Operating System

Goal of the Subject

The goal of this book is to provide all important concepts of operating systems such as Processes and Threads, Mutual Exclusion, CPU Scheduling, Deadlock, Memory Management, Virtual Memory and File Systems.

- Understand the purpose of the operating system
- Distinguish between a resource, a program, and a process
- Solutions of semaphores
- Describe various memory page replacement algorithms
- Describe how files are stored in secondary storage

Operating System

INTRODUCTION

An operating system is a program that acts as an intermediary between a user of a computer and the computer hardware. Two primary aims of an operating systems are to manage resources (e.g. CPU time, memory) and to control users and software. The book consists of topic wise *Examples* and *Student Assignment* questions which test the important concepts from the lesson and provide practice problems. This subject includes the following chapters.

1. **Basic Concepts of OS:** In this chapter we discuss Basic Concepts of OS, Types of Operating System, Dual Mode Operations and System Call.
2. **Processes and Threads:** In this chapter we discuss Process, Operations on a Process, Scheduling, Thread and Co-operating Processes (Inter Process Communication).
3. **CPU Scheduling:** In this chapter we discuss Goals of CPU Scheduling and Scheduling Algo.
4. **Process Synchronization:** In this chapter we discuss Synchronization, Critical-Section Problem, Synchronization Techniques, Semaphores and Classical Problems of Synchronization with Semaphore Solution.
5. **Deadlock and Concurrency:** In this chapter we discuss Concurrency, Deadlock, Conditions for a deadlock, Methods of Handling Deadlocks, Prevention, Avoidance and Detection & Recovery.
6. **Memory Management:** In this chapter we discuss Logical (Virtual) Vs Physical Address Space, Memory Allocation Techniques, Internal Fragmentation and External Fragmentation, Paging, Segmentation, Segmented Paging and Buddy System.
7. **Virtual Memory:** In this chapter we discuss Page Fault, Page Replacement, Dynamic Paging Algorithms and Frame Allocation.
8. **File System:** In this chapter we discuss Directories, File Management System, File Allocation Methods and Free Space Management.
9. **IO System:** In this chapter we discuss I/O System Structure, Magnetic Storage Devices, Disk Scheduling and Disk Scheduling Algorithms.

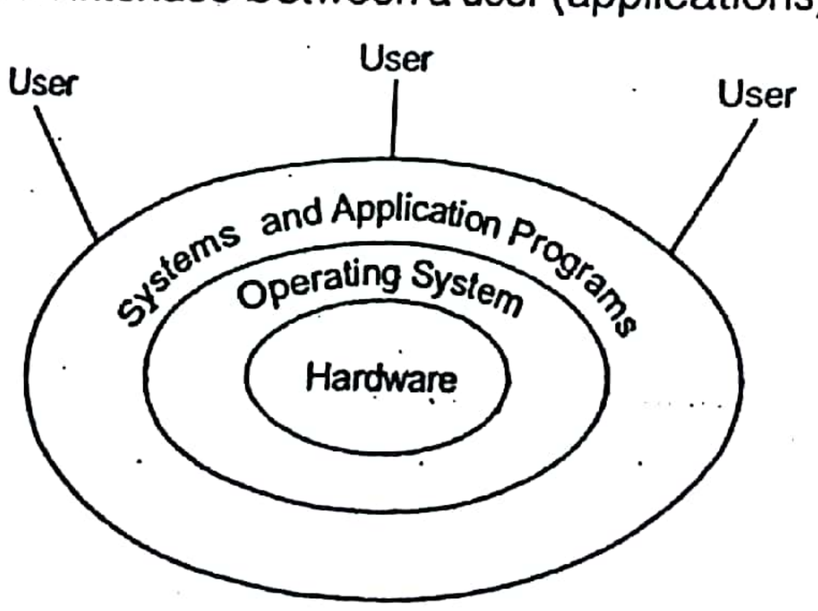
■■■■

CHAPTER 01

Basic Concepts of OS

1.1 Operating System (OS)

It is a program that acts as an interface between a user (applications) and the computer hardware.

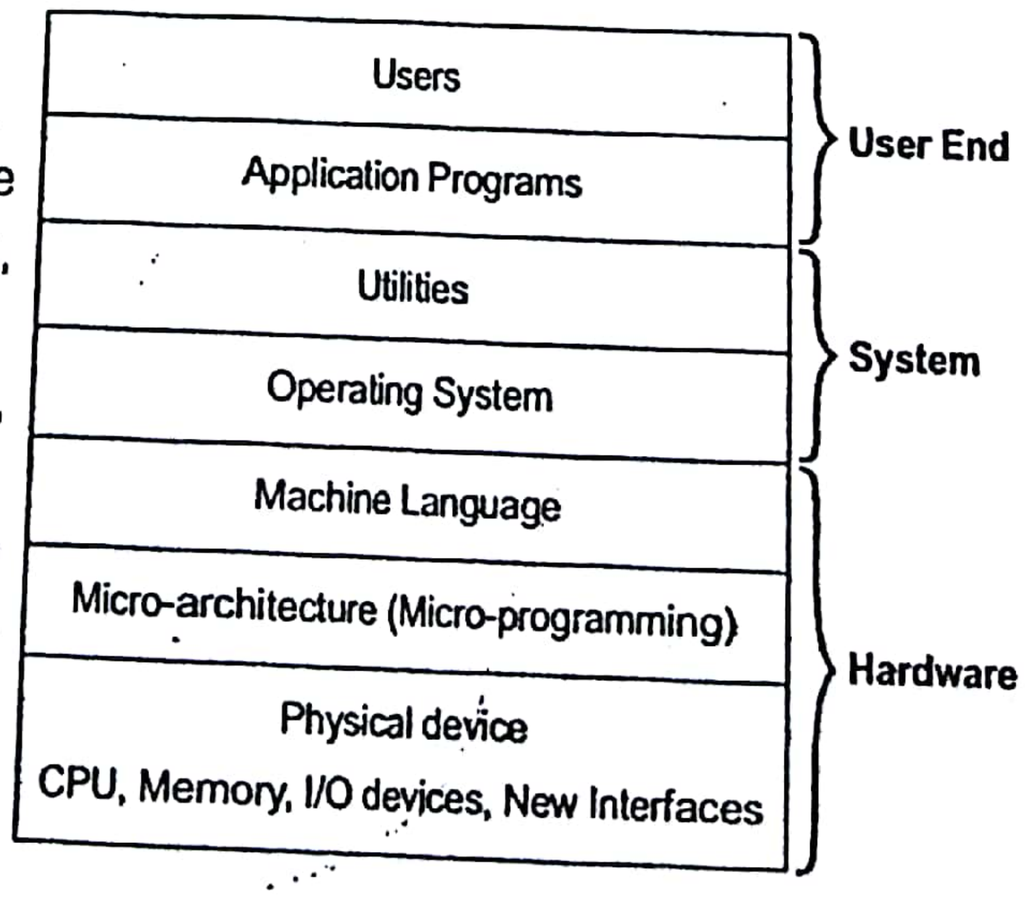


Operating System allows the compute system in more convenient to we and efficient resource utilization.

1.2 Structure of Computer System

A computer system consists of :

- **Users:** People, other computers, machines, etc.
- **Application programs:** Compilers, database systems, video games, business applications, web-browsers, etc.
- **System programs:** Shells, editors, compilers, development tools, etc.
- **Operating System:** It is a system program which controls and coordinates the use of hardware among application programs.
- **Hardware:** CPU, Disk, Memory, I/O devices, etc.



NOTE: Firmware (BIOS) is a software which is permanently stored on chip but upgradable. It loads the operating system during the boot.

1.3 Layered View of Operating System Services

1. **User View:** The OS is an interface, hides the details which must be performed and presents a virtual machine to the user that makes easier to use.
   OS provides the following services to the user.
   (*i*) Execution of a program
   (*ii*) Access to I/O devices
   (*iii*) Controlled access to files
   (*iv*) Error detection (Hardware failures, and software errors)
2. **Hardware View:** The operating system manages the resources efficiently in order to offer the services to the user programs.
   OS acts as resources managers:
   (*i*) Allocation of resources
   (*ii*) Controlling the execution of a program
   (*iii*) Controls the operations of I/O devices
   (*iv*) Protection of resources
   (*v*) Monitors the data
3. **System View:** OS is a program that functions in the same way as other programs. It is a set of instructions that are executed by the processor.
   OS acts as a program to perform the following:
   (*i*) Hardware upgrades
   (*ii*) New services
   (*iii*) Fixes the issues of resources
   (*iv*) Controls the user and hardware operations

Goals of operating system: Primary goal is convenience and secondary goal is efficiency.

1.4 History of Operating System

Tube based:

- Hardware can run one program at a time (Serial Processing) (1945-1955)
- Numerical calculations

Transits based:

- High level languages (Fortran)
- (Batch programing) (1955-1965)

IC Circuits based:

- Time sharing and multiprogramming (1965-1980)
- Business Applications, Scientific and Engineering Applications

Working and PCs:

- Real-time, Embedded, parallel, Network programming (1980-2000)

Networking:

- Parallel Computer Architectures (2000-Present)
- High Speed Networks.

