OPERATING SYSTEM ASSIGNMENT

CSE-316

NAME: UTSAB SEN

ROLL NO: 22

REGISTRATION NO.: 11709492

SECTION: K17TA

Email Address: utsabsen1999@gmail.com

GitHub Link:

Account : https://github.com/UtsabSen

Program : https://github.com/UtsabSen/Multilevel-Queue-Scheduling-Using-Preemptive-SJF-And-Round-Robin

Code: Mention solution code assigned to you

```
#include<iostream>      // Include required header files
#include<stdio.h>
#include<conio.h>
#include<windows.h>
using namespace std;

int total_process, total_time = 0, total_queue, TQ = 2;     // Define global
variables of type integer

struct processes{        // Define structure
    int AT, BT, queue, CT = 0, TAT = 0, WT = 0, fixed_BT;
}*process;       //Define structure variable

int max_queue(){         // Define max_queue function of type integer
    int max = 0;
    for (int i = 0; i < total_process; i++) {
        if (process[i].queue > max){
            max = process[i].queue;      // Return queue processes
        }
    }
    return max;      // Return maximum queue value
}

int high_priority_queue(){       // Define high_priority_queue function of type
integer
    for (int i = 0; i < total_queue; i++) {
        for (int j = 0; j < total_process; j++) {
            if((process[j].queue == i && process[j].BT > 0) && (total_time >=
process[j].AT)){
                return process[j].queue;         // Return processes from the queue
which has higher priority
            }
        }
    }
}

int least_BT(int queue){         // Define least_BT function of type integer
    int small_BT = 0, flag = 0;
    for (int i = 0; i < total_process; i++) {
        if (process[i].BT > 0 && process[i].queue == queue && total_time >=
process[i].AT){      // Check for the minimum burst time of processes
            small_BT = process[i].BT;
            flag = i;
            break;
        }
    }

    for (int i = 0; i < total_process; i++) {
        if (small_BT > process[i].BT && process[i].BT > 0 && process[i].queue ==
queue && total_time >= process[i].AT){      // Check burst time, arrival time of
processes
            small_BT = process[i].BT;
            flag = i;
        }
    }
```

```
    return flag;        // Return process id using flag variable with smallest
burst time
}

int complete_process(){     // Define complete_process function of type integer
    for (int i = 0; i < total_process; i++){
        if (process[i].BT > 0){     // Check if burst time of a process is left or
not
            return 0;       // Return false
        }
    }
    return 1;       // Return True
}

int arrive_process(){       // Define arrive_process function of type integer
    for (int i = 0; i < total_process; i++) {
        if(process[i].BT > 0 && total_time >= process[i].AT){       // Check which
process should arrive next
            return 0;       // Return false
        }
    }
    return 1;       // Return true
}

void process_execution(){       // Define process_execution function of type void

    int t[50] = {0};        // Integer array of size 50 with default value zero
    int c;

    int checkpoint = 1;
    while (!complete_process()){        // Check if any process is complete or not
by checking its burst time       // Call complete_process function
        if (!arrive_process()){     // Check  which process should arrive next
// Call arrive_process function
            int p_id = least_BT(high_priority_queue());     // Store process id of
smallest burst time in p_id variable
            for (int i = 0; i < TQ; i++) {
                process[p_id].BT--;     // Decrement burst time
                total_time++;       // Count the time of process execution
                if(checkpoint == 1){        // Check when process should start
                    c = process[p_id].AT;
                    checkpoint = 0;
                    cout << "Start: " << c << "\t";
                }
                c++;
                cout << "P" << (p_id+1) <<" -> "<< c << " \t";      // Print
process id with process execution time in a form of gantt chart in the console
                t[i] = c;
                Sleep(200);     // Slow down print gantt chart by 20 mili second

                for (int j = 0; j < total_process; j++) {
                    if(j != p_id && total_time > process[j].AT && process[j].BT >
0){       // Check arrival time and burst time of a process
                        process[j].WT++;        // Increment waiting time
                    }
                }

                if (process[p_id].BT == 0){     // Check burst time is zero or not
                    process[p_id].CT = total_time;      // Calculate completion
time
                    process[p_id].TAT = process[p_id].CT - process[p_id].AT;
// Calculate Turn Around Time
                    break;
                }

                if (high_priority_queue() < process[p_id].queue){       // Check
priority
                    break;
```

```
                }
            }
        } else{
            total_time++;        // Increment time of process execution
        }
    }
}

int main() {
    system("color 1F");      // Change console color with blue background and bright
white font color
    char welcome[100] = "\t\t\t\t\t\t\t-:Multilevel queue scheduler:-\n";    //
Heading of the program
    for(int i = 0; i < strlen(welcome); i++){        // Print the heading with gap
of 5 mili second
        cout << welcome[i];
        Sleep(50);
    }

    char question[500] = "\nQuestion: 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.\n";        // Store the question in question character array of size
500
    for(int i = 0; i < strlen(question); i++){       // Print the question with gap
of 0.5 mili second
        cout << question[i];
        Sleep(5);
    }

    char c[100] = "\nEnter total process: ";
    for(int i = 0; i < strlen(c); i++){
        cout << c[i];
        Sleep(50);
    }
    cin >> total_process;        // Take total process as an input from the user
    process = new processes[total_process];

    for (int i = 0; i < total_process; i++) {
        cout << "\nProcess P" << (i+1) << endl;      // Print process id
        cout << "\tP" << (i+1) << " Arrival Time: ";
        cin >> process[i].AT;        // Take arrival time as an input from the user
        cout << "\tP" << (i+1) << " Burst Time: ";
        cin >> process[i].BT;        // Take burst time as an input from the user
        cout << "\tP" << (i+1) << " Queue: ";
        cin >> process[i].queue;         // Take queue number as an input from the
user
        process[i].fixed_BT = process[i].BT;         // Backup burst time
    }

    total_queue = max_queue();       // Call max_queue function and store the
maximum queue in total_queue variable

    system("cls");       // Clear the console screen

    for(int i = 0; i < strlen(welcome); i++){        // Print the heading with gap
of 5 mili second
        cout << welcome[i];
        Sleep(50);
    }
    cout << "\nGantt Chart\n";
    process_execution();         // Call process_execution function and print the
```

```
gantt chart

    cout << "\n\nAll process executed\n";

    cout << "\n\nProcess\t Arrival Time\t Burst Time\t Queue\t Completion Time\t
Turn Around Time\t Waiting Time\t\n\n";        // Print table header

    for (int i = 0; i < total_process; i++) {        // Print all processes process
id, arrival time, burst time, queue, completion time, turn around time, waiting
time
        cout << "  P"<<(i+1) << "\t\t" << process[i].AT << "\t\t" <<
process[i].fixed_BT << "\t  " << process[i].queue <<
             "\t\t" << process[i].CT << "\t\t\t" << process[i].TAT << "\t\t\t" <<
process[i].WT << endl << endl;
    }

    float total_TAT = 0, total_WT = 0;
    float avg_TAT = 0, avg_WT = 0;
    for (int i = 0; i < total_process; i++) {        // Calculate total turn around
time
        total_TAT += process[i].TAT;
    }
    avg_TAT = total_TAT / total_process;        // Calculate average turn around
time

    for (int i = 0; i < total_process; i++) {        // Calculate total waiting time
        total_WT += process[i].WT;
    }
    avg_WT = total_WT / total_process;       // Calculate average waiting time

    cout << "\nAverage Turn Around Time: " << avg_TAT << endl;       // Print
average turn around time
    cout << "\nAverage Waiting Time: " << avg_WT << endl << endl;       // Print
average waiting time

    cout << "\nPress any key to continue...\n\n";       // Take user input for
further details

    getch();        // Take user input but do not store it

    char about[200] = "\n\t\tName: Utsab Sen\n\t\tRegistration No:
11709492\n\t\tRoll No: 22\n\t\tSection: K17TA\n\t\tLOVELY PROFESSIONAL
UNIVERSITY\n\t\t...Thank you...\n";       // Student(My) details
    for(int i = 0; i < strlen(about); i++){      // Print student details with gap
of 1.5 mili second
        cout << about[i];
        Sleep(15);
    }
}
```

1. Explain the problem in terms of operating system concept? (Max 200 word)
Description:

This problem is based on Multilevel Queue Scheduling using Pre-emptive Shortest Job First and Round Robin with Time Quantum of 2. In this problem total number of processes, arrival time (AT), burst time (BT), queue number of each processes is given by the user input. Firstly, we assign the processes in two different queues based on their priority (Smallest number consider as highest priority). To solve the problem firstly we see the arrival time of all the process. After finding the shortest arrival time of the process we put it in ready queue. When two process has the same arrival time, we check for the priority of process as mentioned in the question. When arrival time and priority both will be same for the two processes, we check for the shortest burst time to be executed. For higher priority queue, we use Pre-emptive Shortest Job First and put the processes in a gantt chart and for the lower priority queue, we use Round Robin algorithm with Time Quantum of 2 and put the processes in the gantt chart. We get the Completion Time (CT) when each process completes their process. After getting the Completion Time (CT) we calculate Turn Around Time (TAT) by using the formula TAT = CT – AT and calculate the Waiting Time (WT) by using the formula WT = TAT – BT. After this we have to fine average Turn Around Time (TAT) and average Waiting Time (WT) by dividing the total Time Around Time (TAT) and total Waiting Time (WT) by total number of processes respectively. In this way we can solve Multilevel Queue Scheduling using Pre-emptive Shortest Job First and Round Robin with Time Quantum of 2.

2. Write the algorithm for proposed solution of the assigned problem.

Algorithm:

1. Traverse until all the process gets completely executed.
2. See the Arrival Time (AT) of each process and put the process in ready queue which has shortest arrival time.
    a. If more than one process has same arrival time execute the process with higher priority.
    b. If more than one process has same arrival time as well as has the same priority then execute the process with shortest burst time.
3. Execute the arrived process and decrease Burst Time (BT) according to the Time Quantum.
4. Make Gantt Chart to calculate the Completion Time (CT) for each process.
5. Execute the previous steps until remaining time is zero.
6. Calculate Time Around Time of each process using TAT = CT – AT.
7. Calculate Waiting Time of each process using WT = TAT – BT.
8. Calculate average Turn Around Time and average Waiting Time.

3. Calculate complexity of implemented algorithm. (Student must specify complexity of each line of code along with overall complexity)

Complexity of Pre-emptive Shortest Job First is **O (n log n)**
Complexity of Round Robin is **O (1)**
Complexity of Multilevel Queue Scheduling using Pre-emptive Shortest Job First and Round Robin is **max (O (n log n), O (1))** which is **O (n log n)**

Description (purpose of use):

- According to this algorithm short processes are executed first and longer process executed afterwards.
- In this algorithm longer process has equal time sharing for process execution.
- For short process Pre-emptive Shortest Job First is efficient where as Round Robin is the best algorithm for longer processes. So, overall Multilevel Queue

Scheduler Algorithm is best for both the cases even when short and long processes execute together.

4. Explain all the constraints given in the problem. Attach the code snippet of the implemented constraint.

- Higher priority is considered as lowest number.
- Higher priority should execute first.
- Perform Pre-emptive Shortest Job First algorithm in higher priority queue
- Perform Round Robin in Lower priority queue

5. If you have implemented any additional algorithm to support the solution, explain the need and usage of the same.

Description:

In this program some inbuild system functions are called.

- system("cls") : Clear the console
- system("color 1F") : Colour the console. First letter is for background colour and Second letter is for font colour.

  To see all the following colours: Open Command Prompt and hit enter after typing "color ?"

0 = Black	**8 = Gray**
1 = Blue	**9 = Light Blue**
2 = Green	**A = Light Green**
3 = Aqua	**B = Light Aqua**
4 = Red	**C = Light Red**
5 = Purple	**D = Light Purple**
6 = Yellow	**E = Light Yellow**
7 = White	**F = Bright White**

- Sleep(50) : Makes the program sleep for 5 milli seconds.

6. Explain the boundary conditions of the implemented code.

Description:

- In this program user should not enter Process ID, Arrival Time, Burst Time and Queue as negative value.
- This program is designed for only windows. This program does not run on Linux or Unix system because of some required header files which is not present in Linux and Unix.

7. Explain all the test cases applied on the solution of assigned problem.
Description:

Process	AT	BT	Queue	CT	TAT	WT
P1	0	5	1	5	5	0
P2	1	3	1	9	8	5
P3	2	3	2	12	10	7
P4	4	1	1	6	2	1

Avg TAT = 6.25 Avg WT = 3.25

Process	AT	BT	Queue	CT	TAT	WT
P1	0	8	1	13	13	5
P2	1	4	1	7	6	2
P3	2	6	2	25	23	17
P4	1	2	2	15	14	12
P5	3	1	1	5	2	1
P6	2	4	2	19	17	13

Avg TAT = 12.5 Avg WT = 8.33

Process	AT	BT	Queue	CT	TAT	WT
P1	0	4	2	12	12	8
P2	1	2	1	3	2	0
P3	2	4	1	7	5	1
P4	3	2	2	9	6	4

Avg TAT = 6.25 Avg WT = 3.25

8. Have you made minimum 5 revisions of solution on GitHub?

- YES

GitHub All Commits Link: https://github.com/UtsabSen/Multilevel-Queue-Scheduling-Using-Preemptive-SJF-And-Round-Robin/commits/master