A NEW & EASY WAY TO LEARN

AzDocuments.in

Get All Vtu 18th Scheme notes for all branches here

UNIT -1

OS - notes.

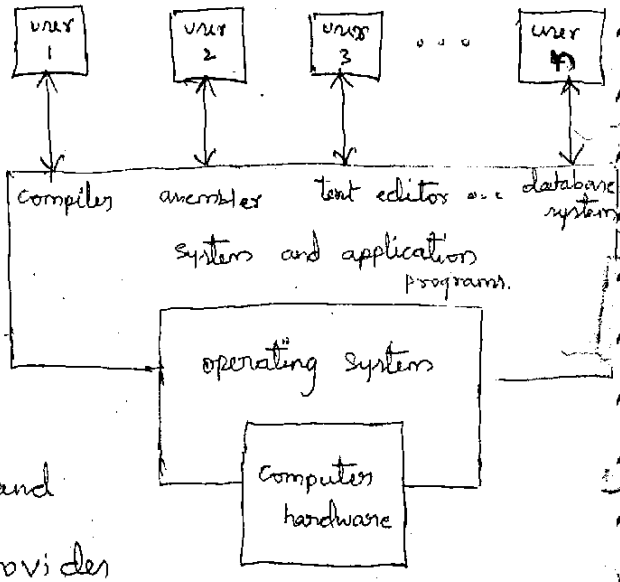
Introduction

Definition of OS :- An operating system is a program that manages the computer hardware. It also provides a basis for application programs and acts as an intermediary between the computer user and the computer hardware.

1.1) What Operating Systems Do (Roll of O.S.)

A computer system can be divided roughly into four components. as shown below.

(a) The hardware
(b) The operating system
(c) The application programs
(d) The users.



a) The hardware :- The central processing unit (CPU), the memory, and the input/output (I/O) devices, provides the basic computing resources for the system.

b) The application programs :- Such as word Processors, spread sheets, compilers, and web browsers.

The operating system controls and coordinates the use of the hardware among the various application programs for the various users. Operating system provides an environment within which other programs can do useful work.

1.1.1) User View :-

*) The user's view of the computer varies according to the interface being used. (PC, laptop, mini computer, mainframe etc)

*) Personal Computer is designed for one user to monopolize (obtain) its resources.

*) The goal is to maximize the work that the user is performing.
Max. rep use
mainframe, minicomputer -> share resources & exchange info
Workstations :- dedicated resource & share also info. & rep. lots
*) Operating System is designed mostly for ease of use.
handheld :- performance battery life

*) Some attention paid to performance and none paid to resource utilization. OS -> program running all times on computer [kernel]

1.1.2) System View :-

*) Operating System is the program most intimately (confidentially) involved with the hardware.

*) This content says Operating system as a resource allocator. [An OS is similar to a government, a resource allocator and a control program. As a resource allocator, it acts as manager of resources (hardware & software) and allocates them to specific programs and users as necessary for tasks.]

*) The operating system acts as the manager of resources.

*) An operating system is a control program which manages the execution of user programs to prevent errors.

1.2) Computer-System Organization :- (Structure of Computer system)

1.2.1) Computer-System Operation :-

A modern general-purpose computer system consists of one or more CPUs and a number of device controllers

2.2) Storage structure :-

The CPU can load instructions only from main memory (RAM), so any programs to run must be stored there.

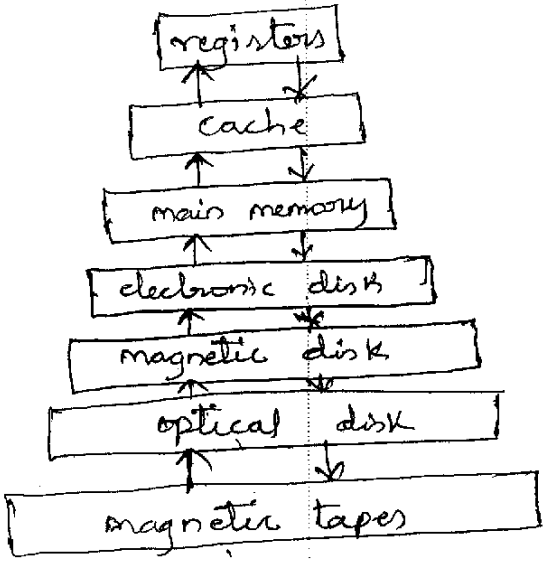
Main memory commonly is implemented in a semiconductor technology called dynamic random-access memory (DRAM).

All forms of memory provide an array of words. Each word has its own address.

The load instruction moves a word from main memory to an internal register within the CPU, whereas the store instruction moves the content of a register to main memory.

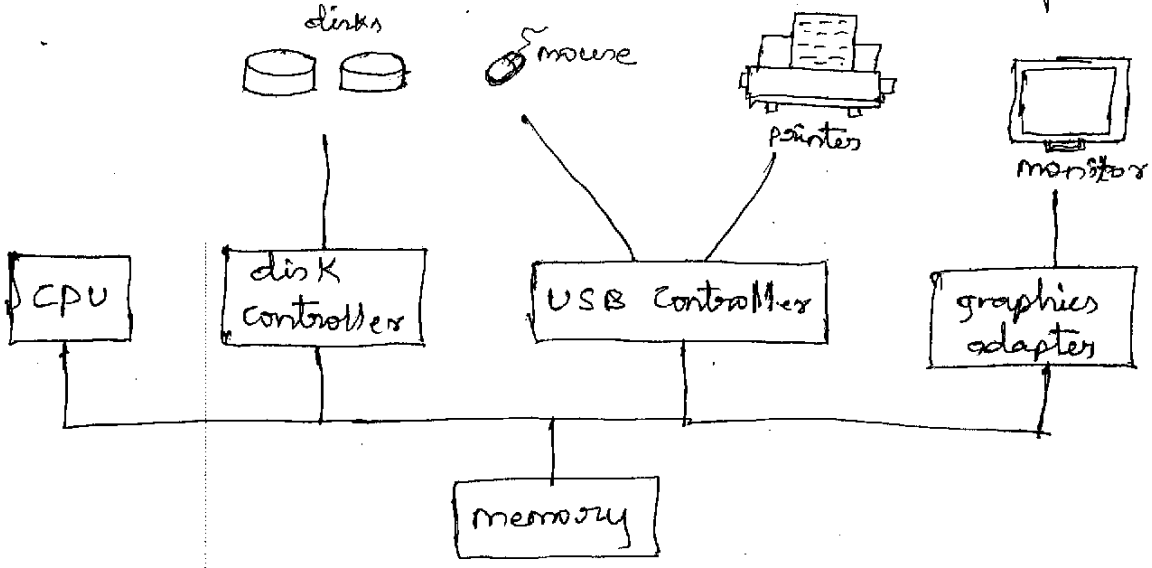
Von-Neumann architecture, first fetches an instruction from memory and stores that instruction in the instruction register. The instruction is then decoded and may cause operands to be fetched from memory and stored in some internal register. After the instruction on the operands has been executed, the result may be stored back in memory (RAM).

The wide variety of storage systems in a computer system can be organized in a hierarchy according to speed and cost. The higher levels are expensive, but they are fast. The top 4 levels of memory in figure may be constructed using semiconductor memory.



storage device hierarchy.

connected through a common bus that provides access to shared memory. As shown in the following figure



) When a computer is powered up it needs to have an initial program to run. This initial program is known as bootstrap program, tends to be simple.

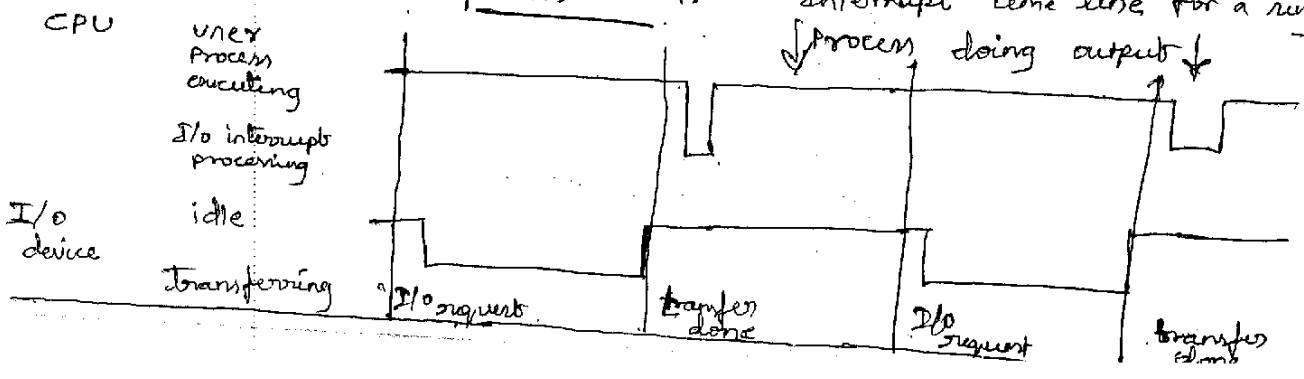
Typically, it is stored in read-only memory (ROM). In general term it is called as firmware.

The operating system starts executing the first process, such as "init", and waits for some event to occur.

Occurrence of an event is usually signaled by an interrupt from either the hardware or the software.

Hardware may trigger an interrupt at any time by nding a signal to the CPU.

Software may trigger an interrupt by executing a special peration called a system call.



1.2.3) I/O structure :-

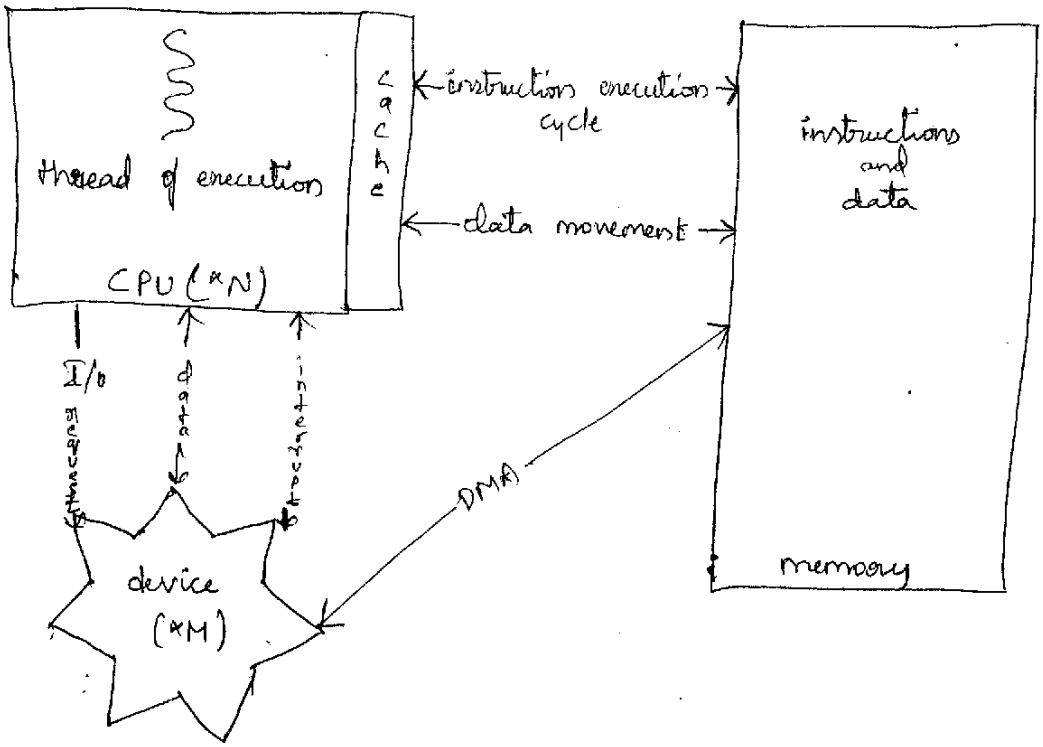
*) Storage is only one of many types of I/O devices wi a computer.

*) A large portion of operating system code is dedicated managing I/O because of performance, reliability and the varying nature of the devices.

*) More than one device may be attached to the SCSI (small computer-systems interface) controller. ↳ set of standards for physically connecting data b/w computer & peripheral

*) The device controller of the I/O device transfers an enti block of data directly to or from its own buffer storag to memory, with no intervention by the CPU, is called a "direct memory access" (DMA).

~~1.2.4) Computer → System Architecture~~ :-



How a modern computer system works.

1.3) Computer - System Architecture :-

Computer system can be categorized roughly according to the number of general-purpose processors used.

1.3.1) Single Processor Systems :-

*) Most systems use a single processor.

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

*) All systems have other Special-purpose processors called device specific processors.

*) These special-purpose processors run a limited instruction set and do not run user processes.

*) The operating system cannot communicate with special-purpose processors, they do their jobs autonomously.

*) The use of special purpose microprocessors is common & does not turn a single-processor system into a multiprocessor

1.3.2) Multiprocessor Systems :-

*) Multiprocessor systems also known as parallel systems, tightly coupled systems.

Multiprocessor systems have three main advantages :

(1) Increased throughput :-

*) By increasing the number of processors, we expect to get more work done in less time.

*) When multiple processors cooperate on a task, a certain amount of overhead is incurred in keeping all the parts working correctly.

(2) Economy of Scale :-

*) Multiprocessor systems can cost less than equivalent multiple single-processor systems, because they can share peripherals, mass storage, and power supplies.

*) If several programs operate on the same set of data, it is cheaper to store those data on one disk.

(3) Increased reliability :-

*) If functions can be distributed properly among several processors, then the failure of one processor will not halt the system, only slow it down.

*) If we have ten processors and one fails, then each of the remaining nine processors can pick up a share of the work of the failed processor. Thus the entire system runs only 10 percent slower, rather than failing altogether.

The ability to continue providing service proportional to the level of surviving hardware is called graceful degradation.

Some systems go beyond graceful degradation and are called fault tolerant

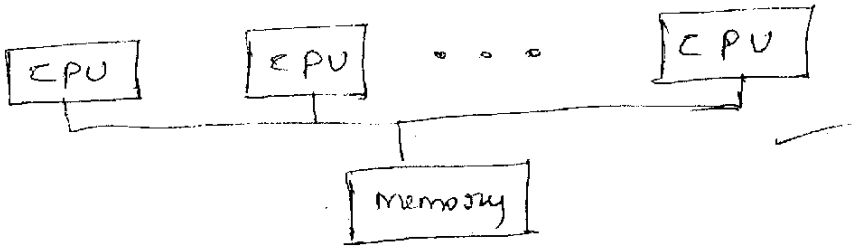
The multiprocessor systems are of two types.

(1) Asymmetric multiprocessing

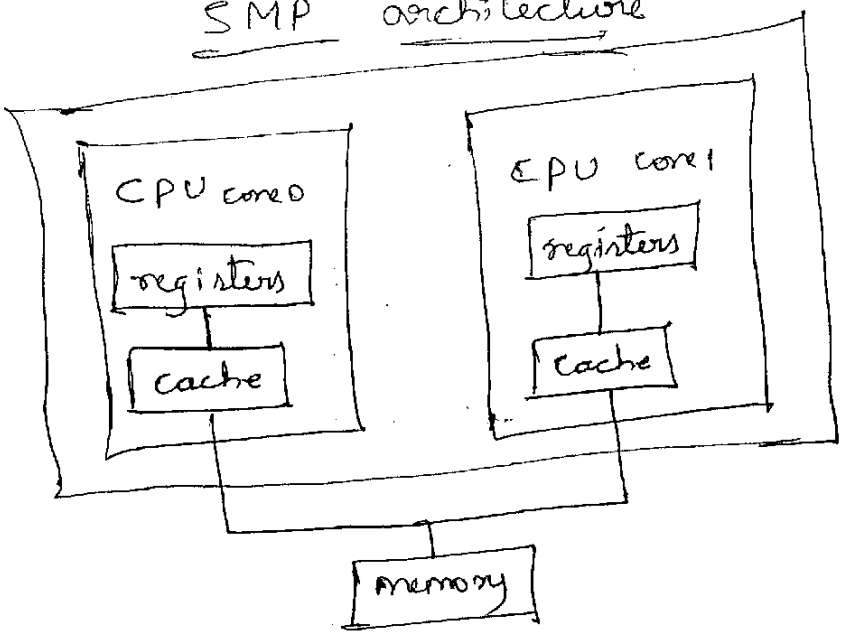
(2) Symmetric multiprocessing

(1) Asymmetric multiprocessing :- Here each processor is assigned a specific task. A master processor controls the system. It defines the master slave relationship. The master processor schedules and allocates work to the slave processors.

(2) Symmetric multiprocessing :- Here each processor performs all tasks within operating system. In SMP means that all processors are peers; no master-slave relationship exists between processors.



SMP architecture



A dual-core design with 2 cores placed on the same chip.

The recent trend in CPU design is to include multiple computing cores on a single chip.

