Adama Science and Technology University
School of Electrical Engineering and Computing

Department of Software Engineering

Operating System Group Project

Name	ID		
1.ABDU AHMED	UGR/30024/15		
2.ABDURAHMAN ALIYI	UGR/30040/15		
3.ESMAEL SHIKUR	UGR/30480/15		
4.Nazira Worku	UGR/31080/15		
5.Fitsum Tafese	UGR/25418/14	4th	CSE add

Submitted date:16/05/2025
Submitted to: Mr Edris

```
#############################################
# CPU Scheduling Algorithms - Shell Script
# With Descriptions and Code
#############################################

#################################################
# 1. First-Come, First-Served (FCFS)
#################################################
# Description:
# FCFS is the simplest scheduling algorithm.
# Processes are executed in the order they arrive.
# It is non-preemptive and fair but can lead to
# poor average waiting time if long jobs arrive first.
#
# Purpose:
# To provide fairness by serving jobs in their order
# of arrival. Best used for batch systems.
```

1, First-Come, First-Served (FCFS)

Description

FCFS (First-Come, First-Served) is the simplest CPU scheduling algorithm. Processes are executed in the exact order in which they arrive in the ready queue. Once a process starts execution, it runs until completion (non-preemptive).

Purpose

The main purpose of FCFS is to ensure **fairness** — every process gets a turn based on its arrival time. It's useful in systems where predictability is more important than performance.

```
fcfs() {
  echo -e "\n--- FCFS Scheduling ---"
  echo -n "Enter number of processes: "
  read n

  for ((i = 0; i < n; i++)); do
    echo -n "Enter Arrival Time and Burst Time for Process P$i (format: AT BT): "
    read at bt
    arrival[$i]=$at
    burst[$i]=$bt
    pid[$i]=$i
  done
```

```
  for ((i = 0; i < n - 1; i++)); do
    for ((j = i + 1; j < n; j++)); do
      if ((arrival[i] > arrival[j])); then
        t=${arrival[i]}; arrival[i]=${arrival[j]}; arrival[j]=$t
        t=${burst[i]}; burst[i]=${burst[j]}; burst[j]=$t
        t=${pid[i]}; pid[i]=${pid[j]}; pid[j]=$t
      fi
    done
```

```
  done
```

```
  wt[0]=0
  tat[0]=${burst[0]}
  total_wt=0
  total_tat=${tat[0]}
```

```
  for ((i = 1; i < n; i++)); do
    wt[i]=$((tat[i-1] - arrival[i] + arrival[i-1]))
    if ((wt[i] < 0)); then wt[i]=0; fi
    tat[i]=$((wt[i] + burst[i]))
    total_wt=$((total_wt + wt[i]))
    total_tat=$((total_tat + tat[i]))
  done
```

```
  echo -e "\nPID\tArrival\tBurst\tWaiting\tTurnaround"
  for ((i = 0; i < n; i++)); do
    echo -e "P${pid[i]}\t${arrival[i]}\t${burst[i]}\t${wt[i]}\t${tat[i]}"
  done
```

```
  avg_wt=$((total_wt / n))
avg_tat=$((total_tat / n))
```

```
echo -e "\nAverage Waiting Time: $avg_wt"
echo -e "Average Turnaround Time: $avg_tat"
```

```
}
```

```
# ✅ This line is crucial to run the function
fcfs
```

When we run our code here is the out put

```
hp@DESKTOP-J1G48NA MINGW64 ~/Desktop/UniversityProject/4th-year/os/FCFS
$ ./fcfs.sh

--- FCFS Scheduling ---
Enter number of processes: 3
Enter Arrival Time and Burst Time for Process P0 (format: AT BT): 2 3
Enter Arrival Time and Burst Time for Process P1 (format: AT BT): 4 5
Enter Arrival Time and Burst Time for Process P2 (format: AT BT): 6 7

PID     Arrival Burst   Waiting Turnaround
P0      2       3       0       3
P1      4       5       1       6
P2      6       7       4       11

Average Waiting Time: 1
Average Turnaround Time: 6
```

2,Shortest Job First (SJF) - Non-Preemptive

Description

SJF (Shortest Job First) scheduling selects the process with the shortest burst (execution) time from the ready queue. It is non-preemptive, meaning the selected process runs until it completes.

Purpose

SJF aims to **minimize average waiting time** and **turnaround time** by executing the quickest tasks first, making it ideal for batch systems with known burst times.

```
sjf() {
  echo -e "\n--- SJF Scheduling ---"
  echo -n "Enter number of processes: "
  read n

  for ((i = 0; i < n; i++)); do
    echo -n "Enter Burst Time for P$i: "
    read bt
    burst[$i]=$bt
    pid[$i]=$i
  done
```

```
  for ((i = 0; i < n - 1; i++)); do
    for ((j = i + 1; j < n; j++)); do
      if ((burst[i] > burst[j])); then
        t=${burst[i]}; burst[i]=${burst[j]}; burst[j]=$t
        t=${pid[i]}; pid[i]=${pid[j]}; pid[j]=$t
      fi
    done
  done
```

```
  wt[0]=0
  tat[0]=${burst[0]}
  total_wt=0
  total_tat=${tat[0]}
```

```
  for ((i = 1; i < n; i++)); do
    wt[i]=$((wt[i-1] + burst[i-1]))
    tat[i]=$((wt[i] + burst[i]))
    total_wt=$((total_wt + wt[i]))
    total_tat=$((total_tat + tat[i]))
  done
```

```
  echo -e "\nPID\tBurst\tWaiting\tTurnaround"
  for ((i = 0; i < n; i++)); do
    echo -e "P${pid[i]}\t${burst[i]}\t${wt[i]}\t${tat[i]}"
  done
```

```
  avg_wt=$((total_wt / n))
avg_tat=$((total_tat / n))
```

```
echo -e "\nAverage Waiting Time: $avg_wt"
echo -e "Average Turnaround Time: $avg_tat"
```

```
}
# call to run
sjf
```

When we run the code here is the out put

```
hp@DESKTOP-J1G48NA MINGW64 ~/Desktop/UniversityProject/4th-year/os/SJF
$ ./sjf.sh

--- SJF Scheduling ---
Enter number of processes: 3
Enter Burst Time for P0: 2
Enter Burst Time for P1: 3
Enter Burst Time for P2: 1

PID     Burst   Waiting Turnaround
P2      1       0       1
P0      2       1       3
P1      3       3       6

Average Waiting Time: 1
Average Turnaround Time: 3
```

3 Priority Scheduling - Non-Preemptive

Description

Priority Scheduling assigns each process a priority and schedules processes based on their priority values. A process with a **lower numeric priority** value is treated as **higher priority**. Like FCFS and SJF, it is non-preemptive.

Purpose

The goal is to **execute important tasks first**, often used in systems where some processes are more critical than others, such as real-time or emergency-response systems.

Shell Script Code:

```
priority() {
  echo "----- Priority Scheduling ---------"
  echo -n "Enter number of processes: "
  read n

  for ((i=0; i<n; i++)); do
    echo -n "Enter Burst Time and Priority for P$i (format: BT PR): "
    read bt pr
    burst[$i]=$bt
    priority[$i]=$pr
    pid[$i]=$i
  done
```

```
  for ((i=0; i<n-1; i++)); do
    for ((j=i+1; j<n; j++)); do
      if ((priority[i] > priority[j])); then
        temp=${priority[i]}; priority[i]=${priority[j]}; priority[j]=$temp
        temp=${burst[i]}; burst[i]=${burst[j]}; burst[j]=$temp
        temp=${pid[i]}; pid[i]=${pid[j]}; pid[j]=$temp
```

```
      fi
    done
  done
```

```
  wt[0]=0
  tat[0]=${burst[0]}
  total_wt=0
  total_tat=${tat[0]}
```

```
  for ((i=1; i<n; i++)); do
    wt[i]=$((wt[i-1] + burst[i-1]))
    tat[i]=$((wt[i] + burst[i]))
    total_wt=$((total_wt + wt[i]))
    total_tat=$((total_tat + tat[i]))
  done
```

```
  echo -e "\nPID\tPriority\tBurst\tWaiting\tTurnaround"
  for ((i=0; i<n; i++)); do
    echo -e "P${pid[i]}\t${priority[i]}\t\t${burst[i]}\t${wt[i]}\t${tat[i]}"
  done
```

```
  avg_wt=$((total_wt / n))
avg_tat=$((total_tat / n))
```

```
echo -e "\nAverage Waiting Time: $avg_wt"
echo -e "Average Turnaround Time: $avg_tat"
}
# call it to run
priority
```

When we run the code we get

```
hp@DESKTOP-J1G48NA MINGW64 ~/Desktop/UniversityProject/4th-year/os/PRIORITY
$ ./priority.sh
----- Priority Scheduling -----
Enter number of processes: 3
Enter Burst Time and Priority for P0 (format: BT PR): 2 3
Enter Burst Time and Priority for P1 (format: BT PR): 1 3
Enter Burst Time and Priority for P2 (format: BT PR): 4 5

PID     Priority        Burst   Waiting Turnaround
P0      3               2       0       2
P1      3               1       2       3
P2      5               4       3       7

Average Waiting Time: 1
Average Turnaround Time: 4
```

4 Round Robin Scheduling

Description

Round Robin (RR) scheduling assigns a fixed time quantum (or time slice) to each process in the ready queue and cycles through them. If a process doesn't finish within its time slice, it's sent to the back of the queue.

Purpose

The purpose of RR is to ensure **responsive and fair multitasking** in time-sharing systems. It's particularly effective in **interactive environments**, like user-facing operating systems.

```
round_robin() {
  echo "----- Round Robin Scheduling---------"
  echo -n "Enter number of processes: "
  read n

  for ((i=0; i<n; i++)); do
    echo -n "Enter Burst Time for P$i: "
    read bt
    burst[$i]=$bt
    remaining[$i]=$bt
    pid[$i]=$i
  done
```

```
  echo -n "Enter Time Quantum: "
  read tq
```

```
  time=0
  completed=0
  for ((i=0; i<n; i++)); do wt[i]=0; done
```

```
  while ((completed < n)); do
    for ((i=0; i<n; i++)); do
      if ((remaining[i] > 0)); then
        if ((remaining[i] > tq)); then
          time=$((time + tq))
          remaining[i]=$((remaining[i] - tq))
        else
          time=$((time + remaining[i]))
          wt[i]=$((time - burst[i]))
          remaining[i]=0
          ((completed++))
        fi
      fi
    done
  done
```

```
  total_wt=0
  total_tat=0
```

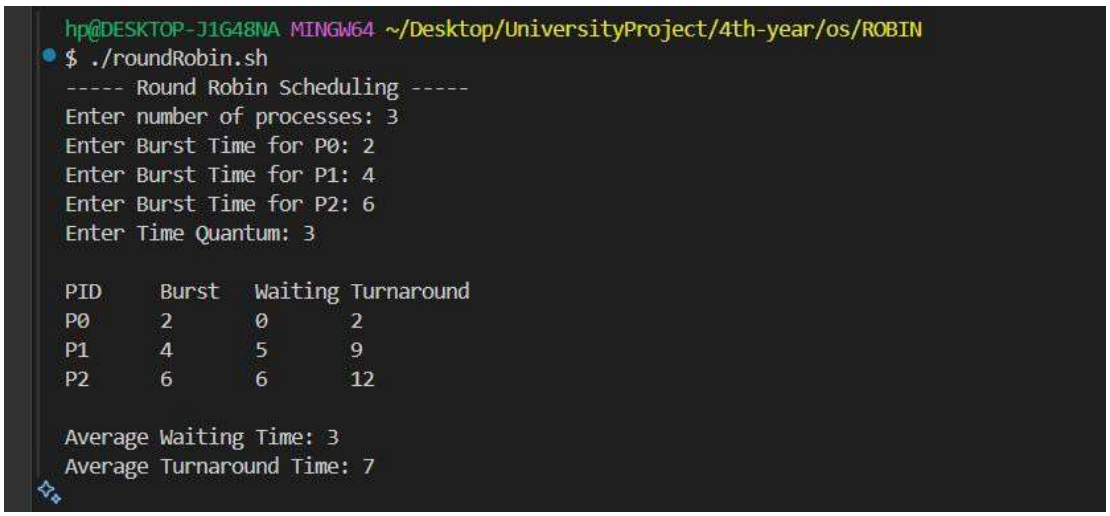
```
  echo -e "\nPID\tBurst\tWaiting\tTurnaround"
  for ((i=0; i<n; i++)); do
    tat[i]=$((wt[i] + burst[i]))
    total_wt=$((total_wt + wt[i]))
    total_tat=$((total_tat + tat[i]))
    echo -e "P${pid[i]}\t${burst[i]}\t${wt[i]}\t${tat[i]}"
  done
```

```
  avg_wt=$((total_wt / n))
avg_tat=$((total_tat / n))
```

```
echo -e "\nAverage Waiting Time: $avg_wt"
echo -e "Average Turnaround Time: $avg_tat"
}
```

```
# call it to run
round_robin
```

When we run the code



5 Multi-Level Queue Scheduling (MLQ)

Description

Multi-Level Queue Scheduling (MLQ) divides the ready queue into multiple queues, each for a different category of processes (e.g., system processes, interactive, batch jobs). Each queue may use a different scheduling algorithm, and priority is assigned to the queues themselves.

Purpose

MLQ is designed to **segregate and prioritize types of tasks**, enabling systems to treat time-sensitive tasks (like UI or real-time processes) differently from background tasks or long-running jobs. It's ideal for complex, multi-user environments.

```
mlq() {
  echo "----- Multi-Level Queue Scheduling ----------"
  echo "Assuming 2 Queues:"
  echo "1. System Processes (High Priority - FCFS)"
  echo "2. User Processes (Low Priority - FCFS)"
```

```
  echo -n "Enter number of system processes: "
  read s
  echo -n "Enter number of user processes: "
  read u
```

```
  echo "Enter burst times for system processes:"
  for ((i=0; i<s; i++)); do
    echo -n "P$i: "; read sbt[$i]
  done
```

```
  echo "Enter burst times for user processes:"
  for ((i=0; i<u; i++)); do
    echo -n "P$((i+s)): "; read ubt[$i]
  done
```

```
  echo -e "\n-- Executing System Processes (FCFS) --"
  time=0
  for ((i=0; i<s; i++)); do
    wt=$time
    time=$((time + sbt[i]))
    tat=$time
    echo "System P$i - Waiting: $wt, Turnaround: $tat"
  done
```

```
  echo -e "\n-- Executing User Processes (FCFS) --"
  for ((i=0; i<u; i++)); do
    wt=$time
    time=$((time + ubt[i]))
    tat=$time
    echo "User P$((i+s)) - Waiting: $wt, Turnaround: $tat"
  done
}
```

```
# Call the function to run
mlq
```

When we run the code here is the output

```
hp@DESKTOP-J1G48NA MINGW64 ~/Desktop/UniversityProject/4th-year/os/MLQ
$ ./mlq.sh
----- Multi-Level Queue Scheduling -----
Assuming 2 Queues:
1. System Processes (High Priority - FCFS)
2. User Processes (Low Priority - FCFS)
Enter number of system processes: 3
Enter number of user processes: 2
Enter burst times for system processes:
P0: 1
P1: 3
P2: 4
Enter burst times for user processes:
P3: 5
P4: 6

-- Executing System Processes (FCFS) --
P4: 6

P4: 6

-- Executing System Processes (FCFS) --
System P0 - Waiting: 0, Turnaround: 1
System P1 - Waiting: 1, Turnaround: 4
P4: 6
P4: 6

-- Executing System Processes (FCFS) --
System P0 - Waiting: 0, Turnaround: 1
System P1 - Waiting: 1, Turnaround: 4
System P2 - Waiting: 4, Turnaround: 8
P4: 6
```


```
P4: 6
P4: 6

-- Executing System Processes (FCFS) --
System P0 - Waiting: 0, Turnaround: 1
System P1 - Waiting: 1, Turnaround: 4
System P2 - Waiting: 4, Turnaround: 8
```


```
-- Executing User Processes (FCFS) --
User P3 - Waiting: 8, Turnaround: 13
User P4 - Waiting: 13, Turnaround: 19
```

6. Multi-Level Feedback Queue Scheduling

Description:
Unlike Multi-Level Queue, this algorithm allows processes to move between queues based on behavior and age. Shorter jobs are promoted to faster queues.

Purpose:
Balances responsiveness and efficiency. Good for systems with mixed job types.

Shell Script Code (Simplified Simulation):

```
mlfq() {
  echo "----- Multi-Level Feedback Queue Scheduling----------"
  echo -n "Enter number of processes: "
  read n

  for ((i=0; i<n; i++)); do
    echo -n "Enter Burst Time for P$i: "
    read bt
    burst[$i]=$bt
    remaining[$i]=$bt
    pid[$i]=$i
    queue[$i]=0
  done
```

```
  tq=(4 8)  # Two queues with different time quantums
  level=0
  completed=0
  time=0
```

```
  while ((completed < n)); do
    progress=0
    for ((i=0; i<n; i++)); do
      if ((remaining[i] > 0 && queue[i] == level)); then
        progress=1
        q=${tq[level]}
        if ((remaining[i] > q)); then
          time=$((time + q))
          remaining[i]=$((remaining[i] - q))
          ((queue[i]++))
        else
          time=$((time + remaining[i]))
          wt[i]=$((time - burst[i]))
          remaining[i]=0
          ((completed++))
        fi
      fi
    done
    ((level++))
    if ((level > 1)); then level=0; fi
    if ((progress == 0)); then break; fi
  done
```

```
  echo -e "\nPID\tBurst\tWaiting\tTurnaround"
  total_wt=0
  total_tat=0
  for ((i=0; i<n; i++)); do
    tat[i]=$((wt[i] + burst[i]))
    total_wt=$((total_wt + wt[i]))
    total_tat=$((total_tat + tat[i]))
    echo -e "P${pid[i]}\t${burst[i]}\t${wt[i]}\t${tat[i]}"
  done
```

```
  avg_wt=$((total_wt / n))
avg_tat=$((total_tat / n))
```

```
echo -e "\nAverage Waiting Time: $avg_wt"
echo -e "Average Turnaround Time: $avg_tat"
}
```

```
# call function to run
mlfq
```

Here is the output of the code when we run it