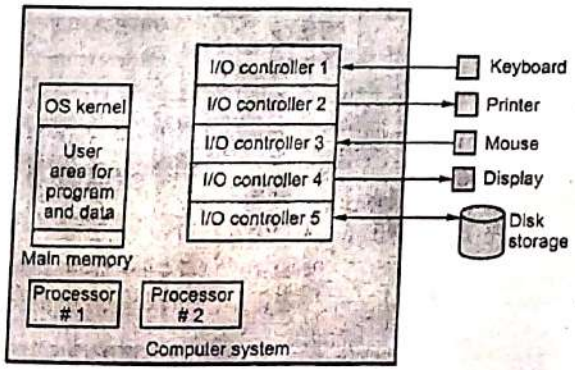


Fig. 1.1.2 OS as a resource manager

- The operating system is responsible for managing the all resources. A portion of the OS is in main memory. This portion of the OS is called kernel.
- User program and data is also stored in remaining parts of the memory. Allocation of main memory is controlled by operating system with the help of memory management hardware.
- I/O device is controlled by OS and it decides when an I/O device can be used by program in execution. Processor is one type of resource and OS control the execution of user program on the processor.
- Modern OS allows multiple programs to run at the same time. If multiple users are using computer then there is need of managing and protecting the memory, I/O devices and other resources.
- Resource management includes sharing resources in different ways. Time and space are the two concept for resource sharing.
  1. **Time :** Time slot is allocated to each program first one gets to use the resource then another and so on.
  2. **Space :** Consider the example of main memory. Main memory is normally divided up among several running programs, so each one can be resident at the same time.

University Questions

1. *What is operating system ? State and explain the basic functions of operating system.* **SPPU : Aug.-18, Insem, Marks 6**
2. *Elaborate the functions of operating system.* **SPPU : Dec.-18, -19, Insem, Marks 5**

1.2 The Evolution of Operating Systems

SPPU : June-19

- Types of an operating system are a grouping that differentiates or identifies the operating system based on how it works the type of hardware it controls and the applications it supports.

1.2.1 Batch System

- Batch system process a collection of jobs, called a batch. Batch is a sequence of user jobs.
- Job is a predefined sequence of commands, programs and data that are combined into a single unit.

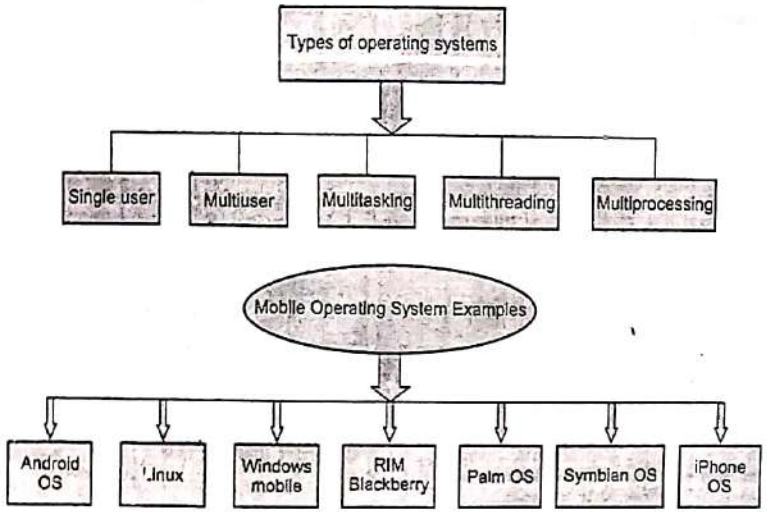


Fig. 1.2.1

- Each job in the batch is independent of other jobs in the batch. A user can define a job control specification by constructing a file with a sequence of commands.
- Jobs with similar needs were batched together to speed up processing. Card readers and tape drives are the input device in batch systems. Output devices are tape drives, card punches and line printers.
- Primary function of the batch system is to service the jobs in a batch one after another without requiring the operator's intervention. There is no need for human/user interaction with the job when it runs, since all the information required to complete job is kept in files.
- Some computer systems only did one thing at a time. They had a list of instructions to carry out and these would be carried out one after the other. This is called a **serial system.** The mechanics of development and preparation of programs in such environments are quite slow and numerous manual operations involved in the process.
- **Batch monitor** is used to implement batch processing system. Batch monitor is also called **kernel**. Kernel resides in one part of the computer main memory.
- The memory allocator managed the main memory space. Batch monitor controls the sequence of events. Main memory store the batch monitor and users program and data (jobs). Fig. 1.2.2 shows memory layout for a batch system.

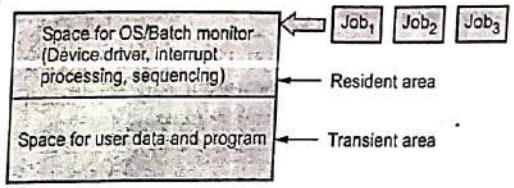


Fig. 1.2.2 Memory layout for batch system

- Computer operator gives a command to start the processing of a batch, the kernel sets up the processing of the first job. Job was selected from the job queue and loaded into main memory. When a job completed execution, its memory was released and the output for the job was copied.
- When a job is completed, it returns control to the monitor, which immediately reads in the next job.
- Fig. 1.2.3 shows concept of batch system.

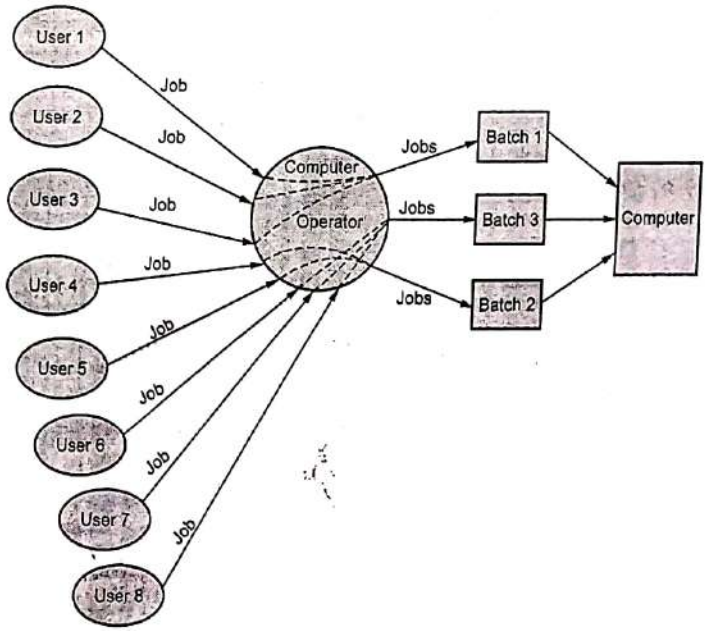


Fig. 1.2.3 Concept of batch system

- Scheduling is also simple in batch system. Jobs are processed in the order of submission i.e. first come first served fashion.
- When a job completes execution, its memory is released and the output for the job gets copied into an output spool for later printing.

1.2.1.1 Spooling

- Spooling an acronym for **simultaneous peripheral operation on line.** Spooling uses the disk as a large buffer for outputting data to printers and other devices. It can also be used for input, but is generally used for output.
- Its main use is to prevent two users from alternating printing lines to the line printer on the same page, getting their output completely mixed together. It also helps in reducing idle time and overlapped I/O and CPU.
- Batch system often provides simple forms of file management. Access to file is serial. Batch systems do not require any time critical device management.
- Batch systems are inconvenient for users because users cannot interact with their jobs to fix problems. There may also be long turnaround times. Example of this system is generating monthly bank statement.
- An optimization used to minimize the discrepancy between CPU and I/O speeds is spooling. It overlaps of one job with computation of other job.
- The spooler for instance could be reading the input of one job while printing the output of different job.
- Spooling refers to putting jobs in a buffer, a special area in memory or on a disk where a device can access them when it is ready.
- Spooling is useful because device access data at different rates. The buffer provides a waiting station where data can rest while the slower device catches up.
- Computer can perform I/O in parallel with computation, it becomes possible to have the computer read a deck of cards to a tape, drum or disk and to write out to a tape printer while it was computing. This process is called **spooling.**
- The most common spooling application is print spooling.
- Spooling batch system were the first and are the simplest of the multiprogramming systems.

Advantages of spooling

1. The spooling operation uses a disk as a very large buffer.
2. Spooling is however capable of overlapping I/O operation for one job with processor operations for another job.

1.2.1.2 Advantages and Disadvantages of Batch System

Advantages of batch system

1. Move much of the work of the operator to the computer.
2. Increased performance since it was possible for job to start as soon as the previous job finished.

Disadvantages of batch system

1. Turn around time can be large from user standpoint.
2. Program debugging is difficult.
3. There was possibility of entering jobs in infinite loop.
4. A job could corrupt the monitor, thus affecting pending jobs.
5. Due to lack of protection scheme, one batch job can affect pending jobs.

1.2.2 Multiprogramming Operating System

- CPU remains idle in batch system. At any time either CPU or I/O device was idle in batch system. To keep CPU busy, more than one program/job must be loaded for execution. It increases the CPU utilizations. So multiprogramming increases the CPU utilization.
- Resource management is the main aim of multiprogramming operating system. File system, command processor, I/O control system and transient area are the essential components of a single user operating system.
- Multiprogramming operating system divides the transient area to store the multiple programs and provides resource management to the operating system.
- The concurrent execution of programs improves the utilization of system resources. A program in execution is called a "Process", "Job" or a "Task".
- When two or more programs are in the memory at the same time, sharing the processor is referred to the multiprogramming operating system.
- Fig. 1.2.4 shows the memory layout for a multiprogramming operating system.

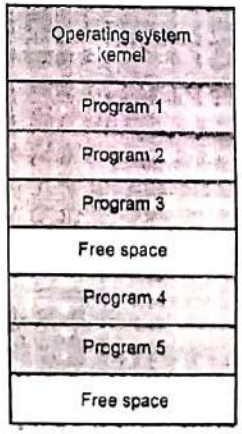


Fig. 1.2.4 Multiprogramming OS memory layout

- Operating system keeps number of programs into the memory. It selects one program from the memory and executes it. All the programs that enter the system are kept in the job pool.
- Job pool consists of all processes residing on disk and waiting to allocate primary memory. Job scheduling concept is used if there is no space for process in the primary memory.
- Memory management is also required to manage the memory for process. CPU scheduling is applied for selecting process from memory.
- When computer loads more than one program in to the memory, CPU executes one program and I/O system is busy with other programs.
- Multiprogramming operating system do not provide user interaction with the program.
- Fig. 1.2.5 Shows working of multiprogramming OS.

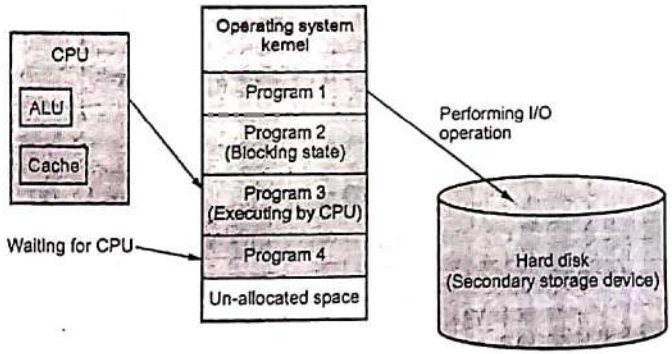


Fig. 1.2.5 Working of multiprogramming OS

- In multiprogramming operating system, programs are competing for resources. A function of the multiprogramming operating system is CPU scheduling, memory management and I/O management.
- Suppose there are four programs for execution. All four programs are loaded into the memory. CPU select first program for execution. Normally programs contain instruction for CPU and I/O operation.
- CPU bound instructions : c = a + b
- I/O bound instructions : printf, scanf etc.
- When any I/O instruction is encounter in the program, CPU select next program for processing. CPU select second program for execution and I/O system select

first program for performing I/O operation. Multiprogramming operating system monitors the state of all active programs and system resources.

Advantages :

1. CPU utilization is high.
2. It increases the degree of multiprogramming.

Disadvantages :

1. CPU scheduling is required.
2. Memory management is also required.

1.2.3 Time Sharing

- In an interactive system, many users directly interact with the computer from terminals connected to the computer.
- Processor's time which is shared among multiple users simultaneously is termed as time-sharing.
- Time sharing is logical extension of multiprogramming OS. Fig. 1.2.6 shows time sharing system.

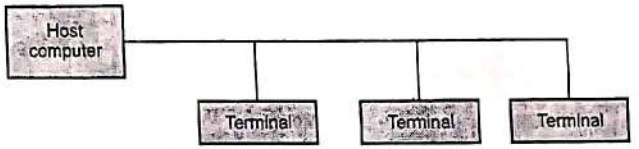


Fig. 1.2.6 Time sharing

- Time sharing is a method that allows multiple users to share resources at the same time. Multiple users in various locations can use a specific computer system at a time
- Time sharing is essentially a rapid time division multiplexing of the processor time among several processes. The processor switching is so frequent that it almost seems each process has its own dedicated processor.
- Time sharing OS is designed to provide a quick response to sub-requests made by users.

- The processor time is shared between multiple users at a time. The processor allows each user program to execute for a small time quantum. Moreover, time sharing systems use multiprogramming and multitasking.
- The operating system provides immediate feedback to the user and response time can be in seconds, depending on the number of active users.
- The scheduling technique used by a time-sharing kernel is called round robin scheduling with time slicing.
- If the time slice elapses before the process completes servicing of a sub-request, the kernel pre-empts the process, moves it to the end of the scheduling queue and schedules another process.
- The main difference between Multi-programmed Batch Systems and Time-Sharing Systems is that in case of Multi-programmed batch systems, the objective is to maximize processor use, whereas in Time Sharing Systems, the objective is to minimize response time.
- **Features of time sharing system :**
  1. User interaction with program is possible.
  2. Time sharing system uses medium term scheduling such as round robin for the foreground.
  3. Each user is given a time slice for executing job in round robin fashion.
- **Advantages of Timesharing operating systems :**
  1. Provides the advantage of quick response.
  2. Avoids duplication of software.
  3. Reduces CPU idle time.
- **Disadvantages of Time-sharing operating systems :**
  1. Problem of reliability.
  2. Question of security and integrity of user programs and data.
  3. Problem of data communication.

1.2.3.1 Difference between Time Sharing OS and Multiprogramming OS

Sr. No.	Time sharing OS	Multiprogramming OS
1.	It enables execution of multiple tasks and processes at the same processes to increases CPU performance.	Multiple programs reside in the main memory simultaneously to improve CPU utilization.
2.	It is based on the concept of time sharing.	It is based on the concept of context switching.
3.	The idea is to allow multiple processes to run simultaneously via time sharing.	The idea is to reduce the CPU idle time for as long as possible.
4.	It takes less time to execute the task allocation.	It takes more time to execute the process.
5.	Principle objective is minimize response time.	Principle objective is maximize processor use.

University Question

1. *Describe in brief the evolution of Operating System.* **SPPU : June-19, Insem, Marks 5**

1.3 Developments Leading to Modern Operating Systems

SPPU : Aug.-18, June-19

- For software development professionals and computer science students, Modern Operating Systems gives a solid conceptual overview of operating system design, including detailed case studies of Unix/Linux and Windows 2000.
- What makes an operating system modern ?
- According to author Andrew Tanenbaum, it is the awareness of high-demand computer applications primarily in the areas of multimedia, parallel and distributed computing, and security.
- The development of faster and more advanced hardware has driven progress in software, including enhancements to the operating system. It is one thing to run an old operating system on current hardware, and another to effectively leverage current hardware to best serve modern software applications.
- Following are the different categories of new types of operating systems :
  1. Microkernel architecture
  2. Multithreading
  3. Symmetric multiprocessing
  4. Distributed operating system
  5. Object oriented system design.

- In a **monolithic kernel**, most Kernel services are part of the kernel, itself. This does not mean that the services cannot be independent of each other. However, the software is very tightly integrated to the rest of the kernel. This makes monolithic kernels very fast and efficient, compared to other designs.
- A **Microkernel** is a kernel design that provides no OS services at all, only the mechanisms needed to implement those services. Because of this, the Kernel itself is usually quite small compared to Monolithic kernels. For example, a microkernel may implement low level memory management and thread management, and Inter Process Communication. The Kernel would use external user mode services, such as device drivers and file systems, rather than everything implemented as part of the kernel because of this, if an external service crashes, the system may still be functional, and the system will not crash.
- Process is collection of threads.
- Thread is a light weight process. A thread is flow of execution through the process code. It contains program counter, stack pointer and stack for data storage.
- Multithreading : Process creates number of multiple threads. These all thread can run simultaneously.

