## 2.1 PROCESS CONCEPTS

Processes **& Programs**:

- Process is a dynamic entity. A process is a sequence of instruction execution process exists in a limited span of time. Two or more process may execute the same program by using its own data & resources.
- A program is a static entity which is made up of program statement. Program contains the instruction. A program exists in a single space. A program does not execute by itself.
- A process generally consists of a process stack which consists of temporary data & data section which consists of global variables.
- It also contains program counter which represents the current activities.
- A process is more than the program code which is also called text section.

**Process State**:

The process state consist of everything necessary to resume the process execution if it is somehow put aside temporarily. The process state consists of at least following:

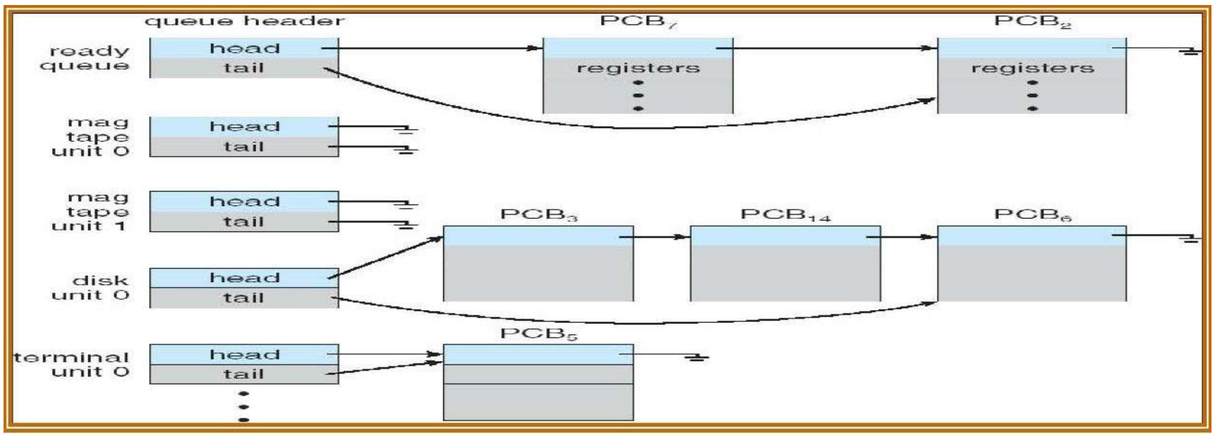
x Code for the program. x Program's
static data. x Program's dynamic data.
x Program's procedure call stack. x
Contents of general purpose registers.
x Contents of program counter (PC) x
Contents of program status word (PSW).
x Operating Systems resource in use.

## 2.2 PROCESS SCHEDULING

PROCESS SCHEDULING QUEUES

The following are the different types of process scheduling queues.

1. Job queue – set of all processes in the system
2. Ready queue – set of all processes residing in main memory, ready and waiting to execute
3. Device queues – set of processes waiting for an I/O device
4. Processes migrate among the various queues



**Ready Queue And Various I/O Device Queues**

**Ready Queue**:

The process that are placed in main m/y and are already and waiting to executes are placed in a list called the ready queue. This is in the form of linked list. Ready queue header contains pointer to the first & final PCB in the list. Each PCB contains a pointer field that points next PCB in ready queue.

**Device Queue**:The list of processes waiting for a particular I/O device is called device. When the CPU is allocated to a process it may execute for some time & may quit or interrupted or wait for the occurrence of a particular event like completion of an I/O request but the I/O may be busy with some other processes. In this case the process must wait for I/O. This will be placed in device queue. Each device will have its own queue.

