(Operating System - 22516)

Unit III : Process Management.

(Weightage 14 marks)

— TRANJAL SAVE

Introduction - Process Management.

- The operating system manages many kinds of activities ranging from user ~~operating~~ programs to system programs.
- The major activities/functions of an operating system in regard to process management are :
  1) Creation and deletion of user and system processes.
  2) Suspension and resumption of processes.
  3) A mechanism for process synchronization, process communication and deadlock handling.

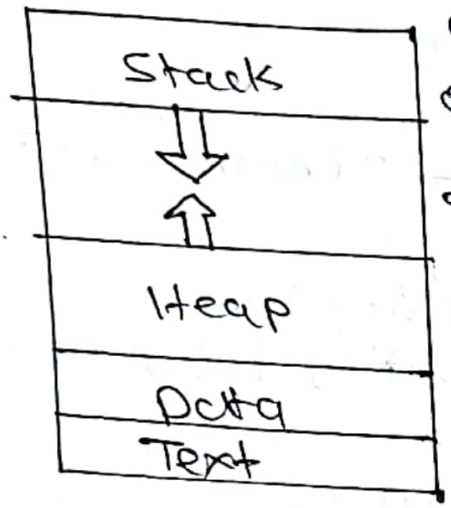
Program & Process

Program : A program is series or set of instructions to perform a particular task.

Process : A process is a program in execution which competes for CPU time & others resources.

Program	Process
~~Active~~ It is passive entity	It is active entity
Program is static	Process is dynamic
It does not have any control block	Process has own control block called PCB.
It contains instructions.	It is sequence of instruction execution
Time span is unlimited	Time span is limited
Program is loaded in secondary storage	It is loaded in to main memory

Process in Memory



When you run a program (like opening an app on your phone or computer) it become an process.
The Operating system (OS) takes the program from storage (like your hard drive) and load into memory (RAM).
In memory, the process is divided into different parts, each serving a specific purpose.

Text Segment: The text segment contains the program code or instructions that tells computer what to do.

Data segment: The data segment holds global variables and pre-set data that the program uses.

Heap: The heap is used for dynamic memory allocation, where new data is stored as needed while program runs.

Stack: The stack is used for managing function calls and local variables. It's a temporary work's space that grows and shrinks as program runs.

Example video from VLC player.

- Holds code that tells the player how to decode and play video. "Blueprint"
- Store Default settings (volume, screen) used during playback
- Dynamically store video frames and audio samples as movie plays
- Holds temporarily data during calls (player)

Process + Process States, Process Control Block (PCB)

Process state

Process is program in execution.

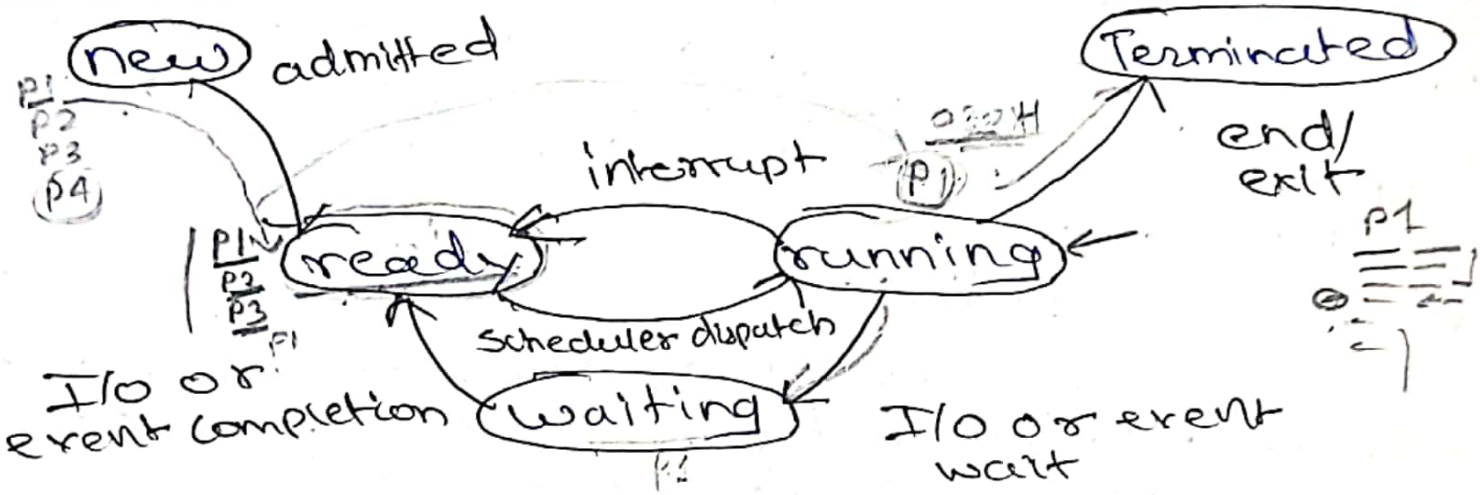
Process from creation to end or execute the process goes through five states that is Process State.

