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.

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

int main(void) {

pid\_t pid = fork();

if(pid == 0) {

printf("Child => PPID: %d PID: %d\n", getppid(), getpid());

exit(EXIT\_SUCCESS);

}

else if(pid > 0) {

printf("Parent => PID: %d\n", getpid());

printf("Waiting for child process to finish.\n");

wait(NULL);

printf("Child process finished.\n");

}

else {

printf("Unable to create child process.\n");

}

return EXIT\_SUCCESS;

}

1. Identify the system calls to copy the content of one file to another and illustrate the same using a C program.

#include<stdio.h>

#include<stdlib.h>

int main()

{

FILE \*f1,\*f2;

char filename[100],c;

f1=fopen("D:\DEVC++\test2.c","r");

f2=fopen("D:\DEVC++\test1.txt","w");

c=fgetc(f1);

while(c!=EOF)

{

fputc(c,f2);

c=fgetc(f1);

}

printf("CONTENTS COPIED SUCCESSFULLY....");

fclose(f1);

fclose(f2);

}

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

#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);

}

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

#include<stdio.h>

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

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

}

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

#include<stdio.h>

struct priority\_scheduling {

char process\_name;

int burst\_time;

int waiting\_time;

int turn\_around\_time;

int priority;

};

int main() {

int number\_of\_process;

int total = 0;

struct priority\_scheduling temp\_process;

int ASCII\_number = 65;

int position;

float average\_waiting\_time;

float average\_turnaround\_time;

printf("Enter the total number of Processes: ");

scanf("%d", & number\_of\_process);

struct priority\_scheduling process[number\_of\_process];

printf("\nPlease Enter the Burst Time and Priority of each process:\n");

for (int i = 0; i < number\_of\_process; i++) {

process[i].process\_name = (char) ASCII\_number;

printf("\nEnter the details of the process %c \n", process[i].process\_name);

printf("Enter the burst time: ");

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

printf("Enter the priority: ");

scanf("%d", & process[i].priority);

ASCII\_number++;

}

for (int i = 0; i < number\_of\_process; i++) {

position = i;

for (int j = i + 1; j < number\_of\_process; j++) {

if (process[j].priority > process[position].priority)

position = j;

}

temp\_process = process[i];

process[i] = process[position];

process[position] = temp\_process;

}

process[0].waiting\_time = 0;

for (int i = 1; i < number\_of\_process; i++) {

process[i].waiting\_time = 0;

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

process[i].waiting\_time += process[j].burst\_time;

}

total += process[i].waiting\_time;

}

average\_waiting\_time = (float) total / (float) number\_of\_process;

total = 0;

printf("\n\nProcess\_name \t Burst Time \t Waiting Time \t Turnaround Time\n");

printf("------------------------------------------------------------\n");

for (int i = 0; i < number\_of\_process; i++) {

process[i].turn\_around\_time = process[i].burst\_time + process[i].waiting\_time;

total += process[i].turn\_around\_time;

printf("\t %c \t\t %d \t\t %d \t\t %d", process[i].process\_name, process[i].burst\_time, process[i].waiting\_time, process[i].turn\_around\_time);

printf("\n-----------------------------------------------------------\n");

}

average\_turnaround\_time = (float) total / (float) number\_of\_process;

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

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

return 0;

}

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

#include<stdio.h>

int main()

{

int burst\_time[20],p[20],waiting\_time[20],tat[20],i,j,n,total=0,pos,temp;

float avg\_waiting\_time,avg\_tat;

printf("please enter number of process: ");

scanf("%d",&n);

printf("\n enter the Burst Time:\n");

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

{

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

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

p[i]=i+1;

}

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

{

pos=i;

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

{

if(burst\_time[j]<burst\_time[pos])

pos=j;

}

temp=burst\_time[i];

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

burst\_time[pos]=temp;

temp=p[i];

p[i]=p[pos];

p[pos]=temp;

}

waiting\_time[0]=0;

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

{

waiting\_time[i]=0;

