**SUBMITTED BY:** Prachi Aggarwal

**COURSE:** Bsc(H) Computer Science

**ROLL NO:** 21570015

**SUBJECT:** Operating System

**UNIQUE PAPER CODE:** 32341302

**SEMESTER:** IIIrd

**YEAR:** 2nd

**SUBMITTED TO:** Vandana

**Examination Rollno:** 21033570042

**TOPIC:** Practical File

**Q1) 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.**

**a)**

#include<iostream>

#include<unistd.h>

using namespace std;

int main(){

for(int i=0; i<4; i++){

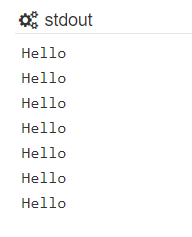
cout<<"Hello"<<endl;

fork();

}

return 0;

}



**b)**

#include<iostream>

#include<unistd.h>

using namespace std;

int main(){

int x,pid;

pid= fork();

if(pid<0){

cout<<"Fork Failed"<<endl;

return 1;

}

else if(pid==0){

cout<<"Child process running"<<endl;

cout<<"pid: "<<pid<<endl;

cout<<"Child process id:"<<getpid()<<endl;

}

else{

cout<<"Parent process running"<<endl;

cout<<"pid: "<<pid<<endl;

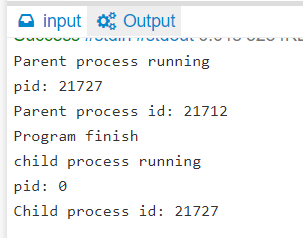
cout<<"Parent process id:"<<getpid()<<endl;

}

cout<<"Finish program"<<endl;

return 0;

}



c)

#include <iostream>

#include<unistd.h>

#include<sys/types.h>

#include<wait.h>

using namespace std;

int main(){

int pid ;

pid=fork();

if(pid<0){

cout<<"Fork Failed"<<endl;

return 1;

}

else if(pid==0){

cout<<"Child process running"<<endl;

cout<<"Child process id:"<<getpid()<<endl;

}

else{

cout<<"Parent"<<endl;

cout<<"Parent will wait"<<endl;

wait(NULL);

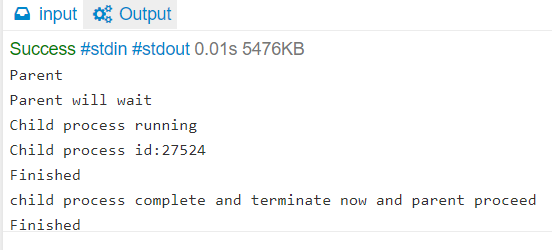
cout<<"child process complete and terminate now and parent proceed"<<endl;

}

cout<<"Finished"<<endl;

return 0;

}



**Q2) Write a program to report behaviour of Linux kernel including kernel version, CPU type and model. (CPU information).**

**And**

**Q3) Write a program to report behaviour of Linux kernel including information on 19 configured memory, amount of free and used memory. (memory information).**

**Ans)**

#include<iostream>

#include<stdlib.h>

#include<stdio.h>

using namespace std;

int main(){

int ch;

cout<"\nEnter your choice:\n";

cout<<"1 for kernal version....\n2)for CPU type mode1...\n3)for information of memory configured, amount of free and used memory....\n";

cin>>ch;

switch(ch){

case 1:

cout<<"\nKernal version is\n";

system("cat/proc/sys/kernel/osrelease");

break;

case 2:

cout<<"\ncpu type & model is:\n";

system("awk 'NR==5{print $5}'/proc/cpuinfo");

break;

case 3:

cout<<"\namount of memory configured in system\n";

system("awk 'NR==4{print $0}'/proc/meminfo");

cout<<"\nmemeory currently free in system\n";

system("awk 'NR==5{print $0}'/proc/meminfo");

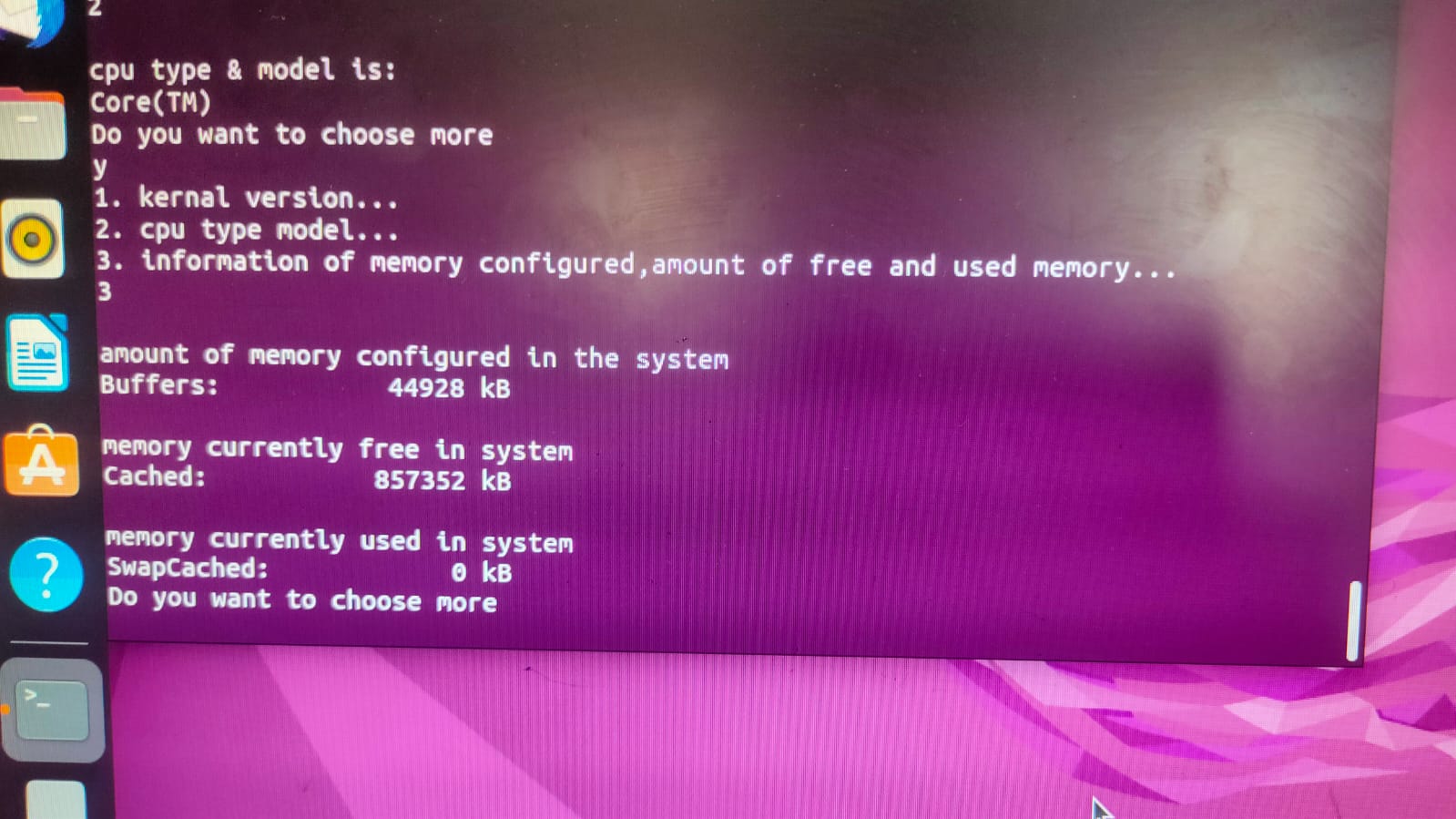
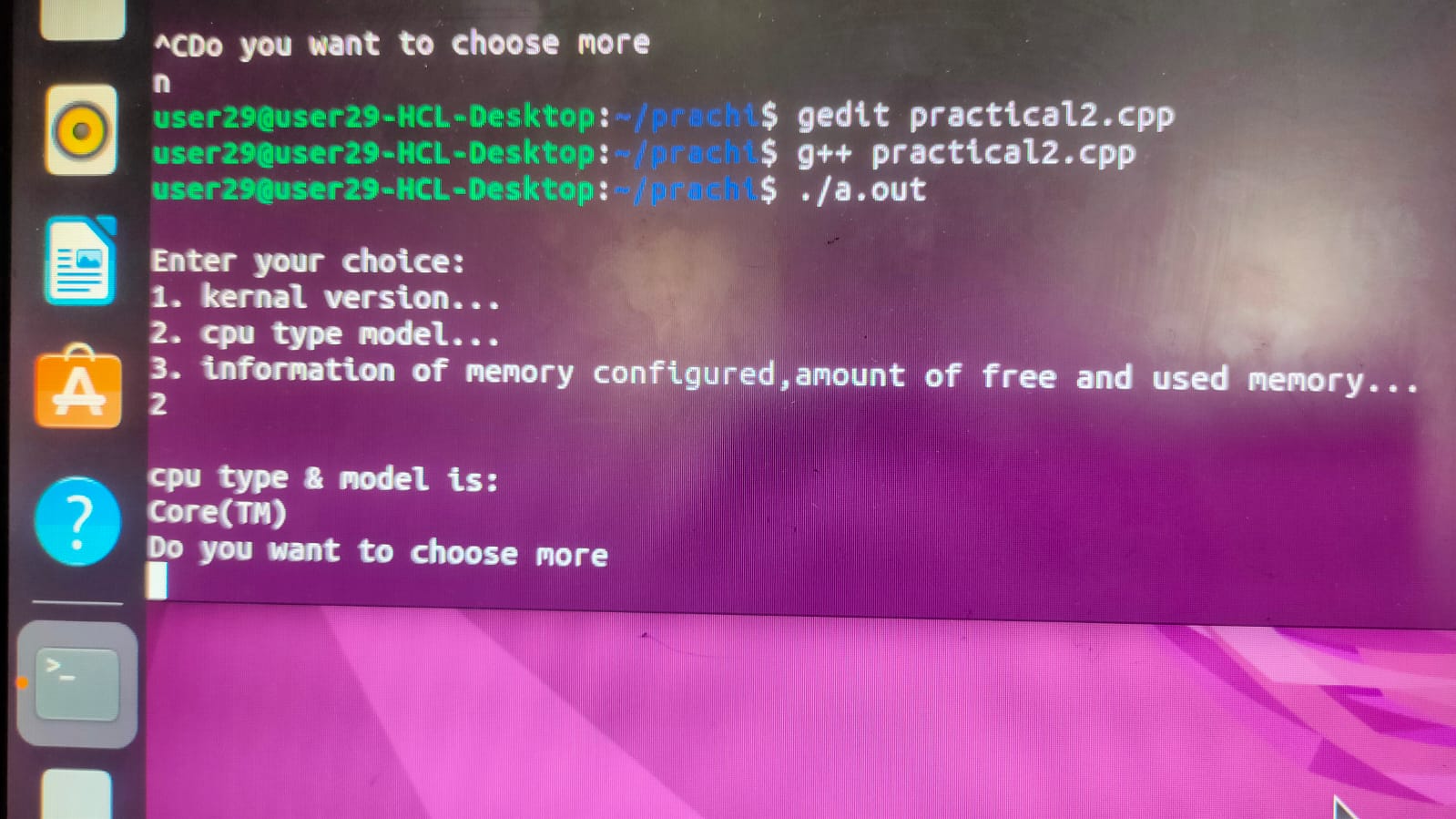
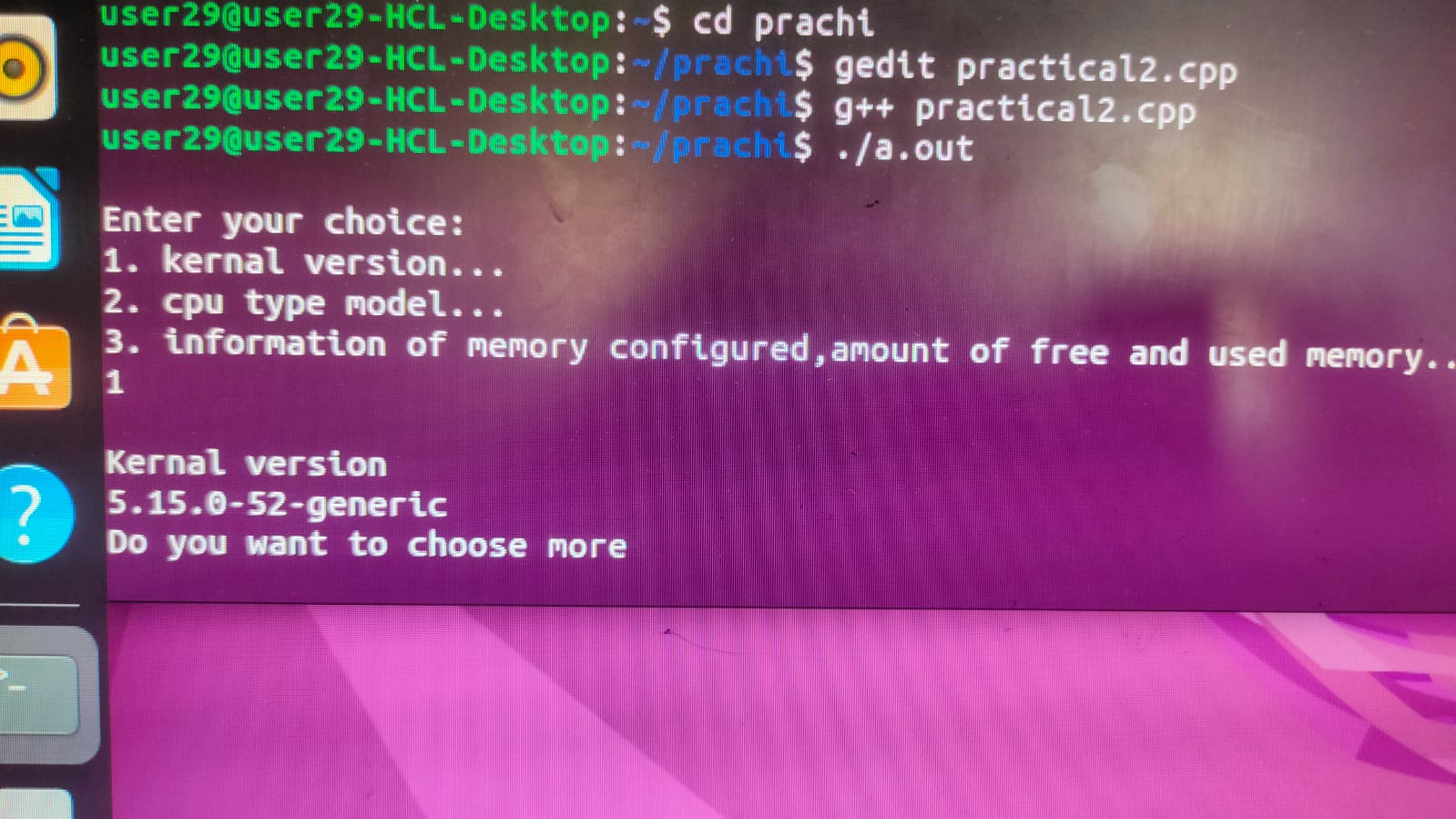
cout<<"\nmemory currently used in system\n";

system("awk 'NR==6{print $0}'/proc/meminfo");

break;

}

}



**Q4) Write a program to print file details including owner access permissions, file access time, where file name is given as argument.**

**Ans)**

#include<iostream>

#include<stdlib.h>

#include<stdio.h>

#include<sys/stat.h>

#include<unistd.h>

using namespace std;

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

int i;

struct stat s;

if(argc<2){

cout<<"\nenter filename\n";

}

for(i=1;i<argc;i++){

cout<<"file:"<<argv[i]<<"\n";

if(stat(argv[i],&s)<0)

cout<<"error to obtaining stats\n";

else

{

cout<<"owner UID:";

cout<<s.st\_uid;

cout<<"\n";

cout<<"group ID:";

cout<<s.st\_gid ;

cout<<"\n";

cout<<"Access permissions:";

cout<<s.st\_mode;

cout<<"\n";

cout<<"Access time:";

cout<<s.st\_atime ;

cout<<"\n";

cout<<"File size:";

cout<<s.st\_size ;

cout<<"\n";

cout<<"file size (in blocks): ";

cout<<s.st\_blksize;

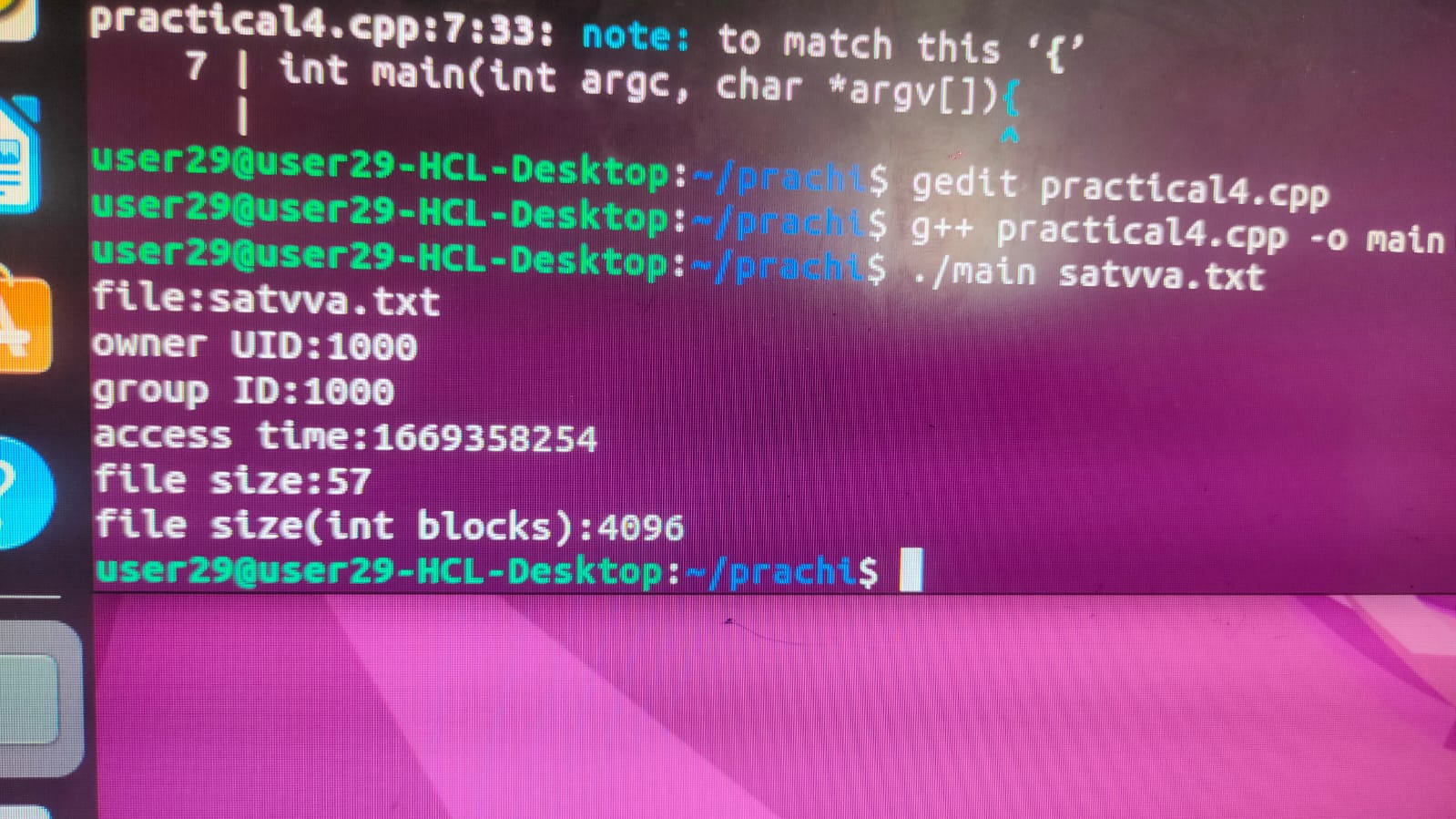
cout<<"\n";

}

}

return 0;

}



**Q5) Write a program to copy files using system calls.**

**Ans)**

#include <stdio.h>

#include <unistd.h>

#include <fcntl.h>

void main()

{

char buf;

int fd\_one, fd\_two;

fd\_one = open("first\_file", O\_RDONLY);

if (fd\_one == -1)

{

printf("Error opening first\_file\n");

close(fd\_one);

return;

}

fd\_two = open("second\_file",

O\_WRONLY | O\_CREAT,

S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IROTH);

while(read(fd\_one, &buf, 1))

{

write(fd\_two, &buf, 1);

}

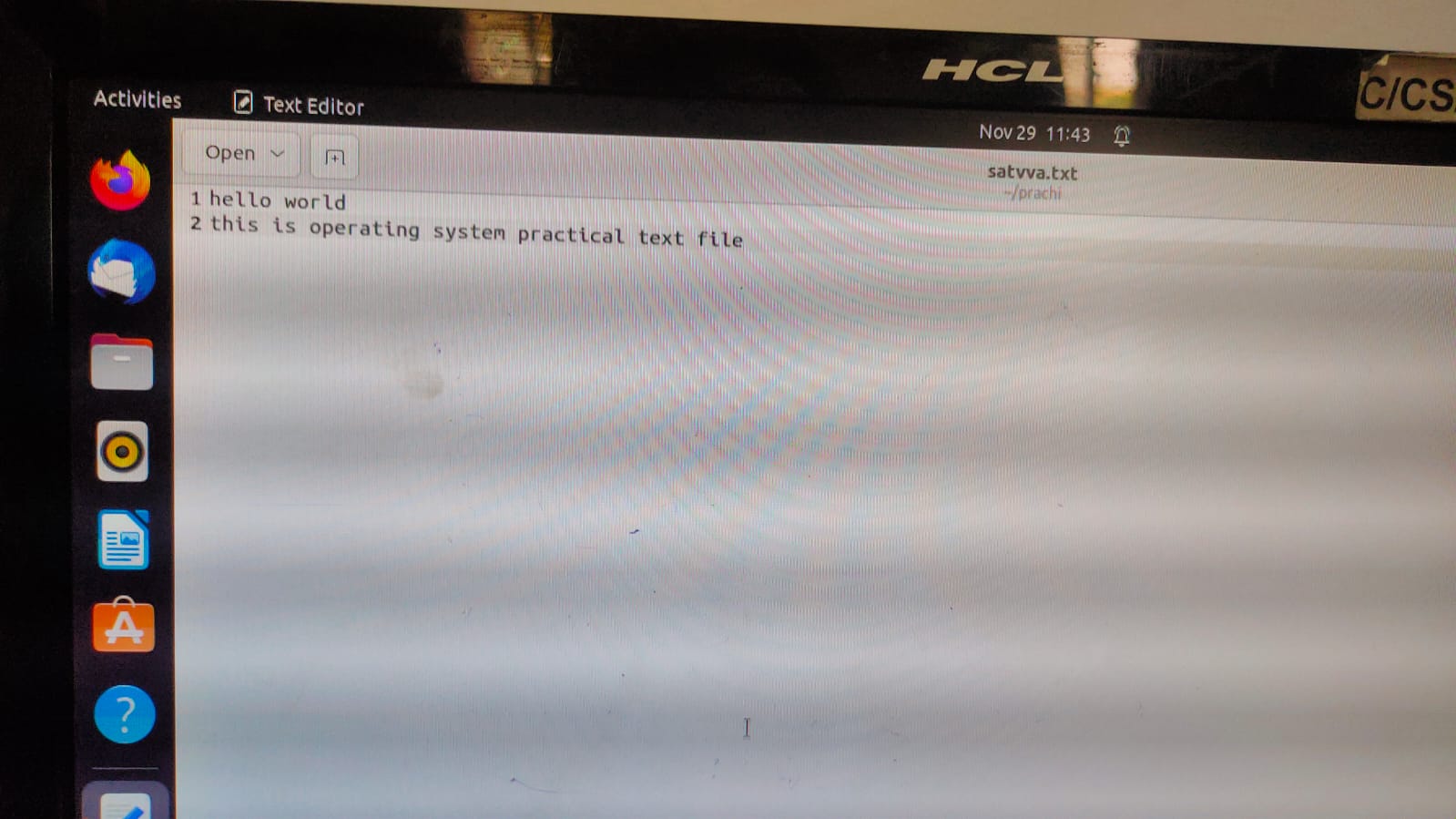
printf("Successful copy");

close(fd\_one);

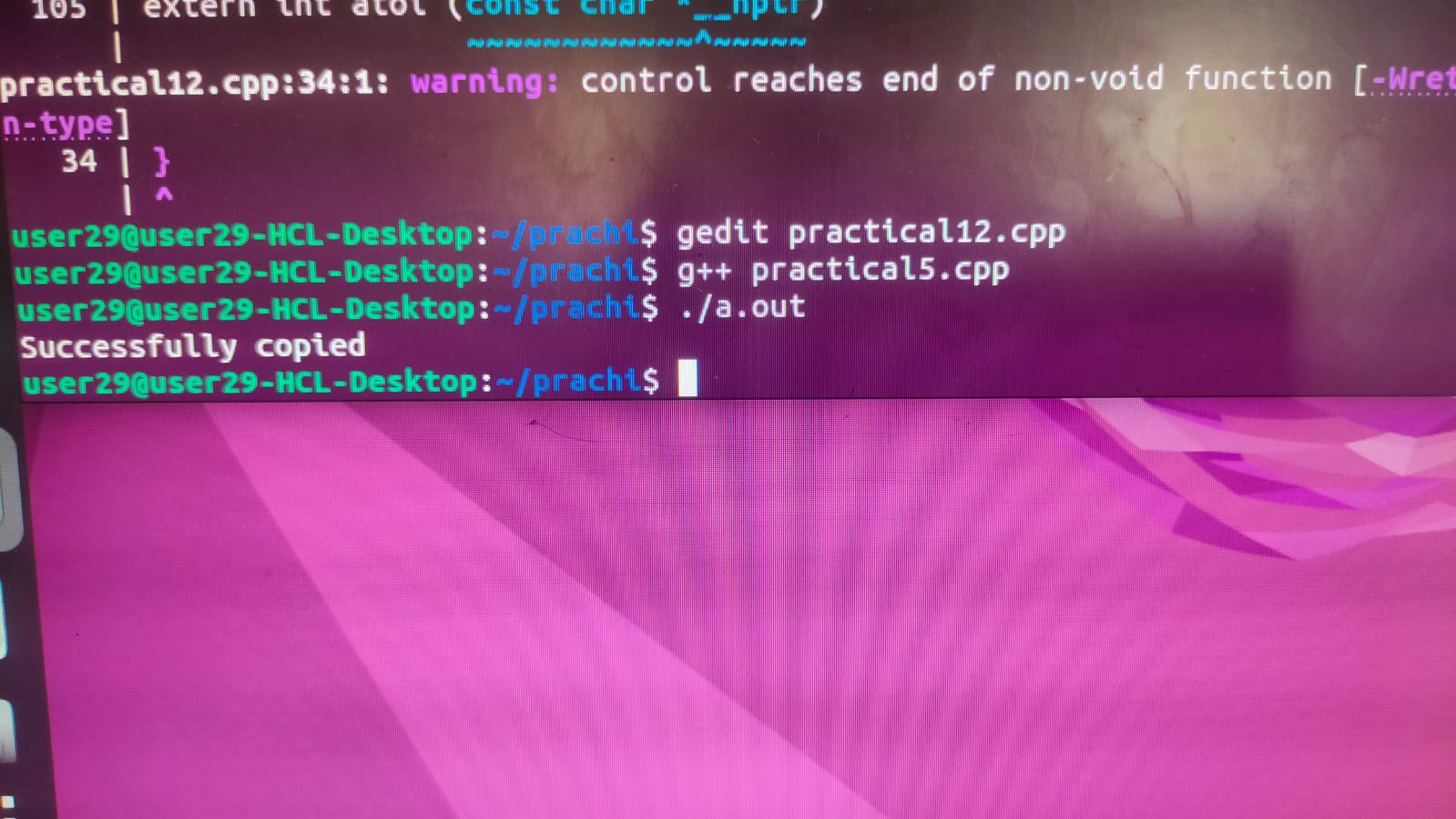
close(fd\_two);

}

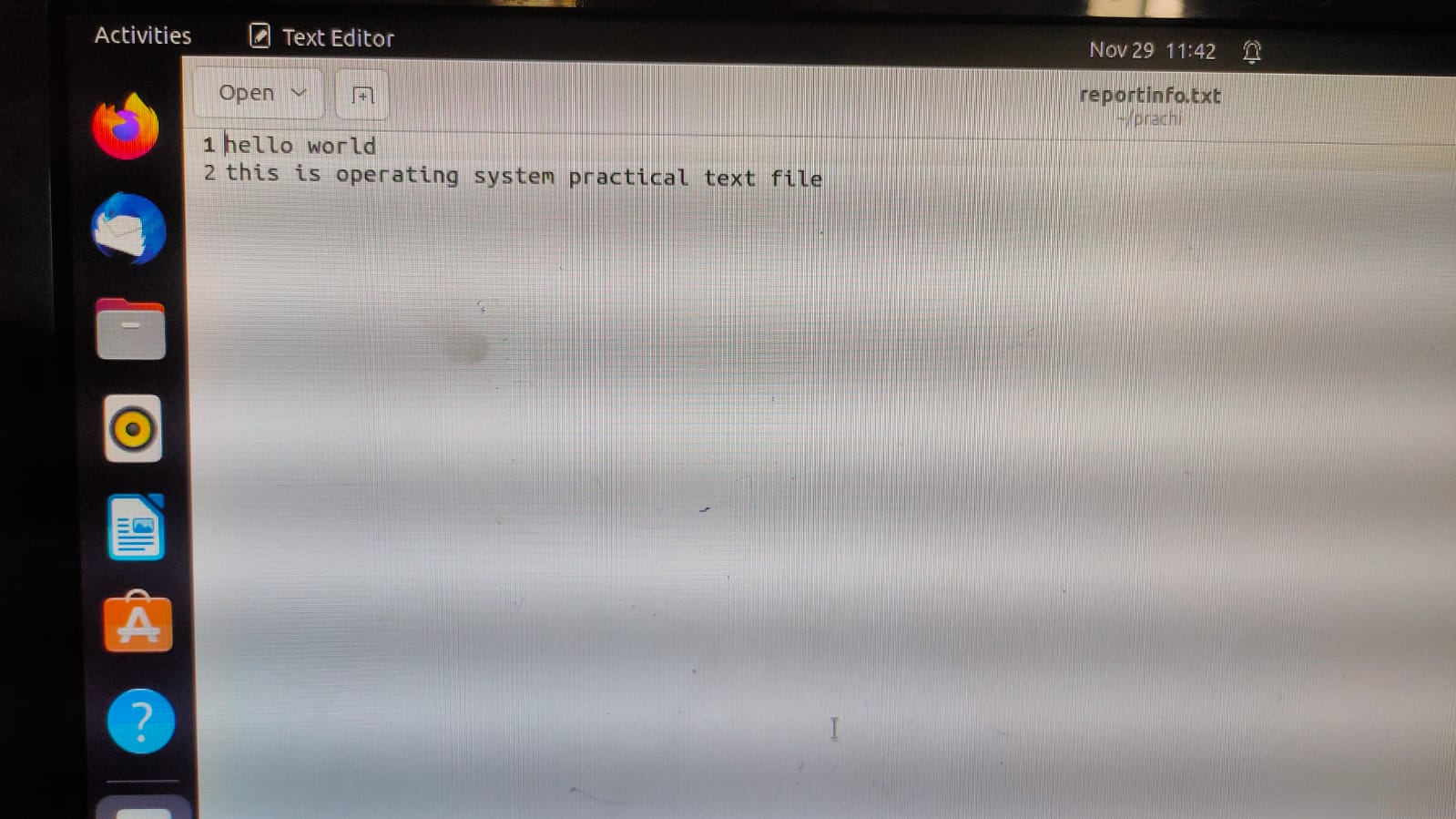
**Satvva.txt**

****

**Output**



**Reportinfo.txt**

****

**Q6) Write a program to implement FCFS scheduling algorithm.**

**Ans)**

#include<iostream>

using namespace std;

typedef struct algo{

float at;

float bt;

float tt;

float et;

float wt;

}T;

class FCFS{

private:

int size;

T \*arr;

public:

FCFS(int s){

size = s;

arr = new T[size];

}

void input(){

cout<<"Enter the arrival time of each process:"<<endl;

for(int i=0; i<size; i++){

cin>>arr[i].at;

}

cout<<"Enter the burst time of each process:"<<endl;

for(int i=0; i<size;i++){

cin>>arr[i].bt;

}

}

void algorithm(){

input();

int avg\_t, avg\_w;

int time =0 ;

for(int i=0;i<size; i++){

time += arr[i].bt;

arr[i].et=time;

}

cout<<"\*\*\*\*\*\*\*\*\*\*FCFS\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"Process\t\tArrival Time\tBurst Time\tExit Time\tTurnAround Time\tWaiting Time\n";

for(int i=0; i<size;i++){

arr[i].tt=arr[i].et-arr[i].at;

avg\_t += arr[i].tt;

arr[i].wt= arr[i].tt-arr[i].bt;

avg\_w += arr[i].wt;

cout<<"P"<<i<<"\t\t\t"<<arr[i].at<<"\t"<<arr[i].bt<<"\t"<<arr[i].et<<"\t"<<arr[i].tt<<"\t\t"<<arr[i].wt<<endl;

}

cout<<"Average turnaround time is: "<<endl;

cout<<(avg\_t/size)<<endl;

cout<<"Average waiting time is: "<<endl;

cout<<(avg\_w/size)<<endl;

}

}

};

int main(){

int proces;

cout<<"Enter the number of processes"<<endl;

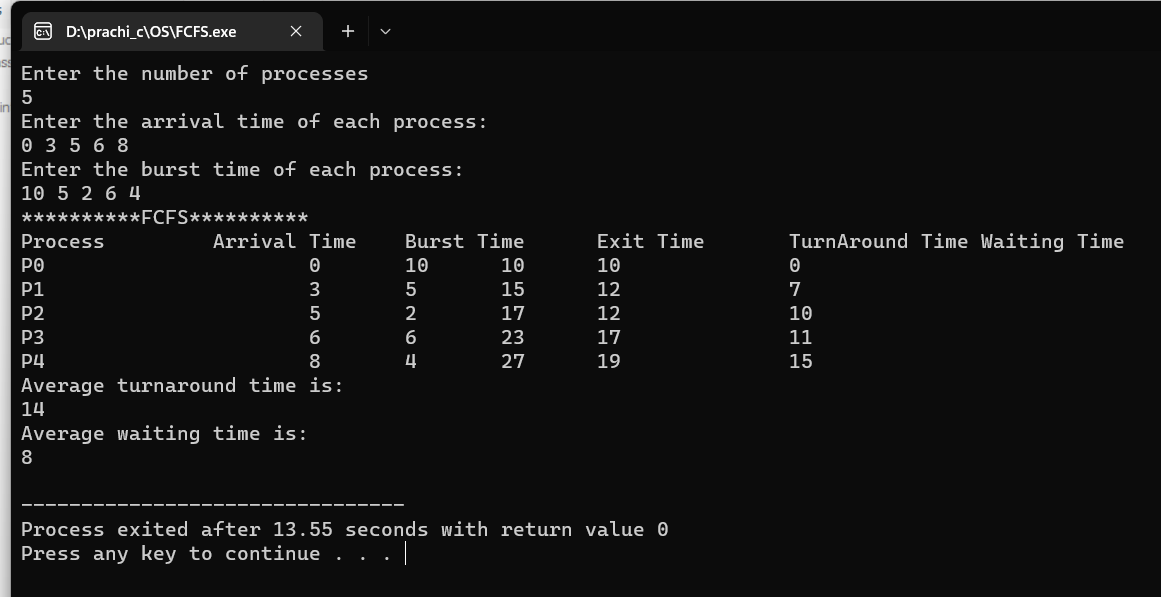
cin>>proces;

FCFS s(proces);

s.algorithm();

return 0;

}



**Q7) Write a program to implement Round Robin scheduling algorithm.**

#include<iostream>

using namespace std;

typedef struct process{

float at;

float bt;

float et;

float tt;

float wt;

float rmBt;

int pr;

} T;

class Robin{

private:

int size;

T \*arr;

int quant;

public:

Robin(int s, int q)

{

quant= q;

size=s;

arr= new T[size];

}

void input(){

cout<<"\*\*\*\*\*\*\*\*\*\*Arrival Time\*\*\*\*\*\*\*\*"<<endl;

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

{

cout<<"AT of P"<<i<<"= ";

cin>>arr[i].at;

}

cout<<"\*\*\*\*\*\*\*\*\*\*\*Burst Time\*\*\*\*\*\*\*\*\*\*"<<endl;

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

{

cout<<"BT of P"<<i<<"= ";

cin>>arr[i].bt;

}

}

void algorithm(){

input();

for(int i=0; i<size; i++){

arr[i].rmBt=arr[i].bt;

}

int time = 0;

while(1){

bool done = true;

for(int i=0; i<size; i++){

if(arr[i].rmBt > 0){

done=false;

if(arr[i].rmBt > quant){

time += quant;

arr[i].rmBt -= quant;

}

else{

time = time + arr[i].rmBt ;

arr[i].wt = time - arr[i].bt;

arr[i].rmBt = 0;

}

}

}

if(done==true){

break;

}

}

for(int i=0; i<size; i++){

arr[i].tt = arr[i].bt + arr[i].wt ;

}

int avg\_tt;

int avg\_wt;

for(int i=0; i<size; i++){

avg\_tt += arr[i].tt;

avg\_wt += arr[i].wt;

}

cout<<"\t\t\t\*\*Round Robin\*\*"<<endl;

cout<<"Process\t\tArrival Time\tBurst Time\tTurnAround Time\t\tWaiting Time\n";

for(int i=0; i<size;i++){

cout<<"P"<<i<<"\t\t\t"<<arr[i].at<<"\t "<<arr[i].bt<<"\t\t\t"<<arr[i].tt<<"\t\t\t"<<arr[i].wt<<endl;

}

cout<<"Average turnaround time is: "<<(avg\_tt/size)<<endl;

cout<<"Average waiting time is: "<<(avg\_wt/size)<<endl;

}

};

int main(){

int proces;

cout<<"Enter the number of processes"<<endl;

cin>>proces;

int quantum;

cout<<"Enter the time quantum for each process"<<endl;

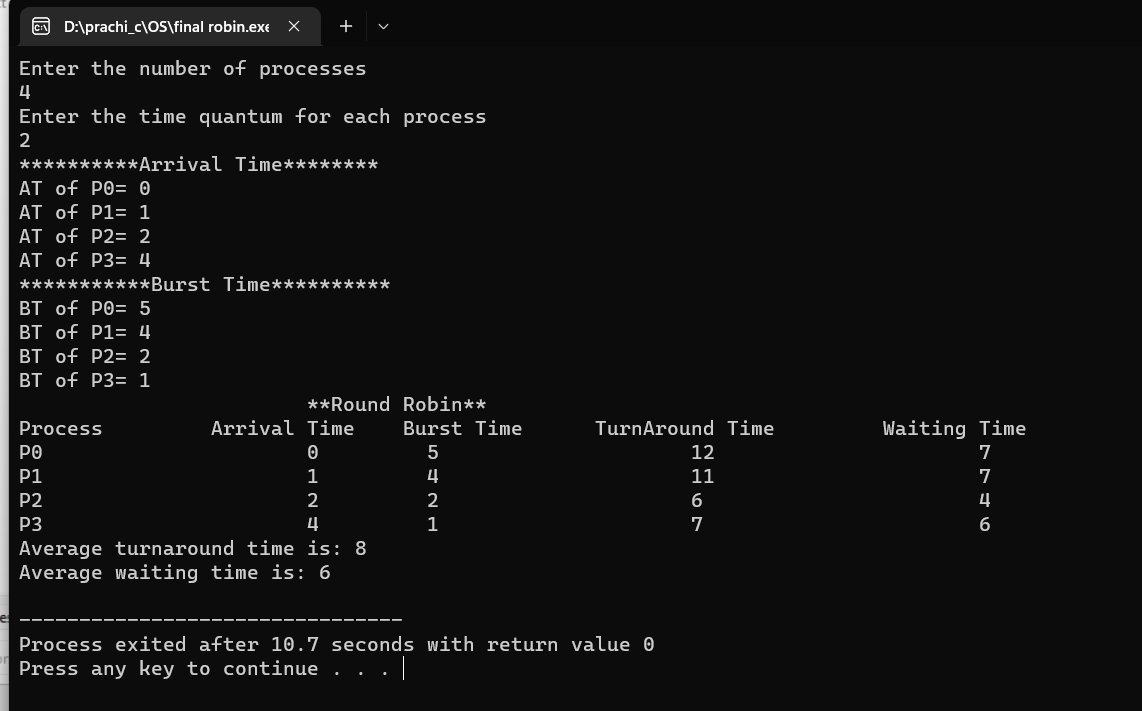
cin>>quantum;

Robin s(proces,quantum);

s.algorithm();

return 0;

}

****

**Q8) Write a program to implement SJF scheduling algorithm.**

**Ans)**

#include<iostream>

using namespace std;

typedef struct algo{

bool done = false;

float at;

float bt;

float tt;

float et;

float wt;

}T;

class SJF{

private:

int size;

T \*arr;

public:

SJF(int s){

size = s;

arr = new T[size];

}

void input(){

cout<<"Enter the arrival time of each process:"<<endl;

for(int i=0; i<size; i++){

cin>>arr[i].at;

}

cout<<"Enter the burst time of each process:"<<endl;

for(int i=0; i<size;i++){

cin>>arr[i].bt;

}

}

void algorithm(){

input();

float time=0;

int count=0;

float min=100;

int index;

while(count<size){

for(int i=0; i<size;i++){

if(arr[i].at<=time && arr[i].done!=true){

if(arr[i].bt<min){

min=arr[i].bt;

index=i;

}

}

}

time += min;

arr[index].et=time;

min=100;

count++;

arr[index].done=true;

}

for(int i=0;i<size;i++){

arr[i].tt=arr[i].et-arr[i].at;

}

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

{

arr[i].wt= arr[i].tt-arr[i].bt;

}

cout<<"\*\*\*\*\*\*\*\*\*\*SJF\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"Process\t\tArrival Time\tBurst Time\tExit Time\tTurnAround Time\tWaiting Time\n";

for(int i=0; i<size;i++){

cout<<"P"<<i<<"\t\t\t"<<arr[i].at<<"\t"<<arr[i].bt<<"\t"<<arr[i].et<<"\t"<<arr[i].tt<<"\t\t"<<arr[i].wt<<endl;

}

}

};

int main(){

int proces;

cout<<"Enter the number of processes"<<endl;

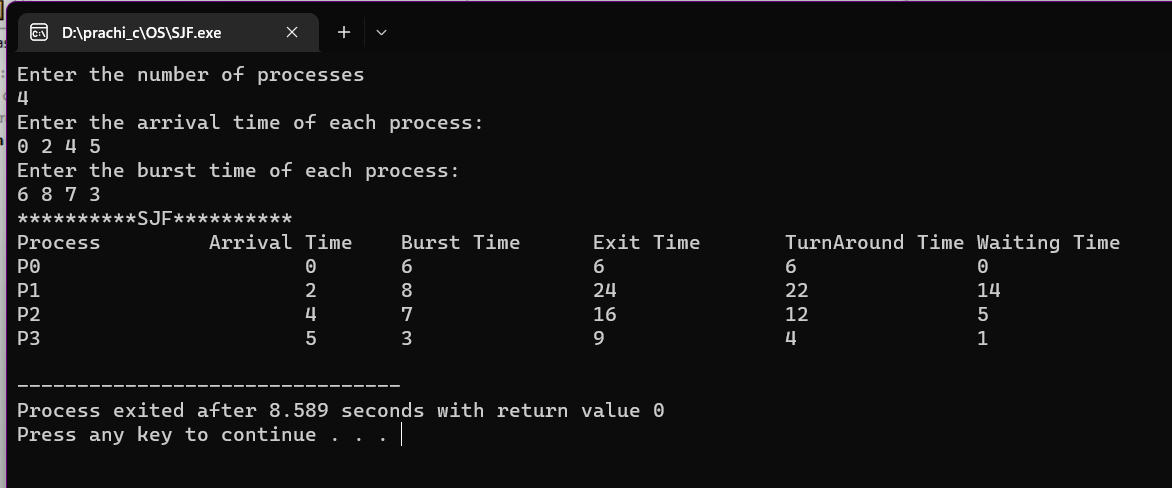
cin>>proces;

SJF s(proces);

s.algorithm();

return 0;

}



**Q9) Write a program to implement non-preemptive priority based scheduling algorithm.**

**Ans)**

#include<iostream>

using namespace std;

typedef struct process{

float at;

float bt;

float et;

float tt;

float wt;

int pr;

bool done=false;

} T;

class Priority{

int size;

T \*arr;

public:

Priority(int s){

size= s;

arr = new T[size];

}

void algorithm()

{

float time=0.0;

int count=0;

int high;

int index;

cout<<"Enter the priority of each process"<<endl;

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

{

cin>>arr[i].pr;

}

cout<<"Enter the arrival time of each process"<<endl;

for(int i=0; i<size; i++){

cin>>arr[i].at;

}

cout<<"Enter the burst time of each process"<<endl;

for(int i=0; i<size; i++){

cin>>arr[i].bt;

}

while(count<size){

high=50;

for(int i=0 ;i<size; i++){

if(time>=arr[i].at && arr[i].done!=true){

if(arr[i].pr<high){

high = arr[i].pr;

index=i;

}

}

}

time += arr[index].bt;

arr[index].et= time;

count++;

arr[index].done=true;

}

for(int i=0;i<size;i++){

arr[i].tt=arr[i].et-arr[i].at;

}

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

{

arr[i].wt= arr[i].tt-arr[i].bt;

}

cout<<"\*\*\*\*\*\*\*\*\*\*PRIORITY SCHEDULING\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"Process\t\tArrival Time\tPriority\tBurst Time\tExit Time\tTurnAround Time\tWaiting Time\n";

for(int i=0; i<size;i++){

cout<<"P"<<i<<"\t\t\t"<<arr[i].at<<"\t"<<arr[i].pr<<"\t"<<arr[i].bt<<"\t"<<arr[i].et<<"\t"<<arr[i].tt<<"\t\t"<<arr[i].wt<<endl;

}

}

};

int main(){

int proces;

cout<<"Enter the number of processes"<<endl;

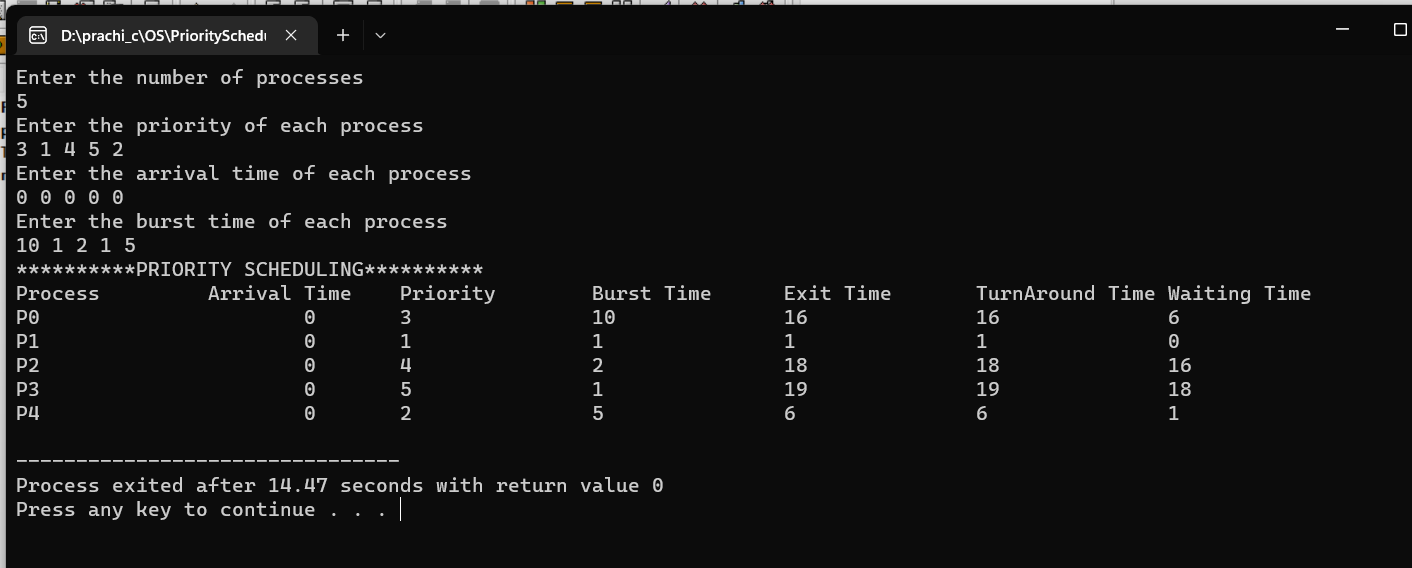
cin>>proces;

Priority s(proces);

s.algorithm();

return 0;

}



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

#include<iostream>

using namespace std;

typedef struct process{

float at;

float bt;

float et;

float comp=0;

float tt;

float wt;

int pr;

bool done=false;

} T;

class Priority{

T \*arr;

int size;

public:

Priority(int s)

{

size=s;

arr = new T[size];

}

void input(){

cout<<"\*\*\*\*\*\*\*\*\*\*Arrival Time\*\*\*\*\*\*\*\*"<<endl;

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

{

cout<<"AT of P"<<i<<"= ";

cin>>arr[i].at;

}

cout<<"\*\*\*\*\*\*\*\*\*\*\*Burst Time\*\*\*\*\*\*\*\*\*\*"<<endl;

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

{

cout<<"BT of P"<<i<<"= ";

cin>>arr[i].bt;

}

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

{

cin>>arr[i].pr;

}

}

void algorithm(){

input();

int time =0 ;

int count=0;

int index;

int max=0;

while(count<size){

for(int i=0; i<size && arr[i].done==false; i++){

for(int j=0; j<size; j++){

if(arr[j].at<=time){

if(arr[j].pr>max && arr[j].done==false){

max=arr[j].pr;

index=j;

}

}

}

if(arr[size-1].at<=time){

for(int a=0; a<arr[index].bt && arr[index].comp<arr[index].bt; a++){

arr[index].comp += 1;

time += 1;

}

arr[index].et=time;

arr[index].done = true;

count++;

}

else{

for(int a=0; time<arr[index+1].at && arr[index].comp<=arr[index].bt; a++){

arr[index].comp += 1;

time += 1;

}

if(arr[index].comp==arr[index].bt){

arr[index].et=time;

arr[index].done=true;

count++;

}

}

max=0;

}

}

for(int i=0;i<size;i++){

arr[i].tt=arr[i].et-arr[i].at;

}

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

{

arr[i].wt= arr[i].tt-arr[i].bt;

}

cout<<"\*\*\*\*\*\*\*\*\*\*PRIORITY SCHEDULING\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"Process\t\tArrival Time\tPriority\tBurst Time\tExit Time\tTurnAround Time\tWaiting Time\n";

for(int i=0; i<size;i++){

cout<<"P"<<i<<"\t\t\t"<<arr[i].at<<"\t"<<arr[i].pr<<"\t\t"<<arr[i].bt<<"\t\t"<<arr[i].et<<"\t\t"<<arr[i].tt<<"\t\t"<<arr[i].wt<<endl;

}

}

};

int main(){

int proces;

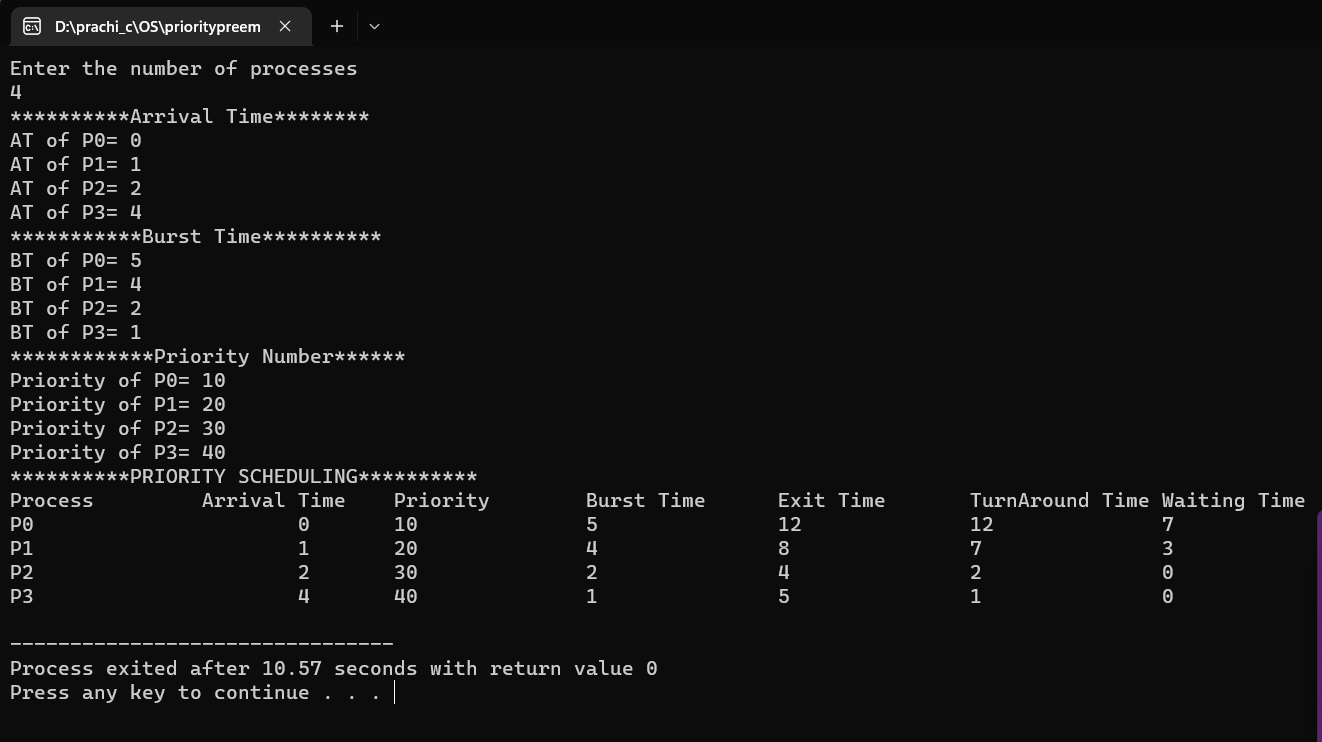
cout<<"Enter the number of processes"<<endl;

cin>>proces;

Priority s(proces);

s.algorithm();

return 0;

}****

**Q11. Write a program to implement SRJF scheduling algorithm.**

**Ans)**

#include <iostream>

using namespace std;

int main()

{

float at[5], bt[5], flag[5], ta[5], wt[5], tbt[5], et = 0, small, time = 0;

int i, j, index, k, n, done = 0;

cout << "enter the no. of process \n";

cin >> n;

cout << "enter the arriavl time\n";

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

{

cin >> at[i];

}

cout << "enter the burst time \n ";

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

{

cin >> bt[i];

}

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

{

tbt[i] = bt[i];

flag[i] = 0;

}

do

{

i = 0;

while (flag[i] == 1)

i++;

small = bt[i];

index = i;

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

{

if (flag[j] != 1)

{

if ((at[i] == at[j] && i != j) || (at[i] != at[j] && time >= at[j]))

if (bt[j] < small)

{

small = bt[j];

index = j;

}

}

}

k = index;

if (k < n - 1)

{

k++;

for (; k < n && at[k] <= time; k++)

;

if (at[k] < (time + bt[index]) && k < n)

{

et = at[k] - time;

}

else

et = bt[index];

}

time = time + et;

if (bt[index] - et == 0)

{

flag[index] = 1;

done++;

ta[index] = time - at[index];

wt[index] = ta[index] - tbt[index];

}

else

{

bt[index] = bt[index] - et;

}

} while (done != n);

cout<<endl;

cout << "process no."

<< "\t"

<< "burst time"

<< "\t"

<< "arrivaltime"

<< "\t"

<< "waiting time"

<< "\t"

<< "turnaround" << endl;

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

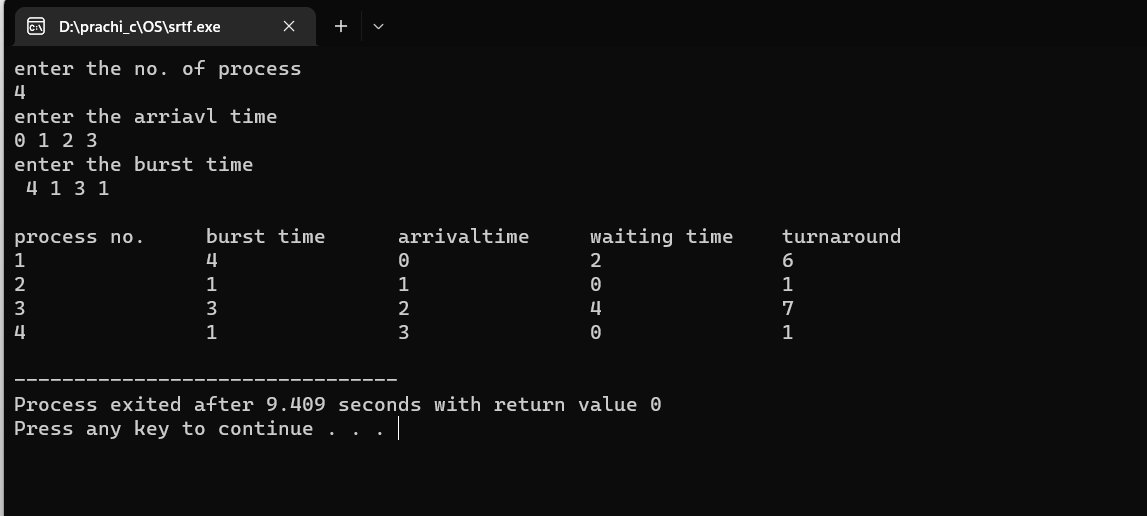
{

cout << i + 1 << "\t\t" << tbt[i] << "\t\t" << at[i] << "\t \t" << wt[i] << "\t \t" << ta[i] << "\n";

}

return 0;

}



**Q12. Write a program to calculate sum of n numbers using thread library.**

#include<unistd.h>

#include<sys/types.h>

#include<iostream>

#include<pthread.h>

using namespace std;

int sum;

int \*par;

void \*runner(void \*param);

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

{

par=(int\*)argv[0];

pthread\_t tid;

pthread\_attr\_t attr;

if(argc !=2)

{

cout<<"Error !"<<endl;

return 1;

}

if(atoi(argv[1])<0)

{

cout<<"no. Should be +ive"<<endl;

return 1;

}

pthread\_attr\_init(&attr);

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

pthread\_join(tid, NULL);

cout<<"Sum = "<<sum<<endl;

}

void \*runner(void \*param)

{

int i;

sum=0 ;

for(i=1;i<=\*par; i++)

sum += i;

pthread\_exit(0);

}

**Q13. Write a program to implement first-fit, best-fit and worst-fit allocation strategies.**

**Ans)**

#include<iostream>

using namespace std;

class allocation

{

private:

int size;

int \*arr;

int hole;

int \*holeArr;

public:

allocation(int s, int h){

size=s;

hole=h;

arr= new int[size];

holeArr= new int[hole];

}

void input()

{

cout<<"Enter the process size"<<endl;

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

{

cout<<"Enter the size of process P"<<i;

cin>>arr[i];

}

cout<<"Enter the sizes of the holes"<<endl;

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

{

cin>>holeArr[i];

}

}

void firstFit(){

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

{

for(int j=0; j<hole; j++)

{

if(arr[i]<holeArr[j]){

holeArr[j] -= arr[i];

break;

}

}

}

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

{

cout<<holeArr[i]<<" ";

}

}

void bestFit(){

int min= 5000;

int index;

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

{

for(int j=0; j<hole;j++)

{

if(arr[i]<=holeArr[j]){

if(min>holeArr[j]){

min = holeArr[j];

index =j;

}

}

}

holeArr[index] -= arr[i];

min= 5000;

}

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

{

if(holeArr[i]<0){

cout<<"Not able to get any hole"<<endl;

break;

}

cout<<holeArr[i]<<" ";

}

}

void worstFit(){

int max= 0;

int index;

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

{

for(int j=0; j<hole;j++)

{

if(arr[i]<=holeArr[j]){

if(max<holeArr[j]){

max = holeArr[j];

index =j;

}

}

}

holeArr[index] -= arr[i];

max= 0;

}

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

{

if(holeArr[i]<0){

cout<<"Not able to get any hole"<<endl;

break;

}

cout<<holeArr[i]<<" ";

}

}

};

int main()

{

int size, hole;

cout<<"Enter the number of processes"<<endl;

cin>>size;

cout<<"Enter the number of holes available in physical memory"<<endl;

cin>>hole;

allocation a(size, hole);

a.input();

int choice;

char ch;

do{

cout<<"Enter your choice:\n1)FirstFit\n2)BestFit\n3)WorstFit\n"<<endl;

cin>>choice;

switch(choice){

case 1:

a.firstFit() ;

break;

case 2:

a.bestFit() ;

break;

case 3:

a.worstFit() ;

break;

default:

cout<<"Wrong choice enter again"<<endl;

break;

}

cout<<"Do you want to choose more(y/n)?"<<endl;

cin>>ch;

}while(ch=='y');

return 0;

}

