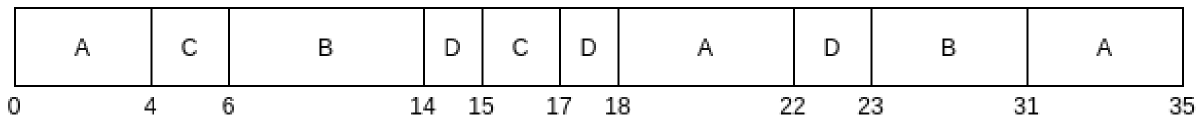
OS ASSIGNMENT - 1

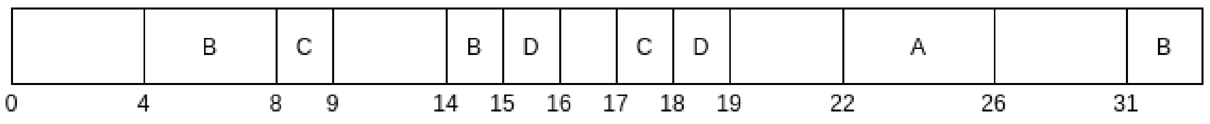
19pw13
Madhumitha

1)

CPU Gantt Chart :



I/O Gantt Chart :



Turnaround time = Completion Time - Arrival Time

Waiting time = Turnaround time - Burst Time

Process	**Turnaround Time**	**Waiting Time**
A	35-0 = 35	35-20 = 15
B	32-2 = 30	30-18 = 12
C	18-3 = 15	15-6 = 9
D	23-7 = 16	16-5 = 11

Average Waiting time: (15+12+9+11) / 4 = 11.75 units

2) Which algorithm will be suitable for CPU Scheduling in the following types of OS? Why ?

I) Batch OS:

Shortest Job First is generally the best algorithm for batch jobs as there is no user interaction. So by using SJF we can maximize the throughput of the CPU, thereby increasing the degree of CPU Utilization. Another alternative might be First Come First Serve.

II) Interactive OS:

Round Robin is the best CPU scheduling algorithm for Interactive OS as any delay is not accepted and processes are needed to be responded quickly. All processes are treated fairly. We just need to set the quantum time appropriately. Another alternative is Priority based scheduling. A better solution will be to have different priority classes or or level and round robin the processes in them.

III) Real Time OS:

Preemptive Priority Scheduling is the best CPU scheduling algorithm for soft Real Time OS as each process is critical and requires a certain amount of urgency and degree of importance. So as long as the age of the process is considered by the scheduler then preemptive priority scheduling is the most suitable as it allows for carrying on tasks which are most important to execute first.

What is the purpose of a command line interpreter ? Why is it usually separated from Kernel ?

- The **Command line interpreter** reads commands from the user or from a file of commands and executes them , usually by turning them into one or more system calls.

- It is usually not part of the kernel since the command line interpreter is subjected to changes.

What happens on a context switch? Should context switches happen frequently or infrequently? Justify

- On a context switch, the OS must save the current execution context in the OS's PCB process and change the program's states to "ready"," waiting" or "terminate" as appropriate
- The OS then selects a process to execute from the ready queue, changes its status and loads it onto the hardware

· It should happen frequently enough so that all the jobs make process before they run.It should happen infrequently enough that their overhead does not take up a significant amount of the total system CPU time. Usually it's between 1-2% of total system time