1.5 Types of Operating System

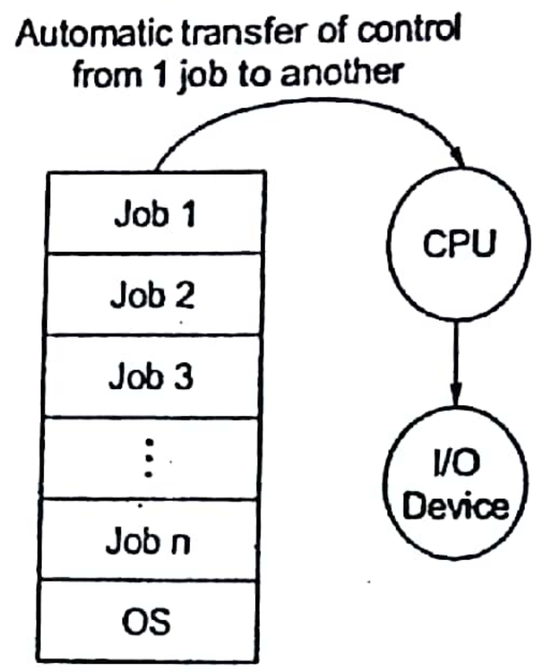
1. Serial OS
2. Batch OS
3. Interactive OS
4. Multiprogrammed OS

5. Time sharing OS
6. Real time OS
7. Network OS
8. Parallel OS
9. Distributed OS
10. Clustered OS
11. Handheld OS

1.5.1 Simple Batch Systems

The operating system in early computers was fairly simple. Its major task was to transfer control automatically from one job to the next. The operating system was always in memory. To speed up processing, jobs with similar needs were batched together and were run through the computer as a group. Thus, the programmers would leave their programs with the operator. The operator would sort programs into batches with similar requirements, and as the computer become available, would run each batch.

- Systems allowed automatic job sequencing by a resident operating system and greatly improved the overall utilization of the computer.
- In batch system there is lack of interaction between the user and the job while the job is executing.
- The CPU utilization was still low. In this execution environment the CPU is often idle. This idleness occurs because the speeds of the mechanical input/output devices are intrinsically slower than those of electronic devices.
- Batch system are appropriate for executing large jobs that need little interaction. *Example:* IBM OS/2.



Batch System

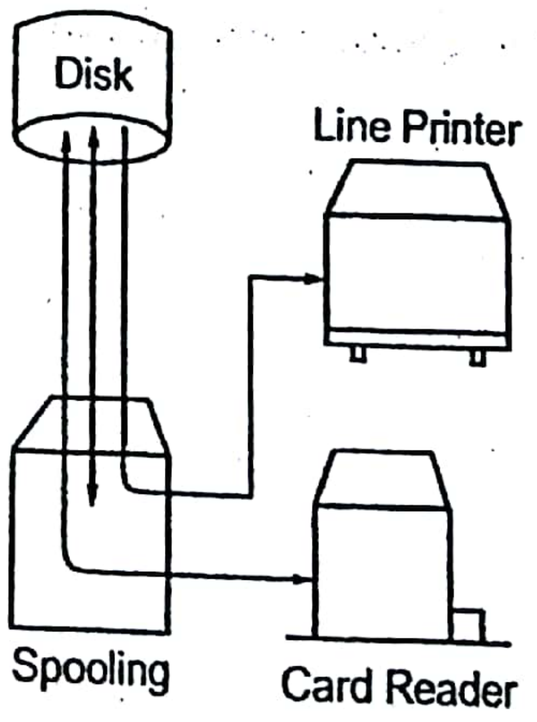
Spooling

Spool is acronym for simultaneous peripheral operation On-Line. Spooling overlaps I/O of one job with the computation of other job. Spooling uses the disk as a huge buffer for reading as far ahead as possible on input devices and for storing output files until the output devices are able to accept them. A spool is a buffer that holds output for a device, such sa a printer, that cannot accept interleaved data streams.

Although a printer can serve only one job at a time, several applications may wish to print their output concurrently, without having their output mixed together. The operating system solves this problem by intercepting as output to the printer. Each applications' output is spooled to a separate disk file.

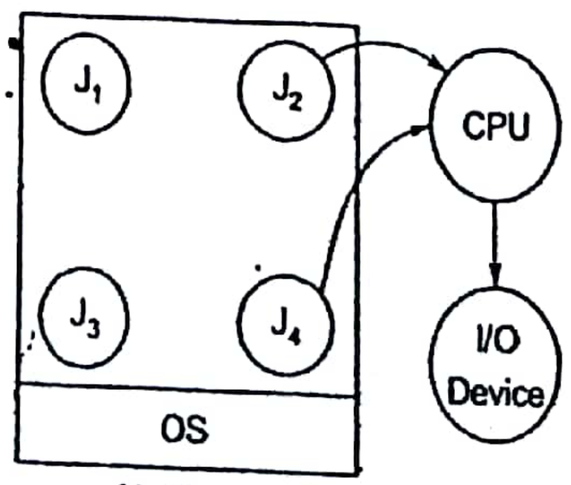
When an application finishes printing, the spooling system queue picks the next spooled-file for input to the printer.

The spooling system copies the queued spool files to the printer one at a time. Even in a simple system, the spooler may be reading the input of one job while printing the output of a different job. Spooling can keep both the CPU and the input/output devices working at much higher rates.



1.5.2 Multi-Programmed Systems

- Several jobs are kept in main memory at the same time and the CPU is multiplexed among them, to increase CPU utilization.
- A job pool on the disk consists of a number of jobs that are ready to be executed. Subsets of these jobs reside in the memory for execution.
- The operating system picks and executes one of the jobs in memory.

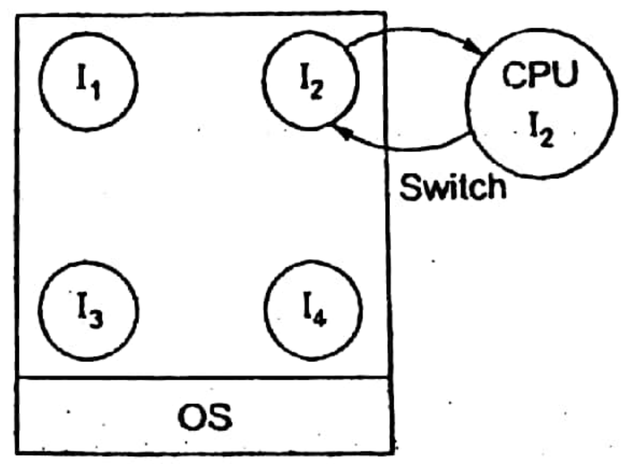


Multi-programming System

- When this job in execution needs an input/output operation to complete, Instead of waiting for the job to complete the input/output, it switches to the subset of jobs waiting for CPU.
- In a non multi programmed system the CPU would sit idle. In multiprogramming system the operating system simple switches to and executes another job.
- If several jobs are ready to be brought into memory and there is not enough room for all of them, then the system must choose among then. Making this decision is job scheduling.
  Example: Windows and Unix.

1.5.3 Time Sharing System (Multitasking)

- Time sharing or multitasking is a logical extension of multiprogramming. Multiple jobs are executed by the CPU switching between them, but the switches occur is frequently that the users may interact with each program while it is running.
- An interactive computer system provides direct communication between the user and the system. The user gives instructions to the operating system or to a program directly and receives an immediate response.
- A time shared operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time shared computer.
- A time shared operating system allows the many users to share the computer simultaneously. Since each action or command in a time shared system tends to be short, only a little CPU time is needed for each user. As the system switches rapidly from one user to the next, each user is given the impression that she has her own computer, whereas actually one computer is being shared among many users.



Multi-tasking System

1.5.4 Real Time System

Used when there are rigid time requirements on the operating of a processor or the flow of data. Systems that control scientific experiments, medical imaging systems, industrial control systems, and some display systems are real-time systems. A real time operating system has well defined, fixed time constraints. Processing must be done within the defined constraints or the system will fail. *Example*: RTOS

Types of Real Time Systems

1. **Hard real time system:** Guarantees that critical tasks completed on time. This goal requires that all delays in the system be bounded from the retrieval of stored data to the time that it takes the operating system to finish any request made to it. Kernel delays need to be bounded and more restrictive. *Example*: Satelite, Missile System.
2. **Soft real time system:** A less restrictive type of real time system is a soft real time system, where a critical real time task gets priority over other tasks, and retains that priority until it completes. Soft real time is an achievable goal that can be mixed with other types of systems. However they have more limited utility than do hard real time system. Given their late of deadline support they are risky to use for industrial control and robotics. *Example*: Banking system.

1.5.5 Parallel Systems (Multiprocessors)

- Multiprocessor system have more than one processor in close communication, sharing the computer bus, the clock, and sometimes memory and peripheral devices. These systems are referred to as tightly coupled systems.

