OPERATNG SYSTEMS

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.

Code :

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

Output :

Process ID: 3044

Parent Process ID: 3044

---

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

Code:

```
#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;
}
```

Output :

```
Enter the filename to open for reading
f6.txt
Enter the filename to open for writing
f7.txt
```

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.

Code:

```
#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);
        }
```

Output:

```
Enter number of process: 4
Enter Burst Time:
P1: 12
P2: 14
P3: 15
P4: 16
P        BT      WT      TAT
P1       12      0       12
P2       14      12      26
P3       15      26      41
P4       16      41      57
Average Waiting Time= 19.750000
Average Turnaround Time= 34.000000
--------------------------------
Process exited after 17.9 seconds with return value 0
Press any key to continue . . .
```

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

Code:

```
#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);
}
```

Output:

```
Enter number of process:3
nEnter Burst Time:
p1:45
p2:32
p3:18
nProcesst    Burst Time    tWaiting TimetTurnaround Time
np3tt  18tt    0ttt18np2tt  32tt    18ttt50np1tt  45tt    50ttt95nnAverage Waiting Time=22.666666nAverage Turnaround Time=54.333332n
--------------------------------
Process exited after 8.49 seconds with return value 0
Press any key to continue . . .
```

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

Code:

```
#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;
}
```

Output:

```
Enter the total number of Processes: 3

Please Enter the  Burst Time and Priority of each process:

Enter the details of the process A
Enter the burst time: 2
Enter the priority: 1

Enter the details of the process B
Enter the burst time: 10
Enter the priority: 3

Enter the details of the process C
Enter the burst time: 6
Enter the priority: 2


Process_name     Burst Time      Waiting Time     Turnaround Time
-------------------------------------------------------------
          B               10             0               10
-------------------------------------------------------------
          C               6              10              16
-------------------------------------------------------------
          A               2              16              18
-------------------------------------------------------------


 Average Waiting Time : 8.666667
 Average Turnaround Time: 14.666667
```

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

Program:-

```
#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();
}
```

Output:

```
 Total number of process in the system: 3

 Enter the Arrival and Burst time of the Process[1]
 Arrival time is:       2

Burst time is:  33334

 Enter the Arrival and Burst time of the Process[2]
 Arrival time is:       23

Burst time is:  45

 Enter the Arrival and Burst time of the Process[3]
 Arrival time is:       27

Burst time is:  67
Enter the Time Quantum for the process:         9

 Process No              Burst Time              TAT             Waiting Time
Process No[2]            45                              121                     76
Process No[3]            67                              175                     108
Process No[1]            33334                           33444                   110
 Average Turn Around Time:      98.000000
 Average Waiting Time:  11246.666992
```

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

Code:

```
#include<iostream>
using namespace std;
void swap(int *a, int *b) {
   int temp = *a;
   *a = *b;
   *b = temp;
}
void arrangeArrival(int num, int mat[][3]) {
   for(int i=0; i<num; i++) {
      for(int j=0; j<num-i-1; j++) {
         if(mat[1][j] > mat[1][j+1]) {
            for(int k=0; k<5; k++) {
               swap(mat[k][j], mat[k][j+1]);
            }
         }
      }
   }
}

void completionTime(int num, int mat[][3]) {
   int temp, val;
   mat[3][0] = mat[1][0] + mat[2][0];
   mat[5][0] = mat[3][0] - mat[1][0];
   mat[4][0] = mat[5][0] - mat[2][0];
```

```
    for(int i=1; i<num; i++) {
      temp = mat[3][i-1];
      int low = mat[2][i];
      for(int j=i; j<num; j++) {
        if(temp >= mat[1][j] && low >= mat[2][j]) {
          low = mat[2][j];
          val = j;
        }
      }
      mat[3][val] = temp + mat[2][val];
      mat[5][val] = mat[3][val] - mat[1][val];
      mat[4][val] = mat[5][val] - mat[2][val];
      for(int k=0; k<6; k++) {
        swap(mat[k][val], mat[k][i]);
      }
    }
}
int main() {
   int num = 3, temp;
   int mat[6][3] = {1, 2, 3, 3, 6, 4, 2, 3, 4};
   cout<<"Before Arrange...\n";
   cout<<"Process ID\tArrival Time\tBurst Time\n";
   for(int i=0; i<num; i++) {
      cout<<mat[0][i]<<"\t\t"<<mat[1][i]<<"\t\t"<<mat[2][i]<<"\n";
   }
   arrangeArrival(num, mat);
   completionTime(num, mat);
   cout<<"Final Result...\n";
   cout<<"Process ID\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\n";
   for(int i=0; i<num; i++) {
      cout<<mat[0][i]<<"\t\t"<<mat[1][i]<<"\t\t"<<mat[2][i]<<"\t\t"<<mat[4][i]<<"\t\t"<<mat[5][i]<<"\n
";
   }
}
```