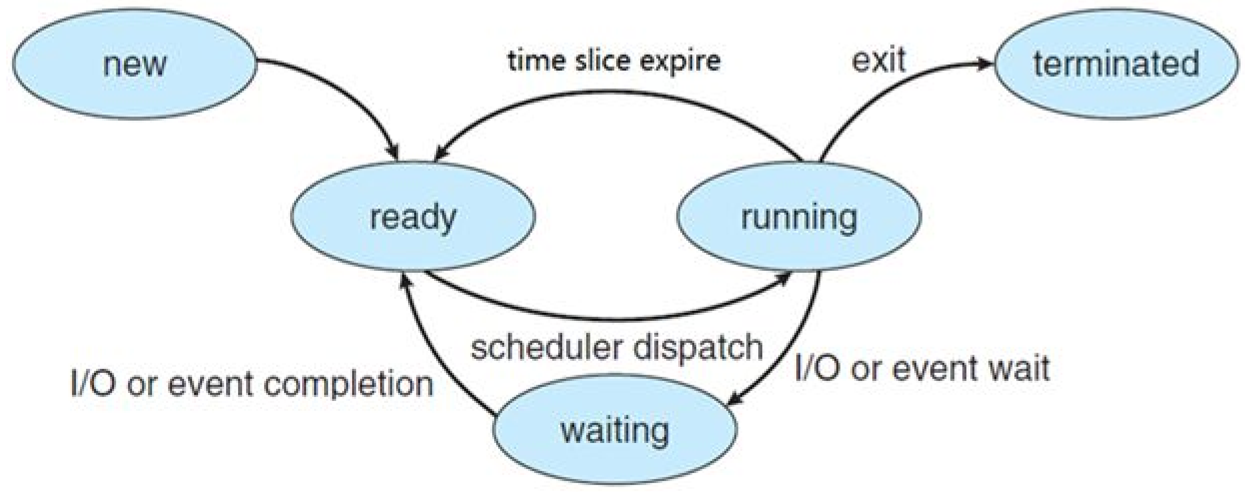
What is blocked suspended and ready suspended state of a process? Why are these states needed?

The processes that were initially in the blocked state in main memory waiting for some event but lack of memory forced them to be suspended and gets placed in main memory are said to be in blocked suspended state

The processes initially in ready state in main memory but lack of memory forced them to be suspended and gets placed in secondary memory are said to be in ready suspended state

These states are required as the processor is faster than I/O devices, so many processes could be waiting for I/O and it is better to swap these processes to secondary memory which frees up the processor to execute other processes.

3)



- **New -> Ready**

The state of the process changes from New to Ready when the CPU is ready to handle more processes than it is currently handling.

- **Ready -> Running**

The state of the process changes from Ready to Running when the OS chooses one of the processes in ready state.

- **Running -> Exit**

The state of the process changes from Running to Exit when the process is completed and is terminated by the OS.

- **Running -> ready**

The most common reason for this transition is that the running processes has
reached the max allowable time or some other high priority process has come in for
execution

- **Running - >Blocked**

A process is put to blocked state if it requests for OS services like an I/O operation, or resource.

- **Blocked -> ready**
  This transition occurs when the event for which the process has been waiting occurs and the process is ready for further processing.

- **Ready -> exit/Blocked -> exit**

The termination of a parent process terminates all its child processes(unless the child was disowned), so even if the child process is ready or waiting for an IO operation, it will generally exit.

a) The process with PID 4 will get executed next as the current process is pointing to that process.
b) The process with PID 13 will get executed as it is not blocked and has highest priority in the processes which ready
c)

Stack	**PID**	**Status**	**Priority**	**Next**	**CPU Burst**
0	4	Running	1	4	33
1	5	Blocked	10	2	134
2	7	Blocked	7	1	58
3	11	Ready	5	0	150
4	13	Ready	8	3	145

d)

Stack	**PID**	**Status**	**Priority**	**Next**	**CPU Burst**
0	4	Ready	1	4	33
1	5	Ready	10	0	134
2	7	Running	7	1	58

3	11	Ready	5	2	150
4	13	Ready	8	3	145

e)

Stack	**PID**	**Status**	**Priority**	**Next**	**CPU Burst**
0	4	Ready	1	4	33
1	5	Blocked	10	1	134
2	7	Running	7	5	58
3	11	Ready	5	2	150
4	13	Ready	8	3	145
5	24	Ready	6	0	-

4)

Consider the implementation of all the short-term scheduling algorithms.

(i) Which short-term scheduling algorithms are effectively impossible to implement? Explain why?

Shortest Job First is effectively impossible to implement as we can never accurately predict how much time the process requires.

(ii) Which algorithms require a Timer interrupt for the CPU?

