*Operating Systems*

Practical File

*Submitted By:*

Name: Mayank Kharab Roll No: 20201437

Course: B.Sc. (Hons) Computer Science

*Index*

1. Write a program (using fork() and/or exec() commands) where parent and child execute:
   1. same program, same code.
   2. same program, different code.
   3. before terminating, the parent waits for the child to finish its task.
2. Write a program to report behavior of Linux kernel including kernel version, CPU type and model. (CPU information)
3. Write a program to report behavior of Linux kernel including information on 19 configured memory, amount of free and used memory. (memory information)
4. Write a program to print file details including owner access permissions, file access time, where file name is given as argument.
5. Write a program to copy files using system calls.
6. Write a program to implement FCFS scheduling algorithm.
7. Write a program to implement Round Robin scheduling algorithm.
8. Write a program to implement SJF scheduling algorithm.
9. Write a program to implement non-preemptive priority based scheduling algorithm.
10. Write a program to implement preemptive priority based scheduling algorithm.
11. Write a program to implement SJRF scheduling algorithm.
12. Write a program to calculate sum of n numbers using thread library.
13. Write a program to implement first-fit, best-fit and worst-fit allocation strategies.

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

* 1. same program, same code.
  2. same program, different code.
  3. before terminating, the parent waits for the child to finish its task.

# A1.

**(a)** #include<iostream> #include<unistd.h> using namespace std; int main()

{

int x,y;

x=fork();

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

y=fork();

cout<<"\n y = "<<y;

cout<<"\n Bye"<<endl;

return 0;

}

# OUTPUT:



**(b)**

#include<iostream> #include<sys/wait.h> #include<unistd.h> using namespace std; int main(){

int x; x=fork();

if(x==0){

execlp("/home/kali/Desktop/practical5","practical5",NULL);

}

else{

wait(NULL);

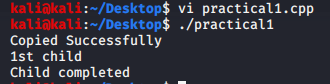
cout<<"Child completed\n";

}

return(0);

}

# OUTPUT:



**(c)**

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

int main()

{

pid\_t id = fork();

// child process if(id == 0)

{

cout<<"\n Child Terminated";

}

// parent process else

{

wait(NULL);

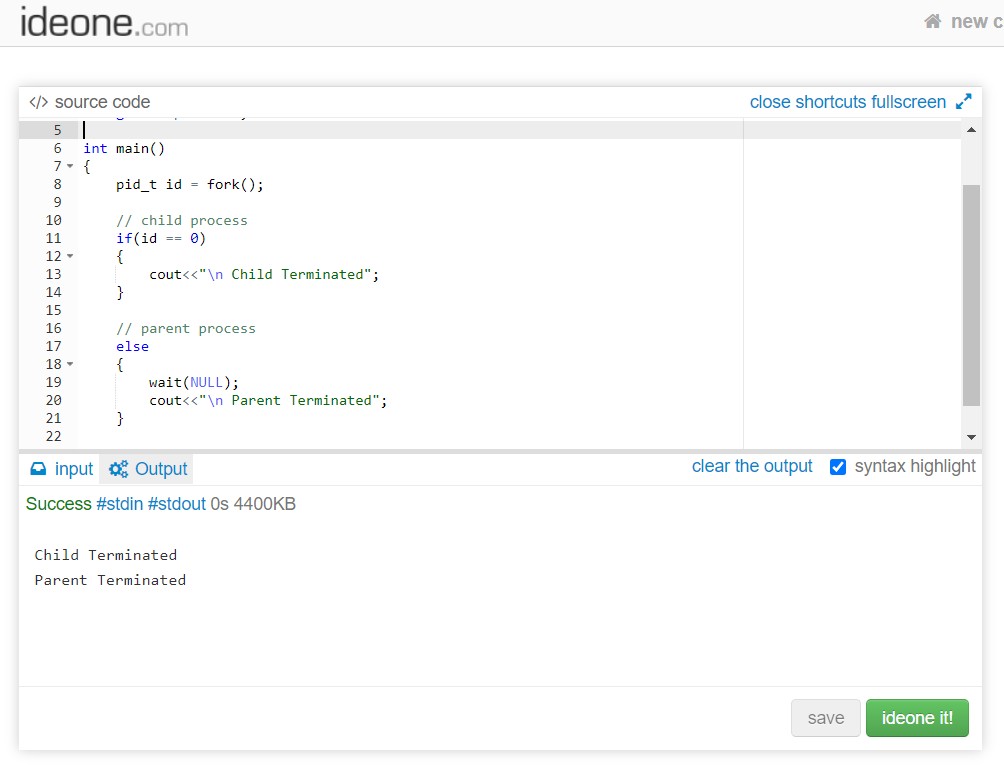
cout<<"\n Parent Terminated";

}

return 0;

}

# OUTPUT:



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

# A2.

#include <iostream> #include <sys/sysinfo.h> #include<unistd.h> #include<string.h> using namespace std;

void defVersion()

{

cout << "\nCPU type and model: \n";

system("grep 'model name' /proc/cpuinfo | awk '{print $4, $5, $6, $7, $8}'");

cout << "Kernel version: \n";

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

cout << "Amount of time since the system was last booted: \n"; system("cat /proc/uptime | awk '{print $1\n}'");

}

void secVersion()

{

cout << "\nPRINT IN AWK\n";

cout << "Amount of time user has spent in:\nUser mode:\n";

system("grep 'cpu' /proc/stat | awk 'NR == 1 {print $2}'"); cout << "System mode: \n";

system("grep 'cpu' /proc/stat | awk ' NR == 1 {print $4}'"); cout << "Idle: \n";

system("grep 'cpu' /proc/stat | awk 'NR == 1 {print $5}'");

cout << "\nSHELL SCRIPT\n";

cout << "Amount of time user has spent in:\n"; system("Q2SecVersion");

cout << "\nNumber of disk requests:\n"; system("grep 'intr' /proc/stat | awk '{print $17}'");

cout << "Number of context switches: \n"; system("grep 'ctxt' /proc/stat | awk '{print $2}'");

cout << "Time at which system was booted:\n"; system("grep 'btime' /proc/stat | awk '{print $2}'");

cout << "Time at which system was booted:\n"; system("Q2EpochToLocal");

cout << "Number of processes created:\n";