- One advantage of building this kind of system is increased throughput. By increasing the number of processors we hope to get more work done in a shorter period of time.
- Multiprocessors can also save money because the processors can share peripherals, cabinets and power supplies. If several programs are to operate on the same set of data, it is cheaper to store those data on one disk and to have all the processors share them, rather than to have many computers with local disks and many copies of the data.
- Another advantage is increased reliability.
- If functions can be distributed properly among several processors, then the failure of one processor will not halt the system, but rather will only slow it down.
- The ability to continue providing service proportional to the level of surviving hardware is called graceful degradation. Systems that are designed for graceful degradation also called fault-tolerant.

Symmetric Multiprocessing Model

In this each processor runs an identical copy of the operating system and these copies communicate with one another as needed.

Asymmetric Multiprocessing Model

In this model each processor is assigned a specific task. A master processor controls the system; the other processors either look to master for instruction or have predefined tasks. This scheme defines a master-slave relationship. The master processor schedules and allocates work to the slave processors.

1.5.6 Distributed Systems

The processors do not share memory or a clock, instead each processor has its own local memory. The systems are also referred as loosely coupled systems.

Advantages of Distributed Systems

- **Resource sharing:** If a number of different sites are connected to one another then, a user at one site may be able to use the resources available at another.
- **Computation speedup:** If a particular computation can be partitioned into a number of subcomputations that can run concurrently then a distributed system may allow us to distribute the computation among the various sites to run that computation concurrently.
- **Reliability:** If one site fails in a distributed system, the remaining silt can potentially continue operating.
- **Communication:** When many sites are connected to one another by a communication network, the processes at different sites have the opportunity to exchange information.

1.5.7 Difference between Distributed OS and Network OS

Network Operating System	Distributed Operating System
1. Each computer has its own OS 2. It allows interaction between the machines by having a common communication architecture 3. Independent machines accessed by the user	1. Common OS shared by a network of computers 2. Single OS controlling the network 3. Dependent machines accessed by the user to share the resources

1.6 Dual Mode Operations

A processor can support two modes of execution:

1. Kernel / Protected / Supervisor / System/Monitor / Privileged mode.
2. User mode / Non-privileged mode

Operating system runs in Kernel mode and user programs run in user mode. Mode bit is used to decide the mode of operating system. If mode bit is 0, it operates in kernel mode. Otherewise it operates in user mode when mode bit is 1.

User mode (Non-privileged mode)	Mode bit = 1
Kernel mode (Privileged mode)	Mode bit = 0

1.6.1 Kernel Mode

A process running in Kernel mode has following:

- Full access to machine instruction set.
- Direct access to hardware (memory, I/O devices, etc).

OS and device drivers must run in Kernel mode. MS DOS only support Kernel mode.

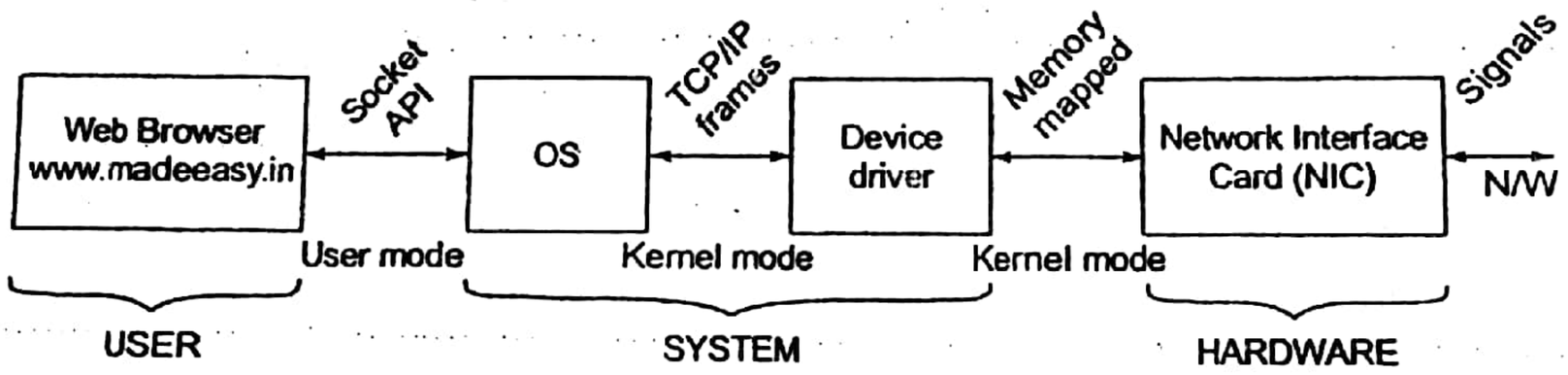
Privileged instructions: Set timer, Clear memory, Disable interrupts, Access I/O devices, etc.

1.6.2 User Mode

A process running in user mode has following:

- Access to the limited set of machine instructions.
- No direct access to hardware.
- Hardware access is coordinated by OS.

Non-privileged instructions: Read clock, Generate trap, User to Kernel mode switch, etc.



1.6.3 Privileged Instructions and Non-privileged Instructions

Privileged Instructions	Non-Privileged Instructions
I/O operations	Reading system time
Context switching	Reading the states of CPU
Disabling the interrupts	Sending the final printout of printer
Set system lock	
Remove process from memory	
Changing memory LOC of process	

Note: In boot time the system always start in kernel. The OS always run in kernel mode.

1.7 Functions of Operating System

- It controls all of computer resources.
- It provides valuable services to user programs.
- It coordinates the execution of user programs.
- It provides resources to user programs.
- It provides an interface (virtual machine) to the user.
- It hides the complexity of software.
- It supports the multiple execution modes.
- It monitors the execution of user programs to prevent errors.

1.8 Operating System Components

- **Process Management:** Operating system manages the process creation and deletion.
- **Main Memory Management:** Operating system keeps track of allocation and deallocation of main memory.
- **Secondary Storage Management:** Operating system uses secondary storage (Disk) to extend main memory.
- **I/O System Management:** Operating system uses I/O system to manage the devices.
- **File Management:** Operating system manages the file creation, deletion and permissions.
- **Protection System:** Operating system protects the resources by authorization.
- **Interface:** Operating system provides an interface to the user using the facilities by "command interpreter" or "Graphical user interfaces".
- **Networking:** The processors in the system are connected through a communication network. Communication takes place using a protocol. A distributed system provides user access to various system resources.

1.9 System Call

System call provides the services of operating system to the user programs via Application Program Interface (API). It provides an interface between a process and operating system to allow user level processes to request services of operating system. System calls are the only entry points into the Kernel System. All programs needing resources must use system calls.

Services Provided by System Calls

1. Process creation and management
2. Main memory management
3. File Access, Directory and File system management
4. Device handling (I/O)
5. Protection
6. Networking, etc.

Types of System Calls

1. **Process control:** end, abort, create, terminate, allocate and free memory.
2. **File management:** create, open, close, delete, read file etc.
3. Device management
4. Information maintenance
5. Communication

1.10 Interrupts

There are two types of interrupts:

1. **Interrupt:** Interrupt is a hardware generated which changes the flow of execution within the system. Interrupt handler deals with hardware generated interrupts to control such interrupts. Interrupt can be used to signal the completion of I/O.
   - External event
   - Asynchronous event
   - Independent of the currently executed process instructions.

