ARYABHATTA COLLEGE

Department of Computer Science
University of Delhi

Examination Roll no - 21059570007
Semester - 3rd
B.Sc. H Computer Science

OPERATING SYSTEM
PRACTICAL FILE

Submitted by :
Devashish Kumar

Submitted to :
Mr. Deepak Sharma

S.NO.	TITLE	PAGE NO.
1.	Write a program (using fork() and/or exec() commands) where parent and child execute: a) same program, same code. b) same program, different code. c) before terminating, the parent waits for the child to finish its task, both for above mentioned cases a) and b).	
2.	Write a program to show how multiple fork() system calls work.	
3.	Write a program to report behaviour of Linux kernel including kernel version, CPU typeand model. (CPU information).	
4.	Write a program to report behaviour of Linux kernel including information onconfigured memory, amount of free and used memory. (Memory information).	
5.	Write a program to print file details including owner access permissions, file accesstime, where file name is given as command line argument.	
6.	Write a program to copy files using system calls.	
7.	Write a program to implement FCFS scheduling algorithm.	
8.	Write a program to implement Round Robin scheduling algorithm.	
9.	Write a program to implement SJF scheduling algorithm.	
10.	Write a program to implement non-preemptive priority based scheduling algorithm.	
11.	Write a program to implement preemptive priority based scheduling algorithm.	
12.	Write a program to implement SRTF scheduling algorithm.	

13.	Write a program to calculate sum of n numbers using thread library.	
14.	Write a program to implement first-fit, best-fit and worst-fit allocation strategies.	

1- (a)

```
#include<iostream>
#include<unistd.h>
#include<sys/types.h>
#include<sys/wait.h>
using namespace std;
int main()
{
int code=fork();
int status;
if(code<0)
{
cout<<"UNSUCCESSFUL"<<endl;
}
else
{
cout<<"ProcessID = "<< getpid() << endl;
cout<<"return code : " << code << endl;
}
return 0;
}
```

OUTPUT:

~

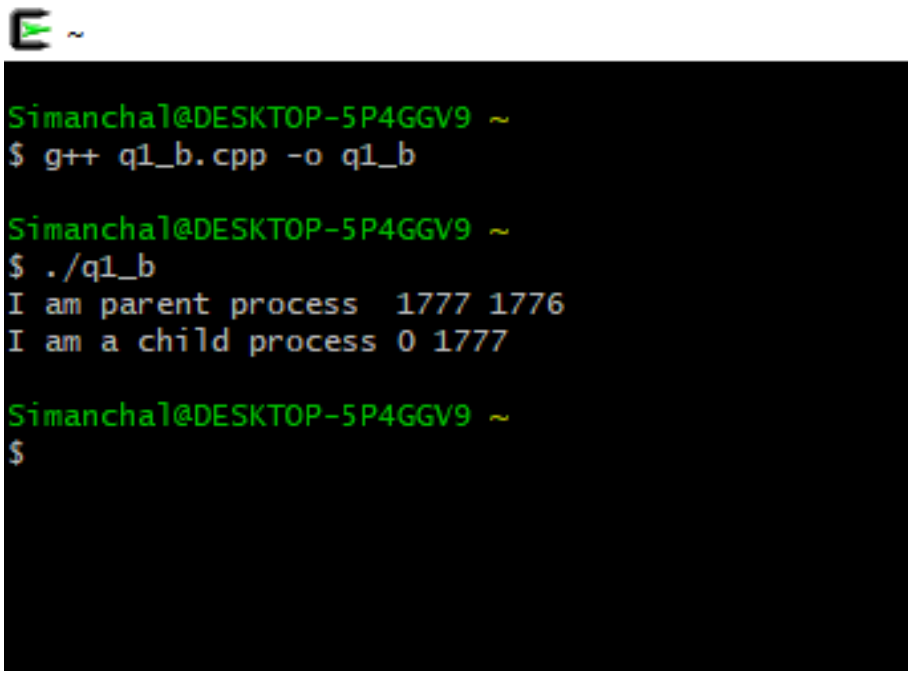
```
Simanchal@DESKTOP-5P4GGV9 ~
$ g++ q1_a.cpp -o q1_a

Simanchal@DESKTOP-5P4GGV9 ~
$ ./q1_a
ProcessID = 1768
ProcessID = 1769
return code : 1769
return code : 0

Simanchal@DESKTOP-5P4GGV9 ~
$
```

(b)

```
#include<iostream>

#include<unistd.h>

using namespace std;

int main()

{

int pid = fork();

if (pid<0)

{

cout<<"UNSUCCESSFUL"<<endl;

return -1;

}

else if(pid==0)

{

cout<<"I am a child process" <<" "<< pid <<" " <<getpid()<<endl;

}

else

{

sleep(-5);
```

```
cout<<"I am parent process " <<" " << pid <<" " << getpid() <<endl;
}
return 0;
}
```

OUTPUT:

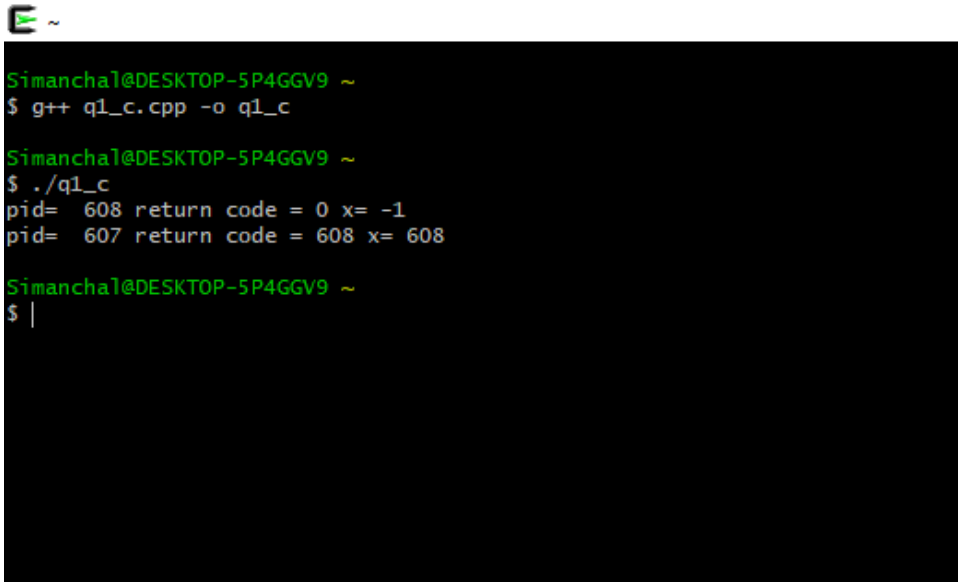


(a)-(c)

```
#include<iostream>
#include<unistd.h>
#include<sys/types.h>
#include<sys/wait.h>
using namespace std;
int main()
{
        int code = fork();
        int status,x;
        if(code<0)
        {
                cout<<"UNSUCCESSFUL"<<endl;
        }
```

```
        else //Run same code for child and parent process
        {
                x = wait(&status);
                cout<<"pid= " <<" "<< getpid() <<" "<< "return code = " << code<<" "<<"x="<<" " << x <<endl;
        }
        return 0;
}
```

OUTPUT:



(b)-(c)

```
#include<iostream>
#include<sys/types.h>
#include<sys/wait.h>
#include<unistd.h>
#include<stdio.h>
using namespace std;
int main()
{
int status;
int pid=fork();
int x;
```

```
if(pid<0)
{
cout<<"Child process cannot be created\n";
return -1;
}
else if(pid==0)
{
cout<<"Child Executing : "<<pid<<"\n";
cout<<"\nI am Child. Child process id: "<<getpid()<<"\n";
execlp("/bin/ls", "ls", NULL);
}
else
{
x=wait(&status);
cout<<"\nChild complete \n";
cout<<"\nI am Parent. Parent process id: "<<getpid()<<"\n";
cout<<"\nInfo returned by wait(&status): "<<x<<" which is = child process
id\n";
}
return 0;
}
```

OUTPUT:

~

```
Simanchal@DESKTOP-5P4GGV9 ~
$ g++ q1_d.cpp -o q1_d

Simanchal@DESKTOP-5P4GGV9 ~
$ ./q1_d
Child Executing : 0

I am Child. Child process id: 17
 FCFS.cpp   'New Text Document.txt'   q1_a.exe   q1_c.exe   sjf.exe
 FCFS.exe    RoundRobin.cpp           q1_b.cpp   q1_d.cpp
 Hello.cpp   RoundRobin.exe           q1_b.exe   q1_d.exe
 Hello.exe   q1_a.cpp                 q1_c.cpp   sjf.cpp

Child complete

I am Parent. Parent process id: 16

Info returned by wait(&status): 17 which is = child process id

Simanchal@DESKTOP-5P4GGV9 ~
$
```

2-code:

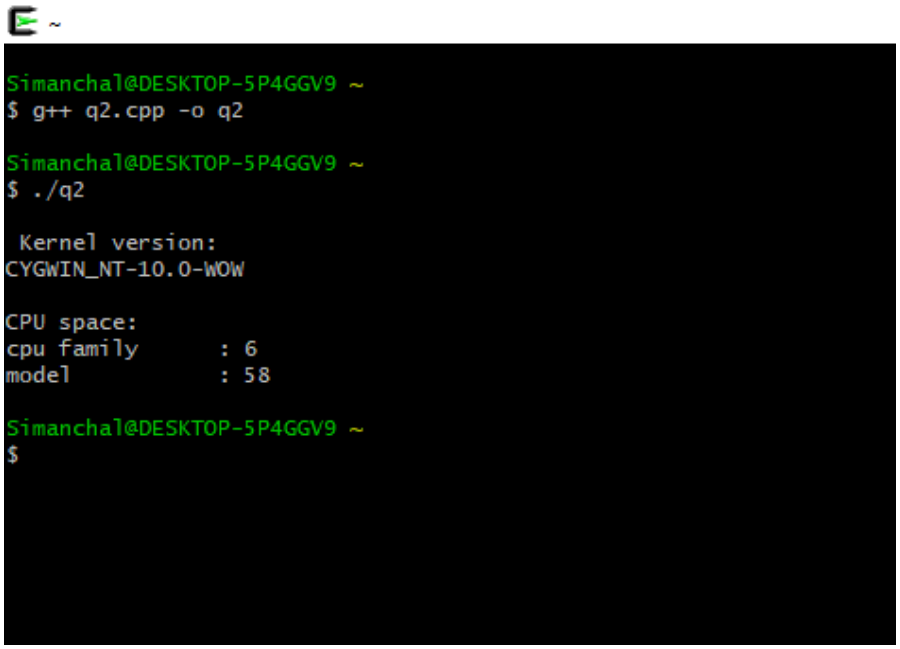
OUTPUT:

3- Write a program to report behaviour of Linux kernel including kernel version, CPU typeand model. (CPU information).

-code:

```
#include<iostream>

using namespace std;

int main()

{

        cout<<"\n Kernel version:\n";

        system("uname -s");

        cout<<"\nCPU space: \n";

        system("cat /proc/cpuinfo |awk 'NR==3,NR==4{print}' \n");

        return 0;

}
```

OUTPUT:



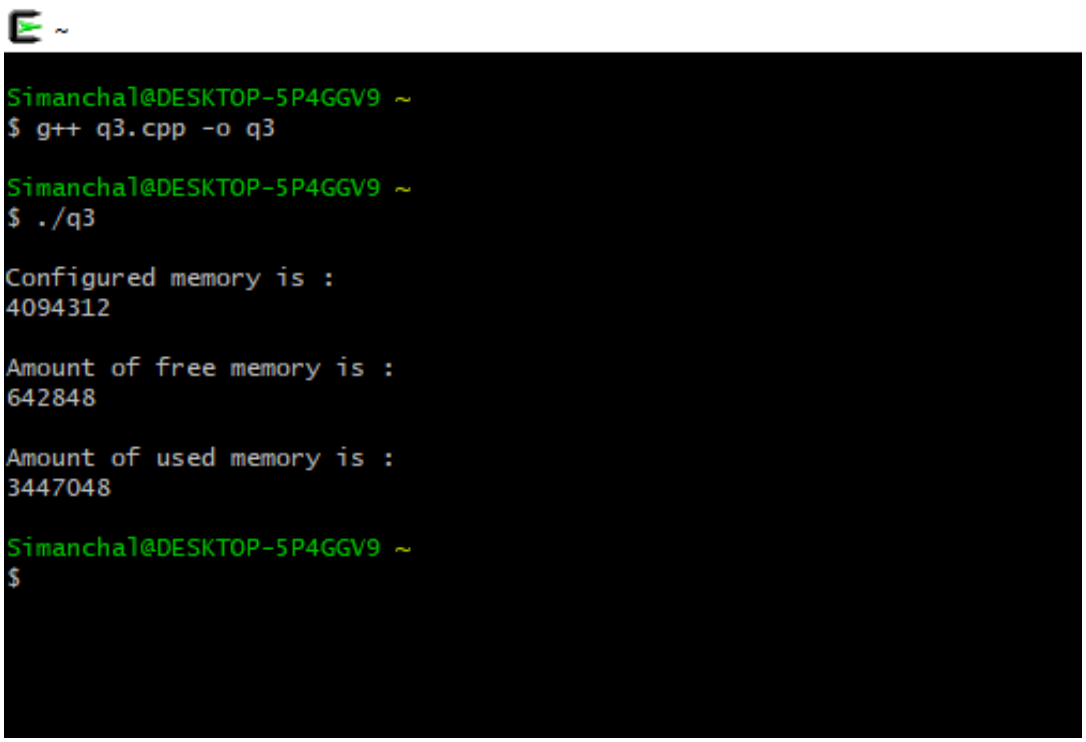
QUESTION 4:

Write a program to report behaviour of Linux kernel including information onconfigured memory, amount of free and used memory. (Memory information).

code:

```
#include<iostream>

using namespace std;

int main()

{

        cout<<"\nConfigured memory is :\n";

        system("cat /proc/meminfo |awk 'NR==1{print $2}'\n");

        cout<<"\nAmount of free memory is :\n";

        system("cat /proc/meminfo |awk 'NR==2{print $2}'\n");

        cout<<"\nAmount of used memory is :\n";

        system("cat /proc/meminfo |awk '{if (NR==1) a=$2; if (NR==2) b=$2 } END {print a-b}'\n");


        return 0;

}
```

OUTPUT:



QUESTION 5:-

Write a program to print file details including owner access permissions, file accesstime, where file name is given as command line argument.

code:

```
#include<iostream>
#include<stdlib.h>
#include<stdio.h>
#include<unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
using namespace std;
int main(int argc, char** argv)
        {
                if(argc !=2)
```

```
        {
                cout<<"\nEnter file name!\n";
                return 1;
        }
    struct stat fileStat;
        if(stat(argv[1],&fileStat)<0)
        return 1;
    cout<<"\nFile details for "<< argv[1]<<" are :\n";
    cout<<"File Size: "<<fileStat.st_size<<" bytes\n";
    cout<<" time of last access is : "<<ctime(&fileStat.st_atime);
    cout<<" time of last modification is : " << ctime(&fileStat.st_mtime);
    cout<<" time of last change is : "<< ctime(&fileStat.st_ctime);
    cout<<"File Permissions: \t";
    cout<<( (S_ISDIR(fileStat.st_mode)) ? "d" : "-");
    cout<<( (fileStat.st_mode & S_IRUSR) ? "r" : "-");
    cout<<( (fileStat.st_mode & S_IWUSR) ? "w" : "-");
    cout<<( (fileStat.st_mode & S_IXUSR) ? "x" : "-");
    cout<<( (fileStat.st_mode & S_IRGRP) ? "r" : "-");
    cout<<( (fileStat.st_mode & S_IWGRP) ? "w" : "-");
    cout<<( (fileStat.st_mode & S_IXGRP) ? "x" : "-");
    cout<<( (fileStat.st_mode & S_IROTH) ? "r" : "-");
    cout<<( (fileStat.st_mode & S_IWOTH) ? "w" : "-");
    cout<<( (fileStat.st_mode & S_IXOTH) ? "x" : "-");
    cout<<endl;
return 0;
}
```

OUTPUT:

~

```
Simanchal@DESKTOP-5P4GGV9 ~
$ g++ q4.cpp -o q4

Simanchal@DESKTOP-5P4GGV9 ~
$ ./q4

Enter file name!

Simanchal@DESKTOP-5P4GGV9 ~
$ ./q4.exe q2.cpp

File details for q2.cppare :
File Size: 305bytes
time of last access is : Fri Dec  3 18:54:36 2021
time of last modification is : Fri Dec  3 18:54:14 2021
time of last change is : Fri Dec  3 18:54:14 2021
File Permissions:       -rwxr-xr-x

Simanchal@DESKTOP-5P4GGV9 ~
$
```

QUESTION 6:-

Write a program to copy files using system calls.

code:

```
#include <iostream>

#include <stdlib.h>

#include <fcntl.h>

#include <errno.h>

#include<unistd.h>

#include<sys/types.h>

#define BUFF_SIZE 1024

using namespace std;

int main(int argc, char* argv[])

{

int srcFD, destFD, nbread, nbwrite ;

char *buff[BUFF_SIZE];

if(argc != 3 || argv[1] == "--help")

{

cout<<"\nUsage: cpcmd source_file destination_file\n";
```

```
exit(EXIT_FAILURE);
}
srcFD = open(argv[1],O_RDONLY);
if(srcFD == -1)
{
cout<<"\nError opening file "<<argv[1]<<" errno = \n"<<errno;
exit(EXIT_FAILURE);
}
destFD = open(argv[2],O_WRONLY | O_CREAT | O_TRUNC, S_IRUSR |
S_IWUSR |S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH);
if(destFD == -1)
{
cout<<"\nError opening file "<<argv[2]<<" errno = \n"<<errno;
exit(EXIT_FAILURE);
}
while((nbread = read(srcFD,buff,BUFF_SIZE)) > 0)
{
if(write(destFD,buff,nbread) != nbread)
cout<<"\nError in writing data to \n"<<argv[2];
}
if(nbread == -1)
cout<<"\nError in reading data from \n"<<argv[1];
if(close(srcFD) == -1)
cout<<"\nError in closing file \n"<<argv[1];
if(close(destFD) == -1)
cout<<"\nError in closing file \n"<<argv[2];
exit(EXIT_SUCCESS);
}
```

OUTPUT:

~

```
Simanchal@DESKTOP-5P4GGV9 ~
$ g++ q5.cpp -o q5

Simanchal@DESKTOP-5P4GGV9 ~
$ ./q5.exe A.cpp B.txt

Simanchal@DESKTOP-5P4GGV9 ~
$
```

QUESTION 7:-

Write a program to implement FCFS scheduling algorithm.

code:

```
#include<iostream>

using namespace std;


int main()

{

        int n;

   cout<<"=================================================";

        cout<< "\n               FCFS          "<<endl;
```

```
cout<<"\n=================================================="<<endl;
cout<<"\nEnter number of process:";
cin>>n;

int burst_time[n];
for(int i=1; i<=n; i++)
{
	cout<<"Enter Burst time for P"<<i<<": ";
	cin>>burst_time[i];
}

int wait_time[n];

wait_time[1]=0;
for(int i=2; i<=n; i++)
{
	wait_time[i]=wait_time[i-1]+burst_time[i-1];
}

int turnaround_time[n];
for(int i=1; i<=n; i++)
{
	turnaround_time[i]=wait_time[i]+burst_time[i];
}

float total_wait_time=0, total_turnaround_time=0;
float avg_wait_time, avg_turnaround_time;
```

```
    for(int i=1;i<=n;i++)
    {
        total_wait_time+= wait_time[i];
        total_turnaround_time+= turnaround_time[i];
    }

    cout<<"          Burst Time    Waiting Time    Turnaround Time"<<endl;
    for(int i=1; i<=n; i++)
    {
        cout<<"P"<<i<<"       "<<burst_time[i]<<"                    "<<wait_time[i]<<"
"<<turnaround_time[i]<<endl;
    }

    avg_wait_time= total_wait_time/n;
    avg_turnaround_time= total_turnaround_time/n;

    cout<<"\nAverage wait time ="<<avg_wait_time<<endl;
    cout<<"\nAverage turnaround time ="<<avg_turnaround_time<<endl;
return 0;
}
```

OUTPUT:

~

```
Simanchal@DESKTOP-5P4GGV9 ~
$ g++ FCFS.cpp -o FCFS

Simanchal@DESKTOP-5P4GGV9 ~
$ ./FCFS
==================================================
                  FCFS

==================================================

Enter number of process:5
Enter Burst time for P1: 6
Enter Burst time for P2: 2
Enter Burst time for P3: 8
Enter Burst time for P4: 3
Enter Burst time for P5: 4
        Burst Time    Waiting Time    Turnaround Time
P1       6              0               6
P2       2              6               8
P3       8              8               16
P4       3              16              19
P5       4              19              23

Average wait time =9.8

Average turnaround time =14.4

Simanchal@DESKTOP-5P4GGV9 ~
$
```

QUESTION 8:-

Write a program to implement Round Robin scheduling algorithm.

CODE:

```
#include <iostream>

#include <vector>

using namespace std;

int main()

{



   cout<<"==================================================";
```

```
cout<< "\n                ROUND ROBIN          "<<endl;
cout<<"\n================================================="<<endl;
    int count,j,n,time,remain,flag=0,time_quantum,i=0;
    int wt=0,tat=0,at[20],bt[20],rt[20],gantt[20][2];
    cout<<"\nEnter no of Processes : ";
    cin>>n;
    cout<<"Enter Time Quantum : ";
    cin>>time_quantum;
    remain=n;
    cout<<"*****ENTER DETAILS*****"<<endl;
    for(count=0;count<n;count++)
    {
        cout<<"\nPId : "<<count+1;
        cout<<"\nArrival Time : ";
        cin>>at[count];
        cout<<"Burst Time : ";
        cin>>bt[count];
        rt[count]=bt[count];
    }
    cout<<"\nPId\tAt\tbt\n";
    for(count=0; count<n; count++)
    {
        cout<<count+1<<"\t"<<at[count]<<"\t"<<bt[count]<<"\n";
    }
    cout<<"\n\nPId\tTAT\tWT\n";
    for(time=0,count=0;remain!=0;)
    {
        if(rt[count]<=time_quantum && rt[count]>0)
```

