Operating Systems
Course Code: MCC510

Operating System:

- Program or software → Hardware
- Act as an intermediary → User and Hardware
- Resource Allocator → Available Resources
- Deals with Conflict of Interest → Efficient use of resources
- Control Program → Execution of Program

Goal and Role of OS

Goal of Operating System:

- Execute User Program
- Solve the user Program
- Convenient use of System
- Use of Computer Hardware

Role of Operating System:

- User Point of View
- System Point of View

Software

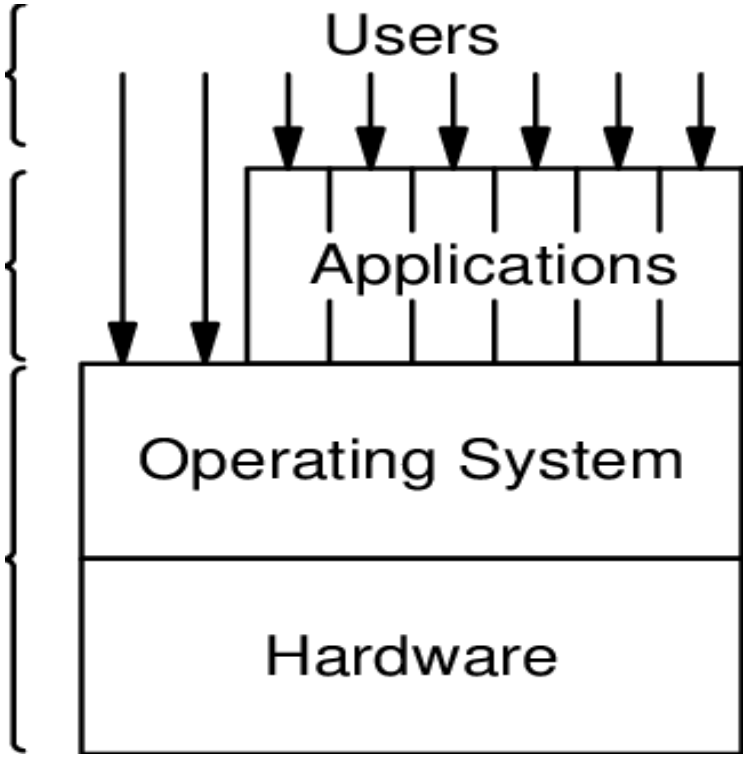
System Software:

- Includes OS, Compilers and all utilities
- Consists low level Program
- Example-UNIX, Window etc.

Application Software:

- Include Program
- Example-Word Processor, Spread sheet, Data Base Management Systems etc.

Computer System Structure



Kernel

Kernel:

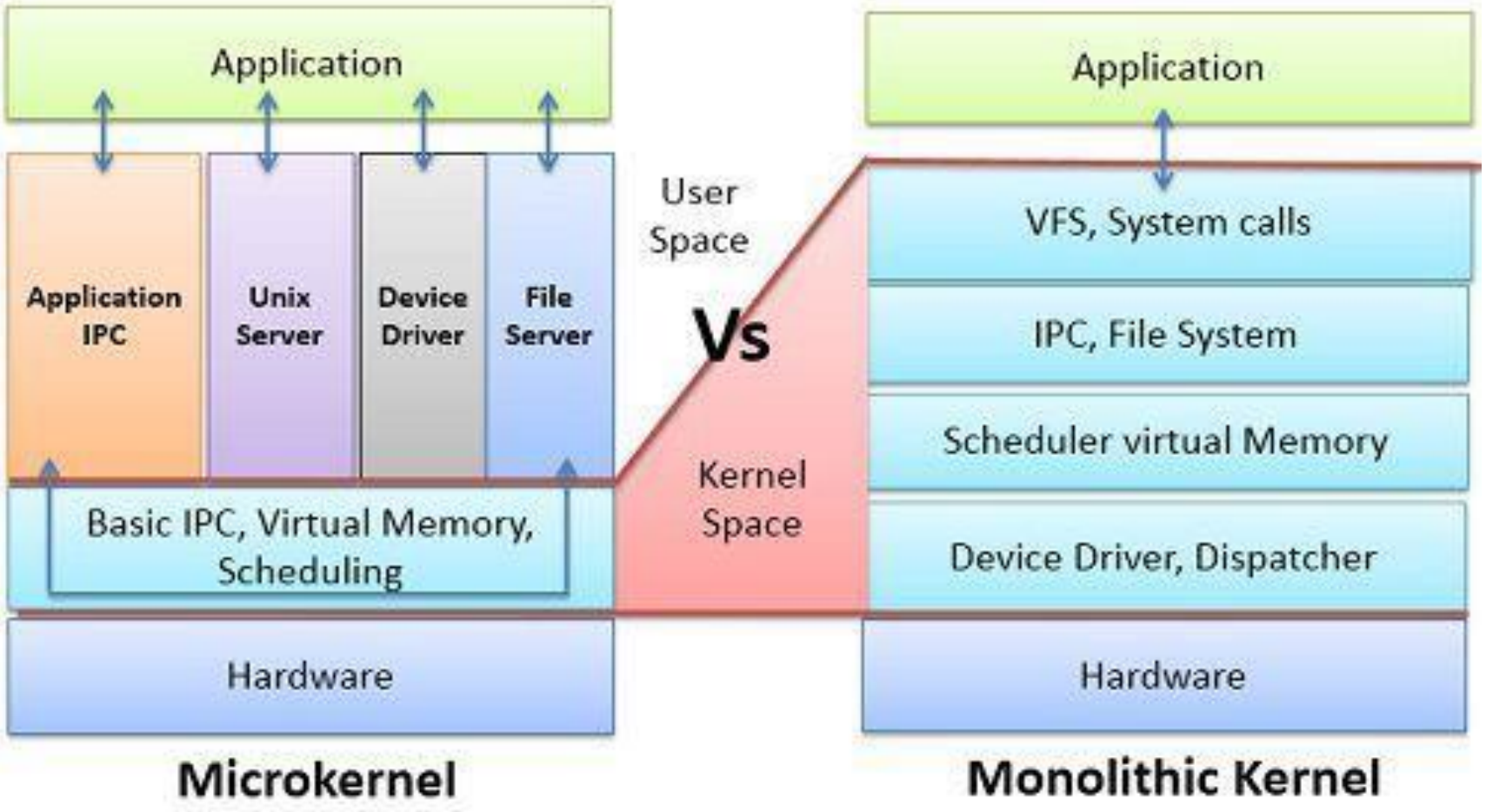
- One program running all times on the computer is known as **Kernel** which connects the application software to the hardware of computer.

Types:

- Monolithic Kernel- [User and Kernel service lies in same address space, Size is larger, Using KS, System Call is used, Not extendable.] e.g., Microsoft Windows, Linux, BSD (OpenBSD, NetBSD, FreeBSD), Solaris, DOS, OpenVMS, etc.

- Micro Kernel- [User and Kernel service lies in different address space, Size is smaller, Using KS, Message passing is used, Easily extendable.] e.g., QNX, minix, Symbian, Mac OS X, L4Linux, Integrity, K42, etc.

Difference between Micro and Monolithic Kernel



Kernel

Advantages Monolithic kernel :

- Simple to design and implement
- Simplicity provides speed on simple hardware
- It can be expanded using a module system
- Time tested and design well known

Disadvantages Monolithic kernel :

- Runtime loading and unloading are not possible because of the module system
- If code base size increases, maintain is difficult
- Fault tolerance is low.

Kernel

Advantages Microkernel :

- Allows the addition of new services
- Architecture design provides a uniform interface on requests
- Architecture support flexibility and the object-oriented operating system
- Modular design helps to Stance reliability.
- Microkernel fends itself to distributed system support

Disadvantages Micro kernel :

- Providing services are expensive
- Context switch or a function call needed when the drivers are implemented as procedures or processes, respectively
- The performance can be indifferent and may lead to some problems

Booting and Bootstrap

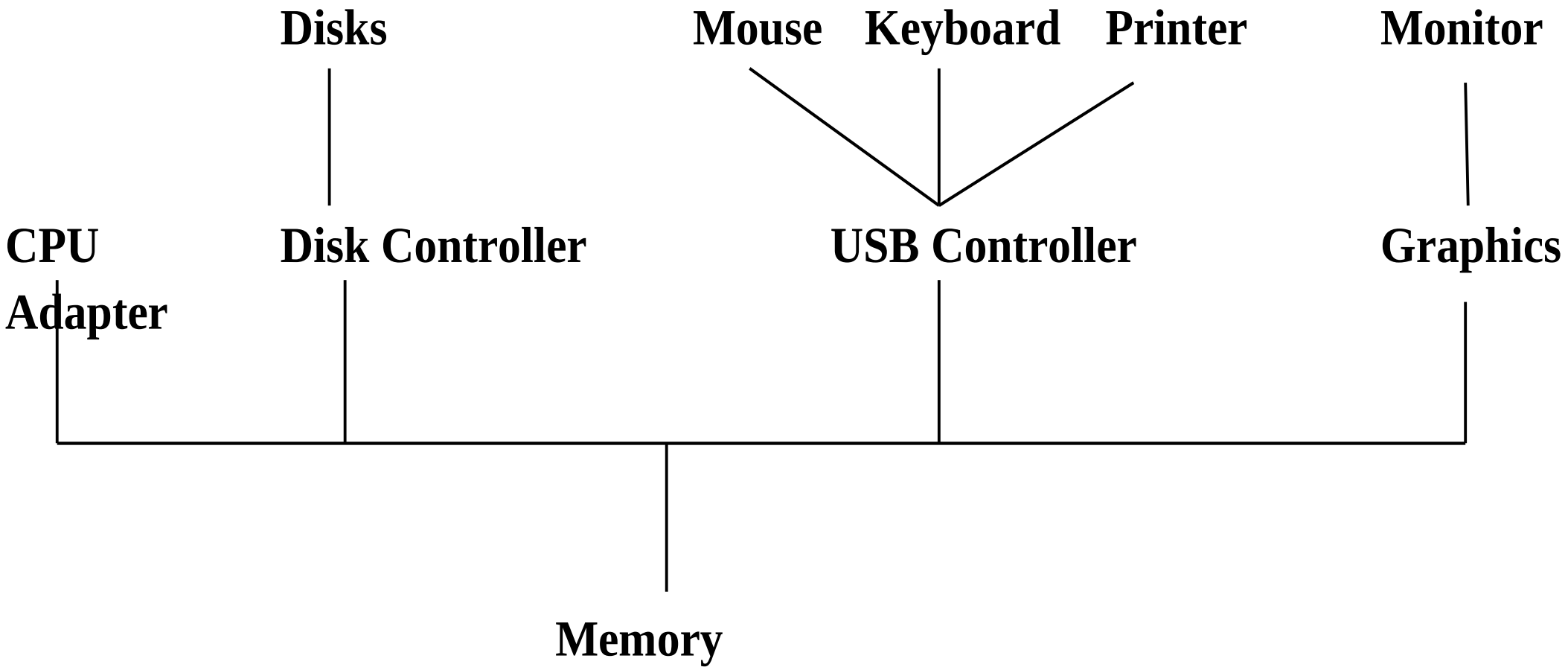
Booting:

- The process of loading an OS software from secondary memory i.e., hard disk into primary memory i.e., RAM(Random Access Memory)is known as booting.

Bootstrap Program:

- It is loaded at power up or reboot which typically stored in ROM(Read only memory) or EPROM(Erasable programmable read only memory) generally known as firmware (within the computer hardware). Initialize all aspect of the system and loads OS kernel and start execution.

Computer System Architecture



Interrupt

Interrupt:

- The occurrence of an event is usually signaled by an interrupt from either by hardware or software. Hardware may trigger an interrupt at any time by sending a signal to the CPU usually by the way of system bus.
- However, software may trigger an interrupt by executing a special operation known as system call.
- If CPU is interrupted then it stops what it is doing and immediately transfer execution to a fixed location.

Interrupt

Common Function of Interrupt :

- Interrupt transfer control to the interrupt service routine generally through the interrupt vector which contains the addresses of all service routines.
- Interrupt architecture must save the address of interrupted instruction.
- Incoming interrupt are disabled while another interrupt is processed to prevent a lost of interrupt.
- A trap is a software generated interrupt caused either by an error or by user request
- An OS is interrupt driven that means there is no process to execute, no I/O devices to service, no user to whom to respond and OS simply waiting for something to happen.

Interrupt

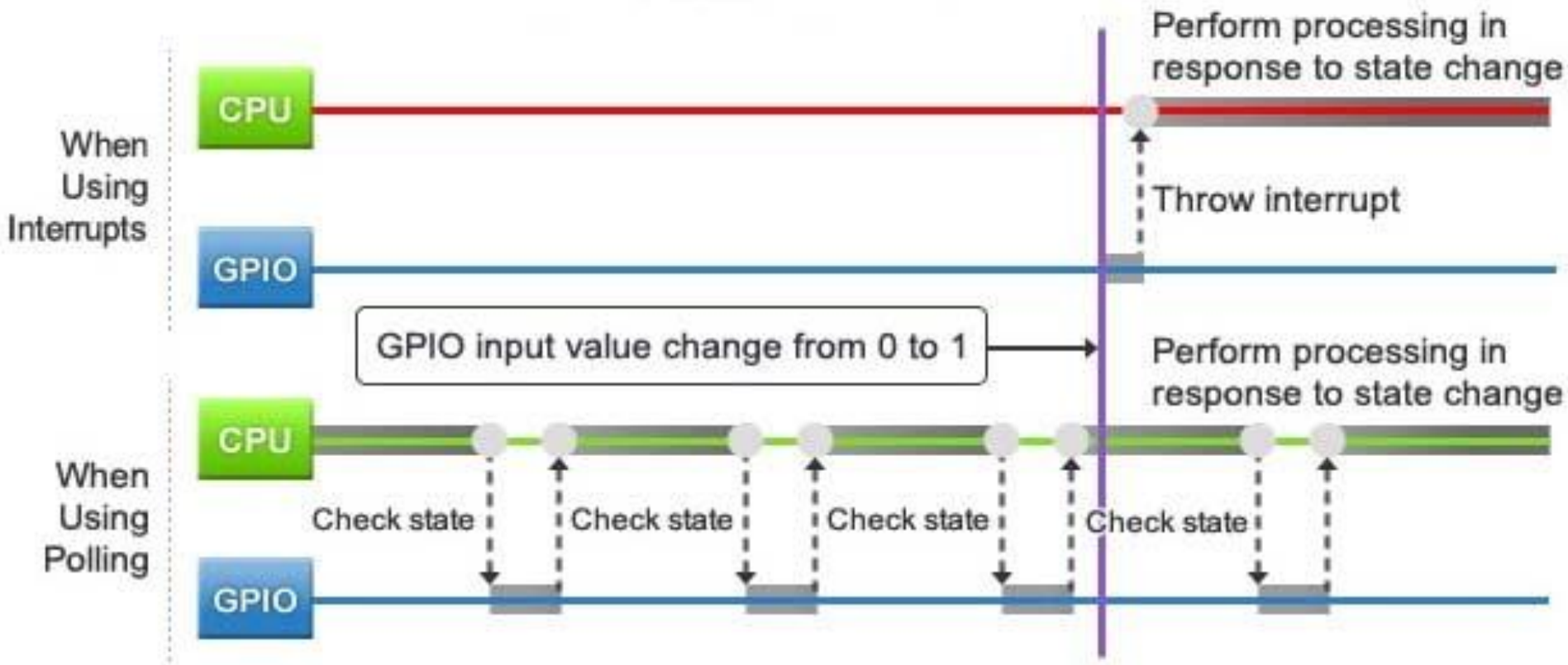
Interrupt Handling:

- The OS preserve the state of CPU by storing registers and the program counter.
- Determine which type of interrupt has occurred (Polling or Vector interrupt system).
- Separate segment of code which determines that what type of action should be taken for each type of interrupt.

Polling:

- A polling based program is non interrupt driven continuously polls or tests whether or not data are ready to be received or transmitted. This scheme is less efficient than the interrupt scheme.

Interrupt/Polling



Category of OS

Multiuser:

- It allows two or more users to run program at the same time. The OS of mainframe and minicomputer are multiuser system. e.g., Multiple Virtual Storage (MVS), UNIX. It is also a time sharing system.

Multiprocessing:

- It refers to a computer system ability to support more than one process at same time. It is also known as parallel processing. E.g., Windows NT, 2000, XP, and Unix.
- It is needed for efficiency such as Single user can not keep CPU busy all the times; Organizes jobs (Code and Data) so that CPU always have one job to execute; Subset of total job in a system kept in memory; One job selected and run via job scheduling; If it has to wait (i.e., I/O) then OS switches to another job

Category of OS

Multitasking:

- It allows to run more than one program concurrently. It is the ability to execute more than one task at the same time. The term multitasking and multiprocessing are often used interchangeably. It may be preemptive or cooperative. It is also known as time sharing which is logical extension in which CPU switches jobs so frequently that users can interact with each job while it is running, **creating interactive computing**. e.g., Microsoft Windows 2000, IBM's **OS**/390, and Linux
- Response time < 1 sec.
- Each user has at least one job executing in memory → Process
- If several job ready to run at the same time → CPU Scheduling
- If process do not fit into memory then Swapping moves them in and out to run.
- Virtual memory allows execution of processes not completely in memory.

Multithreading:

- It allows different parts of a single program to run concurrently. It is an ability of an OS to execute different parts of program called threads simultaneously. e.g., Solaris

Real Time:

- Real time OS is a system that respond to input immediately. e.g., LynxOS, QNX, RTAI, RTLinux, Symbian *OS*, VxWorks, Windows CE, Monta Vista Linux.

OS Operations

- Interrupt driven by hardware
- Software error or request creates exception or trap
- Other process problem include infinite loop
- Processes modifying each other or OS
- Dual mode operation allows OS to protect itself and other system components such as User mode and kernel mode.
- Mode bit provided by hardware that provide ability to distinguish when the system is running in user mode or kernel mode.
- Some instructions designated as privileged only executable in kernel mode
- System call changes mode to kernel and return from call resets it to user.

Protection and Security

Protection:

- It is a mechanism for controlling access of processes or users to resources defined by OS

Security:

- It is the defense of the system against internal and external attacks huge range including denial of service, worm, viruses, identity theft, theft of services etc.
- System generally distinguish among users to find who can do what
- User identities-User ID Security ID include name and associated number one per user
- User ID then associated with files, processes of that user to find access of control
- Group Identifier (Group ID) allows set of users to be defined and controls managed then also associated with each process and files.
- Privilege escalation allows user to change to effective ID with additional rights

OS Services

One set of OS services provide functions that are helpful to users:

- User Interface(UI)-Almost all OS have a UI varies between CLI,GUI and Batch
- Program Execution-The system must be able to load a program in to memory and to run that program, end execution either normally or abnormally indicating error
- File system manipulation-The file system is of particular interest. Program needs to read and write files and directories, create and delete them, search them, list file information, permission and management
- Communications-Processes may exchange information on the same computer or between computers over networks. Communication may be made via message passing or shared memory.
- Error detection-OS needs to be constantly aware about possible errors

OS Services

Another set of OS functions exists for efficient operation of the system itself via Resource sharing:

- Resource Allocation-If multiple users or jobs running concurrently then resources must be allocated to each of them. Many type of resources such as CPU cycles, Main memory and file storage may have special allocation code however I/O devices may have general request and release code

- Accounting-To keep track of users regarding use of type of computer resources

- Protection and security-Protection involves ensuring that all access to system resources is controlled. However, security of the system from outsiders requires user authentication, extend to defend external I/O devices

OS Interface

Command Line Interface (CLI):

- It allows direct command entry sometimes implemented in kernel and sometimes by system program and even sometime multiple flavour implemented such as Shells

- Initially fetches a command from user and execute it. Sometimes command built in and sometimes just name of the programs. If we add some new features then it do not require shell modification.

OS Interface

Graphical User Interface (GUI):

- It is a user friendly desktop metaphor interface which consists usually mouse, key board and monitor. Icons represent files, programs, actions etc.
- The various mouse buttons over objects in the interface cause various actions which provide information, options, execute function, open directory known as folder.

Note:

- Many systems now include both CLI and GUI interfaces.
- Microsoft windows is GUI with CLI command shell.
- Apple Mac OS X as aqua GUI interface with UNIX kernel underneath and shells available
- Solaris is CLI with optional GUI interfaces such as JAVA desktop

System calls:

- It is a programming interface to the services provided by OS and typically written in high level language(C or C++).It is mostly assessed by programs via high level Application Program Interface (API) rather than direct system call use. For example: Three most common APIs are Win 32 API for windows, POSIX API for POSIX based systems including virtually all versions of UNIX,LINUX and Mac OS X and JAVA API for the JAVA virtual machin(JVM).

Example of System Call:

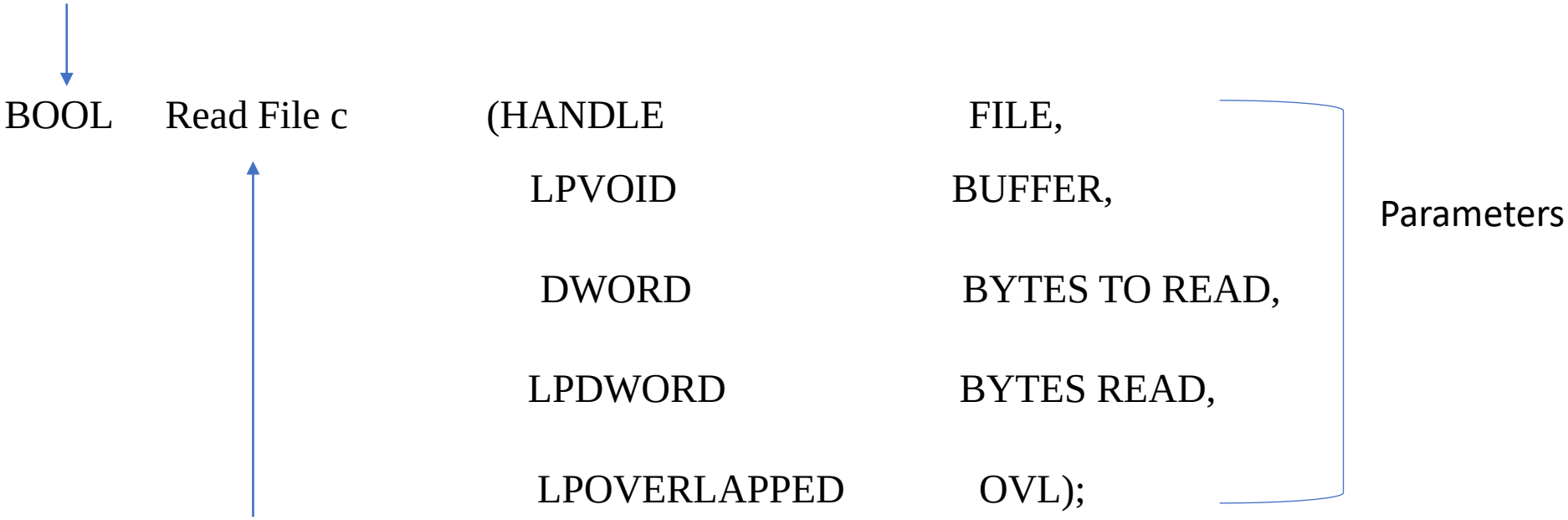
System calls sequence to copy the contents of one file to another file. Source to Destination

System Call Sequence
Acquire input file name
Write prompt to screen
Accept input
Acquire output file name
Write prompt to screen
Accept input
Open the input file
If file does not exist, abort
Create output file
If file exist, abort
Loop
Read from input file
Write to output file
Until read fails
Close output file
Write completion message to screen
Terminate normally

Example of Standard API:

Consider the Read File () function in the Win 32 API- a function for reading from a file

Return value

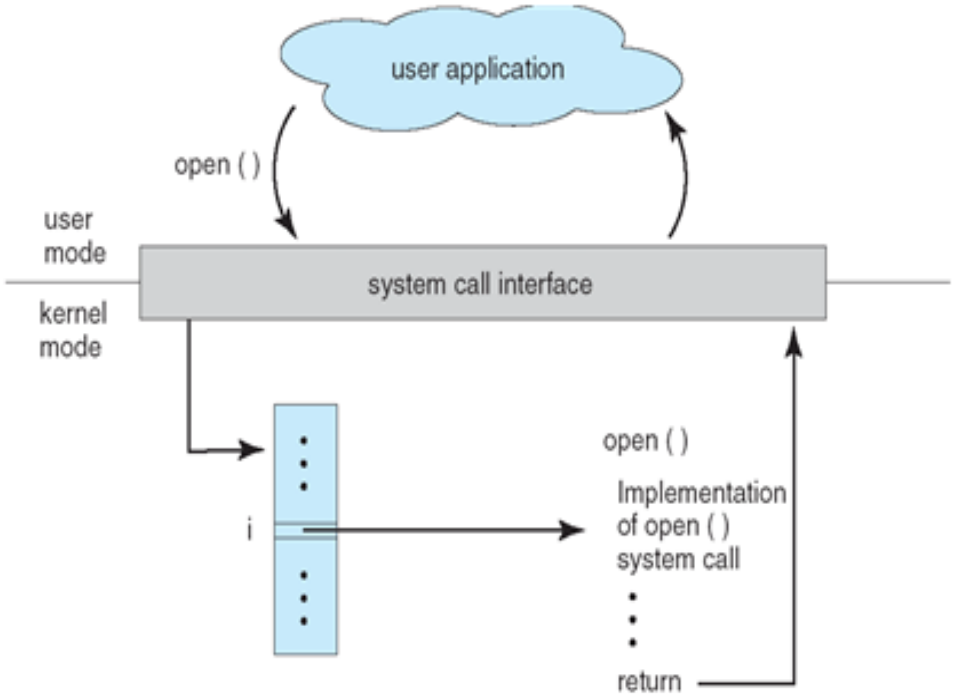


HANDLE file - file to be read
LPVOID buffer - a buffer where the data will be read in to and written from
DWORD bytes to read - the number of bytes to be read in to the buffer
LPDWORD bytes to read - the number of bytes read during the last read
LPOVERLAPPED ovl - indicates if overlapped I/O is being used.

System Call Implementation:

- Normally, a number associated with each system call.
- System call interface maintains a table indexed according to these numbers.
- System call interface invokes intended system call in OS kernel and returns status of the system call and any return values.
- The caller need know nothing about how the system call is implemented.
- Just need to obey the API and understand what OS will do as a result call.
- Most details of OS interface hidden from programmer by API managed by run type support library (Set of functions built into libraries included with compiler).

