CSE 316-OPERATING SYSTEMS

LOVELY
PROFESSIONAL
UNIVERSITY

Submitted to: RICHA SHARMA

Submitted by: M.SIVA NAGI REDDY

Registration no.: 12201812

Section : K22ZC

Roll no:32

QUESTION:

Design a scheduler with multilevel queue having two queues which will schedule the processes on the basis of pre-emptive shortest remaining processing time first algorithm (SROT) followed by a scheduling in which each process will get 2 units of time to execute. Also note that queue 1 has higher priority than queue 2. Consider the following set of processes (for reference)with their arrival times and the CPU burst times in milliseconds.

Process	Arrival-Time	Burst-Time
P1	0	5
P2	1	3
P3	2	3
P4	4	1

Calculate the average turnaround time and average waiting time for each process. The input for number of processes and their arrival time, burst time should be given by the user.

ANSWER:-

```
#include <stdio.h>
#include <stdlib.h>

typedef struct Process {
    int pid;
    int arrival_time;
    int burst_time;
    int remaining_burst_time;
    int completion_time;
    int turnaround_time;
    int waiting_time;
} Process;

void calculateTurnaroundTime(Process* processes, int n) {
    for (int i = 0; i < n; i++) {
        processes[i].turnaround_time =
processes[i].completion_time - processes[i].arrival_time;
    }
}

void calculateWaitingTime(Process* processes, int n) {
```

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

void schedule(Process* processes, int n) {
    int time = 0;
    int quantum = 2;

    // Queue 1 (SROT)
    while (1) {
        int shortest_remaining = -1;
        for (int i = 0; i < n; i++) {
        if (processes[i].arrival_time <= time &&
processes[i].remaining_burst_time > 0) {
            if (shortest_remaining == -1 ||
processes[i].remaining_burst_time <
processes[shortest_remaining].remaining_burst_time) {
                shortest_remaining = i;
            }
        }
    }
```

```
        if (shortest_remaining == -1) {
            break;
        }

        if (processes[shortest_remaining].remaining_burst_time
<= quantum) {
            time +=
processes[shortest_remaining].remaining_burst_time;
            processes[shortest_remaining].remaining_burst_time
= 0;
            processes[shortest_remaining].completion_time =
time;
        } else {
            time += quantum;
            processes[shortest_remaining].remaining_burst_time -
= quantum;
        }
    }

    // Queue 2 (Fixed time slice of 2 units)
    for (int i = 0; i < n; i++) {
        if (processes[i].remaining_burst_time > 0) {
```

```
            time += quantum;
            processes[i].remaining_burst_time -= quantum;
        }
    }
}

int main() {
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);

    Process* processes = (Process*)malloc(n * sizeof(Process));

    printf("Enter arrival time and burst time for each process:\n");
    for (int i = 0; i < n; i++) {
        processes[i].pid = i + 1;
        scanf("%d %d", &processes[i].arrival_time, &processes[i].burst_time);
        processes[i].remaining_burst_time = processes[i].burst_time;
    }
```

```
    schedule(processes, n);

    calculateTurnaroundTime(processes, n);
    calculateWaitingTime(processes, n);

    printf("\nProcess  Turnaround Time  Waiting Time\n");
    for (int i = 0; i < n; i++) {
        printf("P%d\t\t%d\t\t%d\n", processes[i].pid,
processes[i].turnaround_time, processes[i].waiting_time);
    }

    free(processes);

    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct Process {
    int pid;
    int arrival_time;
    int burst_time;
    int remaining_burst_time;
    int completion_time;
    int turnaround_time;
    int waiting_time;
} Process;

void calculateTurnaroundTime(Process* processes, int n) {
    for (int i = 0; i < n; i++) {
        processes[i].turnaround_time = processes[i].completion_time - processes[i].arrival_time;
    }
}

void calculateWaitingTime(Process* processes, int n) {
    for (int i = 0; i < n; i++) {
        processes[i].waiting_time = processes[i].turnaround_time - processes[i].burst_time;
    }
}

void schedule(Process* processes, int n) {
    int time = 0;
    int quantum = 2;

    // Queue 1 (SROT)
    while (1) {
        int shortest_remaining = -1;
        for (int i = 0; i < n; i++) {
        if (processes[i].arrival_time <= time && processes[i].remaining_burst_time > 0) {
        if (shortest_remaining == -1 || processes[i].remaining_burst_time < processes[shortest_remaining].remaining_burst_time) {
                shortest_remaining = i;
                }
            }
        }

        if (shortest_remaining == -1) {
            break;
        }

        if (processes[shortest_remaining].remaining_burst_time <= quantum) {
            time += processes[shortest_remaining].remaining_burst_time;
            processes[shortest_remaining].remaining_burst_time = 0;
            processes[shortest_remaining].completion_time = time;
        } else {
```

```
            processes[shortest_remaining].remaining_burst_time = 0;
            processes[shortest_remaining].completion_time = time;
        } else {
            time += quantum;
            processes[shortest_remaining].remaining_burst_time -= quantum;
        }
    }

    // Queue 2 (Fixed time slice of 2 units)
    for (int i = 0; i < n; i++) {
        if (processes[i].remaining_burst_time > 0) {
            time += quantum;
            processes[i].remaining_burst_time -= quantum;
        }
    }
}

int main() {
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);

    Process* processes = (Process*)malloc(n * sizeof(Process));

    printf("Enter arrival time and burst time for each process:\n");
    for (int i = 0; i < n; i++) {
```

```
    Process* processes = (Process*)malloc(n * sizeof(Process));

    printf("Enter arrival time and burst time for each process:\n");
    for (int i = 0; i < n; i++) {
        processes[i].pid = i + 1;
        scanf("%d %d", &processes[i].arrival_time, &processes[i].burst_time);
        processes[i].remaining_burst_time = processes[i].burst_time;
    }

    schedule(processes, n);

    calculateTurnaroundTime(processes, n);
    calculateWaitingTime(processes, n);

    printf("\nProcess  Turnaround Time  Waiting Time\n");
    for (int i = 0; i < n; i++) {
        printf("P%d\t\t%d\t\t%d\n", processes[i].pid, processes[i].turnaround_time, processes[i].waiting_time);
    }

    free(processes);

    return 0;
}
```

OUTPUT:-

```
Enter the number of processes: 4
Enter arrival time and burst time for each process:
0
1
2
4
5
3
3
1
Process  Turnaround Time  Waiting Time
P1       1        0
P2       -2       -6
P3       -5       -8
P4       -3       -4
```