* Multiprocessing adds CPUs to increase computing power
* If the CPU has an integrated memory controller, then adding CPUs can also increase the amount of memory addressable in the system.
* A recent trend in CPU design is to include multiple computing cores on a single chip.
* Finally, blade servers are a recent development in which multiple processor boards, I/O boards, and networking boards are placed in the same chassis.

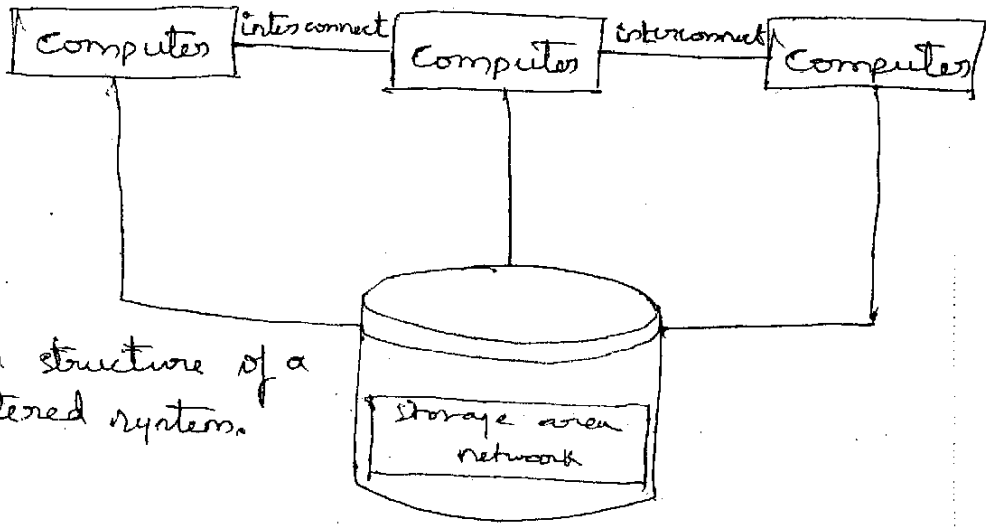
1.3.3) Clustered Systems :-

* Another type of multiple-CPU system is the clustered system.
* Clustered systems differ from multiprocessor systems, however, in that they are composed of two or more individual systems.
* Clustered computers share storage and are closely linked via a local-area network (LAN) or InfiniBand.
* Clustering is usually used to provide high-availability service that is service will continue even if one or more systems in the cluster fail.
* Clustering can be structured
  (a) asymmetrically or
  (b) symmetrically

- Asymmetrical Clustering :- In asymmetric clustering, one machine in hot stand by mode, while the other is running the applications. The hot standby host machine does nothing but monitor the active server. If that server fails, the hot standby host becomes the Active server.

- Symmetric mode Clustering :- In symmetric mode, two & more hosts are running applications and are monitoring each other. This mode is obviously more efficient, as it uses all of the available hardware. More than one application be available to run.

*) Parallelization :- Which consists of dividing a programme into seperate components that run in parallel on individual computers in the cluster.



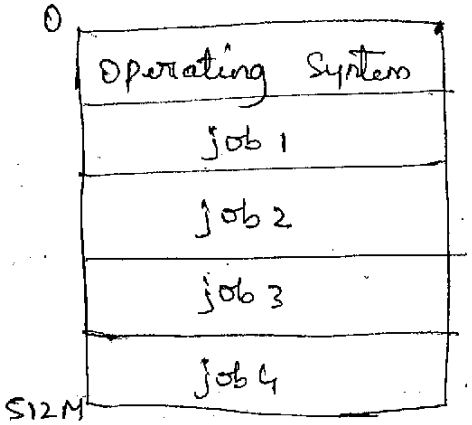
General structure of a clustered system.

:) Many of these improvements are made possible by storage area networks [SANs].

4) Operating System Structure

:) One of the most important aspects of operating system is the ability to multi programme.

- A single programme cannot, in general, keep either the CPU & the I/O devices busy at all times.
- Multiprogramming increases CPU utilization by organising jobs so that the CPU always has one to execute.
- The operating system keeps several jobs in memory simultaneously. Since, in general, main memory is too small to accommodate all jobs, the jobs are kept initially on the disk in the job pool.



Memory layout for a multiprogramming system.

- In a non-multiprogrammed system, the CPU would sit idle. In a multiprogrammed system, the operating system simply switches to, and executes another job.
- Time sharing is a logical extension of multiprogramming. In time sharing systems, the CPU executes multiple jobs by switching among them, but the switches occur so frequently that the users can interact with each program while it is running.
- A program loaded into memory and executing is called a process.
- If several jobs are ready to be brought into memory, and if there is not enough space for all of them, then the system must choose among them. Making this decision is job scheduling and it is done by job scheduler.

-) In a time sharing system, the operating system must ensure reasonable response time, which is some times accomplished through swapping, where processes are swapped in and out of main memory to the disk. A more common method for achieving this goal is virtual memory.

) The main advantage of virtual memory scheme is that it enables users to run programs that are larger than actual physical memory.

.5) Operating-System Operations :-

*) modern operating systems are interrupt driven.

) Events are almost always signaled by the occurrence of an interrupt or a trap.

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

) If a process gets stuck in an infinite loop, this loop could prevent the correct operation of many other processes, where one error program might modify another program, the data of another program, or even the operating system itself.

.5.1) Dual-mode operation :-

*) In order to ensure the proper execution of the operating system, we must be able to distinguish between the execution of operating system code and user defined code.

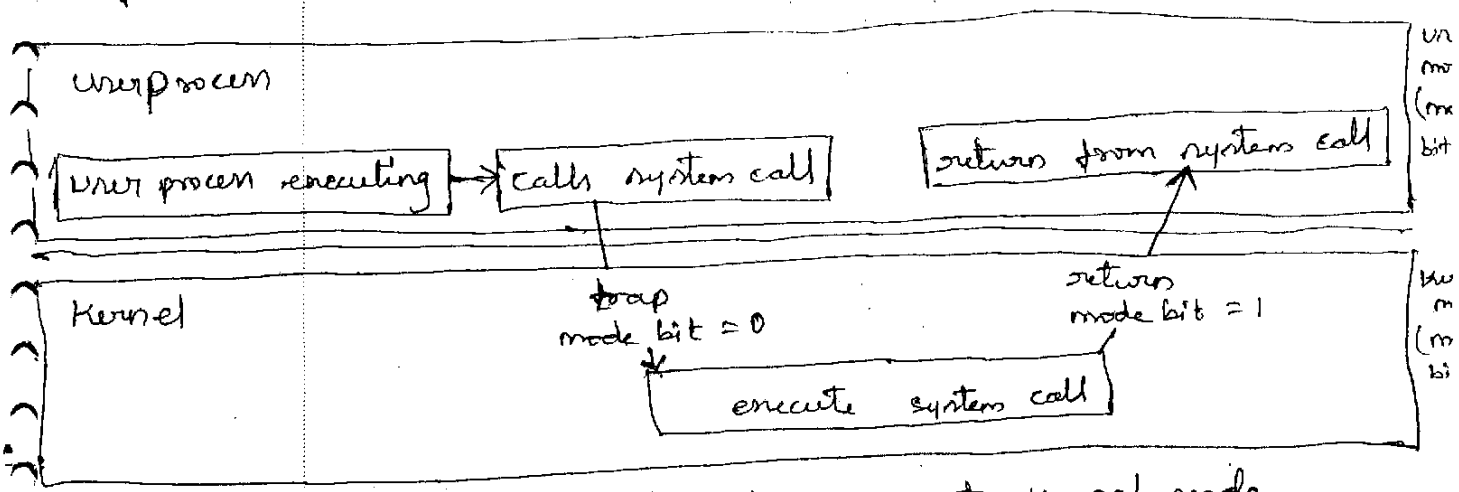
* At the very least, we need two separate modes of operation

1) User mode
2) Kernel mode (supervisor mode, system mode)

* A bit called the mode bit, is added to the hardware of the computer to indicate the current mode. Kernel (0), user (1).

* With the mode bit, we are able to distinguish between a task that is executed on behalf of the operating system and one that is executed on behalf of the user.

* When the computer system is executing on behalf of a user application, the system is in user mode. However, when a user application requests a service from the operating system, it must transit from user to kernel mode to fulfill the request.



Transition from user to kernel mode.

* At system boot time, the hardware starts in kernel mode. The operating system is then loaded and starts user applications in user mode. Whenever a trap or interrupt occurs, the hardware switches from user mode to kernel mode.

*) The lack of a hardware-supported dual mode can cause serious shortcomings in an operating system. For example :- MS-Dos was written for the Intel 8088 architecture, which has no mode bit and therefore no dual mode. A user program can change or write over the operating system code. (lack of security).

1.5.2) Timer :-

*) we can use the timer to prevent a user program from running too long.

*) We cannot allow a user program to get stuck in an infinite loop or to fail to call system services and never return control to the operating system. To accomplish this goal, we can use a timer.

*) A timer can be set to interrupt the computer after a specified period. This period may be fixed or variable

Ex:- A program with a 6-minute time limit would have its counter initialized to ~~120~~ 360. Every second, the timer interrupts and the counter is decremented by 1. As long as the counter is positive, control is returned to the user program. When the counter becomes negative, the operating system terminates.

1.2) Process Management :-

- A program in execution, is a process. → Active entity
- A user program (passive entity) such as a compiler is a process.
- A process needs certain resources - including CPU time, memory, files and I/O devices, to accomplish its task.
- When the process terminates, the operating system will reclaim any reusable resources.
- A program by itself is not a process. A program is a passive entity, like the contents of a file stored on disk, whereas a process is an active entity.
- A single threaded process has one program counter specifying the next instruction to execute.

The operating system is responsible for the following activities in connection with process management.

1) Scheduling processes and threads on the CPUs.
2) Creating and deleting both user and system processes.
3) Suspending and resuming processes.
4) Providing mechanism for process synchronization. (same time, speed)
5) Providing mechanism for process communication.
6) Providing mechanism for deadlock handling

1.3) Memory Management :-

- The main memory is central to the operation of a modern computer system.

*) main memory is a large array of words or bytes, ranging in size from hundreds of thousands to billions.

1) Each word or byte has its own address.

2) The main memory is generally the only large storage device that the CPU is able to address and access directly

The operating system is responsible for the following activities in connection with memory management.

1) Keeping track of which parts of memory are currently being used and by whom.
2) Deciding which process and data to move into and out of memory.
3) Allocating and deallocating memory space as needed.

1.8) Storage Management :-

*) The operating system defines a logical storage unit as a file.

1.8.1) File-System Management :-

*) File management is one of most visible components of an operating system.

*) Computers can store information on several different types of physical media. Magnetic disk, optical disk, and magnetic tape are the most common.

*) A file is a collection of related information defined by its creator.

* When multiple users have access to files, it may be desirable to control by whom and in what ways (for eg. read, write, append) files may be accessed.

The operating system is responsible for the following activities in connection with file management:

1) Creating and deleting files.
2) Creating and deleting directories to organize files.
3) Supporting primitives for manipulating files and directories.
4) Mapping files onto secondary storage.
5) Backing up files on stable (nonvolatile) storage media.

1.8.2) Mass - Storage Management :- (Non-volatile devices)

* Main memory is too small to accommodate all data and programs, and because the data that it holds are lost when power is lost, the computer system must provide secondary storage to backup main memory.
* Most modern computer systems use disks as the principal on-line storage medium for both programs and data.
* Most programs including compilers, assemblers, word processors, editors and formatters are stored on a disk until loaded into memory and then use the disk as both the source and destination of their processing.

The operating system is responsible for the following activities is Connection with disk management.

1) Free space management
2) storage allocation
3) Disk scheduling.

*) Secondary storage is used frequently, so it must be used efficiently.

*) Magnetic tape drives and their tapes and CD and DVD drives and platters are typical tertiary storage devices.

1.8.3) Caching :-

*) Caching is an important principle of computer systems.

*) The most recent information used is stored in cache memory.

*) When CPU needs a particular piece of information, it first checks whether it is in cache. If it is, CPU use the information directly from the cache; if it is not, CPU use the information from the source. (it might be in hard disk).

*) Caches have limited size, so cache management is an important design problem. Careful selection of the cache size and of a replacement policy can result in greatly increased performance.

Performance of Various levels of storage

level	1	2	3	4
Name	registers	cache	main memory	disk storage
Typical size	< 1 KB	> 16 MB	> 16 GB	> 100 GB
Implementation technology	CMOS Custom memory with multiple ports	CMOS SRAM	CMOS DRAM	magnetic disk
Access time (ns)	0.25 - 0.5	0.5 - 25	80 - 250	5000.000
Bandwidth [MB/sec]	20,000 - 100,000	5000 - 10,000	1000 - 5000	20 - 150
Managed by	Compiler	hardware	operating system	operating system
Backed up by	Cache	Main memory	disk	CD or tape

Performance of various levels of storage.

*) main memory can be viewed as a fast cache for secondary storage, since data in secondary storage must be copied into main memory for use, and data must be in main memory before being moved to secondary storage for safekeeping.

migration of integer A from disk to register.

In a hierarchical storage structure, the same data may appear in different levels of the storage system. For example, suppose that an integer A that is to be incremented by 1 is located in file B, the file B resides on magnetic disk. The increment operation proceeds by first issuing an I/O operation to copy the disk block on which A resides to main memory. This operation is followed by copying A to

the cache and to an internal register. Thus, the copy of A appears in several places : on the magnetic disk, in main memory, in the cache, and in an internal register. Once the increment takes place in the internal register, the value of A differs in the various storage systems. The value of A becomes the same only after the new value of A is written from the internal register back to the magnetic disk.

8.4) I/O Systems :-

*) One of the purposes of an operating system is to hide the peculiarities (characters) of specific hardware devices from the user.

The I/O subsystem consists of several components :

1) A memory - management component that includes buffering, caching and spooling.
2) A general device - driver interface.
3) Drivers for specific hardware devices.

Only the device driver knows the peculiarities of the specific device to which it is assigned.

9) protection and Security :-

*) If a computer system has multiple users and allows the concurrent execution of multiple processes, then access to data must be regulated.

* Mechanisms ensure that the resources can be operated by the processes that have gained proper authorization from the operating system.

* Ex. memory addressing hardware ensures that a process can execute only within its own address space.

* The timer ensures that no process can gain control of the CPU without eventually relinquishing (final, quit) control

* Device control registers are not accessible to users, so the integrity (goodness) of the various peripheral devices is protected.

* Protection is a mechanism for controlling the access of processes / users to the resources defined by a computer system.

* Security :- The job of security is to defend a system from external and internal attacks.

* Security should protect the system from the attacks such as viruses, worms, denial-of-service (which use all of a system's resources and so keep legitimate users out of the system)

* Protection and Security require the system to be able to distinguish among all its users. most operating systems maintain a list of user IDs and associated passwords.

1.10) Distributed Systems :-

*) A distributed system is a collection of physically separate (possibly heterogeneous) computer systems networked to provide the users with access to the various resources that the system maintains.

*) Advantages of distributed System :-

1) Increases computation speed
2) Increases functionality
3) Increases data availability
4) Increases reliability.

) A network, in the simplest terms, is a communication path between two or more systems.

Distributed systems depend on networking for their functionality.

Networks are characterized based on the distances between their nodes.

1) A local area network (LAN) connects computers within a room, a floor, or a building. (Ethernet, Wi-Fi)
2) A wide-area network (WAN) usually links buildings, cities, or countries. A global company may have a WAN to connect its offices worldwide. (telephone) (www)
3) A Metropolitan-area network (MAN) could link buildings within a city. (VTU, BMS)
4) BlueTooth and 802.11 devices use wireless technology to communicate over a distance of several feet, creating small area networks. (SMA).

* A network operating system is an operating system that provid features such as file sharing across the network and that ~~has~~ includes a communication scheme that allows different processes on different computers to exchange messages.

1.11) Special-Purpose Systems :-

There are other classes of computer systems whose functions are more limited and whose objective is to deal with limited computation domains.

1.11.1) Real-Time Embedded Systems :-

* Embedded systems designed to work for very specific tasks.
* Operating system provide limited features.
* These devices are found everywhere, from car engines and manufacturing robots to DVD's and microwave ovens.
* Some embedded systems use general-purpose ~~computers~~ operating systems such as unix, others are hardware devices with a special-purpose embedded operating system providing just the functionality desired. Yet others are hardware devices with application-specific integrated circuits (ASICs) that perform their tasks without an operating system.
* Embedded systems almost always run real-time operating systems.

11.2) Multimedia Systems :-

*) Most operating systems are designed to handle conventional data such as text files, programs, word-processing documents, and spreadsheets. Ex: UNIX, MS-DOS

*) Multimedia describes a wide range of applications in popular use today. These include audio files such as MP3, DVD movies, video conferencing, and short video clips.

11.3) Handheld systems :-

*) Handheld systems include personal digital assistants (PDAs), such as palm and Pocket-PCs, and cellular telephones, many of which use special-purpose embedded operating systems.

*) The amount of physical memory in a handheld depends on the device, but typically it is somewhere between 1MB to 10GB. As a result the operating system and applications must manage memory efficiently.

