Submitted By :

Name : Aman Ahmed

Reg. No : 11801727

Section : K18EN

Roll No :30

Group : 2

Email :copininja889@gmail.com

GitHub Link : https://github.com/mishu11/OS_PROJECT

Submitted To :

Priya Virdi

<u>INDEX</u>

QUE.8 Sudesh Sharma is a Linux expert who wants to have an online system where he can handle student queries. Since there can be multiple requests at any time he wishes to dedicate a fixed amount of time to every request so that everyone gets a fair share of his time. He will log into the system from 10am to 12am only. He wants to have separate requests queues for students and faculty. Implement a strategy for the same. The summary at the end of the session should include the total time he spent on handling queries and average query time.

CODE

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

struct process_Struct {
    char process_name[20];
    int arrival_time, burst_time, completion_time, remaining;
}temp_Struct;


void faculty_Queue(int no_of_process) {

    int count, arrival_Time, burst_Time, quantum_time;
    struct process_Struct faculty_Process[no_of_process];

    for(count = 0; count < no_of_process; count++) {
        printf("Enter the details of Process[%d]", count+1);
        puts("");
        printf("Process Name : ");
        scanf("%s ", faculty_Process[count].process_name);

        printf("Arrival Time : ");
        scanf("%d", &faculty_Process[count].arrival_time);

        printf("Burst Time : ");
        scanf("%d ", &faculty_Process[count].burst_time);
        puts("");
    }
    printf("Now, enter the quantum time for FACULTY queue : ");
    scanf("%d", &quantum_time);
```

```
    for(count = 0; count < no_of_process; count++) {
        for(int x = count +1; x < count; x++){
            if(faculty_Process[count].arrival_time >
faculty_Process[x].arrival_time) {
                temp_Struct = faculty_Process[count];
                faculty_Process[count] = faculty_Process[x];
                faculty_Process[x] = temp_Struct;
            }
        }
    }

    for(count = 0; count < no_of_process; count++) {
        faculty_Process[count].remaining =
faculty_Process[count].burst_time;
        faculty_Process[count].completion_time = 0;
    }

    int total_time, queue, round_robin[20];
    total_time = 0;
    queue = 0;
    round_robin[queue] = 0;


    int flag, x, n, z, waiting_time = 0;
    do {
        for(count = 0; count < no_of_process; count++){
            if(total_time >= faculty_Process[count].arrival_time){
                z = 0;
                for(x = 0; x <= queue; x++) {
                    if(round_robin[x] == count) {
                        z++;
                    }
                }
                if(z == 0) {
                    queue++;
                    round_robin[queue] == count;
                }
            }
        }

        if(queue == 0) {
            n = 0;
        }
        if(faculty_Process[n].remaining == 0) {
            n++ ;
        }
```

```
        if(n > queue) {
            n = (n - 1) % queue;
        }
        if(n <= queue) {
            if(faculty_Process[n].remaining > 0) {
                if(faculty_Process[n].remaining < quantum_time){
                    total_time += faculty_Process[n].remaining;
                    faculty_Process[n].remaining = 0;
                }else {
                    total_time += quantum_time;
                    faculty_Process[n].remaining -= quantum_time;
                }
                faculty_Process[n].completion_time = total_time;
            }
            n++;
        }
        flag = 0;

        for(count = 0; count < no_of_process; count++) {
            if(faculty_Process[count].remaining > 0) {
                flag++;
            }
        }
    }while(flag != 0);


    puts("\n\t\t\t*****");
    puts("\t\t\t   ROUND ROBIN ALGORITHM OUTPUT   ");
    puts("\t\t\t*****\n");
    printf("\n|\tProcess Name\t  |\tArrival Time\t  |\tBurst Time\t
|\tCompletion Time  \t|\n");

    for(count = 0; count < no_of_process; count++){
        waiting_time = faculty_Process[count].completion_time -
faculty_Process[count].burst_time -
faculty_Process[count].arrival_time;

        printf("\n|\t  %s\t    |\t  %d\t   |\t  %d\t   |\t  %d\t
|\n", faculty_Process[count].process_name,
faculty_Process[count].arrival_time,
faculty_Process[count].burst_time,
faculty_Process[count].completion_time);
    }

