System Programming and Operating Systems Lab

ASSIGNMENT 5

Name: Shrirang Mhalgi
Roll No: 322008
Batch: B 1

1 Date of Completion:

06/02/2019

2 Aim:

Write a java program (using OOP features) to implement following Scheduling algorithms: FCFS, SJF (Preemtive), Priority (non-Preemtive) and Round Robin (Preemtive).

3 Objectives:

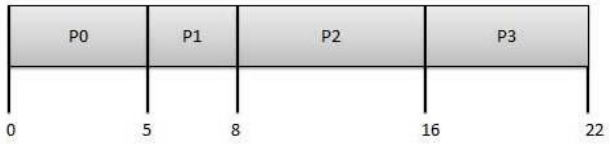
To implement various Scheduling Algorithms

4 Theory:

The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.
Process scheduling is an essential part of a Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing. Process Scheduler schedules different processes to be assigned to the CPU based on particular scheduling algorithms. There are four popular process scheduling algorithms which we are going to discuss

1. First-Come, First-Served (FCFS) Scheduling
   (a) Jobs are executed on first come, first serve basis.
   (b) It is a non-preemptive, pre-emptive scheduling algorithm.
   (c) Easy to understand and implement.
   (d) Its implementation is based on FIFO queue.
   (e) Poor in performance as average wait time is high.

Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16



Wait time of each process is as follows

Process	Wait Time : Service Time - Arrival Time
P0	0
P1	4
P2	6
P3	13

Average Wait Time: (0+4+6+13) / 4 = 5.75

2. Shortest-Job-First (SJF) Scheduling

(a) This is also known as shortest job first, or SJF

(b) This is a non-preemptive, pre-emptive scheduling algorithm.

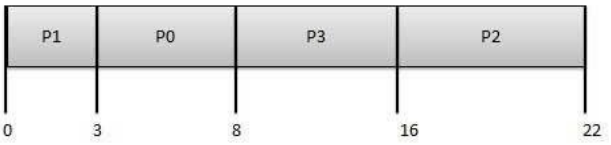
(c) Best approach to minimize waiting time.

(d) Easy to implement in Batch systems where required CPU time is known in advance.

(e) Impossible to implement in interactive systems where required CPU time is not known.

(f) The processer should know in advance how much time process will take.

Process	Arrival Time	Execute Time	Service Time
P0	0	5	3
P1	1	3	0
P2	2	8	16
P3	3	6	8



Wait time of each process is as follows

Process	Wait Time : Service Time - Arrival Time
P0	3
P1	0
P2	14
P3	5

Average Wait Time: (3+0+14+5) / 4 = 5.50

3. Priority Scheduling

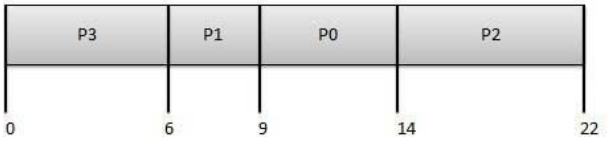
(a) Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems.

(b) Each process is assigned a priority. Process with highest priority is to be executed first and so on.

(c) Processes with same priority are executed on first come first served basis.

(d) Priority can be decided based on memory requirements, time requirements or any other resource requirement.

Process	Arrival Time	Execute Time	Priority	Service Time
P0	0	5	1	9
P1	1	3	2	6
P2	2	8	1	14
P3	3	6	3	0



Wait time of each process is as follows

Process	Wait Time : Service Time - Arrival Time
P0	9 - 0 = 9
P1	6 - 1 = 5
P2	14 - 2 = 12
P3	0 - 0 = 0

Average Wait Time: (9+5+12+0) / 4 = 6.5

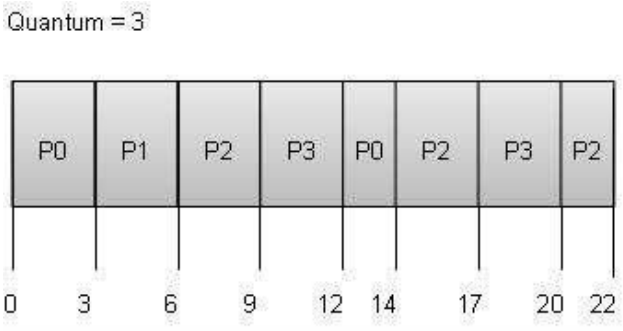
4. Round Robin(RR) Scheduling

(a) Round Robin is the preemptive process scheduling algorithm.

(b) Each process is provided a fix time to execute, it is called a quantum.

(c) Once a process is executed for a given time period, it is preempted and other process executes for a given time period.

(d) Context switching is used to save states of preempted processes.



Wait time of each process is as follows

Process	Wait Time : Service Time - Arrival Time
P0	(0 - 0) + (12 - 3) = 9
P1	(3 - 1) = 2
P2	(6 - 2) + (14 - 9) + (20 - 17) = 12
P3	(9 - 3) + (17 - 12) = 11

Average Wait Time: (9+2+12+11) / 4 = 8.5

These algorithms are either non-preemptive or preemptive. Non-preemptive algorithms are designed so that once a process enters the running state, it cannot be preempted until it completes its allotted time, whereas the preemptive scheduling is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state.

5 Algorithm:

FCFS :

1. Start the process
2. Accept the number of processes in the ready Queue
3. For each process in the ready Q, assign the process id and accept the CPU burst time
4. Set the waiting of the first process as 0 and its burst time as its turn around time
5. for each process in the Ready Q calculate

(a) Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n-1)

(b) Turn around time for Process(n)= waiting time of Process(n)+ Burst time for process(n)

6. Calculate

(a) Average waiting time = Total waiting Time / Number of process

(b) Average Turnaround time = Total Turnaround Time / Number of process

7. Stop the process

SJF :

1. Start the process
2. Accept the number of processes in the ready Queue
3. For each process in the ready Q, assign the process id and accept the CPU burst time
4. Start the Ready Q according the shortest Burst time by sorting according to lowest to highest burst time.
5. Set the waiting time of the first process as 0 and its turnaround time as its burst time.
6. For each process in the ready queue, calculate

(a) Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n-1)

(b) Turn around time for Process(n)= waiting time of Process(n)+ Burst time for process(n)

7. Calculate

(a) Average waiting time = Total waiting Time / Number of process

(b) Average Turnaround time = Total Turnaround Time / Number of process

8. Stop the process

Priority Scheduling :

1. Start the process
2. Accept the number of processes in the ready Queue
3. For each process in the ready Q, assign the process id and accept the CPU burst time, priority
4. Start the Ready Q according the priority by sorting according to lowest to highest burst time and process.
5. Set the waiting time of the first process as 0 and its turnaround time as its burst time.
6. For each process in the ready queue, calculate

(a) Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n-1)

(b) Turn around time for Process(n)= waiting time of Process(n)+ Burst time for process(n)

7. Calculate

(a) Average waiting time = Total waiting Time / Number of process

(b) Average Turnaround time = Total Turnaround Time / Number of process

8. Stop the process

Round Robin :

1. Start the process
2. Accept the number of processes in the ready Queue and time quantum (or) time slice
3. For each process in the ready Q, assign the process id and accept the CPU burst time
4. Calculate the no. of time slices for each process where No. of time slice for process(n) = burst time process(n)/time slice
5. If the burst time is less than the time slice then the no. of time slices =1.
6. Consider the ready queue is a circular Q, calculate

(a) Waiting time for process(n) = waiting time of process(n-1)+ burst time of process(n-1) + the time difference in getting the CPU from process(n-1)

(b) Turn around time for process(n) = waiting time of process(n) + burst time of process(n)+ the time difference in getting CPU from process(n).

7. Calculate

(a) Average waiting time = Total waiting Time / Number of process

(b) Average Turnaround time = Total Turnaround Time / Number of process

8. Stop the process

6 Flowchart:

[H]

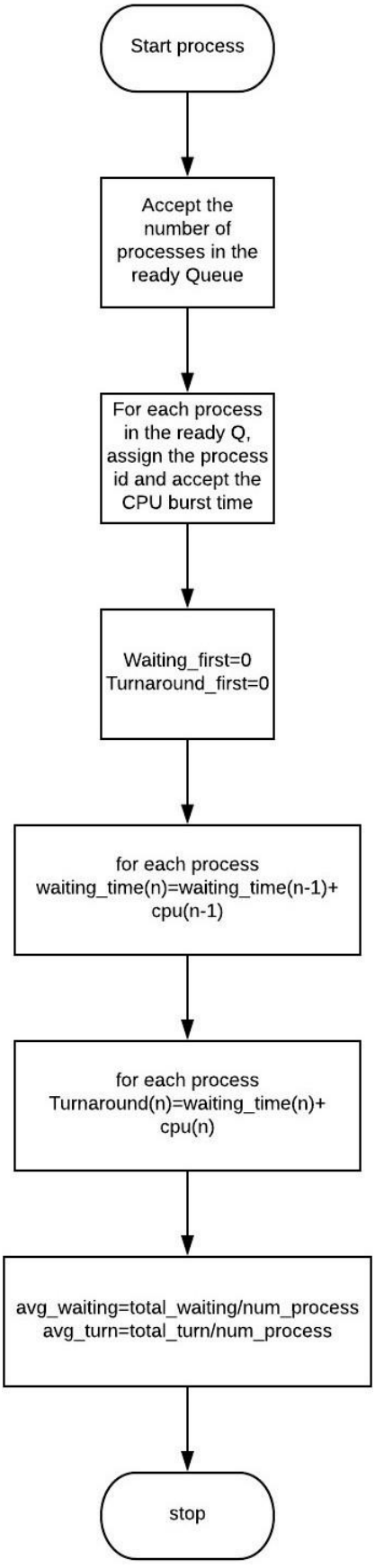


Figure 1: FCFS

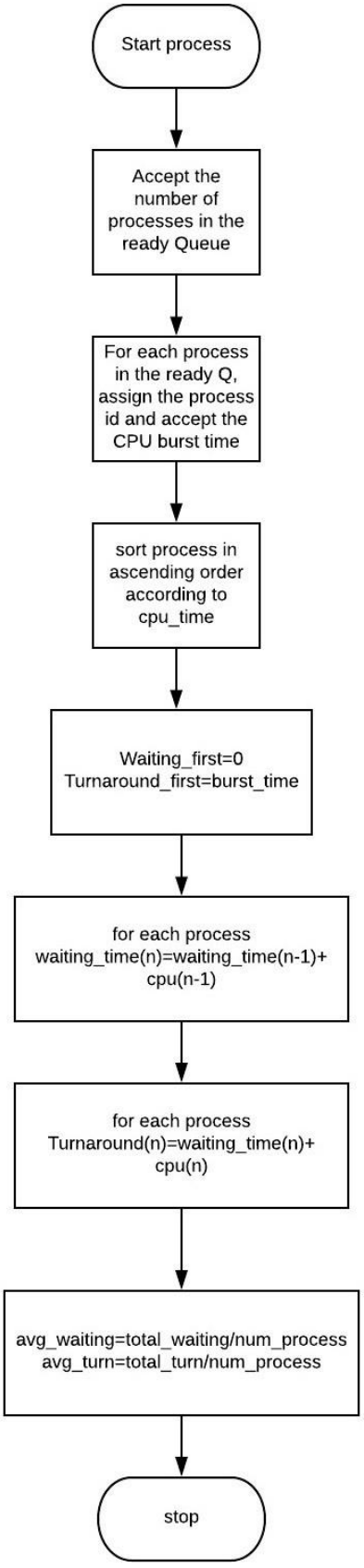

Start process
Accept the number of processes in the ready Queue
For each process in the ready Q, assign the process id and accept the CPU burst time
sort process in ascending order according to cpu_time
Waiting_first=0
Turnaround_first=burst_time
for each process
waiting_time(n)=waiting_time(n-1)+ cpu(n-1)
for each process
Turnaround(n)=waiting_time(n)+ cpu(n)
avg_waiting=total_waiting/num_process
avg_turn=total_turn/num_process
stop

Figure 2: SJF

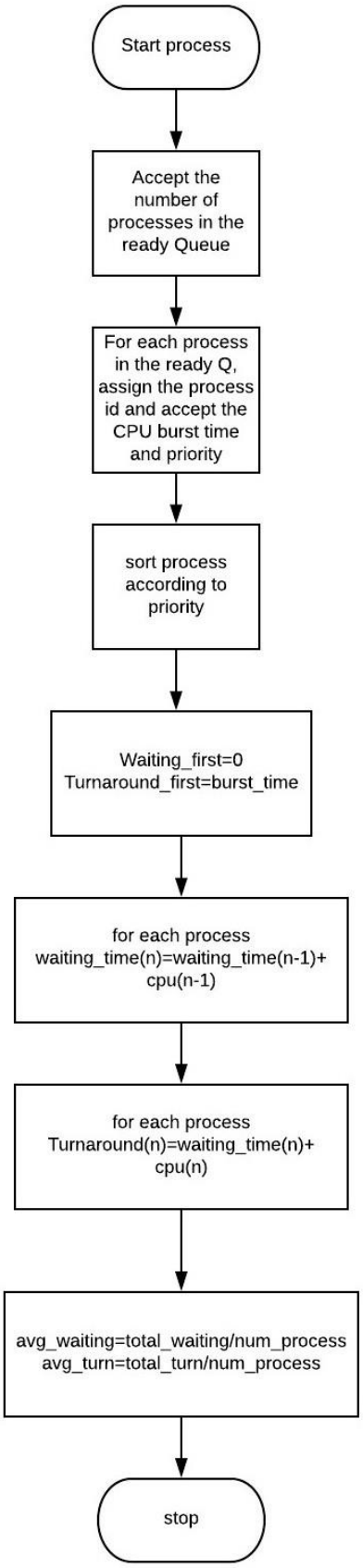


Figure 3: Priority Scheduling

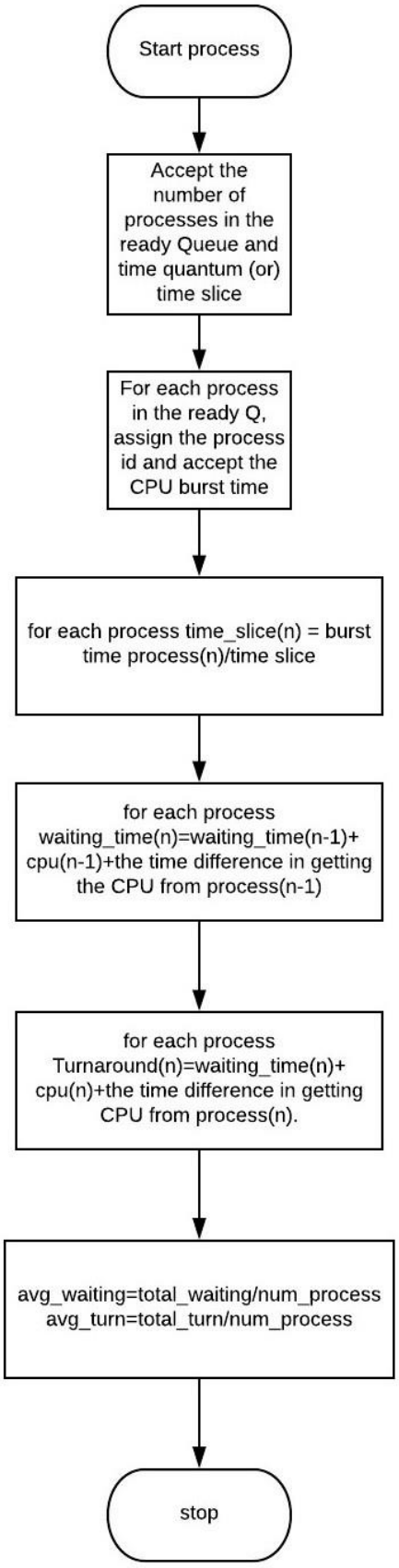


Figure 4: Round Robin

7 Code:

1. FCFS:

```
import java.io.*;
import java.util.Scanner;
public class FCFS
{

int cpu_time,arrival_time,process_num,priority,wait_time;
Scanner s=new Scanner(System.in);

FCFS(int no)
{
process_num=no+1;
cpu_time=arrival_time=priority=0;
}

void input()
{
System.out.print("Enter the CPU time for P"+process_num+":");
cpu_time=s.nextInt();

//System.out.print("Enter the arrival time :");
//arrival_time=s.nextInt();

//System.out.println("Enter its priority  :");
//priority=s.nextInt();

}


static void cal_time(FCFS[] obj,int  no)
{

// ------Waiting  time-----------
int count=0;
float avg=0;
System.out.println();
for(int  i=0;i<no;i++)
{
avg+=count;
obj[i].wait_time=count;
System.out.println("Waiting time for P"+(i+1)+" :"+obj[i].wait_time);
count+=obj[i].cpu_time;
}

System.out.println("Average  Waiting  time  :"+(avg/no));

// ------TurnAround  time-----------
```

```
System.out.println();
count=0;avg=0;
for(int i=0;i<no;i++)
{

System.out.println("TurnAround time for P"+(i+1)+" :"+(obj[i].wait_time+obj[i].cpu_
count+=obj[i].wait_time+obj[i].cpu_time;
avg+=obj[i].wait_time+obj[i].cpu_time;
}
System.out.println(avg);
System.out.println("Average TurnAround time :"+(avg/no));

}


public static void main(String[] args)
{
Scanner in=new Scanner(System.in);
System.out.print("Enter the number of processes :");
int num=in.nextInt();

FCFS[] obj=new FCFS[num];

for(int i=0;i<num;i++)
{
obj[i]=new FCFS(i);
obj[i].input();
}

FCFS.cal_time(obj,num);

}

}
```

2. **SJF:**

```
import java.io.*;
import java.util.Scanner;

public class SJFS
{
int cpu_time,process_num,wait_time,arrival_time;
Scanner s=new Scanner(System.in);
SJFS(int no)
{
cpu_time=0;
process_num=no;
wait_time=0;
arrival_time=0;
```

```
}

public void input()
{
System.out.print("Enter the CPU time for P"+(process_num)+" :");
cpu_time=s.nextInt();
System.out.print("Enter the arrival time for P"+(process_num)+" :");
arrival_time=s.nextInt();
}


public static SJFS minimum(SJFS[] obj,int no,int time)
{
SJFS ref=obj[0];
for(int i=1;i<no;i++)
{
if(obj[i].cpu_time!=0 && obj[i].arrival_time<=time && obj[i].cpu_time<ref.cpu_time)
{
ref=obj[i];
}
}

return ref;
}

public static void process(SJFS[] obj,int no)
{
SJFS ref;
int time=0;
while(!check_all(obj,no))
{
ref=minimum(obj,no,time);
System.out.println("P"+ref.process_num + " :" +(time+1));
ref.cpu_time=ref.cpu_time-1;
time++;
}
}


public static boolean check_all(SJFS[] process,int no)
{
boolean var=true;
for(int i=0;i<no;i++)
{
if(process[i].cpu_time!=0)
{
var=false;
break;
}
}
```

```
return var;
}


public static void main(String[]  args)
{
Scanner in=new Scanner(System.in);
System.out.print("Enter the number of processes :");
int num=in.nextInt();

SJFS[] obj=newSJFS[num];

SJFS ref;

for(int  i=0;i<num;i++)
{
obj[i]=new SJFS(i+1);
obj[i].input();
}

//sort on basis of arrival time

for(int  i=0;i<num;i++)
{
for(int  j=i+1;j<num;j++)
{
if(obj[i].arrival_time>obj[j].arrival_time)
{
ref=obj[i];
obj[i]=obj[j];
obj[j]=ref;

}
}
}

SJFS.process(obj,num);


}
}
```

3. Priority Scheduling:

```
import java.io.*;
import java.util.Scanner;
public class Priority
{
```

```
int cpu_time,arrival_time,process_num,priority,wait_time;
Scanner s=new Scanner(System.in);

Priority(int no)
{
process_num=no+1;
cpu_time=arrival_time=priority=0;
}

void input()
{
System.out.print("Enter the CPU time for P"+process_num+":");
cpu_time=s.nextInt();

//System.out.print("Enter the arrival time :");
//arrival_time=s.nextInt();

System.out.print("Enter its priority :");
priority=s.nextInt();
}


static void cal_time(Priority[] obj,int no)
{

// ------Waiting time-----------
int count=0;
float avg=0;
System.out.println();
for(int i=0;i<no;i++)
{
avg+=count;
obj[i].wait_time=count;
System.out.println("Waiting time for P" + (obj[i].process_num) +" :"+obj[i].wait_ti
count+=obj[i].cpu_time;
}

System.out.println("Average Waiting time :"+(avg/no));

// ------TurnAround time-----------

count=0;avg=0;
for(int i=0;i<no;i++)
{

System.out.println("TurnAround time for P"+(obj[i].process_num)+" :"+(obj[i].wait_t
count+=obj[i].wait_time+obj[i].cpu_time;
avg+=obj[i].wait_time+obj[i].cpu_time;
}
System.out.println(avg);
```

```
System.out.println("Average TurnAround time :"+(avg/no));
}



public static void main(String[]  args)
{
Scanner in=new Scanner(System.in);
System.out.print("Enter the number of processes :");
int num=in.nextInt();

Priority[] obj=new Priority[num];

Priority ref;



for(int i=0;i<num;i++)
{
obj[i]=new Priority(i);
obj[i].input();
}

for(int i=0;i<num;i++)
{
for(int j=i+1;j<num;j++)
{
if(obj[i].priority>obj[j].priority)
{
ref=obj[i];
obj[i]=obj[j];
obj[j]=ref;

}
}
}

Priority.cal_time(obj,num);

}

}
```

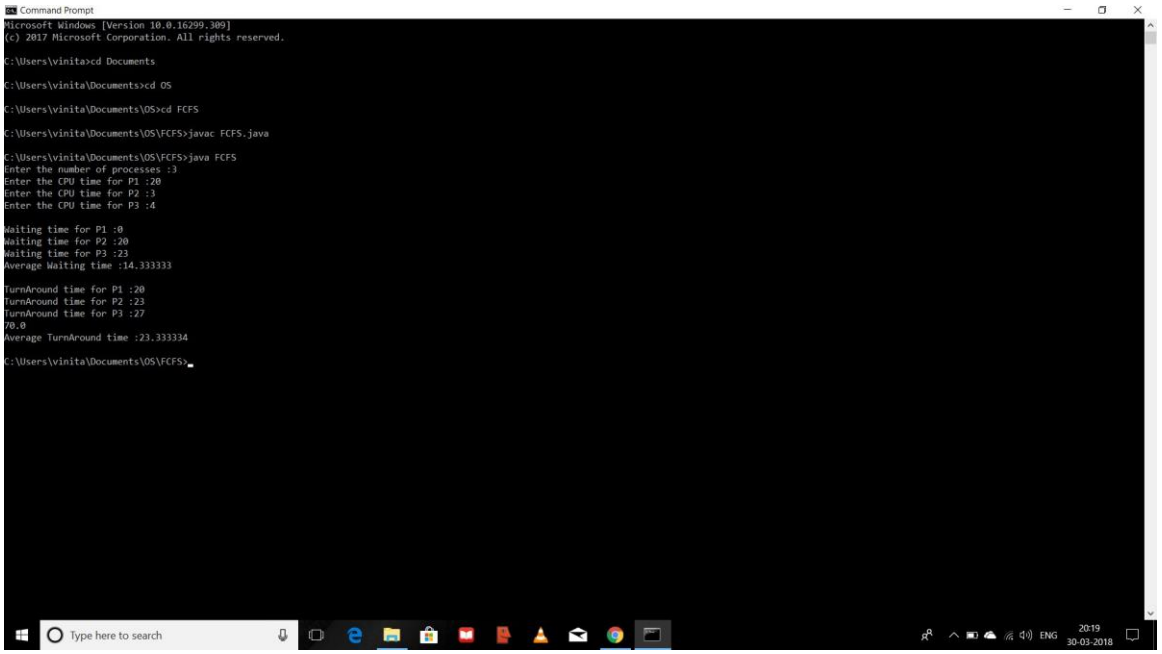
4. Round Robin:

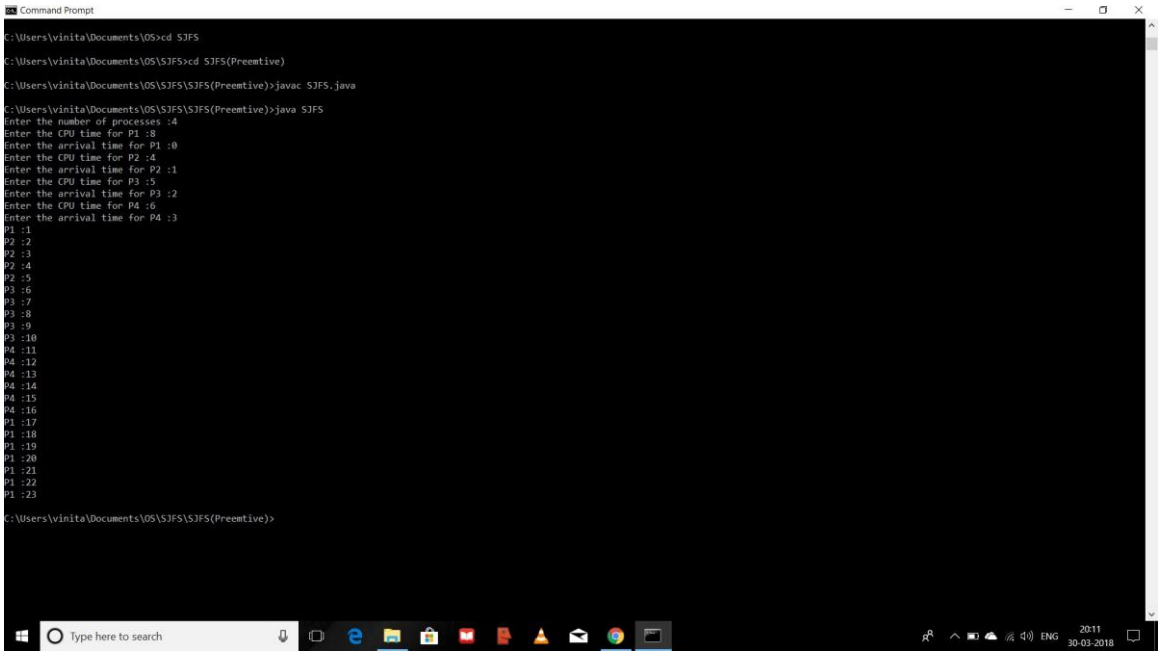
```
import java.io.*;
import java.util.Scanner;
public class RR
```

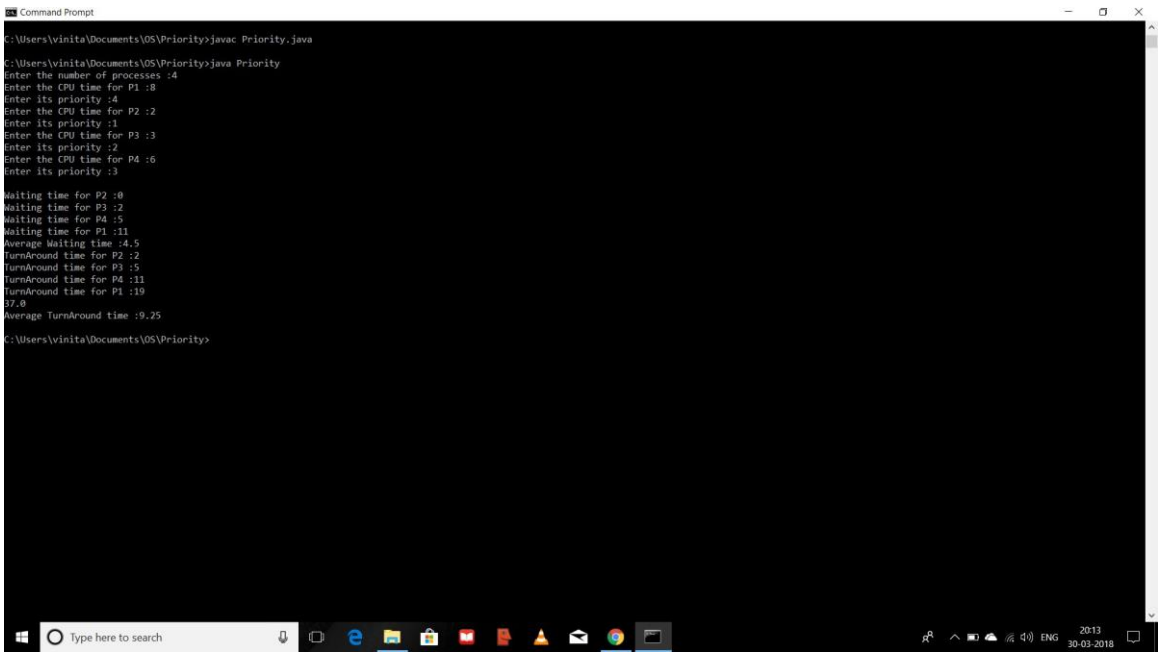
```
{
int process_num,cpu_time;
Scanner s=new Scanner(System.in);

RR(int no)
{
cpu_time=0;
process_num=no+1;
}

public void input()
{
System.out.print("Enter the CPU time for P"+(process_num)+" :");
cpu_time=s.nextInt();
}

public static void process(RR[] obj,int num,int time_slice)
{
while(!check_all(obj,num))
{
for(int i=0;i<num;i++)
{
if(obj[i].cpu_time!=0 )
{
if(obj[i].cpu_time>=time_slice)
{
obj[i].cpu_time=obj[i].cpu_time-time_slice;
System.out.println("P" + obj[i].process_num +" :"+time_slice);
}

else
{
System.out.println("P" + obj[i].process_num +" :"+obj[i].cpu_time);
obj[i].cpu_time=0;
}
}
}
}
}


public static boolean check_all(RR[] process,int no)
{
boolean var=true;
for(int i=0;i<no;i++)
{
if(process[i].cpu_time!=0)
{
var=false;
break;
```

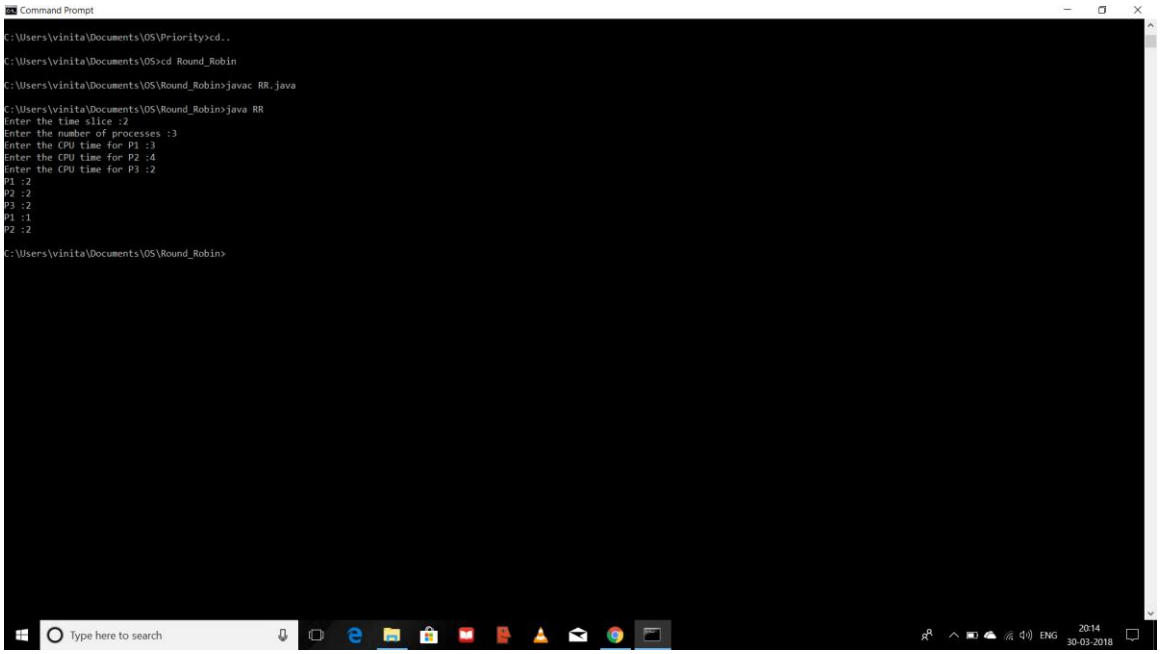
```
}
}

return var;
}


public static void main(String[]  args)
{
Scanner in=new Scanner(System.in);

System.out.print("Enter the time slice :");
int time_slice=in.nextInt();

System.out.print("Enter the number of processes:");
int num=in.nextInt();

RR[]  obj=new  RR[num];

for(int  i=0;i<num;i++)
{
obj[i]=new RR(i);
obj[i].input();
}

RR.process(obj,num,time_slice);
}

}
```

8 Output:


Command Prompt
Microsoft Windows [Version 10.0.16299.309]
(c) 2017 Microsoft Corporation. All rights reserved.
C:\Users\vinita>cd Documents
C:\Users\vinita\Documents>cd OS
C:\Users\vinita\Documents\OS>cd FCFS
C:\Users\vinita\Documents\OS\FCFS>javac FCFS.java
C:\Users\vinita\Documents\OS\FCFS>java FCFS
Enter the number of processes :3
Enter the CPU time for P1 :20
Enter the CPU time for P2 :3
Enter the CPU time for P3 :4
Waiting time for P1 :0
Waiting time for P2 :20
Waiting time for P3 :23
Average Waiting time :14.333333
TurnAround time for P1 :20
TurnAround time for P2 :23
TurnAround time for P3 :27
70.0
Average TurnAround time :23.33334
C:\Users\vinita\Documents\OS\FCFS>
Type here to search
ENG
20:19
30-03-2018


Command Prompt
C:\Users\vinita\Documents\OS>cd SJFS
C:\Users\vinita\Documents\OS\SJFS>cd SJFS(Preemtive)
C:\Users\vinita\Documents\OS\SJFS\SJFS(Preemtive)>javac SJFS.java
C:\Users\vinita\Documents\OS\SJFS\SJFS(Preemtive)>java SJFS
Enter the number of processes :4
Enter the CPU time for P1 :8
Enter the arrival time for P1 :0
Enter the CPU time for P2 :4
Enter the arrival time for P2 :1
Enter the CPU time for P3 :5
Enter the arrival time for P3 :2
Enter the CPU time for P4 :6
Enter the arrival time for P4 :3
P1 :1
P2 :2
P2 :3
P2 :4
P2 :5
P3 :6
P3 :7
P3 :8
P3 :9
P3 :10
P4 :11
P4 :12
P4 :13
P4 :14
P4 :15
P4 :16
P1 :17
P1 :18
P1 :19
P1 :20
P1 :21
P1 :22
P1 :23
C:\Users\vinita\Documents\OS\SJFS\SJFS(Preemtive)>
Type here to search
ENG
20:11
30-03-2018





9 Conclusion:

Hence we have studied that-

1. CPU scheduling concepts like context switching, types of schedulers, different timing parameter like waiting time, turnaround time, burst time, etc.
2. Different CPU scheduling algorithms like FIFO, SJF,Etc.
3. FIFO is the simplest for implementation but produces large waiting times and reduces system performance.
4. SJF allows the process having shortest burst time to execute first.