*) Currently many Handheld devices such as Smartphones, Tablets use virtual memory techniques.

*) A second issue of concern to developers of handheld devices is the speed of the processor. used in the devices. processors for most handheld devices run at fraction of the speed of a processor. in a PC. Faster processor require more power.

*) The last issue confronting program designers for handheld devices is I/O. A lack of physical space limits input methods to small keyboards, handwriting recognition, or small screen-based keyboard.

*) Some handheld devices use wireless technology, such as Blue Tooth or 802.11, allowing remote access to e-mail and web browsing.

OS Concepts

1.12) Computing Environments :

This topic will conclude a brief overview of how these are used in a variety of computing environments.

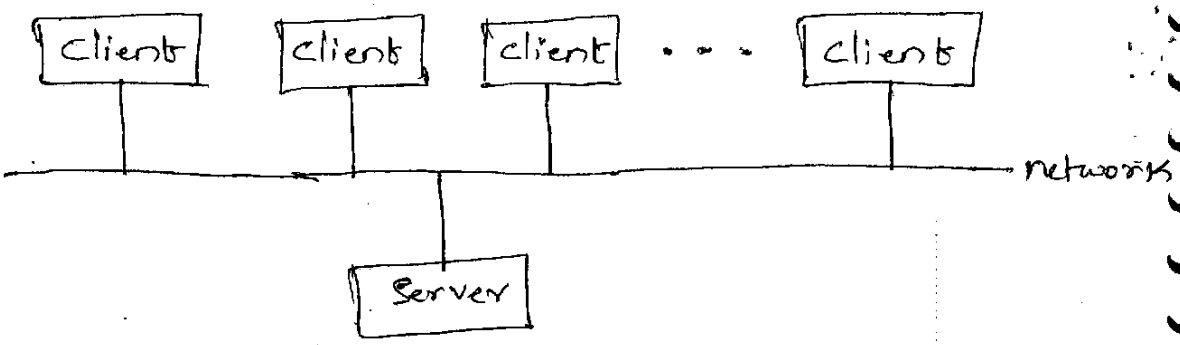
1.12.1) Traditional Computing :

*) Consider the "typical office environment." Just a few years ago, this environment consisted of PCs connected to a network, with servers providing file and print services. Portability was achieved by use of laptop computers.

*) At home, most users had a single computer with a slow modem connection to the office. Today, network-connection with huge bandwidth are relatively inexpensive.

1.12.2) Client-Server Computing :

All PCs have become faster, more powerful, and cheaper designers have shifted away from centralized system architecture.



General structure of a client-server system.

Server systems can be broadly categorized as :

1) Compute servers
2) File servers

1) The compute-server system provides an interface to which a client can send a request to perform an action (for example, read data); in response, the server executes the action and sends back results to the client

2) The file-server system provides a file-system interface where clients can create, update, read and delete files.

1.12.3) Peer-to-peer computing

Another structure for a distributed system is the peer-to-peer (P2P) system model. In this model, clients and servers are not distinguished from one another; instead all nodes within the system are considered peers, and each may act as either a client or a server, depending on

whether it is requesting or providing a service.

*) To participate in a peer-to-peer system, a node must first join the network of peers. Once a node has joined the network, it can begin providing services to and requesting services from other nodes in the network.

Determining what services are available is accomplished in one of two general ways:

1) When a node joins a network, it registers its service with a centralized lookup service on the network. Any node wants a specific service first contacts this centralized lookup service to determine which node provides the service.

2) A peer acting as a client must first discover what node provides a desired service by broadcasting a request for the service to all other nodes in the network. peer can use discovery protocol to discover services provided by other peers.

1.1.2.4) web-Based computing :-

The implementation of web-based computing has given rise to new categories of devices, such as load balancers which can distribute network connections among a pool of similar servers.

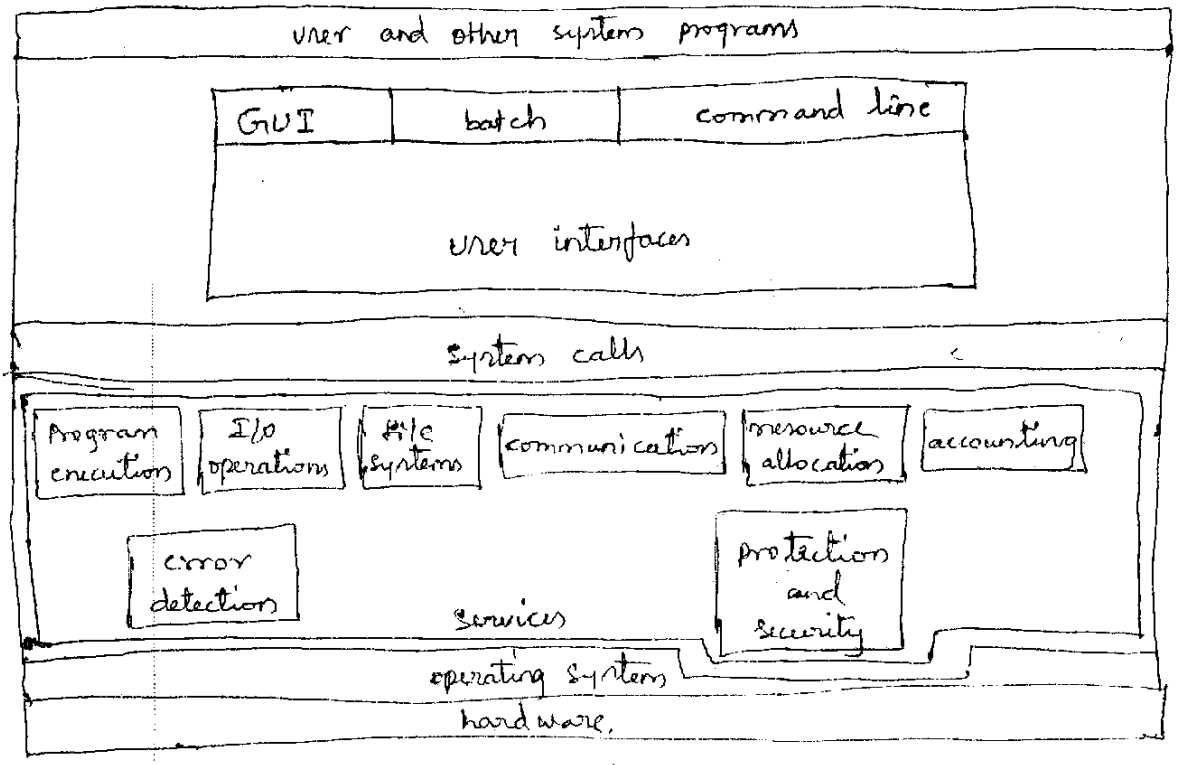
UNIT - I

Chapter - 2

System structures

2.1) Operating System Services :-

*) An operating system provides an environment for the execution of programs.



A view of operating System services.

One set of operating-system services provides functions that are helpful to the user.

1) User interface :- Almost all operating systems have a user interface (UI). There are 3 form of interface

1) DTrace command-line interface
2) batch interface
3) graphical user interface (GUI)

(1) DTrace Command-line interface :- Which uses text commands and a method for entering them.

(2) batch interface :- in which commands are entered into files and those files are executed.

(3) graphical user interface :- Here, the interface is a window system with a pointing device to direct I/O, choose from menus, and make selections and a keyboard to enter text.

2) Program execution :- The system must be able to load a program into memory & to run that program. The program must be able to end its execution, either normally or abnormally.

3) I/O operations :- A running program may require I/O, where the I/O may get from file or an I/O device. For efficiency and protection, users usually cannot control I/O devices directly.

4) File system manipulation :- programs need to read and write files and directories. They also need to create and delete them by name, search for a given file, and list file information. Finally some programs include permissions management to allow or deny access to files or directories based on file ownership.

5) Communications :- There are many chances in which one process needs to exchange information with another process. Such communication may occur between processes that are executing on the same computer or between processes that are executing on different computer system connected by a computer network.

6) Error detection :- The operating system needs to be constantly aware of possible errors.

*) Errors may occur in the CPU and memory hardware in I/O devices.

*) For each type of error, the operating system should take the appropriate action.

Another set of operating-system functions exists not for helping the user but rather for ensuring the efficient operation of the system itself.

1) Resource allocation :- When there are multiple users or multiple jobs running at the same time, resources must be allocated to each of them. many different types of resources are managed by the operating system. (CPU cycles, main memory, file storage)

2) Accounting :- we want to keep tracks of which users use how much and what kinds of computer resources. This usage statistics may be a valuable tool for researchers wish to reconfigure the system to improve computing services

3) Protection and Security :- The owners of information stored in a multiuser or networked computer system may want to control use of that information.

*) When several separate processes execute concurrently, it should not be possible for one process to interfere with others or with the operating system itself.

*) Protection involves ensuring that all access to system resources is controlled.

*) If a system is to be protected and secure, precautions must be instituted throughout it. A chain is only as strong as its weakest link.

2.2) User Operating - System Interface :-

There are several ways for users to interface with the operating system. The 2 fundamental approaches are

1) provides a command-line interface, (or) command interpreter.

2) Graphical user interface (GUI).

2.2.1) Command Interpreter :-

*) Some operating systems include the command interpreter in the kernel.

*) Others such as Window XP and UNIX, treat the command interpreter as a special program.

*) On systems with multiple command interpreters, are known as shells.

*) For example, on UNIX and Linux systems, there are several different shells a user may choose from including the Bourne shell, C shell, Bourne-Again shell, the Korn shell, etc.

*) The main function of the command interpreter is to get and execute the next user-specified command.

*) Many of the commands given at this level manipulate files: create, delete, list, print, copy, execute and so on.

*) EXL rm file.txt (file.txt → parameter), would search for a file called rm, load the file into memory, and execute it with the parameter file.txt

2.2.2) Graphical user Interfaces :-

*) A second strategy for interfacing with the operating system is through a user friendly graphical user interface, or GUI.

*) Here, rather than entering commands directly via a command-line interface, users employ a mouse-based window-and-menu system characterized by a desktop metaphor.

*) The user moves the mouse to position its pointer on images, or icons on the screen (the desktop) that represent programs, files, directories and system functions. Depending on mouse pointer's location, clicking a button on the mouse can invoke a program, select a file or directory - known as a folder.

*) GUI's first appeared due in part to research taking place in the early 1970s at Xerox PARC research facility.

However graphical interfaces became more widespread with the advent of Apple macintosh computers in the 1980s.

*) Mac OS X adopted the Aqua interface.

*) Microsoft's first version of windows - Version 1.0 - was based on the addition of a GUI interface to the MS-DOS operating system.

2.3) System Calls :-

*) System calls provide an interface to the services made available by an operating system.

*) These calls are generally available as routines written in C and C++.

*) For writing a simple program to read data from one file and copy them to another file. The first input that the program will need is the names of the two files. The input file and the output file.

*) These files names can be specified in many ways, depending on the operating-system Design.

*) Once the two files names are obtained, the program must open the input file and create the output file. Each of these operations requires another system call.

*) Now that both files are setup, we enter a loop that reads from the input file (a system call) and writes to the output file (another system call). Each read & write must return status information.

*) Frequent systems execute thousands of system calls per second. This sequence is shown in figure.

Example 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 doesn't exist, abort
create output file
if file exist abort
loop
Read from input file
write to output file
until loop fails
close output file
write completion msg to screen
Terminate normally.

Example of how system calls are used.

*) Typically, application developers design program according to an application programming interface (API).

*) The API specifies a set of functions that are available to an application programmer, including the parameters that are passed to each function and the return values the programmers can expect.

*) Three of the most common APIs available to application programmers are.

1) The Win32 API for windows systems
2) The POSIX API for All versions of UNIX, Linux, Mac OS X
3) Java API for designing programs that run on the Java virtual machine.

each operating system has its own name for each system call.

*) The functions that make up an API typically invoke the actual system calls on behalf of the application programmer.

Ex:- The Win32 function CreateProcess() (API) actually calls the NTCreateProcess() system call in the windows Kernel.

Advantages of using API

1) Program portability.
2) The same program can be run on different machines which supports same API.

Example of standard API

1) ReadFile() function in the Win32 API - a function for reading from a file.

return value.

~~BOOL~~ ~~ReadFile~~

```
BOOL ReadFile (  HANDLE     file,
                 LPVOID     buffer,
                 DWORD      bytesToRead,
                 LPDWORD    bytesRead,
                 LPOVERLAPPED ovl);
```

ReadFile ← function name

(...) ← parameters

*) HANDLE file – the file to be read

*) LPVOID buffer – a buffer where the data will be read into and written from

*) DWORD bytesToRead – the number of bytes to be read into the buffer.

*) LPDWORD bytesRead – the number of bytes read during the last Read.

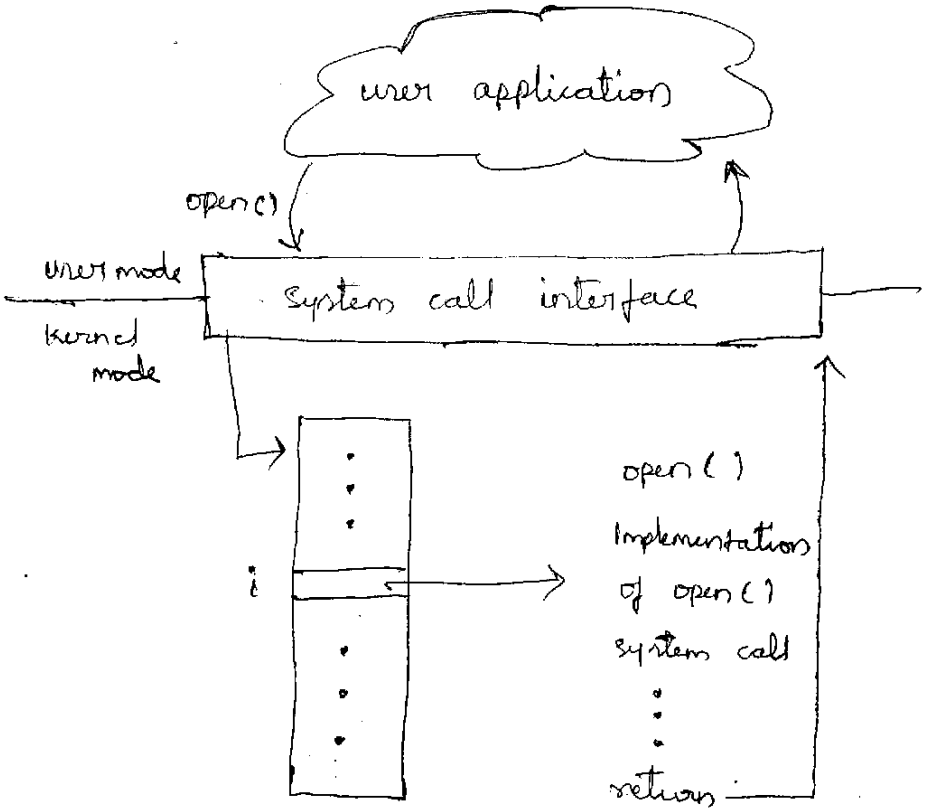
*) LPOVERLAPPED – indicates if overlapped I/O is being used.

* The following figure shows how the operating system handles a user application invoking the open() system call.

Pass parameters to OS in 3 ways

i) Registers if not enough ii) Block/table address is passed as param in register

iii) Stack.

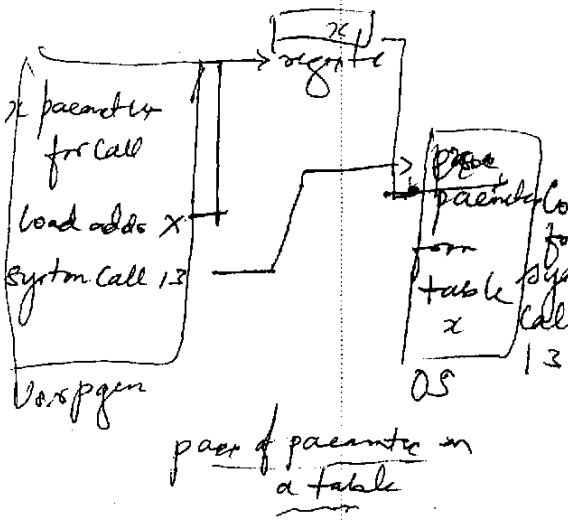


The handling of a user application invoking the open() system call.

2.4) Types of System calls :-

System calls can be grouped roughly into six major categories :

1) process control
2) file manipulation
3) device manipulation
4) information maintenance
5) communications &
6) Protection.



Types of System Calls :-

2.4.1) Process control :-

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

*) If a system call is made to terminate the currently running program abnormally, or if the program runs into a problem and causes an error trap, a dump of memory is sometimes taken and an error message generated.

