Name : Anupam Kunwar
Reg : 19BCE1369
WEEK-7

Q1.
Prompt the use for entering number of processes and their details.
❒Number of processes

❒Process arrival time

❒Process CPU burst time requirement

❒Process priority

Categorize the processes into three different queues based on the specific range of priorities. Use round robin for the highest priority queue with a time quanta of 3. Use shortest job first for the middle level queue. Use first come first serve for the low priority queue.

You can switch CPU between the queues in a round robin manner with a time quanta of 15. Consider sufficient number of process to analyses the average waiting time of the processes. Calculate the average waiting time for each queue also.

Solution :

```
#include <iostream>
using namespace std;

struct process
{
int priority;
int burst_time;
int arrival_time;
int tat_time;
int total_time = 0;
};

struct queues
{
int priority_start;
int priority_end;
int total_time = 0;
int avg_wait_time;
int length = 0;
process *p;
bool executed = false;
};

bool notComplete(queues q[])
{
bool a = false;
int countInc = 0;
for (int i = 0; i < 3; i++)
{
countInc = 0;
for (int j = 0; j < q[i].length; j++)
{
if (q[i].p[j].burst_time != 0)
{
a = true;
}
else
{
countInc += 1;
}
```

```
}
if (countInc == q[i].length)
{

q[i].executed = true;
}
}
return a;
}

void sort_sjf(queues q)
{
//Queue q has to be sorted according to burst-time of processes
for (int i = 1; i < q.length; i++)
{
for (int j = 0; j < q.length - 1; j++)
{
if (q.p[j].burst_time < q.p[j + 1].burst_time)
{
process temp = q.p[j + 1];
q.p[j + 1] = q.p[j];
q.p[j] = temp;
}
}
}
}

void sort_fcfs(queues q)
{
//Queue q has to be sorted according to arrival-time of processes
for (int i = 1; i < q.length; i++)
{
for (int j = 0; j < q.length - 1; j++)
{
if (q.p[j].arrival_time < q.p[j + 1].arrival_time)
{
process temp = q.p[j + 1];
q.p[j + 1] = q.p[j];
q.p[j] = temp;
}
}
}
}
```

```
}

void checkCompleteTimer(queues q[])
{
bool a = notComplete(q);
for (int i = 0; i < 3; i++)
{
if (q[i].executed == false)
{
for (int j = 0; j < q[i].length; j++)
{
if (q[i].p[j].burst_time != 0)
{
q[i].p[j].total_time += 1;
}
}
q[i].total_time += 1;
}
}
}

int main()
{

//Initializing 3 queues with specific priority range
queues q[3];
q[0].priority_start = 7;
q[0].priority_end = 9;
q[1].priority_start = 4;
q[1].priority_end = 6;
q[2].priority_start = 1;
q[2].priority_end = 3;

int no_of_processes, priority_of_process, burst_time_of_process,
arrival_time_of_process;
//Prompt User for entering Processes and assigning it to respective
queues.
cout << "Enter the number of processes\n";
cin >> no_of_processes;
process p1[no_of_processes];

for (int i = 0; i < no_of_processes; i++)
```

```
{
cout << "Enter the priority of the process : ";
cin >> priority_of_process;
cout << "Enter the burst time of the process : ";
cin >> burst_time_of_process;
cout << "Enter arrival time of process : ";
cin >> arrival_time_of_process;
p1[i].priority = priority_of_process;
p1[i].burst_time = burst_time_of_process;
p1[i].tat_time = burst_time_of_process;
p1[i].arrival_time = arrival_time_of_process;
for (int j = 0; j < 3; j++)
{
if (q[j].priority_start <= priority_of_process && priority_of_process <=
q[j].priority_end)
{
q[j].length++;
}
}
}

for (int i = 0; i < 3; i++)
{
int len = q[i].length;
q[i].p = new process[len];
}

int a = 0;
int b = 0;
int c = 0;

for (int i = 0; i < 3; i++)
{
for (int j = 0; j < no_of_processes; j++)
{
if ((q[i].priority_start <= p1[j].priority) && (p1[j].priority <=
q[i].priority_end))
{
if (i == 0)
{
q[i].p[a++] = p1[j];
}
```

```
else if (i == 1)
{
q[i].p[b++] = p1[j];
}
else
{
q[i].p[c++] = p1[j];
}
}
}
}

a--;
b--;
c--;
cout << "\n";
for (int i = 0; i < 3; i++)
{
cout << "Queue " << i + 1 << " : \t";
for (int j = 0; j < q[i].length; j++)
{
cout << q[i].p[j].priority << "->";
}
cout << "NULL\n";
}
cout << "\n";
//While RR on multiple queues is not complete, keep on repeating
int timer = 0;
int l = -1;
int rr_timer = 3;
int counter = 0;
int countersjf = 0;
int counterfcfs = 0;
while (notComplete(q))
{
if (timer == 15)
{
timer = 0;
}
l += 1;
if (l >= 3)
{
```

```
l = l % 3;
}

//Process lth queue if its already not executed
//If its executed change the value of l
if (q[l].executed == true)
{
l += 1;
if (l >= 3)
{
l = l % 3;
}
continue;
}

//Finally you now have a queue which is not completely executed
//Process the incomplete processes over it

if (l == 0)
{
cout << "Executing "
<< "Queue " << l + 1 << " with RR approach\n";
//Round Robin Algorithm for q=3
if (rr_timer == 0)
{
rr_timer = 3;
}

for (int i = 0; i < q[l].length; i++)
{
if (q[l].p[i].burst_time == 0)
{
counter++;
continue;
}
if (counter == q[l].length)
{
break;
}
while (rr_timer > 0 && q[l].p[i].burst_time != 0 && timer != 15)
{
q[l].p[i].burst_time--;
```

```
checkCompleteTimer(q);
rr_timer--;
timer++;
}
if (timer == 15)
{
break;
}
if (q[l].p[i].burst_time == 0 && rr_timer == 0)
{
rr_timer = 3;
if (i == (q[i].length - 1))
{
i = -1;
}
continue;
}
if (q[l].p[i].burst_time == 0 && rr_timer > 0)
{
if (i == (q[i].length - 1))
{
i = -1;
}
continue;
}
if (rr_timer <= 0)
{
rr_timer = 3;
if (i == (q[i].length - 1))
{
i = -1;
}
continue;
}
}
}

else if (l == 1)
{
cout << "Executing "
<< "Queue " << l + 1 << " with SJF approach\n";
sort_sjf(q[l]); //sorting queue according to burst time
```

```
//SJF Scheduling(Non-preemptive)
for (int i = 0; i < q[l].length; i++)
{
if (q[l].p[i].burst_time == 0)
{
countersjf++;
continue;
}
if (countersjf == q[l].length)
{
break;
}
while (q[l].p[i].burst_time != 0 && timer != 15)
{
q[l].p[i].burst_time--;
checkCompleteTimer(q);
timer++;
}
if (timer == 15)
{
break;
}
if (q[l].p[i].burst_time == 0)
{
continue;
}
}
}
else
{
cout << "Executing "
<< "Queue " << l + 1 << " with FCFS approach\n";
//FCFS
sort_fcfs(q[l]); //sorting queue according to arrival time
for (int i = 0; i < q[l].length; i++)
{
if (q[l].p[i].burst_time == 0)
{
counterfcfs++;
continue;
}
if (counterfcfs == q[l].length)
```

```
{
break;
}
while (q[l].p[i].burst_time != 0 && timer != 15)
{
q[l].p[i].burst_time--;
checkCompleteTimer(q);
timer++;
}
if (timer == 15)
{
break;
}
if (q[l].p[i].burst_time == 0)
{
continue;
}
}
}
cout << "\n";
}

int sum_tt = 0;
int sum_wt = 0;
int wtx;
int wty;
cout << "\n\nProcess | Turn Around Time | Waiting Time\n";
for (int i = 0; i < 3; i++)
{
cout << "Queue " << i + 1 << "\n";
for (int j = 0; j < q[i].length; j++)
{
wty = q[i].p[j].total_time - q[i].p[j].tat_time;
if (wty < 0)
wty = 0;
cout << "Process P" << j + 1 << "\t" << q[i].p[j].total_time << "\t\t " << wty
<< "\n";
sum_tt += q[i].p[j].total_time;
sum_wt += wty;
wtx = sum_wt;
}
q[i].avg_wait_time = wtx / q[i].length;
```

```
wtx = 0;
}

for (int i = 0; i < 3; i++)
{
cout << "\nTotal Time taken for queue " << i + 1 << " to execute: " <<
q[i].total_time << "\n";
cout << "Average waiting time for queue " << i + 1 << " : " <<
q[i].avg_wait_time << "\n";
}

cout << "\nThe average turnaround time for all process is : " << sum_tt /
no_of_processes << endl;
cout << "\nThe average waiting time for all process is : " << sum_wt /
no_of_processes << endl;
}
```

Output :

```
piratepanda@SastaPC:~/Documents/oslab/week6$ g++ queue.cpp
piratepanda@SastaPC:~/Documents/oslab/week6$ ./a.out
Enter the number of processes
9
Enter the priority of the process : 9
Enter the burst time of the process : 3
Enter arrival time of process : 2
Enter the priority of the process : 8
Enter the burst time of the process : 2
Enter arrival time of process : 3
Enter the priority of the process : 7
Enter the burst time of the process : 3
Enter arrival time of process : 4
Enter the priority of the process : 6
Enter the burst time of the process : 1
Enter arrival time of process : 2
Enter the priority of the process : 5
Enter the burst time of the process : 2
Enter arrival time of process : 4
Enter the priority of the process : 4
Enter the burst time of the process : 3
Enter arrival time of process : 1
Enter the priority of the process : 3
Enter the burst time of the process : 1
Enter arrival time of process : 3
Enter the priority of the process : 2
```

.

```
Enter the burst time of the process : 3
Enter arrival time of process : 2
Enter the priority of the process : 1
Enter the burst time of the process : 2
Enter arrival time of process : 3

Queue 1 :       9->8->7->NULL
Queue 2 :       6->5->4->NULL
Queue 3 :       3->2->1->NULL

Executing Queue 1 with RR approach

Executing Queue 2 with SJF approach

Executing Queue 3 with FCFS approach

Executing Queue 3 with FCFS approach
```

.

```
Process     | Turn Around Time | Waiting Time
Queue 1
Process P1      2                   0
Process P2      4                   2
Process P3      7                   4
Queue 2
Process P1      10                  7
Process P2      12                  10
Process P3      13                  12
Queue 3
Process P1      14                  13
Process P2      16                  14
Process P3      19                  16

Total Time taken for queue 1 to execute: 7
Average waiting time for queue 1 : 2

Total Time taken for queue 2 to execute: 13
Average waiting time for queue 2 : 11

Total Time taken for queue 3 to execute: 19
Average waiting time for queue 3 : 26

The average turnaround time for all process is : 10

The average waiting time for all process is : 8
```