1.3.1 Kernel

- The Kernel is a software code that resides in the central core of a operating system. It has complete control over the system. A Kernel is different than the snell. The shell is the outermost part of an operating system and a program that interacts with user commands. Fig. 1.3.1 shows the position of the Kernel.

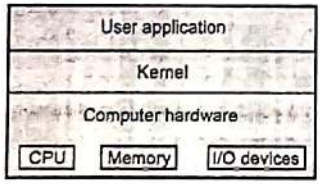


Fig. 1.3.1 Kernel

- The kernel does not interact directly with the user. But it interacts with the shell, other programs and hardware.
- When operating system boots, kernel is the first part of the operating system to load into memory. Kernel remains in memory for the entire duration of the computer session. The kernel code is usually loaded into a protected area of memory.
- The kernel performs its tasks in kernel space. Executing processes and handling interrupts are performed by kernel in kernel space. User performs its task in user area of memory. Memory is divided into system area and user area. This memory

separation is made in order to prevent user data and kernel data from interfering with each other.

- When a computer crashes, it actually means the kernel has crashed. Single program crash is not a kernel crash. The kernel provides services for process management, memory management, file management and I/O management. System calls are used for providing this type of services.
- Kernel content will change according to the operating system but typically it includes :
  1. Scheduler
  2. Supervisor
  3. Interrupt handler
  4. Memory manager.
- Scheduler allocates the kernel's processing time to various processes. Supervisor grants the permission to use computer system resources to each process. Interrupt handler handles all requests from the various hardware devices which compete for the kernel's services. Memory manager allocates space in memory for all users of the kernel services.

Types of kernel :

1. Monolithic kernel
2. Microkernel
3. Hybrid kernel
4. Exo-kernel.

1.3.2 Monolithic Kernel

- Traditional UNIX operating system uses monolithic kernel architecture. The entire operating system runs as a single program in kernel mode. Program contains operating system core function and device drivers.
- Most of the operation performed by kernel is via system call. Required system calls are made within programs and a checked copy of the request is passed through a system call. Fig. 1.3.2 shows monolithic kernel.
- LINUX operating system and FreeBSD uses modern monolithic kernel architecture. It loads the modules at run time. There is easy access of kernel function as required and minimize the code running in kernel space.
- Monolithic kernel used in Windows 95, Windows 98, Linux and FreeBSD etc.

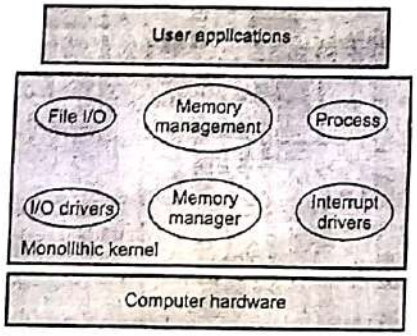


Fig. 1.3.2 Monolithic kernel

Advantages :

1. Simple to design and implement.
2. Simplicity provides speed on simple hardware.
3. It can be expanded using module system.
4. Time tested and design well known.

Disadvantages :

1. Runtime loading and unloading is not possible because of module system.
2. If code base size increases, maintain is difficult.
3. Fault tolerance is low.

1.3.3 Microkernel

- Microkernel provides minimal services like defining memory address space, IPC and process management. It is small operating core. Hardware resource management is implemented whenever process is executing. The function of microkernel is to provide a communication facility between the client programs. It also provides facility to various services which are running in user space. Message passing method is for communication two processes.
- Microkernel runs in kernel mode and rest run in normal user processes. Microkernel also provides more security and reliability. Most of the services are running as user rather than kernel processes. By running device driver and file system as a separate user process, a error in one can crash only single component. Fig. 1.3.3 shows microkernel.

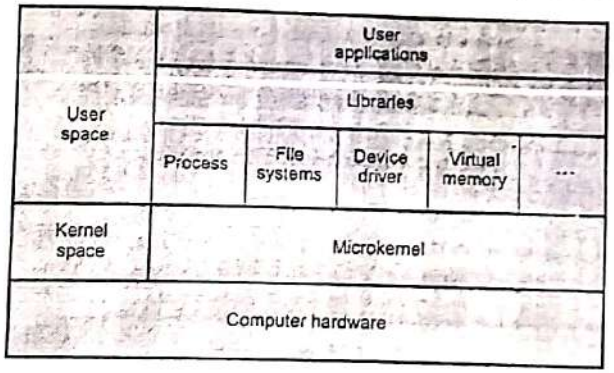


Fig. 1.3.3 Microkernel

- Mach operating system uses microkernel architecture. Microkernel in Windows NT operating system provides portability and modularity. The kernel is surrounded by a number of compact subsystems so that the task of implementing Windows NT on a variety of platform is easy.
- Microkernel architecture assigns only a few essential functions to the kernel, including address space, IPC and basic scheduling. QNX is a real time operating system that is also based upon the microkernel design.
- Main disadvantage is poor performance due to increased system overhead from message passing.

Advantages of microkernel :

1. Microkernel allows the addition of new services.
2. Microkernel architecture design provides a uniform interface on requests made by a process.
3. Microkernel architecture supports object oriented operating system.
4. Modular design helps to enhance reliability.
5. Microkernel lends itself to distributed system support.
6. Microkernel architecture support flexibility. User can add or subtract services according to the requirement.

1.3.4 Comparison between Monolithic Kernel and Microkernel

Sr. No.	Monolithic Kernel	Microkernel
1.	Kernel size is large.	Kernel size is small.
2.	OS is complex to design.	OS is easy to design, implement and install.
3.	Request may be serviced faster.	Request may be serviced slower than monolithic Kernel.
4.	All the operating system services are included in the Kernel.	Kernel provides only IPC and low level device management services.
5.	No message passing and no context switching are required while the Kernel is performing the job.	Microkernel requires message passing and context switches.

University Questions

1. *Differentiate between monolithic and microkernel architectures.* **SPPU : Aug.-18, Insem, Marks 4**
2. *Explain the difference between a monolithic and a microkernel with advantages and disadvantages.* **SPPU : June-19, Insem, Marks 5**

1.4 Virtual Machines

SPPU : Aug.-17, Dec.-17

- In a pure virtual machine architecture the operating system gives each process the illusion that it is the only process on the machine. The user writes an application as if only its code were running on the system.
- Each user interacts with the computer by typing commands to the virtual machine on a virtual system console and receiving results back from the machine as soon as they are computed.
- Each user directs the virtual machine to perform different commands. These commands are then executed on the physical machine in a multiprogramming environments.
- Virtualization is an abstraction layer that decouples the physical hardware from the operating system to deliver greater IT resource utilization and flexibility.
- It allows multiple virtual machines, with heterogeneous operating systems to run in isolation, side-by-side on the same physical machine. Fig. 1.4.1 shows virtual machine.

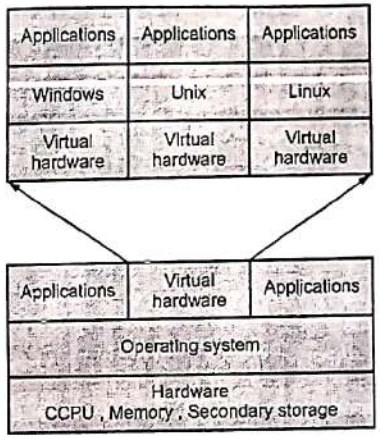


Fig. 1.4.1 Virtual machine

- Each virtual machine has its own set of virtual hardware (e.g., RAM, CPU, NIC, etc.) upon which an operating system and applications are loaded. The operating system creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory.
- The main components of virtual machine are the control program, conversational monitor system, remote spooling communication and interactive problem control system. The control program creates the environments in which virtual machines can executes. It also manages the real machines underlying the virtual machine environment.
- The java virtual machine is one of the most widely used virtual machines.
- Virtual machines tend to be less efficient than real machines because they access the hardware indirectly.

Benefits :

1. There is no overlap amongst memory as each Virtual Memory has its own memory space.
2. Virtual machines are completely isolated from the host machine and other virtual machines.
3. Data does not leak across virtual machines.

Virtualization

- Virtualization means running multiple machines on a single hardware. The "Real" hardware invisible to operating system. OS only sees an abstracted out picture. Only Virtual Machine Monitor (VMM) talks to hardware.

- It is "a technique for hiding the physical characteristics of computing resources from the way in which other systems, applications, or end users interact with those resources. This includes making a single physical resource appear to function as multiple logical resources; or it can include making multiple physical resources appear as a single logical resource."
- It is divided into two main categories :
  1. Platform virtualization involves the simulation of virtual machines.
  2. Resource virtualization involves the simulation of combined, fragmented, or simplified resources.

Hypervisor

- In computing, a hypervisor is a virtualization platform that allows multiple operating systems to run on a host computer at the same time. The term usually refers to an implementation using full virtualization.
- Hypervisors are currently classified in two types :
  1. Type 1 hypervisor is software that runs directly on a given hardware platform. A "guest" operating system thus runs at the second level above the hardware.
  2. Type 2 hypervisor is software that runs within an operating system environment. A "guest" operating system thus runs at the third level above the hardware.
- Bochs and QEMU are PC emulators that allow operating systems such as Windows or Linux to be run in the user-space of a Linux operating system.
- VMware is a popular commercial full-virtualization solution that can virtualizes unmodified operating systems.
- Xen is an open source para-virtualization solution that requires modifications to the guest operating systems but achieves near native performance by collaborating with the hypervisor.
- Microsoft Virtual PC is a para-virtualization virtual machine approach.
- User-mode Linux (UML) is another para-virtualization solution that is open source. Each guest operating system executes as a process of the host operating system.
- Cooperative Linux, is a virtualization solution that allows two operating systems to cooperatively share the underlying hardware.
- Linux-Vserver is an operating system-level virtualization solution for GNU/Linux systems with secure isolation of independent guest servers.
- The Linux KVM is virtualization technology that has been integrated into the mainline Linux kernel. Runs as a single kernel loadable module, a Linux kernel running on virtualization-capable hardware is able to act as a hypervisor and support unmodified Linux and Windows guest operating systems.

- Fig. 1.4.2 shows para-virtualization architecture. In Para-virtualization, the virtual machine does not necessarily simulate hardware, but instead offers a special API that can only be used by modifying the "guest" OS. This system call to the hypervisor is called a "hypercall" in Xen.

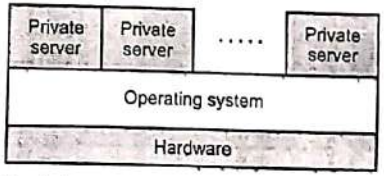


Fig. 1.4.2 Para-virtualization architecture

- Para-virtualization shares the process with the guest operating system

Problems with para-virtualization

1. Para-virtualized systems won't run on native hardware
2. There are many different para-virtualization systems that use different commands, etc.

Platform virtualization

- The creation of a virtual machine using a combination of hardware and software is referred to as platform virtualization.
- Platform virtualization is performed on a given hardware platform by "host" software, which creates a simulated computer environment for its "guest" software.
- The "guest" software, which is often itself a complete operating system, runs just as if it were installed on a stand-alone hardware platform.
- Typically, many such virtual machines are simulated on a given physical machine.
- For the "guest" system to function, the simulation must be robust enough to support all the guest system's external interfaces, which may include hardware drivers.

Resource virtualization

- The basic concept of platform virtualization, was later extended to the virtualization of specific system resources, such as storage volumes, name spaces, and network resources.
- Resource aggregation, spanning, or concatenation combines individual components into larger resources or resource pools. For example : RAID and volume managers combine many disks into one large logical disk.

- Virtual Private Network (VPN), Network Address Translation (NAT), and similar networking technologies create a virtualized network namespace within or across network subnets. Multiprocessor and multi-core computer systems often present what appears as a single, fast processor.

University Questions

1. *Explain the concept of virtual machine with its benefits.* **SPPU : Aug.-17, Insem, Marks 4**
2. *What is a virtual machine ? Explain the concept of virtualization.* **SPPU : Dec.-17, Insem, Marks 5**

1.5 OS Services

SPPU : Aug.-17

- Operating system provides different types of services to different users. It provides services to program like load of data into memory, allocating disk for storage, files or directory for open or read etc.
- Services will change according to the operating system. May be the two different operating system provides same type of services with different names. OS makes programmer job easy by providing different services.
- Following are the list of services provided by operating systems :

1. Program execution	2. Input-output operation
3. Error detection	4. File and directory operation
5. Communication	6. Graphical User Interface

1. **Program execution :** Before executing the program, it is loaded into the memory by operating system. Once program loads into memory, its start execution. Program finishes its execution with error or without error. It is up to the user for next operation.
2. **Input - output operation :** Program is combination of input and output statement. While executing the program, it requires I/O device. OS provides the I/O devices to the program.
3. **Error detection :** Error is related to the memory, CPU, I/O device and in the user program. Memory is full, stack overflow, file not found, directory not exist, printer is not ready, attempt to access illegal memory are the example of error detection.
4. **File and directory operation :** User wants to read and writes the file and directory. User wants to search the file/directory, rename file, and modify the file etc. user also create the file or directory. All these operation is performed by user by using help of operating system.

5. **Communication :** Communication may be inter-process communication and any other type of communication. Data transfer is one type of communication. Communication is in between two process of same machine or two process of different machine. Pipe, shared memory, socket and message passing are the different methods of communication.
6. **Graphical user interface :** User interacts with operating system by using user interface. All operating system provides user interface. Command line interface and batch interface are two types of user interface used in the operating system. In command line interface, user enters the text command for performing operation. Batch interface uses files. A file contains the command for various operations. When file executes, command output is displayed on the screen.

- All above services provided by operating system is only for single user. If suppose there are multiple user in the system, then operating system provides some additional services. These services are resource allocation, accounting, protection of data and security.
- Different types of resources are managed by the operating system. Multiple users require different resource for their execution. One type of resource may be required by two different users. So resource allocation is necessary.
- Accounting means keeping information of resources and users. Which types of resources are allocated to which user and whether particular user has permission for using this type of resources.

University Question

1. *State and explain different services provided by an operating system.* **SPPU : Aug.-17, Insem, Marks 6**

1.6 Introduction to Linux OS

- Linux is developed in 1991 by Linus Torvalds. It is used in most of the computers, ranging from super computers to embedded system. Linux is a free operating system based on UNIX standards.
- Linux is multi user, multi-tasking, time sharing and monolithic kernel etc.
- Linux is Free Open Source Software. Free means liberty and not related to price or cost and Open means source code is available and anybody can contribute to the development.

- The core Linux operating system kernel is entirely original, but it can run much existing free UNIX software, resulting in an entire UNIX-compatible operating system free from proprietary code.
- **Kernel** is a main program of UNIX system. It controls hardwares, CPU, memory, hard disk, network card etc.
- Shell is an interface between user and kernel. Shell interprets your input as commands and passes them to kernel.

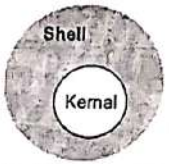


Fig. 1.6.1 Shell and Kernel

Design principles

- Linux is a multiuser, multitasking system with a full set of UNIX-compatible tools.
- Its file system adheres to traditional UNIX semantics, and it fully implements the standard UNIX networking model.
- Main design goals are speed, efficiency and standardization.
- Linux is designed to be compliant with the relevant POSIX documents; at least two Linux distributions have achieved official POSIX certification.
- The Linux programming interface adheres to the SVR4 UNIX semantics, rather than to BSD behaviour.

1.6.1 Components of Linux System

- Linux operating system consists of three components :
  1. Kernel
  2. System library
  3. System utility
- Fig. 1.6.2 shows Linux operating system components.
  1. **Kernel :** Kernel is the heart of the Linux operating system. It is responsible for all major activities of Linux OS. It provides important abstraction to hide low level hardware information to user or application program. Kernel is combination of various modules and it interacts directly with the underlying hardware. The Linux kernel consists of several important parts : Process management, memory management, hardware device drivers, file system drivers, network management etc.

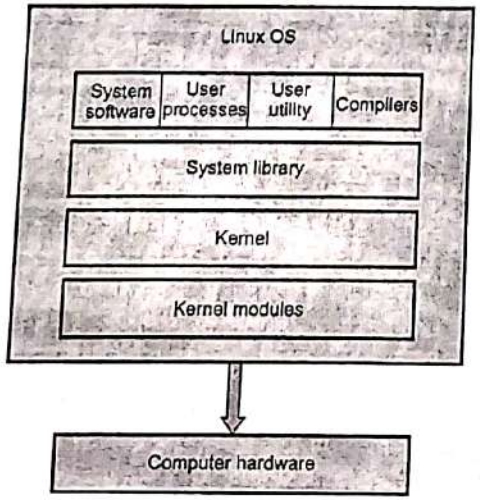


Fig. 1.6.2 Linux OS components

  2. **System Library :** System libraries are special functions or programs. Libraries are used for accessing kernel's features. These libraries implements most of the functionalities of the operating system and do not requires kernel module's code access rights.
  3. **System Utility :** System utility programs are responsible to do specialized, individual level tasks.
- Privileged mode is called kernel mode in Linux system. All the kernel code executes in the privileged mode with full access to the hardware resource of the computer. In Linux operating system, no user mode code is built into the kernel.

