Name : Manikanth goudar

Reg_no : 192311300

Course: Operating system for supercomputers

Course_code: CSA0477

LAB-PRACTICAL QUESTIONS

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.

```
int main() {
    FILE *file = fopen("employee.dat", "r+b");

    if (file == NULL) {
        // If file doesn't exist, create it
        file = fopen("employee.dat", "w+b");
        if (file == NULL) {
printf("Unable to create file.\n");
            return 1;
        }
    }

    int choice, id;

    while (1) {
printf("\nEmployee Management System\n");
printf("1. Add Employee\n");
printf("2. Display Employee\n");
printf("3. Update Employee\n");
printf("4. List All Employees\n");
printf("5. Delete Employee\n");
printf("6. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

    switch (choice) {
      case 1:
addEmployee(file);
        break;
      case 2:
printf("Enter Employee ID to display: ");
scanf("%d", &id);
displayEmployee(file, id);
        break;
      case 3:
printf("Enter Employee ID to update: ");
scanf("%d", &id);
updateEmployee(file, id);
        break;
      case 4:
listAllEmployees(file);
        break;
      case 5:
printf("Enter Employee ID to delete: ");
scanf("%d", &id);
deleteEmployee(file, id);
        break;
      case 6:
fclose(file);
```

```
            return 0;
          default:
printf("Invalid choice. Try again.\n");
     }
  }
}
```

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.

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

void sortRequests(int requests[], int n) {
    for (int i = 0; i< n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (requests[j] >requests[j + 1]) {
                // Swap the elements
                int temp = requests[j];
                requests[j] = requests[j + 1];
requests[j + 1] = temp;
            }
        }
    }
}
```

```
int calculateSeekTime(int requests[], int n, int start, int direction) {
    int totalSeekTime = 0;

sortRequests(requests, n);

    int startPos;
    for (int i = 0; i< n; i++) {
        if (requests[i] >= start) {
startPos = i;
            break;
        }
    }

    if (direction == 1) {
        for (int i = startPos; i< n; i++) {
totalSeekTime += abs(requests[i] - start);
            start = requests[i];
        }
        for (int i = startPos - 1; i>= 0; i--) {
totalSeekTime += abs(requests[i] - start);
            start = requests[i];
        }
    } else {
        for (int i = startPos - 1; i>= 0; i--) {
            totalSeekTime += abs(requests[i] - start);
            start = requests[i];
        }
        for (int i = startPos; i< n; i++) {
            totalSeekTime += abs(requests[i] - start);
            start = requests[i];
        }
    }
    return totalSeekTime;
}

int main() {
    int n, start, direction;

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

    int requests[n];
    printf("Enter the disk request positions: ");
    for (int i = 0; i< n; i++) {
        scanf("%d", &requests[i]);
    }

    Printf("Enter the starting position of the disk head: ");
    Scanf("%d", &start);

    Printf("Enter the direction (1 for right, -1 for left): ");
    Scanf("%d", &direction);

    int totalSeekTime = calculateSeekTime(requests, n, start, direction);

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

    return 0;
}
```

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

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

void cscan(int arr[], int n, int head, int disk_size) {
    int seek_count = 0;
    int distance = 0;
    int curr = 0;
    int left = 0;
    int right = 0;

    int left_arr[50], right_arr[50];

    for (int i = 0; i< n; i++) {
        if (arr[i] < head) {
left_arr[left++] = arr[i];
        } else {
right_arr[right++] = arr[i];
        }
    }

qsort(left_arr, left, sizeof(int), compare);
qsort(right_arr, right, sizeof(int), compare);

    for (int i = 0; i< right; i++) {
        distance = abs(head - right_arr[i]);
seek_count += distance;
        head = right_arr[i];
    }

seek_count += abs(head - (disk_size - 1));
    head = 0;  // Move to the beginning of the disk

    for (int i = left - 1; i>= 0; i--) {
        distance = abs(head - left_arr[i]);
seek_count += distance;
        head = left_arr[i];
    }
```

```
printf("Total Seek Time: %d\n", seek_count);
}

int compare(const void *a, const void *b) {
    return (*(int *)a - *(int *)b);
}

int main() {
    int n, head, disk_size;

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

    int arr[n];
printf("Enter the disk requests: ");
    for (int i = 0; i< n; i++) {
scanf("%d", &arr[i]);
    }

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

printf("Enter the size of the disk (total cylinders): ");
scanf("%d", &disk_size);

    // Call the cscan function to simulate the algorithm
cscan(arr, n, head, disk_size);

    return 0;
}
```

the First in First Out paging technique of memory management.