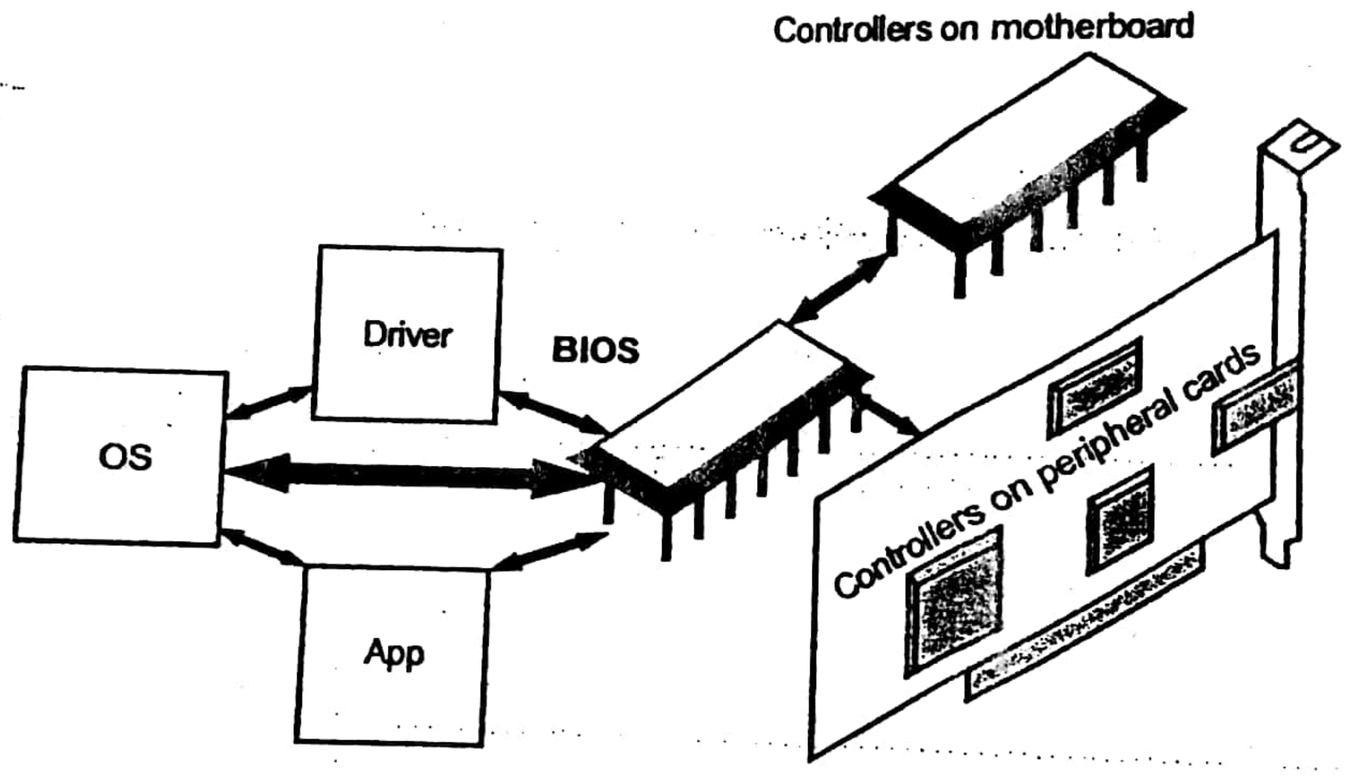
   Examples: Clock interrupt, I/O interrupt and Memory fault.

2. **Trap:** Trap is a software generated signal either to call operating system routines or to catch the arithmetic errors.
   - Exceptions and system calls
   - Synchronous event
   - Internal (exceptions) events or external events.

1.11 Booting of OS

Hardware doesn't know where the operating system resides and how to load it. Hence it needs a special program to do this job i.e., Bootstrap loader. *Example:* BIOS (Boot Input Output System).

Bootstrap loader locates the kernel, loads it into main memory and starts its execution. In some systems, a simple bootstrap loader fetches a more complex boot program from disk, which in turn loads the kernel.



BIOS Interaction

Booting Process

- Reset event on CPU (power up, reboot) causes instruction register to be loaded with a predefined memory location. It contains a jump instruction that transfers execution to the location of Bootstrap program.
- This program is form of ROM, since RAM is in unknown state at system startup. ROM is convenient as it needs no initialization and can't be affected by virus.
- Run diagnostics to determine the state of machine. If diagnostics pass, booting continues.
- Runs a **Power-On Self Test (POST)** to check the devices that the computer will rely on, are functioning.
- BIOS goes through a preconfigured list of devices until it finds one that is bootable. If it finds no such device, an error is given and the boot process stops.
- Initializes CPU registers, device controllers and contents of the main memory. After this, it loads the OS.
- On finding a bootable device, the BIOS loads and executes its **boot sector**. In the case of a hard drive, this is referred to as the **Master Boot Record (MBR)** and is often not OS specific.
- The MBR code checks the **partition table** for an active partition. If one is found, the MBR code loads that partitions **boot sector** and executes it.
- The boot sector is often **operating system** specific, however in most operating systems its main function is to load and execute a **kernel**, which continues startup.

- If there is no active partition or the active partition's boot sector is invalid, the MBR may load a secondary boot load and pass control to it and this secondary boot loader will select a partition (often via user input) and load its boot sector.
- Examples of secondary boot loaders:
  (a) GRUB : GRand Unified Bootloader
  (b) LILO : LInux LOader
  (c) NTLDR : NT Loader
- System such as cellular phones, PDAs and game consoles stores entire OS on ROM. Done only for small OS, simple supporting hardware, and rugged operation.
- Changing bootstrap code would require changing ROM chips.
  EPROM : Erasable Programmable ROM

1.12 Types of Kernel Designs

Monolithic Kernel

- All OS services operate in kernel space.
- Good performance.
- **Disadvantages:** Dependencies between system component. Complex and huge (millions(!) of lines of code).

Microkernel

- Minimalist approach
- IPC, virtual memory, thread scheduling
- Put the rest into user space
- Device drivers, networking, file system, user interface
- More stable with less services in kernel space
- **Disadvantages:** Lots of system calls and context switches.
  Example: Mach, L4, AmigaOS, Minix, K42.

Hybrid Kernel

It combines the best of both worlds

- Speed and simple design of a monolithic kernel
- Modularity and stability of a microkernel
- Still similar to a monolithic kernel
  Example: Windows NT, NetWare, BeOS

Exokernel

- Follows end-to-end principle
- Extremely minimal
- Fewest hardware abstractions as possible
- Just allocates physical resources to apps
- **Disadvantages:** More work for application developers. *Example:* Nemesis, ExOS

Q.1 In a multiprogramming and time-sharing environment, several users share the system simultaneously. This situation can result in various security problems.

(a) What are two such problems?

(b) Can we ensure the same degree of security in a time-shared machine as we have in a dedicated machine? Explain your answer.

Ans: (a) Stealing or copying one's programs or data; using system resources (CPU, memory, disk space, peripherals) without proper accounting.

(b) Probably not, since any protection scheme devised by humans can inevitably be broken by a human, and the more complex the scheme, the more difficult it is to feel confident of its correct implementation.

Q.2 What is the main advantage of multiprogramming?

Ans: Multiprogramming makes efficient use of the CPU by overlapping the demands for the CPU and its I/O devices from various users. It attempts to increase CPU utilization by always having something for the CPU to execute.

Q.3 List the steps that are necessary to run a program on a completely dedicated machine.

Ans: (a) Reserve machine (b) Manually load program into memory (c) Load starting address and begin execution (d) Monitor and control execution of program form console.

Q.4 Writing an operating system that can operate without interference from malicious or undebugged user programs requires some hardware assistance. Name three hardware aids for writing an operating system.

Ans: (a) Monitor/user mode (b) Privileged instructions (c) Timer (d) Memory protection.

Q.5 What is the purpose of the command interpreter? Why is it usually separate from the kernel?

Ans: It reads commands from the user or from a file of commands and executes them, usually by turning them into one or more system calls. It is usually not part of the kernel since the command interpreter is subject to changes.

Q.6 What is the main advantage of the layered approach to system design?

Ans: As in all cases of modular design, designing an operating system in a modular way has several advantages. The system is easier to debug and modify because changes affect only limited sections of the system rather than touching all sections of the operating system. Information is kept only where it is needed and is accessible only within a defined and restricted area, so any bugs affecting that data must be limited to a specific module or layer.

Q.7 What are the main advantages of the microkernel approach to system design?

Ans: Benefits typically include the following (a) adding a new service does not require modifying the kernel, (b) it is more secure as more operations are done in user mode than in kernel mode, and (c) a simpler kernel design and functionality typically results in a more reliable operating system.

Summary

- **Multitasking:** Interaction between multiple processes on the same processor.
- **Multiprogramming:** Interaction between multiple processes in the system. (either on same processor or on different processor)
- **Multiprocessing:** Interaction between multiple processors to execute the processes parallely.
- **Concurrency includes:**
  - Communication among processes/threads.
  - Sharing system resources.
  - Cooperative processing of shared data.
  - Synchronization of process/thread activities.
  - Organized CPU scheduling.
  - Solving deadlock and starvation problems.
- **Concurrency arises:**
  - Interaction between multiple processes running on one CPU.
  - Interaction between multiple threads running in one process.
  - Interaction between multiple processors running multiple processes/threads.
- **Multicomputing:** Interaction between multiple computers running distributed processes.

Student's Assignment

Q.1 The software that contains the core components of the operating system is called
(a) Controller (b) Root
(c) Kernel (d) None of the above

Q.2 When a computer is switched on, where is the operating system loaded?
(a) BIOS (b) ROM
(c) POST (d) RAM

Q.3 On which chip is the BIOS program permanently stored?
(a) RAM (b) ROM
(c) SIMBA (d) none of these

Q.4 Spooling helps because
(a) it is a more secure method of accessing data
(b) print jobs go more smoothly with less stop and go
(c) the computer is released to do other things while still printing
(d) None of the above

Q.5 When a computer is "swapping", it is
(a) moving data from hard drive to floppy drive
(b) moving data from memory to the swap file on the hard drive
(c) moving data between registers in memory
(d) None of the above

Q.6 The Operating System is responsible for
(a) controlling peripheral devices such as monitor, printers, disk drives
(b) provide an interface that allows users to choose programs to run/manipulate files
(c) manage users' files on disk
(d) All of the above

Q.7 Which of the following does an operating system do in a stand-alone computer system?
(a) manages the user's files
(b) provides the interface to allow the user to communicate with the computer
(c) controls the various peripherals
(d) All of the above

Q.8 Which of the following is true about a terminal on a time-sharing computer system?

