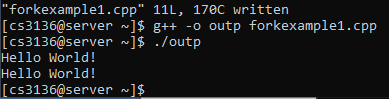
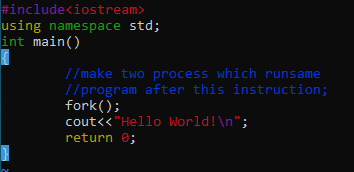
**Q1). Write a program (using fork() and/or exec() commands) where parent and child execute:**

**a) same program, same code.**

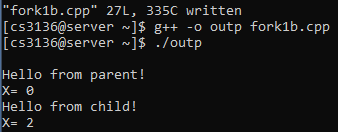
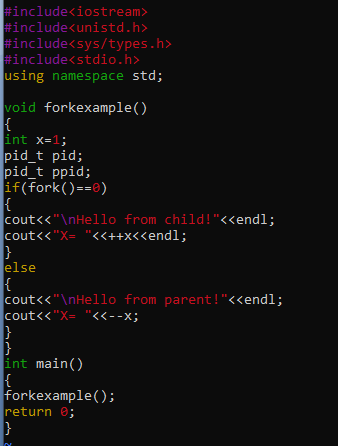
**b) same program, different code.**

**c) before terminating, the parent waits for the child to finish its task.**

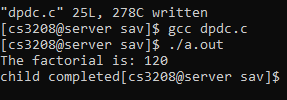
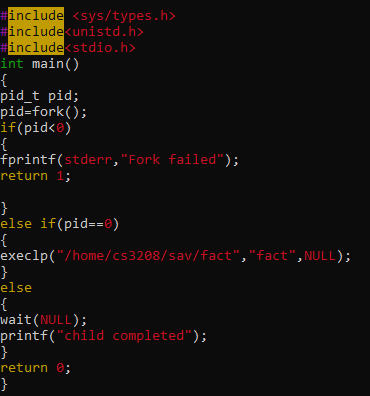
**a)**



**b)**



**c)**



a) same program, same code.

#include<iostream>

Using namespace std;

Int main(){

Fork();

Cout<<”hello world”<<endl;

}

b) same program, different code.

#include<iostream>

#include<unistd.h>

#include<stdio.h>

#include<sys/types.h>

Using namespace std;

Int main(){

Int x=1;

Pid\_t pid;

Pid\_t ppid;

If (fork()==0){

Cout<<”hello from child”<<”x”<<++x<<endl;

}else {

Cout<<”hello from parent”<<”x”<🡨x<<endl;

}

Return 0;

}

***G++ -o file file.cpp***

***./file***

c) before terminating, the parent waits for the child to finish its task.

#include<iostream>

#include<unistd.h>

#include<stdio.h>

#include<sys/types.h>

Using namespace std;

Int main(){

Int x=1;

Pid\_t pid;

Pid\_t ppid;

If (fork()==0){

Execlp(“/home/cs\_\_/folder/fact”, “fact”, NULL);

}else {

Wait(NULL);

Printf(“child completed”);

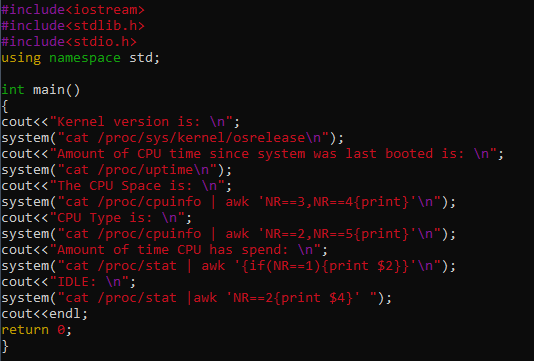
}

Return 0;

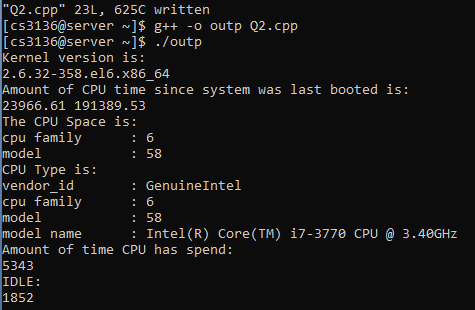
}

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

* **/proc/cpuinfo**
* **/proc/sys/kernel/osrelease**
* **/proc/uptime**
* **/proc/stat**
* **/proc/meminfo**
* **/proc/loadavg**

****

**OUTPUT:**

****

#include<iostream>

#include<unistd.h>

#include<stdio.h>

#include<stdlib.h>

#include<sys/types.h>

Using namespace std;

Int main(){

cout<<”kernel version”<<endl;

system(“cat proc/sys/kernel/osrelease”)

/proc/uptime //amt of time cpu is up since last booted

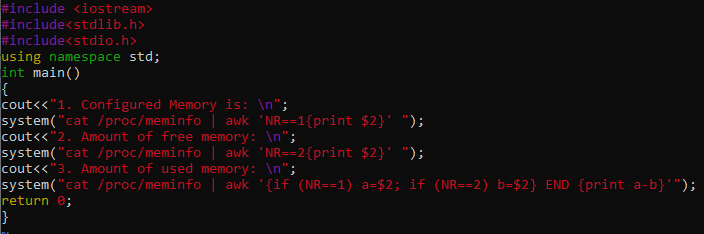
/proc/cpuinfo | awk ‘NR==3, NR==4{print}’ CPU SPACE

/proc/cpuinfo | awk ‘NR==2, NR==5{print}’ CPU TYPE

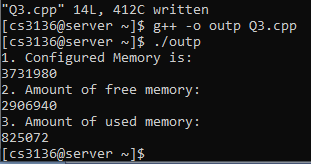
/proc/stat | awk ‘ {if (NR==1) {print $2}}’ CPU time spent

/proc/stat | awk ‘NR==2{print $4}’

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

****

**OUTPUT:**

****

#include<iostream>

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

Int main(){

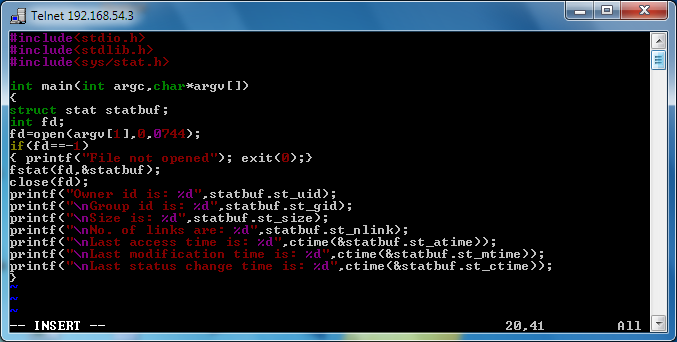
/proc/meminfo | awk ‘NR==1 {print $2}’ //configured memory

