INDIAN INSTITUTE OF TECHNOLOGY (INDIAN SCHOOL OF MINES) DHANBAD

Department of Computer Science and Engineering Operating System Lab (CSC15202)

Monsoon Semester (2020-21)

Name: Vivek Kumar Verma
Admission Number: 18JE0940

Index

Exp No	Aim	Date	Signature	Remarks
1	Basics ofUNIX Commands.	August 8, 2020		
2	Shell Programming	August 18, 2020		
3	Implementation ofCPU Scheduling. A) FCFS B) SJF	August 25, 2020		
4	Priority Scheduling	September 1, 2020		
5	Round Robin Scheduling	September 1, 2020		
6	Implementation of SJF (Preemptive and Non-preemptive)	September 8, 2020		
7	Multilevel Queue Scheduling Algorithm - I	September 22, 2020		
8	Multilevel Queue Scheduling Algorithm - II	September 22, 2020		
9	Producer & Consumer Algorithm	September 29, 2020		
10	Dining Philosophers Problem	October 6, 2020		
11	Banker's and Deadlock Prevention Algorithm	October 13, 2020		
12	Simulate Page Replacement Algorithms: FIFO	October 20, 2020		
13	Simulate Page Replacement Algorithms: LRU	October 20, 2020		

14	Simulate Fixed Partitioning: First Fit, Best Fit and Worst Fit	October 27, 2020		
15	Simulate Partitioning: First Fit, Best Fit and Worst Fit	October 27, 2020		

Experiment Number 1: Basics of UNIX commands

Date: August 11, 2020

Aim: To write and learn the basic Linux commands.

Linux Command	DOS Command	Description
pwd	cd	"Print WorkingDirectory". Shows thecurrent location in the directory tree.
cd	cd, chdir	"Change Directory".When typed all by itself, it returns you to your home directory.
cd directory	cd directory	Change into the specified directoryname. Example: cd /usr/src/linux
cd ~		"~" is an alias for your home directory. It can be used as a shortcut to your "home", orother directories relative to your home.
cd..	cd..	Move up one directory. For example, if you are in /home/vic and you type "cd ..", you will end up in /home.
cd -		Return to previous directory. An easy way to get back to your previous location!
ls	dir /w	List all files in the current directory, in column format.
ls directory	dir directory	List the files in the specified directory. Example: ls /var/log
ls -l	dir	List files in "long" format, one file per line. This also shows you additional info about the file, such as ownership, permissions, date, and size.
ls -a	dir /a	List all files, including "hidden" files. Hidden files are those files that begin with a ".", e.g.The .bash_history file in your home directory.
ls -ld directory		A "long" list of "directory", but instead of showing the directory contents, show the directory's detailed information. For example, compare the output of the following twocommands: ls -l /usr/bin ls -ld /usr/bin
ls /usr/bin/d*	dir d*.*	List all files whose names begin with the letter "d" in the /usr/bindirectory.

Linux Command	*DOS Command*	*Description*
pwd	cd	"Print WorkingDirectory". Shows thecurrent location in the directory tree.
cd	cd, chdir	"Change Directory".When typed all by itself, it returns you to your home directory.
cd directory	cd directory	Change into the specified directoryname. Example: cd /usr/src/linux
cd ~		"~" is an alias for your home directory. It can be used as a shortcut to your "home", orother directories relative to your home.
cd..	cd..	Move up one directory. For example, if you are in /home/vic and you type "cd ..", you will end up in /home.
cd -		Return to previous directory. An easy way to get back to your previous location!
ls	dir /w	List all files in the current directory, in column format.
ls directory	dir directory	List the files in the specified directory. Example: ls /var/log
ls -l	dir	List files in "long" format, one file per line. This also shows you additional info about the file, such as ownership, permissions, date, and size.
ls -a	dir /a	List all files, including "hidden" files. Hidden files are those files that begin with a ".", e.g.The .bash_history file in your home directory.
ls -ld directory		A "long" list of "directory", but instead of showing the directory contents, show the directory's detailed information. For example, compare the output of the following twocommands: ls -l /usr/bin ls -ld /usr/bin
ls /usr/bin/d*	dir d*.*	List all files whose names begin with the letter "d" in the /usr/bindirectory.

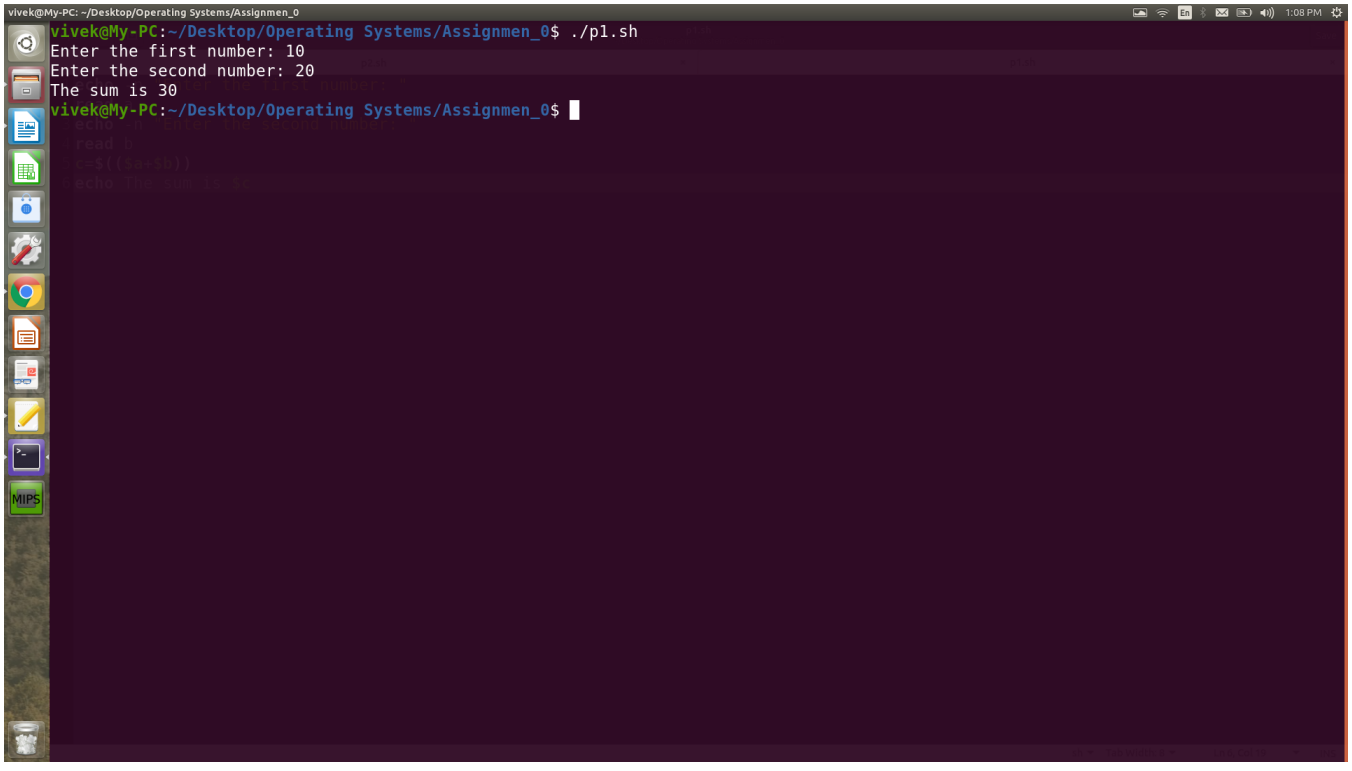
Experiment Number 2: Shell programming

Date: August 18, 2020

Aim: To write the given programs to implement Shell Programming.

Q1.Write a shell program to add two numbers.
Sol:

```
echo -n "Enter the first number: "
read a
echo -n "Enter the second number: "
read b
c=$(($a+$b))
echo The sum is $c
```

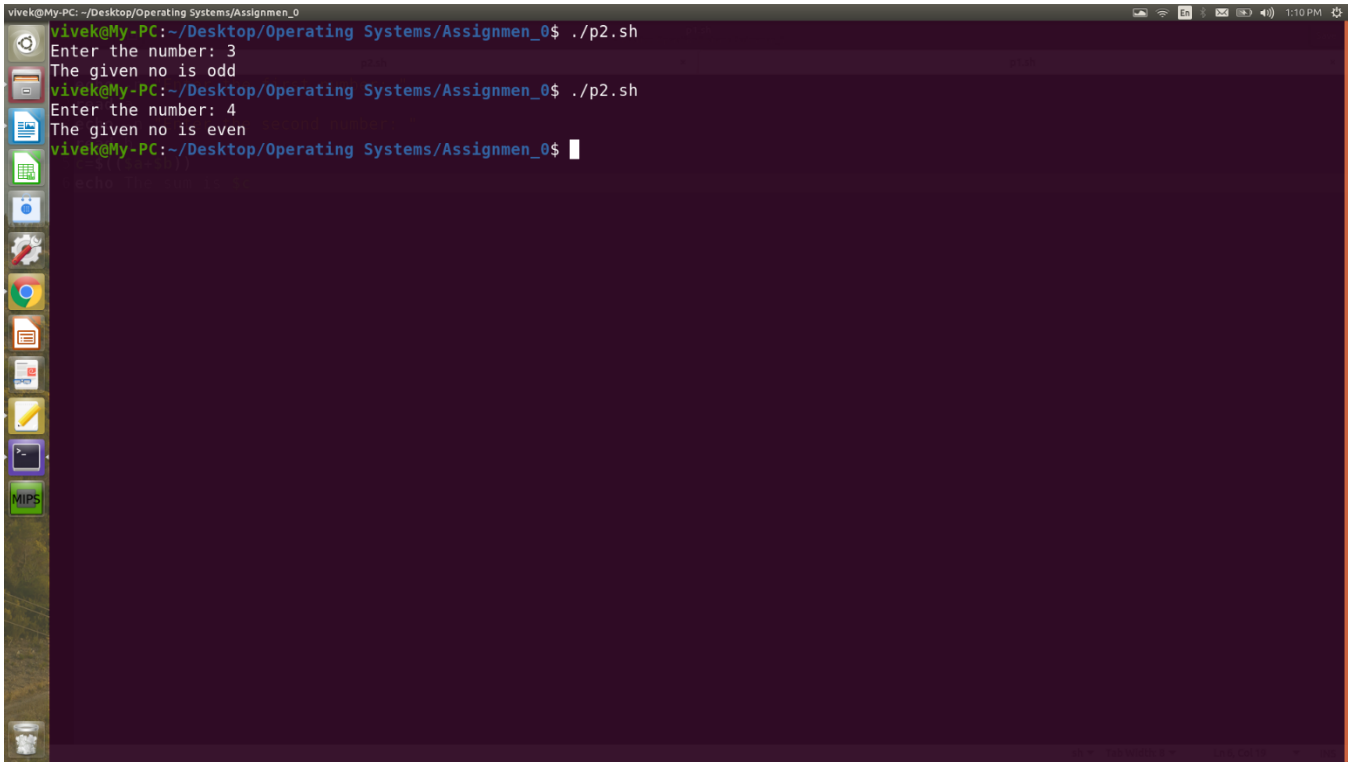


Q2 Write a shell program to find whether a number is even or odd.
Sol:

```
echo -n "Enter the number: "
read x
b=$(($x%2))
if [ $b -eq 0 ]; then
echo The given no is even
```

```
else
echo The given no is odd
fi
```

Output

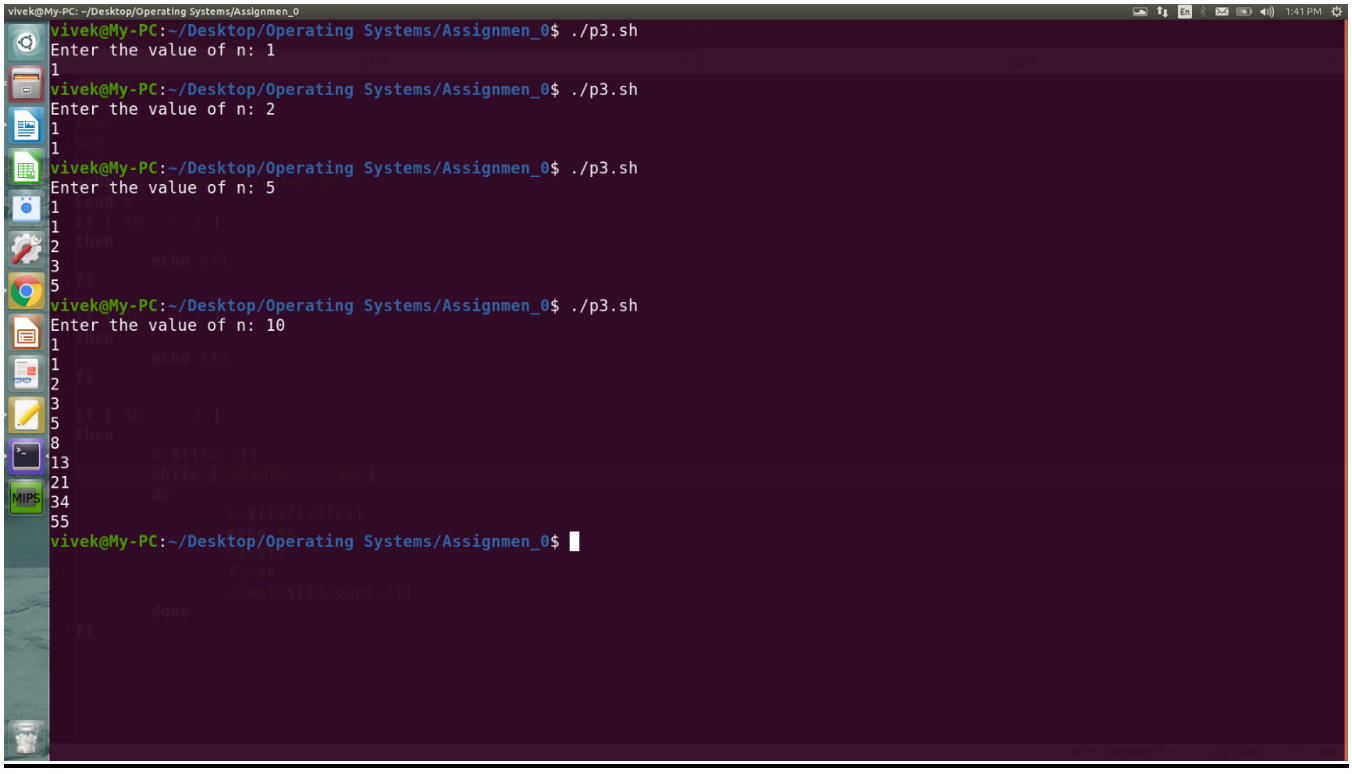


Q3 Write a shell program for Fibonacci series.
Sol.

```
count=1
f1=1
f2=1
k=0
n=0
echo -n "Enter the value of n: "
read n
if [ $n -ge 1 ]
then
        echo $f1
fi

if [ $n -ge 2 ]
then
        echo $f2
fi
```

```
if [ $n -gt 2 ]
then
        while [ $count -le $(($n-2)) ]
        do
                k=$(($f1+$f2))
                echo $k
                f1=$f2
                f2=$k
                count=$(($count+1))
        done
fi
```

Output



Experiment Number 3: Implementation of CPU Scheduling.

Date: August 25, 2020

Aim: To write the program to implement following CPU scheduling algorithms.

1. FCFS
2. SJF

1. Algorithm for FCFS

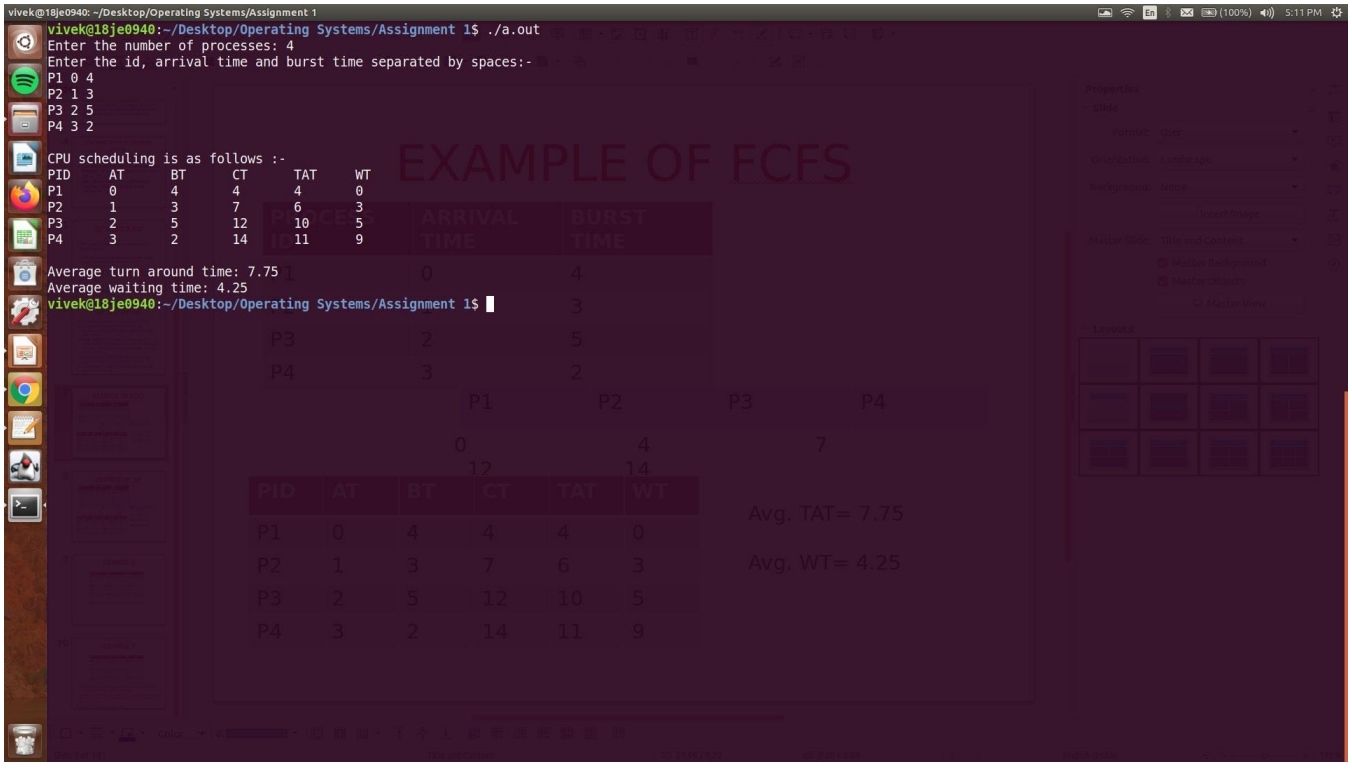
1.	Start the program.
2.	Get the number of processes, arrival time and burst times. (also ID's if required)
3.	Compute the response time, completion time, waiting time and turnaround time.
4.	The waiting time of all the processes is summed then average value time is calculated.
5.	The waiting time of each process and average times are displayed
6.	Same is done for turnaround time as well.
7.	Stop the program