As process executes, it changes state.

Different process state are as follows.

1) New 2) Ready 3) Running 4) Waiting 5) Terminated.

Diagram



New state: When a process enters into the system it is in new state.
In this process state, process is created.
In new state process in job pool

Ready state: When the process is loaded into the main memory, it is ready for execution.
In this state the process is waiting for processor allocation.

Running: When CPU is available, System selects one process from main memory and executes all the instructions from that process.
So, when process is in execution, it is in running state.
Single user system: 1 process in running state.
multi user system: multiple process in running state.

Waiting State : When process is in execution, may request for I/O resources. If resources is not available, process goes in waiting state. When resources are available, then process goes back to ready state.

Terminated State : When the process completes its execution, it goes into the terminated State.
In this state memory occupied by process is released.

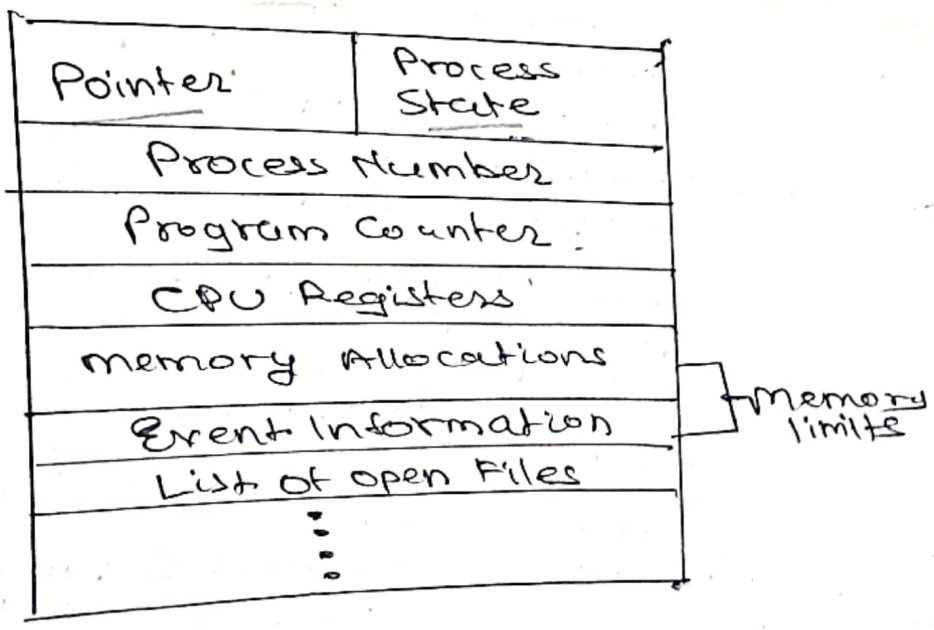
Process Control Block (PCB)

Process is defined as program in execution, which competes for CPU time and other resources. Process is also called as job, task or unit of work.