1.7 BASH Shell Scripting

SPPU : Aug.-17,18, Dec.-17

- Bash is the shell or command language interpreter, for the GNU operating system. The name is an acronym for the 'Bourne-Again SHell'.
- Bash is largely compatible with sh and incorporates useful features from the Korn shell ksh and the C shell csh.
- Bash is based on the Bourne shell, sh, originally written by Stephen Bourne. Bash is extremely portable and can be built on most UNIX systems because many of the environment dependent variables are determined at build time. Bash has been ported as a shell for several non-UNIX platforms like QNX, Minix, OS/2, Windows 95 and Windows NT.
- Startup files are scripts that are read and executed by Bash when it starts.
- Interactive means user can enter commands. The shell is not running because a script has been activated. A login shell means that user got the shell after authenticating to the system, usually by giving your user name and password. A non-login shell means that you did not have to authenticate to the system.
- Error messages are printed if configuration files exist but are not readable. If a file does not exist, bash searches for the next. Shell scripts are just ordinary text files.
- Commands & variables read when shell is started. If variables are placed in system-wide init files, available to every shell. Commands and aliases cannot be exported so must be placed in user-specific init.
- If the shell is a login shell, it looks for one of the following (in order)
  1. ~/.bash_profile
  2. ~/.bash_login

3. ~/.profile

- If it's an interactive, non-login shell, it reads **~/.bashrc.**

Bash configuration files

Files	Description
.bashrc	Main configuration file and it contains list of commands. When shell is open, these commands are executed.
.bash_profile	This file maintains the list of all login users.
.bash_history	Maintain the history of bash shell
.bash_logout	This file exists in users home directory. Executed by bash when login shell exits.

- Check your shell type :

```
$ echo $SHELL
```

1.7.1 Executing Script

- A shell script is a sequence of commands for which you have a repeated use. This sequence is typically executed by entering the name of the script on the command line.
- To create a shell script, open a new empty file in your editor. Any text editor will do : **vim, emacs, gedit, dtpad** are all valid.
- A bash script is a file containing a list of commands to be executed by the bash shell. The very simplest scripts contain a set of commands that you would normally enter from the keyboard.
- Bash comments start with a **hash mark (#)** and continue to the end of the line. As on the command line, user can break a single logical line onto multiple physical lines by escaping the new line with a backslash. User can also put more than one statement on a line by separating the statements with semicolons.
- Consider the following example :

```
#! /bin/bash
# script to turn the screen blue
echo "Hello, It is a cool day"
```

- The first line tells Linux to use the bash interpreter to run this script. Script always starts with the two character '#!' which is called as she-bang. This is to indicate that the file is a script and should be executed using the interpreter **(/bin/bash)** specified by the rest of the first line in the file.
- Second line is comment line because hash (#) is used for comments.

- When you execute the command "bash hello.sh", it starts the non-interactive shell and passes the filename as an argument to it.
- To run the script :

```
$ chmod 741 hello.sh
$ ./hello.sh
Hello, It is a cool day
```

- The first line tells the operating system which shell to spawn to execute the script. In the above example, bash interpreter which interprets the script and executes the commands one by one from top to bottom.
- The **"echo"** is a command which simply outputs the argument we give to it. It is also used to print the value of the variable.
- When the bash is invoked as an interactive shell, it first reads and executes commands from **/etc/profile.** If **/etc/profile** does not exist, it reads and executes the commands from **~/.bash_profile, ~/.bash_login** and **~/.profile** in the given order.

Command line arguments

- Arguments or variables may be passed to a shell script. Simply list the arguments on the command line when running a shell script. In the shell script, $0 is the name of the command run (usually the name of the shell script file); $1 is the first argument, $2 is the second argument, $3 is the third argument, etc.
- The variable $# contains the number of command line arguments that were supplied and the variable $* contains all the arguments at once. The variable $# reports the number of command line arguments passed to the shell script program.

University Questions

1. *Write a shell script for sorting a given list of numbers using any sorting strategy.* **SPPU : Aug.-17, Insem, Marks 6**
2. *Write a shell program to check if a given string is palindrome or not.* **SPPU : Dec.-17, Insem, Marks 5**
3. *Write a shell script to check if given string is a palindrome or not.* **SPPU : Aug.-18, Insem, Marks 6**

1.8 Basic Shell Commands

SPPU : Aug.-17, 18, June-18, Dec.-19

- From a certain point of view, Linux can be seen primarily as a set of utilities bonded together by a common kernel.

- Linux utilities come in two basic flavors : open source and commercial. Some commercial products, such as RealPlayer, are available for free, while others, such as the BRU tape backup software, are available for a fee. The majority of Linux utilities are free, open-source packages.
- Traditional UNIX utilities evolved in the command-line environment. Because of that they can be joined together to form powerful shell scripts and time based jobs. Some of the newer, graphical utilities are designed to be clicked and pointed at in the X Window System.

1.8.1 Internal and External Commands

- Linux commands are nothing but the text written in the shell or terminal that the terminal understands. Linux commands usually perform certain specific operations such as editing a text file, making, removing and moving a directory etc. The shell or the terminal of Linux provides a use set of commands and allows us to write the script that is able to perform various operations.
- Commands in Linux are usually divided into two sections internal commands and the external commands.

1. **External commands :** These are the most commonly used utilities and programs such as cat, ls and so forth. External commands are the commands that are executed by the kernel. These commands will have a process id running for it.
2. **Internal commands :** The shell has a number of built-in commands such as cd and echo which don't generate a process. Internal commands are the commands that are executed directly by the shell. These commands will not have a separate process running for each.

1.8.2 File Handling Utilities

- We discuss some of the more important directory and file handling commands.

1. pwd : Print [current] working directory

- *pwd* displays the full absolute path to your current location in the file system. So,

```
$ pwd <Enter>
/usr/bin
```

implies that */usr/bin* is the current working directory.

2. ls : list directory

- *ls* lists the contents of a directory. If no target directory is given, then the contents of the current working directory are displayed. So, if the current working directory is /,


```
$ ls <Enter>
bin  dev home mnt  share usr var
boot etc lib  proc sbin  tmp vol
```

- Actually, *ls* doesn't show you all the entries in a directory - files and directories that begin with a dot (.) are hidden.
- The reason for this is that files that begin with a. usually contain important configuration information and should not be changed under normal circumstances. If you want to see all files, *ls* supports the *-a* option :

```
$ ls -a
```

- Even this listing is not that helpful - there are no hints to properties such as the size, type and ownership of files, just their names.
- To see more detailed information, use the *-l* option (long listing), which can be combined with the *-a* option as follows :

```
$ ls -l /user/rakshita <Enter>
total 3
```

- r w - r - - r - -	1	rakshita student	13	Jan 1	10:39	data1
-r w x r w x r w x	1	rakshita student	19	Feb 10	21:35	data2
d r w x r w x r - -	2	rakshita student	32	Jan 16	16:50	prog-files

```
$
```

- The first character in column 1 can be '-' (ordinary file), 'd' (directory file) or 'b', 'c', 'p' (special file).
- The output of the ls -l command can be explained as in Table 1.8.1.

Column	Description
1	File type and FAP
2	Number of links
3	File Owner
4	Group Owner (group name)
5	File size (in bytes)
6, 7 and 8	Day and time of last modification in the file
9	Name of file

Table 1.8.1 Output of ls -l command

3. cd : Change directory

- Allows you to change your current working directory to any other you have permission to enter. Change directory by itself sends you to your home dir. The address can be relative or absolute.

```
$ pwd <Enter>
/home/iresh
$ cd program <Enter>
$ pwd <Enter>
/home/iresh/program
$
```

Another use of cd command

```
$ pwd <Enter>
/user/rakshita
$ cd /user <Enter>
$ pwd <Enter>
/user
$
```

- Note that the complete path-name has been specified with the cd command. UNIX also allows relative path-names with commands. For example, rakshita can enter the following command after logging in, to change to the parent directory of his HOME directory :

```
$ pwd <Enter>
/user/rakshita
$ cd .. <Enter>
$ pwd <Enter>
/user
$
```

- The two dots refer to the parent directory of the current directory. The cd command without any path-name always takes the user back to the HOME directory.

4. mkdir : Make directory

- The *mkdir* command is used to create directories.

```
$ mkdir prog-files <Enter>
$
```

- The subdirectory prog-files is created under the current directory. However, the new directory does not become the current directory. Complete path-names can be specified with *mkdir*.

```
$ mkdir /user/rakshita/prog-files <Enter>
```

5. rmdir : Removing a directory

- The *rmdir* removes the directory specified.

```
$ rmdir cob-prog <Enter>
$
```

where cob-prog is the directory which is deleted

- A directory can be deleted only if it is :

i. Empty (does not contain files or subdirectories)

ii. Not the current directory.

- Complete path-names may also be specified with rmdir.

```
$ rmdir /user/gomes/cob-prog <Enter>
```

6. cat : Concatenate

- The *cat* command displays the contents of the file specified.

```
$ cat data1 <Enter>
A sample file
$
```

- The command assumes that data1 is in the current. Complete path-names can also be specified to display a file in another directory. The *cat* command can also display more than one file as shown in the following command :

```
$ cat data1 data2 <Enter>
A sample file
Another sample file          (contents of data2)
$
```

7. cp : Copy

- The *cp* command duplicates the contents of the source file into a target file.

```
$ cp data1 data3 <Enter>
```

- In the above example, the contents of data1 are copied to a new file data3. If data3 already exists, its contents will be overwritten by the contents of data1.
- Complete path-names can be specified with the *cp* command to copy files across directories .

8. rm : Removing files

- The *rm* (remove) command is used to delete files.

```
$ rm data1 <Enter>
$
```

- If the file is not located in the current directory, the complete path-names have to be given.

```
$ rm /user/gomes/data1 <Enter>
```

9. mv : Move (Renaming files)

- The *mv* command changes the name of a file or directory. It can also be used to move files from one directory to another.

```
$ mv data3 newfile <Enter>
$
$ mv /user/rakshita/prog-files  /user/rakshita/programs <Enter>
$
```

- In UNIX, a file can be moved to another directory (not copied) as shown below :

```
$ mv data3 /user/rakshita/programs/data3 <Enter>
$
```

1.8.3 Security File Permissions

- File Access Permissions (FAP) refer to the permissions associated with a file with respect to the following :
  1. The File Owner
  2. The Group Owner
  3. Other Users.

Table 1.8.2 summarizes the access permissions available.

Access type	Denoted by	What it means
Read	r	Allows displaying, copying and compiling a file
Write	w	Allows editing and deleting of a file
Execute	x	Allows execution of a file

Table 1.8.2

- The File Access Permissions for a directory is to be interpreted in the following manner :

a. Without read permission, the user is not allowed to :
- List contents of directory
- Remove the directory using *rmdir*.

b. Without *write* permission, the user is not allowed to :
- Copy files to the directory
- Remove files from the directory
- Rename files in the directory
- Make a subdirectory
- Remove a subdirectory from the directory
- Move files to and from the directory.

c. Without *execute* permission, the user is not allowed to :
- Display the contents of a directory file from within the directory
- Change to the directory
- Display a file in the directory
- Copy a file to or from the directory.

1. Wildcard characters
- The command is executed on the files with names that match the pattern.
- Table 1.8.3 lists the wildcards with a description.

Character	Purpose
*	Matches none or one character or a string of more than one character
?	Matches exactly one character
[ ]	Matches exactly one of a specified set of characters

Table 1.8.3

2. *chmod* command
- Access permissions associated with a file or directory can be changed using the *chmod* command. Permissions associated with a file can be changed only by the owner of the file.
- Consider a file **prime.c** with the following permissions :

```
-r w x r w x r w x   1   rakshita   student   20   Jan   1   11:20   prime.c
```

- To change the permissions for the file, the File Owner has to specify :
  1. The type of user for whom permission is to be changed
  2. The type of permission which is to be changed
  3. Whether the permission is to be given or revoked
  4. The name of the file for which permissions are to be changed.
- UNIX allows you to change the FAP for a specific user-type. The command to do that is :

```
$ chmod u+r prime.c <Enter>
$
```

- Here,

'u' = indicates File Owner,

'+' = indicates that the permission is to be given,

'r' = indicates the read permission and prime.c is the filename.

- The File Access Permissions for the file prime.c will now appear as follows :

```
-r w x r w x r w x   1   rakshita   student   20   Jan   1   11:20   prime.c
```

- To remove the read permission from the Group Owner, the command is :

```
$ chmod g-r prime.c <Enter>
$
```

Here,

'g' = indicates Group Owner

'-' = indicates that the permission is to be removed.

- The permissions will now appear as follows :

```
-r w x - w x r w x    1    rakshita    student    20    Jan    1    11:20    prime.c
```

- To remove the execute permission from the Other Users, the command is :
  $ chmod o-x prime.c <Enter>
  $

Here, 'o' indicates Other Users

- The permissions will now appear as follows :

```
-r w x - w x r w-    1    rakshita    student    20    Jan    1    11:20    prime.c
```

- UNIX also allows the File Owner to change permissions for all categories of users for a file.
- To remove *write* permission from all users, the command is :
  $ chmod -w prime.c <Enter>
  $
- The FAP for the file prime.c will now appear as follows :

```
-r - x - - x r - -1    rakshita    student    20    Jan    1    11:20    prime.c
```

- To add back the *write* permission, the File Owner can give the following command :
  $ chmod +w prime.c <Enter>
  $

1.8.4 Process Utilities

1. ps : Process status

- *ps* command display some process attributes. The basic unit of execution in UNIX is called a process.
- If you run the Linux *ps* command by itself, it only shows very basic information about the processes you are currently running. For example, if you issue the basic command like this without any arguments :
  $ ps

Output from this command looks something like this :

```
PID    TTY       TIME   CMD
4343   ttys000   0:00.35  -bash
2617   ttys001   0:00.65  -bash
18201  ttys003   0:00.27  -bash
```

- The PID column shows the process-id, the second column shows the TTY (terminal) the process is attached to, the TIME column shows how much

CPU time the process has used, and the CMD column shows the command that is running.

- *ps* command **options** are :

Options	Meaning
-a	Displays all processes on a terminal, with the exception of group leaders.
-c	Displays scheduler data.
-d	Displays all processes with the exception of session leaders.
-e	Displays all processes.
-f	Displays a full listing.
-glist	Displays data for the *list* of group leader IDs.
-j	Displays the process group ID and session ID.
-l	Displays a long listing
-plist	Displays data for the *list* of process IDs.
-slist	Displays data for the *list* of session leader IDs.
-tlist	Displays data for the *list* of terminals.
-ulist	Displays data for the *list* of usernames.

Example :
$ ps –ef <Enter>

This leads to much more output :

UID	PID	PPID	C	STIME TTY	TIME CMD
root	1	0	0	Oct21 ?	00:00:01 init [3]
root	2	1	0	Oct21 ?	00:00:00 [migration/0]
root	3	1	0	Oct21 ?	00:00:00 [ksoftirqd/0]
root	4	1	0	Oct21 ?	00:00:00 [watchdog/0]
root	5	1	0	Oct21 ?	00:00:00 [migration/1]
root	6	1	0	Oct21 ?	00:00:00 [ksoftirqd/1]
root	7	1	0	Oct21 ?	00:00:00 [watchdog/1]
root	8	1	0	Oct21 ?	00:00:00 [events/0]
root	9	1	0	Oct21 ?	00:00:00 [events/1]
root	10	1	0	Oct21 ?	00:00:00 [khelper]

root	11	1	0	Oct21 ?	00:00:00 [kthread]
root	15	11	0	Oct21 ?	00:00:00 [kblockd/0]
root	16	11	0	Oct21 ?	00:00:00 [kblockd/1]
root	17	11	0	Oct21 ?	00:00:00 [kacpid]
root	91	11	0	Oct21 ?	00:00:00 [cqueue/0]
root	92	11	0	Oct21 ?	00:00:00 [cqueue/1]
root	95	11	0	Oct21 ?	00:00:00 [khubd]

2. nice

- Processes in the UNIX system are usually executed with equal priority. UNIX offers the nice command, which is used with the and operator to reduce the priority of the jobs.
- In the UNIX scheme of things, this priority is indicated by a number that can vary from a –20 to a +19 and is known as the nice number of the task. The programs with the highest priority have the lowest nice value, so a value of –20 makes a task important in the UNIX scheme.
- The nice command is used with its -n option, along with an argument in the range of –20 to 19, in order from highest to lowest priority (the lower the number, the higher the priority).
- Process started with a value of 18 cannot be lowered to 10 with nice –10, except by root.
- Nice is a built-in command in the C-shell. Nice values are system dependent.
- Syntax : *nice -number argument*
- For instance, to execute a program called a.out at a low priority, enter

  $ nice -n 19 a.out <Enter>
- If you have already started a process without the nice command, you can use the *renice* command to change the priority of the process.