CHAPTER 02

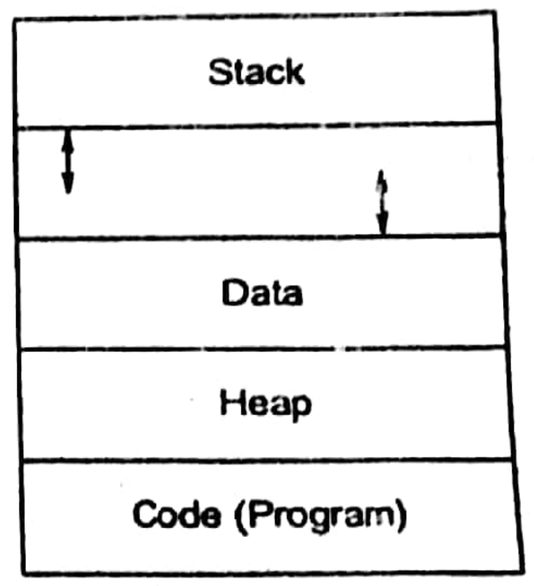
Processes and Threads

2.1 Process

A process is an activity of executing a program. It is a program under execution. Every process needs certain resources to complete its task.

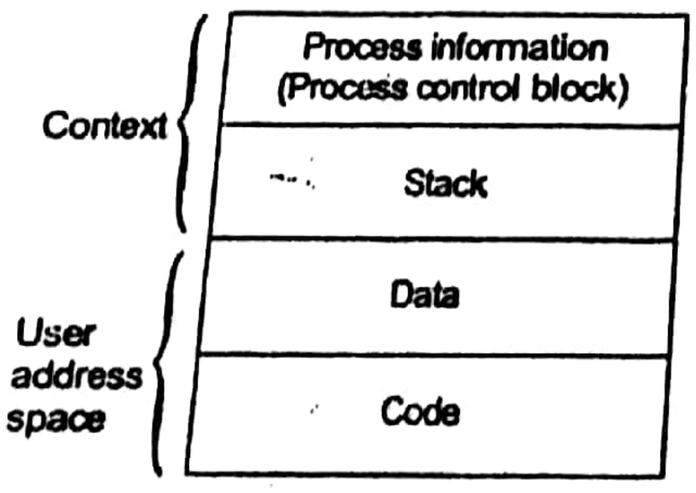
There are two types of processes:

1. **User process:** User processes are executed in user mode and user processes can be preempted while executing.
2. **System process:** System processes are executed in privileged mode. System process executed automatically without preemption.



Abstraction view of a process

2.1.1 Process Description



Every process has an image consists of three components.

1. Executable program [code section]
2. Associated data needed by the program [data section]
3. Execution context of the process [context section]

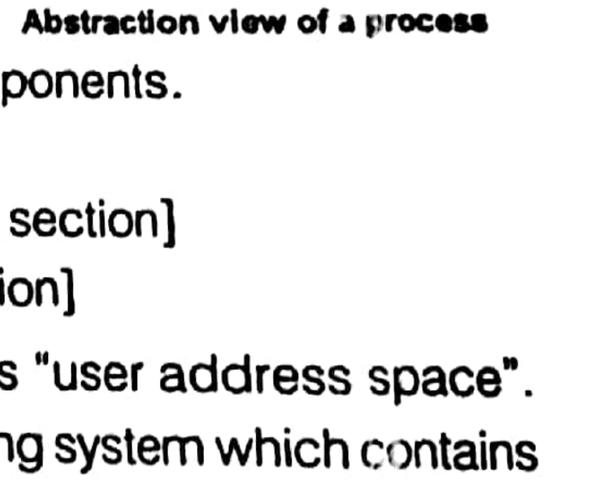
Code and data section of a program are called as "user address space". Context section of a process is managed by operating system which contains stack and process control information.

2.1.2 Process Control Block (PCB)

A process context saved in particular block to control the process is called as "Process Control Block". PCB is also called as task control block. Every process has a PCB that contains the following information.

1. Process Identification Data
2. CPU state information
3. Process control information

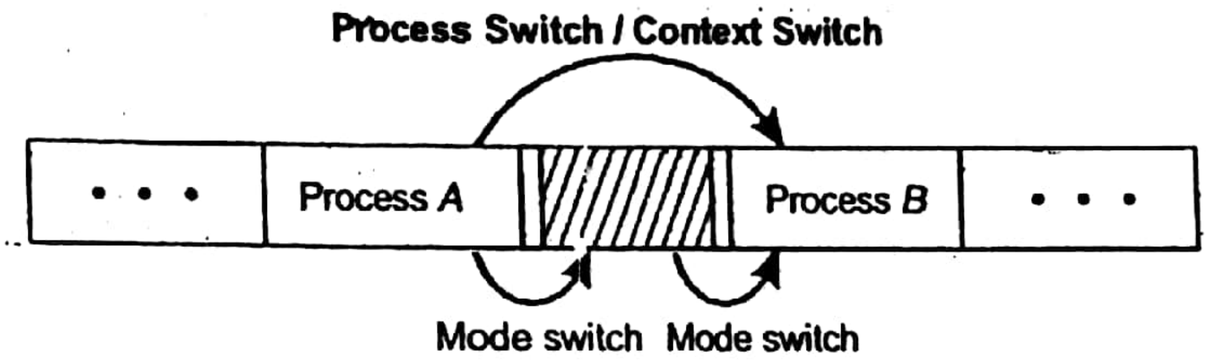
OS maintains a process to manage the processes by maintaining PCB of every process as an entry of process table.



Process Control Block

2.1.3 Process Switch (Context Switch)

1. **Process switch:** Process switch also called as context switch which involves saving the current CPU information, updating the control information and restore the CPU information.
   Process switch includes the following steps:
   (*i*) Save CPU context [Mode switch from user mode to Kernel mode using mode bit].
   (*ii*) Update PCB of current process.
   (*iii*) Move PCB of current process to appropriate queue.
   (*iv*) Select another process for execution [By CPU scheduler].
   (*v*) Update PCB of selected process.
   (*vi*) Update memory management structures.
   (*vii*) Restore CPU context of new PCB [Mode switch from Kernel mode to user mode].



2. Context switch time is dependent on hardware and it is overhead because during context switch the system does not useful work.

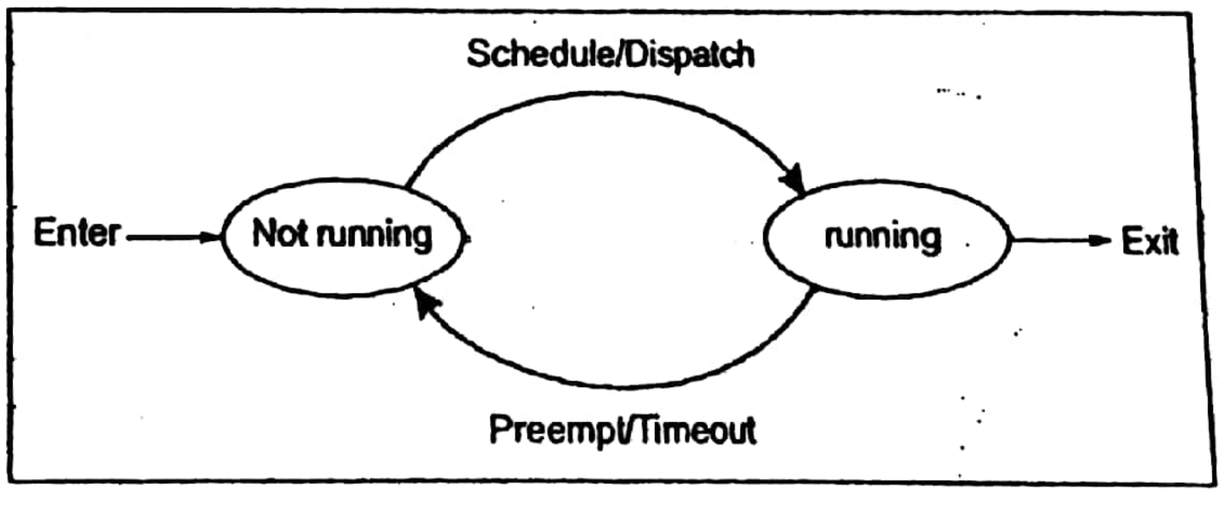
2.2 Process State Models

Process state defines the current activity of the process. Process states are: new, running, ready, blocked, suspended (suspended ready, suspended blocked), etc.