Process Control Block

Process control Block is data structure that contains information of process related to it. The process control block is also known as Task control Block.

Diagram



Pointer: Pointer points to another process control Block. Pointer is used for maintaining scheduler list.

2) Process state: This information is about the current state of the process.
The state may be new, ready, running, and waiting, halted and so on.

3) Process Number: Each process is identified by its process number called Process Identification Number (PID).
Every process has unique process-id through which it is identified.
It is provided by OS. Process id of two process cannot be same as its unique.

4) Program Counter: The counter indicates the address of next instruction to be executed for this process.

5) CPU Registers: Various CPU registers are there where process need to be stored for execution for running state. include, accumulators, index registers, stack pointers and General purpose registers.

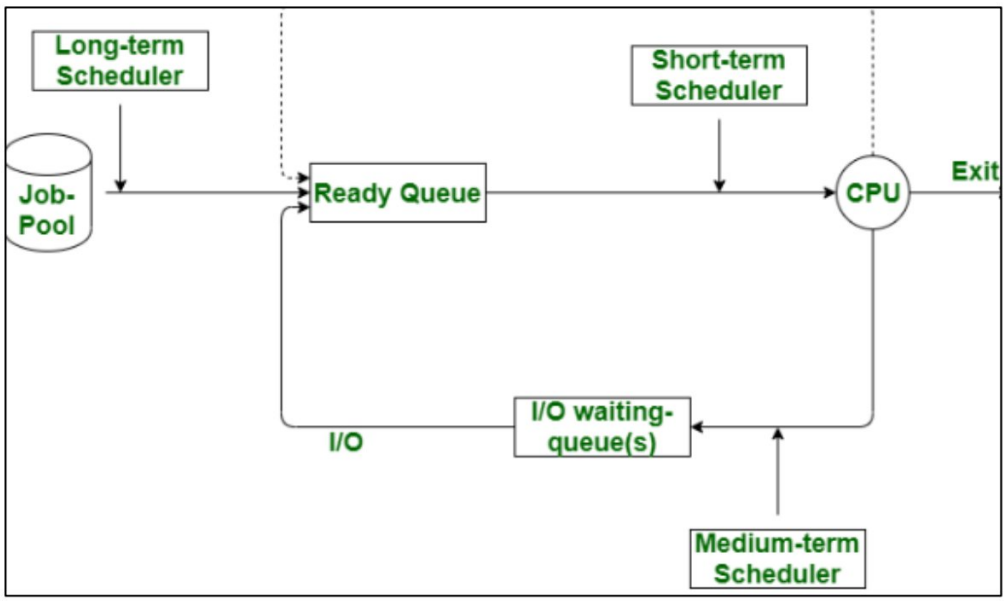
6) Process Privileges: This is required to allow/ disallow access to system resources.

7) CPU scheduling Information: Process priority and other scheduling information which is required to schedule the process

8) Memory management Information: This includes the information of page table, memory limits, Segment table depending on memory used by OS.

9) Accounting information: This includes the amount of CPU used for process executed, time limits etc---

10) I/O status Info: This includes a list of I/O device allocated to process.



Long-term scheduler:

- It picks programs from a big pool (job pool) and puts them into the main memory.
- It decides how many programs can run at once (this is called the degree of multiprogramming).
- A program can be:
- I/O-bound (spends more time doing input/output tasks).
- CPU-bound (spends more time doing calculations).
- The scheduler tries to balance both types to keep the system running smoothly.
- It runs rarely, only when a program finishes and leaves the system.
- When it picks a program, the program's state changes from new to ready.

Short-term scheduler:

- Also called the CPU scheduler.
- It picks a program from the ready queue (programs waiting to run) and gives it to the CPU.
- This scheduler works often because the CPU needs tasks quickly.
- When it picks a program, the program's state changes from ready to running.

