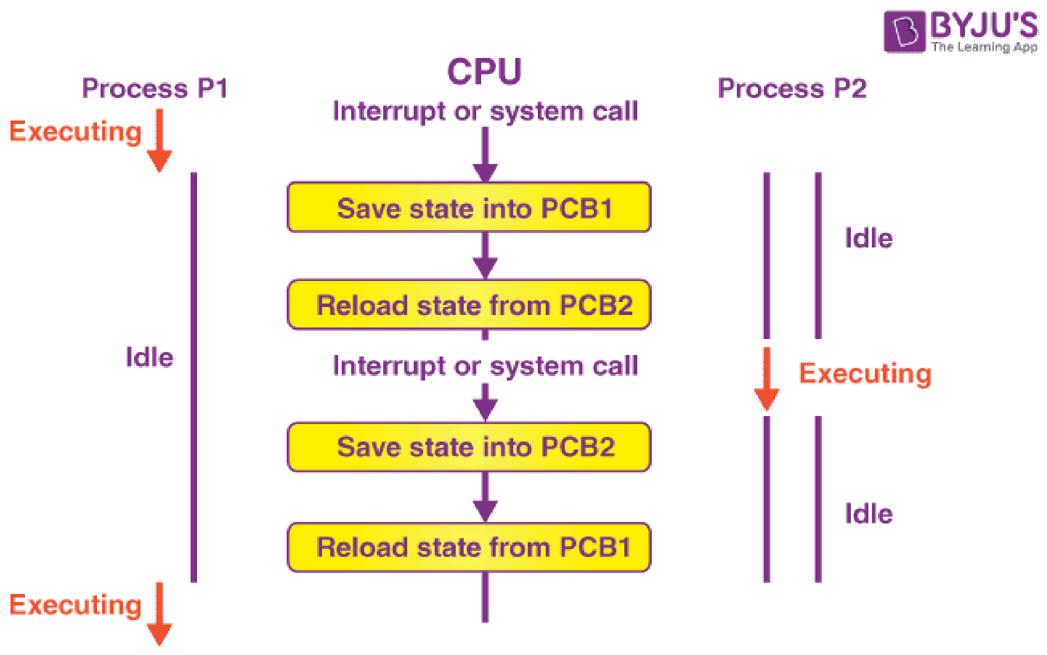
What is Context Switching in OS?

- Context switching refers to a technique/method used by the OS to switch processes from a given state to another one for the execution of its function using the CPUs present in the system. When switching is performed in the system, the old running process's status is stored as registers, and the CPU is assigned to a new process for the execution of its tasks. While new processes are running in a system, the previous ones must wait in the ready queue. The old process's execution begins at that particular point at which another process happened to stop it. It describes the features of a multitasking OS where multiple processes share a similar CPU to perform various tasks without the requirement for further processors in the given system.



Why Do We Need Context Switching?

- Context switching helps in sharing a single CPU among all processes so as to complete the execution and store the status of the system's tasks. Whenever the process reloads in a system, its execution begins at the very point in which there is conflict.

What is the PCB?

- PCB refers to a data structure that is used in the OS to store all the data-related info to the process. For instance, whenever a process is formed in the OS, the info of the process is updated, information about the process switching, and the process terminated in the PCB. The PCB contains,

    1. Process state
    2. Program counter
    3. CPU registers
    4. CPU scheduling information
    5. Memory-management information
    6. Accounting information
    7. I/O status information

What is a Process Scheduler in OS?

- Process Scheduling is responsible for selecting a processor process based on a scheduling method as well as removing a processor process. It's a crucial component of a multiprogramming operating system. Process scheduling makes use of a variety of scheduling queues (Job Queue, Ready Queue, Wait Queue)

- Processes in an operating system go through several states during their lifecycle, and the transition between these states often involves decisions made by different types of schedulers. Here are the different process states along with the types of schedulers involved:

1. New State:
- **Scheduler Involved**: Long-term Scheduler (or Job Scheduler)
- **Justification**:
    - The long-term scheduler decides which processes from the pool of new processes are to be admitted into the ready queue for execution.
    - It manages the degree of multiprogramming by controlling the number of processes in the ready queue, balancing system throughput and responsiveness.

2. Ready State:
- **Scheduler Involved**: Short-term Scheduler (or CPU Scheduler)
- **Justification**:
    - The short-term scheduler selects which process from the ready queue will execute next on the CPU.

    - It makes decisions frequently and quickly to ensure efficient CPU utilization and responsiveness to events.