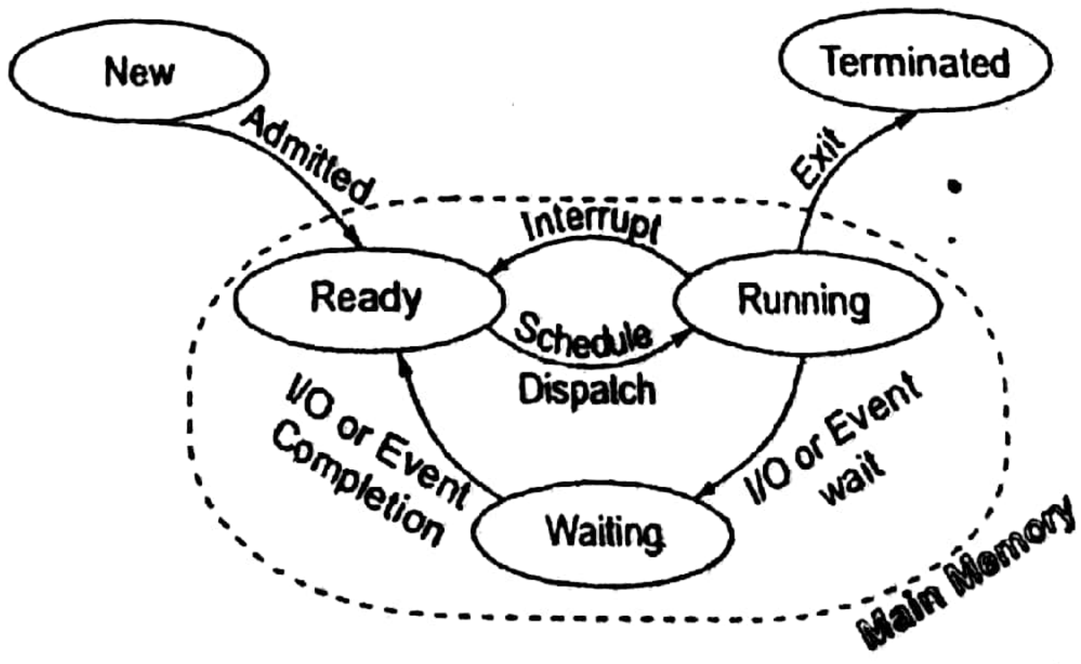
2.2.1 Types of Process State Models

Number of process states are decided by the type of process state model.

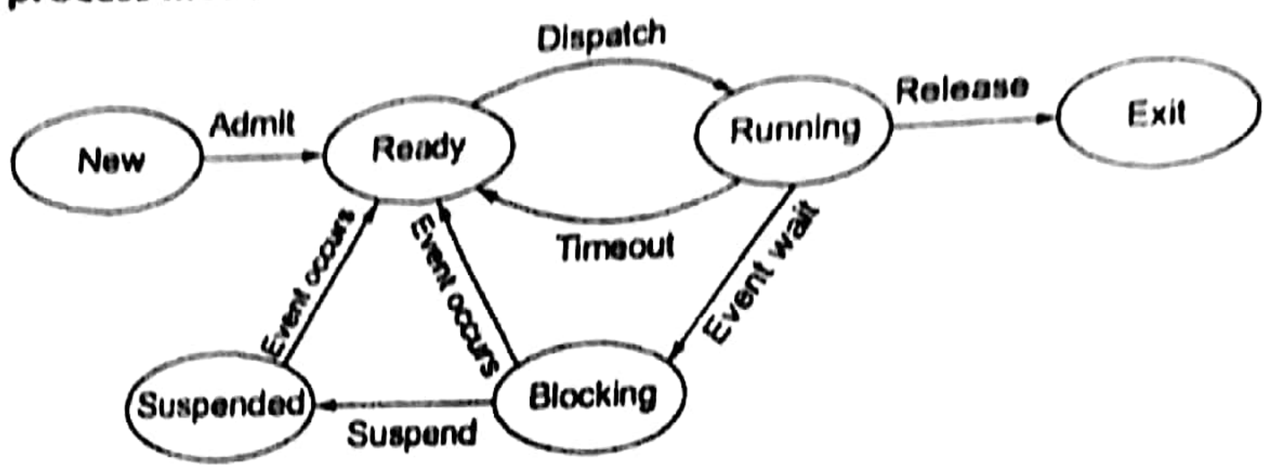
1. Two state process model:



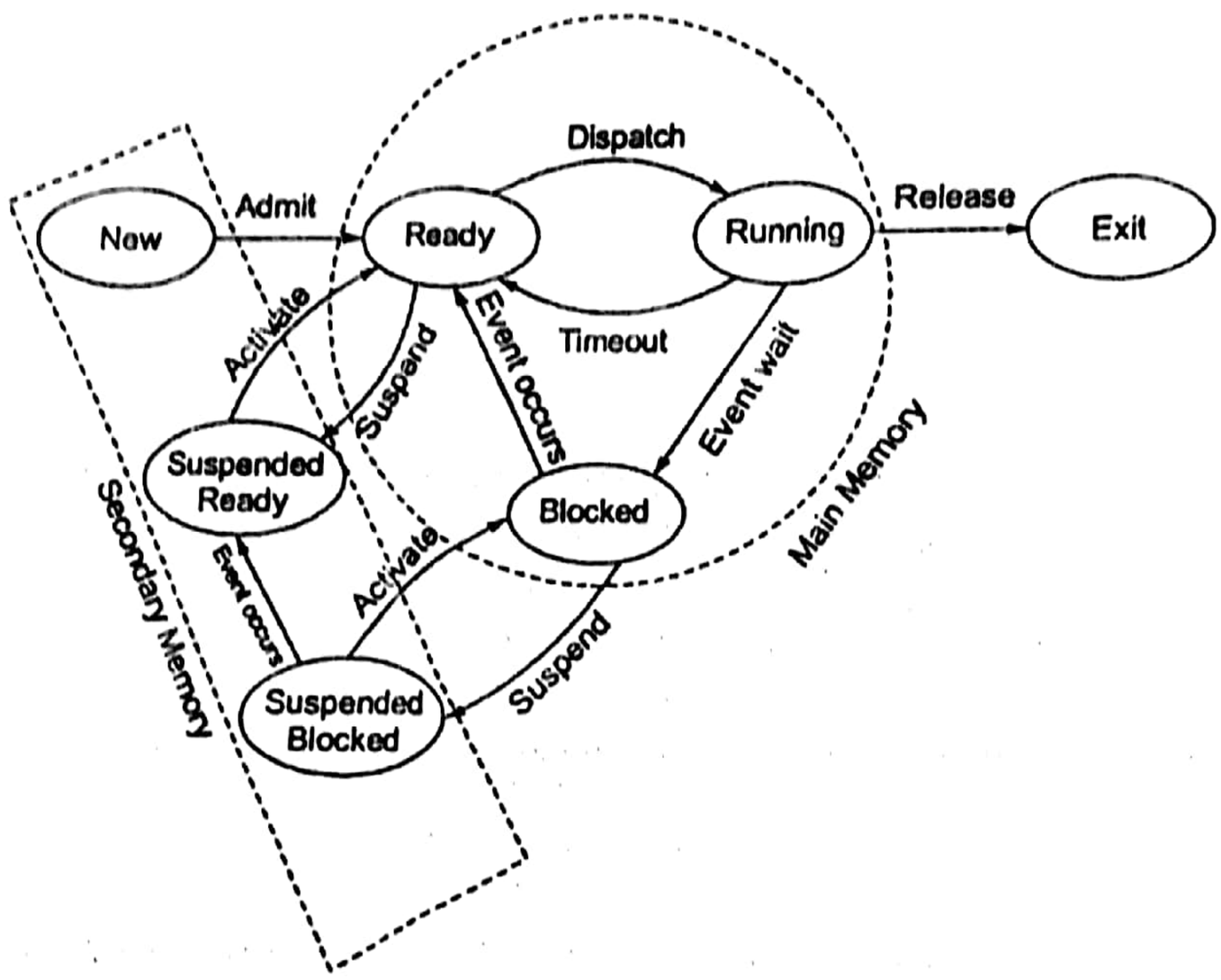
2. Five state process model:



3. **Six state process model:**



4. **Seven state process model:**



2.2.2 Queues / States Description

- *New:* For every new process, first PCB is created and added to a "new" queue.
- *Ready:* When a process is ready, the program and data are loaded, and PCB of that process kept in "Ready" queue. The process is available for execution and available in main memory.
- *Blocked:* The process is in main memory awaiting an event will be in blocked. When event occurs, the process enters into "Ready" queue.
- *Running:* The process currently executing is in running state atmost one process is in running state at any time.
- *Suspended blocked:* The process is in secondary memory and awaiting an event will be in suspended blocked. The process in "Blocked" will move to "Suspended blocked" due to many reasons such as blocked process might be consuming more memory.
- *Suspended ready:* The process is in the secondary memory but is available for execution whenever it is loaded into memory. The process moved from "Ready" queue to "Suspended Ready" queue due to more priority for other processes and several reasons are exist.
- *Exit:* A process which completed its execution will be terminated by swapping out.

2.2.3 Active and Inactive Jobs

The following processes (active) are in main memory.

- Ready
- Running
- Blocked

The following processes (inactive) are in secondary memory.

- New
- Suspended blocked
- Suspended ready
- Exit

2.2.4 Actions Performed by OS

Timeout/Preemption: The process receives a timer interrupt and relinquishes control back to the OS dispatcher. The OS puts the process in "Ready" queue and dispatches another process to the CPU.

Dispatch: A process in "Ready" queue has been chosen to be the next running process.

Event Wait (Block/I/O Wait): A process invokes an I/O system call for an event that blocks the currently executing process. OS puts the process in "Blocked" queue and dispatches another process to the CPU from "Ready" queue.

Event Occurs (Unblock/ I/O Completion): An I/O system sends an interrupt to CPU to inform the completion of its task. OS may then decide to unblock the process which is waiting in blocked "queue" and puts in "Ready" queue.

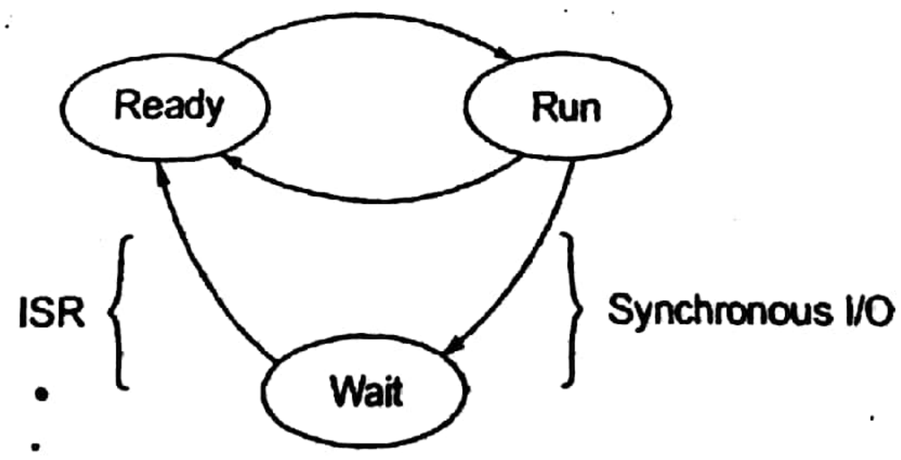
2.2.5 Types of Process

Processes may be categorized as:

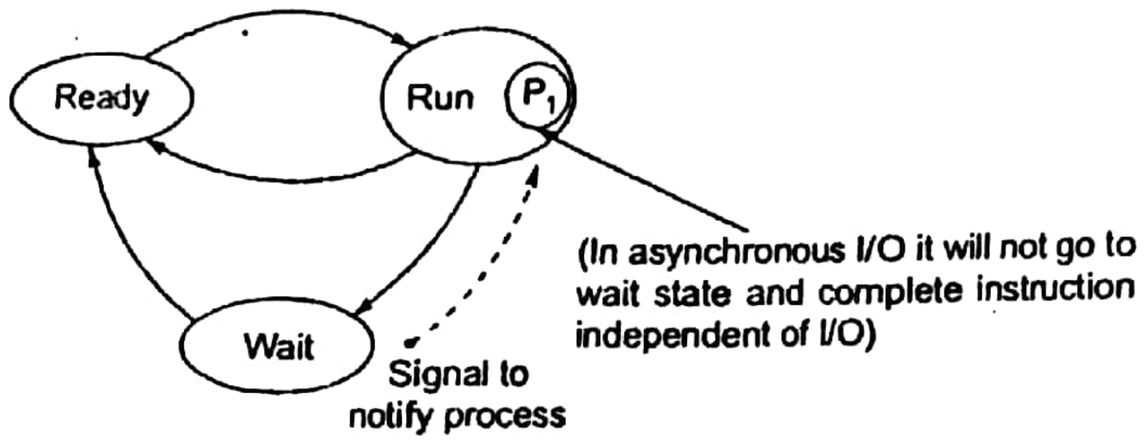
- CPU-bound: Process does not need much I/O service, almost always want the CPU.
- I/O-bound: Short CPU burst times, needs lots of I/O service.
- Interactive: Short CPU burst times, lots of time waiting for user input (keyboard, mouse).

2.2.6 Types of I/O (Syncronous and Asyncronous I/O)

1. **Synchronous I/O:** The process programming I/O operation will be blocked in the block state till I/O operation is completed. Once I/O operations is completed ISR (Interrupt Service Routine) will initiated which places the process from block to ready state.



2. **Asynchronous I/O:** An asynchronous I/O while initiating I/O request a handler function will be registered. The process is not placed in block state it continues to execute remaining code after initiating I/O request. At the point of I/O request is completed the signal mechanism is used to notify the process that data is available and registered handler function is asynchronously invoked. All information about process will be store in handler function like type, what it does etc.?



2.2.7 Process Memory Image

Memory image of a process has following 5 sections which are used while running a process.

- **Text Segment/Test Region:** It stores the machine instructions of stored program and size is fixed.
- **Data Segment:** It stores initialized static and global data of stored program and size is fixed.
- **BSS (Block Stated by Symbol) Segment:** It stores uninitialized static data that is defined in the program (e.g., global uninitialized strings, numbers, structures). Size of BSS is fixed.
- **Heap Segment:** It is dynamically allocated memory. Memory can be allocated and deallocated dynamically at run time.
- **Stack Segment:** It used to maintain the call stack, which holds return addresses, local variables, temporary data, and saved registers. Stack is dynamic.

2.2.8 Important Commands (System Calls)

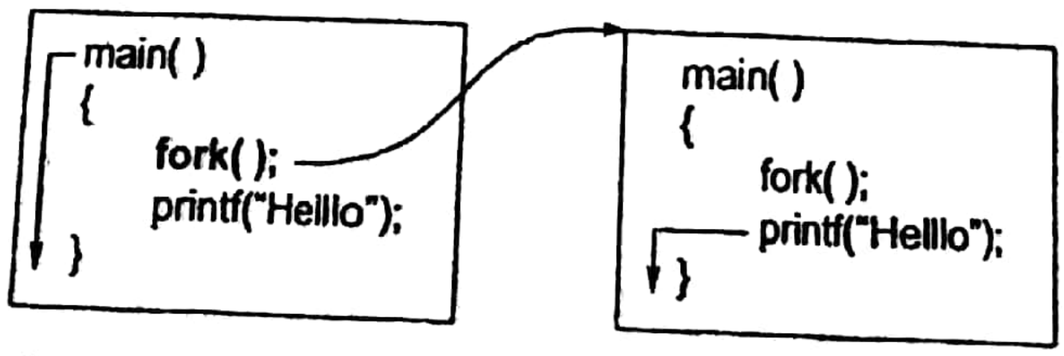
- **Fork:** *Fork* creates a new process. The new process is a copy of the parent. It's running the same program and has the same open files. It is, however, a copy and not a reference to the parent. Any memory modifications or changes to open file status will be invisible to the parent and vice versa. fork system call : (i) Returns –ve value if process creation is unsuccessful, (ii) Returns +ve value if process creation is sucecesful, (iii) Returns 0 to newly created process.
  The parent and child process have same virtual address but physical address will be different.

NOTE: For N fork statement $2^N - 1$ child process will be created.

Example - 2.1 Find the number of child processes created for the following code and also find how many times "Hello" is printed.

```
main( )
{
    fork( );
    printf("Hello");
}
```

Solution:

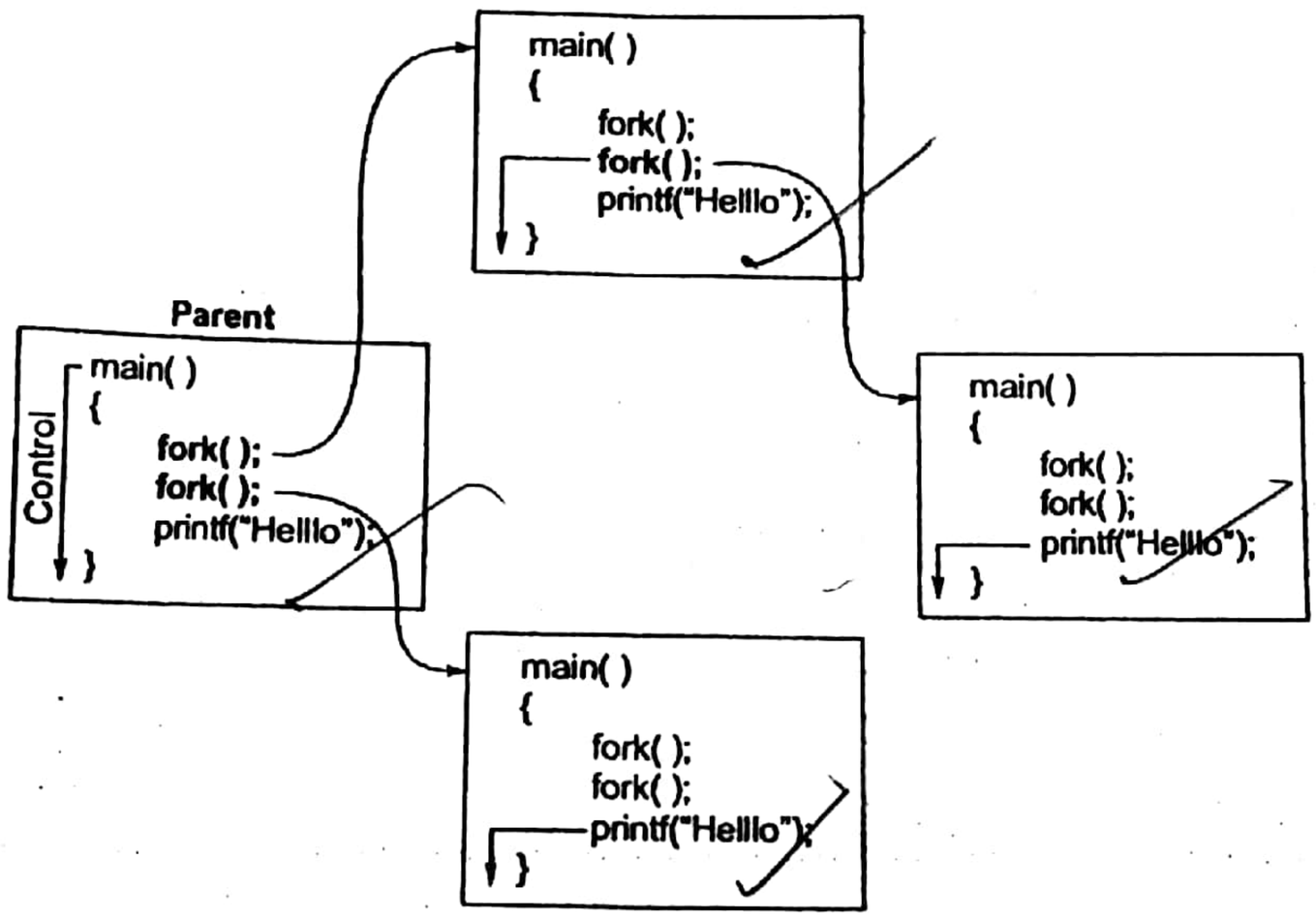


Only one child process is created. "Hello" is printed twice for the given code.

Example - 2.2 Find the number of child processes created for the following code and also find how many times "Hello" is printed.

```
main( )
{
    fork( );
    fork( );
    printf("Hello");
}
```

Solution:



Three child processes are created. "Hello" is printed four times.

- **execve:** *execve* does not create a new process. It loads a new program from the file system to overwrite the current process, reinitializing the process' memory map. It is often used immediately after *fork* to allow have the child process created by *fork* to load and run a new program.
- **exit:** *exit* tells the operating system to terminate the current process.
- **wait:** *wait* allows a parent to wait for and detect the termination of child processes.
- **signal:** *signal* allows a process to detect signals, which include software exceptions, user-defined signals, and death of child signals.
- **kill:** *kill* sends a signal to a specified process. The signal can usually be detected with *signal*. It does not kill the process unless the signal is a kill (or terminate) signal.

Event	Starting State	Ending State
Program started by user	Does not exist	Running
Scheduler dispatches process	Ready	Ready
I/O complete	Waiting	Ready
Process admitted to ready queue for first time	New	Ready
Scheduler preemtps (interrupts) process	Running	Ready
Process finishes execution (task is complete)	Running	Terminated
Process initiates I/O	Running	Waiting

Events of a Process

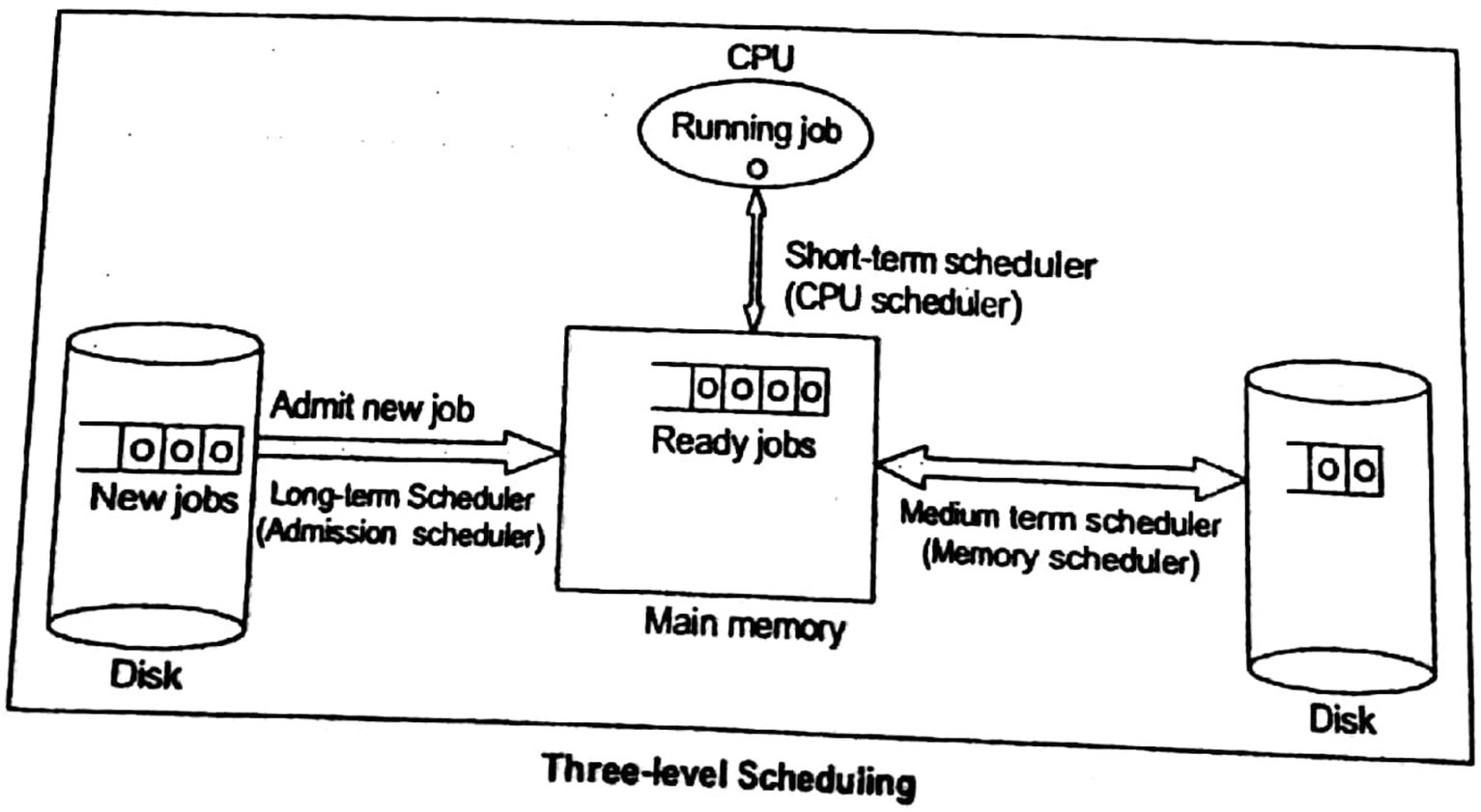
2.3 Operations on a Process

1. **Process creation:** The following events lead to the process creation.
   (a) System initialization (system booting) creates several background processes (email, logon, etc).
   (b) A user requests to create a new process.
   (c) Already running to process (existing) can create a new process.
   (d) Batch system takes initiation of a batch job
2. **Process termination:** The following events lead to the process termination
   (a) *Process-triggered:* When a process completes its execution it executes "exit" system call to indicate to the OS that it has finished.
   (b) *OS-triggered:* "Service errors" like no memory left for allocation and I/O errors "Preemption" of a process when total time limit exceeded. In both the cases the process can be terminated, called as "total errors".
   (c) *Hardware Interrupt triggered:* Arithmetic errors, out of bounds memory access, etc. are program bugs which terminates a running process.
   (d) *Software interrupt triggered:* A process can be terminated by another process by executing system call to kill the process that informs OS.
3. **Process blocking:** When a process invokes an I/O system call that blocks the process and OS put this process in block mode (waiting for I/O) and dispatches another process to CPU.
4. **Process preemption:** When a timeout occurs, the process receives a timer interrupt and relinquishes the control back to OS dispatcher. OS puts the current process in "Ready mode" and dispatches another process to CPU.

2.4 Scheduling

Scheduling depends on three scheduler in the system

1. Long-term scheduler
2. Medium-term scheduler
3. Short-term scheduler



Three-level Scheduling

2.4.1 Long-term Scheduler (LTS)

- It involves in a decision to add a job to the pool of processes to be executed.
- It controls the degree of programming by taking the decision that the number of processes can be ready for execution by keeping in ready queue (in main memory).