Concept of Scheduling:

- Scheduling is an important function of an operating system. The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.
- The scheduler is the kernel component (module/program) responsible for deciding which program should be executed on the CPU. A scheduler selects a job/task which is to be submitted next for execution.

4.1.2 CPU – I/O Burst Cycles

(W-16, 18, S-18)

- CPU scheduling is greatly affected by how a process behaves during its execution. Almost all the processes continue to switch between CPU (for processing) and I/O devices (for performing I/O) during their execution.
- The success of CPU scheduling depends upon the observed property of processes such as process execution is a cycle of CPU execution and I/O waits.
- Processes alternate back and forth between these two states. Process execution begins with a CPU burst. It is followed by an I/O burst, which is followed by another CPU burst, then another I/O burst and so on.
- Eventually the last CPU burst will end with a system request to terminate execution, rather than another I/O burst.
- In short, we can say that the process execution comprises alternate cycles of CPU burst (the time period elapsed in processing before performing the next I/O operation) and I/O burst (the time period elapsed in performing I/O before the next CPU burst).
- Fig. 4.3 shows the sequence of CPU and I/O bursts.

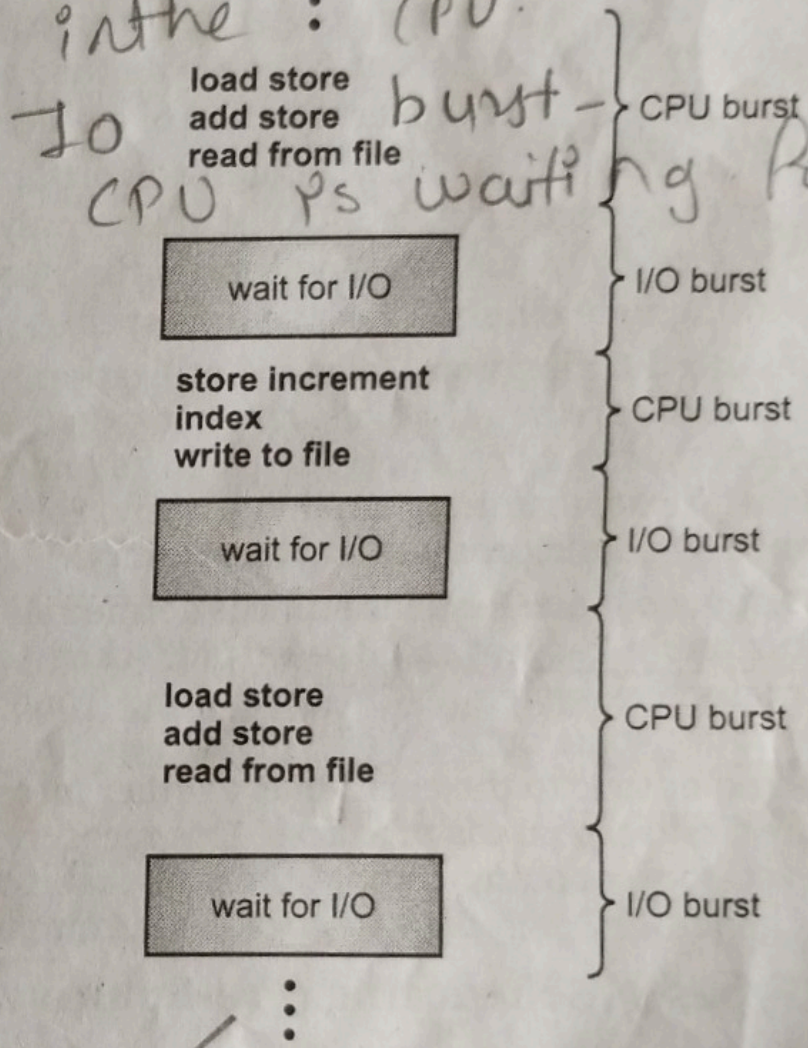

Fig. 4.3: Execution is an Alternating Sequence of CPU and I/O Bursts

- To realize a system's scheduling objectives, the scheduler should consider process behavior.
  1. **CPU Bound Process:** The process which spends more time in computations or with CPU and very rarely with the I/O devices is called as CPU bound process.
  2. **I/O Bound Process:** The process which spends more time in I/O operation than computation (time spends with CPU) is I/O bound process.