Output:

Before Arrange...

Process ID	Arrival Time	Burst Time
1	3	2
2	6	3
3	4	4

Final Result...

Process ID	Arrival Time	Burst Time	Waiting Time	Turnaround Time
1	3	2	0	2
3	4	4	1	5
2	6	3	3	6

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

Code:

```
#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();
}
```

Output:

```
 Total number of process in the system: 4

 Enter the Arrival and Burst time of the Process[1]
 Arrival time is:       1

Burst time is:  23

 Enter the Arrival and Burst time of the Process[2]
 Arrival time is:       2

Burst time is:  32

 Enter the Arrival and Burst time of the Process[3]
 Arrival time is:       3

Burst time is:  2

 Enter the Arrival and Burst time of the Process[4]
 Arrival time is:       4

Burst time is:  45
Enter the Time Quantum for the process:         5

 Process No              Burst Time              TAT             Waiting Time
Process No[3]            2                               9                       7
Process No[1]            23                              64                      41
Process No[2]            32                              85                      53
Process No[4]            45                              98                      53
 Average Turn Around Time:      38.500000
 Average Waiting Time:  64.000000
```

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

Code:

```
#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);

}

Output:

```
vbnet

Data written to shared memory: Hello, shared memory!
```

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

Code:

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

Output:

```
arduino

Producer: Data sent to message queue: Hello, message queue!
Consumer: Data received from message queue: Hello, message queue!
```

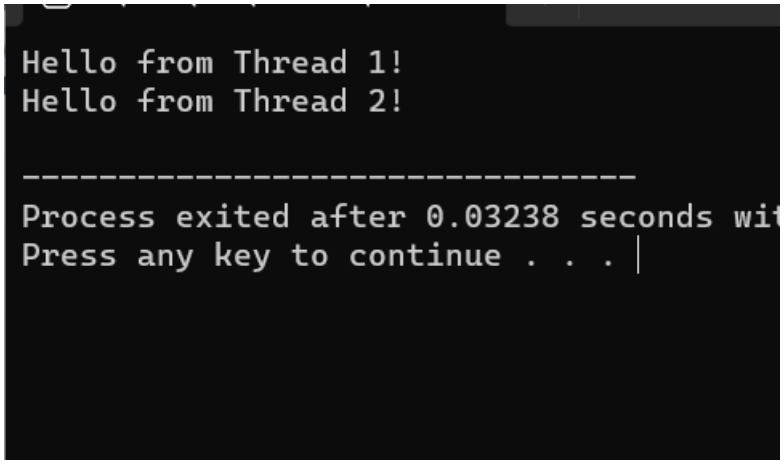
11. Illustrate the concept of multithreading using a C program.

Code:

```
#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);
}
```

Output:



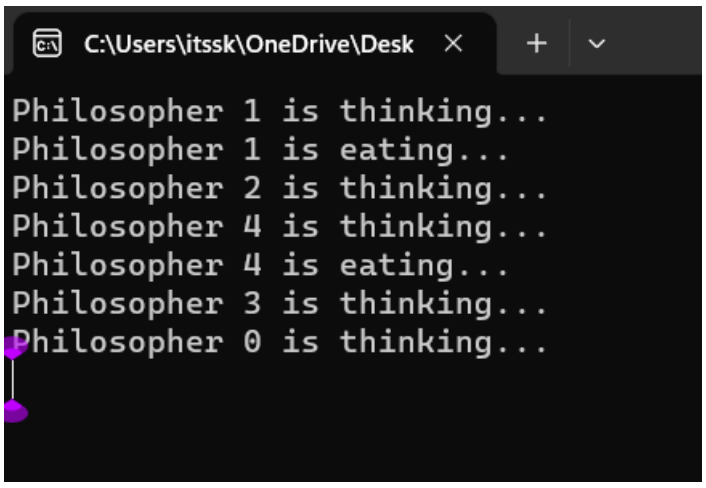
12.Design a C program to simulate the concept of Dining-Philosophers problem.

Code:

```
#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);
}
```

Output:



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

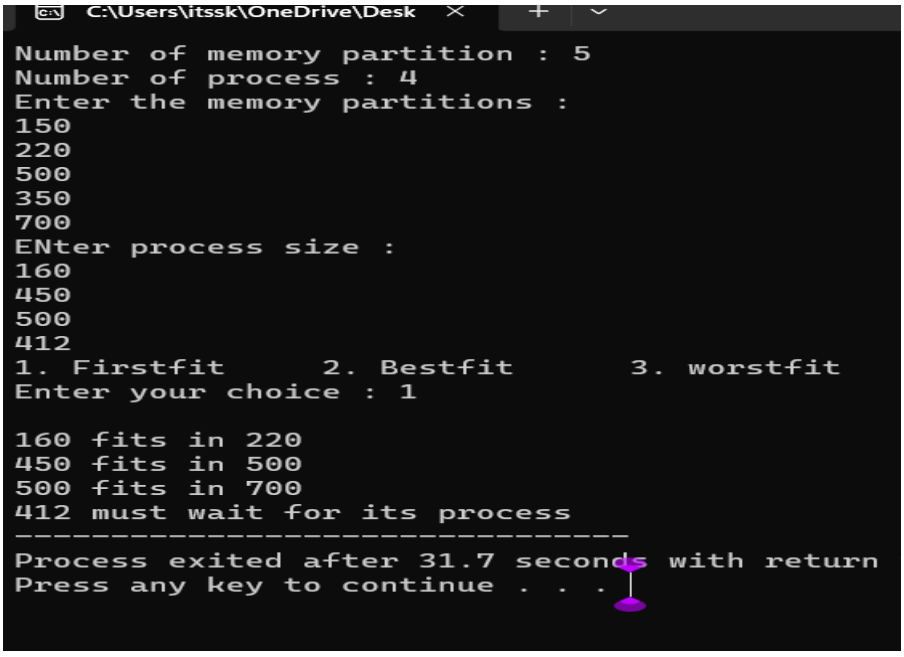
Code:

```
#include<stdio.h>
void bestfit(int mp[],int p[],int m,int n){int
        j=0;
        for(int i=0;i<n;i++){
                if(mp[i]>p[j]){
                        printf("\n%d fits in %d",p[j],mp[i]);
                        mp[i]=mp[i]-p[j++];
                        i=i-1;
                }
        }
        for(int i=j;i<m;i++)
        {
                printf("\n%d must wait for its process",p[i]);
        }
}

void rsort(int a[],int n){ for(int
        i=0;i<n;i++){
                for(int j=0;j<n;j++){
                        if(a[i]>a[j]){
                                int    t=a[i];
                                a[i]=a[j];
                                a[j]=t;
                        }
                }
        }
}

void sort(int a[],int n){ for(int
        i=0;i<n;i++){
                for(int j=0;j<n;j++){
                        if(a[i]<a[j]){
```

