**Programme Name - B.Sc. Computer Science (Hons.)**

**Semester - III**

**Paper Title - Operating System Practicals**

**Submitted by - Khusi Singh**

**Examination Roll No - 19044570049**

**College RollNo - CSC/19/88**

**College Name - Mata Sundri College for Women, University of Delhi**

**College Address - Mata Sundri Lane, New Delhi 110002**

Q1. Write a program (using fork() and/or exec() commands) where parent and child execute :

(a) same program, same code.

*Code:*

//Same Program,Same Code(Both process run concurrently)

#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<<"\nUnsuccessful \n";

}

else

{

cout <<"PId : " <<getpid()<<"\n";

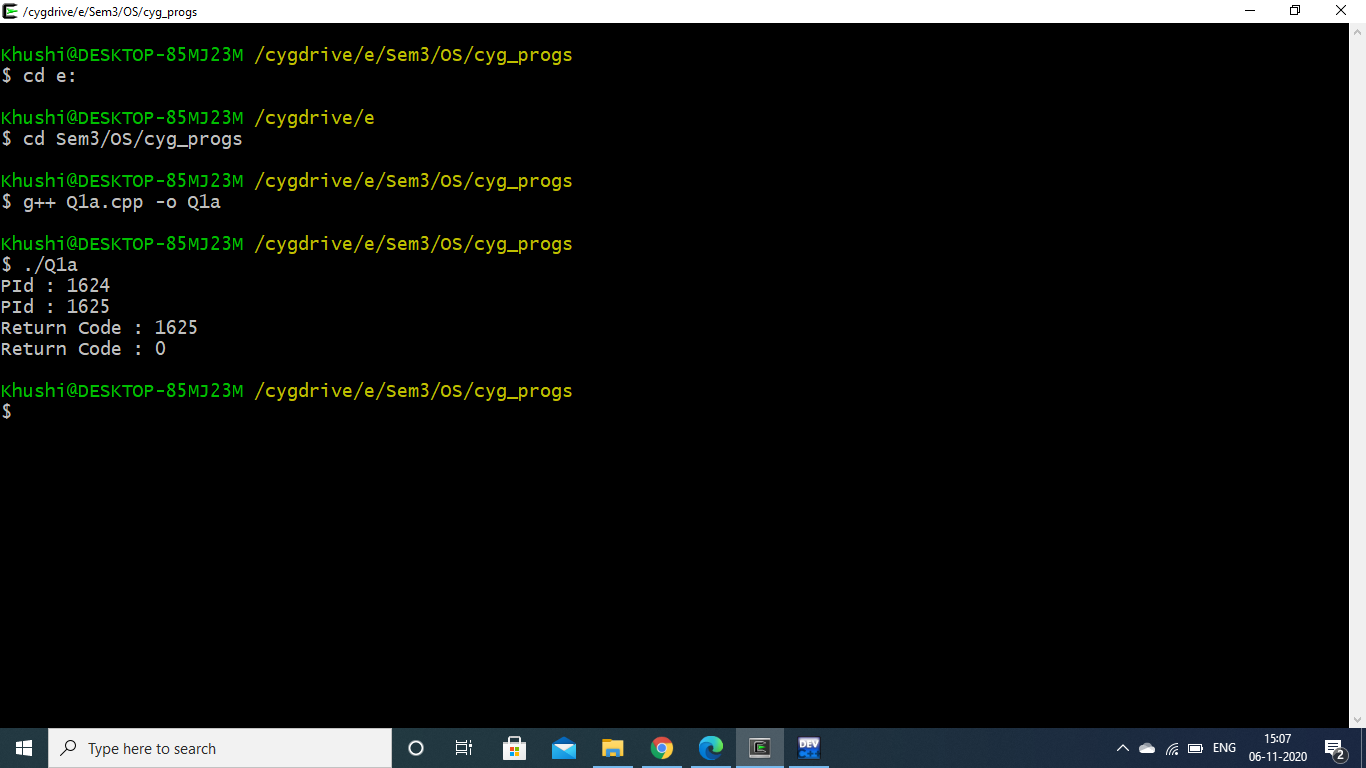
cout<<"Return Code : "<<code<<"\n";

}

return 0;

}

*Output:*

**

(b) same program, different code.

*Code:*

//Same Program,Different Code(Both process run concurrently)

#include<iostream>

#include<unistd.h>

using namespace std;

int main()

{

int pid=fork();

if(pid<0)

{

cout<<"Unsuccessful\n";

return -1;

}

else if(pid==0)

{

cout<<"I am Child Process \n";

cout<<"PId : "<<getpid()<<"\n";

}

else

{

sleep(.5);

cout<<"I am Parent Process \n";

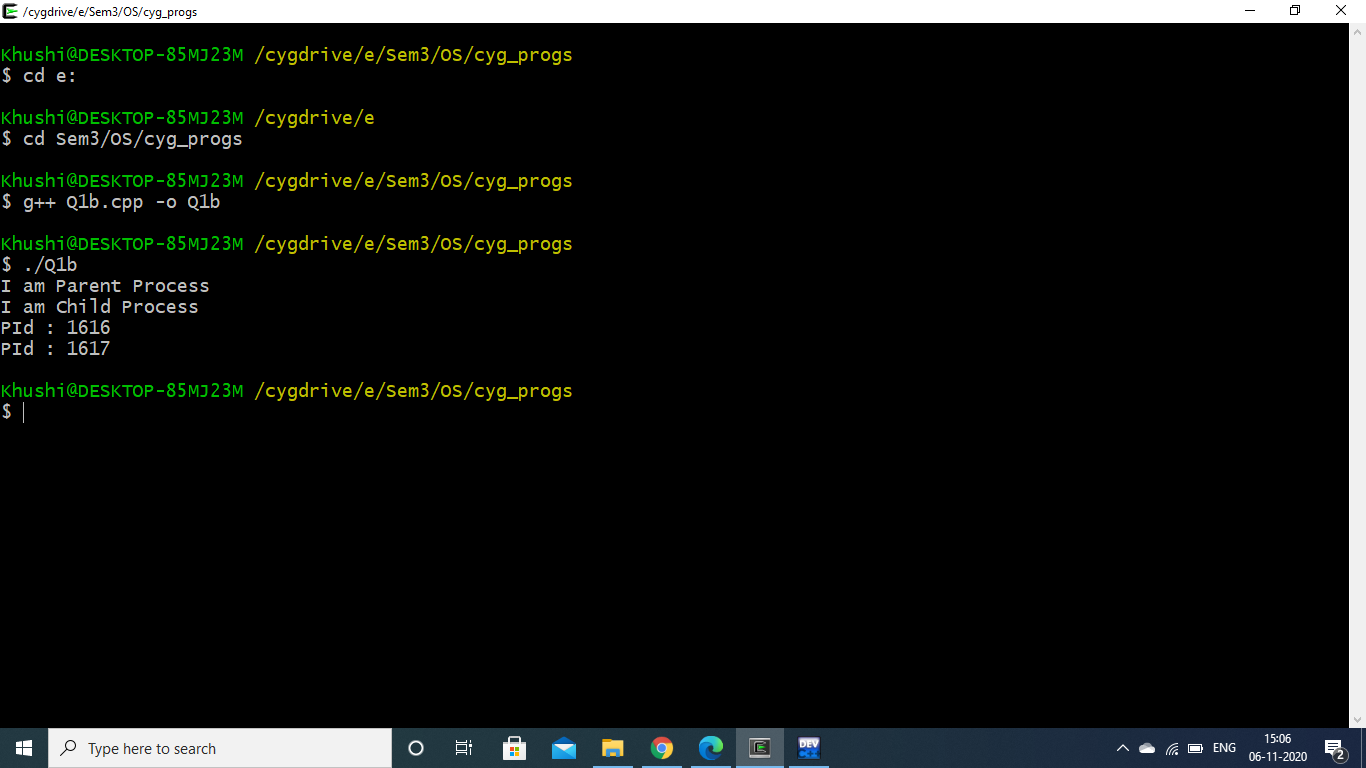
cout<<"PId : "<<getpid()<<"\n";

}

return 0;

}

*Output:*

**

(c) Before terminating, the parent waits for the child to finish its task, both for above mentioned cases (a) and (b).

*Code: (a)*

//Same Program,Same Code(Parent process waits for child to finish)

#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\n";

}

else

{

x=wait(&status);

cout <<"PId : "<<getpid()<<"\n";

cout<<"Return code : "<<code<<"\n";

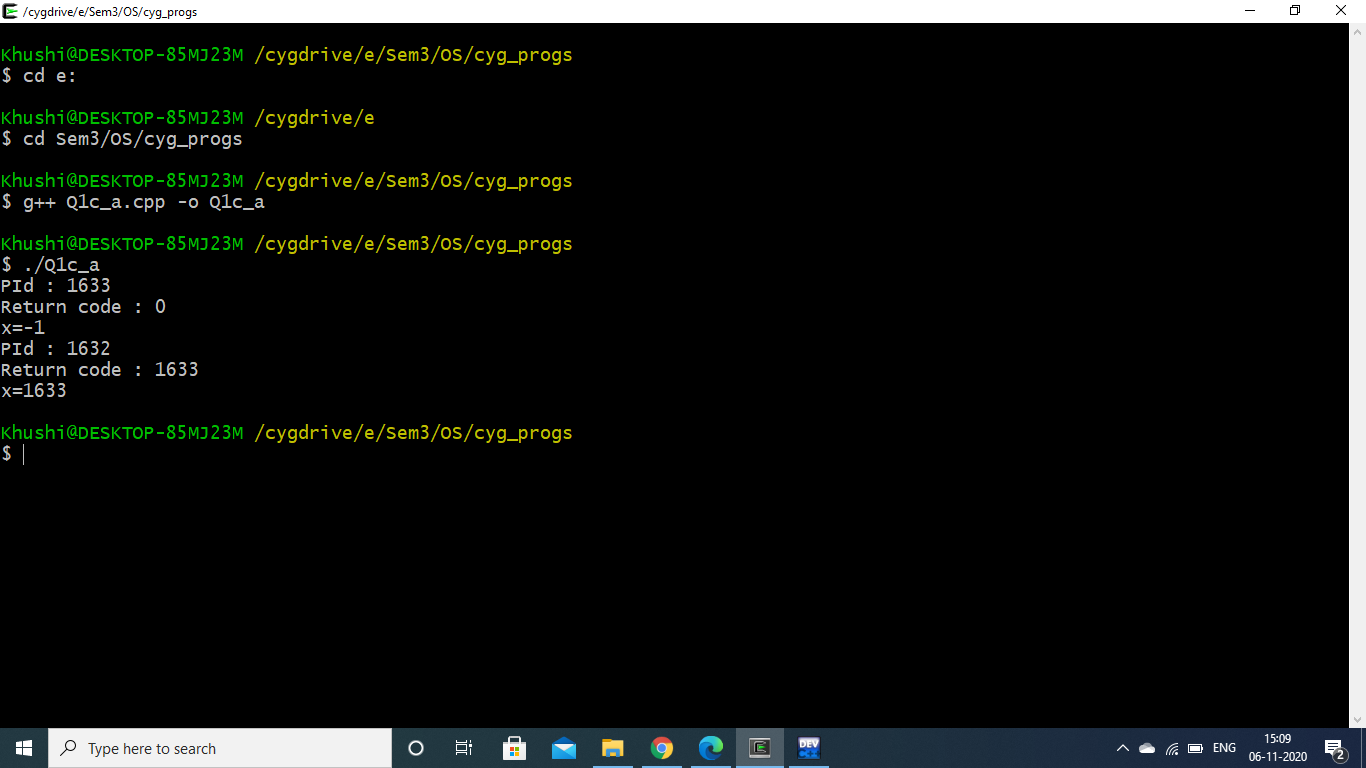
cout<<"x="<<x<<"\n";

}

return 0;

}

*Output:*



*Code: (b)*

//Same Program,Different Code(Parent process waits for child to finish)

#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<<"Unsuccessful\n";

return -1;

}

else if(pid==0)

{

cout<<"I am Child Process \n";

cout<<"PId : "<<getpid()<<"\n";

}

else

{

wait(&status);

cout<<"I am Parent Process \n";

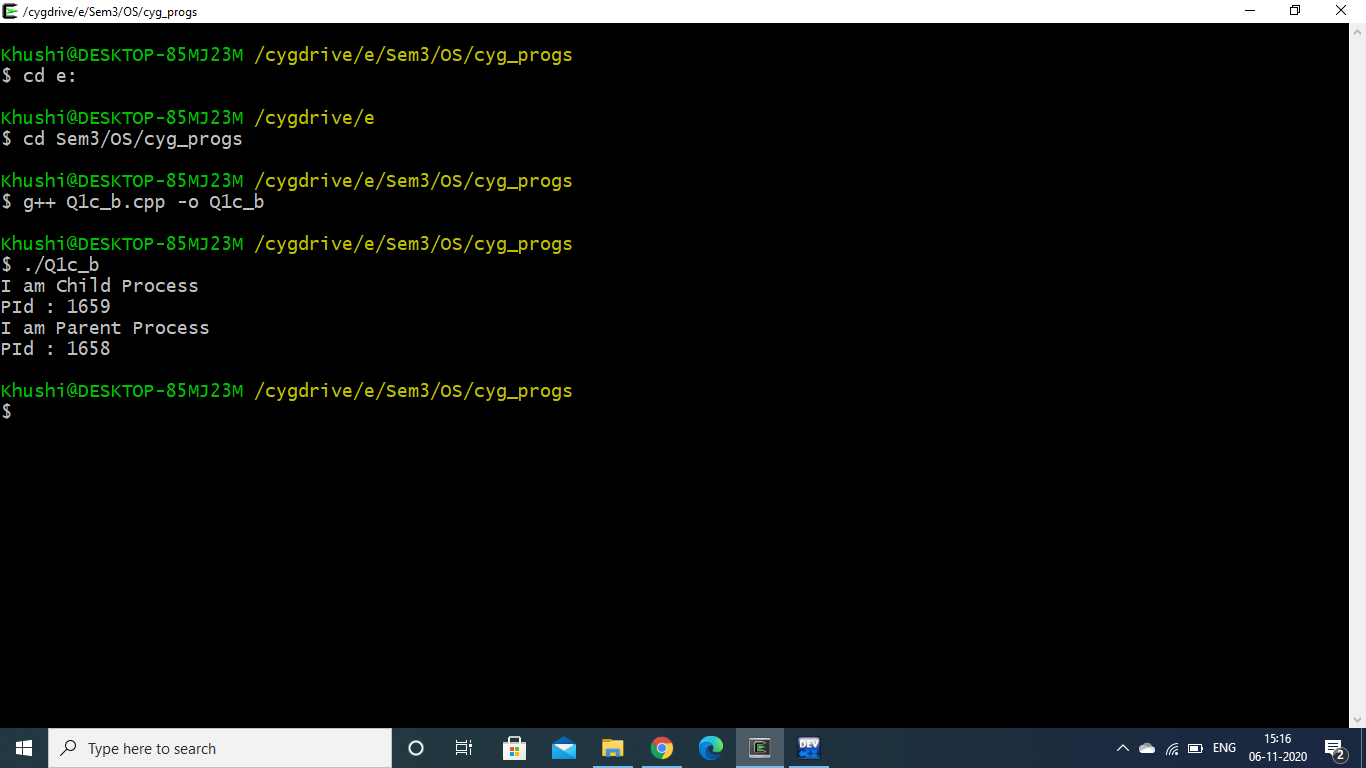
cout<<"PId : "<<getpid()<<"\n";

}

return 0;

}

*Output:*



Q2. Write a program to show how multiple fork() system calls work.

*Code:*

//Write a program to show how multiple fork() system calls work.

#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<<"\nUnsuccessful \n";

else if(ID1==0)

cout <<" I am child pid = " << getpid() <<" return ID1: "<< ID1<<"\n";

else

cout <<" I am parent pid = " << getpid() <<" return ID1: "<< ID1<<"\n";

if(ID2<0)

cout<<"\nUnsuccessful \n";

else if(ID2==0)

cout <<" I am child pid = " << getpid() <<" return ID2: "<< ID2<<"\n";

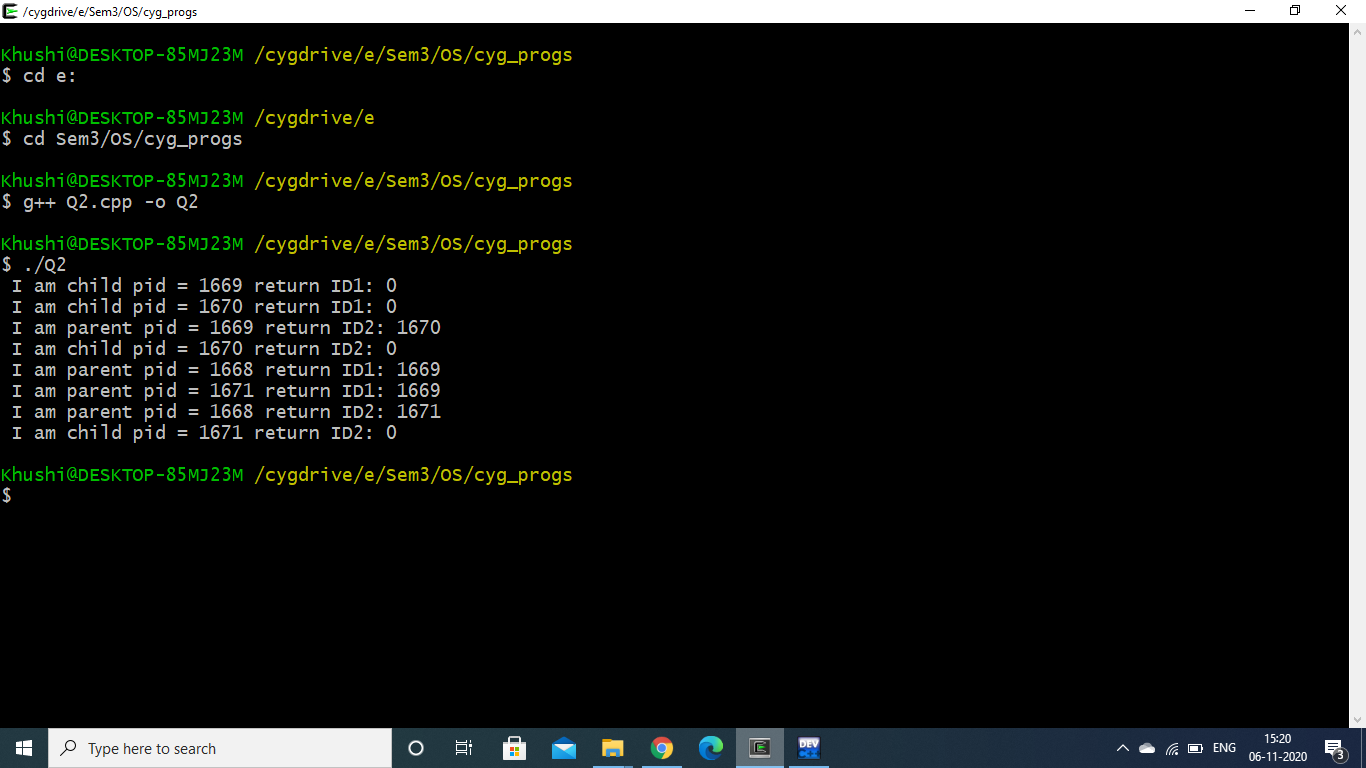
else

cout <<" I am parent pid = " << getpid() <<" return ID2: "<< ID2<<"\n";

return 0;

}

*Output:*



Q3. Write a program to report behaviour of Linux kernel including kernel version, CPU type and model. (CPU information)

*Code:*

#include<iostream>

using namespace std;

int main()

{

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

system("uname -s");

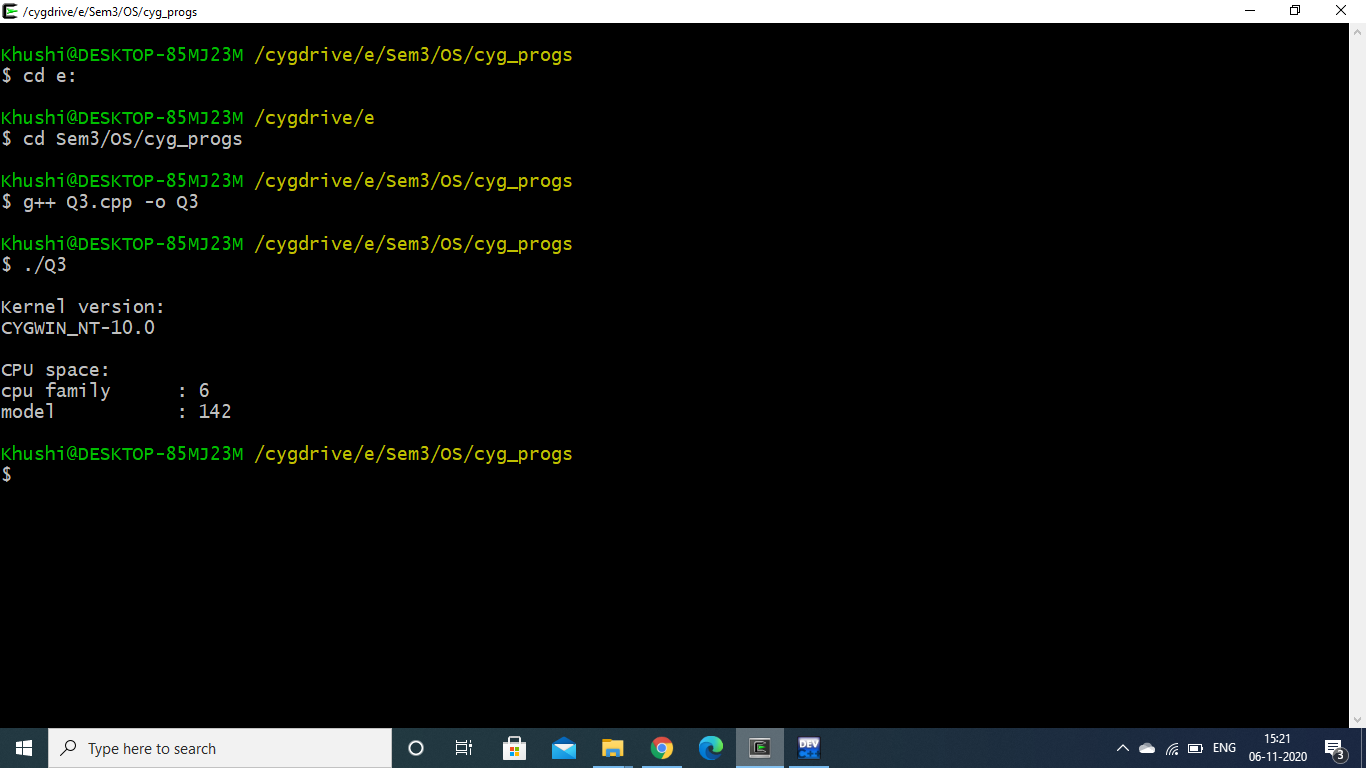
cout<<"\nCPU space: \n";

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

return 0;

}

*Output:*



Q4. Write a program to report behaviour of Linux kernel including information on configured 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");

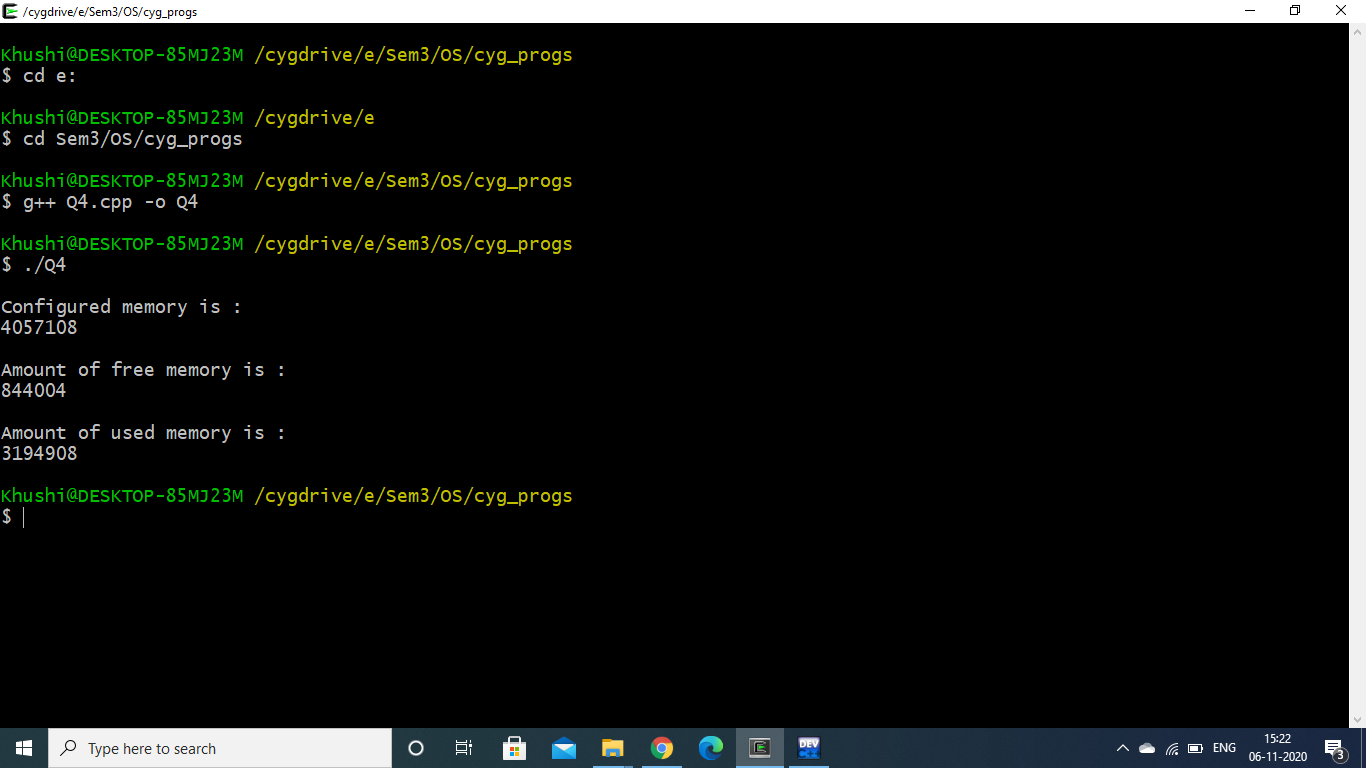
cout<<"\nAmount of used memory is :\n";

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

return 0;

}

*Output:*



Q5. Write a program to print file details including owner access permissions, file access

time, where file name is given as command line argument.

*Code:*

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

#include<iostream>

#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" : "-");

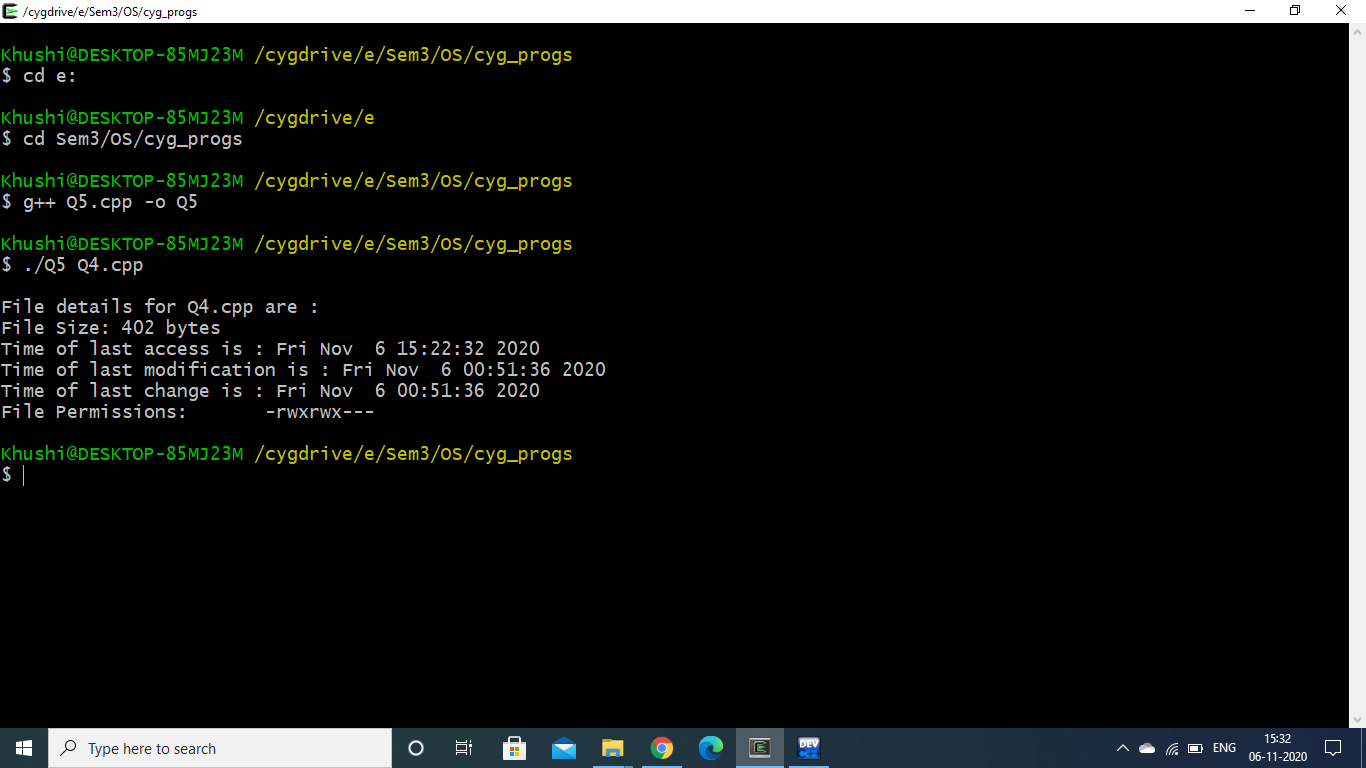
cout<<((fileStat.st\_mode & S\_IXOTH) ? "x" : "-");

cout<<endl;

return 0;

}

*Output:*

**

Q6. Write a program to copy files using system calls.

*Code:*

//Q6. Write a program to copy files using system calls.

#include <iostream>

#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];

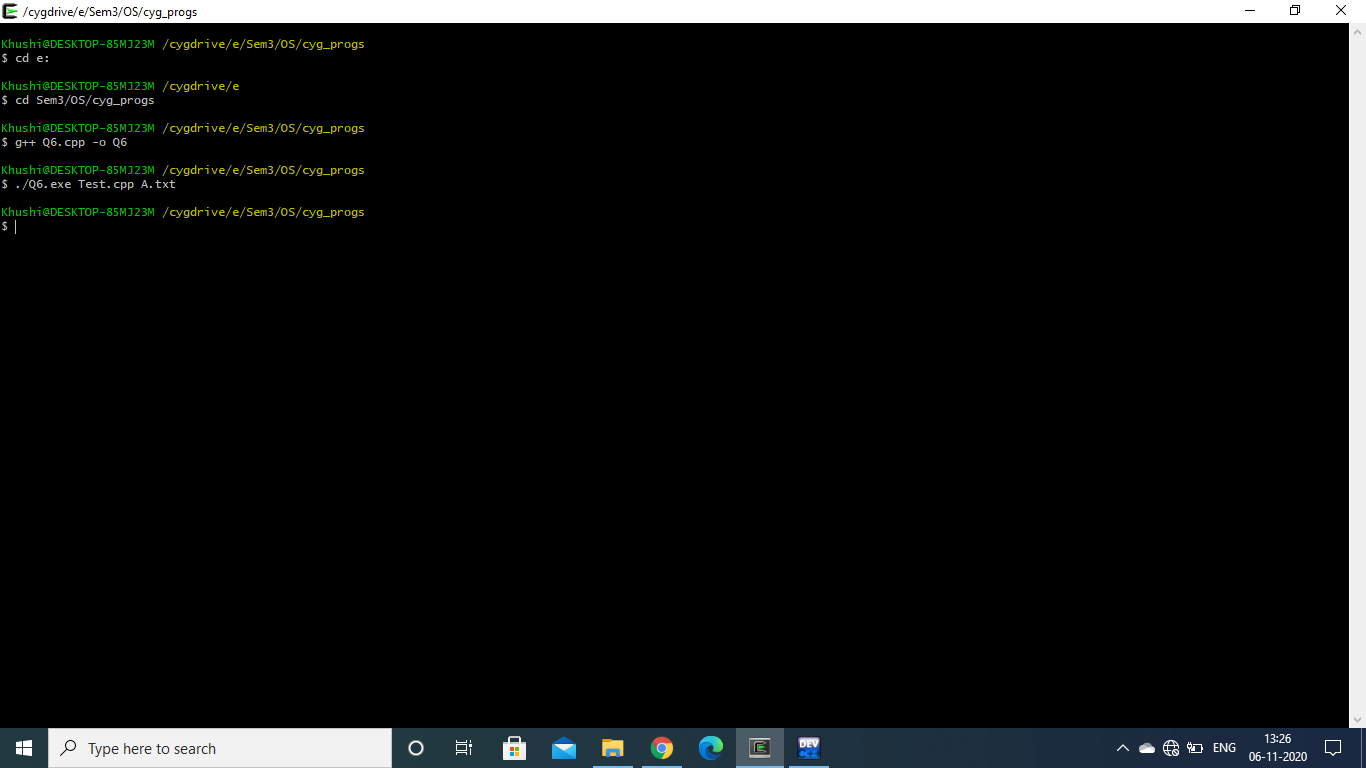
if(close(destFD)==-1)

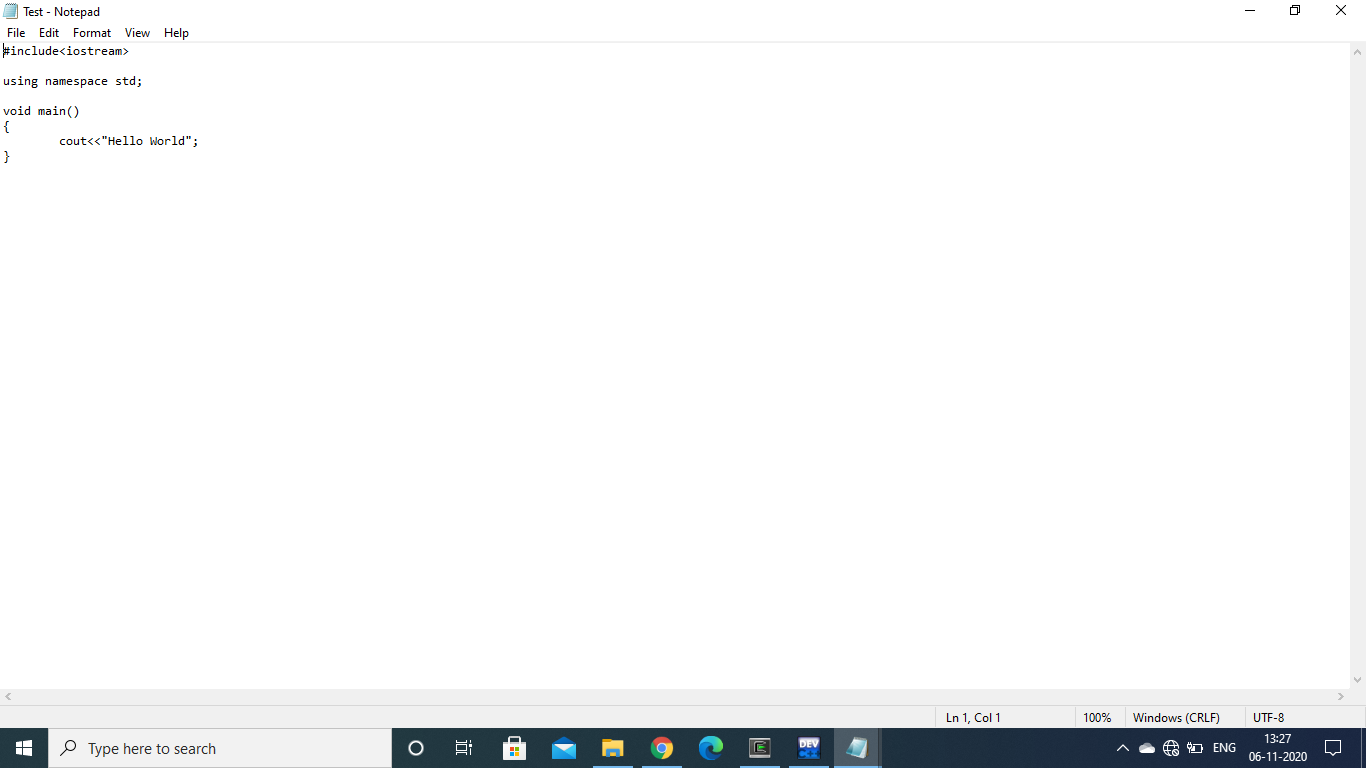
cout<<"\nError in closing file \n"<<argv[2];

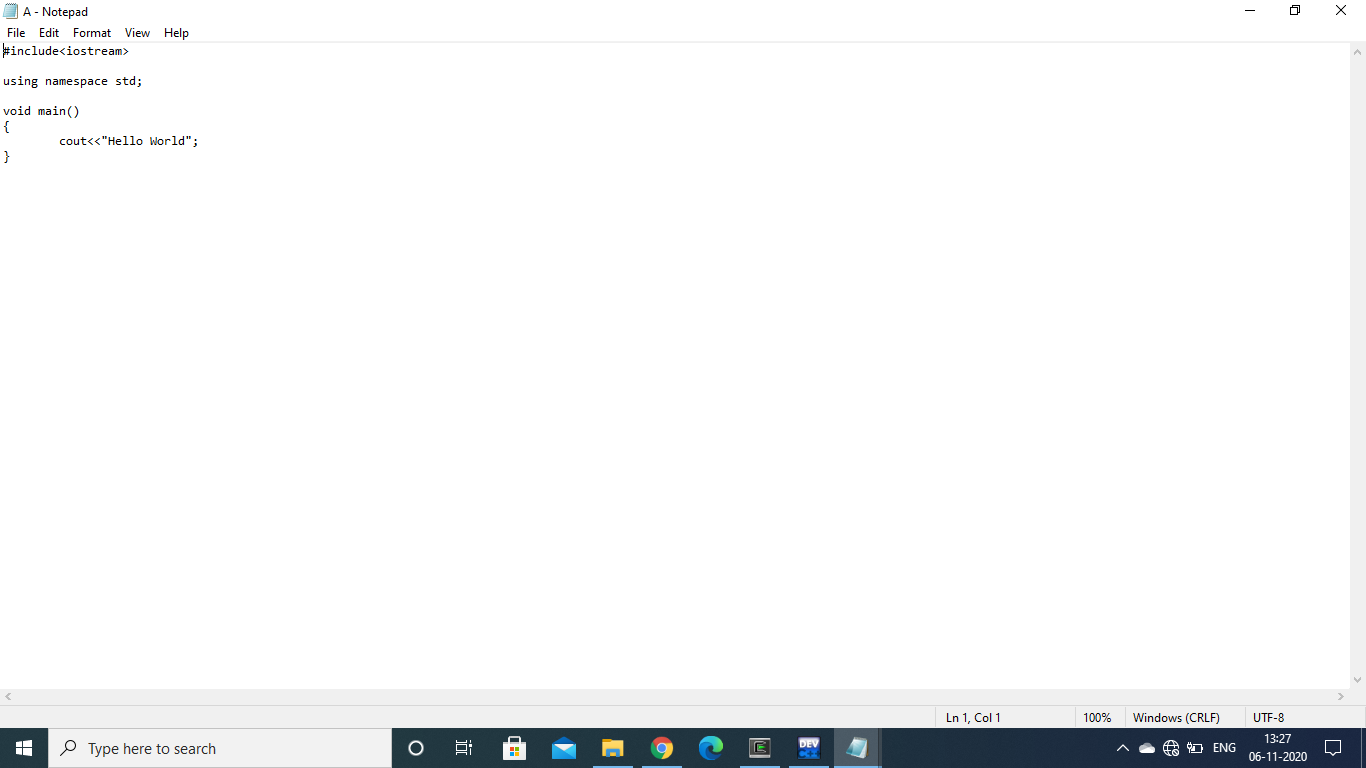
exit(EXIT\_SUCCESS);

}

*Output:*

**

**

**

Q7. Write a program to implement FCFS scheduling algorithm.

*Code:*

//First Come First Serve basis Scheduling Algorithm.

#include<iostream>

using namespace std;

class Process

{

int n;

int pid;

int at;

int bt;

Process \*proc;

public:

void input();

void sort\_at();

void fcfs();

};

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<<"-----Process P"<<i+1<<"-----"<<endl;

cout<<"PId : ";

cin>>proc[i].pid;

cout<<"Arrival Time : ";

cin>>proc[i].at;

cout<<"Burst Time : ";

cin>>proc[i].bt;

cout<<endl;

}

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 :: fcfs()

{

sort\_at();

int t,ct,wt,tat;

t=proc[0].at;

cout<<"\nExecution sequence as follow:\n";

cout<<"\nPId\tAT\tBT\tST\tCT\tWT\tTAT\n";

for(int i=0;i<n;i++)

{

ct=t+proc[i].bt;

wt=t-proc[i].at;

tat=ct-proc[i].at;

cout<<proc[i].pid<<"\t"<<proc[i].at<<"\t"<<proc[i].bt<<"\t"<<t<<"\t"<<ct<<"\t"<<wt<<"\t"<<tat<<endl;

t=ct;

if(t<proc[i].at)

{

t=proc[i+1].at;

}

}

}

int main()

{

Process p;

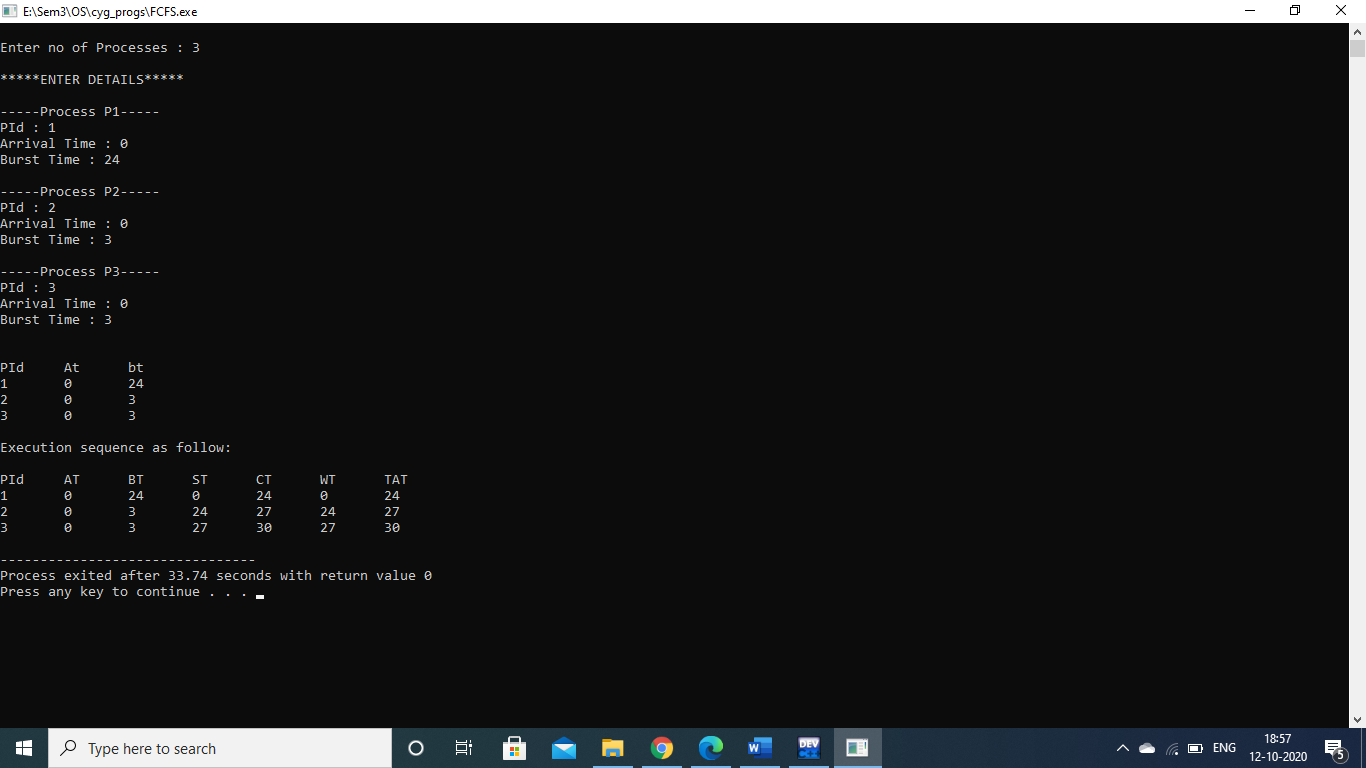
p.input();

p.fcfs();

return 0;

}

*Output:*



Q8. Write a program to implement Round Robin scheduling algorithm.

*Code:*

//Round Robin Scheduling Algorithm

#include<iostream>

using namespace std;

int main()

{

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++)

{

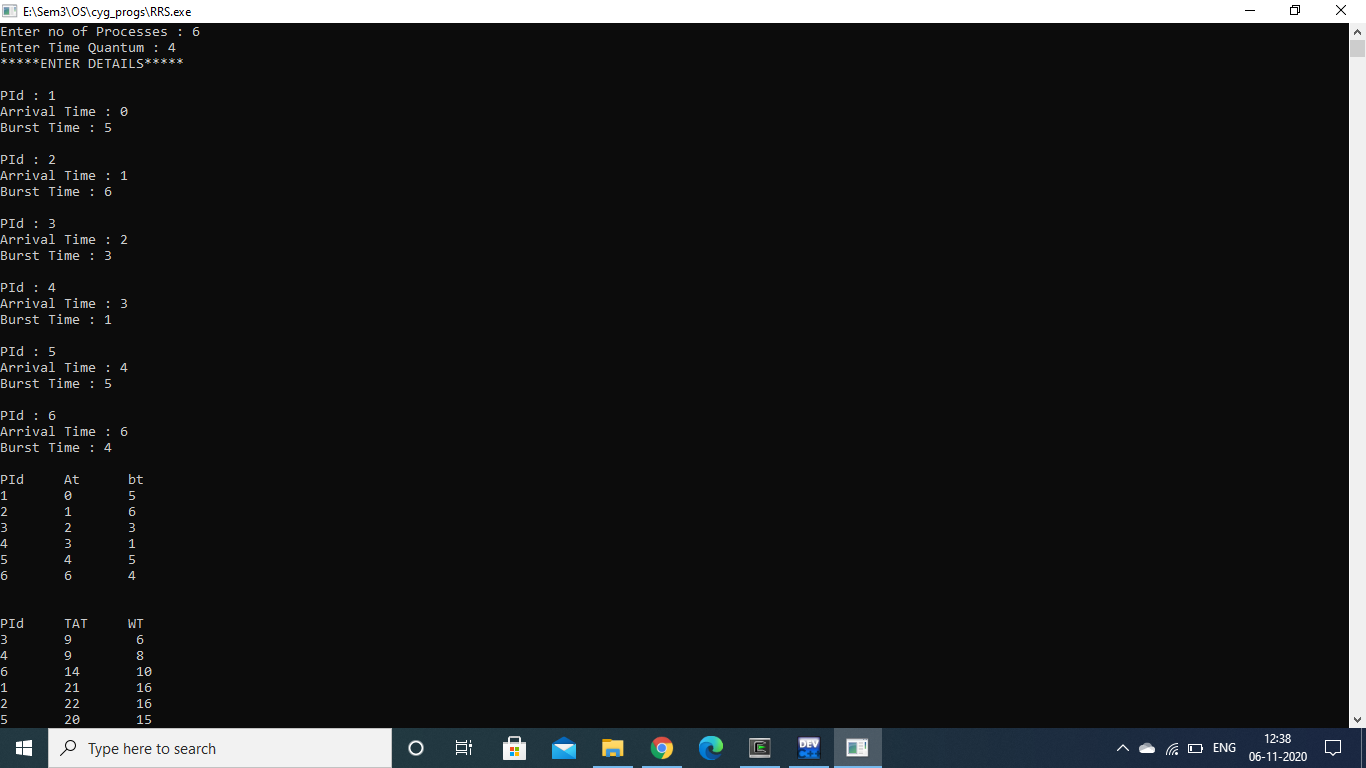
cout<<gantt[k][0]+1<<"\t"<<gantt[k][1]<<"\t"<<endl;

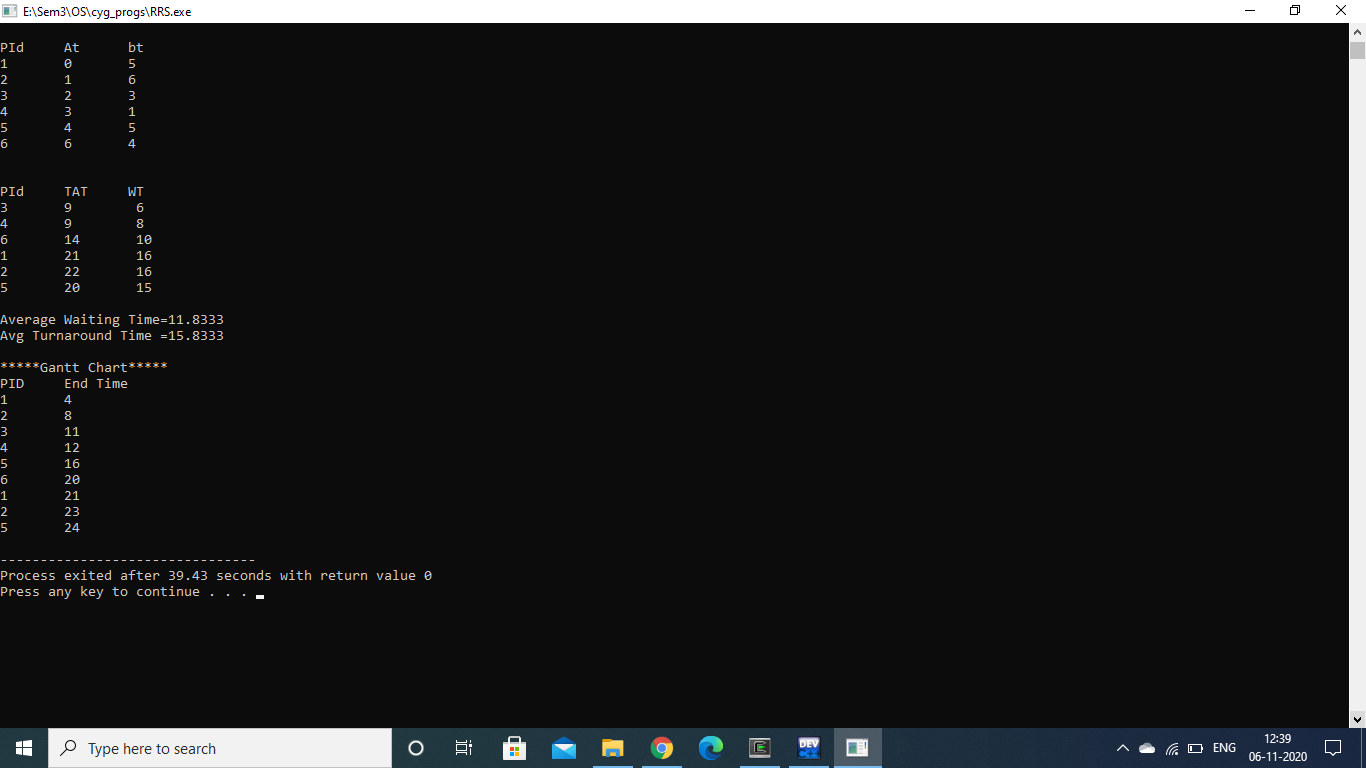
}

return 0;

}

*Output:*





Q9. Write a program to implement the SJF scheduling algorithm.

*Code:*

//SJF

#include<iostream>

using namespace std;

class Process

{

int n;

int pid;

int at;

int bt;

int completed\_flag;

Process \*proc;

public:

void input();

void sort\_at();

void sjf();

};

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<<"-----Process P"<<i+1<<"-----"<<endl;

cout<<"PId : ";

cin>>proc[i].pid;

cout<<"Arrival Time : ";

cin>>proc[i].at;

cout<<"Burst Time : ";

cin>>proc[i].bt;

cout<<endl;

proc[i].completed\_flag=0;

}

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 :: sjf()

{

int ct;

sort\_at();

ct=proc[0].at;

cout<<"Execution sequence as follow:\nSelected process info :\n";

cout<<"\nPId\tAT\tBT\tST\tCT\tWT\tTAT\n";

for(int count=0; count<n; )

{

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].bt>proc[i].bt)

selected\_process=i;

}

else if (proc[i].at>ct)

break;

}

if(selected\_process==-1)

{

ct++;

continue;

}

cout<<proc[selected\_process].pid<<"\t"<<proc[selected\_process].at<<"\t"<<proc[selected\_process].bt<<"\t"<<ct<<"\t"<<ct+proc[selected\_process].bt<<"\t"<<ct-proc[selected\_process].at<<"\t"<<(ct+proc[selected\_process].bt)-proc[selected\_process].at<<"\n";

proc[selected\_process].completed\_flag=1;

ct=ct+proc[selected\_process].bt;

count++;

}

}

int main()

{

Process p;

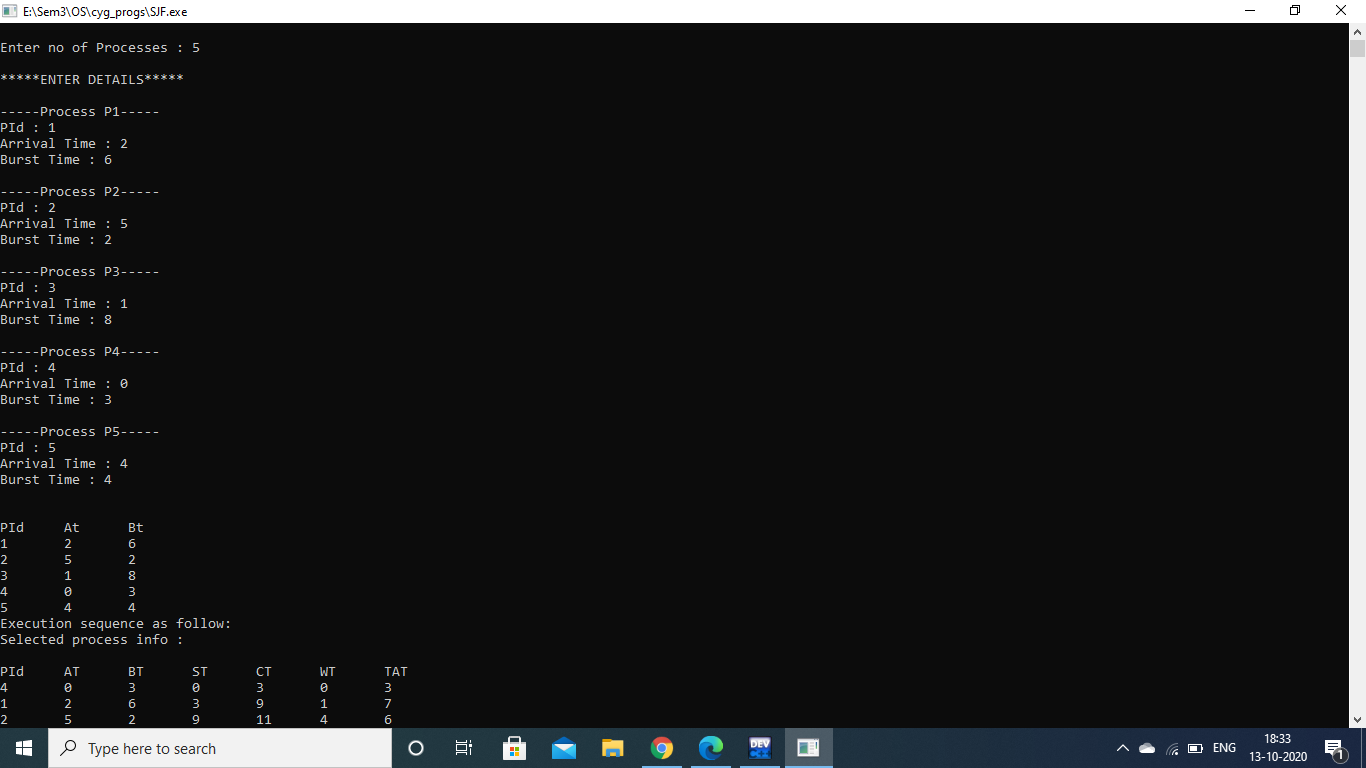
p.input();

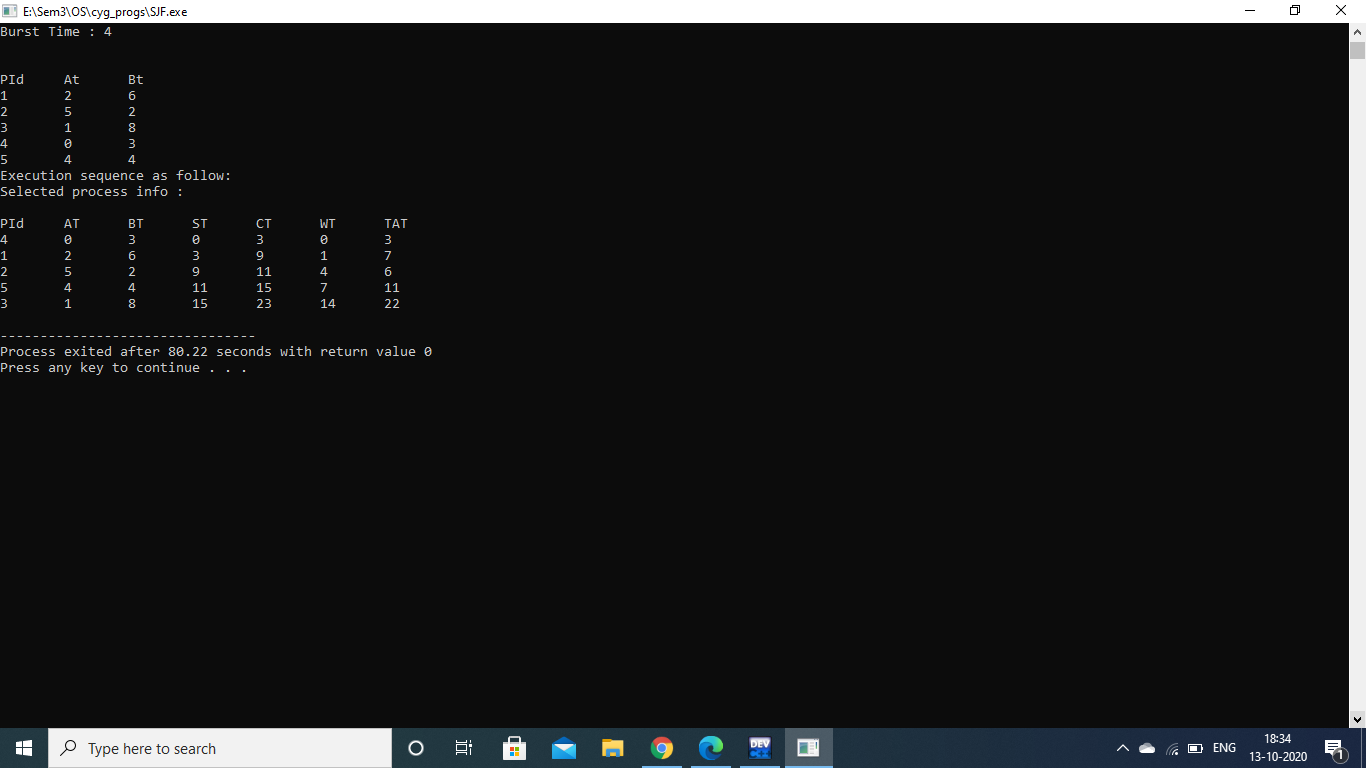
p.sjf();

return 0;

}

*Output:*





Q10. Write a program to implement a non-preemptive priority based scheduling algorithm.

*Code:*

//Non-preemptive priority based scheduling algorithm.

#include<iostream>

using namespace std;

class Process

{

int pid;

int at; //arrival time

int bt; //brust time

int priority;

int n;

int completed\_flag;

Process \*proc;

public:

void input();

void sort\_at();

void nonprem\_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<<"-----Process P"<<i+1<<"-----"<<endl;

cout<<"PId : ";

cin>>proc[i].pid;

cout<<"Arrival Time : ";

cin>>proc[i].at;

cout<<"Burst Time : ";

cin>>proc[i].bt;

cout<<"Priority : ";

cin>>proc[i].priority;

proc[i].completed\_flag=0;

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 :: nonprem\_priority()

{

int ct;

sort\_at();

ct=proc[0].at;

cout<<"Execution sequence as follow:\nSelected process info :\n";

cout<<endl;

cout<<"\nPId\tAT\tBT\tPriority\tST\tCT\tWT\tTAT\n";

for(int count=0;count<n; )

{

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)

{

ct++;

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+proc[selected\_process].bt<<"\t"<<ct-proc[selected\_process].at<<"\t"<<proc[selected\_process].bt+(ct-proc[selected\_process].at)<<"\n";

proc[selected\_process].completed\_flag=1;

ct=ct+proc[selected\_process].bt;

count++;

}

}

int main()

{

Process p;

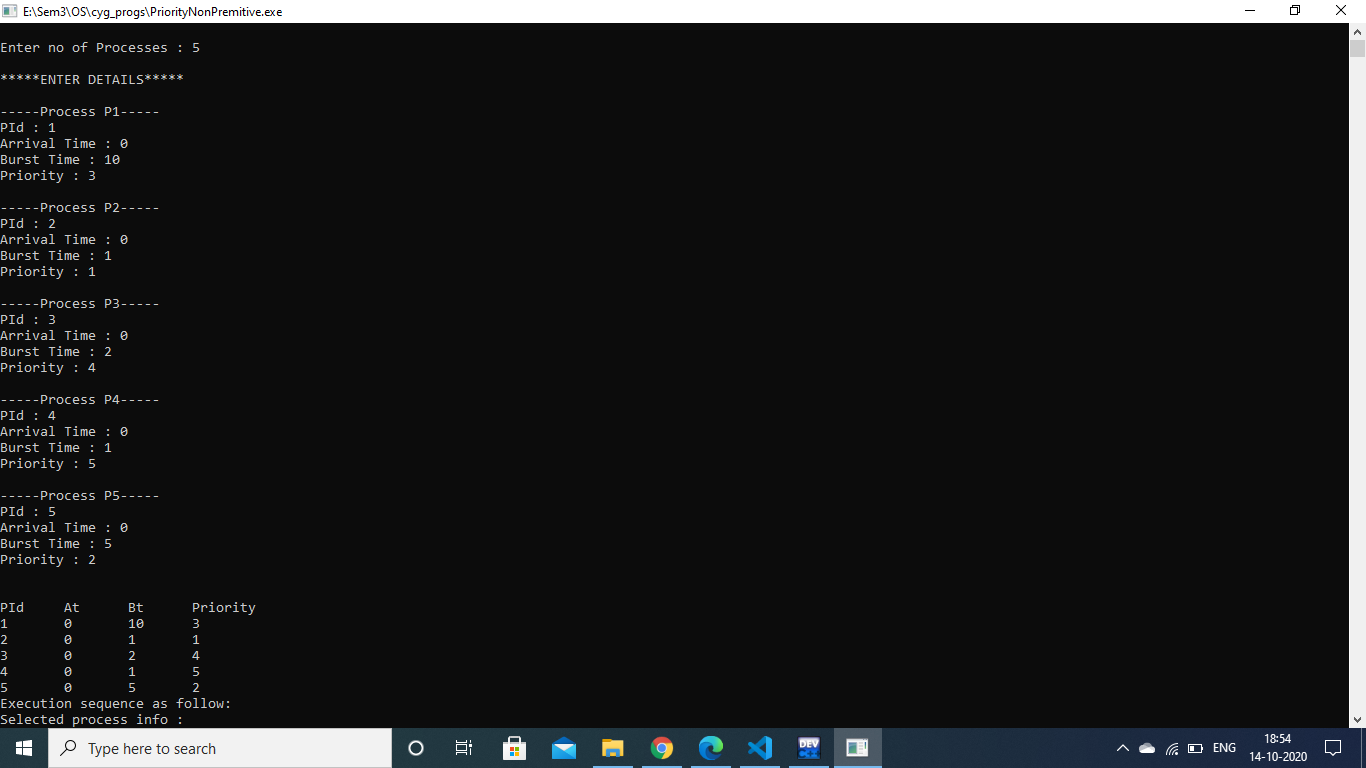
p.input();

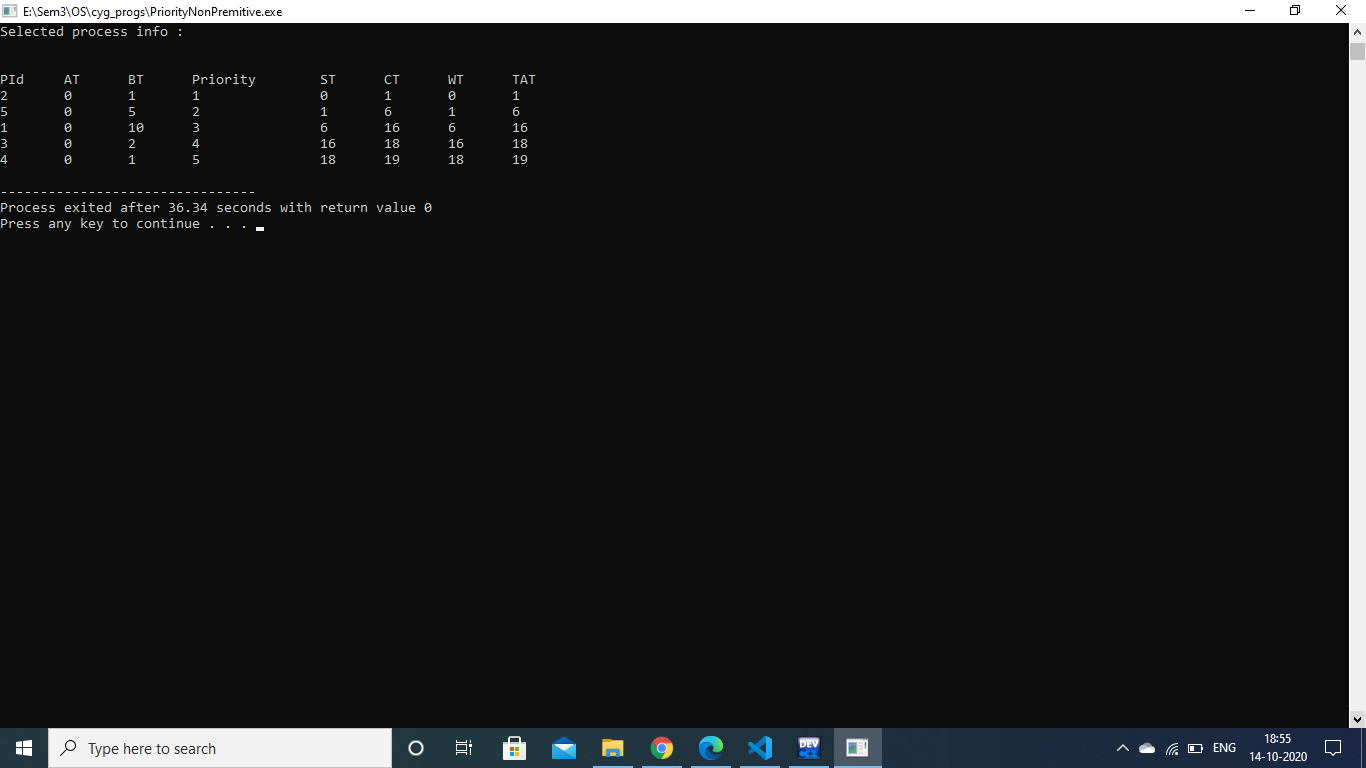
p.nonprem\_priority();

return 0;

}

*Output:*

**

**

Q11. Write a program to implement preemptive priority based scheduling algorithm.

*Code:*

//Preemptive priority based scheduling algorithm.

#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<<"-----Process P"<<i+1<<"-----"<<endl;

cout<<"PId : ";

cin>>proc[i].pid;

cout<<"Arrival Time : ";

cin>>proc[i].at;

cout<<"Burst Time : ";

cin>>proc[i].bt;

cout<<"Priority : ";

cin>>proc[i].priority;

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()

{

Process p;

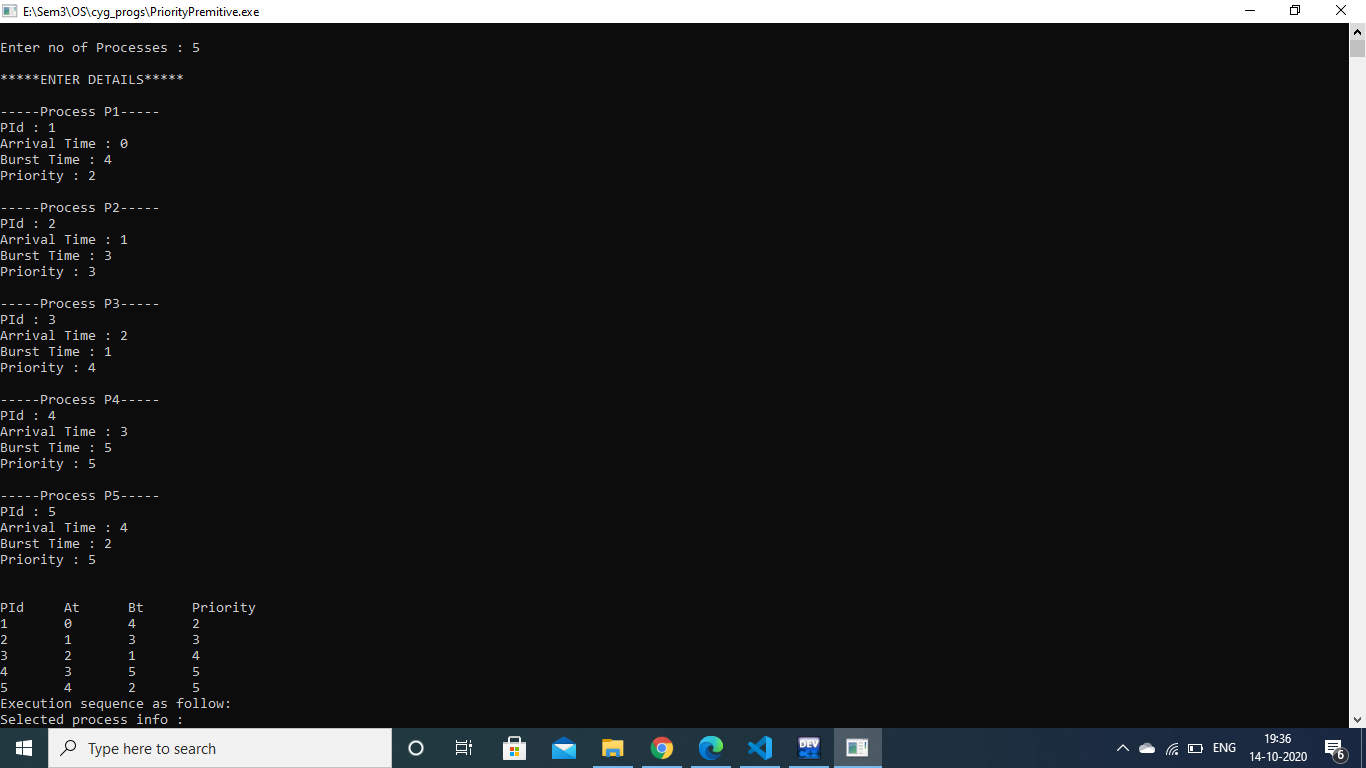
p.input();

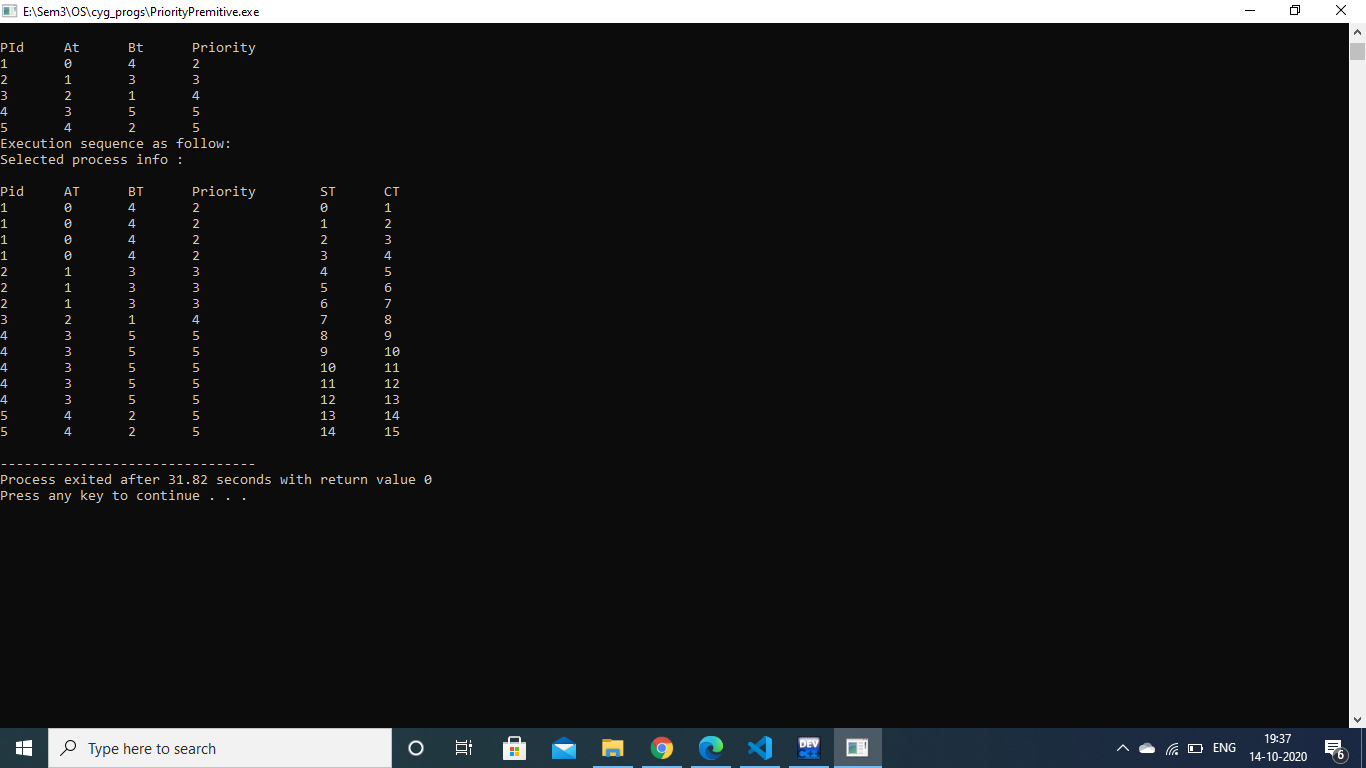
p.prem\_priority();

return 0;

}

*Output:*





Q12. Write a program to implement SRTF scheduling algorithm.

*Code:*

//Shortest Remaining Time First

#include<iostream>

using namespace std;

class Process

{

int n;

int pid;

int at;

int bt;

int rt;

int completed\_flag;

Process \*proc;

public:

void input();

void sort\_at();

void srtf();

};

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<<"-----Process P"<<i+1<<"-----"<<endl;

cout<<"PId : ";

cin>>proc[i].pid;

cout<<"Arrival Time : ";

cin>>proc[i].at;

cout<<"Burst Time : ";

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\tAT\tBT\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].rt>proc[i].rt)

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"<<ct<<"\t"<<ct+1<<"\t"<<"\n";

proc[selected\_process].rt--;

if(proc[selected\_process].rt==0)

{

proc[selected\_process].completed\_flag=1;

count++;

}

}

}

int main()

{

Process p;

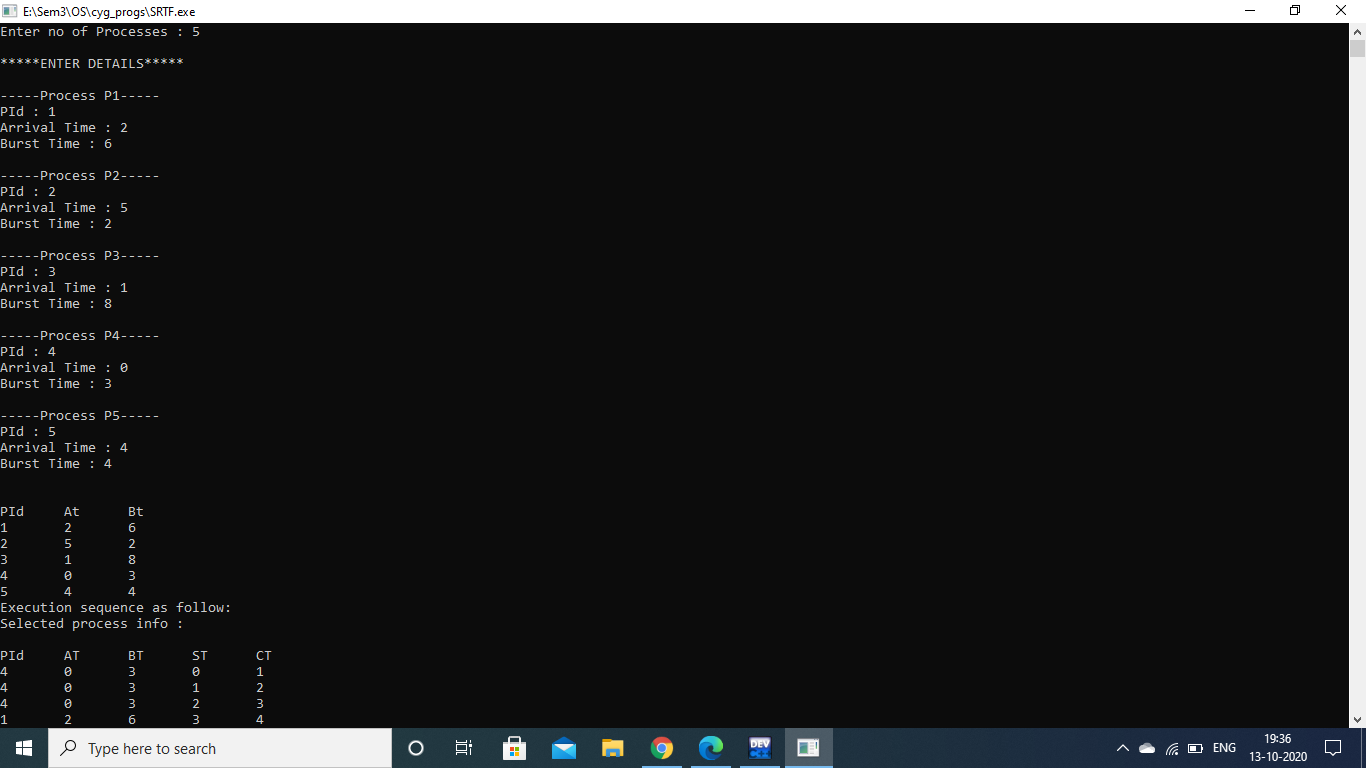
p.input();

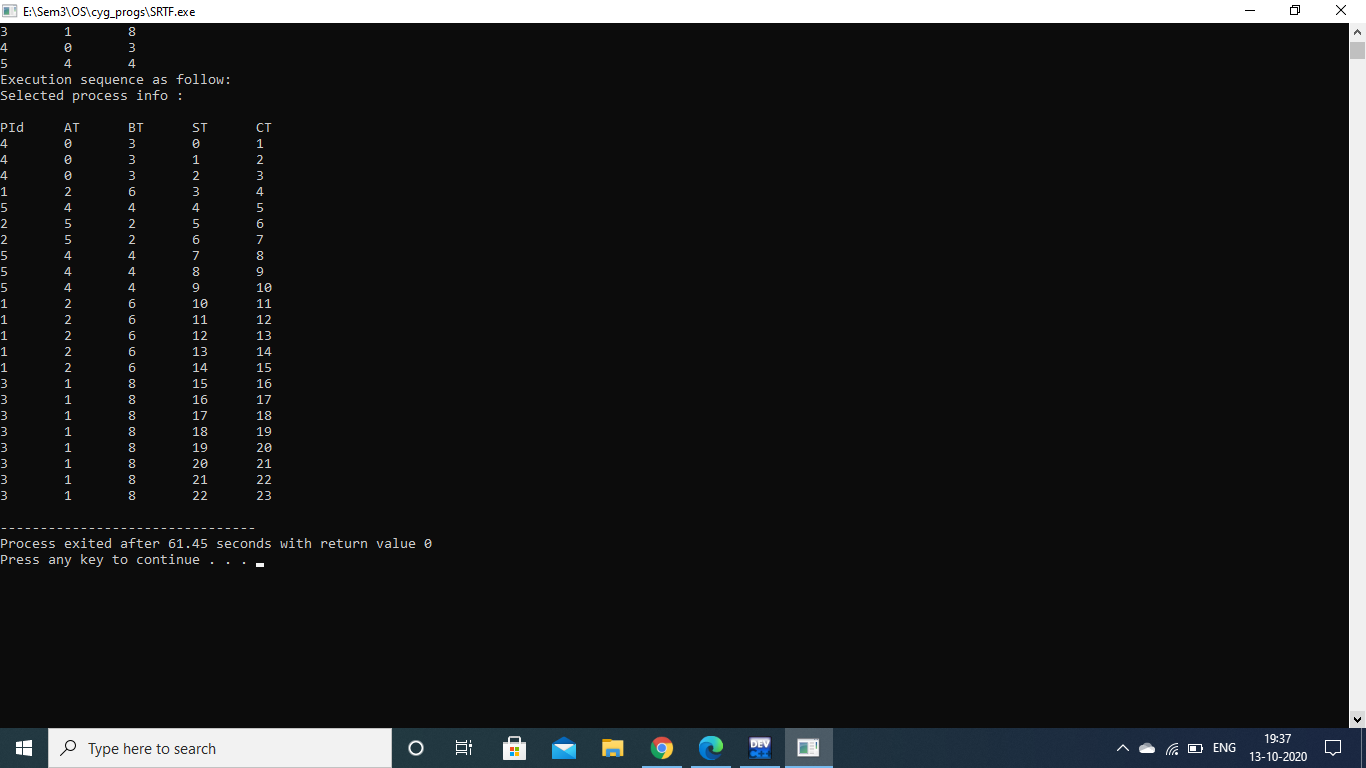
p.srtf();

return 0;

}

*Output:*





Q13. Write a program to calculate the sum of n numbers using a thread library.

*Code:*

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

#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);

pthread\_create(&tid,&attr,runner,argv[1]);

pthread\_join(tid,NULL);

cout<<"\nSUM is: "<<sum<<endl;

return 0;

}

void\* runner(void\* param)

{

int i,upper=atoi((const char\*)param);

sum=0;

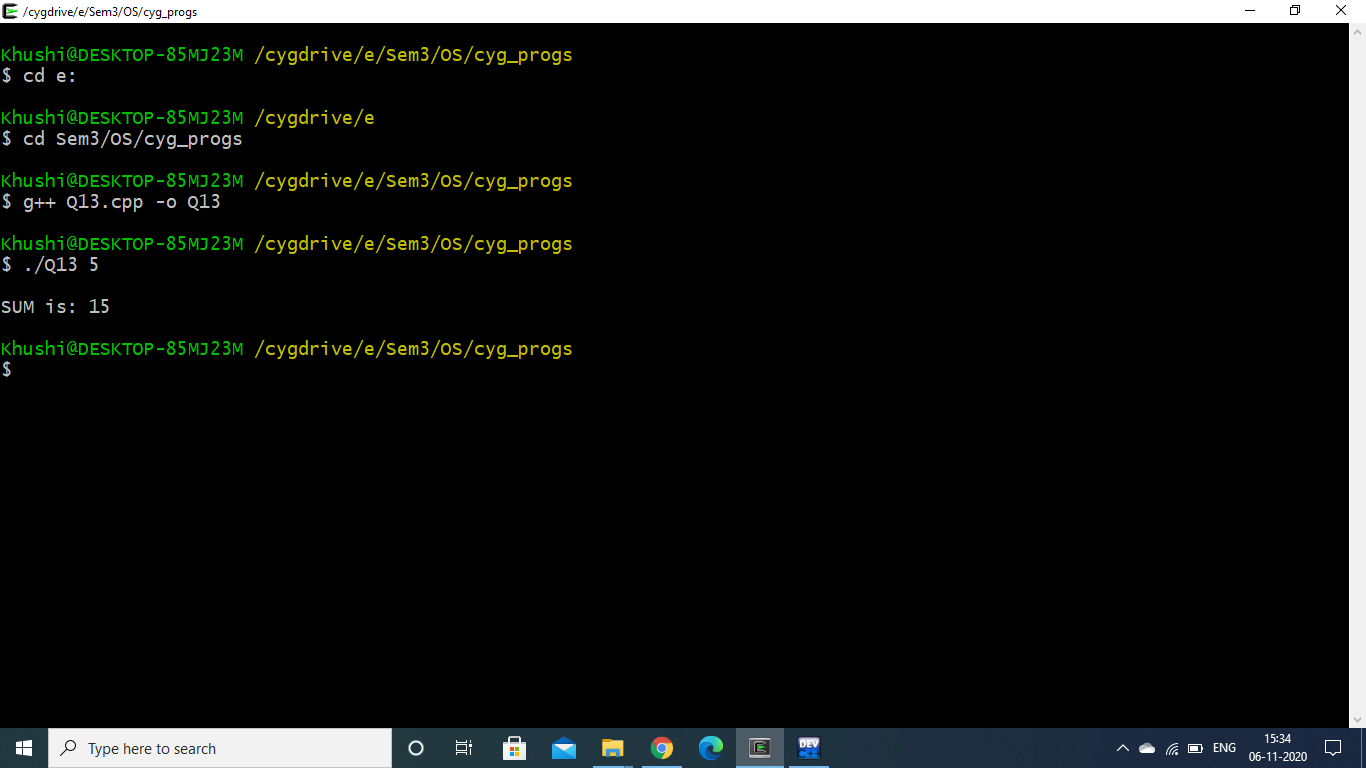
for(i=1;i<=upper;i++)

sum+=i;

pthread\_exit(0);

}

*Output:*

**

Q14. 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): ";

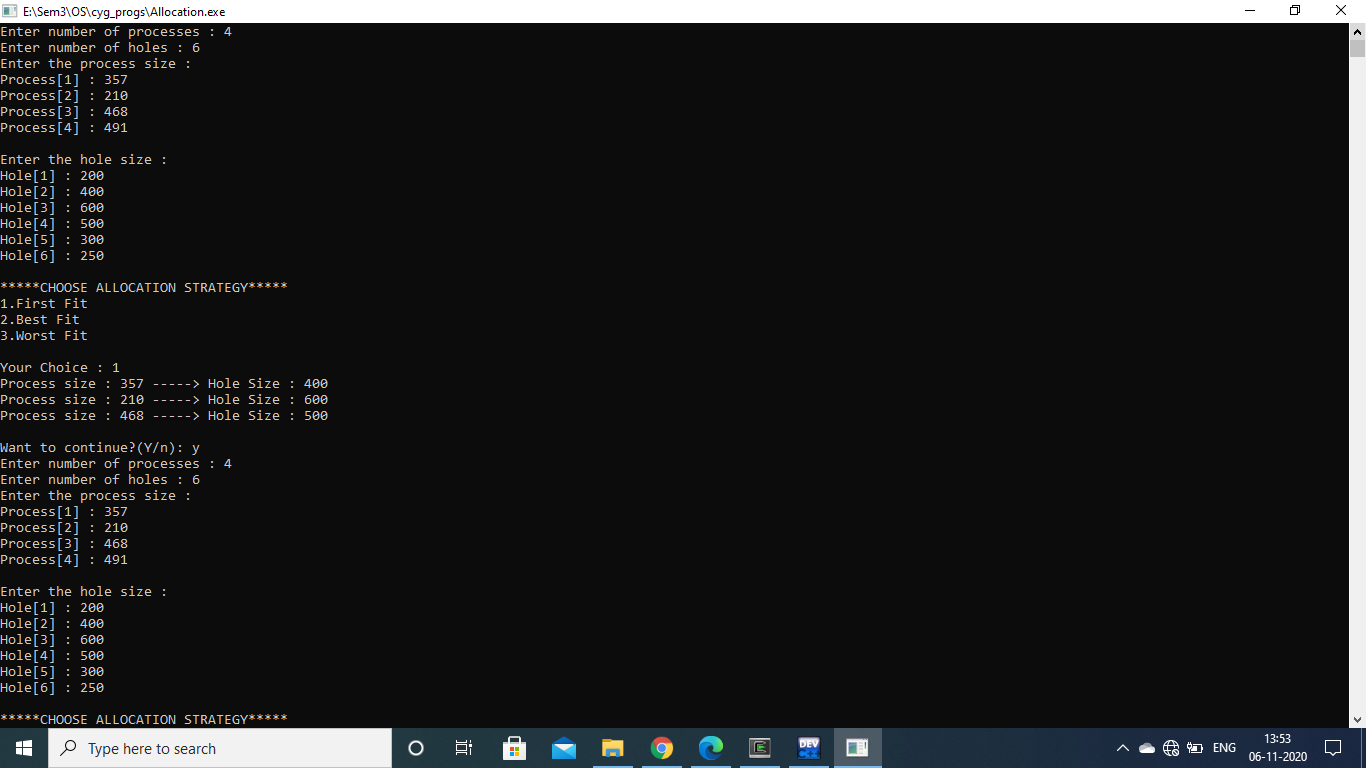
cin>>ans;

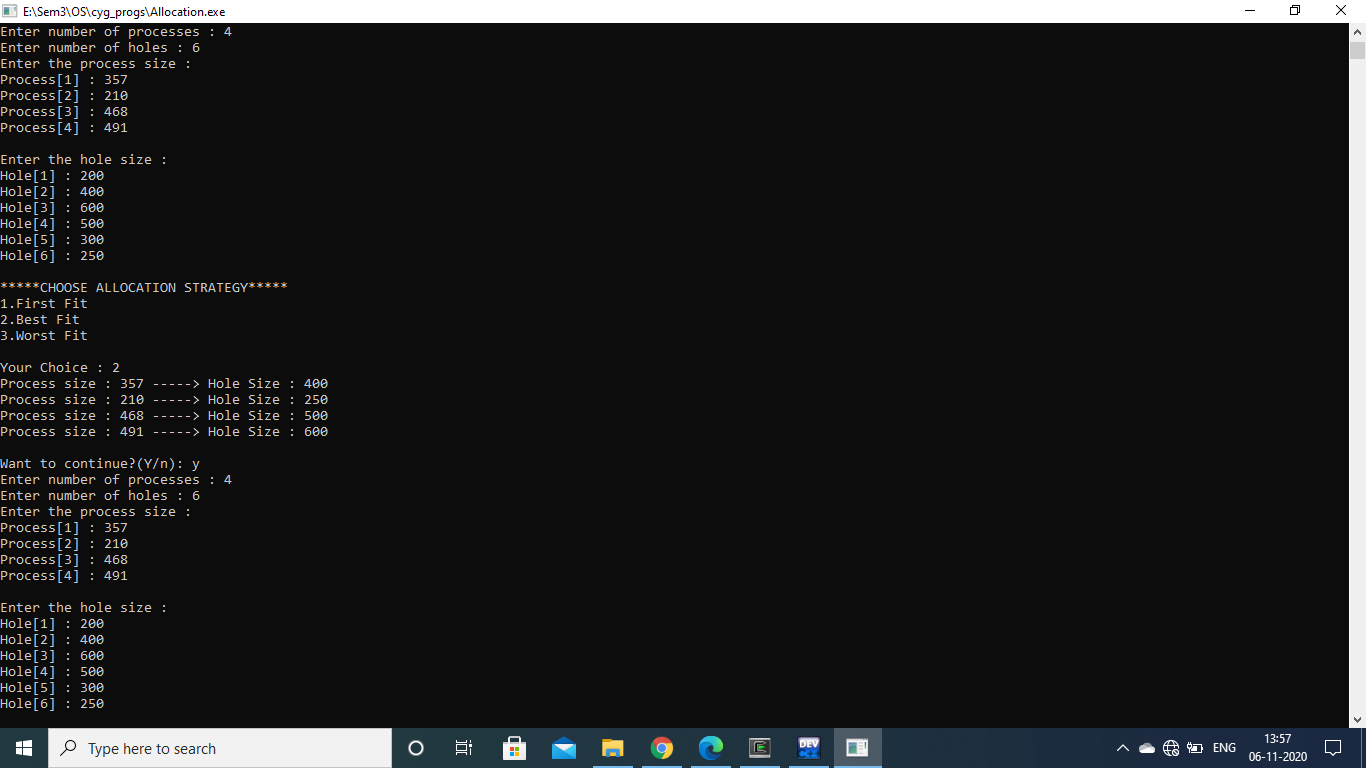
}while(ans=='Y' || ans=='y');

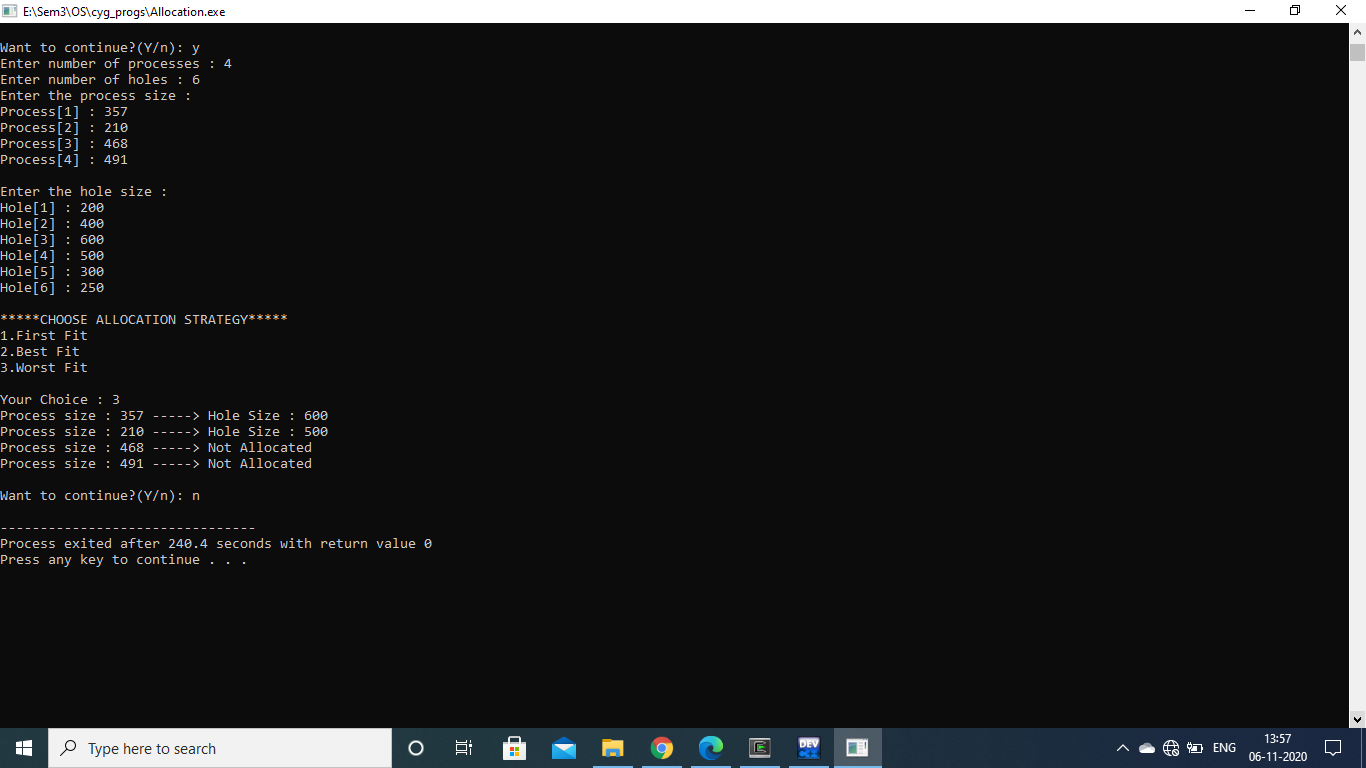
return 0;

}

*Output:*

**

**

**