Lab Sheet 2

Name	:	Lokesh Chandra Basu
E.No.	:	10114026
Course	:	Operating Systems Lab

Question 1

Design and implement a Scheduler using a non-preemptive Shortest Job First (SJF) algorithm.

Algorithm

Algorithm used :

```
//to create the Job Queue
 take input from the user for the number of queues in each of the Job Queue
Initialize count to 0
        While count is less then input number
                generate a random number to be stored in queue as pID
                count = count +1

//to move processes from  the Ready Queue to Running Queue base on their
  Burst Time

if same arrival time
        search for the shortest Burst Time among all the processes and then
        move it to the Running Queue and so on
else
        search for the shortest Burst Time from Waiting Queue and also from
        ready  Queue ab then move that process into Running Queue

do not terminate the program unless the Waiting Queue and Ready Queue are
both empty
```

Gantt Chart

Procees	Execution Tiem(in ms)
P1	5
P2	15
P3	10
P4	7
P5	3

Job Queue with processes and execution time (assuming all arrives at **same time**) :

P1	**P2**	**P3**	**P4**	**P5**

After Long Term Scheduler or Job Scheduler :

Ready Queue :

P5	**P1**	**P4**	**P3**	**P5**

0 3 8 15 25 40

time elapse = 0ms

After Short Term Scheduler or Scheduler Dispatch :

Ready Queue :

P1	**P4**	**P3**	**P5**

3 8 15 25 40

Running Queue :

P5 (Running)

0 3

time elapse = 3ms

After Short Term Scheduler or Scheduler Dispatch :

Ready Queue :

P4	**P3**	**P5**

8 15 25 40

Running Queue :

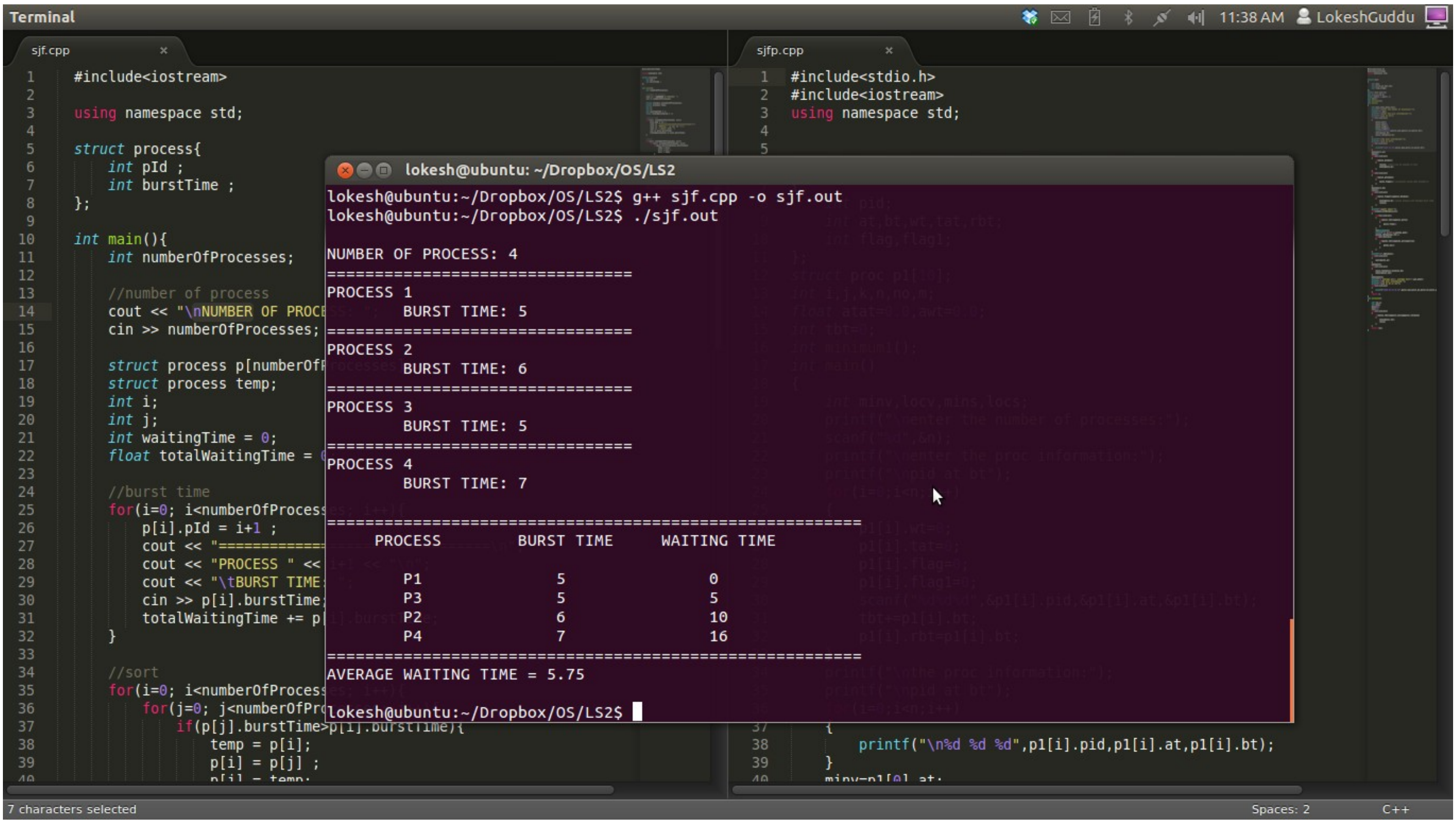
P5 (Terminated)	**P1(Running)**

0 5 20

time elapse = 20ms (just after P2 terminates)

And So on...

Output



```
#include<iostream>

using namespace std;

struct process{
	int pId ;
	int burstTime ;
};

int main(){
	int numberOfProcesses;

	//number of process
	cout << "\nNUMBER OF PROCE
	cin >> numberOfProcesses;

	struct process p[numberOfP
	struct process temp;
	int i;
	int j;
	int waitingTime = 0;
	float totalWaitingTime = 

	//burst time
	for(i=0; i<numberOfProcess
		p[i].pId = i+1 ;
		cout << "============
		cout << "PROCESS " <<
		cout << "\tBURST TIME
		cin >> p[i].burstTime
		totalWaitingTime += p
	}

	//sort
	for(i=0; i<numberOfProcess
		for(j=0; j<numberOfPr
			if(p[j].burstTime>p[i].burstTime){
				temp = p[i];
				p[i] = p[j] ;
```

```
#include<stdio.h>
#include<iostream>
using namespace std;
```

```
	{
		printf("\n%d %d %d",p1[i].pid,p1[i].at,p1[i].bt);
	}
```

lokesh@ubuntu: ~/Dropbox/OS/LS2

```
lokesh@ubuntu:~/Dropbox/OS/LS2$ g++ sjf.cpp -o sjf.out
lokesh@ubuntu:~/Dropbox/OS/LS2$ ./sjf.out

NUMBER OF PROCESS: 4
================================
PROCESS 1
        BURST TIME: 5
================================
PROCESS 2
        BURST TIME: 6
================================
PROCESS 3
        BURST TIME: 5
================================
PROCESS 4
        BURST TIME: 7

========================================================
     PROCESS        BURST TIME     WAITING TIME

        P1              5               0
        P3              5               5
        P2              6               10
        P4              7               16
========================================================
AVERAGE WAITING TIME = 5.75

lokesh@ubuntu:~/Dropbox/OS/LS2$ 
```

gcc version 4.6.3 (Ubuntu/Linaro 4.6.3-1 ubuntu5)

Question 2

Design and implement a Scheduler using a preemptive Shortest Job First (SJF) algorithm.

Algorithm

Algorithm used :

```
//to create the Job Queue
 take input from the user for the number of queues in each of the Job Queue
Initialize count to 0
    While count is less then input number
        generate a random number to be stored in queue as pID
        count = count +1

//to move processes from  the Ready Queue to Running Queue base on their
  Burst Time

if same arrival time
    search for the shortest Burst Time among all the processes and then
    move it to the Running Queue and so on
else
    search for the shortest Burst Time from Waiting Queue and also from
    ready  Queue ab then move that process into Running Queue

do not terminate the program unless the Waiting Queue and Ready Queue are
both empty


//applying preemptive SJF
if different arrival time
    then if any process in Waiting Queue or Ready Queue has less Burst
    Time then the time left for execution of the process in the Running Queue
    then remove the process from Running Queue and move it into Waiting
    Queue and move the shortest process into Running Queue.
```

Gantt Chart

Procees	**Execution Tiem(in ms)**	**Arrival Time(ms)**
P1	**5**	**0**
P2	**15**	**0**
P3	**10**	**5**
P4	**7**	**10**
P5	**3**	**20**

Job Queue with processes and execution time (assuming all arrives at **same time**) :

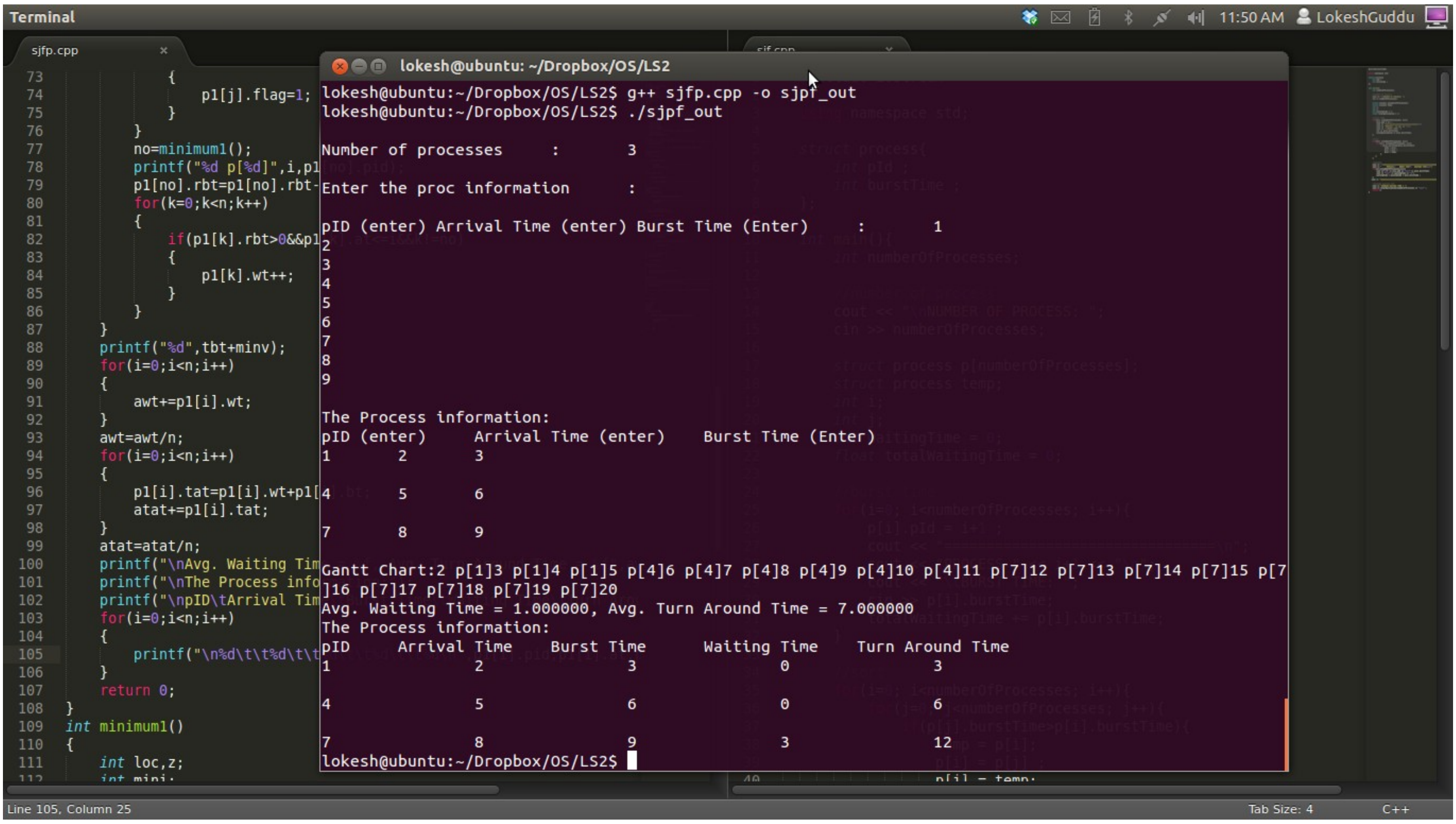
P1	**P2**	**P3**	**P4**	**P5**

Processes Queue after Termination Queue :

P1	**P3**	**P5**	**P4**	**P2**

0 5 15 18 25 40

Output



```
            {
                p1[j].flag=1;
            }
        }
        no=minimum1();
        printf("%d p[%d]",i,p1
        p1[no].rbt=p1[no].rbt-
        for(k=0;k<n;k++)
        {
            if(p1[k].rbt>0&&p1
            {
                p1[k].wt++;
            }
        }
    }
    printf("%d",tbt+minv);
    for(i=0;i<n;i++)
    {
        awt+=p1[i].wt;
    }
    awt=awt/n;
    for(i=0;i<n;i++)
    {
        p1[i].tat=p1[i].wt+p1[
        atat+=p1[i].tat;
    }
    atat=atat/n;
    printf("\nAvg. Waiting Tim
    printf("\nThe Process info
    printf("\npID\tArrival Tim
    for(i=0;i<n;i++)
    {
        printf("\n%d\t\t%d\t\t
    }
    return 0;
}
int minimum1()
{
    int loc,z;
    int mini;
```

```
lokesh@ubuntu:~/Dropbox/OS/LS2$ g++ sjfp.cpp -o sjpf_out
lokesh@ubuntu:~/Dropbox/OS/LS2$ ./sjpf_out

Number of processes     :       3

Enter the proc information      :

pID (enter) Arrival Time (enter) Burst Time (Enter)     :       1
2
3
4
5
6
7
8
9


The Process information:
pID (enter)     Arrival Time (enter)    Burst Time (Enter)
1       2       3

4       5       6

7       8       9

Gantt Chart:2 p[1]3 p[1]4 p[1]5 p[4]6 p[4]7 p[4]8 p[4]9 p[4]10 p[4]11 p[7]12 p[7]13 p[7]14 p[7]15 p[7
]16 p[7]17 p[7]18 p[7]19 p[7]20
Avg. Waiting Time = 1.000000, Avg. Turn Around Time = 7.000000
The Process information:
pID     Arrival Time    Burst Time      Waiting Time    Turn Around Time
1               2               3               0               3

4               5               6               0               6

7               8               9               3               12
lokesh@ubuntu:~/Dropbox/OS/LS2$ 
```

gcc version 4.6.3 (Ubuntu/Linaro 4.6.3-1 ubuntu5)