  Syntax : *renice -number process_id*

1.8.5 Disk Utilities

1. The df Command

- The df command lists the current total disk usage for all file systems accessible by your workstation.

  Syntax : df -k

2 The du command

- To determine your current disk quota usage, use the du command.

  Syntax : *du [options] directory*
- By default, the *du* command by itself will give a summary of quota usage for the directory specified and all subdirectories below it.

  du /users/username
- For a summary of total disk usage, use the *-s* option.

  du -s /users/username

Networking commands

1. FTP : File Transfer Protocol

- Transfer files to or from a remote computer on the Internet. Some computers permit anonymous FTP that is accessible to anyone on the network, even if you don't have an account on that computer.
- To connect your local machine to the remote machine, type.

  ftp machinename

where machinename is the full machine name of the remote machine, e.g. vtubooks.com.

- If the name of the machine is unknown, you may type

  ftp machinennumber
- where *machinennumber* is the net address of the remote machine, e.g. 129.13.45.123.
- In either case, this command is similar to logging onto the remote machine. If the remote machine has been reached successfully, FTP responds by asking for a loginname and password.
- When you enter your own loginname and password for the remote machine, it returns the prompt

  ftp>

and permits you access to your own home directory on the remote machine. You should be able to move around in your own directory and to copy files to and from your local machine using the FTP interface commands given below :

- Common FTP commands

Commands	Meaning
?	To request help or information about the FTP commands
ascii	To set the mode of file transfer to ASCII
binary	To set the mode of file transfer to binary
mput	To copy multiple files from the local machine to the remote machine

open	To open a connection with another computer
bye	To exit the FTP environment
cd	To change directory on the remote machine
close	To terminate a connection with another computer
delete	To delete (remove) a file in the current remote directory
get	To copy one file from the remote machine to the local machine
help	To request a list of all available FTP commands
lcd	To change directory on your local machine
ls	To list the names of the files in the current remote directory
mkdir	To make a new directory within the current remote directory
mget	To copy multiple files from the remote machine to the local machine
put	To copy one file from the local machine to the remote machine
pwd	To find out the pathname of the current directory on the remote machine
quit	To exit the FTP environment
rmdir	To remove (delete) a directory in the current remote directory

2. Telnet

- Telnet operates in a client/server environment, meaning that the remote machine is configured as a server, and consequently waits for the other machine to request a service from it.
- The remote machine is sending data to be displayed; the user feels like they are working directly on the remote machine.
- In UNIX, the service is provided by what is called a daemon, a small task that runs in the background.
- The command to initiate a Telnet session is usually :

telnet server_name

server_name represents the name or IP address of the remote machine that the user wants to connect to.

- You can also give its IP address, for example :

telnet 125.63.121.77

- Finally, you can also specify which port to use by putting the port number after the IP address or server name :

telnet 125.63.121.77 80

Telnet commands

Command	Description
?	show help
close	Close Telnet session

display	Show connection settings onscreen
environ	For defining the operating system's environmental variables
logout	For logging out
mode	Switches between the transfer modes ASCII and BINARY
open	Opens another connection from the current one
quit	Leaves the Telnet application
set	Changes the connection settings

1.8.6 Other Commands

(i) Grep :

- The name graph as its origin in the phrase "Get Regular Expression and Print". Grep is a full-blown regular-expression matcher.
- Grep is the original command and it uses basic regular expressions to select the lines to process.
- The grep command allows you to search one file or multiple files for lines that contain a pattern. It displays the name of the file that contains the matched line. Exit status is 0 if matches were found, 1 if no matches were found and 2 if errors occurred.
- The syntax for the grep command is :

grep [options] pattern [files]

Options :

-b : Display the block number at the beginning of each line.

-c : Display the number of matched lines.

-h : Display the matched lines, but do not display the filenames.

-I : Ignore case sensitivity.

-l : Display the filenames, but do not display the matched lines.

-n : Display the matched lines and their line numbers.

-s : Silent mode.

-v : Display all lines that do NOT match.

-w : Match whole word.

(II) Cut :

- Cut command is used to select sections of text from each line of files. You can use the cut command to select fields or columns from a line by specifying a delimiter

or you can select a portion of text by specifying the range or characters. Basically the cut command slices a line and extracts the text

- Writes out selected bytes, characters, or fields from each line of a file.
- Syntax :

  cut [options] [list] [character] [file]

Options :

-c : List Specifies character positions. For example, if you specify -c 1-72, the cut command writes out the first 72 characters in each line of the file.

-f : List Specifies a list of fields assumed to be separated in the file by a delimiter character, which is by default the tab character.

(iii) Finger

- The finger displays information about the system users.
- Syntax :

  finger [-l] [-m] [-p] [-s] [username]

Options :

-l : Force long output format.

-m : Match arguments only on user name (not first or last name).

-p : Suppress printing of the .plan file in a long format printout.

-s : Force short output format.

(iv) suid

- SUID (Set User-ID) is one of the most beautiful concepts in UNIX. The common definition given for SUID is, it is an advanced file permission which allows an user to execute a script as if the owner of the script is executing it, and the famous example used for SUID is the passwd command.
- A program that changes its UID is called a SUID program (set-UID); a program that changes its GID is called a SGID program (set-GID). A program can be both SUID and SGID at the same time.
- When a SUID program is run, its effective UID[22] becomes that of the owner of the file, rather than of the user who is running it.
- Any program can be SUID, SGID, or both SUID and SGID. For example, any user can become the superuser simply by running a SUID copy of csh that is owned by root.
- Under most versions of UNIX, you can create shell scripts that are SUID or SGID. That is, you can create a shell script and, by setting the shell script's owner to be root and setting its SUID bit, you can force the shell script to execute with superuser privileges.

(v) wc command

- It is often useful to know the number of lines, words and characters in a file. The wc utility accomplishes this task. By default, it prints out all three of these fields.
- Syntax :

  wc|options|filename
- The command is userful when combined with other commands. For example, to find the number of files in a directory, enter

  $ ls - 1|wc - 1

(vi) cpio

- The cpio command is one of standard UNIX backup utilities. It stands for "copy in/out". It is much less well known and rarely used UNIX utility in comparison with tar. But in certain areas it is more powerful then tar due to its ability to handle links and special files, append back volumes and span types, allowing you to create incremental backup sets and full systems backups without losing data integrity.
- The cpio preserves permissions, times and ownership of files and subdirectories.
- cpio works as a filter accepting standard input and writing to standard output. The input to cpio is the list of files. That means that results of ls or find command can be piped directly into cpio. You can specify a device or file to which cpio will send making regular cpio backups.
- For example to backup the contents of a directory you can use ls command. The device file can be a regular file, for example, /tmp/mybackup :

  $ ls | cpio-oacv > /tmp/mybackup

(vii) tar archive files (create & extract)

- On UNIX platform, tar command is the primary archiving utility. Understanding various tar command options will help you master the archive file manipulation.
- This is the basic command to create a tar archive.

  $ tar cvf archive_name.tar dirname

In the above command :

i. c : Create a new archive.

ii. v : Verbosely list files which are processed.

iii. f : Following is the archieve file name.

University Questions

1. *Explain the following shell commands with example.*
   i) Chmod ii) Grep iii) Cat iv) Sort. **SPPU : Aug.-17, Insem, Marks 4**
2. *Explain the significance of following shell commands*
   i) in ii) wc iii) umask iv) cut v) grep **SPPU : June-18, Insem, Marks 5**
3. *Explain the following shell commands with example :*
   i) echo ii) grep iii) touchi v) ls. **SPPU : Aug.-18, Insem, Marks 4**
4. *Write a shell program to check if a given string is a palindrome or not.*
   SPPU : Dec.-19, Insem, Marks 5

1.9 Multiple Choice Questions

Q.1 ——— acts as an intermediary between the user of a computer and the computer hardware.

[a] Kernel [b] Operating system
[c] System software [d] All ofthese

Q.2 ——— program manages the execution of user programs to prevent errors and improper use of the computer.

[a] Batch [b] Operating system
[c] Control [d] None on these

Q.3 Multiprogramming increases ——— utilization by organizing jobs so that the CPU always has one to execute.

[a] hard disk [b] memory
[c] CPU [d] pen drive

Q.4 Time sharing requires an interactive computer system, which provides ——— communication between the user and the system.

[a] Indirect [b] direct
[c] direct and indirect [d] message

Q.5 A trap is a software-generated ——— caused either by an error or by a specific request from a user program that an operating-system service be performed.

[a] interrupt [b] request
[c] system call [d] message

Q.6 The hardware allows privileged instructions to be executed only in ——— mode.

[a] user [b] protected
[c] kernel [d] system

Q.7 Time sharing is logical extension of ———.

[a] multitasking [b] multiprogramming
[c] real time [d] batch

Q.8 Time sharing operating system uses ——— scheduling and multiprogramming to provide each user with a small portion of a time shared computer.

[a] job [b] process
[c] hard disk [d] CPU

Q.9 Which is the layer of a computer system between the hardware and the user program ?

[a] Operating environment [b] Operating system
[c] System environment [d] None of these

Q.10 The operating system manages ———.

[a] memory [b] processor
[c] disk and I/O devices [d] all of these

Q.11 Unix operating system is an ———.

[a] time sharing operating system
[b] multi-user operating system
[c] multi-tasking operating system
[d] All of these

Q.12 Dual mode uses user mode and monitor mode for working of ———.

[a] operating system [b] kernel
[c] batch system [d] system call

Q.13 Two views of operating systems are ———.

a software view and hardware view

b inside view and outside view

c user view and system view

d program view and process view

Answer Keys for Multiple Choice Questions

Q.1	b	Q.2	c
Q.3	c	Q.4	b
Q.5	a	Q.6	c
Q.7	b	Q.8	d
Q.9	b	Q.10	d
Q.11	d	Q.12	a
Q.13	c		

❑❑❑

UNIT - II

2 Process Management

Syllabus

Process : Concept of a Process, Process States, Process Description, Process Control.
Threads : Processes and Threads, Concept of Multithreading, Types of Threads, Threadprogramming Using Pthreads.
Scheduling : Types of Scheduling, Scheduling Algorithms, First Come First Served, Shortest Job First, Priority, Round Robin.

Contents

2.1 Concept of a Process

SPPU : Dec.-17, June-18,19

- Process term is used in MULTICS system in the 1960. Process is an asynchronous activity.
- Process is an active entity that requires a set of resources, including a processor, program counter, registers to perform its function. Multiple processes may be associated with one program.
- Process means a program in execution. Process execution must progress in sequential order. It also called task. Task is a single instance of an executable program. Fig. 2.1.1 shows memory layout for a process.
- Each process has its own address space. Address space is divided into two regions :
  1. Text region 2. Data region 3. Stack region
- **Text region :** It stores the code that that the processor executes.
- **Data region :** It stores variables and dynamically allocated memory that the process uses during execution.
- **Stack region :** It stores instructions and local variables for active procedure calls.
- Process is a dynamic entity that executes a program on a particular set of data using resources allocated by the operating system.
- A process can run to completion only when all requested hardware and software resources have been allocated to the process.
- Program is a passive entity. Program becomes process when executable file loaded into memory. A process may be independent of other processes in the system.
- After booting operating system, it creates foreground processes and background processes.

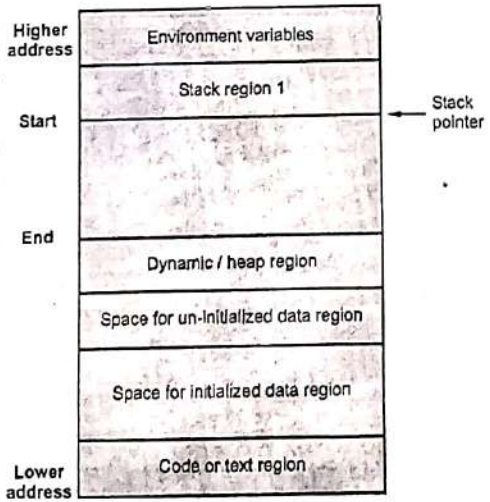


Fig. 2.1.1 Memory layout for a process

- Foreground processes interact with user and background processes is used by system. Background processes are related e-mail, web pages, news and printing.

- A process is used as a fundamental unit for resource allocation in operating system. A process normally has its own private memory area in which it runs.
- Text region contains executable instructions. It is be placed below the heap or stack.
- An initialized data region contains the static and global variables that are initialized by the user.
- Un-initialized data region contains all static and global variables that are initialized by kernel to zero.
- Heap is used for dynamic memory allocation and stack contains program counter.

2.1.1 Difference between Process and Program

Sr. No.	Process	Program
1.	Process is active entity.	Program is passive entity.
2.	Process is a sequence of instruction executions.	Program contains the instructions.
3.	Process exists in a limited span of time.	A program exists at single place and continues to exist.
4.	Process is a dynamic entity.	Program is a static entity.

2.1.2 Process Control Block

- Operating system keeps an internal data structure to describe each process it manages.
- When OS creates process, it creates this process descriptor. In some operating system, it calls Process Control Block (PCB).
- Fig. 2.1.2 shows process control block.
- Process control block will change according to the operating system. PCB is also called **task control block.**
- The PCB is identified by an integer Process ID (PID). When a process is running, its hardware state is inside the CPU.

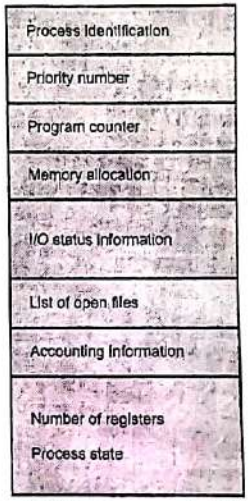


Fig. 2.1.2 Process control block

- When the OS stops running a process, it saves the register's values in the PCB.
- When a process is created by operating system, it allocates a PCB for it. OS initializes PCB and puts PCB on the correct queue. Following information is stored in process control block.

1. **Process identification** : Each process is uniquely identified by the user's identification and a pointer connecting it to its descriptor.
2. **Priority number** : Operating system allocates the priority number to each process. According to the priority number it allocates the resources.
3. **Program counter** : The PC indicates the address of the next instruction to be executed for this current process.
4. **Memory allocation** : It contains the value of the base registers, limit registers and the page tables depending on the memory system used by the operating system.
5. **I/O status information** : It maintains information about the open files, list of I/O devices allocated to the process etc.
6. **List of open files** : Process uses number of files for operation. Operating system keeps track of all opened file by this process.
7. **Process state** : Process may be in any one of the state : new, ready, running, and waiting, terminate.

- When process changes the state, the operating system must update information in the process's process control block. Process control block maintain other information which is not included in PCB block diagram. This information includes CPU scheduling, file management, input-output management information.
- Fig. 2.1.3 shows process table with PCB.
- When process is created, hardware registers and flags values are set by loader or linker.
- Operating system maintains pointers to each process's PCB in a per user process table or system wide process table. This information is used to access PCB quickly.

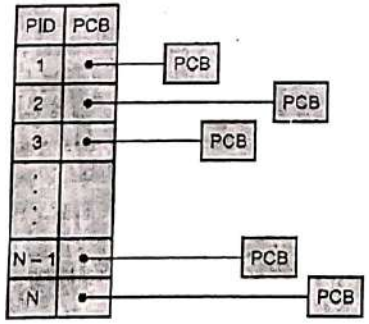


Fig. 2.1.3 Process table with PCB

2.1.3 Process States

- Each process has an execution state which indicates what process is currently doing. The process descriptor is the basic data structure used to represent the specific state for each process.