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

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

void print_permissions(mode_t mode) {
printf("Permissions: ");
printf((S_ISDIR(mode)) ? "d" : "-");  // Directory check
printf((mode & S_IRUSR) ? "r" : "-");  // Owner read
printf((mode & S_IWUSR) ? "w" : "-");  // Owner write
printf((mode & S_IXUSR) ? "x" : "-");  // Owner execute
printf((mode & S_IRGRP) ? "r" : "-");  // Group read
printf((mode & S_IWGRP) ? "w" : "-");  // Group write
printf((mode & S_IXGRP) ? "x" : "-");  // Group execute
printf((mode & S_IROTH) ? "r" : "-");  // Others read
printf((mode & S_IWOTH) ? "w" : "-");  // Others write
printf((mode & S_IXOTH) ? "x" : "-");  // Others execute
printf("\n");
}

int main() {
    char filename[] = "testfile.txt";  // Change to your file path



    struct stat file_stat;
    if (stat(filename, &file_stat) == -1) {
perror("stat");
        return 1;
    }
print_permissions(file_stat.st_mode);


    if (chmod(filename, S_IRUSR | S_IWUSR | S_IXUSR | S_IRGRP |
S_IXGRP | S_IROTH | S_IXOTH) == -1) {
perror("chmod");
        return 1;
    }

printf("Permissions after modification:\n");


    if (stat(filename, &file_stat) == -1) {
perror("stat");
        return 1;
    }
print_permissions(file_stat.st_mode);

    return 0;
}
```

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

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

void createFile();
void writeFile();
void readFile();
void appendFile();
void deleteFile();

int main() {
    int choice;
```

```
  while (1) {
printf("\n--- File Management Operations ---\n");
printf("1. Create/Open File\n");
printf("2. Write to File\n");
printf("3. Read from File\n");
printf("4. Append to File\n");
printf("5. Delete File\n");
printf("6. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

    switch (choice) {
      case 1:
createFile();
        break;
      case 2:
writeFile();
        break;
      case 3:
readFile();
        break;
      case 4:
appendFile();
        break;
      case 5:
deleteFile();
        break;
      case 6:
printf("Exiting program.\n");
exit(0);
      default:
printf("Invalid choice! Please try again.\n");
    }
    }
    return 0;
}

void createFile() {
    FILE *file;
    char filename[100];

printf("Enter
```

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

```
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <time.h>

void list_directory(const char *path) {
    struct dirent *entry;
    DIR *dp = opendir(path);

    if (dp == NULL) {
perror("opendir");
        return;
    }

    // Print directory contents
    while ((entry = readdir(dp)) != NULL) {
printf("%s\n", entry->d_name);
    }

closedir(dp);
```

```
}

int main(int argc, char *argv[]) {
    if (argc == 1) {
        // If no directory is provided, use the current directory
list_directory(".");
    } else {
        // List the directory pas
```

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

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int matchPattern(const char *line, const char *pattern) {

    return strstr(line, pattern) != NULL;
}

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

    if (file == NULL) {
perror("Error opening file");
        return;
    }

    char line[1024];



    while (fgets(line, sizeof(line), file) != NULL) {

        if (matchPattern(line, pattern)) {
printf("%s", line);
        }
    }

fclose(file);
}

int main(int argc, char *argv[]) {
    // Check if correct arguments are
```

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

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

#define MAX 5  // Maximum size of the buffer
int buffer[MAX];  // Shared buffer
int in = 0, out = 0;  // Buffers indices for producer and consumer

sem_t empty, full, mutex;

void *producer(void *arg) {
    int item;

    while (1) {
        item = rand() % 100;  // Produce an item
sem_wait(&empty);      // Wait for empty space in buffer
sem_wait(&mutex);      // Ensure mutual exclusion
        buffer[in] = item;     // Insert item into buffer
printf("Produced: %d at index %d\n", item, in);
        in = (in + 1) % MAX;   // Move to next buffer index
sem_post(&mutex);      // Release mutual exclusion
sem_post(&full);       // Signal that there is a new item in the buffer
sleep(1);  // Simulate time taken to produce
```

```
    }

void *consumer(void *arg) {
    int item;

    while (1) {
sem_wait(&full);      // Wait for full space in buffer
sem_wait(&mutex);
        item = buffer[out];
printf("Consumed: %d from index %d\n", item, out);
        out = (out + 1) % MAX;
sem_post(&mutex);
sem_post(&empty);
sleep(1);
    }
}

int main() {
pthread_t pr
```