API-System Call-OS Relationship:



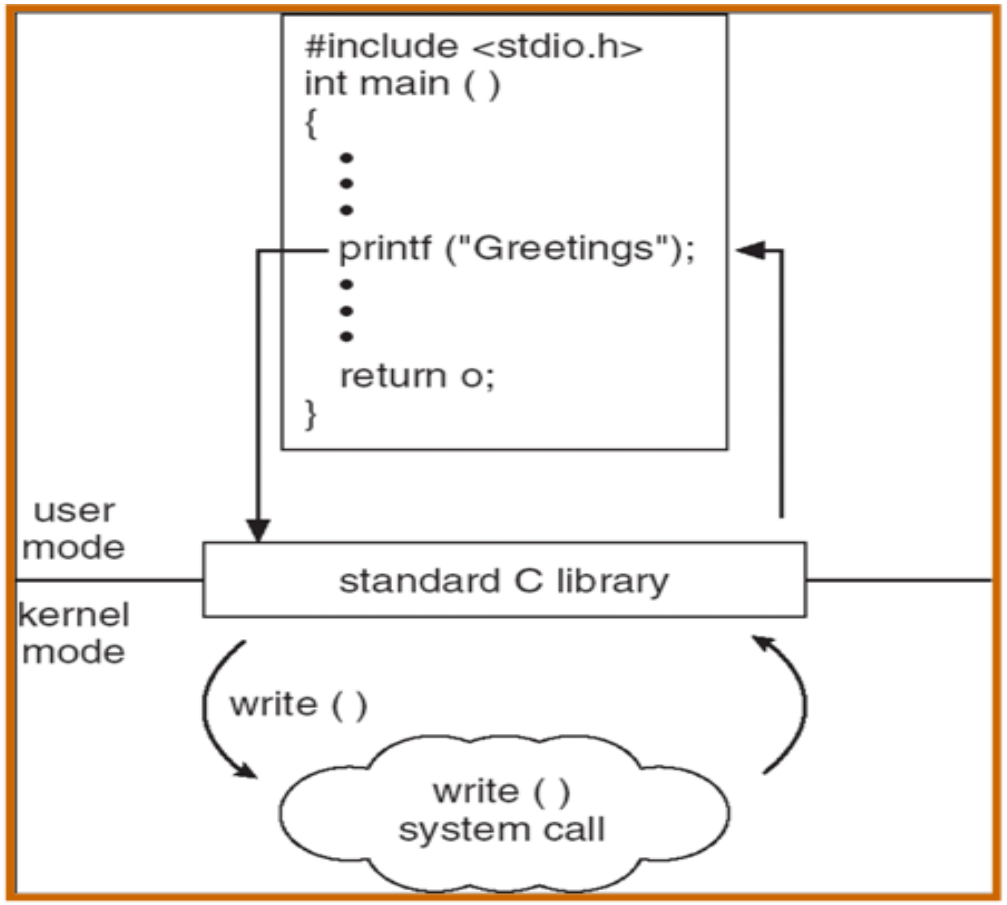
The interface to the services provided by the OS has two parts:

I. Higher language interface -a part of a system library
   - Executes in user mode
   - Implemented to accept a standard procedure call
   - Traps to the Part II

II. Kernel part
   - Executes in system mode
   - Implements the system service
   - May cause blocking the caller (forcing it to wait)
   - After completion returns back to user
   - (report the success or failure of the call)

Example of Standard C Library :

C program invoking printf() library call, which calls write() system call



Types of System Call:

- Process Control- end, abort, load, execute, create and terminate process, get set process attributes, wait for time, wait event , signal event, allocate and free memory.
- File management- create, delete file, open, closed, read write, reposition, set get process attribute.
- Device Management- request, release device, read write, reposition, set get process attribute.
- Information Maintenance-get/set time date, system data, process file or device attribute.
- Communications-create, delete communication, send receive messages, transfer status information, attach/detach, remove device.

System Call Parameter Passing:

- More information is required than simply identity of desired system call.
- Exact type and amount of information vary according to OS and call.
- Three general methods used to pass parameters to the OS

1. Simplest: pass the parameters in registers. In some cases may be more parameters than register.
2. Parameters stored in block, or table, in memory and address of block passed as a parameter in a register. This approach is taken by Linux and Solaris
3. Parameters placed or pushed on to the stack (It contains temporary data such as function parameters, return addresses and local variables) by the program and popped off (To take out) the stack by the operating system.

Block and stack methods do not limit the number or length of parameters being passed so some OS will prefer bock and stack method.

Stack- 1st out and Last in

Queue-1st in 1st out

System Program:

- It provides a convenient environment for program development and execution which can be divided into the following:

File manipulation

Status information

File modification

Programming language support

Programming loading and execution

Communications

Application Program:

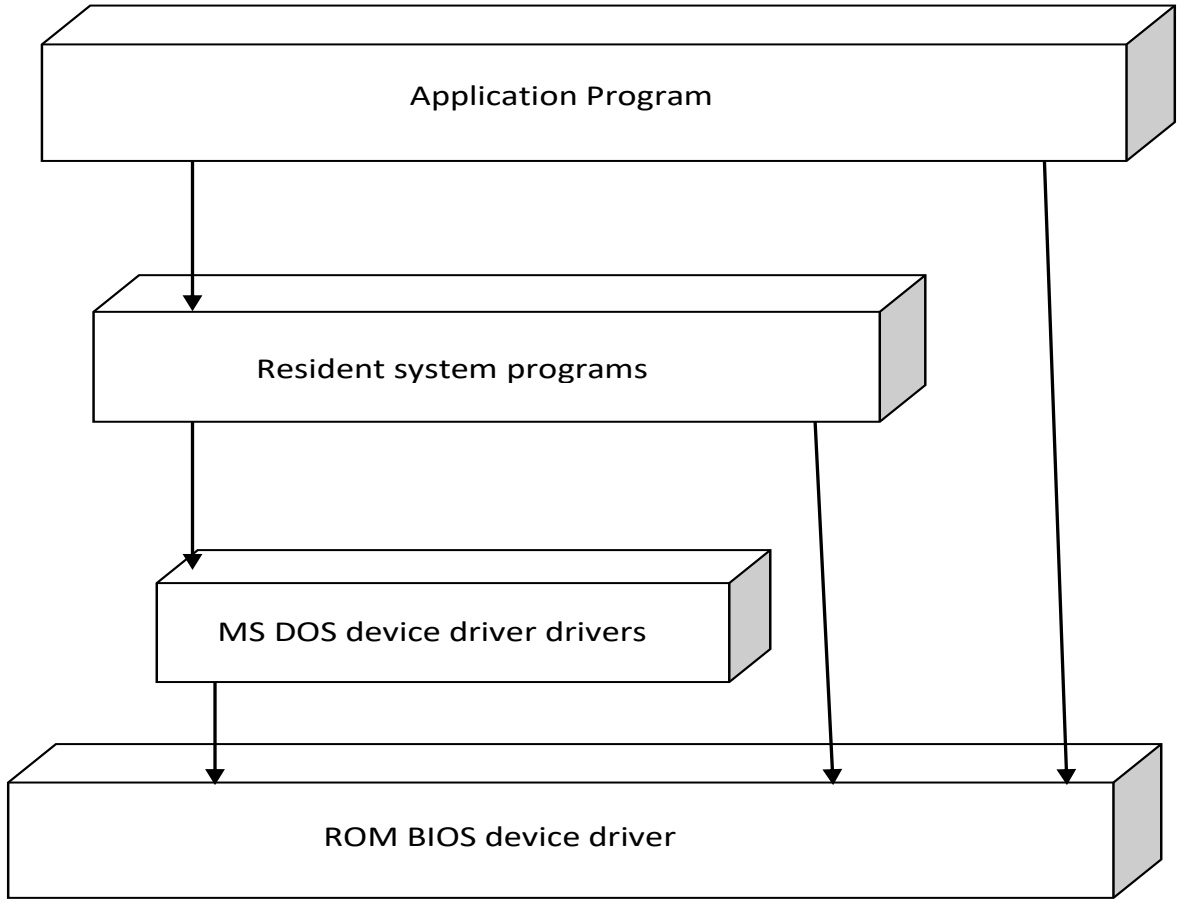
- Most users view of the operation system is defined by system program not the actual system calls.
- Provides a convenient environment for program development and execution. Some of them are simply user interfaces to system calls and others are considerably more complex.
- File management-create, delete, copy, rename, print, dump, list, and generally manipulate files and directories.
- Status Information: Some ask the system for information like date, time, amount of available memory, disk space, and number of users others provide detailed performance, logging and debugging information. These programs format and print the output to the terminal or other output devices. Some systems implement a registry used to store and retrieve configuration information
- File Modification: Text editors to create and modify files, Special commands to search contents of files or perform transformation of the text
- Programming Language support: Compilers, assemblers, debuggers and interpreters sometimes provided.
- Program loading execution: Absolute loaders, relocatable loaders, linkage editors and overlay loaders, debugging systems for high level and machine language required.
- Communications: Provide the mechanism for creating virtual connections among processes, users, and computer systems. Allows users to send messages to one anothers screens, browse web pages, send electronic mail messages, log in remotely, transfer files from one machine to another.

Operating Systems Design and Implementation:

- Design and implementation of OS not solvable but some approaches have proven successful.
- Internal structure of different OS can vary widely
- Start by defining goals and specifications
- Affected by choice of hardware and type of system
- **User Goals**-OS should be convenient to use, easy to learn, reliable, safe and fast
- **System goals**- OS should be easy to design, implement and maintain as well as flexible, reliable, error free and efficient
- Important principle to separate Policy- decide what will be done and mechanism- find the way to do something The separation of policy from mechanism is a very important principle which allows maximum flexibility if policy decisions are to be changed later.

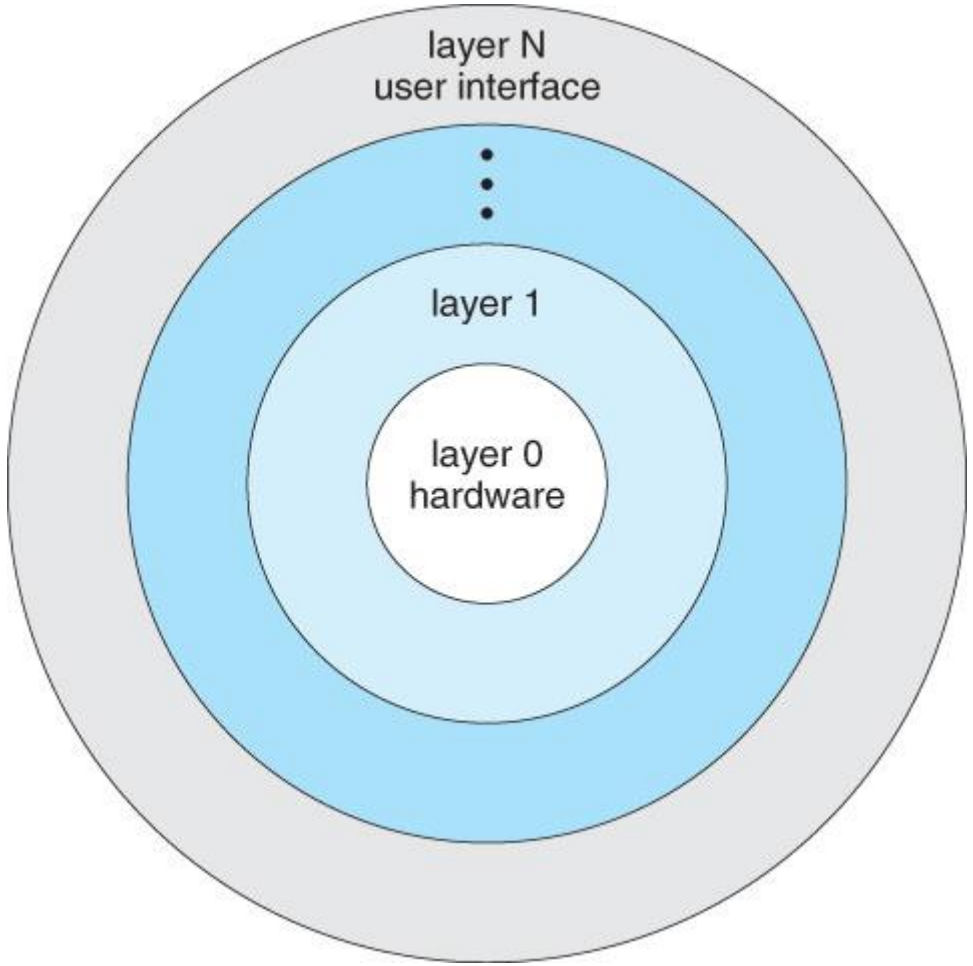
Simple Structure:

- MS DOS - It is written to provide the most functionality in the least space not divided into modules although MSDOS has some structure in which interfaces and levels of functionality are not well separated.



Simple Structure:

- **Layered Structure** – OS is divided into number of layers each built on top of the lower layer. The bottom layer is known as hardware and the highest layer is known as user Interface.



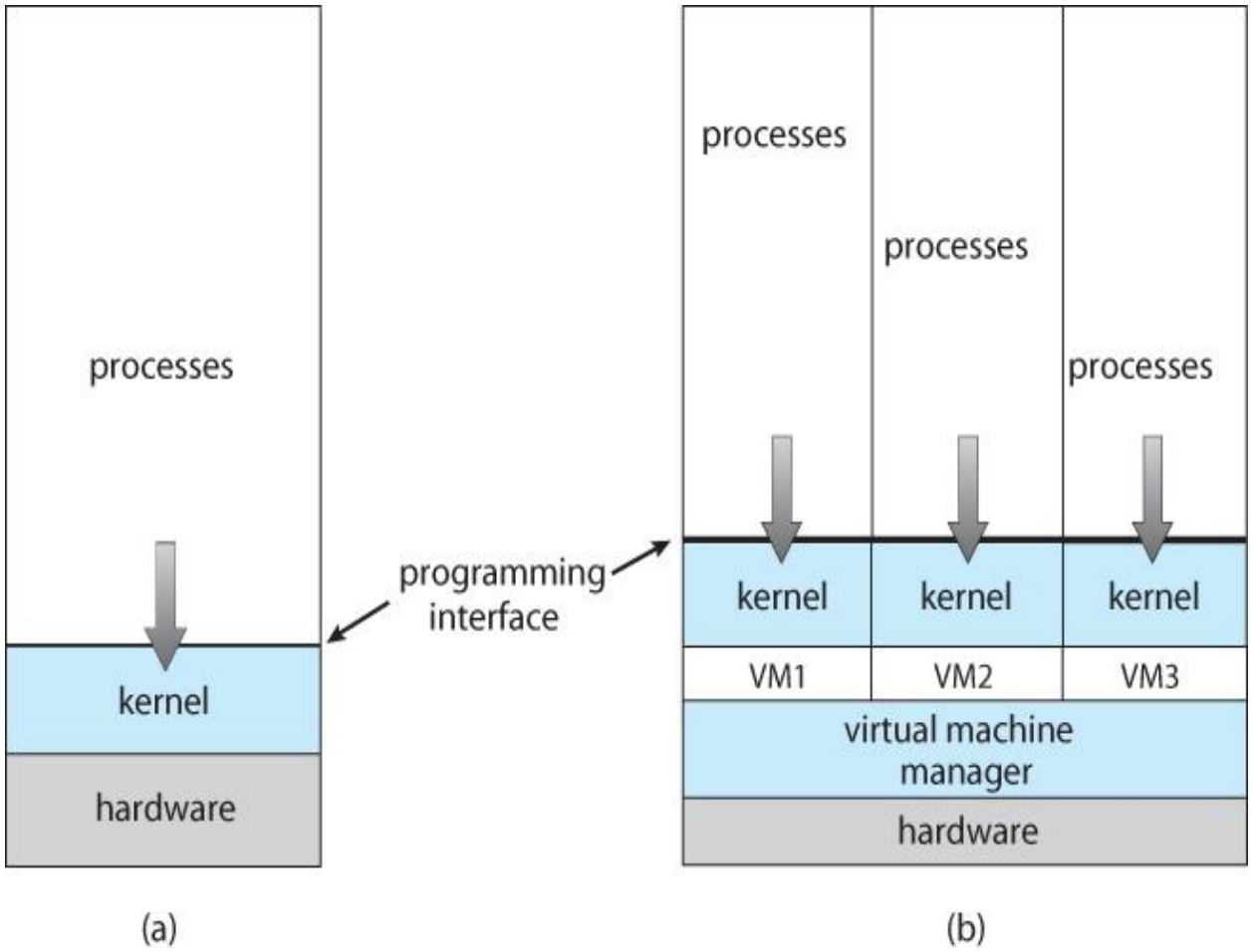
System Generation (SYSGEN):

- OS are designed to run many class of machines and the system must be configured for each specific computer site.
- SYSGEN program obtained information regarding the specific configuration of hardware system.
- Booting-Starting a computer by loading a kernel
- Bootstrap program-Code stored in ROM that is able to locate the kernel, load it into memory and starts its execution.

Virtual Machines:

- A virtual machine (VM) is a software implementation of a machine (i.e. a computer) that executes programs like a physical machine.
- Virtual machines are separated into two major categories, based on their use and degree of correspondence to any real machine.
- A **system virtual machine** provides a complete system platform which supports the execution of a complete OS.
- A **process virtual machine** is designed to run a single program, which means that it supports a single process.
- An essential characteristic of a virtual machine is that the software running inside is limited to the resources and abstractions provided by the virtual machine.
- A virtual machine was originally defined by **Popek and Goldberg** as "an efficient, isolated duplicate of a real machine". Current use includes virtual machines which have no direct correspondence to any real hardware.

Virtual Machine/Non-Virtual machine:



Advantages/Disadvantages:

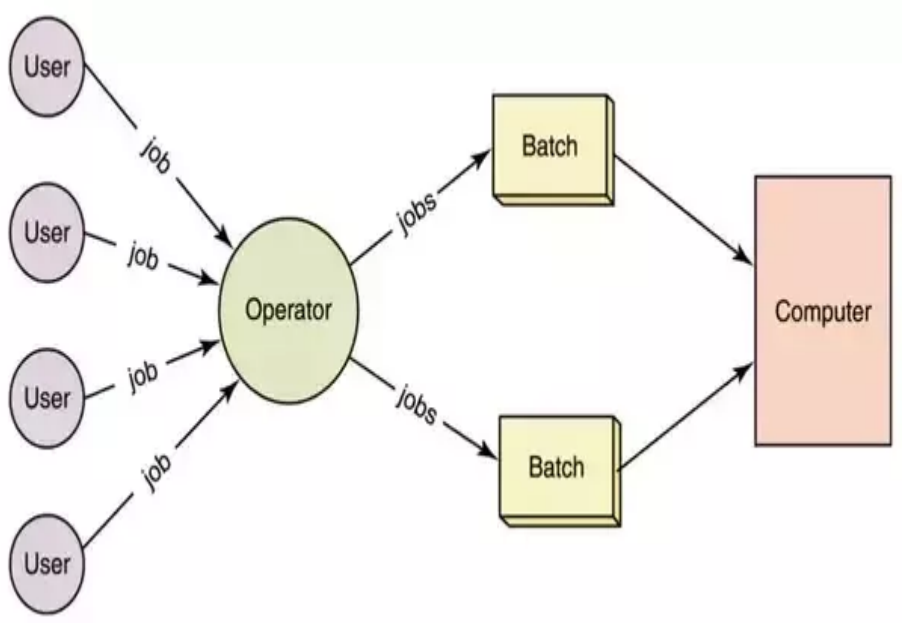
- A virtual machine is less efficient than a real machine when it accesses the hardware indirectly.
- When multiple VMs are concurrently running on the same physical host, each VM may exhibit a varying and unstable performance (Speed of Execution, and not results), which highly depends on the workload imposed on the system by other VMs, unless proper techniques are used for temporal isolation among virtual machines.
- Virtual machines concept provide complete protection of system resources since each VM is isolated from all other VMs. However, this isolation permits no direct sharing of resources.
- A virtual machine system is a perfect vehicle for OS research and development. System development is done on the VM instead of physical machine and therefore does not disrupt the normal system operation.
- The virtual concept is difficult to implement due to the effort required for providing an exact duplicate to the underlying machine.

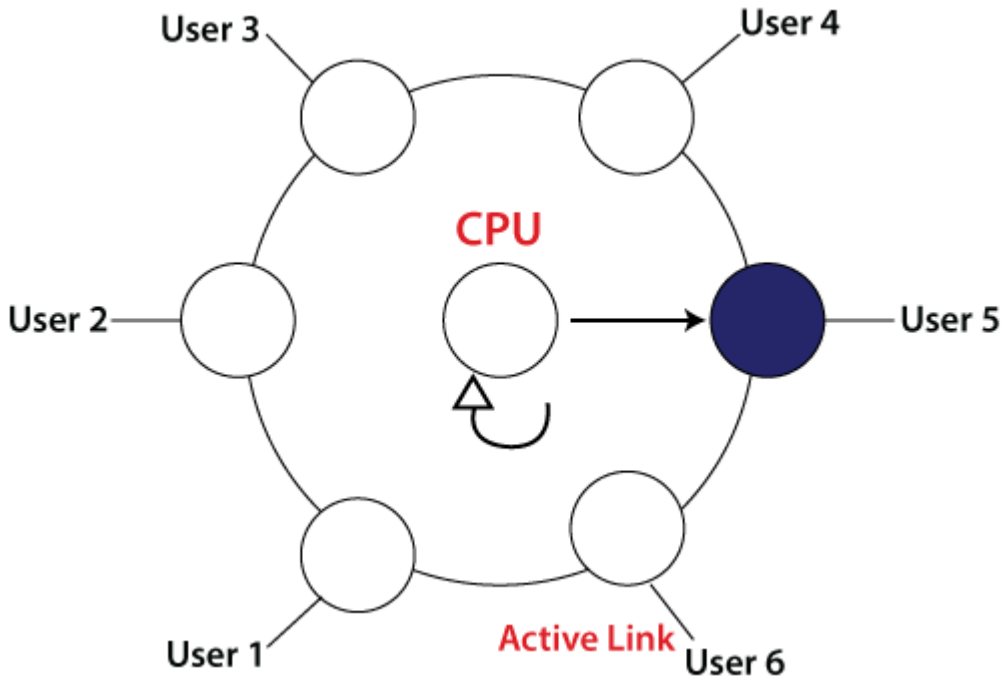
Operating Systems
Course Code: MCC510

Process Concept:

An Operating system executes a variety of programs via

- Batch system – Jobs
- Time shared systems - User programs or tasks



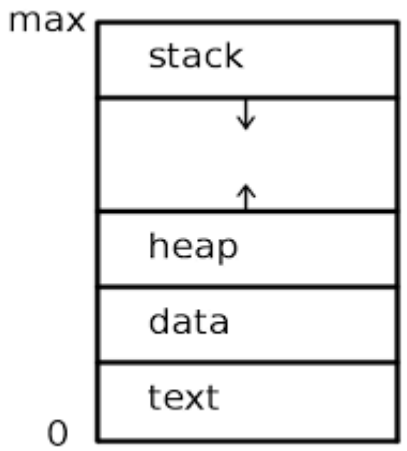


Operating Systems
Course Code: MCC510

Process:

- A program in execution is known as process and execution must be progress in sequential fashion
- A process includes:
  - program counter (The counter indicates the address of the next execution to be executed for this purpose),
  - Stack (Contains temporary data such as function, parameters, return address and local variable)
  - Data Section (includes global variable)

- Process in Memory:
  - Heap-It is a memory dynamically allocated during process run time

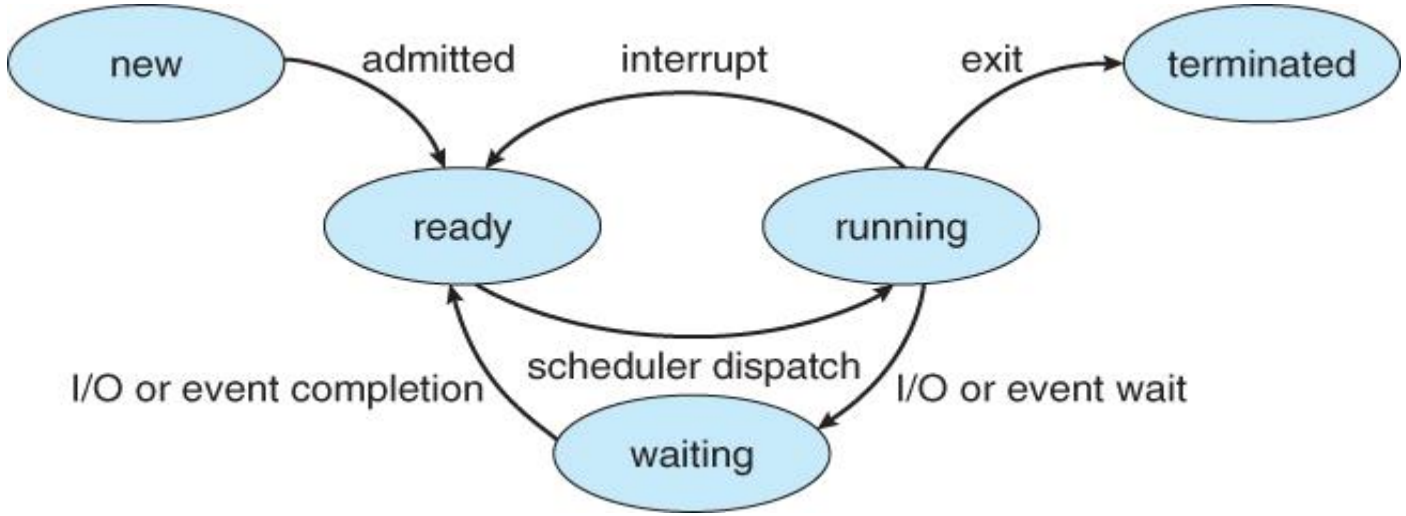


Operating Systems
Course Code: MCC510

Process States:

As a process executes, it changes state as follows:

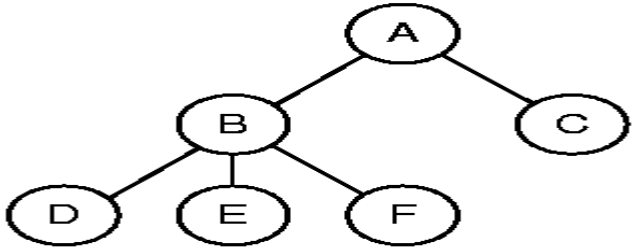
- New: The process is being executed
- Running: Instructions are being executed
- Waiting: The process is waiting for some event to occur
- Ready: The process is waiting to be assigned to a processor
- Terminated: The process has finished execution



Operating Systems
Course Code: MCC510

Process Creation:

- Parent process creates children processes which in turn create other processes and forming a tree of processes.



- Resource Sharing:
  - Parents and children share all resources
  - Children share subset of parent's resources
  - parent and child share no resources.
- Execution:
  - Parents and children execute concurrently
  - Parent waits until children terminate.
- Address space:
  - Child duplicates of parent
  - Child has a program loaded into it

Operating Systems
Course Code: MCC510

Process Creation:

- UNIX example:
- fork system call creates new process
- The new process is an exact copy of the parent and continues execution from the same point as its parent.
- Exec system call used after a fork to replace the processes memory space with a new program
- The only difference between parent and child is the value returned from the fork call.
  - 0 for the child.
  - the process id (pid) of the child, for the parent.
- Using fork and exec we can write a simple command line interpreter.

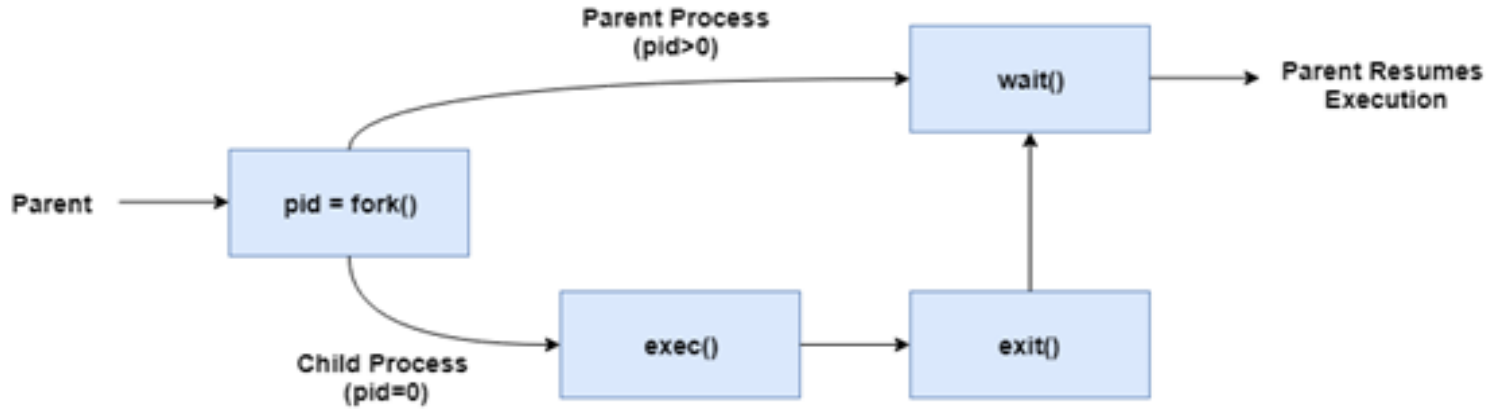
```
while (true) {
read_command_line(&command,&parameters);
if(fork()!=0) {
waitpid(-1,&status,0);
} else {
execve(command,parameters,0);
}

}
```

Operating Systems
Course Code: MCC510

Process Creation:

A process may be created by another process using fork(). The creating process is called the parent process and the created process is the child process. A child process can have only one parent but a parent process may have many children. Both the parent and child processes have the same memory image, open files and environment strings. However, they have distinct address spaces.



Process Creation Using fork()

Operating Systems
Course Code: MCC510

Process Termination:

- Process termination occurs when the process is terminated. The exit() system call is used by most operating systems for process termination. Process executes last statement and asks the OS to terminate it via exit().
  - Exit(or return) status value from child is received by parent via wait()
  - Process resources are deallocated by OS

- Parents may terminate execution of children process via kill() function.
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - If parent exit; some OS do not allow child to continue, if its parent terminates.

- All children terminated that means cascading termination

Operating Systems
Course Code: MCC510

Reason for Process Termination :

- A process may be terminated after its execution is usually completed. This process leaves the processor and releases all its resources.
- A child process may be terminated if its parent process requests for its termination.
- A process can be terminated if it tries to use a resource that it is not allowed. e.g., A process can be terminated for trying to write into a read only file.
- If an I/O failure occurs for a process, it can be terminated. e.g., If a process requires the printer and it is not working, then the process will be terminated.
- In most cases, if a parent process is terminated then its child processes are also terminated. This is done because the child process cannot exist without the parent process.
- If a process requires more memory than is currently available in the system, then it is terminated because of memory scarcity.

Operating Systems
Course Code: MCC510

Process Control Block:

- Each process is represented in the OS by a PCB also known as Task Control Block(TCB) which contains information associated with specific process such as
  - Process State
  - Program counter
  - CPU Registers
  - CPU Scheduling Information
  - Memory management information
  - Accounting information
  - I/O status information

Operating Systems
Course Code: MCC510

Process Scheduling:

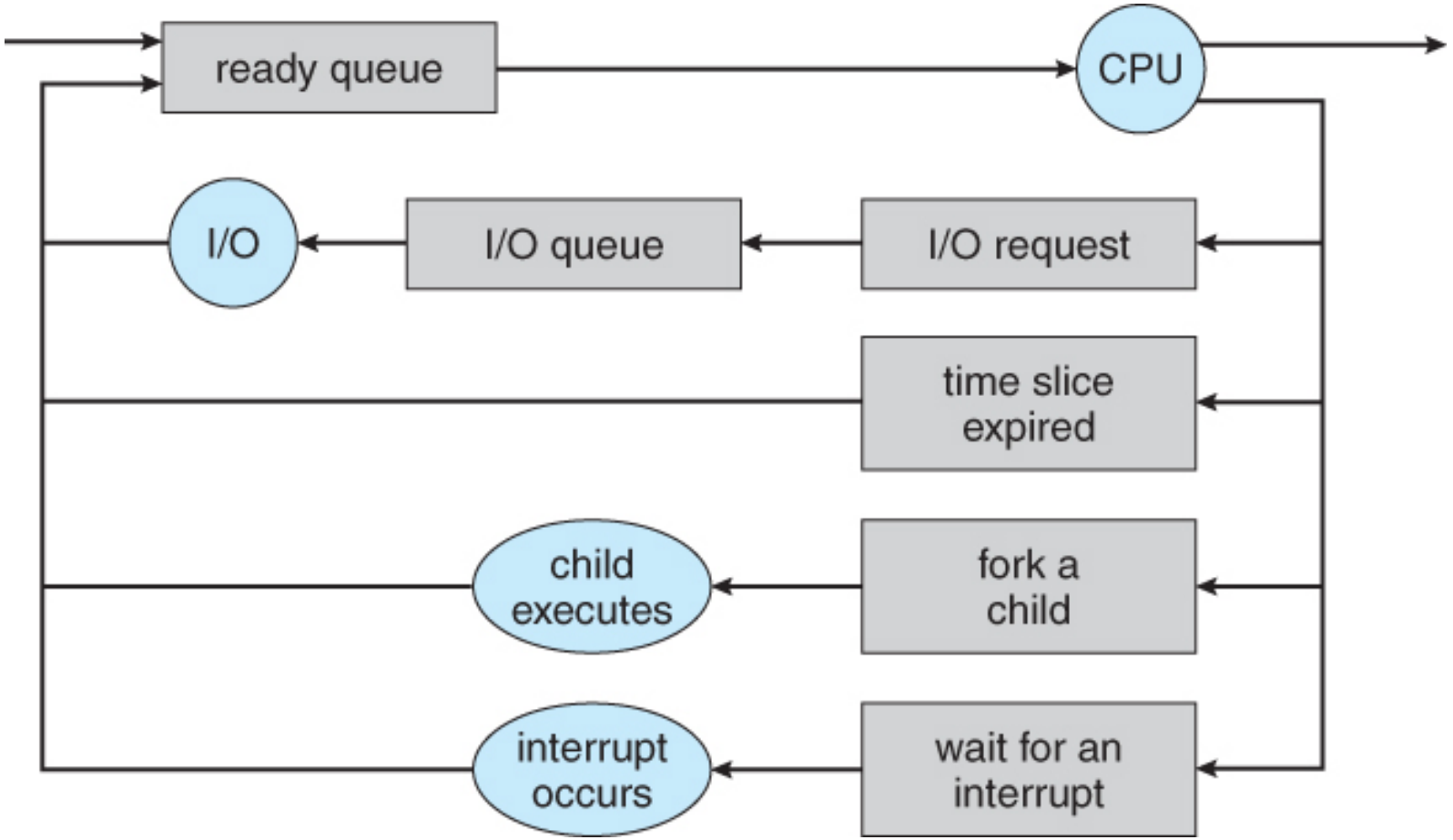
- **Process Scheduling** is an OS jobs that schedules processes of different states. It allows OS to allocate a time interval of CPU execution for each process. Process scheduling system is used to keeps the CPU busy all the time.

Process Scheduling Queues:

- **Job queue -** Set of all process in the system
- **Ready queue** - Set of all process in the main memory, ready and waiting to execute
- **Device queue** - Set of all process waiting for an I/O device
- Processes migrate among the various queues.

Operating Systems
Course Code: MCC510

Representation of Process Scheduling:



Operating Systems
Course Code: MCC510

Schedulers:

Long Term Scheduler(Job Schedular):

- Select which processes should be brought into the ready queue
- The primary objective is to provide balanced mix of jobs such as I/O bound and processor bound.
- It also control the degree of multiprogramming

Short Term Scheduler(CPU Schedular):

- Select which processes should be executed next and allocate to CPU
- The main objective is to increase system performance accordance with chosen set of criteria
- It is the change of ready state to running state of the process.

Operating Systems
Course Code: MCC510

Comparison between Schedulers:

Long Term Scheduler:

- It is a Job Schedular
- Speed is less
- control the degree of multiprogramming
- Absent or minimal in time sharing system
- Select processes from a pool and load them into memory for execution
- Process state is new to ready
- Select a good process, mix of I/O bound and CPU bound.

Operating Systems
Course Code: MCC510

Comparison between Schedulers:

Short Term Scheduler :

- It is a CPU Schedular
- Speed is very fast
- Less control over the degree of multiprogramming
- Minimal in time sharing system
- Select among the processes that are ready to execute
- Process state is ready to running
- Select a new process for a CPU quite frequently

Operating Systems
Course Code: MCC510

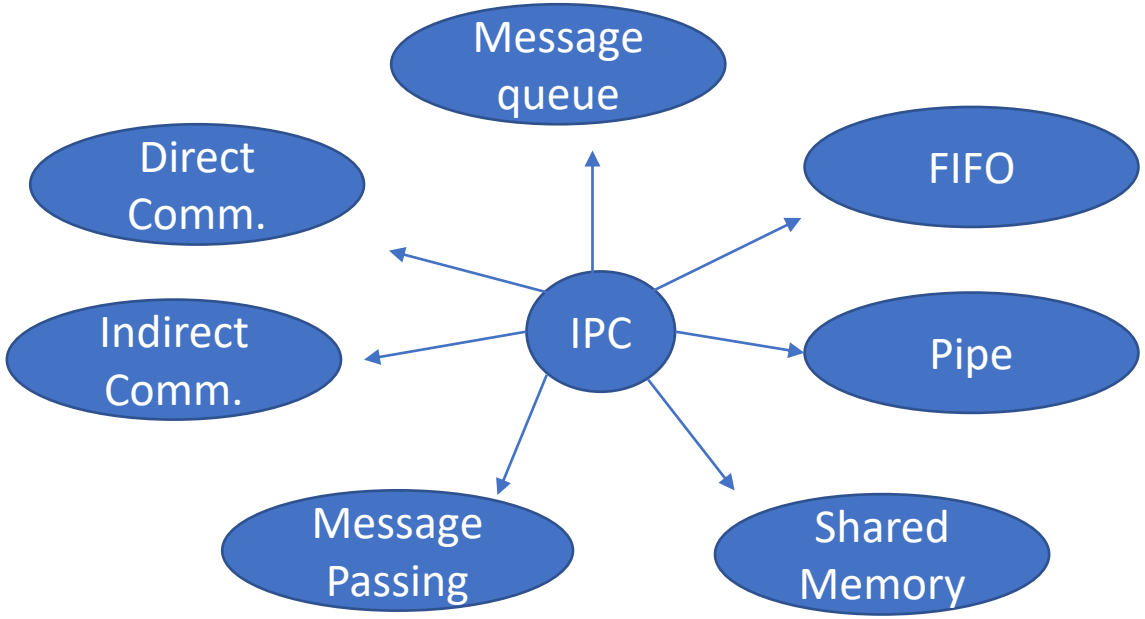
Context Switch:

- When CPU switches to another process (Because of an interrupt), the system must save the state of old process and load the saved state for new process
- Context switch time is overhead and the system does no useful work while switching
- Context switch time is highly dependent on hardware support
- When the process is switched, the information stored as
  - Program counter
  - Scheduling information
  - Base and limit register
  - Currently used register
  - Changed state
  - I/O state
  - Accounting

Operating Systems
Course Code: MCC510

Inter Process Communication(IPC):

- It is a mechanism which allows processes to communicate with each other and synchronize their actions. The communication between these processes can be seen as a method of co-operation between them.



Operating Systems
Course Code: MCC510

Inter Process Communication:

Independent Process:

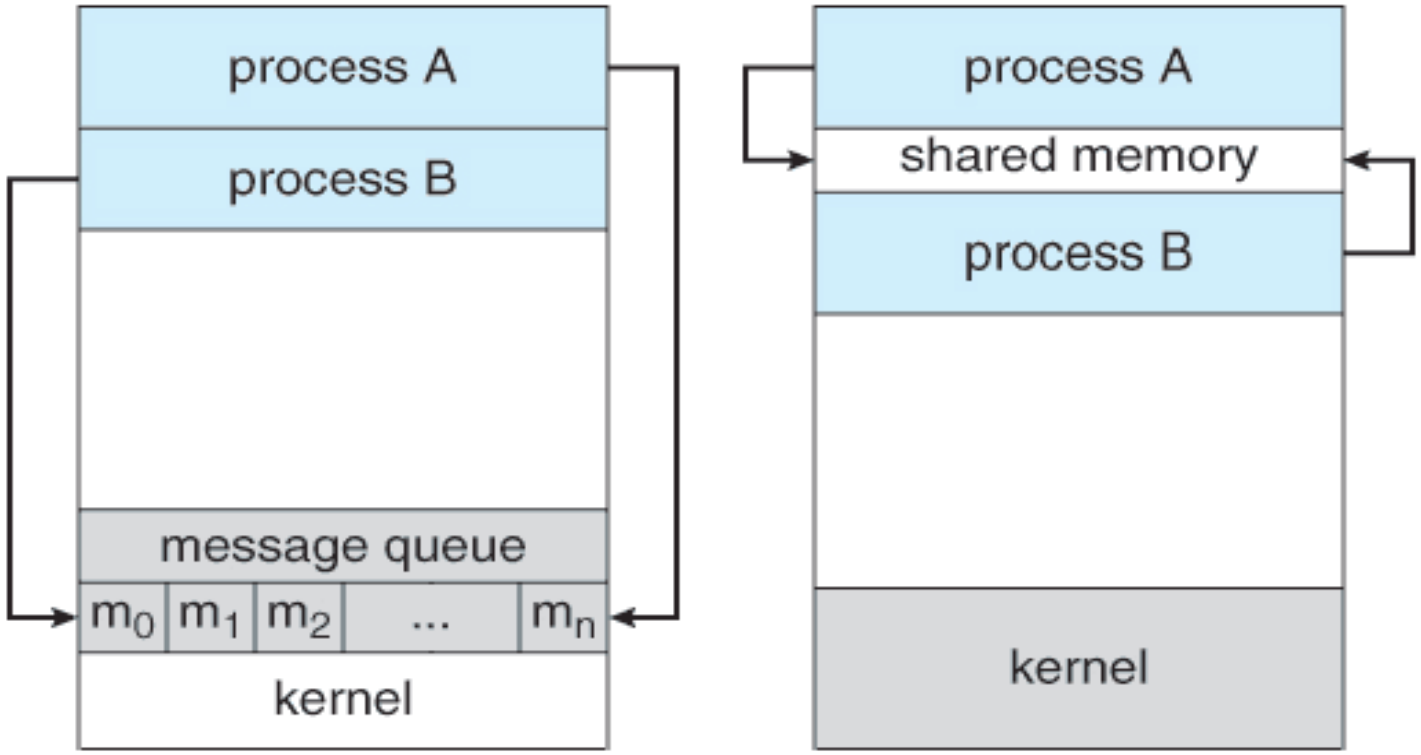
- It is one that can not affect or be affected by the execution of another process

Cooperating Process:

- It is one that can affect or be affected by the execution of another process in the system.
- Advantages:
  - Information sharing of the same piece of data
  - Computation speed up via break a task into some smaller task
  - Modularity dividing the system functions
  - Convenience for doing the multiple task
- It requires an IPC mechanism which allow them to exchange data and information

Operating Systems
Course Code: MCC510

Fundamental Models of Inter Process Communication:



(a) Message Passing (b) Shared Memory

Operating Systems
Course Code: MCC510

Fundamental Models of Inter Process Communication:

Message Passing:

- It is mechanism to allow processes to communicate and to synchronize their actions
- No address space needs to be shared and particularly useful in a distributed processing environment
- Message passing facility provides two operations
  - Send(message)-size can be fixed or variable
  - Receive(message)
- If P and Q want to communicate then they need to
  - establish a communication link between them
  - Exchange messages via send and receive

Operating Systems
Course Code: MCC510

Fundamental Models of Inter Process Communication:

Shared Memory:

- It requires communicating processes to establish a region of shared memory
- Information is exchanged by reading and writing data in the shared memory
- Producer consumer problem
  - A producer process produces information that is consumed by a consumer process
  - Unbounded buffer-places no practical limit on the size of the buffer
  - Bounded buffer-assumes that there is a fixed size of the buffer

Operating Systems
Course Code: MCC510

Direct Communication:

- Processes must name each other explicitly
  - Send(P, message)-send a message to process P
  - Receive(Q, message)- receive a message from process Q
- Properties of communication link:
  - Links are established automatically between every pair of processes that want to communicate
  - A link is associated with exactly one pair of communicating processes
  - Between each pair there exists exactly one link
  - The link may be unidirectional but usually it is bidirectional

Operating Systems
Course Code: MCC510

Indirect Communication:

- Messages are directed and received from mailboxes (referred to Port)
- Each mailbox has unique id
- Processes can communicate only if they share a mailbox
  - Send(P, message)-send a message to mailbox P
  - Receive(P, message)- receive a message from mailbox P
- Properties of communication link:
  - Links is established between a pair of processes only if both have shared mailbox
  - A link may be associated with more than two processes
  - Between each pair of processes, there may be many different links, with each link corresponding to one mailbox
- Mechanism provided by the operating system
  - Create a new mailbox
  - Send and receive messages through the mailbox
  - Destroy the mailbox

Message Queues :

- A message queue is a linked list of messages stored within the kernel. It is identified by a message queue identifier. This method offers communication between single or multiple processes with full-duplex capacity.

FIFO:

- Communication between two unrelated processes. It is a full-duplex method, which means that the first process can communicate with the second process, and the opposite can also happen.

Pipe:

- Pipe is widely used for communication between two related processes. This is a half-duplex method, so the first process communicates with the second process. However, in order to achieve a full-duplex, another pipe is needed.

Synchronization:

- Message passing may be either blocking or non- blocking
- **Blocking** is considered as synchronous
  - Blocking send has the sender block until the messages is received
  - Blocking receive has the receiver block until a message is available
- **Non-Blocking** is considered as asynchronous
  - Non-blocking send has the sender send the messages and continue
  - Non-blocking receive has the receiver receive a valid messages or null

Buffering:

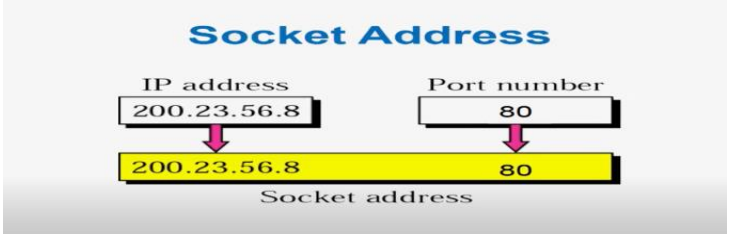
- Whether communication is direct or indirect, messages exchanged by communicating processes reside in a temporary queue
- These queue can be implemented in three ways
  - **Zero capacity**-queue has a maximum length of zero that means sender must block until the recipient receives the message
  - **Bounded capacity**-queue has a finite length of n that means sender must wait if queue is full
  - **Unbounded capacity**-queue length is unlimited that means sender never blocks

Operating Systems
Course Code: MCC510

Communication in Client Server System:

Sockets:

- It is defined as an endpoint for communication and identified by an IP address concatenated with a port number
- In general, sockets use client server architecture.[A **client** is a program or a process which connects to another program (the server) and lets it carry out a specific task. In particular it might require supplying some data from the server] [A **server** is a program or a process which provides services for clients. For example, it might supply some data or a result of processing data to clients] [A **protocol** is a language of communication among programs; in particular between a client and a server]
- The server waits for incoming client requests by listening to a specified port. Once a request is received, the server accepts a connection from the client socket to complete the connection.
- Server implementing specific services such as telnet, ftp, http listen to well-known ports i.e., 23,21 and 80 respectively.



Operating Systems
Course Code: MCC510

Port number with Protocol:

- Note that port is identified for each address and protocol by a 16-bit number known as port number. It is associated with an IP address of the host as well as the type of protocol used for communication. Well-known port (0-1023), registered port (1024-49151) and Dynamic or private port (49152-65535).

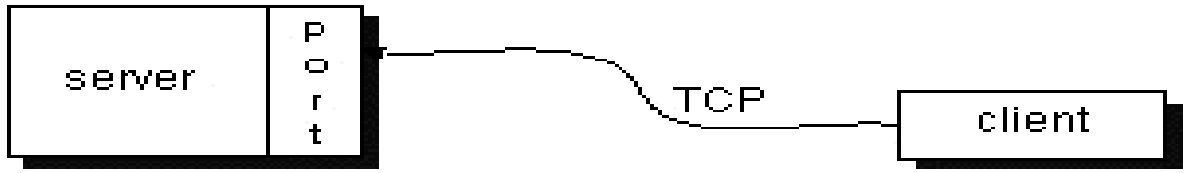
Port Number	Protocol
7	ECHO[the echo server]
20	FTP[File Transfer Protocol]-Data
21	FTP-commands
22	SSH Remote login Protocol
23	Telnet
25	SMTP[Simple mail transfer protocol]
37	Time
70	Gopher[Internet Gopher Protocol]-Hypertext link
80	HTTP[Hypertext Transfer Protocol]
110	POP3[Post Office Protocol]
119	NNTP[Network News Transfer Protocol]
123	NTP[Network Time Protocol]
143	IMAP[Internet Mail Access Protocol]
194	IRC[Internet Relay Chat]
443	HTTPS[TSL/SSL based HTTP]
989	FTPS[TSL/SSL based]-data
990	FTPS[TSL/SSL based]-command
992	Telnets [TSL/SSL based telnet]
993	IMAPS[TSL/SSL based IMAP4]
994	IRCS[TSL/SSL based IRC]
995	POP3S[TSL/SSL based POP3]
2628	DICT[the dictionary service]

Operating Systems
Course Code: MCC510

Communication in Client Server System:

Sockets:

- **TCP (Transmission Control Protocol)** is a connection-based protocol that provides a reliable flow of data between two computers. It guarantees that the sent data are not lost and arrive in the proper order to the receiver.
- Identification of hosts in the network is made using IP (Internet Protocol) addresses. An IP address is a 32-bit number (or a 128-bit for IPv6) typically represented using the dot notation as a sequence of four (or eight) numbers separated with dots (e.g. 192.33.71.12)

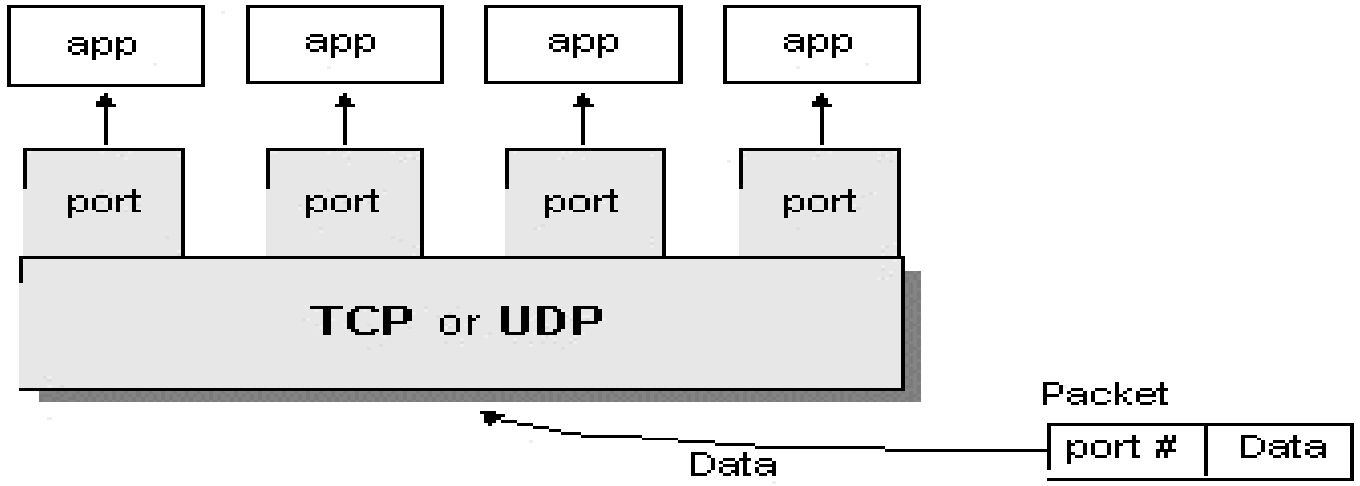


Operating Systems
Course Code: MCC510

Communication in Client Server System:

Sockets:

- **UDP (User Datagram Protocol)** is a protocol that sends independent packets of data, called datagrams, from one computer to another with no guarantees about arrival. UDP is not connection-based like TCP. Datagrams may arrive to the receiver in an arbitrary order, some of them might be lost.

