Ex. No.: 6a)
Date:

FIRST COME FIRST SERVE

Aim:

To implement First-come First- serve (FCFS) scheduling technique

Algorithm:

1. Get the number of processes from the user.
2. Read the process name and burst time.
3. Calculate the total process time.
4. Calculate the total waiting time and total turnaround time for each process 5. Display the process name & burst time for each process. 6. Display the total waiting time, average waiting time, turnaround time

Program Code:

```
echo "Enter the number of process:"; read n

for ((i=0; i<n; i++)); do
    echo "Enter burst time for process $i:"; read bt[i]
done

wt[0]=0; tat[0]=${bt[0]}
for ((i=1; i<n; i++)); do
    wt[i]=$((wt[i-1] + bt[i-1]))
    tat[i]=$((wt[i] + bt[i]))
done

echo -e "Process\tBurst Time\tWaiting Time\tTurn Around Time"
for ((i=0; i<n; i++)); do
    echo -e "$i\t${bt[i]}\t\t${wt[i]}\t\t${tat[i]}"
    twt=$((twt + wt[i]))
    ttat=$((ttat + tat[i]))
done

echo "Average waiting time is: $(echo "scale=1; $twt / $n" | bc)"
echo "Average Turn around Time is: $(echo "scale=1; $ttat / $n" | bc)"
```

Sample Output:

```
Enter the number of process:
3
Enter the burst time of the processes:
24 3 3

Process        Burst Time      Waiting Time   Turn Around Time

   0              24               0               24

   1               3              24               27

   2               3              27               30

Average waiting time is:  17.0
Average Turn around Time is:  19.0
```

Result:

The FCFS scheduling algorithm has been successfully implemented, demonstrating the ability to process tasks in the order they arrive.

Ex. No.: 6b)
Date:

SHORTEST JOB FIRST

Aim:

To implement the Shortest Job First (SJF) scheduling technique

Algorithm:

1. Declare the structure and its elements.
2. Get number of processes as input from the user.
3. Read the process name, arrival time and burst time
4. Initialize waiting time, turnaround time & flag of read processes to zero. 5. Sort based on burst time of all processes in ascending order 6. Calculate the waiting time and turnaround time for each process. 7. Calculate the average waiting time and average turnaround time. 8. Display the results.

Program Code:

```
echo "Enter the number of processes:"
read n

declare -a bt
declare -a wt
declare -a tat

echo "Enter the burst time of the processes:"
for ((i = 0; i < n; i++)); do
    read bt[$i]
done

sorted_bt=($(printf "%s\n" "${bt[@]}" | sort -n))

wt[0]=0
tat[0]=${sorted_bt[0]}
total_wt=0
total_tat=${sorted_bt[0]}

for ((i = 1; i < n; i++)); do
    wt[$i]=$((wt[$i - 1] + sorted_bt[$i - 1]))
    tat[$i]=$((wt[$i] + sorted_bt[$i]))
    total_wt=$((total_wt + wt[$i]))
    total_tat=$((total_tat + tat[$i]))
done
avg_wt=$(echo "scale=2; $total_wt / $n" | bc)
avg_tat=$(echo "scale=2; $total_tat / $n" | bc)
```

```
echo "Process  Burst Time  Waiting Time  Turn Around Time"
for ((i = 0; i < n; i++)); do
    echo "   $((i +
1))         ${sorted_bt[$i]}            ${wt[$i]}             ${tat[$i]}"
done

echo "Average waiting time is: $avg_wt"
echo "Average Turn Around Time is: $avg_tat"
```

Sample Output:

```
Enter the number of process:
4
Enter the burst time of the processes:
8 4 9 5

Process        Burst Time      Waiting Time   Turn Around Time

   2              4               0              4
   4              5               4              9
   1              8               9             17
   3              9              17             26

Average waiting time is: 7.5
Average Turn Around Time is: 13.0
```

Result:

The SJF scheduling algorithm has been successfully implemented, demonstrating the ability to prioritize and execute tasks based on their shortest execution time

Ex. No.: 6c)
Date:

PRIORITY SCHEDULING

Aim:

To implement priority scheduling technique

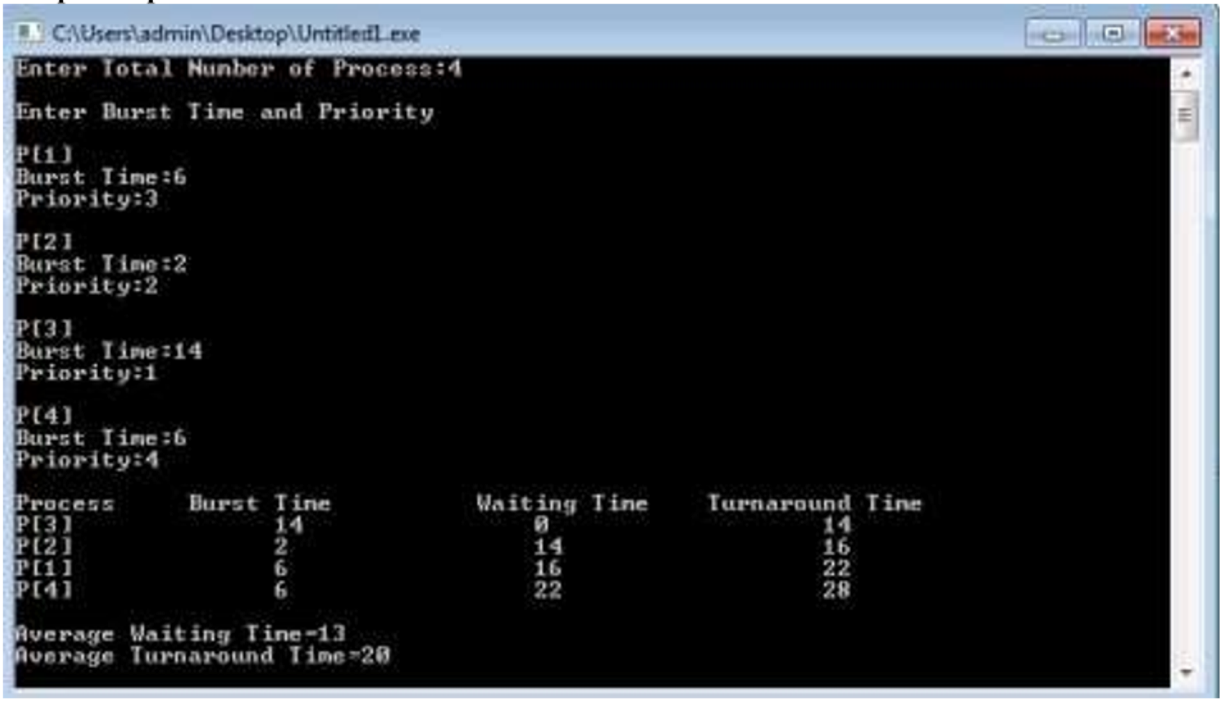
Algorithm:

1. Get the number of processes from the user.
2. Read the process name, burst time and priority of process.
3. Sort based on burst time of all processes in ascending order based priority
4. Calculate the total waiting time and total turnaround time for each process
5. Display the process name & burst time for each process.
6. Display the total waiting time, average waiting time, turnaround time

Program Code:

```
# Read the number of processes
echo -n "Enter number of processes: "
read n

# Declare arrays
declare -a burst_time priority waiting_time turnaround_time process

# Read burst time and priority for each process
for ((i=0; i<n; i++)); do
    process[i]=$((i+1))  # Process ID
    echo -e "P[$((i+1))]\nBurst Time: "
    read burst_time[i]
    echo -n "Priority: "
    read priority[i]
done

# Sort processes by priority (lower number = higher priority)
for ((i=0; i<n-1; i++)); do
    for ((j=i+1; j<n; j++)); do
        if (( priority[i] > priority[j] )); then
            # Swap priority
            temp=${priority[i]}
            priority[i]=${priority[j]}
            priority[j]=$temp

            # Swap burst time
            temp=${burst_time[i]}
            burst_time[i]=${burst_time[j]}
            burst_time[j]=$temp

            # Swap process ID
            temp=${process[i]}
            process[i]=${process[j]}
            process[j]=$temp
        fi
    done
done

# Calculate waiting time
waiting_time[0]=0
for ((i=1; i<n; i++)); do
    waiting_time[i]=$((waiting_time[i-1] + burst_time[i-1]))
done

# Calculate turnaround time
for ((i=0; i<n; i++)); do
    turnaround_time[i]=$((waiting_time[i] + burst_time[i]))
done

# Display results
echo -e "\nProcess\tBurst Time\tWaiting Time\tTurnaround Time"
total_wt=0
total_tat=0
for ((i=0; i<n; i++)); do
    echo -e
"P[${process[i]}]\t${burst_time[i]}\t\t${waiting_time[i]}\t\t${turnaround_time[i]}"
    total_wt=$((total_wt + waiting_time[i]))
    total_tat=$((total_tat + turnaround_time[i]))
done

# Calculate and print average times
avg_wt=$(echo "scale=2; $total_wt / $n" | bc)
avg_tat=$(echo "scale=2; $total_tat / $n" | bc)
echo -e "\nAverage Waiting Time: $avg_wt"
echo -e "Average Turnaround Time: $avg_tat"
```

Sample Output:



```
Enter Total Number of Process:4

Enter Burst Time and Priority

P[1]
Burst Time:6
Priority:3

P[2]
Burst Time:2
Priority:2

P[3]
Burst Time:14
Priority:1

P[4]
Burst Time:6
Priority:4

Process      Burst Time          Waiting Time    Turnaround Time
P[3]            14                  0                  14
P[2]            2                   14                 16
P[1]            6                   16                 22
P[4]            6                   22                 28

Average Waiting Time=13
Average Turnaround Time=20
```

Result:

The priority scheduling algorithm has been successfully implemented, demonstrating the ability to execute tasks based on assigned priority levels.

Ex. No.: 6d)

Date

ROUND ROBIN SCHEDULING

Aim:

To implement the Round Robin (RR) scheduling technique

Algorithm:

1. Declare the structure and its elements.
2. Get number of processes and Time quantum as input from the user.
3. Read the process name, arrival time and burst time
4. Create an array **rem_bt[]** to keep track of remaining burst time of processes which is initially copy of bt[] (burst times array)
5. Create another array **wt[]** to store waiting times of processes. Initialize this array as 0. 6. Initialize time : t = 0
7. Keep traversing the all processes while all processes are not done. Do following for i'th process if it is not done yet.

```
a- If rem_bt[i] > quantum
(i) t = t + quantum
(ii) bt_rem[i] -= quantum;
b- Else // Last cycle for this process
(i) t = t + bt_rem[i];
(ii) wt[i] = t - bt[i]
(iii) bt_rem[i] = 0; // This process is over
```

8. Calculate the waiting time and turnaround time for each process.
9. Calculate the average waiting time and average turnaround time.
10. Display the results.

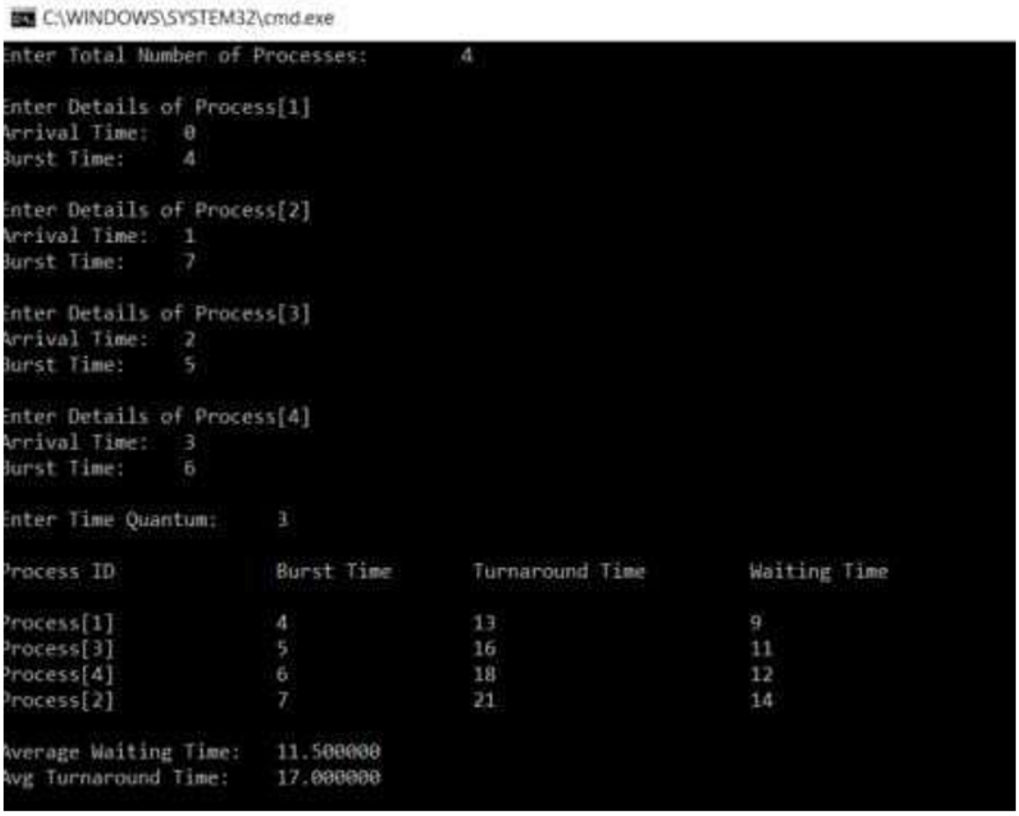
Program Code:

```
echo -n "Enter Total Number of Processes: "
read n

declare -a bt at wt tat remaining_bt

for ((i=0; i<n; i++))
do
    echo "Enter Details of Process[$((i+1))]"
    echo -n "Arrival Time: "
    read at[i]
    echo -n "Burst Time: "
    read bt[i]
    remaining_bt[i]=${bt[i]}
done
```

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

time=0
done_processes=0

# Initialize waiting time array
for ((i=0; i<n; i++))
do
    wt[i]=0
done

while ((done_processes < n))
do
    for ((i=0; i<n; i++))
    do
      if ((remaining_bt[i] > 0))
        then
            if ((remaining_bt[i] > tq))
            then
                time=$((time + tq))
                remaining_bt[i]=$((remaining_bt[i] - tq))
            else
                time=$((time + remaining_bt[i]))
                wt[i]=$((time - bt[i] - at[i]))
                remaining_bt[i]=0
                ((done_processes++))
            fi
         fi
    done
done

# Calculate turnaround time
for ((i=0; i<n; i++))
do
    tat[i]=$((bt[i] + wt[i]))
done

# Display results
echo -e "\nProcess ID\tBurst Time\tTurnaround Time\tWaiting Time"
total_wt=0
total_tat=0

for ((i=0; i<n; i++))
do
```

```
    echo -e "Process[$((i+1))]\t${bt[i]}\t\t${tat[i]}\t\t${wt[i]}"
    total_wt=$((total_wt + wt[i]))
    total_tat=$((total_tat + tat[i]))
done

avg_wt=$(echo "scale=2; $total_wt / $n" | bc)
avg_tat=$(echo "scale=2; $total_tat / $n" | bc)

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

Sample Output



```
Enter Total Number of Processes:        4

Enter Details of Process[1]
Arrival Time:   0
Burst Time:     4

Enter Details of Process[2]
Arrival Time:   1
Burst Time:     7

Enter Details of Process[3]
Arrival Time:   2
Burst Time:     5

Enter Details of Process[4]
Arrival Time:   3
Burst Time:     6

Enter Time Quantum:     3

Process ID              Burst Time      Turnaround Time         Waiting Time

Process[1]              4               13                      9
Process[3]              5               16                      11
Process[4]              6               18                      12
Process[2]              7               21                      14

Average Waiting Time:   11.500000
Avg Turnaround Time:    17.000000
```

Result:
The Round Robin scheduling algorithm has been successfully implemented, demonstrating the ability to fairly distribute CPU time among multiple processes using time slicing