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

waiting\_time[i]+=burst\_time[j];

total+=waiting\_time[i];

}

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

total=0;

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

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

{

tat[i]=burst\_time[i]+waiting\_time[i];

total+=tat[i];

printf("\np%d\t\t %d\t\t %d\t\t\t%d",p[i],burst\_time[i],waiting\_time[i],tat[i]);

}

avg\_tat=(float)total/n;

printf("\n\n the average Waiting Time=%f",avg\_waiting\_time);

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

}

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

#include<stdio.h>

int main() {

int time, burst\_time[10], at[10], sum\_burst\_time = 0, smallest, n, i;

int sumt = 0, sumw = 0;

printf("enter the no of processes : ");

scanf("%d", & n);

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

printf("the arrival time for process P%d : ", i + 1);

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

printf("the burst time for process P%d : ", i + 1);

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

sum\_burst\_time += burst\_time[i];

}

burst\_time[9] = 9999;

for (time = 0; time < sum\_burst\_time;) {

smallest = 9;

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

if (at[i] <= time && burst\_time[i] > 0 && burst\_time[i] < burst\_time[smallest])

smallest = i;

}

printf("P[%d]\t|\t%d\t|\t%d\n", smallest + 1, time + burst\_time[smallest] - at[smallest], time - at[smallest]);

sumt += time + burst\_time[smallest] - at[smallest];

sumw += time - at[smallest];

time += burst\_time[smallest];

burst\_time[smallest] = 0;

}

printf("\n\n average waiting time = %f", sumw \* 1.0 / n);

printf("\n\n average turnaround time = %f", sumt \* 1.0 / n);

}

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

#include<stdio.h>

#include<conio.h>

int main()

{

int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10];

float avg\_wt, avg\_tat;

printf(" Total number of process in the system: ");

scanf("%d", &NOP);

y = NOP;

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

{

printf("\n Enter the Arrival and Burst time of the Process[%d]\n", i+1);

printf(" Arrival time is: \t");

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

printf(" \nBurst time is: \t");

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

temp[i] = bt[i];

}

printf("Enter the Time Quantum for the process: \t");

scanf("%d", &quant);

printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time ");

for(sum=0, i = 0; y!=0; )

{

if(temp[i] <= quant && temp[i] > 0)

{

sum = sum + temp[i];

temp[i] = 0;

count=1;

}

else if(temp[i] > 0)

{

temp[i] = temp[i] - quant;

sum = sum + quant;

}

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

{

y--;

printf("\nProcess No[%d] \t\t %d\t\t\t\t %d\t\t\t %d", i+1, bt[i], sum-at[i], sum-at[i]-bt[i]);

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

tat = tat+sum-at[i];

count =0;

}

if(i==NOP-1)

{

i=0;

}

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

{

i++;

}

else

{

i=0;

}

}

avg\_wt = wt \* 1.0/NOP;

avg\_tat = tat \* 1.0/NOP;

printf("\n Average Turn Around Time: \t%f", avg\_wt);

printf("\n Average Waiting Time: \t%f", avg\_tat);

getch();

}

1. Illustrate the concept of inter-process communication using shared memory with a C 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.

#include <stdio.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#define MAX 10

struct mesg\_buffer

{

long mesg\_type;

char mesg\_text[100];

} message;

int main()

{

key\_t key;

int msgid;

key = ftok("progfile", 65);

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

message.mesg\_type = 1;

printf("Write Data : ");

fgets(message.mesg\_text,MAX,stdin);

msgsnd(msgid, &message, sizeof(message), 0);

printf("Data send is : %s \n", message.mesg\_text);

return 0;

}

1. Illustrate the concept of multithreading using a C program.

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<pthread.h>

void \*myThreadFun(void \*vargp)

{

sleep(1);

printf("Printing GeeksQuiz from Thread \n");

return NULL;

}

int main()

{

pthread\_t thread\_id;

printf("Before Thread\n");

pthread\_create(&thread\_id, NULL, myThreadFun, NULL);

pthread\_join(thread\_id, NULL);

printf("After Thread\n");

exit(0);