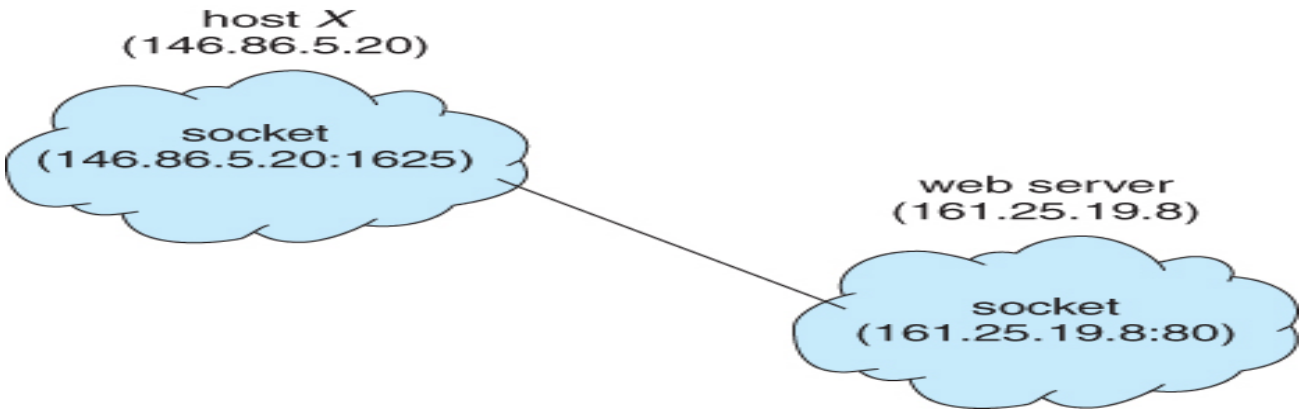


Operating Systems
Course Code: MCC510

Communication in Client Server System:

Interaction between client and server through Sockets:

- The server binds a socket to a specific port number and starts waiting for clients
- A client initializes a connection with the service specified by its host name and port number
- The server accepts the connection made by the client and creates a new socket for communicating with it
- From the client point of view, usually, the socket was used to initialize the connection



Operating Systems
Course Code: MCC510

Communication in Client Server System:

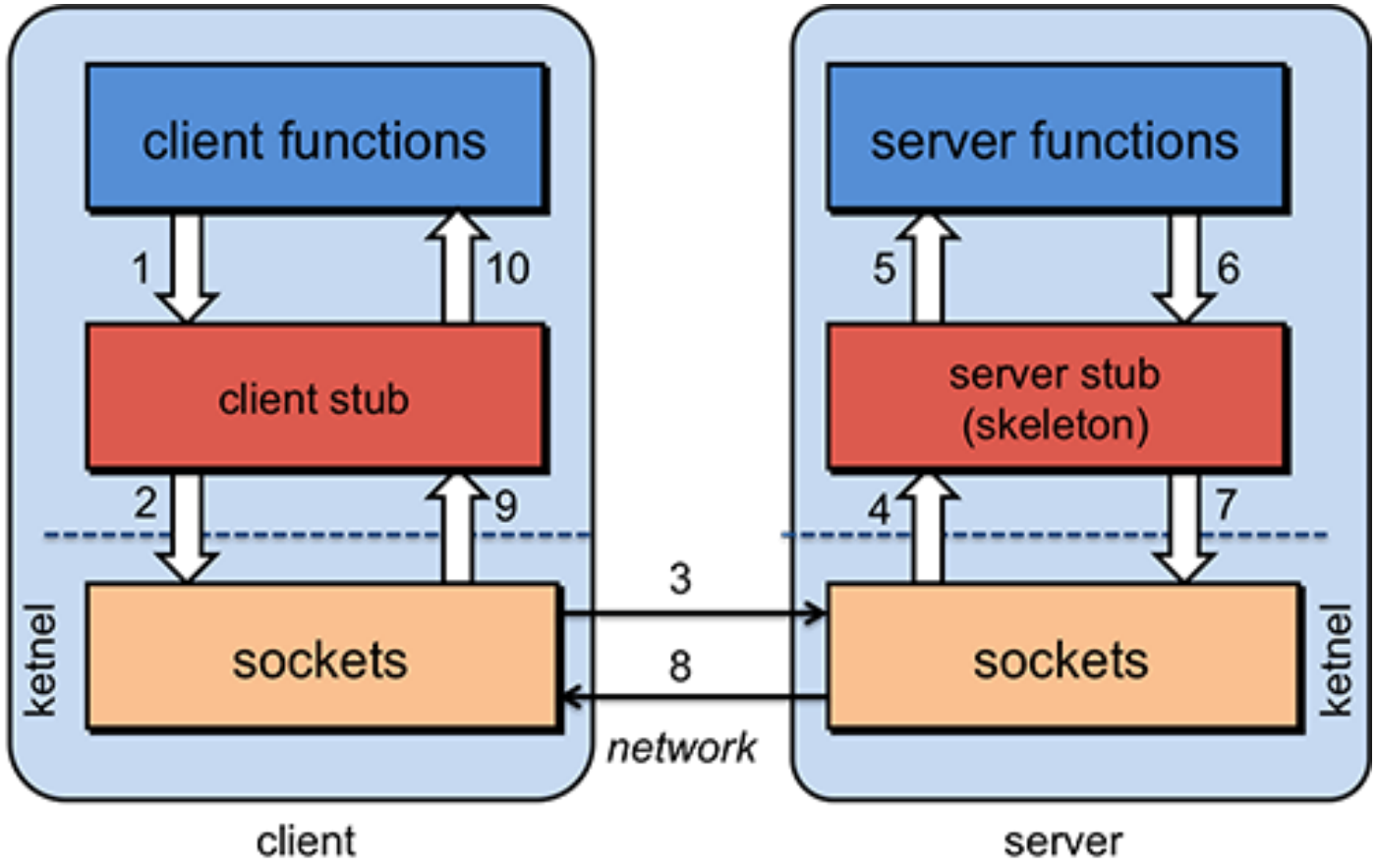
Remote Procedure Call(RPC):

- The main idea of RPC is to allow a local computer(client) to remotely call procedure on a remote computer(server)
- It is a message based communication scheme to provide remote service
- RPC abstracts the procedure call mechanism for use between the systems with network connection
- The RPC hides the details that allow the communication to take place by providing a **stub** on the client side.[**Stub** is a piece of code used for converting parameters passed during RPC so that remote function call looks like a local function call for the remote computer]
- Stub locates the port on the server and marshals the parameters. Parameter marshalling involves packaging the parameters into a form that can transmitted over a network.
- Stub then transmit a message to the server using message passing and the stub on server side receives this message and invokes the procedure on the server.
- RPC occurs when a process calls a procedure on a remote application.

Operating Systems
Course Code: MCC510

Communication in Client Server System:

Remote Procedure Call(RPC):

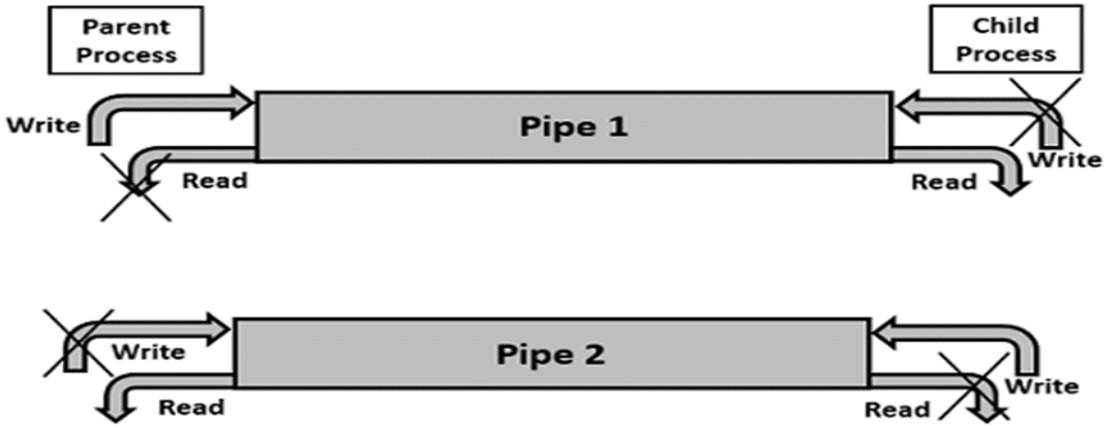


Operating Systems
Course Code: MCC510

Communication in Client Server System:

Pipes:

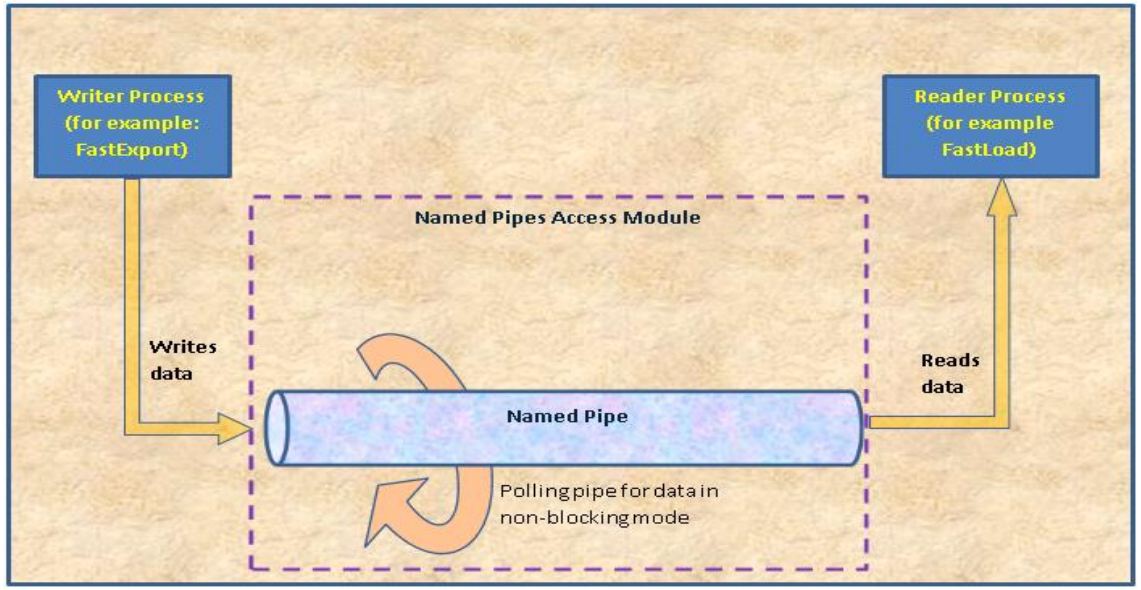
- A pipe acts as a conduit allowing two processes to communicate
- **Ordinary Pipe**-It allow two processes to communicate in standard producer consumer fashion in which producer write to one end of pipe (the write end) and the consumer reads from the another end(the read end). As a result ordinary pipe are unidirectional allowing only one way communication. If two way communication is required then two pipes must be used with each pipe sending data in different direction.



Operating Systems
Course Code: MCC510

Communication in Client Server System:

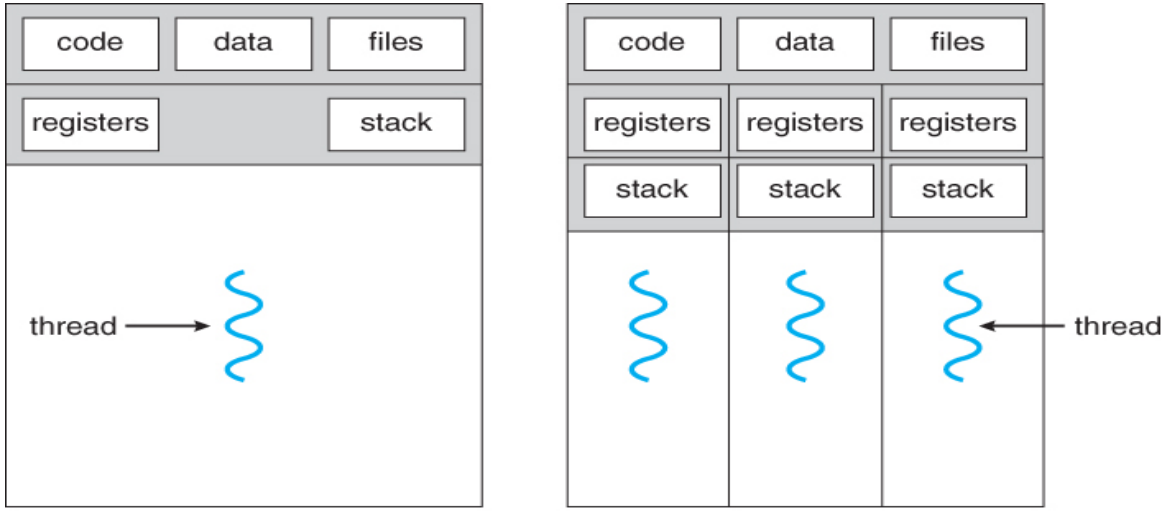
- **Named Pipe**-It is also known as FIFO for its behaviour and it is an extension to the traditional pipe concept on UNIX. It can be used to transfer instruction from one application to other without the use of intermediate temporary file. It provide much more powerful communication tool that can be bidirectional and no parent-child relationship is required.



Operating Systems
Course Code: MCC510

Threads:

- It is a basic unit of CPU utilization which comprises a thread ID, a program counter, a register set and a stack. It shares with other threads belonging to the same process its code section and other OS resources such as open files and signals.
- Threads are popular way to improve application performance through parallelism. A thread is sometimes known as light weight process.
- Single threaded process- It has a single thread of control. MS DOS support single user process and a single thread
- Multi threaded Process-It has multiple thread of control that can perform more than one task at a time. UNIX support multiuser process but only support one thread per process, Solaris supports multiple thread.



single-threaded process multithreaded process

Operating Systems
Course Code: MCC510

Advantages of Thread:

- **Responsiveness**-Multithreading is an interactive application may allow a program to continue running even if a part of it is blocked or is performing a lengthy operation, thereby increasing responsiveness.
- **Resource sharing**-By default, threads share the memory and the resources of the process to which they belong. The benefit of sharing code and data is that it allows an application to have several different threads of activity within the same address space.
- **Economy**-Allocating memory and resources for process creation is a costly affairs as threads share the resources of the process to which they belong. It is more economical to create context switch threads.
- **Utilization of MP architectures**-The benefit of multithreading can be greatly increased in a multiprocessor architecture where threads may be running in parallel on different processors.

Operating Systems
Course Code: MCC510

Types of Thread:

- **User Level Thread**-All the work of thread management is done by the application and kernel is not aware of the existence of threads. The thread library contains code for creating and destroying threads for passing message and data between threads.
- Advantages-
  - Thread switching does not require kernel mode privileges
  - It can run on any operating system
  - Scheduling can be application specific
  - It is faster to create and manage
- Disadvantages-
  - In a typical OS, most system calls are blocking
  - Multithreaded application cannot take advantage of multiprocessing

Operating Systems
Course Code: MCC510

Types of Thread:

- **Kernel Level Thread**-Thread management done by the kernel. There is no thread management code in the application area. Kernel threads are supported directly by the OS. e.g., Windows XP/2000, Solaris, Linux,Tru64 UNIX, Mac OS X
- Advantages-
  - Kernel can simultaneously schedule multiple threads from the same process on multiple processes
  - If one thread in a process is blocked then the kernel can schedule another thread of the same process
  - Kernel routine themselves multithreaded.
- Disadvantages-
  - Kernel thread are generally slower to create and mange
  - Transfer of control from one thread to another thread within the same process requires a mode switch to the kernel.

Operating Systems
Course Code: MCC510

Difference between User and Kernel level Threads:

- **User Level thread**
  - Faster to create and manage
  - Implemented by thread library at the user level
  - Run on any OS
  - Support provided at the user level
  - Multithreaded application cannot take advantage of multiprocessing

Kernel Level Thread

- Slower to create and manage
- OS support directly to Kernel threads
- Kernel level threads are specific to the OS
- Support provided by kernel
- Kernel routine themselves multithreaded.

Operating Systems
Course Code: MCC510

Difference between Process and Thread:

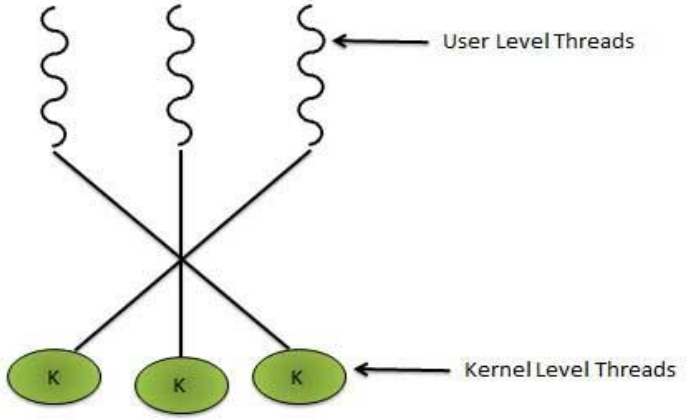
- **Process**
  - Heavy weight process
  - Process switching needs interface with OS
  - In multiple process implementation, each process executes the same code but has its own memory and file resources
  - If one server process is blocked then no other server process can execute until the first process unblocked.
  - In multiple process, each process operates independently

- **Thread**
  - Light weight process
  - Thread switching does not need to call a OS and cause an interrupt to the kernel
  - All threads can share same set of open files, child processes.
  - While one server thread is blocked and waiting, second thread in the same task could run.
  - One thread can read, write or even completely wipe out another thread stack.

Operating Systems
Course Code: MCC510

Multithreading Models:

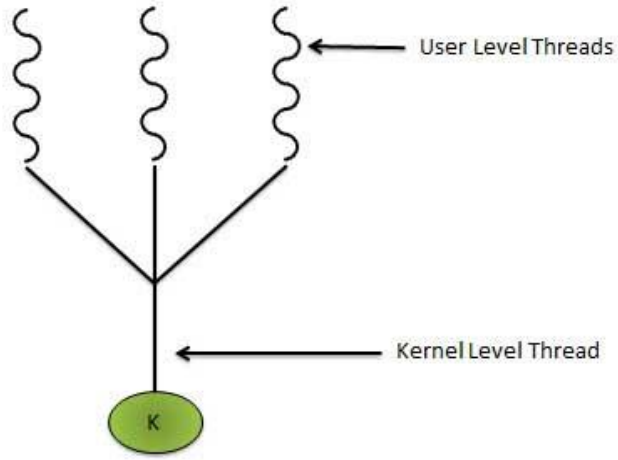
- **Many to Many Model-** In this model, many user level threads multiplexes to the kernel thread of smaller or equal numbers. The number of kernel threads may be specific to either a particular application or a particular machine.
- In this model, developers can create as many user thread as necessary and the corresponding kernel threads can run in parallel on a multiprocessor.
- Examples-Solaris prior to version 9, Windows NT/2000 with the thread fiber package



Operating Systems
Course Code: MCC510

Multithreading Models:

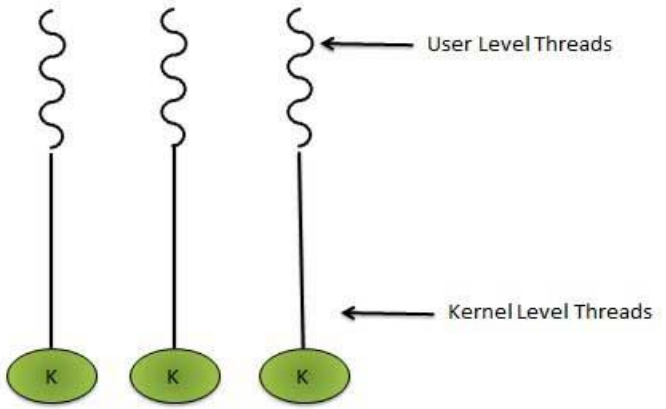
- **Many to One Model-** In this model, many user level threads maps to the one kernel thread.
- Thread management done in user space only.
- If thread makes a blocking system call then the entire process will be blocked.
- Only one thread can access the kernel at a time so multiple thread are unable to run in parallel on a multiprocessor.
- Examples-Solaris Green Thread, GNU Portable thread



Operating Systems
Course Code: MCC510

Multithreading Models:

- **One to One Model-** In this model, there is one to one correspondence between user level thread and kernel level thread
- It provides more concurrency than many to one model by allowing another thread to run when thread makes a blocking system call. It also allow multiple thread to run in parallel on a multiprocessor.
- The only drawback to this model is that creating a user thread requires creating the corresponding kernel thread. Because the overhead of creating kernel threads can burden the performance of an application.
- Examples- Windows NT/XP/2000, Linux, Solaris 9 and later



Thread Library:

- A thread library provides the programmer with an API for creating and managing threads.
- Two Approaches for Implementing a Thread library.
  - The first approach is to provide a library entirely in user space with no kernel support. All code and data structures for the library exist in user space.
  - The second approach is to implement a kernel level library supported directly by the OS. The code and data structures for the library exist in the kernel space.
- There are three main thread libraries which are in use such as Posix P threads, Win 32 threads and Java threads.

Operating Systems
Course Code: MCC510

Threading Issues:

- Semantics of fork() and exec() system calls
- Thread cancellation
- Signal Handling
- Thread Pools
- Thread Specific data
- Schedular Activation

Operating Systems
Course Code: MCC510

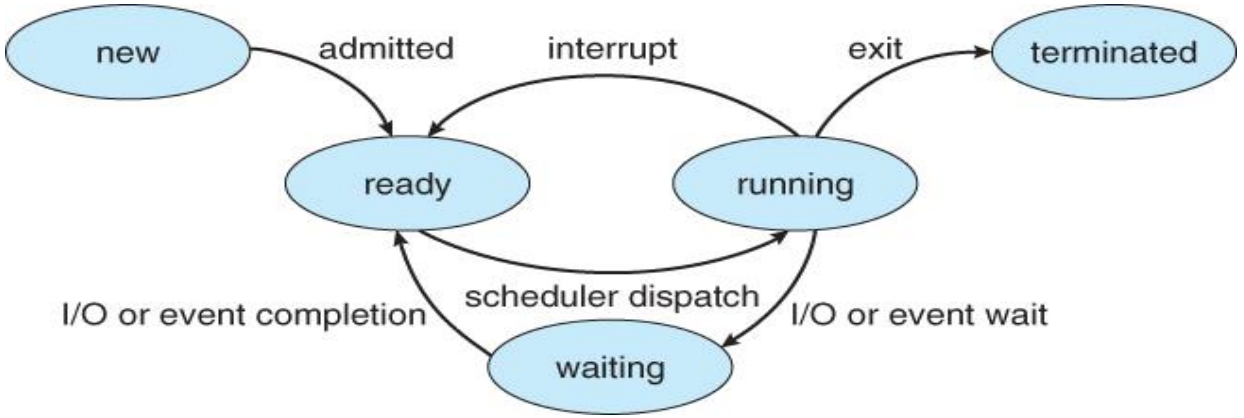
CPU Scheduling:

- **Basic Concepts**
  - Maximum CPU utilization obtained with multiprogramming
  - CPU I/O Burst Cycle-Process execution consists of a cycle of CPU execution and I/O wait
  - CPU burst distribution

Operating Systems
Course Code: MCC510

CPU Schedular:

- It selects the processes in the memory which are in ready to execute and allocate the CPU.
- CPU scheduling decisions may take place when a process
  - Switches from running to waiting state
  - Switches from running to ready state
  - Switches from waiting to ready state
  - Switches from running to terminated state
- Scheduling under 1st and last is non-preemptive
- Scheduling under 2nd and 3rd is preemptive



Operating Systems
Course Code: MCC510

CPU Scheduling Criteria:

- **CPU Utilization-** Keep the CPU as busy as possible. Conceptually, it ranges from 0-100% but in real system, it ranges from 40-90%
- **Throughput-** The number of processes that are completed per unit time is called throughput.
- **Turnaround Time-** Amount of time to execute a particular process. It is the interval of the time of submission of the process to the time of completion of the process.
- **Waiting Time-** Amount of time a process has spent waiting in the ready queue.
- **Response Time-** Amount of time it takes from when request was submitted until the first response is produced not the output.

Operating Systems
Course Code: MCC510

Optimization criteria:

- Maximize CPU utilization
- Maximize Throughput
- Minimize Turnaround time
- Minimize Waiting time
- Minimize Response time

Operating Systems
Course Code: MCC510

Scheduling Algorithms:

- First Cum First Serve (FCFS)
- Shortest Job First (SJF)
- Priority
- Round Robin (RR)

Operating Systems
Course Code: MCC510

Scheduling Algorithms:

- First Cum First Serve (FCFS)
  - FCFS is non-preemptive algorithm. Once the CPU has been allocated to a process, that process keeps the CPU until it releases it either by terminating or by requesting I/O. It is a troublesome algorithm for time sharing systems.
  - For Example- Let us suppose that the processes are arrived in a order P1,P2,P3
    - Process Burst Time
    - P1 24
    - P2 3
    - P3 3
  - Gannt Chart-

P1	P2	P3
0 24	27	30

Average Waiting Time: (0+24+27)/3 = 17

Average Turnaround Time: (24+27+30)/3 = 27

Operating Systems
Course Code: MCC510

Scheduling Algorithms:

- First Cum First Serve (FCFS)
  - For Example- Let us suppose that the processes are arrived in a order P2,P3,P1
    - Process Burst Time
    - P1 24
    - P2 3
    - P3 3
  - Gannt Chart-

P2	P3	P1
0 3	6	30

Average Waiting Time: (0+3+6)/3 = 3

Average Turnaround Time: (3+6+30)/3 = 13

Operating Systems
Course Code: MCC510

Scheduling Algorithms:

- Shortest Job First(SJF)
  - It associates with each process the length of its next CPU burst time and use these lengths to schedule the process with the shortest time
  - SJF are both Non-preemptive and preemptive
  - In Non-preemptive case, once CPU given to the process it can not be preempted until completes its CPU burst time.
  - In preemptive case, if a new process arrives with CPU burst length less than remaining time of current executing process then preempt. His scheme is also known as Shortest Remaining Time First(SRTF)
  - SJF is optimal- It gives average waiting time for a given set of processes.

Operating Systems
Course Code: MCC510

Scheduling Algorithms:

- Shortest Job First(SJF)
  - For Non Preemptive case- Let us suppose that the processes P1,P2,P3,P4 are arrived as follows:

Process	Arrival Time	Burst Time
P1	0.0	7
P2	2.0	4
P3	4.0	1
P4	5.0	4

  - Gannt Chart-

P1	P3	P2	P4
0 7	8	12	16

Average Waiting Time: (0+6+3+7)/4 = 4

Average Turnaround Time: (7+10+4+11)/4 = 8

Operating Systems
Course Code: MCC510

Scheduling Algorithms:

- Shortest Job First(SJF)
  - For Preemptive case- Let us suppose that the processes P1,P2,P3,P4 are arrived as follows:

Process	Arrival Time	Burst Time
P1	0.0	7
P2	2.0	4
P3	4.0	1
P4	5.0	4

  - Gannt Chart-

P1	P2	P3	P2	P4	P1
0 2	4	5	7	11	16

Average Waiting Time: (9+1+0+2)/4 = 3

Average Turnaround Time: (16+7+5+11)/4 = 9.75

Operating Systems
Course Code: MCC510

Scheduling Algorithms:

- **Priority Scheduling**
  - A priority number is associated with each process
  - The CPU is allocated to the process with the highest priority(smallest integer will be the highest priority)
  - It is non-Preemptive
  - SJF is a priority scheduling where priority is the predicted next CPU burst time.
  - The main problem with the priority scheduling is Starvation i.e., low priority process may never execute.
  - The solution is aging i.e., as time progress, the priority of a process in the ready queue is increased.