*) The dump is written to disk and may be examined by a debugger. [find/correct bugs]

*) System calls under process control

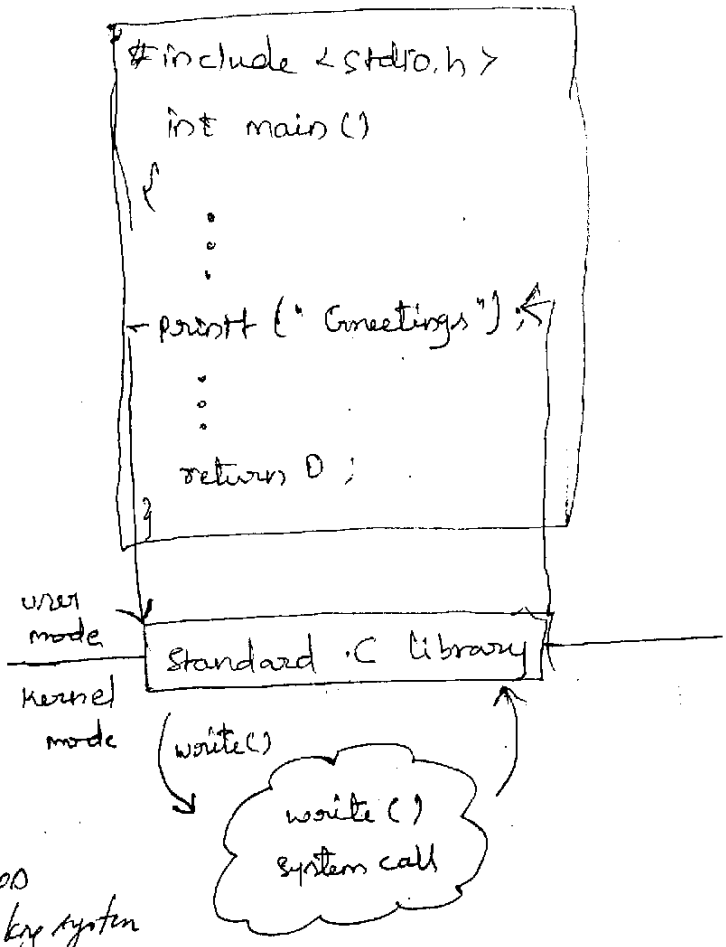
1) end, abort
2) load, execute
3) create process, terminate process
4) get process attributes, set process attributes
5) wait for time
6) wait event, signal event
7) allocate and free memory

*) Examples of windows and UNIX system calls for process control.

	Windows	UNIX
Process control	CreateProcess()	fork()
	ExitProcess()	exit()
	WaitForSingleObject()	wait()

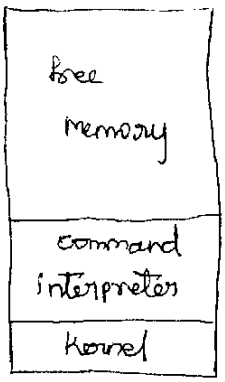
Example of STANDARD C Library

An an example, let's assume a C program invokes the Printf() statement.

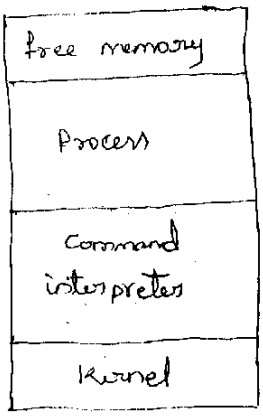


MS-DOS
Single tasking system

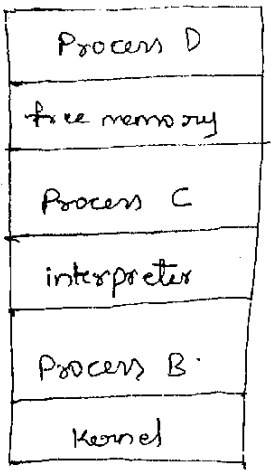
Standard C library handling of write().



MS-DOS execution (a) At system startup



(b) Running a program.



FreeBSD running multiple programs.

2.4.2) File Management :-

*) Here, we can identify several common system calls dealing with files.

*) we first need to able to create and delete files

*) Once the file is created, we can open it, we may also read, write, or reposition (rewinding or skipping to the end of the file)

*) Finally, we need to close the file, indicating that we are no longer using it.

*) File attributes include the file name, a file type, Protection codes, accounting information, and so on. atleast 2 system calls, get file attribute and set file attribute, are required for this function.

*) Some operating systems provide many more calls, such as calls for file move and copy.

Ex:- CreateFile(), ~~Boot~~ open() | ReadFile(), read() | writeFile(), write()

2.4.3) Device Management :-

*) A Process may need several resources to execute -

1) main memory
2) disk drives
3) access to file

request device, release device
read, write, reposition
get device attributes, set
logically attach, detach devices

*) If the resources are available, they can be granted, and control can be returned to the user process. Other wise, the process will have to wait until sufficient resources are available.

*) The various resources controlled by the operating system are of 2 types

1) physical devices (HDD, tapes)
2) virtual devices (files)

*) If there are multiple users of the system, the system the system may require us to first request the device, to ensure exclusive use of it. After we are finished with the device, we release it. These functions are similar to the open and close system calls for files.

Windows	Unix
ReadConsole()	read()
WriteConsole()	write()

2.4.4) Information Maintenance :-

*) Many system calls exist simply for the purpose

of transferring information between the user program and the operating system.

*) Ex: Most systems have a system call to return the current time and date. get time/date, set

*) Another set of system calls is helpful in debugging a program. Many systems provide system calls to dump memory. This provision is useful for debugging. get process, file/device attributes, set

*) A program trace lists each system call as it is executed.

2.4.5) Communication :-

There are two common models of interprocess communication

1) The message passing model. create, delete comm'n connection
2) shared memory model. send, receive msg
transfer status info
attach/detach remote memory

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

*) Messages can be exchanged between the processes either directly or indirectly through a common mailbox.

*) Before communication can take place, a connection must be opened. The name of the other communicator must be known, be it another process on the same system or a process on another computer connected by a communications network.

*) Each computer in a network has a host name by which it is commonly known.

*) A host also has a network identifier, such as an IP address. Similarly, each Process has a process name, and this name is translated into an identifier by which the operating system can refer to the process. The get hostid and get procesid system calls do this translation

*) The identifiers are then passed to the general purpose open and close calls provided by the file system or to specific open connection and close connection system calls, depending on the system's model of communication.

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

*) Normally, the operating system tries to prevent one Process from accessing another process's memory. Shared Memory requires that two or more processes agree to remove this restriction. They can then exchange information by reading and writing data in the shared areas.

*) The Form of the data and the location are determined by the processes and are not under the operating system's control. The processes are also responsible for ensuring that they are not writing to the same location simultaneously.

Both of the models that is

1) menage paring model &
2) shared memory model

are common in operating systems, and most systems implement both.

*) Menage passing is useful for exchanging smaller amounts of data, because no conflicts need be avoided.

*) Menage paring is also easier to implement than shared memory for inter computer communication.

*) Shared memory allows maximum speed and convenience of communication, since it can be done at memory speeds when it takes place within a computer.

2.4.6) protection :-

*) protection provides a mechanism for controlling accen to the resources provided by a computer system.

*) Historically, protection was a concern only on multiprogrammed computer systems with several users. However, with the advents of networking and the internet, all computer systems, from servers to PDAs, must be concerned with protection.

*) Typically, system calls providing protection include set permission and get permission, which manipulate the permission settings of resources such as files and disks.

*) The allow user and deny user system calls specify whether particular users can - or cannot - be allowed access to certain resources.

2.5) System Programs :-

*) Another aspect of a modern system is the collection of system programs.

*) According to logical computer hierarchy. At the lowest level is hardware. Next is the operating system, then the system programs, and finally the application programs.

*) System programs, also known as system utilities, provide a convenient environment for program development and execution.

System programs can be divided into these (six) categories :-

1) File management :- These programs create, delete, copy, rename, print, dump, list, and generally manipulate files and directories.

2) Status information :- Some programs simply ask the system for the date, time, amount of available memory or disk space, number of users status information. Typically, these programs format and print the output to the terminal or other output devices or files or display it in a window of the GUI.

3) File modification :- Several text editors may be available to create and modify the contents of files stored on disk or other storage devices.

4) programming - language support :- Compilers, assemblers, debuggers, and interpreters for common programming languages (such as C, C++, Java, VB) are often provided to the user with the operating system.

5) program loading and execution :- Once a program is assembled or compiled, it must be loaded into memory to be executed. The system may provide absolute loaders, ~~reco~~ relocatable loaders, linkage editors.

6) Communications :- These programs provide the mechanism for creating virtual connections among processes, users, and computer systems. They allow users to send messages to one another's screens, to transfer files from one machine to another.

*) In addition to system programs, most operating systems are supplied with programs that are useful in solving common problems or performing common operations. Such application programs include web browsers, word processors & text formatters, spreadsheets, database systems, compilers & games.

*) When a user's computer running mac OS X operating system, the user might see GUI, and alternatively one of the users may have command-line user shells. Both use the same set of system calls, but the system calls look different & act in different ways.

*) Some times the same user can use multiple operating systems on the same hardware sequentially or concurrently

2.6) Operating-System Design and Implementation :-

Here we discuss about the problems we face in designing and implementing an operating system.

2.6.1) Design goals :-

*) The first problem is designing a system is to define goals and specifications.

*) At the highest level, the design of the system will be affected by the choice of hardware and the type of system: batch, time shared, single user, multi user, distributed, real time or general purpose.

*) The requirements can be divided into 2 basic groups:

1) user goals
2) system goals.

*) Users can derive certain obvious properties in a system. The system should be convenient to use, easy to learn and to use, reliable, safe and fast.

*) A similar set of requirements can be defined by those people who must design, create, maintain and operate the system. The system should be easy to design, implement and maintain; and it should be flexible, reliable, error free and efficient.

2.6.2) Mechanisms and Policies :-

*) Mechanisms determine how to do something

*) Policies determine what will be done.

Ex:- The timer construct is a mechanism for ensuring CPU protection, but ~~deciding~~ deciding how long the timer is to be set for a particular user/program is a policy decision.

*) policies are likely to change across places or over time.

*) A general mechanism insensitive to changes in policy would be more desirable.

*) In the worst case, each change in policy would require a change in the underlying mechanism.

*) Micro-kernel based operating systems implement a basic set of primitive building blocks. These blocks are almost policy free, allowing more advanced mechanisms and policies to be added via user-created kernel modules or via user programs themselves.

*) policy decisions are important for all resource allocation.

  *) policy decision has to be made whether to allocate or not to allocate resource.

  2) how and what resources must be decided by mechanism.

2.6.3) Implementation :-

*) Once an operating system is designed, it must be implemented. Traditionally operating systems have been written in assembly language.

*) Now, however, they are most commonly written in higher-level languages such as C or C++.

*) The linux and window XP operating systems are written mostly in C, although there are some small sections of assembly code for device drivers and for saving and restoring the state of registers.

*) The advantages of using a higher-level language are

  1) The code can be written faster

2) The code is more compact

3) Easier to understand.

4) Easier to debug

*) Operating system is far easier to port. - to move to some other hardware.

Eg: MS-DOS was written in Intel 8088 assembly language

The Linux OS, in contrast, is written mostly in C and is available on a number of different CPUs, including Intel 80x86, Motorola 680X0, MIPS RX000. etc

*) The only possible disadvantage of implementing an operating system in a higher-level language are reduced speed and increased storage requirements.

*) In addition, although operating systems are large, only a small amount of the code is critical to high performance; the memory manager and the CPU scheduler are probably the most critical routines.

*) After the system is written and is working correctly, bottleneck routines can be identified and can be replaced with assembly-language equivalents.

*) To identify bottlenecks, we must be able to monitor system performance.

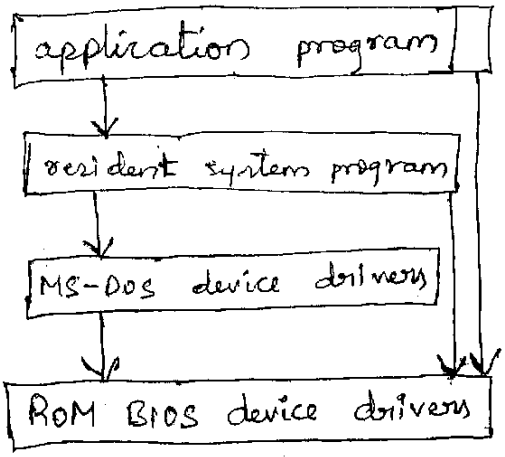
*) All interesting events are logged with their time and important parameters are written to a file. (log file).

*) later, an analysis program can process the log file to determine system performance and to identify bottlenecks and inefficiencies.

2.7) Operating system structure :-

*) A common approach is to partition the task into small components rather than have one monolithic system.

*) Each of these modules should be a well-defined portion of the system, with carefully defined inputs, outputs, and functions.



MS-DOS layer structure.

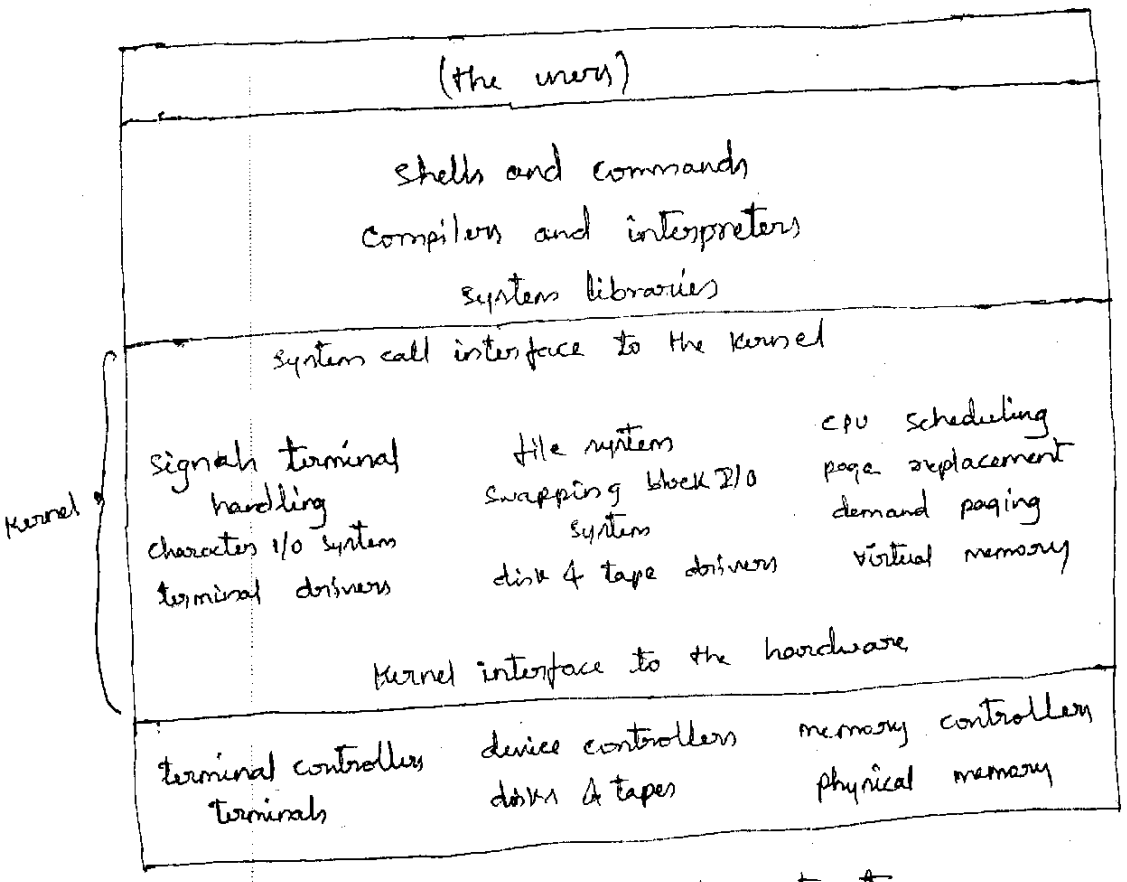
*) In MS-DOS, the interfaces and levels of functionality are not well separated.

Disadvantages:

*) For instance, application programs are able to access the basic I/O routines to write directly to the display and disk drives. Such freedom leaves MS-DOS vulnerable to virus programs, causing entire system crash.

*) Another example of limited structuring is the original UNIX operating system.

*) Like MS-DOS, UNIX initially was limited by hardware functionality. It consists of two separable parts: the kernel and the system programs.



Traditional UNIX system structure.

