FAIZAN CHOUDHARY

20BCS021

OS LAB

17th February 2022

CODE: (code pasted in this format for readability)

```
#include <iostream>
#include <string.h>
#include <algorithm>
using namespace std;
const int SIZE = 50;

struct process
{
    int pid;
    int burst;
    int arrival;
    int start;
    int completion;
    int waiting;
    int turnaround;
    int response;
};
process pr[SIZE];
int n;

struct Gantt
{
    int idx;
    int start;
    int end;
};
Gantt g[SIZE];
int cnt=0;                          // to count number of indexed processes for Gantt
chart

// ready queue (circular queue) FIFO
int ready_queue[SIZE];
int front=-1, rear=-1;

int current_time=0, time_quantum;
int remaining[SIZE];                  // to store remaining burst time for each process
int temp[SIZE];                       // to store remaining burst times for Gantt chart
bool completed[SIZE] = {false};       // to store if the process is completed or not
int idx;
int num = 0;                          // to store the number of processes completed
double tot_ct = 0, tot_wt =0, tot_tat = 0, tot_rt =0;
```

```
// comparing wrt arrival time
bool compare1 (process &p1, process &p2) {
    return p1.arrival < p2.arrival;
}

// comparing wrt pid
bool compare2 (process &p1, process &p2) {
    return p1.pid < p2.pid;
}

// to insert a process in the ready queue
void insertProcess (int l) {
    if (front == -1)
        front = 0;
    rear = (rear + 1) % SIZE;
    ready_queue[rear] = l;
}

// deleting from ready queue
void executeProcess () {
    if (front == -1)
        return;
    if (front == rear)
        front=rear=-1;
    else
        front = (front + 1) % SIZE;
}

void RR () {
    // sorting wrt arrival times
    sort (pr,pr+n,compare1);
    // inserting first process
    insertProcess(0);
    completed[0] = true;

    // loop until the num of processes executed is equal to no of processes input by user
    while (num != n) {
        // dispatching the process at the front of ready queue
        idx = ready_queue[front];
        executeProcess();

        // if the remaining burst time for a process is equal to the current process at
that idx, update current_time and start time of process
        if (remaining[idx] == pr[idx].burst) {
            pr[idx].start = max (current_time, pr[idx].arrival);
            current_time = pr[idx].start;
        }

        // if the burst time remaining for a process is greater than the time quantum
        if (remaining[idx] > time_quantum) {
            temp[idx] = remaining[idx];
            remaining[idx] -= time_quantum;
            current_time += time_quantum;
        }
```

```
        else {
            // if the process has remaining burst time less than time quantum
            current_time += remaining[idx];
            temp[idx] = remaining[idx];
            remaining[idx] = 0;
            // updating no of processes completed
            num++;

            pr[idx].completion = current_time;
            pr[idx].turnaround = pr[idx].completion - pr[idx].arrival;
            pr[idx].waiting = pr[idx].turnaround - pr[idx].burst;
            pr[idx].response = pr[idx].start - pr[idx].arrival;

            tot_tat += pr[idx].turnaround;
            tot_wt += pr[idx].waiting;
            tot_ct += pr[idx].completion;
            tot_rt += pr[idx].response;
        }

        for (int i=1; i<n; i++) {
            if (remaining[i] > 0 && pr[i].arrival <= current_time && completed[i] ==
false) {
                insertProcess(i);
                completed[i] = true;
            }
        }

        if (remaining[idx] > 0)
            insertProcess(idx);

        // if queue is empty
        if (front == -1) {
            for (int i=1; i<n; i++) {
                if (remaining[i] > 0) {
                    insertProcess(i);
                    completed[i] = true;
                    break;
                }
            }
        }
        // for Gantt chart
        g[cnt].idx = idx;
        if (current_time - time_quantum < 0)
            g[cnt].start = 0;
        else if (temp[idx] < time_quantum)
            g[cnt].start = current_time - time_quantum + 1;
        else
            g[cnt].start = current_time - time_quantum;
        g[cnt].end = current_time+1;
        cnt++;
    }
    g[cnt].end = current_time;
}
```

```
void display () {
    int time = 0;
    sort(pr,pr+n,compare2);
    cout<<"\n\nProcess | Burst Time | Arrival Time | Completion Time | Waiting Time |
Turnaround Time | Response Time\n";
    cout<<"_____
_____\n\n";

    for (int i=0; i<n; i++) {
        printf("   P%d        %2d            %2d                %2d               %2d
        %2d              %2d\n", pr[i].pid, pr[i].burst, pr[i].arrival, pr[i].completion,
pr[i].waiting, pr[i].turnaround, pr[i].response);
    }

    cout<<"_____
_____\n\n";

    printf("\nAverage Completion time: %.2f",tot_ct / (float) n);
    printf("\nAverage Waiting time: %.2f", tot_wt / (float) n);
    printf("\nAverage Turnaround time: %.2f",tot_tat / (float) n);
    printf("\nAverage Response time: %.2f\n",tot_rt / (float) n);
}

void displayGantt () {
    cout<<"\nGantt chart: \n";
    int time = 0;
    for (int i=0; i<cnt; i++) {
        cout<<"| ";
        cout<<"P"<<pr[g[i].idx].pid<<" ";
    }
    cout<<"|\n";
    int i;
    for (i=0; i<cnt; i++) {
        if (g[i].start > 9)
            cout<<g[i].start<<"   ";
        else if (g[i].start <= 9)
            cout<<g[i].start<<"    ";
    }
    cout<<g[i].end<<endl;
}

int main () {
    cout<<"\nFAIZAN CHOUDHARY\n20BCS021\n";
    cout<<"\nRound Robin Scheduling Algorithm\n";

    cout<<"\nEnter the number of processes: ";
    cin>>n;
    int bt[n], at[n];                   // burst time and arrival time

    cout<<"\nEnter burst time for each process: ";
    for (int i=0; i<n; i++)
        cin>>bt[i];
    cout<<"\nEnter arrival time for each process: ";
    for (int i=0; i<n; i++)
```

```
        cin>>at[i];

    for (int i=0; i<n; i++) {
        // pr[i].pid = k[i];
        pr[i].pid = i+1;
        pr[i].arrival = at[i];
        pr[i].burst = bt[i];
        remaining[i] = pr[i].burst;
    }

    cout<<"\nEnter the time quantum: ";
    cin>>time_quantum;

    RR ();              // logic for calculating various times
    display ();         // displaying calculated values of time
    displayGantt ();    // printing Gantt chart

    return 0;
}
```