Medium-term scheduler:

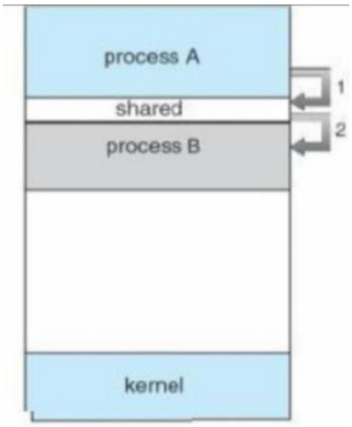
- If a program is running but gets blocked (e.g., waiting for input), it is swapped out (moved to storage).
- When there's space in the main memory, this scheduler picks a program from storage and puts it back in the ready queue.
- It works closely with the long-term scheduler to decide what to load into the memory.

Inter-process communication: Cooperating processes require an Inter- process communication (IPC) mechanism that will allow them to exchange data and information

Inter-process communication (IPC) allows processes to exchange data and information.

. There are two models of IPC

1. **Shared Memory**



A specific memory area is shared by processes that want to communicate.

Processes can read and write data in this shared memory.

Processes must:

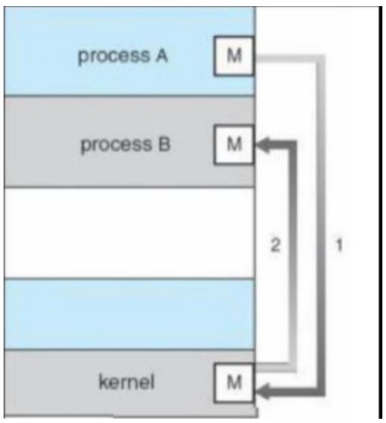
Attach to the shared memory.

Decide how to organize the data and avoid writing to the same spot at the same time.

The operating system does not control how they use the shared memory.

Once set up, processes access the memory like normal, without help from the OS.

2. **Message Passing**



Processes exchange messages to communicate.

Useful when processes are on different computers in a network.

Processes do not share memory; they use the kernel to send and receive messages.

There must be a communication link between the processes, with one link for each pair of communicating processes.

In short:

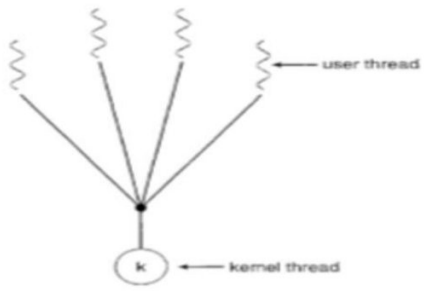
Shared Memory: Fast but needs careful coordination between processes.

Message Passing: Safer and works well over networks, but involves more overhead

Multithreading model

Many systems provide support for both user and kernel threads, resulting in different multithreading models. Following are three multithreading model:

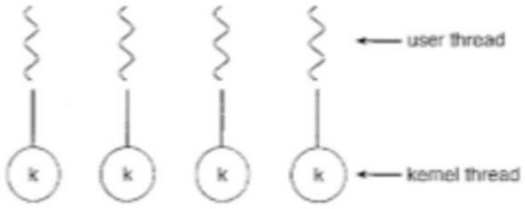
1. Many-to-One Model

- **Description**:
  Maps many user threads to one kernel thread.
  Thread management is done in user space, making it efficient.
-
- 

- **Key Issues**:
  - **Blocking**: If one thread makes a blocking system call, the entire process blocks.
  - **No Parallelism**: Only one thread can access the kernel at a time, so it cannot utilize multiple processors.
- **Examples**:
  Green threads in Solaris.
- **Advantages**:
  - Easy to implement and efficient in single-core systems.
  - Low overhead since only one kernel thread is needed.
- **Disadvantages**:
  - Blocking affects all threads.
  - No true parallelism on multiprocessor systems.

