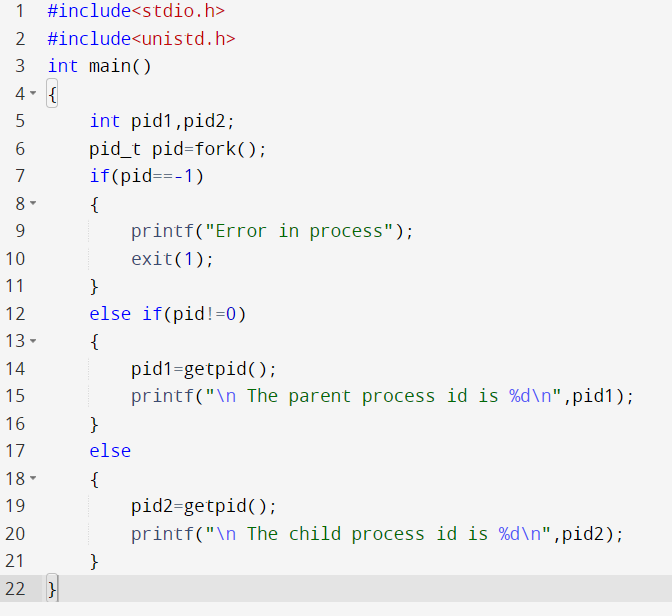
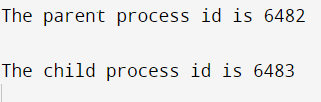
**OPERATING SYSTEMS(CSA0469)**

**PROGRAM 1:**

**CODE:**

****

**OUTPUT:**

****

**PROGRAM 2:**

**CODE:**

#include <stdio.h>

#include <stdlib.h>

int main()

{

FILE \*fptr1, \*fptr2;

char filename[100], c;

printf("Enter the filename to open for reading \n");

scanf("%s", filename);

fptr1 = fopen(filename, "r");

if (fptr1 == NULL)

{

printf("Cannot open file %s \n", filename);

exit(0);

}

printf("Enter the filename to open for writing \n");

scanf("%s", filename);

fptr2 = fopen(filename, "w");

if (fptr2 == NULL)

{

printf("Cannot open file %s \n", filename);

exit(0);

}

c = fgetc(fptr1);

while (c != EOF)

{

fputc(c, fptr2);

c = fgetc(fptr1);

}

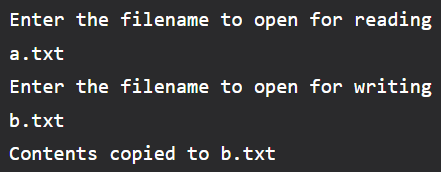
printf("\nContents copied to %s", filename);

fclose(fptr1);

fclose(fptr2);

}

**OUTPUT:**



**PROGRAM 3:**

**CODE:**

#include<stdio.h>

int main()

{

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

printf("Enter total number of processes(maximum 20):");

scanf("%d",&n);

printf("\nEnter Process Burst Time\n");

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

{

printf("P[%d]:",i+1);

scanf("%d",&bt[i]);

}

wt[0]=0;

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

{

wt[i]=0;

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

wt[i]+=bt[j];

}

printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time");

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

{

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

avwt+=wt[i];

avtat+=tat[i];

printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);

}

avwt/=i;

avtat/=i;

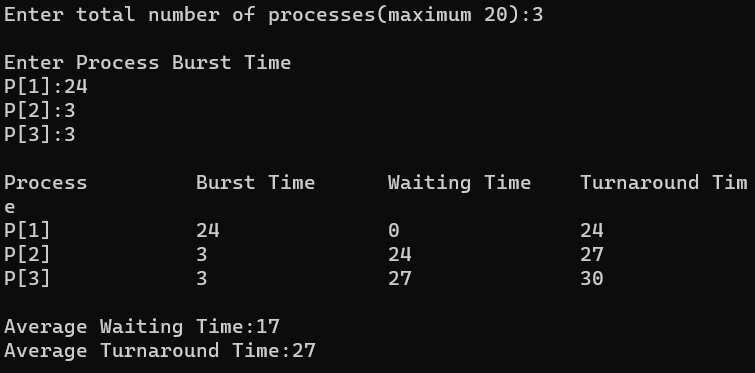
printf("\n\nAverage Waiting Time:%d",avwt);

printf("\nAverage Turnaround Time:%d",avtat);

return 0;

}

**OUTPUT:**



**PROGRAM 4:**

**CODE:**

#include<stdio.h>

int main()

{

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

printf("Enter number of process:");

scanf("%d",&n);

printf("\n Enter Burst Time:\n");

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

{

printf("p%d:",i+1);

scanf("%d",&bt[i]);

p[i]=i+1;

}

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

{

pos=i;

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

{

if(bt[j]<bt[pos]) pos=j;

}

temp=bt[i]; bt[i]=bt[pos];

bt[pos]=temp; temp=p[i];

p[i]=p[pos]; p[pos]=temp;

}

wt[0]=0;

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

{

wt[i]=0;

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

wt[i]+=bt[j];

total+=wt[i];

}

avg\_wt=(float)total/n;

total=0;

printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");

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

{

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

printf("\np%d\t\t %d\t\t %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);

}

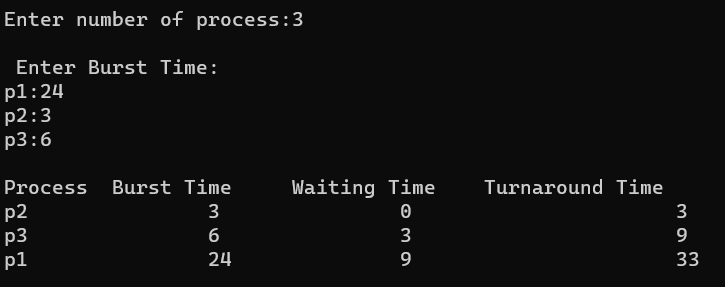
avg\_tat=(float)total/n;

printf("\n\nAverage Waiting Time=%f",avg\_wt);

printf("\nAverage Turnaround Time=%f\n",avg\_tat);

}

**OUTPUT:**



**PROGRAM 5:**

**CODE:**

#include<stdio.h>

int main()

{

int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;

float avg\_wt,avg\_tat;

printf("Enter number of process:");

scanf("%d",&n);

printf("\n Enter Burst Time:\n");

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

{

printf("p%d:",i+1);

scanf("%d",&bt[i]);

p[i]=i+1;

}

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

{

pos=i;

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

{

if(bt[j]>bt[pos])

pos=j;

}

temp=bt[i]; bt[i]=bt[pos];

bt[pos]=temp; temp=p[i];

p[i]=p[pos]; p[pos]=temp;

}

wt[0]=0;

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

{

wt[i]=0;

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

wt[i]+=bt[j];

total+=wt[i];

}

avg\_wt=(float)total/n;

printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");

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

{

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

printf("\np%d\t\t %d\t\t %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);

}

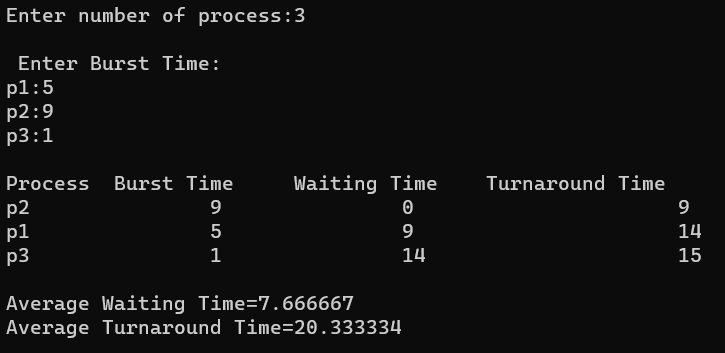
avg\_tat=(float)total/n;

printf("\n\nAverage Waiting Time=%f",avg\_wt);

printf("\nAverage Turnaround Time=%f\n",avg\_tat);

}

**OUTPUT:**



**PROGRAM 6:**

**CODE:**

#include<stdio.h>

int main()

{

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

printf("Enter Total Number of Process:");

scanf("%d",&n);

printf("\nEnter Burst Time and Priority\n");

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

{

printf("\nP[%d]\n",i+1);

printf("Burst Time:");

scanf("%d",&bt[i]);

printf("Priority:");

scanf("%d",&pr[i]);

p[i]=i+1;

}

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;

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;

total=0;

printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");

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

{

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

total+=tat[i];

printf("\nP[%d]\t\t %d\t\t %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);

}

avg\_tat=total/n;

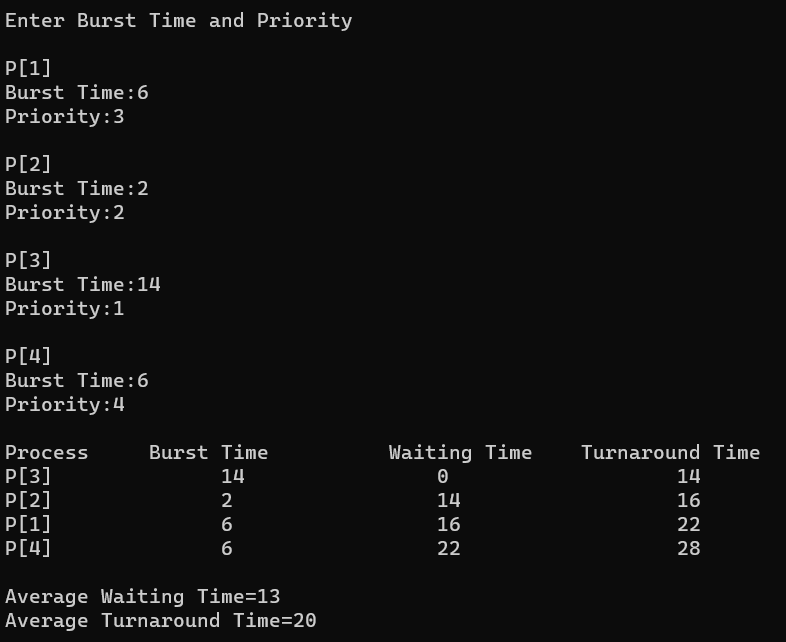
printf("\n\nAverage Waiting Time=%d",avg\_wt);

printf("\nAverage Turnaround Time=%d\n",avg\_tat);

return 0;

}

**OUTPUT:**



**PROGRAM 7:**

**CODE:**

#include<stdio.h>

int main()

{

int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;

float avg\_wt,avg\_tat;

printf("Enter number of process:");

scanf("%d",&n);

printf("\nEnter Burst Time:\n");

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

{

printf("p%d:",i+1);

scanf("%d",&bt[i]);

p[i]=i+1;

}

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

{

pos=i;

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

{

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

pos=j;

}

temp=bt[i];

bt[i]=bt[pos];

bt[pos]=temp;

temp=p[i];

p[i]=p[pos];

p[pos]=temp;

}

wt[0]=0;for(i=1;i<n;i++)

{

wt[i]=0;

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

wt[i]+=bt[j];

total+=wt[i];

}

avg\_wt=(float)total/n;

total=0;

printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");

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

{

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

total+=tat[i];

printf("\np%d\t\t %d\t\t %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);

}

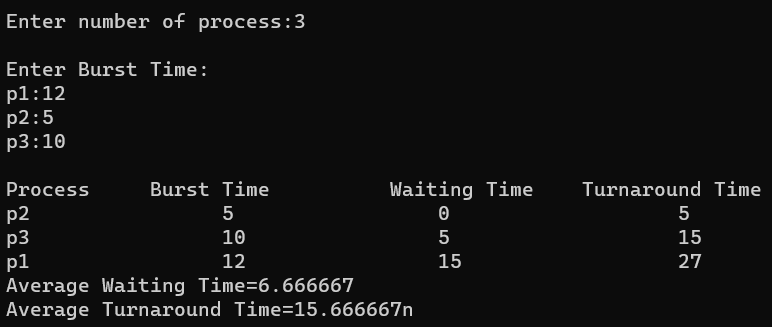
avg\_tat=(float)total/n;

printf("\nAverage Waiting Time=%f",avg\_wt);

printf("\nAverage Turnaround Time=%fn",avg\_tat);

}

**OUTPUT:**



**PROGRAM 8:**

**CODE:**

#include <stdio.h>

int main()

{

int i, total = 0, x, limit, counter = 0, t\_quantum;

int wait\_time = 0, turnaround\_time = 0, arrival\_time[10], burst\_time[10], temp[10];

float average\_wait\_time, average\_turnaround\_time;

printf("\nEnter Total Number of Processes: ");

scanf("%d", &limit);

x = limit;

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

{

printf("\nProvide the details for Process[%d]\n", i + 1);

printf("Arrival Time:\t");

scanf("%d", &arrival\_time[i]);

printf("Burst Time:\t");

scanf("%d", &burst\_time[i]);

temp[i] = burst\_time[i];

}

printf("\nEnter Time Quantum:\t");

scanf("%d", &t\_quantum);

printf("\nProcess ID\t\tBurst Time\t Turnaround Time\t Waiting Time\n");

for (total = 0, i = 0; x != 0;)

{

if (temp[i] <= t\_quantum && temp[i] > 0)

{

total = total + temp[i];

temp[i] = 0;

counter = 1;

}

else if (temp[i] > 0)

{

temp[i] = temp[i] - t\_quantum;

total = total + t\_quantum;

}

if (temp[i] == 0 && counter == 1)

{

x--;

printf("\nProcess[%d]\t\t%d\t\t %d\t\t\t %d", i + 1, burst\_time[i], total - arrival\_time[i], total - arrival\_time[i] - burst\_time[i]);

wait\_time = wait\_time + total - arrival\_time[i] - burst\_time[i];

turnaround\_time = turnaround\_time + total - arrival\_time[i];

counter = 0;

}

if (i == limit - 1)

{

i = 0;

}

else if (arrival\_time[i + 1] <= total)

{

i++;

}

else

{

i = 0;

}

}

average\_wait\_time = wait\_time \*1.0 / limit;

average\_turnaround\_time = turnaround\_time \*1.0 / limit;

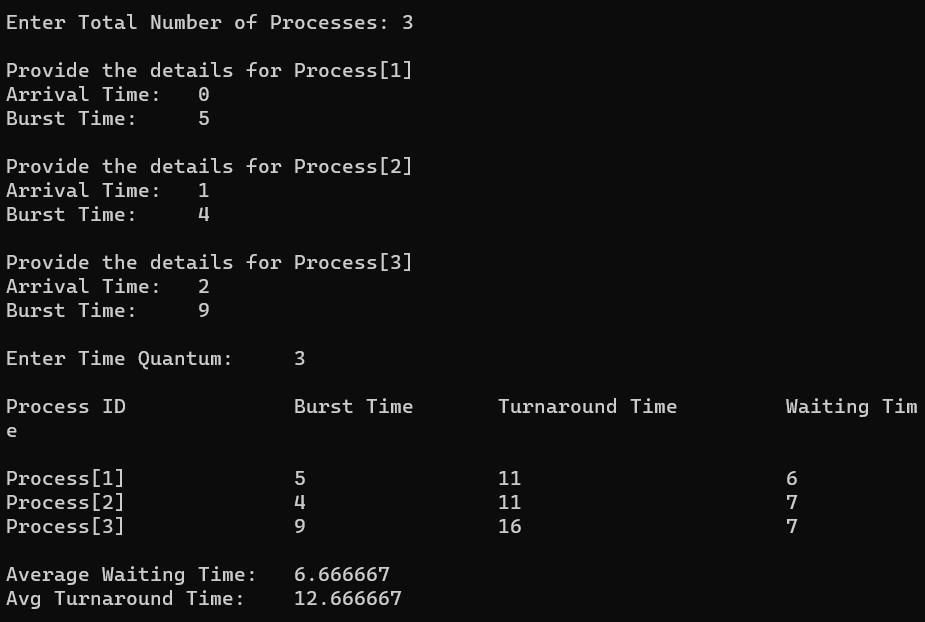
printf("\n\nAverage Waiting Time:\t%f", average\_wait\_time);

printf("\nAvg Turnaround Time:\t%f\n", average\_turnaround\_time);

return 0;

}

**OUTPUT:**



**PROGRAM 9:**

**CODE:**

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/shm.h>

#include<string.h>

**int** main()

{

**int** i;

**void** \*shared\_memory;

**char** buff[100];

**int** shmid;

shmid=shmget((key\_t)2345, 1024, 0666|IPC\_CREAT);

printf("Key of shared memory is %d\n",shmid);

shared\_memory=shmat(shmid,NULL,0);

printf("Process attached at %p\n",shared\_memory);

printf("Enter some data to write to shared memory\n");

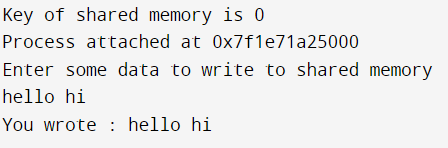
read(0,buff,100);

strcpy(shared\_memory,buff);

printf("You wrote : %s\n",(**char** \*)shared\_memory);

}

**OUTPUT:**



**PROGRAM 10:**

**CODE:**

#include<stdlib.h>

#include<stdio.h>

#include<string.h>

#include<unistd.h>

#include<sys/types.h>

#include<sys/ipc.h>

#include<sys/msg.h>

#define MAX\_TEXT 512 //maximum length of the message that can be sent allowed

struct my\_msg{

long int msg\_type;

char some\_text[MAX\_TEXT];

};

int main()

{

int running=1;

int msgid;

struct my\_msg some\_data;

char buffer[50]; //array to store user input

msgid=msgget((key\_t)14534,0666|IPC\_CREAT);

if (msgid == -1) // -1 means the message queue is not created

{

printf("Error in creating queue\n");

exit(0);

}

while(running)

{

printf("Enter some text:\n");

fgets(buffer,50,stdin);

some\_data.msg\_type=1;

strcpy(some\_data.some\_text,buffer);

if(msgsnd(msgid,(void \*)&some\_data, MAX\_TEXT,0)==-1) // msgsnd returns -1 if the message is not sent

{

printf("Msg not sent\n");

}

if(strncmp(buffer,"end",3)==0)

{

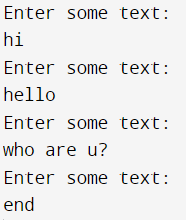
running=0;

}

}

}

**OUTPUT:**



**PROGRAM 11:**

**CODE:**

#include <iostream>

#include <cstdlib>

#include <pthread.h>

using namespace std;

#define NUM\_THREADS 5

void \*PrintHello(void \*threadid) {

long tid;

tid = (long)threadid;

printf("Hello World! Thread ID, %d

", tid);

pthread\_exit(NULL);

}

int main () {

pthread\_t threads[NUM\_THREADS];

int rc;

int i;

for( i = 0; i < NUM\_THREADS; i++ ) {

cout << "main() : creating thread, " << i << endl;

rc = pthread\_create(&threads[i], NULL, PrintHello, (void \*)i);

if (rc) {

printf("Error:unable to create thread, %d

", rc);

exit(-1);

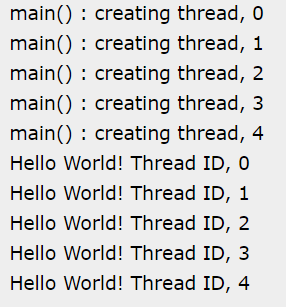
}

}

pthread\_exit(NULL);

}

**OUTPUT:**



**PROGRAM 12:**

**CODE:**

#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

#include<semaphore.h>

#include<unistd.h>

sem\_t room;

sem\_t chopstick[5];

void \* philosopher(void \*);

void eat(int);

int main()

{

int i,a[5];

pthread\_t tid[5];

sem\_init(&room,0,4);

for(i=0;i<5;i++)

sem\_init(&chopstick[i],0,1);

for(i=0;i<5;i++){

a[i]=i;

pthread\_create(&tid[i],NULL,philosopher,(void \*)&a[i]);

}

for(i=0;i<5;i++)

pthread\_join(tid[i],NULL);

}

void \* philosopher(void \* num)

{

int phil=\*(int \*)num;

sem\_wait(&room);

printf("\nPhilosopher %d has entered room",phil);

sem\_wait(&chopstick[phil]);

sem\_wait(&chopstick[(phil+1)%5]);

eat(phil);

sleep(2);

printf("\nPhilosopher %d has finished eating",phil);

sem\_post(&chopstick[(phil+1)%5]);

sem\_post(&chopstick[phil]);

sem\_post(&room);

}

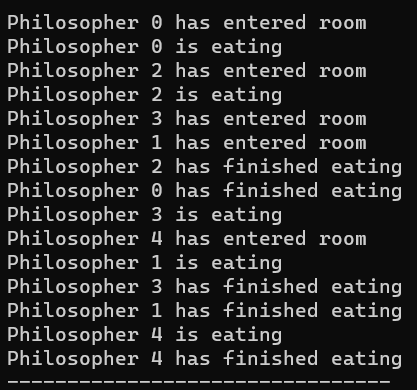
void eat(int phil)

{

printf("\nPhilosopher %d is eating",phil);

}

**OUTPUT:**



**PROGRAM 13**

**CODE:**

#include<stdio.h>

int main()

{

int bsize[10], psize[10], bno, pno, flags[10], allocation[10], i, j;

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

{

flags[i] = 0;

allocation[i] = -1;

}

printf("Enter no. of blocks: ");

scanf("%d", &bno);

printf("\nEnter size of each block: ");

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

scanf("%d", &bsize[i]);

printf("\nEnter no. of processes: ");

scanf("%d", &pno);

printf("\nEnter size of each process: ");

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

scanf("%d", &psize[i]);

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

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

if(flags[j] == 0 && bsize[j] >= psize[i])

{

allocation[j] = i;

flags[j] = 1;

break;

}

printf("\nBlock no.\tsize\t\tprocess no.\t\tsize");

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

{

printf("\n%d\t\t%d\t\t", i+1, bsize[i]);

if(flags[i] == 1)

printf("%d\t\t\t%d",allocation[i]+1,psize[allocation[i]]);

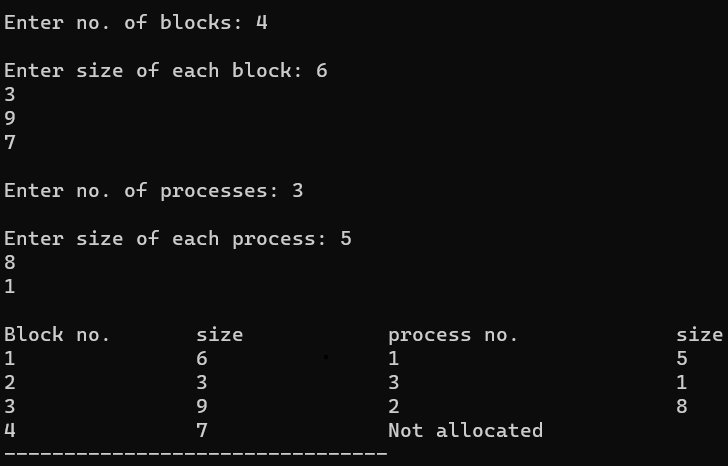
else

printf("Not allocated");

}

}

**OUTPUT:**



**PROGRAM 14**

**CODE:**

#include<stdio.h>

#include<conio.h>

#include<string.h>

int main()

{

int nf=0,i=0,j=0,ch;

char mdname[10],fname[10][10],name[10];

printf("Enter the directory name:");

scanf("%s",mdname);

printf("Enter the number of files:");

scanf("%d",&nf);

do

{

printf("Enter file name to be created:");

scanf("%s",name);

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

{

if(!strcmp(name,fname[i]))

break;

}

if(i==nf)

{

strcpy(fname[j++],name);

nf++;

}

else

printf("There is already %s\n",name);

printf("Do you want to enter another file(yes - 1 or no - 0):");

scanf("%d",&ch);

}

while(ch==1);

printf("Directory name is:%s\n",mdname);

printf("Files names are:");

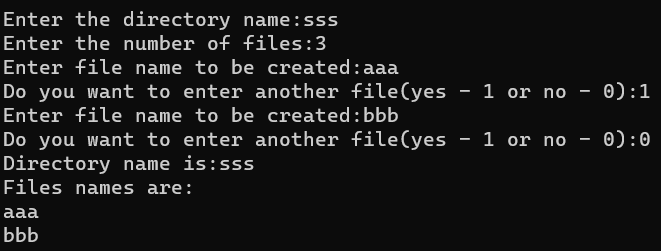
for(i=0;i<j;i++)

printf("\n%s",fname[i]);

getch();

}

**OUTPUT:**



**PROGRAM 15**

**CODE:**

#include<stdio.h>

#include<conio.h>

struct st

{

char dname[10];

char sdname[10][10];

char fname[10][10][10];

int ds,sds[10];

}dir[10];

int main()

{

int i,j,k,n;

printf("enter number of directories:");

scanf("%d",&n);

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

{

printf("enter directory %d names:",i+1);

scanf("%s",&dir[i].dname);

printf("enter size of directories:");

scanf("%d",&dir[i].ds);

for(j=0;j<dir[i].ds;j++)

{

printf("enter subdirectory name and size:");

scanf("%s",&dir[i].sdname[j]);

scanf("%d",&dir[i].sds[j]);

for(k=0;k<dir[i].sds[j];k++)

{

printf("enter file name:");

scanf("%s",&dir[i].fname[j][k]);

}

}

}

printf("\ndirname\t\tsize\tsubdirname\tsize\tfiles");

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

{

printf("%s\t\t%d",dir[i].dname,dir[i].ds);

for(j=0;j<dir[i].ds;j++)

{

printf("\t%s\t\t%d\t",dir[i].sdname[j],dir[i].sds[j]);

for(k=0;k<dir[i].sds[j];k++)

printf("%s\t",dir[i].fname[j][k]);

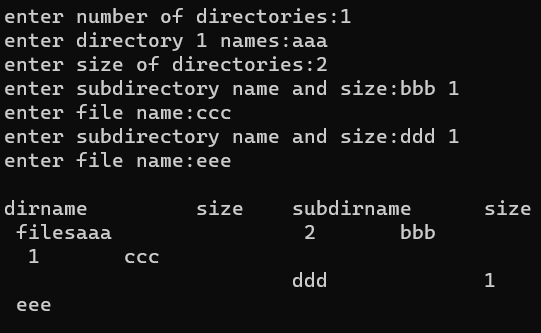
printf("\n\t\t");

}

printf("\n"); }

getch(); }

**OUTPUT:**



**PROGRAM 16**

**CODE:**

#include <stdio.h>

struct employee {

char name[20];

int id\_number;

};

int main()

{

FILE \*fp;

struct employee e = {"", 0};

if (!(fp = fopen( "employee.txt", "r+" )))

{

printf("File cannot be opened.");

return 0;

}

while(1)

{

printf("Enter id number from (1 to 100) , -1 to end input : ");

scanf("%d",&e.id\_number);

if(e.id\_number == -1)

break;

printf("Enter name : ");

scanf("%e",e.name);

fseek(fp,(e.id\_number-1)\*sizeof(e),0);

fwrite(&e, sizeof(e), 1, fp);

}

fclose(fp);

return 0;

}

**PROGRAM 17**

**CODE:**

#include<stdio.h>

int main() {

int p, c, count = 0, i, j, alc[5][3], max[5][3], need[5][3], safe[5], available[3], done[5], terminate = 0;

printf("Enter the number of process and resources");

scanf("%d %d", & p, & c);

printf("enter allocation of resource of all process %dx%d matrix", p, c);

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

for (j = 0; j < c; j++) {

scanf("%d", & alc[i][j]);

}

}

printf("enter the max resource process required %dx%d matrix", p, c);

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

for (j = 0; j < c; j++) {

scanf("%d", & max[i][j]);

}

}

printf("enter the available resource");

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

scanf("%d", & available[i]);

printf("\n need resources matrix are\n");

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

for (j = 0; j < c; j++) {

need[i][j] = max[i][j] - alc[i][j];

printf("%d\t", need[i][j]);

}

printf("\n");

}

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

done[i] = 0;

}

while (count < p) {

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

if (done[i] == 0) {

for (j = 0; j < c; j++) {

if (need[i][j] > available[j])

break;

}

if (j == c) {

safe[count] = i;

done[i] = 1;

for (j = 0; j < c; j++) {

available[j] += alc[i][j];

}

count++;

terminate = 0;

} else {

terminate++;

}

}

}

if (terminate == (p - 1)) {

printf("safe sequence does not exist");

break;

}

}

if (terminate != (p - 1)) {

printf("\n available resource after completion\n");

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

printf("%d\t", available[i]);

}

printf("\n safe sequence are\n");

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

printf("p%d\t", safe[i]);

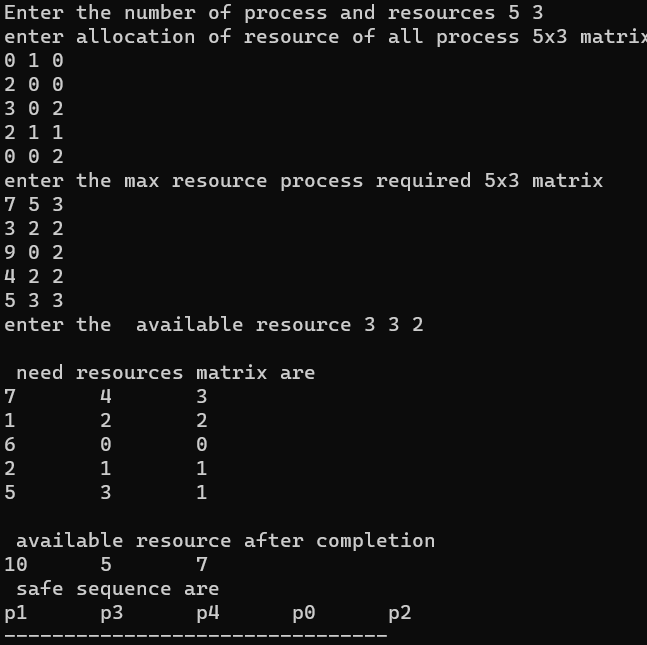
}

}

return 0;

}

**OUTPUT:**



**PROGRAM 18**

**CODE:**

#include <stdio.h>

#include <stdlib.h>

int mutex = 1;

int full = 0;

int empty = 10, x = 0;

void producer()

{

--mutex;

++full;

--empty;

x++;

printf("\nProducer produces"

"item %d",

x);

++mutex;

}

void consumer()

{

--mutex;

--full;

++empty;

printf("\nConsumer consumes "

"item %d",

x);

x--;

++mutex;

}

int main()

{

int n, i;

printf("\n1. Press 1 for Producer"

"\n2. Press 2 for Consumer"

"\n3. Press 3 for Exit");

for (i = 1; i > 0; i++) {

printf("\nEnter your choice:");

scanf("%d", &n);

switch (n) {

case 1:

if ((mutex == 1)

&& (empty != 0)) {

producer();

}

else {

printf("Buffer is full!");

}

break;

case 2:

if ((mutex == 1)

&& (full != 0)) {

consumer();

}

else {

printf("Buffer is empty!");

}

break;

case 3:

exit(0);

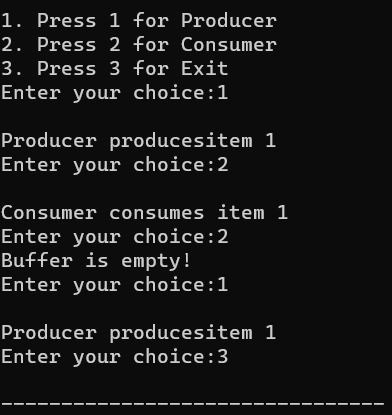
break;

}

}

}

**OUTPUT:**



**PROGRAM 19**

**CODE:**

#include <pthread.h>

#include <stdio.h>

pthread\_mutex\_t lock;

void\* func1(void\* arg) {

pthread\_mutex\_lock(&lock);

printf("Thread 1: Locked\n");

printf("Thread 1: Unlocked\n");

pthread\_mutex\_unlock(&lock);

return NULL;

}

void\* func2(void\* arg) {

pthread\_mutex\_lock(&lock);

printf("Thread 2: Locked\n");

printf("Thread 2: Unlocked\n");

pthread\_mutex\_unlock(&lock);

return NULL;

}

int main() {

pthread\_t thread1, thread2;

pthread\_mutex\_init(&lock, NULL);

pthread\_create(&thread1, NULL, func1, NULL);

pthread\_create(&thread2, NULL, func2, NULL);

pthread\_join(thread1, NULL);

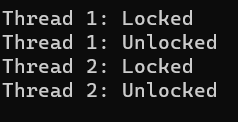
pthread\_join(thread2, NULL);

pthread\_mutex\_destroy(&lock);

return 0;

}

**OUTPUT:**



**PROGRAM 20**

**CODE:**

#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>

sem\_t wrt;

pthread\_mutex\_t mutex;

int cnt = 1;

int numreader = 0;

void \*writer(void \*wno)

{

sem\_wait(&wrt);

cnt = cnt\*2;

printf("Writer %d modified cnt to %d\n",(\*((int \*)wno)),cnt);

sem\_post(&wrt);

}

void \*reader(void \*rno)

{

pthread\_mutex\_lock(&mutex);

numreader++;

if(numreader == 1) {

sem\_wait(&wrt);

}

pthread\_mutex\_unlock(&mutex);

printf("Reader %d: read cnt as %d\n",\*((int \*)rno),cnt);

pthread\_mutex\_lock(&mutex);

numreader--;

if(numreader == 0) {

sem\_post(&wrt);

}

pthread\_mutex\_unlock(&mutex);

}

int main()

{

pthread\_t read[10],write[5];

pthread\_mutex\_init(&mutex, NULL);

sem\_init(&wrt,0,1);

int a[10] = {1,2,3,4,5,6,7,8,9,10};

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

pthread\_create(&read[i], NULL, (void \*)reader, (void \*)&a[i]);

}

for(int i = 0; i < 5; i++) {

pthread\_create(&write[i], NULL, (void \*)writer, (void \*)&a[i]);

}

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

pthread\_join(read[i], NULL);

}

for(int i = 0; i < 5; i++) {

pthread\_join(write[i], NULL);

}

pthread\_mutex\_destroy(&mutex);

sem\_destroy(&wrt);

return 0;

}

**OUTPUT:**

**PROGRAM 21**

**CODE:**

#include <stdio.h>

void implimentWorstFit(int blockSize[], int blocks, int processSize[], int processes)

{

int allocation[processes];

int occupied[blocks];

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

allocation[i] = -1;

}

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

occupied[i] = 0;

}

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

{

int indexPlaced = -1;

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

{

if(blockSize[j] >= processSize[i] && !occupied[j])

{

if (indexPlaced == -1)

indexPlaced = j;

else if (blockSize[indexPlaced] < blockSize[j])

indexPlaced = j;

}

}

if (indexPlaced != -1)

{

allocation[i] = indexPlaced;

occupied[indexPlaced] = 1;

blockSize[indexPlaced] -= processSize[i];

}

}

printf("\nProcess No.\tProcess Size\tBlock no.\n");

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

{

printf("%d \t\t\t %d \t\t\t", i+1, processSize[i]);

if (allocation[i] != -1)

printf("%d\n",allocation[i] + 1);

else

printf("Not Allocated\n");

}

}

int main()

{

int blockSize[] = {100, 50, 30, 120, 35};

int processSize[] = {40, 10, 30, 60};

int blocks = sizeof(blockSize)/sizeof(blockSize[0]);

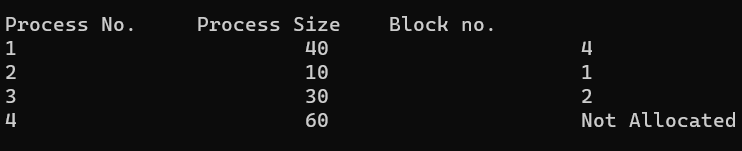
int processes = sizeof(processSize)/sizeof(processSize[0]);

implimentWorstFit(blockSize, blocks, processSize, processes);

return 0;

}

**OUTPUT:**



**PROGRAM 22**

**CODE:**

#include<stdio.h>

int main()

{

int fragment[20],b[20],p[20],i,j,nb,np,temp,lowest=9999;

int barray[20],parray[20];

printf("\nEnter the number of blocks:");

scanf("%d",&nb);

printf("Enter the number of processes:");

scanf("%d",&np);

printf("\nEnter the size of the blocks:-\n");

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

{

printf("Block no.%d:",i);

scanf("%d",&b[i]);

}

printf("\nEnter the size of the processes :-\n");

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

{

printf("Process no.%d:",i);

scanf("%d",&p[i]);

}

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

{

for(j=1;j<=nb;j++)

{

if(barray[j]!=1)

{

temp=b[j]-p[i];

if(temp>=0)

if(lowest>temp)

{

parray[i]=j;

lowest=temp;

}

}

}

fragment[i]=lowest;

barray[parray[i]]=1;

lowest=10000;

}

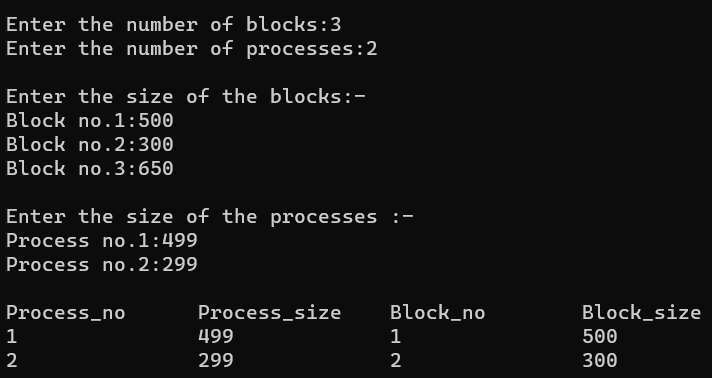
printf("\nProcess\_no\tProcess\_size\tBlock\_no\tBlock\_size\tFragment");

for(i=1;i<=np && parray[i]!=0;i++)

printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,p[i],parray[i],b[parray[i]],fragment[i]);

}

**OUTPUT:**



**PROGRAM 23**

**CODE:**

#include<stdio.h>

int main()

{

int bsize[10], psize[10], bno, pno, flags[10], allocation[10], i, j;

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

{

flags[i] = 0;

allocation[i] = -1;

}

printf("Enter no. of blocks: ");

scanf("%d", &bno);

printf("\nEnter size of each block: ");

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

scanf("%d", &bsize[i]);

printf("\nEnter no. of processes: ");

scanf("%d", &pno);

printf("\nEnter size of each process: ");

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

scanf("%d", &psize[i]);

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

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

if(flags[j] == 0 && bsize[j] >= psize[i])

{

allocation[j] = i;

flags[j] = 1;

break;

}

printf("\nBlock no.\tsize\t\tprocess no.\t\tsize");

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

{

printf("\n%d\t\t%d\t\t", i+1, bsize[i]);

if(flags[i] == 1)

printf("%d\t\t\t%d",allocation[i]+1,psize[allocation[i]]);

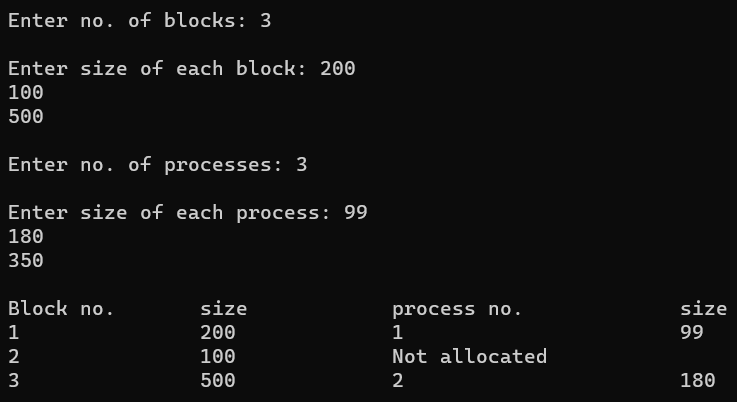
else

printf("Not allocated");

}

}

**OUTPUT:**



**PROGRAM 24**

**CODE:**

#include<unistd.h>

#include<fcntl.h>

#include<sys/stat.h>

#include<sys/types.h>

#include<stdio.h>

int main()

{

int n,fd;

printf("Enter text to write in the file:\n");

n= read(0, buff, 50);

fd=open("file",O\_CREAT | O\_RDWR, 0777);

write(fd, buff, n);

write(1, buff, n);

int close(int fd);

return 0;

}

**OUTPUT:**

**PROGRAM 25**

**CODE:**

#include<stdio.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<fcntl.h>

#include<stdlib.h>\*/

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

{

int n, fd;

char buff[50];

fd=open(argv[1],0);

if(fd!=-1)

{

while ((n=read(fd, buff, sizeof(buff)))>0)

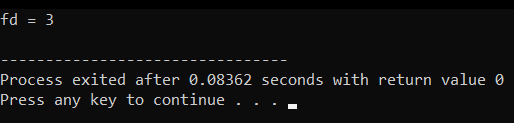
write (1, buff, n);

}

close(fd);

}

**OUTPUT:**



**PROGRAM 26**

**CODE:**

#include <stdio.h> #include <stdlib.h> int main() {

FILE \*file;

file = fopen("example.txt", "w"); if (file == NULL) {

printf("Error opening the file for writing.\n"); return 1;

}

fprintf(file, "Hello, World!\n");

fprintf(file, "This is a C file management example.\n"); fclose(file);

file = fopen("example.txt", "r"); if (file == NULL) {

printf("Error opening the file for reading.\n"); return 1;

}

char buffer[100];

while (fgets(buffer, sizeof(buffer), file) != NULL) { printf("%s", buffer);

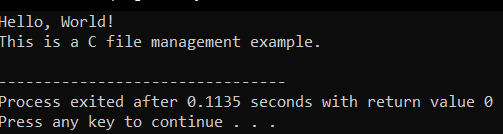
}

fclose(file);

return 0;

}

**OUTPUT:**



**PROGRAM 27**

**CODE:**

#include<stdio.h> #include<dirent.h>

int main()

{

char fn[10], pat[10], temp[200]; FILE \*fp;

printf("\n Enter file name : "); scanf("%s", fn);

printf("Enter the pattern: "); scanf("%s", pat);

fp = fopen(fn, "r");

while (!feof(fp)) { fgets(temp, sizeof(fp), fp); if (strcmp(temp, pat))

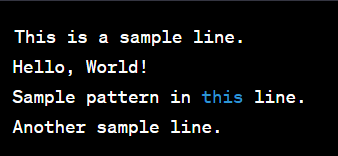
printf("%s", temp);

}

fclose(fp); return 1;

}

**OUTPUT:**



**PROGRAM 28**

**CODE:**

#include <stdio.h> #include <stdlib.h> #include <string.h>

#define MAX\_LINE\_LENGTH 1024

void searchFile(const char \*pattern, const char \*filename) { FILE \*file = fopen(filename, "r");

if (file == NULL) { perror("Error opening file"); exit(1);

}

char line[MAX\_LINE\_LENGTH]; while (fgets(line, sizeof(line), file)) {

if (strstr(line, pattern) != NULL) { printf("%s", line);

}

}

fclose(file);

}

int main(int argc, char \*argv[]) { if (argc != 3) {

fprintf(stderr, "Usage: %s <pattern> <filename>\n", argv[0]); return 1;

}

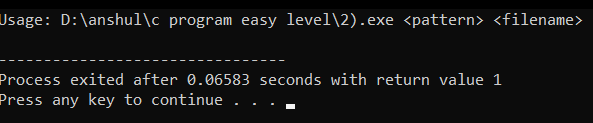
const char \*pattern = argv[1]; const char \*filename = argv[2];

searchFile(pattern, filename);

return 0;

}

**OUTPUT:**



**PROGRAM 29**

**CODE:**

#include <stdio.h> #include <stdlib.h> int mutex = 1;

int full = 0;

int empty = 10, x = 0; void producer()

{

--mutex;

++full;

--empty; x++;

printf("\nProducer produces" "item %d",

x);

++mutex;

}

void consumer()

{

--mutex;

--full;

++empty;

printf("\nConsumer consumes " "item %d",

x);

x--;

++mutex;

}

int main()

{

int n, i;

printf("\n1. Press 1 for Producer"

"\n2. Press 2 for Consumer" "\n3. Press 3 for Exit");

#pragma omp critical for (i = 1; i > 0; i++)

{

printf("\nEnter your choice:"); scanf("%d", &n);

switch (n) { case 1:

if ((mutex == 1)

&& (empty != 0)) { producer();

}

else

{

printf("Buffer is full!");

}

break; case 2:

if ((mutex == 1) && (full != 0)) { consumer();

}

else {

printf("Buffer is empty!");

}

break;

case 3:

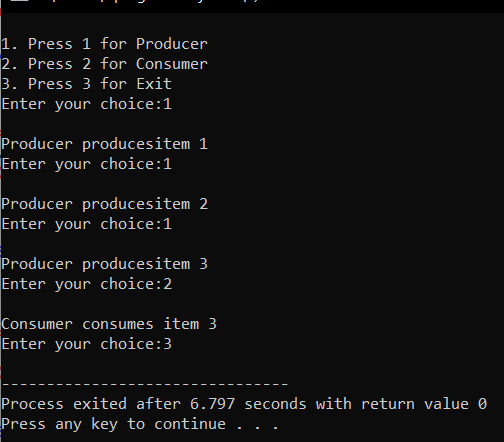
exit(0); break;

}

}

}

**OUTPUT:**



**PROGRAM 30**

**CODE:**

#include <pthread.h> #include <stdio.h> #include <stdlib.h> void\* func(void\* arg)

{

pthread\_detach(pthread\_self()); printf("Inside the thread\n"); pthread\_exit(NULL);

}

void fun()

{

pthread\_t ptid;

pthread\_create(&ptid, NULL, &func, NULL); printf("This line may be printed"

" before thread terminates\n"); if(pthread\_equal(ptid, pthread\_self()))

{

printf("Threads are equal\n");

}

Else

printf("Threads are not equal\n"); pthread\_join(ptid, NULL);

printf("This line will be printed" " after thread ends\n");

pthread\_exit(NULL);

}

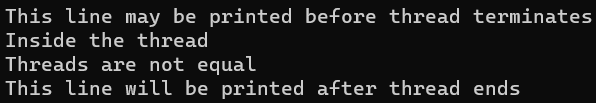
int main()

{

fun(); return 0;

}

**OUTPUT:**



**PROGRAM 31**

**CODE:**

#include <stdio.h>

#define MAX\_FRAMES 3 // Maximum number of frames in memory

void printFrames(int frames[], int n) { for (int i = 0; i < n; i++) {

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

printf(" - ");

} else {

printf(" %d ", frames[i]);

}

}

printf("\n");

}

int main() {

int referenceString[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2}; int n = sizeof(referenceString) / sizeof(referenceString[0]);

int frames[MAX\_FRAMES];

int framePointer = 0; // Points to the current frame to be replaced

for (int i = 0; i < MAX\_FRAMES; i++) {

frames[i] = -1; // Initialize all frames to -1 (indicating empty)

}

printf("Reference String: "); for (int i = 0; i < n; i++) {

printf("%d ", referenceString[i]);

}

printf("\n\n");

printf("Page Replacement Order:\n");

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

int page = referenceString[i]; int pageFound = 0;

// Check if the page is already in memory for (int j = 0; j < MAX\_FRAMES; j++) {

if (frames[j] == page) { pageFound = 1; break;

}

}

if (!pageFound) {

printf("Page %d -> ", page); frames[framePointer] = page;

framePointer = (framePointer + 1) % MAX\_FRAMES; printFrames(frames, MAX\_FRAMES);

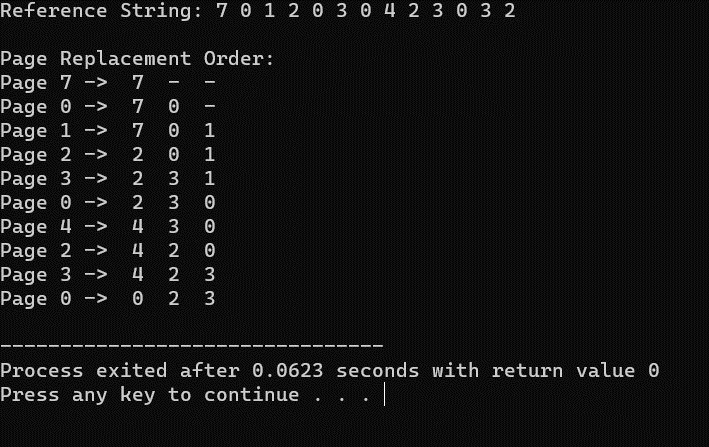
}

}

return 0;

}

**OUTPUT:**



**PROGRAM 32**

**CODE:**

#include <stdio.h> #include <stdlib.h>

#define MAX\_FRAMES 3

void printFrames(int frames[], int n) { for (int i = 0; i < n; i++) {

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

printf(" - ");

} else {

printf(" %d ", frames[i]);

}

}

printf("\n");

}

int main() {

int frames[MAX\_FRAMES];

int usageHistory[MAX\_FRAMES]; // To store the usage history of pages

for (int i = 0; i < MAX\_FRAMES; i++) {

frames[i] = -1; // Initialize frames to -1 (empty) usageHistory[i] = 0; // Initialize usage history

}

int pageFaults = 0;

int referenceString[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2}; int n = sizeof(referenceString) / sizeof(referenceString[0]);

printf("Reference String: "); for (int i = 0; i < n; i++) {

printf("%d ", referenceString[i]);

}

printf("\n\n");

printf("Page Replacement Order:\n"); for (int i = 0; i < n; i++) {

int page = referenceString[i]; int pageFound = 0;

// Check if the page is already in memory (a page hit) for (int j = 0; j < MAX\_FRAMES; j++) {

if (frames[j] == page) { pageFound = 1;

// Update the usage history by incrementing other pages for (int k = 0; k < MAX\_FRAMES; k++) {

if (k != j) { usageHistory[k]++;

}

}

usageHistory[j] = 0; // Reset the usage counter for the used page break;

}

}

if (!pageFound) {

printf("Page %d -> ", page);

used)// Find the page with the maximum usage counter (least recently

int lruPage = 0;

for (int j = 1; j < MAX\_FRAMES; j++) {

if (usageHistory[j] > usageHistory[lruPage]) { lruPage = j;

}

}

int replacedPage = frames[lruPage]; frames[lruPage] = page; usageHistory[lruPage] = 0;

if (replacedPage != -1) {

printf("Replace %d with %d: ", replacedPage, page);

} else {

printf("Load into an empty frame: ");

}

printFrames(frames, MAX\_FRAMES); pageFaults++;

}

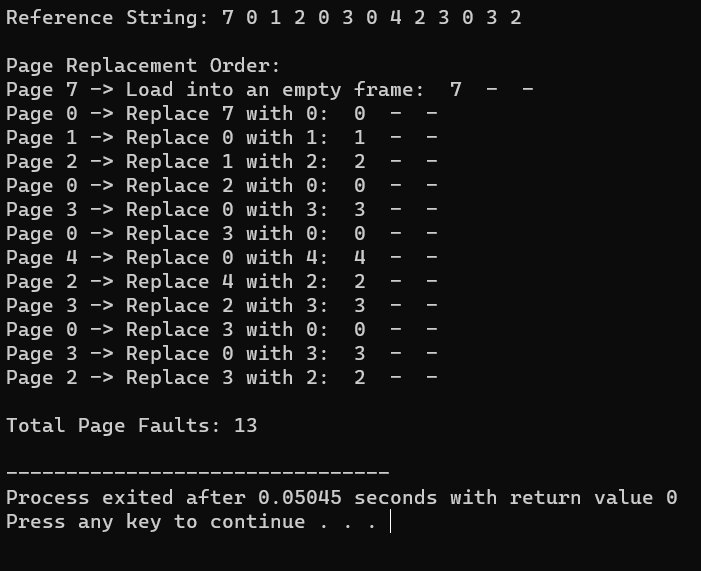
}

printf("\nTotal Page Faults: %d\n", pageFaults);

return 0;

}

**OUTPUT:**



**PROGRAM 33**

**CODE:**

#include <stdio.h> #include <stdlib.h>

#define MAX\_FRAMES 3

void printFrames(int frames[], int n) { for (int i = 0; i < n; i++) {

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

printf(" - ");

} else {

printf(" %d ", frames[i]);

}

}

printf("\n");

}

int main() {

int frames[MAX\_FRAMES];

for (int i = 0; i < MAX\_FRAMES; i++) { frames[i] = -1; // Initialize frames to -1 (empty)

}

int pageFaults = 0;

int referenceString[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2}; int n = sizeof(referenceString) / sizeof(referenceString[0]);

printf("Reference String: "); for (int i = 0; i < n; i++) {

printf("%d ", referenceString[i]);

}

printf("\n\n");

printf("Page Replacement Order:\n"); for (int i = 0; i < n; i++) {

int page = referenceString[i];

int pageFound = 0;

// Check if the page is already in memory (a page hit) for (int j = 0; j < MAX\_FRAMES; j++) {

if (frames[j] == page) { pageFound = 1; break;

}

}

if (!pageFound) {

printf("Page %d -> ", page);

int optimalPage = -1; int farthestDistance = 0;

for (int j = 0; j < MAX\_FRAMES; j++) { int futureDistance = 0;

for (int k = i + 1; k < n; k++) {

if (referenceString[k] == frames[j]) { break;

}

futureDistance++;

}

if (futureDistance > farthestDistance) { farthestDistance = futureDistance; optimalPage = j;

}

}

frames[optimalPage] = page;

printFrames(frames, MAX\_FRAMES); pageFaults++;

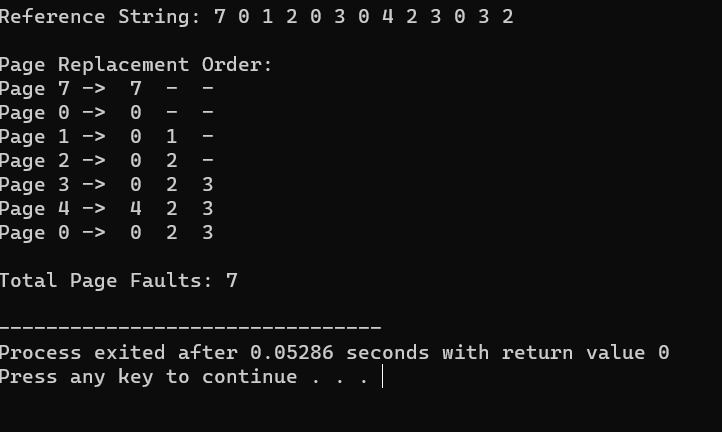
}

}

printf("\nTotal Page Faults: %d\n", pageFaults);

return 0;

}

**OUTPUT**

**PROGRAM 34**

**CODE:**

#include <stdio.h> #include <stdlib.h>

// Structure to represent a record struct Record {

int recordNumber;

char data[256]; // Adjust the size as needed for your records

};

int main() { FILE \*file;

struct Record record; int recordNumber;

// Open or create a file in write mode (for writing records) file = fopen("sequential\_file.txt", "w");

if (file == NULL) {

printf("Error opening the file.\n");

return 1;

}

// Write records sequentially to the file

printf("Enter records (Enter '0' as record number to exit):\n"); while (1) {

printf("Record Number: "); scanf("%d", &record.recordNumber); if (record.recordNumber == 0) {

break;

}

// Input data for the record printf("Data: ");

scanf(" %[^\n]", record.data);

// Write the record to the file

fwrite(&record, sizeof(struct Record), 1, file);

}

fclose(file);

// Reopen the file in read mode (for reading records) file = fopen("sequential\_file.txt", "r");

if (file == NULL) {

printf("Error opening the file.\n"); return 1;

}

// Read a specific record from the file while (1) {

printf("Enter the record number to read (0 to exit): "); scanf("%d", &recordNumber);

if (recordNumber == 0) { break;

}

// Read and display records up to the requested record while (fread(&record, sizeof(struct Record), 1, file)) {

printf("Record Number: %d\n", record.recordNumber); printf("Data: %s\n", record.data);

if (record.recordNumber == recordNumber) { break;

}

}

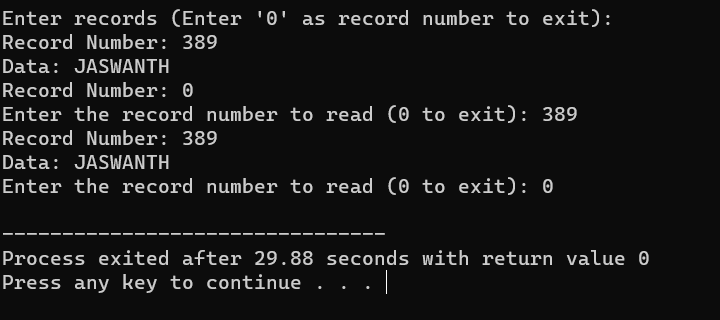
rewind(file); // Reset the file pointer to the beginning of the file

}

fclose(file); return 0;

}

**OUTPUT:**



**PROGRAM 35**

**CODE:**

#include <stdio.h>

#include <stdlib.h>

// Structure to represent a block struct Block {

int blockNumber;

char data[256]; // Adjust the size as needed for your blocks

};

int main() { FILE \*file;

struct Block block; int blockNumber;

// Create an index block that contains pointers to data blocks int indexBlock[100] = {0}; // Adjust the size as needed

// Open or create a file in write mode (for writing blocks) file = fopen("indexed\_file.txt", "w");

if (file == NULL) {

printf("Error opening the file.\n"); return 1;

}

// Write blocks and update the index block

printf("Enter blocks (Enter '0' as block number to exit):\n"); while (1) {

printf("Block Number: ");

scanf("%d", &block.blockNumber); if (block.blockNumber == 0) {

break;

}

// Input data for the block printf("Data: ");

scanf(" %[^\n]", block.data);

// Write the block to the file

fwrite(&block, sizeof(struct Block), 1, file);

// Update the index block with the pointer to the data block indexBlock[block.blockNumber] = ftell(file) - sizeof(struct Block);

}

fclose(file);

// Reopen the file in read mode (for reading blocks) file = fopen("indexed\_file.txt", "r");

if (file == NULL) {

printf("Error opening the file.\n"); return 1;

}

// Read a specific block from the file while (1) {

printf("Enter the block number to read (0 to exit): "); scanf("%d", &blockNumber);

if (blockNumber == 0) { break;

}

if (indexBlock[blockNumber] == 0) { printf("Block not found.\n");

} else {

// Seek to the data block using the index block fseek(file, indexBlock[blockNumber], SEEK\_SET); fread(&block, sizeof(struct Block), 1, file);

printf("Block Number: %d\n", block.blockNumber); printf("Data: %s\n", block.data);

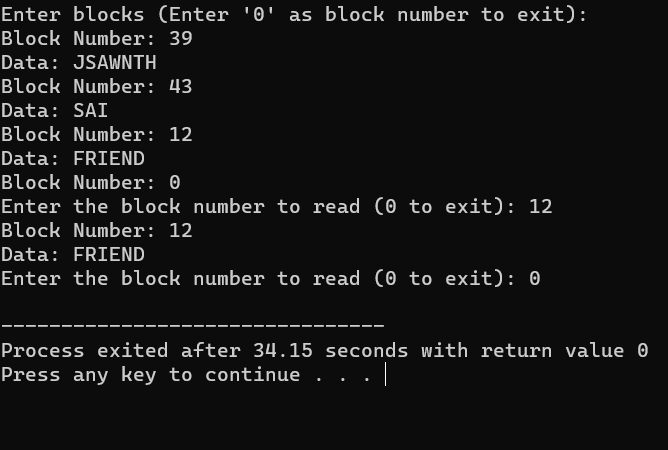
}

}

fclose(file); return 0;

}

**OUTPUT:**



**PROGRAM 36**

**CODE:**

#include <stdio.h> #include <stdlib.h>

// Structure to represent a block struct Block {

char data[256]; // Adjust the size as needed for your blocks struct Block\* next;

};

int main() {

struct Block\* firstBlock = NULL; // Pointer to the first block in the linked list

struct Block\* lastBlock = NULL; // Pointer to the last block in the linked list

int blockCount = 0; // Count of blocks in the linked list

int blockNumber; char data[256]; char choice;

printf("Linked Allocation Simulation\n");

while (1) {

printf("Enter 'W' to write a block, 'R' to read a block, or 'Q' to quit: "); scanf(" %c", &choice);

if (choice == 'Q' || choice == 'q') { break;

}

if (choice == 'W' || choice == 'w') { printf("Enter data for the block: "); scanf(" %[^\n]", data);

// Create a new block

struct Block\* newBlock = (struct Block\*)malloc(sizeof(struct Block)); for (int i = 0; i < 256; i++) {

newBlock->data[i] = data[i];

}

newBlock->next = NULL;

if (blockCount == 0) {

// This is the first block firstBlock = newBlock; lastBlock = newBlock;

} else {

// Link the new block to the last block lastBlock->next = newBlock; lastBlock = newBlock;

}

blockCount++;

} else if (choice == 'R' || choice == 'r') {

printf("Enter the block number to read (1-%d): ", blockCount); scanf("%d", &blockNumber);

if (blockNumber < 1 || blockNumber > blockCount) { printf("Invalid block number. The valid range is 1-%d.\n",

blockCount);

} else {

struct Block\* currentBlock = firstBlock; for (int i = 1; i < blockNumber; i++) {

currentBlock = currentBlock->next;

}

printf("Block %d Data: %s\n", blockNumber, currentBlock->data);

}

}

}

// Free the allocated memory for blocks before exiting struct Block\* currentBlock = firstBlock;

while (currentBlock != NULL) {

struct Block\* nextBlock = currentBlock->next;

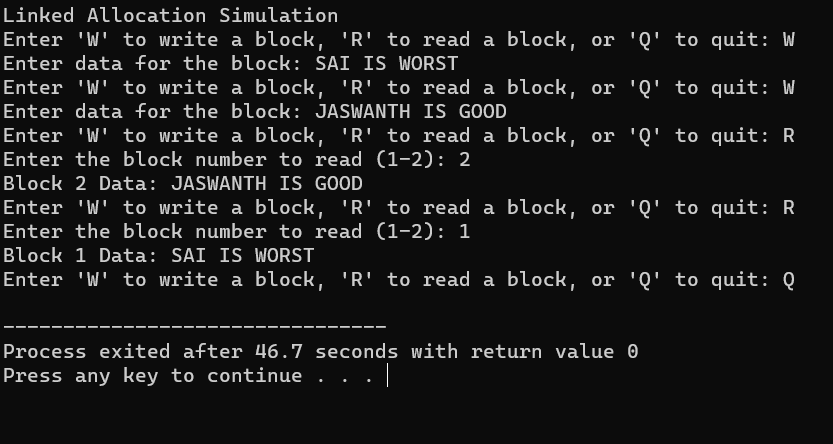
free(currentBlock); currentBlock = nextBlock;

}

return 0;

}

**OUTPUT:**



**PROGRAM 37**

**CODE:**

#include <stdio.h> #include <stdlib.h>

int main() {

int n, head, seek\_time = 0;

printf("Enter the number of disk requests: "); scanf("%d", &n);

int request\_queue[n];

printf("Enter the disk request queue:\n"); for (int i = 0; i < n; i++) {

scanf("%d", &request\_queue[i]);

}

printf("Enter the initial position of the disk head: "); scanf("%d", &head);

// FCFS Scheduling

printf("\nFCFS Disk Scheduling:\n");

printf("Head Movement Sequence: %d", head); for (int i = 0; i < n; i++) {

seek\_time += abs(head - request\_queue[i]); head = request\_queue[i];

printf(" -> %d", head);

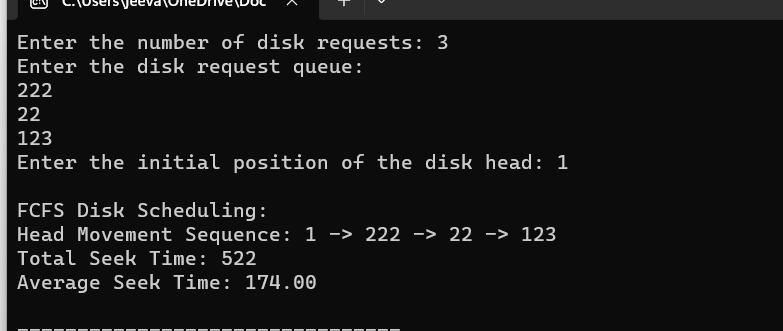
}

printf("\nTotal Seek Time: %d\n", seek\_time); printf("Average Seek Time: %.2f\n", (float) seek\_time / n);

return 0;

}

**OUTPUT:-**



**PROGRAM 38**

**CODE:**

#include <stdio.h> #include <stdlib.h>

int main() {

int n, head, seek\_time = 0;

printf("Enter the number of disk requests: "); scanf("%d", &n);

int request\_queue[n];

printf("Enter the disk request queue:\n");

for (int i = 0; i < n; i++) { scanf("%d", &request\_queue[i]);

}

printf("Enter the initial position of the disk head: "); scanf("%d", &head);

// Sort the request queue to simplify SCAN algorithm for (int i = 0; i < n - 1; i++) {

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

if (request\_queue[i] > request\_queue[j]) { int temp = request\_queue[i]; request\_queue[i] = request\_queue[j]; request\_queue[j] = temp;

}

}

}

// SCAN (Elevator) Scheduling

printf("\nSCAN (Elevator) Disk Scheduling:\n"); int start = 0;

int end = n - 1;

int current\_direction = 1; // 1 for moving right, -1 for moving left

while (start <= end) {

if (current\_direction == 1) {

for (int i = start; i <= end; i++) {

if (request\_queue[i] >= head) {

seek\_time += abs(head - request\_queue[i]); head = request\_queue[i];

start = i + 1; break;

}

}

current\_direction = -1; // Change direction

} else {

for (int i = end; i >= start; i--) {

if (request\_queue[i] <= head) {

seek\_time += abs(head - request\_queue[i]); head = request\_queue[i];

end = i - 1; break;

}

}

current\_direction = 1; // Change direction

}

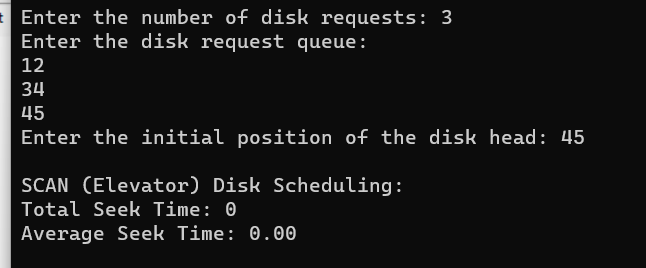
}

printf("Total Seek Time: %d\n", seek\_time); printf("Average Seek Time: %.2f\n", (float)seek\_time / n);

return 0;

}

## Output:-



**PROGRAM 39**

**CODE:**

#include <stdio.h>

#include <stdlib.h>

int main() {

int n, head, seek\_time = 0;

printf("Enter the number of disk requests: "); scanf("%d", &n);

int request\_queue[n];

printf("Enter the disk request queue:\n"); for (int i = 0; i < n; i++) {

scanf("%d", &request\_queue[i]);

}

printf("Enter the initial position of the disk head: "); scanf("%d", &head);

// Sort the request queue for simplicity for (int i = 0; i < n - 1; i++) {

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

if (request\_queue[i] > request\_queue[j]) { int temp = request\_queue[i]; request\_queue[i] = request\_queue[j]; request\_queue[j] = temp;

}

}

}

// C-SCAN Scheduling

printf("\nC-SCAN Disk Scheduling:\n"); int start = 0;

int end = n - 1;

while (start <= end) {

for (int i = start; i <= end; i++) { if (request\_queue[i] >= head) {

seek\_time += abs(head - request\_queue[i]); head = request\_queue[i];

start = i + 1;

}

}

// Move the head to the end in the current direction seek\_time += abs(head - 0);

head = 0;

// Change direction to the opposite side seek\_time += abs(head - request\_queue[end]); head = request\_queue[end];

end = n - 2; // Exclude the last request, as it has already been served

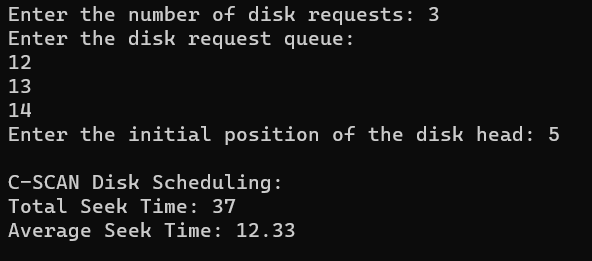
}

printf("Total Seek Time: %d\n", seek\_time); printf("Average Seek Time: %.2f\n", (float)seek\_time / n);

return 0;

}

## OUTPUT:-



**PROGRAM 40**

**CODE:**

#include <stdio.h> #include <stdlib.h> #include <sys/stat.h>

int main() {

char filename[] = "file.txt";

int new\_permissions = S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IWGRP | S\_IROTH; // rw-rw-r--

if (chmod(filename, new\_permissions) == 0) { printf("File permissions changed successfully.\n");

} else {

perror("chmod"); return 1;

}

return 0;

}

## OUTPUT:

