DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

Course: Operating System IAT-1 Solutions

Code: 15CS64

Question1 : Define Operating Systems. Explain the Dual-mode operations of Operating System with a neat diagram.

Solutions:

Operating System

An OS is a program that acts as an intermediary between
- Computer-user
- Computer-hardware

It also provides a basis for application-programs

Goals of OS:
- To execute programs.
- To make solving user-problems easier.
- To make the computer convenient to use.

Some OS are designed to be Convenient , others to be efficient, and others some combination of the two

The OS (also called kernel) is the one program running at all times on the computer.

Different types of OS:
- Mainframe OS is designed to optimize utilization of hardware.
- Personal computer (PC) OS supports complex game, business application.
- Handheld computer OS is designed to provide an environment in which a user can easily interface with the computer to execute programs.

Dual-Mode Operation

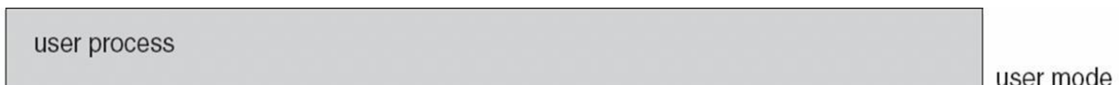
Since the operating system and the user programs share the hardware and software resources of the computer system, it has to be made sure that an error in a user program cannot cause problems to other programs and the Operating System running in the system.

The approach taken is to use a hardware support that allows us to differentiate among various modes of execution.

The system can be assumed to work in two separate **modes** of operation:
- **user mode** and
- **kernel mode** (**supervisor mode, system mode, or privileged mode).**

A hardware bit of the computer, called the **mode bit**, is used to indicate the current mode: kernel (0) or user (1). With the mode bit, we are able to distinguish between a task that is executed by the operating system and one that is executed by the user.

When the computer system is executing a user application, the system is in user mode. When a user application requests a service from the operating system (via a system call), the transition from user to kernel mode takes place.



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 (that is, changes the mode bit from 1 to 0). Thus, whenever the operating system gains control of the computer, it is in kernel mode.

The dual mode of operation provides us with the means for protecting the operating system from errant users—and errant users from one another.

The hardware allows privileged instructions to be executed only in kernel mode. If an attempt is made to execute a privileged instruction in user mode, the hardware does not execute the instruction but rather treats it as illegal and traps it to the operating system. The instruction to switch to user mode is an example of a privileged instruction.

Initial control is within the operating system, where instructions are executed in kernel mode. When control is given to a user application, the mode is set to user mode. Eventually, control is switched back to the operating system via an interrupt, a trap, or a system call.

b) Timer

Operating system uses timer to control the CPU. A user program cannot hold CPU for a long time, this is prevented with the help of timer.
A timer can be set to interrupt the computer after a specified period. The period may be **fixed** (for example, 1/60 second) or **variable** (for example, from 1 millisecond to 1 second).

Fixed timer – After a fixed time, the process under execution is interrupted.

Variable timer – Interrupt occurs after varying interval. This is implemented using a fixed-rate clock and a counter. The operating system sets the counter. Every time the clock ticks, the counter is decremented. When the counter reaches 0, an interrupt occurs.

Before changing to the user mode, the operating system ensures that the timer is set to interrupt. If the timer interrupts, control transfers automatically to the operating system, which may treat the interrupt as a fatal error or may give the program more time.

OS Services

Process Management

A program under execution is a process. A process needs resources like CPU time, memory, files, and I/O devices for its execution. These resources are given to the process when it is created or at run time. When the process terminates, the operating system reclaims the resources.

The program stored on a disk is a **passive entity** and the program under execution is an **active entity**. A single-threaded process has one program **counter specifying** the next instruction to execute. The CPU executes one instruction of the process after another, until the process

completes. A multithreaded process has multiple program counters, each pointing to the next instruction to execute for a given thread.
The operating system is responsible for the following activities in connection with process management:

- Scheduling process and threads on the CPU
- Creating and deleting both user and system processes
- Suspending and resuming processes
- Providing mechanisms for process synchronization
- Providing mechanisms for process communication

Memory Management

Main memory is a large array of words or bytes. Each word or byte has its own address. Main memory is the storage device which can be easily and directly accessed by the CPU. As the program executes, the central processor reads instructions and also reads and writes data from main memory.

To improve both the utilization of the CPU and the speed of the computer's response to its users, general-purpose computers must keep several programs in memory, creating a need for memory management.

The operating system is responsible for the following activities in connection with memory management:
Keeping track of which parts of memory are currently being used by user.
Deciding which processes and data to move into and out of memory.
Allocating and deallocating memory space as needed.

Storage Management

There are three types of storage management i) File system management ii) Mass-storage management iii) Cache management.

File-System Management

File management is one of the 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. Each of these media has its own characteristics and physical organization. Each medium is controlled by a device, such as a disk drive or tape drive, that also has its own unique characteristics.

A file is a collection of related information defined by its creator. Commonly, files represent programs and data. Data files may be numeric, alphabetic, alphanumeric, or binary. Files may be free-form (for example, text files), or they may be formatted rigidly (for example, fixed fields).

The operating system implements the abstract concept of a file by managing mass storage media. Files are normally organized into directories to make them easier to use. When multiple users have access to files, it may be desirable to control by whom and in what ways (read, write, execute) files may be accessed.

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

- Creating and deleting files
- Creating and deleting directories to organize files
- Supporting primitives for manipulating files and directories
- Mapping files onto secondary storage
- Backing up files on stable (nonvolatile) storage media

Mass-Storage Management

As the main memory is too small to accommodate all data and programs, and as the data that it holds are erased when power is lost, the computer system must provide secondary storage to back up main memory. Most modern computer systems use disks as the 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. Hence, the proper management of disk storage is of central importance to a computer system. The operating system is responsible for the following activities in connection with disk management:

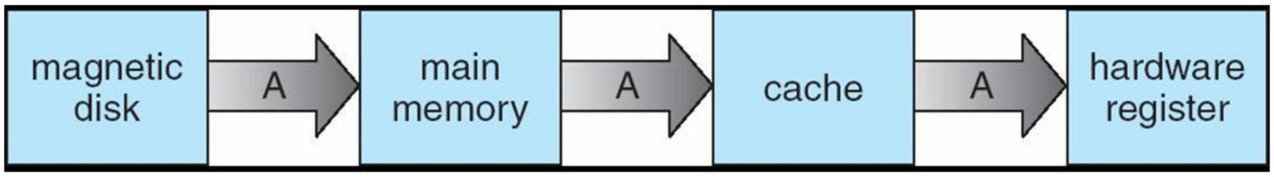
- Free-space management
- Storage allocation
- Disk scheduling

As the secondary storage is used frequently, it must be used efficiently. The entire speed of operation of a computer may depend on the speeds of the disk. Magnetic tape drives and their tapes, CD, DVD drives and platters are **tertiary storage** devices. The functions that operating systems provides include mounting and unmounting media in devices, allocating and freeing the devices for exclusive use by processes, and migrating data from secondary to tertiary storage.

Caching

Caching is an important principle of computer systems. Information is normally kept in some storage system (such as main memory). As it is used, it is copied into a faster storage system—the cache—as temporary data. When a particular piece of information is required, first we check whether it is in the cache. If it is, we use the information directly from the cache; if it is not in cache, we use the information from the source, putting a copy in the cache under the assumption that we will need it again soon.

Because caches have limited size, **cache management** is an important design problem. Careful selection of the cache size and page replacement policy can result in greatly increased performance.
The movement of information between levels of a storage hierarchy may be either explicit or implicit, depending on the hardware design and the controlling operating-system software. For instance, data transfer from cache to CPU and registers is usually a hardware function, with no operating-system intervention. In contrast, transfer of data from disk to memory is usually controlled by the operating system.
In a hierarchical storage structure, the same data may appear in different levels of the storage system. For example, suppose to retrieve an integer A from magnetic disk to the processing program. The 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.



In a multiprocessor environment, in addition to maintaining internal registers, each of the CPUs also contains a local cache. In such an environment, a copy of A may exist simultaneously in several caches. Since the various CPUs can all execute concurrently, any update done to the value of A in one cache is immediately reflected in all other caches where A resides. This situation is called **cache coherency,** and it is usually a hardware problem (handled below the operating-system level).

I/O Systems
One of the purposes of an operating system is to hide the peculiarities of specific hardware devices from the user. The I/O subsystem consists of several components:

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

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

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. For that purpose, mechanisms ensure that files, memory segments, CPU, and other resources can be operated on by only those processes that have gained proper authorization from the operating system.
If a computer system has multiple users and allows the concurrent execution of multiple processes, then access to data must be regulated. For that purpose, there are mechanisms which ensure that files, memory segments, CPU, and other resources can be operated on by only those processes that have gained proper authorization from the operating system.
For example, 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 for a long time. Device-control registers are not accessible to users, so the integrity of the various peripheral devices is protected.

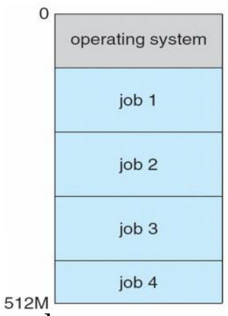
Protection is a mechanism for controlling the access of processes or users to the resources defined by a computer system. This mechanism must provide means for specification of the controls to be imposed and means for enforcement.
Protection improves reliability. A protection-oriented system provides a means to distinguish between authorized and unauthorized usage. A system can have adequate protection but still be prone to failure and allow inappropriate access.
Consider a user whose authentication information is stolen. Her data could be copied or deleted, even though file and memory protection are working. It is the job of **security** to defend a system from external and internal attacks. Such attacks spread across a huge range and include viruses and worms, denial-of service attacks etc.
Protection and security require the system to be able to distinguish among all its users. Most operating systems maintain a list of user names and associated **user identifiers (user IDs).** When a user logs in to the system, the authentication stage determines the appropriate user ID for the user.

Question2: Distinguish between the following terms:

i) Symmetric &amp; Asymmetric Multiprocessors
ii) ii) Multiprocessor systems and clustered systems.

Solution:

Multiprogramming
One of the most important aspects of operating systems is the ability to multi-program. A single user cannot keep either the CPU or the I/O devices busy at all times. **Multiprogramming** increases CPU utilization by organizing jobs, so that the CPU always has one to execute.



The operating system keeps several jobs in memory simultaneously as shown in figure. This set of jobs is a subset of the jobs kept in the job pool. Since the number of jobs that can be kept simultaneously in memory is usually smaller than the number of jobs that can be kept in the job pool (in secondary memory). The operating system picks and begins to execute one of the jobs in memory. Eventually, the job may have to wait for some task, such as an I/O operation, to complete. In a non-multi-programmed system, the CPU would sit idle. In a multi-programmed system the operating system simply switches and executes, another job. When *that* job needs to wait, the CPU is switched to *another* job, and so on.

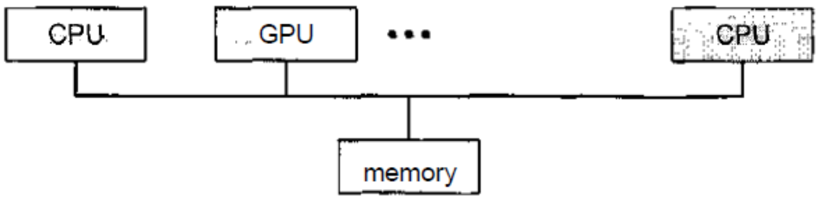
Eventually, the first job finishes waiting and gets the CPU back. Thus the CPU is never idle.
Multi-programmed systems provide an environment in which the various system resources (for example, CPU, memory, and peripheral devices) are utilized effectively, but they do not provide for user interaction with the computer system.

Multitasking

In **Time sharing** (or **multitasking) systems,** a single 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. The user feels that all the programs are being executed at the same time. Time sharing requires an **interactive** (or **hands-on) computer system,** which provides direct communication between the user and the system. The user gives instructions to the operating system or to a program directly, using a input device such as a keyboard or a mouse, and waits for immediate results on an output device. Accordingly, the **response time** should be short—typically less than one second.
A time-shared operating system allows many users to share the computer simultaneously. As the system switches rapidly from one user to the next, each user is given the impression that the entire computer system is dedicated to his use only, even though it is being shared among many users.

Multiprocessor

A **multiprocessor system** is a computer system having two or more CPUs within a single computer system, each sharing main memory and peripherals. Multiple programs are executed by multiple processors parallel.



Multiprocessor Systems (parallel systems or **tightly coupled systems) –**

Systems that have two or more processors in close communication, sharing the computer bus, the clock, memory, and peripheral devices are the multiprocessor systems.

Multiprocessor systems have three main advantages:

1. **Increased throughput -** In multiprocessor system, as there are multiple processors execution of different programs take place simultaneously. Even if the number of processors is increased the performance cannot be simultaneously increased. This is due to the overhead incurred in keeping all the parts working correctly and also due to the competition for the shared resources. The speed-

up ratio with *N* processors is not *N*, rather, it is less than N. Thus the speed of the system is not has expected.

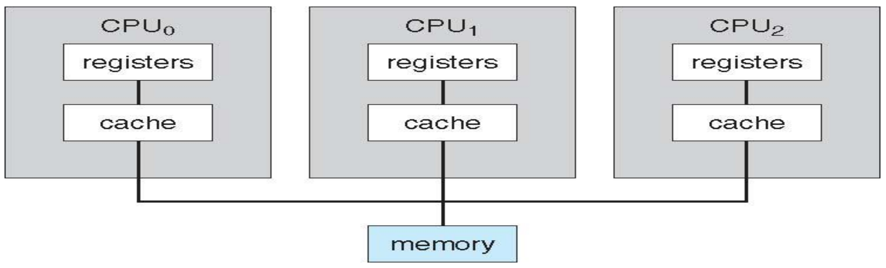
2. **Economy of scale -** Multiprocessor systems can cost less than equivalent number of many single-processor systems. As the multiprocessor systems share peripherals, mass storage, and power supplies, the cost of implementing this system is economical. If several processes are working on the same data, the data can also be shared among them.