Operating Systems
Course Code: MCC510

Scheduling Algorithms:

- **Priority Scheduling**

- For Example- Let us suppose that the processes P1,P2,P3,P4,P5 are arrived as follows:

Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

- Gannt Chart-

P2	P5	P1	P3	P4
0 1	6	16	18	19

Average Waiting Time: (6+0+16+18+1)/5 = 8.2

Average Turnaround Time: (16+1+18+19+6)/5 = 12

Operating Systems
Course Code: MCC510

Scheduling Algorithms:

➢ Round Robin(RR) Scheduling

- It is a preemptive algorithm that select the process that has been waiting the longest. After a specified time quantum, the running process is preempted and a new selection of process is made.
- In the RR algorithm, each process gets a small unit of CPU time(time quantum), usually 10-100 milli seconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are n processes in the ready queue and the time quantum is q then each process gets 1/n of the CPU time in chunks of at most q time units at once. No process waits more than (n-1)q time units.
- Performance: If q is large the FIFO.
- if q is small then it must be large with respect to context switch otherwise overhead is too high.

Operating Systems
Course Code: MCC510

Scheduling Algorithms:

➢ Round Robin (RR) Scheduling

- For Example- Let us suppose that the processes P1,P2,P3,P4 are arrived as follows with time quantum q=20:

Process	Burst Time
P1	53
P2	17
P3	68
P4	24

- Gannt Chart-

P1	P2	P3	P4	P1	P3	P4	P1	P3	P3
0 20	37	57	77	97	117	121	134	154	162

Average Waiting Time:

([(0-0)+(77-20)+(121-97)]+ (20-0)+[(37-0)+(97-57)+(134-117)]+[(57-0)+(117-77)])/4 = (81+20+94+97)/4=73

Average Turnaround Time: (134+37+162+121)/4 = 113.5

Operating Systems
Course Code: MCC510

Scheduling Algorithms:

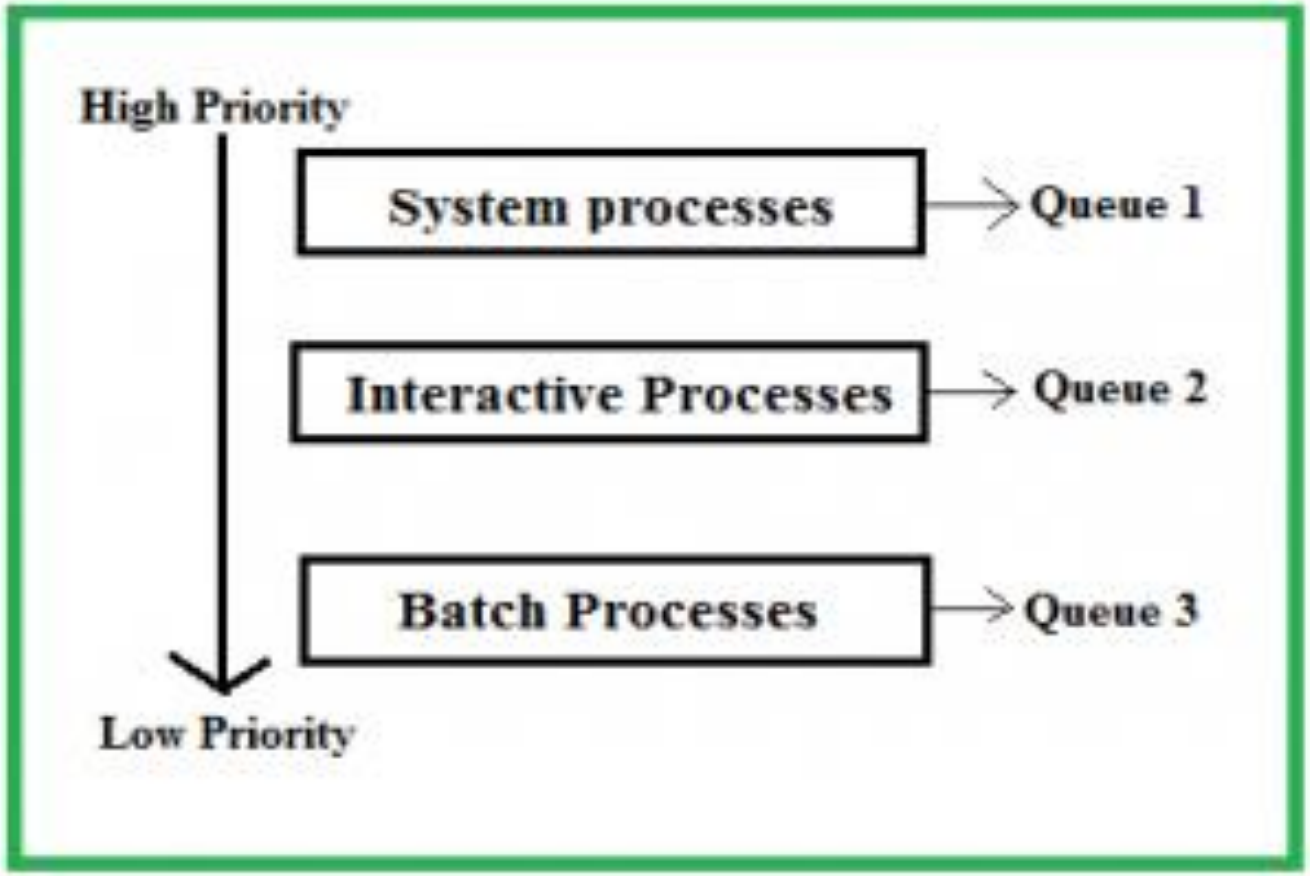
➢ Multilevel Queue Scheduling

- It is used when processes can classified in the group
- Ready queue is partitioned into separate queues:
  - Foreground(interactive)
  - Background(Batch)
- Each queue has its own scheduling algorithm
- Foreground having RR and Background having FCFS
- Scheduling must be done between the queues
- Fixed priority scheduling(i.e., serve all from foreground then from background). There will be a possibility of starvation
- Time slice-Each queue gets a certain amount of CPU time which it can schedule amongst its processes i.e., 80% to for foreground in RR and 20% to background in FCFS.

Operating Systems
Course Code: MCC510

Scheduling Algorithms:

- **Multilevel Queue Scheduling**



Operating Systems
Course Code: MCC510

Process Synchronization:

- **Basic Need of Process Synchronization**

P1	P2
Int. Shared=7	Int. Shared=7
Int X =Shared;	Int Y =Shared;
X++;	Y--;
Sleep(2);	Sleep(2);
Shared=X;	Shared=Y;
Terminated	Terminated
Outcome: X=8	Outcome: Y=6

If the process is not synchronize, then the outcome is wrong.

Operating Systems
Course Code: MCC510

Race Condition:

- Situations where the correctness of the computation performed by cooperating processes can be affected by the relative timing of the processes execution is known as **race condition**. To be considered correct computation, cooperating process may not be subject to race condition.
- When cooperating processes run on a system with a single processor, concurrency is simulated by the processes sharing the CPU. The scheduling algorithm determines the relative timing of cooperating processes. The result may differ widely from an idealized perception of parallel execution. It is not only time dependent on the scheduling algorithm used but it is also dependent on the system load created by other process on the system.
- On multiprocessor system, there is no guarantee that all processors work at equivalent speeds. It depends on which processes are allocated to which processors, timing can differ drastically.

Operating Systems
Course Code: MCC510

Critical Section:

- The code executed by a process can be grouped into sections. Some of which require access to shared resources and others do not require. The section of code that require access to shared resources are called **critical section.**
- To avoid race conditions, a mechanism is needed to appropriately synchronize the execution within critical section.

	P1	P2
# include	___	___
	main()	main()
	___	___
	Non critical Section	Non critical Section
	Entry Section	Entry Section
	Critical Section	Critical Section
	Exit Section	Exit Section
	___	___

Solution to Critical Section Problem:

- A solution to the critical section problem must satisfy the following requirements:
  - Mutual Exclusion
  - Progress
  - Bounded Wait
  - No Assumption

Operating Systems
Course Code: MCC510

Synchronization Hardware:

- Many system provide hardware support for critical sections
- Like uniprocessors which could disable interrupts
  - Current running code would execute without preemption
  - Generally too inefficient on multiprocessor system
- Modern machines provide special atomic hardware instructions (Non-interruptible)
  - Either test memory word and set value
  - Or, swap contents of two memory words.

Operating Systems
Course Code: MCC510

Test and Set Instruction:

- It is a special assembly language instruction that does two operations automatically.
  - Instructions can not be interrupted in the middle
  - It is not necessarily to deniable the interrupts
- **Structure of Test and Set instruction**

```
Boolean Test and Set (Boolean *target)
{
        Boolean rv= *target;
        *target= True;
        return rv;
}
```

Operating Systems
Course Code: MCC510

Solution Using Test and Set Instruction:

- Shared Boolean variable lock, initialized to false

```
while (true)
 {
        while (Test and Set (&lock));
        / *do nothing
        // critical section
        Lock=false;
        //remainder section
}
```

Operating Systems
Course Code: MCC510

Swap Instruction:

- It is just exchange the values of variables a and b

- **Structure of Swap Instruction**

```
void swap(boolean *a, boolean *b)
{
        Boolean temp = *a;
        *a = *b;
        *b = temp;
}
```

Solution Using Swap Instruction:

- Shared Boolean variable lock, initialized to false
- Each process has a local Boolean variable key.

```
while (true)
 {
        key = true;
        while (key==true);
        Swap(&lock, &key)
        // critical section
        Lock=false;
        //remainder section
}
```

Operating Systems
Course Code: MCC510

Semaphore:

- It is a controlling synchronization by using an abstract data type proposed by Dijkstra in 1965
- It is easily implemented in OS and provide a general purpose solution for controlling access to critical sections.
- It can also be treated as synchronization tool that does not required busy waiting.
  - Busy waiting- If the process is in its critical section then any other process that tries to enter its critical section must loop continuously at the entry code. This continual looping is known as busy waiting. Note that busy waiting waist CPU cycles and hence it makes use of CPU less efficient.
- Semaphore is represented by an integer variable S

Operating Systems
Course Code: MCC510

Semaphore:

- Two standard operations modify semaphore
  - Wait() - Originally called P() named for the Dutch word **Probern**, “to test”
  - Signal () - Originally called V() named for the Dutch word **Verhogen**, “to increment”
- Less complicated
- It can only be accessed via two atomic operations:

```
wait (S)
 {
while (S <= 0);
// no-op
S --;
}

Signal(S)
{
S++;
}
```

Operating Systems
Course Code: MCC510

Semaphore as General Synchronization Tool:

- Counting Semaphore
  - Integer variable can range over an unrestricted domain
- Binary Semaphore
  - Integer variable ranges only between 0 and 1.
  - It is simpler to implement
  - It is also known as mutex locks [Mutex is a library locked or unlocked semaphore is a notion of counting or a queue of more than one lock and unlock request]. It provides mutual exclusion.
    - **Structure**

```
Semaphore S;
//initialized to 1
wait(S);
Critical Section
Signal(S);
```

Semaphore Implementation:

- No two processes can execute wait() and signal() on the same semaphore at the same time
- So, semaphore implementation becomes critical section problem where the wait and signal code are placed in critical sections. This could now have busy waiting in critical section implementation
  - Implementation code is too short
  - Little busy waiting if critical section rarely occupied
- Note that applications may spend lots of time in critical sections and therefore this is not a good general solution.

Operating Systems
Course Code: MCC510

Semaphore Implementation without Busy waiting:

- With each semaphore there is an associated waiting queue. Each entry in a waiting queue has two data items
  - Value(of type integer)
  - Pointer to next record in the list
- Two Operations
  - Block- place the process invoking the wait operation on the appropriate waiting queue
  - Wakeup- Remove one of the processes in the waiting queue and place it in the ready queue.
- Implementation of Wait

```
wait (S)
{
value--;
if (value<0)
      {
       add this process to waiting queue
       block();
      }
}
```

Operating Systems
Course Code: MCC510

Semaphore Implementation without Busy waiting:

- Implementation of Signal

```
Signal (S)
{
value++;
if (value<=0)
      {
remove a process from the waiting queue
       wakeup();
      }
}
```

- Deadlock and Starvation
  - Deadlock- two or more processes are waiting indefinitely for an event that can be caused by only one of the waiting processes.
  - Starvation-It is indefinite blocking. A process may never be removed from the semaphore in which it is suspended.

Operating Systems
Course Code: MCC510

Classical Problems of Synchronization:

- Bounded Buffer Problem
- Readers and Writers Problem
- Dining Philosopher Problem

Operating Systems
Course Code: MCC510

Classical Problems of Synchronization:

- **Readers and Writers Problem**
  - A data set is shared among a number of concurrent processes
    - Readers-Only read the data set and do not perform any updates
    - Writers-Can both read and write
  - Problem
    - Allow multiple readers to read at the same time
    - Only one writer can access the shared data at the same time.
  - Shared Data
    - Semaphore mutex initialized to 1
    - Semaphore wrt initialized to 1
    - Integer readcount initialized to 0

Operating Systems
Course Code: MCC510

- **Readers and Writers Problem Continued**
  - **Structure of Writer process**

```
while(true)
{
wait(wrt);
// writing is performed
signal(wrt);
}
```

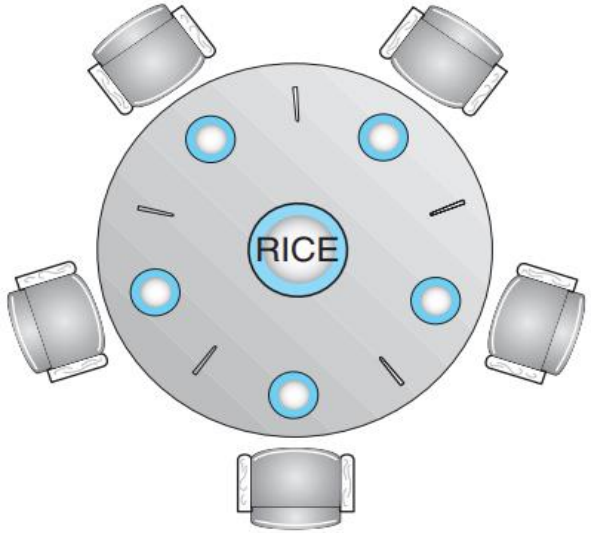
  - **Structure of Reader process**

```
while(true)
{
wait(mutex);
readcount++;
If (readcount = = 1) wait(wrt);
signal(mutex)
// reading is performed
wait(mutex);
readcount--;
If (readcount = = 0) signal(wrt);
signal(mutex)
}
```

  - Any number of readers may simultaneously be reading from a file. Only a writer at a time may write to the file and no reader can be reading while a writer is writing using a semaphore.

Operating Systems
Course Code: MCC510

- **Dining Philosophers Problem**:
  - Five philosopher are sitting at round table. In the centre of the round table there is a bowl of rice. Between each pair of philosopher there is a single chopstick. A philosopher is in one of the three states: thinking, hungry or eating. At a various of times a thinking philosopher gets hungry. A hungry philosopher attempts to pick up one of the adjacent chopstick, then the other but not both at the same time, If the philosopher is able to obtain the pair of chopstick (not already in use) then philosopher eats for a period of time. After eating the philosopher puts the chopstick down and returns to thinking.



Operating Systems
Course Code: MCC510

- **Dining Philosophers Problem**:
  - **Structure of philosopher i**

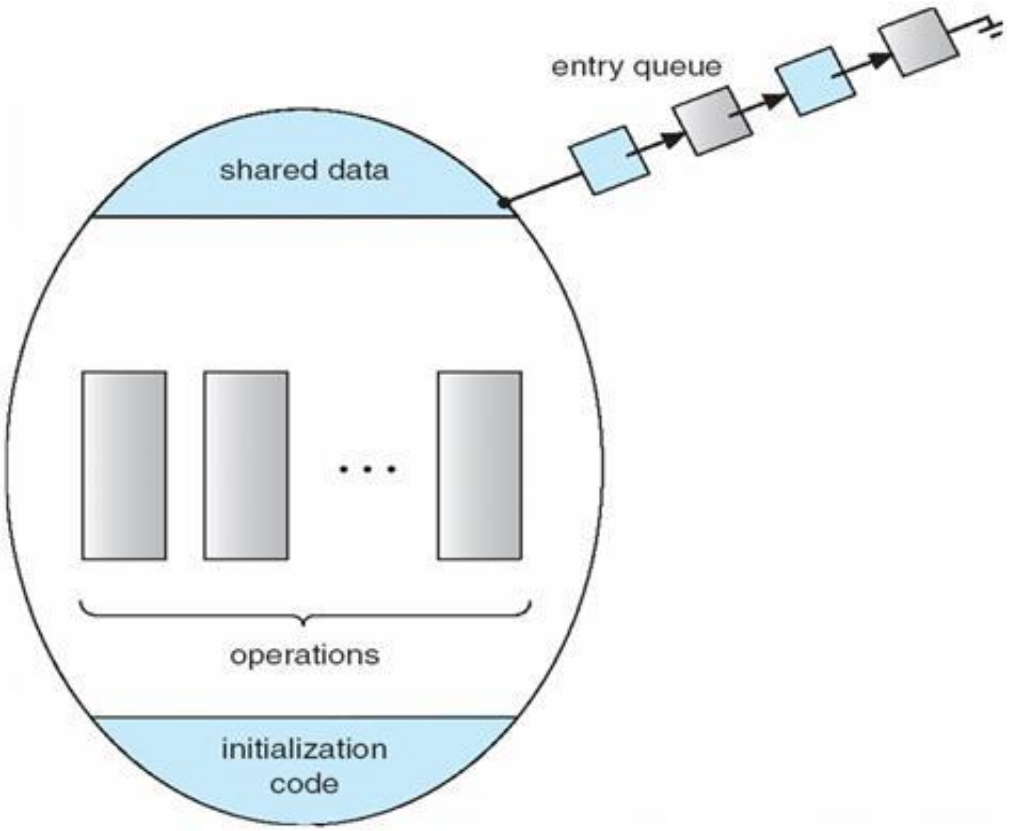
```
while(true)
{
wait(chopstick[i]);
wait(chopstick[(i+1)%5]);
// eat
signal(chopstick[i]);
signal(chopstick[(i+1)%5]);
// think
}
```

- **Monitor**:
  - A monitor is a programming language construct and guarantee that appropriate access to critical sections. The code placed before and after a critical section to control access to that critical section, is generated by the compiler. Controlled access is provided by the language not by the programmer.
  - In other words, a high level abstraction that provides a convenient and effective mechanism for process synchronization
  - Only one process may be active within the monitor at a time.

```
monitor monitor-name
{
// shared variable declarations
procedure P1 (…) { …. }
…
procedure Pn (…) {……}
Initialization code ( ….) { … }
…
}
```

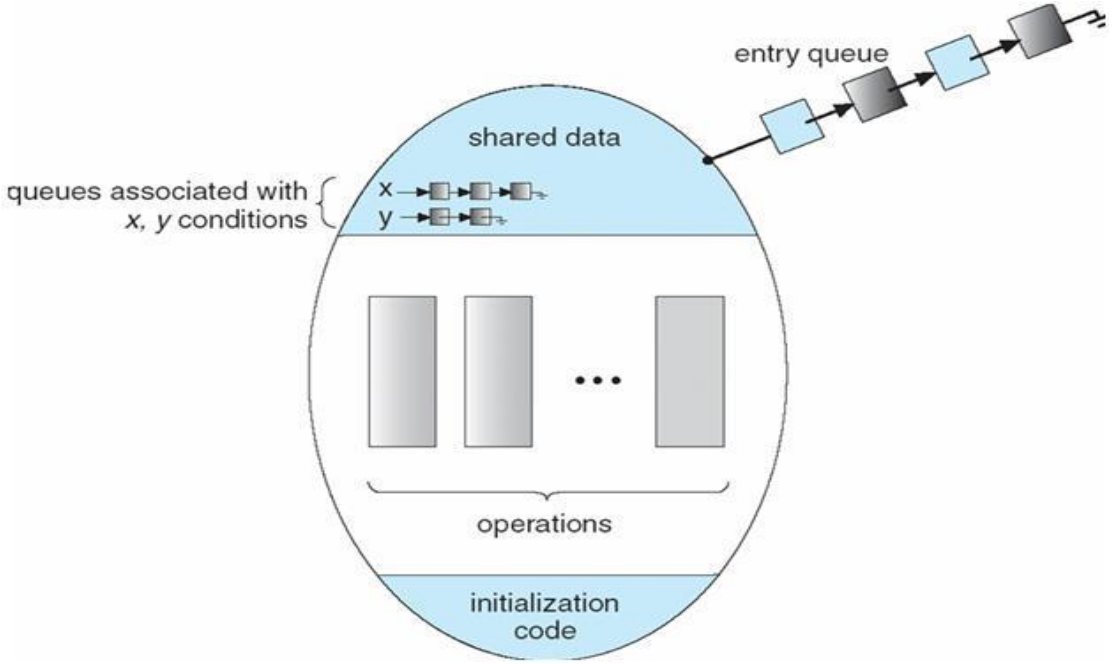
Operating Systems
Course Code: MCC510

- **Schematic view of Monitor**:



Operating Systems
Course Code: MCC510

- **Monitor with condition variable**:
  - condition x, y;
  - Two operations on a condition variable:
  - x.wait () – a process that invokes the operation is suspended.
  - x.signal () – resumes one of processes (if any) that invoked x.wait ()



Operating Systems
Course Code: MCC510

- **Solution to DP**

```
monitor DP
        {
enum { THINKING; HUNGRY, EATING) state [5] ;
        condition self [5];
      void pickup (int i) {
        state[i] = HUNGRY;
              test(i);
    if (state[i] != EATING) self [i].wait;
              }

 void putdown (int i) {
        state[i] = THINKING;
// test left and right neighbors
              test((i + 4) % 5);
              test((i + 1) % 5);
               }
```

```
void test (int i) {
                if ( (state[(i + 4) % 5] != EATING)
&&
                (state[i] == HUNGRY) &&
                (state[(i + 1) % 5] != EATING) ) {
                    state[i] = EATING ;
                            self[i].signal () ;
                 }
            }
    initialization_code() {
                for (int i = 0; i < 5; i++)
                state[i] = THINKING;
            }
}
```

Each philosopher *I* invokes the operations pickup() and putdown() in the following sequence:

```
DiningPhilosophters.pickup (i);
 EAT
 DiningPhilosophers.putdown (i);
```

Operating Systems
Course Code: MCC510

- **Monitor Implementation Using Semaphores**
  - Variables

```
semaphore mutex;  // (initially = 1)
semaphore next;   // (initially = 0)
int next-count = 0;
```

- Each procedure ***F*** will be replaced by

```
wait(mutex);
…
body of F;
…
if (next_count > 0)
    signal(next)
else
    signal(mutex);
```

- Mutual exclusion within a monitor is ensured.

Monitor Implementation

- For each condition variable ***x***, we have:

```
semaphore x_sem; // (initially = 0)
int x-count = 0;
```

- The operation x.wait can be implemented as:

```
x-count++;
if (next_count > 0)
    signal(next);
else
    signal(mutex);
wait(x_sem);
x-count--;
```

- The operation x.signal can be implemented as:

```
if (x-count > 0) {
    next_count++;
    signal(x_sem);
    wait(next);
    next_count--;
}
```

- **A Monitor to Allocate Single Resource**

```
monitor Resource Allocator
{
boolean busy;
condition x;
void acquire(int time) {
if (busy)
x.wait(time);
busy = TRUE;
}
void release() {
busy = FALSE;
x.signal();
}
initialization code() {
busy = FALSE;
}
}
```

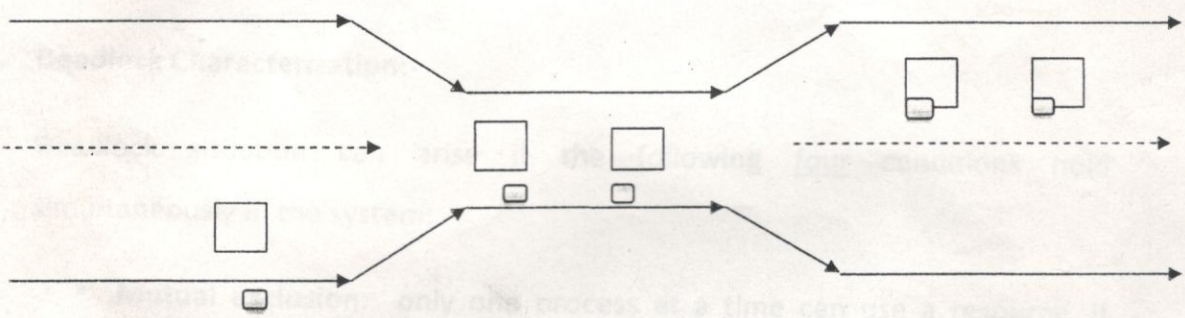
Chapter - 7

Deadlock

Deadlock occurs when each process in a set of processes controls a resource that another process in the set has requested. Each process blocks waiting for its requested resource to become available.

Example - A system has 2 disk drives; P_1 and P_2 each hold one disk drive and each need the other one.

Bridge Crossing Example



- Traffic only in one direction
- The resource is a one-lane bridge
- If a deadlock occurs, it can be resolved if one car backs up (preempt resources and rollback)
- Several cars may have to be backed up if a deadlock occurs
- Starvation is possible

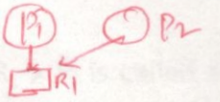
System Model-:

- Resources are partitioned into several steps, each consisting of some number of identical instances. *CPU cycles, memory space, I/O devices are examples of resource types*. If a system has two CPUs, then the resource type CPU has two instances.
- Each resource type R_i has *1 or more* instances.
- Each process utilizes a resource as follows:-
  1. request
  2. use
  3. release

Deadlock Characterization:-

Deadlock situation can arise if the following four conditions hold simultaneously in the system:

- **Mutual exclusion:** only one process at a time can use a resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.



- **Hold and wait:** a process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes .

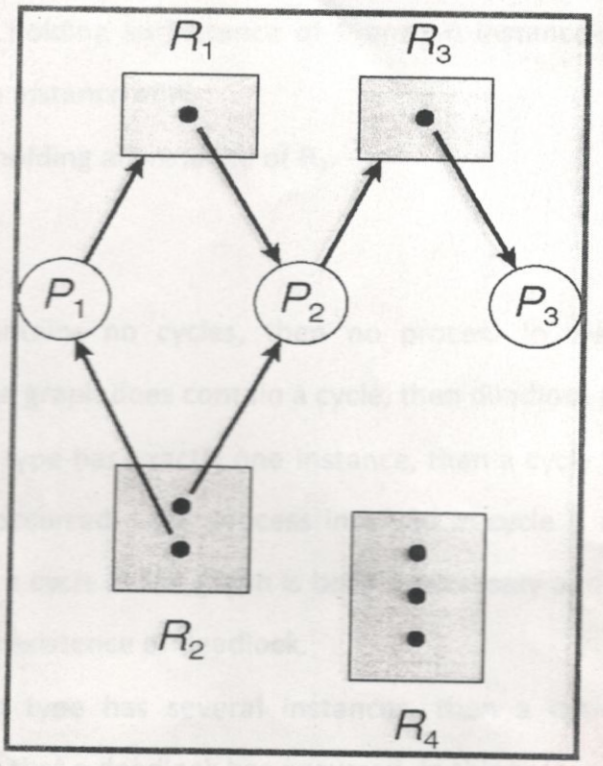
- **No preemption:** resources cannot be preempted; i.e. a resource can be released only voluntarily by the process holding it, after that process has completed its task.

- **Circular wait:** there exists a set $\{P_0, P_1, ..., P_0\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by P_2, ..., P_{n-1} is waiting for a resource that is held by P_n, and P_n is waiting for a resource that is held by P_0

Resource-Allocation Graph:-

Deadlock can be described more precisely in terms of a directed graph called a system resource allocation graph. This graph consists of a set of vertices V and a set of edges E.

- V is partitioned into two types of nodes:
  - $P = \{P_1, P_2, ..., P_n\}$, the set consisting of all the active processes in the system.
  - $R = \{R_1, R_2, ..., R_m\}$, the set consisting of all resource types in the system.
- $P_i \rightarrow R_j$; signifies that process P_i has requested an instance of resource type R_j and is currently waiting for that resource.
- $R_j \rightarrow P_i$; signifies that an instance of resource type R_j has been allocated to process P_i
- A directed edge $P_i \rightarrow R_j$ is called a request edge; and $R_j \rightarrow P_i$ is called an assignment edge.



The resource allocation graph shown above depicts the following situation:-

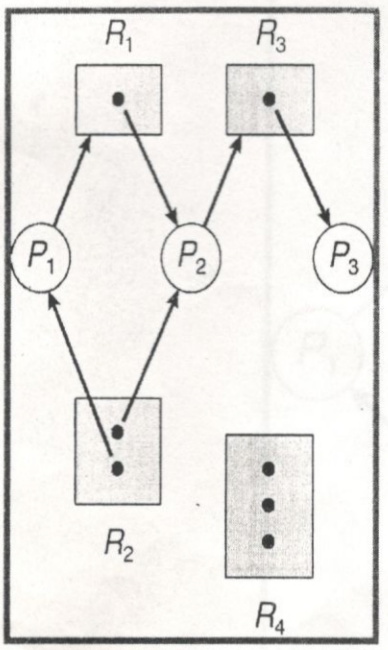
- The sets P,R, and E:
  - $P= \{P_1,P_2,P_3\}$
  - $R= \{R_1,R_2,R_3,R_4\}$
  - $E= \{P_1 \rightarrow R_1,\ P_2 \rightarrow R_3,\ R_1 \rightarrow P_2,\ R_2 \rightarrow P_2,\ R_2 \rightarrow P_1,\ R_3 \rightarrow P_3\}$
- Resources instances:
  - One instance of resource type R_1
  - Two instance of resource type R_2
  - One instance of resource type R_3
  - Three instance of resource type R_4
- Process states:
  - Process P_1 is holding an instance of resource type R_2 and is waiting for an instance of resource type R_1.

- Process P_2 is holding an instance of $R_1$and an instance of R_2 and is waiting for an instance of R_3.
- Process P_3 is holding an instance of R_3.

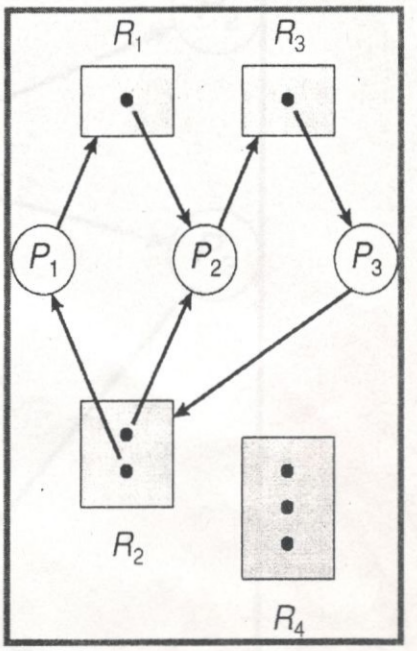
Notes:

- If the graph contains no cycles, then no process in the system is deadlocked. If the graph does contain a cycle, then deadlock may exist.
- If each resource type has exactly one instance, then a cycle implies that a deadlock has occurred. Each process involved in cycle is deadlocked. Thus in this case a cycle in the graph is both a necessary and a sufficient condition for the existence of deadlock.
- If each resource type has several instances, then a cycle does not necessarily imply that a deadlock has occurred. In this case, a cycle in the graph is a necessary but not a sufficient condition for the existence of deadlock.

To illustrate above concepts let us consider following resource allocation graph:-



Before P_3 requested an instance of R_2



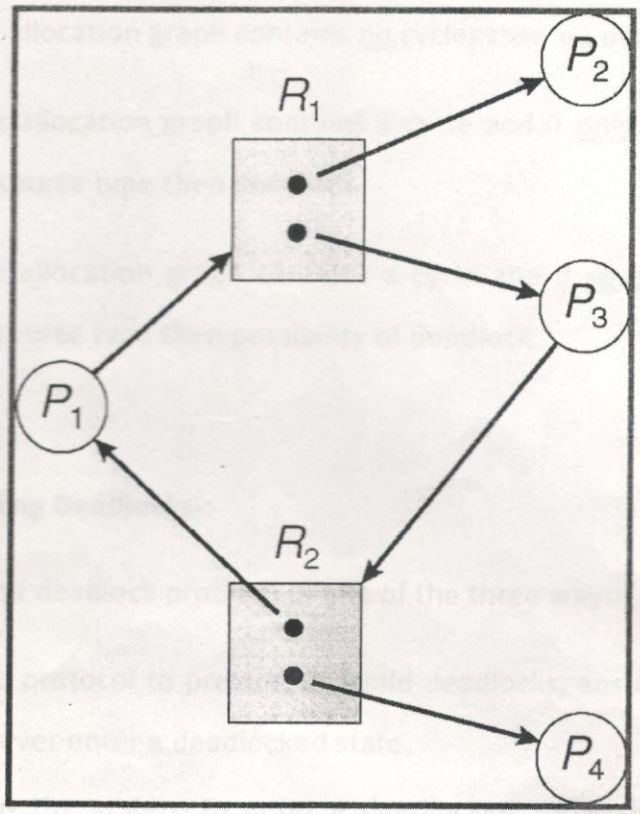
After P_3 requested an instance of R_2(deadlock)

Suppose that process P3 requests an instance of resource type R2. Since no resource instance is currently available, a request edge P3→R2 is added to the graph. At this point , two minimal cycles exists in the system:

- $P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$
- $P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$

Processes P1, P2, P3 are deadlocked.

Resource allocation Graph with A Cycle But No Deadlock:-



In this example we also have a cycle:

$P_1 \rightarrow R_1 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$

There is no deadlock. Process P_4 may release its instance of resource type R_2. That resource can then be allocated to P3, thereby breaking the cycle.

Relationship of cycles to deadlocks:-

- If a resource allocation graph contains no cycles then no deadlock .
- If a resource allocation graph contains a cycle and if only one instance exists per resource type then deadlock.
- If a resource allocation graph contains a cycle and if several instances exists per resource type then possibility of deadlock.

Methods for Handling Deadlocks-:

We can deal with the deadlock problem in one of the three ways:

- We can use a protocol to prevent or avoid deadlocks, ensuring that the system will never enter a deadlocked state.
- We can allow the system to enter a deadlocked state, detect it, and recover
- We can ignore the problem altogether and pretend that deadlocks never occur in the system.

Deadlock Prevention:-

Deadlock prevention provides a set of methods for ensuring that at least one of necessary condition (Mutual exclusion, Hold and wait, No pre-emption, Circular wait) cannot hold.

- **Mutual exclusion-:**

The Mutual exclusion condition must hold for nonsharable resources. For example a printer cannot be simultaneously shared by several processes.

- **Hold and wait-:**

we must guarantee that whenever a process requests a resource, it does not hold any other resources

  - This Require a process to request and be allocated all its resources before it begins execution, or allow a process to request resources only when the process has none
  - Result(disadvantage): Low resource utilization; starvation possible.

- **No Preemption –:**
  - If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released
  - Preempted resources are added to the list of resources for which the process is waiting
  - A process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.

- **Circular wait-:**

one way to ensure that this condition never holds is to impose a total ordering of all resource types and to require that each process requests resources in an increasing order of enumeration.

We let R= $\{R_1, R_2, R_3, \ldots\ldots, R_n\}$ be the set of resource types. We assign to each resource type a unique integer no., which allows us to compare two resources and to determine whether one precedes another in our ordering.

Deadlock Avoidance-:

Requires that the operating system has some additional *a priori* information available concerning which resources a process will request and use during lifetime.

- Simplest and most useful model requires that each process declare the *maximum number* of resources of each type that it may need

- The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition.

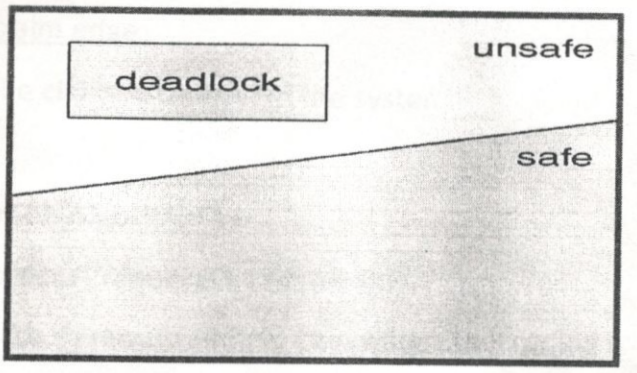
- A resource-allocation *state* is defined by the number of available and allocated resources, and the maximum demands of the processes.

Safe state-:

- When a process requests an available resource, the system must decide if immediate allocation leaves the system in a safe state
- A system is in a safe state only if there exists a safe sequence
- A sequence of processes $<P_1, P_2, ..., P_n>$ is a safe sequence for the current allocation state if, for each P_i, the resource requests that P_i can still make, can be satisfied by currently available resources plus resources held by all P_j, with $j < i$.
- That is:
    1. If the P_i resource needs are not immediately available, then P_i can wait until all P_j have finished

2. When P_j is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate
3. When P_i terminates, P_{i+1} can obtain its needed resources, and so on

Safe, Unsafe , Deadlock State-:



- If a system is in safe state then no deadlocks
- If a system is in unsafe state then possibility of deadlock
- Avoidance ensures that a system will never enter an unsafe state .

Avoidance algorithms-:

- For a single instance of a resource type, use a resource-allocation graph
- For multiple instances of a resource type, use the banker's algorithm

Resource-Allocation Graph Scheme-:

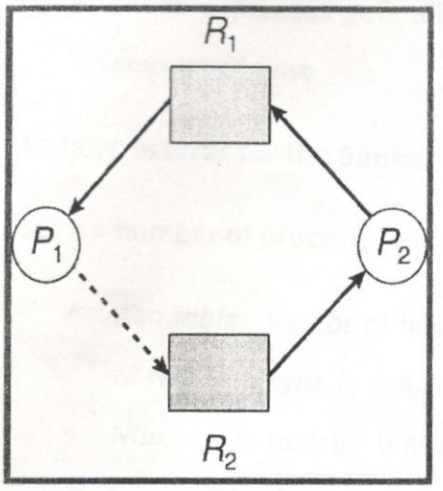
- *Claim edge* P_i ------>R_j indicates that process P_i may request resource R_j; which is represented by a dashed line .
- A claim edge converts to a request edge when a process **requests** a resource
- A request edge converts to an assignment edge when the resource is **allocated** to the process
- When a resource is **released** by a process, an assignment edge reconverts to a claim edge
- Resources must be **claimed *a priori*** in the system.

Resource-Allocation Graph Algorithm-:

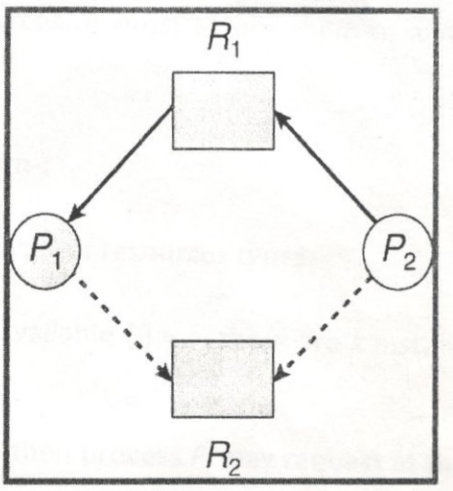
- Suppose that process P_i requests a resource R_j
- The request can be granted only if converting the request edge to an assignment edge does not result in the formation of a cycle in the resource allocation graph.

If no cycle exists, then the allocation of the resource will leave the system in a safe state; otherwise put it into unsafe state. In that case process P_i will have to wait for its request to satisfied.

To illustrate this algorithm, suppose that P_2 requests R_2. Although R_2 is currently free we cannot allocate it to P_2, since this action will create a cycle in the graph. A cycle as mentioned indicates that the system is in an unsafe state. If P_1 requests R_2 and P_2 requests R_1 then a deadlock will occur.



Resource allocation Graph for deadlock avoidance



An unsafe state in graph

Banker's Algorithm-:

- Used when there exists **multiple** instances of a resource type
- Each process must **a priori** claim maximum use
- When a process requests a resource, it may have to wait
- When a process gets all its resources, it must return them in a finite amount of time

Data Structures for the Banker's Algorithm-:

Let n = number of processes, and m = number of resources types.

- ***Available:*** Vector of length m. If available $[j] = k$, there are k instances of resource type R_j available.
- ***Max:*** $n \times m$ matrix. If $Max\ [i,j] = k$, then process P_i may request at most k instances of resource type R_j.
- ***Allocation:*** $n \times m$ matrix. If $Allocation[i,j] = k$ then P_i is currently allocated k instances of R_j.
- ***Need:*** $n \times m$ matrix. If $Need[i,j] = k$, then P_i may need k more instances of R_j to complete its task.

$$Need\ [i,j] = Max[i,j] - Allocation\ [i,j]$$

Operating Systems
Course Code: MCC510

Memory Management:

- **Objective**
  - To provide detailed description of various ways of organizing memory hardware
  - To discuss various memory management techniques

Operating Systems
Course Code: MCC510

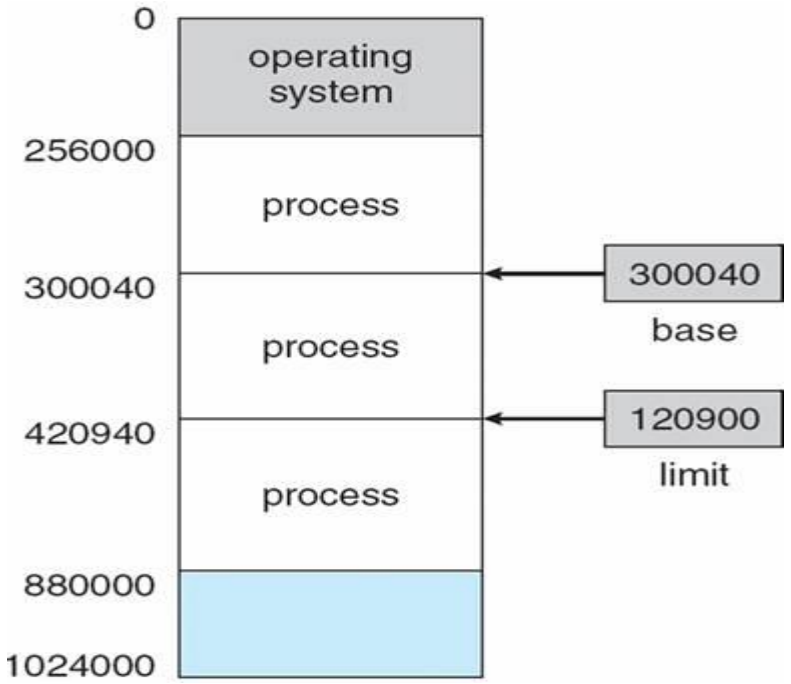
Background Information:

- Program must be brought from disk into memory and placed within a process for it to be run
- Main memory and registers are only storage where CPU can access directly
- Register access in one CPU clock or less
- Main memory can take many cycles
- Cache sits between main memory and CPU registers
- Protection of memory required to ensure correct operation

Operating Systems
Course Code: MCC510

Base and Limit Registers:

- A pair of base and limit registers define the logical address space.



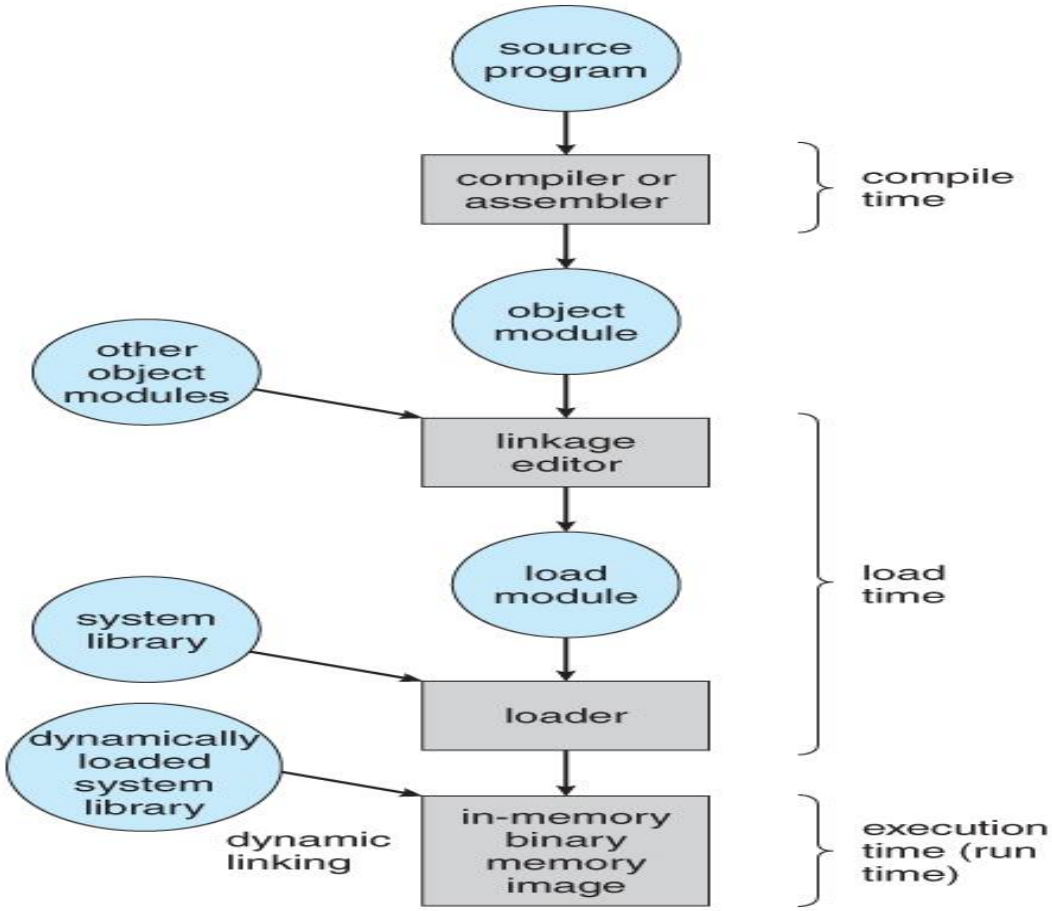
Operating Systems
Course Code: MCC510

Binding of Instructions and Data to Memory:

- Address binding of instructions and data to memory addresses can happen at three different stages:
- **Compile time -** If memory location known a priori, **absolute code** can be generated; must recompile code if starting location changes
- **Load Time -** If memory location is not known at compile time, must generate relocatable code
- **Execution time -** Binding delayed until run time if the process can be moved during its execution from one memory segment to another. Need hardware support for address maps (e.g., base and limit registers)

Operating Systems
Course Code: MCC510

Multistep Processing of a User Program :



Logical and Physical Address Spaces :

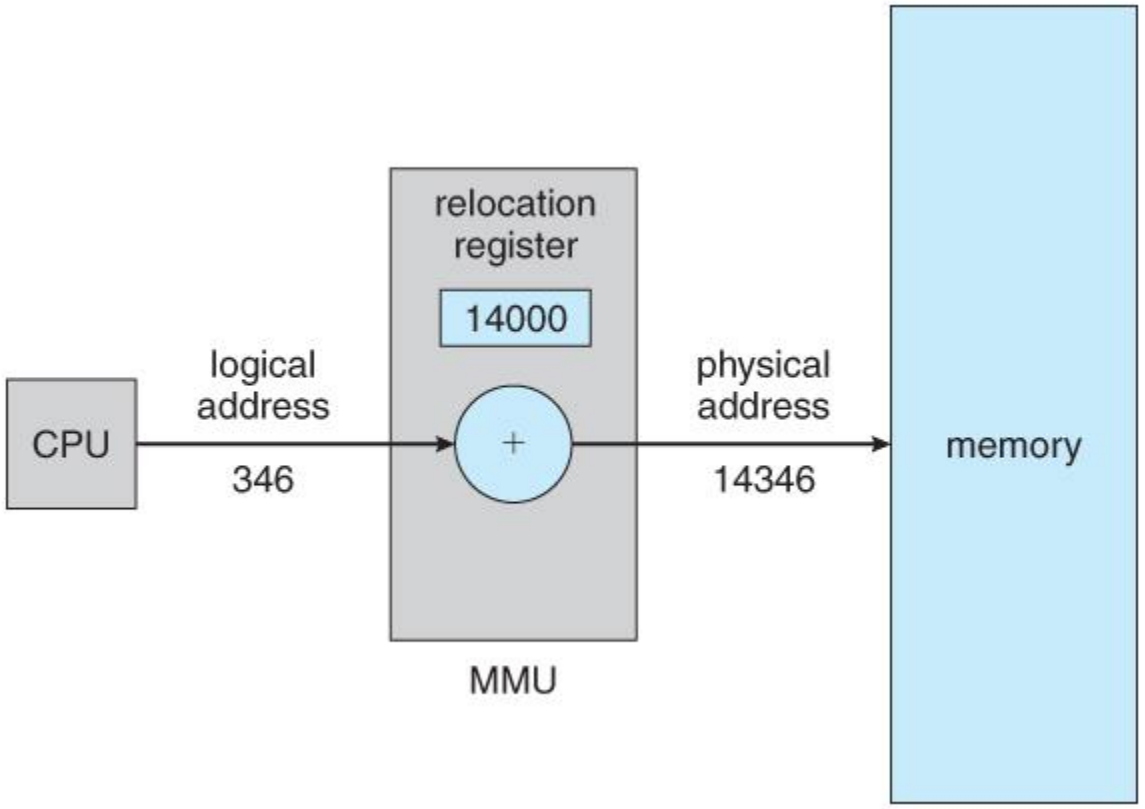
- The concept of a logical address space that is bound to a separate **physical address space** is central to proper memory management
- **Logical address** – generated by the CPU; also referred to as **virtual address**
- **Physical address** – address seen by the memory unit
- Logical and physical addresses are the same in compile-time address-binding schemes; logical (virtual) and physical addresses differ in execution- time address-binding scheme.