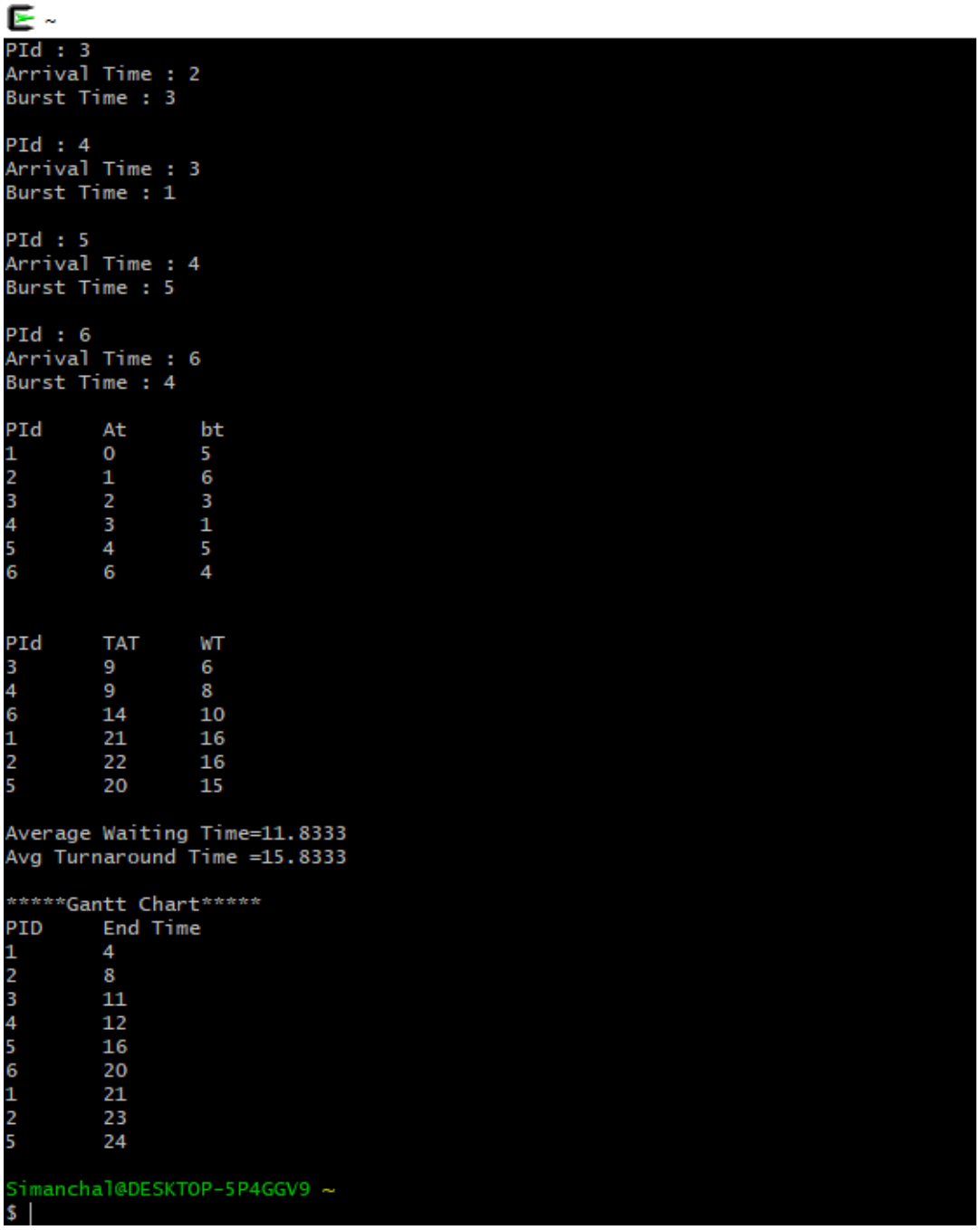
```
{
        time+=rt[count];
        rt[count]=0;
        gantt[i][0]= count;
        gantt[i][1]= time;
        i++;
        flag=1;
}
else if(rt[count]>0)
{
        rt[count]-=time_quantum;
        time+=time_quantum;
        gantt[i][0]= count;
        gantt[i][1]= time;
        i++;
}
if(rt[count]==0 && flag==1)
{
        remain--;
        cout<<count+1<<"\t"<<time-at[count]<<"\t"<<time-at[count]-bt[count]<<"\n";
        wt+=time-at[count]-bt[count];
        tat+=time-at[count];
        flag=0;
}
if(count==n-1)
        count=0;
else if(at[count+1]<=time)
        count++;
```

```
            else
                count=0;
            }
            cout<<"\nAverage Waiting Time="<<wt*1.0/n<<endl;
            cout<<"Avg Turnaround Time ="<<tat*1.0/n<<endl;
            cout<<endl<<"*****Gantt Chart*****"<<endl<<"PID\tEnd Time\t"<<endl;
            for(int k=0;k<i;k++)
            {
                cout<<gantt[k][0]+1<<"\t"<<gantt[k][1]<<"\t"<<endl;
            }
            return 0;
        }
```

OUTPUT:

~

```
Simanchal@DESKTOP-5P4GGV9 ~
$ g++ RoundRobin.cpp -o RoundRobin

Simanchal@DESKTOP-5P4GGV9 ~
$ ./RoundRobin
=================================================
                  ROUND ROBIN

=================================================

Enter no of Processes : 6
Enter Time Quantum : 4
*****ENTER DETAILS*****

PId : 1
Arrival Time : 0
Burst Time : 5

PId : 2
Arrival Time : 1
Burst Time : 6

PId : 3
Arrival Time : 2
Burst Time : 3

PId : 4
Arrival Time : 3
Burst Time : 1

PId : 5
Arrival Time : 4
Burst Time : 5

PId : 6
Arrival Time : 6
Burst Time : 4

PId     At      bt
1       0       5
2       1       6
3       2       3
4       3       1
5       4       5
6       6       4


PId     TAT     WT
3       9       6
```



QUESTION 9:-

Write a program to implement SJF scheduling algorithm.

CODE:

```
#include <iostream>

using namespace std;
```

```
int mat[10][6];


void swap(int* a, int* b)
{
   int temp = *a;
   *a = *b;
   *b = temp;
}


void arrangeArrival(int num, int mat[][6])
{
          for (int i = 0; i < num; i++)
          {
                    for (int j = 0; j < num - i - 1; j++)
                    {
          if (mat[j][1] > mat[j + 1][1])
                              {
             for (int k = 0; k < 5; k++)
                                        {
                swap(mat[j][k], mat[j + 1][k]);
             }
          }
       }
   }
}


void completionTime(int num, int mat[][6])
{
```

```
    int temp, val;
    mat[0][3] = mat[0][1] + mat[0][2];
    mat[0][5] = mat[0][3] - mat[0][1];
    mat[0][4] = mat[0][5] - mat[0][2];

    for (int i = 1; i < num; i++)
            {
        temp = mat[i - 1][3];
        int low = mat[i][2];
        for (int j = i; j < num; j++)
                    {
            if (temp >= mat[j][1] && low >= mat[j][2])
                                {
                low = mat[j][2];
                val = j;
            }
        }
        mat[val][3] = temp + mat[val][2];
        mat[val][5] = mat[val][3] - mat[val][1];
        mat[val][4] = mat[val][5] - mat[val][2];
        for (int k = 0; k < 6; k++) {
            swap(mat[val][k], mat[i][k]);
        }
            }
}

int main()
{
```

```
cout<<"=========================================================================="<<endl;
      cout<<"                    SJf                   "<<endl;

cout<<"=========================================================================="<<endl;

   int num, temp;

   cout << "Enter number of Process: ";
   cin >> num;

   cout << "...Enter the process ID...\n";
   for (int i = 0; i < num; i++)
         {
      cout << "...Process " << i + 1 << "...\n";
      cout << "Enter Process Id: ";
      cin >> mat[i][0];
      cout << "Enter Arrival Time: ";
      cin >> mat[i][1];
      cout << "Enter Burst Time: ";
      cin >> mat[i][2];
   }

   cout << "Before Arrange...\n";
   cout << "Process ID\tArrival Time\tBurst Time\n";
   for (int i = 0; i < num; i++)
         {
```

```
        cout << mat[i][0] << "\t\t" << mat[i][1] << "\t\t"
<< mat[i][2] << "\n";
    }

    arrangeArrival(num, mat);
    completionTime(num, mat);

    cout << "Final Result.."<<endl;
    cout << "Process ID\tArrival Time\tBurst Time\tWaiting "
          "Time\tTurnaround Time\n";
    for (int i = 0; i < num; i++)
          {
        cout << mat[i][0] << "\t\t" << mat[i][1] << "\t\t"
<< mat[i][2] << "\t\t" << mat[i][4] << "\t\t"
<< mat[i][5] << "\n";
    }

    return 0;
}
```

OUTPUT:

```
Simanchal@DESKTOP-5P4GGV9 ~
$ g++ sjf.cpp -o sjf

Simanchal@DESKTOP-5P4GGV9 ~
$ ./sjf
=====================================================================
                        SJf
=====================================================================
Enter number of Process: 3
...Enter the process ID...
...Process 1...
Enter Process Id: 1
Enter Arrival Time: 2
Enter Burst Time: 6
...Process 2...
Enter Process Id: 2
Enter Arrival Time: 5
Enter Burst Time: 2
...Process 3...
Enter Process Id: 3
Enter Arrival Time: 1
Enter Burst Time: 8
Before Arrange...
Process ID      Arrival Time    Burst Time
1               2               6
2               5               2
3               1               8
Final Result..
Process ID      Arrival Time    Burst Time      Waiting Time    Turnaround Time
3               1               8               0               8
2               5               2               4               6
1               2               6               9               15

Simanchal@DESKTOP-5P4GGV9 ~
$
```

QUESTION 10:-

Write a program to implement non-preemptive priority based scheduling algorithm.