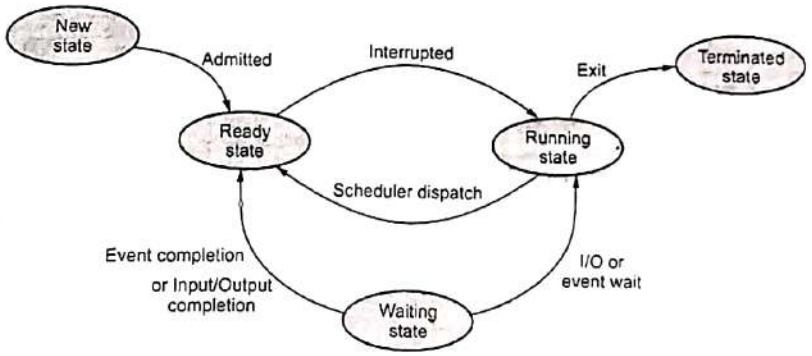


Fig. 2.1.4 Process state diagram

- Fig. 2.1.4 shows a process state diagram. A state diagram is composed of a set of states and transitions between states.
- State diagram is used by process manager to determine the type of service to provide to the process.
- The process states are as follows :
  1. New 2. Ready 3. Running 4. Waiting 5. Terminated
- **New** : Operating system creates new process by using fork( ) system call. These process are newly created process and resources are not allocated.
- **Ready** : The process is competing for the CPU. Process reaches to the head of the list (queue).
- **Running** : The process that is currently being executed. Operating system allocates all the hardware and software resources to the process for execution.
- **Waiting** : A process is waiting until some event occurs such as the completion of an input-output operation.
- **Terminated** : A process completes its operations and releases all resources.
- Operating system maintains a ready list of ready process and a blocked list of blocked processes. The ready list is maintained in priority order and blocked list is typically unordered.
- The act of assigning a processor to the first process on the ready list is called **dispatching** and is performed by a system entity called the **dispatcher.**
- When process is in new state, the program remains in the secondary storage. Here process itself is not in the main memory and space is not allocated.

- The process exists a system because of two reasons :
  1. Process is terminated when it completes its operation.
  2. Process aborts due to an unrecoverable error.
- To prevent any one process from continuously using the system, the operating system sets a hardware interrupting clock to allow a process to run for a specific time interval or time quantum.
- The process may request a resource when it is in the running state. In most of the operating system, if a running process requests an immediately available resources the process is allowed to continue in the running state.
- From the blocked state, the process can move to the ready state only by being allocated the requested resource.
- User only initiate process state transition is blocked and remaining all other state transitions is initiated by the operating system.

Sr. No.	State transition	Remarks
1.	Ready to running	Process is dispatched.
2.	Running to ready	Process time slice expires.
3.	Running to blocked	When a process blocks.
4.	Blocked to ready	When the event for which it has been waiting occurs.
5.	Ready to exit	This is possible when parent process may terminate child process at any time.
6.	Running to exit	When currently running process completes its operation then OS terminates the process.

2.1.4 Suspended Processes

- Each process to be executed must be loaded into main memory. In uniprocessor system, processor remains idle because of execution speed mismatch of processor and I/O devices. An I/O device is slower than processor.
- Main memory accommodates multiple processes and the processor select next process when one process is blocked. So even with multiprogramming operating system, a processor could be idle most of the time. Main memory size is increased for accommodating more processes but cost will increase.
- **Suspended state :** when all of the processes in main memory are in the blocked state, the operating system can suspend one process by putting it in the suspend

state and transferring it to disk. The space that is freed in main memory can then be used to bring in another process.

- Characteristics of a suspended process :
  1. The process cannot execute immediately.
  2. The process may or may not be waiting for an event.
  3. The process was placed in a suspended state by a parent process, or itself or by operating system.
  4. The process may not be removed from suspended state until its turn comes.

University Questions

1. *How PCB helps in process state management? Explain the structure of PCB.* **SPPU : Dec.-17, Marks 5**
2. *Draw and explain process state diagram.* **SPPU : June-18, Marks 5**
3. *Draw process state transition diagram and explain each state in it.* **SPPU : June-19, Marks 5**

2.2 Process Description

- Operating systems are considered as a manager of the various hardware resources. Goal of the OS is the management of hardware resources and control of processes accessing the resources. Fig. 2.2.1 shows the processes with resources.

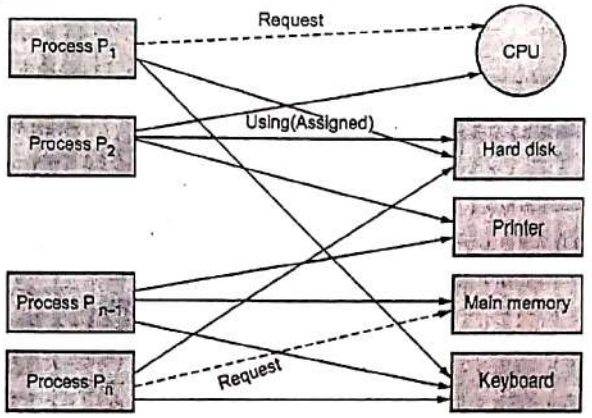


Fig. 2.2.1 Process and resources

- OS as a service provider, naturally also imposes control on processes that consume the services. That is both processes and resources are under the operating system's control. Control table is used by OS.

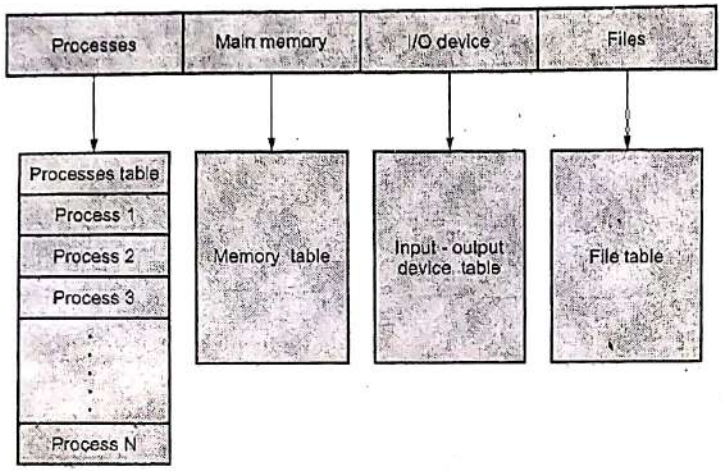


Fig. 2.2.2 OS control tables

Control structures

- The operating system constructs and maintains tables of information about each entity that it is managing. Fig. 2.2.2 shows that four different types of tables are maintained by the operating system.

1. **Process table :** Operating system must maintain process tables to manage processes. Process image contains user data, user program, stack and process control block.
2. **Memory tables :** Operating system uses memory table for keeping track of both main memory and secondary memory. Main memory is used by operating system and processes. For operating system, memory is reserved and it cannot allocate to the user process. Following information is stored in the memory table.
   a. Allocation of main memory to user processes.
   b. Allocation of secondary memory to user processes.
   c. Protection attributes
3. **Input-Output tables :** This helps operating system for managing I/O devices. It maintains information like :
   a. Device availability
   b. Status of I/O operation
   c. Location of evice in main memory
4. **File tables :** It maintains information about the existence of files and their location on secondary memory with attributes.

2.3 Process Control

- Following operations are performed on the process :
  1. Process creation 2. Process termination
- A mechanism for process creation and termination via system calls. Programmers usually access these system calls via application interface (API) / library.

2.3.1 Modes of Execution

- Most of the processor support two modes of execution. Certain instructions can only be executed in the more privileged more.
- Certain regions of memory can only be accessed in the more privileged mode.
- The less privileged mode is referred to as the user mode because user programs typically would execute in this mode.
- More privileged mode is referred as the system mode or kernel mode.
- Function of Operating system Kernel :
  1. **Process Management**
     - Process creation and termination
     - Process switching
     - Process scheduling and dispatching
     - Process synchronization
     - Support for interprocess communication
     - Management of process control blocks
  2. **Memory Management**
     - Swapping
     - Allocation of address space to processes
     - Page and segment management
  3. **I/O Management**
     - Buffer Management
     - Allocation of I/O channels and device to processes

2.3.2 Process Creation

- Operating system creates the process in following situations :
  1. Starting of new batch job. 2. User request for creating new process.
  3. To provide new services by OS. 4. System call from currently running process.

- Operating system creates a new process with the specified or default attributes and identifier. A process may create several new sub-process.
- Parent process is creating process and the new processes are called the children of the process. When operating system creates process, it builds the data structure for managing process and allocates address space in primary main memory.
- Operating system creates foreground and background process. Process is identified by unique process identifier (PID) in UNIX and windows operating system. PID value is an integer number.
- All processes in UNIX are created using the fork( ) system call. The forking process is called the parent process. The new process is called the child process.
- Both the parent and child process have their own and private memory. Open files are shared between parent and child.
- If the parent changes the value of its variable, the modification will only affect the variable in the parent process's address space. Other address spaces created by fork( ) calls will not be affected even though they have identical variable names.
- When a process is created, OS assigns some attributes. These are priority, privilege level, requirement of memory, access right, memory protection, PID etc. To perform operation, process needs software and hardware resources. It includes CPU time, files, memory, I/O device.
- Relation between parent process and child process is as follows :
  1. Parent process continues to execute concurrently with its child process.
  2. Parent process waits until some or all of its children have terminated.

```
void main( )
    {
        printf("Operating System\n");
        fork( );
            printf("Technical Publications\n");
    return 0;
}
```

- In above program Operating System is printed only once and Technical Publications is printed two times.

2.3.3 Process Termination

- When process finishes its normal execution then that process is terminate. Operating system delete that process using exit ( ) system call. After deleting process, memory space becomes free.

- OS passes the child's exit status to the parent process and then discards the process. At the same time, it de-allocate all the resources hold by this process.
- Following are the various reasons for process termination :
  1. Normal completion of operation
  2. Memory is not available
  3. Time slice expired
  4. Parent termination
  5. Failure of I/O
  6. Request from parent process
  7. Misuse of access rights

2.4 Threads

SPPU : Aug.-17, June-19

- Thread is a dispatchable unit of work. It consists of thread ID, program counter, stack and register set. Thread is also called a Light Weight Process (LWP). Because they take fewer resources then a process. A thread is easy to create and destroy.
- It shares many attributes of a process. Threads are scheduled on a processor. Each thread can execute a set of instructions independent of other threads and processes. Fig. 2.4.1 shows a thread.
- Every program has at least one thread. Programs without multithreading executes sequentially. That is, after executing one instruction the next instruction in sequence is executed.
- Thread is a basic processing unit to which an operating system allocates processor time and more than one thread can be executing code inside a process.

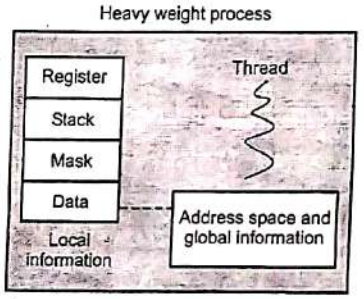


Fig. 2.4.1 Thread

- When user double click on an icon by using mouse, the operating system creates a process and that process has one thread that runs the icon's code.
- Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control.
- Normally, an operating system will have a separate thread for each different activity. The OS will have a separate thread for each process and that thread will perform OS activities on behalf of the process. In this condition, each user process is backed by a kernel thread. When process issues a system call to read a file, the process's thread will take over, find out which disk accesses to generate, and issue the low level instructions required to start the transfer. It then suspends until the disk finishes reading in the data.

- When process starts up a remote TCP connection, its thread handles the low-level details of sending out network packets.
- In Remote Procedure Call (RPC), servers are multithreaded. Server receives the messages using a separate thread.
- Different operating system uses threads in different ways.
  1. Free memory is managed by kernel thread in LINUX OS.
  2. Solaries uses a kernel thread for interrupt handling.

2.4.1 Thread Advantages

1. Context switching time is minimized.
2. Thread support for efficient communication.
3. Resource sharing is possible using threads.
4. A thread provides concurrency within a process.
5. It is more economical to create and context switch threads.

2.4.2 Difference between Thread and Process

Sr. No.	Thread	Process
1.	Thread is also called lightweight process.	Process is also called heavyweight process.
2.	Operating system is not required for thread switching.	Operating system interface is required for process switching.
3.	One thread can read, write or even completely clean another threads stack.	Each process operates independently of the other process.
4.	All threads can share same set of open files and child processes.	In multiple processing, each process executes the same code but has its own memory and file resources.
5.	If one thread is blocked and waiting then second thread in the same task can run.	If one server process is blocked then other server process cannot execute until the first process is unblocked.
6.	Uses fewer resources.	Uses more resources.

2.4.3 Thread Lifecycle

- Fig. 2.4.2 shows a thread lifecycle. A thread has one of the following states.

1. **New :** Thread is just created.

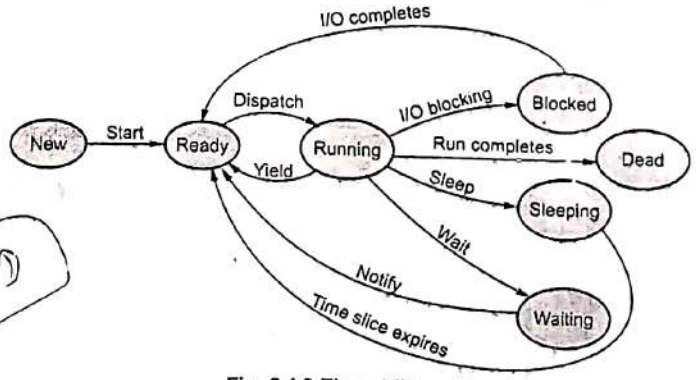


Fig. 2.4.2 Thread lifecycle

2. **Ready :** Thread's start( ) method invoked and now it is executing. OS put thread into Ready queue.
3. **Running :** Highest priority ready thread enters the running state. Thread is assigned a processor and now is running.
4. **Blocked :** This is the state when a thread is waiting for a lock to access an object.
5. **Waiting :** Here thread is waiting indefinitely for another thread to perform an action.
6. **Sleeping :** Thread sleep for a specified time of period. When sleeping time expires, it enters to ready state. CPU is not used by sleeping thread.
7. **Dead :** When thread completes task or operation.

University Questions

1. *Differentiate between process and thread.* **SPPU : Aug.-17, Marks 2**
2. *Define thread ? List and explain different thread scheduling approaches.* **SPPU : June-19, Marks 5**

2.5 Types of Threads

SPPU : Aug.-18, Dec.-18, 19

- Threads are of two types :
  1. User level thread 2. Kernel level thread
- All modern operating system support threading model. Implementation of thread will change according to the operating system.

2.5.1 User Level Thread

- User level thread uses user space for thread scheduling. These threads are **transparent** to the operating system. User level threads are created by runtime libraries that cannot execute privileged instructions.

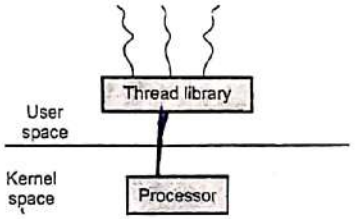


Fig. 2.5.1 User level thread

- User-level threads have low overhead but it can achieve high performance in computation. User-level threads are managed entirely by the run-time system.
- User-level threads are small and faster. A thread is simply represented by a PC, registers, stack and small thread control block. Fig. 2.5.1 shows user level thread.
- The code for creation and destroying thread, message passing and data transfer, thread scheduling is included into thread library. Kernel is unaware of user level thread.
- User level threads do not invoke the Kernel for scheduling decision.
- User level thread are also called **many to one mapping** thread because the operating system maps all threads in a multithreaded process to a single execution context. The operating system considers as each multithreaded processes as a single execution unit.
- Example : POSIX Pthreads and Mach C-threads.

Advantages :

1. Kernel mode privilege does not require for thread switching.
2. These threads are fast to create and manage.
3. User level thread works even if the OS does not support threads.
4. User level threads are more portable.
5. Threading library controls flow of thread.

Disadvantages :

1. If thread blocks, the Kernel may block the all threads.
2. Not suitable for multiprocessor system.
3. User level threads also do not support system wide scheduling priority.

2.5.2 Kernel Level Thread

- In Kernel level thread, thread management is done by Kernel. Operating systems support the Kernel level thread.

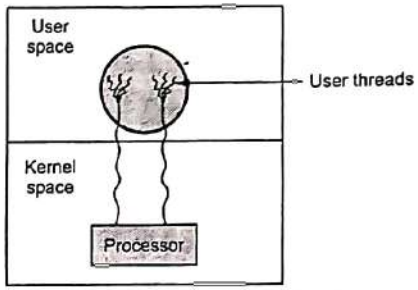


Fig. 2.5.2 Kernel level thread

- Since Kernel managing threads, Kernel can schedule another thread if a given thread blocks rather than blocking the entire processes. Fig. 2.5.2 shows Kernel level thread.
- Kernel level thread support one to one thread mapping. This mapping requires each user thread with kernel thread. Operating system performs this mapping.
- Threads are constructed and controlled by system calls. The system knows the state of each thread.
- Thread management code is not included in the application code. It is only API to the Kernel thread. Windows operating system uses this facility.
- Any application can be programmed to be multithreaded. All of the threads within an application are supported within a single process.
- Kernel performs scheduling on a thread basis. The Kernel support for scheduling and management, thread creation only in Kernel space.
- Kernel level threads are slower than user level threads.
- Example : Windows 95/98/NT, Sun Solaris and Digital UNIX.