```
                                int    t=a[i];
                                a[i]=a[j];
                                a[j]=t;
                        }
                }
        }
}
void firstfit(int mp[],int p[],int m,int n){
        sort(mp,n);
        sort(p,m);
        bestfit(mp,p,m,n);
}
void worstfit(int mp[],int p[],int m,int n){
        rsort(mp,n);
        sort(p,m);
        bestfit(mp,p,m,n);
}
int main(){
        int  m,n,mp[20],p[20],ch; printf("Number of
        memory partition : ");scanf("%d",&n);

        printf("Number of process : ");
        scanf("%d",&m);
        printf("Enter the memory partitions : \n");for(int
        i=0;i<n;i++){
                scanf("%d",&mp[i]);
        }
        printf("ENter process size : \n");
        for(int i=0;i<m;i++){
                scanf("%d",&p[i]);
        }
```

```
        printf("1. Firstfit\t2. Bestfit\t3. worstfit\nEnter your choice : ");
        scanf("%d",&ch);
        switch(ch){
        case 1:
        bestfit(mp,p,m,n);
        break;
        case 2:
                firstfit(mp,p,m,n);
                break;
                case 3:
                        worstfit(mp,p,m,n);
                        break;
        default:
                printf("invalid");
                break;
        }
}
```

Output:

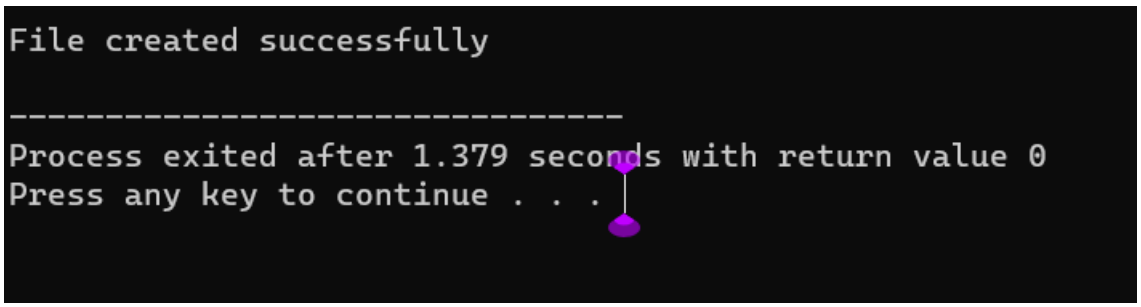


14.Design a C program to organize the file using two level directorystructure.

Code:

```
#include<stdio.h>
#include <string.h>
    int main() {
      char mainDirectory[] = "C:/Users/itssk/OneDrive/Desktop";char
      subDirectory[] = "os";
      char fileName[] = "example.txt";char
      filePath[200];
      char mainDirPath[200];
      snprintf(mainDirPath, sizeof(mainDirPath), "%s/%s/", mainDirectory,subDirectory);
      snprintf(filePath, sizeof(filePath), "%s%s", mainDirPath, fileName);FILE *file
      = fopen(filePath, "w");
      if (file == NULL) { printf("Error
          creating file.\n");return 1;
      }
      fprintf(file, "This is an example file content.");
      printf("File created successfully: %s\n");
}
```

OUTPUT :



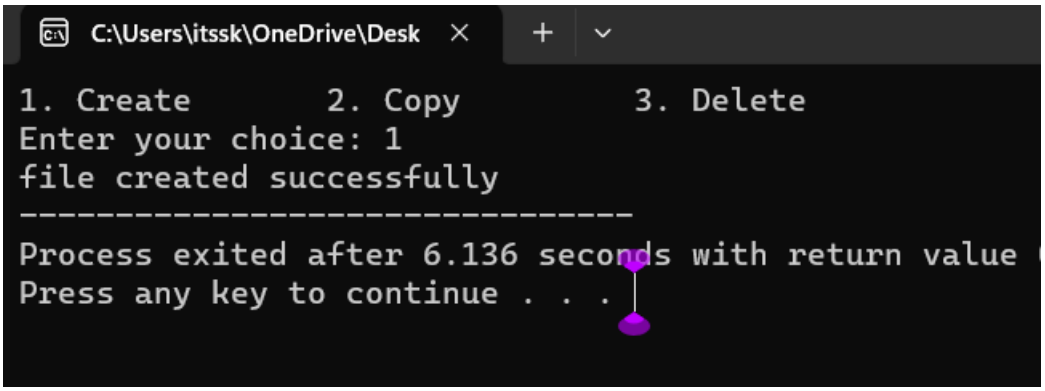
15. **Construct a C program to organize the file using single leveldirectory**

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#define BUFFER_SIZE 4096void
copy(){
        const char *sourcefile=
"C:/Users/itssk/OneDrive/Desktop/sasi.txt";
    const char *destination_file="C:/Users/itssk/OneDrive/Desktop/sk.txt";int
    source_fd = open(sourcefile, O_RDONLY);
```

```
    int dest_fd = open(destination_file, O_WRONLY | O_CREAT |O_TRUNC,
0666);
    char buffer[BUFFER_SIZE]; ssize_t
    bytesRead, bytesWritten;
    while ((bytesRead = read(source_fd, buffer, BUFFER_SIZE)) > 0) {bytesWritten =
        write(dest_fd, buffer, bytesRead);
    }
    close(source_fd);
    close(dest_fd);
    printf("File copied successfully.\n");
}
void create()
{
char path[100];
        FILE *fp; fp=fopen("C:/Users/itssk/OneDrive/Desktop/sasi.txt","w");
        printf("file created successfully");
}

int main(){
        int n;
        printf("1. Create \t2. Copy \t3. Delete\nEnter your choice: " );
        scanf("%d",&n);
        switch(n){
                case 1:
        create(); break;
        case 2:
                copy();
                break;
                case 3:
                        remove("C:/Users/itssk/OneDrive/Desktop/sasi.txt");printf("Deleted
                        successfully");
}}
```