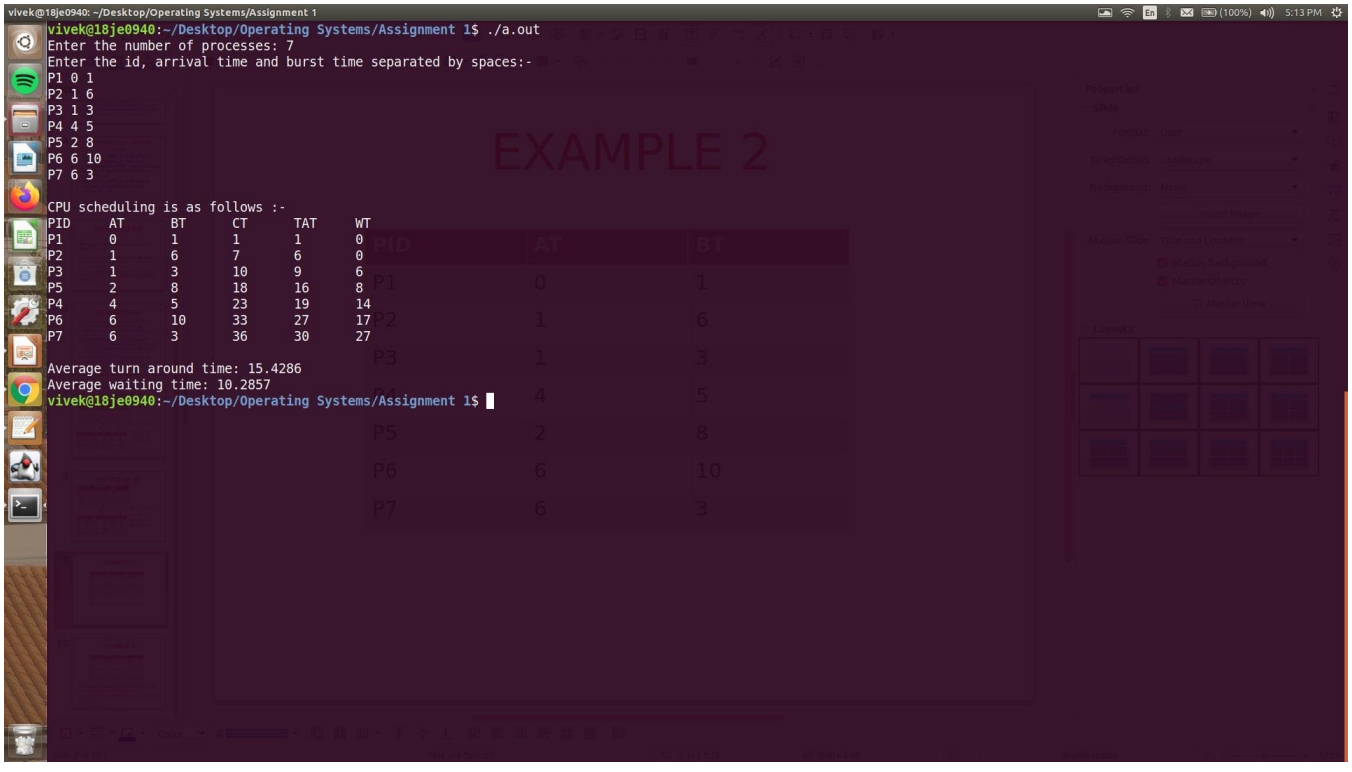
C++ implementation

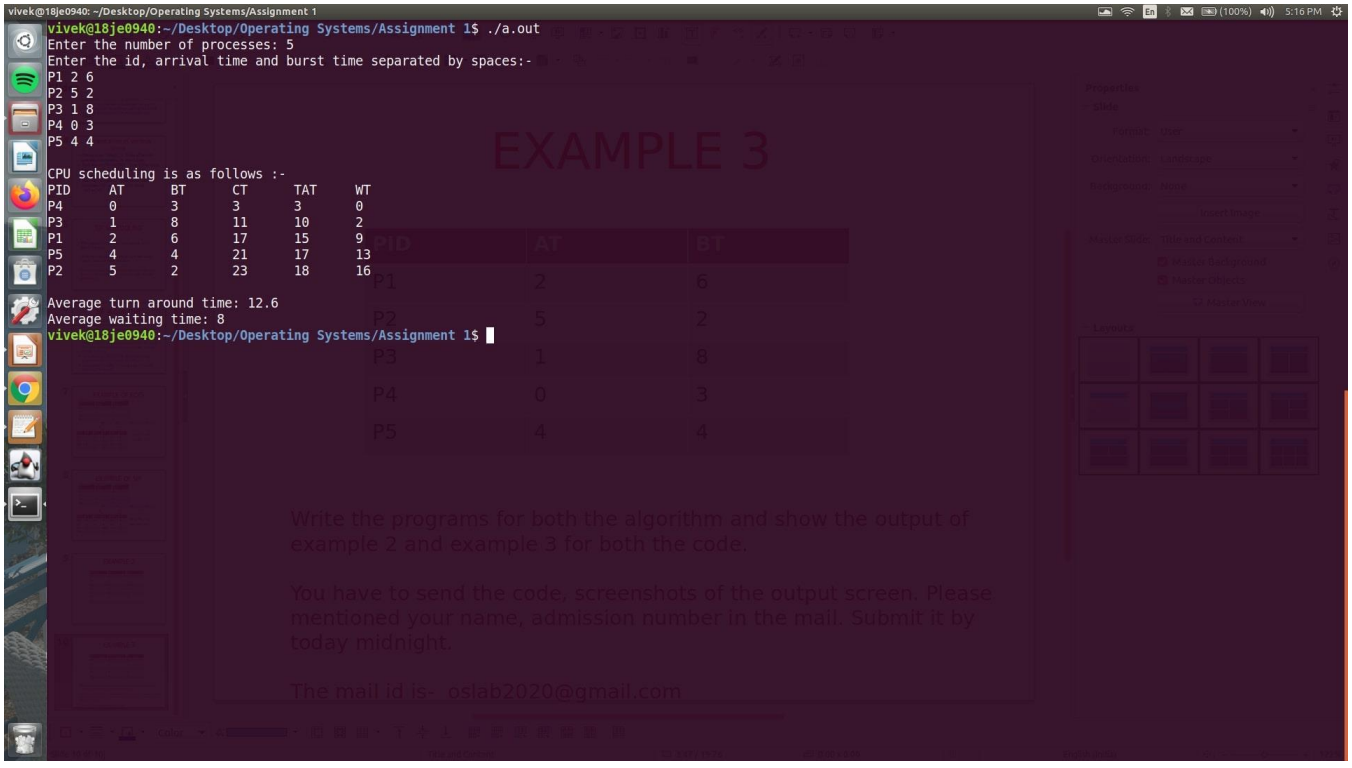
```
#include<bits/stdc++.h>

using namespace std;

struct process
{
	int at,bt,wt,ct,tat;
	string id;

	process()
	{
		id = "";
		at = bt = wt = ct = tat = -1;
	}
};

bool compare(process p1, process p2)
{
	if(p1.at==p2.at)
	{
		return p1.id < p2.id;
	}
	return p1.at < p2.at;
}

int main()
```

```
{
	int n,i;
	cout<<"Enter the number of processes: ";
	cin>>n;
	process p1[n];
	cout<<"Enter the id, arrival time and burst time separated by spaces:-"<<endl;
	for(i=0;i<n;++i)
	{
		cin>>p1[i].id>>p1[i].at>>p1[i].bt;
	}
	sort(p1,p1+n,compare);
	int time = 0;
	float tat_av = 0, wat_avg = 0;
	for(i=0;i<n;++i)
	{
		time += p1[i].bt;
		p1[i].ct = time;
		p1[i].tat = p1[i].ct - p1[i].at;
		p1[i].wt = p1[i].tat - p1[i].bt;
		tat_av += p1[i].tat;
		wat_avg += p1[i].wt;
	}
	cout<<"\nCPU scheduling is as follows :-\n";
	cout<<"PID\tAT\tBT\tCT\tTAT\tWT\n";
	for(i=0;i<n;++i)
	{

	cout<<p1[i].id<<"\t"<<p1[i].at<<"\t"<<p1[i].bt<<"\t"<<p1[i].ct<<"\t"<<p1[i].tat<<"\t"<<
p1[i].wt<<endl;
	}
	cout<<endl;
	cout<<"Average turn around time: "<<tat_av / n<<endl;
	cout<<"Average waiting time: "<<wat_avg / n<<endl;
	return 0;
}
```

Output


vivek@18je0940:~/Desktop/Operating Systems/Assignment 1$ ./a.out
Enter the number of processes: 4
Enter the id, arrival time and burst time separated by spaces:-
P1 0 4
P2 1 3
P3 2 5
P4 3 2
CPU scheduling is as follows :-
PID AT BT CT TAT WT
P1 0 4 4 4 0
P2 1 3 7 6 3
P3 2 5 12 10 5
P4 3 2 14 11 9
Average turn around time: 7.75
Average waiting time: 4.25
vivek@18je0940:~/Desktop/Operating Systems/Assignment 1$


vivek@18je0940:~/Desktop/Operating Systems/Assignment 1$ ./a.out
Enter the number of processes: 7
Enter the id, arrival time and burst time separated by spaces:-
P1 0 1
P2 1 6
P3 1 3
P4 4 5
P5 2 8
P6 6 10
P7 6 3
CPU scheduling is as follows :-
PID AT BT CT TAT WT
P1 0 1 1 1 0
P2 1 6 7 6 0
P3 1 3 10 9 6
P5 2 8 18 16 8
P4 4 5 23 19 14
P6 6 10 33 27 17
P7 6 3 36 30 27
Average turn around time: 15.4286
Average waiting time: 10.2857
vivek@18je0940:~/Desktop/Operating Systems/Assignment 1$


vivek@18je0940: ~/Desktop/Operating Systems/Assignment 1
vivek@18je0940:~/Desktop/Operating Systems/Assignment 1$ ./a.out
Enter the number of processes: 5
Enter the id, arrival time and burst time separated by spaces:-
P1 2 6
P2 5 2
P3 1 8
P4 0 3
P5 4 4
CPU scheduling is as follows :-
PID AT BT CT TAT WT
P4 0 3 3 3 0
P3 1 8 11 10 2
P1 2 6 17 15 9
P5 4 4 21 17 13
P2 5 2 23 18 16
Average turn around time: 12.6
Average waiting time: 8
vivek@18je0940:~/Desktop/Operating Systems/Assignment 1$
EXAMPLE 3
PID AT BT
P1 2 6
P2 5 2
P3 1 8
P4 0 3
P5 4 4
Write the programs for both the algorithm and show the output of example 2 and example 3 for both the code.
You have to send the code, screenshots of the output screen. Please mentioned your name, admission number in the mail. Submit it by today midnight.
The mail id is- oslab2020@gmail.com

2. Algorithm for SJF (non-preemptive)

1.	Start the program.
2.	Get the number of processes, arrival time and burst times. (also ID's if required)
3.	Compute the response time, completion time, waiting time and turnaround time.
4.	The waiting time of all the processes is summed then average value time is calculated.
5.	The waiting time of each process and average times are displayed
6.	Same is done for turnaround time as well.
7.	Stop the program

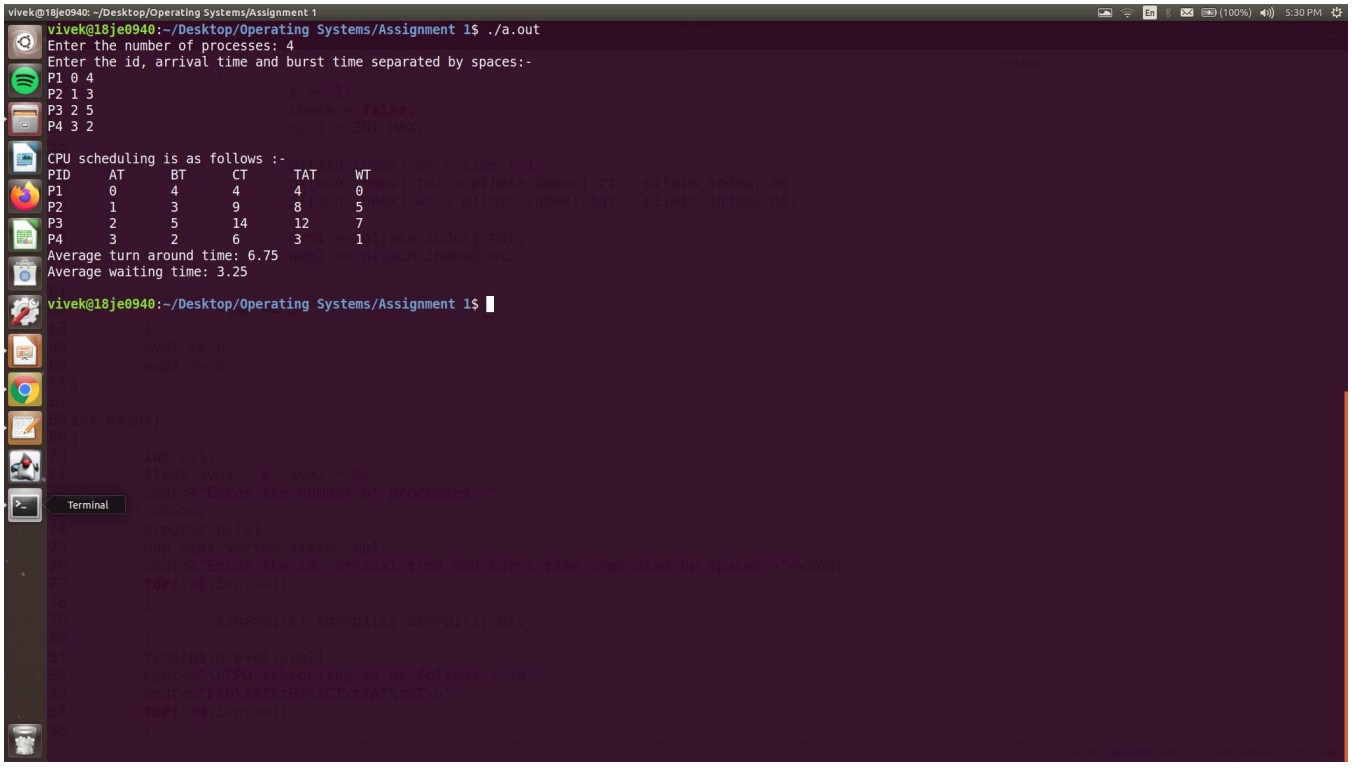
C++ implementation

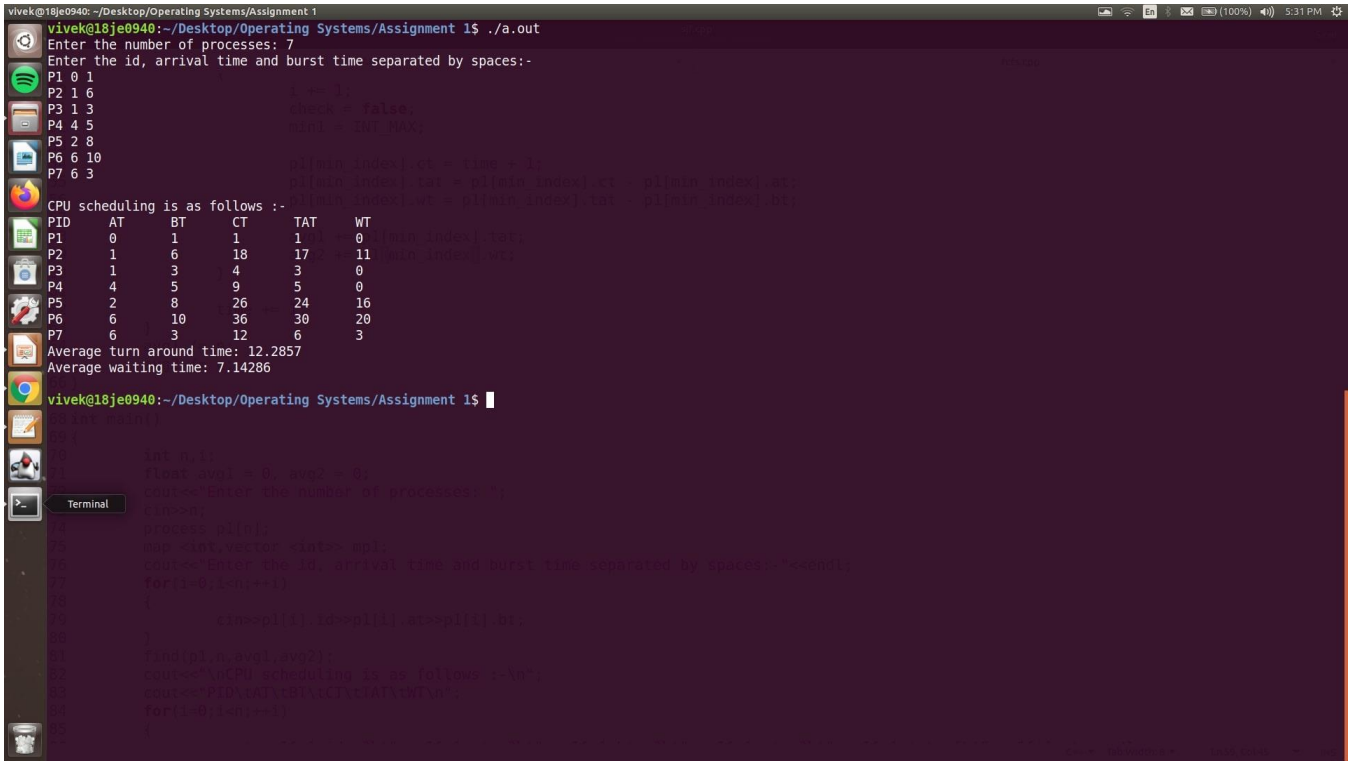
```
#include<bits/stdc++.h>

using namespace std;

struct job
{
	int at, bt, st, ct, tat, wt;
	string id;
};

void non_preemptive(job j1[], float &avg1, float &avg2, int n)
{
	avg1 = 0;
	avg2 = 0;
	int rem[n],i;
	for(i=0;i<n;++i)
	{
		rem[i] = j1[i].bt;
	}
	int completed = 0,time = 0;
	while(completed<n)
	{
		int min_time = INT_MAX,min_index = -1;
		for(i=0;i<n;++i)
		{
			if(j1[i].at<=time and rem[i]>0 and rem[i]<min_time)
			{
				min_time = rem[i];
				min_index = i;
			}
		}
		if(min_index==-1)
		{
```

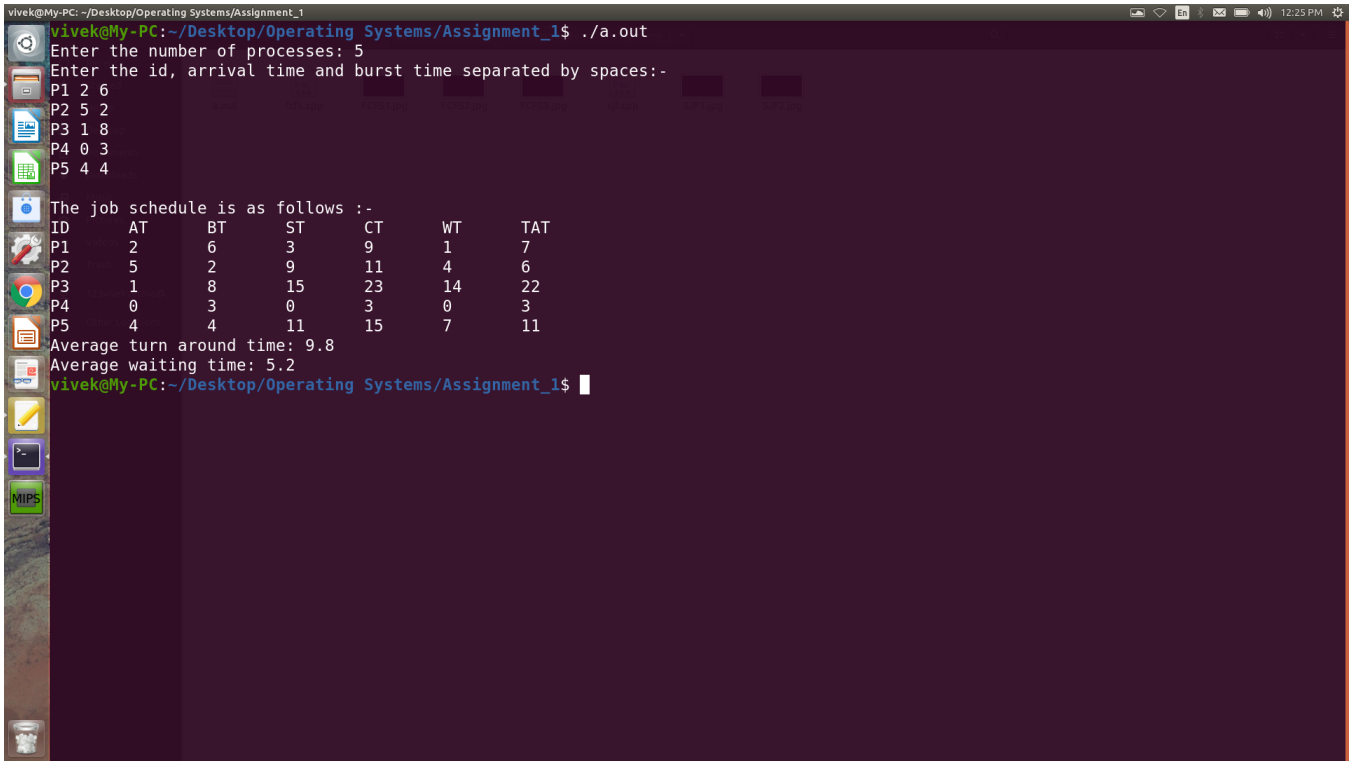
```
                time = time + 1;
                continue;
            }
            j1[min_index].st = time;
            j1[min_index].ct = time + j1[min_index].bt;
            rem[min_index] = 0;

            time = time + j1[min_index].bt;

            min_time = rem[min_index];

            completed += 1;
            min_time = INT_MAX;

            j1[min_index].tat = j1[min_index].ct - j1[min_index].at;
            j1[min_index].wt = j1[min_index].tat - j1[min_index].bt;

            avg1 += j1[min_index].tat;
            avg2 += j1[min_index].wt;

            //time = time + 1;
        }
        avg1 = avg1 / n;
        avg2 = avg2 / n;
        cout<<"ID\tAT\tBT\tST\tCT\tWT\tTAT"<<endl;
        for(i=0;i<n;++i)
        {

        cout<<j1[i].id<<"\t"<<j1[i].at<<"\t"<<j1[i].bt<<"\t"<<j1[i].st<<"\t"<<j1[i].ct<<"\t"<<j1[i
].wt<<"\t"<<j1[i].tat<<endl;
        }
}

int main()
{
        int n,i;
        cout<<"Enter the number of processes: ";
        cin>>n;
        job j1[n];
        cout<<"Enter the id, arrival time and burst time separated by spaces:-"<<endl;
        for(i=0;i<n;++i)
        {
                cin>>j1[i].id>>j1[i].at>>j1[i].bt;
        }
        float avg1,avg2;
        cout<<"\nThe job schedule is as follows :-\n";
        non_preemptive(j1,avg1,avg2,n);
```

```
        cout<<"Average turn around time: "<<avg1<<endl;
        cout<<"Average waiting time: "<<avg2<<endl;
        return 0;
    }
```

Output




vivek@18je0940: ~/Desktop/Operating Systems/Assignment 1
vivek@18je0940:~/Desktop/Operating Systems/Assignment 1$ ./a.out
Enter the number of processes: 7
Enter the id, arrival time and burst time separated by spaces:-
P1 0 1
P2 1 6
P3 1 3
P4 4 5
P5 2 8
P6 6 10
P7 6 3
CPU scheduling is as follows :-
PID AT BT CT TAT WT
P1 0 1 1 1 0
P2 1 6 18 17 11
P3 1 3 4 3 0
P4 4 5 9 5 0
P5 2 8 26 24 16
P6 6 10 36 30 20
P7 6 3 12 6 3
Average turn around time: 12.2857
Average waiting time: 7.14286
vivek@18je0940:~/Desktop/Operating Systems/Assignment 1$
Terminal


vivek@My-PC: ~/Desktop/Operating Systems/Assignment_1
vivek@My-PC:~/Desktop/Operating Systems/Assignment_1$ ./a.out
Enter the number of processes: 5
Enter the id, arrival time and burst time separated by spaces:-
P1 2 6
P2 5 2
P3 1 8
P4 0 3
P5 4 4
The job schedule is as follows :-
ID AT BT ST CT WT TAT
P1 2 6 3 9 1 7
P2 5 2 9 11 4 6
P3 1 8 15 23 14 22
P4 0 3 0 3 0 3
P5 4 4 11 15 7 11
Average turn around time: 9.8
Average waiting time: 5.2
vivek@My-PC:~/Desktop/Operating Systems/Assignment_1$
MIPS

Result: The following CPU scheduling algorithms were implemented successfully.