Advantages :

1. Each thread can be treated separately.
2. A thread blocking in the Kernel does not block all other threads in the same process.
3. Kernel routines itself as multithreaded.

Disadvantages :

1. Slower than the user level thread.
2. There will be overhead and increased in Kernel complexity.

2.5.3 Difference between User Level and Kernel Level Thread

Sr. No.	User level threads	Kernel level threads
1.	User level threads are faster to create and manage.	Kernel level threads are slower to create and manage.
2.	Implemented by a thread library at the user level.	Operating system support directly to Kernel threads.
3.	User level thread can run on any operating system.	Kernel level threads are specific to the operating system.
4.	Support provided at the user level called user level thread.	Support may be provided by Kernel is called Kernel level threads.
5.	Multithread application cannot take advantage of multiprocessing.	Kernel routines themselves can be multithreaded.
6.	Example : POSIX Pthreads and Mach C-threads	Example : Windows 95/98/NT, Sun Solaris and Digital UNIX
7.	User level threads are also called many to one mapping thread.	Kernel level thread support one to one thread mapping.

University Question

1. *Differentiate between user level and kernel level threads.*

SPPU : Aug.-18, Marks 2, Dec.-18, 19, Marks 5

2.6 Concept of Multithreading

- Operating system uses user level thread and kernel level thread. User level threads are managed without kernel support. Operating system support and manage the kernel level threads.
- Modern operating system support kernel level threads. Kernel performs multiple simultaneous tasks in operating system. In most of the application, user threads are mapped with kernel level threads.
- Different methods of mapping is used in operating system are as follows :
  1. One to one 2. Many to one 3. Many to many

1. One to one

- In this model, one user level thread maps with one kernel level thread. If there are three user threads then system creates three separate kernel thread. This model provides more concurrency because of separate thread. Fig. 2.12.1 shows one to one mapping.

- In this method, the operating system allocates data structure that represents kernel threads. Here multiple threads are run in parallel on multiprocessor system.
- Number of threads in the system increases, the amount of memory required is also increases.
- Windows 95/XP and Linux operating system uses this one to one thread mapping.
- Only overhead in this method is creation of kernel level thread for user level thread. Because of this overhead, system performance is slowdown.

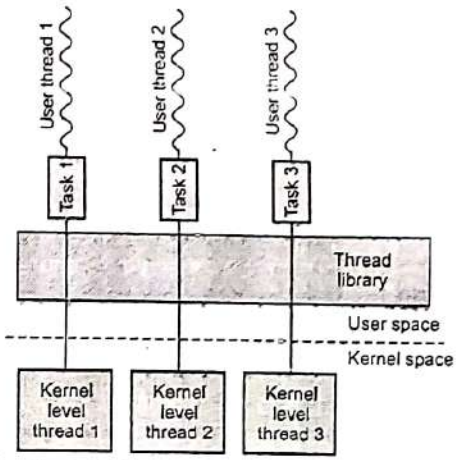


Fig. 2.6.1 One to one multithreading model

2. Many to one

- Many to one mapping means many user thread maps with one kernel thread. Fig. 2.6.2 shows many to one mapping.
- Operating system blocks the entire multi-threaded process when a single thread blocks because the entire multi-threaded process is a single thread of control. So when operating system receive any a blocking I/O request, it blocks the entire process.

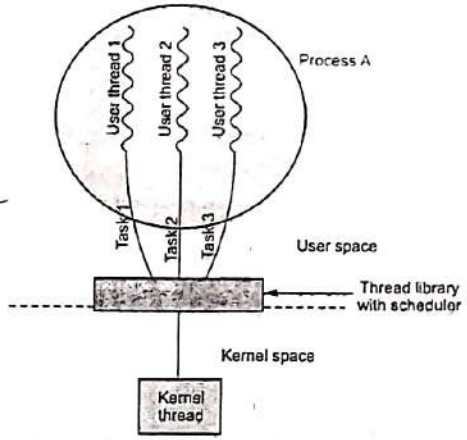


Fig. 2.6.2 Many to one multithreading model

- Thread library handle thread management in user space. The many-to-one model does not allow individual processes to be split across multiple CPUs.
- This type of relationship provides an effective context-switching environment, easily implementable even on simple kernels with no thread support.
- Green threads for Solaris and GNU portable threads implement the many-to-one model in the past, but few systems continue to do so today.

- **Advantage** : System performance is improved by customizing the thread library scheduling algorithm.

3. Many to many

- In the many to many mapping, many user-level threads maps with equal number of kernel threads. Fig. 2.6.3 shows many to many thread mapping.

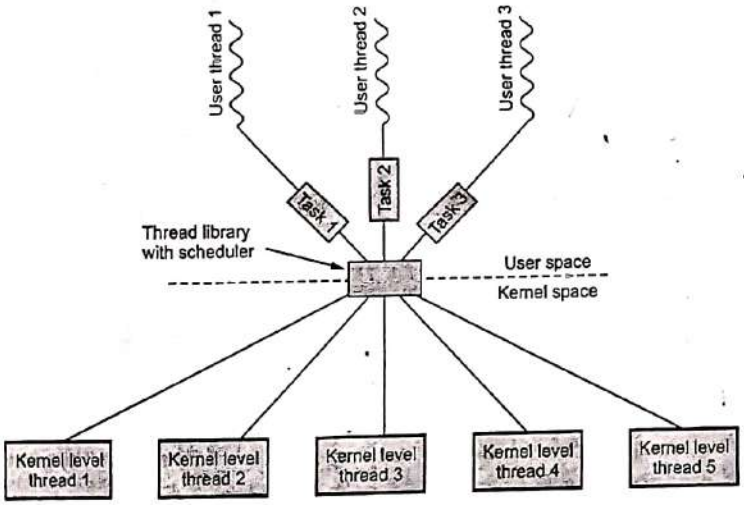


Fig. 2.6.3 Many to many multithreading model

- Many to many mapping is also called M-to-N thread mapping. Thread pooling is used to implement this method.
- Users can create required number of thread and there is no limitation for user. Again here blocking Kernel system calls do not block the entire process.
- This model support for splitting processes across multiple processors.
- **Limitations** : Operating system design becomes complicated.

Example 2.6.1 *Provide two programming examples in which multithreading does not provide better performance than a single-threaded solution.*

Solution :

a. A Web server that services each request in a separate thread.

b. A parallelized application such as matrix multiplication where different parts of the matrix may be worked on in parallel.

c. An interactive GUI program such as a debugger where a thread is used to monitor user input, another thread represents the running application, and a third thread monitors performance.

2.7 Thread Programming using Pthreads

SPPU : June-18

- The subroutines which comprise the Pthreads API can be informally grouped into four major groups :

1. **Thread management :** Routines that work directly on threads - creating, detaching, joining, etc. They also include functions to set/query thread attributes (joinable, scheduling etc.)
2. **Mutexes :** Routines that deal with synchronization, called a "mutex", which is an abbreviation for "mutual exclusion". Mutex functions provide for creating, destroying, locking and unlocking mutexes.
3. **Condition variables :** Routines that address communications between threads that share a mutex. Based upon programmer specified conditions. This group includes functions to create, destroy, wait and signal based upon specified variable values. Functions to set/query condition variable attributes are also included.
4. **Synchronization :** Routines that manage read/write locks and barriers.

Basic thread functions include creation and termination of threads. There are five such thread function.

2.7.1 pthread_create Function

- A program is started by exec, a single thread is created known as initial thread or main thread. Additional threads are created by pthread_create

```
#include<pthread.h>
    int pthread_create (pthread_t *tid, const pthread_attr_t *attr,
                        void *(*func) (void*), void* arg);
```

- Each thread is identified by a thread ID, of which data type is pthread_t. Each thread has numerous attributes, its priority, its initial stack size. The thread starts by calling this function and then terminates explicitely or implicitly. The address of function is specified as **func** argument which is called with a single pointer argument, **arg**.

2.7.2 pthread_join Function

- A thread can be terminated by calling **pthread_join.** In Unix process **pthread_create** is similar to **fork** and **pthread_join** is similar to **wait pid.**

```
#include<pthread.h>
int pthread_ join (pthread_t tid, void **status);
```

- The tid is specified as the wait. But there is no way to wait for any of the threads.

2.7.3 pthread_self Function

- Each thread has an ID which identifies it within a given process. This thread ID is returned by **pthread_create** and is used by **pthread_join.**
- A thread fetches this value for itself by using **ptherad_self.**

```
#include<pthread.h>
    pthread_t pthread_self (void);
```

2.7.4 pthread_detach Function

- A thread is joinable or detachable. When joinable thread terminates, its thread ID and exit status are retained until another thread calls pthread_join.
- A detached thread is like a daemon process i.e. when it is terminated all its resources are released. The pthread_detach function changes the specified thread so that it is detached.

```
#include<pthread.h>
    int pthread_detach (pthread_t tid);
```

2.7.5 pthread_exit Function

- A way for a thread to terminate is to call **pthread_exit.**

```
#include<pthread.h>
    void pthread_exit (void *status);
```

- But if thread is not detached, its thread ID and exit status are retained for a later pthread_join by any other thread in calling processes.

2.7.6 pthread_sigmask, pthread_kill

The prototype of the *pthread_sigmask* and *pthread_kill* functions are :

```
#include <signal.h>
#include <pthread.h>
int pthread_sigmask (int mode, sigset_t *sigsetp, sigset_t *oldstp);
int pthread_kill (pthread_t tid, int signum);
```

- The *pthread_sigmask* function sets the signal mask of a calling thread. The *sigsetp* argument contains one or more of the signal numbers to be applied to the calling thread.
- The mode argument specifies how the signal specified in the *sigsetp* argument is to be used. Possible value of mode argument is declared in <signal.h> header and their meanings are :

Mode value	Meaning
SIG_BLOCK	Adds signals contained in the *sigsetp* argument to the thread signal mask.
SIG_UNBLOCK	Removes signals contained in the *sigsetp* argument from the thread signal mask.
SIG_SETMASK	Replace the thread signal mask with the signal specified in the *sigsetp* argument.

- The *pthread_kill* function sends a signal, as specified in the *signum* argument, to a thread whose ID is given by the *tid* argument. The sending and receiving threads must be in the same process.
- If the default action for a signal is to terminate the process, then sending the signal to a thread will still kill the entire process.

2.7.7 sched_yield

- The function prototype of the *sched_yield* API is :

```
#include <pthread.h>
int sched_yield (void);
```

- The *sched_yield* function is called by a thread to yield its execution to other threads with the same priority. This function returns 0 on success and –1 when fails.

2.7.8 Thread Attributes

- Thread attributes provide a mechanism for fine-tuning the behavior of individual threads. You may create and customize a thread attribute object to specify other values for the attributes.
- We can use the *pthread_attr_t* structure to modify the default attributes, and associate these attributes with threads that we create. We use the *pthread_attr_init* function to initialize the *pthread_attr_t* structure. After calling *pthread_attr_init*, the *pthread_attr_t* structure contains the default values for all the thread attributes supported by the implementation.
- The function for changing the attributes are as follows :

```
#include <pthread.h>
int pthread_attr_init(pthread_attr_t *attr);
int pthread_attr_destroy(pthread_attr_t *attr);
```

- To deinitialize a *pthread_attr_t* structure, we call *pthread_attr_destroy*. If an implementation of *pthread_attr_init* allocated any dynamic memory for the attribute object, *pthread_attr_destroy* will free that memory. In addition, *pthread_attr_destroy* will initialize the attribute object with invalid values, so if it is used by mistake, *pthread_create* will return an error.

- Steps for specify customized thread attributes :
  1. Create a *pthread_attr_t* object. The easiest way is simply to declare an automatic variable of this type.
  2. Call *pthread_attr_init*, passing a pointer to this object. This initializes the attributes to their default values.
  3. Modify the attribute object to contain the desired attribute values.
  4. Pass a pointer to the attribute object when calling *pthread_create*.
  5. Call *pthread_attr_destroy* to release the attribute object. The *pthread_attr_t* variable itself is not deallocated; it may be reinitialized with *pthread_attr_init*.
- A single thread attribute object may be used to start several threads. It is not necessary to keep the thread attribute object around after the threads have been created.
- For Linux application programming tasks, only one thread attribute is typically of interest. This attribute is the threads detach state.
- A thread may be created as a joinable thread or as a detached thread. A joinable thread, like a process, is not automatically cleaned up by Linux when it terminates.
- Instead, the thread's exit state hangs around in the system until another thread calls *pthread_join* to obtain its return value. Only then are its resources released.
- A detached thread, in contrast, is cleaned up automatically when it terminates. Because a detached thread is immediately cleaned up, another thread may not synchronize on its completion by using *pthread_join* or obtain its return value.
- To set the detach state in a thread attribute object, use *pthread_attr_set* detach state. The first argument is a pointer to the thread attribute object, and the second is the desired detach state. Because the joinable state is the default, it is necessary to call this only to create detached threads; pass *PTHREAD_CREATE_DETACHED* as the second argument.

University Question

1. *Explain the following functions with reference to 'C*
   i) pthread_create()
   ii) pthread_join()

SPPU : June-18, Marks 5

2.8 Scheduling : Types of Scheduling

SPPU : May-18, Dec.-18,19, June-19

- An OS must allocate resources amongst competing processes.
- The resource is allocated by means of scheduling - determines which processes will wait and which will progress.
- The resource provided by a processor is execution time.
- Aim is to assign processes to be executed by the processor in a way that meets system objectives, such as response time, throughput, and processor efficiency.

Scheduling Objectives :

- The scheduling function should
  1. Share time fairly among processes
  2. Prevent starvation of a process
  3. Use the processor efficiently
  4. Have low overhead
  5. Prioritise processes when necessary (e.g. real time deadlines)

2.8.1 Process Scheduling

- CPU utilization is maximizing by using multiprogramming concept. Processor is not idle, it is executing a process. Processor scheduler selects one process for execution from the ready queue.

Scheduling Queue

- Scheduling queue is queue of processes or input-output devices. When the user process enters into the system, they put into the job queue. Job queue consists of all processes of the system.
- Operating system maintains different types of queues for different purposes. The queue are ready queue, device queue etc. Fig. 2.8.1 shows ready queue and device queue.

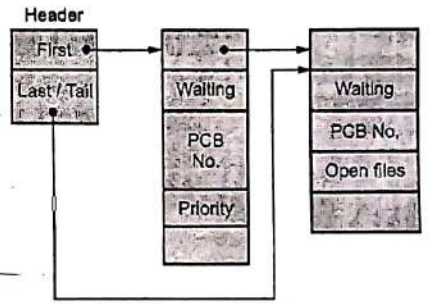


Fig. 2.8.1 Ready and device queue

- **Ready queue :** The processes which are ready and waiting to execute are kept in the ready queue. Ready queue is stored in main memory. Linked list is used for representing ready queue. Pointer field of PCB is used for this.

- **Device queue :** Device queue is a queue for which multiple processes are waiting for a particular I/O device. Each device has maintained its own device queue.
- Process scheduling is represented by queueing diagram. Newly created process is kept into the ready queue and waits for processor. When CPU select the process for executing, following things happens :

1. Process may require I/O device to perform operation. So it is put into the I/O queue.
2. Process may create child process and wait for child process termination.
3. Because of interrupt, process preempted from CPU and put into the ready queue.

2.8.2 Types of Scheduling

- Schedulers are used to handles process scheduling. It is one type of system software and selects the jobs from the system and decide which process to run. Schedulers are of three types -
  1. Long term scheduler 2. Short term scheduler 3. Medium term scheduler
- Fig. 2.8.2 shows queueing diagram for process scheduling.

Long term scheduler

- Long term scheduler is also called job scheduler. It determines which process are admitted to the system for processing. Processes are selected from the queue and loads into the main memory for execution.
- Long term scheduling controls the degree of multiprogramming in multitasking systems. It provides a balanced mix of jobs, such as I/O bound and CPU bound.
- Long term scheduling is used in real time operating system. Time sharing operating system has no long term scheduler.

Medium term scheduler

- Medium term scheduler is part of swapping function. Sometimes it removes the process from memory. It also reduces the degree of multiprogramming.
- If process makes an I/O request and it is in memory then operating system takes this process into suspended states. Once the process becomes suspended, it cannot make any progress towards completion.
- In this situation, the process is removed from memory and makes free space for other process.
- The suspended process is stored in the secondary storage device i.e. hard disk. This process is called swapping.