CODE:

```
#include<iostream>

using namespace std;


int ct=0,i=0,wp[25]={0},comp[25];


void TurnAroundTime(int i,int at[],int tat[],int qt)
```

```
{
	tat[i] = qt-at[i];
	comp[i]=qt;
}

void WaitingTime(int tat[], int n, int bt[], int wt[])
{
	for(int i=0;i<n;i++)
	{
		wt[i]=tat[i]-bt[i];
	}
}

void waitingQueue(int dup_bt[],int n,int processes[],int at[],int tat[])
{
	for(int i=0;i<n;i++)
	{
		int j=0; j=ct;
		if(dup_bt[i]!=0)
		{
			ct=ct+dup_bt[i];
			dup_bt[i]=0;
			TurnAroundTime(i,at,tat,ct);
		}
	}
}

void ReadyQueue(int processes[],int bt[],int at[],int n,int pri[])
```

```
{
        int dup_bt[n],tat[n],wt[n];
        for(int i=0;i<n-1;i++)
        {
                for(int j=0;j<n-1;j++)
                {
                        if(pri[j]<pri[j+1])
                        {
                                int t=0;
                                t=at[j];
                                at[j]=at[j+1];
                                at[j+1]=t;
                                t=processes[j];
                                processes[j]=processes[j+1];
                                processes[j+1]=t;
                                t=bt[j];
                                bt[j]=bt[j+1];
                                bt[j+1]=t;
                                t=pri[j];
                                pri[j]=pri[j+1];
                                pri[j+1]=t;
                        }
                        else if(pri[j]==pri[j+1]&&at[j]>at[j+1])
                        {
                                int t=0;
                                t=at[j];
                                at[j]=at[j+1];
                                at[j+1]=t;
```

```
                        t=processes[j];
                        processes[j]=processes[j+1];
                        processes[j+1]=t;
                        t=bt[j];
                        bt[j]=bt[j+1];
                        bt[j+1]=t;
                        t=pri[j];
                        pri[j]=pri[j+1];
                        pri[j+1]=t;
                }
        }
    }
    cout<<"\nProcesses "<<" Arrival Time "<<" Burst Time "<<" Priority \n";
    for (int i=0;i<n;i++)
    {
        cout<<"    "<<processes[i]<<"\t\t"<<at[i]<<"\t      "<<bt[i]<<"\t       "<<pri[i]<<"\t
"<<endl;
    }
    for(int i=0;i<n;i++)
        dup_bt[i] = bt[i];
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n;j++)
        {
            if(dup_bt[j]!=0)
            {
                if(ct==at[j]||ct>at[j])
                {
```

```
                                        i=ct;
                                        int l=0;
                                        ct=ct+dup_bt[j];
                                        dup_bt[j]=0;
                                        TurnAroundTime(j,at,tat,ct);
                                        j=n;
                                    }
                                }
                            }
                        }
                        waitingQueue(dup_bt,n,processes,at,tat);
                        WaitingTime(tat,n,bt,wt);
                        cout<<"\nProcesses "<<" Arrival Time "<<" Burst Time "<<" Turn Around Time "<<" Compilation Time "<<" Waiting Time "<<"\n";
                        for(int i = 0 ; i < n ; i++)
                                cout<<"    "<<processes[i]<<"\t\t "<<at[i]<<"\t      "<<bt[i]<<"\t          "<<tat[i]<<"\t "<<comp[i]<<"\t            "<<wt[i]<<"\n";
}

int main()
{

cout<<"=============================================================================";
   cout<< "\n               NON PREEMPTIVE PRIORITY SCHEDULING          "<<endl;

cout<<"=============================================================================="<<endl;
```

```
        int n;
        cout<<"Enter no. of Processes: ";
        cin>>n;
        cout<<endl;
        int processes[n];
        int bt_time[n];
        int a_time[n];
        int priority[n];
        for(int i=0;i<n;i++)
        {
                processes[i]=i+1;
                cout<<"Enter Burst Time of P["<<i+1<<"]: ";
                cin>>bt_time[i];
                cout<<"Enter Arrival Time of P["<<i+1<<"]: ";
                cin>>a_time[i];
                cout<<"Enter Priority of P["<<i+1<<"]: ";
                cin>>priority[i];
                cout<<endl;
        }
        cout<<endl;
        cout<<"Processes "<<" Arrival Time "<<" Burst Time "<<" Priority \n";
        for(int i=0;i<n;i++)
        {
                cout<<"    "<<processes[i]<<"\t\t"<<a_time[i]<<"\t     "<<bt_time[i]<<"\t
"<<priority[i]<<"\t    "<<endl;
   }
   ReadyQueue(processes,bt_time,a_time,n,priority);
```

```
    return(0);
}
```

OUTPUT:

```
~
Simanchal@DESKTOP-5P4GGV9 ~
$ g++ nonpreemptive.cpp -o nonpreemptive

Simanchal@DESKTOP-5P4GGV9 ~
$ ./nonpreemptive
================================================================================
                  NON PREEMPTIVE PRIORITY SCHEDULING
================================================================================
Enter no. of Processes: 5

Enter Burst Time of P[1]: 4
Enter Arrival Time of P[1]: 0
Enter Priority of P[1]: 2

Enter Burst Time of P[2]: 3
Enter Arrival Time of P[2]: 1
Enter Priority of P[2]: 3

Enter Burst Time of P[3]: 1
Enter Arrival Time of P[3]: 2
Enter Priority of P[3]: 4

Enter Burst Time of P[4]: 5
Enter Arrival Time of P[4]: 3
Enter Priority of P[4]: 5

Enter Burst Time of P[5]: 2
Enter Arrival Time of P[5]: 4
Enter Priority of P[5]: 5


Processes  Arrival Time  Burst Time  Priority
    1           0            4         2
    2           1            3         3
    3           2            1         4
    4           3            5         5
    5           4            2         5

Processes  Arrival Time  Burst Time  Priority
    4           3            5         5
    5           4            2         5
    3           2            1         4
    2           1            3         3
    1           0            4         2
```

```
Processes  Arrival Time  Burst Time  Turn Around Time  Compilation Time  Waiting
 Time
    4           3            5            6                 9              1
    5           4            2            7                 11             5
    3           2            1            10                12             9
    2           1            3            14                15             11
    1           0            4            4                 4              0

Simanchal@DESKTOP-5P4GGV9 ~
$
```

QUESTION 11:-

Write a program to implement preemptive priority based scheduling algorithm.

CODE:

```
#include <iostream>
using namespace std;
#include<iostream>
using namespace std;
class Process
{
        int pid;
        int at;
        int bt;
        int rt;
        int priority;
        int n;
        int completed_flag;
        Process *proc;
    public:
        void input();
        void sort_at();
        void prem_priority();
};

void Process :: input()
{
    cout<<"\nEnter no of Processes : ";
    cin>>n;
```

```
        cout<<endl;
        proc=new Process[n];
        cout<<"ENTER DETAILS"<<endl;
        cout<<endl;
        for(int i=0;i<n;i++)
        {
                cout<<"Enter the PId for Process "<<i+1<<" : ";
                cin>>proc[i].pid;
                cout<<"Enter the Process Arrival Time for Process " <<i+1<<" : ";
                cin>>proc[i].at;
                cout<<"Enter the Process Burst Time for Process "<<i+1<<" : ";
                cin>>proc[i].bt;
                cout<<"Enter the Priority for Process "<<i+1<<" : ";
                cin>>proc[i].priority;
                cout<<endl;
                proc[i].completed_flag=0;
                proc[i].rt=proc[i].bt;
                cout<<endl;
        }
        cout<<"\nPId\tAt\tBt\tPriority\n";
        for(int i=0; i<n; i++)
        {
                cout<<proc[i].pid<<"\t"<<proc[i].at<<"\t"<<proc[i].bt<<"\t"<<proc[i].priority<<"\n";
        }
}
void Process :: sort_at()
{
        for (int i=0; i<n; i++)
```

```
        {
                for(int j=0; j<n-i-1; j++)
                {
                        if(proc[j].at>proc[j+1].at)
                        {
                                Process temp=proc[j];
                                proc[j]=proc[j+1];
                                proc[j+1]=temp;
                        }
                }
        }
}

void Process :: prem_priority()
{
        int ct;
        sort_at();
        cout<<"Execution sequence as follow:\nSelected process info :\n";
        cout<<"\nPid\tAT\tBT\tPriority\tST\tCT\n";
        for(int count=0,ct=proc[0].at; count<n;ct++ )
        {
                int selected_process=-1;
                for(int i=0; i<n; i++)
                {
                        if(proc[i].at<=ct && proc[i].completed_flag!=1)
                        {
                                if(selected_process==-1)
                                selected_process=i;
```

```
                    else if(proc[selected_process].priority>proc[i].priority)
                    selected_process=i;
                }
                else if (proc[i].at>ct)
                    break;
            }
            if(selected_process==-1)
            {
                continue;
            }

cout<<proc[selected_process].pid<<"\t"<<proc[selected_process].at<<"\t"<<proc[selected_process].bt<<"\t"<<proc[selected_process].priority<<"\t\t"<<ct<<"\t"<<ct+1<<"\n";
            proc[selected_process].rt--;
            if(proc[selected_process].rt==0)
            {
                proc[selected_process].completed_flag=1;
                count++;
            }
        }
}

int main()
{
   int n;
   cout <<
"================================================================================";
   cout << "\n              PREEMPTIVE PRIORITY SCHEDULING          " << endl;
```

```
   cout <<
"================================================================================"
<< endl;
        Process p;
p.input();
p.prem_priority();
return 0;
}
```

