**Operating system**

**Name: ROHITH.K.S Regno: 192210070**

**1.Create a new process by invoking the appropriate system call. Get the process identifier of the currently running process and its respective parent using system calls and display the same using a C program.**

PROGRAM:

#include<stdio.h>

#include<unistd.h>

int main(){

Pid\_t child\_pid=fork();

If (child\_pid == -1){

Perror(“fork failed);

Return 1;

}

else{

printf(" parent Process : PID= %d, child PID=%d\n", getpid() ),child\_pid);}

return 0;

}

INPUT AND OUTPUT:



**2.To identify the system calls to copy the content of one file to another and illustrate the same using a C program.**

Program:

#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);

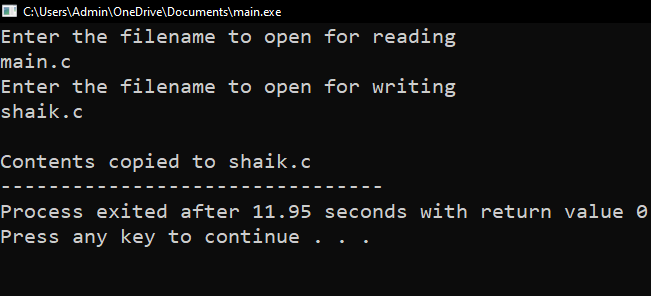
fclose(fptr1);

fclose(fptr2);

return 0;

}

INPUT AND OUTPUT:



**3.ToDesign a CPU scheduling program with C using First Come First Served technique with the following considerations. a. All processes are activated at time 0. b. Assume that no process waits on I/O devices.**

PROGRAM:

#include<stdio.h>

void main()

{

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

{

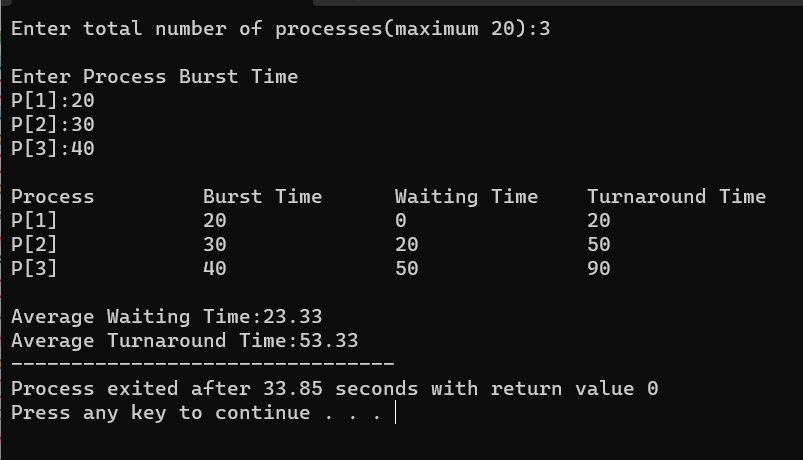
tat[i]=bt[i]+wt[i]; avwt+=wt[i]; avtat+=tat[i];printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);

} avwt/=i; avtat/=i;printf("\n\nAverage Waiting Time:%.2f",avwt);

printf("\nAverage Turnaround Time:%.2f",avtat);

}

INPUT AND OUTPUT:



**4.Construct a scheduling program with C that selects the waiting process with the smallest execution time to execute next.**

PROGRAM:

#include <stdio.h>

struct Process {

int id;

int burst\_time;

};

int main() {

int n;

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

scanf("%d", &n);

struct Process processes[n];

int remaining\_time[n];

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

processes[i].id = i + 1;

printf("Enter burst time for Process %d: ", i + 1);

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

remaining\_time[i] = processes[i].burst\_time;

}

int current\_time = 0;

printf("\nProcess execution order:\n");

while (1) {

int smallest = -1;

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

if (remaining\_time[i] > 0) {

if (smallest == -1 || remaining\_time[i] < remaining\_time[smallest]) {

smallest = i;

}

}

}

if (smallest == -1) {

break;

}

printf("Process %d (Burst Time: %d) is executing from time %d to ", processes[smallest].id, processes[smallest].burst\_time, current\_time);

current\_time += 1;

remaining\_time[smallest] -= 1;

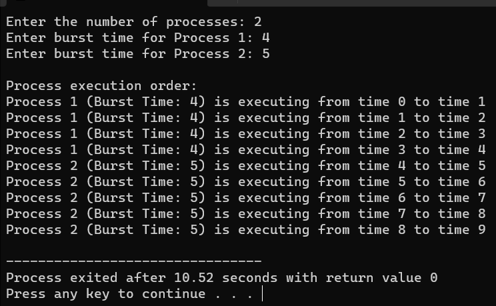
printf("time %d\n", current\_time);

}

return 0;

}

OUTPUT:



**5. Construct a scheduling program with C that selects the waiting process with the highest priority to execute next.**

PROGRAM:

#include <stdio.h>

struct Process {

int id;

int priority;

int burst\_time;

};

int main() {

int n;

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

scanf("%d", &n);

struct Process processes[n];

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

processes[i].id = i + 1;

printf("Enter priority and burst time for Process %d: ", i + 1);

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

}

printf("Process Execution Order:\n");

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

int highest\_priority\_idx = 0;

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

if (processes[j].priority < processes[highest\_priority\_idx].priority) {

highest\_priority\_idx = j;

}

}

printf("Executing Process %d (Priority: %d, Burst Time: %d)\n",

processes[highest\_priority\_idx].id, processes[highest\_priority\_idx].priority, processes[highest\_priority\_idx].burst\_time);

for (int j = highest\_priority\_idx; j < n - 1; j++) {

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

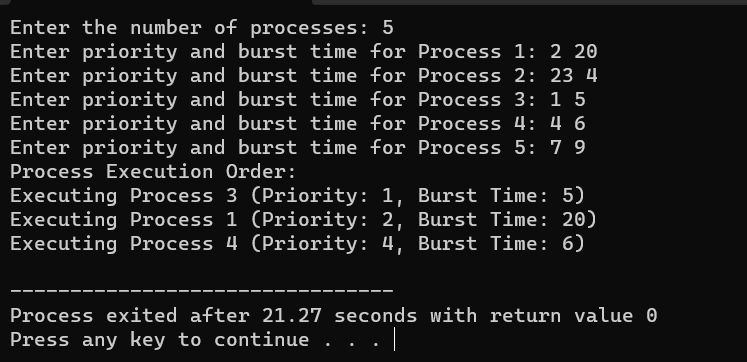
}

n--;

}

return 0;

**}OUTPUT:**



**6.** **Construct a C program to implement pre-emptive priority scheduling algorithm**.

PROGRAM:

#include <stdio.h>

struct Process {

int id, arrival\_time, burst\_time, priority;

};

int main() {

int n;

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

scanf("%d", &n);

struct Process processes[n];

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

processes[i].id = i + 1;

printf("Enter arrival time, burst time, and priority for process %d: ", processes[i].id);

scanf("%d %d %d", &processes[i].arrival\_time, &processes[i].burst\_time, &processes[i].priority);

}

int current\_time = 0;

int total\_time = 0;

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

total\_time += processes[i].burst\_time;

}

printf("Gantt Chart: ");

while (current\_time < total\_time) {

int highest\_priority = 9999; // A high value to represent the lowest priority

int selected\_process = -1;

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

if (processes[i].arrival\_time <= current\_time && processes[i].burst\_time > 0 && processes[i].priority < highest\_priority) {

highest\_priority = processes[i].priority;

selected\_process = i;

}

}

if (selected\_process == -1) {

printf("Idle ");

current\_time++;

} else {

printf("P%d ", processes[selected\_process].id);

processes[selected\_process].burst\_time--;

current\_time++;

}

}

printf("\n\nProcess\tArrival Time\tBurst Time\tPriority\n");

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

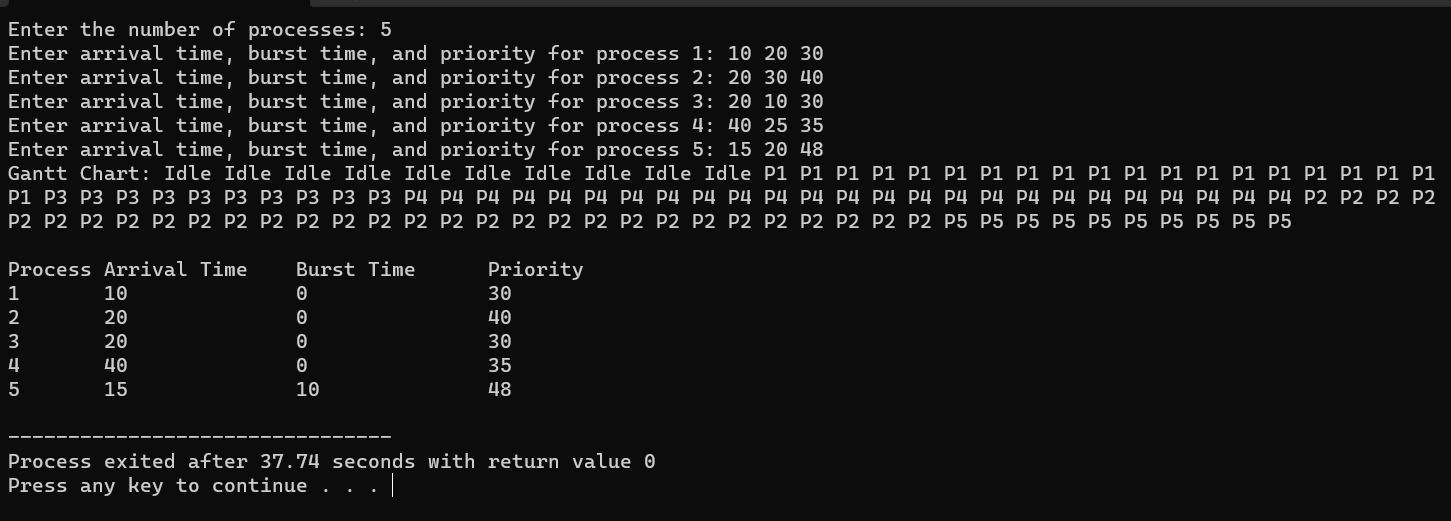
printf("%d\t%d\t\t%d\t\t%d\n", processes[i].id, processes[i].arrival\_time, processes[i].burst\_time, processes[i].priority);

}

return 0;

}

**OUTPUT:**

****

**7.** **Construct a C program to implement non-preemptive SJF algorithm**

PROGRAM:

#include <stdio.h>

struct Process {

int id, arrival\_time, burst\_time;

};

int main() {

int n;

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

scanf("%d", &n);

struct Process processes[n];

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

processes[i].id = i + 1;

printf("Enter arrival time and burst time for process %d: ", processes[i].id);

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

}

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

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

if (processes[i].burst\_time > processes[j].burst\_time) {

struct Process temp = processes[i];

processes[i] = processes[j];

processes[j] = temp;

}

}

}

int waiting\_time = 0, turnaround\_time = 0;

printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");

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

turnaround\_time += processes[i].burst\_time;

printf("%d\t%d\t\t%d\t\t%d\n", processes[i].id, processes[i].burst\_time, waiting\_time, turnaround\_time);

waiting\_time += processes[i].burst\_time;

}

float avg\_waiting\_time = (float)waiting\_time / n;

float avg\_turnaround\_time = (float)turnaround\_time / n;

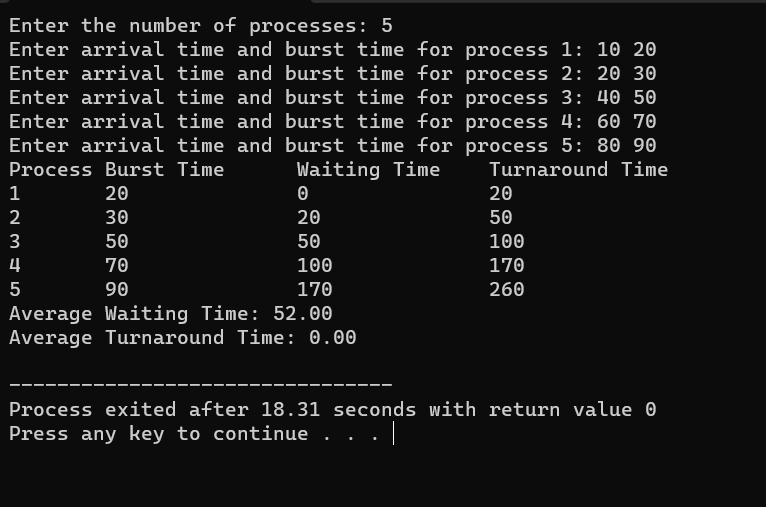
printf("Average Waiting Time: %.2f\n", avg\_waiting\_time);

printf("Average Turnaround Time: %.2f\n");

return 0;

}

OUTPUT:



**8. Construct a C program to simulate Round Robin scheduling algorithm with C**

**PROGRAM:**

#include <stdio.h>

struct Process {

int id, burstTime, remainingTime;

};

void roundRobin(struct Process processes[], int n, int quantum) {

int time = 0, completed = 0;

while (completed < n) {

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

if (processes[i].remainingTime > 0) {

int execTime = (processes[i].remainingTime < quantum) ? processes[i].remainingTime : quantum;

processes[i].remainingTime -= execTime;

time += execTime;

printf("Process %d runs for %d units.\n", processes[i].id, execTime);

if (processes[i].remainingTime == 0) {

completed++;

printf("Process %d has completed.\n", processes[i].id);

}

}

}

}

}

int main() {

int n, quantum;

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

scanf("%d", &n);

struct Process processes[n];

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

processes[i].id = i + 1;

printf("Enter burst time for process %d: ", processes[i].id);

scanf("%d", &processes[i].burstTime);

processes[i].remainingTime = processes[i].burstTime;

}

printf("Enter time quantum: ");

scanf("%d", &quantum);

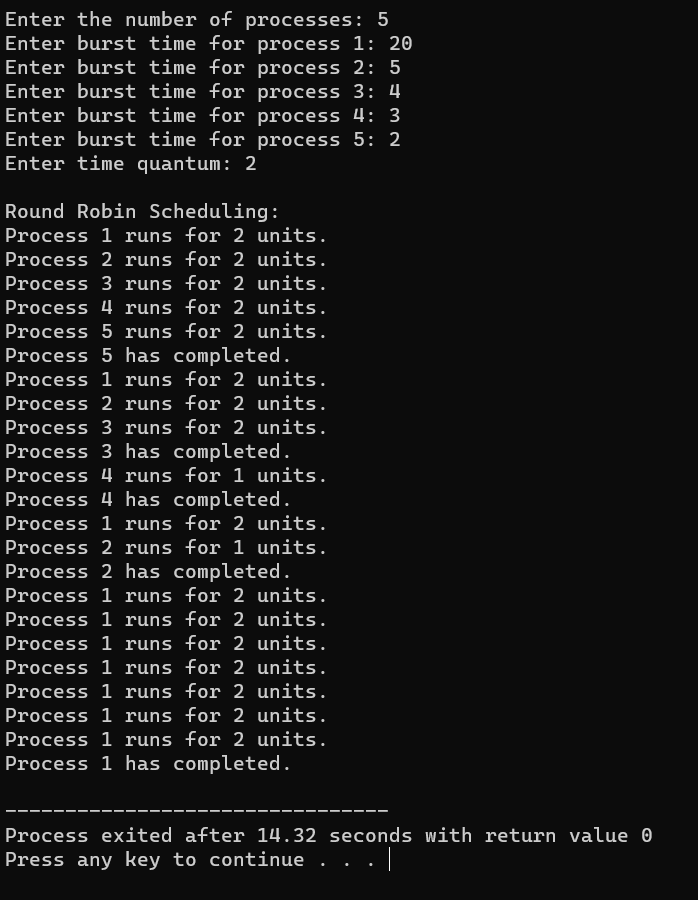
printf("\nRound Robin Scheduling:\n");

roundRobin(processes, n, quantum);

return 0;

}

OUTPUT:



**9. Illustrate the concept of inter-process communication using shared memory with a C program.**

Program:

#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);

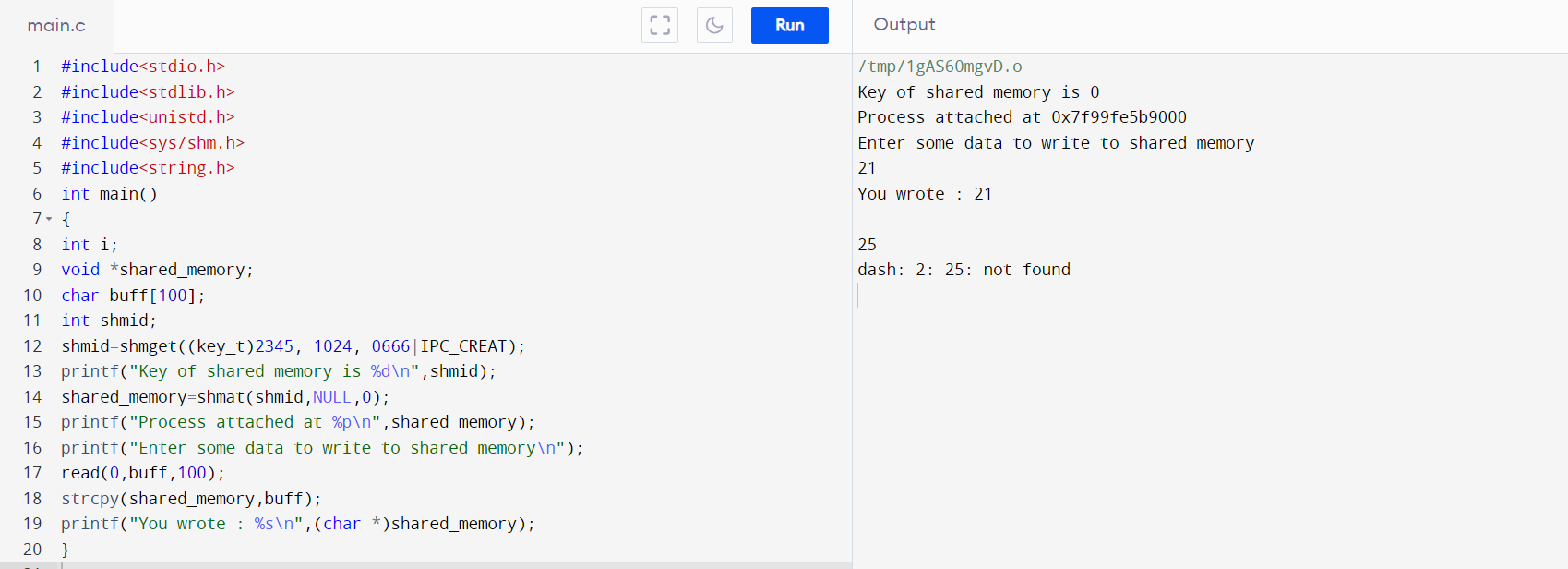
printf("Enter some data to write to shared memory\n");

read(0,buff,100);

strcpy(shared\_memory,buff);

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

}



10. Illustrate the concept of inter-process communication using message queue with a C program.

Program:

#include <stdio.h>

struct clientData

{

unsigned int acctNum;

char lastName[ 15 ];

char firstName[ 10 ];

double balance;

};

int main( void )

{

unsigned int i;

struct clientData blankClient = { 0, "", "", 0.0 };

FILE \*cfPtr;

if ( ( cfPtr = fopen( "credit.dat", "wb" ) ) == NULL )

{

puts( "File could not be opened." );

}

else

{

for ( i = 1; i <= 100; ++i )

{

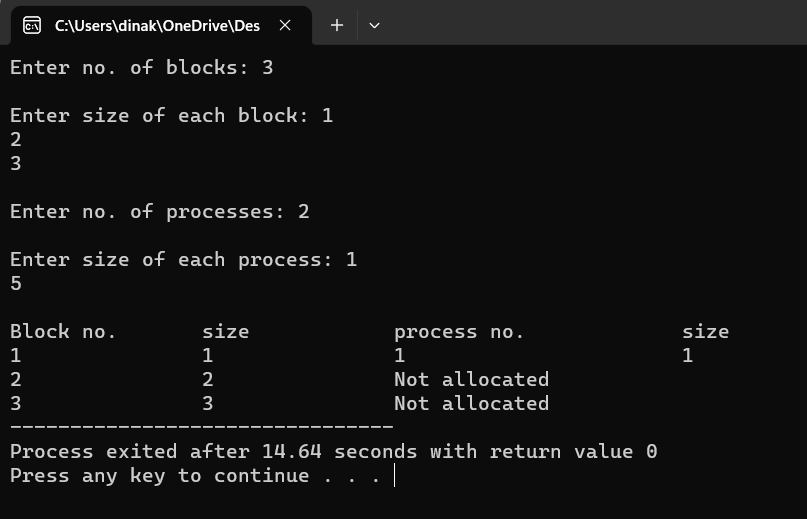
fwrite( &blankClient, sizeof( struct clientData ), 1, cfPtr );

}

fclose ( cfPtr );

}

}



**11. Illustrate the concept of multithreading using a C program.**

Program:

#include<stdio.h>

#include <pthread.h>

int g = 0;

void \*myThreadFun(void \*vargp)

{ int \*myid = (int \*)vargp;

static int s = 0; ++s; ++g;

printf("Thread ID: %d, Static: %d, Global: %d\n", \*myid, ++s, ++g);

}

int main()

{ int i;

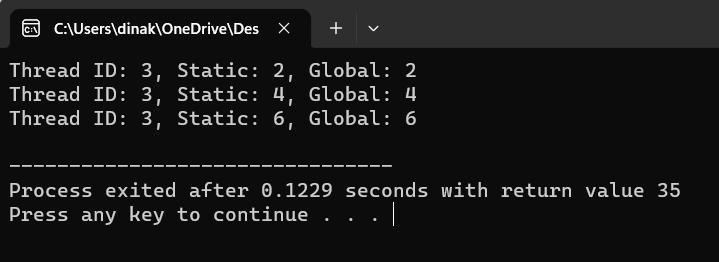
pthread\_t tid;

for (i = 0; i < 3; i++)

pthread\_create(&tid, NULL, myThreadFun, (void \*)&tid);

pthread\_exit(NULL);

}



**12. Design a C program to simulate the concept of Dining-Philosophers problem**

Program:

#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);

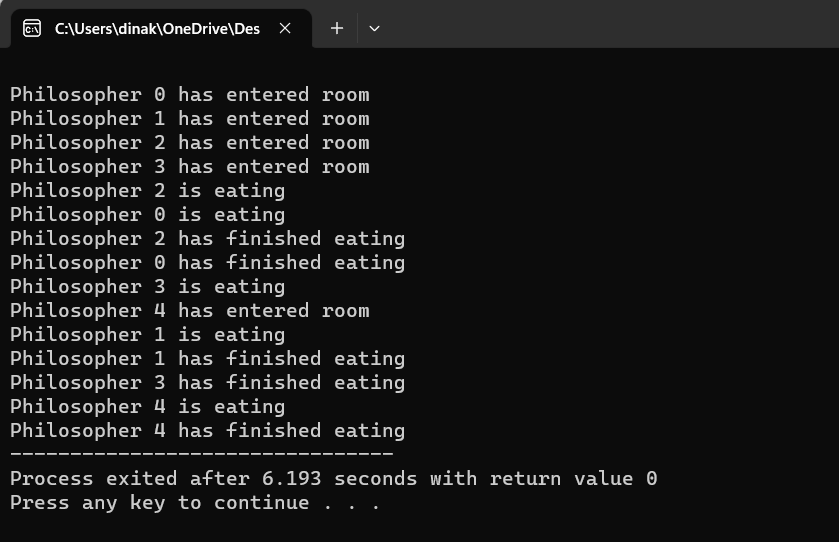
}

void eat(int phil)

{

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

}



**13. Construct a C program for implementation of memory allocation using first fit strategy.**

Program:

#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)

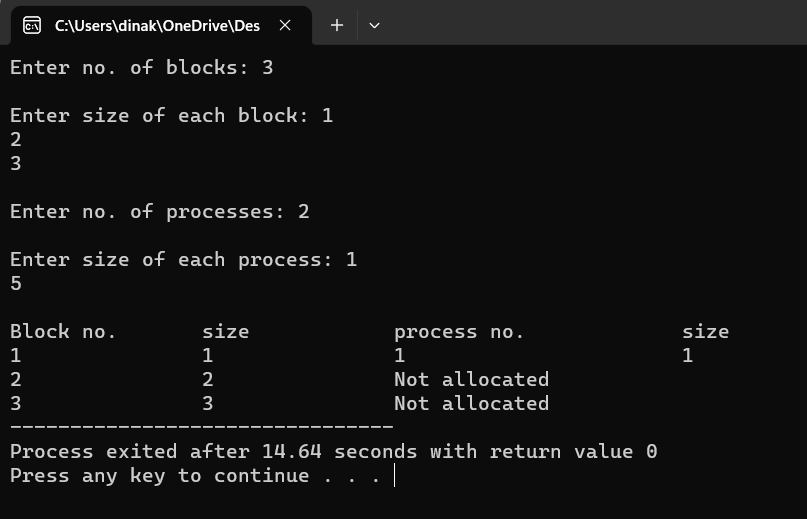
printf("%d\t\t\t%d",allocation[i]+1,psize[allocation[i]]);

else

printf("Not allocated");

}

}



**14. Construct a C program to organize the file using single level directory.**

Program:

#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);

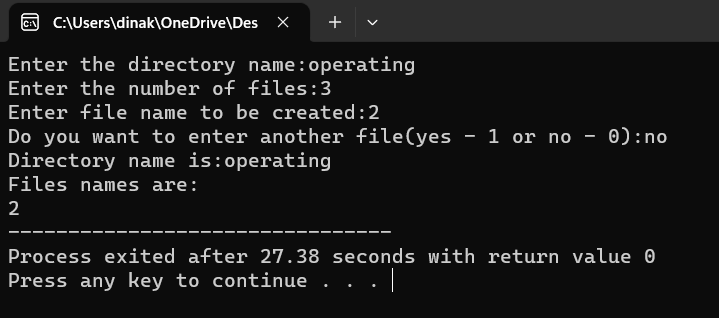
printf("Files names are:");

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

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

getch();

}



**15. Design a C program to organize the file using two level directory structure**

**Program:**

#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");

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

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");

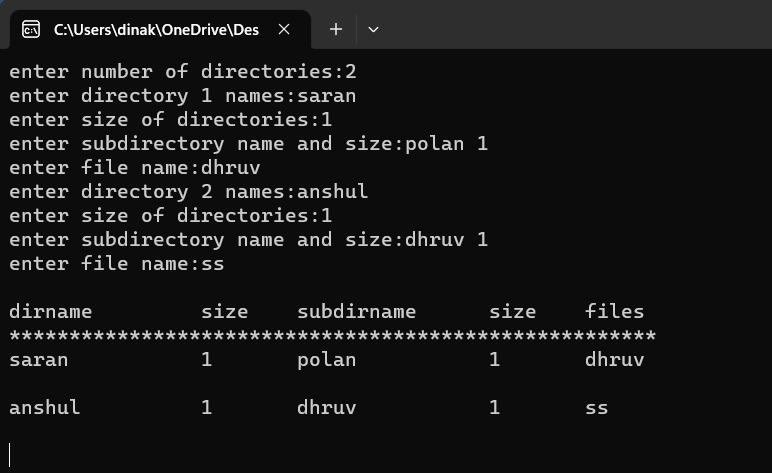
}

printf("\n");

}

getch();

}



16. Develop a C program for implementing random access file for processing the employee details

Program:

#include <stdio.h>

struct clientData

{

unsigned int acctNum;

char lastName[ 15 ];

char firstName[ 10 ];

double balance;

};

int main( void )

{

unsigned int i;

struct clientData blankClient = { 0, "", "", 0.0 };

FILE \*cfPtr;

if ( ( cfPtr = fopen( "credit.dat", "wb" ) ) == NULL )

{

puts( "File could not be opened." );

}

else

{

for ( i = 1; i <= 100; ++i )

{

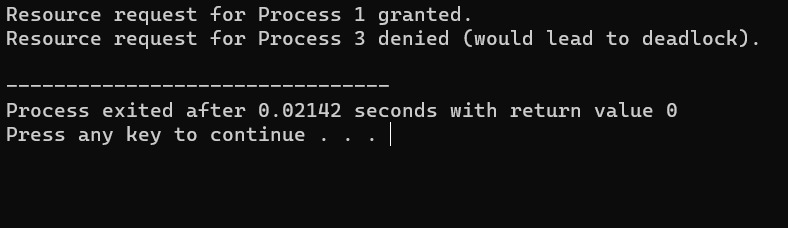
fwrite( &blankClient, sizeof( struct clientData ), 1, cfPtr );

}

fclose ( cfPtr );

}

}



17. Illustrate the deadlock avoidance concept by simulating Banker’s algorithm with C.

Program:

#include <stdio.h>

// Define the number of processes and resources

#define NUM\_PROCESSES 5

#define NUM\_RESOURCES 3

// Available resources

int available[NUM\_RESOURCES] = {3, 3, 2};

// Maximum demand of each process

int max\_demand[NUM\_PROCESSES][NUM\_RESOURCES] = {

{7, 5, 3},

{3, 2, 2},

{9, 0, 2},

{2, 2, 2},

{4, 3, 3}

};

// Allocated resources to processes

int allocation[NUM\_PROCESSES][NUM\_RESOURCES] = {

{0, 1, 0},

{2, 0, 0},

{3, 0, 2},

{2, 1, 1},

{0, 0, 2}

};

// Need resources for each process

int need[NUM\_PROCESSES][NUM\_RESOURCES];

// Function to check if the system is in a safe state

int isSafeState(int process, int request[]) {

// Step 1: Check if the request is less than or equal to the need

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

if (request[i] > need[process][i]) {

return 0; // Request exceeds need

}

}

// Step 2: Check if the request is less than or equal to the available resources

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

if (request[i] > available[i]) {

return 0; // Request exceeds available resources

}

}

// Try to allocate the resources and see if it remains safe

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

available[i] -= request[i];

allocation[process][i] += request[i];

need[process][i] -= request[i];

}

int finish[NUM\_PROCESSES] = {0};

int work[NUM\_RESOURCES];

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

work[i] = available[i];

}

int count = 0;

while (count < NUM\_PROCESSES) {

int found = 0;

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

if (finish[i] == 0) {

int j;

for (j = 0; j < NUM\_RESOURCES; j++) {

if (need[i][j] > work[j]) {

break;

}

}

if (j == NUM\_RESOURCES) {

for (j = 0; j < NUM\_RESOURCES; j++) {

work[j] += allocation[i][j];

}

finish[i] = 1;

found = 1;

count++;

}

}

}

if (!found) {

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

available[i] += request[i];

allocation[process][i] -= request[i];

need[process][i] += request[i];

}

return 0; // System is not in a safe state

}

}

// If we reach here, the system is in a safe state

return 1;

}

// Function to allocate resources to a process

void allocateResources(int process, int request[]) {

if (isSafeState(process, request)) {

printf("Resource request for Process %d granted.\n", process);

} else {

printf("Resource request for Process %d denied (would lead to deadlock).\n", process);

}

}

int main() {

// Initialize the 'need' matrix

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

for (int j = 0; j < NUM\_RESOURCES; j++) {

need[i][j] = max\_demand[i][j] - allocation[i][j];

}

}

// Test resource allocation requests

int process = 1;

int request[NUM\_RESOURCES] = {1, 0, 2};

allocateResources(process, request);

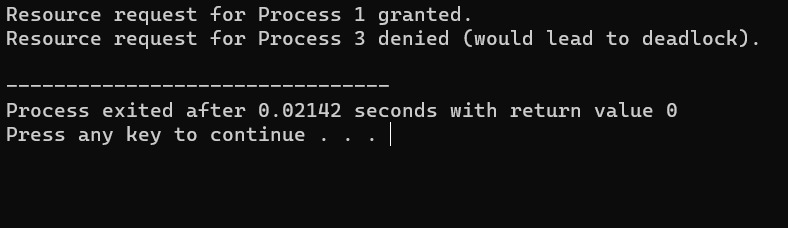
process = 3;

int request2[NUM\_RESOURCES] = {0, 2, 0};

allocateResources(process, request2);

return 0;

}



18. Construct a C program to simulate producer-consumer problem using semaphores.

Program:

#include<stdio.h>

#include<stdlib.h>

int mutex=1,full=0,empty=3,x=0;

int main()

{

int n;

void producer();

void consumer();

int wait(int);

int signal(int);

printf("\n1.Producer\n2.Consumer\n3.Exit");

while(1)

{

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;

}

}

return 0;

}

int wait(int s)

{

return (--s);

}

int signal(int s)

{ return(++s);

}void producer()

{ mutex=wait(mutex);

full=signal(full);

empty=wait(empty);

x++;

printf("\nProducer produces the item %d",x);

mutex=signal(mutex);

}void consumer()

{

mutex=wait(mutex);

full=wait(full);

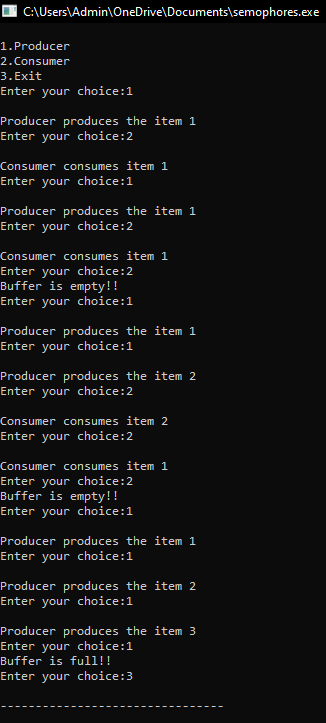
empty=signal(empty);

printf("\nConsumer consumes item %d",x);

x--;

mutex=signal(mutex);

}



19. Design a C program to implement process synchronization using mutex locks.

Program:

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#define NUM\_THREADS 2

#define NUM\_INCREMENTS 10000

int sharedCounter = 0;

pthread\_mutex\_t mutex;

void\* incrementCounter(void\* threadID) {

int id = \*((int\*)threadID);

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

pthread\_mutex\_lock(&mutex);

sharedCounter++;

pthread\_mutex\_unlock(&mutex);

}

printf("Thread %d finished\n", id);

pthread\_exit(NULL);

}

int main() {

pthread\_t threads[NUM\_THREADS];

int threadIDs[NUM\_THREADS];

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

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

threadIDs[i] = i;

int result = pthread\_create(&threads[i], NULL, incrementCounter, &threadIDs[i]);

if (result) {

fprintf(stderr, "Error creating thread %d: %d\n", i, result);

exit(-1);

}

}

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

pthread\_join(threads[i], NULL);}

pthread\_mutex\_destroy(&mutex);

printf("Shared Counter: %d\n", sharedCounter);

return 0;

}

20.Construct a C program to simulate Reader-Writer problem using Semaphores.

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

#define NUM\_READERS 5

#define NUM\_WRITERS 2

int shared\_data = 0;

sem\_t mutex, writeblock;

int reader\_count = 0;

void \*reader(void \*arg) {

while (1) {

usleep(100000); // Sleep for a short time to simulate reading

sem\_wait(&mutex);

reader\_count++;

if (reader\_count == 1) {

sem\_wait(&writeblock);

}

sem\_post(&mutex);

// Reading the shared data

printf("Reader is reading: %d\n", shared\_data);

sem\_wait(&mutex);

reader\_count--;

if (reader\_count == 0) {

sem\_post(&writeblock);

}

sem\_post(&mutex);

}

}

void \*writer(void \*arg) {

while (1) {

sem\_wait(&writeblock);

// Modifying the shared data

shared\_data++;

printf("Writer is writing: %d\n", shared\_data);

sem\_post(&writeblock);

usleep(100000); // Sleep for a short time to simulate writing

}

}

int main() {

pthread\_t reader\_threads[NUM\_READERS];

pthread\_t writer\_threads[NUM\_WRITERS];

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

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

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

pthread\_create(&reader\_threads[i], NULL, reader, NULL);

}

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

pthread\_create(&writer\_threads[i], NULL, writer, NULL);

}

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

pthread\_join(reader\_threads[i], NULL);

}

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

pthread\_join(writer\_threads[i], NULL);

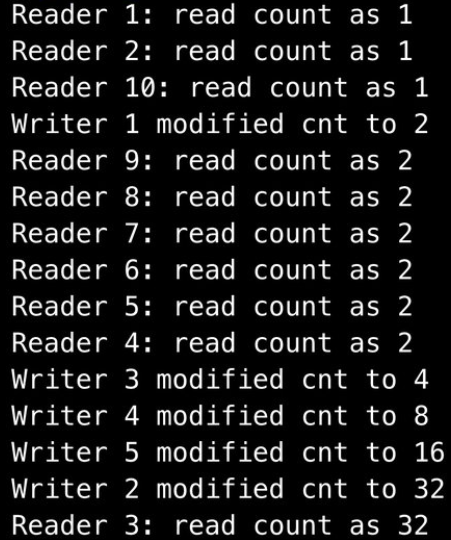
}

sem\_destroy(&mutex);

sem\_destroy(&writeblock);

return 0;

}



21. Develop a C program to implement worst fit algorithm of memory management.

#include <stdio.h>

#define MAX\_MEMORY 1000

int memory[MAX\_MEMORY];

// Function to initialize memory

void initializeMemory() {

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

memory[i] = -1; // -1 indicates that the memory is unallocated

}

}

// Function to display memory status

void displayMemory() {

int i, j;

int count = 0;

printf("Memory Status:\n");

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

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

count++;

j = i;

while (memory[j] == -1 && j < MAX\_MEMORY) {

j++;

}

printf("Free memory block %d-%d\n", i, j - 1);

i = j - 1;

}

}

if (count == 0) {

printf("No free memory available.\n");

}

}

// Function to allocate memory using worst-fit algorithm

void allocateMemory(int processId, int size) {

int start = -1;

int blockSize = 0;

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

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

if (blockSize == 0) {

start = i;

}

blockSize++;

} else {

blockSize = 0;

}

if (blockSize >= size) {

break;

}

}

if (blockSize >= size) {

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

memory[i] = processId;

}

printf("Allocated memory block %d-%d to Process %d\n", start, start + size - 1, processId);

} else {

printf("Memory allocation for Process %d failed (not enough contiguous memory).\n", processId);

}

}

// Function to deallocate memory

void deallocateMemory(int processId) {

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

if (memory[i] == processId) {

memory[i] = -1;

}

}

printf("Memory released by Process %d\n", processId);

}

int main() {

initializeMemory();

displayMemory();

allocateMemory(1, 200);

displayMemory();

allocateMemory(2, 300);

displayMemory();

deallocateMemory(1);

displayMemory();

allocateMemory(3, 400);

displayMemory();

return 0;

}



22. Construct a C program to implement best fit algorithm of memory management.

#include <stdio.h>

#define MAX\_MEMORY 1000

int memory[MAX\_MEMORY];

// Function to initialize memory

void initializeMemory() {

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

memory[i] = -1; // -1 indicates that the memory is unallocated

}

}

// Function to display memory status

void displayMemory() {

int i, j;

int count = 0;

printf("Memory Status:\n");

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

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

count++;

j = i;

while (memory[j] == -1 && j < MAX\_MEMORY) {

j++;

}

printf("Free memory block %d-%d\n", i, j - 1);

i = j - 1;

}

}

if (count == 0) {

printf("No free memory available.\n");

}

}

// Function to allocate memory using best-fit algorithm

void allocateMemory(int processId, int size) {

int start = -1;

int blockSize = MAX\_MEMORY;

int bestStart = -1;

int bestSize = MAX\_MEMORY;

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

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

if (blockSize == MAX\_MEMORY) {

start = i;

}

blockSize++;

} else {

if (blockSize >= size && blockSize < bestSize) {

bestSize = blockSize;

bestStart = start;

}

blockSize = 0;

}

}

if (bestSize >= size) {

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

memory[i] = processId;

}

printf("Allocated memory block %d-%d to Process %d\n", bestStart, bestStart + size - 1, processId);

} else {

printf("Memory allocation for Process %d failed (not enough contiguous memory).\n", processId);

}

}

// Function to deallocate memory

void deallocateMemory(int processId) {

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

if (memory[i] == processId) {

memory[i] = -1;

}

}

printf("Memory released by Process %d\n", processId);

}

int main() {

initializeMemory();

displayMemory();

allocateMemory(1, 200);

displayMemory();

allocateMemory(2, 300);

displayMemory();

deallocateMemory(1);

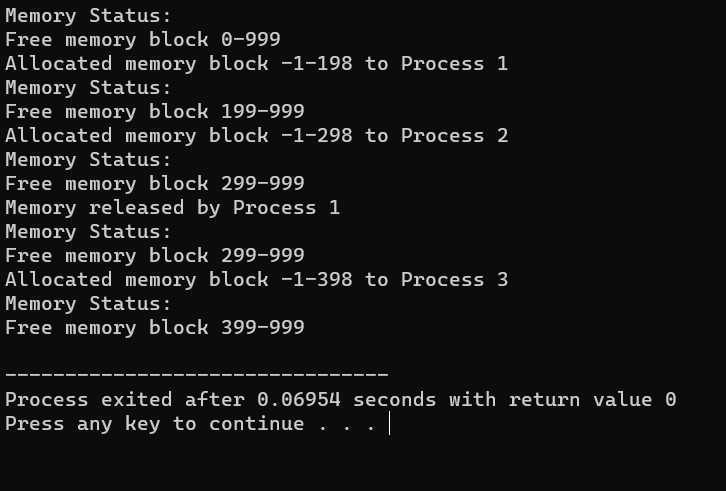
displayMemory();

allocateMemory(3, 400);

displayMemory();

return 0;

}



23. Construct a C program to implement first fit algorithm of memory management.

#include <stdio.h>

#define MAX\_MEMORY 1000

int memory[MAX\_MEMORY];

// Function to initialize memory

void initializeMemory() {

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

memory[i] = -1; // -1 indicates that the memory is unallocated

}

}

// Function to display memory status

void displayMemory() {

int i, j;

int count = 0;

printf("Memory Status:\n");

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

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

count++;

j = i;

while (memory[j] == -1 && j < MAX\_MEMORY) {

j++;

}

printf("Free memory block %d-%d\n", i, j - 1);

i = j - 1;

}

}

if (count == 0) {

printf("No free memory available.\n");

}

}

// Function to allocate memory using first-fit algorithm

void allocateMemory(int processId, int size) {

int start = -1;

int blockSize = 0;

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

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

if (blockSize == 0) {

start = i;

}

blockSize++;

} else {

blockSize = 0;

}

if (blockSize >= size) {

break;

}

}

if (blockSize >= size) {

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

memory[i] = processId;

}

printf("Allocated memory block %d-%d to Process %d\n", start, start + size - 1, processId);

} else {

printf("Memory allocation for Process %d failed (not enough contiguous memory).\n", processId);

}

}

// Function to deallocate memory

void deallocateMemory(int processId) {

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

if (memory[i] == processId) {

memory[i] = -1;

}

}

printf("Memory released by Process %d\n", processId);

}

int main() {

initializeMemory();

displayMemory();

allocateMemory(1, 200);

displayMemory();

allocateMemory(2, 300);

displayMemory();

deallocateMemory(1);

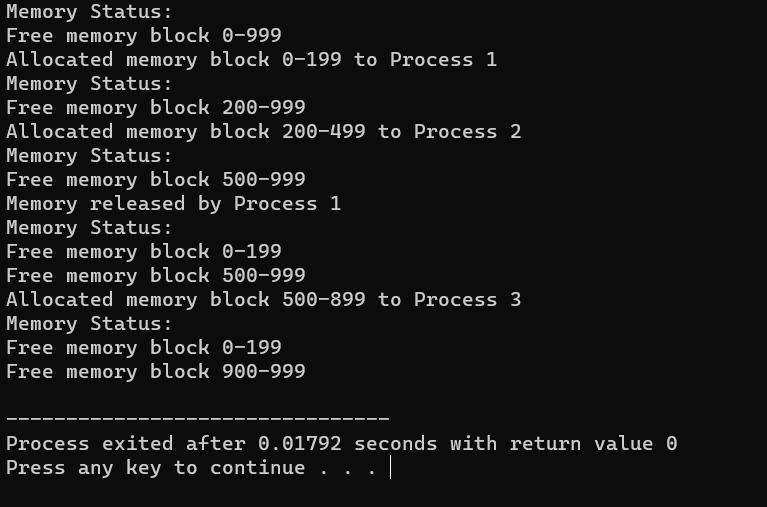
displayMemory();

allocateMemory(3, 400);

displayMemory();

return 0;

}



24. Design a C program to demonstrate UNIX system calls for file management.

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/types.h>

#include <sys/stat.h>

int main() {

int fd;

char buffer[100];

// Creating a new file

fd = creat("sample.txt", S\_IRWXU);

if (fd == -1) {

perror("create");

exit(1);

} else {

printf("File 'sample.txt' created successfully.\n");

close(fd);

}

// Opening an existing file for writing

fd = open("sample.txt", O\_WRONLY | O\_APPEND);

if (fd == -1) {

perror("open");

exit(1);

} else {

printf("File 'sample.txt' opened for writing.\n");

}

// Writing data to the file

write(fd, "Hello, World!\n", 14);

printf("Data written to 'sample.txt'.\n");

close(fd);

// Opening the file for reading

fd = open("sample.txt", O\_RDONLY);

if (fd == -1) {

perror("open");

exit(1);

} else {

printf("File 'sample.txt' opened for reading.\n");

}

// Reading data from the file

int bytesRead = read(fd, buffer, sizeof(buffer));

if (bytesRead == -1) {

perror("read");

exit(1);

} else {

printf("Data read from 'sample.txt':\n");

write(STDOUT\_FILENO, buffer, bytesRead);

}

close(fd);

// Deleting the file

if (remove("sample.txt") == -1) {

perror("remove");

exit(1);

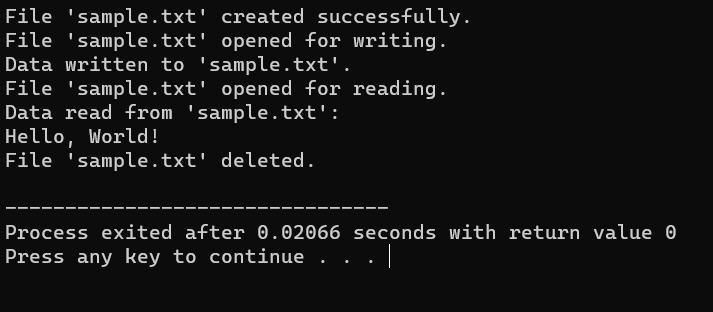
} else {

printf("File 'sample.txt' deleted.\n");

}

return 0;

}



25) Construct a C program to implement the I/O system calls of UNIX (fcntl, seek, stat, opendir, readdir)

Program:

#include<stdio.h>

#include<fcntl.h>

#include<errno.h>

extern int errno;

int main()

{

int fd = open("foo.txt", O\_RDONLY | O\_CREAT);

printf("fd = %d\n", fd);

if (fd ==-1)

{

printf("Error Number % d\n", errno);

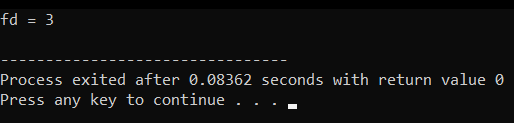
perror("Program");

}

return 0;

}

Output:



26) Construct a C program to implement the file management operations.

Program:

#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);

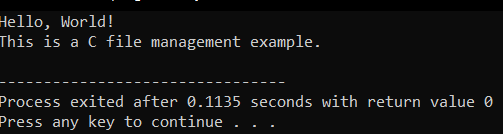
}

fclose(file);

return 0;

}

Output:



27) Develop a C program for simulating the function of ls UNIX Command.

Program:

#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);

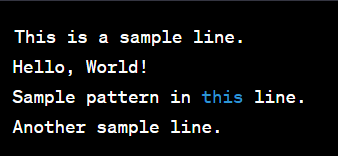
}

fclose(fp);

return 1;

}

Output:



28) Write a C program for simulation of GREP UNIX command

Program:

#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];

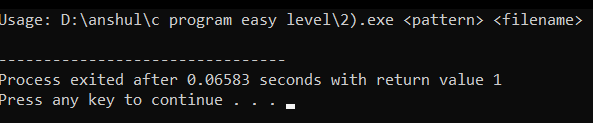
const char \*filename = argv[2];

searchFile(pattern, filename);

return 0;

}

Output:



29) Write a C program to simulate the solution of Classical Process Synchronization Problem

Program:

#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);

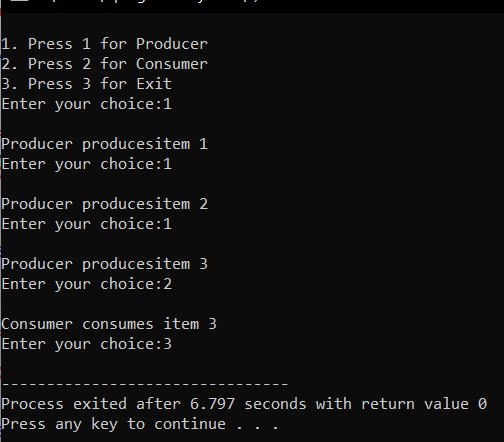
break;

}

}

}

Output:



30. Write C programs to demonstrate the following thread related concepts.

#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);

}

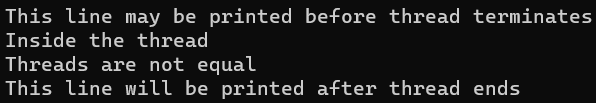
int main()

{

fun();

return 0;

}



31. Construct a C program to simulate the First in First Out paging technique of memory management.

#include<stdio.h>

int main()

{

int i,j,n,a[50],frame[10],no,k,avail,count=0;

printf("\n ENTER THE NUMBER OF PAGES:\n");

scanf("%d",&n);

printf("\n ENTER THE PAGE NUMBER :\n");

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

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

printf("\n ENTER THE NUMBER OF FRAMES :");

scanf("%d",&no);

for(i=0;i<no;i++)

frame[i]= -1;

j=0;

printf("\tref string\t page frames\n");

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

{

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

avail=0;

for(k=0;k<no;k++)

if(frame[k]==a[i])

avail=1;

if (avail==0)

{

frame[j]=a[i];

j=(j+1)%no;

count++;

for(k=0;k<no;k++)

printf("%d\t",frame[k]);

}

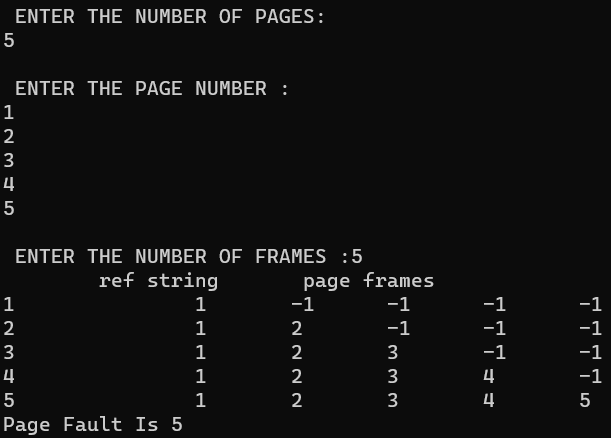
printf("\n");

}

printf("Page Fault Is %d",count);

return 0;

}



32. Construct a C program to simulate the Least Recently Used paging technique of memory management.

#include<stdio.h>

int findLRU(int time[], int n)

{

int i, minimum = time[0], pos = 0;

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

{

if(time[i] < minimum)

{

minimum = time[i];

pos= i;

}

}

return pos;

}

int main()

{

int no\_of\_frames, no\_of\_pages, frames[10], pages[30], counter = 0, time[10], flag1, flag2, i, j, pos, faults = 0;

printf("Enter number of frames: ");

scanf("%d", &no\_of\_frames);

printf("Enter number of pages: ");

scanf("%d", &no\_of\_pages);

printf("Enter reference string: ");

for(i = 0; i<no\_of\_pages; ++i)

{

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

}

for(i = 0; i<no\_of\_frames; ++i)

{

frames[i] = -1;

}

for(i = 0; i<no\_of\_pages; ++i)

{

flag1 = flag2 = 0;

for(j = 0; j <no\_of\_frames; ++j)

{

if(frames[j] == pages[i])

{

counter++;

time[j] = counter;

flag1 = flag2 = 1;

break;

}

}

if(flag1 == 0)

{

for(j = 0; j <no\_of\_frames; ++j)

{

if(frames[j] == -1)

{

counter++;

faults++;

frames[j] = pages[i];

time[j] = counter;

flag2 = 1;

break;

}

}

}

if(flag2 == 0){

pos = findLRU(time, no\_of\_frames);

counter++;

faults++;

frames[pos] = pages[i];

time[pos] = counter;

}

printf("\n");

for(j = 0; j <no\_of\_frames; ++j){

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

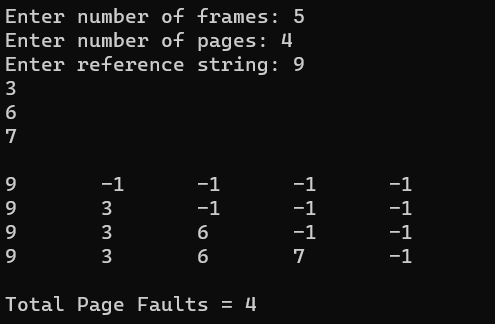
}

}

printf("\n\nTotal Page Faults = %d", faults);

return 0;

}



33. Construct a C program to simulate the optimal paging technique of memory management

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

int main()

{

int no\_of\_frames, no\_of\_pages, frames[10], pages[30], temp[10], flag1, flag2, flag3, i, j, k, pos, max, faults = 0;

printf("Enter number of frames: ");

scanf("%d", &no\_of\_frames);

printf("Enter number of pages: ");

scanf("%d", &no\_of\_pages);

printf("Enter page reference string: ");

for(i = 0; i<no\_of\_pages; ++i){

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

}

for(i = 0; i<no\_of\_frames; ++i){

frames[i] = -1;

}

for(i = 0; i<no\_of\_pages; ++i){

flag1 = flag2 = 0;

for(j = 0; j <no\_of\_frames; ++j){

if(frames[j] == pages[i]){

flag1 = flag2 = 1;

break;

}

}

if(flag1 == 0){

for(j = 0; j <no\_of\_frames; ++j){

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

faults++;

frames[j] = pages[i];

flag2 = 1;

break;

}

}

}

if(flag2 == 0){

flag3 =0;

for(j = 0; j <no\_of\_frames; ++j){

temp[j] = -1;

for(k = i + 1; k <no\_of\_pages; ++k){

if(frames[j] == pages[k]){

temp[j] = k;

break;

}

}

}

for(j = 0; j <no\_of\_frames; ++j){

if(temp[j] == -1){

pos = j;

flag3 = 1;

break;

}

}

if(flag3 ==0){

max = temp[0];

pos = 0;

for(j = 1; j <no\_of\_frames; ++j){

if(temp[j] > max){

max = temp[j];

pos = j;

}

}

}

frames[pos] = pages[i];

faults++;

}

printf("\n");

for(j = 0; j <no\_of\_frames; ++j){

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

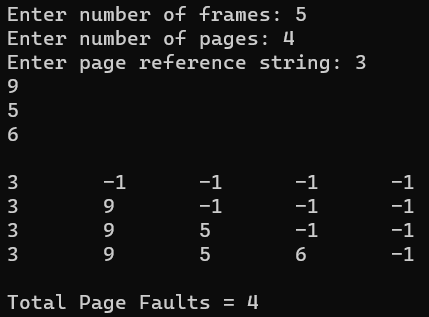
}

}

printf("\n\nTotal Page Faults = %d", faults);

return 0;

}



34. Consider a file system where the records of the file are stored one after another both physically and logically. A record of the file can only be accessed by reading all the previous records. Design a C program to simulate the file allocation strategy.

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

int main()

{

int f[50], i, st, len, j, c, k, count = 0;

for(i=0;i<50;i++)

f[i]=0;

printf("Files Allocated are : \n");

x : count=0;

printf("Enter starting block and length of files: ");

scanf("%d%d", &st,&len);

for(k=st;k<(st+len);k++)

if(f[k]==0)

count++;

if(len==count)

{

for(j=st;j<(st+len);j++)

if(f[j]==0)

{

f[j]=1;

printf("%d\t%d\n",j,f[j]);

}

if(j!=(st+len-1))

printf("The file is allocated to disk\n");

}

else

printf("The file is not allocated \n");

printf("Do you want to enter more file(Yes - 1/No - 0)");

scanf("%d", &c);

if(c==1)

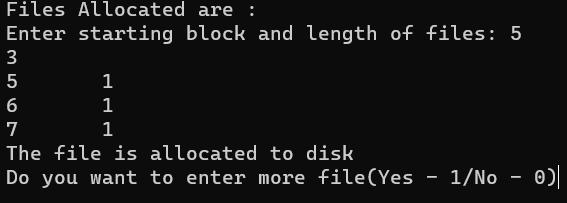
goto x;

else

exit(0);

getch();

}



35. Consider a file system that brings all the file pointers together into an index block. The ith entry in the index block points to the ith block of the file. Design a C program to simulate the file allocation strategy.

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

int main()

{

int f[50], index[50],i, n, st, len, j, c, k, ind,count=0;

for(i=0;i<50;i++)

f[i]=0;

x:printf("Enter the index block: ");

scanf("%d",&ind);

if(f[ind]!=1)

{

printf("Enter no of blocks needed and no of files for the index %d on the disk : \n", ind);

scanf("%d",&n);

}

else

{

printf("%d index is already allocated \n",ind);

goto x;

}

y: count=0;

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

{

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

if(f[index[i]]==0)

count++;

}

if(count==n)

{

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

f[index[j]]=1;

printf("Allocated\n");

printf("File Indexed\n");

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

printf("%d-------->%d : %d\n",ind,index[k],f[index[k]]);

}

else

{

printf("File in the index is already allocated \n");

printf("Enter another file indexed");

goto y;

}

printf("Do you want to enter more file(Yes - 1/No - 0)");

scanf("%d", &c);

if(c==1)

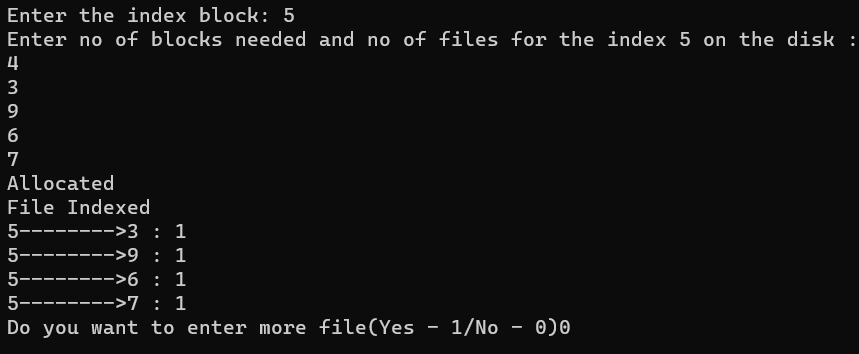
goto x;

else

exit(0);

getch();

}



36. With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file. Each block contains a pointer to the next block. Design a C program to simulate the file allocation strategy.

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

main()

{

int f[50], p,i, st, len, j, c, k, a;

for(i=0;i<50;i++)

f[i]=0;

printf("Enter how many blocks already allocated: ");

scanf("%d",&p);

printf("Enter blocks already allocated: ");

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

{

scanf("%d",&a);

f[a]=1;

}

x: printf("Enter index starting block and length: ");

scanf("%d%d", &st,&len);

k=len;

if(f[st]==0)

{

for(j=st;j<(st+k);j++)

{

if(f[j]==0)

{

f[j]=1;

printf("%d-------->%d\n",j,f[j]);

}

else

{

printf("%d Block is already allocated \n",j);

k++;

}

}

}

else

printf("%d starting block is already allocated \n",st);

printf("Do you want to enter more file(Yes - 1/No - 0)");

scanf("%d", &c);

if(c==1)

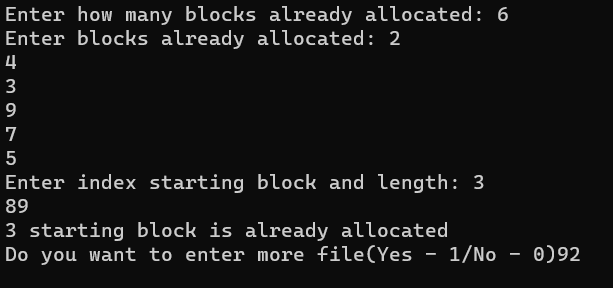
goto x;

else

exit(0);

getch();

}



37.Construct a C program to simulate the First Come First Served disk scheduling algorithm.

#include<stdio.h>

#include<stdlib.h>

int main()

{

int RQ[100],i,n,TotalHeadMoment=0,initial;

printf("Enter the number of Requests\n");

scanf("%d",&n);

printf("Enter the Requests sequence\n");

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

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

printf("Enter initial head position\n");

scanf("%d",&initial);

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

{

TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);

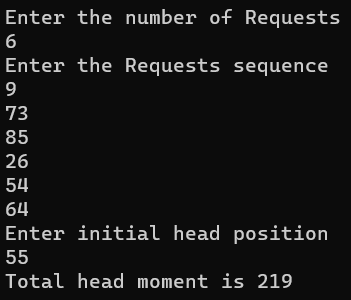
initial=RQ[i];

}

printf("Total head moment is %d",TotalHeadMoment);

return 0;

}



38. Design a C program to simulate SCAN disk scheduling algorithm.

#include <stdio.h>

#include <math.h>

int main()

{

int queue[20], n, head, i, j, k, seek = 0, max, diff, temp, queue1[20],

queue2[20], temp1 = 0, temp2 = 0;

float avg;

printf("Enter the max range of disk\n");

scanf("%d", &max);

printf("Enter the initial head position\n");

scanf("%d", &head);

printf("Enter the size of queue request\n");

scanf("%d", &n);

printf("Enter the queue of disk positions to be read\n");

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

{

scanf("%d", &temp);

if (temp >= head)

{

queue1[temp1] = temp;

temp1++;

}

else

{

queue2[temp2] = temp;

temp2++;

}

}

for (i = 0; i < temp1 - 1; i++)

{

for (j = i + 1; j < temp1; j++)

{

if (queue1[i] > queue1[j])

{

temp = queue1[i];

queue1[i] = queue1[j];

queue1[j] = temp;

}

}

}

for (i = 0; i < temp2 - 1; i++)

{

for (j = i + 1; j < temp2; j++)

{

if (queue2[i] < queue2[j])

{

temp = queue2[i];

queue2[i] = queue2[j];

queue2[j] = temp;

}

}

}

for (i = 1, j = 0; j < temp1; i++, j++)

queue[i] = queue1[j];

queue[i] = max;

for (i = temp1 + 2, j = 0; j < temp2; i++, j++)

queue[i] = queue2[j];

queue[i] = 0;

queue[0] = head;

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

{

diff = abs(queue[j + 1] - queue[j]);

seek += diff;

printf("Disk head moves from %d to %d with seek %d\n", queue[j],

queue[j + 1], diff);

}

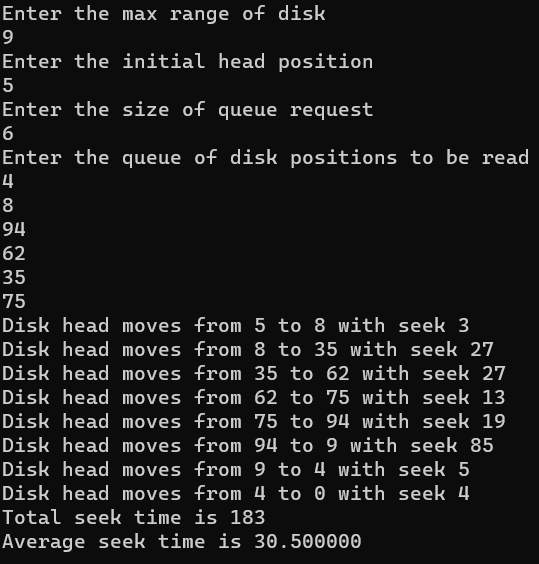
printf("Total seek time is %d\n", seek);

avg = seek / (float)n;

printf("Average seek time is %f\n", avg);

return 0;

}



39. Develop a C program to simulate C-SCAN disk scheduling algorithm.

#include <stdio.h>

#include <math.h>

int main()

{

int queue[20], n, head, i, j, k, seek = 0, max, diff, temp, queue1[20],

queue2[20], temp1 = 0, temp2 = 0;

float avg;

printf("Enter the max range of disk\n");

scanf("%d", &max);

printf("Enter the initial head position\n");

scanf("%d", &head);

printf("Enter the size of queue request\n");

scanf("%d", &n);

printf("Enter the queue of disk positions to be read\n");

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

{

scanf("%d", &temp);

if (temp >= head)

{

queue1[temp1] = temp;

temp1++;

}

else

{

queue2[temp2] = temp;

temp2++;

}

}

for (i = 0; i < temp1 - 1; i++)

{

for (j = i + 1; j < temp1; j++)

{

if (queue1[i] > queue1[j])

{

temp = queue1[i];

queue1[i] = queue1[j];

queue1[j] = temp;

}

}

}

for (i = 0; i < temp2 - 1; i++)

{

for (j = i + 1; j < temp2; j++)

{

if (queue2[i] > queue2[j])

{

temp = queue2[i];

queue2[i] = queue2[j];

queue2[j] = temp;

}

}

}

for (i = 1, j = 0; j < temp1; i++, j++)

queue[i] = queue1[j];

queue[i] = max;

queue[i + 1] = 0;

for (i = temp1 + 3, j = 0; j < temp2; i++, j++)

queue[i] = queue2[j];

queue[0] = head;

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

{

diff = abs(queue[j + 1] - queue[j]);

seek += diff;

printf("Disk head moves from %d to %d with seek %d\n", queue[j],

queue[j + 1], diff);

}

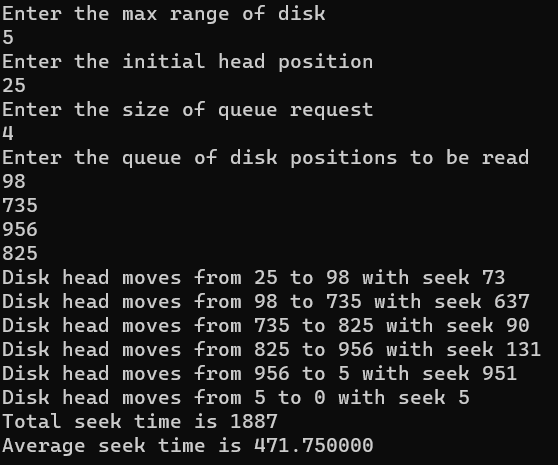
printf("Total seek time is %d\n", seek);

avg = seek / (float)n;

printf("Average seek time is %f\n", avg);

return 0;

}



40. Illustrate the various File Access Permission and different types users in Linux.

#include<stdio.h>

int main()

{

FILE \*fp;

fp = fopen;

if(!fp)

{

printf("Error in opening file\n");

return 0;

}

printf("Position of the pointer : %ld\n",ftell(fp));

char ch;

while(fread(&ch,sizeof(ch),1,fp)==1)

{

printf("%c",ch);

}

printf("\nPosition of the pointer : %ld\n",ftell(fp));

rewind(fp);

printf("\n USING REWIND Position of the pointer : %ld\n",ftell(fp));

printf("\nUSING FSEEK.....");

fseek(fp, 6, 0);

while(fread(&ch,sizeof(ch),1,fp)==1)

{

printf("%c",ch);

}

fclose(fp);

return 0;

}