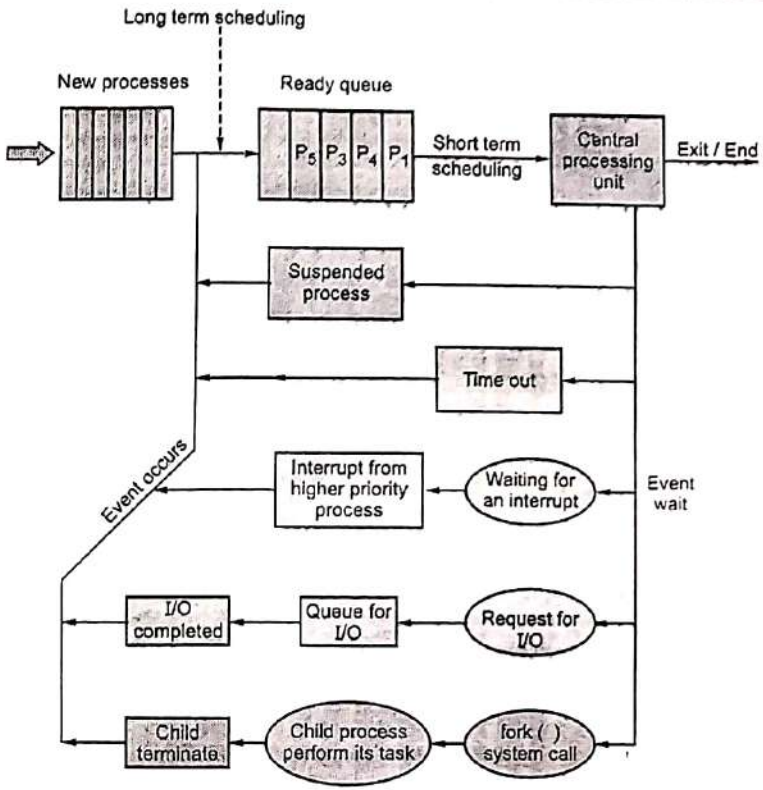


Fig. 2.8.2 Process scheduling queueing diagram

Short term scheduler

- Short term scheduler is also called CPU scheduler. It selects the process from queue which are ready to execute and allocate the CPU for execution.
- Short term scheduler is faster that long term scheduler. This scheduler makes scheduling decisions much more frequently than the long-term or mid-term schedulers.
- A scheduling decision will at a minimum have to be made after every time slice, and these are very short.
- It is also known as dispatcher.

2.8.3 Difference between Long Term, Short Term and Medium Term Scheduling

Sr. No.	Long term	Short term	Medium term
1.	It is job scheduler.	It is CPU scheduler.	It is swapping.
2.	Speed is less than short term scheduler.	Speed is very fast.	Speed is in between both.
3.	It controls the degree of multiprogramming.	Less control over degree of multiprogramming.	Reduce the degree of multiprogramming.
4.	Absent or minimal in time sharing system.	Minimal in time sharing system.	Time sharing system use medium term scheduler.
5.	It select processes from pool and load them into memory for execution.	It select from among the processes that are ready to execute.	Process can be reintroduced into memory and its execution can be continued.
6.	Process state is (New to Ready).	Process state is (Ready to Running)	-
7.	Select a good process, mix of I/O bound and CPU bound.	Select a new process for a CPU quite frequently.	-

2.8.4 Context Switch

- A context switch is the switching of the CPU from one process or thread to another. A context is the contents of a CPU's registers and program counter at any point in time.
- Switching the CPU to another process requires performing a state save of the current process and a state restore of a different process. This task is known as a context switch.
- A context switch can mean a register context switch, a task context switch, a thread context switch or a process context switch.
- A register is a small amount of very fast memory inside of a CPU that is used to speed the execution of computer programs by providing quick access to commonly used values.
- A program counter is a specialized register that indicates the position of the CPU in its instruction sequence and which holds either the address of the instruction being executed or the address of the next instruction to be executed, depending on the specific system.

- Context switching can be described in more detail as the Kernel performing the following activities with regard to processes (including threads) on the CPU :
  1. Suspending the progression of one process and storing the CPU's state (i.e., the context) for that process somewhere in memory.
  2. Retrieving the context of the next process from memory and restoring it in the CPU's registers and
  3. Returning to the location indicated by the program counter in order to resume the process.
- Context switches can occur only in Kernel mode (system mode). Kernel mode is a privileged mode of the CPU in which only the Kernel runs and which provides access to all memory locations and all other system resources.
- Other programs, including applications, initially operate in user mode, but they can run portions of the Kernel code via system calls. Software context switching can be used on all CPUs and can be used to save and reload only the state that needs to be changed.
- To use the hardware context switch you need to tell the CPU where to save the existing CPU state and where to load the new CPU state from. The CPU state is always stored in a special data structure called a TSS (Task State Segment).
- Context switch times are highly dependent on hardware support. Context switching represents a substantial cost to the system in terms of CPU time and it can be the most costly operation on an operating system.
- There are three situations where a context switch needs to occur. They are multitasking, interrupt handling, user and Kernel mode switching.

University Questions

1. *Explain different types of schedulers in operating system.* **SPPU : May-18, Marks 5**
2. *Specify the role of schedulers in operating system.* **SPPU : Dec.-18, Marks 5**
3. *Specify the role of long term, short term and medium term scheduler in operating system with diagram.* **SPPU : June-19, Marks 5**
4. *Explain the concept of context switching with the help of a neat diagram.* **SPPU : Dec.-19, Marks 5**

2.9 CPU Scheduling

SPPU : Aug.-17

- In a multiprogramming environment, usually more programs to be executed than could possibly be run time at one time. In CPU scheduling it switches from one

process to another process. CPU resource management is commonly known as scheduling.

- Objective of the multiprogramming is to increases the CPU utilization. CPU scheduling is one kind of fundamental operating system functions.
- User program contains combination of CPU burst cycle and I/O burst cycles. Process starts with CPU burst cycle then I/O burst cycle again CPU burst cycle again I/O burst cycle. In this way the process completes its executions. Process will terminates after final CPU burst cycle completions.
  Program execution sequence : CPU burst cycle → I/O burst cycle → CPU burst cycle → I/O burst cycle → CPU burst cycle → Program terminates.
- A CPU bound process tends to use the processor time that the system allocates for it. An I/O bound process tends to use the processor only briefly before generating an I/O request. The CPU bound processes spend most of their time using the processor.

CPU Scheduler

- CPU scheduler is also called as short scheduler. System programmers use parameters like program size, resources required for program and any other special devices are needed to determine scheduling policies.
- Scheduling mechanism is the part of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.
- Scheduler is responsible for multiplexing processes on the CPU.

2.9.1 Preemptive and Non-preemptive Scheduling

- The scheduling policy determines when it is time for process to be removed from the CPU and which ready process should be allocated the CPU next. The scheduling mechanism is composed of several different parts, depending on exactly how it is implemented in any particular operating system.
- CPU scheduling is divided into two types : Preemptive scheduling and non-preemptive scheduling.
- **Preemptive scheduling :** A scheduling method that interrupts the processing of a process and transfers the CPU to another process is called a preemptive CPU scheduling. The process switches from running state to the ready state and waiting state to the ready state.

- **Non-preemptive scheduling :** Non-preemptive operation usually proceeds towards completion uninterrupted. Once the system has assigned a processor to a process, the system cannot remove that processor from the process. The process switches from running state to the waiting state and termination of process.
- Preemptive scheduling increases the cost and it has higher overheads. This scheduling method is useful only in the high priority processes require rapid response.
- Non-preemptive scheduling is simple to implement. It requires some hardware platform. This method is attractive because of its simplicity. Windows 3.1 and Apple Macintosh operating system uses this scheduling method.

2.9.2 Difference between Preemptive and Non-preemptive Scheduling

Sr. No.	Preemptive scheduling	Non-preemptive scheduling
1.	Preemptive scheduling allows a process to be interrupted in the midst of its execution, taking the CPU away and allocating it to another process.	Non-preemptive scheduling ensures that a process relinquishes control of the CPU only when it finishes with its current CPU burst.
2.	Preemptive scheduling incurs a cost associated with access shared data.	Non-preemptive scheduling does not increase the cost.
3.	It also affects the design of the operating system Kernel.	It does not affects the design of OS Kernel.
4.	Preemptive scheduling is more complex.	Simple, but very inefficient.
5.	Example : Round robin method.	Example : First come first serve method.

2.9.3 CPU Scheduling Criteria

- Depending on the system, CPU scheduling criteria will change.
  1. **Throughput :** CPU scheduling should attempt to service the maximum number of processes per unit time. The higher is the number, the more work is done by the system.
  2. **Waiting time :** The average period of time a process spends waiting. Process is normally in the ready queue in waiting time.
  3. **Turnaround time :** Turnaround time start from process submission to completion of process.

     Turnaround time = Burst time + Waiting time

4. **Response time :** It is the time from the submission of a request until the first response is produced.
5. **CPU utilization :** It is average function of time during which the processor is busy.
6. **Fairness :** Avoid the process from the starvation. All the processes must be given equal opportunity to execute.
7. **Priority :** If the operating system assigns priorities to processes, the scheduling mechanism should favor the higher priority processes.
8. **Predictability :** A given process always should run in about the same amount of time under similar system loads.

- Depending upon the nature of operations the scheduling policy may differ. The CPU utilization and throughput are the system centered parameters. Fairness is affect by user and system.

2.9.4 Dispatcher

- Using dispatcher, CPU selects the process from the short term scheduler. Functions of the dispatcher is as follows :
  1. Switching context
  2. Switching to user mode
  3. Jump to proper location in the user program for restarting that program.
- Dispatcher is a small program that switches the processor from one process to another.

Dispatch Latency : It is time taken by dispatcher to stop running one process and start other process running is called as dispatch latency.

University Question

1. *Define context switch.* **SPPU : Aug.-17, Marks 2**

2.10 Scheduling Algorithms

SPPU : Aug.-17, 18, Oct.-18

- CPU scheduling algorithms are as follows :
  1. First In First Out (FIFO) scheduling
  2. Shortest job first scheduling
  3. Priority scheduling
  4. Round robin scheduling

2.10.1 First Come First Serve Scheduling

- This method of scheduling means that the first process to request the processor gets it until it finished execution. With this algorithm, processes are assigned the CPU in the order they request it.

- Normally there is a single queue of ready processes. When the first process enters the system from the outside, it is started immediately and allowed to run as long as it wants it. At the same time, other process enters into the system; they are put onto the end of the queue.
- New process enters the tail of the queue and the schedule selects the process from the head of the queue.
- When new process enters into the system, its process control block is linked to the end of the ready queue and it is removed from the front of the queue.
- When a process waits or blocks, it is removed from the queue and it queues up again in FCFS queue when it gets ready.
- FCFS is non-preemptive CPU scheduling algorithm. It is also called First In First Out method (FIFO).
- FCFS is simple to implement because it uses a FIFO queue. This algorithm is fine for most of the batch operating systems.
- FCFS is not useful in scheduling interactive processes because it cannot guarantee short response time. This algorithm performs much better for long processes than short ones.
- Turnaround time is unpredictable with the FCFS algorithm. Average waiting time of the FCFS is often quite long.
- Real life analogy is **buying tickets.**

Example 2.10.1 *Consider the following set of process that arrives at time 0, with the length of the CPU burst given in milliseconds. Calculate the average waiting time.*

Process	Burst time
P_1	14
P_2	2
P_3	4

Solution :

Gantt chart :

Such a diagram is called "Gantt charts", showing when each CPU burst uses the CPU. Here, each CPU burst comes from a different thread.

Waiting Time :

Process	Waiting time
P_1	0
P_2	14
P_3	16

$$\text{Average waiting time} = \frac{\text{Sum of } P_1, P_2 \text{ and } P_3 \text{ waiting time}}{3}$$

$$= \frac{0+14+16}{3} = 10$$

- If the processes change their order of arrive i.e. P_2, P_3, P_1, then the results will be different than first one and it is shown below :

Gantt chart :

Waiting Time :

Process	Waiting time
P_1	6
P_2	0
P_3	2

$$\text{Average waiting time} = \frac{\text{Sum of } P_1, P_2 \text{ and } P_3 \text{ waiting time}}{3}$$

$$= \frac{6+0+2}{3} = 2.666$$

Convey Effect :

- To reduce I/O device utilization, all I/O bound processes will be waiting excessively long for.processor bound ones. This is called as **convey effect.**
- If one process monopolizes the system, the extent of its overall effect on system performance depends on the scheduling policy and whether the process is processor bound or I/O bound.

Advantages

1. Simple to implement
2. Fair

Disadvantages

1. Waiting time depends on arrival order
2. Convoy effect : short process stuck waiting for long process
3. Also called head of the line blocking

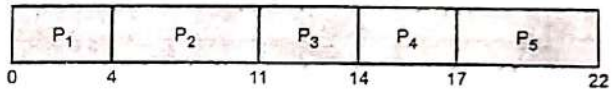
Example 2.10.2 *Consider the following set of process that arrive at time 0, with the length of CPU burst given in milliseconds. Calculate the average waiting time when the processes arrive in the following order :*

a. P_1, P_2, P_3, P_4, P_5

Provide the Gantt chart for the same.

Process	Burst time
P_1	4
P_2	7
P_3	3
P_4	3
P_5	5

Solution : Gantt chart :



Waiting time :

Process	Waiting time
P_1	0 – 0 = 0
P_2	4 – 0 = 4
P_3	11 – 0 = 11
P_4	14 – 0 = 14
P_5	17 – 0 = 17

$$\text{Average waiting time} = \frac{0+4+11+14+17}{5} = \frac{46}{5} = 9.2$$

Turnaround time : Turnaround time = Waiting time + Burst time

Process	Turnaround time
P_1	0 + 4 = 4
P_2 ws	4 + 7 = 11
P_3	11 + 3 = 14

P_4	14 + 3 = 17
P_5	17 + 5 = 22

Average turnaround time = $\frac{4+11+14+17+22}{5} = \frac{68}{5} = 13.60$

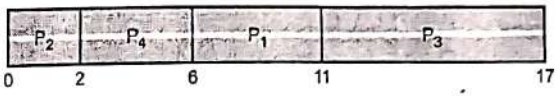
2.10.2 Shortest Job First Scheduling

- Shortest job first scheduling algorithm is also known as Shortest Job Next (SJN) scheduling algorithm. It handles the process based on length of their CPU cycle time. It reduces average waiting time over FIFO algorithm.
- SJF is a non-preemptive CPU scheduling algorithm.
- It does not work in interactive system because users do not estimate in advance the CPU time required to run their processes.
- SJF scheduling algorithm is used frequently in long term scheduling.
- When a process request a CPU, it must inform the system for how long it wants to use the CPU.
- When CPU become available, the system allocates into processes with the least expected execution time.
- To break ties, it follows the FCFS algorithm.
- Preemptive version of SJF is called as Shortest Remaining Time Next (SRTN).
- Preemptive SJF algorithm will preempt the currently exceutig process, whereas a non preemptive SJF algorithm will allow the currently running process to finish its CPU burst.
- SIF selects processes for service in a manner ensuring the next one will complete and leave the system as soon as possible.

Example 2.10.3 *Calculate the average waiting time and average turnaround time. Provide the Gantt chart for the same.*

Process	*Burst time*
P_1	5
P_2	2
P_3	6
P_4	4

Solution : Gantt chart :



Waiting time :

Process	Waiting time
P_1	6 – 0 = 6
P_2	0 – 0 = 0
P_3	11 – 0 = 11
P_4	2 – 0 = 2

Average waiting time = $\frac{6+0+11+2}{4} = \frac{19}{4} = 4.75$

Turnaround time : Turnaround time = Waiting time + Burst time

Process	Turnaround time
P_1	6 + 5 = 11
P_2	0 + 2 = 2
P_3	11 + 6 = 17
P_4	2 + 4 = 6

Average turnaround time = $\frac{11+2+17+6}{4} = \frac{36}{4} = 9$

- The SJF scheduling algorithm is optimal only when all of the processes are available at the same time and the CPU estimates are available.
- SJF scheduling algorithm may be **preemptive or non-preemptive.**

2.10.3 Priority Scheduling

- Priority CPU scheduling algorithm is preemptive and non-preemptive algorithm. It is one of the most common scheduling algorithm in batch system. Priority can be assigned by a system admin using characteristics of the process.
- In this scheduling algorithm, CPU select higher priority process first. If the priority of two process is same then FCFS scheduling algorithm is applied for solving the problem.

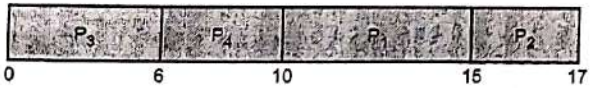
- The priority of a process determines how quickly its request for a CPU will be granted if other processes make competing requests.
- Each process is assigned a priority number for the purpose of CPU scheduling.
- The priority number is normally a non negative integer number.
- Non preemptive priority scheduling algorithm will simply put the new process at the head of the ready queue.
- Preemptive priority scheduling algorithm will preempt the CPU if the priority of the newly arrived process is higher than the priority of the currently running process.