2. One-to-One Model

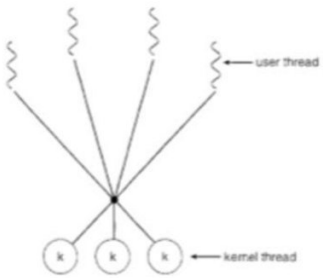
- **Description**:
  Maps each user thread to a single kernel thread.
  Allows multiple threads to run in parallel on multiprocessors.



- **Key Issues**:
  - **Overhead**: Creating a user thread requires creating a corresponding kernel thread, which can burden the system.
  - **Thread Limit**: Most systems impose limits to prevent performance issues.
- **Examples**:
  Linux and Windows operating systems.
- **Advantages**:
  - Better concurrency; other threads can run when one blocks.
  - Utilizes multiple CPUs effectively.
- **Disadvantages**:
  - High overhead due to kernel thread creation.
  - Limited number of threads can be supported.

3. Many-to-Many Model

- **Description**:
  Maps many user threads to a smaller or equal number of kernel threads.
  Developers can create many user threads, while kernel threads efficiently manage execution.



- **Key Benefits**:
    - Allows parallel execution on multiprocessor systems.
    - If a thread blocks, the kernel schedules another thread to run.
- **Advantages**:
    - Combines benefits of both Many-to-One and One-to-One models.
    - Supports true concurrency on multiprocessor systems.
    - No limit on the number of user threads.
- **Disadvantages**:
    - Overhead of managing kernel threads.
    - Complex implementation.

User Thread & Kernal Thread

User-Level Threads

Managed by the application, not the operating system (Kernel).

The thread library handles creating, destroying, and managing threads.

The operating system is unaware of user-level threads.

Advantages:

- Fast to create and manage threads.
- Does not require Kernel privileges for switching threads.
- Can run on any operating system.
- Allows application-specific scheduling of threads.

Disadvantages:

- Limited Parallelism: Can't fully use multi-core processors.
- Blocking Issues: If one thread blocks (e.g., during I/O), all threads in the process are blocked.
- No True Parallelism: All user threads run on a single kernel thread.
- Not Suitable for Multiprocessor Systems: Can't use multiple CPUs effectively.

Kernel-Level Threads

Managed by the Kernel (operating system).

Kernel handles thread creation, scheduling, and management.

Threads are treated as independent entities by the operating system.

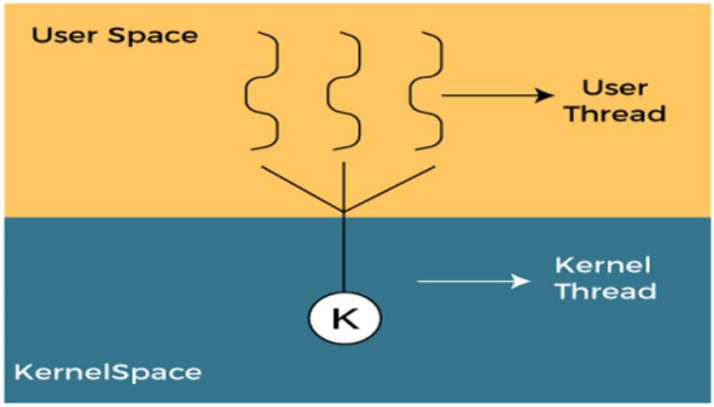
Advantages:

- True parallel execution on multi-core systems.

- If one thread blocks, the Kernel schedules another thread from the same process.
- Kernel routines can be multithreaded.