}
```

```c
void student_Queue(int no_of_process) {

    int count, arrival_Time, burst_Time, quantum_time;
    struct process_Struct student_Process[no_of_process];

    for(count = 0; count < no_of_process; count++) {
        printf("Enter the details of Process[%d]", count+1);
        puts("");
        printf("Process Name : ");
        scanf("%s", student_Process[count].process_name);

        printf("Arrival Time : ");
        scanf("%d", &student_Process[count].arrival_time);

        printf("Burst Time : ");
        scanf("%d", &student_Process[count].burst_time);
    }
    printf("Now, enter the quantum time for STUDENT queue : ");
    scanf("%d", &quantum_time);


    // if the ARRIVAL time is same then scheduling is based on FCFS.
    for(count = 0; count < no_of_process; count++) {
        for(int x = count +1; x < count; x++){
            if(student_Process[count].arrival_time >
student_Process[x].arrival_time) {
                temp_Struct = student_Process[count];
                student_Process[count] = student_Process[x];
                student_Process[x] = temp_Struct;
            }
        }
    }

    for(count = 0; count < no_of_process; count++) {
        student_Process[count].remaining =
student_Process[count].burst_time;
        student_Process[count].completion_time = 0;
    }

    int total_time, queue, round_robin[20];
    total_time = 0;
    queue = 0;
    round_robin[queue] = 0;
}
```

```c
int main(int argc, char const *argv[]) {
    int select_queue, no_of_process;

    puts("Please choose a queue to post your query : ");
    puts("1. FACULTY queue.");
    puts("2. STUDENT queue.");
    printf("> ");
    scanf("%d", &select_queue);

    switch(select_queue) {
        case 1 :
                printf("Enter number of process for FACULTY queue : 
");
                scanf("%d", &no_of_process);

                faculty_Queue(no_of_process);

                break;

        case 2 :
                printf("Enter number of process for STUDENT queue : 
");
                scanf("%d", &no_of_process);

                student_Queue(no_of_process);

                break;

        default :
                printf("Please selet the correct option by running 
the program again.");
    }

    return 0;
}
```

Algorithm:

- **Round Robin** is a CPU scheduling algorithm where each process is assigned a fixed time slot in a cyclic way. ... One of the most commonly used **technique** in CPU scheduling as a core. It is preemptive as processes are assigned CPU only for a fixed slice of time at most.

COMPLEXITY

- The time **complexity** of **round-robin** scheduling algorithms is O(1). It is easy to realize and is suitable to use in high-speed.

Activity on github:

- I have uploaded the project on the GitHub. And I have also done the revisions of the project on the GitHub as my project is going on.
- I have made more than 5 revisions on that project and also uploaded the documented description file with it.

SCREENSHOTS

INPUT:

```
C:\Users\AMAN AHMED\Desktop\Untitled1.exe
Please choose a queue to post your query :
1. FACULTY queue.
2. STUDENT queue.
> 1
Enter number of process for FACULTY queue : 2
Enter the details of Process[1]
Process Name : p1
Arrival Time : 0
Burst Time : 7

Enter the details of Process[2]
Process Name : p2
Arrival Time : 3
Burst Time : 5

Now, enter the quantum time for FACULTY queue : 6
```

OUTPUT:

```
Please choose a queue to post your query :
1. FACULTY queue.
2. STUDENT queue.
> 1
Enter number of process for FACULTY queue : 2
Enter the details of Process[1]
Process Name : p1
Arrival Time : 0
Burst Time : 7

Enter the details of Process[2]
Process Name : p2
Arrival Time : 3
Burst Time : 5

Now, enter the quantum time for FACULTY queue : 6

                        ********************************************
                        *****    ROUND ROBIN ALGORITHM OUTPUT    *****
                        ********************************************

|               Process Name  | Arrival Time |  Burst Time |    Completion Time  |

|                         p1       |      0         |       7        |       24        |

|                         p2       |      3         |       5        |       11        |

--------------------------------
Process exited after 24.68 seconds with return value 0
Press any key to continue . . .
```