Example 2.10.4 *Consider the following set of process that arrive at time 0, with the length of CPU burst given in milliseconds. Calculate the average waiting time and average turnaround time. Provide the Gantt chart for the same. (Time slice = 2)*

Process	Burst time	Priority
P_1	5	3
P_2	2	4
P_3	6	1
P_4	4	2

Solution : For non-preemptive priority scheduling

Gantt Chart :



Waiting time :

Process	Waiting time
P_1	10 – 0 = 10
P_2	15 – 0 = 15
P_3	0 – 0 = 0
P_4	6 – 0 = 6

Average waiting time $= \frac{10+15+0+6}{4} = \frac{31}{4} = 7.75$

Turnaround time : Turnaround time = Waiting time + Burst time

Process	Turnaround time
P_1	10 + 5 = 15
P_2	15 + 2 = 17
P_3	0 + 6 = 6
P_4	6 + 4 = 10

Average turnaround time $= \frac{15+17+6+10}{4} = \frac{48}{4} = 12$

Problem with Priority Scheduling

- Waiting time is more for lower priority process even if there required CPU burst time is less. Priority scheduling algorithm faces the starvation problem. Starvation problem can be solved by using **aging techniques**.
- In aging, the priority of the process will increase which is waiting for long time in the ready queue.

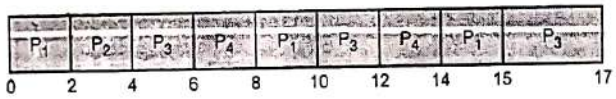
2.10.4 Round Robin Scheduling

- Round robin is a preemptive scheduling algorithm. It is used in interactive system.
- Here process are given a limited amount of time of processor time called a time slice or time quantum. If a process does not complete before its quantum expires, the system preempts it and gives the processor to the next waiting process. The system then places the preempted process at the back of the ready queue.
- Processes are placed in the ready queue using a FIFO scheme. With the RR algorithm, the principle design issue is the length of the time quantum to be used. For short time slice, processes will move through the system relatively quickly. It increases the processing overheads.

Example 2.10.5 *Consider the following set of process that arrive at time 0, with the length of CPU burst given in milliseconds. Calculate the average waiting time and average turnaround time. Provide the Gantt chart for the same. (Time slice = 2)*

Process	Burst time
P_1	5
P_2	2
P_3	6
P_4	4

Solution : Gantt chart :



Waiting time :

Process	Waiting time
P_1	0 – 0 + 8 – 2 + 14 – 10 = 10
P_2	2 – 0 = 2
P_3	4 – 0 + 10 – 6 + 15 – 12 = 11
P_4	6 – 0 + 12 – 8 = 10

Average waiting time = $\frac{10+2+11+10}{4} = \frac{33}{4} = 8.25$

Turnaround time : Turnaround time = Waiting time + Burst time

Process	Turnaround time
P_1	10 + 5 = 15
P_2	2 + 2 = 4
P_3	11 + 6 = 17
P_4	10 + 4 = 14

Average turnaround time = $\frac{15+4+17+14}{4} = \frac{50}{4} = 12.5$

2.10.5 Comparison between FCFS and RR

Sr. No.	FCFS	Round robin
1.	FCFS decision made is non-preemptive.	RR decision made is preemptive.
2.	It has minimum overhead.	It has low overhead.
3.	Response time may be high.	Provides good response time for short processes.
4.	It is troublesome for time sharing system.	It is mainly designed for time sharing system.
5.	The workload is simply processed in the order of arrival.	It is similar like FCFS but uses time quantum.
6.	No starvation in FCFS.	No starvation in RR.

2.10.6 Comparison of CPU Scheduling Algorithm

Algorithm	Policy type	Used in	Advantages	Disadvantages
FCFS	Non-preemptive	Batch	1. Easy to implement. 2. Minimum overhead.	1. Unpredictable turn around time. 2. Average waiting is more.
RR	Preemptive	Interactive	1. Provides fair CPU allocation. 2. Provides reasonable response times to interactive users.	1. Requires selection of good time slice.
Priority	Non-preemptive	Batch	1. Ensures fast completion of important jobs.	1. Indefinite postponement of some jobs. 2. Faces starvation problem.
SJF	Non-preemptive	Batch	1. Minimizes average waiting time. 2. SJF algorithm is optimal.	1. Indefinite postponement of some jobs. 2. Cannot be implemented at the level of short term scheduling. 3. Difficulty is knowing the length of the next CPU request.

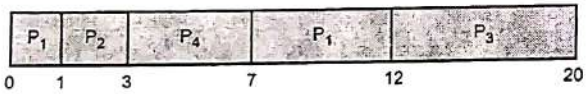
2.10.7 Shortest Remaining Time Next

- Preemptive SJF is called *shortest remaining time first*. In this algorithm, the scheduler selects the process with the smallest estimated run time to completion.
- This scheduler gives minimum wait times in theory but in certain situation due to preemption overhead, shortest process first might performance better.
- An advantageous of this method is that, the short processes are handled very quickly. The system requires very little overhead since it only makes a decision when a process completes or a new process is added.

- When a new process is admitted the SRTN algorithm only needs to compare the currently executing process with the new process, ignoring all other processes currently waiting to execute.
- Consider the four processes with their arrival and burst time.

Process	Burst Time	Arrival time
P_1	6	0
P_2	2	1
P_3	8	2
P_4	4	3

- The execution of the process is shown by the Gantt chart :



- Now we calculate waiting time and turnaround time for all processes.

Process	Waiting Time	Turnaround time
P_1	(0 - 0) + (7 -1) = 6	6 + 6 = 12
P_2	1 - 1 = 0	2 + 0 = 2
P_3	12 - 2 = 10	8 + 10 = 18
P_4	3 - 3 = 0	4 + 0 = 4

Average waiting time = $\frac{6+0+10+0}{4} = \frac{16}{4} = 4$

Average waiting time = $\frac{12+2+18+4}{4} = \frac{36}{4} = 9$

Advantages :

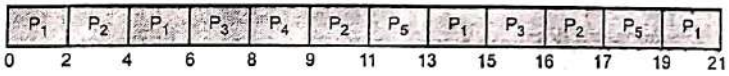
1. Very short processes get very good service.
2. The penalty ratios are small; this algorithm works extremely well in most cases
3. This algorithm provably gives the highest throughput of all scheduling algorithms if the estimates are exactly correct.

Example 2.10.6 *For the table given below, calculate average waiting time and average turnaround time and draw a Gantt Chart illustrating the process execution using following scheduling algorithms.*

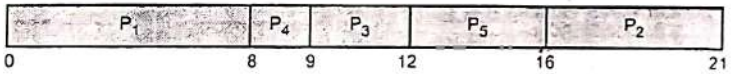
i) RR (Time slice - 2 units) ii) SJF (non - preemptive) **SPPU : Aug.-17, Marks 8**

Process	Arrival Time	Burst Time
P_1	0	8
P_2	1	5
P_3	3	3
P_4	4	1
P_5	6	4

Solution. : i) RR (time slice 2 units) : Gantt chart



ii) SJF (non-preemptive) : Gantt chart



Waiting time and turnaround time :

Process	Waiting time		Turnaround time	
	RR	SJF	RR	SJF
P_1	0 - 0 + 4 - 2 + 13 - 6 + 19 - 15 = 13	0 - 0 = 0	21	8
P_2	2 - 1 + 9 - 4 + 16 - 11 = 11	16 - 1 = 15	16	20
P_3	6 - 3 + 15 - 8 = 10	9 - 3 = 6	13	9
P_4	8 - 4 = 4	8 - 4 = 4	5	5
P_5	11 - 6 + 17 - 13 = 9	12 - 6 = 6	13	10

Average waiting time for RR = $\frac{13+11+10+4+9}{5} = 9.4$

Average waiting time for SJF = $\frac{0+15+6+4+6}{5} = 6.2$

Average turnaround time for RR = $\frac{21+16+13+5+13}{5} = 13.6$

Average turnaround time for SJF = $\frac{8+20+9+5+10}{5} = 10.4$

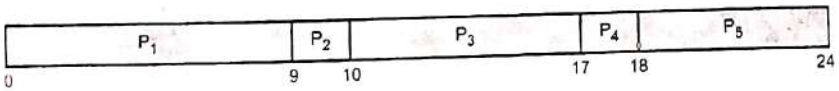
Example 2.10.7 *For the table given below, calculate average waiting time and average turnaround time, also draw a Gantt chart illustrating the process execution using following scheduling algorithms.*

i) FCFS ii) SJF (preemptive)

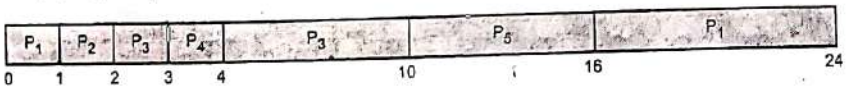
SPPU : Aug.-17, Marks 8

Process	Arrival Time	Burst Time
P_1	0	9
P_2	1	1
P_3	2	7
P_4	3	1
P_5	4	6

Solution : i) FCFS : Gantt chart



ii) SJF (preemptive) : Gantt chart



Waiting time and turnaround time :

Process	Waiting time		Turnaround time	
	FCFS	SJF	FCFS	SJF
P_1	0 – 0 = 0	0 – 0 + 16 – 1 = 15	0 + 9 = 9	15 + 9 = 24
P_2	9 – 1 = 8	1 – 1 = 0	8+1 = 9	0 + 1 = 1
P_3	10 – 2 = 8	2 – 2 + 4 – 3 = 2	8 + 7 = 15	2 + 7 = 9
P_4	17 – 3 = 14	3 – 3 = 0	14 + 1 = 15	0 + 1 = 1
P_5	18 – 4 = 14	10 – 4 = 6	14 + 6 = 20	6 + 6 = 12

Average waiting time for FCFS = $\frac{0+8+8+14+14}{5} = 8.8$

Average waiting time for SJF = $\frac{15+0+2+0+6}{5} = 4.6$

Average turnaround time for FCFS = $\frac{9+9+15+15+20}{5} = 13.6$

Average turnaround time for SJF = $\frac{24+1+9+1+12}{5} = 9.4$

Example 2.10.8 *For the table given below, calculate average waiting time and average turnaround time and draw a Gantt chart illustrating the process execution using following scheduling algorithms.*

i) Round robin (time slice - 2 units) ii) Priority (non-preemptive)

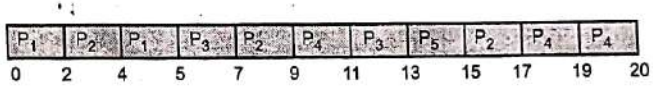
Process	Arrival Time	Burst Time	Priority
P_1	0	3	5
P_2	2	6	2
P_3	4	4	4
P_4	6	5	3
P_5	8	2	1

Note : For priority scheduling, minimum value indicates higher priority.

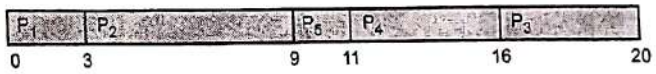
SPPU : Oct.-18, Marks 8

Solution : Gantt chart :

i) Round robin (time slice = 2 units)



ii) Priority (non - preemptive)



Waiting time and turnaround time :

Process	Waiting time: Round robin	Waiting time: Priority (non-preemptive)	turnaround time: Round robin	turnaround time: Priority (non-preemptive)
P_1	0 – 0 + 4 – 2 = 2	0 – 0 = 0	5	3
P_2	2 – 2 + 7 – 4 + 15 – 9 = 9	3 – 2 = 1	15	21
P_3	5 – 4 + 11 – 7 = 5	16 – 4 = 12	9	13
P_4	9 – 6 + 17 – 11 = 11	11 – 6 = 5	16	21
P_5	13 – 8 = 5	9 – 8 = 1	7	9

Average waiting time of Round Robin = (2 + 9 + 5 + 11 + 5) / 5 = 6.4

Average waiting time of Priority (non-preemptive) = (0 + 1 + 12 + 5 + 1) / 5 = 3.8

Average turnaround time of Round Robin = (5 + 15 + 9 + 16 + 7) / 5 = 10.4

Average turnaround time of Priority (non-preemptive) = (3 + 21 + 13 + 21 + 9) / 5 = 13.4

Example 2.10.9 *For the table given below, calculate average waiting time and average turnround time and draw a Gantt chart illustrating the process execution using following scheduling algorithms.*

i) SJF (non-preemptive) ii) Priority (preemptive)

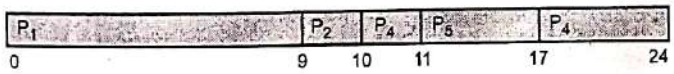
Process	Arrival Time	Burst Time	Priority
P_1	0	9	3
P_2	1	1	2
P_3	2	7	1
P_4	3	1	5
P_5	4	6	4

Note : For priority scheduling, minimum value indicates higher priority.

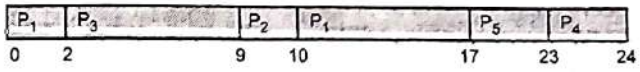
SPPU : Aug.-18, Marks 8

Solution : Gantt Chart :

i) SJF (non - preemptive)



ii) Priority (Preemptive)



Waiting time and turnaround time :

Process	Waiting time: SJF (non-preemptive)	Waiting time: Priority (Preemptive)	Turnaround time: SJF (non-preemptive)	Turnaround time: Priority (Preemptive)
P_1	0 – 0 = 0	0 – 0 + 10 – 2 = 8	9	17
P_2	9 – 1 = 8	9 – 1 = 8	9	9
P_3	17 – 2 = 15	2 – 2 = 0	22	7
P_4	10 – 3 = 7	23 – 3 = 20	8	21
P_5	11 – 4 = 7	17 – 4 = 13	13	20

Average waiting time of SJF (non-preemptive) = (0 + 8 + 15 + 7 + 7) / 5 = 7.4

Average waiting time of Priority (preemptive) = (8 + 8 + 0 + 20 + 13) / 5 = 9.8

Average turnaround time of SJF (non-preemptive) = (9 + 9 + 22 + 8 + 13) / 5 = 12.2

Average turnaround time of Priority (preemptive) = (17 + 9 + 7 + 21 + 20) / 5 = 14.8

2.11 Multiple Choice Questions

Q.1 A _____ is the unit of work in a modern time-sharing system.

a) program b) process

c) task d) device driver

Q.2 A process is more than the program code, which is sometimes known as the _____.

a) text section b) data section

c) stack d) heap

Q.3 A program becomes a process when an executable file is loaded into _____.

a) hard disk b) CPU

c) memory d) all of these

Q.4 PCB stands for _____.

[a] process control block [b] program control block
[c] process control bus [d] program control bus

Q.5 As processes enter the system, they are put into a _____ queue, which consists of all processes in the system.

[a] device [b] ready
[c] job [d] memory

Q.6 Short term scheduler is also called _____ scheduler.

[a] job [b] CPU
[c] medium [d] device

Q.7 The interval from the time of submission of a process to the time of completion is termed as _____.

[a] waiting time [b] turnaround time
[c] response time [d] throughput

Q.8 Which algorithm is defined in Time quantum ?

[a] Shortest job scheduling algorithm
[b] Round robin scheduling algorithm
[c] Priority scheduling algorithm
[d] Multilevel queue scheduling algorithm

Q.9 Aging is a technique to avoid _____ in a scheduling system.

[a] deadlock [b] blocking
[c] starvation [d] convey effect

Q.10 A new process is created by _____ system call.

[a] create [b] read
[c] write [d] fork

Q.11 SJF stands for _____.

[a] Short Job First [b] Shortest Job First
[c] Simple Job First [d] Second Job First

Q.12 The SJF algorithm is a special case of the general priority-scheduling algorithm.

[a] FCFS [b] round robin
[c] SJF [d] all of these

Q.13 A preemptive priority scheduling algorithm will preempt the CPU if the priority of the newly arrived process is higher than the priority of the _____.

[a] parent process [b] init process
[c] all process [d] currently running process

Q.14 A non-preemptive priority scheduling algorithm will simply put the new process at the head of the _____ queue.

[a] device [b] ready
[c] suspended [d] none of these

Q.15 Which of the following in NOT process states ?

[a] New [b] Ready
[c] Waiting [d] Dispatcher

Answer Keys for Multiple Choice Questions

Q.1	b	Q.2	a	Q.3	c	Q.4	a
Q.5	c	Q.6	b	Q.7	b	Q.8	b
Q.9	c	Q.10	d	Q.11	b	Q.12	c
Q.13	d	Q.14	b	Q.15	d