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

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

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

void* print_message(void* ptr) {
printf("Hello from the thread!\n");
    return NULL;
}

int main() {
pthread_tthread_id;
```

```
    if (pthread_create(&thread_id, NULL, print_message, NULL)) {
printf("Error creating thread\n");
        return 1;
    }



pthread_join(thread_id, NULL);

printf("Main thread finished\n");

    return 0;
}
```

31. Construct a C program to simulate

management.

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

#define MAX_EMPLOYEES 10
struct Employee {
    int id;
    char name[100];
    float salary;
};



struct Employee employees[MAX_EMPLOYEES];
int employeeCount = 0;

void addEmployee() {
    if (employeeCount>= MAX_EMPLOYEES) {
printf("Error: Cannot add more employees, storage full.\n");
        return;
    }

    struct Employee newEmployee;
printf("Enter employee ID: ");
scanf("%d", &newEmployee.id);
getchar(); // To capture the newline character
printf("Enter employee name: ");
fgets(newEmployee.name, 100, stdin);
newEmployee.name[strcspn(newEmployee.name, "\n")] = '\0'; // Remove
newline
printf("Enter employee salary: ");
scanf("%f", &newEmployee.salary);

    employees[employeeCount] = newEmployee;
employeeCount++;
printf("Employee added successfully!\n");
}

void removeEmployee() {
    int id, found = 0;
printf("Enter employee ID to remove: ");
scanf("%d", &id);

    for (int i = 0; i<employeeCount; i++) {
        if (employees[i].id == id) {
            for (int j = i; j <employeeCount - 1; j++) {
                employees[j] = employees[j + 1];
            }
employeeCount--;
printf("Employee with ID %d removed successfully!\n", id);
            found = 1;
            break;
        }
    }

    if (!found) {
printf("Error: Employee not found.\n");
    }
}


void displayEmployees() {
    if (employeeCount == 0) {
printf("No employees to display.\n");
        return;
    }

printf("\nEmployee List:\n");
printf("ID\tName\t\tSalary\n");
printf("----------------------------------------\n");
    for (int i = 0; i<employeeCount; i++) {
printf("%d\t%s\t%.2f\n", employees[i].id, employees[i].name,
employees[i].salary);
    }
}
void searchEmployee() {
    int id, found = 0;
printf("Enter employee ID to search: ");
scanf("%d", &id);

    for (int i = 0; i<employeeCount; i++) {
        if (employees[i].id == id) {
printf("\nEmployee found:\n");
printf("ID: %d\n", employees[i].id);
printf("Name: %s\n", employees[i].name);
printf("Salary: %.2f\n", employees[i].salary);
```

```
            found = 1;
            break;
        }
    }

    if (!found) {
printf("Error: Employee not found.\n");
    }
}
int main() {
    int choice;

    while (1) {
printf("\n===== Employee Management System =====\n");
printf("1. Add Employee\n");
printf("2. Remove Employee\n");
printf("3. Display Employees\n");
printf("4. Search Employee\n");
printf("5. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

        switch (choice) {
            case 1:
addEmployee();
                break;
            case 2:
removeEmployee();
                break;
            case 3:
displayEmployees();
                break;
            case 4:
```

```
searchEmployee();
                break;
            case 5:
printf("Exiting the system...\n");
exit(0);
            default:
printf("Invalid choice, please try again.\n");
        }
    }

    return 0;
}
```

32. Construct a C program to simulate management.

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

#define MAX_EMPLOYEES 100
struct Employee {
    int id;
    char name[100];
    float salary;
};

struct Employee employees[MAX_EMPLOYEES];
int employeeCount = 0;



void addEmployee() {
    if (employeeCount>= MAX_EMPLOYEES) {
printf("Error: Cannot add more employees, storage full.\n");
        return;
    }

    struct Employee newEmployee;
printf("Enter employee ID: ");
scanf("%d", &newEmployee.id);
getchar();
printf("Enter employee name: ");
fgets(newEmployee.name, 100, stdin);
newEmployee.name[strcspn(newEmployee.name, "\n")] = '\0'; // Remove
newline
printf("Enter employee salary: ");
scanf("%f", &newEmployee.salary);

    employees[employeeCount] = newEmployee;
employeeCount++;
printf("Employee added successfully!\n");
}


void removeEmployee() {
    int id, found = 0;
printf("Enter employee ID to remove: ");
scanf("%d", &id);

    for (int i = 0; i<employeeCount; i++) {
        if (employees[i].id == id) {
            for (int j = i; j <employeeCount - 1; j++) {
                employees[j] = employees[j + 1];
            }
employeeCount--;
printf("Employee with ID %d removed successfully!\n", id);
            found = 1;
            break;
        }
    }

    if (!found) {
printf("Error: Employee not found.\n");
    }
}

void displayEmployees() {
    if (employeeCount == 0) {
printf("No employees to display.\n");
        return;
    }

printf("\nEmployee List:\n");
printf("ID\tName\t\tSalary\n");
printf("----------------------------------------\n");
    for (int i = 0; i<employeeCount; i++) {
printf("%d\t%s\t%.2f\n", employees[i].id, employees[i].name,
employees[i].salary);
    }
}

void searchEmployee() {
    int id, found = 0;
printf("Enter employee ID to search: ");
scanf("%d", &id);

    for (int i = 0; i<employeeCount; i++) {
        if (employees[i].id == id) {
printf("\nEmployee found:\n");
printf("ID: %d\n", employees[i].id);
printf("Name: %s\n", employees[i].name);
printf("Salary: %.2f\n", employees[i].salary);
            found = 1;
            break;
        }
    }

    if (!found) {
printf("Error: Employee not found.\n");
    }
}
int main() {
    int choice;

    while (1) {
printf("\n===== Employee Management System =====\n");
printf("1. Add Employee\n");
printf("2. Remove Employee\n");
printf("3. Display Employees\n");
printf("4. Search Employee\n");
printf("5. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

        switch (choice) {
            case 1:
addEmployee();
                break;
            case 2:
removeEmployee();
                break;
            case 3:
displayEmployees();
                break;
            case 4:
searchEmployee();
                break;
            case 5:
printf("Exiting the system...\n");
exit(0);
            default:
printf("Invalid choice, please try again.\n");
        }
    }

    return 0;
}
```

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

```
#include <stdio.h>

#define MAX_FRAMES 3

void optimal_page_replacement(int pages[], int n, int frames) {
    int memory[frames];  // Array representing frames in memory
    int i, j, k, page_faults = 0, page_found;



    for (i = 0; i< frames; i++) {
        memory[i] = -1;
    }

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

```
        for (j = 0; j < frames; j++) {
            if (memory[j] == pages[i]) {
page_found = 1;
                break;
            }
        }

        if (page_found == 0) {
page_faults++;
            int farthest = -1, replace_index = -1;

                        for (j = 0; j < frames; j+
```

34. Consider a file system 33. Construct a C program to simulate the optimal paging technique of memory management

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.

. 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

36. Illustrate the concept 35of multithreading using a C program.

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

void* print_message(void* msg) {
printf("%s\n", (char*)msg);
    return NULL;
}

int main() {
pthread_t thread1, thread2;
```

```
    char* msg1 = "Hello from Thread 1";
    char* msg2 = "Hello from Thread 2";

pthread_create(&thread1, NULL, print_message, (void*)msg1);
pthread_create(&thread2, NULL, print_message, (void*)msg2);

pthread_join(thread1, NULL);
pthread_join(thread2, NULL);

printf("Main thread ends.\n");

    return 0;
}
```

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

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

#define NUM_PHILOSOPHERS 5

pthread_mutex_tforks[NUM_PHILOSOPHERS];

void* philosopher(void* num) {
    int id = *(int*)num;

    while (1) {
printf("Philosopher %d is thinking.\n", id);
usleep(1000000);

pthread_mutex_lock(&forks[id]);
pthread_mutex_lock(&forks[(id + 1) % NUM_PHILOSOPHERS]);
printf("Philosopher %d is eating.\n", id);
usleep(1000000);

pthread_mutex_unlock(&forks[id]);
pthread_mutex_unlock(&forks[(id + 1) % NUM_PHILOSOPHERS]);
    }

    return NULL;
}

int main() {
pthread_tphilosophers[NUM_PHILOSOPHERS];
    int ids[NUM_PHILOSOPHERS];

    for (int i = 0; i< NUM_PHILOSOPHERS; i++) {
pthread_mutex_init(&forks[i], NULL);
        ids[i] = i;
    }

    for (int i = 0; i< NUM_PHILOSOPHERS; i++) {
pthread_create(&philosophers[i], NULL, philosopher, (void*)&ids[i]);
    }

    for (int i = 0; i< NUM_PHILOSOPHERS; i++) {
pthread_join(philosophers[i], NULL);
    }

    for (int i = 0; i< NUM_PHILOSOPHERS; i++) {
pthread_mutex_destroy(&forks[i]);
    }

    return 0;
}
```

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

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

#define MAX_BLOCKS 10
#define MAX_PROCESSES 5

typedef struct {
    int size;
    int is_free;
} Block;

Block memory[MAX_BLOCKS];

void first_fit(int process_size) {
    for (int i = 0; i< MAX_BLOCKS; i++) {
        if (memory[i].is_free&& memory[i].size>= process_size) {
printf("Allocating process of size %d to block %d\n", process_size, i);
            memory[i].is_free = 0;
            return;
        }
    }
printf("No suitable block found for process of size %d\n", process_size);
}

void best_fit(int process_size) {
    int best_idx = -1;
    for (int i = 0; i< MAX_BLOCKS; i++) {
        if (memory[i].is_free&& memory[i].size>= process_size) {
            if (best_idx == -1 || memory[i].size< memory[best_idx].size) {
best_idx = i;
            }
        }
    }
    if (best_idx != -1) {
printf("Allocating process of size %d to block %d\n", process_size, best_idx);
        memory[best_idx].is_free = 0;
    } else {
printf("No suitable block found for process of size %d\n", process_size);
    }
}

void worst_fit(int process_size) {
    int worst_idx = -1;
    for (int i = 0; i< MAX_BLOCKS; i++) {
        if (memory[i].is_free&& memory[i].size>= process_size) {
            if (worst_idx == -1 || memory[i].size> memory[worst_idx].size) {
worst_idx = i;
            }
        }
    }
    if (worst_idx != -1) {
printf("Allocating process of size %d to block %d\n", process_size,
worst_idx);
        memory[worst_idx].is_free = 0;
    } else {
printf("No suitable block found for process of size %d\n", process_size);
    }
}

int main() {
    // Initialize memory blocks
    for (int i = 0; i< MAX_BLOCKS; i++) {
        memory[i].size = (i + 1) * 50;
        memory[i].is_free = 1;
    }

first_fit(100);
best_fit(60);
worst_fit(150);

    return 0;
}
```

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

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

#define MAX_FILES 10

typedef struct {
    char filename[100];
} File;

typedef struct {
    File files[MAX_FILES];
    int count;
} Directory;

void create_file(Directory* dir, const char* filename) {
    if (dir->count < MAX_FILES) {
strcpy(dir->files[dir->count].filename, filename);
dir->count++;
    } else {
printf("Directory is full!\n");
    }
}

void list_files(Directory* dir) {
    if (dir->count == 0) {
printf("Directory is empty.\n");
    } else {
printf("Files in directory:\n");
        for (int i = 0; i<dir->count; i++) {
printf("%s\n", dir->files[i].filename);
        }
    }
}

int main() {
    Directory dir = {.count = 0};

create_file(&dir, "file1.txt");
create_file(&dir, "file2.txt");
create_file(&dir, "file3.txt");

list_files(&dir);

    return 0;
}
```

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

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

#define MAX_FILES 5
#define MAX_SUBDIRS 3

typedef struct {
    char filename[100];
} File;

typedef struct {
    char dirname[100];
    File files[MAX_FILES];
    int file_count;
} Subdirectory;

typedef struct {
    Subdirectory subdirs[MAX_SUBDIRS];
    int subdir_count;
} Directory;

void create_file(Directory* dir, int subdir_index, const char* filename) {
    if          (subdir_index>=          dir->subdir_count          ||          dir-
>subdirs[subdir_index].file_count>= MAX_FILES) {
printf("Error: Cannot create file.\n");
        return;
    }

    strcpy(dir->subdirs[subdir_index].files[dir-
>subdirs[subdir_index].file_count].filename, filename);
dir->subdirs[subdir_index].file_count++;
}

void create_subdir(Directory* dir, const char* subdir_name) {
    if (dir->subdir_count< MAX_SUBDIRS) {
strcpy(dir->subdirs[dir->subdir_count].dirname, subdir_name);
dir->subdirs[dir->subdir_count].file_count = 0;
dir->subdir_count++;
    } else {
printf("Error: Cannot create subdirectory.\n");
    }
}

void list_files(Directory* dir) {
    for (int i = 0; i<dir->subdir_count; i++) {
printf("Subdirectory: %s\n", dir->subdirs[i].dirname);
        for (int j = 0; j <dir->subdirs[i].file_count; j++) {
printf("\t%s\n", dir->subdirs[i].files[j].filename);
        }
    }
}

int main() {
    Directory dir = {.subdir_count = 0};

create_subdir(&dir, "Documents");
create_subdir(&dir, "Images");

create_file(&dir, 0, "file1.txt");
create_file(&dir, 1, "image1.jpg");

list_files(&dir);

    return 0;
}
```