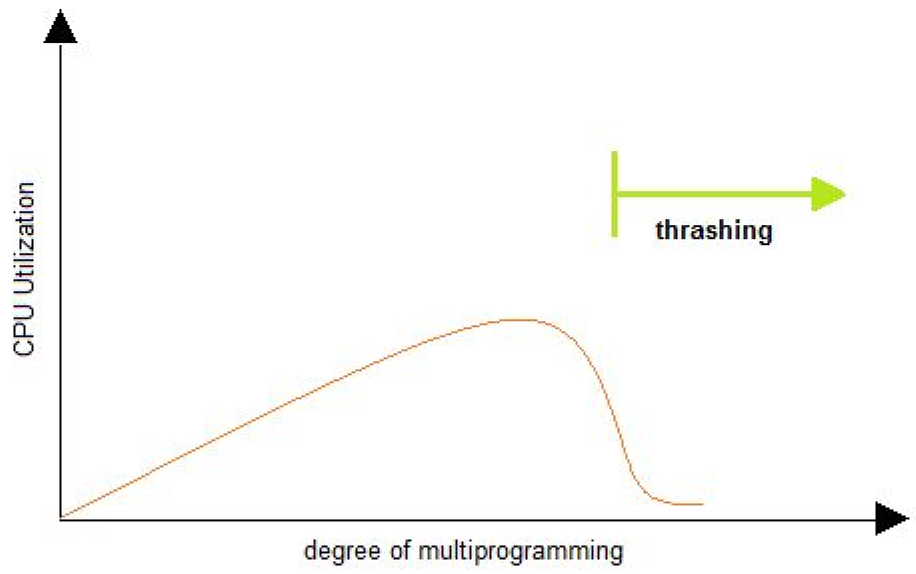
Round Robin requires timer interrupt as each process is allocated a predefined quantum and needs to be interrupted on a timely basis.In round robin scheduling, after one predefined time quantum or time slice, the process is interrupted and context switch occurs. The time per process needs to be maintained by a Timer and the Timer interrupt interrupts after the predefined time quanta.

(iii) Why RR Is the Only Real Scheduling Algorithm That Can Be Used in Practice?

Round Robin is used in practice as it is *simple, easy to implement* and most importantly *starvation-free*. In FCFS the processes are starved if the processes with large CPU-burst enters. SJF is hard to implement as the time taken by the process is unpredictable. In Priority Scheduling the low priority processes are starved leaving RR which is viable.

5)

The phenomenon observed in the graph is called ***Thrashing****.* Another way to refer to it might be oversubscription of the CPU. This occurs when processes with a low amount of frames are queued for execution and the CPU wastes a lot of time just doing context switches and loading pages, that is trying to load from Virtual memory and spends a major amount of its time in servicing page faults that occur due to it.



<u>Methods to prevent Thrashing:</u>

- **The Working Set Model :** This model is based on the Locality Model. It states that if we allocate enough frames to a process to accommodate its current locality, it will only fault whenever it moves to some new locality. But if the allocated frames are lesser than the size of the current locality, the process is bound to thrash.

- **The Page Fault Frequency:** If the page fault rate falls below the lower limit, frames can be removed from the process. Similarly, if the page fault rate exceeds the upper limit, more number of frames can be allocated to the process.

- **Context Switching:** When a *Context Switch* occurs, the state of the process or a thread is stored so that it can be resumed at a later point. Context switching is desirable because it allows for multiprogramming without the use of extra processors.

What is meant by Context-switching? Why are context switches desirable?

Context Switching involves storing the context or state of a process so that it can be reloaded when required and execution can be resumed from the same point as earlier. This is a feature of a multitasking operating system and allows a single CPU to be shared by multiple processes.

Due to some reasons, a running process is interrupted and the Operating System assigns another process to the running state and turns control over to the process. A process switch may occur any time the OS has gained control from the currently running process. Possible events giving OS control are: interrupts, trap and supervisor call.

Ex: Process A might need some input, so it goes for input (waiting state). Process B which has been scheduled next, comes from ready state to running state and starts utilizing CPUs resources.

A context switching helps to share a single CPU across all processes to complete its execution and store the system's tasks status. When the process reloads in the system, the execution of the process starts at the same point where it stopped the previous time.

6)

Does this algorithm give the highest priority to new processes?

No, scheduler alternates between old and new ,hence equal priority is given.This algorithm does not give priority to new processes as average burst time is greater than 2 milliseconds most of the processes are moved to the old queue.

Is this algorithm starvation free?

This algorithm is starvation free as the scheduler is basically following FCFS as every process is executed when its chance comes.Every job that is not executed from the new queue eventually ends up in the old queue and from there eventually reaches the head of the queue and is executed.

Is this algorithm fair to all processes. By "fair" we mean every process has a wait time approximately equal to the average wait time, assuming all processes have close to the same execution time

No, it is not fair to every process because processes that arrive in the new queue at a point of time where the CPU wants to select from the new queue the waiting time will be 0 to 2 milliseconds whereas the processes that reside in the old queue will keep accumulating wait time.

7) How does the mixture of I/O bound processes with CPU bound processes maximize system utilization?

CPU bound – process which mostly takes up the CPU's processing capabilities

I/O bound – process which mostly needs a considerable amount of time doing I/O operations.

If there is a mixture of I/O bound and CPU bound processes, then both these resources can be utilised effectively and their idle time would be relatively less. While one process uses the CPU, the other could use the I/O (controlled by the DMA controller) and therefore both won't be idle.

If the majority of the processes are CPU bound, I/O is idle (wait queue is mostly empty). If the majority of the processes are I/O bound, the CPU is idle (ready queue is mostly empty).

Why is this more important in batch systems than in multiprocessing systems?

A mixture of CPU and I/O bound processes reduces CPU idle time. This is more important in batch processing because multiprocessing already has higher CPU utilisation (it is optimised for that). So, implementing it in a system that has lesser CPU utilisation (which is batch systems) will help improve it a lot more and will be more beneficial to the system.

Example Event that triggers each transition:

1. Running -> Blocked

   Say a process is running and it requires I/O. So an I/O interrupt is issued and it goes to blocked state.

2. Running -> Ready

   In pre-emptive scheduling, say in a Priority Scheduling algorithm implementation, if a process is running and a new process with higher priority enters the system, then the currently running process is pre-empted (goes from running to ready state) and the new process begins utilising the CPU.

3. Ready -> Running

   When the execution of one process gets over, it moves to a terminated state. In that situation, the CPU free's up for the next process and according to the scheduling algorithm, one process is chosen to utilise the CPUs resources. This process moves from Ready to Running State.

4. Blocked-> Ready

Say, a process was waiting for an I/O operation and it was in blocked state. Now, that operation is over and it is ready to go back to utilising the processor. So, it moves from blocked to ready state, to show that it is a viable candidate for executing (utilising the CPU) next.

8)

FIRST-COME/FIRST SERVED:

1	R	R	R	R	R	R	R	R	R	W	W	W	W	W	W	W	R	R	R	R	R	R					
2	D	D	D	D	D	D	D	D	D	R	R	W	W	W	W	W	W	D	D	D	D	D	R	R	R	R	R

SHORTEST JOB FIRST (NON - PREEMPTIVE):

1	D	D	R	R	R	R	R	R	R	R	R	W	W	W	W	W	W	W	R	R	R	R	R	R
2	R	R	W	D	D	D	D	D	D	D	D	R	R	R	R	R								

PREEMPTIVE SHORTEST JOB FIRST:

1	D	D	R	D	D	D	D	D	R	R	R	R	R	R	R	R	W	W	W	W	W	W	W	R	R	R	R	R	R
2	R	R	W	R	R	R	R	R																					

ROUND ROBIN WITH QUANTUM OF 3:

1	R	R	R	D	D	R	R	R	D	D	D	R	R	R	W	W	W	W	W	W	W	R	R	R	R	R	R
2	D	D	D	R	R	W	D	D	R	R	R	D	D	D	R	R											

		Waiting time			***Time finished***	
Algorithm	CPU Utilization	Process 1	Process 2	Average	Process 1	Process 2
FCFS	81.5 %	0	14	7	22	27
SJF	91.66 %	2	8	5	24	16
PSJF	75.86 %	7	0	3.5	29	8
RR 3	81.48 %	5	8	7.5	27	16