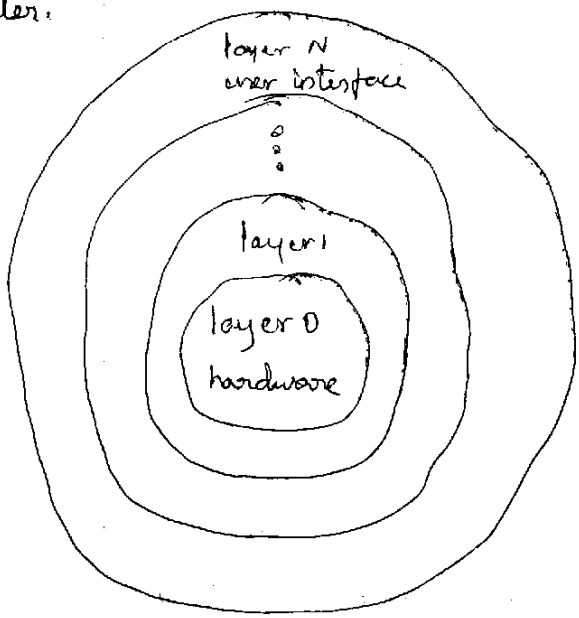
*) Everything below the systems-call interface and above the physical hardware is the kernel.

*) The kernel provides the file system, CPU scheduling, memory management, and other operating-system functions through system calls.

2.7.2) Layered Approach :-

*) With proper hardware support, operating systems can be broken into pieces that are smaller and more appropriate than those allowed by the original MS-DOS and UNIX systems.

*) Operating system can then retain much greater control over the computer and over the applications that make use of that computer.



A layered operating system.

*) A system can be made modular in many ways. One method is layered approach, in which the operating system is broken into a number of layers (levels)

*) The bottom layer (layer 0) is the hardware; the highest (layer N) is the user interface.

*) A typical operating-system layer - say, layer M consists of data structures and a set of routines that can be invoked by higher-level layers. layer M in turn, can invoke operations on lower-level layers.

*) The main advantage of layered approach is simplicity of construction and debugging.

*) The layers are selected so that each uses functions and services of only lower-level layers.

*) The first layer can be debugged without any concern for the rest of the system, because, by definition, it uses only the basic hardware to implement its functions. Once the first layer is debugged, its correct functioning can be assumed while the second layer is debugged, and so on.

*) If an error is found during the debugging of a particular layer, the error must be on that layer, because the layers below it are already debugged.

*) Each layer hides the existence of certain data structures, operations, and Hardware from higher-level layers.

*) The major difficulty with the layered approach involves appropriately defining the various layers. Because a layer can use only lower-level layers, careful planning is necessary.

*) Backing-store (disk space used by virtual-memory algorithms) drivers would normally be above the CPU scheduler, because the drivers may need to wait for I/O and the CPU can be rescheduled during this time.

*) A final problem with layered implementations is that they tend to be less efficient than other types. for instance, when user program executes an I/O operation, it executes a system call that is trapped to the I/O layer, which calls the memory management layer, which is turn calls the CPU-scheduling layer, when is then passed to the hardware. At each layer, the parameters may be modified, data may need to be passed, and so on. It causes overhead to the system call.

2.7.3) Micro Kernels :-

*) We have already seen that as UNIX expanded, the kernel became large and difficult to manage.

*) In the mid-1980's, researchers at carnegie mellon university developed an operating system called Mach that Modularized the kernel using the microkernel approach.

*) This method structures the operating system by removing all non-essential components from the kernel and implementing them as system and user-level programs.

*) Microkernels provide minimal process and memory management, in addition to a communication facility.

*) The main function of the microkernel is to provide a communication facility between the client program and the various services that are also running in user space.

*) One Advantage of the microkernel approach is ease of extending the operating system. All new services are added to user space and consequently do not require modification of the kernel.

*) The microkernel operating system is easier to port from one hardware design to another.

*) The microkernel also provides more security and reliability, since most services are running as user- rather than kernel process. If a service fails, the rest of the operating system remains untouched.

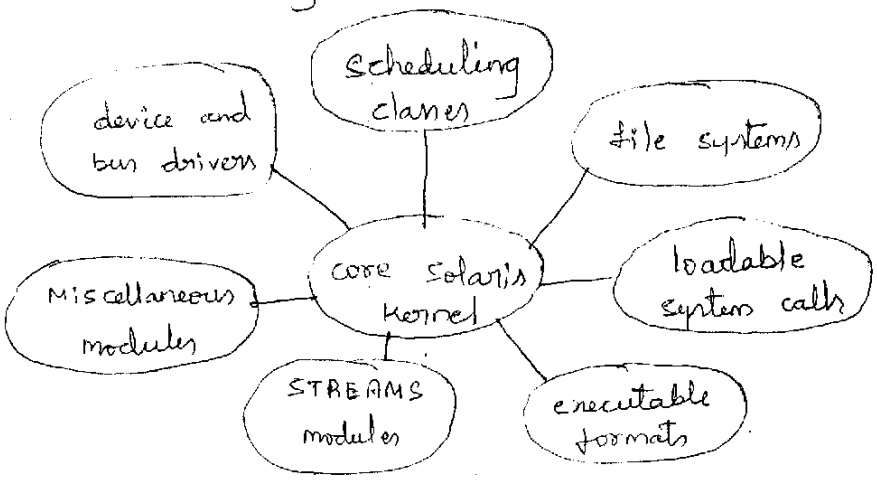
*) Unfortunately, microkernels can suffer from performance decreases due to increased system function overhead.

*) By the time Windows XP was designed, its architecture was more monolithic than microkernel.

2.7.4) modules :-

*) perhaps the best current methodology for operating-system design involves using object-oriented programming techniques to create a modular kernel.

*) Here, the kernel has a set of core components and dynamically links in additional services either during boot time or during run time.



Solaris loadable modules.

There are seven types of loadable kernel modules:

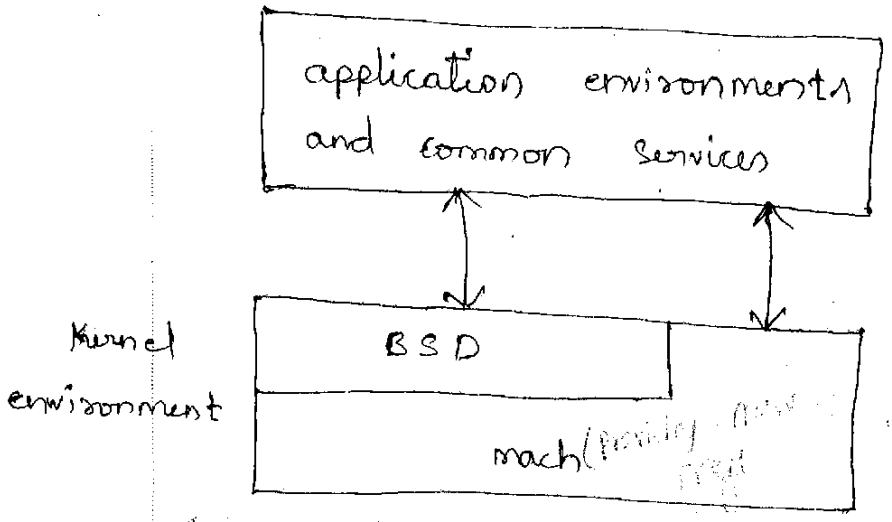
*) Such design allows the kernel to provide core services yet also allows certain features to be implemented dynamically.

*) For example, device and bus drivers for specific hardware can be added to the kernel, and support for different file systems can be added as loadable modules.

*) The loadable modules is more flexible than a layered system is that any module can call any other module.

*) Here the primary module has only core functions and knowledge of how to load and communicate with other modules.

*) loadable modules are more efficient, because modules do not need to invoke menage passing in order to communicate.



*) The apple mac os x operating system uses a hybrid structure (It is also called as Darwin)

*) It is a layered system in which one layer consists of the mach microkernel.

*) The top layers include applications environments and a set of services providing a graphical interface to applications.

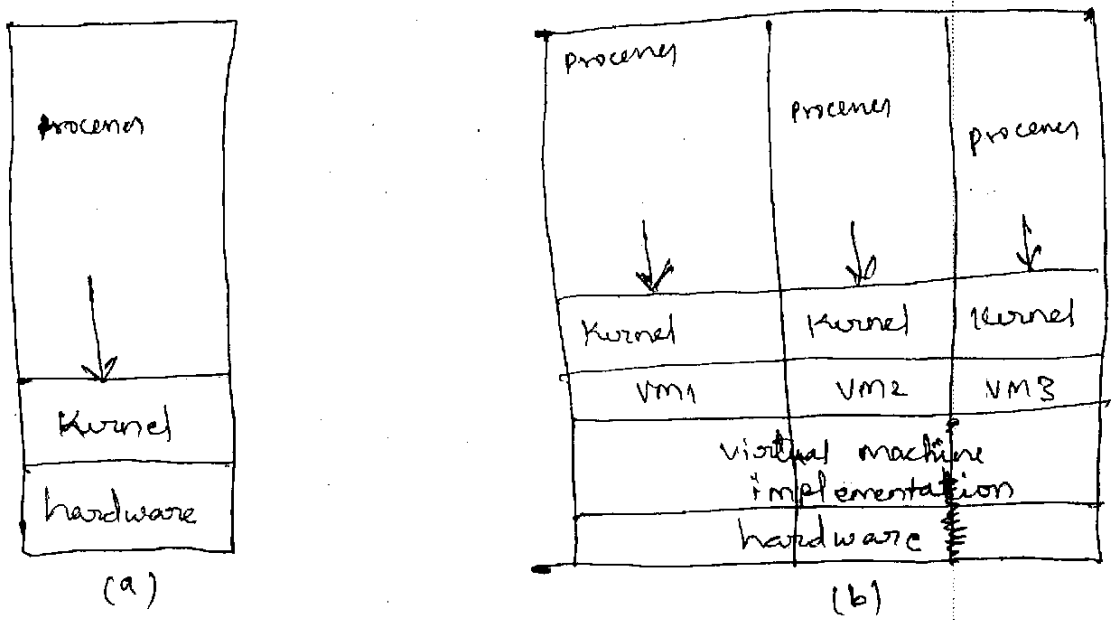
*) Below these layers there is Mach microkernel and the BSD kernel. Mach provides memory management; support for remote procedure calls (RPCs) and

and interprocess communication (IPC) facilities, including message passing and thread scheduling.

2.8) Virtual Machines :-

*) The fundamental idea behind a virtual machine is to abstract the hardware of a single computer (the CPU, memory, disk drives, network interface cards) into several different execution environments, thereby creating the illusion that each separate execution environment is running its own private computer.

*) By using CPU scheduling and virtual memory techniques, an operating system host can create the illusion that a process has its own processor with its own memory.



System models (a) Nonvirtual machine (b) virtual machine

2.8.1) History :- X

*) Virtual machine first appeared commercially on IBM mainframes via the VM operating system in 1972.

*) IBM VM370 divided a mainframe into multiple virtual machines, each running its own operating system.

*) A major difficulty with the IBM VM370 is from disk systems. Suppose that physical machine had three disk drives but wanted to support seven virtual machines.

*) The solution was to provide virtual disks termed minidisks in IBM's mini operating system.

*) The system implemented each minidisk by allocating as many tracks on the physical disks as the minidisk needed. → Implementation

2.8.2) Benefits :-

There are several reasons for creating a virtual machine.

*) One important advantage is that the host system is protected from the virtual machines, just as the virtual machines are protected from each other.

*) A virus inside a guest operating system might damage that operating system but is unlikely to affect the host or the other guests.

*) Every virtual machine is completely isolated from all other virtual machines, there are no protection problems.

*) There are no direct sharing of resources. Two approaches to provide sharing have been implemented.

1) It is possible to share a file-system volume (minidisk) and thus to share files.

2) It is possible to define a network of virtual machines, each of which can send information over virtual communications network.

*) The power of the operating system makes changing it particularly dangerous. Because the operating system executes in kernel mode, a wrong change in a pointer could cause an error that would destroy the entire file system. Thus it is necessary to test all changes to the operating system carefully.

*) The operating system, runs & controls the entire machine. Therefore, the current system must be stopped and taken out of use while changes are made and tested. This period is commonly called system development time.

*) Another advantage of virtual machines for developers is that multiple operating systems can be running on the developer's workstation concurrently. This virtualized workstation allows for rapid porting and testing of programs in varying environments.

*) A major ~~add~~ advantage of virtual machines in production data-centers are is system consolidation, which involves taking two or more separate systems & running them in virtual machines on one system.

Ex: Vmware.

2.8.3) Simulation + X

*) Virtualization is the most common because it makes guest operating systems and applications "believe" they are running on native hardware.

*) Another methodology is simulation, in which the host system has one system architecture and the guest system was compiled for a different architecture.

Ex: Suppose a company has replaced its outdated computer system with a new system but would like to continue to run certain important programs that were compiled for the old system.

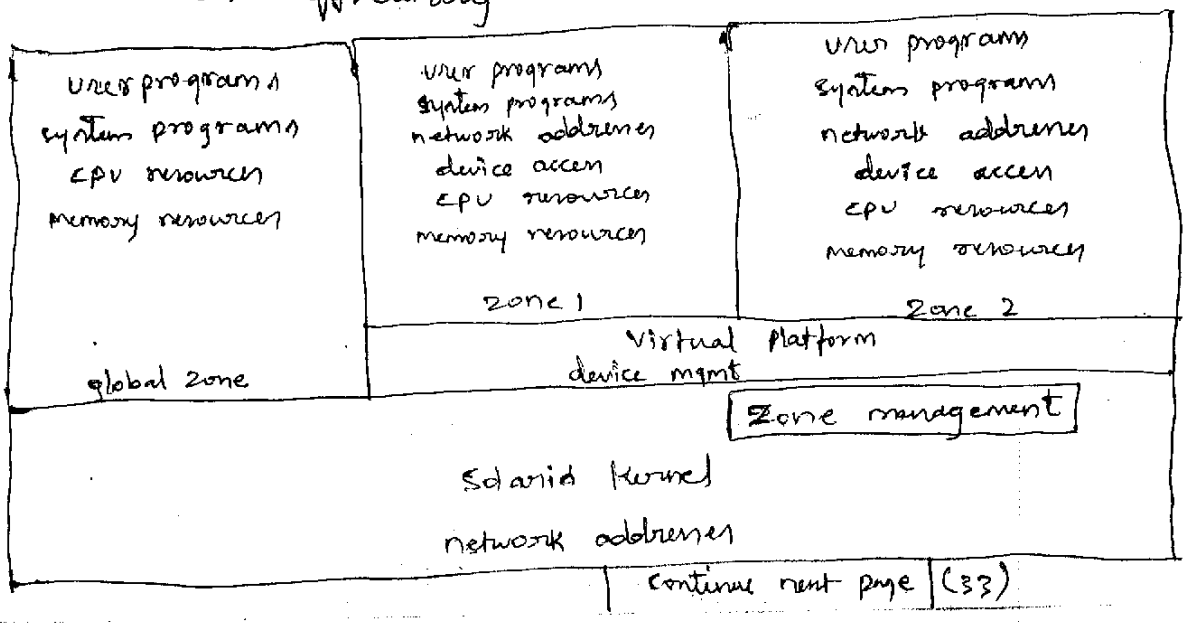
*) So the old programs could be run in an emulator. that translates each of the outdated system's instructions into the native instruction set of the new system.

*) Emulation can increase the life of programs and allow us to explore old architectures without having an actual old machine, but its major challenge is performance.

2.8.4) Para – Virtualization :-

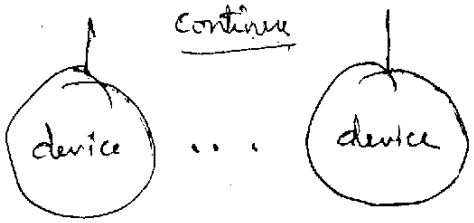
*) Rather than creating illusion that a guest operating system has its own computer system, para-virtualization presents the guest with a system that is similar but not identical to the guest's preferred system.

*) para-virtualization is designed to make use of resources efficiently



continue next page (33)

Continue



Solaris 10 with two Containers

*) Solaris 10 includes containers, or zones, that create a virtual layer between the operating system and the application.

*) In this system, only one Kernel is installed, and the hardware is not virtualized.

*) Rather, the operating system is virtualized.

*) One or more zones can be created, and each can have its own applications, user accounts, and so on.

*) Solaris 10 with 2 containers and global user space.

2.8.5) Implementation :-

*) Although the virtual-machine concept is useful, it is difficult to implement.

*) Much work is required to provide an exact duplicate of the underlying machine. This machine has 2 modes

1) user mode.
2) Kernel mode.

*) The virtual machine software can run in Kernel mode, since it is the operating system. VM executes User mode Guest and Virtual OS & kernel on physical user mode

*) When a system call is made by a program running on a virtual machine in virtual user mode, it will cause a transfer to the virtual machine monitor in the real machine.

*) When the virtual-machine monitor gains control, it can change the register contents and program counter for the virtual machine to simulate the effect of the system call.

*) The major difference, of course, is time. Whereas the real I/O might have taken 100 milliseconds, the virtual I/O might take ten or more time.

*) Without some level of hardware support, virtualization would be impossible.