3. Running State:

- **Scheduler Involved**: CPU Scheduler
- **Justification**:
    - While a process is running, the CPU scheduler may still be involved in decisions related to process preemption (e.g., when a higher priority process becomes ready).
    - It may also handle scheduling decisions when the running process voluntarily relinquishes the CPU (e.g., through blocking for I/O).

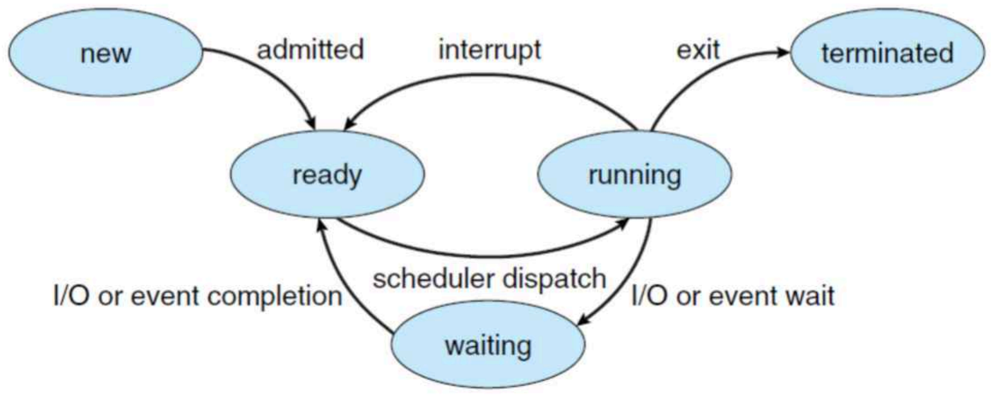
4. Wait State:

- **Scheduler Involved**: I/O Scheduler (in some systems)
- **Justification**:
    - When a process is blocked waiting for I/O (e.g., disk, network), an I/O scheduler manages the queue of pending I/O requests.
    - It decides the order in which pending I/O operations are dispatched to the appropriate I/O devices, optimizing device utilization and minimizing latency.

5. Terminate State:

- **No Scheduler Involved**:
    - Once a process completes its execution, no scheduler is actively involved.
    - The operating system handles final cleanup tasks (e.g., releasing resources, notifying parent process) before the process is completely removed from the system.

Process State Diagram



Types of Process Schedulers:

Process schedulers are divided into three categories,

1. **Long-Term Scheduler or Job Scheduler**: The job scheduler is another name for the Long-Term scheduler. It selects processes from the pool (or the secondary memory) and then maintains them in the primary memory's ready queue. The Multiprogramming degree is mostly controlled by the Long-Term Scheduler. The goal is to select the best mix of IO and CPU bound processes from the pool of jobs.
2. **Short-Term Scheduler or CPU Scheduler**: CPU scheduler is another name for Short-Term scheduler. It chooses one job from the ready queue and then sends it to the CPU for processing.
3. **Medium-Term Scheduler**: The switched-out processes are handled by the Medium-Term scheduler. If the running state processes require some IO time to complete, the state must be changed from running to waiting. It stops the process from executing in order to make space for other processes.

Summary:

- **Long-term Scheduler**: Decides which new processes to admit into the system, influencing the degree of multiprogramming.
- **Short-term Scheduler (CPU Scheduler)**: Selects which process from the ready queue will execute next on the CPU, optimizing CPU utilization.
- **I/O Scheduler**: Manages the queue of I/O requests for blocked processes, optimizing device utilization and reducing I/O latency.

The degree of multiprogramming describes the maximum number of processes that a single-processor system can accommodate efficiently[2]. The primary factor affecting the degree of multiprogramming is the amount of memory available to be allocated to executing processes.

Zombie Processes and their Solutions:

- **Zombie state:** When a process is created in UNIX using a *fork()* system call, the parent process is cloned. If the parent process calls the *wait()* system call, then the execution of the parent is suspended until the child is terminated.
- **Solution:** Using *wait()* system call. When the parent process calls *wait()*, after the creation of a child, it indicates that, it will wait for the child to complete and it will reap the exit status of the child. The parent process is suspended (waits

in a waiting queue) until the child is terminated. It must be understood that during this period, the parent process does nothing just wait.

CPU bound means the program is bottlenecked by the CPU, or central processing unit, while I/O bound means the program is bottlenecked by I/O, or input/output, such as reading or writing to disk, network, etc.
A CPU-bound process requires more CPU time or spends more time in the running state. An I/O-bound process requires more I/O time and less CPU time. An I/O-bound process spends more time in the waiting state.
If the job scheduler selects more IO bound processes, all of the jobs may become stuck, the CPU will be idle for the majority of the time, and multiprogramming will be reduced as a result. AND VICE VERSA.