~

```
Simanchal@DESKTOP-5P4GGV9 ~
$ g++ preemptive.cpp -o preemptive

Simanchal@DESKTOP-5P4GGV9 ~
$ ./preemptive
================================================================================
                  PREEMPTIVE PRIORITY SCHEDULING
================================================================================

Enter no of Processes : 5

ENTER DETAILS

Enter the PId for Process 1 : 1
Enter the Process Arrival Time for Process 1 : 0
Enter the Process Burst Time for Process 1 : 4
Enter the Priority for Process 1 : 2


Enter the PId for Process 2 : 2
Enter the Process Arrival Time for Process 2 : 1
Enter the Process Burst Time for Process 2 : 3
Enter the Priority for Process 2 : 3


Enter the PId for Process 3 : 3
Enter the Process Arrival Time for Process 3 : 2
Enter the Process Burst Time for Process 3 : 1
Enter the Priority for Process 3 : 4


Enter the PId for Process 4 : 4
Enter the Process Arrival Time for Process 4 : 3
Enter the Process Burst Time for Process 4 : 5
Enter the Priority for Process 4 : 5


Enter the PId for Process 5 : 5
Enter the Process Arrival Time for Process 5 : 4
Enter the Process Burst Time for Process 5 : 2
Enter the Priority for Process 5 : 5



PId     At      Bt      Priority
1       0       4       2
2       1       3       3
3       2       1       4
4       3       5       5
```

```
PId     At      Bt      Priority
1       0       4       2
2       1       3       3
3       2       1       4
4       3       5       5
5       4       2       5
Execution sequence as follow:
Selected process info :

Pid     AT      BT      Priority        ST      CT
1       0       4       2               0       1
1       0       4       2               1       2
1       0       4       2               2       3
1       0       4       2               3       4
2       1       3       3               4       5
2       1       3       3               5       6
2       1       3       3               6       7
3       2       1       4               7       8
4       3       5       5               8       9
4       3       5       5               9       10
4       3       5       5               10      11
4       3       5       5               11      12
4       3       5       5               12      13
5       4       2       5               13      14
5       4       2       5               14      15

Simanchal@DESKTOP-5P4GGV9 ~
$
```

QUESTION 1 2:-

Write a program to implement SRTF scheduling algorithm.

CODE:

```
#include<iostream>
#include<stdlib.h>
#include<stdio.h>
using namespace std;
int n;
class Process
{
        int n,pid,at,bt,tat,st,wt,ct,rt;
        int completed_flag;
        Process *proc;
    public:
        void input();
        void sort_at();
        void srtf();
};

void Process :: input()
{
    cout<<"Enter no of Processes :"<<" " <<endl;
    cin>>n;
    cout<<"\n";
```

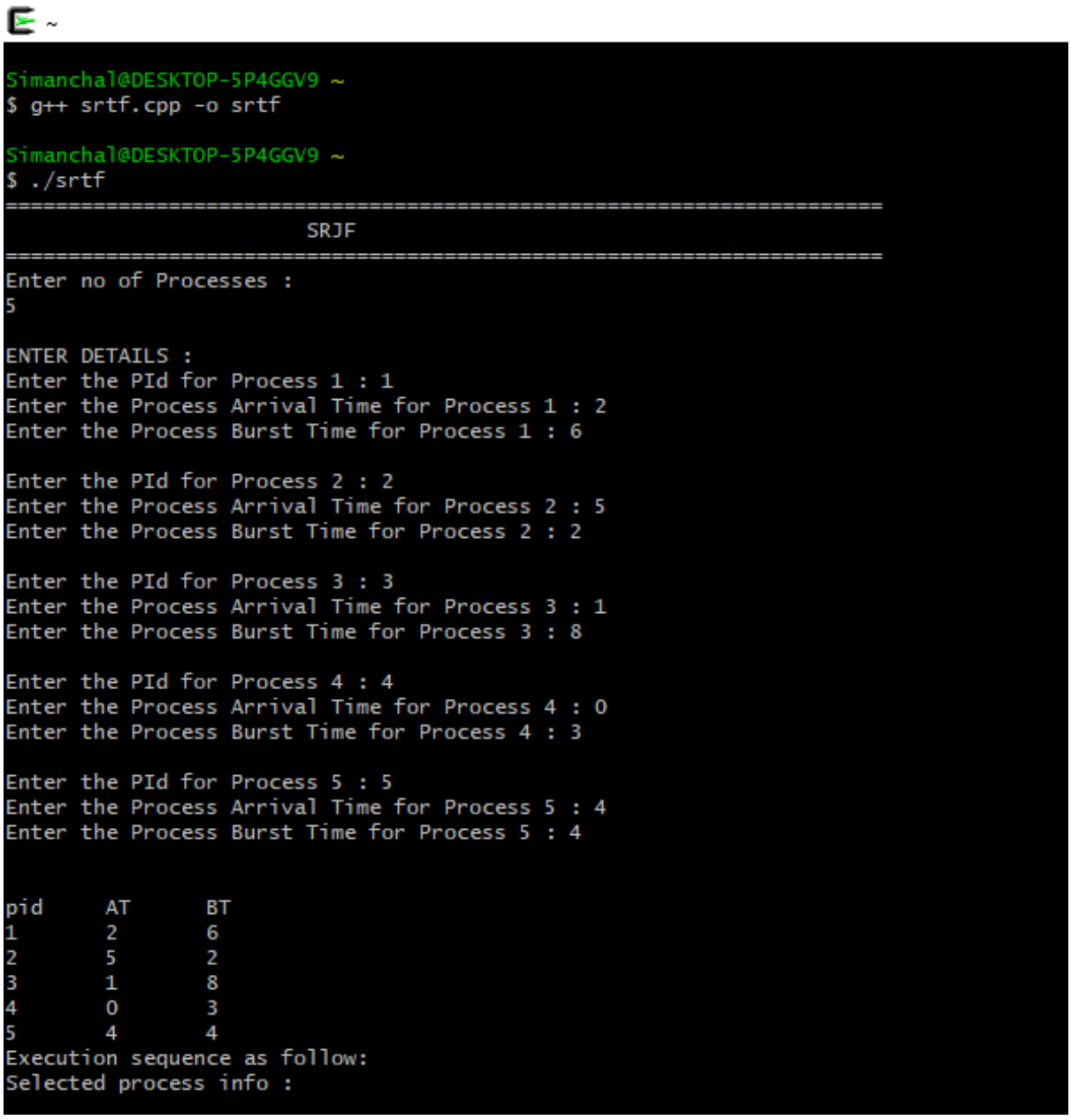
```
	proc=new Process[n];
	cout<<"ENTER DETAILS :" <<endl;
	for(int i=0; i<n; i++)
	{
		cout<<"Enter the PId for Process "<<i+1<<" : ";
		cin>>proc[i].pid;
		cout<<"Enter the Process Arrival Time for Process "<<i+1<<" : ";
		cin>>proc[i].at;
		cout<<"Enter the Process Burst Time for Process "<<i+1<<" : ";
		cin>>proc[i].bt;
		cout<<endl;
		proc[i].completed_flag=0;
		proc[i].rt=proc[i].bt;
	}
	cout<<"\npid\tAT\tBT\n";
	for(int i=0; i<n; i++)
	{
		cout<<proc[i].pid<<"\t"<<proc[i].at<<"\t"<<proc[i].bt<<"\n";
	}
}

void Process :: sort_at()
{
	for (int i=0; i<n; i++)
	{
		for(int j=0; j<n-i-1; j++)
		{
			if(proc[j].at>proc[j+1].at)
```