*) All major general purpose CPUs provide some amount of hardware support for virtualization.
[Advanced Micro Devices]
Ex: AMD virtualization technology is found in several AMD processors. It defines two new modes of operation host and guest. Benefits

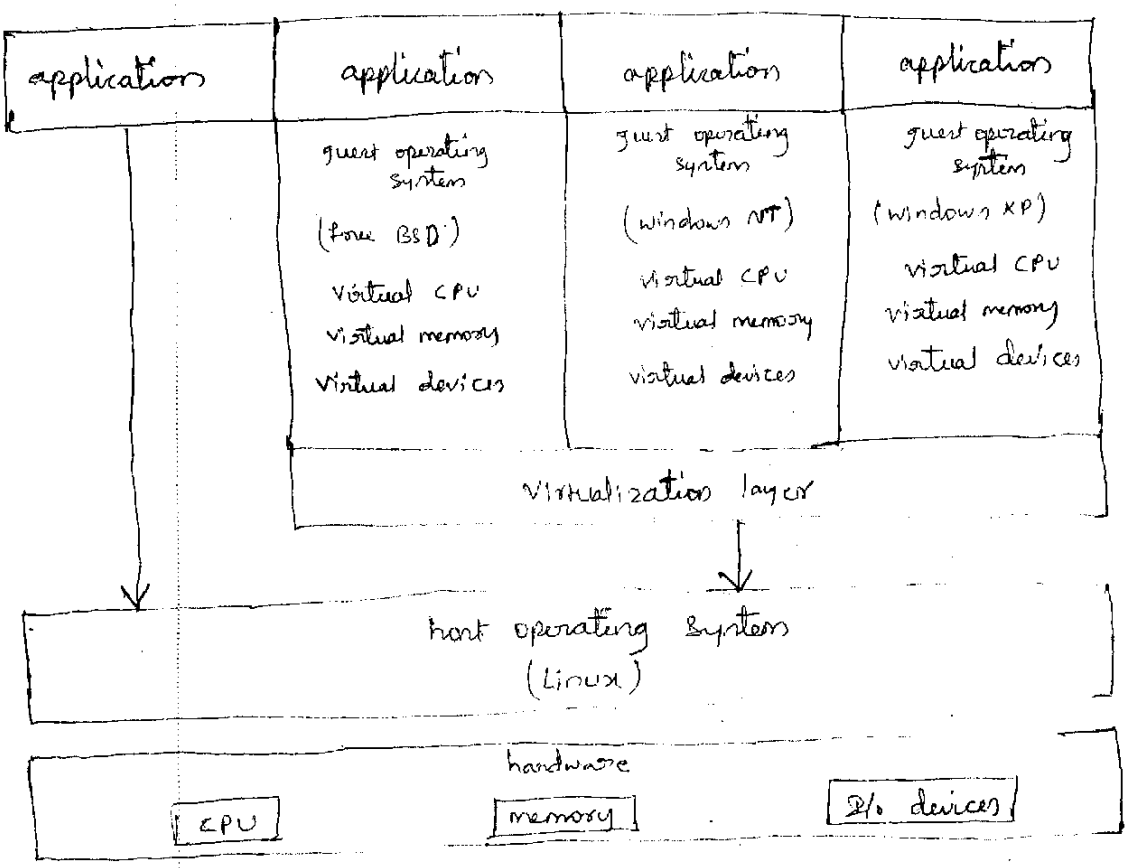
2.8.6) Examples:

2.8.6.1) VMware:

*) VMware workstation is a popular commercial application that abstracts Intel 80x86 and compatible hardware

into isolated virtual machines.

*) VMware workstation runs as an application on a host operating system such as windows or Linux and allows this host system to concurrently run several different guest operating systems as independent virtual machines.



VMware architecture

*) The above figure shows the architecture. In this scenario, Linux is running as the host operating system and free BSD, Windows NT, and windows XP are running as guest operating systems.

*) Each virtual machine has its own virtual CPU, memory,

disk drives, network interfaces and so fourth.

*) The physical disk the guest owns and manages is really just a file with in the file system of the host operating system.

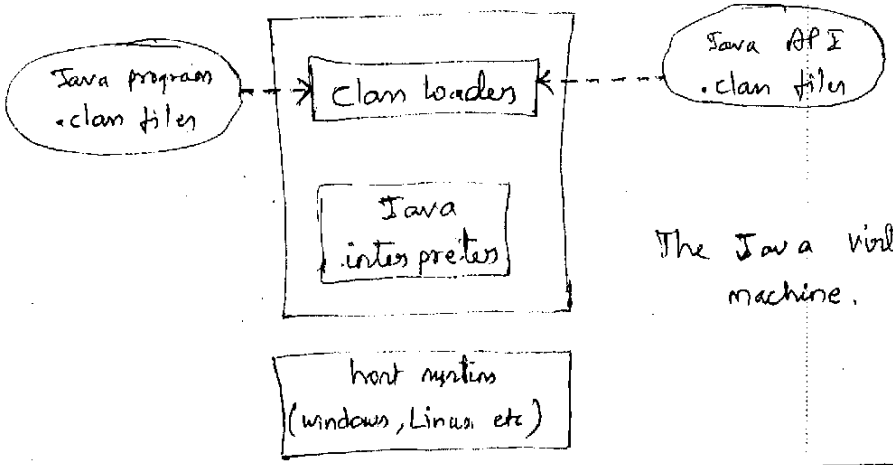
*) To create an identical guest instance, we can simply copy the file. copying the file to another location protects the guest instance against a disaster at the original site.

2.8.6.2) The Java virtual machine :-

*) Java is a popular object-oriented programming language introduced by Sun micro systems in 1995.

*) Java provides language specification, large API library and Java virtual machine. (JVM)

*) Java objects are specified with the class construct; a Java program consists of one or more classes. For each Java class, the compiler produces an architecture-neutral bytecode output (.class) file that will run on any implementation of the JVM.



The Java virtual machine.

*) The JVM is a specification for an abstract computer. It consists of a class loader and a Java interpreter that executes the architecture-neutral bytecodes. As shown in the above diagram.

*) The class loader loads the compiled .class files from both the Java program and the Java API for execution by the Java interpreter.

*) After a class is loaded, the verifier checks that the .class file is valid Java bytecode and does not overflow or underflow the stack.

*) JVM also automatically manages memory by performing garbage collection - the practice of reclaiming memory from objects no longer in use and returning it to the system.

*) The JVM may be implemented in software on top of a host operating system, such as Windows, Linux or Mac OS X, or as part of a web browser.

*) JVM may be implemented in hardware on a chip specifically designed to run Java programs.

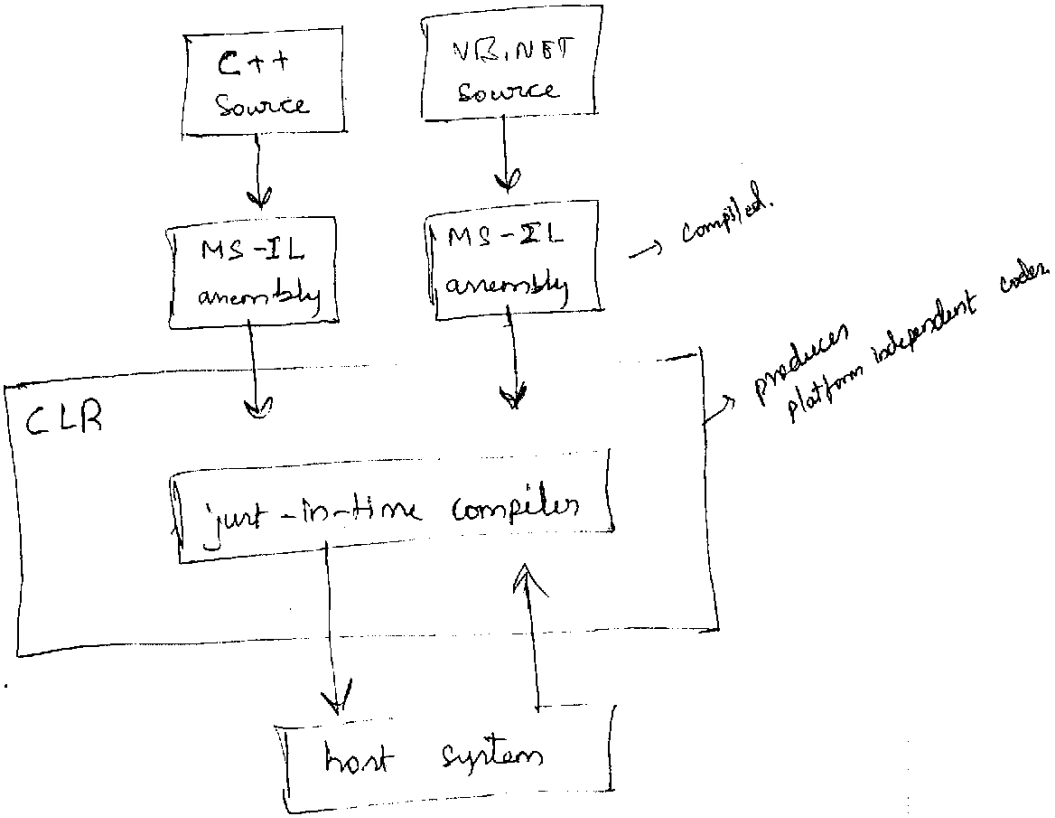
*) JVM uses a faster software technique called just-in-time (JIT) compiler).

The .NET Framework :-

*) The .NET Framework is a collection of technologies, including a set of class libraries, and an execution environment that come togeather to provide a platform for developing software.

*) .NET platform supports programming languages that are .NET Compatible.

*) .NET Framework contains CLR (common language Runtime) using which it produces the platform independent codes.



Architecture of the CLR for the .NET Framework

*) .NET framework provides an environment for execution of programs written in any of the languages targeted at the .NET Framework.

*) programs written in languages such as C# and VB.NET are compiled into an intermediate architecture-independent language called microsoft Intermediate language (MS-IL).

*) The compiled code will be having extensions of either .EXE or .DLL.

2.10) Operating-system Generation :-

*) It is possible to design, code and implement an operating system specifically for one machine.

*) More commonly, however, operating systems are designed to run on any of a class of machines with a variety of peripheral configurations.

*) The operating system is normally distributed on disk or CD-ROM. To generate a system, we use a special program called SYSGEN (system generation) that gives the specific configuration of the hardware system.

- The following kinds of information must be determined

1) what CPU is to be used? what options are installed? for multiple CPU systems, each CPU must be described.

2) How much memory is available? Some systems will determine this value themselves by referencing memory location after memory location until an "illegal address" fault is generated. This procedure defines the final legal address and hence the amount of available memory.

3) What devices are available? The system will need to know how to address each device (the device number), the device interrupt number, and any special device characteristics.

4) What operating-system options are used? These options might include how many buffers of which sizes would be used, what type of CPU-scheduling algorithm is desired, and so on.

*) Once this information is determined, it can be used in several ways. At one extreme, a system administrator can use it to modify a copy of the source code of the operating system.

2.10) System Boot

*) After an operating system is generated, it must be made available for use by the hardware.

*) But hardware doesn't know where does Kernel present to load. So, On most computer systems, bootstrap program locates the Kernel and loads it into main memory, and starts its execution.

When a CPU is powered up or rebooted the instruction register is loaded with a predefined memory location, and execution starts there. At that location is the initial bootstrap program. This program is in ROM (Read only memory).

*) Boot - strap program can perform a variety of tasks

1) One task is to run diagnostics to determine the state of the machine.

2) It can initialize all aspects of the system, from CPU registers to device controllers and the contents of main memory.

Some systems such as cellular phones, PDA's and game consols store the entire operating system in ROM. problem here is change in operating system will change ROM. TO resolve this problem some systems are using (EEPROM) Electronic Erasable programmable read-only memory.

*) A disk that has a boot partition is called a boot disk.

UNIT - 2

Chapter 1

Process Concept

Early computer systems allowed only one program to be executed at a time. This program had complete control of the system and had access to all the system's resources.

In contrast, current-day computer systems allow multiple programs to be loaded into memory and executed concurrently.

3.1) Process Concept :-

*) A batch system executes jobs, whereas a time-shared system has user programs, or tasks.

*) Even on a single-user system such as Microsoft windows, a user may be able to run several programs at one time: a word processor, a web browser and an e-mail package

3.1.1) The Process

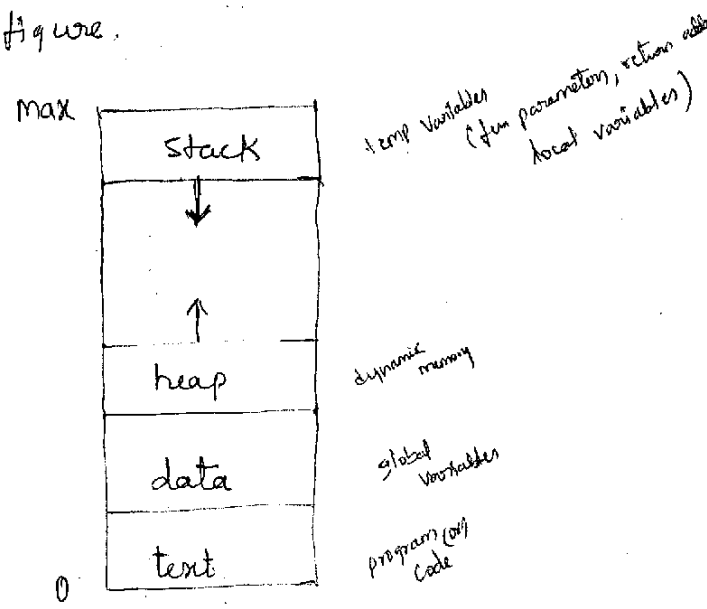
*) A Process is more than the program code, which is sometimes known as the text section.

*) It also includes the current activity, as represented by the value of the program counter and the contents of the processor's registers.

*) A process generally also includes the process stack, which contains temporary data (such as function parameters, return addresses, and local variables), and a data section, which contains global variables.

*) A process may also include a heap, which is memory that is dynamically allocated during process running.

The structure of a process in memory is shown in below figure.



Process in memory.

*) We can say that a program by itself is not a process; a program is a passive entity, such as a file containing a list of instructions stored on disk (often called an executable file), whereas a process is an active entity, with a program counter specifying the next instruction to execute and a set of associated resources.

*) A program becomes a process when an executable file is loaded into memory.

*) Two common techniques for loading executable files are double-clicking an icon representing the executable file and entering the name of the executable file on the command line (as in prog.exe or a.out).

*) Although two processes may be associated with the same program, they are considered as two separate execution sequences.

*) For instance, several users may be running different copies of the web browser program (creating tabs).

*) Each of these is a separate process; and although the text sections are equivalent, the data, heap, and stack sections vary.

3.1.2) process state :-

*) As a process executes, it changes state. The state of a process is defined in part by the current activity of that process.

*) Each process may be in one of the following states:

1) New :- The process is being created.
2) Running :- Instructions are being executed.
3) waiting :- The process is waiting for some event to occur (such as an I/O completion or reception of a signal)
4) Ready :- The process is waiting to be assigned to a processor
5) Terminated :- The process has finished execution.

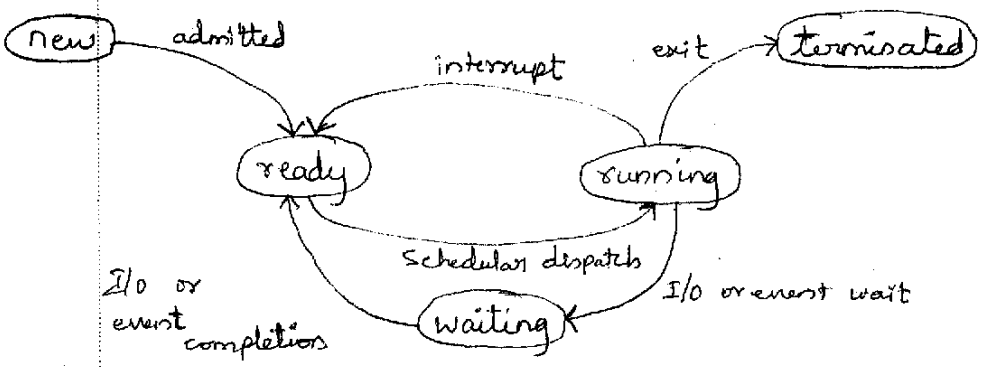


Diagram of process state

*) These names are arbitrary, and they vary across O.S.

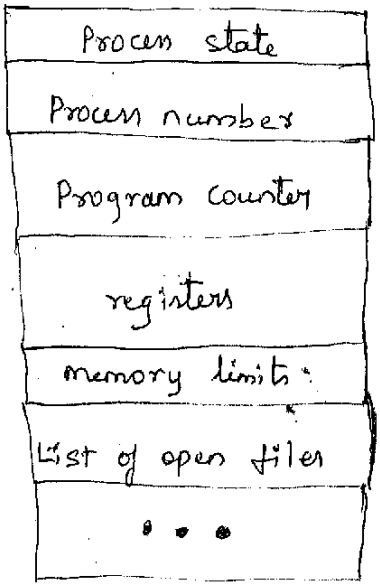
*) The states that they represent are found on all systems.

