System Programming and Operating Systems Lab

**ASSIGNMENT 5**

**Name: Shrirang Mhalgi Roll No: 322008**

**Batch: B1**

# Date of Completion:

06/02/2019

# Aim:

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

# Objectives:

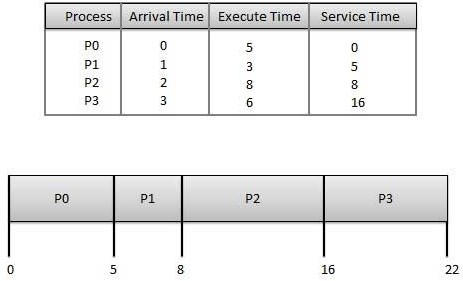
To implement various Scheduling Algorithms

# 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 oper- ating 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
     1. Jobs are executed on first come, first serve basis.
     2. It is a non-preemptive, pre-emptive scheduling algorithm.
     3. Easy to understand and implement.
     4. Its implementation is based on FIFO queue.
     5. Poor in performance as average wait time is high.

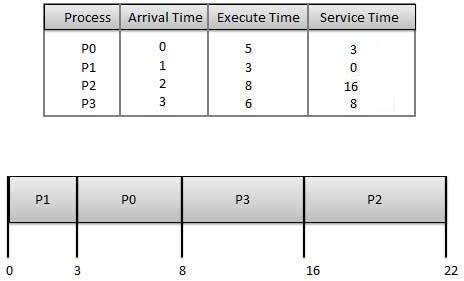


Wait time of each process is as follows

|  |  |
| --- | --- |
| 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

* 1. Shortest-Job-First (SJF) Scheduling
     1. This is also known as shortest job first, or SJF
     2. This is a non-preemptive, pre-emptive scheduling algorithm.
     3. Best approach to minimize waiting time.
     4. Easy to implement in Batch systems where required CPU time is known in advance.
     5. Impossible to implement in interactive systems where required CPU time is not known.
     6. The processer should know in advance how much time process will take.

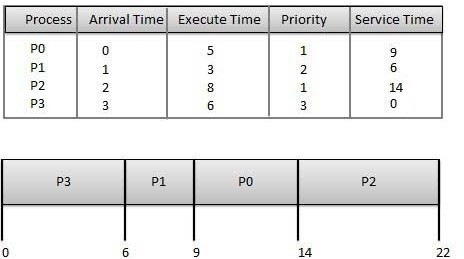


Wait time of each process is as follows

|  |  |
| --- | --- |
| 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

* 1. Priority Scheduling
     1. Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems.
     2. Each process is assigned a priority. Process with highest priority is to be executed first and so on.
     3. Processes with same priority are executed on first come first served basis.
     4. Priority can be decided based on memory requirements, time requirements or any other resource requirement.

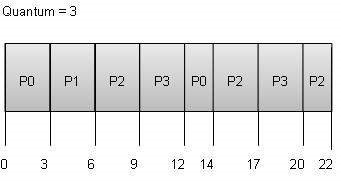


Wait time of each process is as follows

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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

* 1. Round Robin(RR) Scheduling
     1. Round Robin is the preemptive process scheduling algorithm.
     2. Each process is provided a fix time to execute, it is called a quantum.
     3. Once a process is executed for a given time period, it is preempted and other process executes for a given time period.
     4. 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.

# 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
     1. Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n- 1)
     2. Turn around time for Process(n)= waiting time of Process(n)+ Burst time for pro- cess(n)
  6. Calculate
     1. Average waiting time = Total waiting Time / Number of process
     2. 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
   1. Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n- 1)
   2. Turn around time for Process(n)= waiting time of Process(n)+ Burst time for pro- cess(n)
7. Calculate
   1. Average waiting time = Total waiting Time / Number of process
   2. 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
   1. Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n- 1)
   2. Turn around time for Process(n)= waiting time of Process(n)+ Burst time for pro- cess(n)
7. Calculate
   1. Average waiting time = Total waiting Time / Number of process
   2. 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
   1. 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)
   2. 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
   1. Average waiting time = Total waiting Time / Number of process
   2. Average Turnaround time = Total Turnaround Time / Number of process
8. Stop the process

# Flowchart:

[H]

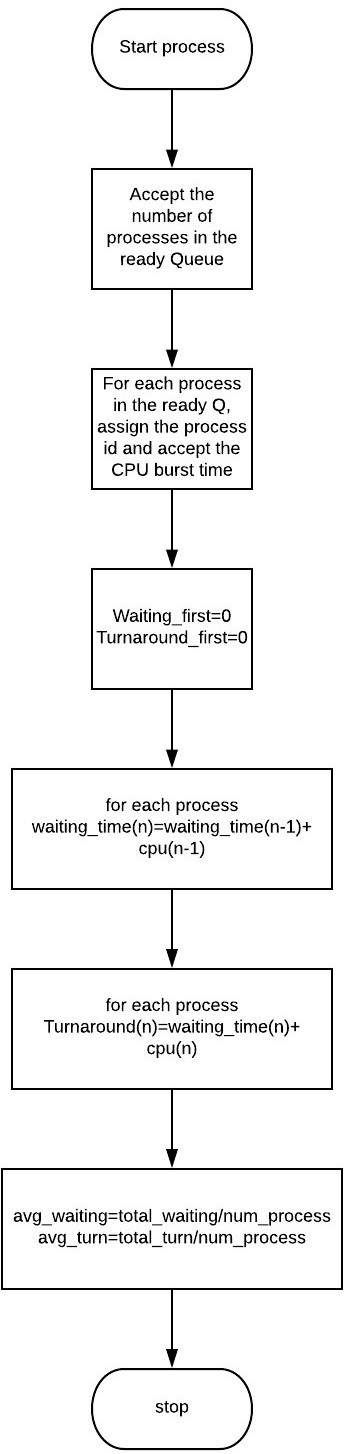


Figure 1: FCFS

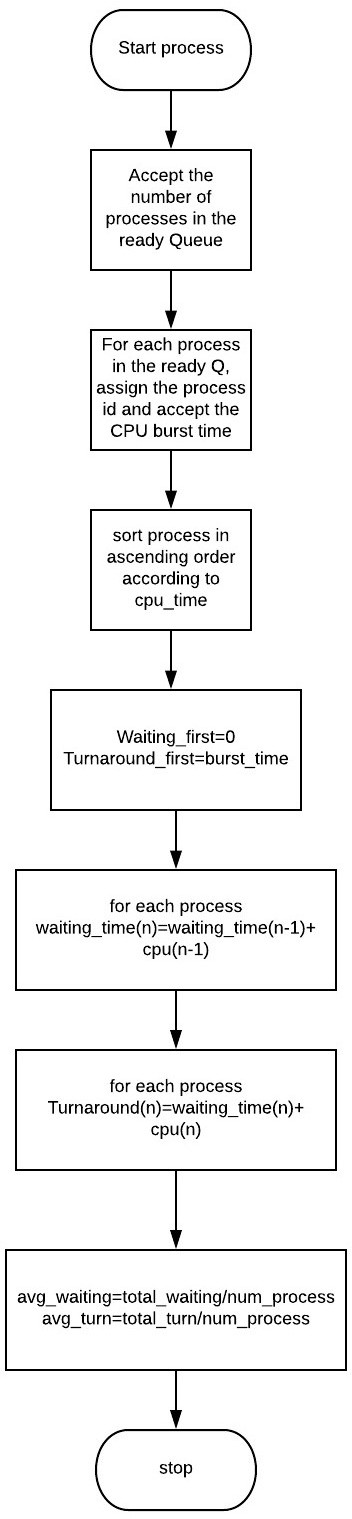


Figure 2: SJF

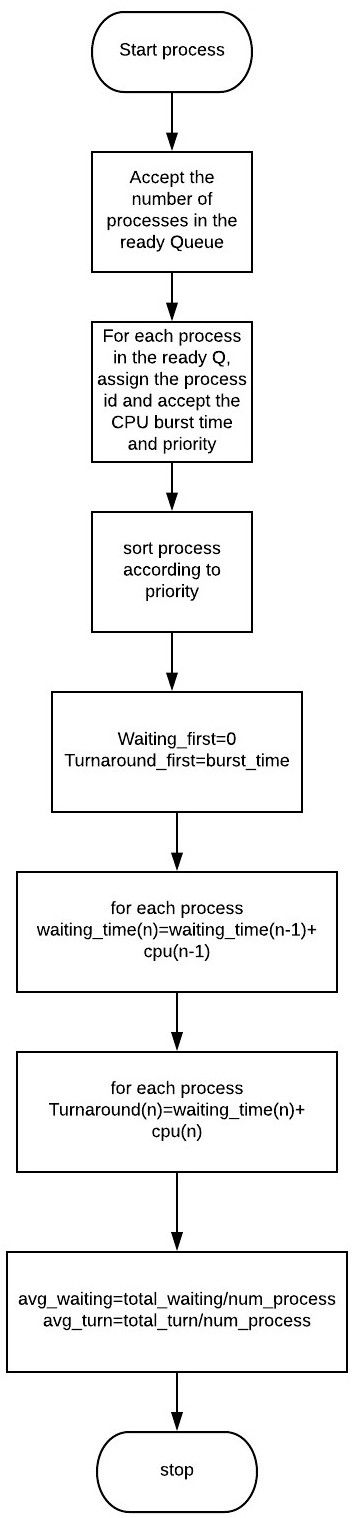


Figure 3: Priority Scheduling

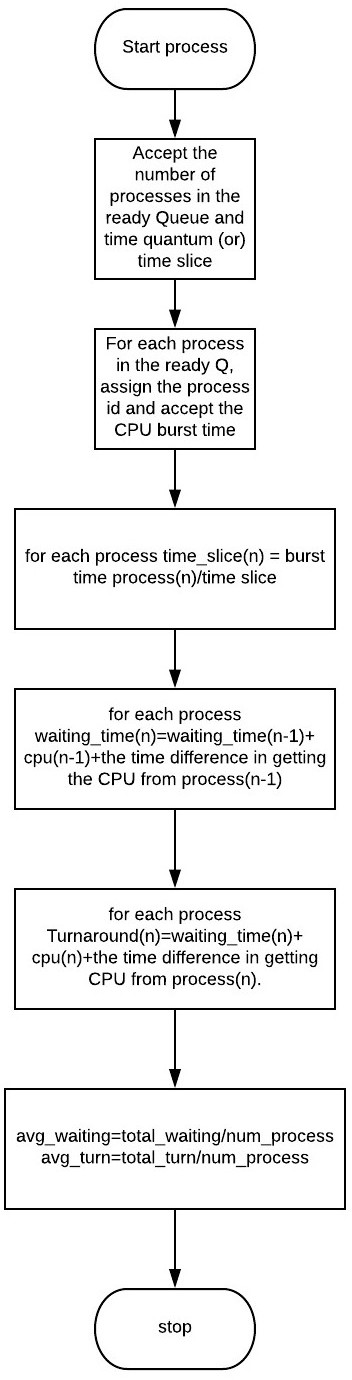


Figure 4: Round Robin

# 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);

}

}

* 1. 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=new SJFS[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);

}

}

* 1. 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);

}

}

* 1. 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();

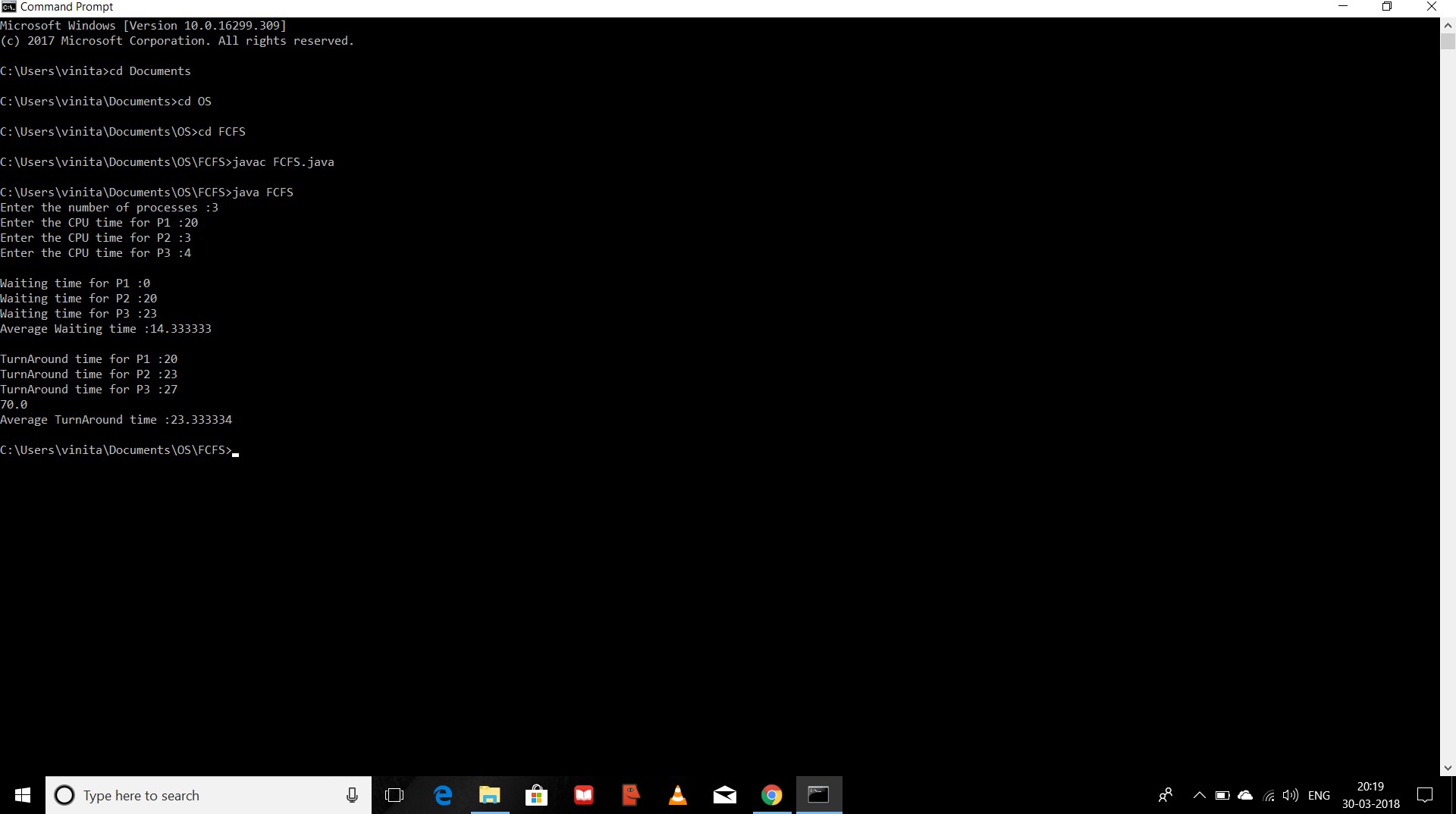
}

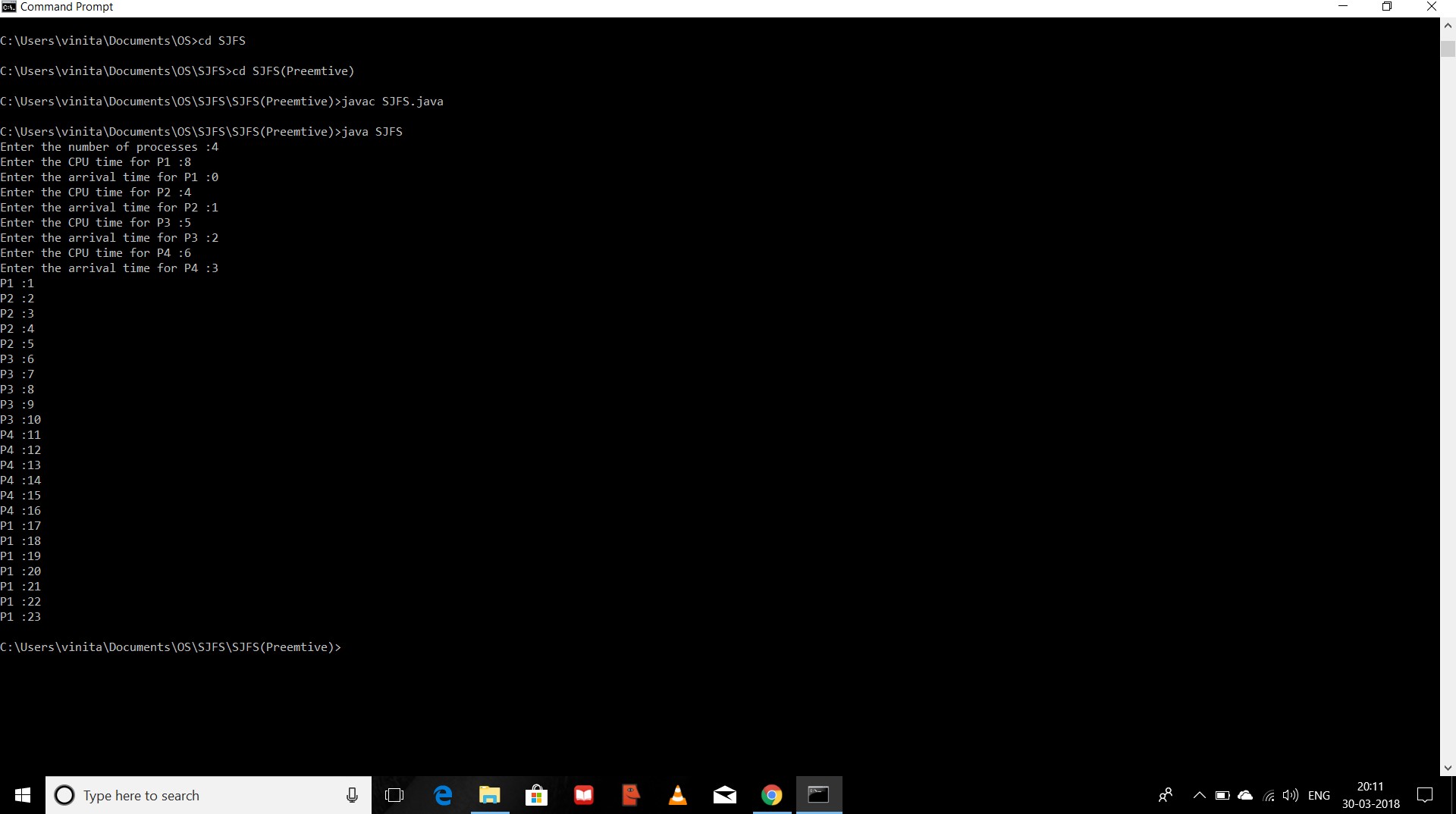
RR.process(obj,num,time\_slice);

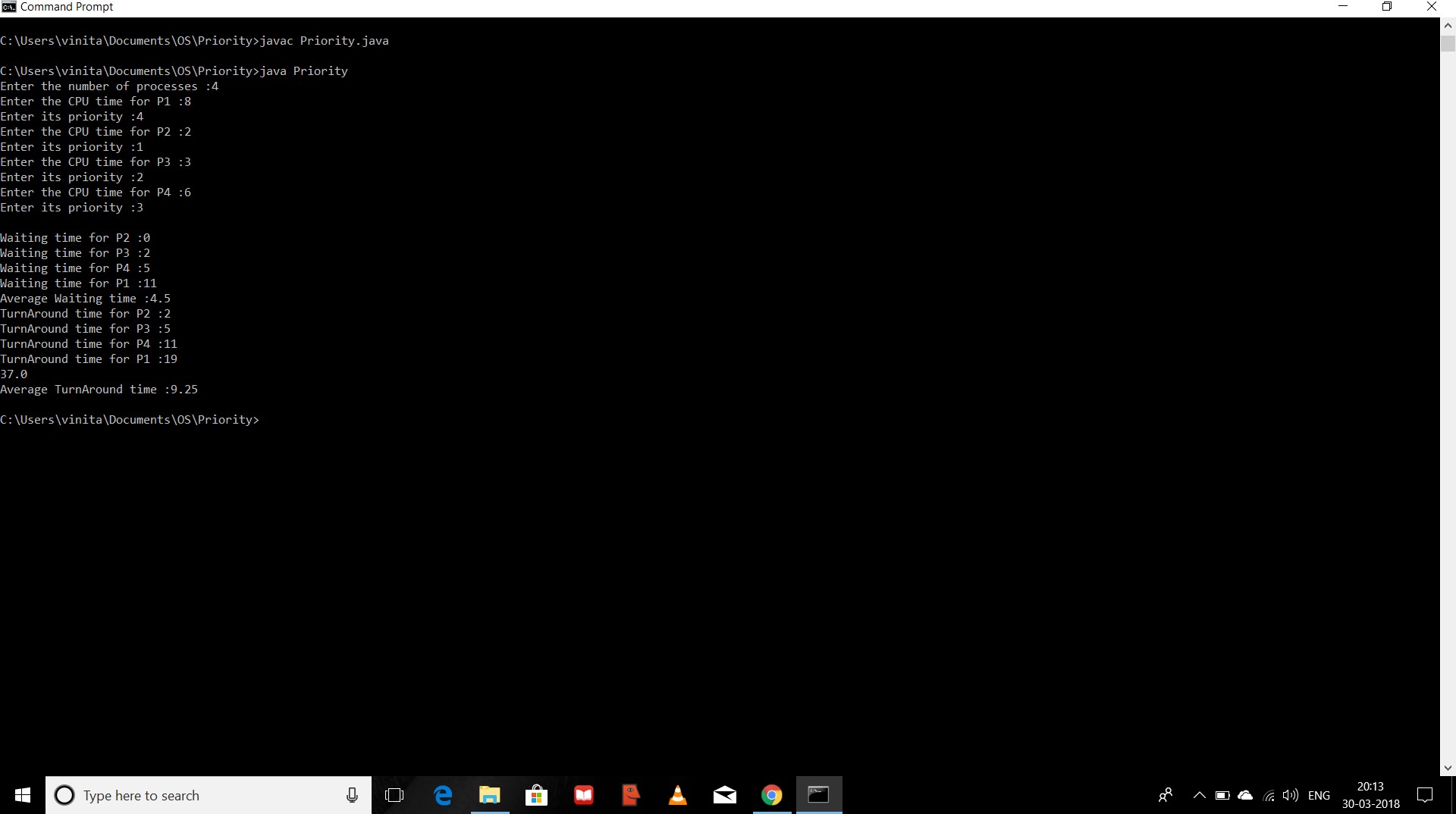
}

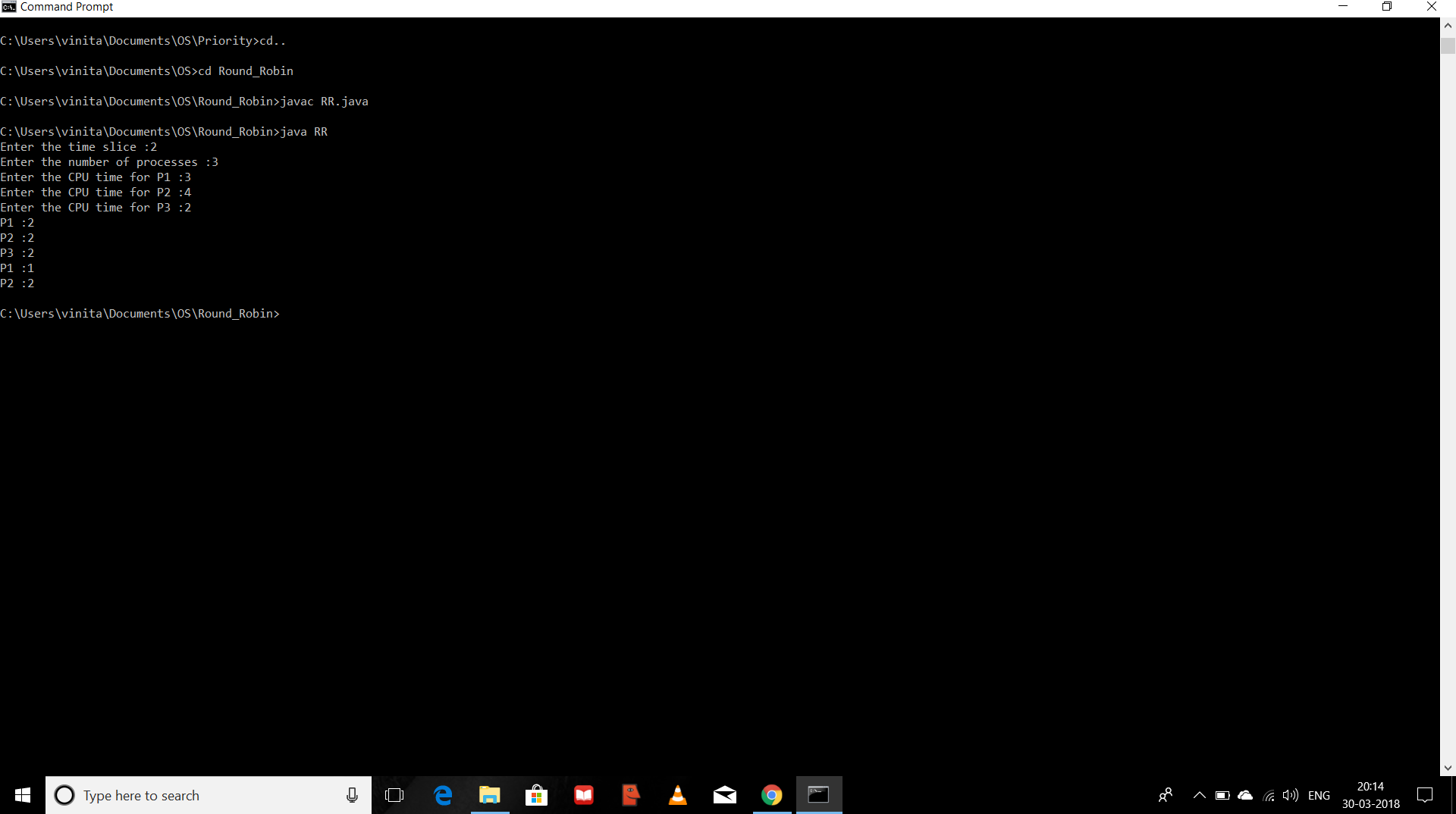
}

# Output:









1. **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.