OUTPUT :



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

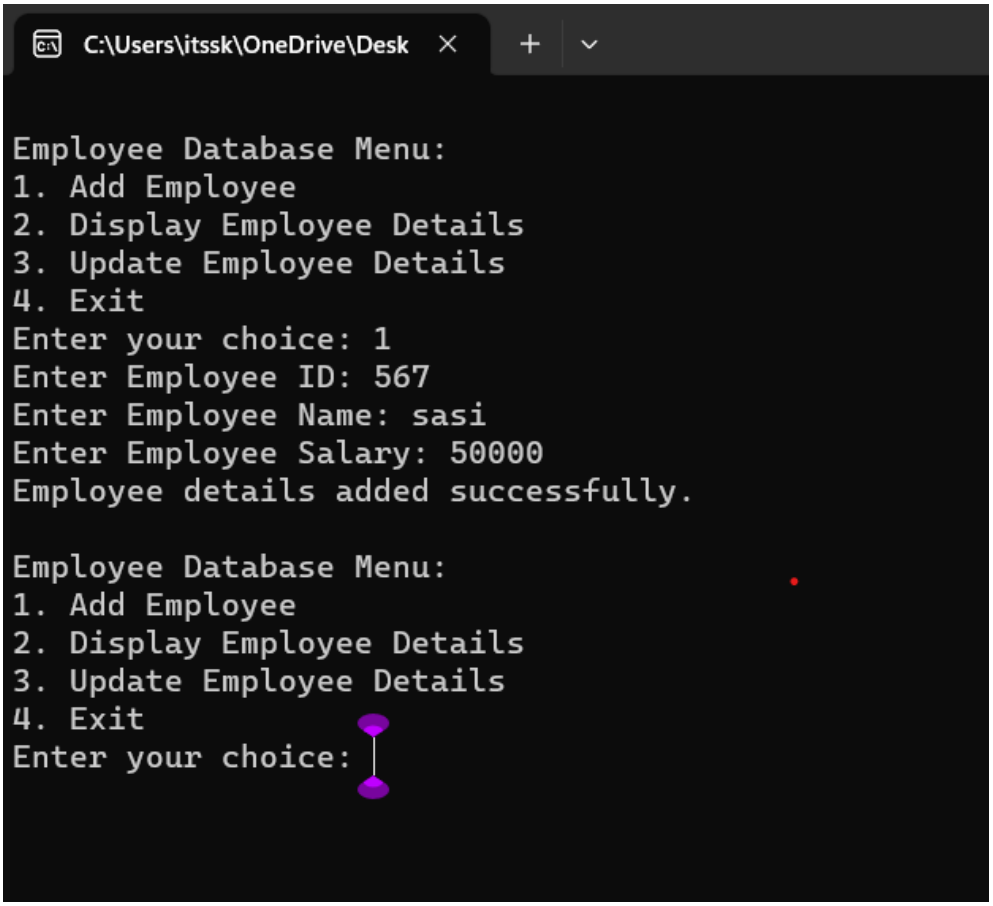
Code:

```
#include<stdio.h>
#include<stdlib.h>
struct Employee {
    int empId;
    char empName[50];
    float empSalary;};
int main() { FILE
    *filePtr;
    struct Employee emp;
    filePtr = fopen("employee.dat", "rb+");if
    (filePtr == NULL) {
        filePtr = fopen("employee.dat", "wb+");if
        (filePtr == NULL) {
            printf("Error creating the file.\n");
            return 1;     }
    }
    int choice;
    do {
        printf("\nEmployee      Database      Menu:\n");
        printf("1. Add Employee\n");
        printf("2.    Display    Employee    Details\n");
```

```
        printf("3. Update Employee Details\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice); switch
        (choice) {
            case 1:
                printf("Enter Employee ID: "); scanf("%d",
                &emp.empId); printf("Enter Employee
                Name: ");
                scanf("%s", emp.empName);
                printf("Enter Employee Salary: ");
                scanf("%f", &emp.empSalary);
                fseek(filePtr, (emp.empId - 1) * sizeof(struct Employee),
SEEK_SET);
                fwrite(&emp, sizeof(struct Employee), 1, filePtr);
                printf("Employee details added successfully.\n");break;
            case 2:
                printf("Enter Employee ID to display: ");
                scanf("%d", &emp.empId);
                fseek(filePtr, (emp.empId - 1) * sizeof(struct Employee),
SEEK_SET);
                fread(&emp, sizeof(struct Employee), 1, filePtr);
                printf("Employee ID: %d\n", emp.empId); printf("Employee
                Name: %s\n", emp.empName); printf("Employee Salary:
                %.2f\n", emp.empSalary);break;
            case 3:
                printf("Enter Employee ID to update: ");
                scanf("%d", &emp.empId);
                fseek(filePtr, (emp.empId - 1) * sizeof(struct Employee),
SEEK_SET);
                fread(&emp, sizeof(struct Employee), 1, filePtr);
                printf("Enter Employee Name: ");
                scanf("%s", emp.empName);
                printf("Enter Employee Salary: ");
                scanf("%f", &emp.empSalary);
```

```
            fseek(filePtr, (emp.empId - 1) * sizeof(struct Employee),
SEEK_SET);
            fwrite(&emp, sizeof(struct Employee), 1, filePtr);
            printf("Employee details updated successfully.\n");break;
        case 4:
            break;
        default:
            printf("Invalid choice. Please try again.\n");
    }
} while (choice != 4);
fclose(filePtr);
return 0;
```

OUTPUT :



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

Code:

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

Output:

Sample input:

********** **Banker's Algo** **********

Enter the no of Processes 5

Enter the no of resources instances 3

Enter the Max Matrix

7 5 3

3 2 2

9 0 2

2 2 2

4 3 3

Enter the Allocation Matrix

0 1 0

2 0 0

3 0 2

2 1 1

0 0 2

Enter the available Resources

3 3 2

Output:

Process	Allocation	Max	Available
P1	0 1 0	7 5 3	3 3 2
P2	2 0 0	3 2 2	
P3	3 0 2	9 0 2	
P4	2 1 1	2 2 2	
P5	0 0 2	4 3 3	

P1->P3->P4->P2->P5->

The system is in safe state

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

Code:

```
#include<stdio.h>
#include<stdlib.h>
int mutex=1,full=0,empty=3,x=0;
int main()
{
   int n;
   void producer();
   void consumer();
   int wait(int);
   int signal(int);
   printf("\n1.Producer\n2.Consumer\n3.Exit");
   while(1)
   {
      printf("\nEnter your choice:");
```

```
        scanf("%d",&n);
        switch(n)
        {
           case 1:    if((mutex==1)&&(empty!=0))
                         producer();
                      else
                         printf("Buffer is full!!");
                      break;
           case 2:    if((mutex==1)&&(full!=0))
                         consumer();
                      else
                         printf("Buffer is empty!!");
                      break;
           case 3:
                      exit(0);
                      break;
        }
    }
    return 0;
}
int wait(int s)
{
    return (--s);
}
int signal(int s)
{
    return(++s);
}
void producer()
{
    mutex=wait(mutex);
    full=signal(full);
```

```
   empty=wait(empty);
   x++;
   printf("\nProducer produces the item %d",x);
   mutex=signal(mutex);
}
void consumer()
{
   mutex=wait(mutex);
   full=wait(full);
   empty=signal(empty);
   printf("\nConsumer consumes item %d",x);
   x--;
   mutex=signal(mutex);
}
```

---

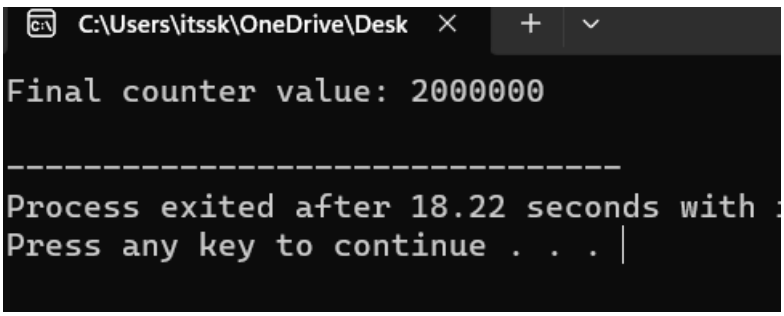
19. Design a C program to implement process synchronization using mutex locks.

PROGRAM :

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

// Shared
variablesint
counter = 0;
pthread_mutex_t mutex;

// Function to be executed by
threadsvoid
*threadFunction(void *arg) {
    int i;
    for (i = 0; i < 1000000; ++i) {          }
```

```
    return NULL;
}

int main() {
  pthread_mutex_init(&mutex,
   NULL);pthread_t thread1,
   thread2;
    pthread_create(&thread1, NULL, threadFunction, NULL);
    pthread_create(&thread2, NULL, threadFunction, NULL);

    // Wait for the threads to
    finish
    pthread_join(thread1,
    NULL);
    pthread_join(thread2,
    NULL);

    // Destroy the mutex
    pthread_mutex_destroy(&mutex);

    // Print the final value of the counter
    printf("Final counter value: %d\n",
    counter);


    return 0;
}
```

OUTPUT :



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

PROGRAM :

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

sem_t mutex, writeBlock;
int data = 0, readersCount = 0;

void *reader(void
        *arg) {int
        i=0;
    while (i<10) {
        sem_wait(&
        mutex);
        readersCount
        ++;
        if (readersCount == 1)
            {
            sem_wait(&writeB
            lock);
        }
        sem_post(&mutex);

        // Reading operation
        printf("Reader reads data: %d\n", data);

        sem_wait(&
        mutex);
        readersCount
```

```
        --;
        if (readersCount == 0)
            {
            sem_post(&writeB
            lock);
        }
        sem_post(&mutex);
        i++;
    }
}
void *writer(void
          *arg) {int
          i=0;
    while (i<10) {
        sem_wait(&writeB
        lock);
        // Writing
        operation
        data++;
        printf("Writer writes data: %d\n", data);
        sem_post(&writeB
        lock);i++;
    }
}
int main() {
    pthread_t readers,
    writers;
    sem_init(&mutex, 0,
    1);
    sem_init(&writeBlock, 0, 1);
   pthread_create(&readers, NULL, reader,
   NULL);pthread_create(&writers, NULL, writer,
```

```
    NULL); pthread_join(readers, NULL);
    pthread_join(writers, NULL);
    sem_destroy(&mutex);
     sem_destroy(&writeBl
    ock);return 0;
}
```

OUTPUT :