*) It is important to realize that only one process can be running on any processor at any instant. many processes may be ready and waiting.

3.1.3) Process Control Block :-

*) Each Process is represented in the operating system by a process control block (PCB), also called task control block.

*) A PCB is shown in following. It contains many pieces of information associated with a specific process



Process control block (PCB).

1) process state :- The state may be new, running, waiting, halted and so on.

2) Program Counter :- The counter indicates the address of the instruction to be executed for this Process

3) CPU registers :- The registers vary in number and type, depending on the computer architecture. They include

accumulators, index registers, stack pointers, and general purpose registers. Along with the program counter, the state information must be saved when an interrupt occurs. As shown in following figure.

4) CPU-scheduling information :- This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.

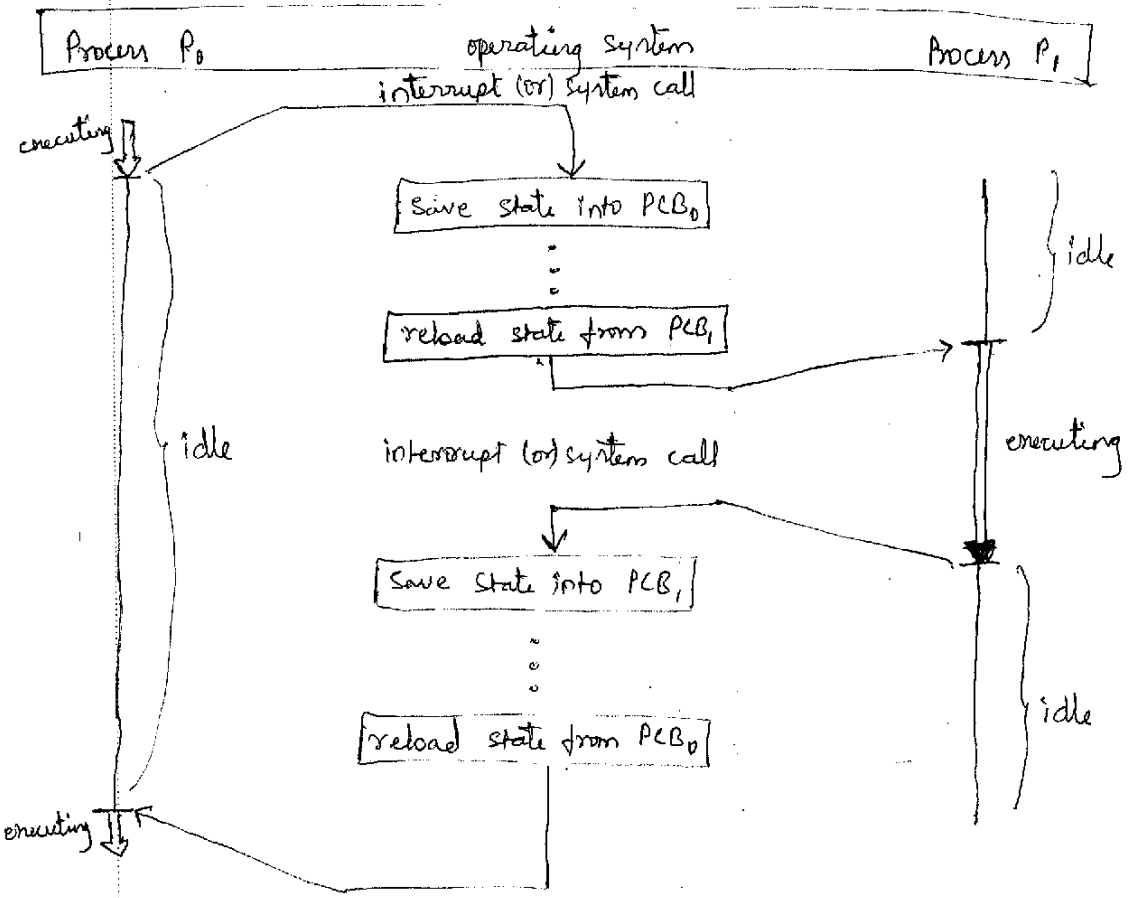


Diagram showing CPU switch from process to process.

5) Memory management information :- This information may include ~~each~~ values of base and limit registers, the page tables, or the segment tables, depending on the memory system used by the operating system.

6) Accounting Information :- This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.

7) I/O status information :- This information includes the list of I/O devices allocated to the Process, a list of open files, and so on.

3.1.4) Threads :-

*) A process is a program that performs a single thread of execution.

*) For example, when a process is running a word-processor program, a single thread of instructions is being executed. This single thread of control allows the process to perform only one task at one time.

*) Many modern operating systems have extended the process concept to allow a process to have multiple threads of execution and thus to perform more than one task at a time.

*) A Thread is a smallest execution part of a program that can be managed by a ~~process~~ scheduler.

3.2) Process Scheduling :-

*) The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization.

*) The objective of time sharing is to switch the CPU among processes so frequently that users can interact with each program while it is running.

*) To meet these objectives, the process scheduler selects an available process (possibly from a set of several available processes) for program execution on the CPU.

*) For a single-processor system, there will never be more than one running process. If there are more processes, the rest will have to wait until the CPU is free and can be rescheduled.

3.2.1) Scheduling Queues :-

*) As processes enter the system, they are put into a job queue, which consists of all processes in the system.

*) The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the ready queue. This queue is generally stored as a linked list.

*) A ready-queue header contains pointers to the first and final PCBs in the list. Each PCB includes a pointer field that points to the next PCB in the ready queue.

*) A list of processes waiting for a particular I/O device is called a device queue. Each device has its own device queue.

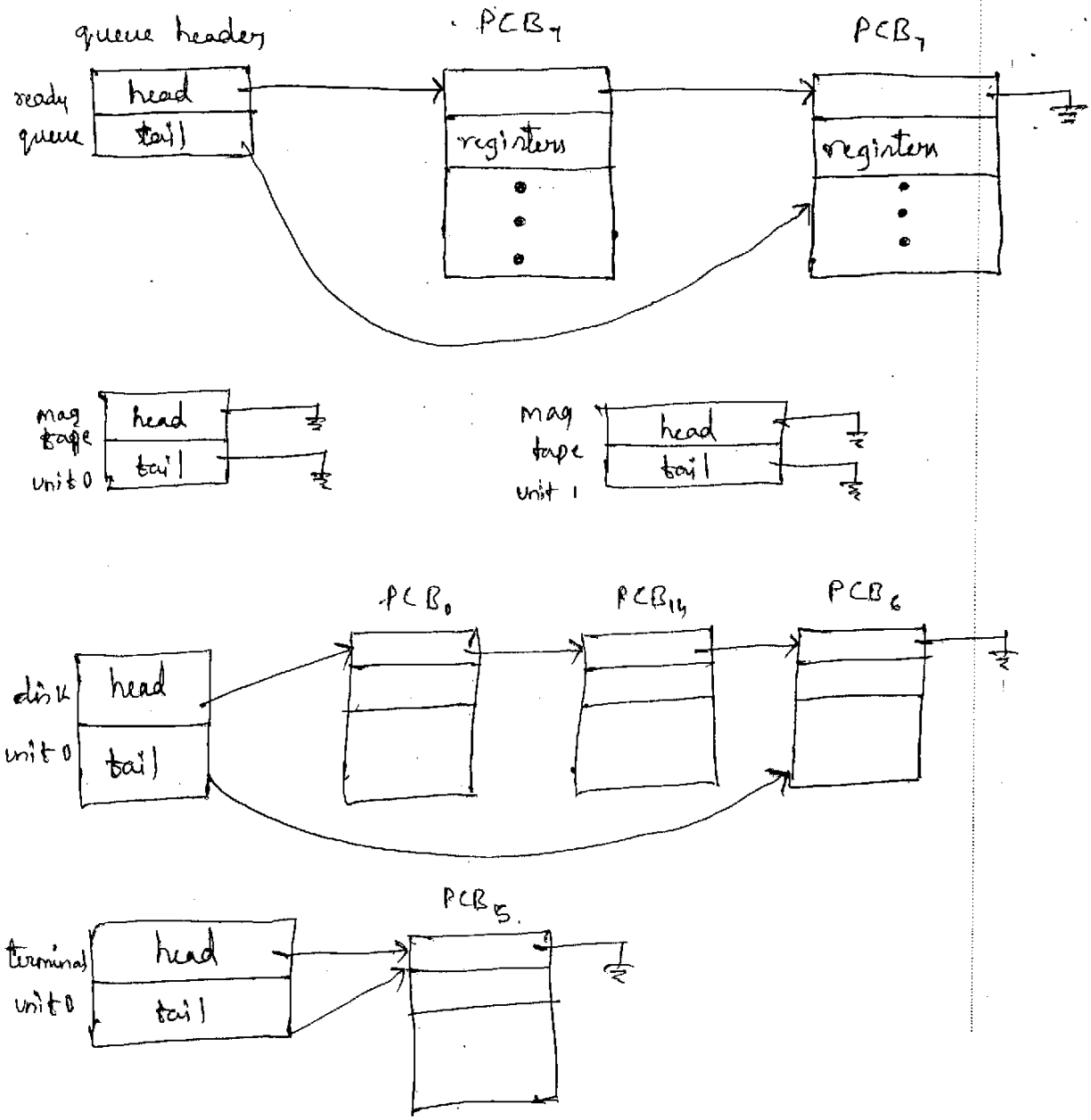
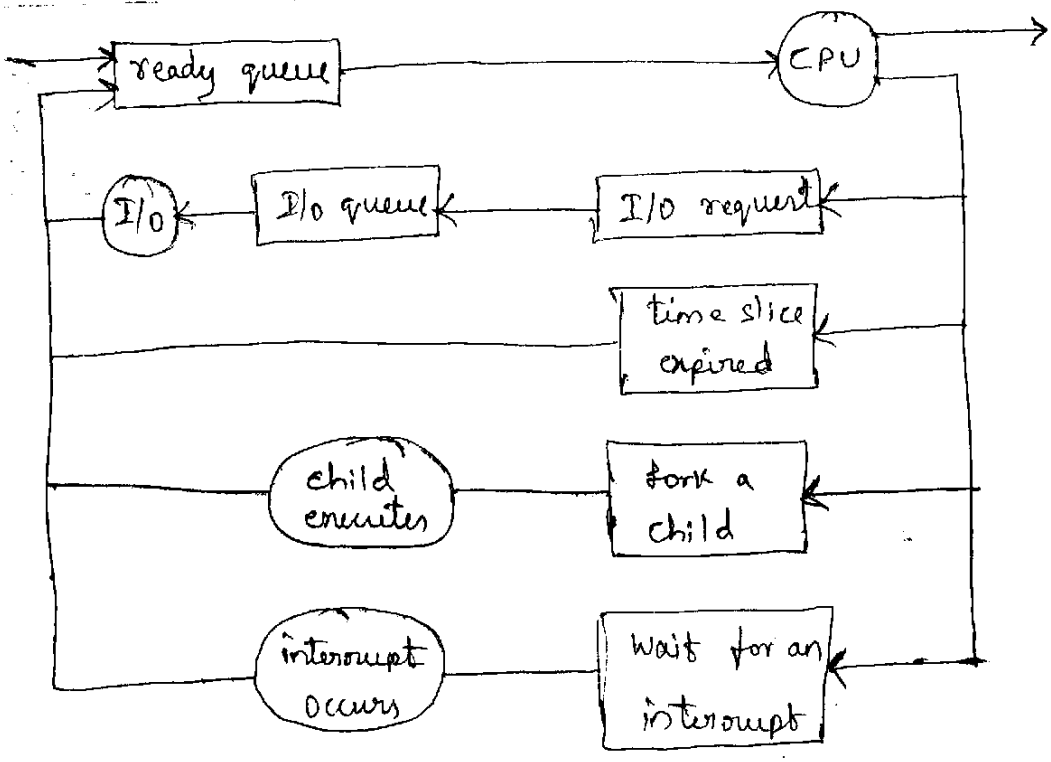


figure ↑ The ready queue and various I/O device queues

*) A common representation of process scheduling is a queueing diagram as shown in figure (next page(5)).

*) Each rectangular box represents a queue.
*) Circle represents the resources that serve the queues.
*) Two types of queues are present

1) The ready queue.
2) Set of device queues.



queueing-diagram representation of process scheduling

3.2.2) Schedulers :-

*) Processes will be present in a queue, Operating system should select a process from the queue. This selection process is carried out by the appropriate Schedulers.

*) If the number of waiting processes are huge then these processes are spooled to a mass-storage device, where they are kept for later execution.

*) The long-term scheduler, or job scheduler, selects processes from this pool and loads them into memory for execution. (pool to ready queue)

*) The short-term scheduler, or CPU scheduler, selects from among the processes that are ready to execute and allocates the CPU to one of them.

*) The short time scheduler must be fast. because if it takes 10 milliseconds to decide which process to execute of type 100 milliseconds, then $10/(100+10) = 9$ percent of the cpu time is wasted.

*) The long-term scheduler controls the degree of multiprogramming. The average rate of process creation must be equal to the average departure rate of processes leaving the system.

*) Thus long-term scheduler may need to be invoked only when a process leaves the system.

*) processes can be described in 2 ways:

(1) I/O bound.
(2) cpu bound.

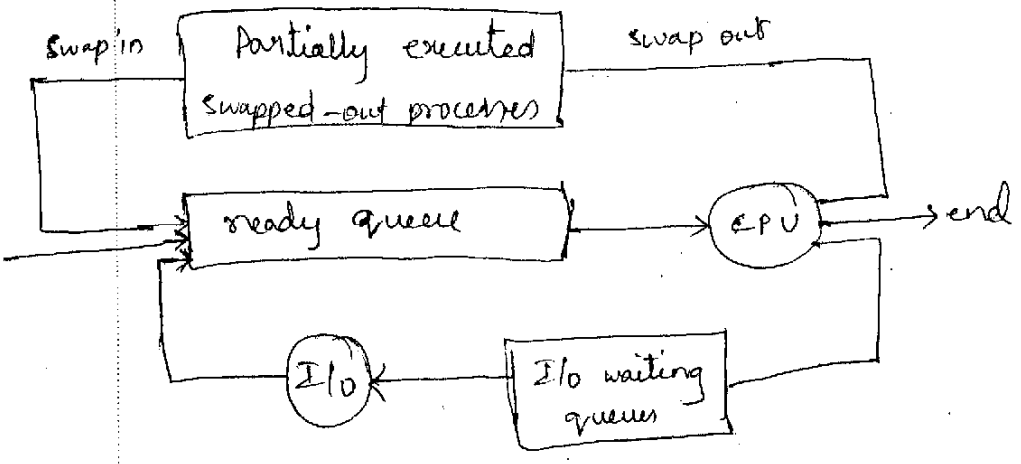
(1) An I/O bound process is one that spends more of its time doing I/O than it spends doing computations.

(2) A CPU-bound process, in contrast, generates I/O requests infrequently, using more of its time doing computations.

*) long-term scheduler should select a combination of CPU-bound and I/O-bound processes.

*) Some operating systems, such as time-sharing systems, may introduce an additional, intermediate level of scheduling know as medium-term scheduler.

*) The key idea behind a medium-term scheduler is that sometimes it can be advantageous to remove process from memory and thus reduce the degree of multiprogramming. Later, the process can be reintroduced into memory, and its execution can be continued where it left off.



Addition of medium-term scheduling to the queueing diagram.

3.2.3) Context switch :

*) Switching the CPU to another requires performing a state save of the current process and a state restore of a different process. This task is known as a context switch.

*) Interrupts cause the operating system to change a CPU from its current task and to run a kernel's task. Such operations happen frequently on general-purpose systems.

*) When an interrupt occurs, the system needs to save the current context of the process running on the CPU so that it can restore that context when its processing is done, essentially suspending the process and then resuming it. The context is represented in the PCB of the process.

