**CPU SCHEDULING**

**BASIC CONCEPTS:**

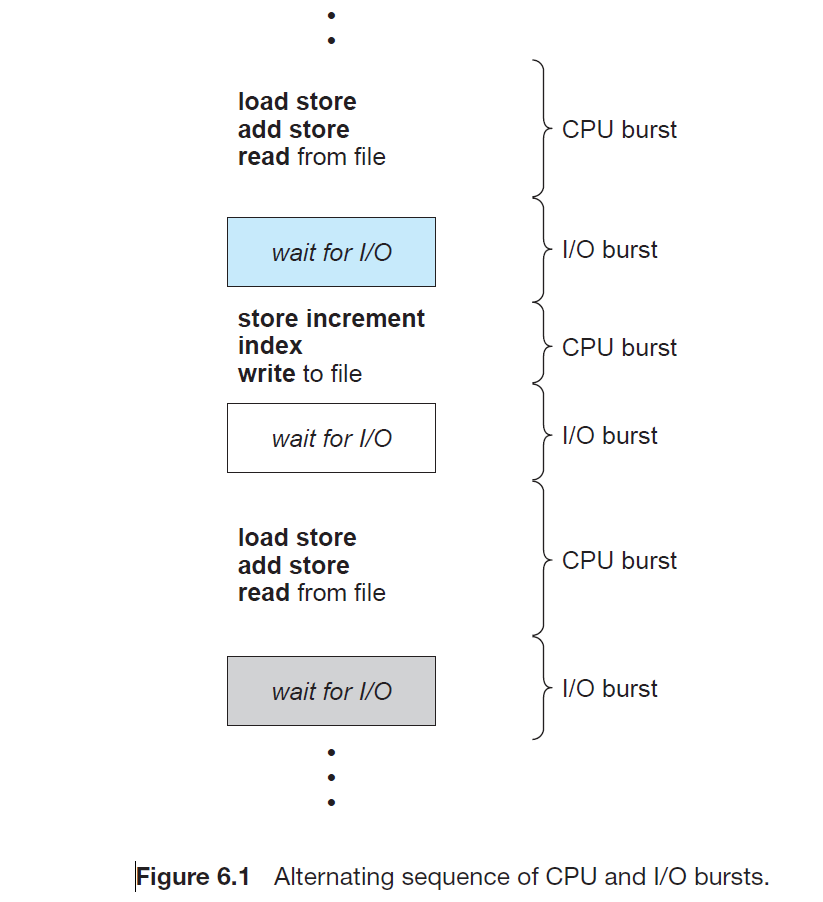
In case of a single processor system only a single process can run at a time. Other processes must wait until the CPU is free and can be rescheduled. This shows the motivation for CPU Scheduling. The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization. Several processes are kept in memory at the same time. When one process has to wait, the operating system takes the CPU away from that process and gives the CPU to another process. This pattern continues. Every time one process has to wait, another process can take over use of the CPU. Scheduling of this kind is a fundamental operating-system function.

**1)CPU — I/O Burst Cycle:**

**CPU Burst:** CPU Burst means interval of time a process is being executed on CPU.

**I/O Burst: I/O** Burst means interval of time a process spends doing a single I/O operation and CPU is waiting for I/O for further execution.

Process execution consists of a cycle of CPU execution and I/O wait. Processes alternate between these two states. Process execution begins with a CPU burst. That is followed by an I/O burst, which is followed by another CPU burst, then another I/O burst, and so on. Eventually, the final CPU burst ends with a system request to terminate execution.



**2)CPU Scheduler:**

Whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed. The selection process is carried out by the short-term scheduler, or CPU scheduler. The scheduler selects a process from the processes in memory that are ready to execute and allocates the CPU to that process. There are various scheduling algorithms which determine which process to select.

### **3)Preemptive Scheduling:**

CPU-scheduling decisions may take place under the following four circumstances:

1.When a process switches from the running state to the waiting state (for example, as the result of an I/O request or an invocation of wait() for the termination of a child process).

2. When a process switches from the running state to the ready state (for example, when an interrupt occurs).

3. When a process switches from the waiting state to the ready state (for

example, at completion of I/O).

4. When a process terminates.

Out of the above mentioned 4 cases, in case 1 and 4 there is no choice in terms of CPU scheduling. A new process (if one exists in the ready queue) must be selected for execution. Whereas in cases 2 and 3 there is a choice whether to remain idle while the process is in waiting state or to select another process from the ready queue.

When scheduling takes place only under circumstances 1 and 4, then that the scheduling scheme is nonpreemptive or cooperative. Otherwise, it is preemptive. Under **nonpreemptive scheduling**, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state.

Under **Preemptive Scheduling,** once the CPU has been allocated to a process, in middle of that process running the operating system can take out a running process from CPU forcefully due to some reasons.

 In this, the resources are allocated to execute the process for a certain period. After this, the process is taken away in the middle and is placed in the ready queue its bursts time is left and this process will stay in ready line until it gets its turn to execute. Suppose if a process which has the highest priority arrives, then this process does not wait for the complete execution of the current process. Instead of it, the ongoing process is interrupted in between and is placed in the ready queue until the process which has the highest priority does its execution. Thus in this way, all the processes which are in the available line get some time to run.

In preemptive scheduling there can be race conditions when data is shared among several processes. Consider the case of two processes that share data. While one process is updating the data, it is preempted so that the second process can run. The second process then tries to read the data, which are in an inconsistent state.

The preemption also affects the design of the Operating System kernel. During the processing of a system call, the kernel may be busy with an activity on behalf of a process. Such activities may involve changing important kernel data (such as I/O queues).We need to ensure that if a process is preempted in the middle of these changes it does not create chaos in the OS. This is usually done by waiting for the system call to complete or for an I/O block to be called before context switching.

**4)Dispatcher:**

Another component involved in the CPU-scheduling function is the dispatcher. Only the selection of which process will use the CPU next is done by the CPU scheduler. The dispatcher module is the one that gives control of the CPU to the process selected by the short-term scheduler.

This function involves the following:

* Switching context
* Switching to user mode
* Jumping to the proper location in the user program to restart that program from where it left last time.

The dispatcher should be as fast as possible, given that it is invoked during every process switch. The time taken by the dispatcher to stop one process and start another process is known as the **Dispatch Latency**. The dispatch latency should be as less as possible because the CPU does not do any useful work during this period.