1. First Come First Serve
2. Shortest Job First (Non-preemptive)

Experiment Number 4: Implementation of priority scheduling.

Date: September 1, 2020

Aim: To write the program to implement priority CPU scheduling.

Algorithm for Priority Scheduling

1.	Start the program.
2.	Get the number of processes, arrival time, burst times and priority. (also ID's if required)
3.	Compute the response time, completion time, waiting time and turnaround time based on priorities.
4.	The waiting time of all the processes is summed then average value time is calculated.
5.	The waiting time of each process and average times are displayed
6.	Same is done for turn around time as well.
7.	Stop the program

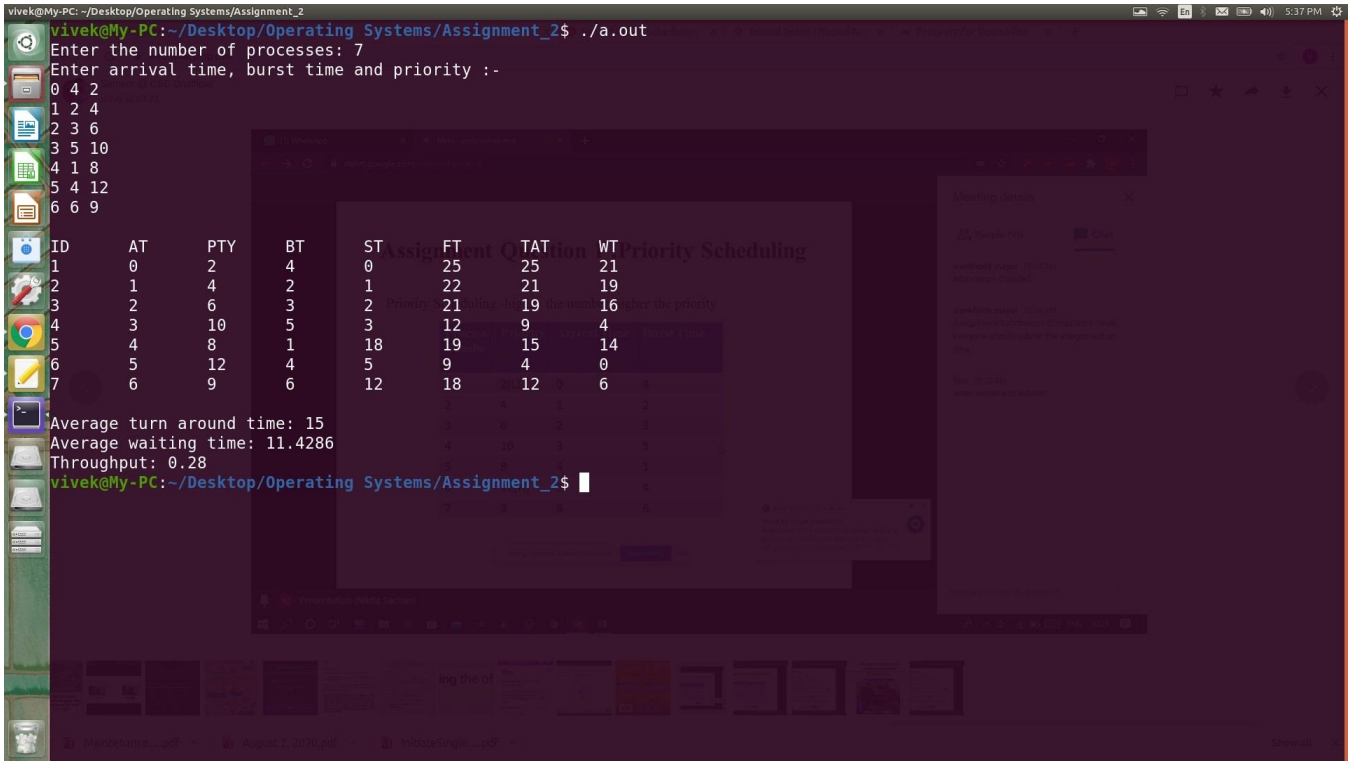
C++ implementation

```
#include<bits/stdc++.h>

using namespace std;

struct process
{
	int id,at,bt,priority,st,ft,tat,wt;
};

int main()
{
	int n,i;
	cout<<"Enter the number of processes: ";
	cin>>n;
	process p1[n];
	int remaining[n];
	cout<<"Enter arrival time, burst time and priority :-\n";
	for(i=0;i<n;++i)
	{
		cin>>p1[i].at>>p1[i].bt>>p1[i].priority;
		p1[i].id = i + 1;
		remaining[i] = p1[i].bt;
	}
	int completed = 0,time = 0;
	bool visited[n] = {};
	float avg1 = 0,avg2 = 0;
	int t1 = INT_MAX,t2 = INT_MIN;
```

```
while(completed<n)
{
	int min1 = INT_MIN,idx = -1;
	for(i=0;i<n;++i)
	{
		if(p1[i].priority > min1 and remaining[i]>0 and p1[i].at <= time)
		{
			min1 = p1[i].priority;
			idx = i;
		}
		if(p1[i].priority==min1 and remaining[i]>0 and p1[i].at <= time)
		{
			if(p1[idx].at>p1[i].at)
			{
				idx = i;
			}
		}
	}
	if(idx==-1)
	{
		time += 1;
		continue;
	}
	if(visited[idx]==false)
	{
		p1[idx].st = time;
		visited[idx] = true;
	}
	time += 1;
	remaining[idx] -= 1;
	if(remaining[idx]==0)
	{
		completed += 1;
		p1[idx].ft = time;
		p1[idx].tat = p1[idx].ft - p1[idx].at;
		p1[idx].wt = p1[idx].tat - p1[idx].bt;
		avg1 += p1[idx].tat;
		avg2 += p1[idx].wt;
		t1 = min(t1,p1[idx].at);
		t2 = max(t2,p1[idx].ft);
	}
}
cout<<"\nID\tAT\tPTY\tBT\tST\tFT\tTAT\tWT\n";
for(i=0;i<n;++i)
{

cout<<p1[i].id<<"\t"<<p1[i].at<<"\t"<<p1[i].priority<<"\t"<<p1[i].bt<<"\t"<<p1[i].st<<"
```

```
\t"<<p1[i].ft<<"\t"<<p1[i].tat<<"\t"<<p1[i].wt<<"\r"<<endl;
        }
        cout<<"\nAverage turn around time: "<<avg1 / n<<endl;
        cout<<"Average waiting time: "<<avg2 / n<<endl;
        cout<<"Throughput: "<<(float)n / (float)(t2-t1)<<endl;
        return 0;
}
```

Output



Result: Priority CPU scheduling algorithm was implemented and verified successfully

Experiment Number 5: Implementation of Round Robin Scheduling

Date: September 1, 2020

Aim: To write the program to implement round robin scheduling.

Algorithm for Priority Scheduling

1.	Get the number of process and their burst time and two different quantum times and do steps 2 through 6 for both.
2.	Initialize the array for Round Robin circular queue as '0' in both cases.
3.	The burst time of each process is divided and the quotients are stored on the round Robinarray.
4.	According to the array value the waiting time for each process andthe average time are calculated as line the otherscheduling.
5.	The waiting time for each process and average times are displayed.
6.	Same is done for turnaround time as well.
7.	Compare the average waiting time, turnaround time and throughput for both time quantums.
8.	Stop the program

C++ Implementation

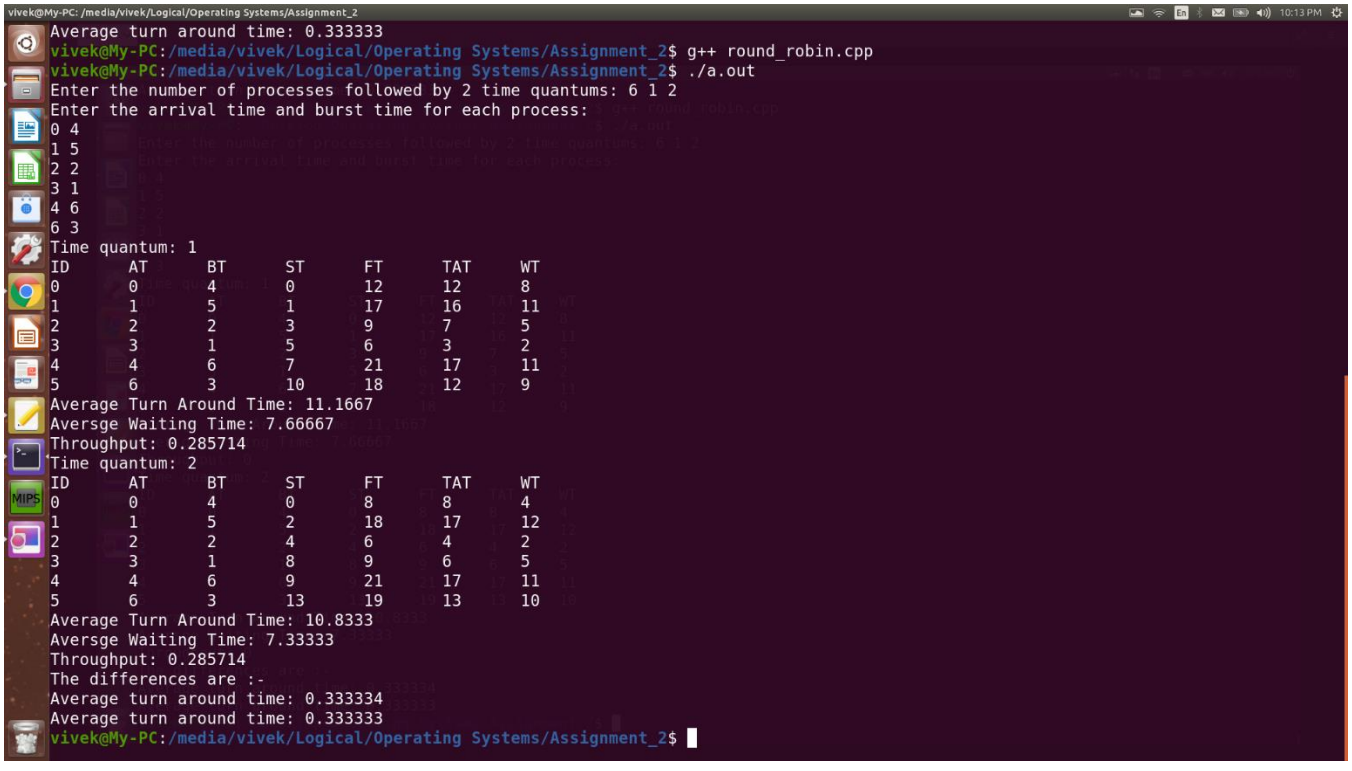
```
#include<bits/stdc++.h>

using namespace std;

struct process
{
	int at,bt,tat,wt,ct,rt,id;
	process()
	{
		at = bt = tat = wt = ct = rt = id;
	}
};

void update_queue(queue <int>&q1, process p1[], int time, int n, bool pushed[])
{
	int i;
	for(i=0;i<n;++i)
	{
		if(p1[i].at<=time and pushed[i]==false)
		{
			q1.push(p1[i].id);
			pushed[i] = true;
		}
	}
}
```

```
void round_robin(process p1[], int n, int q, float &avg1, float &avg2)
{
	avg1 = 0, avg2 = 0;
	int time1 = 0,completed = 0;
	queue <int> ready;
	int rem[n],i,j;
	bool pushed[n] = {}, visited[n] = {};
	for(i=0;i<n;++i)
	{
		rem[i] = p1[i].bt;
	}
	update_queue(ready,p1,time1,n,pushed);
	while(completed<n)
	{
		if(ready.empty()==false)
		{
			int temp = ready.front();
			ready.pop();
			if(visited[temp]==false)
			{
				visited[temp] = true;
				p1[temp].rt = time1;
			}
			if(rem[temp]<=q)
			{
				//updating time variable
				time1 = time1 + rem[temp];
				//assigning completion time
				p1[temp].ct = time1;
				//updating remaining time
				rem[temp] = 0;
				//computing turn around time
				p1[temp].tat = p1[temp].ct - p1[temp].at;
				//computing waiting time
				p1[temp].wt = p1[temp].tat - p1[temp].bt;
				//updatint complete variable
				completed = completed + 1;
				//computing the averages
				avg1 += p1[temp].tat;
				avg2 += p1[temp].wt;
				//updating ready queue
				update_queue(ready,p1,time1,n,pushed);
			}
			else
			{
				//updating remaning time
```

```
                        rem[temp] = rem[temp] - q;
                        //updating time variable
                        time1 = time1 + q;
                        update_queue(ready,p1,time1,n,pushed);
                        ready.push(temp);
                    }
                }
                else
                {
                    time1 += q;
                    update_queue(ready,p1,time1,n,pushed);
                }
        }
        avg1 /= n;
        avg2 /= n;
        cout<<"Time quantum: "<<q<<endl;
        cout<<"ID\tAT\tBT\tST\tFT\tTAT\tWT\n";
        for(i=0;i<n;++i)
        {
        cout<<p1[i].id<<"\t"<<p1[i].at<<"\t"<<p1[i].bt<<"\t"<<p1[i].rt<<"\t"<<p1[i].ct<<"\t"<<
p1[i].tat<<"\t"<<p1[i].wt<<"\r"<<endl;
        }
        cout<<"Average Turn Around Time: "<<avg1<<endl;
        cout<<"Aversge Waiting Time: "<<avg2<<endl;
        int t1,t2;
        t1 = INT_MAX;
        t2 = INT_MIN;
        for(i=0;i<n;++i)
        {
            t1 = min(p1[i].at,t1);
            t2 = max(p1[i].ct,t2);
        }
        cout<<"Throughput: "<<1.0 * n / (t2 - t1)<<endl;
}

int main()
{
        int n,q1,q2,i;
        cout<<"Enter the number of processes followed by 2 time quantums: ";
        cin>>n>>q1>>q2;
        process p1[n];
        cout<<"Enter the arrival time and burst time for each process: "<<endl;
        for(i=0;i<n;++i)
        {
            p1[i].id = i;
            cin>>p1[i].at>>p1[i].bt;
        }
```

```
        float a,b,c,d;
        round_robin(p1,n,q1,a,b);
        round_robin(p1,n,q2,c,d);
        cout<<"The differences are :-\n";
        cout<<"Average turn around time: "<<abs(c-a)<<endl;
        cout<<"Average turn around time: "<<abs(d-b)<<endl;
        return 0;
    }
```

Output



Result: Round Robin CPU scheduling algorithm was implemented and verified successfully.

Experiment Number 6: Implementation of SJF (Preemptive & Non-preemptive Scheduling)

Date: September 8, 2020

Aim: To write the program to implement preemptive and non-preemptive scheduling and compare the results of the two.

Algorithm for Preemptive SJF

1.	Get the number of process and their burst time, arrival time and IDs if needed.
2.	Select the next job from the ready queue which has the shortest remaining time.
3.	If in between a new process arrives with even shorter remaining time, the CPU turns to that process immediately.
4.	Computer waiting time, turnaround time and response time for each process.
5.	The waiting time for each process and average times are displayed.
6.	Same is done for turnaround time as well.
7.	Compute the above quantities using non-preemptive algorithm as well and compare the values.
7.	Stop the program.