The process scheduling is represented using a queuing diagram. Queues are represented by the rectangular box & resources they need are represented by circles. It contains two queues ready queue & device queues. Once the process is assigned to CPU and is executing the following events can occur,

1.20 It can execute an I/O request and is placed in I/O queue.

1.21 The process can create a sub process & wait for its termination.

1.22 The process may be removed from the CPU as a result of interrupt and can be put back into ready queue.

## Schedulers:

The following are the different type of schedulers

1. **Long-term scheduler (or job scheduler) –** selects which processes should be brought into the ready queue.
2. **Short-term scheduler (or CPU scheduler) –** selects which process should be executed next and allocates CPU.
3. **Medium-term schedulers**

**->** Short-term scheduler is invoked very frequently (milliseconds) . (must be fast)

-> Long-term scheduler is invoked very infrequently (seconds, minutes) (may be slow)

-> The long-term scheduler controls the *degree of multiprogramming* -

>Processes can be described as either:

x I/O-bound process – spends more time doing I/O than computations, many short CPU bursts

**CPU-bound process – spends more time doing computations; few very long CPU bursts**

## 2.3 OPERATION ON PROCESS

### Process Creation

In general-purpose systems, some way is needed to create processes as needed during operation. There are four principal events led to processes creation.

x System initialization. x Execution of a process Creation System
calls by a running process. x A user request to create a new process.
x Initialization of a batch job.

Foreground processes interact with users. Background processes that stay in background sleeping but suddenly springing to life to handle activity such as email, webpage, printing, and so on. Background processes are called daemons. This call creates an exact clone of the calling process. A process may create a new process by some create process such as 'fork'. It choose to does so, creating process is called parent process and the created one is called the child processes. Only one parent is needed to create a child process. Note that unlike plants and animals that use sexual representation, a process has only one parent. This creation of process (processes) yields a hierarchical structure of processes like one in the figure. Notice that each child has only one parent but each parent may have many children. After the fork, the two processes, the parent and the child, have the same memory image, the same environment strings and the same open files. After a process is created, both the parent and child have their own distinct address space. If either process changes a word in its address space, the change is not visible to the other process.

Following are some reasons for creation of a process

x User logs on. x User starts a program. x Operating systems creates process to
provide service, e.g., to manage printer. x Some program starts another process,
e.g., Netscape calls *xv* to display a picture.

### Process Termination

A process terminates when it finishes executing its last statement. Its resources are returned to the system, it is purged from any system lists or tables, and its process control block (PCB) is erased i.e., the PCB's memory space is returned to a free memory pool. The new process terminates the existing process, usually due to following reasons:

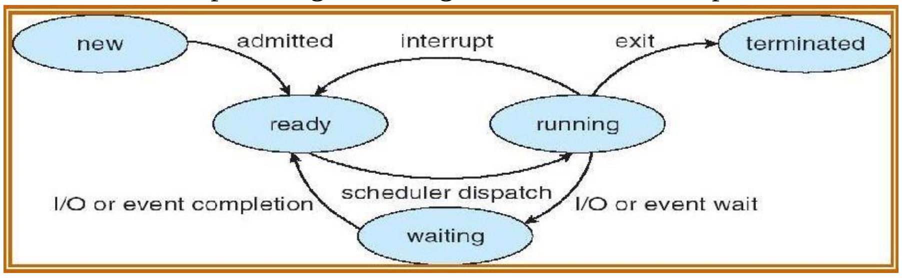
x **Normal Exist** Most processes terminates because they have done their job. This call is exist in UNIX. x

**Error Exist** When process discovers a fatal error. For example, a user tries to compile a program that does not exist.

**Fatal Error** An error caused by process due to a bug in program for example, executing an illegal instruction, referring non-existing memory or dividing by zero. x

**Killed by another Process** A process executes a system call telling the Operating Systems to terminate some other process. In UNIX, this call is kill. In x some systems when a process kills all processes it created are killed as well (UNIX does not work this way).

**Process States :**A process goes through a series of discrete process states.



x **New State** The process being created. X
**Terminated State** The process has finished execution. X
**Blocked**
**(waiting) State** When a process blocks, it does so because logically it cannot continue, typically because it is waiting for input that is not yet available. Formally, a process is said to be blocked if it is waiting for some event to happen (such as an I/O completion) before it can proceed. In this state a process is unable to run until some external event happens.
x **Running State** A process is said t be running if it currently has the CPU, that is, actually using the CPU at that particular instant. x

**Ready State** A process is said to be ready if it use a CPU if one were available. It is runable but temporarily stopped to let another process run.

Logically, the 'Running' and 'Ready' states are similar. In both cases the process is willing to run, only in the case of 'Ready' state, there is temporarily no CPU available for it. The 'Blocked' state is different from the 'Running' and 'Ready' states in that the process cannot run, even if the CPU is available.