/proc/meminfo | awk ‘NR==2 {print $2}’ //free memory

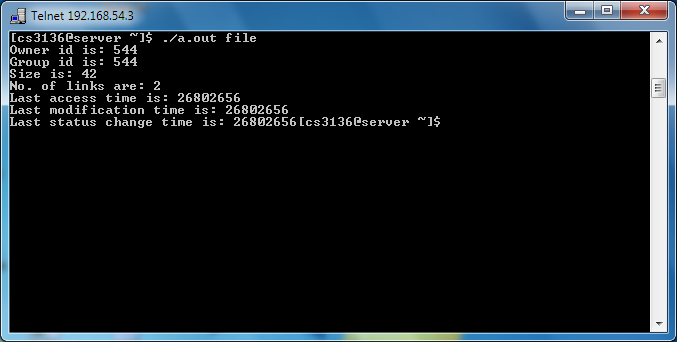
/proc/meminfo | awk ‘{if (NR==1) a=$2; if (NR==2) b=$2} END{print a-b}’ //used memory

}

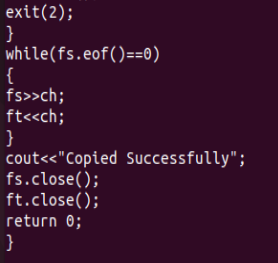
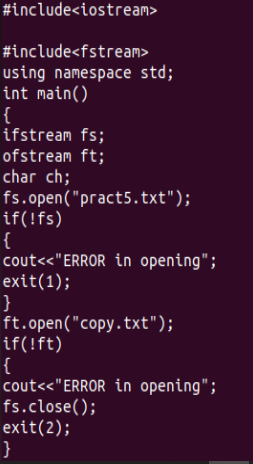
**Q4). Write a program to print file details including owner access permissions, file access time, where file name is given as argument.**



**OUTPUT:**



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



**OUTPUT:**



**prac5.txt**



**Copy.txt**



**Q6). Write a program to implements FCFS scheduling algorithm AT=0.**

#include<iostream>

using namespace std;

int main()

{

cout<<"\n This program implement FCFS scheduling";

cout<<"\n --------------------------------------";

int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;

cout<<"\nEnter total number of processes(maximum 20):";

cin>>n;

cout<<"\nEnter Process Burst Time\n";

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

{

cout<<"P["<<i+1<<"]:";

cin>>bt[i];

}

wt[0]=0; //waiting time for first process is 0

//calculating waiting time

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

{

wt[i]=0;

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

wt[i]+=bt[j];

}

cout<<"\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time";

//calculating turnaround time

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

{

tat[i]=bt[i]+wt[i];

avwt+=wt[i];

avtat+=tat[i];

cout<<"\nP["<<i+1<<"]"<<"\t\t"<<bt[i]<<"\t\t"<<wt[i]<<"\t\t"<<tat[i];

}

avwt/=i;

avtat/=i;

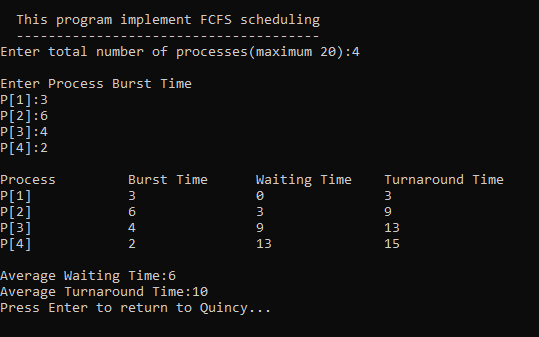
cout<<"\n\nAverage Waiting Time:"<<avwt;

cout<<"\nAverage Turnaround Time:"<<avtat;

return 0;

}

**OUTPUT:**

****

**Q7). Write a program to implements Round Robin scheduling algorithm AT=0.**

#include <iostream>

#include <vector>

/\*at = Arrival time,

bt = Burst time,

time\_quantum= Quantum time

tat = Turn around time,

wt = Waiting time\*/

using namespace std;

int main(){

int i,n,time,remain,temps=0,time\_quantum;

int wt=0,tat=0;

cout<<"\n This program implement Round Robin scheduling";

cout<<"\n ---------------------------------------------";

cout<<"\nEnter the total number of process: ";

cin>>n;

remain=n;

// assigning the number of process to remain variable

vector<int>at(n);

vector<int>bt(n);

vector<int>rt(n);

//dynamic array declaration using vector method of (STL)

//STL standard template library of C++

cout<<"Enter the Arrival time, Burst time for All the processes:"<<endl;

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

{

cin>>at[i];

cin>>bt[i];

rt[i]=bt[i];

}

cout<<"Enter the value of time QUANTUM:"<<endl;

cin>>time\_quantum;

cout<<"\n\nProcess\t:Turnaround Time:Waiting Time\n\n";

for(time=0,i=0;remain!=0;)

{

if(rt[i]<=time\_quantum && rt[i]>0)

{

time += rt[i];

//Addition using shorthand operators

rt[i]=0;

temps=1;

}

else if(rt[i]>0)

{

rt[i] -= time\_quantum;

//Subtraction using shorthand operators

time += time\_quantum;

//Addition using shorthand operators

}

if(rt[i]==0 && temps==1)

{

remain--;

//Desplaying the result of wating, turn around time:

printf("P{%d}\t:\t%d\t:\t%d\n",i+1,time-at[i],time-at[i]-bt[i]);

cout<<endl;

wt += time-at[i]-bt[i];

tat += time-at[i];

temps=0;

}

if(i == n-1)

i=0;

else if(at[i+1] <= time)

i++;

else

i=0;