Disadvantages:

- Slower to create and manage threads due to Kernel involvement.
- Switching threads requires a mode switch to the Kernel, which adds overhead.
-



-

bject: Operating Systems **Subject Code:** 22516

	Ans.	**i)kill** **Syntax: kill pid** Kill command is used to stop execution of particular process by sending an interrupt signal to the process **ii)Sleep** **Syntax: sleep NUMBER[SUFFIX]** The sleep command pauses the execution for specified time in command. **iii)Wait** **Syntax: wait [pid]** Wait command waits for running process to complete and return the exit status. **iv) Exit** **Syntax: exit** used to quit the shell **(OR)** **Syntax: exit[n]** The terminal window will close and return a status of n	***Each command Syntax ½M*** ***Explanation ½ M***

ɔ)

ns.

Differentiate between process and thread (any two Points). Also discuss the benefits of multithreaded programming.

	Process	Thread
Definition	A process is a program under execution i.e. an active program.	A thread is a lightweight process that can be managed independently by a scheduler.
Running mechanism	Processes run in separate memory spaces.	Threads within the same process run in a shared memory space.
Concept	Process is heavy weight and any program is in execution.	It is a lightweight process and a segment of a process.
PCB	The process has its own Process Control Block, Stack, and Address Space.	Thread has Parents' PCB, its own Thread Control Block, and Stack and common Address space.
Context Switching	Context switching between the process is more expensive	Context switching between threads of the same process is less expensive
Dependency	Processes are independent.	Threads are dependent
Controlling	Process is controlled by the operating system.	Threads are controlled by programmer in a program

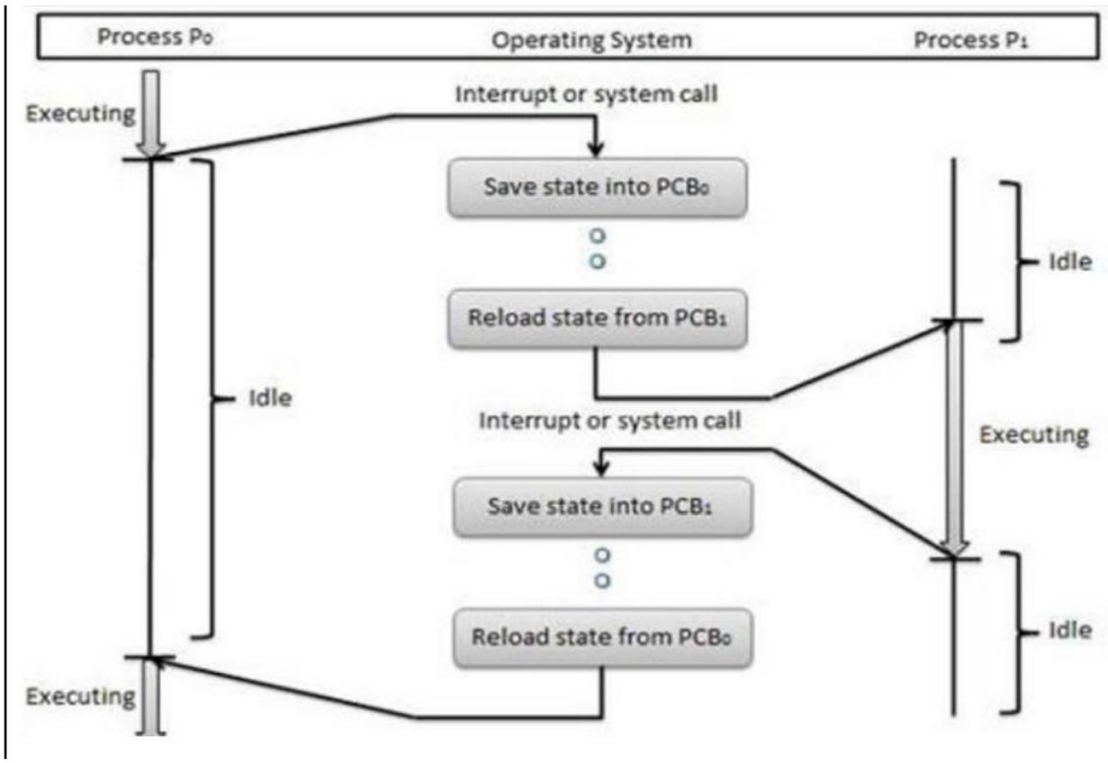
Benefits of Multithreading

1. **Improved Responsiveness**:
    - Allows an application to remain responsive even when performing long tasks (e.g., user interface remains active during background operations).
2. **Better Resource Utilization**:
    - Threads can share resources such as memory and data, reducing overhead compared to creating multiple processes.

