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<unistd.h>
int main(){
 printf("Process ID: %d\n", getpid() );
 printf("Parent Process ID: %d\n", getpid() );
  return 0;}
```

2. 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 *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;}
```

3. Design 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.

```
#include <stdio.h>
Int main(){
Int A[100][4];
Int i, j, n, total = 0, index, temp;
Float avg_wt, avg_tat;
Printf("Enter number of process: ");
Scanf("%d", &n);
Printf("Enter Burst Time:\n");
For (i = 0; i < n; i++) {
        Printf("P%d: ", i + 1);
        Scanf("%d", &A[i][1]);
        A[i][0] = i + 1; }
For (i = 0; i < n; i++) {
        Index = i;
        For (j = i + 1; j < n; j++)
                If (A[j][1] < A[index][1])
                        Index = j;
        Temp = A[i][1];
        A[i][1] = A[index][1];
        A[index][1] = temp;
        Temp = A[i][0];
        A[i][0] = A[index][0];
        A[index][0] = temp;     }
A[0][2] = 0;
For (    i = 1; i < n; i++) {
```

```
        A[i][2] = 0;
        For (j = 0; j < i; j++)
                A[i][2] += A[j][1];
        Total += A[i][2];}
Avg_wt = (float)total / n;
Total = 0;
Printf("P        BT      WT      TAT\n");
For (i = 0; i < n; i++) {
        A[i][3] = A[i][1] + A[i][2];
        Total += A[i][3];
Printf("P%d %d %d %d\n", A[i][0],A[i][1], A[i][2], A[i][3]);  }
Avg_tat = (float)total / n;
Printf("Average Waiting Time= %f", avg_wt);
Printf("\nAverage Turnaround Time= %f", avg_tat);}
```

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

```
#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("nProcesst    Burst Time    tWaiting TimetTurnaround Time");
      For(i=0;i<n;i++)    {
        Tat[i]=bt[i]+wt[i];
        Total+=tat[i];
        Printf("np%dtt  %dtt    %dttt%d",p[i],bt[i],wt[i],tat[i]);    }
      Avg_tat=(float)total/n;
      Printf("nnAverage Waiting Time=%f",avg_wt);
      Printf("nAverage Turnaround Time=%fn",avg_tat);}
```

5. 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;}
```

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

```
#include <stdio.h>
Void swap(int *a,int *b){
   Int temp=*a;
   *a=*b;
   *b=temp;}
Int main(){
   Int n;
   Printf("Enter Number of Processes: ");
   Scanf("%d",&n);
   Int b[n],p[n],index[n];
   For(int i=0;i<n;i++)  {
      Printf("Enter Burst Time and Priority Value for Process %d: ",i+1);
      Scanf("%d %d",&b[i],&p[i]);
      Index[i]=i+1;}
   For(int i=0;i<n;i++){
      Int a=p[i],m=i;
      For(int j=i;j<n;j++) {
         If(p[j] > a)   {
            A=p[j];
            M=j;}  }
```

```
    Swap(&p[i], &p[m]);
    Swap(&b[i], &b[m]);
    Swap(&index[i],&index[m]);}
  Int t=0;
  Printf("Order of process Execution is\n");
  For(int i=0;i<n;i++){
    Printf("P%d is executed from %d to %d\n",index[i],t,t+b[i]);
    T+=b[i];}
  Printf("\n");
  Printf("Process Id Burst Time Wait Time TurnAround Time\n");
  Int wait_time=0;
  For(int i=0;i<n;i++){
    Printf("P%d %d %d %d\n",index[i],b[i],wait_time,wait_time + b[i]);
    Wait_time += b[i];}
  Return 0;}
```

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

```
#include <stdio.h>
Int main(){
  Int A[100][4];
  Int i, j, n, total = 0, index, temp;
  Float avg_wt, avg_tat;
  Printf("Enter number of process: ");
  Scanf("%d", &n);
  Printf("Enter Burst Time:\n");
  For (i = 0; i < n; i++) {
    Printf("P%d: ", i + 1);
    Scanf("%d", &A[i][1]);
    A[i][0] = i + 1;}
  For (i = 0; i < n; i++) {
    Index = i;
    For (j = i + 1; j < n; j++)
      If (A[j][1] < A[index][1])
        Index = j;
```

```
        Temp = A[i][1];
        A[i][1] = A[index][1];
        A[index][1] = temp;
        Temp = A[i][0];
        A[i][0] = A[index][0];
        A[index][0] = temp;}
    A[0][2] = 0;
    For (i = 1; i < n; i++) {
        A[i][2] = 0;
        For (j = 0; j < i; j++)
            A[i][2] += A[j][1];
        Total += A[i][2];}
    Avg_wt = (float)total / n;
    Total = 0;
    Printf("P     BT     WT     TAT\n");
    For (i = 0; i < n; i++) {
        A[i][3] = A[i][1] + A[i][2];
        Total += A[i][3];
        Printf("P%d     %d     %d      %d\n", A[i][0],
            A[i][1], A[i][2], A[i][3]);}
    Avg_tat = (float)total / n;
    Printf("Average Waiting Time= %f", avg_wt);
    Printf("\nAverage Turnaround Time= %f", avg_tat);}
```

8. 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(); }

9. 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;}
```

11.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);}
```

12.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);}
```

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

```
#include <stdio.h>
#include <stdlib.h>
Int main(){
   Int* ptr;
   Int n, i;
```

```
Printf(“Enter number of elements:”);
Scanf(“%d”,&n);
Printf(“Entered number of elements: %d\n”, n);
Ptr = (int*)malloc(n * sizeof(int));
If (ptr == NULL) {
   Printf(“Memory not allocated.\n”);
   Exit(0);}
Else {
   Printf(“Memory successfully allocated using malloc.\n”);
   For (i = 0; i < n; ++i) {
      Ptr[i] = i + 1;}
   Printf(“The elements of the array are: “);
   For (i = 0; i < n; ++i) {
      Printf(“%d, “, ptr[i]); }}
Return 0;}
```

14. 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();}
```

15. 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();     }
```

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

```
#include<stdio.h>
Int main(){
   FILE *fp;
   Fp=fopen("prepbytes.txt","r");
   If(!fp) {
      Printf("Error: File cannot be opened\n") ;
      Return 0;}
      Printf("Position pointer in the beginning : %ld\n",ftell(fp));
   Char ch;
   While(fread(&ch,sizeof(ch),1,fp)==1){
      Printf("%c",ch);}
   Printf("\nSize of file in bytes is : %ld\n",ftell(fp));
   Fclose(fp);
   Return 0;}
```

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

```
#include<stdio.h>
#include<conio.h>
```

```
Int max[100][100];
Int alloc[100][100];
Int need[100][100];
Int avail[100];
Int n,r;
Void input();
Void show();
Void cal();
Int main(){
Int i,j;
Printf("********** Banker's Algo ************\n");
Input();
Show();
Cal();
Getch();
Return 0;}
Void input(){
Int i,j;
Printf("Enter the no of Processes\t");
Scanf("%d",&n);
Printf("Enter the no of resources instances\t");
Scanf("%d",&r);
Printf("Enter the Max Matrix\n");
For(i=0;i<n;i++){
For(j=0;j<r;j++){
Scanf("%d",&max[i][j]);}}
Printf("Enter the Allocation Matrix\n");
For(i=0;i<n;i++){
For(j=0;j<r;j++){
Scanf("%d",&alloc[i][j]);}}
Printf("Enter the available Resources\n");
For(j=0;j<r;j++){
Scanf("%d",&avail[j]);}}
```

```
Void show(){
Int i,j;
Printf("Process\t Allocation\t Max\t Available\t");
For(i=0;i<n;i++){
Printf("\nP%d\t ",i+1);
For(j=0;j<r;j++){
Printf("%d ",alloc[i][j]);}
Printf("\t");
For(j=0;j<r;j++){
Printf("%d ",max[i][j]);}
Printf("\t");
If(i==0){
For(j=0;j<r;j++)
Printf("%d ",avail[j]);}}}
Void cal(){
Int finish[100],temp,need[100][100],flag=1,k,c1=0;
Int safe[100];
Int i,j;
For(i=0;i<n;i++){
Finish[i]=0;}
For(i=0;i<n;i++){
For(j=0;j<r;j++){
Need[i][j]=max[i][j]-alloc[i][j];}}
Printf("\n");
While(flag){
Flag=0;
For(i=0;i<n;i++){
Int c=0;
For(j=0;j<r;j++){
If((finish[i]==0)&&(need[i][j]<=avail[j])){
C++;
If(c==r){
For(k=0;k<r;k++){
```

```
    Avail[k]+=alloc[i][j];
    Finish[i]=1;
    Flag=1;}
    Printf("P%d->",i);
    If(finish[i]==1){
    I=n;}}}}}}
    For(i=0;i<n;i++){
    If(finish[i]==1){
    C1++;}
    Else{
    Printf("P%d->",i);}}
    If(c1==n){
    Printf("\n The system is in safe state");}
    Else{
    Printf("\n Process are in dead lock");
    Printf("\n System is in unsafe state");}}
```

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);}
```

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

```
#include <pthread.h>
#include <stdio.h>
```

```
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
Pthread_t tid[2];
Int counter;
Pthread_mutex_t lock;
Void* trythis(void* arg) {
    Pthread_mutex_lock(&lock);
    Unsigned long i = 0;
    Counter += 1;
    Printf("\n Job %d has started\n", counter);
    For (i = 0; i < (0xFFFFFFFF); i++) ;
    Printf("\n Job %d has finished\n", counter);
    Pthread_mutex_unlock(&lock);
    Return NULL; }
Int main(void) {
    Int i = 0;
    Int error;
    If (pthread_mutex_init(&lock, NULL) != 0) {
            Printf("\n mutex init has failed\n");
            Return 1; }
    While (i < 2) {
            Error = pthread_create(&(tid[i]),
            NULL,
            &trythis, NULL);
            If (error != 0)
    Printf("\nThread can't be created :[%s]",
    Strerror(error));
            I++; }
    Pthread_join(tid[0], NULL);
    Pthread_join(tid[1], NULL);
    Pthread_mutex_destroy(&lock);
    Return 0; }
```

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

```
#include<semaphore.h>
#include<stdio.h>
#include<stdlib.h>
Sem_t x,y;
Pthread_t tid;
Pthread_t writerthreads[100],readerthreads[100];
Int readercount;
Void *reader(void* param){
   Sem_wait(&x);
   Readercount++;
   If(readercount==1)
   Sem_wait(&y);
   Sem_post(&x);
   Printf("\n%d reader is inside",readercount);
   Sem_wait(&x);
   Readercount--;
   If(readercount==0) {
      Sem_post(&y);   }
   Sem_post(&x);
   Printf("\n%d Reader is leaving",readercount+1);}
Void *writer(void* param){
   Printf("\nWriter is trying to enter");
   Sem_wait(&y);
   Printf("\nWriter has entered");
   Sem_post(&y);
   Printf("\nWriter is leaving");}
Int main(){
   Int n2,i;
   Printf("Enter the number of readers:");
   Scanf("%d",&n2);
   Int n1[n2];
   Sem_init(&x,0,1);
```

```
Sem_init(&y,0,1);
For(i=0;i<n2;i++){
    Pthread_create(&writerthreads[i],NULL,reader,NULL);
    Pthread_create(&readerthreads[i],NULL,writer,NULL);    }
For(i=0;i<n2;i++)   {
    Pthread_join(writerthreads[i],NULL);
    Pthread_join(readerthreads[i],NULL);}}
```

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

```
#include<stdio.h>
#include<conio.h>
#define max 25
Void main(){
Int frag[max],b[max],f[max],i,j,nb,nf,temp,highest=0;
 Static int bf[max],ff[max];
Clrscr();
Printf("\n\tMemory Management Scheme – Worst Fit");
 Printf("\nEnter the number of blocks:");
 Scanf("%d",&nb);
Printf("Enter the number of files:");
Scanf("%d",&nf);
Printf("\nEnter the size of the blocks:-\n");
For(i=1;i<=nb;i++){
Printf("Block %d:",i);
Scanf("%d",&b[i]);}
Printf("Enter the size of the files :-\n");
For(i=1;i<=nf;i++){
Printf("File %d:",i);
Scanf("%d",&f[i]);}
For(i=1;i<=nf;i++){
For(j=1;j<=nb;j++){
If(bf[j]!=1)    //if bf[j] is not allocated{
Temp=b[j]-f[i];
If(temp>=0)
```

```
If(highest<temp){
Ff[i]=j;
Highest=temp;}}
Frag[i]=highest;
Bf[ff[i]]=1;
 Highest=0;}
Ff[i]=j;
Highest=temp;}
Printf("\nFile_no:\tFile_size:\tBlock_no:\tBlock_size:\tFragement;
For(i=1;i<=nf;i++)
Printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
Getch();}
```

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

```
#include<stdio.h>
#include<process.h>
Void main(){
Int a[20],p[20],i,j,n,m;
Printf("Enter no of Blocks.\n");
Scanf("%d",&n);
For(i=0;i<n;i++){
  Printf("Enter the %dst Block size:",i);
Scanf("%d",&a[i]);}
Printf("Enter no of Process.\n");
Scanf("%d",&m);
For(i=0;i<m;i++){
  Printf("Enter the size of %dst Process:",i);
   Scanf("%d",&p[i]);
} For(i=0;i<n;i++){
For(j=0;j<m;j++) {
If(p[j]<=a[i] {   Printf("The Process %d allocated to %d\n",j,a[i]);
 P[j]=10000;
  Break;    } }}
For(j=0;j<m;j++){
```

```
If(p[j]!=10000)  {
Printf("The Process %d is not allocated\n",j);  }}}
```

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

```
#include<stdio.h>
Void 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\td",allocation[i]+1,psize[allocation[i]]);
                Else
                        Printf("Not allocated");}}
```

24. Design a C program to demonstrate UNIX system calls for file management. 26. Construct a C program to implement the file management operations.

```
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <stdio.h>
Int main()
{Int n, fd;
   Char buff[50];
   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;}
```

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

```
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <stdio.h>
Int main()
{Int fd[2];
Char buf1[25] = "just a test\n";
Char buf2[100];
Fd[0] = open("tfile",O_RDWR);
Fd[1] = open("tfile",O_RDWR);
Write(fd[0],buf1,strlen(buf1));
Printf("\nEnter your text now…");
Gets(buf1);
Write(fd[0],buf1,strlen(buf1));
Write(1, buf2, read(fd[1],buf2,sizeof(buf2)));
Close(fd[0]);
```

```
Close(fd[1]);
Printf("\n");
Return 0;}
```

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

```
# include<stdio.h>
# define PERROR -1
Main(argc ,argv)
Int argc;
Char *argv[];{
Char op;
If((argc<3)(argc>4)){
Fprintf(stderr,"ERROR");
Exit(1);}
If(argc==4){
Printf("\n Are you sure to move the file (y/n)");
Scanf("%c",&op);
If(tolower(op)=='y'){
If(link(argv[1],argv[2])==PERROR){
Perror(argv[0]);}
If(unlink(argv[1]==PERROR){
Perror(argv[1]);}}
Else{
Else(1);}
If(argc==3){
If(link(argv[1],argv[2])==PERROR){
Perror(argv[1]);}}
Else{Exit(1);}}
If(argc==3){
If(link(argv[1],argv[2])==PERROR){
Perror(argv[1]);}}
Exit(1);}
```

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

```
#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;}
```

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

```
#include<pthread.h>
#include<stdio.h>
#include<semaphore.h>
#include<unistd.h>
Void *fun1();
 Void *fun2();
 Int shared=1;
 Sem_t s;
 Int main(){
 Sem_init(&s,0,1);
 Pthread_t thread1, thread2;
 Pthread_create(&thread1, NULL, fun1, NULL);
 Pthread_create(&thread2, NULL, fun2, NULL);
 Pthread_join(thread1, NULL);
 Pthread_join(thread2,NULL);
 Printf("Final value of shared is %d\n",shared);  }
Void *fun1(){
```

```
        Int x;
        Sem_wait(&s);
        X=shared;
        Printf("Thread1 reads the value as %d\n",x);
        X++;
        Printf("Local updation by Thread1: %d\n",x);
        Sleep(1);
        Shared=x;
 Printf("Valueof shared variable updated by Thread1 is: %d\n",shared);
        Sem_post(&s);}
    Void *fun2(){
        Int y;
        Sem_wait(&s);
        Y=shared;
        Printf("Thread2 reads the value as %d\n",y);
        y--;
        printf("Local updation by Thread2: %d\n",y);
        sleep(1);
        shared=y;
    printf("Value of shared variable updated by Thread2 is: %d\n",shared);
        sem_post(&s);}
```

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

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

```
#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);}
```

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

```
#include <stdio.h>
Int main(){
   Int incomingStream[] = {4, 1, 2, 4, 5};
   Int pageFaults = 0;
   Int frames = 3;
   Int m, n, s, pages;
   Pages = sizeof(incomingStream)/sizeof(incomingStream[0]);
   Printf("Incoming \t Frame 1 \t Frame 2 \t Frame 3");
   Int temp[frames];
   For(m = 0; m < frames; m++){
      Temp[m] = -1;}
   For(m = 0; m < pages; m++){
      S = 0;
      For(n = 0; n < frames; n++){
         If(incomingStream[m] == temp[n]) {
            S++;
            pageFaults--;}}
      pageFaults++;
      if((pageFaults <= frames) && (s == 0)){
         Temp[m] = incomingStream[m];}
```

```
        Else if(s == 0){
            Temp[(pageFaults – 1) % frames] = incomingStream[m];}
        Printf("\n");
        Printf("%d\t\t\t",incomingStream[m]);
        For(n = 0; n < frames; n++){
            If(temp[n] != -1)
                Printf(" %d\t\t\t", temp[n]);
            Else
                Printf(" - \t\t\t");} }
    Printf("\nTotal Page Faults:\t%d\n", pageFaults);
    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>
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 <stdlib.h>
Void recurse(int files[]){
   Int flag = 0, startBlock, len, j, k, ch;
   Printf("Enter the starting block and the length of the files: ");
   Scanf("%d%d", &startBlock, &len);
   For (j = startBlock; j < (startBlock + len); j++){
      If (files[j] == 0)
         Flag++; }
   If (len == flag){
      For (int k = startBlock; k < (startBlock + len); k++){
```

```
        If (files[k] == 0){
            Files[k] = 1;
            Printf("%d\t%d\n", k, files[k]); }}
      If (k != (startBlock + len – 1))
        Printf("The file is allocated to the disk\n");}
   Else
   Printf("The file is not allocated to the disk\n");
   Printf("Do you want to enter more files?\n");
   Printf("Press 1 for YES, 0 for NO: ");
   Scanf("%d", &ch);
   If (ch == 1)
     Recurse(files);
   Else
     Exit(0);
   Return;}
Int main(){
   Int files[50];
   For (int i = 0; i < 50; i++)
     Files[i] = 0;
   Printf("Files Allocated are :\n");
   Recurse(files);
   Return 0;}
```

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

```
#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();}
```

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>
Void main(){
Int f[50], p,i, st, len, j, c, k, a;
Clrscr();
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 schedul

```
#include<stdio.h>
#include<stdlib.h>
Int main(){
  Int ReadyQueue[100],i,n,TotalHeadMov=0,initial;
  Scanf(“%d”,&n);
  For(i=0;i<n;i++){
  Scanf(“%d”,&ReadyQueue[i]);  }
```

```
  Scanf("%d",&initial);
  For(i=0;i<n;i++)  {
    TotalHeadMov=TotalHeadMov+abs(ReadyQueue[i]-initial);
    Initial=ReadyQueue[i];}
  Printf("Total Head Movement=%d",TotalHeadMov);
}ing algorithm.
```

38. Design a C program to simulate SCAN disk scheduling algorithm.
39. 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;}
```

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

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
Int main(int argc, char *argv[]){
  Char *filename;
  Struct stat fs;
  Int r;
```

```
If( argc<2 )    {
   Puts(“Filename required”);
   Exit(1);  }
Filename = argv[1];
Printf(“Obtaining permission mode for ‘%s’:\n”,filename);
R = stat(filename,&fs);
If( r==-1 ) {
   Fprintf(stderr,”File error\n”);
   Exit(1); }
Printf(“Permission bits: %X\n”,fs.st_mode);   Return(0); }
```