3.3) Operations on Processes :-

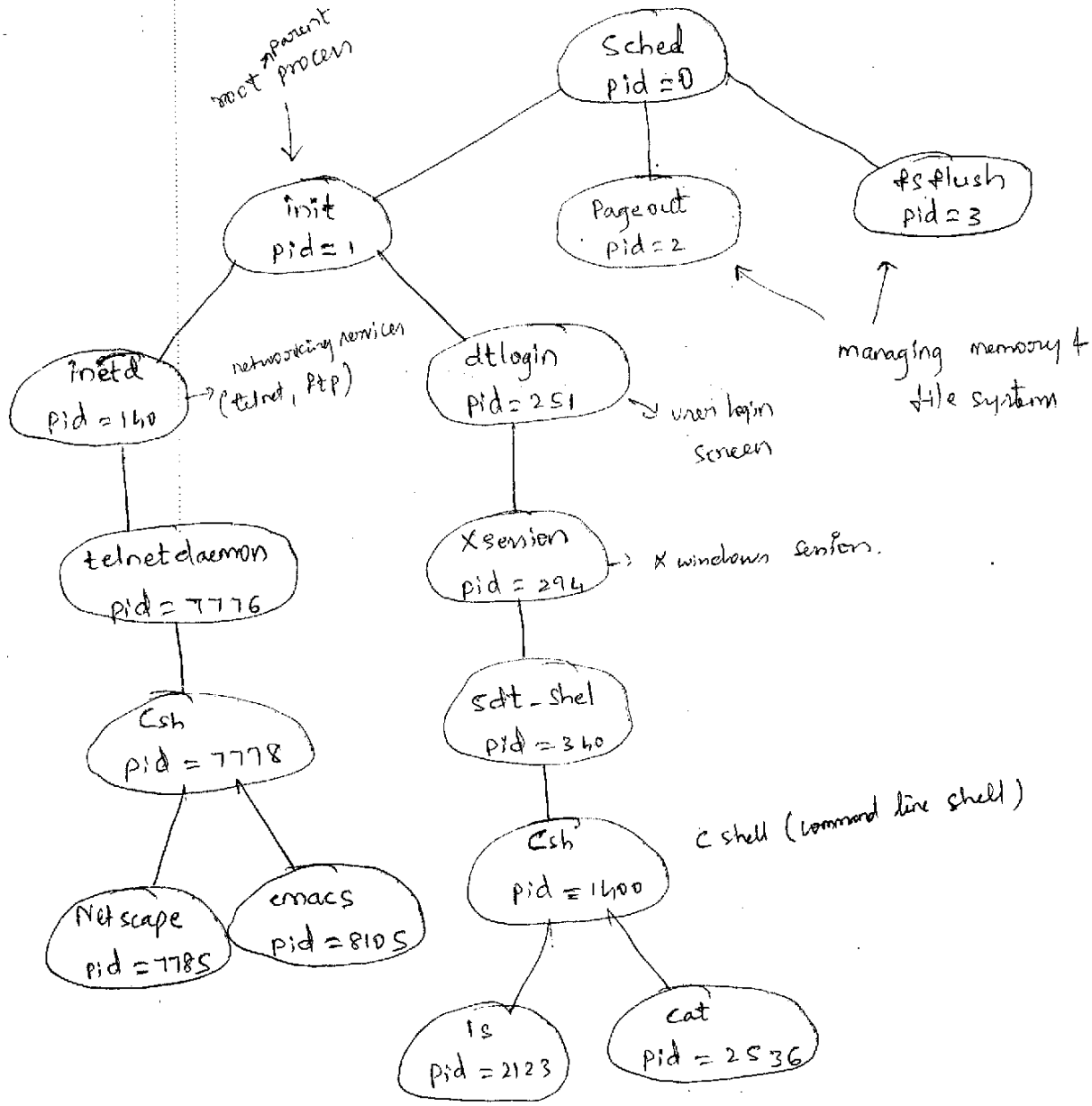
*) The processes in most systems can execute concurrently, and they may be created and deleted dynamically. Thus, these systems must provide a mechanism for process creation and termination.

3.3.1) Process Creation :-

*) A process may create several new processes, via a create-process system call, during the course of execution. The creating process is called a parent process, and the new processes are called the children of that process. Each of these new processes may in turn create other processes, forming a tree of processes.

*) Most operating systems identify processes according to a unique process identifier (pid), which is typically an

integer number. following figure illustrates a typical process tree for the solaris operating system, showing the name of each process and its Pid.



A tree of processes on a typical solaris system.

*) In general, a process will need certain resources (cpu time, memory files, I/o devices) to accomplish its task.

*) When a process creates a subprocess, that subprocess may be able to obtain its resources directly from the

operating system.

*) When a process creates a new process, two possibilities exist in terms of execution:

(1) The parent continues to execute concurrently with its children.

(2) The parent waits until some or all its children have terminated.

*) There are also two possibilities in terms of the address space of the new process.

(1) The child process is a duplicate of the parent process (it has the same program and data as the parent).

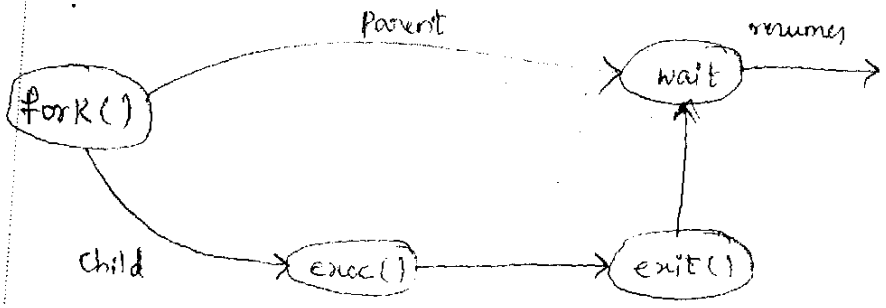
(2) The child process has a new program loaded into it.

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
    pid_t pid;

    pid = fork(); /* fork a child process */

    if (pid < 0) { /* error occurred */

    fprintf(stderr, "Fork Failed");
        return 1; }

    else if (pid == 0) { /* child process */
        execlp("/bin/ls", "ls", NULL); }
```

```
else { /* parent process */
/* parent will wait for the child to complete */
    wait (NULL);
    Printf("child Complete");
}
```

creating a separate process using the UNIX fork() system call.

here, A new process is created by the fork() system call. The new process consists of a copy of the address space of the original process. This mechanism allows the parent process to communicate easily with its child process. Both processes continue execution at the instruction after the fork(), with one difference : the return code for the fork() is zero for the new (child) process, whereas the (nonzero) process identifier of the child is returned to the parent.



Process Creation using fork() system call.

*) When ever the fork() system call is called by parent, The parent waits for the child process to complete with the wait() system call. When the child process completes the parent process resumes from the wait() as shown in the figure above.

Creating a seperate process using the Win32 API

```
#include<stdio.h>
#include<windows.h>

int main(VOID)
{
   STARTUPINFO si;
   PROCESS_INFORMATION pi;

ZeroMemory(&si, sizeof(si)); // allocate memory
  si.cb = sizeof(si);
Zeromemory(&pi, sizeof(pi));

// create child process
if(!CreateProcess(NULL, "C:\\WINDOWS\\System32\\mspaint.exe", NULL,
   NULL, FALSE, 0, NULL, NULL, &si, &pi))
{
  fprintf(stderr, "Create Process Failed");
     return -1;
}
// parent will wait for the child to complete
~~WaitForSingle~~
WaitForSingleObject(pi.hProcess, INFINITE);
   printf("child Complete");
   //close handles
   CloseHandle(pi.hProcess);
   CloseHandle(pi.hThread);  }
```

use command line

command line

don't inherit process handle

don't inherit Thread handle

disable handle inheritance

no creation flags

use parents environment block

use parents existing directory

*) processes are created in the Win32 API using the CreateProcess() function.

*) CreateProcess() expects ten parameters.

*) The C program shown in previous page illustrates the CreateProcess() function, which creates a child process that loads the application mspaint.exe

*) Two parameters passed to CreateProcess() are instances of the STARTUPINFO and PROCESS_INFORMATION

*) STARTUPINFO specifies properties such as

(1) Window size
(2) appearance
(3) Handles to standard I/O files.

*) PROCESS_INFORMATION specifies :

(1) identifiers to the newly created process.
(2) identifiers to the threads.

*) ZeroMemory() function to allocate memory for each of these structures before proceeding with CreateProcess().

3.3.2) process Termination :-

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

*) All the resources of the process - including physical and virtual memory, open files, and I/O buffers are deallocated by the operating system.

*) Termination can occur in other circumstances as well. A process can cause the termination of another process via an appropriate system call.

*) Usually, such a system call can be invoked only by by the parent of the process that is to be terminated.

*) A Parent may terminate the execution of one of its children for a variety of reasons, such as these:

(1) The child has exceeded its usage of some of the resources that it has been allocated.

(2). The task assigned to the child is no longer required.

(3) The parent is exiting, and the operating system does not allow a child to continue if its parent terminates.

*) Some systems do not allow a child to exist if it's parent has terminated. In such systems, if a process terminates, then all its children must also be terminated. This phenomenon, referred to as cascading termination, is normally initiated by the operating system.

Under direct communication, each process that wants to communicate must explicitly name the recipient or sender of the communication. In this scheme, the send() and receive() primitives are defined as:

(1) send(P, message) - Send a message to process P.

(2) Receive(Q, message) - Receive a message from process Q.

A communication link in this scheme has the following properties:

(1) A link is established automatically between every pair of processes that want to communicate. The processes need to know only each other's identity to communicate.

(2) A link is associated with exactly two processes.

(3) Between each pair of processes, there exists exactly one link.

*) The above scheme exhibits symmetry in addressing; that is both the sender process and the receiver process must name the other to communicate.

*) The asymmetry scheme of addressing. Here only the sender names the recipient; the recipient is not required to name the sender. In this scheme, the send() and receive() primitives are defined as follows:

(1) send(P, message) - Send a message to process P.

(2) Receive(id, message) - Receive a message from any process; the variable id is set to the name of the process with which communication has taken place.

(47)

*) With indirect communication, the messages are sent to and received from mailboxes, or ports

*) A mailbox can be viewed as an object into which messages can be placed by processes and from which messages can be removed.

*) A Process can communicate with some other process via a number of different mailboxes.

*) Two Processes can communicate only if the processes have a shared Mailbox.

The Send() and receive() primitives are defined as follows:

(1) Send(A, message) - Send a message to mailbox A.

(2) receive(A, message) - Receive a message from mailbox A.

In this scheme, a communication link has the following properties.

(1) A link is established between a pair of processes only if both members of the pair have a shared mailbox.

(2) A link may be associated with more than two processes.

(3) Between each pair of communicating processes, there may be number of different links, where each link will be having its own mailbox.

```
item nextProduced;
while (true) {
    /* produce an item in nextProduced */
    while (((in+1) % BUFFER_SIZE) == out)
        ; /* do nothing */
    buffer[in] = nextProduced;
    in = (in+1) % BUFFER_SIZE;
}
```

The Producer Process

```
item nextConsumed;
while (true) {
    while (in == out)
        ; // do nothing
    nextConsumed = buffer[out];
    out = (out+1) % BUFFER_SIZE;
    /* Consume the item in nextConsumed */
}
```

The Consumer process.

3.4.2) Menage - Passing Systems :-

*) Menage passing provides a mechanism to allow processes to communicate without sharing the same (48)

address space and is particularly useful in a distributed environment, where the communicating processes may reside on different computers connected by a network.

*) A menage-passing facility provides at least two operations

(1) Send (menage)

(2) Receive (menage).

*) Menages sent by a process can be of either fixed or variable size.

*) If process P and Q want to communicate, they must send menages to and receive menages from each other.

*) A communication link must exist between them. This link can be implemented in a variety of ways.

(1) Direct or Indirect communication

(2) Synchronous or asynchronous communication

(3) Automatic or explicit buffering.

3.4.2.1) Naming :

Processes that want to communicate must have a way to refer to each other.

They can use either direct or indirect communication.

3.4) Inter process Communication :-

*) Processes executing concurrently in the operating system may be either independent processes or cooperating processes.

*) A Process is independent if it cannot affect or be affected by the other processes executing in the system. Independent process does not share data with any other process.

*) A process is cooperating if it can affect or be affected by the other processes executing in the system. Clearly, any process that shares data with other processes is a cooperating process.

There are several reasons for providing an environment that allows process cooperation :

(1) Information Sharing :- Since several users may be interested in the same piece of information, we must provide an environment to allow concurrent access to such information.

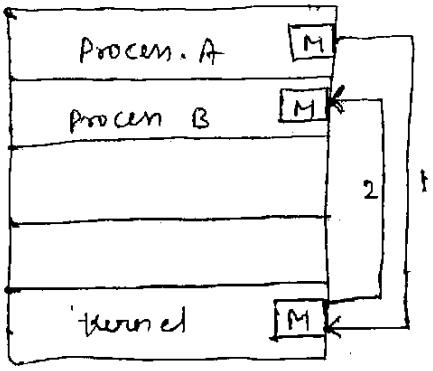
(2) Computation speedup :- If we want a particular task to run faster, we must break it into subtasks, each of which will be executing in parallel with the others.

(3) Modularity :- System has to construct in modular fashion that is dividing the system functions into separate processes or threads.

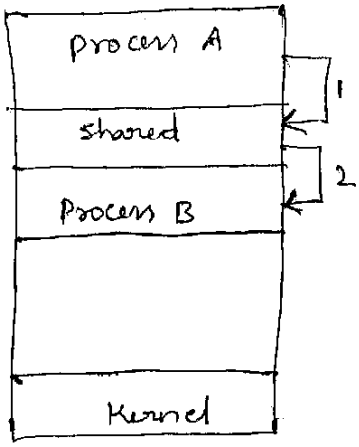
(4) Convenience :- Even an individual user may work on many tasks at the same time. For instance, a user may be editing, printing, and compiling in parallel.

*) There are two fundamental models of interprocess communication.

(1) Shared Memory.
(2) Menage passing.



menage passing.



shared memory.

(1) Shared-Memory System :-

*) Interprocen communication using shared memory requires communicating procenes to establish a region of shared memory.

*) In the shared-memory model, a region of memory that is shared by cooperating procenes is established. procenes can then exchange information by reading and writing data to the shared region.

*) Typically, a shared-memory region resides in the address space of the procen creating the shared memory segment.

*) Normally the operating (OS) tries to prevent on process from accessing another process's memory. Shared memory requires that two or more processes agree to remove this restriction. They can then exchange information by reading and writing data in the shared areas.

*) To illustrate the concept of cooperating processes, let's consider the producer - consumer problem.

*) A Producer process produces information that is consumed by a consumer process. For example a compiler may produce assembly code which is consumed by an assembler. The assembler, in turn, may produce object modules, which are consumed by the loader.

In a client-Server model. we generally think of a server as a producer and a client as a consumer.

Ex:- web server produces HTML files and images, which are consumed by the client web browser requesting the resource.

*) with the help of buffer a producer can produce items into buffer where an consumer can consume items from the buffer.

*) Two types of buffer can be used.

(1) bounded buffer.

(2) unbounded buffer.

(1) bounded buffer :- Assumes a fixed buffer size. In this case, the consumer must wait if the buffer is empty, and the producer must wait if the buffer is full.

(2) Unbounded buffer :- places no practical limit on the size of the buffer. The consumer may have to wait for new items, but the producer can always produce new items.

*) The following variables reside in a region of memory shared by the producer and consumer process:

```
#define BUFFER_SIZE 10

typedef struct {
    . . .
} item;

item buffer[BUFFER_SIZE];
int in=0;
int out=0;
```

*) The shared buffer is implemented as a circular array with two logical pointers : in and out.

*) The variable in points to the next free position in the buffer ; out points to the first full position in the buffer.

*) The buffer is empty when in == out

*) The buffer is full when ((in+1) % BUFFER_SIZE) == out.

*) The code for producer process and consumer process is shown in next page. (page 12).

P.T.O

*) A mailbox may be owned either by a process or by the operating system.

*) If the mailbox is owned by a process then the mailbox is part of the address space of the process.

3.4.2.2) Synchronization :-

Message passing may be either blocking or nonblocking also known as Synchronous and asynchronous.

(1) Blocking send :- The sending process is blocked until the message is received by the receiving process or by the mailbox.

(2) Nonblocking send :- The sending process sends the message and resumes operation.

(3) Blocking receive :- The receiver blocks until a message is available.

(4) Nonblocking receive :- The receiver retrieves either a valid message or a null.

3.4.2.3) Buffering :-

Whether communication is direct or indirect, messages exchanged by communicating processes reside in a temporary queue. Basically such queues can be implemented in three ways:

(1) Zero Capacity :- The queue has a maximum length of zero; thus the link cannot have any messages waiting in it. In this case, the sender must block

(51)

until the recipient receives the message.

(2) Bounded capacity :- The queue has finite length thus, at most n messages can reside in it. If the queue is not full when a new message is sent, the message is placed in the queue, and sender can continue execution without waiting.

(3) Unbounded capacity :- The queue's length is potentially infinite; thus any number of messages can wait in it. The sender never blocks.

Chapter - 2

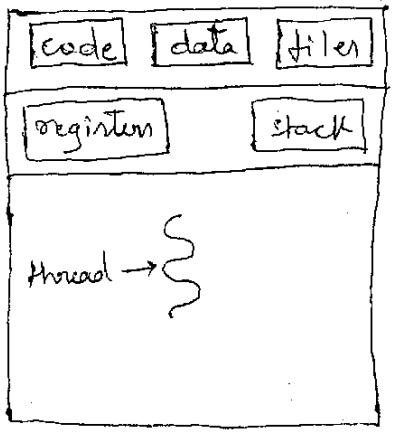
Multithreaded Programming

4.1) Overview :-

A Thread is a basic unit of CPU utilization, it comprises a thread ID, a program counter, a register set, and a stack.

*) A traditional (or heavyweight) process has a single thread of control. If a process has multiple threads of control, it can perform more than one task at a time.

Single-threaded Process



Multithreaded Process