}

1. Design a C program to simulate the concept of Dining-Philosophers problem #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);

}

1. Construct a C program for implementation the various memory allocation strategies.

#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");

}

}

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

#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();

}

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

#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();

}

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

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define FILE\_NAME "employee.dat"

#define RECORD\_SIZE sizeof(struct Employee)

struct Employee {

int id;

char name[30];

int age;

float salary;

};

void addEmployee(struct Employee e) {

FILE \*fp;

fp = fopen(FILE\_NAME, "ab");

if (fp == NULL) {

printf("Could not open file %s", FILE\_NAME);

return;

}

fwrite(&e, RECORD\_SIZE, 1, fp);

fclose(fp);

}

void readEmployee(int id) {

struct Employee e;

FILE \*fp;

fp = fopen(FILE\_NAME, "rb");

if (fp == NULL) {

printf("Could not open file %s", FILE\_NAME);

return;

}

fseek(fp, (id - 1) \* RECORD\_SIZE, SEEK\_SET);

fread(&e, RECORD\_SIZE, 1, fp);

printf("Employee with ID %d:\n", id);

printf("Name: %s\n", e.name);

printf("Age: %d\n", e.age);

printf("Salary: %.2f\n", e.salary);

fclose(fp);

}

int main() {

struct Employee e1 = {1, "John Doe", 25, 50000.0};

struct Employee e2 = {2, "Jane Doe", 30, 60000.0};

addEmployee(e1);

addEmployee(e2);

readEmployee(1);

readEmployee(2);

}

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

#include <stdio.h>

int main()

{

int r[1][10], av[1][10];

int all[10][10], max[10][10], ne[10][10], w[10],safe[10];

int i=0, j=0, k=0, l=0, np=0, nr=0, count=0, cnt=0;

printf("enter the number of processes in a system");

scanf("%d", &np);

printf("enter the number of resources in a system");

scanf("%d",&nr);

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

{

printf("Enter no. of instances of resource R%d " ,i);

scanf("%d", &r[0][i]);

av[0][i] = r[0][i];

}

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

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

all[i][j] = ne[i][j] = max[i][j] = w[i]=0;

printf("Enter the allocation matrix");

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

{

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

{

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

av[0][j] = av[0][j] - all[i][j];

}

}

printf("Enter the maximum matrix");

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

{

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

{

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

}

}

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

{

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

{

ne[i][j] = max[i][j] - all[i][j];

}

}

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

{

printf("pocess P%d", i);

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

{

printf("\n allocated %d\t",all[i][j]);

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

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

}

printf("\n \n");

}

printf("\nAvailability ");

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

printf("R%d %d\t", i, av[0][i]);

printf("\n ");

printf("\n safe sequence");

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

{

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

{

cnt = 0;

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

{

if(ne[i][j] <= av[0][j] && w[i]==0)

cnt++;

}

if(cnt == nr)

{

k++;

safe[k] = i;

for(l=1; l<=nr; l++)

av[0][l] = av[0][l] + all[i][l];

printf("\n P%d ",safe[k]);

printf("\t Availability ");

for(l=1; l<=nr; l++)

printf("R%d %d\t", l, av[0][l]);

w[i]=1;

}

}

}

}

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

#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);

}

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

#include <pthread.h>

#include <stdio.h>

#include <unistd.h>

pthread\_mutex\_t mutex;

void \*thread\_func(void \*arg)

{

pthread\_mutex\_lock(&mutex);

printf("Entered thread\_func\n");

sleep(5);

printf("Exiting thread\_func\n");

pthread\_mutex\_unlock(&mutex);

return NULL;

}

int main()

{

pthread\_t tid;

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

pthread\_create(&tid, NULL, thread\_func, NULL);

pthread\_mutex\_lock(&mutex);

printf("Entered main\n");

sleep(5);

printf("Exiting main\n");

pthread\_mutex\_unlock(&mutex);

pthread\_join(tid, NULL);

pthread\_mutex\_destroy(&mutex);

return 0;

}

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

#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;

}

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

#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;

}

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

#include <stdio.h>

struct process

{

int size;

int flag;

int holeid;

} p[10];

struct hole

{

int hid;

int size;

int actual;

} h[10];

int main()

{

int i, np, nh, j;

void bsort(struct hole[], int);

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

scanf("%d", &nh);

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

{

printf("Enter size for hole H%d : ",i);

scanf("%d", &h[i].size);

h[i].actual = h[i].size;

h[i].hid = i;

}

printf("\nEnter number of process : " );

scanf("%d",&np);

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

{

printf("enter the size of process P%d : ",i);

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

p[i].flag = 0;

}

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

{

bsort(h, nh);

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

{

if(p[i].flag != 1)

{

if(p[i].size <= h[j].size)

{

p[i].flag = 1;

p[i].holeid = h[j].hid;

h[j].size -= p[i].size;

}

}

}

}

printf("\n\tBest fit\n");

printf("\nProcess\tPSize\tHole");

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

{

if(p[i].flag != 1)

printf("\nP%d\t%d\tNot allocated", i, p[i].size);

else

printf("\nP%d\t%d\tH%d", i, p[i].size, p[i].holeid);

}

printf("\n\nHole\tActual\tAvailable");

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

printf("\nH%d\t%d\t%d", h[i].hid, h[i].actual,

h[i].size);

printf("\n");

}

void bsort(struct hole bh[], int n)

{

struct hole temp;

int i,j;

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

{

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

{

if(bh[i].size > bh[j].size)

{

temp = bh[i];

bh[i] = bh[j];

bh[j] = temp;

}

}

}

}

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

#include <stdio.h>

struct process

{

int size;

int flag;

int holeid;

} p[10];

struct hole

{

int size;

int actual;

} h[10];

main()

{

int i, np, nh, j;

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

scanf("%d", &nh);

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

{

printf("Enter size for hole H%d : ",i);

scanf("%d", &h[i].size);

h[i].actual = h[i].size;

}

printf("\nEnter number of process : " );

scanf("%d",&np);

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

{

printf("enter the size of process P%d : ",i);

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

p[i].flag = 0;

}

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

{

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

{

if(p[i].flag != 1)

{

if(p[i].size <= h[j].size)

{

p[i].flag = 1;

p[i].holeid = j;

h[j].size -= p[i].size;

}

}

}

}

printf("\n\tFirst fit\n");

printf("\nProcess\tPSize\tHole");

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

{

if(p[i].flag != 1)

printf("\nP%d\t%d\tNot allocated", i, p[i].size);

else

printf("\nP%d\t%d\tH%d", i, p[i].size, p[i].holeid);

}

printf("\n\nHole\tActual\tAvailable");

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

printf("\nH%d\t%d\t%d", i, h[i].actual, h[i].size);

printf("\n");

}

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

#include <fcntl.h>

#include <stdio.h>

#include <unistd.h>

#include <sys/stat.h>

int main()

{

int fd, nbytes;

char buffer[1024];

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

if (fd == -1) {

perror("open");

return 1;

}

nbytes = read(fd, buffer, sizeof(buffer));

if (nbytes == -1) {

perror("read");

return 1;

}

printf("Read %d bytes: %s\n", nbytes, buffer);

if (close(fd) == -1) {

perror("close");

return 1;

}

return 0;

}

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

#include<sys/stat.h>

#include<stdio.h>

#include<fcntl.h>

#include<sys/types.h>

int main()

{

int n,i=0;

int f1,f2;

char c,strin[100];

f1=open("data",O\_RDWR|O\_CREAT|O\_TRUNC);

while((c=getchar())!='\n')

{

strin[i++]=c;

}

strin[i]='\0';

write(f1,strin,i);

close(f1);

f2=open("data",O\_RDONLY);

read(f2,strin,0);

printf("\n%s\n",strin);

close(f2);

return 0;