3. **Increased reliability-** In multiprocessor systems functions are shared among several processors. If one processor fails, the system is not halted, it only slows down. The job of the failed processor is taken up, by other processors.
   Two techniques to maintain 'Increased Reliability' - graceful degradation & fault tolerant
   - **Graceful degradation –** As there are multiple processors when one processor fails other process will take up its work and the system goes down slowly.
   - **Fault tolerant –** When one processor fails, its operations are stopped, the system failure is then detected, diagnosed, and corrected.

The HP Nonstop system uses both hardware and software duplication to ensure continued operation despite faults. The system consists of multiple pairs of CPUs. Both processors in the pair execute same instruction and compare the results. If the results differ, then one CPU of the pair is at fault, and both are halted. The process that was being executed is then moved to another pair of CPUs, and the instruction that failed is restarted. This solution is expensive, since it involves special hardware and considerable hardware duplication.

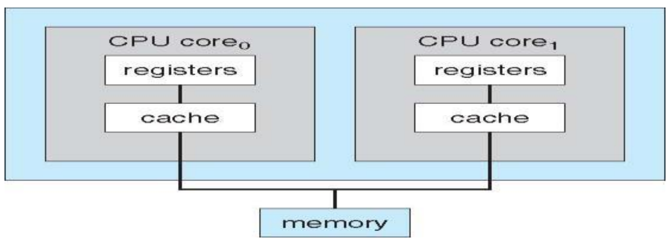
There are two types of multiprocessor systems –
- **Asymmetric multiprocessing**
- **Symmetric multiprocessing**

1) **Asymmetric multiprocessing – (Master/Slave architecture)** Here each processor is assigned a specific task, by the master processor. A master processor controls the other processors in the system. It schedules and allocates work to the slave processors.
2) **Symmetric multiprocessing (SMP) –** All the processors are considered as peers. There is no master-slave relationship. All the processors have its own registers and CPU, only memory is shared.



The benefit of this model is that many processes can run simultaneously. *N* processes can run if there are *N* CPUs—without causing a significant deterioration of performance. Operating systems like Windows, Windows XP, Mac OS X, and Linux—now provide support for SMP.

A recent trend in CPU design is to include multiple compute **cores** on a single chip. The communication between processors within a chip is faster than communication between two single processors.



Clustered Systems
Clustered systems are two or more individual systems connected together via network and sharing software resources. Clustering provides **high-availability** of resources and services. The service will continue even if one or more systems in the cluster fail. High availability is generally obtained by storing a copy of files (s/w resources) in the system.

There are two types of Clustered systems – **asymmetric** and **symmetric**

In **asymmetric clustering** – one system is in **hot-standby mode** while the others are 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.

In **symmetric clustering** – two or more systems are running applications, and are monitoring each other. This mode is more efficient, as it uses all of the available hardware. If any system fails, its job is taken up by the monitoring system.

Other forms of clusters include parallel clusters and clustering over a wide-area network (WAN). Parallel clusters allow multiple hosts to access the same data on the shared storage. Cluster technology is changing rapidly with the help of **SAN (storage-area networks)**. Using SAN resources can be shared with dozens of systems in a cluster that are separated by miles.

Question3: What is a thread? Explain with a neat diagram. What is the need for multithreaded processes? Indicate the four major benefits of multithreaded programming.

Solution:

Threads:

A thread is the smallest unit of processing that can be performed in an OS. In most modern operating systems, a thread exists within a process - that is, a single process may contain multiple threads.

1. Multi-threaded Programming

- A thread is a basic unit of CPU utilization.
- It consists of
  → thread ID
  → PC
  → register-set and
  →a stack.
- It shares with other threads belonging to the same process its code-section & data-section and other OS resources, such as open files and signals.
- A traditional (or **heavy weight**) 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. Such a process is called **multi-threaded process** (Figure 2.1).