MMU:

- Hardware device that maps virtual to physical address
- In MMU scheme, the value in the relocation (base) register is added to every address generated by a user process at the time it is sent to memory
- The user program deals with *logical* addresses; it never sees the *real* physical addresses

Operating Systems
Course Code: MCC510

Dynamic Relocation Using a Relocation Register:



Dynamic Loading:

- Routine is not loaded until it is called.
- For better memory-space utilization; unused routine is never loaded
- For useful when large amounts of code are needed to handle infrequently occurring cases
- No special support from the operating system is required, implemented through program design

Dynamic Linking:

- It is postponed until execution time
- Small piece of code, stub, used to locate the appropriate memory resident library routine
- Stub replaces itself with the address of the routine and execute the routine.
- OS need to check if routine is in processes memory address or not.
- It is particularly useful for libraries
- System is also known as shared library

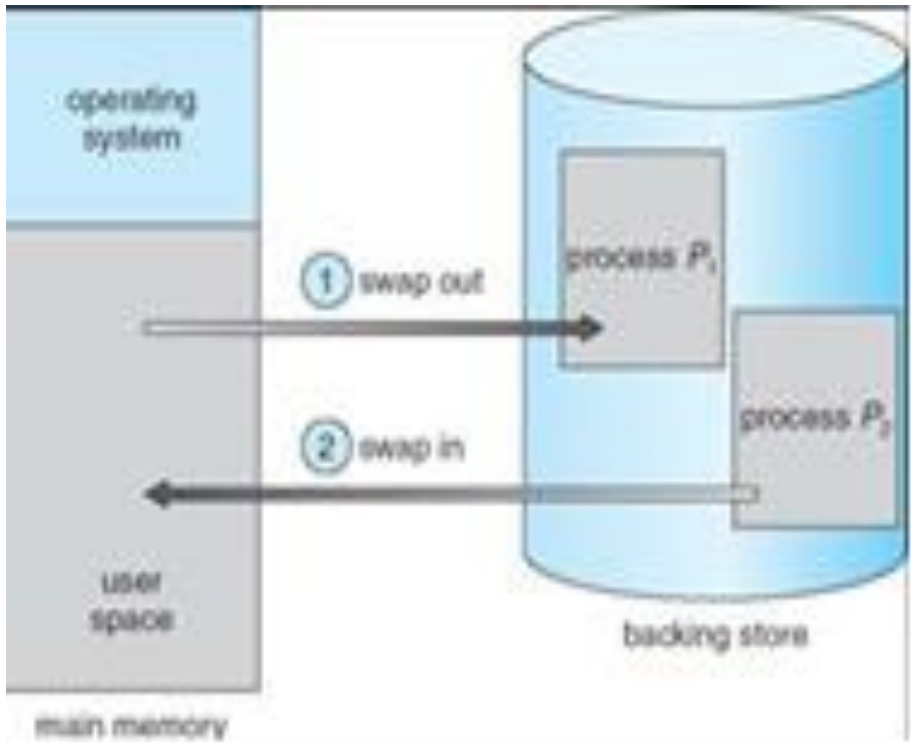
Operating Systems
Course Code: AMC15106

Swapping:

- A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution
- **Backing store** –disk large enough to accommodate copies of all memory images for all users
- **Roll out, roll in** – swapping variant used for priority- based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed
- Major part of swap time is transfer time. The total transfer time is directly proportional to the amount of memory swapped
- Modified version of swapping are found on many systems such as UNIX,LINUX and Windows
- System maintains a **ready queue** of ready-to-run processes which have memory images on disk

Operating Systems
Course Code: AMC15106

Swapping:



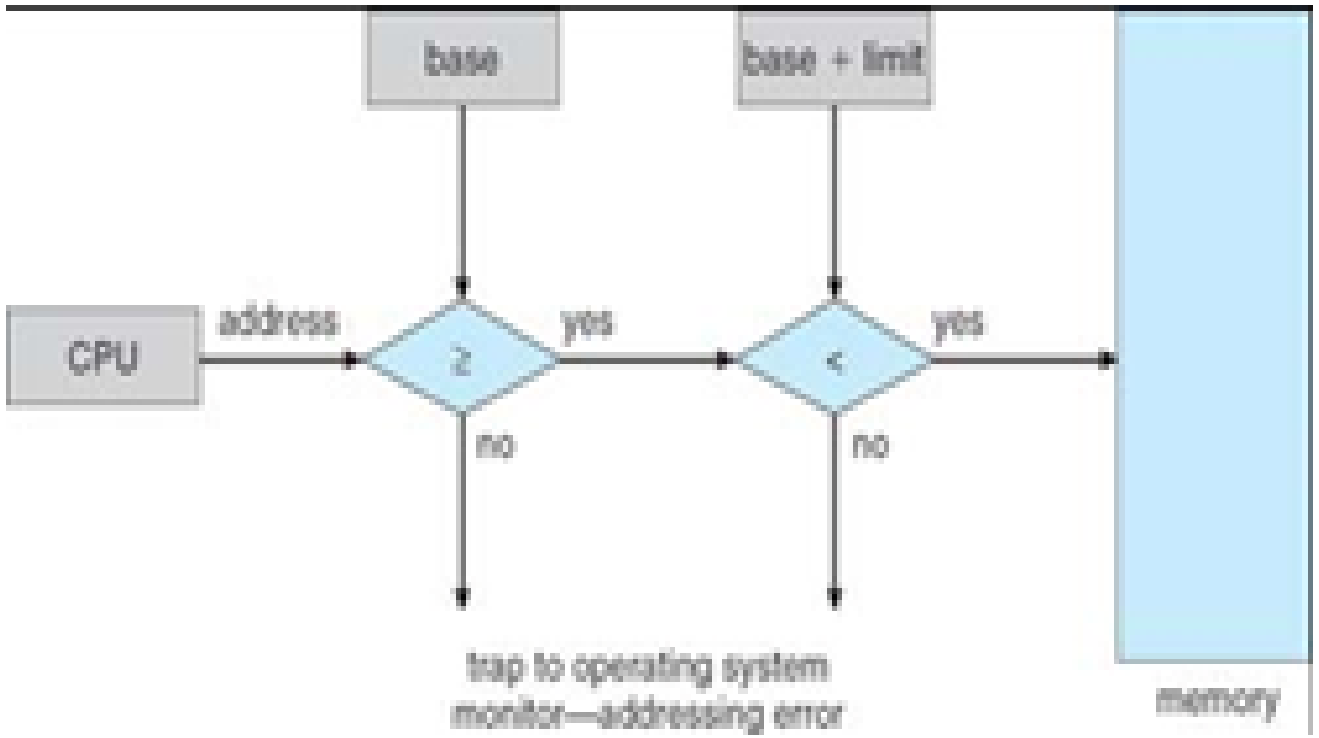
Operating Systems
Course Code: AMC15106

Contiguous Allocation:

- Main memory usually divided into two partitions:
  - Resident operating system, usually held in low memory
  - User processes then held in high memory
- Relocation registers used to protect user processes from each other, and from changing operating-system code and data
  - Base register contains value of smallest physical address
  - Limit register contains range of logical addresses – each logical address must be less than the limit register
  - MMU maps logical address *dynamically.*

Operating Systems
Course Code: AMC15106

HW Address Protection with Base and Limit Registers:



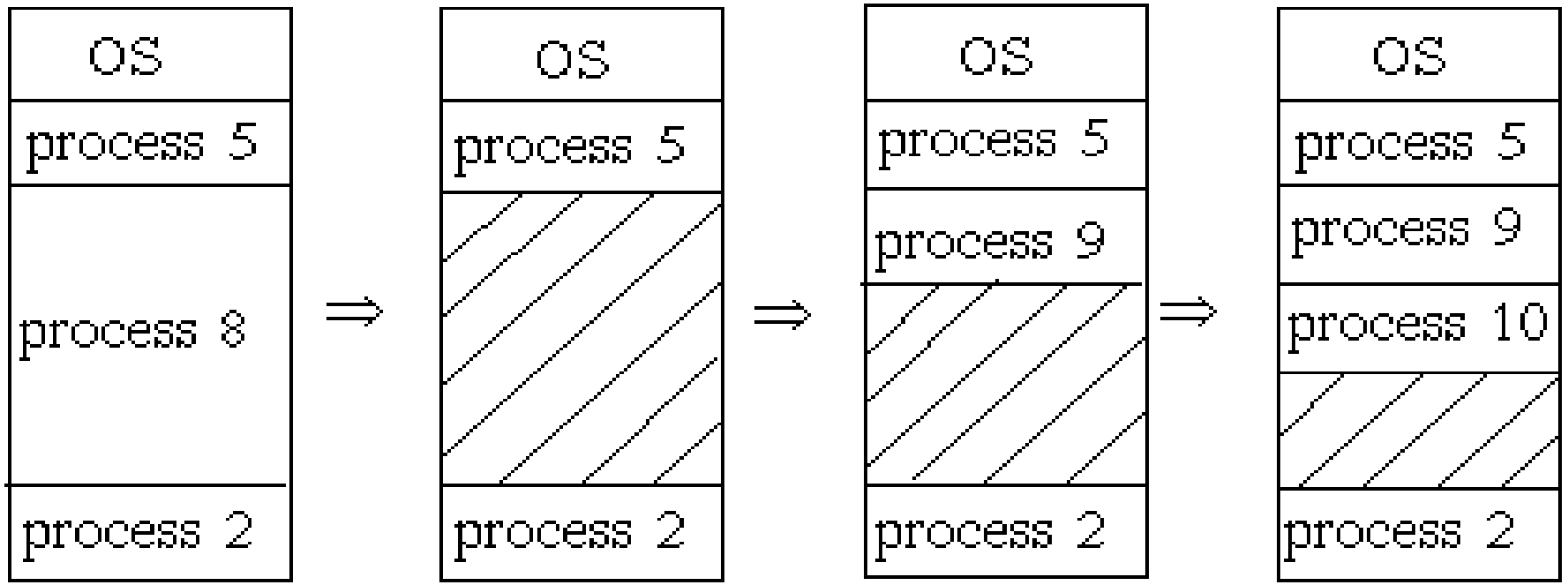
Operating Systems
Course Code: AMC15106

Contiguous Allocation continued:

- Multiple-partition allocation
  - Logical +relocation
- Hole - block of available memory; holes of various size are scattered throughout memory
- When a process arrives, it is allocated memory from a hole large enough to accommodate it
- Operating system maintains information about:
  - a) allocated partitions b) free partitions (hole)

Operating Systems
Course Code: AMC15106

Contiguous Allocation continued:



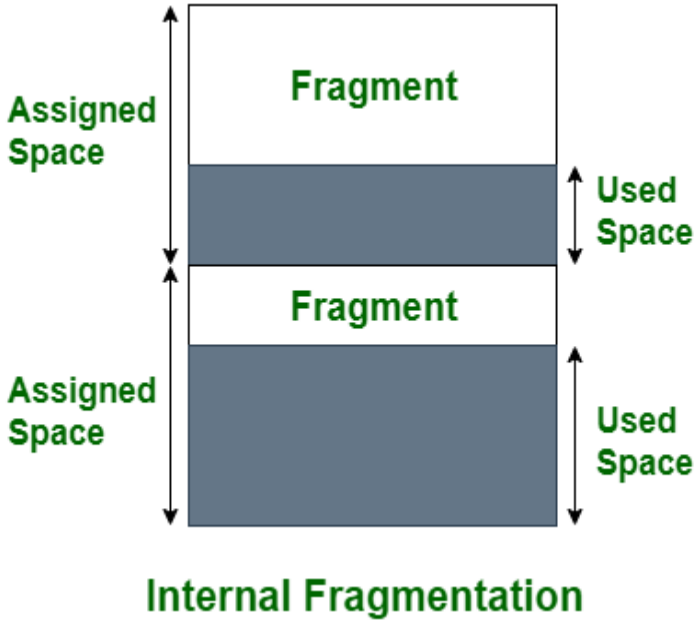
Dynamic Storage Allocation Problem:

- How to satisfy a request of size *n* from a list of free holes
  - **First-fit**: Allocate the *first* hole that is big enough
  - **Best-fit**: Allocate the *smallest* hole that is big enough; must search entire list
    - Produces the smallest leftover hole
  - **Worst-fit**: Allocate the *largest* hole; must also search entire list
    - Produces the largest leftover hole
- First-fit and best-fit better than worst-fit in terms of speed and storage utilization

Operating Systems
Course Code: AMC15106

Fragmentation:

- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used.

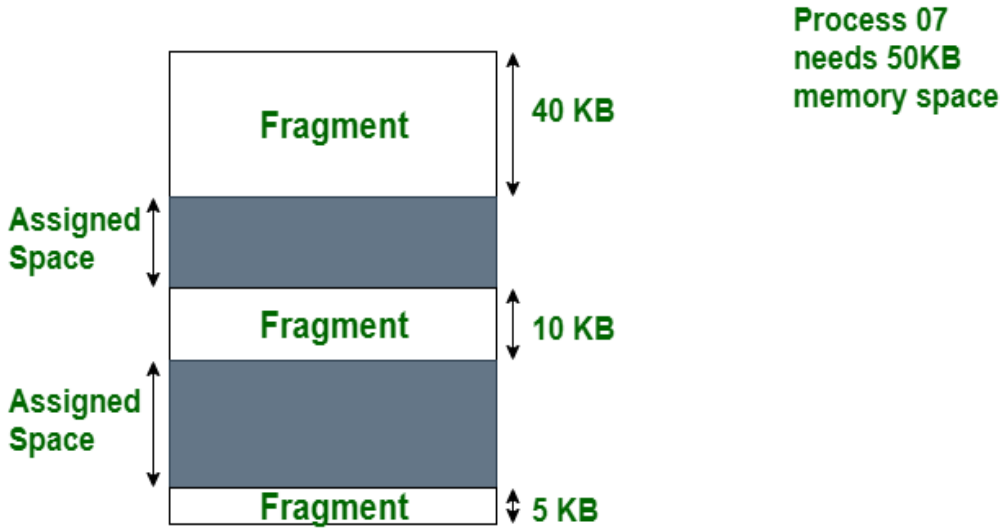


Internal Fragmentation

Operating Systems
Course Code: AMC15106

Fragmentation:

- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous
- Reduce external fragmentation by **compaction**
  - Shuffle memory contents to place all free memory together in one large block
  - Compaction is possible *only* if relocation is dynamic, and is done at execution time



Operating Systems
Course Code: AMC15106

Difference Between Internal and External Fragmentations:

- In internal fragmentation fixed-sized memory, blocks square measure appointed to process. However, In external fragmentation, variable-sized memory blocks square measure appointed to method.
- Internal fragmentation happens when the method or process is larger than the memory. However, External fragmentation happens when the method or process is removed.
- The solution of internal fragmentation is best-fit block. However, Solution of external fragmentation is compaction, paging and segmentation.
- Internal fragmentation occurs when memory is divided into fixed sized partitions. However, External fragmentation occurs when memory is divided into variable size partitions based on the size of processes.
- The difference between memory allocated and required space or memory is called Internal fragmentation. However, The unused spaces formed between non-contiguous memory fragments are too small to serve a new process, is called External fragmentation.

Operating Systems
Course Code: AMC15106

External Fragmentation Due to variable Size Partitioning:

- P1=300, P2=25, P3=125, P4=50

Occupied		Occupied		Occupied
50	150	300	350	600

- First Fit

Occupied	P2 25	P3 125	Occupied	P1 300	P4 50	Occupied

Operating Systems
Course Code: AMC15106

External Fragmentation Due to variable Size Partitioning:

- P1=300, P2=25, P3=125, P4=50

Occupied		Occupied		Occupied
50	150	300	350	600

- Best Fit

Occupied	P3 125	25	Occupied	P1 300	P2 25	25	Occupied

Operating Systems
Course Code: AMC15106

External Fragmentation Due to variable Size Partitioning:

- P1=300, P2=25, P3=125, P4=50

Occupied		Occupied		Occupied
50	150	300	350	600

- Worst Fit

Occupied	P2 25	P3 125	Occupied	P1 300	P4 50	Occupied

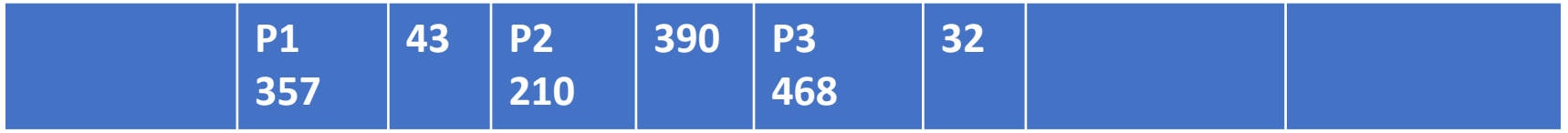
Operating Systems
Course Code: AMC15106

Internal Fragmentation Due to Fixed Size Partitioning:

- P1=357, P2=210, P3=468, P4=491

200 400 600 500 300 250

- First Fit



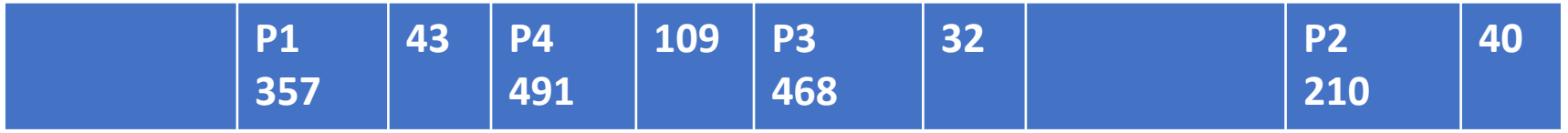
Operating Systems
Course Code: AMC15106

Internal Fragmentation Due to Fixed Size Partitioning:

- P1=357, P2=210, P3=468, P4=491

200 400 600 500 300 250

- Best Fit



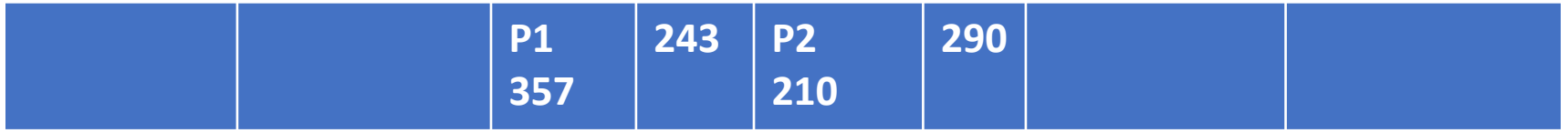
Operating Systems
Course Code: AMC15106

Internal Fragmentation Due to Fixed Size Partitioning:

- P1=357, P2=210, P3=468, P4=491

200 400 600 500 300 250

- Worst Fit



Operating Systems
Course Code: AMC15106

Paging:

- Logical address space of a process can be non contiguous; process is allocated physical memory whenever the latter is available.
- Divide physical memory into fixed-sized blocks called **frames** (size is power of 2).
- Divide logical memory into blocks of same size called **pages.**
- Keep track of all free frames.
- To run a program of size ***n*** pages, need to find *n* free frames and load program.
- Set up a page table to translate logical to physical addresses.
- Internal fragmentation.

Operating Systems
Course Code: AMC15106

Address Translation Scheme:

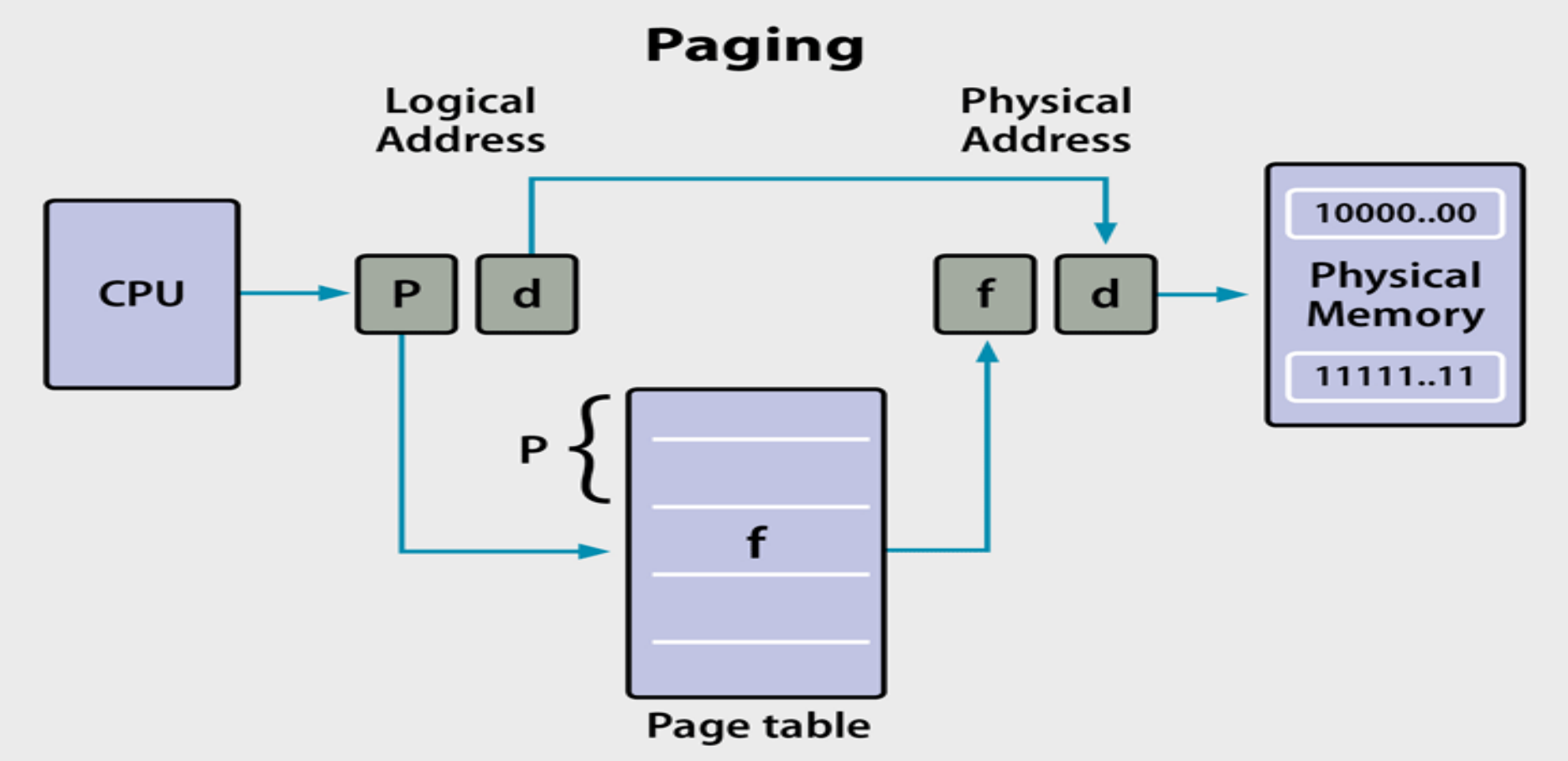
- Address generated by CPU is divided into:
- **Page number (*p*)** – used as an index into a *page table* which contains base address of each page in physical memory
- **Page offset (d)** – combined with base address to define the physical memory address that is sent to the memory unit

Page Number	Page Offset
p	d
m-n	n

- For given logical address space of size 2m and page size is 2n

Operating Systems
Course Code: AMC15106

Paging Hardware:



Operating Systems
Course Code: AMC15106

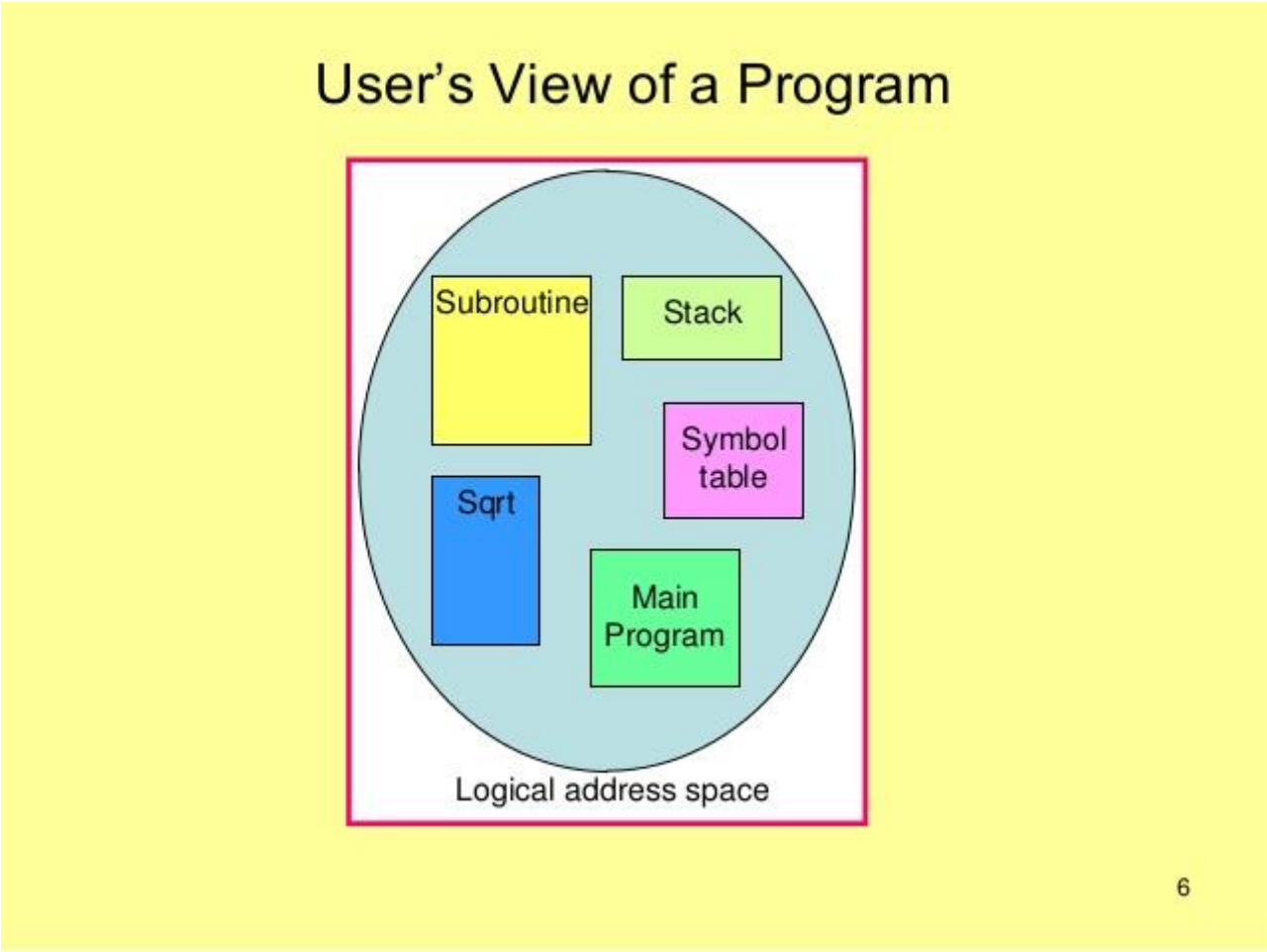
Segmentation:

- Memory-management scheme that supports user view of memory
- A program is a collection of segments. A segment is a logical unit such as

  Main program,
  procedure,
  function,
  method,
  object,
  local variables,
  global variables,
  common block,
  stack,
  symbol table, arrays

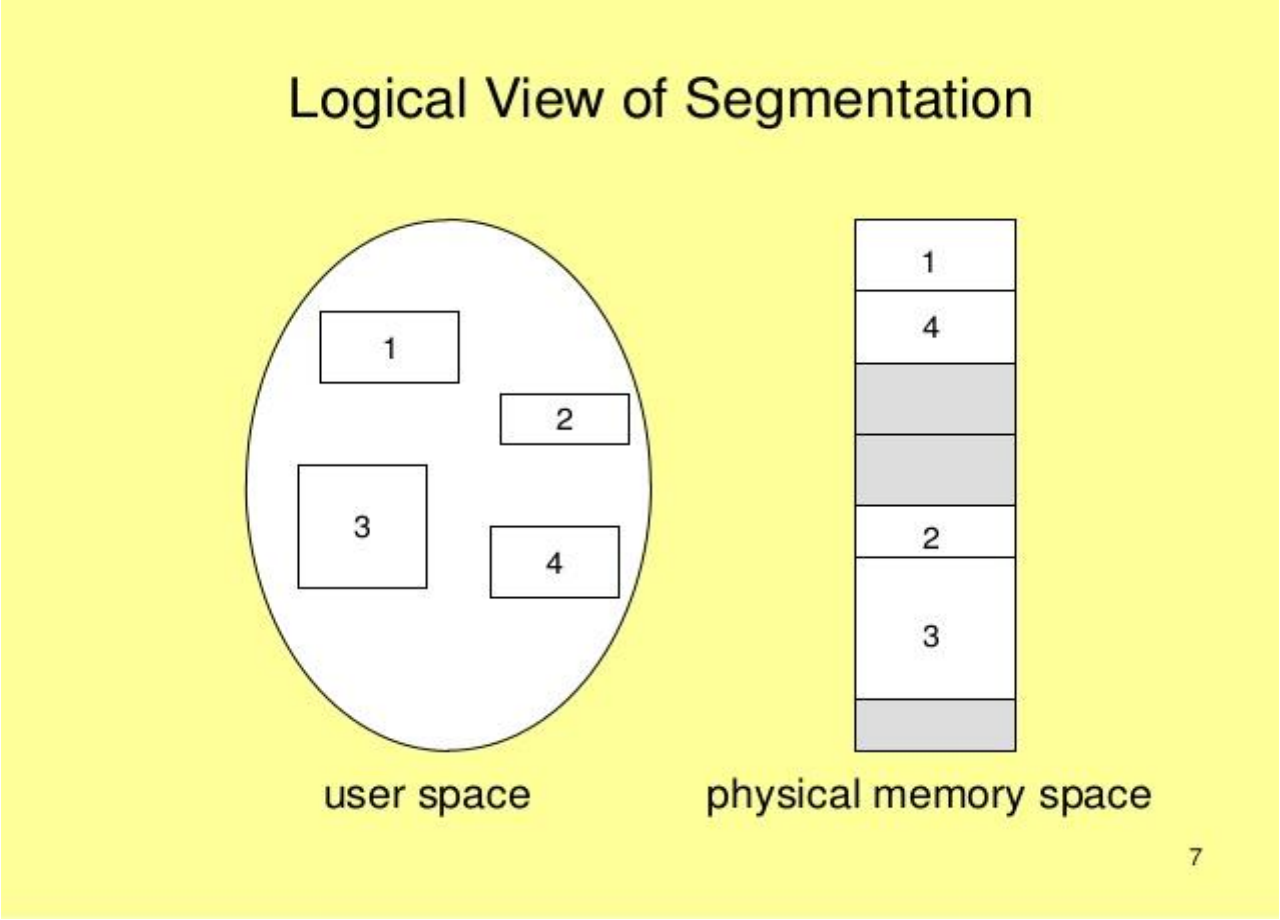
Operating Systems
Course Code: AMC15106

User view of Program:



Operating Systems
Course Code: AMC15106

Logical View of Segmentation:



Operating Systems
Course Code: AMC15106

Segmentation Architecture:

- Logical address consists of a two tuple:

  <segment-number, offset>

- **Segment table** – maps two-dimensional physical addresses; each table entry has:
  - **base** – contains the starting physical address where the segments reside in memory
  - **limit** – specifies the length of the segment
- **Segment-table base register (STBR)** points to the segment table's location in memory
- **Segment-table length register (STLR)** indicates number of segments used by a program;

  segment number ***s*** is legal if ***s*** < **STLR**

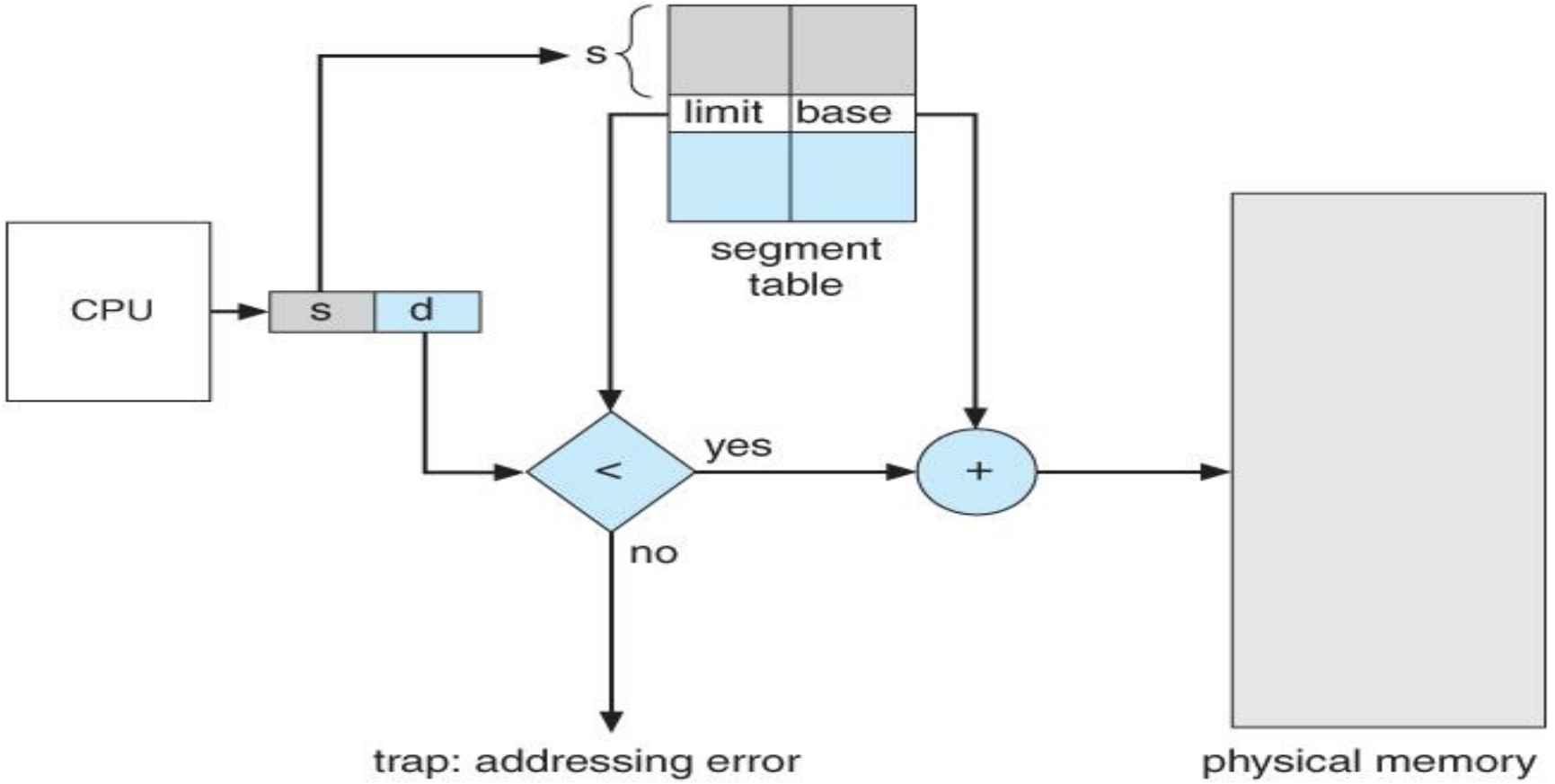
Operating Systems
Course Code: AMC15106

Segmentation Architecture:

- Protection
  - With each entry in segment table associate:
  - validation bit = 0 ⇒ illegal segment
  - read/write/execute privileges
- Protection bits associated with segments; code sharing occurs at segment level.
- Since segments vary in length, memory allocation is a dynamic storage-allocation problem.

Operating Systems
Course Code: AMC15106

Segmentation Hardware:



Operating Systems
Course Code: AMC15106

Virtual Memory:

- It involves separation of user logical memory from physical memory. This separation allows an extremely large virtual memory to be provided for programmers where only a smaller physical memory is available.
- Only part of the program needs to be in memory for execution.
- Logical address space > > Physical address space
- Allows address spaces to be shared by several processes and also allows for efficient process creation

Operating Systems
Course Code: AMC15106

Virtual Memory:

- Virtual memory can be implemented via Demand paging and Segmentation
- Demand Paging brings a page into memory only when it is needed, less I/O is needed, less memory is needed, faster response and more users.
- Page is needed which means that reference to it. It will be invalid reference that may abort or not in memory which brings into memory
- V is a valid bit- associated page in the LA is both legal and in memory
- I is invalid bit-the page either is not valid(not in LA) or is valid but currently on the disk
- Lazy swapper-never swaps a page into memory unless page is needed.
- Swapper that deals with pages is a pager.

Page Fault:

- If there is a reference to a page, first reference to that page will trap to operating system: page fault
- OS looks at another table to decide invalid reference or just not in memory
- Get empty frame
- Swap page into frame
- Reset tables
- Set validation bit v
- Restart the instruction that cause page fault

Performance of Demand Paging:

- If page fault rate p is lying between 0 and 1 then
  - if p=0 then no page fault
  - if p=1 then every reference is a page fault
- Effective Access Time(EAT)

  =(1-p)* memory access + p * (page fault overhead + swap page out + swap page in + restart overhead)

What happens when there is no free frame?

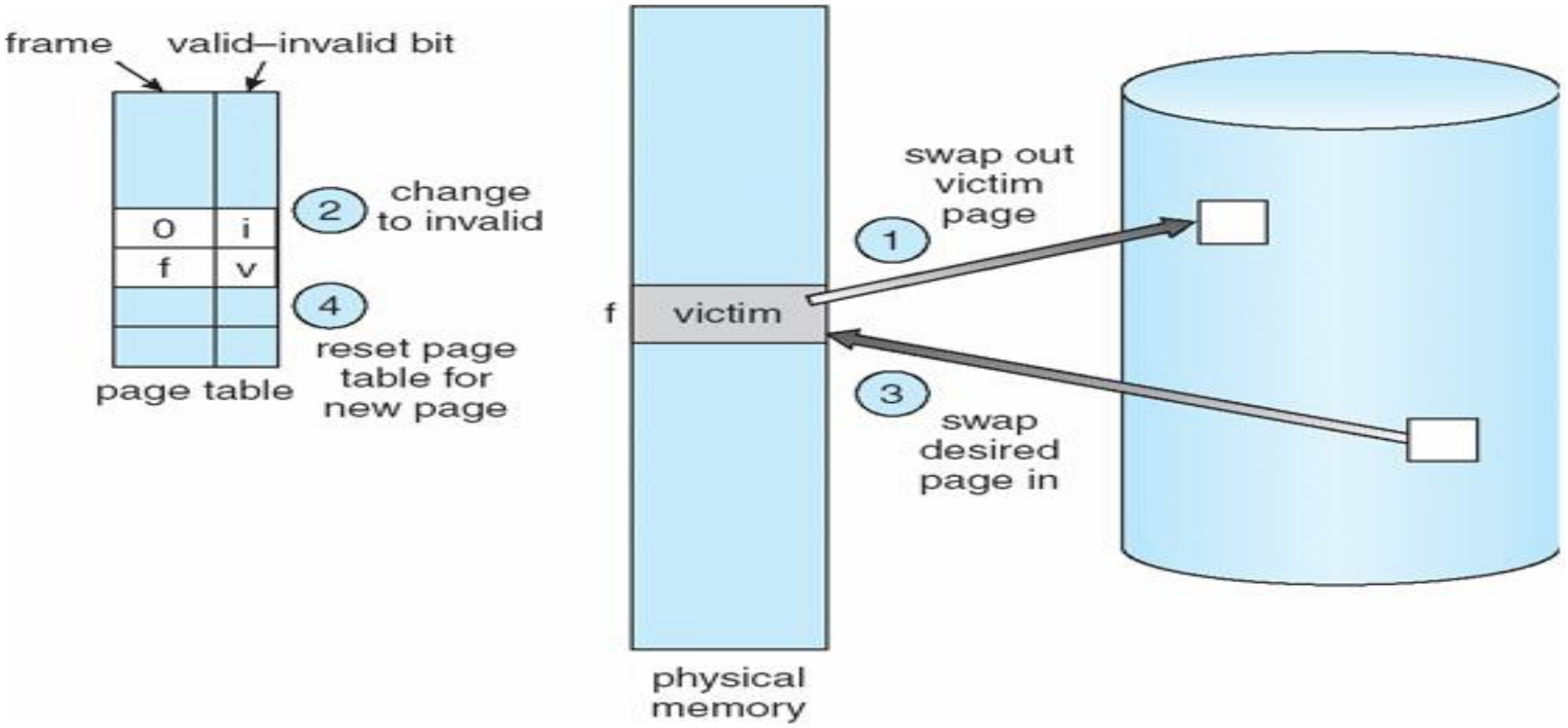
- Page replacement- Find some pages in memory but not really in use, swap it out.
- Algorithm
- Performance- minimum number of page fault.
- Same page may be brought into memory several times
- Prevent user allocation of memory by modifying page fault service routine to include page replacement.
- Use modify (dirty) bit to reduce overhead of page transfers-only modified pages are written to disk.
- Page replacement completes separation between logical memory and physical memory. Large virtual memory can be provided on a smaller physical memory.

Basic for Page Replacement Algorithm:

- Find the location of desired page on disk
- Find a free frame
- If there is a free frame then use it otherwise use page replacement algorithm, to select victim frame
- Bring the desired page into the newly free frame and update the page and frame tables
- Restart the process

Operating Systems
Course Code: AMC15106

Basic for Page Replacement Algorithm:



Operating Systems
Course Code: AMC15106

Page Replacement Algorithm:

- Algorithm evaluation will be made by running it on a particular string of memory references (reference string) and computing the no. of page faults on that string.
- The reference string is considered as
  - 1, 2, 3, 4, 1, 2, 5, 1, 2, 3,4,5
- FIFO algorithm
- Consider 3 frame (3 pages can be in memory at a time per process)

1	4	5
2	1	3
3	2	4

- 9 page fault

Operating Systems
Course Code: AMC15106

Page Replacement Algorithm:

- The reference string is considered as
  - 1, 2, 3, 4, 1, 2, 5, 1, 2, 3,4,5
- FIFO algorithm
- Consider 4 frame (4 pages can be in memory at a time per process)

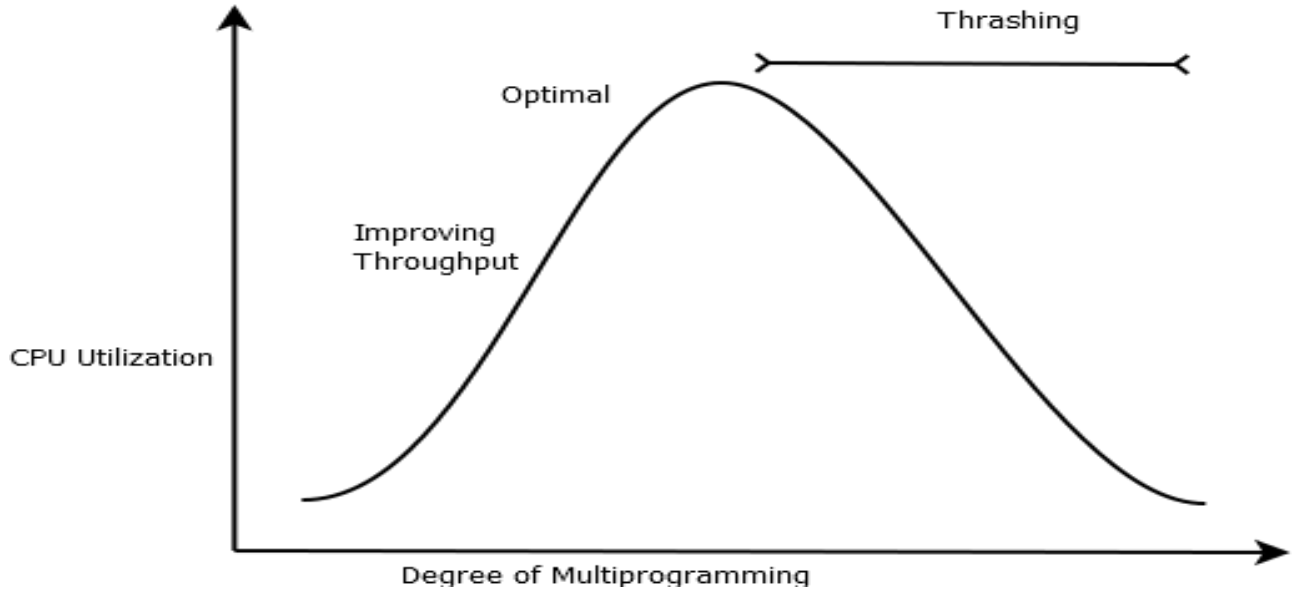
1	5	4
2	1	5
3	2	
4	3	

- 10 page fault
- Belady's Anomaly: More frames, more page fault

Operating Systems
Course Code: AMC15106

Thrashing:

- If a process does not have enough pages then the page fault rate is very high which leads to low CPU utilization.
- OS thinks that it needs to increase the degree of multiprogramming and another process added to the system.
- A process is busy swapping pages in and out is known as thrashing.
- Thrashing occurs if the sum of size of locality is greater than total memory size.



Operating System Lab Manuals

Subject: Operating Systems Lab Code: MCC510 L T P: 0-0-3

1. Introduction to Shell Programming
2. Syntax, various commands, algorithm for Shell Programming
3. Execution of Shell Programming
4. Shell Programming continued
5. Programming based on Processes and Threads
6. Processes and Threads continued
7. CPU Scheduling algorithms-FCFS & SJF
8. CPU Scheduling algorithms-RR & Priority
9. Programming based on Deadlock
10. Programming based on Deadlock continued

Unix Commands	**Description**
cd	Change directory
cd-	Return to previous directory
mkdir	Make directory
find	Find files
cat	It display file contents
pwd	To know about present working directory
ls	List all files in a directory
ls -l	List all files in long format, one file per line
ls -a	List all files including hidden files
mv	To rename the existing file
cp	To copy one or more files
man	It displays the manual pages for a chosen Unix command
rm	It removes files or directories
echo	It displays a line of text on the screen
clear	Clear the screen
who	Displays data about all the user have logged into the system currently

Exercise: Enter these commands at the UNIX prompt, and try to interpret the output:

i) echo hello world
ii) passwd
iii) date
iv) hostname
v) uname -a
vi) uptime
vii) who am i
viii) who
ix) id

x) last
xi) finger
xii) top (you may need to press q to quit)
xiii) echo $SHELL
xiv) man "automatic door"
xv) man ls (you may need to press q to quit)
xvi) man who (you may need to press q to quit)
xvii) lost
xviii) clear
xix) cal 2000
xx) bc -l (type quit or press Ctrl-d to quit)
xxi) echo 5+4 | bc -l
xxii) history

1. Write a shell script program to read two numbers and perform basic arithmetic operations (+ , - , * , / , %)

Algorithm:
Step 1: Start
Step 2: Read two integers a, b
Step 3: Calculate Sum= a + b
Diff= a – b
Product= a * b
Div=a / b
Rem=a % b
Step 4: Display Sum, Diff, Product, Div and Rem
Step 5: Stop

2. Write a shell script to read three integer numbers and print the largest among three numbers.

Algorithm:
Step 1: Start
Step 2: Declare variables a, b and c.
Step 3: Read variables a, b and c.
Step 4: if a>b
if a>c
Display a is the largest number.
else
Display c is the largest number.
else
if b>c
Display b is the largest number.
else
Display c is the greatest number.
Step 5: Stop

3. Write a shell script program to read a character from keyboard and check whether it is vowel or not.

Algorithm:
Step1: Start
Step2: Declare variable ch.
Step3: Read the value of ch.
Step4: if (ch=='A' || ch=='a' || ch=='E' || ch=='e'|| ch=='I' || ch=='i'|| ch=='O' || ch=='o' || ch=='U' || ch == 'u') then
Display "Entered character is Vowel"
goto Step 6
else
Step5: Display "Entered character is not Vowel"
goto Step 6
Step 6: Stop

4. Write a shell script to print out the Fibonacci series up to a limit.

Algorithm:
Step 1: Start
Step 2: Declare variables n, a ← 0, b ← 1, c, i
Step 3: Read values of n
Step 4: Display a, b
Step 5: Assign i←2
Step 6: if i < n then goto step 7 otherwise goto step10
Step 7: calculate c ← a+b,
i ← i+1
a ← b, b ← c
Display the value of c
goto step 6
Step 10: Stop

5. To write a shell script to check whether the given number is prime or not.

Algorithm:
Step 1: Start
Step 2: Read an integer n
Step 3: Assign i=2, j=0
Step 4: Is i < n then r =n % i. otherwise goto step 8
Step 5: Is r=0 then increment i and j value by i. otherwise go to step 6
Step 6: Increment i value by one
Step 7: Is j=0 then print number is prime and goto step 10
Step 8: Is j != 0 then print number is not a prime number
Step 9: Stop

6. To write a shell script to find the Armstrong numbers between 1 to N.

Algorithm:
Step 1: Start
Step 2: When i equal to 0 and less than or equal to N, calculate increment value of i.
Step 3: Assign value of i to temp and n.

Step 4: Assign value of ams equal to zero.
Step 5: When n not equal to zero calculate
rem←n%10;
ams=ams+rem*rem*rem
n←n/10
Step 6: If temp equal to ams then print the value of ams.
Step 7: Thus for each value of i, values of ams is printed.
Step 8: Stop the program.

7. Write a shell script to perform Conversion of temperature in Celsius to Fahrenheit and Fahrenheit to Celsius.

Algorithm:
Step 1: Start
Step 2: Input the choice as 1 or 2
Step 3: Is choice is 1 then goto step 4 otherwise goto step 7
Step 4: Input temperature in Celsius
Step 5: Calculate Fahrenheit F =((9/5)*c) +32
Step 6: Print Fahrenheit F and goto step 10
Step 7: Input temperature in Fahrenheit
Step 8: Calculate Celsius C=((5/9)*(f-32))
Step 9: Print Celsius C
Step 10: Stop

8. Write a shell script to read an integer find out the reverse of the integer using function and check whether integer is palindrome or not.

Algorithm:
Step 1: Start
Step 2: read n
Step 3: copy n into m for later use. Also, initialize rn;
Step 5: while n is not zero
1. r = n % 10
2. n = n/10
3. rn = rn*10 + r;
Step 6: if m equal rn then the number is palindrome.
Step 7: Else Print number is not palindrome
Step 8: Stop

9. Write a shell script to read an integer find out the factorial of the integer.

Algorithm:
Step1: Start
Step2: Read a number ‘n’ and fact=1
Step3: if n==1 then
Return (1)
Step4: else
For i=0 to i<n
Factorial=fact*fact(n-1)
Return(fact)

Step4: Stop

10. Write a shell script program to read an array of 'n' integers and perform linear search operation.

Algorithm:
Step 1: Start
Step 2: Read the array A of 'n' elements, f=0
Step 3: Read the element 'x' to be searched in A
Step 4: Set i to 0
Step 5: if i > n then go to step 10
Step 6: if A[i] = x then f=1 and go to step 9
Step 7: Set i to i + 1
Step 8: Go to Step 5
Step 9: Print Element x Found at index i+1 and go to step 11
Step 10: if f=0 then Print element not found
Step 11: Stop

11. Write a shell script program to read an array of 'n' integers and sort number in ascending order using bubble sort technique.

Algorithm:
Step1: Start
Step2: Read the number of array elements
Step3: for i = 0 to n-1
Read array[i]
Step4: for i = 0 to n-1
for j = 0 to n-i-1
if (array[i]>array[j+1]) then
Temp=array[j]
Array[j]=array[j+1]
Array[j+1]=temp
Step7: Display array elements
Step8: Stop

12. Write a shell script program to read an array of 'n' integers and perform binary search.

Algorithm:
Step 1: Start
Step 2: Read the array a of n elements, f=0
Step 3: Sort using any algorithm
Step 4: Read the element to be searched in x
Step 5: Set L=0 the lower limit and u=n-1 the upper limit
Step 6: Repeat the steps 7, 8, 9, 10 until u>=L
Step 7: mid = (L+u)/2
Step 8: when a[mid]==x f=1 print the search is successful, display the position goto step 12
Step 9: when a[mid]<x L=mid+1
Step 10: when a[mid]>x u=mid-1
Step 11: if f==0 print search is unsuccessful

Step 12: Stop

SCHEDULING ALGORITHMS

1. First Come First Serve Scheduling (FCFS Scheduling)

i) Jobs are executed on first come and first serve basis
ii) It is a non pre-emptive scheduling algorithm
iii) It is easy to understand and implement
iv) Its implementation is based on first in first out (FIFO) queue
v) It is poor in performance as average waiting time is high

AIM: To write the program to implement CPU & scheduling algorithm for first come first serve scheduling.

Algorithm:

1. Start the program.
2. Get the number of processes and their burst time.
3. Initialize the waiting time for process 1 and 0.
4. Process for (i=2; i<=n; i++),wt.p[i]=p[i-1]+bt.p[i-1].
5. The waiting time of all the processes is summed then average value time is calculated.
6. The waiting time of each process and average times are displayed
7. Stop the program

2. Shortest Job First Scheduling (SJF Scheduling)

i) It is a non pre-emptive scheduling algorithm
ii) It is best approach to minimize waiting time
iii) It is easy to implement in batch systems where required CPU time is known in advance
iv) Impossible to implement in interactive systems where required CPU time is not known
v) The processor should know in advance how much time process will take

To write a program to implement cpu scheduling algorithm for shortest job first scheduling.

Algorithm:

1. Start the program. Get the number of processes and their burst time.
2. Initialize the waiting time for process 1 as 0.
3. The processes are stored according to their burst time.
4. The waiting time for the processes are calculated as follows:
   for(i=2;i<=n;i++).wt.p[i]=p[i=1]+bt.p[i-1].
5. The waiting time of all the processes summed and then the average time is calculate
6. The waiting time of each processes and average time are displayed.
7. Stop the program.

3. Priority Scheduling

i) SJF scheduling is special case of priority scheduling
ii) Priority is associated with each process
iii) CPU is allotted to the process with the highest priority

iv) For the case of equal priority, processes are scheduled on the basis of FCFS
v) It is a non pre-emptive scheduling
vi) Priority can be decided based on memory or time requirements or any other resource requirements

To write a 'C' program to perform priority scheduling.

Algorithm:
1. Start the program.
2. Read burst time, waiting time, turn the around time and priority.
3. Initialize the waiting time for process 1 and 0.
4. Based up on the priority process are arranged
5. The waiting time of all the processes is summed and then the average waiting time
6. The waiting time of each process and average waiting time are displayed based on the priority.
7. Stop the program.

4. Round Robin Scheduling

i) It is pre-emptive process scheduling algorithm
ii) Each process is provided a fix time to execute, it is called a quantum
iii) Once a process is executed for a given time period, it will be pre-empted at that given time and other process will execute for a given time period

To write a program to implement CPU scheduling for Round Robin Scheduling.

Algorithm:
1. Get the number of process and their burst time.
2. Initialize the array for Round Robin circular queue as '0'.
3. The burst time of each process is divided and the quotients are stored on the round Robin array.
4. According to the array value the waiting time for each process and the average time are calculated as line the other scheduling.
5. The waiting time for each process and average times are displayed.
6. Stop the program.

PIPE PROCESSING

To write a program for create a pipe processing

Algorithm:
1. Start the program.
2. Declare the variables.
3. Read the choice.
4. Create a piping processing using IPC.
5. Assign the variable lengths
6. "strcpy" the message lengths.
7. To join the operation using IPC.
8. Stop the program

SIMULATE ALGORITHM FOR DEADLOCK PREVENTION

Algorithm:

1. Start the program
2. Attacking Mutex condition: never grant exclusive access. But this may not be possible for several resources.
3. Attacking preemption: not something you want to do.
4. Attacking hold and wait condition: make a process hold at the most 1 resource
5. At a time. Make all the requests at the beginning. Nothing policy. If you feel, retry.
6. Attacking circular wait: Order all the resources.
7. Correct order so that there are no cycles present in the resource graph. Resources numbered 1 ... n.
8. Resources can be requested only in increasing
9. Order. i.e., you cannot request a resource whose no is less than any you may be holding.
10. Stop the program

Sd.
(M K Singh)
Professor/M&C