4.1.3 Scheduling Criteria (S-16, 17, 18, W-16, 17, 18)

- For scheduling purposes, the scheduler may consider some performance measures and optimization criteria. Different CPU scheduling algorithms have different properties and may favor one class of processes over another.
- In choosing which algorithm to use in a particular situation/condition, the properties of various algorithms must be considered.
- Many criteria have been suggested for comparing CPU scheduling algorithms criteria, which are used, include CPU utilization, balanced utilization, throughput, waiting time, turnaround time and response time.
  1. **CPU Utilization:** CPU utilization is defined as, the percentage of time the CPU is busy in executing processes. For higher utilization, CPU must be kept as busy as possible, that is, there must be some process running at all times. CPU Utilization may range from 0 to 100 percent. In a real system it should range from 40 percent to 90 percent.
  2. **Throughput:** If the CPU is busy executing processes, then work is being done. One measure of work is the number of processes that are completed per time unit called throughput. For long processes, this rate may be one process per hour, for short transactions, throughput might be 10 processes per second. Throughput is defined as, the total number of processes that a system can execute per unit of time.
  3. **Turnaround Time (TAT):** It is the difference between the time a process enters the system and the time it exits the system. From the point of view of a particular process, the important criterion is how long it takes to execute that process. The interval from the time of submission of a process to the time of completion is the turnaround time. Turnaround Time is the sum; of the periods spent waiting to get into memory writing in the ready queue, executing on the CPU and doing I/O. Turnaround time is defined as, the amount of time that has rolled by from the time of creation to the termination of a process. (S-18)
  4. **Waiting Time:** The CPU scheduling algorithm does not affect the amount of time during which a process executes of does I/O, if affects only the amount of time that a process spends waiting in the ready queue. Waiting time is the sum of the periods spent waiting in the ready queue. Waiting time is defined as, the time spent by a process while waiting in the ready queue.
  5. **Balanced Utilization:** Balanced utilization is defined as, the percentage of time all the system resources are busy. It considers not only the CPU utilization but the utilization of I/O devices, memory, and all other resources. To get more work done by the system, the CPU and I/O devices must he kept running simultaneously. For this, it is desirable to load a mixture of CPU-bound and I/O-bound processes in the memory.
  6. **Response Time:** Response time is defined as, the time elapsed between the moment when a user initiates a request and the instant when the system starts responding to this request. In an interactive system TAT (Turnaround Time) may not be best criterion. Often a process can produce some output early and can continue computing new results while previous results are being output to the user. Thus another measure is the time from submission of a request until the first response is produced. This measure called response time is the amount of time it takes to start responding, but not the time that it takes to output that response. The Turnaround Time is generally limited by the speed of the output device.

4.1.4 Types of Scheduling (Pre-Emptive and Non-Preemptive) (S-16, 19, W-16, 17, 18)

- A scheduler is an operating system module (program) that selects a job which is to be admitted next for execution.
- CPU scheduling is the process of selecting a process and allocating the processor to the selected process for execution.

- Scheduling types are depending upon how the process is executed (i.e., whether the CPU is provided to a process completely or provided in cycles with breaks in between i.e. the process is in the waiting or suspended state after certain period of time).
- CPU scheduling decisions may take place under the following four conditions:
  1. When a process switches from the running state to the waiting state (for example, I/O request, or invocation of wait for termination of one of the child processes).
  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, completion of I/O).
  4. When a process terminates.
- Depending on above situations, there are two types of scheduling namely Pre-emptive Scheduling and Non-Preemptive Scheduling.

4.1.4.1 Pre-Emptive Scheduling

(S-16, 19, W-16, 17, 18)

- A pre-emptive scheduling allows a higher priority process to replace a currently running process, even if its time slot is not completed or it has not requested for any I/O.
- If a higher priority process is enters the system, the currently running process is stopped and the CPU transfers the control to the higher priority process.
- The currently running process may be interrupted and moved to the ready state by the operating system. Window-95 introduces pre-emptive scheduling and all subsequent versions of Windows operating systems have used pre-emptive scheduling.
- The Mac OS X operating system for the Macintosh also uses preemptive scheduling.
- The pre-emptive scheduling algorithms is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state.
- The advantage of pre-emptive scheduling algorithms is they allow real multiprogramming.
- The disadvantages of pre-emptive scheduling algorithms are they are complex and they lead the system to race condition.