C++ Implementation

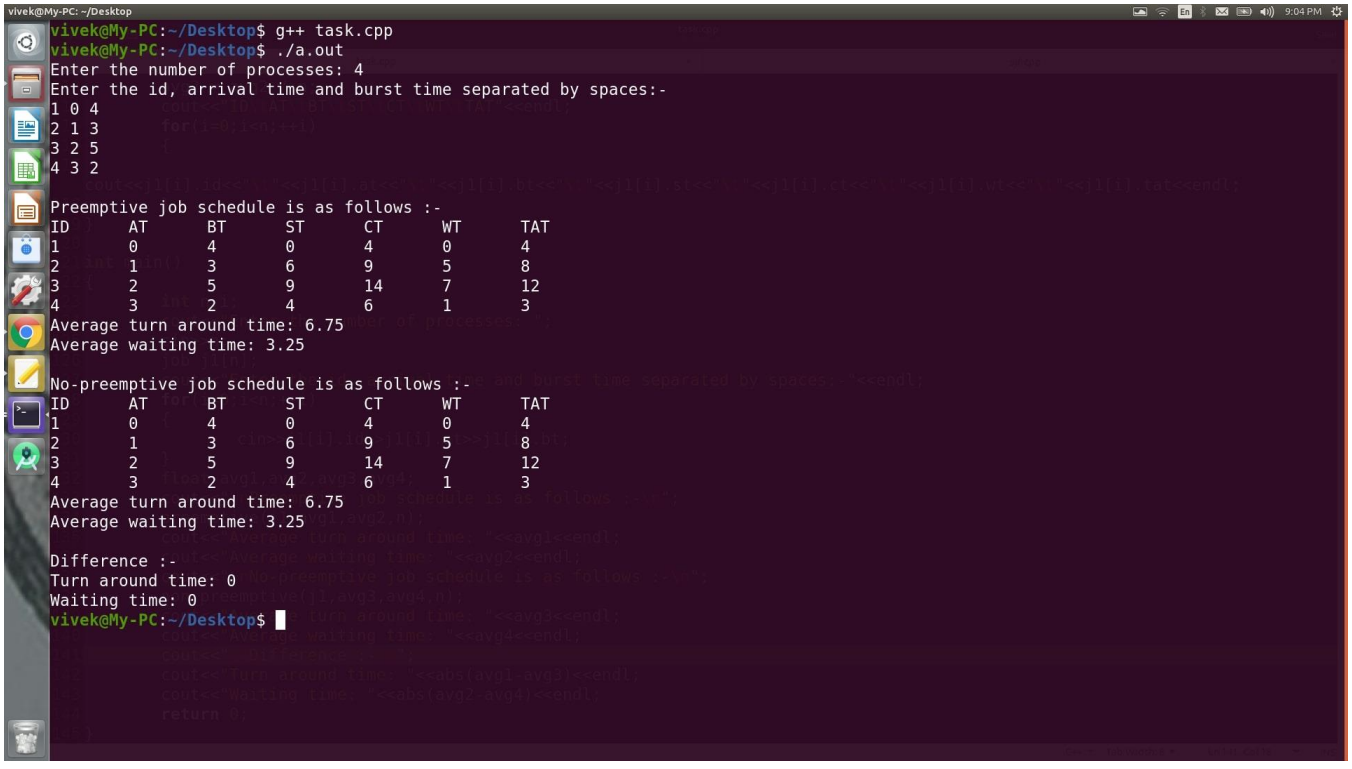
```
#include<bits/stdc++.h>

using namespace std;

struct job
{
	int id, at, bt, st, ct, tat, wt;
};

void preemptive(job j1[], float &avg1, float &avg2, int n)
{
	avg1 = 0;
	avg2 = 0;
	int rem[n],i;
	bool start[n] = {};
	for(i=0;i<n;++i)
	{
		rem[i] = j1[i].bt;
	}
	int completed = 0,time = 0;
	while(completed<n)
	{
		int min_time = INT_MAX,min_index = -1;
```

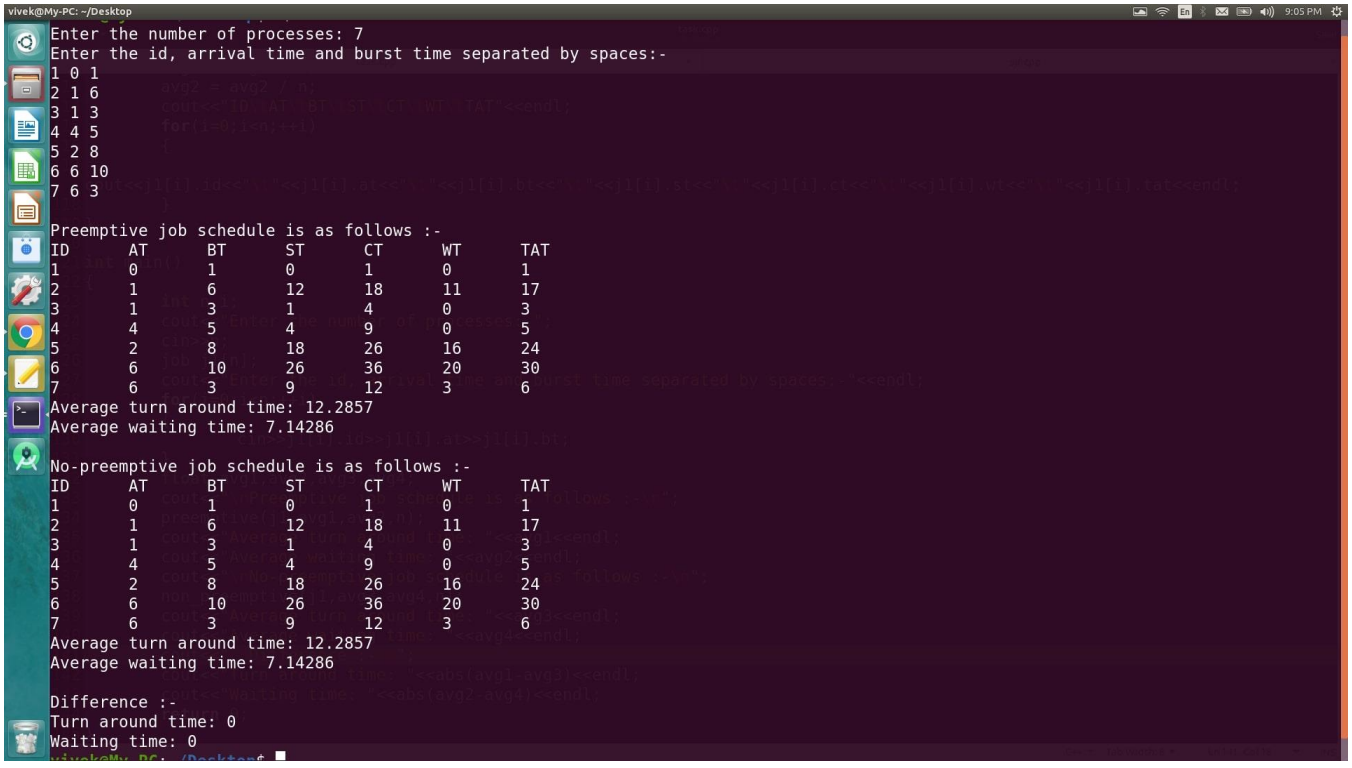
```
                for(i=0;i<n;++i)
                {
                        if(j1[i].at<=time and rem[i]>0 and rem[i]<min_time)
                        {
                                min_time = rem[i];
                                min_index = i;
                        }
                }
                if(min_index==-1)
                {
                        time = time + 1;
                        continue;
                }
                if(start[min_index]==false)
                {
                        j1[min_index].st = time;
                        start[min_index] = true;
                }
                rem[min_index] -= 1;
                min_time = rem[min_index];
                if(min_time == 0)
                {
                        completed += 1;
                        min_time = INT_MAX;

                        j1[min_index].ct = time + 1;
                        j1[min_index].tat = j1[min_index].ct - j1[min_index].at;
                        j1[min_index].wt = j1[min_index].tat - j1[min_index].bt;

                        avg1 += j1[min_index].tat;
                        avg2 += j1[min_index].wt;
                }
                time = time + 1;
        }
        avg1 = avg1 / n;
        avg2 = avg2 / n;
        cout<<"ID\tAT\tBT\tST\tCT\tWT\tTAT"<<endl;
        for(i=0;i<n;++i)
        {

        cout<<j1[i].id<<"\t"<<j1[i].at<<"\t"<<j1[i].bt<<"\t"<<j1[i].st<<"\t"<<j1[i].ct<<"\t"<<j1[i
].wt<<"\t"<<j1[i].tat<<endl;
        }
}

void non_preemptive(job j1[], float &avg1, float &avg2, int n)
{
```

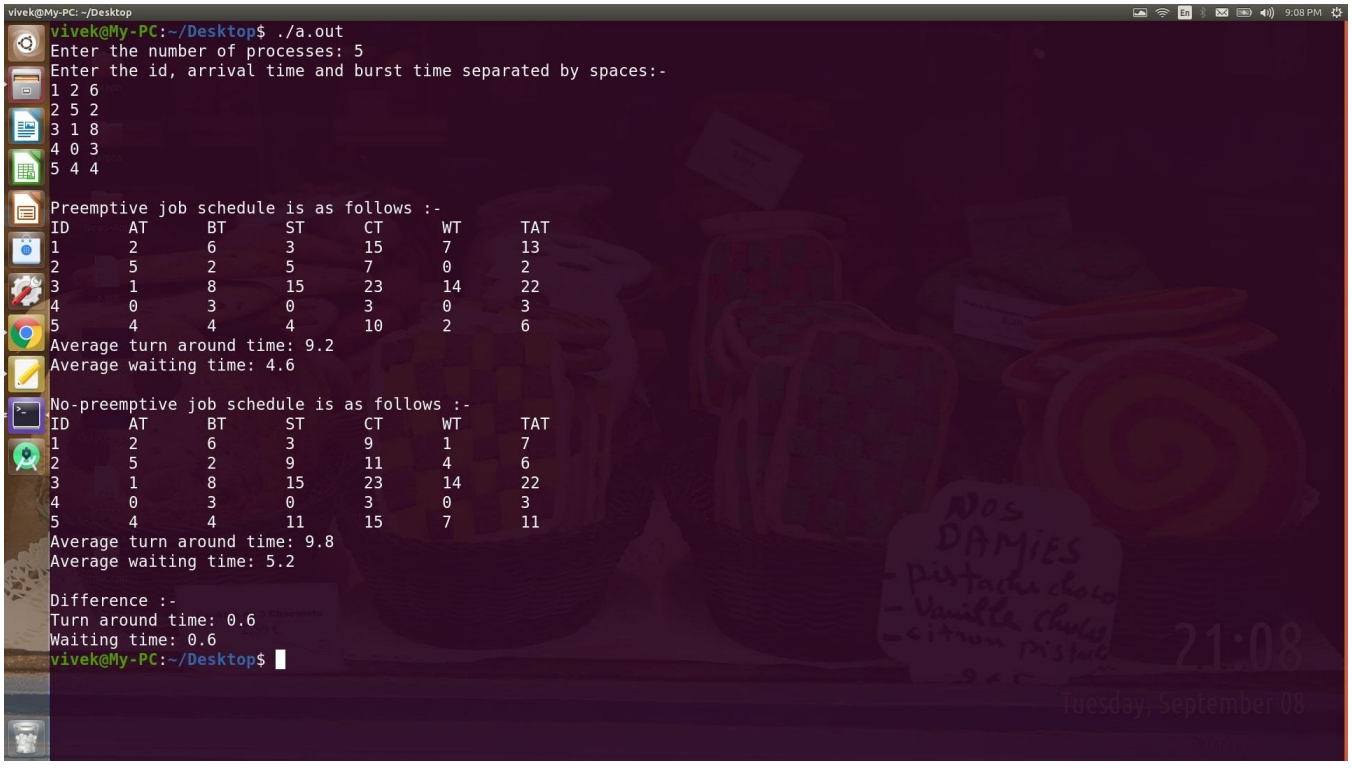
```
avg1 = 0;
avg2 = 0;
int rem[n],i;
for(i=0;i<n;++i)
{
	rem[i] = j1[i].bt;
}
int completed = 0,time = 0;
while(completed<n)
{
	int min_time = INT_MAX,min_index = -1;
	for(i=0;i<n;++i)
	{
		if(j1[i].at<=time and rem[i]>0 and rem[i]<min_time)
		{
			min_time = rem[i];
			min_index = i;
		}
	}
	if(min_index==-1)
	{
		time = time + 1;
		continue;
	}
	j1[min_index].st = time;
	j1[min_index].ct = time + j1[min_index].bt;
	rem[min_index] = 0;

	time = time + j1[min_index].bt;

	min_time = rem[min_index];

	completed += 1;
	min_time = INT_MAX;

	j1[min_index].tat = j1[min_index].ct - j1[min_index].at;
	j1[min_index].wt = j1[min_index].tat - j1[min_index].bt;

	avg1 += j1[min_index].tat;
	avg2 += j1[min_index].wt;

	//time = time + 1;
}
avg1 = avg1 / n;
avg2 = avg2 / n;
cout<<"ID\tAT\tBT\tST\tCT\tWT\tTAT"<<endl;
for(i=0;i<n;++i)
```

```
        {

        cout<<j1[i].id<<"\t"<<j1[i].at<<"\t"<<j1[i].bt<<"\t"<<j1[i].st<<"\t"<<j1[i].ct<<"\t"<<j1[i
].wt<<"\t"<<j1[i].tat<<endl;
        }
}

int main()
{
        int n,i;
        cout<<"Enter the number of processes: ";
        cin>>n;
        job j1[n];
        cout<<"Enter the id, arrival time and burst time separated by spaces:-"<<endl;
        for(i=0;i<n;++i)
        {
                cin>>j1[i].id>>j1[i].at>>j1[i].bt;
        }
        float avg1,avg2,avg3,avg4;
        cout<<"\nPreemptive job schedule is as follows :-\n";
        preemptive(j1,avg1,avg2,n);
        cout<<"Average turn around time: "<<avg1<<endl;
        cout<<"Average waiting time: "<<avg2<<endl;
        cout<<"\nNo-preemptive job schedule is as follows :-\n";
        non_preemptive(j1,avg3,avg4,n);
        cout<<"Average turn around time: "<<avg3<<endl;
        cout<<"Average waiting time: "<<avg4<<endl;
        cout<<"\nDifference :-\n";
        cout<<"Turn around time: "<<abs(avg1-avg3)<<endl;
        cout<<"Waiting time: "<<abs(avg2-avg4)<<endl;
        return 0;
}
```

Output


vivek@My-PC: ~/Desktop
9:04 PM
vivek@My-PC:~/Desktop$ g++ task.cpp
vivek@My-PC:~/Desktop$ ./a.out
Enter the number of processes: 4
Enter the id, arrival time and burst time separated by spaces:-
1 0 4
2 1 3
3 2 5
4 3 2
Preemptive job schedule is as follows :-
ID AT BT ST CT WT TAT
1 0 4 0 4 0 4
2 1 3 6 9 5 8
3 2 5 9 14 7 12
4 3 2 4 6 1 3
Average turn around time: 6.75
Average waiting time: 3.25
No-preemptive job schedule is as follows :-
ID AT BT ST CT WT TAT
1 0 4 0 4 0 4
2 1 3 6 9 5 8
3 2 5 9 14 7 12
4 3 2 4 6 1 3
Average turn around time: 6.75
Average waiting time: 3.25
Difference :-
Turn around time: 0
Waiting time: 0
vivek@My-PC:~/Desktop$





Result: Preemptive and non-preemptive shortest job first CPU scheduling algorithms were implemented successfully and verified.

Experiment Number 7: Implementation of Multilevel Queue Scheduling.

Date: September 22, 2020

Aim: To write the program to implement multilevel queue scheduling taking two queues containing System and User processes respectively where System queue has priority over User queue and each queue is scheduled individually using FCFS.

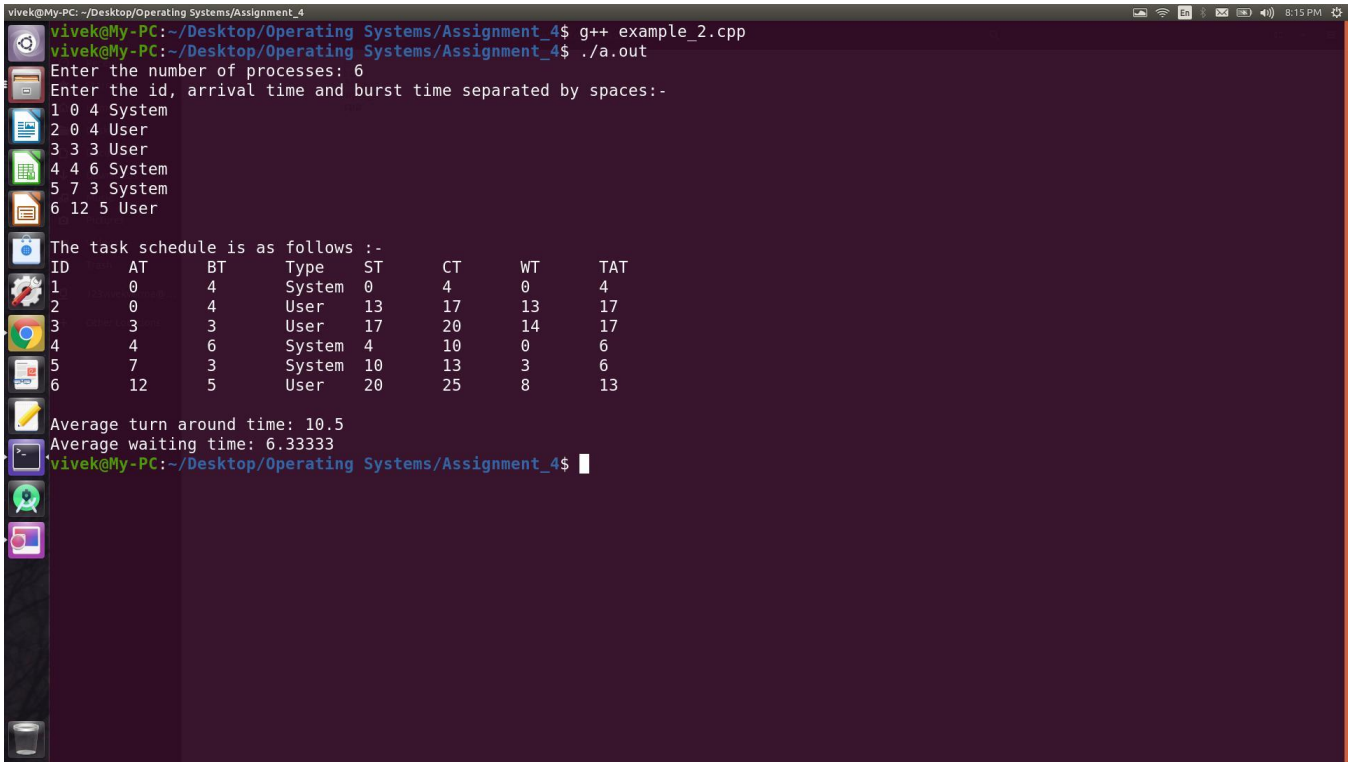
C++ Implementation

```
#include<bits/stdc++.h>

using namespace std;

struct process
{
	int at,st,ct,tat,bt,wt,id;
	string type;
};

void find(process p1[], int n)
{
	int time = 0, completed = 0;
	int time1[n],i;
	float avg1 = 0, avg2 = 0;
	bool visited[n] = {};
	for(i=0;i<n;++i)
	{
		time1[i] = p1[i].bt;
	}
	while(completed<n)
	{
		int idx = -1,at = INT_MAX;
		for(i=0;i<n;++i)
		{
			if(p1[i].at <= time and time1[i]>0 and p1[i].type=="System" and p1[i].at <
at)
			{
				idx = i;
				at = p1[i].at;
			}
		}
		if(idx==-1)
		{
			at = INT_MAX;
			for(i=0;i<n;++i)
			{
```

```
                        if(p1[i].at <= time and time1[i]>0 and p1[i].type=="User" and
p1[i].at < at)
                        {
                                idx = i;
                                at = p1[i].at;
                        }
                }
        }
        if(idx!=-1)
        {
                if(visited[idx]==false)
                {
                        p1[idx].st = time;
                        visited[idx] = true;
                }

                time = time + 1;
                time1[idx] -= 1;

                if(time1[idx]==0)
                {
                        p1[idx].ct = time;

                        completed = completed + 1;

                        p1[idx].tat = p1[idx].ct - p1[idx].at;
                        p1[idx].wt = p1[idx].tat - p1[idx].bt;

                        avg1 += p1[idx].tat;
                        avg2 += p1[idx].wt;
                }
        }
        else
        {
                time = time + 1;
        }
    }
    cout<<"\nThe task schedule is as follows :-\n";
    cout<<"ID\tAT\tBT\tType\tST\tCT\tWT\tTAT"<<endl;
    for(i=0;i<n;++i)
    {

    cout<<p1[i].id<<"\t"<<p1[i].at<<"\t"<<p1[i].bt<<"\t"<<p1[i].type<<"\t"<<p1[i].st<<"\t"
<<p1[i].ct<<"\t"<<p1[i].wt<<"\t"<<p1[i].tat<<endl;
    }
    avg1 = avg1;
    avg2 = avg2;
```

```
        cout<<"\nAverage turn around time: "<<avg1 / n<<endl;
        cout<<"Average waiting time: "<<avg2 / n<<endl;
}

int main()
{
        int n,i;
        cout<<"Enter the number of processes: ";
        cin>>n;
        process p1[n];
        cout<<"Enter the id, arrival time and burst time separated by spaces:-"<<endl;
        for(i=0;i<n;++i)
        {
                cin>>p1[i].id>>p1[i].at>>p1[i].bt>>p1[i].type;
        }
        find(p1,n);
        return 0;
}
```

Output



Result: The given multilevel queue scheduling was implemented successfully and verified.

Experiment Number 8: Implementation of Multilevel Queue Scheduling.

Date: September 22, 2020

Aim: To write the program to implement multilevel queue scheduling taking two queues containing System and User processes respectively where System queue has priority over User queue and System queue has round robin scheduling whereas User queue has FCFS.