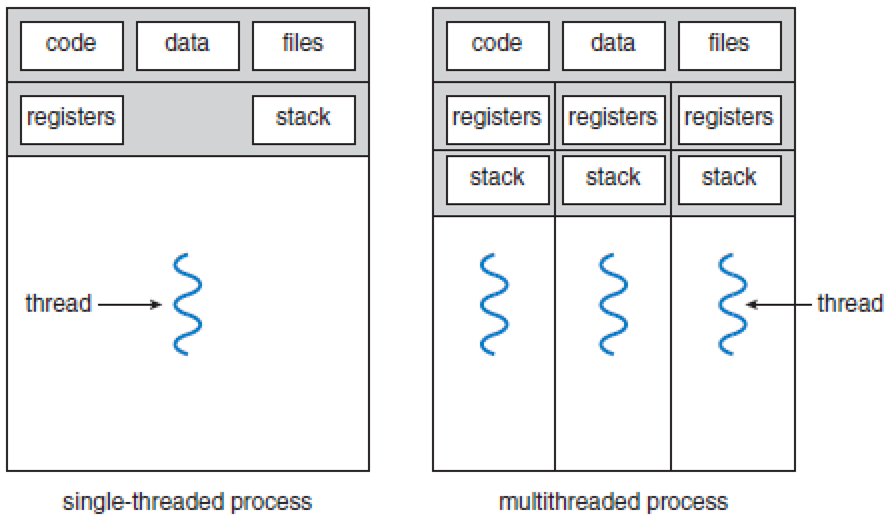


Figure 2.1 Single-threaded and multithreaded processes

2. Motivation

- The software-packages that run on modern PCs are multithreaded. An application is implemented as a separate process with several threads of control.
  - Example1: A word processor may have
    - → first thread for displaying graphics
    - → second thread for responding to keystrokes and
    - → Third thread for performing grammar checking.
  - Example2: A web browser may have one thread display images or text while another thread retrieves data from the network.
- In some situations, a single application may be required to perform several similar tasks.
  - Example1: A web-server accepts client requests for web pages, images, sound, and so forth. In this case the server would create a separate thread that would listen for client requests and create another thread to service the request.
- RPC servers are multithreaded.
  - When a server receives a message, it services the message using a separate thread.This allows the server to service several concurrent requests.
- Most OS kernels are multithreaded; Several threads operate in kernel, and each thread performs a specific task, such as
  - → managing devices or
  - → interrupt handling.

3. Benefits

1) Responsiveness

- A program may be allowed to continue running even if part of it is blocked. Thus, increasing responsiveness to the user. For instance, a multithreaded web browser could still allow user interaction in one thread while an image was being loaded in another thread.

2) *Resource Sharing*

- By default, threads share the memory (and resources) of the process to which they belong. Thus, an application is allowed to have several different threads of activity within the same address-space.

3) *Economy*

- Allocating memory and resources for process-creation is costly. Thus, it is more economical to create and context-switch threads. For example, in Solaris, creating a process is about thirty times slower than creating a thread, and context switching is about five times slower.

4) *Utilization of Multiprocessor Architectures*

- In a multiprocessor architecture, threads may be running in parallel on different processors. Thus, parallelism will be increased.

Question4: What is Inter-Process Communication (IPC)? Explain the two basic communication models of IPC with a neat diagram.

Solution:

Inter Process Communication (IPC)

- Processes executing concurrently in the OS may be 1) Independent processes or 2) Co-operating processes.

1) A process is **independent** if
   i) The process cannot affect or be affected by the other processes.
   ii) The process does not share data with other processes.
2) A process is **co-operating** if
   i) The process can affect or be affected by the other processes.
   ii) The process shares data with other processes.

- Advantages of process co-operation:

1) Information Sharing

- Since many users may be interested in same piece of information (ex: shared file).

2) Computation Speedup

- We must break the task into subtasks.
- Each subtask should be executed in parallel with the other subtasks.
- The speed can be improved only if computer has multiple processing elements such as
  → CPUs or
  → I/O channels.

3) Modularity

- Divide the system-functions into separate processes or threads.

4) Convenience

- An individual user may work on many tasks at the same time.
- For ex, a user may be editing, printing, and compiling in parallel.

- Two basic models of IPC (Figure 1.31): 1) Shared-memory and
  2) Message passing.

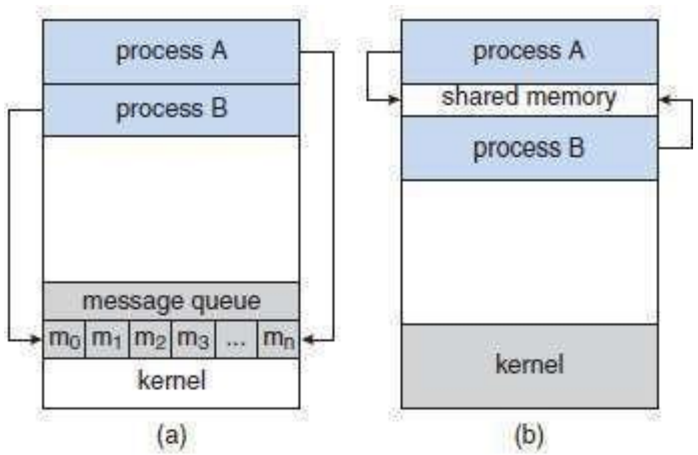


Figure 1.31 Communications models. (a) Message passing. (b) Shared-memory

1. Shared-Memory Systems

- Communicating-processes must establish a region of shared-memory.
- A shared-memory resides in address-space of the process creating the shared-memory. Other processes must attach their address-space to the shared-memory.
- The processes can then exchange information by reading and writing data in the shared-memory.
- The processes are also responsible for ensuring that they are not writing to the same location simultaneously.
- Two types of buffers can be used:
  1) **Unbounded-Buffer** places no practical limit on the size of the buffer.
  2) **Bounded-Buffer** assumes that there is a fixed buffer-size.
- Advantages:
  1) Allows maximum speed and convenience of communication.
  2) Faster.
- For ex, **Producer-Consumer Problem for bounded buffer**:
- Producer-process produces information that is consumed by a consumer-process.
- The following variables reside in a region of memory shared by the producer and consumer processes:

```
#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 and **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 is as follows:

```
item nextProduced;
while (true)
{
                /* Produce an item in nextProduced */
                while (((in = (in + 1) % BUFFER_SIZE ) == out);   /* do nothing -- no free buffers */
                buffer [in] = nextProduced;
                in = (in + 1) % BUFFER SIZE;
 }
```

The code for consumer process is as follows:

```
item nextConsumed;