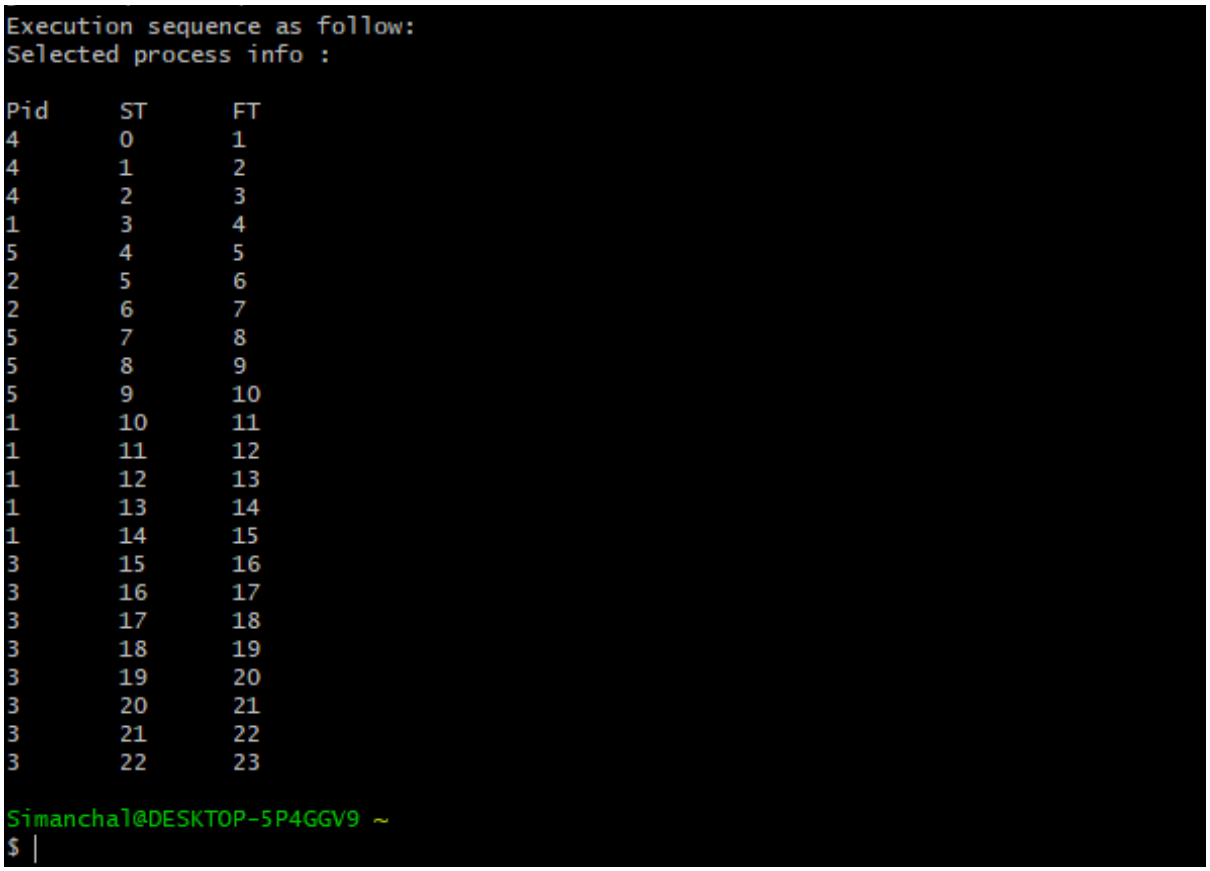
```
                {
                        Process temp=proc[j];
                        proc[j]=proc[j+1];
                        proc[j+1]=temp;
                }
        }
    }
}

void Process :: srtf()
{
    int ct;
    sort_at();
    cout<<"Execution sequence as follow:\nSelected process info :\n";
    cout<<"\nPid\tST\tFT\n";
    for(int count=0, ct= proc[0].at; count<n;ct++ )
    {
        int selected_process=-1;
        for(int i=0; i<n; i++)
        {
            if(proc[i].at<=ct && proc[i].completed_flag!=1)
            {
                if(selected_process==-1)
                selected_process=i;
                else if(proc[selected_process].rt>proc[i].rt)
                selected_process=i;
            }
            else if (proc[i].at>ct)
```

```
                break;
            }
            if(selected_process==-1)
            {
                ct++;
                continue;
            }
            cout<<proc[selected_process].pid<<"\t"<<ct<<"\t"<<ct+1<<"\n";
            proc[selected_process].rt--;
            if(proc[selected_process].rt==0)
            {
                proc[selected_process].completed_flag=1;
                count++;
            }
        }
}

int main()
{

cout<<"=========================================================================="<<endl;
   cout<<"                  SRTF                  "<<endl;

cout<<"=========================================================================="<<endl;

        Process p;
        p.input();
        p.srtf();
```

```
    return 0;
}
```

OUTPUT:





QUESTION 13:-

Write a program to calculate sum of n numbers using thread library.

CODE:

```
#include<pthread.h>
#include<iostream>
using namespace std;
int sum;
void* runner(void* param);
int main(int argc,char *argv[])
{
        pthread_t tid;
        pthread_attr_t attr;
```

```
        if(argc!=2)
        {
                cout<<"\nUsage :a.out<integer value>\n";
                return -1;
        }
        if(atoi(argv[1])<0)
        {
                cout<<"\n%d must be >=0\n"<<atoi((const char*)(argv[1]))<<endl;
                return -1;
        }
        pthread_attr_init(&attr);  //get the default attributes
        pthread_create(&tid,&attr,runner,argv[1]);  //create the thread:
        pthread_join(tid,NULL);  //parent waits for the child thread to finish
        cout<<"\nSUM is: "<<sum<<endl;//output the value of shared data "sum"
        return 0;
}//child thread will begin execution here:


void* runner(void* param)
{
        int i,upper=atoi((const char*)param);
        sum=0;
        for(i=1;i<=upper;i++)
        sum+=i;
        pthread_exit(0);
}
```

OUTPUT:

~

```
Simanchal@DESKTOP-5P4GGV9 ~
$ g++ sum_n.cpp -o sum_n

Simanchal@DESKTOP-5P4GGV9 ~
$ ./sum_n

Usage :a.out<integer value>

Simanchal@DESKTOP-5P4GGV9 ~
$ ./sum_n 7

SUM is: 28

Simanchal@DESKTOP-5P4GGV9 ~
$
```

QUESTION 14:-

Write a program to implement first-fit, best-fit and worst-fit allocation strategies.