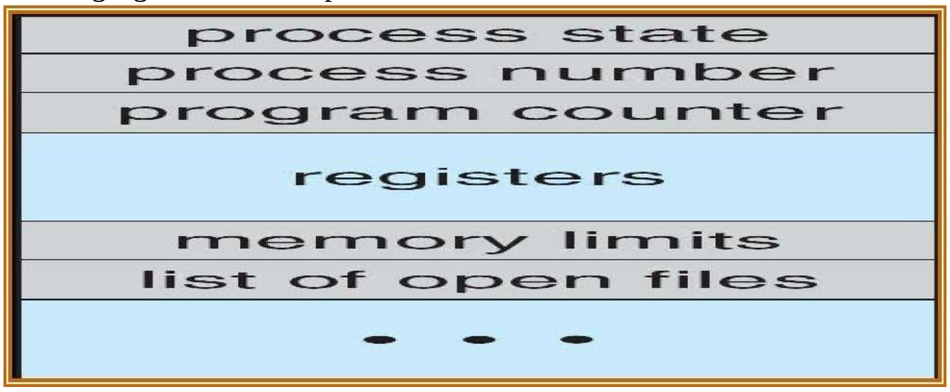
## Process Control Block

A process in an operating system is represented by a data structure known as a process control block (PCB) or process descriptor. The PCB contains important information about the specific process including x The current state of the process i.e., whether it is ready, running, waiting, or whatever.

x Unique identification of the process in order to track "which is which" information. x A pointer to parent process. x Similarly, a pointer to child process (if it exists). x The priority of process (a part of CPU scheduling information). x Pointers to locate memory of processes. x A register save area. x The processor it is running on.

The PCB is a certain store that allows the operating systems to locate key information about a process. Thus, the PCB is the data structure that defines a process to the operating systems.

The following figure shows the process control block.



**Context Switch**:

1. When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process.
2. Context-switch time is overhead; the system does no useful work while switching.
3. Time dependent on hardware support

**Cooperating Processes & Independent Processes**

**Independent process**: one that is independent of the rest of the universe.

x Its state is not shared in any way by any other process.
x Deterministic: input state alone determines results.
x Reproducible.
x Can stop and restart with no bad effects (only time varies). Example: program that sums the integers from 1 to i (input).

There are many different ways in which a collection of independent processes might be executed on a processor: programming: a single process is run to completion before anything else can be run on the processor.

Multiprogramming: share one processor among several processes. If no shared state, then order of dispatching is irrelevant.

multiprocessing: if multiprogramming works, then it should also be ok to run p processes in parallel on separate processors.

. given process runs on only one processor at a time.
A process ma y run on different processors at different times (move state, assume processors are identical).
.Cannot distinguish multiprocessing from multiprogramming on a very fine grain.

**Cooperating prmocesses:**

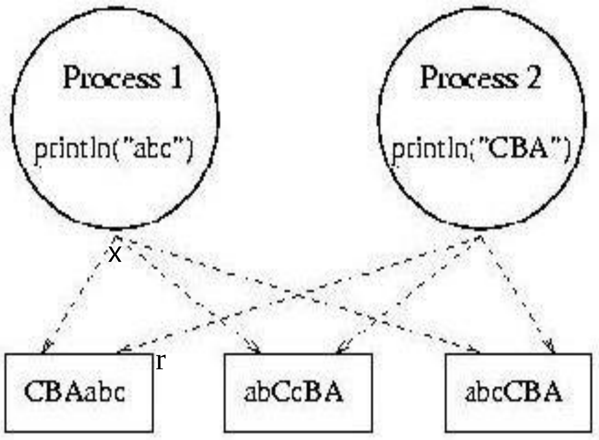
Machine must model the social structures of the people that use it. People cooperate, so machine must support: that cooperation. Cooperation means shared state, e.g. a single file system.

Cooperating processes are those that share state. (May or may not actually be "cooperating")

Behavior a is nondeterministic: depends on relative execution sequence and cannot be predicted priori.