4.1.4.2 Non-Preemptive Scheduling

(S-16, 19, W-16, 17, 18)

- In non-preemptive scheduling once the CPU is assigned to a process, the processor do not release until the completion of that process. It means that a running process has the control of the CPU and other allocated resources until the normal termination of that process.
- In non-pre-emptive 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. This scheduling method was used by Microsoft Windows 3.
- In Microsoft Windows 3 case, once a process is in the running state, it continues to execute until it terminates or blocks itself to wait for I/O or by requesting some operating system services i.e., if a higher priority process enters the system, the running process cannot be forced to release the control.
- Non-preemptive algorithms are designed so that once a process enters the running state, it cannot be preempted until it completes its allotted time.
- The advantages of non-preemptive scheduling algorithms are they are simple and they cannot lead the system to race condition.
- The disadvantage of non-preemptive scheduling algorithms is they do not allow real multi-programming.

Difference between Pre-emptive and Non-preemptive Scheduling: (S-19)

Sr. No.	Preemptive Scheduling	Non-preemptive Scheduling
1.	Even if CPU is allocated to one process, CPU can be preempted to other process if other process is having higher priority or some other fulfilling criteria.	Once, the CPU has been allocated to a process the process keeps the CPU until releases CPU either by terminating or by switching to waiting state.

contd. ...

2.	Throughput is less.	Throughput is high.
3.	It is suitable for real time system.	It is not suitable for real time system.
4.	Algorithm design is complex.	Algorithm design is simple.
5.	Only the processes having higher priority are scheduled.	Processes having any priority can get scheduled.
6.	It does not treat all processes as equal.	It treats all process as equal.
7.	It has high overhead.	It has low overhead.
8.	Circumstances for pre-emptive: • process switch from running to ready state. • process switch from waiting to ready state.	Circumstances for non-preemptive: • process switches from running to waiting state. • process terminates.
9.	Examples: Round Robin, Priority algorithms.	Example: FCFS algorithm.
10.	In pre-emptive scheduling, a running process may be interrupted by another process that needs to execute.	In non-preemptive scheduling, the processor executes a process till termination without any interruption.
11.	The system resources are used efficiently.	The system resources are not used efficiently.
12.	Pre-emption allows the operating system to interrupt the executing task and handle any important task that requires immediate action.	Non-preemption does not allows the operating system to interrupt the executing task and does not handle any important task that requires immediate action till process termination.

4.3 DEADLOCK

(W-18)

- Deadlock can be defined as, the permanent blocking of a set of processes that either compete for system resources or communicate with each other. Deadlock involves conflicting needs for resources by two or more processes.

4.3.2 Necessary Conditions to Deadlock

(S-17, W-17, 18)

- Deadlock is a situation when two or more processes get locked and cannot processed further because of inter-dependability. In real world, deadlocks can arise when two persons wait for phone calls from one another.
- Deadlock is defined as, "a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process".
- Processes involved in a deadlock remain blocked permanently and this affects OS performance indices like throughput and resource efficiency.
- In a multiprogramming environment, multiple processes may try to access s resource. A deadlock is a situation when a when a process waits endlessly for a resource and the requested resource is being used by another process i.e., waiting for some other resource, (See Fig. 4.10).
- A deadlock arises when the four conditions hold true simultaneously in a system. These four conditions are explained below:

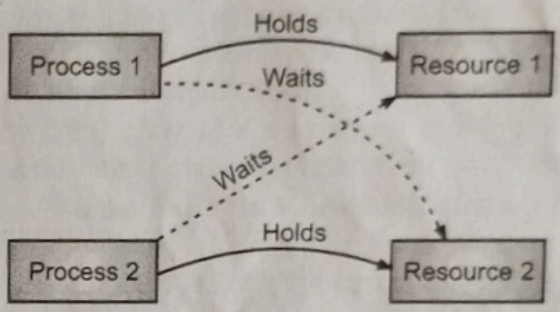

Fig. 4.10: Deadlock

1. **Mutual Exclusion:** At least one resource is held in a non-sharable mode, that is only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released. Each resource is either currently assigned to exactly one process or is available.
2. **Hold and Wait:** There must exist a process that is holding at least one resource and is waiting to acquire additional resources that are currently being held by another process. Process currently holding resources granted earlier can request new resources.
3. **No Pre-emption:** Resources cannot be pre-empted; i.e. resource can only be released voluntarily by the process holding it, after the process has completed its task. Resources previously granted cannot be forcibly taken away from a process. They must be explicitly released by the process holding them.
4. **Circular Wait:** There exist a set (P_0, P_1, ----- P_n) of waiting processes such that P_0 is waiting for a resource which is held by P_1, P_1 is waiting for a resource which is held by P_2. P_{n-1} is waiting for resources which are held by P_n and P_n is waiting for a resource which is held by P_0. Thus there must be a circular chain of two or more processes, each of which is waiting for a resource held by the next member of the chain.

