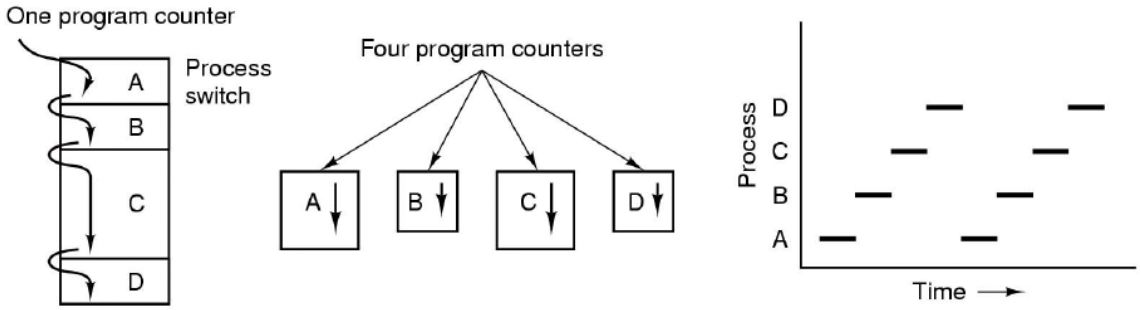
Unit 3 Process Management System.

Processes

1. A program is an algorithm expressed in some notation. A process is the activity of executing the program.
2. Early systems only allowed one program to be executed at a time. Later systems allowed multiprogramming, where many programs are loaded into memory at once and only one can be executed at any time. Programs are termed jobs or processes when executing. A process is sequential and can only do one thing at a time.
3. Multiple user processes run in separate (protected) address space, and can't directly access other memory spaces. Some user programs can share address space (e.g., an interpreter), but only the code (which is read only) is shared. The kernel can access all, however.
4. As the CPU only has one program counter, so the OS must use a system known as process switching.

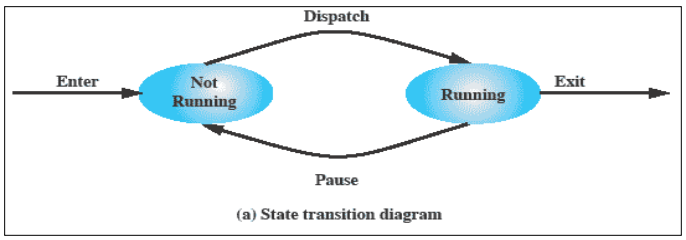


Process States:-

Process is a program in execution. Generally programs reside in secondary memory. When CPU is ready to execute then they are placed in main memory. If we compare process with program, process is a active entity whereas program is a passive entity. A program can have more number of processes. E.g. you can have a calculator program open twice; this is two processes but only one program.
We have two types of process state models:
Every process has different states.

A. Two state models:

This represented a simple model by observing that a process is either beginexecuted by a processor or not. It can be n two states running or not running. When the operating system creates a new process, it enters that into the system in the NOT RUNNING state as shown in fig.



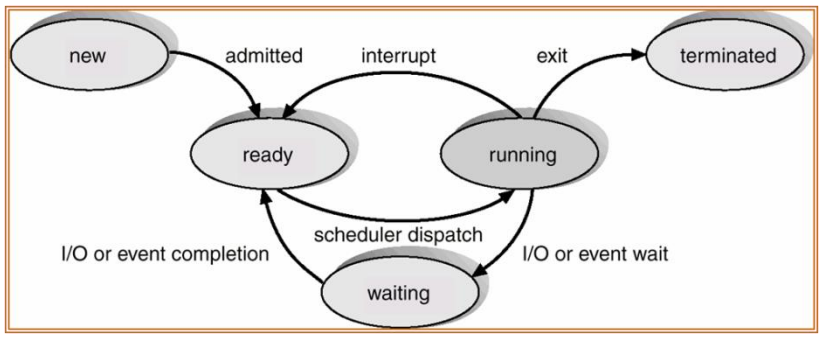
The process is waiting for an opportunity to operate, at times the currently running process will be interrupted and the dispatcher of the operating system will select a new process to run, one of the processes is now in the running state. Processes that are not running must be kept in some queue waiting their turn to be executed.

B. **Five State Processes:**

1. **New:** Process is created.
2. **Ready state**: Process is ready for the execution
3. **Running state:** Process is in execution
4. **Halt or termination**: Process completed execution

5. **Wait**: Process is in wait state(For i/o operation or because of time expire).
Below diagram is the process state diagram.