Behavior is irreproducible. x Example: one process writes "ABC", another writes "CBA". Can get different outputs, cannot tell what comes from which. E.g. which process output first "C" in "ABCCBA"? Note the subtle state sharing that occurs here via the terminal. Not just anything can happen, though. For example, "AABBCC" cannot occur.

1. Independent process cannot affect or be affected by the execution of another process



2. Cooperatinp g process can affect or be affected by the execution of another process

3. Advantages of process cooperation

Information sharing, Computation speed-up , Modularity , Convenience

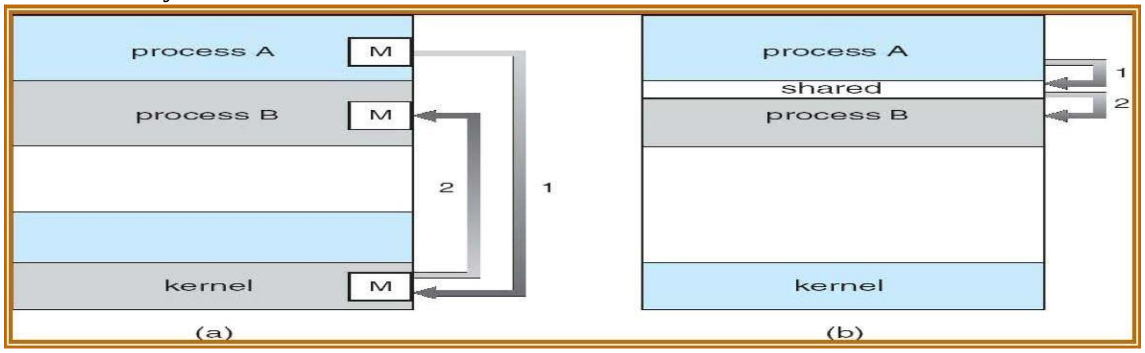
**2.4 INTERPROCESS COMMUNICATION (IPC)**
1. Mechanism for processes to communicate and to synchronize their actions.
2. Message system – processes communicate with each other without resorting to shared variables
3. IPC facility provides two operations:
send(*message*) – message size fixed or variable
receive(*message*)
4. If *P* and *Q* wish to communicate, they need to:exchange messages via send/receive 5. Implementation of communication link

physical (e.g., shared memory, hardware bus) logical (e.g., logical properties)

**Communications Models** there are two

types of communication models

1. Multi programming
2. Shared Memory



**Direct Communication**

1. Processes must name each other explicitly: x send (*P*, *message*) – send a message to process P x

receive(*Q*, *message*) – receive a message from process Q

2. Properties of communication link
x Links are established automatically x A link is associated with exactly one pair of communicating processes x Between each pair there exists exactly one link x The link may be unidirectional, but is usually bi-directional

**Indirect Communication**

1. Messages are directed and received from mailboxes (also referred to as ports)

   x Each mailbox has a unique id

2. Processes can communicate only if they share a mailbox

2. Properties of communication link x Link established only if processes share a common mailbox x A link may be associated with many processes

x Each pair of processes may share several communication links x Link may be unidirectional or bi-directional

3. Operations

. o create a new mailbox
. o send and receive messages through mailbox
. o destroy a mailbox

4.Primitives are defined as: send(*A, message*) – send a message to mailbox A receive(*A, message*) – receive a message from mailbox A

5.Mailbox sharing *P1, P2,* and *P3* share mailbox A *P1*, sends; *P2* and *P3* receive Who gets the message?

6. Solutions Allow a link to be associated with at most two processes

Allow only one process at a time to execute a receive operation Allow the system to select arbitrarily the receiver. Sender is notified who the receiver was.

**<u>Synchronization</u>**

1. Message passing may be either blocking or non-blocking
2. Blocking is considered synchronous

->Blocking send has the sender block until the message is received. ->Blocking receive has the receiver block until a message is available.

3. Non-blocking is considered asynchronous
->Non-blocking send has the sender send the message and continue.
**->**Non-blocking receive has the receiver receive a valid message or null.

**<u>Buffering</u>**

**->Queue of messages attached to the link; implemented in one of three ways**

1. Zero capacity – 0 messages sender must wait for receiver (rendezvous)
2. Bounded capacity – finite length of *n* messages Sender must wait if link full