4.3.3.1 Deadlock Prevention

- In previous section (Section 4.3.2) we stated, a deadlock occurs when all of the four conditions (mutual exclusion, hold and wait, no preemption and circular wait) are satisfied at any point of time.
- The deadlock can be prevented by not allowing all four conditions to be satisfied simultaneously, i.e., by making sure that at least one of the four conditions does not hold. In this section we analyze all four conditions one by one and see how their occurrence can be prevented.
- In simple words, deadlock prevention provides a set of methods for ensuring that at least one of the necessary conditions cannot hold.

Eliminating Mutual Exclusion Condition: (S-18)

- The mutual exclusion condition must hold for non-sharable types of resources. For example, several processes cannot simultaneously share a printer.
- Sharable resources do not require mutually exclusive access, and thus cannot be involved in a deadlock. Read only files are a good example of a sharable resource. If several processes attempt to open a read only file at the same time, they can be granted simultaneous access to the file.
- A process never needs to wait for a sharable resource. It is not possible to prevent deadlock by denying the mutual exclusion condition.

Eliminating Hold and Wait Condition:

- In order to ensure that the hold and wait condition never holds in the system, one must guarantee that whenever a process request a resource it does not hold any other resources.
- In other words, hold and wait condition can be eliminated by not allowing any process to request for a resource until it releases the resources held by it, which is impractical as process may require the resources simultaneously.
- One protocol that can be used requires each process to request and be allocated all of its resources before it begins execution. For example, consider a process which copies from a card reader to a disk file, sorts the disk file and then prints the results to a line printer and copies them to a magnetic tape. If all resources to be requested at the beginning of the process, then the process must initially request the card reader, disk file, line printer and tape drive. It will hold the magnetic tape drive for its entire execution, even though it needs only at the end.
- An alternative protocol allows requesting resources only when it has none. A process may request some resources and use them. Before it can request any additional resources, it must release all the resources that it is currently allocated.
- The disadvantages associated with hold and wait condition includes, **poor resource utilization**, since many of the resources may be allocated but unused for a long period of time and **starvation is possible** (a process that needs several popular resources may have to wait indefinitely while at least one of the resources that it needs is always allocated to some other process).

Eliminating No Preemption Condition: (S-18)

- The third necessary condition in deadlock is that there is no pre-emption of resources that have already been allocated. The elimination of no preemptive condition means a process can release the resource held by it.
- If a process requests for a resource held by some other process then instead of making it wait, all the resources currently held by this process can be preempted.
- The process will be restarted only when it is allocated the requested as well as the preempted resources.
- Note that only those resources can be preempted whose current working state can be saved and can be later restored. For example, the system resources, like disk drives and printer cannot be preempted.

Eliminating Circular Wait Condition: (S-18)

- The circular wait condition of deadlock can be eliminated by assigning a priority number to each available resource and a process can request resources only in increasing order.
- Whenever, a process requests for a resource, the priority number of the required resource is compared with the priority numbers of the resources already held by it.

Banker's Algorithm: (S-17, 19, W-18)

- Dijkstra was the first person to propose an algorithm in 1965 for deadlock avoidance. This is known as Banker's algorithm. Banker's algorithm is a deadlock avoidance algorithm. It is named so because this algorithm is used in banking systems to determine whether a loan can be granted or not.
- Banker's algorithm in the operating system is such that it can know in advance before a resource is allocated to a process, whether it can lead to deadlock ("unsafe state") or it can certainly manage to avoid it ("safe state").
- When a new process enters the system, this process declares the maximum number of instances of each resource type that it will require. This number may not exceed the total number of resources in the system.
- When a user requests a set of resources, the system must determine whether the allocation of these resources will leave the system in a safe state. If the process is in safe state, the resources are allocated; otherwise, the process must wait until some other process releases enough resources.
- Several data structures must be maintained to implement the banker's algorithm, where n is the number of processes in the system and m is the number of resource types. The data structures are:
  - **Available:** A vector of length m indicates the number of available resources of each type. If Available[j]=k, then k instances of resource type R_j are available.
  - **Max:** An n x m matrix defines the maximum demand of each process. If Max[i][j]=k, then process Pi may request at most k instances of resource type Rj.
  - **Allocation:** An n x m matrix defines the number of resources of each type currently allocated to each process. If Allocation[i][j]=k, then process Pi is currently allocated k instances of resource type Rj.

Need: An n x m matrix indicates the remaining resource need of each process. If Need[i][j]=k, then process Pi may need k more instances of resource type Rj to complete its task. Note that Need[i][j]=Max[i][j]- Allocation [i][j].

Safety Algorithm: