OPERATING SYSTEM LAB

Name: Thorat Amey Arun Reg No.: 23MCS1004

LAB EXPERIMENT 6
Scheduling Algorithms

1. Create a thread pass string as your name to the thread and also print your registration number to main thread and print their IDs.

Program:

```
#include <stdio.h>
float avg_waiting_time_sjf=0, avg_turnaround_time_sjf=0;
float avg_waiting_time_fcfs=0, avg_turnaround_time_fcfs=0;

int main() {
   int arrival_time[5] = {0, 2, 4, 7, 3};
   int burst_time[5] = {15, 2, 5, 1, 7};
   int priority[5] = {3, 1, 4, 5, 2};
   int waiting_time[5], turnaround_time[5];
   int completion_time[5], processed[5] = {0};
   int n = 5;
   int choice;

   printf("Select the algorithm:\n");
   printf("1. FCFS\n");
   printf("2. SJF (Non-Preemptive)\n");
   printf("3. To exit\n");

   do{
   scanf("%d", &choice);
   switch (choice) {
      case 1:
        //fcfs(arrival_time, burst_time, n);
      waiting_time[0] = 0;
      for (int i = 1; i < n; i++) {
            waiting_time[i] = waiting_time[i - 1] + burst_time[i - 1];
      }

      for (int i = 0; i < n; i++) {
            turnaround_time[i] = waiting_time[i] + burst_time[i];
      }
      for (int i = 0; i < n; i++) {
```

```
                avg_waiting_time_fcfs += waiting_time[i];
                avg_turnaround_time_fcfs += turnaround_time[i];
        }

        avg_waiting_time_fcfs /= n;
        avg_turnaround_time_fcfs /= n;

        printf("FCFS Scheduling:\n");
        printf("Process\tWaiting Time\tTurnaround Time\n");
        for (int i = 0; i < n; i++) {
                printf("P%d\t%d\t\t%d\n", i + 1, waiting_time[i],
turnaround_time[i]);
        }
        printf("Average Waiting Time: %.2f\n", avg_waiting_time_fcfs);
        printf("Average Turnaround Time: %.2f\n",
avg_turnaround_time_fcfs);
        break;
        case 2:
                        //sjf_non_preemptive(arrival_time, burst_time, n);
        int current_time = 0;
        int remaining_processes = n;

        while (remaining_processes > 0) {
                int shortest_job = -1;
                int shortest_burst = 99999999;

                for (int i = 0; i < n; i++) {
                  if (!processed[i] && arrival_time[i] <= current_time &&
burst_time[i] < shortest_burst) {
                     shortest_burst = burst_time[i];
                     shortest_job = i;
                  }
                }

                if (shortest_job == -1) {
                  current_time++;
                } else {
                  completion_time[shortest_job] = current_time +
burst_time[shortest_job];
```

```
                    waiting_time[shortest_job] = current_time -
arrival_time[shortest_job];
                    turnaround_time[shortest_job] =
completion_time[shortest_job] - arrival_time[shortest_job];
                    current_time = completion_time[shortest_job];
                    processed[shortest_job] = 1;
                    remaining_processes--;
                }
        }


        for (int i = 0; i < n; i++) {
                avg_waiting_time_sjf += waiting_time[i];
                avg_turnaround_time_sjf += turnaround_time[i];
        }

        avg_waiting_time_sjf /= n;
        avg_turnaround_time_sjf /= n;

        printf("SJF (Non-Preemptive) Scheduling:\n");
        printf("Process\tWaiting Time\tTurnaround Time\n");
        for (int i = 0; i < n; i++) {
                printf("P%d\t%d\t\t%d\n", i + 1, waiting_time[i],
turnaround_time[i]);
        }
        printf("Average Waiting Time: %.2f\n", avg_waiting_time_sjf);
        printf("Average Turnaround Time: %.2f\n",
avg_turnaround_time_sjf);
        break;
        default:
                if(avg_waiting_time_sjf<avg_waiting_time_fcfs)
                        printf("\nSJF has low waiting time");
                else
                        printf("\nFCFS has low waiting time");
                if(avg_turnaround_time_sjf<avg_turnaround_time_fcfs)
                        printf("\nSJF has low turn around time");
                else
                        printf("\nFCFS has low turn around time");
                printf("\nExit\n");
```

```
                break;
        }
        }while(choice!=3);

        return 0;
}
```

Output:

```
student1@student1-VirtualBox:~/Desktop$ gcc lab6a.c -o lab6a
student1@student1-VirtualBox:~/Desktop$ ./lab6a
Select the algorithm:
1. FCFS
2. SJF (Non-Preemptive)
3. To exit
1
FCFS Scheduling:
Process Waiting Time    Turnaround Time
P1      0               15
P2      15              17
P3      17              22
P4      22              23
P5      23              30
Average Waiting Time: 15.40
Average Turnaround Time: 21.40
2
SJF (Non-Preemptive) Scheduling:
Process Waiting Time    Turnaround Time
P1      0               15
P2      14              16
P3      14              19
P4      8               9
P5      20              27
Average Waiting Time: 11.20
Average Turnaround Time: 17.20
3

SJF has low waiting time
SJF has low turn around time
Exit
student1@student1-VirtualBox:~/Desktop$
```

2. Write one single C /C++ program to simulate different scheduling algorithm in OS (functions)- Pre-emptive Algorithms
   - Shortest job remaining first
   - Priority (pre-emptive)
   - Round Robin

   Compare the algorithm and print which is having less waiting time.

OPERATING SYSTEM LAB

Name: Thorat Amey Arun Reg No.: 23MCS1004

Sample Data:

Consider the ready queue of OS, the process are present and maintained with their arrival time and expected burst time for execution. Some processes have priority which is also given. Consider the required data to run different scheduling algorithms and analyse the result with respect to average waiting time and turnaround time.

Time quantum = 2ms

S.no	Process ID	Arrival time	Expected Burst time	Priority
1	P1	0	15	3
2	P2	2	2	1
3	P3	4	5	4
4	P4	7	1	5
5	P5	3	7	2

Program:

```
#include <stdio.h>
#include <stdlib.h>
// Structure to represent a process
struct Process {
    int pid;
    int arrival_time;
    int burst_time;
    int priority;
    int remaining_time;
};

// Function to calculate waiting time for each process
void calculateWaitingTime(struct Process processes[], int n, int
waiting_time[]) {
    int remaining_burst_time[n];
    for (int i = 0; i < n; i++) {
        remaining_burst_time[i] = processes[i].burst_time;
    }
    int completed = 0;
    int time = 0;
    while (completed < n) {
        int shortest = -1;
        for (int i = 0; i < n; i++) {
```

```
            if (processes[i].arrival_time <= time && remaining_burst_time[i]
> 0 && (shortest == -1 || remaining_burst_time[i] <
remaining_burst_time[shortest])) {
                shortest = i;
            }
        }
        if (shortest == -1) {
            time++;
        } else {
            remaining_burst_time[shortest]--;
            if (remaining_burst_time[shortest] == 0) {
                completed++;
                waiting_time[shortest] = time + 1 -
processes[shortest].arrival_time - processes[shortest].burst_time;
            }
            time++;
        }
    }
}

// Function to calculate turnaround time for each process
void calculateTurnaroundTime(struct Process processes[], int n, int
waiting_time[], int turnaround_time[]) {
    for (int i = 0; i < n; i++) {
        turnaround_time[i] = processes[i].burst_time + waiting_time[i];
    }
}

// Function to calculate the average waiting time
float calculateAverageWaitingTime(int waiting_time[], int n) {
    float total_waiting_time = 0;
    for (int i = 0; i < n; i++) {
        total_waiting_time += waiting_time[i];
    }
    return total_waiting_time / n;
}

int main() {
    int n;
```

```
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    struct Process processes[n];
    // Input process data
    for (int i = 0; i < n; i++) {
        processes[i].pid = i + 1;
        printf("Enter arrival time for Process P%d: ", i + 1);
        scanf("%d", &processes[i].arrival_time);
        printf("Enter burst time for Process P%d: ", i + 1);
        scanf("%d", &processes[i].burst_time);
        printf("Enter priority for Process P%d: ", i + 1);
        scanf("%d", &processes[i].priority);
        processes[i].remaining_time = processes[i].burst_time;
    }
    int waiting_time[n];
    int turnaround_time[n];
    // Calculate waiting time for SJRF
    calculateWaitingTime(processes, n, waiting_time);
    // Calculate turnaround time for SJRF
    calculateTurnaroundTime(processes, n, waiting_time,
turnaround_time);
    // Print waiting and turnaround times for each process
    printf("\nProcess\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("P%d\t%d\t\t%d\n", processes[i].pid, waiting_time[i],
turnaround_time[i]);
    }
    // Calculate and print average waiting time for SJRF
    float avg_waiting_time_sjrf =
calculateAverageWaitingTime(waiting_time, n);
    printf("Average Waiting Time for SJRF: %.2f\n",
avg_waiting_time_sjrf);
    // Reset waiting time array
    for (int i = 0; i < n; i++) {
        waiting_time[i] = 0;
    }
     // Priority (Preemptive) Scheduling Algorithm
    int completed = 0;
    int time = 0;
```

```
    while (completed < n) {
        int highest_priority = -1;
        for (int i = 0; i < n; i++) {
            if (processes[i].arrival_time <= time &&
processes[i].remaining_time > 0 && (highest_priority == -1 ||
processes[i].priority < processes[highest_priority].priority)) {
                highest_priority = i;
            }
        }
        if (highest_priority == -1) {
            time++;
        } else {
            processes[highest_priority].remaining_time--;
            if (processes[highest_priority].remaining_time == 0) {
                completed++;
                waiting_time[highest_priority] = time + 1 -
processes[highest_priority].arrival_time -
processes[highest_priority].burst_time;
            }
            time++;
        }
    }
    // Calculate turnaround time for Priority (Preemptive)
    calculateTurnaroundTime(processes, n, waiting_time,
turnaround_time);
    // Calculate and print average waiting time for Priority (Preemptive)
    float avg_waiting_time_priority =
calculateAverageWaitingTime(waiting_time, n);
    printf("Average Waiting Time for Priority (Preemptive): %.2f\n",
avg_waiting_time_priority);
    // Reset waiting time array
    for (int i = 0; i < n; i++) {
        waiting_time[i] = 0;
    }
    // Round Robin Scheduling Algorithm (with time quantum = 2ms)
    int time_quantum = 2;
    int remaining_burst_time[n];
    for (int i = 0; i < n; i++) {
        remaining_burst_time[i] = processes[i].burst_time;
```

```
    }
    completed = 0;
    time = 0;
    while (completed < n) {
        for (int i = 0; i < n; i++) {
            if (processes[i].arrival_time <= time && remaining_burst_time[i]
> 0) {
                if (remaining_burst_time[i] <= time_quantum) {
                    time += remaining_burst_time[i];
                    remaining_burst_time[i] = 0;
                    completed++;
                    waiting_time[i] = time - processes[i].arrival_time -
processes[i].burst_time;
                } else {
                    time += time_quantum;
                    remaining_burst_time[i] -= time_quantum;
                }
            }
        }
    }
    // Calculate turnaround time for Round Robin
    calculateTurnaroundTime(processes, n, waiting_time,
turnaround_time);
    // Calculate and print average waiting time for Round Robin
    float avg_waiting_time_rr =
calculateAverageWaitingTime(waiting_time, n);
    printf("Average Waiting Time for Round Robin: %.2f\n",
avg_waiting_time_rr);
    // Compare the algorithms and print the result
    if (avg_waiting_time_sjrf <= avg_waiting_time_priority &&
avg_waiting_time_sjrf <= avg_waiting_time_rr) {
        printf("SJRF has the lowest average waiting time.\n");
    } else if (avg_waiting_time_priority <= avg_waiting_time_sjrf &&
avg_waiting_time_priority <= avg_waiting_time_rr) {
        printf("Priority (Preemptive) has the lowest average waiting
time.\n");
    } else {
        printf("Round Robin has the lowest average waiting time.\n");
    }
```

```
    return 0;
}
```

Output:

```
vboxuser@Ubuntu:~$ cd Desktop
vboxuser@Ubuntu:~/Desktop$ gcc lab6b.c -o lab6b
vboxuser@Ubuntu:~/Desktop$ ./lab6b
Enter the number of processes: 5
Enter arrival time for Process P1: 0
Enter burst time for Process P1: 15
Enter priority for Process P1: 3
Enter arrival time for Process P2: 2
Enter burst time for Process P2: 2
Enter priority for Process P2: 1
Enter arrival time for Process P3: 4
Enter burst time for Process P3: 5
Enter priority for Process P3: 4
Enter arrival time for Process P4: 7
Enter burst time for Process P4: 1
Enter priority for Process P4: 5
Enter arrival time for Process P5: 3
Enter burst time for Process P5: 7
Enter priority for Process P5: 2

Process Waiting Time    Turnaround Time
P1      15              30
P2      0               2
P3      1               6
P4      0               1
P5      7               14
Average Waiting Time for SJRF: 4.60
Average Waiting Time for Priority (Preemptive): 10.40
Average Waiting Time for Round Robin: 8.40
SJRF has the lowest average waiting time.
vboxuser@Ubuntu:~/Desktop$
```