}

1. Construct a C program to implement the file management operations.

# include <stdio.h>

# include <string.h>

int main( )

{

FILE \*filePointer ;

char dataToBeWritten[50]

= "file";

filePointer = fopen("file.c", "w") ;

if ( filePointer == NULL )

{

printf( "file.c file failed to open." ) ;

}

else

{

printf("The file is now opened.\n") ;

if ( strlen ( dataToBeWritten ) > 0 )

{

fputs(dataToBeWritten, filePointer) ;

fputs("\n", filePointer) ;

}

fclose(filePointer) ;

printf("Data successfully written in file file.c\n");

printf("The file is now closed.") ;

}

return 0;

}

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

#include <dirent.h>

#include <stdio.h>

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

DIR \*dp;

struct dirent \*dirp;

if (argc != 2) {

printf("Usage: ls directory\_name\n");

return 1;

}

if ((dp = opendir(argv[1])) == NULL) {

printf("Cannot open directory %s\n", argv[1]);

return 1;

}

while ((dirp = readdir(dp)) != NULL) {

printf("%s\n", dirp->d\_name);

}

closedir(dp);

return 0;

}

1. Write a C program for simulation of GREP UNIX command

#include <stdio.h>

#include <string.h>

#define MAX\_LINE\_LEN 1024

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

char line[MAX\_LINE\_LEN];

FILE \*fp;

if (argc != 3) {

printf("Usage: grep pattern file\n");

return 1;

}

fp = fopen(argv[2], "r");

if (fp == NULL) {

printf("Cannot open file %s\n", argv[2]);

return 1;

}

while (fgets(line, MAX\_LINE\_LEN, fp) != NULL) {

if (strstr(line, argv[1]) != NULL) {

printf("%s", line);

}

}

fclose(fp);

return 0;

}

1. Write a C program to simulate the solution of Classical Process Synchronization Problem

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

#define BUFFER\_SIZE 10

int buffer[BUFFER\_SIZE];

int fill = 0;

int use = 0;

int count = 0;

sem\_t empty;

sem\_t full;

pthread\_mutex\_t mutex;

void put(int value) {

buffer[fill] = value;

fill = (fill + 1) % BUFFER\_SIZE;

count++;

}

int get() {

int tmp = buffer[use];

use = (use + 1) % BUFFER\_SIZE;

count--;

return tmp;

}

void \*producer(void \*arg) {

int i;

for (i = 0; i < 25; i++) {

sem\_wait(&empty);

pthread\_mutex\_lock(&mutex);

put(i);

pthread\_mutex\_unlock(&mutex);

sem\_post(&full);

}

return NULL;

}

void \*consumer(void \*arg) {

int i;

for (i = 0; i < 25; i++) {

sem\_wait(&full);

pthread\_mutex\_lock(&mutex);

int tmp = get();

pthread\_mutex\_unlock(&mutex);

sem\_post(&empty);

printf("%d\n", tmp);

}

return NULL;

}

int main() {

pthread\_t producer\_thread;

pthread\_t consumer\_thread;

sem\_init(&empty, 0, BUFFER\_SIZE);

sem\_init(&full, 0, 0);

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

pthread\_create(&producer\_thread, NULL, producer, NULL);

pthread\_create(&consumer\_thread, NULL, consumer, NULL);

pthread\_join(producer\_thread, NULL);

pthread\_join(consumer\_thread, NULL);

sem\_destroy(&empty);

sem\_destroy(&full);

pthread\_mutex\_destroy(&mutex);

return 0;

}

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

(i) create (ii) join (iii) equal (iv) exit

#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;

}

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

#include<stdio.h>

#define MAX\_PAGE\_FRAMES 100

#define MAX\_PAGE\_REQUESTS 1000

int pageFrames[MAX\_PAGE\_FRAMES];

int pageRequests[MAX\_PAGE\_REQUESTS];

int pageFaults = 0;

int numPageFrames;

int numPageRequests,page\_hit;

void initialize() {

int i;