CODE:

```
#include <iostream>
using namespace std;
class Fit
{
        int p,h;
        int *process,*hole;
    public:
        Fit(int,int);
        void input();
        void firstFit();
        void bestFit();
        void worstFit();
```

```
};
Fit :: Fit(int n1,int n2)
{
	p=n1;
	h=n2;
	process=new int[p];
	hole=new int[h];
}
void Fit :: input()
{
	cout<<"Enter the process size :\n";
	for(int i=0;i<p;i++)
	{
		cout<<"Process[" << i+1 << "] : ";
		cin>>process[i];
	}
	cout<<"\nEnter the hole size :\n";
	for(int i=0;i<h;i++)
	{
		cout<<"Hole[" << i+1 << "] : ";
		cin>>hole[i];
	}
}
void Fit :: firstFit()
{
	int flag=1;
	for(int i=0;i<p;i++)
	{
```

```
		for(int j=0;j<h;j++)
		{
			if(process[i]<=hole[j])
			{
				cout<<"Process size : "<<process[i]<<" -----> Hole Size : "<< hole[j] <<endl;
				int flag=0;
				hole[j]-=process[i];
				break;
			}
		}
	}
}
void Fit :: bestFit()
{
	int loc,temp,min;
	for(int i=0;i<h-1;i++)
	{
		min=hole[i];
		loc=i;
		for(int j=i+1;j<h;j++)
		{
			if(min>hole[j])
			{
				min=hole[j];
				loc=j;
			}
		}
		temp=hole[i];
```

```
                hole[i]=hole[loc];
                hole[loc]=temp;
        }
        for(int i=0;i<p;i++)
        {

for(int j=0;j<h;j++)
                {
                        if(process[i]<=hole[j])
                        {
                                cout<<"Process size : "<<process[i]<<" -----> Hole Size : "<<hole[j]<<endl;
                                hole[j]-=process[i];
                                break;
                        }
                }
        }
}

void Fit :: worstFit()
{
        int flag=1;
        if(p<=h)
        {
                for(int i=0;i<p;i++)
                {
                        for(int j=i+1;j<h;j++)
                        {
                                if(hole[i]<hole[j])
```

```
                        {
                                int temp=hole[i];
                                hole[i]=hole[j];
                                hole[j]=temp;
                        }
                }
        }
        for(int i=0;i<p;i++)
        {
                for(int j=0;j<h;j++)
                {
                        if(process[i]<=hole[j])
                        {
                                cout<<"Process size : "<<process[i]<<" -----> Hole Size :
"<<hole[j]<<endl;
                                flag=0;
                                hole[j]=0;
                                break;
                        }
                        else
                        flag=1;
                }
                if(flag==1)
                cout<<"Process size : "<<process[i]<<" -----> Not Allocated"<<endl;
        }
    }
}
```

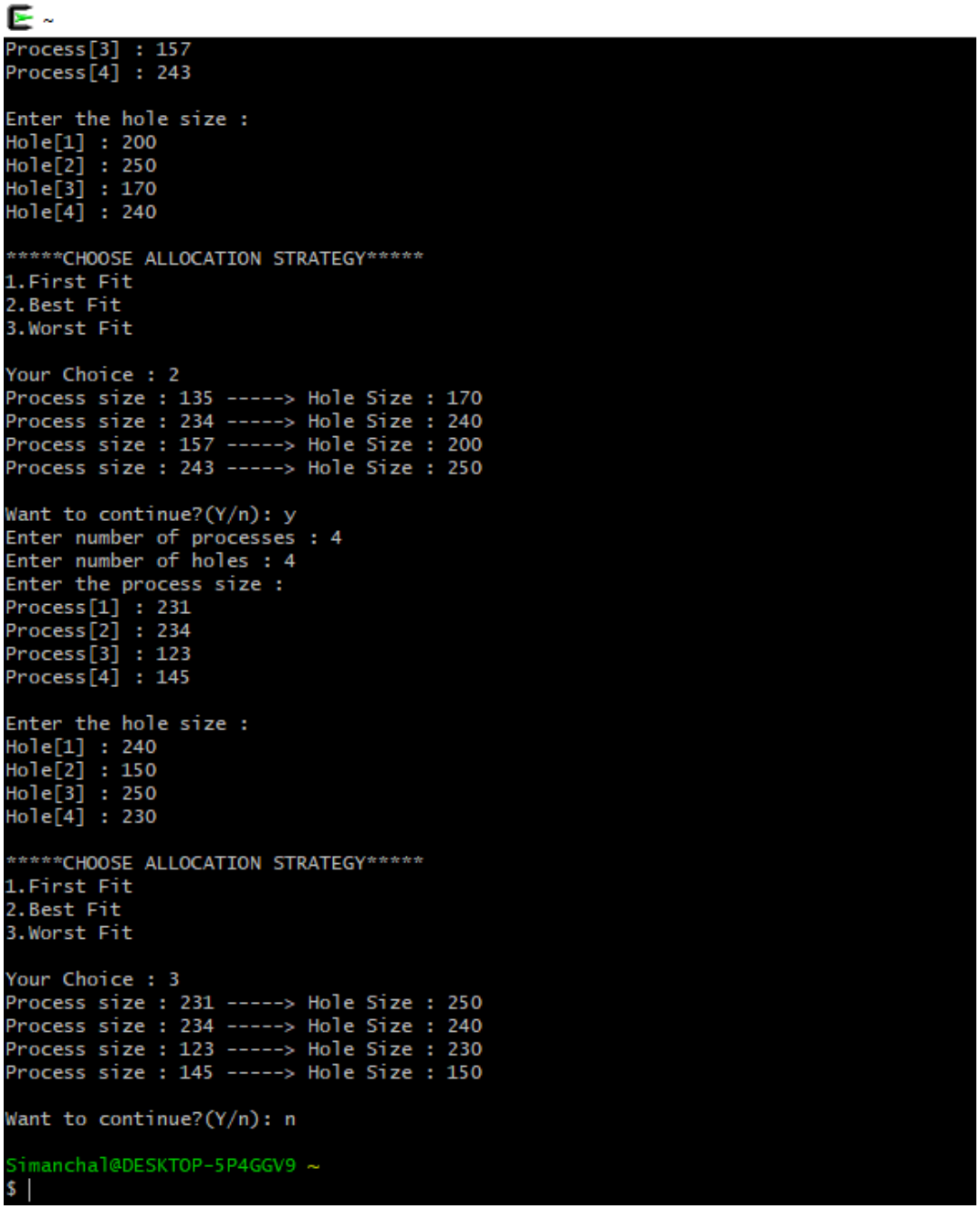
```
int main()
{
char ans='y';
int p,h,choice;
do
{
	cout<<"Enter number of processes : ";
	cin>>p;
	cout<<"Enter number of holes : ";
	cin>>h;
	Fit f(p,h);
	f.input();
	cout<<"\n*****CHOOSE ALLOCATION STRATEGY*****\n";
	cout<<"1.First Fit\n";
	cout<<"2.Best Fit\n";
	cout<<"3.Worst Fit\n";
	cout<<"\nYour Choice : ";
	cin>>choice;
	switch(choice)
	{
		case 1:
			f.firstFit();
			break;
		case 2:
			f.bestFit();
			break;
		case 3:
			f.worstFit();
```

```
                    break;
            default:
                    cout<<"Make a valid choice\n";
                    break;
        }

        cout<<"\nWant to continue?(Y/n): ";
        cin>>ans;
        }while(ans=='Y' || ans=='y');
return 0;
}
```

OUTPUT:

~

```
Simanchal@DESKTOP-5P4GGV9 ~
$ g++ FIT.cpp -o FIT

Simanchal@DESKTOP-5P4GGV9 ~
$ ./FIT
Enter number of processes : 4
Enter number of holes : 4
Enter the process size :
Process[1] : 123
Process[2] : 234
Process[3] : 213
Process[4] : 150

Enter the hole size :
Hole[1] : 230
Hole[2] : 150
Hole[3] : 200
Hole[4] : 250

*****CHOOSE ALLOCATION STRATEGY*****
1.First Fit
2.Best Fit
3.Worst Fit

Your Choice : 1
Process size : 123 -----> Hole Size : 230
Process size : 234 -----> Hole Size : 250
Process size : 150 -----> Hole Size : 150

Want to continue?(Y/n): y
Enter number of processes : 4
Enter number of holes : 4
Enter the process size :
Process[1] : 135
Process[2] : 234
Process[3] : 157
Process[4] : 243

Enter the hole size :
Hole[1] : 200
Hole[2] : 250
Hole[3] : 170
Hole[4] : 240

*****CHOOSE ALLOCATION STRATEGY*****
1.First Fit
2.Best Fit
3.Worst Fit
```



15-CODE:

```
#include<iostream>
#include<unistd.h>
#include<sys/types.h>
#include<sys/wait.h>
using namespace std;
```

```
int main()
{
int ID1=fork();
int ID2=fork();
if(ID1<0)
cout<<"\n Unsuccessful \n";
if(ID1==0)
cout <<" Im child pid = " << getpid() <<" return ID1: "<< ID1<<"\n";
else
cout <<" Im parent pid = " << getpid() <<" return ID1: "<< ID1<<"\n";
if(ID2<0)
cout<<"\n Unsuccessful \n";
if(ID2==0)
cout <<" Im child pid = " << getpid() <<" return ID2: "<< ID2<<"\n";
else
cout <<" Im parent pid = " << getpid() <<" return ID2: "<< ID2<<"\n";
return 0;
}
```

OUTPUT: