# OPERATING SYSTEM: CSET209

# CONTENTS COVERED

- CPU Scheduling Algorithms
- First Come First Served (FCFS)
- Shortest-Job-First (SJF)
- Shortest Remaining Time First (SRTF)

# DIFFERENT TIMES NOTATION IN CPU SCHEDULING

1. **Arrival Time (AT):** Time at which process arrives in the ready queue.
2. **Burst Time (BT):** Duration of execution time required by process in CPU.
3. **Completion Time (CT):** Time at which process completes.
4. **Turn Around Time (TAT):** Total time spent by the process in the system.
5. **Waiting Time (WT):** How much time processes spend in the ready queue waiting their turn to get on the CPU.
6. **Response time (RT):** Response time is the time spent when the process is in the ready state and gets the CPU for the first time.

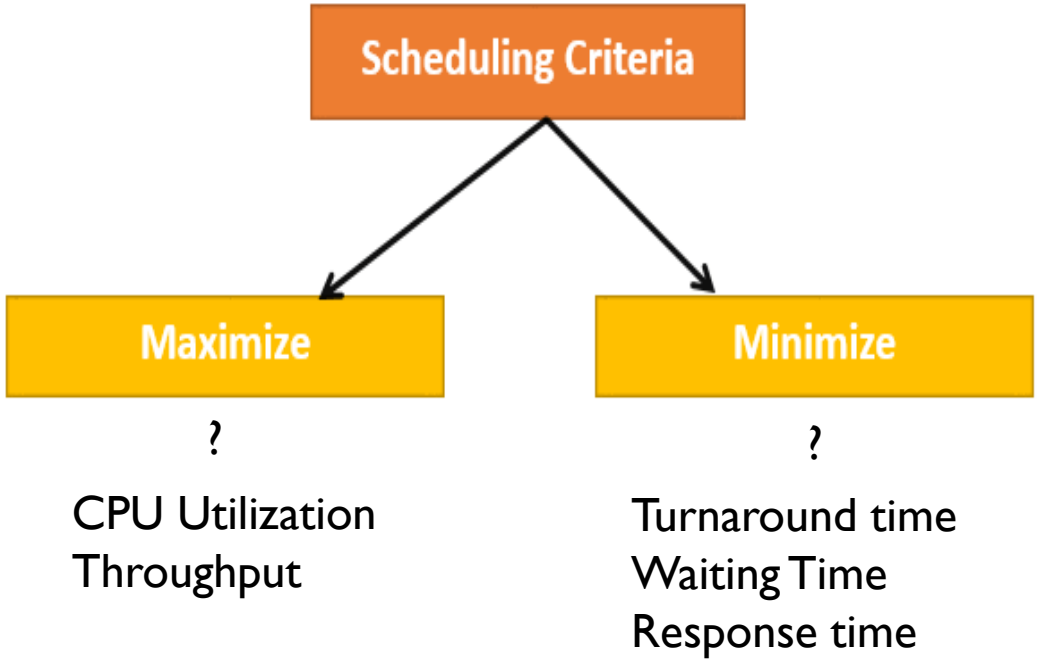
| |
|---|
| Turn Around Time = Completion Time - Arrival Time |
| Waiting Time = Turnaround time - Burst Time |

**The average** TAT and WT are determined by summing the respective TAT and WT of all the processes and divided the sum by the total number of processes (n).

# METRICS TO MEASURE PERFORMANCE OF A CPU SCHEDULING ALGO

**Many criteria have been suggested for comparing CPU scheduling algorithms, including**:

- **CPU utilization** - CPU should be kept as busy as possible
- **Throughput** - Number of processes completed per unit time is called *throughput*.
- **Response time** - Response time is the time from the submission of a request until the first response is produced.



# CPU SCHEDULING

# FCFS SCHEDULING

- **First come first serve** (FCFS) scheduling algorithm simply schedules the jobs **according to their arrival time**. The job which comes first in the ready queue will get the CPU first. The lesser the arrival time of the job, the sooner will the job get the CPU.

# (FCFS) SCHEDULING: EXAMPLE 1

- Let's take an example of The FCFS scheduling algorithm. In the Following schedule, there are 5 processes with process ID **P0, P1, P2, P3 and P4**. P0 arrives at time 0, P1 at time 1, P2 at time 2, P3 arrives at time 3 and Process P4 arrives at time 4 in the ready queue. The processes and their respective Arrival and Burst time are given in the following table.
- The Turnaround time (TAT) and the waiting time (WT) are calculated by using the following formula

```
Turn Around Time = Completion Time - Arrival Time
Waiting Time = Turnaround time - Burst Time
```

The average TAT and WT are determined by summing the respective TAT and WT of all the processes and divided the sum by the total number of processes.

# (FCFS) SCHEDULING: EXAMPLE I

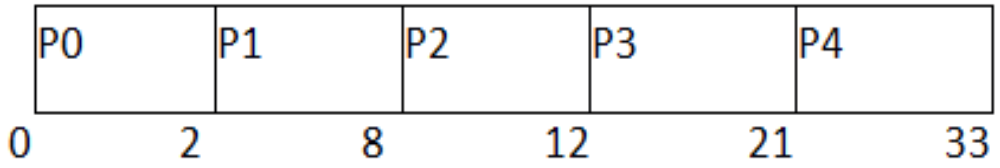
| Process ID | Arrival Time | Burst Time (millisecond) |
|---|---|---|
| 0 | 0 | 2 |
| 1 | 1 | 6 |
| 2 | 2 | 4 |
| 3 | 3 | 9 |
| 4 | 6 | 12 |

# (FCFS) SCHEDULING: EXAMPLE I

| Process ID | Arrival Time | Burst Time (millisecond) | Completion Time | Turn Around Time (TAT) | Waiting Time (WT) |
|---|---|---|---|---|---|
| 0 | 0 | 2 | 2 | 2 | 0 |
| 1 | 1 | 6 | 8 | 7 | 1 |
| 2 | 2 | 4 | 12 | 10 | 6 |
| 3 | 3 | 9 | 21 | 18 | 9 |
| 4 | 6 | 12 | 33 | 27 | 15 |

**Avg TAT Time=64/5 = 12.8 msec**

**Avg Waiting Time=31/5 = 6.2 msec**




Gantt Chart

# (FCFS) SCHEDULING: EXAMPLE 2

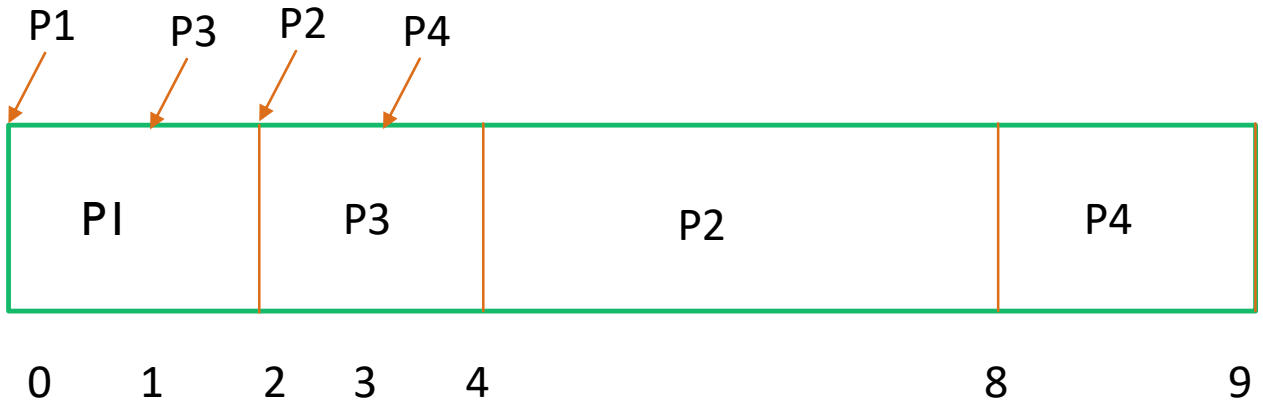
## Example: 2

| Process | Arrival Time | Burst Time (msec) |
|---|---|---|
| P1 | 0 | 2 |
| P2 | 2 | 4 |
| P3 | 1 | 2 |
| P4 | 3 | 1 |

Calculate following:

1. Calculate waiting time for all process
2. Average waiting time
3. Calculate turnaround time for all process
4. Average turnaround time

# (FCFS) SCHEDULING: EXAMPLE 2




Waiting time of P1 = 0 msec; P2 = 2 msec; P3 = 1 ; P4 = 5 msec;
Average waiting time: (0 + 2 + 1+5)/4 = 2 msec

Turn Around time of P1 = 2 msec; P2 = 6 msec; P3 = 3 msec; P4 = 6 msec;
Average turn around time: (2 + 6 + 3+6)/4 = 4.25 msec

# CONVOY EFFECT

**Consider the scenario.**

- One CPU-bound process and several I/O-bound processes.

As the processes flow around the system, the CPU-bound process will get and hold the CPU. During this time, all the other processes will finish their I/O and will move into the ready queue, **waiting for the CPU. While the processes wait in the ready queue, the I/O devices are idle.**

Eventually, the CPU-bound process finishes its CPU burst and moves to an I/O device. All the I/O-bound processes, which have short CPU bursts, execute quickly and move back to the I/O queues. At this point, the **CPU sits idle.** The CPU-bound process will then move back to the ready queue and be allocated the CPU. Again, all **the I/O processes end up waiting in the ready queue** until the CPU-bound process is done.

There is called as **convoy effect** as all the other processes wait for the one big process to get off the CPU.

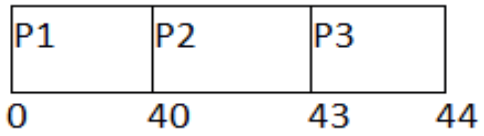
**Consequence:**

- This effect results in lower CPU and device utilization than might be possible if the shorter processes were allowed to go first.

# ILLUSTRATING CONVOY EFFECT IN FCFS

**Example: Consider** 3 processes named as **P1, P2 and P3**. The Burt Time of process P1 is highest.

**Case 1:** In the First scenario, The Process P1 arrives at the first in the queue although; the burst time of the process is the highest among all. Since, the Scheduling algorithm, we are following is FCFS hence the CPU will execute the Process P1 first. In this schedule, the average waiting time of the system will be very high. That is because of the convoy effect. The other processes P2, P3 have to wait for their turn for 40 units of time although their burst time is very low. This schedule suffers from starvation.

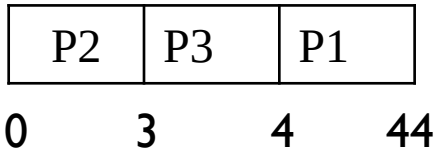



Avg waiting Time = 81/3=27 msec

| Process ID | Arrival Time | Burst Time | Completion Time | Turn Around Time | Waiting Time |
|---|---|---|---|---|---|
| 1 | 0 | 40 | 40 | 40 | 0 |
| 2 | 1 | 3 | 43 | 42 | 39 |
| 3 | 1 | 1 | 44 | 43 | 42 |

# ILLUSTRATING CONVOY EFFECT IN FCFS

- **Case 2:** In the Second scenario, If Process P1 would have arrived at the last of the queue and the other processes P2 and P3 at earlier then the problem of starvation would not be there. Following example shows the deviation in the waiting times of both the scenarios. Although the length of the schedule is same that is 44 units but the waiting time will be lesser in this schedule.




**Avg Waiting Time=6/3 =2 msec**

| Process ID | Arrival Time | Burst Time | Completion Time | Turn Around Time | Waiting Time |
|---|---|---|---|---|---|
| 1 | 1 | 40 | 44 | 43 | 3 |
| 2 | 0 | 3 | 3 | 3 | 0 |
| 3 | 0 | 1 | 4 | 4 | 3 |

# FCFS SCHEDULING

**Advantages of FCFS**

- Simple and Easy to implement.
- Easy to implement, it doesn’t require complex data structures.
- Since processes are executed in the order they arrive, there’s no risk of starvation.

**Disadvantages of FCFS**

- FCFS scheduling **may cause the problem of long average waiting time** if the burst time of the first process is the longest among all the jobs.
- The scheduling method is non preemptive, the process will run to the completion, thus particularly troublesome for interactive systems, where it is important that each process get a share of the CPU at regular intervals.
- Processes that are at the end of the queue, have to wait longer to finish.
- It is not suitable for time-sharing operating systems where each process should get the same amount of CPU time.

```
// Structure to represent a process

struct Process {
    int pid;        // Process ID
    int arrival;    // Arrival Time
    int burst;      // Burst Time
    int waiting;    // Waiting Time
    int turnaround; // Turnaround Time
};
```

Space: 20 Byte (if int take 4 byte)

```
void findWaitingTime(struct Process processes[], int n) {
    processes[0].waiting = 0; // First process has no waiting time

    for (int i = 1; i < n; i++) {
        processes[i].waiting = processes[i - 1].waiting + processes[i - 1].burst;
    }
}

void findTurnaroundTime(struct Process processes[], int n) {
    for (int i = 0; i < n; i++) {
        processes[i].turnaround = processes[i].waiting + processes[i].burst;
    }
}
```

```
void findAverageTime(struct Process processes[], int n) {
    int totalWaiting = 0, totalTurnaround = 0;

    findWaitingTime(processes, n);
    findTurnaroundTime(processes, n);



    for (int i = 0; i < n; i++) {
        totalWaiting += processes[i].waiting;
        totalTurnaround += processes[i].turnaround;
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", processes[i].pid, processes[i].arrival,
                processes[i].burst, processes[i].waiting, processes[i].turnaround);
    }

    printf("\nAverage Waiting Time = %.2f", (float)totalWaiting / n);
    printf("\nAverage Turnaround Time = %.2f\n", (float)totalTurnaround / n);
}
```

# SHORTEST JOB FIRST

# SHORTEST-JOB-FIRST (SJF) SCHEDULING

- In SJF scheduling, the process with the lowest burst time, among the list of available processes in the ready queue, is going to be scheduled next. However, it is very difficult to predict the burst time needed for a process hence this algorithm is very difficult to implement in the system.

**Advantages of SJF**

- Maximum throughput
- Minimum average waiting and turnaround time

- **Disadvantages of SJF**

- May suffer with the problem of starvation
- It is not implementable because the exact Burst time for a process can't be known in advance.

**Two schemes:**

- Non-preemptive – once CPU given to the process it cannot be preempted until completes its CPU burst.
- Preemptive – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is know as the Shortest-Remaining-Time-First (SRTF).

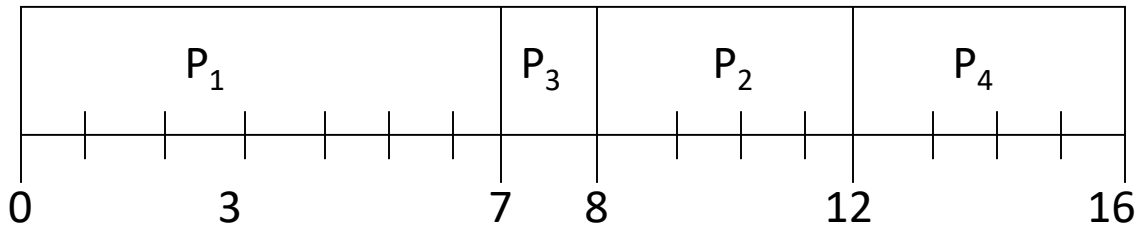
- SJF is optimal – gives minimum average waiting time for a given set of processes.

# Shortest-Job-First (SJF) Example 1

## EXAMPLE OF NON-PREEMPTIVE SJF

| Process | Arrival Time | Burst Time |
|---|---|---|
| $P_1$ | 0.0 | 7 |
| $P_2$ | 2.0 | 4 |
| $P_3$ | 4.0 | 1 |
| $P_4$ | 5.0 | 4 |

- SJF (non-preemptive)

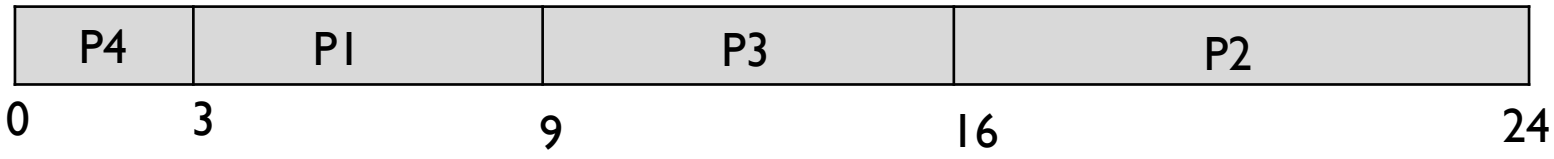



- Average waiting time = (0 + 6 + 3 + 7)/4 = 4
- Turn around time: ??

# EXAMPLE 2: NON-PREEMPTIVE SJF

| Process | Burst Time |
|---|---|
| $P_1$ | 6 |
| $P_2$ | 8 |
| $P_3$ | 7 |
| $P_4$ | 3 |

- SJF scheduling chart




- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7
- Turn around time??

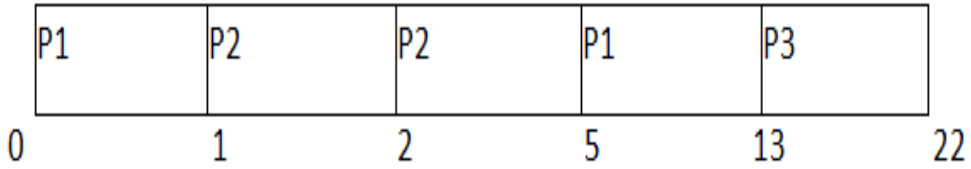
# Algorithm

- Sort all the processes according to the arrival time.
- Then select that process that has minimum arrival time and minimum Burst time.
- After completion of the process make a pool of processes that arrives afterward till the completion of the previous process and select that process among the pool which is having minimum Burst time.

**Time Complexity: O(n^2) // depends on sorting algorithm**
**Auxiliary Space: O(n)**

# Preemptive SRTF Example I

- Consider there are three jobs P1, P2 and P3. P1 arrives at time unit 0; it will be scheduled first for the time until the next process arrives. P2 arrives at 1 unit of time. Its burst time is 4 units which is least among the jobs in the queue. Hence it will be scheduled next.
- At time 2, P3 will arrive with burst time 9. Since remaining burst time of P2 is 3 units which are least among the available jobs. Hence the processor will continue its execution till its completion. Because all the jobs have been arrived so no preemption will be done now and all the jobs will be executed till the completion according to SJF.




Avt TAT = 37/3=12.33 msec
Avg Waiting Time = (4+0+11)/3 = 5 msec.

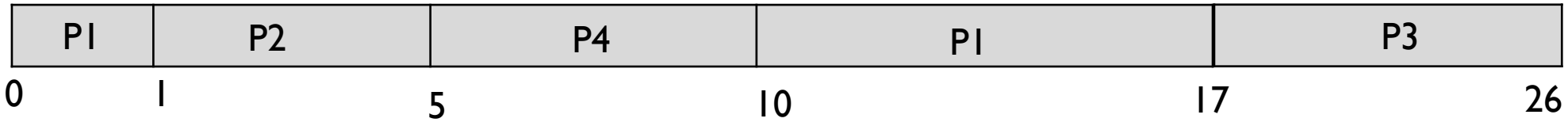
| Process ID | Arrival Time | Burst Time | Completion Time | Turn Around Time | Waiting Time |
|---|---|---|---|---|---|
| 1 | 0 | 9 | 13 | 13 | 4 |
| 2 | 1 | 4 | 5 | 4 | 0 |
| 3 | 2 | 9 | 22 | 20 | 11 |

# PREEMPTIVE SRTF EXAMPLE 2

- Now we add the concepts of varying arrival times and preemption to the analysis

| Process | *Arrival* Time | Burst Time |
|---|---|---|
| $P_1$ | 0 | 8 |
| $P_2$ | 1 | 4 |
| $P_3$ | 2 | 9 |
| $P_4$ | 3 | 5 |

- *Preemptive* SJF Gantt Chart




- Average waiting time = [(10-1)+(1-1)+(17-2)+5-3)]/4 = 26/4 = 6.5 msec

# Algorithm

- Traverse until all process gets completely executed.
  - Find process with minimum remaining time at every single time lap.
  - Reduce its time by 1.
  - Check if its remaining time becomes 0
  - Increment the counter of process completion.
  - Completion time of current process = current_time + 1;
  - Calculate waiting time for each completed process.
    - **wt[i]= Completion time – arrival_time-burst_time**
  - Increment time lap by one.
- Find turnaround time (waiting_time + burst_time).

# SRTF

**Advantages:**

- Short processes are handled very quickly.
- The system also requires very little overhead since it only makes a decision when a process completes or a new process is added.
- When a new process is added the algorithm only needs to compare the currently executing process with the new process, ignoring all other processes currently waiting to execute.

**Disadvantages:**

- Like shortest job first, it has the potential for process starvation.
- Long processes may be held off indefinitely if short processes are continually added.

**Time Complexity:** O(N)
**Auxiliary Space:** O(N)

# THANK YOU
?