C++ Implementation

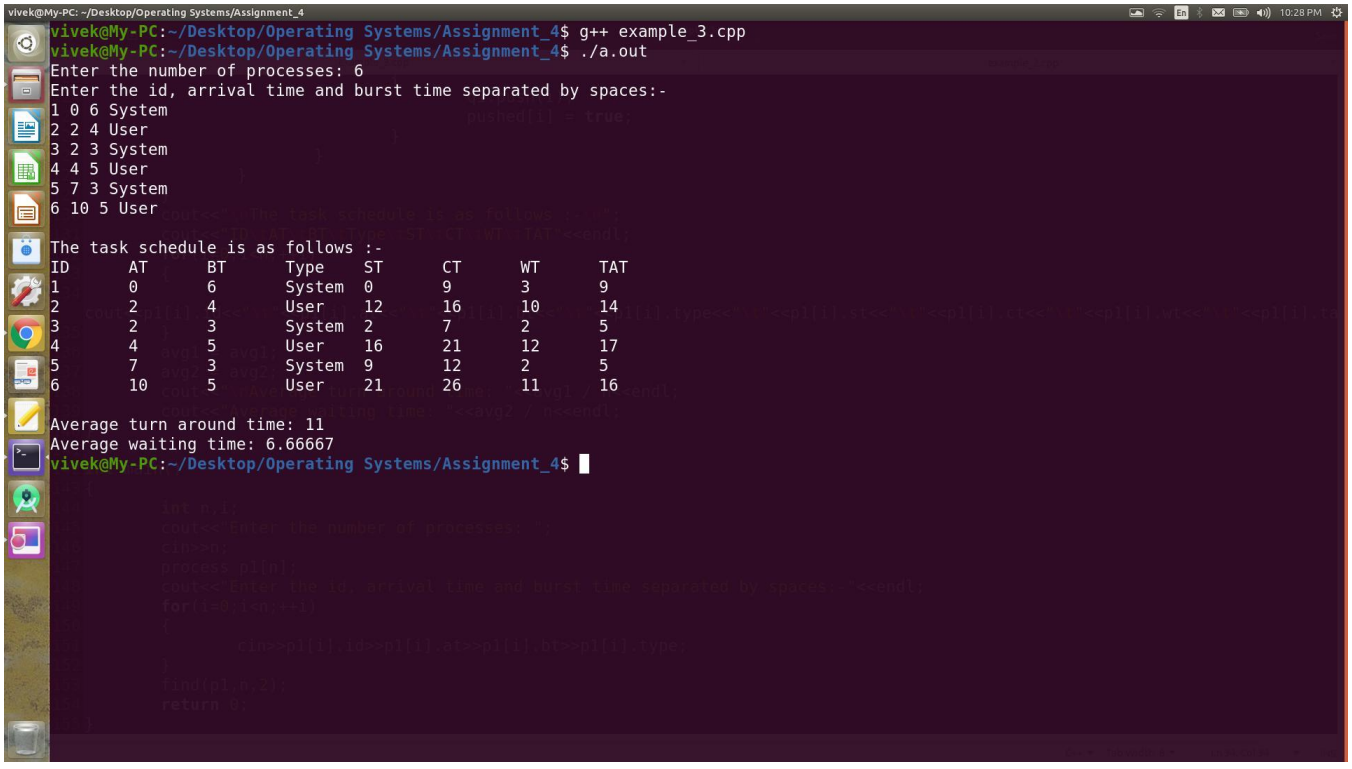
```
#include<bits/stdc++.h>

using namespace std;

struct process
{
	int id,at,bt,st,ct,tat,wt;
	string type;
};

void find(process p1[], int n, int q)
{
	int time = 0,completed = 0,time1[n] = {},i;
	bool pushed[n] = {},visited[n] = {};
	for(i=0;i<n;++i)
	{
		time1[i] = p1[i].bt;
	}
	queue <int> q1;
	float avg1 = 0, avg2 = 0;
	for(i=0;i<n;++i)
	{
		if(pushed[i]==false and p1[i].at<=time and p1[i].type=="System")
		{
			q1.push(i);
			pushed[i] = true;
		}
	}
	while(completed<n)
	{
		if(q1.empty()==false)
		{
			int idx = q1.front();
			q1.pop();
			if(visited[idx]==false)
			{
				p1[idx].st = time;
```

```
                visited[idx] = true;
            }
            if(time1[idx]<=q)
            {
                time = time + time1[idx];
                p1[idx].ct = time;
                time1[idx] = 0;

                completed = completed + 1;

                p1[idx].tat = p1[idx].ct - p1[idx].at;
                p1[idx].wt = p1[idx].tat - p1[idx].bt;

                avg1 += p1[idx].tat;
                avg2 += p1[idx].wt;

                for(i=0;i<n;++i)
                {
                    if(pushed[i]==false and p1[i].at<=time and
p1[i].type=="System")
                    {
                        q1.push(i);
                        pushed[i] = true;
                    }
                }

            }
            else
            {
                time1[idx] = time1[idx] - q;
                time = time + q;
                for(i=0;i<n;++i)
                {
                    if(pushed[i]==false and p1[i].at<=time and
p1[i].type=="System")
                    {
                        q1.push(i);
                        pushed[i] = true;
                    }
                }
                q1.push(idx);
            }
        }
        else
        {
            int idx = -1, at = INT_MAX;
            for(i=0;i<n;++i)
```

```
                {
                        if(time1[i]>0 and p1[i].type=="User" and p1[i].at < at)
                        {
                                idx = i;
                                at = p1[i].at;
                        }
                }
                if(idx==-1)
                {
                        time = time + 1;
                        continue;
                }
                else
                {
                        if(visited[idx]==false)
                        {
                                p1[idx].st = time;
                                visited[idx] = true;
                        }

                        time = time + 1;
                        time1[idx] -= 1;

                        if(time1[idx]==0)
                        {
                                p1[idx].ct = time;

                                completed = completed + 1;

                                p1[idx].tat = p1[idx].ct - p1[idx].at;
                                p1[idx].wt = p1[idx].tat - p1[idx].bt;

                                avg1 += p1[idx].tat;
                                avg2 += p1[idx].wt;
                        }
                }

                for(i=0;i<n;++i)
                {
                        if(pushed[i]==false and p1[i].at<=time and p1[i].type=="System")
                        {
                                q1.push(i);
                                pushed[i] = true;
                        }
                }
        }
}
```

```
        cout<<"\nThe task schedule is as follows :-\n";
        cout<<"ID\tAT\tBT\tType\tST\tCT\tWT\tTAT"<<endl;
        for(i=0;i<n;++i)
        {

        cout<<p1[i].id<<"\t"<<p1[i].at<<"\t"<<p1[i].bt<<"\t"<<p1[i].type<<"\t"<<p1[i].st<<"\t"
<<p1[i].ct<<"\t"<<p1[i].wt<<"\t"<<p1[i].tat<<endl;
        }
        avg1 = avg1;
        avg2 = avg2;
        cout<<"\nAverage turn around time: "<<avg1 / n<<endl;
        cout<<"Average waiting time: "<<avg2 / n<<endl;
}

int main()
{
        int n,i;
        cout<<"Enter the number of processes: ";
        cin>>n;
        process p1[n];
        cout<<"Enter the id, arrival time and burst time separated by spaces:-"<<endl;
        for(i=0;i<n;++i)
        {
                cin>>p1[i].id>>p1[i].at>>p1[i].bt>>p1[i].type;
        }
        find(p1,n,2);
        return 0;
}
```

Output



Result: The given multilevel queue scheduling was implemented successfully and verified.

Experiment Number 9: Implementation of Proudcer & Consumer Problem

Date: September 29, 2020

Aim: To implement Producer & Consumer problem (bounded buffer problem) using semaphores.

Algorithm for Producer & Consumer problem.

1	Declare variable for producer & consumer as pthread tid produce and tid consume.
2	Declare a structure to add items, semaphore variable set as struct.
3	Read number the items to be produced and consumed.
4	Declare and define semaphore function for creation and destroy.
5	Define producerfunction.
6	Define consumer function.
7	Call producer andconsumer.
8	Stop theexecution.

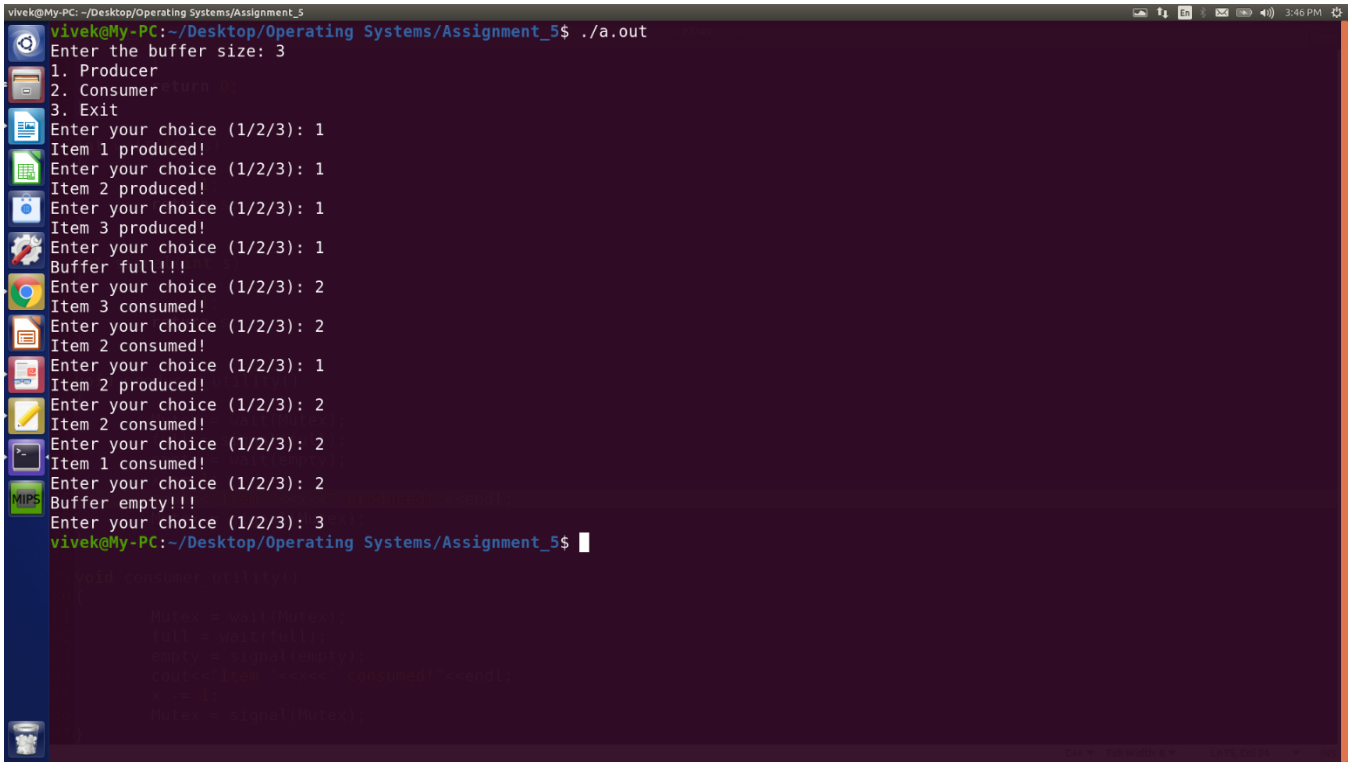
C++ Implementation

```
#include<bits/stdc++.h>

using namespace std;

int Mutex=1,full=0,empty=3,x=0;

int main()
{
        int choice;
        void producer_utility();
        void consumer_utility();
        int wait(int),signal(int);
        cout<<"Enter the buffer size: ";
        cin>>empty;
        cout<<"1. Producer"<<endl;
        cout<<"2. Consumer"<<endl;
        cout<<"3. Exit"<<endl;
        while(true)
        {
                cout<<"Enter your choice (1/2/3): ";
                cin>>choice;
                switch(choice)
                {
                        case 1:
                                if((Mutex==1) and (empty!=0))
                                {
```

```
                producer_utility();
            }
            else
            {
                cout<<"Buffer full!!!"<<endl;
            }
            break;

        case 2:
            if((Mutex==1) and (full!=0))
            {
                consumer_utility();
            }
            else
            {
                cout<<"Buffer empty!!!"<<endl;
            }
            break;

        case 3:
            exit(0);
            break;

        default:
            cout<<"Invalid Choice"<<endl;
    }
  }
  return 0;
}

int wait(int s)
{
  s -= 1;
  return s;
}

int signal(int s)
{
  s += 1;
  return s;
}

void producer_utility()
{
  Mutex = wait(Mutex);
  full = signal(full);
  empty = wait(empty);
```

```
        x += 1;
        cout<<"Item "<<x<<" produced!"<<endl;
        Mutex = signal(Mutex);
}

void consumer_utility()
{
        Mutex = wait(Mutex);
        full = wait(full);
        empty = signal(empty);
        cout<<"Item "<<x<<" consumed!"<<endl;
        x -= 1;
        Mutex = signal(Mutex);
}
```

Output



Result: The Producer & Consumer problem was implemented and verified successfull

Experiment Number 10: Implementation of Dining Philosopher's problem using mutex and semaphores.

Date: October 6, 2020

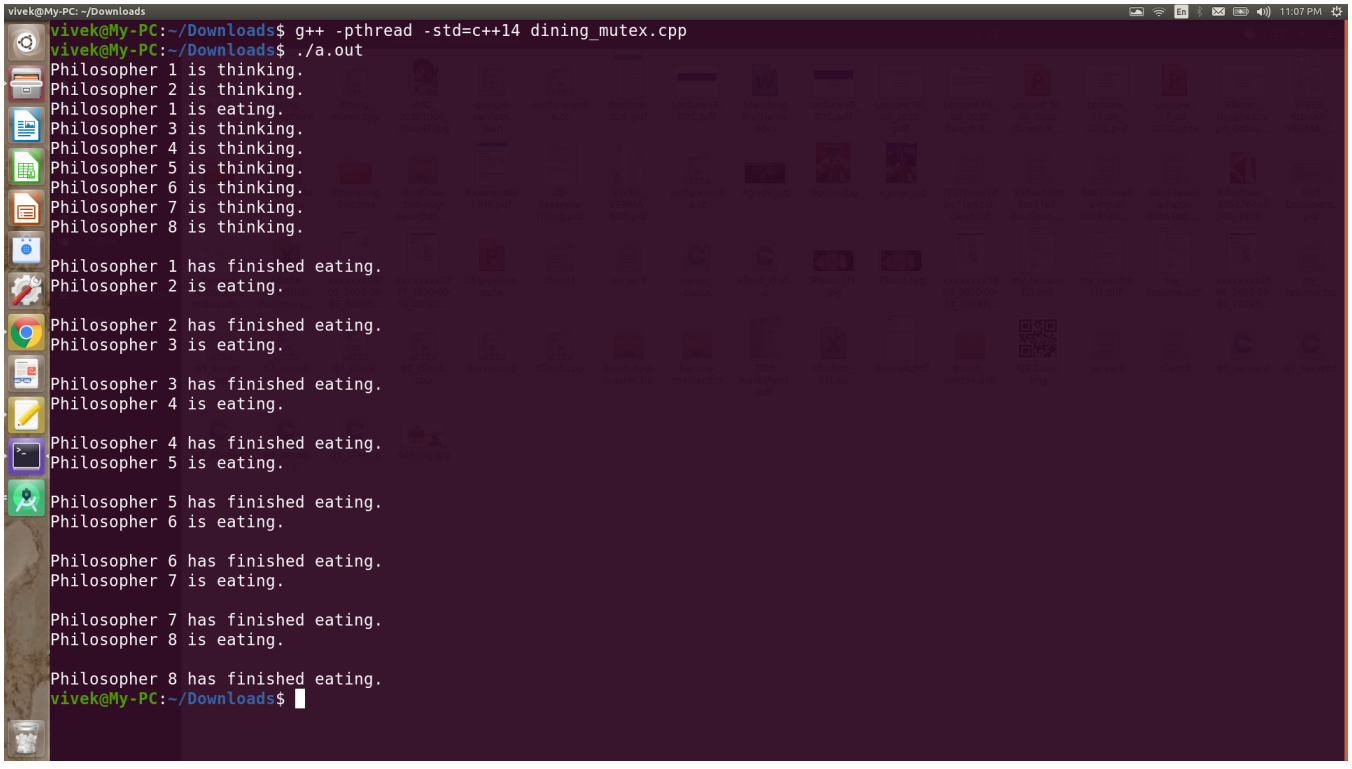
Aim: To implement Dining Philosopher's problem using mutex and semaphores.

C++ implementation using mutex.

```
#include <bits/stdc++.h>
#include <mutex>
#include <thread>

using namespace std;
const int N = 8;

class fork
{
   public:
                fork()
                {
                        ;
                }
                mutex mu;
};

void eat(fork &fork_left, fork& fork_right, int number)
{

      fork_left.mu.lock();
      fork_right.mu.lock();
      cout << "Philosopher " << number << " is eating." << endl;
      this_thread::sleep_for(chrono::seconds(2));
      cout << "\nPhilosopher " << number << " has finished eating." << endl;
      fork_right.mu.unlock();
      fork_left.mu.unlock();
}

int main()
{
   fork forks[N];
   thread philosopher[N];
   cout << "Philosopher " << 1 << " is thinking." << endl;
   philosopher[0] = thread(eat, ref(forks[0]), ref(forks[N-1]), 1);
   for(int i = 1; i < N; ++i)
   {
```

```
        cout << "Philosopher " << (i+1) << " is thinking." << endl;
        philosopher[i] = thread(eat, ref(forks[i]), ref(forks[i-1]), (i+1));
    }

    for(auto &ph: philosopher) ph.join();
    return 0;
}
```

Output

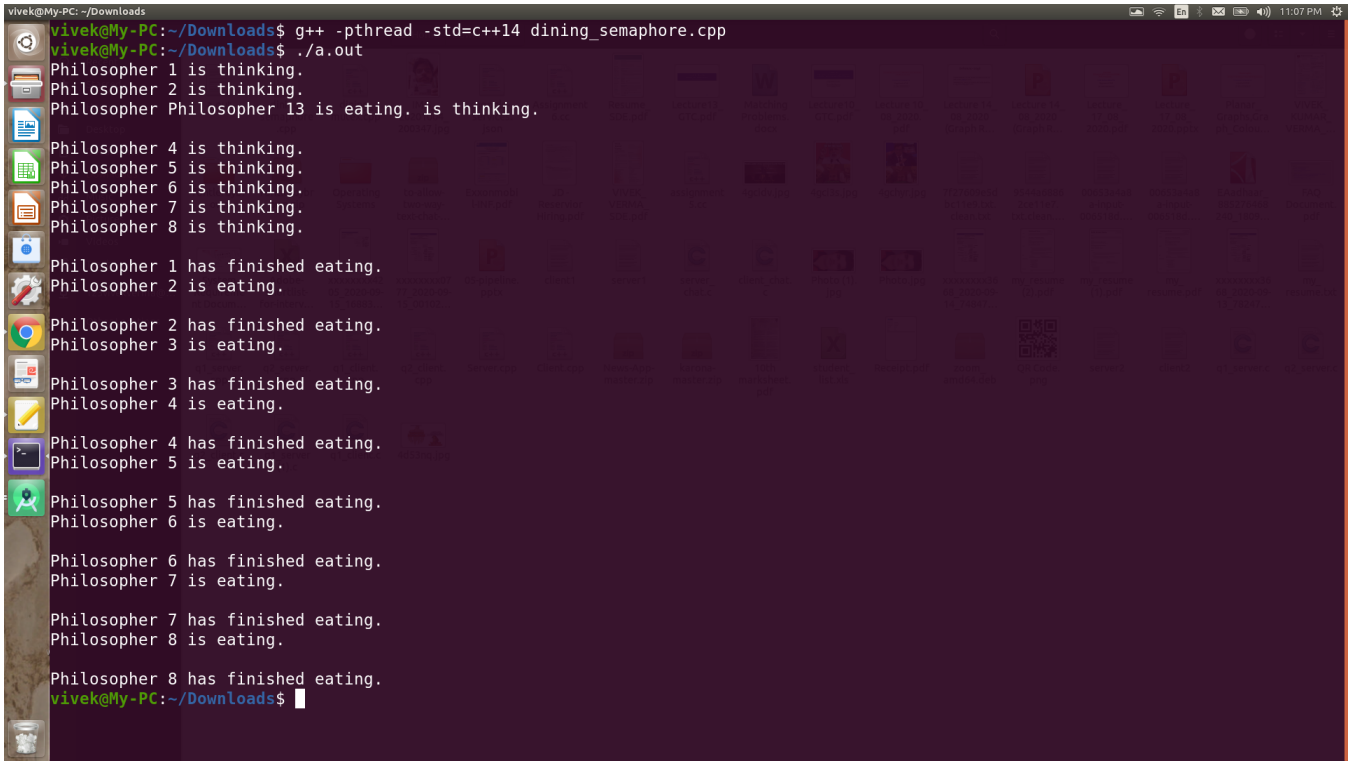


C++ Implementation (using Semaphores)

```
#include <bits/stdc++.h>
#include <mutex>
#include <thread>

using namespace std;
const int N = 8;

class Semaphore
{
        private:
                bool signaled;
                pthread_mutex_t m;
                pthread_cond_t  c;
```

```
                void Lock()
                {
                        pthread_mutex_lock(&m);
                }

                void Unlock()
                {
                        pthread_mutex_unlock(&m);
                }

        public:
                Semaphore();
                void P();
                void V();
};

Semaphore::Semaphore()
{
        signaled = true;
        c = PTHREAD_COND_INITIALIZER;
        m = PTHREAD_MUTEX_INITIALIZER;
}

void Semaphore::P()
{
        Lock();
        while (!signaled)
        {
                pthread_cond_wait(&c, &m);
        }
        signaled = false;
        Unlock();
}

void Semaphore::V()
{
        bool previously_signaled;
        Lock();
        previously_signaled = signaled;
        signaled = true;
        Unlock();
        if (!previously_signaled)
        pthread_cond_signal(&c);
}

void eat(Semaphore &fork_left, Semaphore &fork_right, int number)
```

```
{
	fork_left.P();
	fork_right.P();
	cout << "Philosopher " << number << " is eating." << endl;
	this_thread::sleep_for(chrono::seconds(2));
	cout << "\nPhilosopher " << number << " has finished eating." << endl;
	fork_right.V();
	fork_left.V();
}

int main()
{
	Semaphore forks[N];

	thread philosopher[N];

	cout << "Philosopher " << 1 << " is thinking." << endl;
	philosopher[0] = thread(eat, ref(forks[0]), ref(forks[N-1]), 1);

	for(int i = 1; i < N; ++i)
	{
		cout << "Philosopher " << (i+1) << " is thinking." << endl;
		philosopher[i] = thread(eat, ref(forks[i]), ref(forks[i-1]), (i+1));
	}

	for(auto &ph: philosopher) ph.join();
	return 0;
}
```



Result: The Dining Philosopher's problem was implemented using mutex & semaphores and verified successfully.

Experiment Number 11: Implementation of Banker's and Deadlock Prevention Algorithms

Date: October 13, 2020

Aim: To implement Banker's and Deadlock prevention algorithm.

1. **Deadlock Prevention**
    a. We can prevent Deadlock by eliminating any of the below four conditions.

2. **Eliminate Mutual Exclusion**
    a. It is not possible to dis-satisfy the mutual exclusion because some resources, such as the tape drive and printer, are inherently non-shareable.

3. **Eliminate Hold and wait**
    a. Allocate all required resources to the process before the start of its execution, this wayhold and wait condition is eliminated but it will lead to low device utilization. For example, if a process requires printer at a later time and we have allocated printer before the start of its execution printer will remain blocked till it has completed itsexecution.
    b. The process will make a new request for resources after releasing the current set of resources. This solution may lead tostarvation.

4. **Eliminate No Preemption**
    a. Preempt resources from the process when resources required by other high priority processes.

5. **Eliminate Circular Wait**
    a. Each resource will be assigned with a numerical number. A process can request the resources increasing/decreasing order of numbering.
    b. For Example, if P1 process is allocated R5 resources, now next time if P1 ask for R4, R3 lesser than R5 such request will not be granted, only request for resources more than R5 will be granted.

Algorithm

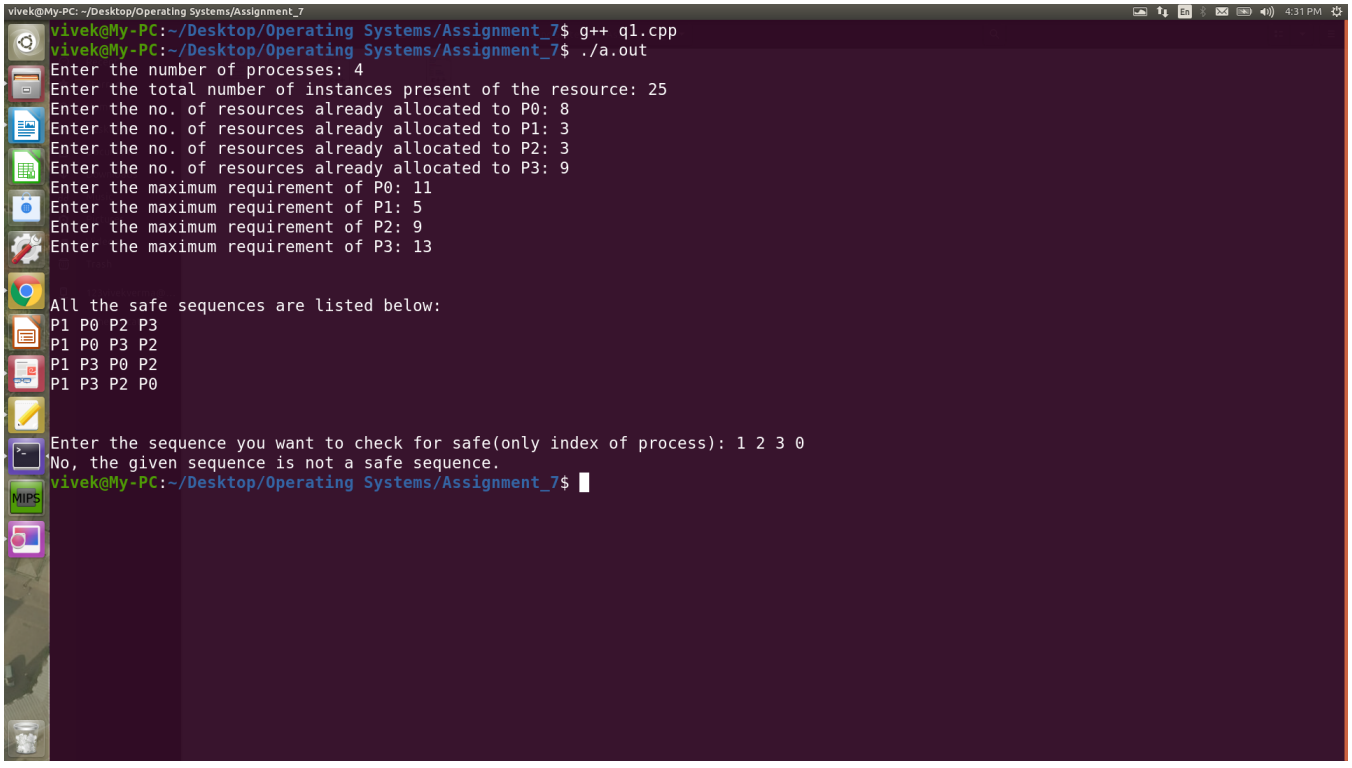
1.	Start the program.
2.	Attacking mutex condition: never grant exclusive access. But this may not be possible for several resources.
3.	Attacking preemption: not something you want to do.
4.	Attacking hold and wait condition: make a process hold at the most 1 resource.
5.	At a time. Make all the requests at the beginning. Nothing policy. If you feel, retry.
6.	Attacking circular wait: Order all the resources. Make sure that the requests are issued in the.
7.	Correct order so that there are no cycles present in the resource graph. Resources numbered 1 ... n.
8.	Resources can be requested only in increasing.
9.	Order. i.e. you cannot request a resource whose no is less than any you may be holding.

C++ Implementation (Baker's algorithm)

```
#include<bits/stdc++.h>

using namespace std;

int main()
{
	int n;
	cout << "Enter the number of processes: ";
	cin >> n;
	int r;
	cout << "Enter the total number of instances present of the resource: ";
	cin >> r;
	vector < vector<int>> safe_sequences;
	vector<int> cur_sequence(n);
	vector<int> allocated(n), maxi(n);
	for(int i = 0; i < n; i++)
	{
		cur_sequence[i] = i;
		cout<<"Enter the no. of resources already allocated to P"<<i<<": ";
		cin>>allocated[i];
	}

	for(int i = 0; i < n; i++)
	{
		cout<<"Enter the maximum requirement of P"<<i<<": ";
		cin>>maxi[i];
	}
	while(true)
	{
		int cont = 0; //indicates that current sequence will lead to a deadlock
		vector <int> need(n);
		int available = r;
		for(int i = 0; i < n; i++)
		{
			available -= allocated[i];
			need[i] = maxi[i] - allocated[i];
		}
		for(int i = 0; i < n; i++)
		{
			if(need[cur_sequence[i]] > available)
			{
				cont = 1;
				break;
			}
			available += allocated[cur_sequence[i]];
```

```
            }
            if(cont==0)
            {
                  safe_sequences.push_back(cur_sequence);
            }
            int flag = next_permutation(cur_sequence.begin(), cur_sequence.end());
            if(!flag)
            {
                  break;
            }
      }
      cout << "\n\nAll the safe sequences are listed below:\n";
      for(int i = 0; i <safe_sequences.size(); i++)
      {
            for(int j = 0; j < n; j++)
            {
                  cout <<"P"<<safe_sequences[i][j]<<" ";
            }
            cout<<"\n";
      }
      cout << "\n\nEnter the sequence you want to check for safe(only index of process): ";
      vector<int> check(n);
      for(int i = 0; i < n; i++)
      {
            cin >> check[i];
      }
      int yes = 0;
      for(int i = 0; i < safe_sequences.size(); i++)
      {
            int flag = 0;
            for(int j = 0; j < n; j++)
            {
                  if(safe_sequences[i][j] != check[j])
                  {
                        flag = 1;
                        break;
                  }
            }
            if(flag == 0)
            {
                  yes = 1;
                  break;
            }
      }
      if(yes)
      {
            cout <<"Yes, the given sequence is a safe sequence.\n";
```

```
        }
        else
        {
                cout<<"No, the given sequence is not a safe sequence.\n";
        }
        return 0;
}
```



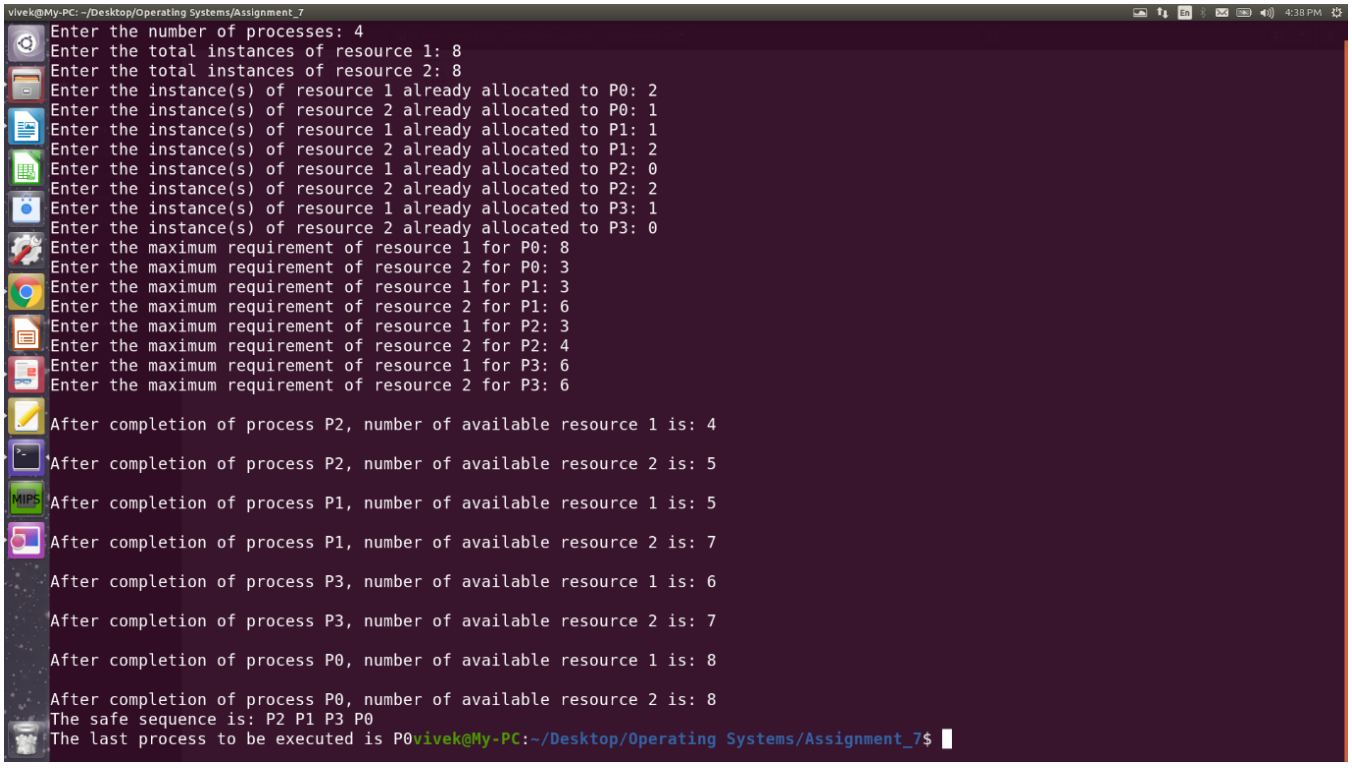
C++ Implementation of deadlock prevention algorithm

```
#include<bits/stdc++.h>

using namespace std;

int main()
{
	int n;
	cout<<"Enter the number of processes: ";
	cin>>n;
	int r1, r2;
	cout<<"Enter the total instances of resource 1: ";
	cin>>r1;
	cout<<"Enter the total instances of resource 2: ";
	cin>>r2;
	int allocated[n][2], maxi[n][2];
	for(int i = 0; i < n; i++)
	{
		cout<<"Enter the instance(s) of resource 1 already allocated to P" << i <<": ";
		cin>>allocated[i][0];
		cout<<"Enter the instance(s) of resource 2 already allocated to P" << i <<": ";
		cin>>allocated[i][1];
	}
```

```
	for(int i = 0; i < n; i++)
	{
		cout<<"Enter the maximum requirement of resource 1 for P"<<i<<": ";
		cin>>maxi[i][0];
		cout<<"Enter the maximum requirement of resource 2 for P"<<i<<": ";
		cin>>maxi[i][1];
	}
	int available1 = r1, available2 = r2;
	for(int i = 0; i < n; i++)
	{
		available1 -= allocated[i][0];
		available2 -= allocated[i][1];
	}
	int need[n][2];
	for(int i = 0; i < n; i++)
	{
		need[i][0] = maxi[i][0] - allocated[i][0];
		need[i][1] = maxi[i][1] - allocated[i][1];
	}
	vector<int> safe_sequence;
	for(int i = 0; i < n; i++)
	{
		int dead_lock = 1;
		for(int j = 0; j < n; j++)
		{
			if(available1 >= need[j][0] && available2 >= need[j][1] && (need[j][0] ||
need[j][1]))
			{
				dead_lock = 0;
				available1 += allocated[j][0];
				need[j][0] = 0;
				available2 += allocated[j][1];
				need[j][1] = 0;
				cout << "\nAfter completion of process P" << j <<", number of
available resource 1 is: "<<available1<<"\n";
				cout << "\nAfter completion of process P" << j <<", number of
available resource 2 is: "<<available2<<"\n";
				safe_sequence.push_back(j);
			}
		}
		if(dead_lock)
		{
			break;
		}
	}
	if(safe_sequence.size() < n)
	{
```

```
            cout << "There is no safe sequence\n";
        }
        else
        {
            cout<<"The safe sequence is: ";
            for(int i = 0; i < n; i++)
            {
                cout << "P" <<safe_sequence[i]<<" ";
            }
            cout<<"\n";
            cout<<"The last process to be executed is "<<"P"<<safe_sequence[n-1];
        }
        return 0;
    }
```

Output



Result: Banker's and Deadlock Prevention Algorithm was implemented and verified successfully.

Experiment Number 12: Stimulate Page Replacement Algorithm: FIFO

Date: October 20, 2020

Aim: To Simulate FIFO page replacement algorithm.

First in First Out (FIFO): This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

Example1: Consider page reference string 1, 3, 0, 3, 5, 6 with 3-page frames. Find number of page faults.

Initially all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots —>**3 Page Faults.**

When 3 comes, it is already in memory so —>**0 Page Faults.**

Then 5 comes, it is not available in memory so it replaces the oldest page slot i.e. 1. —>**1 Page Fault.**

6 comes, it is also not available in memory so it replaces the oldest page slot i.e. 3 —>**1 Page Fault.**

Finally, when 3 come it is not available so it replaces 0 —>**1 Page fault**

Belady's Anomaly: Belady's anomaly proves that it is possible to have more page faults when increasing the number of page frames while using the First in First out (FIFO) page replacement algorithm.

For example, if we consider reference string 3, 2, 1, 0, 3, 2, 4, 3, 2,1, 0, 4 and 3 slots, we get 9 total page faults, but if we increase slots to 4, we get 10-page faults.

Algorithm

1. Start the program
2. Read the number offrames
3. Read the number ofpages
4. Read the pagenumbers
5. Initialize the values in frames to-1
6. Allocate the pages in to frames in First in first outorder.
7. Display the number of pagefaults.
8. Stop theprogram

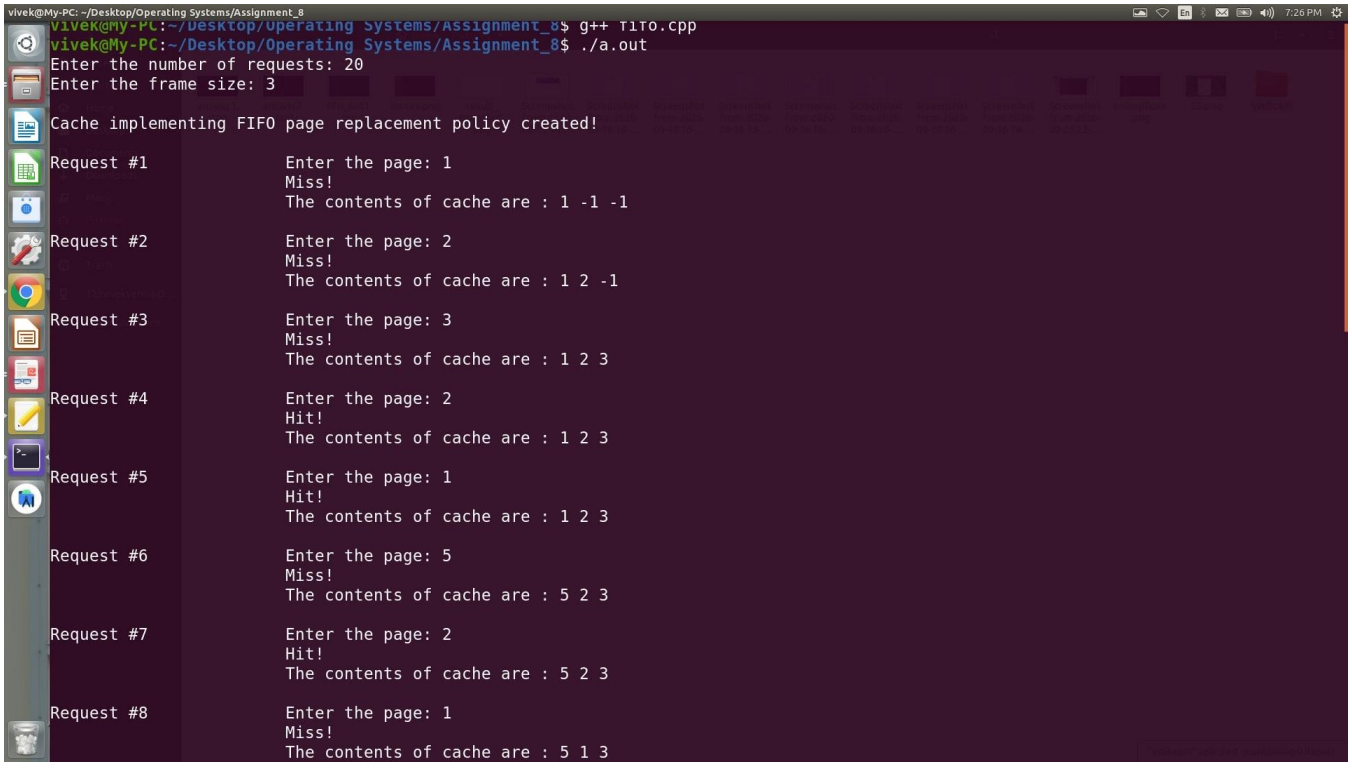
C++ Implementation

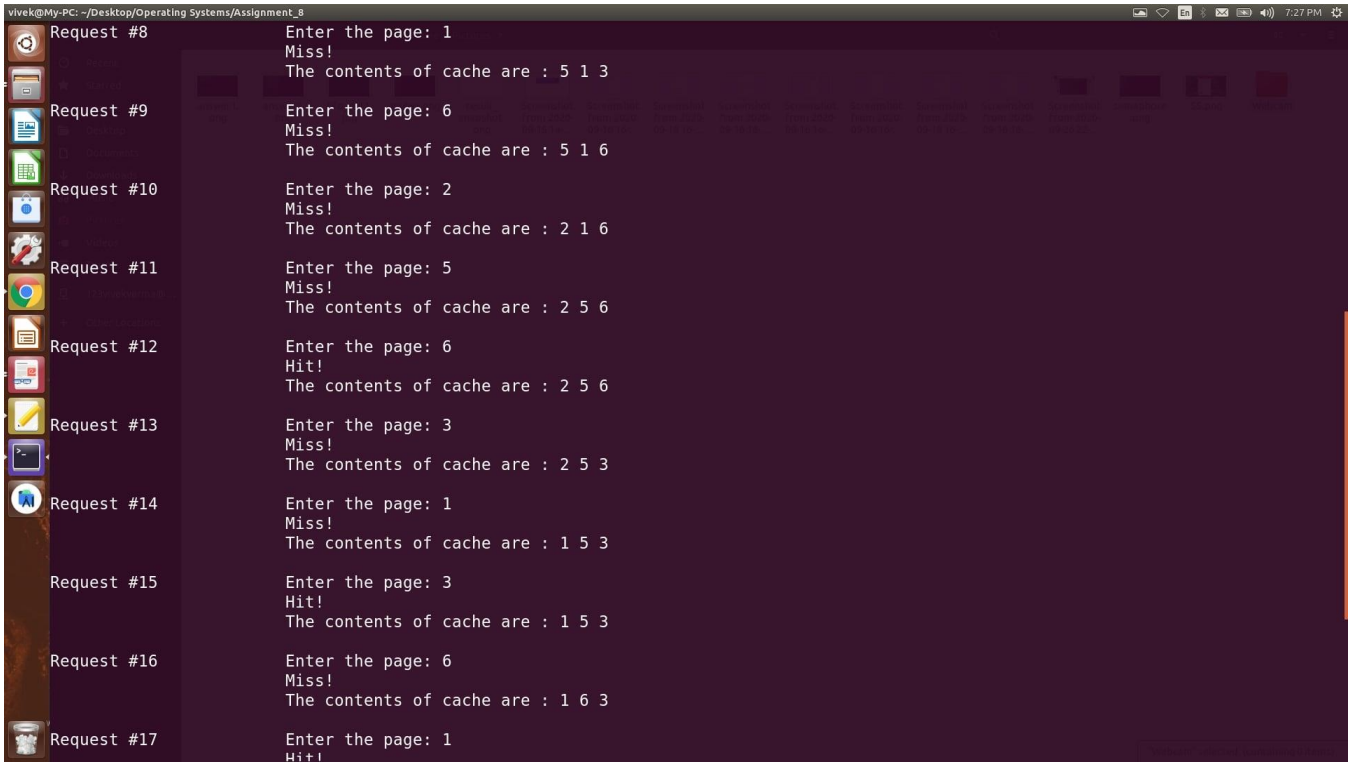
```
#include<bits/stdc++.h>

using namespace std;

class Cache
{
	private:
		int *cache,n,*time1;
		void increase();

	public:
		Cache(int);
		bool find(int x);
		void display();
};

void Cache::display()
{
	cout<<"	\t\tThe contents of cache are : ";
	for(int i=0;i<n;++i)
	{
		cout<<cache[i]<<" ";
	}
	cout<<endl;
}

void Cache::increase()
{
	for(int i=0;i<n;++i)
	{
		if(cache[i]!=-1)
		{
			time1[i] =  time1[i] + 1;
		}
	}
}

Cache::Cache(int n)
{
	cout<<"\nCache implementing FIFO page replacement policy created!\n";
	this -> n = n;
	cache = new int[n];
	time1 = new int[n];
	for(int i=0;i<n;++i)
	{
```

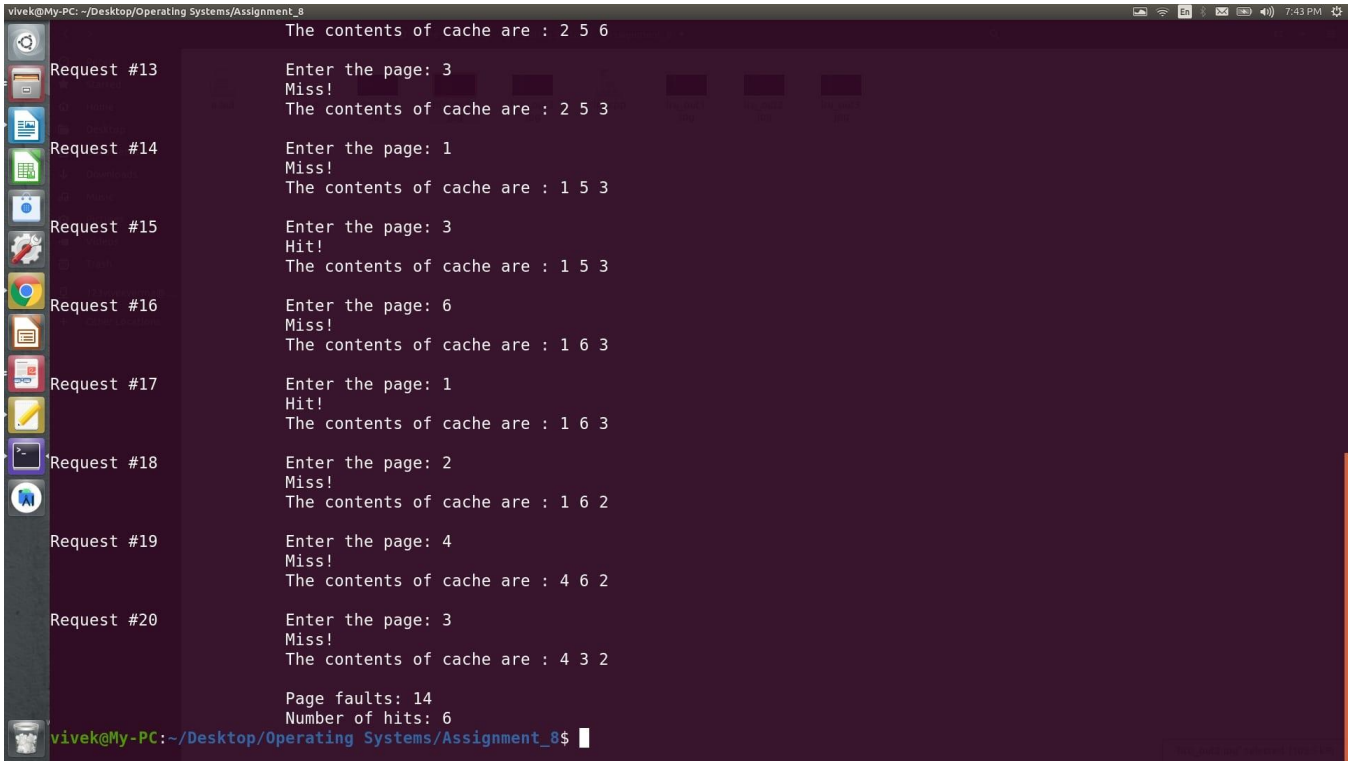
```
                cache[i] = -1;
                time1[i] = INT_MAX;
        }
}

bool Cache::find(int x)
{
        bool hit_miss = false;
        int idx = -1,i;
        for(i=0;i<n;++i)
        {
                if(cache[i]==x)
                {
                        idx = i;
                        break;
                }
        }
        if(idx==-1)
        {
                //miss has occured
                hit_miss = false;
                int last_used = -1, last_used_time = INT_MIN;
                for(i=0;i<n;++i)
                {
                        if(time1[i]>last_used_time)
                        {
                                last_used_time = time1[i];
                                last_used = i;
                        }
                }
                cache[last_used] = x;
                time1[last_used] = 0;
        }
        else
        {
                //hit has occured
                hit_miss = true;
                //time1[idx] = 0; the only difference between both algorithms
        }
        this -> increase();
        return hit_miss;
}

int main()
{
        int n,f,i,page,faults = 0;
        cout<<"Enter the number of requests: ";
```

```
        cin>>n;
        cout<<"Enter the frame size: ";
        cin>>f;
        Cache c1(f);
        cout<<"\n";
        for(i=0;i<n;++i)
        {
                cout<<"Request #"<<i+1<<"\t\t";
                cout<<"Enter the page: ";
                cin>>page;
                if(c1.find(page)==false)
                {
                        faults += 1;
                        cout<<"         \t\tMiss!"<<endl;
                }
                else
                {
                        cout<<"         \t\tHit!"<<endl;
                }
                c1.display();
                cout<<"\n";
        }
        cout<<"         \t\tPage faults: "<<faults<<endl;
        cout<<"         \t\tNumber of hits: "<<n - faults<<endl;
        return 0;
}
```

Output


vivek@My-PC: ~/Desktop/Operating Systems/Assignment_8
7:26 PM
vivek@My-PC:~/Desktop/Operating Systems/Assignment_8$ g++ fifo.cpp
vivek@My-PC:~/Desktop/Operating Systems/Assignment_8$ ./a.out
Enter the number of requests: 20
Enter the frame size: 3
Cache implementing FIFO page replacement policy created!
Request #1 Enter the page: 1
Miss!
The contents of cache are : 1 -1 -1
Request #2 Enter the page: 2
Miss!
The contents of cache are : 1 2 -1
Request #3 Enter the page: 3
Miss!
The contents of cache are : 1 2 3
Request #4 Enter the page: 2
Hit!
The contents of cache are : 1 2 3
Request #5 Enter the page: 1
Hit!
The contents of cache are : 1 2 3
Request #6 Enter the page: 5
Miss!
The contents of cache are : 5 2 3
Request #7 Enter the page: 2
Hit!
The contents of cache are : 5 2 3
Request #8 Enter the page: 1
Miss!
The contents of cache are : 5 1 3


vivek@My-PC: ~/Desktop/Operating Systems/Assignment_8
7:27 PM
Request #8 Enter the page: 1
Miss!
The contents of cache are : 5 1 3
Request #9 Enter the page: 6
Miss!
The contents of cache are : 5 1 6
Request #10 Enter the page: 2
Miss!
The contents of cache are : 2 1 6
Request #11 Enter the page: 5
Miss!
The contents of cache are : 2 5 6
Request #12 Enter the page: 6
Hit!
The contents of cache are : 2 5 6
Request #13 Enter the page: 3
Miss!
The contents of cache are : 2 5 3
Request #14 Enter the page: 1
Miss!
The contents of cache are : 1 5 3
Request #15 Enter the page: 3
Hit!
The contents of cache are : 1 5 3
Request #16 Enter the page: 6
Miss!
The contents of cache are : 1 6 3
Request #17 Enter the page: 1



Result: FIFO page replacement algorithm was implemented and verified successfully.

Experiment Number 13: Simulate Page Replacement Algorithm: LRU

Date: October 20, 2020

Aim: To Simulate LRU page replacement algorithm.

Least Recently Used: In this algorithm page will be replaced which is least recently used.

Example3: Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 with 4-page frames. Find number of page faults.

Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —>**4 Page faults**

0 is already there so —>**0 Page fault.**

When 3 came it will take the place of 7 because it is least recently used —>**1 Page fault**

0 is already in memory so —>**0 Page fault**. 4 will takes place of 1 —>**1 Page Fault**

Now for the further page reference string —>**0 Page fault** because they are already available in the memory.

Algorithm

1. Start
2. Read the number of frames
3. Read the number of pages
4. Read the page numbers
5. Initialize the values in frames to -1
6. Allocate the pages in to frames by selecting the page that has not been used for the longest period of time.
7. Display the number of page faults.
8. Stop

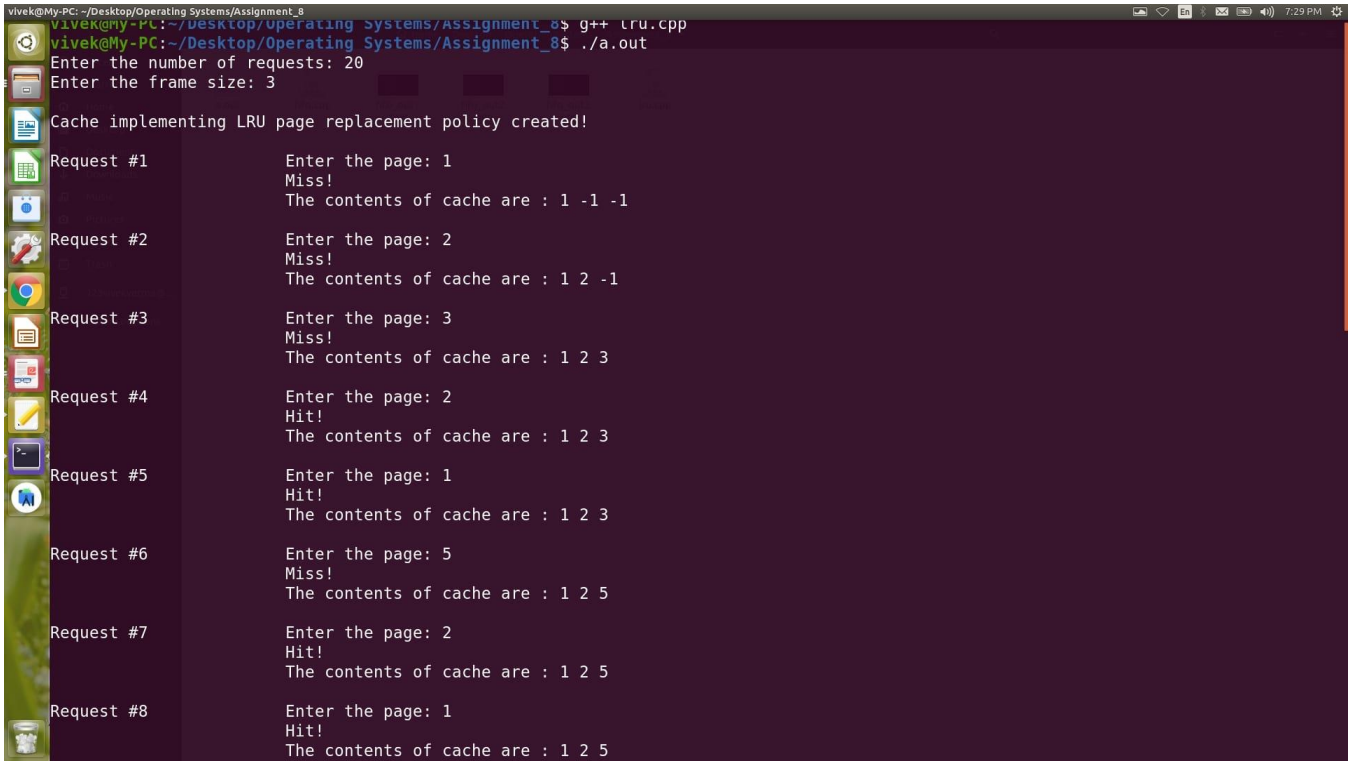
C++ Implementation

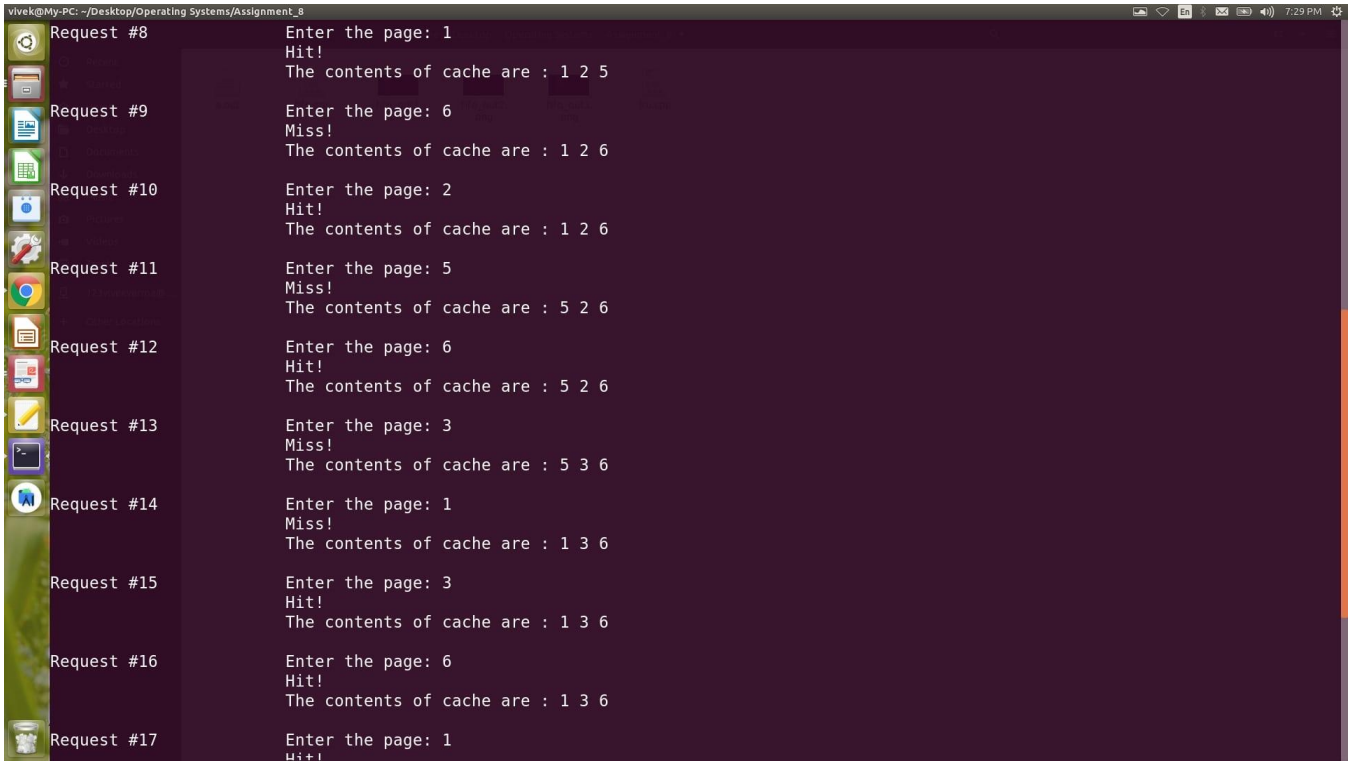
```
#include<bits/stdc++.h>

//implementation differs only on line number 82

using namespace std;

class Cache
{
        private:
```

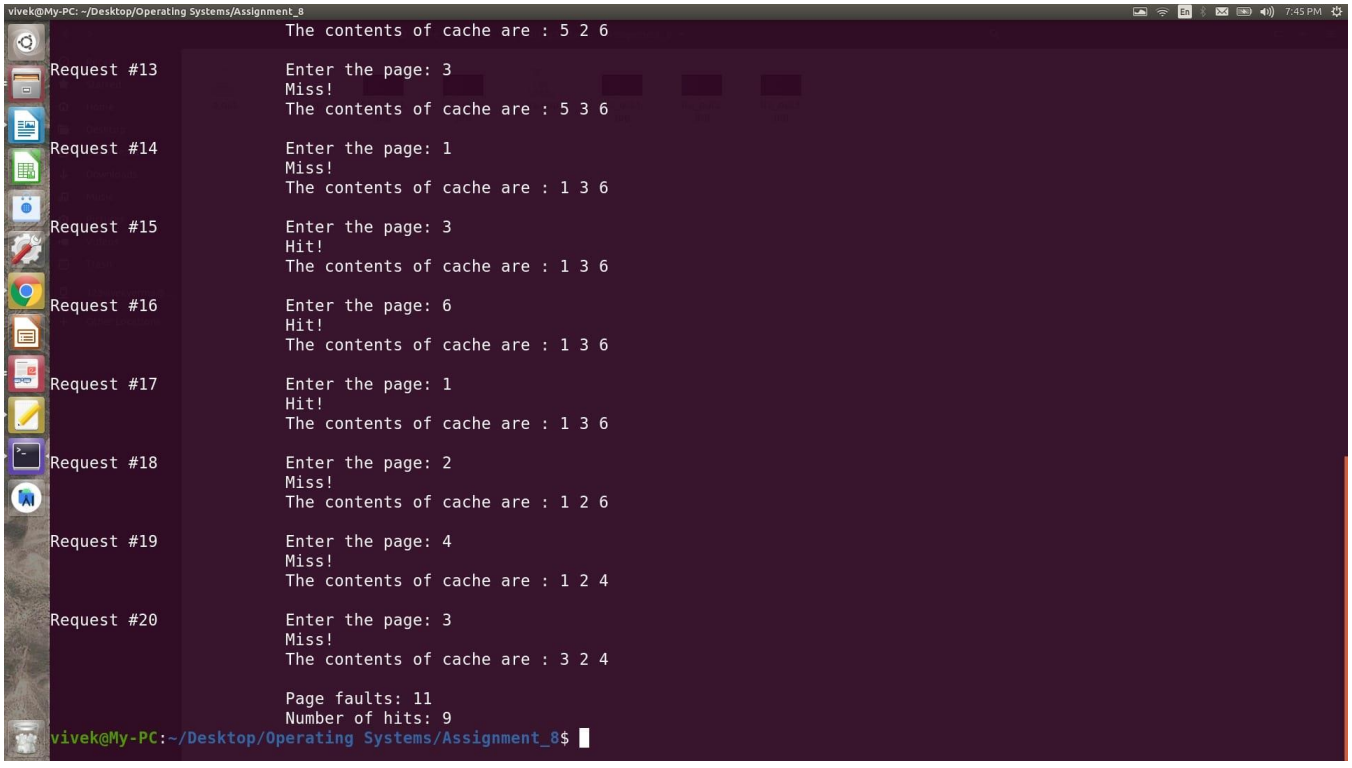
```
		int *cache,n,*time1;
		void increase();

	public:
		Cache(int);
		bool find(int x);
		void display();
};

void Cache::display()
{
	cout<<"	\t\tThe contents of cache are : ";
	for(int i=0;i<n;++i)
	{
		cout<<cache[i]<<" ";
	}
	cout<<endl;
}

void Cache::increase()
{
	for(int i=0;i<n;++i)
	{
		if(cache[i]!=-1)
		{
			time1[i] =  time1[i] + 1;
		}
	}
}

Cache::Cache(int n)
{
	cout<<"\nCache implementing LRU page replacement policy created!\n";
	this -> n = n;
	cache = new int[n];
	time1 = new int[n];
	for(int i=0;i<n;++i)
	{
		cache[i] = -1;
		time1[i] = INT_MAX;
	}
}

bool Cache::find(int x)
{
	bool hit_miss = false;
	int idx = -1,i;
```

```
        for(i=0;i<n;++i)
        {
                if(cache[i]==x)
                {
                        idx = i;
                        break;
                }
        }
        if(idx==-1)
        {
                //miss has occured
                hit_miss = false;
                int last_used = -1, last_used_time = INT_MIN;
                for(i=0;i<n;++i)
                {
                        if(time1[i]>last_used_time)
                        {
                                last_used_time = time1[i];
                                last_used = i;
                        }
                }
                cache[last_used] = x;
                time1[last_used] = 0;
        }
        else
        {
                //hit has occured
                hit_miss = true;
                time1[idx] = 0;
        }
        this -> increase();
        return hit_miss;
}

int main()
{
        int n,f,i,page,faults = 0;
        cout<<"Enter the number of requests: ";
        cin>>n;
        cout<<"Enter the frame size: ";
        cin>>f;
        Cache c1(f);
        cout<<"\n";
        for(i=0;i<n;++i)
        {
                cout<<"Request #"<<i+1<<"\t\t";
                cout<<"Enter the page: ";
```

```
            cin>>page;
            if(c1.find(page)==false)
            {
                  faults += 1;
                  cout<<"       \t\tMiss!"<<endl;
            }
            else
            {
                  cout<<"       \t\tHit!"<<endl;
            }
            c1.display();
            cout<<"\n";
      }
      cout<<"       \t\tPage faults: "<<faults<<endl;
      cout<<"       \t\tNumber of hits: "<<n - faults<<endl;
      return 0;

}
```




vivek@My-PC: ~/Desktop/Operating Systems/Assignment_8
7:29 PM
Request #8 Enter the page: 1
Hit!
The contents of cache are : 1 2 5
Request #9 Enter the page: 6
Miss!
The contents of cache are : 1 2 6
Request #10 Enter the page: 2
Hit!
The contents of cache are : 1 2 6
Request #11 Enter the page: 5
Miss!
The contents of cache are : 5 2 6
Request #12 Enter the page: 6
Hit!
The contents of cache are : 5 2 6
Request #13 Enter the page: 3
Miss!
The contents of cache are : 5 3 6
Request #14 Enter the page: 1
Miss!
The contents of cache are : 1 3 6
Request #15 Enter the page: 3
Hit!
The contents of cache are : 1 3 6
Request #16 Enter the page: 6
Hit!
The contents of cache are : 1 3 6
Request #17 Enter the page: 1



Result: LRU Page Replacement Algorithm was implemented and verified successfully.

Experiment Number 14: Simulate Fixed Partitioning: First Fit, Best Fit and Worst Fit

Date: October 27, 2020

Aim: To simulate first fir, best fit and worst fit on a fixed partitioning disk.

C++ Implementation

```
#include<bits/stdc++.h>

using namespace std;

class memory_fixed
{
	private:
		vector <pair<int,int>> p1;
		vector <int> p2;
		bool *arr;
		int n1, n2;
		vector <int> record;

	public:
		memory_fixed(vector<int>&,vector<int>&);
		void first_fit();
		void best_fit();
		void worst_fit();
		void clear();
		void display();
};

void memory_fixed::clear()
{
	int i,j;
	for(i=0;i<n1;++i)
	{
		p1[i].first = 0;
		arr[i] = false;
		record[i] = -1;
	}
}

void memory_fixed::display()
{
	int i,j,internal = 0,external = 0;
	cout<<left<<setw(15)<<"Partition"<<left<<setw(15)<<"Fragment
```

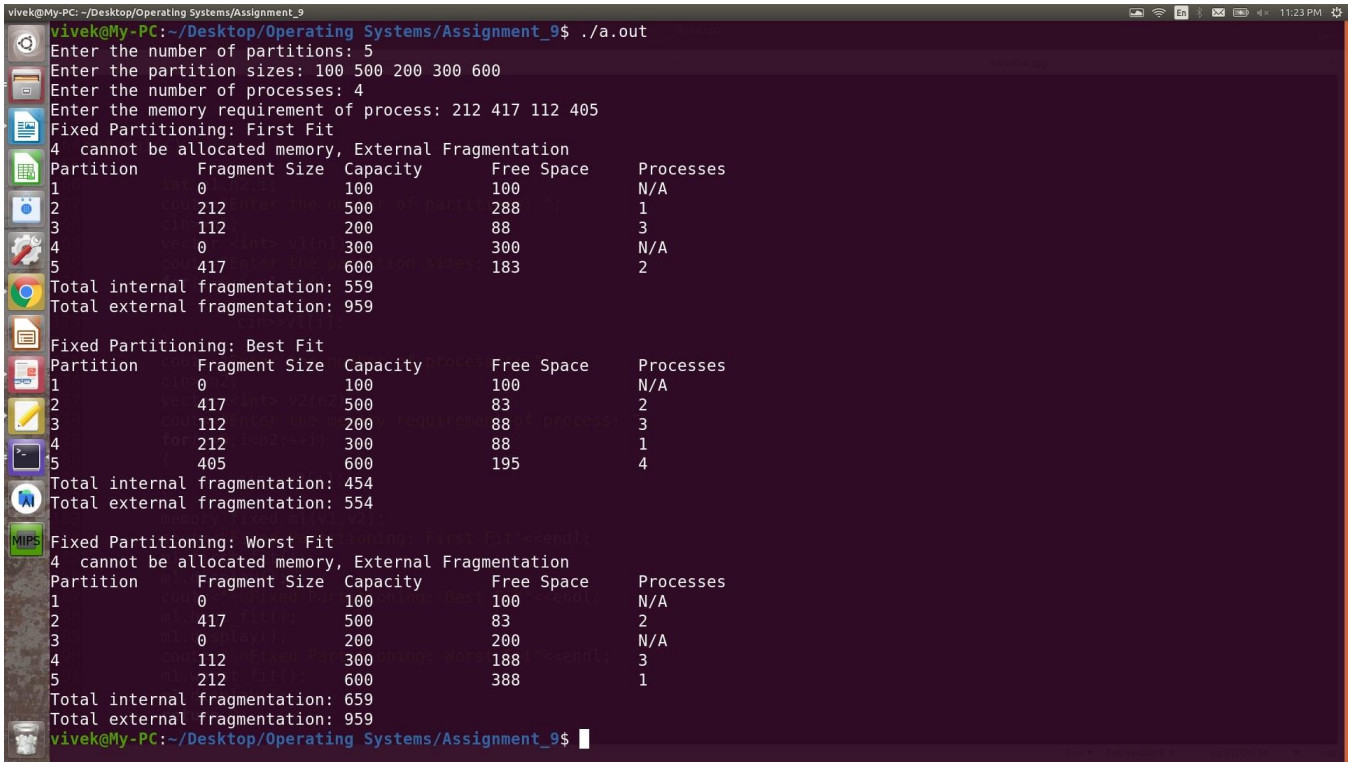
```
Size"<<left<<setw(15)<<"Capacity"<<left<<setw(15)<<"Free
Space"<<left<<setw(15)<<"Processes"<<endl;
        for(i=0;i<n1;++i)
        {

        cout<<left<<setw(15)<<i+1<<left<<setw(15)<<p1[i].first<<left<<setw(15)<<p1[i].secon
d<<left<<setw(15)<<p1[i].second - p1[i].first;
                if(record[i]==-1)
                {
                        cout<<"N/A";
                }
                else
                {
                        internal += p1[i].second - p1[i].first;
                        cout<<record[i];
                }
                cout<<endl;
                external += p1[i].second - p1[i].first;
        }
        cout<<"Total internal fragmentation: "<<internal<<endl;
        cout<<"Total external fragmentation: "<<external<<endl;
}

memory_fixed::memory_fixed(vector <int>&partitions, vector <int>&processes)
{
        p2 = processes;
        n1 = partitions.size();
        n2 = processes.size();
        arr = new bool[n1];
        int i;
        for(i=0;i<n1;++i)
        {
                p1.push_back({0,partitions[i]});
                arr[i] = false;
                record.push_back(-1);
        }
}

void memory_fixed::first_fit()
{
        clear();
        int i,j,flag = 0;
        for(i=0;i<n2;++i)
        {
                flag = 0;
                for(j=0;j<n1;++j)
                {
```

```
                        if(p2[i]<=p1[j].second and arr[j]==false)
                        {
                                ++flag;
                                break;
                        }
                }
                if(flag==1)
                {
                        arr[j] = true;
                        p1[j].first = p2[i];
                        record[j] = i + 1;
                }
                else
                {
                        cout<<(i+1)<<" "<<" cannot be allocated memory, External
Fragmentation"<<endl;
                }
        }
}

void memory_fixed::best_fit()
{
        clear();
        int i,j,best_size,idx = -1;
        for(i=0;i<n2;++i)
        {
                best_size = INT_MAX;
                idx = -1;
                for(j=0;j<n1;++j)
                {
                        if(p1[j].second>= p2[i] and arr[j]==false)
                        {
                                if(p1[j].second<best_size)
                                {
                                        best_size = p1[j].second;
                                        idx = j;
                                }
                        }
                }
                if(idx==-1)
                {
                        cout<<(i+1)<<" "<<" cannot be allocated memory, External
Fragmentation"<<endl;
                }
                else
                {
                        p1[idx].first = p2[i];
```

```
                        arr[idx] = true;
                        record[idx] = i + 1;
                }
        }
}

void memory_fixed::worst_fit()
{
        clear();
        int i,j,best_size,idx = -1;
        for(i=0;i<n2;++i)
        {
                best_size = INT_MIN;
                idx = -1;
                for(j=0;j<n1;++j)
                {
                        if(p1[j].second>= p2[i] and arr[j]==false)
                        {
                                if(p1[j].second>best_size)
                                {
                                        best_size = p1[j].second;
                                        idx = j;
                                }
                        }
                }
                if(idx==-1)
                {
                        cout<<(i+1)<<" "<<" cannot be allocated memory, External
Fragmentation"<<endl;
                }
                else
                {
                        p1[idx].first = p2[i];
                        arr[idx] = true;
                        record[idx] = i + 1;
                }
        }
}

int main()
{
        int n1,n2,i;
        cout<<"Enter the number of partitions: ";
        cin>>n1;
        vector <int> v1(n1);
        cout<<"Enter the partition sizes: ";
        for(i=0;i<n1;++i)
```

```
        {
                cin>>v1[i];
        }
        cout<<"Enter the number of processes: ";
        cin>>n2;
        vector <int> v2(n2);
        cout<<"Enter the memory requirement of process: ";
        for(i=0;i<n2;++i)
        {
                cin>>v2[i];
        }
        memory_fixed m1(v1,v2);
        cout<<"Fixed Partitioning: First Fit"<<endl;
        m1.first_fit();
        m1.display();
        cout<<"\nFixed Partitioning: Best Fit"<<endl;
        m1.best_fit();
        m1.display();
        cout<<"\nFixed Partitioning: Worst Fit"<<endl;
        m1.worst_fit();
        m1.display();
        return 0;
}
```

Output



Result: Fixed Partitioning using first fit, best fit and worst fit was implemented and verified successfully.

Experiment Number 15: Simulate Variable Partitioning: First Fit, Best Fit and Worst Fit

Date: October 27, 2020

Aim: To simulate first fir, best fit and worst fit on a variable partitioning disk.

C++ Implementation

```
#include<bits/stdc++.h>

using namespace std;

class memory_variable
{
	private:
		vector <pair<int,int>> p1;
		vector <int> p2;
		bool *arr;
		int n1, n2;
		vector <vector <int>> record;

	public:
		memory_variable(vector<int>&,vector<int>&);
		void first_fit();
		void best_fit();
		void worst_fit();
		void clear();
		void display();
};

void memory_variable::clear()
{
	int i,j;
	for(i=0;i<n1;++i)
	{
		p1[i].first = 0;
		arr[i] = false;
		record[i].clear();
	}
}

void memory_variable::display()
{
	int i,j,internal = 0, external = 0;
	cout<<left<<setw(15)<<"Partition"<<left<<setw(15)<<"Fragment
```

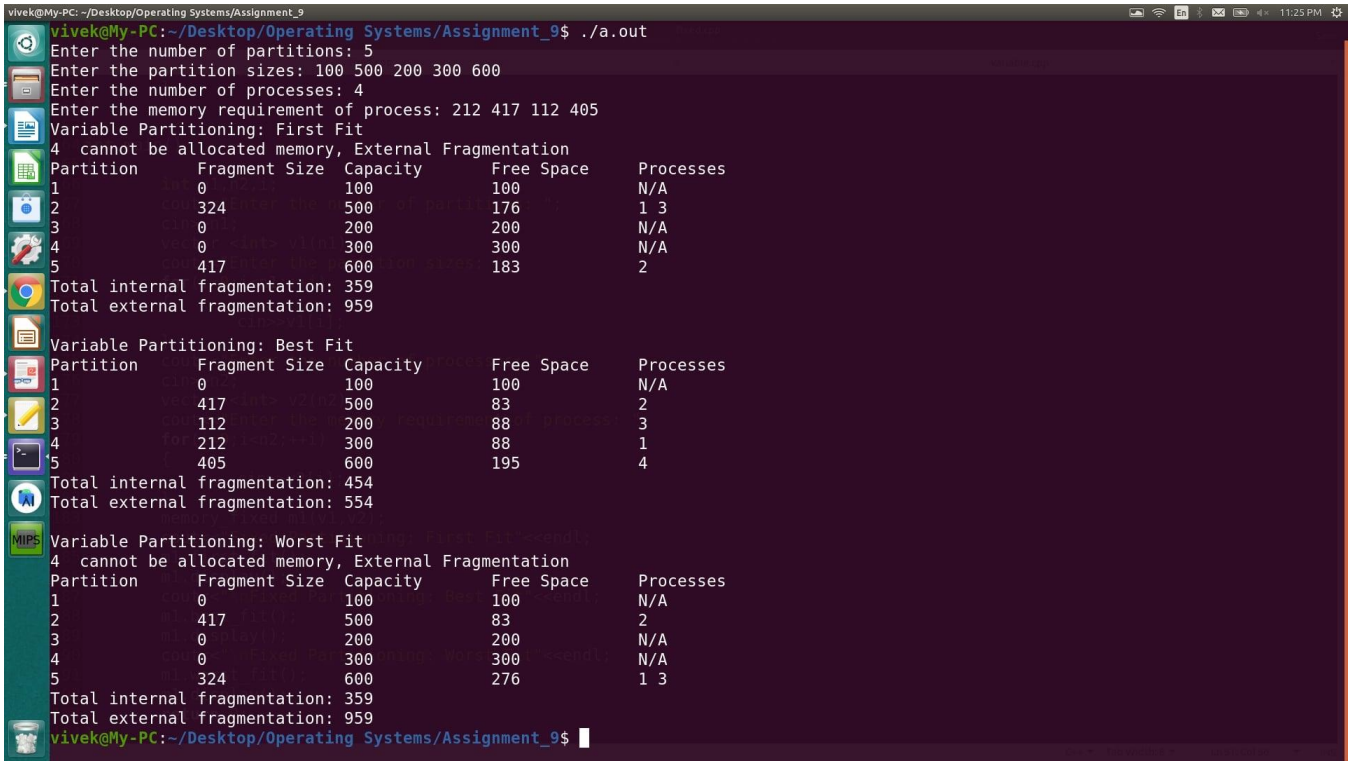
```
Size"<<left<<setw(15)<<"Capacity"<<left<<setw(15)<<"Free
Space"<<left<<setw(15)<<"Processes"<<endl;
        for(i=0;i<n1;++i)
        {
        cout<<left<<setw(15)<<i+1<<left<<setw(15)<<p1[i].first<<left<<setw(15)<<p1[i].secon
d<<left<<setw(15)<<p1[i].second - p1[i].first;
                for(j=0;j<record[i].size();++j)
                {
                        cout<<record[i][j]<<" ";
                }
                if(record[i].size()==0)
                {
                        cout<<"N/A";
                }
                else
                {
                        internal += p1[i].second - p1[i].first;
                }
                cout<<endl;
                external += p1[i].second - p1[i].first;
        }
        cout<<"Total internal fragmentation: "<<internal<<endl;
        cout<<"Total external fragmentation: "<<external<<endl;
}

memory_variable::memory_variable(vector <int>&partitions, vector <int>&processes)
{
        p2 = processes;
        n1 = partitions.size();
        n2 = processes.size();
        arr = new bool[n1];
        int i;
        for(i=0;i<n1;++i)
        {
                p1.push_back({0,partitions[i]});
                arr[i] = false;
                vector <int> temp;
                record.push_back(temp);
        }
}

void memory_variable::first_fit()
{
        clear();
        int i,j,flag = 0;
        for(i=0;i<n2;++i)
        {
```

```
                flag = 0;
                for(j=0;j<n1;++j)
                {
                        if(p2[i]<=p1[j].second - p1[j].first)
                        {
                                ++flag;
                                break;
                        }
                }
                if(flag==1)
                {
                        p1[j].first += p2[i];
                        record[j].push_back(i+1);
                }
                else
                {
                        cout<<(i+1)<<" "<<" cannot be allocated memory, External
Fragmentation"<<endl;
                }
        }
}

void memory_variable::best_fit()
{
        clear();
        int i,j,best_size,idx = -1;
        for(i=0;i<n2;++i)
        {
                best_size = INT_MAX;
                idx = -1;
                for(j=0;j<n1;++j)
                {
                        if(p1[j].second - p1[j].first>= p2[i])
                        {
                                if(p1[j].second<best_size)
                                {
                                        best_size = p1[j].second - p1[j].first;
                                        idx = j;
                                }
                        }
                }
                if(idx==-1)
                {
                        cout<<(i+1)<<" "<<" cannot be allocated memory, External
Fragmentation"<<endl;
                }
                else
```

```
                {
                        p1[idx].first += p2[i];
                        record[idx].push_back(i+1);
                }
        }
}

void memory_variable::worst_fit()
{
        clear();
        int i,j,best_size,idx = -1;
        for(i=0;i<n2;++i)
        {
                best_size = INT_MIN;
                idx = -1;
                for(j=0;j<n1;++j)
                {
                        if(p1[j].second - p1[j].first>= p2[i])
                        {
                                if(p1[j].second - p1[j].second >= best_size)
                                {
                                        best_size = p1[j].second - p1[j].second;
                                        idx = j;
                                }
                        }
                }
                if(idx==-1)
                {
                        cout<<(i+1)<<" "<<" cannot be allocated memory, External
Fragmentation"<<endl;
                }
                else
                {
                        p1[idx].first += p2[i];
                        record[idx].push_back(i + 1);
                }
        }
}

int main()
{
        int n1,n2,i;
        cout<<"Enter the number of partitions: ";
        cin>>n1;
        vector <int> v1(n1);
        cout<<"Enter the partition sizes: ";
        for(i=0;i<n1;++i)
```

```
        {
                cin>>v1[i];
        }
        cout<<"Enter the number of processes: ";
        cin>>n2;
        vector <int> v2(n2);
        cout<<"Enter the memory requirement of process: ";
        for(i=0;i<n2;++i)
        {
                cin>>v2[i];
        }
        memory_variable m1(v1,v2);
        cout<<"Variable Partitioning: First Fit"<<endl;
        m1.first_fit();
        m1.display();
        cout<<"\nVariable Partitioning: Best Fit"<<endl;
        m1.best_fit();
        m1.display();
        cout<<"\nVariable Partitioning: Worst Fit"<<endl;
        m1.worst_fit();
        m1.display();
        return 0;
}
```

Output



Result: Variable Partitioning using first fit, best fit and worst fit was implemented and verified successfully.