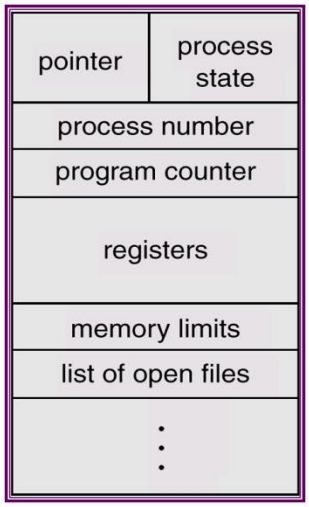
Some possible reasons for these transitions are:



1) **New to ready**:- The operating system creates an process, and prepares to the process to be executed, then the operating system moves the process into "Ready" queue.
2) **Ready to running**: A process is selected out of the ready queue (i.e. scheduled to be run) and is dispatched to the running state.
3) **Running to Waiting**: I/O operation is in fact the most common reason for this type of transition. Here, since a process is waiting for I/o completion and thus do not need CPU for executing any other instruction, CPU is freed by it and CPU in the meantime takes up some other ready process and starts its execution.
4) **Waiting to Ready**: The reason for a process to remain in waiting state is no longer applicable e.g. I/O completes, Timer goes off etc. Thus, the process becomes ready for its execution.
5) **Running to Ready**: The running process is pre-empted because some other process of higher priority has become ready. When the time slot of the processor expired or if processor received any interrupt signal, then the operating system shifted running process to ready state. For example, P1 is executed by processor; in the mean time process P2 generates an interrupts signal to the processor. Then, the processor compares the priorities of process P1 and P2, if P1> P2 then the processor continues the process P1, otherwise processor switched to process P2, and process P1 moved to the ready state.

Process Control Block: -(PCB)

1. To implement the process model, the operating system maintains a table, an array of structures, called the **process table** or **process control block (PCB)** or **Switch frame**. A process structure block is shown in Figure
2. Each entry identifies a process with information such as process state, its program counter, stack pointer, memory allocation, the status of its open files, its accounting and scheduling information.
3. In other words, it must contain everything about the process that must be saved when the process is switched from the running state to the ready state so that it can be restarted later as if it had never been stopped.
4. The following is the information stored in a PCB.



• **Process state**, which may be new, ready, running, waiting or halted;
• **Process number**, each process is identified by its process number, called process ID;
• **Program counter**, which indicates the address of the next instruction to be executed for this process;
• **CPU registers**, which vary in number and type, depending on the concrete microprocessor architecture;
• **Memory management information**, which include base and bounds registers or page table;
• **I/O status information**, composed I/O requests, I/O devices allocated to this process, a list of open files and so on;

• **Processor scheduling information**, which includes process priority, pointers to scheduling queues and any other scheduling parameters;
• **List of open files**.

Context Switch:-

1.A **context switch** (also sometimes referred to as a process switch or a task switch) is the switching of the CPU (central processing unit) from one process or to another. A *context* is the contents of a CPU's registers and program counter at any point in time.

2.Context Switch: Steps

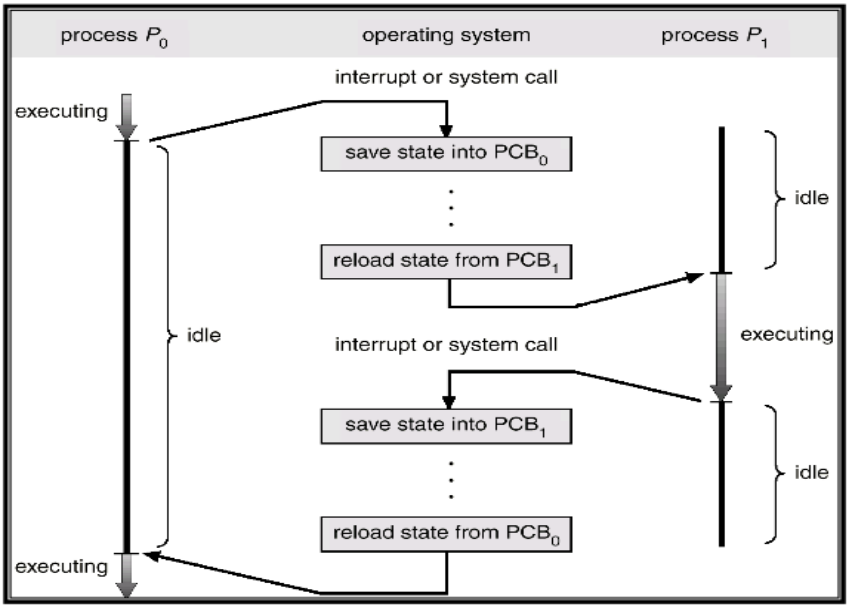
a. In a context switch, the state of the first process must be saved somehow, so that, when the scheduler gets back to the execution of the first process, it can restore this state and continue normally.

b. The state of the process includes all the registers that the process may be using, especially the program counter, plus any other operating system specific data that may be necessary.

c. Often, all the data that is necessary for state is stored in one data structure, called a process control block (PCB). Now, in order to switch processes, the PCB for the first process must be created and saved.

d. The PCBs are sometimes stored upon a per-process stack in the kernel memory, or there may be some specific operating system defined data structure for this information.