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

pageFrames[i] = -1;

}

}

int isInMemory(int page) {

int i;

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

if (pageFrames[i] == page) {

return 1;

}

}

return 0;

}

void replacePage(int page) {

int i;

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

pageFrames[i] = pageFrames[i + 1];

}

pageFrames[numPageFrames - 1] = page;

}

int main() {

int i;

printf("Enter number of page frames: ");

scanf("%d", &numPageFrames);

printf("Enter number of page requests: ");

scanf("%d", &numPageRequests);

printf("Enter page requests:\n");

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

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

}

initialize();

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

int page = pageRequests[i];

if (!isInMemory(page)) {

pageFaults++;

if (pageFrames[numPageFrames - 1] != -1) {

replacePage(page);

} else {

pageFrames[numPageFrames - 1] = page;

}

}

}

printf("Number of page faults: %d\n", pageFaults);

page\_hit=numPageRequests-pageFaults;

printf("Number of page hit: %d\n",page\_hit);

return 0;

}

1. 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,page\_hit;

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);

page\_hit=no\_of\_pages-faults;

printf("\n\n page hit = %d",page\_hit);

return 0;

}

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

#include<stdio.h>

int main()

{

int no\_of\_frames, no\_of\_pages,page\_hit, 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);

page\_hit=no\_of\_pages-faults;

printf("\n\n page hit = %d",page\_hit);

return 0;

}

1. 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>

int main()

{

char name[10][30];

int start[10],length[10],num;

printf("Enter the number of files to be allocated\n");

scanf("%d",&num);

int count=0,k,j;

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

{

printf("Enter the name of the file %d\n",i+1);

scanf("%s",&name[i][0]);

printf("Enter the start block of the file %d\n",i+1);

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

printf("Enter the length of the file %d\n",i+1);

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

for(j=0,k=1;j<num && k<num;j++,k++)

{

if(start[j+1]<=start[j] || start[j+1]>=length[j])

{

}

else

{

count++;

}

}

if(count==1)

{

printf("%s cannot be allocated disk space\n",name[i]);

}

}

printf("File Allocation Table\n");

printf("%s%40s%40s\n","File Name","Start Block","Length");

printf("%s%50d%50d\n",name[0],start[0],length[0]);

for(int i=0,j=1;i<num && j<num;i++,j++)

{

if(start[i+1]<=start[i] || start[i+1]>=length[i])

{

printf("%s%50d%50d\n",name[j],start[j],length[j]);

}

}

return 0;

}

1. 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<stdlib.h>

void 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);

}

1. 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<stdlib.h>

void 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);

}

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;

}

1. 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;

}

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

#include<stdio.h>

#include<stdlib.h>

int main()

{

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

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);

printf("Enter total disk size\n");

scanf("%d",&size);

printf("Enter the head movement direction for high 1 and for low 0\n");

scanf("%d",&move);

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

{

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

{

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

{

int temp;

temp=RQ[j];

RQ[j]=RQ[j+1];

RQ[j+1]=temp;

}

}

}

int index;

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

{

if(initial<RQ[i])

{

index=i;

break;

}

}

if(move==1)

{

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

{

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

initial=RQ[i];

}

TotalHeadMoment=TotalHeadMoment+abs(size-RQ[i-1]-1);

TotalHeadMoment=TotalHeadMoment+abs(size-1-0);

initial=0;

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

{

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

initial=RQ[i];

}

}

else

{

for(i=index-1;i>=0;i--)

{

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

initial=RQ[i];

}

TotalHeadMoment=TotalHeadMoment+abs(RQ[i+1]-0);

TotalHeadMoment=TotalHeadMoment+abs(size-1-0);

initial =size-1;

for(i=n-1;i>=index;i--)

{

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

initial=RQ[i];

}

}

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

return 0;

}

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

#include<stdio.h>

int main()

{

FILE \*fp;

fp = fopen("D:\\DEVC++\\file1.txt","r");

if(!fp)

{

printf("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;

}