OUTPUT:

```
FAIZAN CHOUDHARY
20BCS021

Round Robin Scheduling Algorithm

Enter the number of processes: 5

Enter burst time for each process: 5 3 1 2 3

Enter arrival time for each process: 0 1 2 3 4

Enter the time quantum: 2
```

```
Process | Burst Time | Arrival Time | Completion Time | Waiting Time | Turnaround Time | Response Time
------------------------------------------------------------------------------------------------------
   P2         3             1              12               8              11              1
   P3         1             2               5               2               3              2
   P4         2             3               9               4               6              4
   P5         3             4              14               7              10              5
------------------------------------------------------------------------------------------------------


Average Completion time: 10.60
Average Waiting time: 5.80
Average Turnaround time: 8.60
Average Response time: 2.40

Gantt chart:
| P1 | P2 | P3 | P1 | P4 | P5 | P2 | P1 | P5 |
0    2    4    5    7    9    11   12   13   14
```

```
FAIZAN CHOUDHARY
20BCS021

Round Robin Scheduling Algorithm

Enter the number of processes: 6

Enter burst time for each process: 4 5 2 1 6 3

Enter arrival time for each process: 0 1 2 3 4 6

Enter the time quantum: 2
```

```
Process | Burst Time | Arrival Time | Completion Time | Waiting Time | Turnaround Time | Response Time
--------------------------------------------------------------------------------------------------------

   P1        4            0              8               4              8              0
   P2        5            1             18              12             17              1
   P3        2            2              6               2              4              2
   P4        1            3              9               5              6              5
   P5        6            4             21              11             17              5
   P6        3            6             19              10             13              7
--------------------------------------------------------------------------------------------------------


Average Completion time: 13.50
Average Waiting time: 7.33
Average Turnaround time: 10.83
Average Response time: 3.33

Gantt chart:
| P1 | P2 | P3 | P1 | P4 | P5 | P2 | P6 | P5 | P2 | P6 | P5 |
0    2    4    6    8    9    11   13   15   17   18   19   21
```

```
FAIZAN CHOUDHARY
20BCS021

Round Robin Scheduling Algorithm

Enter the number of processes: 3

Enter burst time for each process: 4 3 5

Enter arrival time for each process: 0 0 0

Enter the time quantum: 2
```

```
Process | Burst Time | Arrival Time | Completion Time | Waiting Time | Turnaround Time | Response Time
--------------------------------------------------------------------------------------------------------

   P1        4            0              8               4              8              0
   P2        3            0              9               6              9              2
   P3        5            0             12               7             12              4
--------------------------------------------------------------------------------------------------------


Average Completion time: 9.67
Average Waiting time: 5.67
Average Turnaround time: 9.67
Average Response time: 2.00

Gantt chart:
| P1 | P2 | P3 | P1 | P2 | P3 | P3 |
0    2    4    6    8    9    11   12
```