e. For example, suppose if two processes P0 and P1 are in ready queue. If CPU is executing Process P0 and Process P1 is in wait state. If an interrupt occurs for Process P0, the operating system suspends the execution of the first process, and stores the current information of Process P0 in its PCB and context to the second process namely Process P1. In doing so, the program counter from the PCB of Process P1 is loaded, and thus execution can continue with the new process. The switching between two processes, Process P0 and Process P1 is illustrated in the Figure given below:



Threads:-

1) A thread (or lightweight process (LWP)) is a basic unit of CPU utilization; it consists of:
   a) Program counter
   b) register set
   c) stack space
2) A thread shares with its peer threads its:
   a) code section
   b) data section

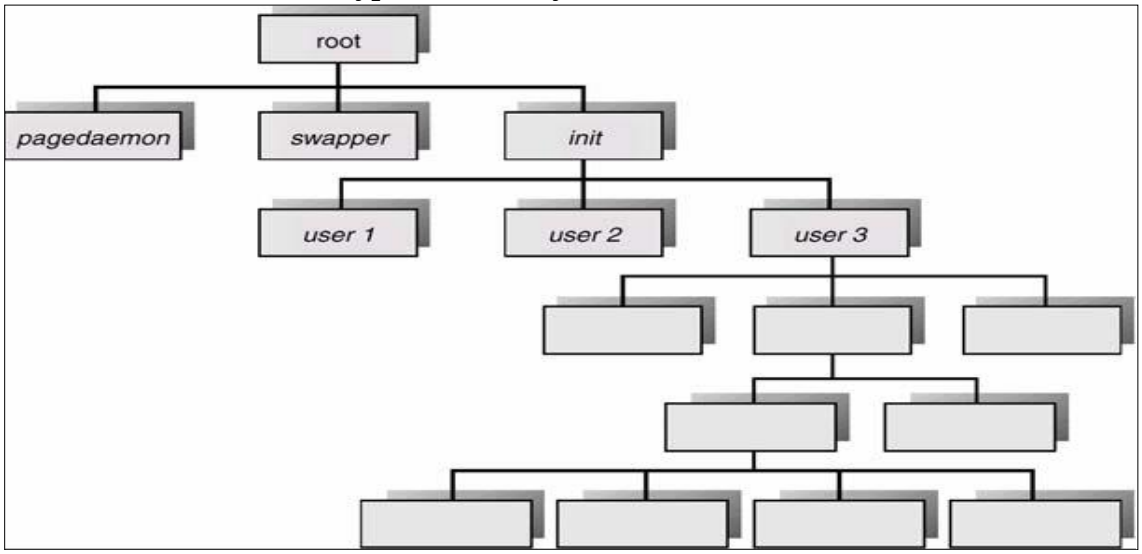
c) operating-system resources
Collectively known as a task.

3) A traditional or heavyweight process is equal to a task with one thread.
4) In a multiple threaded task, while one server thread is blocked and waiting, a second thread in the same task can run.
   a) Cooperation of multiple threads in same job confers higher throughput and improved performance.
   b) Applications that require sharing a common buffer benefit from thread utilization.
5) Threads provide a mechanism that allows sequential processes to make blocking system calls while also achieving parallelism.
6) Kernel-supported threads (Mach and OS/2).
7) User-level threads; supported above the kernel, via a set of library calls at the user level (Project Andrew from CMU).
8) Hybrid approach implements both user-level and kernel supported threads (Solaris 2).

Process Creation

1) A process may create several new processes, via a create-process system call, during execution.
2) Parent process creates children processes, which, in turn create other processes, forming a tree ofprocesses.
3) Resource sharing, such as CPU time, memory, files, I/O devices …
   a. Parent and children share all resources.
   b. Children share subset of parent's resources.
   c. Parent and child share no resources.
4) When a process creates a new process, two possibilities exist in terms of execution:
   a. Parent and children execute concurrently.
   b. Parent waits until children terminate.
5) There are also two possibilities in terms of the address space of the new process:
   a. Child duplicate of parent.
   b. Child has a program loaded into it.
6) UNIX examples
   a. fork system call creates new process
   b. exec system call used after a fork to replace the process' memory space with a new program.

A Tree of Processes ona Typical UNIX System



Process Termination

1) Process executes last statement and asks the operating system to delete it by using the exit system call.
    a. Output data from child to parent via wait system call.
    b. Process' resources are deallocated by operating system.
2) Parent may terminate execution of children processes via abort system call for a variety of reasons, such as:
    a. Child has exceeded allocated resources.
    b. Task assigned to child is no longer required.
    c. Parent is exiting, and the operating system does not allow a child to continue if its parent terminates.

2 marks

1. What is Context Switch?
2. Define Process and list different states of the Process.

4 marks

1. Explain the Creation and Termination of Processes
2. Explain Operation on Process
3. ***Explain Process Control Block (PCB) in detail with the help of diagram.
4. State and explain four reasons for Process Creation