while (true)
{
          while (in == out) ; // do nothing -- nothing to consume
        /*remove an item from the buffer*/
                nextConsumed = buffer[out];
                 out = (out + 1) % BUFFER_ SIZE;
       /*consume the item in nextConsumed*/
    }
```

From the code, the producer process has a local variable ***nextProduced*** *in which the new item to be produced is stored. The consumer process has a local variable* ***nextConsumed*** *in which the item to be consumed is store.*
This scheme allows at most ***BUFFER_SIZE – 1*** *items in the buffer at the same time.*

2. Message-Passing Systems

- These allow processes to communicate and to synchronize their actions without sharing the same address-space.
- For example, a chat program used on the WWW.
- Messages can be of 2 types: 1) Fixed size or
  2) Variable size.
  1) If **fixed-sized messages** are used, the system-level implementation is simple.
     - However, the programming task becomes more difficult.
  2) If **variable-sized messages** are used, the system-level implementation is complex.
     - However, the programming task becomes simpler.
- A communication-link must exist between processes to communicate
- Three methods for implementing a link:
  1) Direct or indirect communication.
  2) Symmetric or asymmetric communication.
  3) Automatic or explicit buffering.
- Two operations:
  1) send(P,message): Send a message to process P.
  2) receive(Q,message): Receive a message from process Q.
- Advantages:
  1) Useful for exchanging smaller amounts of data („." No conflicts need be avoided).
  2) Easier to implement.
  3) Useful in a distributed environment.

4. Naming

- Processes that want to communicate must have a way to refer to each other. They can use either direct or indirect communication.

Direct Communication	Indirect Communication
Each process must explicitly name the recipient/sender.	Messages are sent to/received from mailboxes (or ports).
<u>Properties of a communication link:</u> ➢ 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. ➢ A link is associated with exactly two processes. ➢ Exactly one link exists between each pair of processes.	<u>Properties of a communication link:</u> ➢ A link is established between a pair of processes only if both members have a shared mailbox. ➢ A link may be associated with more than two processes. ➢ A number of different links may exist between each pair of communicating processes.
<u>Symmetric addressing:</u> ➢ Both sender and receiver processes must name the other to communicate. ➢ Primitives: **send** (*P, message*) – send a message to process P **receive**(*Q, message*) – receive a message from process Q	<u>Mailbox owned by a process:</u> ➢ The owner can only receive, and the user can only send. ➢ The mailbox disappears when its owner Process terminates.
<u>Asymmetric addressing:</u> ➢ Only the sender names the recipient; the recipient needn't name the sender. ➢ Primitives: **send** (*P, message*) – send a message to process P **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.	<u>Mailbox owned by the OS:</u> ➢ The OS allows a process to: 1. Create a new mailbox 2. Send & receive messages via it 3. Delete a mailbox. ➢ Primitives: **send(*A, message*) –** send a message to mailbox A **receive(*A, message*) –** receive a message from mailbox A

5. *Synchronization*

- Message passing may be either blocking or non-blocking (also known as synchronous and asynchronous).

Synchronous Message Passing	Asynchronous Message Passing
<u>Blocking send:</u> ➢The sending process is blocked until the message is received by the receiving process or by the mailbox.	<u>Non-blocking send:</u> ➢ The sending process sends the message and resumes operation.
<u>Blocking receive:</u> ➢The receiver blocks until a message is available.	<u>Non-blocking receive:</u> ➢ The receiver retrieves either a valid message or a null.

6. *Buffering*

- Messages exchanged by processes reside in a temporary queue.
- Three ways to implement a queue:

1) *Zero Capacity*

- The queue-length is zero.
- The link can't have any messages waiting in it.
- The sender must block until the recipient receives the message.

2) *Bounded Capacity*

- The queue-length is finite.
- If the queue is not full, the new message is placed in the queue.
- The link capacity is finite.

- If the link is full, the sender must block until space is available in the queue.

3) Unbounded Capacity

- The queue-length is potentially infinite.
- Any number of messages can wait in the queue.
- The sender never blocks.

Question5: Illustrate with a neat sketch, the process states and process control block.

Solution:

Process State

A Process has 5 states. Each process may be in one of the following states –

1. **New** - The process is in the stage of being created.
2. **Ready** - The process has all the resources it needs to run. It is waiting to be assigned to the processor.
3. **Running** – Instructions are being executed.
4. **Waiting** - The process is waiting for some event to occur. For example the process may be waiting for keyboard input, disk access request, inter-process messages, a timer to go off, or a child process to finish.
5. **Terminated -** The process has completed its execution.

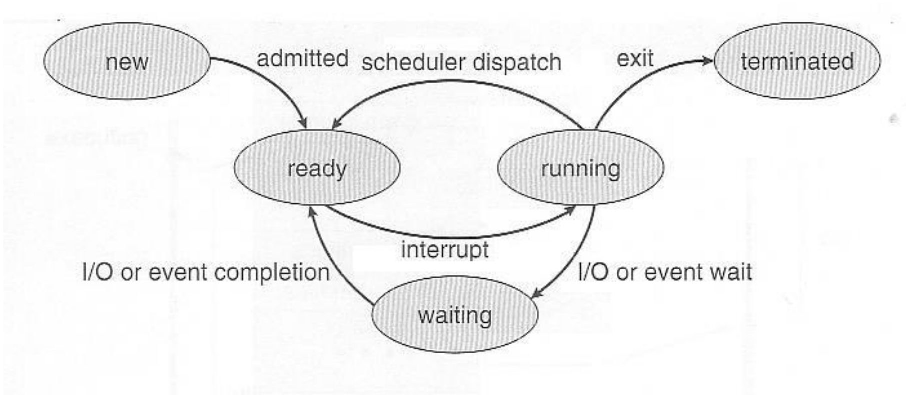


Figure 3.2 Diagram of process state.

Process Control Block

For each process there is a Process Control Block (PCB), which stores the process-specific information as shown below –

Process State – The state of the process may be new, ready, running, waiting, and so on.

Program counter – The counter indicates the address of the next instruction to be executed for this process.

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, this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward.

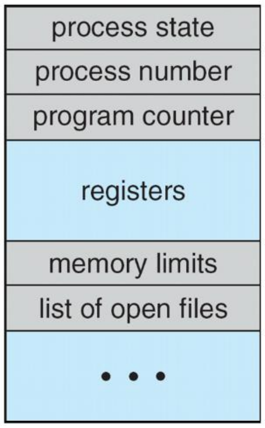
CPU scheduling information- This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.

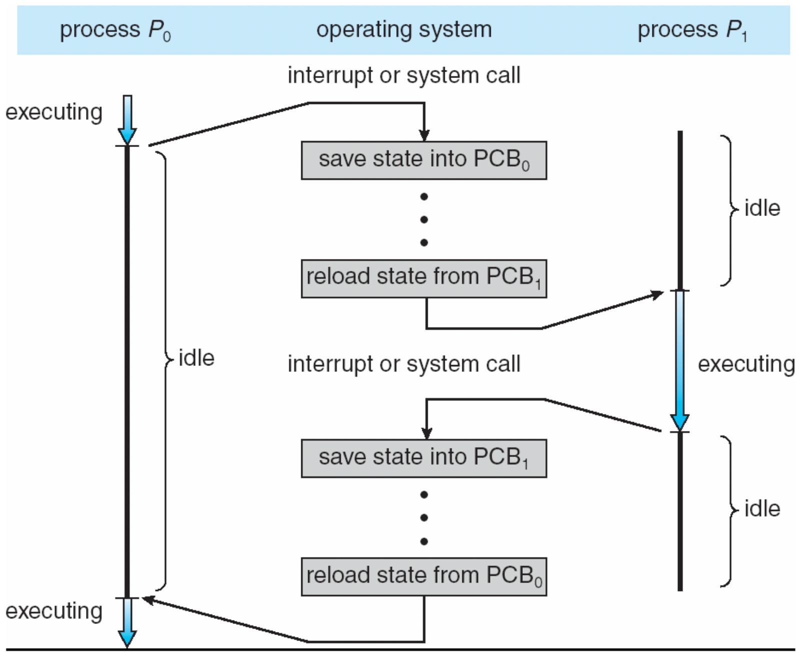
Memory-management information – This include information such as the value of the base and limit registers, the page tables, or the segment tables.

Accounting information – This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.

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.

The PCB simply serves as the repository for any information that may vary from process to process.





Question6: With a neat diagram of VM-Ware architecture, explain the concept of VM and the main advantage of using VM architecture

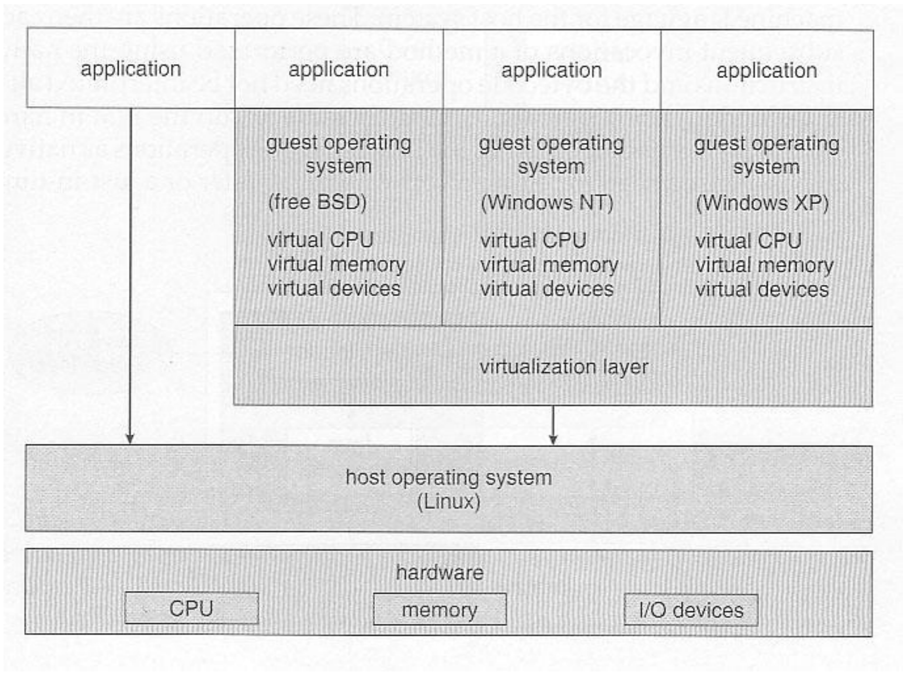
Solution:

VMware

VMware is a popular commercial application that abstracts Intel 80X86 hardware into isolated virtual machines. The virtualization tool runs in the user-layer on top of the host OS. The virtual machines running in this tool believe they are running on bare hardware, but the fact is that it is running inside a user-level application.

VMware 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.

In below scenario, Linux is running as the host operating system; FreeBSD, Windows NT, and Windows XP are running as guest operating systems. The virtualization layer is the heart of VMware, as it abstracts the physical hardware into isolated virtual machines running as guest operating systems. Each virtual machine has its own virtual CPU, memory, disk drives, network interfaces, and so forth.



VMware architecture

Question7: What are system calls? Briefly point out its types.
Solution:

System calls is a mean to access the services of the operating system. Generally written in C or C++, although some are written in assembly for optimal performance.
Types of System Calls: The system calls can be categorized into six major categories:

- Process Control
- File management
- Device management
- Information management
- Communications
- Protection

- Process control
  - end, abort
  - load, execute
  - create process, terminate process
  - get process attributes, set process attributes
  - wait for time
  - wait event, signal event
  - allocate and free memory
- File management
  - create file, delete file
  - open, close
  - read, write, reposition
  - get file attributes, set file attributes
- Device management
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices
- Information maintenance
  - get time or date, set time or date
  - get system data, set system data
  - get process, file, or device attributes
  - set process, file, or device attributes
- Communications
  - create, delete communication connection
  - send, receive messages
  - transfer status information
  - attach or detach remote devices

EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

	Windows	Unix

Process Control	`CreateProcess()`
	`ExitProcess()`
	`WaitForSingleObject()`
		`fork()`
		`exit()`
		`wait()`

File Manipulation	`CreateFile()`
	`ReadFile()`
	`WriteFile()`
	`CloseHandle()`
		`open()`
		`read()`
		`write()`
		`close()`

Device Manipulation	`SetConsoleMode()`
	`ReadConsole()`
	WriteConsole()
		`ioctl()`
		`read()`
		write()

Information Maintenance	`GetCurrentProcessID()`
	`SetTimer()`
	`Sleep()`
		`getpid()`
		`alarm()`
		`sleep()`

Communication	`CreatePipe()`
	`CreateFileMapping()`
	`MapViewOfFile()`
		`pipe()`
		`shmget()`
		`mmap()`

Protection	`SetFileSecurity()`
	`InitlializeSecurityDescriptor()`
	`SetSecurityDescriptorGroup()`
		`chmod()`
		`umask()`
		`chown()`