}

void thirdVersion(int a, int b)

{

struct sysinfo sInfo;

if (sysinfo(&sInfo) < 0)

{

cout << "\nError in finding information about the system"; exit(1);

}

"KB";;

cout << "\nAmount of memory configured into system: " << sInfo.totalram/1024 <<

cout << "\nAmount of memory currently available: " << sInfo.freeram/1024 << "KB"; cout << "\nLoad averages: \n";

for (int i = 1; i <= b/a; i++)

{

cout << i << ". " << sInfo.loads[0] << "\n"; if (i == b/a) break;

sleep(a);

if (sysinfo(&sInfo) < 0)

{

cout << "\nError in finding information about the system"; exit(1);

}

}

}

int main(int ac, char \*av[])

{

switch (ac){

case 1: defVersion(); break;

case 2: if (strcmp(av[1], "-s") == 0)

{

}

else

break;

defVersion(); secVersion();

cout << "\nInvalid option\n";

case 4: if (strcmp(av[1], "-l") == 0)

{

}

else

break;

defVersion(); secVersion();

thirdVersion(atoi(av[2]), atoi(av[3]));

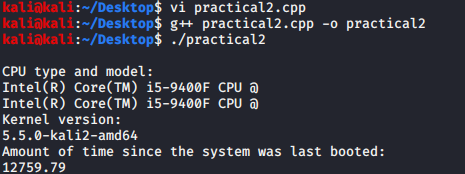
cout << "\nInvalid option\n";

default: cout << "\nInvalid number of arguments\n";

}

}

# OUTPUT:



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

# A3.

#include <iostream> #include <sys/sysinfo.h> #include<unistd.h> #include<string.h> using namespace std;

void defVersion()

{

cout << "\nCPU type and model: \n";

system("grep 'model name' /proc/cpuinfo | awk '{print $4, $5, $6, $7, $8}'");

cout << "Kernel version: \n";

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

cout << "Amount of time since the system was last booted: \n"; system("cat /proc/uptime | awk '{print $1\n}'");

cout<<"The configured memory is; \n";

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

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

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

b}'\n");

}

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

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

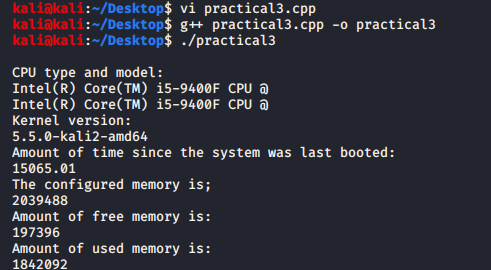
int main(int ac, char \*av[])

{

defVersion(); return 0;

}

# OUTPUT:



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

# A4.

#include<sys/types.h> #include<iostream> #include<sys/stat.h> using namespace std;

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

{

struct stat buf; if(argc<2)

cout<<"\nError : Parameter too low";

else

{

stat(argv[1],&buf);

cout<<"\nFile Name : "<<argv[1]; cout<<"\nUser id : "<<buf.st\_uid; if(S\_ISREG(buf.st\_mode))

cout<<"\n"<<argv[1]<<" is a regular file";

else

cout<<"\n"<<argv[1]<<" is a directory";

cout<<"\npermissions :-"; if(buf.st\_mode & 0400)

cout<<"\nOwner has read permission";

if(buf.st\_mode & 0200)

cout<<"\nOwner has write permission"; if(buf.st\_mode & 0100)

cout<<"\nOwner has execute permission"; if(buf.st\_mode & 0040)

cout<<"\nGroup has read permission"; if(buf.st\_mode & 0020)

cout<<"\nGroup has write permission"; if(buf.st\_mode & 0010)

cout<<"\nGroup has execute permission"; if(buf.st\_mode & 0004)

cout<<"\nOthers has read permission"; if(buf.st\_mode & 0002)

cout<<"\nOthers has write permission"; if(buf.st\_mode & 0001)

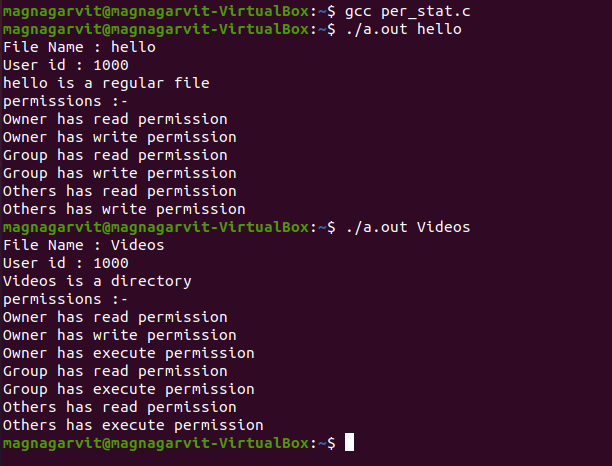
cout<<"\nOthers has execute permission";

}

return 0;

}

# OUTPUT:



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

# A5.

#include<iostream> #include<fstream> using namespace std;

int main()

{

ifstream fs; ofstream ft; char ch;

fs.open("practical5.txt");

if(!fs){

cout<<"ERROR in opening"; exit(1);

}

ft.open("copy.txt");

if(!ft){

cout<<"ERROR in opening"; fs.close();

exit(2); }

while(fs.eof()==0){

fs>>ch; ft<<ch;

}

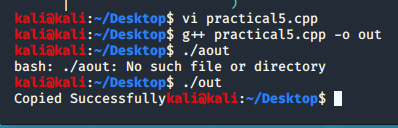
cout<<"Copied Successfully"; fs.close();

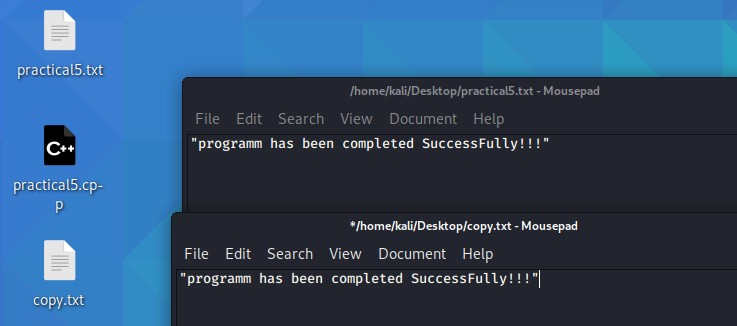
ft.close();

return 0;

}

# OUTPUT:





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

# A6.

#include<iostream> using namespace std;

// Function to find the waiting time for all

// processes

void findWaitingTime(int processes[], int n,

int bt[], int wt[],int at[])

{

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

// calculating waiting time for (int i = 1; i < n ; i++ )

wt[i] = bt[i-1] + wt[i-1]-at[i] ;

}

// Function to calculate turn around time

void findTurnAroundTime( int processes[], int n, int bt[], int wt[], int tat[],int at[])

{

// calculating turnaround time by

// bt[i] - at[i]

for (int i = 0; i < n ; i++) tat[i] = bt[i]+wt[i] -at[i];

}

//Function to calculate average time

void findavgTime( int processes[], int n, int bt[],int at[])

{

int wt[n], tat[n], total\_wt = 0, total\_tat = 0;

//arranging places according to the arrival\_time for(int i=0;i<n-1;i++)

{

for(int j=0;j<n+1;j++)

{

if(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;

}

}

}

//Function to find waiting time of all processes findWaitingTime(processes, n, bt, wt,at);

//Function to find turn around time for all processes findTurnAroundTime(processes, n, bt, wt, tat,at);

//Display processes along with all details

cout << "Processes "<<"Arrival time"<< " Burst time "

<< " Waiting time " << " Turn around time\n";

// Calculate total waiting time and total turn

// around time

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

{

total\_wt = total\_wt + wt[i]; total\_tat = total\_tat + tat[i];

cout << " " << processes[i] << "\t\t"<<at[i]<<"\t\t" << bt[i] <<"\t "

<< wt[i] <<"\t\t " << tat[i] <<endl;

}

cout << "Average waiting time = "

<< (float)total\_wt / (float)n;

cout << "\nAverage turn around time = "

<< (float)total\_tat / (float)n;

}

// Driver code

int main()

{

int n;

cout<<"enter no of process u want :-"; cin>>n;

int processes[n]; int burst\_time[n]; int arrival\_time[n]; for(int i=0;i<n;i++){

processes[i]=i+1; //process id's

cout<<"Enter Burst\_time of process no "<<i+1<<" :"; //burst\_time cin>>burst\_time[i];

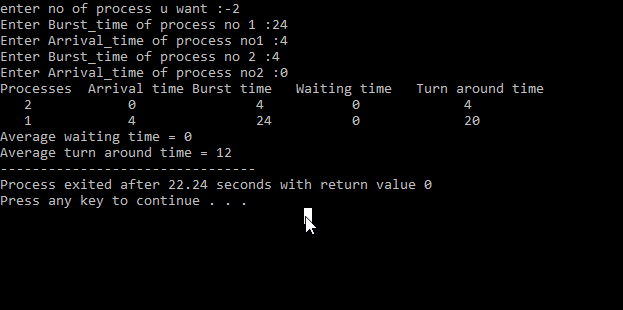
cout<<"Enter Arrival\_time of process no"<<i+1<<" :" ; //arrival\_time cin>>arrival\_time[i];

}

findavgTime(processes, n, burst\_time,arrival\_time); return 0;

}

# OUTPUT:



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

# A7.

#include<iostream> using namespace std;

// Function to find the waiting time for all

// processes

void findWaitingTime(int processes[], int n, int bt[], int wt[], int quantum)

{

// Make a copy of burst times bt[] to store remaining

// burst times. int rem\_bt[n];

for (int i = 0 ; i < n ; i++) rem\_bt[i] = bt[i];

int t = 0;

// Keep traversing processes in round robin manner

// until all of them are not done. while (1)

{

bool done = true;

// Traverse all processes one by one for (int i = 0 ; i < n; i++)

{

// If burst time of a process is greater than 0

// then only need to process further if (rem\_bt[i] > 0)

{

done = false;

if (rem\_bt[i] > quantum)

{

// Increase the value of t i.e. shows

// how much time a process has been processed t += quantum;

// Decrease the burst\_time of current process

// by quantum rem\_bt[i] -= quantum;

}

// If burst time is smaller than or equal to

// quantum. Last cycle for this process else

{

// Increase the value of t i.e. shows

// how much time a process has been processed t = t + rem\_bt[i];

// Waiting time is current time minus time

// used by this process wt[i] = t - bt[i];

// As the process gets fully executed

// make its remaining burst time = 0 rem\_bt[i] = 0;

}

}

}

if (done == true) break;

}

}

void findTurnAroundTime(int processes[], int n,

int bt[], int wt[], int tat[])

{

for (int i = 0; i < n ; i++) tat[i] = bt[i] + wt[i];

}

// Function to calculate average time

void findavgTime(int processes[], int n, int bt[],

int quantum)

{

int wt[n], tat[n], total\_wt = 0, total\_tat = 0;

// Function to find waiting time of all processes

findWaitingTime(processes, n, bt, wt, quantum);

// Function to find turn around time for all processes findTurnAroundTime(processes, n, bt, wt, tat);

cout << "Processes "<< " Burst time "

<< " Waiting time " << " Turn around time\n";

// Calculate total waiting time and total turn

// around time

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

{

total\_wt = total\_wt + wt[i]; total\_tat = total\_tat + tat[i];

cout << " " << i+1 << "\t\t" << bt[i] <<"\t "

<< wt[i] <<"\t\t " << tat[i] <<endl;

}

cout << "Average waiting time = "

<< (float)total\_wt / (float)n;

cout << "\nAverage turn around time = "

<< (float)total\_tat / (float)n;

}

int main()