3. **Faster Execution**:
    - Tasks can be divided among multiple threads and executed in parallel, reducing overall execution time, especially on multicore systems.
4. **Scalability**:
    - Multithreading takes advantage of multiprocessor and multicore systems by running threads simultaneously across different processors.
5. **Cost-Effective**:
    - Creating and managing threads is faster and requires fewer resources than creating processes.
6. **Enhanced Performance for I/O Operations**:
    - One thread can handle I/O operations while others continue processing, leading to better efficiency.
7. **Simplifies Complex Applications**:
    - Threads can be used to separate different functionalities of an application (e.g., one thread for input, another for computation, and another for output).
8. **Support for Asynchronous Processing**:
    - Multithreading allows programs to perform non-blocking operations, enabling tasks to proceed concurrently without waiting for others to complete.

Context Switching

c)	**Explain working of CPU switch from process to process with neat labelled diagram.**	**4M**
Ans.	A CPU switch from process to process is referred as **context switch**. A context switch is a mechanism that store and restore the state or context of a CPU in Process Control block sothat a process execution can be resumed from the same point at a later time. When thescheduler switches the CPU from one process to another process, the context switch savesthe contents of all process registers for the process being removed from the CPU, in its process control block. Context switch includes two operations such as state save and state restore. State save operation stores the current information of running process into its PCB. State restoreoperation restores the information of process to be executed from its PCB. Switching the CPU from one process to another process requires performing state save operation for thecurrently executing process (blocked) and a state restore operation for the process ready for execution. This task is known as context switch.	***Explanation 2M*** ***Diagram 2M*** ***Relevant Explanation shall be considered***



Context Switch: Overview

A **context switch** occurs when the CPU stops running one process and starts running another. This enables multitasking by allowing multiple processes to share the CPU.

What is a Context?

The **context** of a process refers to all the information the CPU needs to execute it. This includes:

- **Register values** (e.g., program counter, general-purpose registers).
- **Process state** (e.g., running, ready, or waiting).
- Other critical details stored in the **Process Control Block (PCB)**.

Steps in a Context Switch

1. **State Save**:
    - The CPU saves the current process's state (register values, program counter, etc.) into its PCB.
    - This ensures that the process can resume later without losing progress.
2. **State Restore**:

- The CPU loads the state of the new process from its PCB.
- The new process resumes execution from where it last paused.

a	**Write Unix command for following:** **i)create a folder OSY ii) create a file FIRST in OSY folder** **iii) List/display all files and directories.** **iv) Write command to clear the screen**	**4M**
Ans	i) create a folder OSY: $mkdir OSY ii)create a file FIRST in OSY folder: $cd OSY $cat>FIRST or $ touch FIRST iii) List/display all files and directories: $ls iv) to clear screen: $clear	**Each correct command- 1M**

b)	**Writer the outputs of following commands** **(i) Wait 2385018** **(ii) Sleep 09** **(iii) PS –u Asha**	**6M**
Ans.	i) Wait command waits until the termination of specified process ID 2385018 ii) Sleep command is used to delay for 9 seconds during the execution of a processi.e. it will pause the terminal for 9 seconds. iii) ps command with -u is used to display data/processes for the specific user Asha.	***2M for each correct output***

f) Give commands to perform following tasks:

i) To add delay in script

ii) To terminate a process