LIST OF PROGRAMS

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 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();
}
```

7. 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);
}
```

8. 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);
}
```

9. 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);
}
```

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

```
#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");
        }
}
```

11. 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();
}
```

12. 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();
}
```

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

14. 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");
}
}
```

15 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);
}
```

16. 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;
}
```

17. 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;
}
```

18. 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;
}
```

19. 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();
}
```

20. 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();
}
```

21. 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();
}
```

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

```
#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);
}
```

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

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

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