}

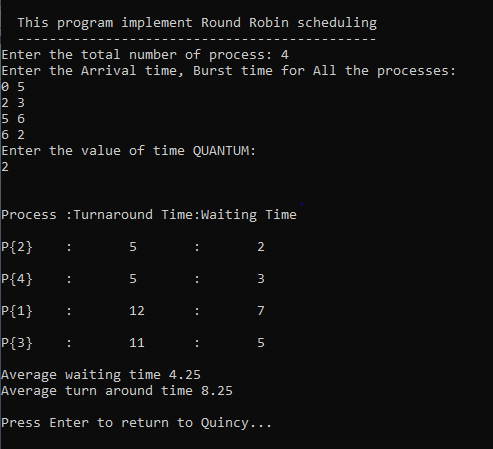
cout<<"Average waiting time "<<wt\*1.0/n<<endl;

cout<<"Average turn around time "<<tat\*1.0/n<<endl;;

return 0;

}

**OUTPUT:**

****

**Q8). Write a program to implement SJF scheduling algorithm AT=0.**

//Implementation fo SHORTEST JOB FIRST(Preemptive) Using C++

#include <iostream>

#include <algorithm>

#include <cstring>

using namespace std;

typedef struct proccess

{

int at,bt,ct,ta,wt,btt;

string pro\_id;

/\*

artime = Arrival time,

bt = Burst time,

ct = Completion time,

ta = Turn around time,

wt = Waiting time

\*/

}Schedule;

bool compare(Schedule a,Schedule b)

{

return a.at<b.at;

}

bool compare2(Schedule a,Schedule b)

{

return a.bt<b.bt;

}

int main()

{

Schedule pro[10];

//An array of Processes

int n,i,j,pcom;

//n = number of processes, i= iteration variable

cout<<"\n This program implement SJF scheduling";

cout<<"\n -------------------------------------";

cout<<"\nEnter the number of Process::";

cin>>n;

cout<<"Enter the Process id, arrival time, burst time:: \n";

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

{

cin>>pro[i].pro\_id;

cin>>pro[i].at;

cin>>pro[i].bt;

pro[i].btt=pro[i].bt;

}

sort(pro,pro+n,compare);

i=0;

pcom=0;

while(pcom<n)

{

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

{

if(pro[j].at>i)

break;

}

sort(pro,pro+j,compare2);

if(j>0)

{

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

{

if(pro[j].bt!=0)

break;

}

if(pro[j].at>i)

{

i=pro[j].at;

}

pro[j].ct=i+1;

pro[j].bt--;

}

i++;

pcom=0;

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

{

if(pro[j].bt==0)

pcom++;

}

}

cout<<"\nProID\tAt\tBT\tCT\tTAT\tWT\n";

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

{

pro[i].ta=pro[i].ct-pro[i].at;

pro[i].wt=pro[i].ta-pro[i].btt;

cout<<pro[i].pro\_id<<"\t"<<pro[i].at<<"\t"<<pro[i].btt<<"\t"<<pro[i].ct<<"\t"<<pro[i].ta<<"\t"<<pro[i].wt;

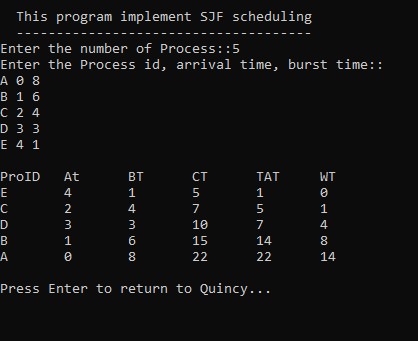
cout<<endl;

}

return 0;

}

**OUTPUT:**

****

**Q9). Write a program to implement non-preemptive priority scheduling algorithm AT=0.**

#include<iostream>

using namespace std;

int main()

{

cout<<"\n This program is for non preemptive priority based scheduling";

cout<<"\n ------------------------------------------------------------";

int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg\_wt,avg\_tat;

cout<<"\nEnter Total Number of Process:";

cin>>n;

cout<<"\nEnter Burst Time and Priority\n";

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

{

cout<<"\nP["<<i+1<<"]\n";

cout<<"Burst Time:";

cin>>bt[i];

cout<<"Priority:";

cin>>pr[i];

p[i]=i+1; //contains process number

}

//sorting burst time, priority and process number in ascending order using selection sort

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

{

pos=i;

for(j=i+1;j<n;j++)

{

if(pr[j]<pr[pos])

pos=j;

}

temp=pr[i];

pr[i]=pr[pos];

pr[pos]=temp;

temp=bt[i];

bt[i]=bt[pos];

bt[pos]=temp;

temp=p[i];

p[i]=p[pos];

p[pos]=temp;

}

wt[0]=0; //waiting time for first process is zero

//calculate waiting time

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

{

wt[i]=0;

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

wt[i]+=bt[j];

total+=wt[i];

}

avg\_wt=total/n; //average waiting time

total=0;

cout<<"\nProcess\t Burst Time \tWaiting Time\tTurnaround Time";

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

{

tat[i]=bt[i]+wt[i]; //calculate turnaround time

total+=tat[i];

cout<<"\nP["<<p[i]<<"]\t\t "<<bt[i]<<"\t\t "<<wt[i]<<"\t\t\t"<<tat[i];

}

avg\_tat=total/n; //average turnaround time

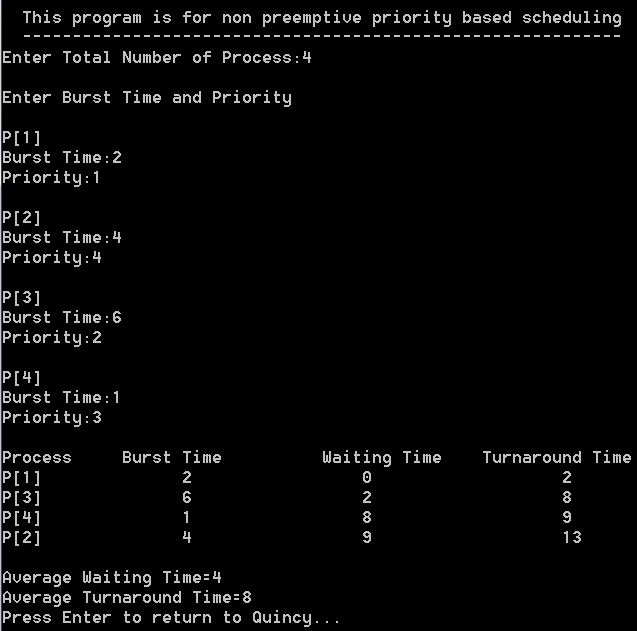
cout<<"\n\nAverage Waiting Time="<<avg\_wt;

cout<<"\nAverage Turnaround Time="<<avg\_tat;

return 0;

}

**OUTPUT:**

****

**Q10). Write a program to implement non-preemptive priority scheduling algorithm AT!=0.**

#include<iostream>

using namespace std;

int main()

{

int a[10],b[10],x[10];

int waiting[10],turnaround[10],completion[10],p[10];

int i,j,smallest,count=0,time,n;

double avg=0,tt=0,end;

cout<<"\n This program for preemptive priority based scheduling";

cout<<"\n -----------------------------------------------------";

cout<<"\nEnter the number of Processes: ";

cin>>n;

cout<<"\nEnter arrival time,burst time,priority of process: ";

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

{

cin>>a[i];

cin>>b[i];

cin>>p[i];

}

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

x[i]=b[i];

p[9]=-1;

for(time=0; count!=n; time++)

{

smallest=9;

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

{

if(a[i]<=time && p[i]>p[smallest] && b[i]>0 )

smallest=i;

}

b[smallest]--;

if(b[smallest]==0)

{

count++;

end=time+1;

completion[smallest] = end;

waiting[smallest] = end - a[smallest] - x[smallest];

turnaround[smallest] = end - a[smallest];

}

}

cout<<"Process"<<"\t"<< "BT"<<"\t"<<"AT" <<"\t"<<"WT" <<"\t"<<"TAT"<< "\t"<<"CT"<<"\t"<<"Priority"<<endl;

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

{

avg = avg + waiting[i];

tt = tt + turnaround[i];

cout<<"p"<<i+1<<"\t"<<x[i]<<"\t"<<a[i]<<"\t"<<waiting[i]<<"\t"<<turnaround[i]<<"\t"<<completion[i]<<"\t"<<p[i]<<endl;

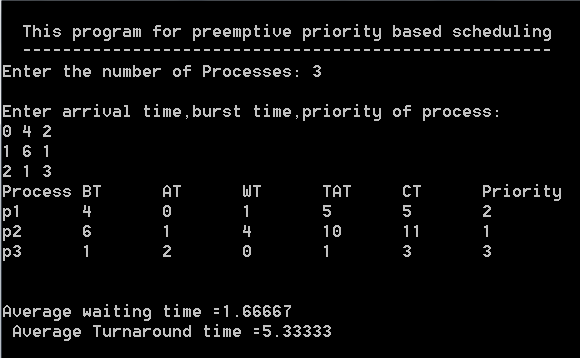
}

cout<<"\n\nAverage waiting time ="<<avg/n;

cout<<"\n Average Turnaround time ="<<tt/n<<endl;

}

**OUTPUT:**

****

**Q11). Write a program to implement SRJF scheduling algorithm AT!=0.**

#include<iostream>

using namespace std;

int main()

{

int a[10],b[10],x[10];

int waiting[10],turnaround[10],completion[10];

int i,j,smallest,count=0,time,n;

double avg=0,tt=0,end;

cout<<"\nEnter the number of Processes: "; //input

cin>>n;

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

{

cout<<"\nEnter arrival time of process: "; //input

cin>>a[i];

}

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

{

cout<<"\nEnter burst time of process: "; //input

cin>>b[i];

}

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

x[i]=b[i];

b[9]=9999;

for(time=0; count!=n; time++)

{

smallest=9;

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

{

if(a[i]<=time && b[i]<b[smallest] && b[i]>0 )

smallest=i;

}

b[smallest]--;

if(b[smallest]==0)

{

count++;

end=time+1;

completion[smallest] = end;

waiting[smallest] = end - a[smallest] - x[smallest];

turnaround[smallest] = end - a[smallest];

}

}

cout<<"Process"<<"\t"<< "burst-time"<<"\t"<<"arrival-time" <<"\t"<<"waiting-time" <<"\t"<<"turnaround-time"<< "\t"<<"completion-time"<<endl;

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

{

cout<<"p"<<i+1<<"\t\t"<<x[i]<<"\t\t"<<a[i]<<"\t\t"<<waiting[i]<<"\t\t"<<turnaround[i]<<"\t\t"<<completion[i]<<endl;

avg = avg + waiting[i];

tt = tt + turnaround[i];

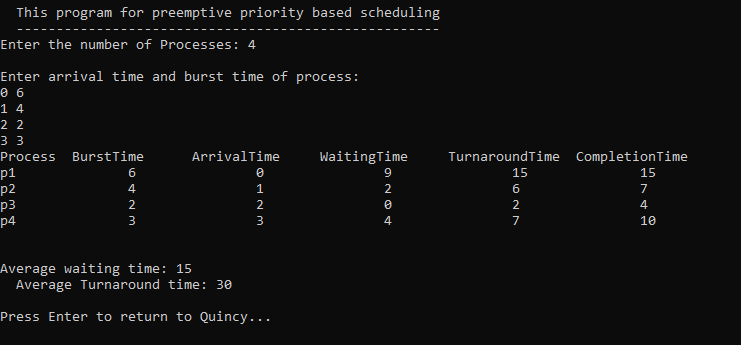
}

cout<<"\n\nAverage waiting time ="<<avg/n;

cout<<" Average Turnaround time ="<<tt/n<<endl;

}

**OUTPUT:**

****

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

#include<iostream>

#include<pthread.h>

using namespace std;

int g[2];

void \*sum(void \*a)

{

int \*arr;

arr = (int\*)a;

int n1, n2, s;

n1 = g[0];

n2 = g[1];

s = n1 + n2;

cout << "Sum=" << s;

return NULL;

}

int main()

{

cout << "Enter first number:" << endl;

cin >> g[0];

cout << "Enter second number:" << endl;

cin >> g[1];

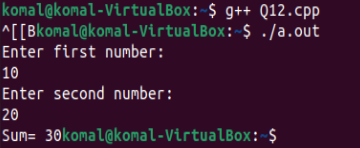
pthread\_t tid\_s;

pthread\_create(&tid\_s, NULL, sum, (void\*)&g);

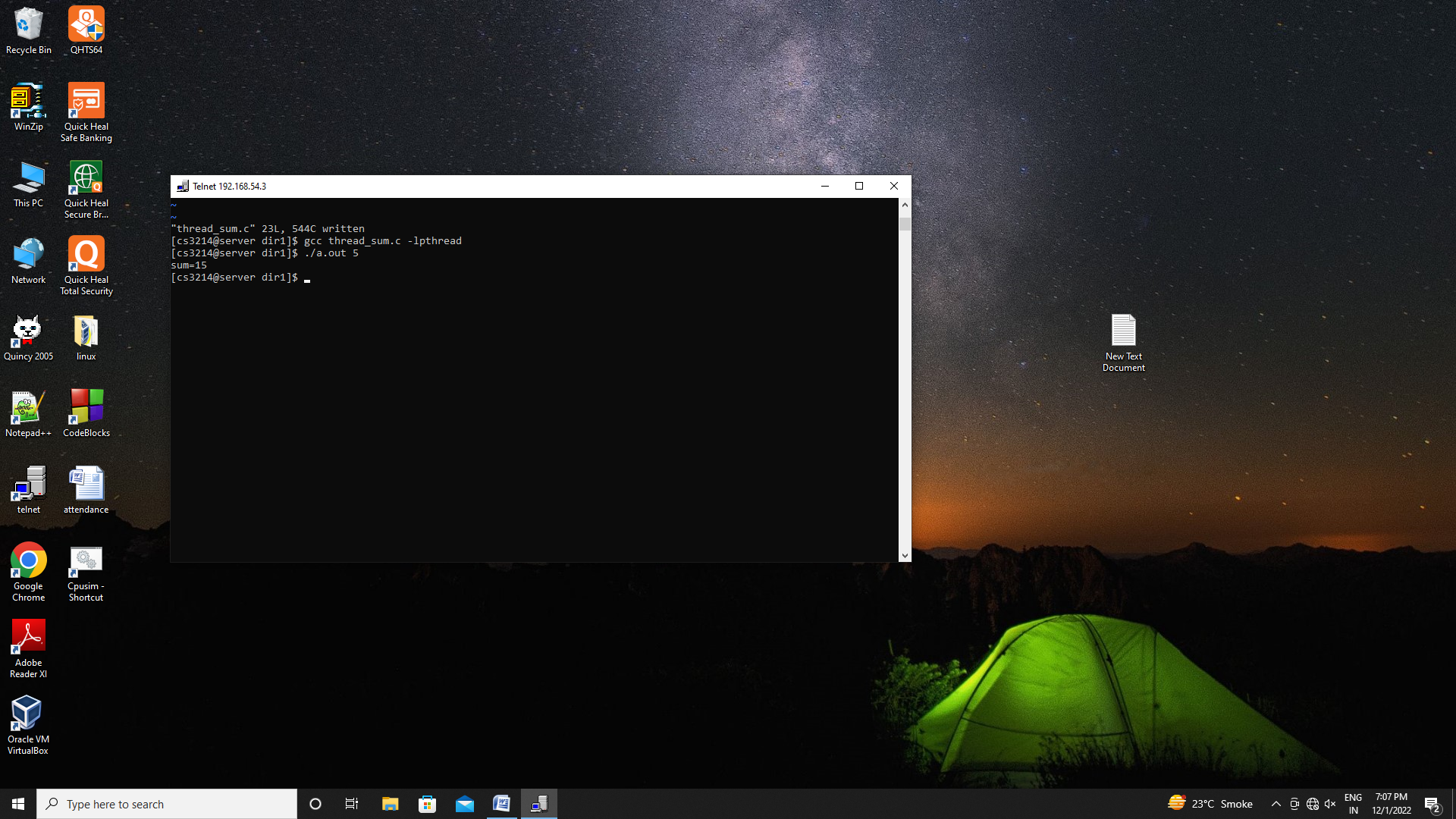
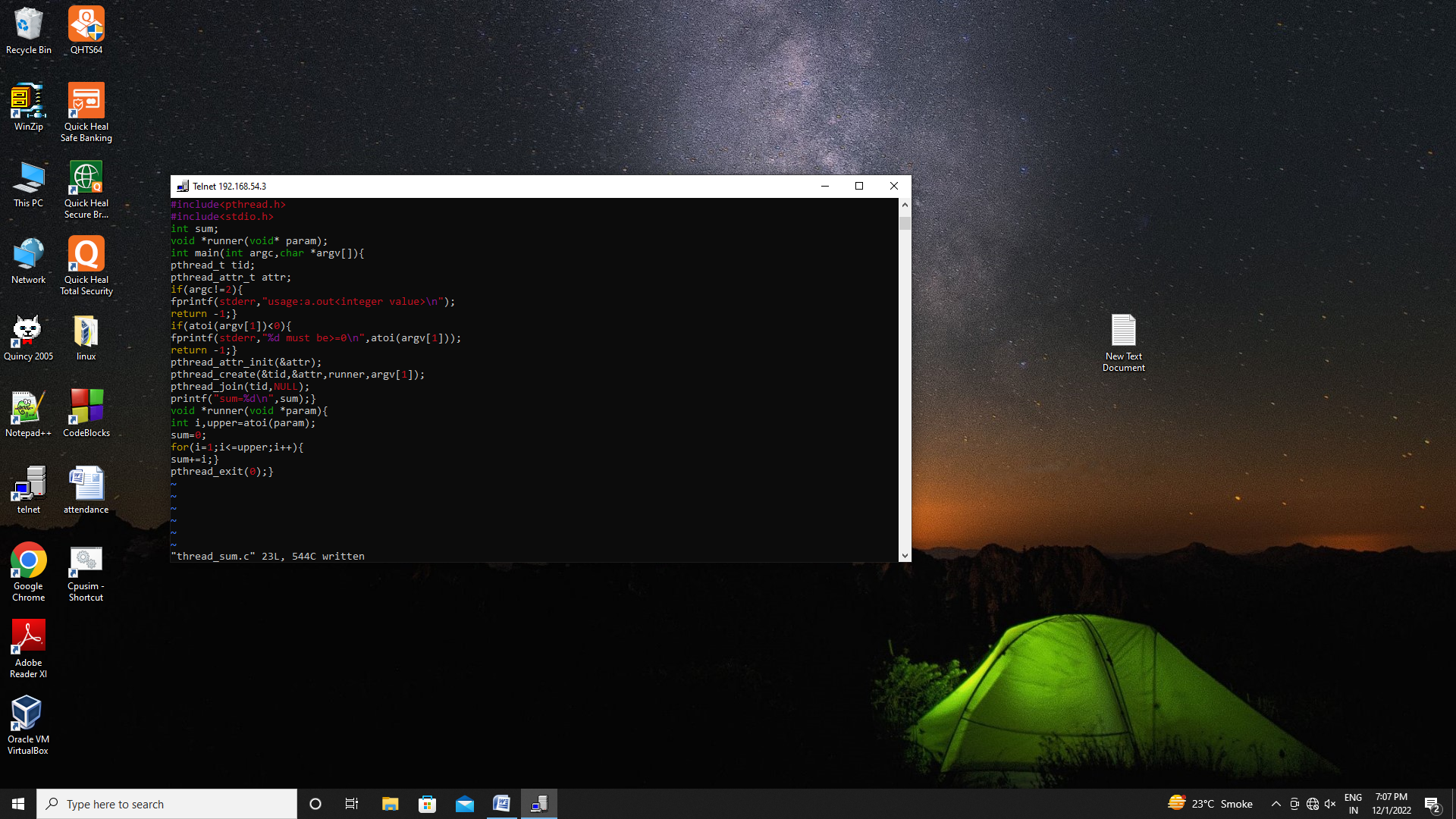
pthread\_join(tid\_s, NULL);

}

**OUTPUT:**



**OR**



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

#include<iostream>

#include<cstring>

using namespace std;

void firstfit(int p, int b, int np[], int nb[])

{

int flags[10], allocation[10], t, i, j;

for (i = 0;i < 10;i++)

{

flags[i] = 0;

allocation[i] = -1;

}

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

for (j = 0;j < b;j++)

if (flags[j] == 0 && nb[j] >= np[i])

{

allocation[j] = i;

flags[j] = 1;

break;

}

cout << "\nBlock no.\tBSize\t\tProcess no.\t\tPSize";

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

{

cout << "\n" << i + 1 << "\t\t" << nb[i] << "\t\t";

if (flags[i] == 1)

cout << allocation[i] + 1 << "\t\t\t" << np[allocation[i]];

else

cout << "Not Allocated" << endl;

}

}

void bestfit(int p, int b, int np[], int nb[])

{

int fragment[20], i, j, temp, lowest = 9999;

static int barray[20], parray[20];

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

{

for (j = 0;j < b;j++)

{

if (barray[j] != 1)

{

temp = nb[j] - np[i];

if (temp >= 0)

if (lowest > temp)

{

parray[i] = j;

lowest = temp;

}

}

}

fragment[i] = lowest;

barray[parray[i]] = 1;

lowest = 10000;

}

cout << "\nProcess No.\tProcess Size\tBlock No\tBlock Size\tFragment";

for (i = 0;i < p && parray[i] != 0;i++)

cout << "\n" << i + 1 << "\t\t" << np[i] << "\t\t" << parray[i] << "\t\t" << nb[parray[i]] << "\t\t" << fragment[i];

}

void worstfit(int p, int b, int np[],int nb[])

{

int f[10];

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

f[i]=nb[i];

int bs[10];

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

bs[i]=nb[i];

int allocation[10];

memset(allocation,-1,sizeof(allocation));

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

{

int wstIdx=-1;

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

{

if(nb[j]>=np[i])

{

if(wstIdx==-1)

wstIdx=j;

else if(nb[wstIdx]<nb[j])

wstIdx=j;

}

}

if(wstIdx!=-1)

{

allocation[i]=wstIdx;

nb[wstIdx]-=np[i];

}

}

cout<<"\nProcess No.\tProcess Size\tBlock No.\tBlock Size\tFragment\n";

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

{

cout<<" "<<i+1<<"\t\t"<<np[i]<<"\t\t";

if(allocation[i]!=-1)

{

int s=allocation[i]+1;

f[s-1]-=np[i];

cout<<s<<"\t\t"<<bs[s-1]<<"\t\t"<<f[s-1];

}

else

cout<<"Not Allowed";

cout<<endl;

}

}

void input(int&p, int&b, int np[], int nb[])

{

cout << "Enter no. of processes: \n";

cin >> p;

cout << "Enter no. of blocks: \n";

cin >> b;

cout << "Enter the block sizes: \n";

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

{

cin >> nb[i];

}

cout << "Enter the process sizes: \n";

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

{

cin >> np[i];

}

}

int main()

{

int p, b;

int np[50], nb[50];

char ch = 'y';

int choice;

cout<<"\tThis Program implement First Fit, Best Fit & Worst Fit\n";

cout<<"\t------------------------------------------------------\n";

while (ch == 'y' || ch == 'Y')

{

cout << "1. Input\n";

cout << "2. First Fit\n";

cout << "3. Best Fit\n";

cout << "4. Worst Fit\n";

cout << "ENTER YOUR CHOICE: ";

cin >> choice;

switch (choice)

{

case 1:

input(p, b, np, nb);

break;

case 2:

firstfit(p, b, np, nb);

break;

case 3:

bestfit(p, b, np, nb);

break;

case 4:

worstfit(p, b, np, nb);

break;

default:

cout << "Error in input\n";

break;

}

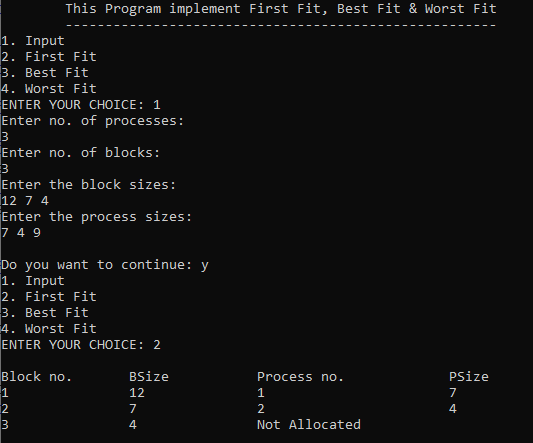
cout << "\nDo you want to continue: ";

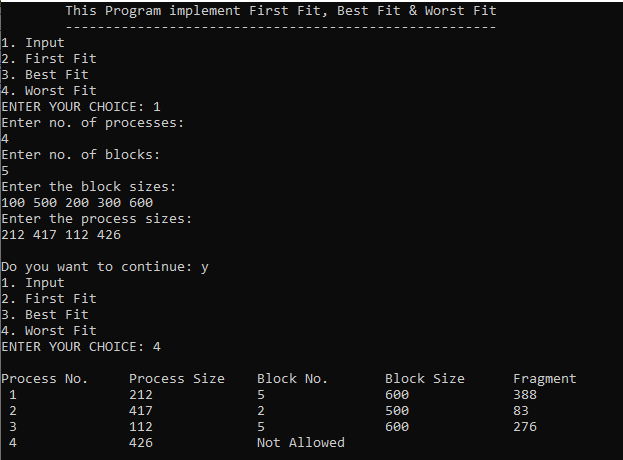
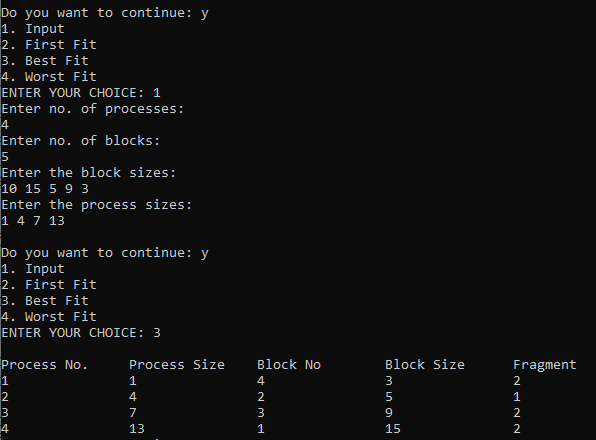
cin >> ch;

}

}

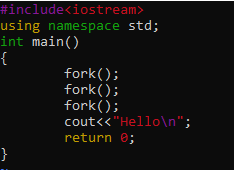
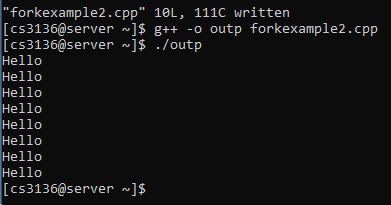
**OUTPUT:**

****

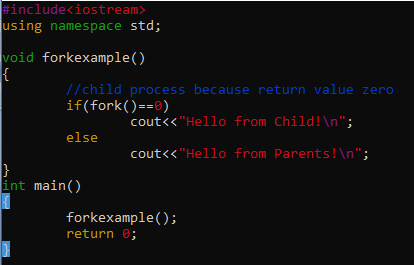
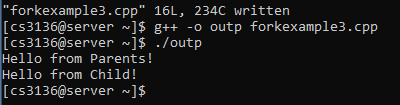
****

**EXTRA:**

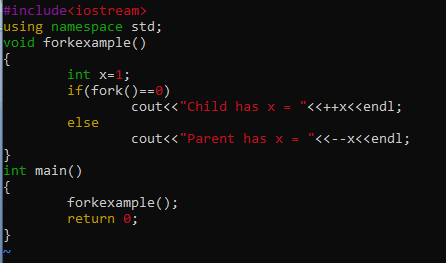
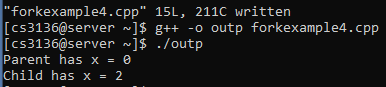
**1.**

** **

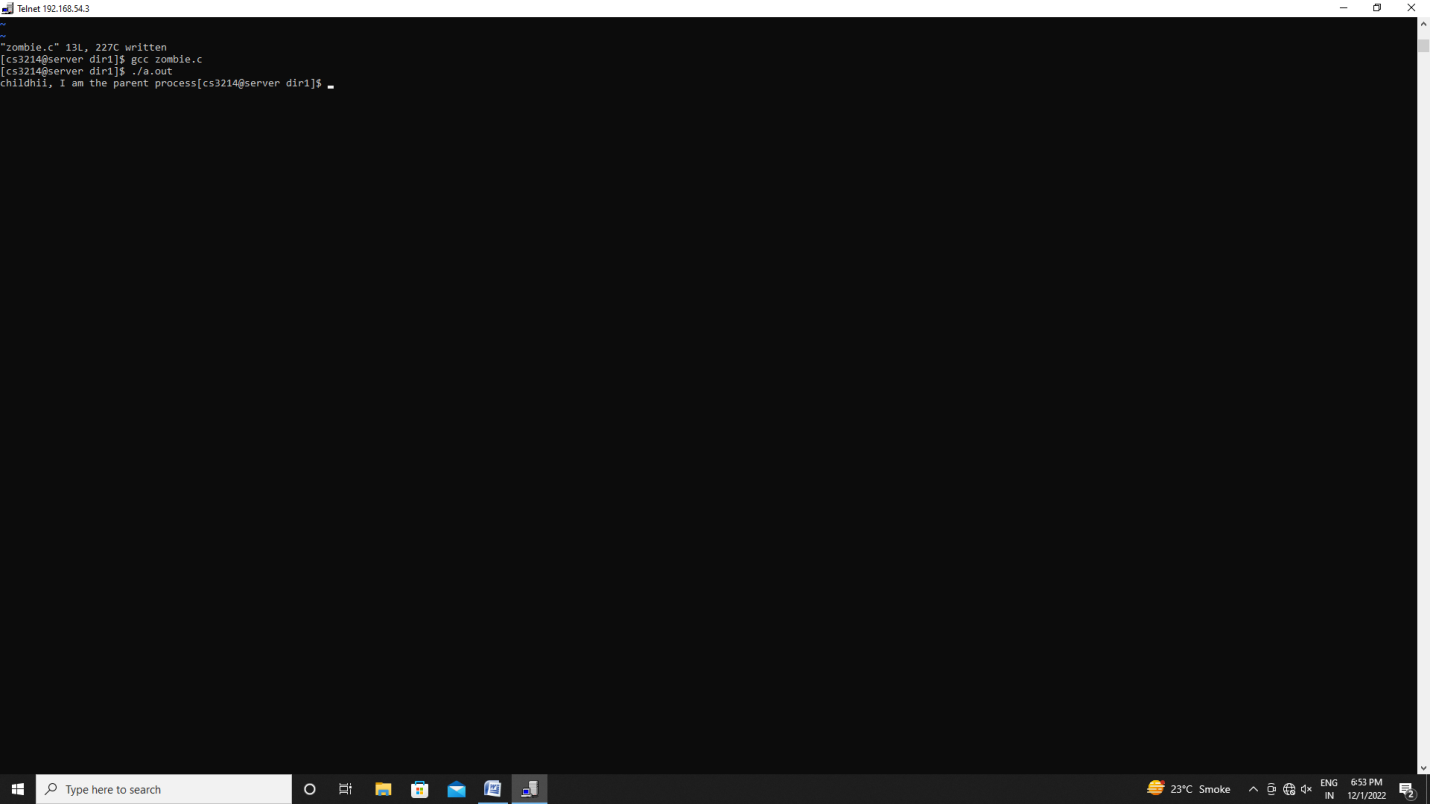
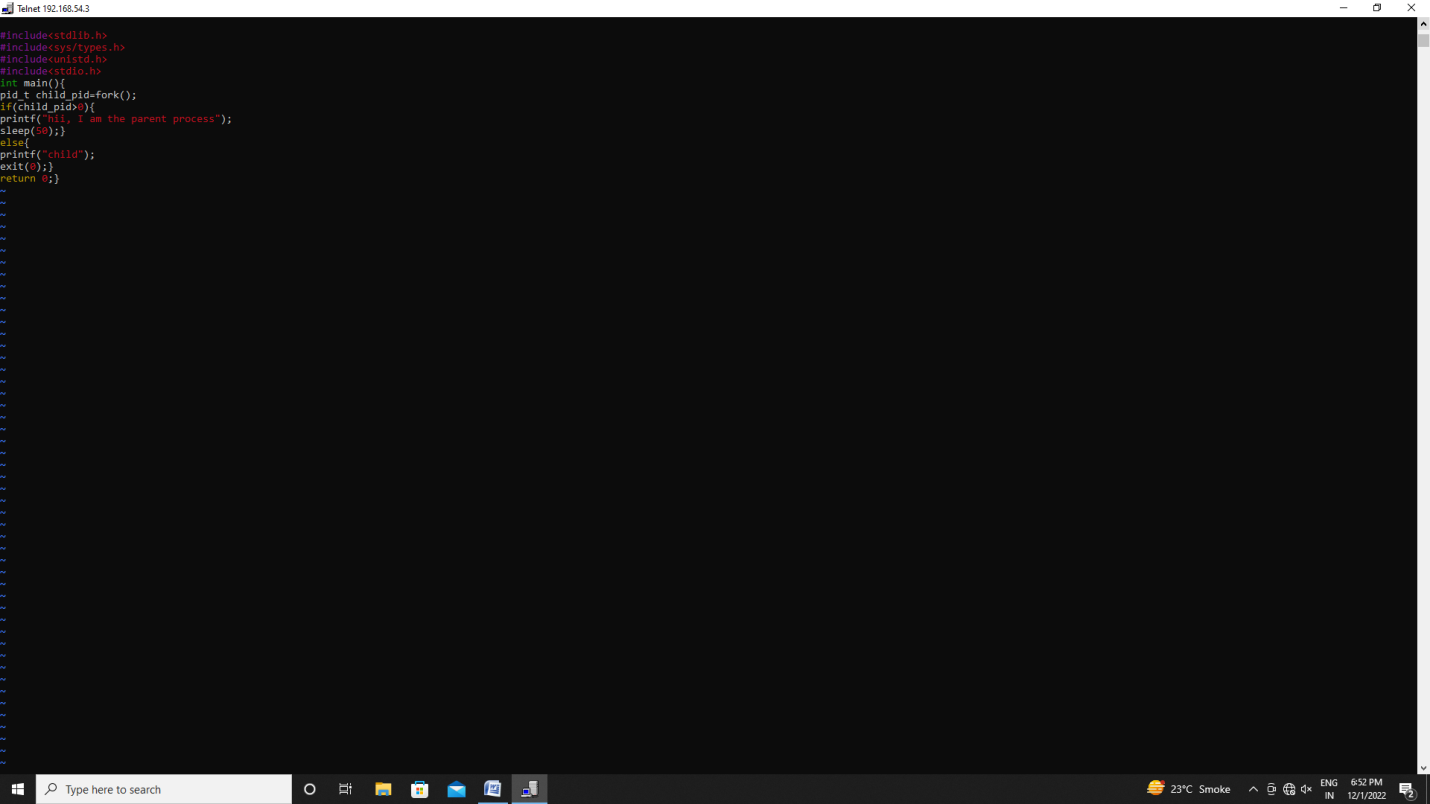
**2.**

** **

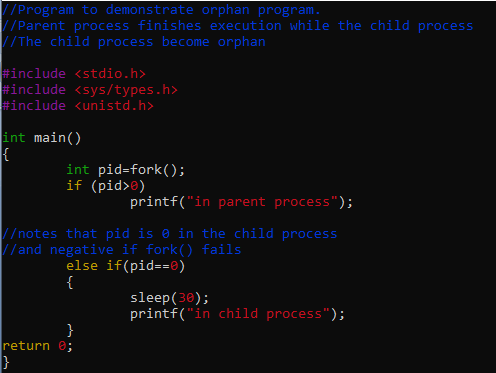
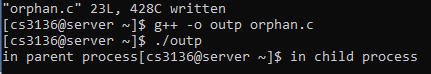
**3.**

** **

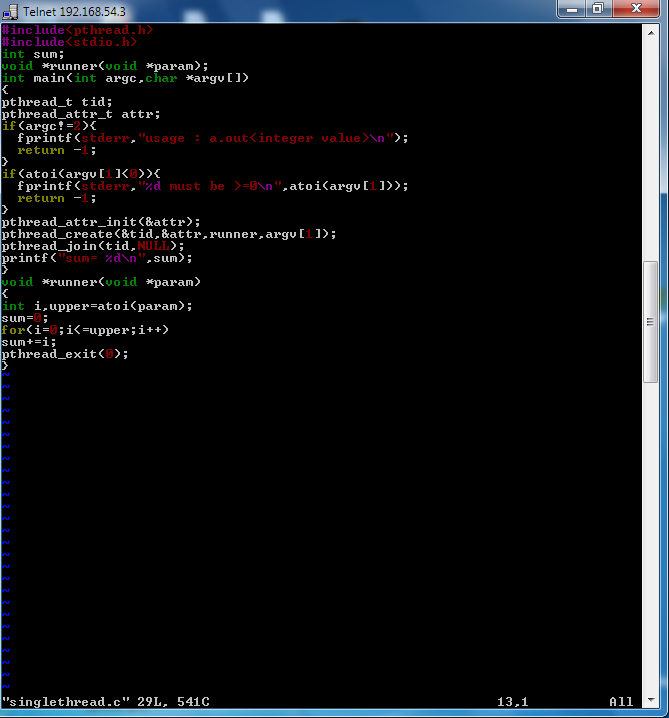
**4. Zombie**



**5. Orphan**

** **

**6. Single thread**



**7. Multithread**