{

int n;

cout<<"enter the number of preocesses"<<endl; cin>>n;

int processes[n];

cout<<"enter the processes"<<endl; for(int i=0;i<n;i++)

{

cin>>processes[i];

}

cout<<"enter burst time"<<endl; int burst\_time[n];

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

{

cin>>burst\_time[i];

}

// Time quantum int quantum ;

cout<<"enter the value of quantum time"<<endl; cin>>quantum;

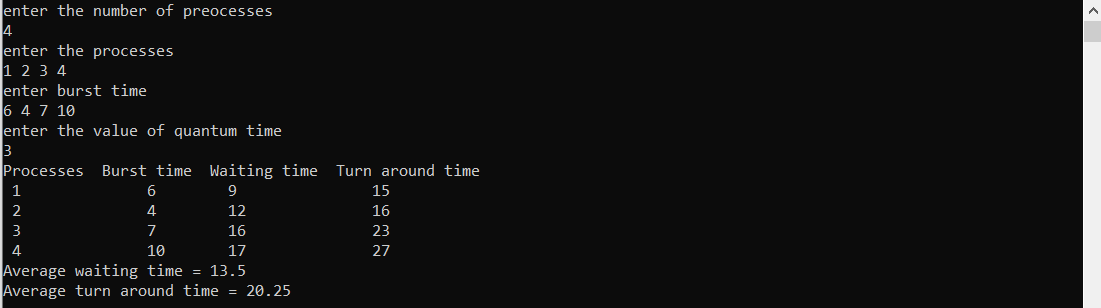
findavgTime(processes, n, burst\_time, quantum);

getchar(); getchar();

return 0;

}

# OUTPUT:



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

# A8.

#include<stdio.h> #include<iostream> using namespace std; int main()

{

int i,n,proc\_id[10],min,k=1,btime=0;

int bt[10],temp,j,at[10],wt[10],tt[10],a[10]; int t=0,m=0;

cout<<"Enter the number of processes: "; cin>>n;

cout<<"\n Enter the arrival time: \n"; for(i=0;i<n;i++)

{

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

cin>>at[i]; cout<<endl;

}

cout<<"Enter the burst time:"<<endl; for(i=0;i<n;i++)

{

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

cin>>bt[i]; cout<<endl; proc\_id[i]=i+1;

}

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

{

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

{

if(at[i]<at[j])

{

temp=proc\_id[j]; proc\_id[j]=proc\_id[i]; proc\_id[i]=temp;

temp=at[j]; at[j]=at[i]; at[i]=temp;

temp=bt[j]; bt[j]=bt[i]; bt[i]=temp;

}

}

}

t=0;

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

{

for(j=0;j<(n-1)-i;j++)

{

if(at[j]==at[j+1])

{

if(bt[j]>bt[j+1])

{ temp=at[j];

at[j]=at[j+1]; at[j+1]=temp;

t=bt[j]; bt[j]=bt[j+1]; bt[j+1]=t;

m=proc\_id[j]; proc\_id[j]=proc\_id[j+1];

proc\_id[j+1]=m;

}

}

}

}

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

{

btime=btime+bt[j]; min=bt[k]; for(i=k;i<n;i++)

{

if (btime>=at[i] &&bt[i]<min)

{

temp=proc\_id[k]; proc\_id[k]=proc\_id[i]; proc\_id[i]=temp;

temp=at[k]; at[k]=at[i]; at[i]=temp;

}

} k++;

temp=bt[k]; bt[k]=bt[i]; bt[i]=temp;

}

wt[0]=0;

a[0]=0;

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

{

a[i]=a[i-1]+bt[i-1];

wt[i]=a[i]-at[i];

}

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

{

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

}

cout<<endl; cout<<"Process\t"<<"Burst\t"<<"Arrival\t"<<"Waiting\t"<<"Turn-around" ; for(i=0;i<n;i++)

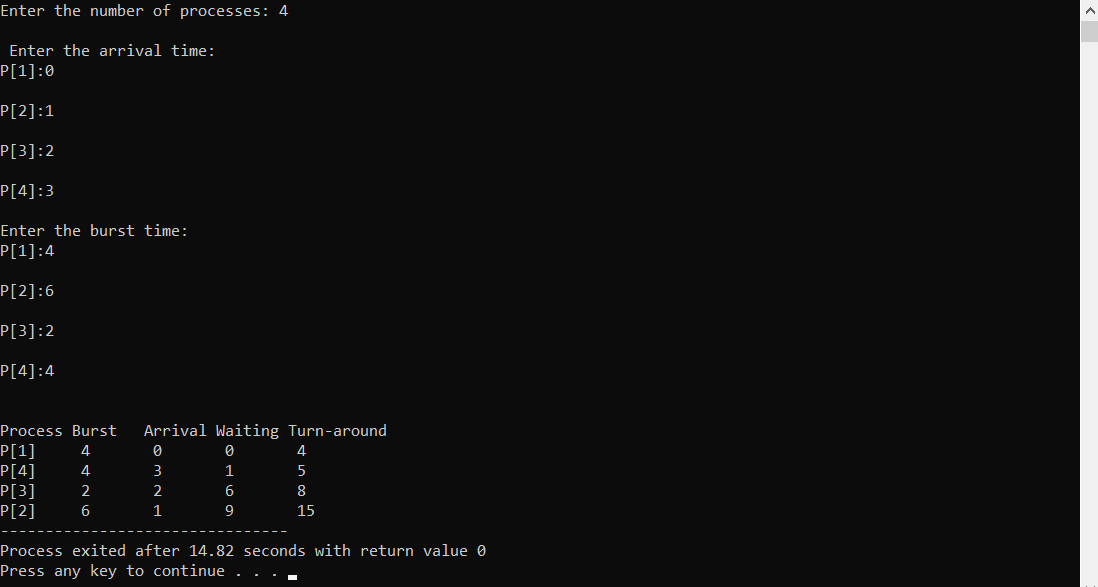
{

cout<<"\n"<<"P["<<proc\_id[i]<<"]"<<" \t "<<bt[i]<<" \t "<<at[i]<<" \t "<<wt[i]<<" \t " <<tt[i];

}

}

# OUTPUT:



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

# A9.

#include<iostream> using namespace std;

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

void findTurnAroundTime(int i,int at[],int tat[],int qt)

{

// calculating turnaround time by

//complete at- start at

tat[i] = qt-at[i]; comp[i]=qt;

//cout<<"\nct"<<comp[i]<<" 3";

}

void findWaitingTime(int tat[], int n, int bt[], int wt[])

{ //calculating waiting time by

//turn around time -burst btime 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;

findTurnAroundTime(i,at,tat,ct);

cout<<"\n ~~~~~~\n"; cout<<j<<" |"<<" P"<<processes[i]<<" | "<< ct

<<"\n";//<<"\t"<<at[j]<<"\t"<<quantum1<<"\n"; cout<<" ~~~~~~";

}

}

}

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

{

int dup\_bt[n],tat[n],wt[n];

//arranging places according to the PRIORITY 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;

}

}

}

//show

cout << "\nProcesses "<< " Arrival time "<< " Burst time "<<"Priotrity \n"; for (int i=0; i<n; i++)

{

cout << " " << processes[i] <<"\t\t"<<at[i] <<"\t\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])

{

l=ct;

int l=0;

ct=ct+dup\_bt[j];

dup\_bt[j]=0; findTurnAroundTime(j,at,tat,ct);

cout<<"\n ~~~~~~\n";

cout<<l<<" |"<<" P"<<processes[j]<<" | "<< ct

<<"\n";//<<"\t"<<at[j]<<"\t"<<quantum1<<"\n";

cout<<" ~~~~~~"; j=n;

//cout<<"\n 5";

}

}

}

}

waitingQueue(dup\_bt,n,processes,at,tat); findWaitingTime(tat,n,bt,wt);

cout<< "\nProcesses "<< " Burst time "<<"\tArrival time "<<"turn around Time"<<"

Complition Time"

<<"waiting Time"<<"\n";

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

cout<<processes[i]<<"\t\t "<<bt[i]<<"\t\t "<<at[i]<<"\t\t "<<tat[i]<<"\t\t "

<<comp[i]<<"\t\t "<<wt[i]<<"\n";

}

//main code int main()

{

int n;

cout<<"enter no of process u want :-"; cin>>n;

int processes[n]; int burst\_time[n]; int arrival\_time[n]; int priority[n];

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

processes[i]=i+1; //process id's

cout<<"Enter Burst\_time of process no "<<i+1<<" :"; //burst\_time cin>>burst\_time[i];

cout<<"Enter Arrival\_time of process no"<<i+1<<" :"; cin>>arrival\_time[i];

cout<<"Enter Priority of process no"<<i+1<<" :"; cin>>priority[i];

}

cout<<"input :- \n";

cout << "Processes "<< " Arrival time "<< " Burst time "<<"Priotrity \n"; for (int i=0; i<n; i++)

{

cout << " " << processes[i] <<"\t\t"<<arrival\_time[i] <<"\t\t"<< burst\_time[i] <<"\t "

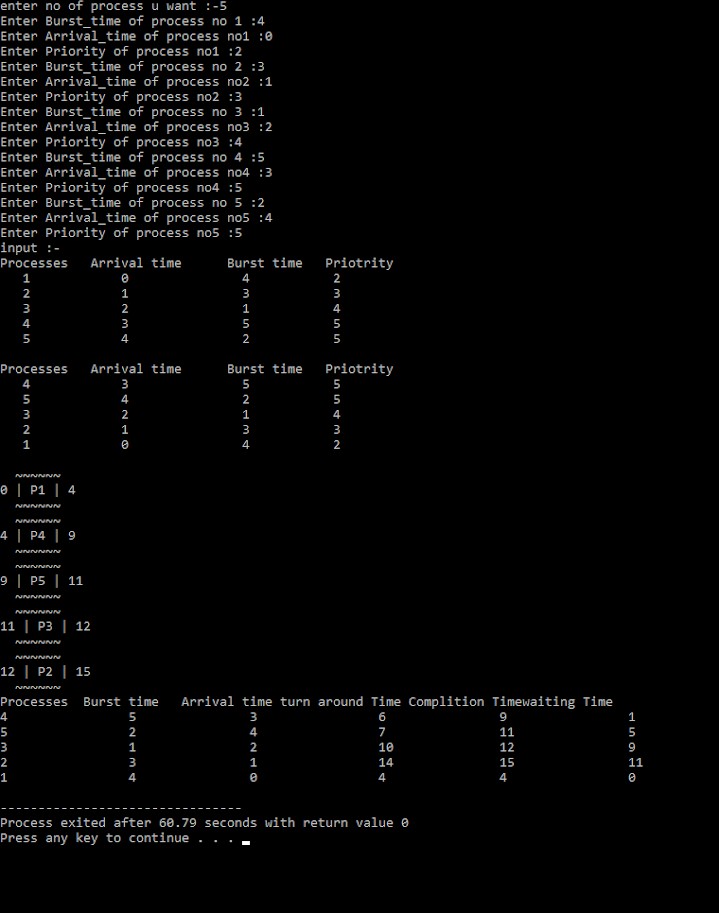
<< priority[i] <<"\t "<<endl;

}

ReadyQueue(processes,burst\_time,arrival\_time,n,priority); return(0);

}

# OUTPUT:



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

# A10.

#include<iostream> using namespace std;

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

void findTurnAroundTime(int i,int at[],int tat[],int qt)

{

// calculating turnaround time by

//complete at- start at

tat[i] = qt-at[i]; comp[i]=qt;

//cout<<"\nct"<<comp[i]<<" 3";

}

void findWaitingTime(int tat[], int n, int bt[], int wt[])

{ //calculating waiting time by

//turn around time -burst btime 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;

findTurnAroundTime(i,at,tat,ct);

cout<<"\n ~~~~~~\n"; cout<<j<<" |"<<" P"<<processes[i]<<" | "<< ct

<<"\n";//<<"\t"<<at[j]<<"\t"<<quantum1<<"\n"; cout<<" ~~~~~~";

}

}

}

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

{

int dup\_at[n],dup\_bt[n],tat[n],wt[n]; for (int i = 0 ; i < n ; i++)

dup\_at[i] = at[i];

//arranging places according to the PRIORITY 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;

}

}

}

//show

cout << "\nProcesses "<< " Arrival time "<< " Burst time "<<"Priotrity \n"; for (int i=0; i<n; i++)

{

cout << " " << processes[i] <<"\t\t"<<at[i] <<"\t\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])

{

if(dup\_bt[j]<ct)

{int z=0; z=ct;

ct=ct+dup\_bt[j]; dup\_bt[j]=0;

findTurnAroundTime(j,at,tat, ct); cout<<"\n ~~~~~~\n";

cout<<z<<" |"<<" P"<<processes[j]<<" | "<< ct <<"\n";

//<<"\t"<<at[j]<<"\t"<<quantum1<<"\n";

cout<<" ~~~~~~";

// cout<<" 1";

}

{ int q=0,z=0; z=ct;

if(i==(n-1))

else if(dup\_bt[j]>ct)

{

ct=ct+dup\_bt[j]; dup\_bt[j]=0;

}

else

{

}

q=dup\_at[i+1]-dup\_at[i]; ct=q+ct;

dup\_bt[j]=dup\_bt[j]-q;

if(dup\_bt[j]==0)

{

findTurnAroundTime(j,at,tat, ct); dup\_bt[j]=0;

//cout<<"\n 4";

}

cout<<"\n ~~~~~~\n";

cout<<z<<" |"<<" P"<<processes[j]<<" | "<< ct <<"\n";

//<<"\t"<<at[j]<<"\t"<<quantum1<<"\n";

cout<<" ~~~~~~";

// cout<<"\n 2";

}

j=n;

//cout<<"\n 5";

}

}

}

}

waitingQueue(dup\_bt,n,processes,at,tat); findWaitingTime(tat,n,bt,wt);

cout<< "\nProcesses "<< " Burst time "<<"\tArrival time "<<"turn around Time"<<"

Complition Time"

<<"waiting Time"<<"\n";

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

cout<<processes[i]<<"\t\t "<<bt[i]<<"\t\t "<<at[i]<<"\t\t "<<tat[i]<<"\t\t "

<<comp[i]<<"\t\t "<<wt[i]<<"\n";

}

//main code int main()

{

int n;

cout<<"enter no of process u want :-"; cin>>n;

int processes[n]; int burst\_time[n]; int arrival\_time[n]; int priority[n];

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

processes[i]=i+1; //process id's

cout<<"Enter Burst\_time of process no "<<i+1<<" :"; //burst\_time cin>>burst\_time[i];

cout<<"Enter Arrival\_time of process no"<<i+1<<" :"; cin>>arrival\_time[i];

cout<<"Enter Priority of process no"<<i+1<<" :"; cin>>priority[i];

}

cout<<"input :- \n";

cout << "Processes "<< " Arrival time "<< " Burst time "<<"Priotrity \n"; for (int i=0; i<n; i++)

{

cout << " " << processes[i] <<"\t\t"<<arrival\_time[i] <<"\t\t"<< burst\_time[i] <<"\t "

<< priority[i] <<"\t "<<endl;

}

ReadyQueue(processes,burst\_time,arrival\_time,n,priority); return(0);

}

# OUTPUT:



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

# A11.

#include<iostream> using namespace std;

// Function to find the waiting time for all

// processes

void findWaitingTime(int processes[], int n,

int bt[], int wt[],int at[])

{

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

int ct=0+bt[0];

// calculating waiting time for (int i = 1; i < n ; i++ ) {

wt[i] = ct-at[i] ;

ct=ct+bt[i]; }

}

// Function to calculate turn around time

void findTurnAroundTime( int processes[], int n, int bt[], int wt[], int tat[],int at[])

{

int ct=0;//Completion\_time

// calculating turnaround time by

// completion time - arrival time for (int i = 0; i < n ; i++) {

ct=ct+bt[i]; tat[i] =ct -at[i];

}

}

//Function to calculate average time

void findavgTime( int processes[], int n, int bt[],int at[])

{

int wt[n], tat[n], total\_wt = 0, total\_tat = 0,t=0;

//arranging places according to the Burst\_time for(int i=0;i<n+1;i++)

{

for(int j=0;j<n-1;j++)

{

if(bt[j]>bt[j+1])

{

t=bt[j]; bt[j]=bt[j+1]; bt[j+1]=t;

t=processes[j]; processes[j]=processes[j+1]; processes[j+1]=t;

t=at[j]; at[j]=at[j+1]; at[j+1]=t;

}

}

for(int j=0;j<n-1;j++)

{

if(j!=0&&at[j]==0&&at[j-1]!=0)

{

t=bt[j]; bt[j]=bt[j-1]; bt[j-1]=t;

t=processes[j]; processes[j]=processes[j-1]; processes[j-1]=t;

t=at[j]; at[j]=at[j-1];

at[j-1]=t;

}

}

}

//Function to find waiting time of all processes findWaitingTime(processes, n, bt, wt,at);

//Function to find turn around time for all processes findTurnAroundTime(processes, n, bt, wt, tat,at);

//Display processes along with all details

cout << "Processes "<<"Arrival time"<< " Burst time "

<< " Waiting time " << " Turn around time\n";

// Calculate total waiting time and total turn

// around time

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

{

total\_wt = total\_wt + wt[i]; total\_tat = total\_tat + tat[i];

cout << " " << processes[i] << "\t\t"<<at[i]<<"\t\t" << bt[i] <<"\t "

<< wt[i] <<"\t\t " << tat[i] <<endl;

}

cout << "Average waiting time = "

<< (float)total\_wt / (float)n;

cout << "\nAverage turn around time = "

<< (float)total\_tat / (float)n;

}

// Driver code int main()

{

int n;

cout<<"enter no of process u want :-"; cin>>n;

int processes[n]; int burst\_time[n]; int arrival\_time[n]; for(int i=0;i<n;i++){

processes[i]=i+1; //process id's

cout<<"Enter Burst\_time of process no "<<i+1<<" :"; //burst\_time cin>>burst\_time[i];

cout<<"Enter Arrival\_time of process no"<<i+1<<" :" ; //arrival\_time cin>>arrival\_time[i];

}

cout<<"input :- \n";

cout << "Processes "<<"Arrival time"<< " Burst time \n"; for (int i=0; i<n; i++)

{

cout << " " << processes[i] << "\t\t"<<arrival\_time[i]<<"\t\t" << burst\_time[i] <<"\t "

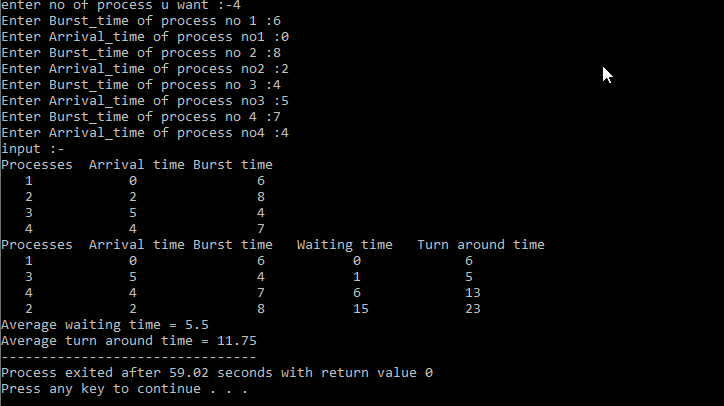
<<endl;

}

findavgTime(processes, n, burst\_time,arrival\_time); return 0;

}

# OUTPUT:



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

# A12.

#include <iostream> #include<pthread.h> using namespace std;

int global[2];

void \*sum\_thread(void \*arg)

{

int \*args\_array; args\_array = (int\*)arg;

int n1,n2,sum; n1=global[0]; n2=global[1]; sum = n1+n2;

cout<<"\n Sum = "<<sum;

return NULL;;

}

int main()

{

cout<<"\n First number: "; cin>>global[0];

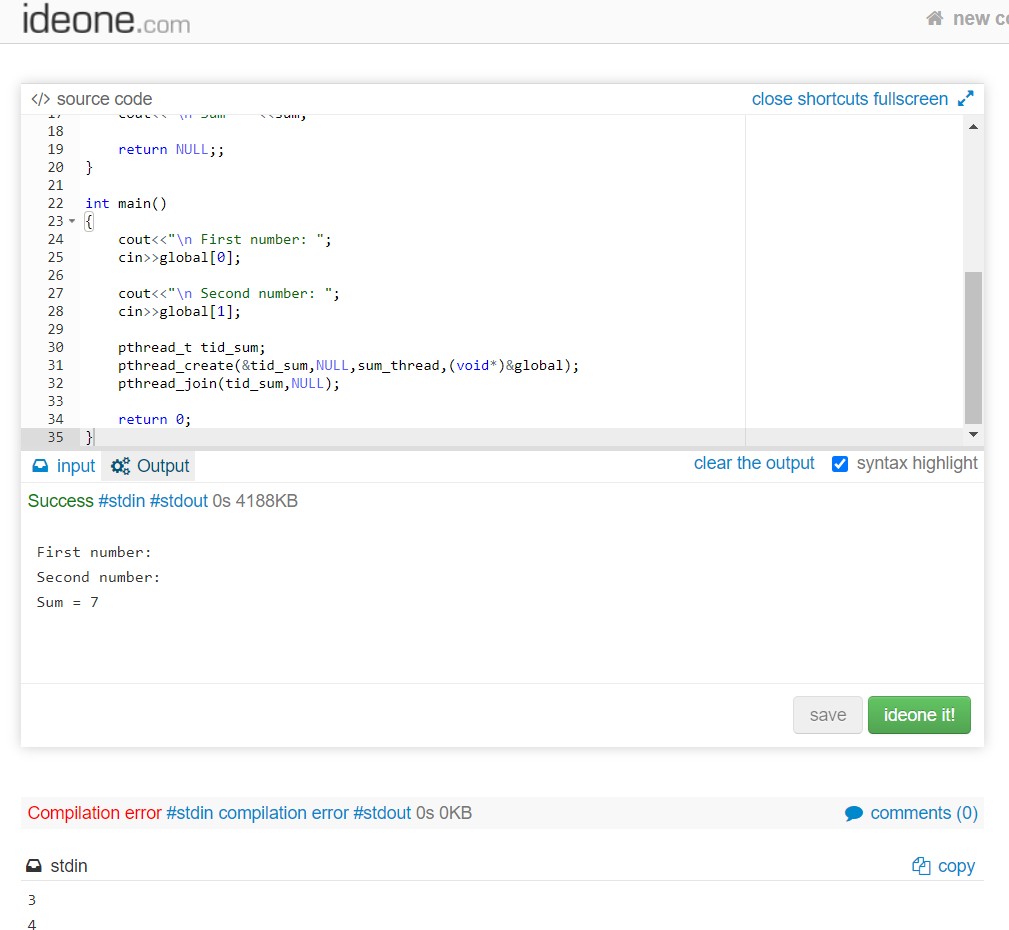
cout<<"\n Second number: "; cin>>global[1];

pthread\_t tid\_sum; pthread\_create(&tid\_sum,NULL,sum\_thread,(void\*)&global); pthread\_join(tid\_sum,NULL);

return 0;

}

# OUTPUT:



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

# A13.

1. **First Fit:**

#include<iostream> using namespace std;

void show(int block[],int burst[],int n,int c,int allocation[]){ cout<<"processes No "<<"size "<<"Allocated at "<<"Block size\n"; for(int i=0;i<c;i++){

if(allocation[i]==-1){

cout<<i+1<<"\t"<<burst[i]<<"\t\t"<<"Not Allocated\n";

}

else{

cout<<i+1<<"\t"<<burst[i]<<"\t\t"<<allocation[i]+1<<"\t"<<block[allocation[i]]<<"\n";

}

}

}

void first\_fit(int block[],int burst[],int n,int c){ int allocation[c];

for(int i=0;i<c;i++) allocation[i] = -1; int memoryused[n]; for(int i=0;i<n;i++) memoryused[i]= 0;

for(int i=0;i<c;i++){//c=No of Processes for(int j=0;j<n;j++){//j=No of blocks

if((block[j]-memoryused[j])>=burst[i]){

memoryused[j]+=burst[i]; allocation[i]=j;

j=n;

}

}

}

show(block,burst,n,c,allocation);

}

int main(){

int n,c;

cout<<"No of Memory blocks you want to enter:-"; cin>>n;

int Block[n];

cout<<"Enter no of process u want"; cin>>c;

int Burst[c];

cout<<"Enter Size of memory blocks resp:-"; for(int i=0;i<n;i++)

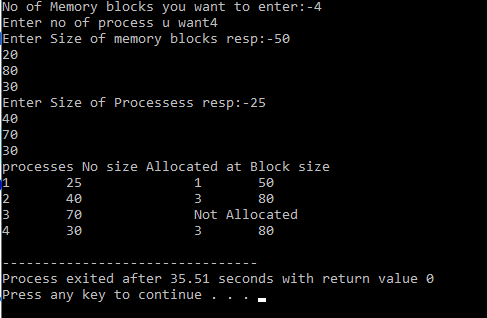
cin>>Block[i];

cout<<"Enter Size of Processess resp:-"; for(int i=0;i<n;i++)

cin>>Burst[i]; first\_fit(Block,Burst,n,c);

}

# OUTPUT:



1. **Best Fit:**

#include<iostream> using namespace std;

void show(int block[],int burst[],int n,int c,int allocation[]){ cout<<"processes size "<<"Allocated at "<<"Block size\n"; for(int i=0;i<c;i++){

if(allocation[i]==-1){

cout<<burst[i]<<"\t\t"<<"Not Allocated\n";

}

else{

cout<<burst[i]<<"\t\t"<<allocation[i]+1<<"\t"<<block[allocation[i]]<<"\n";

}

}

}

void best\_fit(int block[],int burst[],int n,int c){ int allocation[c];

for(int i=0;i<c;i++) allocation[i] = -1; int memoryused[n]; for(int i=0;i<n;i++) memoryused[i]= 0;

for(int i=0;i<n-1;i++)

{

for(int j=0;j<n-1;j++)

{

if(block[j]<block[j+1])

{

int t=0; t=block[j];

block[j]=block[j+1]; block[j+1]=t;

}

}

}

for(int i=0;i<c-1;i++)

{

for(int j=0;j<c-1;j++)

{

if(burst[j]<burst[j+1])

{

int t=0; t=burst[j];

burst[j]=burst[j+1]; burst[j+1]=t;

}

}

}

for(int i=0;i<c;i++){//c=No of Processes for(int j=0;j<n;j++){//j=No of blocks

if((block[j]-memoryused[j])>=burst[i]){

memoryused[j]+=burst[i]; allocation[i]=j;

j=n;

}

}

}

show(block,burst,n,c,allocation);

}

int main(){

int n,c;

cout<<"No of Memory blocks you want to enter:-"; cin>>n;

int Block[n];

cout<<"Enter no of process u want"; cin>>c;

int Burst[c];

cout<<"Enter Size of memory blocks resp:-"; for(int i=0;i<n;i++)

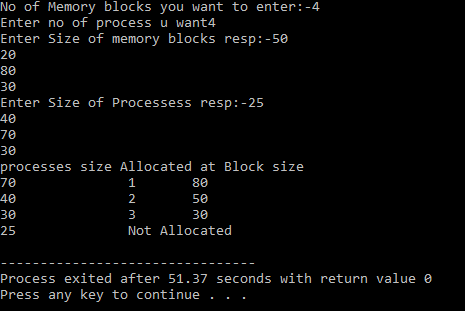
cin>>Block[i];

cout<<"Enter Size of Processess resp:-"; for(int i=0;i<n;i++)

cin>>Burst[i]; best\_fit(Block,Burst,n,c);

}

# OUTPUT:



1. **Worst Fit:**

#include<iostream> using namespace std;

void show(int block[],int burst[],int n,int c,int allocation[]){ cout<<"processes size "<<"Allocated at "<<"Block size\n"; for(int i=0;i<c;i++){

if(allocation[i]==-1){

cout<<burst[i]<<"\t\t"<<"Not Allocated\n";

}

else{

cout<<burst[i]<<"\t\t"<<allocation[i]+1<<"\t"<<block[allocation[i]]<<"\n";

}

}

}

void worst\_fit(int block[],int burst[],int n,int c){ int allocation[c];

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

allocation[i] = -1; int memoryused[n]; for(int i=0;i<n;i++) memoryused[i]= 0;

for(int i=0;i<n-1;i++)

{

for(int j=0;j<n-1;j++)

{

if(block[j]<block[j+1])

{

int t=0; t=block[j];

block[j]=block[j+1]; block[j+1]=t;

}

}

}

for(int i=0;i<c-1;i++)

{

for(int j=0;j<c-1;j++)

{

if(burst[j]>burst[j+1])

{

int t=0; t=burst[j];

burst[j]=burst[j+1]; burst[j+1]=t;

}

}

}

for(int i=0;i<c;i++){//c=No of Processes for(int j=0;j<n;j++){//j=No of blocks

if((block[j]-memoryused[j])>=burst[i]){

memoryused[j]+=burst[i]; allocation[i]=j;

j=n;

}

}

}

show(block,burst,n,c,allocation);

}

int main(){

int n,c;

cout<<"No of Memory blocks you want to enter:-"; cin>>n;

int Block[n];

cout<<"Enter no of process u want"; cin>>c;

int Burst[c];

cout<<"Enter Size of memory blocks resp:-"; for(int i=0;i<n;i++)

cin>>Block[i];

cout<<"Enter Size of Processess resp:-"; for(int i=0;i<n;i++)

cin>>Burst[i]; worst\_fit(Block,Burst,n,c);

}

# OUTPUT:

