CS1035 OPERATING SYSTEMS LAB L T P C

Contact hours - 30 0 0 2 1

LIST OF EXPERIMENTS

1. Write programs using the following system calls of Linux operating system: Fork, exec, getpid, exit, wait, close, stat, opendir, readdir.
2. Write programs using the I/O system calls of Linux operating system (open, read, write,etc), ls, grep Commands.
3. Simulate the following CPU scheduling algorithm.
   A. Round Robin.
   B. SJF
   C. FCFS
   D. Priority
4. Simulate all file allocation strategies.
   A. Sequential
   B. Indexed
   C. Linked
5. Simulate MVT and MFT.
6. Implementation of Bankers Algorithm for Dead Lock Avoidance.
7. Simulate an Algorithm for Dead Lock Detection.
8. Simulate all page replacement algorithms.
   A. FIFO
   B. LRU
   C. LFU
9. Simulate all File Organization Techniques.
   A. Single level directory
   B. Two level
   C. Hierarchical
10. Simulate Paging Technique of memory management.
11. Simulate Shared memory and IPC.
12. Implement Threading & Synchronization Applications.

REFERENCE

Laboratory Manual

https://www.scribd.com/doc/48254171/OS-CD-LAB-MANUAL

file:///D:/OS%20LAB%20MANUAL_0.pdf

Ex.No: 3a

ROUND ROBIN

AIM:

To write a program to implement the Round Robin (RR) CPU scheduling.

ALGORITHM:

Step 1: Start the process

Step 2: Get the number of Processes

Step 3: Get the value for burst time for individual processes

Step 4: Get the value for time quantum

Step 5: Make the CPU scheduler go around the ready queue allocating CPU to each process for the time interval specified

Step 6: Make the CPU scheduler pick the first process and set time to interrupt after quantum. And after it's expiry dispatch the process

Step 7: If the process has burst time less than the time quantum then the process are released by the CPU

Step 8: If the process has burst time greater than time quantum then it is interrupted by the OS and the process is put to the tail of ready queue and the schedule selects next process from head of the queue

Step 9: Calculate the total and average waiting time and turnaround time and display the results

Step 10: Stop the process

PROGRAM:

```
#include<stdio.h>
struct process
{
int at,bt,wt,tat,st,ft,flag,id,tbt;
}p[10],temp;
int n,t,save_et[10],save_id[10],turn,btsum;
float awt,atat;
void read();
void print();
void rndrbn();
void fifoq();
main()
{
int ch;
read();
fifoq();
rndrbn();
print();
}
void read()
{
int i;
printf("Enter no of processes:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter arriving time ,burst time of process p%d : ",(i+1));
scanf("%d%d",&p[i].at,&p[i].bt);
p[i].id=i+1;
p[i].wt=p[i].flag=0;
p[i].tbt=p[i].bt;
btsum+=p[i].bt;
}
printf("Enter time quantum : ");
scanf("%d",&t);
}
void fifoq()
{
int i,j;
for(i=0;i<n;i++)
{
```

```
for(j=0;j<n-i-1;j++)
{
if(p[j].at>p[j+1].at)
{
temp=p[j];
p[j]=p[j+1];
p[j+1]=temp;
}
}
}
}
void rndrbn()
{
int cnt=n;
int i=0;
int et=0;
int sum=0;
float twt=0;
float ttat=0;
while(cnt!=0)
{
if((p[i].bt)>t)
{
et=t;
p[i].bt-=t;
}
else
{
et=p[i].bt;
p[i].bt=0;
}
```

```
p[i].st=sum;
if((p[i].flag)==0)
{
p[i].wt=p[i].st-p[i].at;
p[i].flag++;
}
else
p[i].wt=p[i].wt+(p[i].st-p[i].ft);
sum=sum+et;
p[i].ft=sum;
save_et[turn]=et;
save_id[turn++]=p[i].id;
if((p[i].bt)==0)
{
cnt--;
}
do
{
i=(i+1)%n;
}while((p[i].bt)==0 && cnt!=0);
}
for(i=0;i<n;i++)
{
p[i].tat=p[i].wt+p[i].tbt;
twt+=p[i].wt;
ttat+=p[i].tat;
}
awt=twt/n;
atat=ttat/n;
}
void print()
```

```
{
int i,sum=0;
for(i=0;i<=btsum;i++)
printf("---");
printf("\n");
printf("| ");
for(i=0;i<turn;i++)
printf("  %p*d  |",-(save_et[i]-1),save_id[i]);
printf(" ");
printf("\n");
for(i=0;i<=btsum;i++)
printf("---");
printf("\n");
printf(" ");
for(i=0;i<turn;i++)
{
printf("%p*d    ",-(save_et[i]),sum);
sum+=save_et[i];
}
printf("%d\n",sum);
printf("\nPid\tWT\t TT");
for(i=0;i<n;i++)
{
printf("\n%d\t %d \t%d\n",p[i].id,p[i].wt,p[i].tat);
}
printf("AWT=%f\t\t ATT=%.2f\n",awt,atat);
btsum=0;
}
```

OUTPUT:

```
[2cse@localhost ~]$ ./a.out
Enter no of processes:3
Enter arriving time ,burst time of process p1 : 0 5
Enter arriving time ,burst time of process p2 : 2 6
Enter arriving time ,burst time of process p3 : 3 8
Enter time quantum : 2
-----------------------------------------------------------------
|   1   |  2  |  3  |  1  |  2   |  3  |  1   |  2   |  3   |  3  |
-----------------------------------------------------------------
 0    2      4      6      8     10    12     13     15     17     19

Pid     WT       TT
1        8      13
2        7      13
3        8      16

AWT=7.666667             ATT=14.00
```

RESULT:

Thus the program for Round Robin Scheduling was executed and verified successfully.

Ex.No: 3b

SHORTEST JOB FIRST

AIM:

To write a program for shortest job first (SJF) scheduling algorithm.

ALGORITHM:

Step 1: Start the program.

Step 2: Declare and Initialize the variables.

Step 3: Get the number of process and its burst time.

Step 4: Re-arrange the burst times using "BUBBLE SORT" in ascending order.

Step 5: Calculate the average turnaround time and waiting time of each process.

Step 6: Twt=Twt+(Wt[i]-A[i]);

Step 7: Ttt=Ttt+((Wt[i]+Bu[i])-A[i]);

Step 8: Att=(float)Ttt/n;

Step 9: Awt=(float)Twt/n;

Step 10: Display the results.

Step 11: Stop the program.

PROGRAM:

```
#include<stdio.h>
#include<string.h>
int Twt,Ttt,A[20],Wt[20],n,Bu[20],B[10];
float Att,Awt;
char pname[20][20];
void Getdata();
void Gantt_chart();
void Sjf();
void Getdata()
{
   int i;
   printf("\n Enter the number of processes: ");
   scanf("%d",&n);
   for(i=1;i<=n;i++)
   {
      fflush(stdin);
      printf("\n\n Enter the process name: ");
      scanf("%s",&pname[i]);
      printf("\n Enter The BurstTime for Process %s = ",pname[i]);
      scanf("%d",&Bu[i]);
      printf("\n Enter the Arrival Time for Process %s =  ",pname[i]);
      scanf("%d",&A[i]);
   }
}
void Gantt_chart()
{
   int i;
   printf("\n\nGANTT CHART");
   printf("\n------------------------------------------------------\n");
   for(i=1;i<=n;i++)
      printf("|\t%s\t",pname[i]);
   printf("|\t\n");
   printf("\n-------------------------------------------------------\n");
   printf("\n");
   for(i=1;i<=n;i++)
      printf("%d\t\t",Wt[i]);
  printf("%d",Wt[n]+B[n]);
   printf("\n--------------------------------------------------------\n");
   printf("\n");
 }
void Sjf()
{
   int w,t,i,Tt=0,temp,j;
   char S[10],c[20][20];
   int temp1;
```

```
printf("\n\n SHORTEST JOB FIRST SCHEDULING ALGORITHM \n\n");
Twt=Ttt=0;
w=0;
for(i=1;i<=n;i++)
{
   B[i]=Bu[i];
   S[i]='T';
   Tt=Tt+B[i];
}
for(i=1;i<=n;i++)
{
   for(j=3;j<=n;j++)
   {
      if(B[j-1]>B[j])
      {
         temp=B[j-1];
         temp1=A[j-1];
         B[j-1]=B[j];
         A[j-1]=A[j];
         B[j]=temp;
         A[j]=temp1;
         strcpy(c[j-1],pname[j-1]);
         strcpy(pname[j-1],pname[j]);
         strcpy(pname[j],c[j-1]);
      }
   }
}
Wt[1]=0;
w=w+B[1];
t=w;
S[1]='F';
```

```
    while(w<Tt)
    {
        i=2;
        while(i<=n)
        {
            if(S[i]=='T'&&A[i]<=t)
            {
                Wt[i]=w;
                S[i]='F';
                w=w+B[i];
                t=w;
                i=2;
            }
            else
                i++;
        }
    }
    for(i=1;i<=n;i++)
    {
        Twt=Twt+(Wt[i]-A[i]);
        Ttt=Ttt+((Wt[i]+Bu[i])-A[i]);
    }
    Att=(float)Ttt/n;
    Awt=(float)Twt/n;
    printf("\n\n Average Turn around time=%3.2f ms ",Att);
    printf("\n\n AverageWaiting Time=%3.2f ms",Awt);
    Gantt_chart();
}
void main()
{
    Getdata();
    Sjf();
}
```

OUTPUT:

```
[2cse@localhost ~]$ cc sjf.c
[2cse@localhost ~]$ ./a.out

 Enter the number of processes: 3
 Enter the process name: p1
 Enter The BurstTime for Process p1 = 5
 Enter the Arrival Time for Process p1 =  0
 Enter the process name: p2
 Enter The BurstTime for Process p2 = 8
 Enter the Arrival Time for Process p2 =  1
 Enter the process name: p3
 Enter The BurstTime for Process p3 = 3
 Enter the Arrival Time for Process p3 =  2
 SHORTEST JOB FIRST SCHEDULING ALGORITHM
 Average Turn around time=8.67 ms
 AverageWaiting Time=3.33 ms

GANTT CHART
-----------------------------------------
|      p1      |      p3      |      p2      |
------------------------------------------
0            5            8            16
-------------------------------------------
```

RESULT:

Thus the program to implement the SJF (Shortest Job First) scheduling Algorithm was written, executed and the output was verified successfully.

Ex. No.: 3c

FCFS

AIM:

To write a program for first come first serve (FCFS) scheduling algorithm.

ALGORITHM:

```
Step 1: Start the program the program.
Step 2: Declare and Initialize the variables.
Step 3: Get the number of process, its burst time and arrival time.
Step 4: Calculate the average turnaround time and waiting time of each process.
        Twt=Twt+(Wt[i]-A[i]);
        Ttt=Ttt+((Wt[i]+Bu[i])-A[i]);
        Att=(float)Ttt/n;
        Awt=(float)Twt/n;
Step 5: Display the result.
Step 6: Stop the program.
```

PROGRAM:

```
#include<stdio.h>
#include<string.h>
int n,Bu[20],Twt,Ttt,A[10],Wt[10],w;
float Awt,Att;
char pname[20][20],c[20][20];
void Getdata();
void Gantt_chart();
void Calculate();
void fcfs();
void Getdata()
{   int i;
   printf("\n Enter the number of processes: ");
   scanf("%d",&n);
   for(i=1;i<=n;i++)
   {     fflush(stdin);
```

```
        printf("\n\n Enter the process name: ");
        scanf("%s",&pname[i]);
        printf("\n Enter The BurstTime for Process %s = ",pname[i]);
        scanf("%d",&Bu[i]);
        printf("\n Enter the Arrival Time for Process %s =  ",pname[i]);
        scanf("%d",&A[i]);
    }
}
void Gantt_chart()
{   int i;
    printf("\n\n\t\t\tGANTT CHART\n");
    printf("\n--------------------------------------------------------\n");
    for(i=1;i<=n;i++)
        printf("|\t%s\t",pname[i]);
    printf("|\t\n");
    printf("\n--------------------------------------------------------\n");
    printf("\n");
    for(i=1;i<=n;i++)
        printf("%d\t\t",Wt[i]);
    printf("%d",Wt[n]+Bu[n]);
    printf("\n--------------------------------------------------------\n");
    printf("\n");
 }
void Calculate()
{   int i;
    Wt[1]=0;
    for(i=2;i<=n;i++)
    {
        Wt[i]=Bu[i-1]+Wt[i-1];
    }
    for(i=1;i<=n;i++)
    {     Twt=Twt+(Wt[i]-A[i]);
       Ttt=Ttt+((Wt[i]+Bu[i])-A[i]);
    }
    Att=(float)Ttt/n;
    Awt=(float)Twt/n;
    printf("\n\n Average Turn around time=%3.2f ms ",Att);
    printf("\n\n AverageWaiting Time=%3.2f ms",Awt);
}
void fcfs()
{   int i,j,temp, temp1;
    Twt=0;
    Ttt=0;
    for(i=1;i<=n;i++)
    {
        for(j=i+1;j<=n;j++)
        {
            if(A[i]>A[j])
            {  temp=Bu[i];
               temp1=A[i];
               Bu[i]=Bu[j];
               A[i]=A[j];
               Bu[j]=temp;
               A[j]=temp1;
               strcpy(c[i],pname[i]);
               strcpy(pname[i],pname[j]);
               strcpy(pname[j],c[i]);
            }
        }
    }
    Calculate();
    Gantt_chart();
```

```
}
void main()
{
   int ch;
   Getdata();
   fcfs();
}
```

OUTPUT:

```
[2cse@localhost ~]$ cc fcfs.c
 [2cse@localhost ~]$ ./a.out
 Enter the number of processes: 3
 Enter the process name: p1
 Enter The BurstTime for Process p1 = 4
 Enter the Arrival Time for Process p1 =  0
 Enter the process name: p2
 Enter The BurstTime for Process p2 = 6
 Enter the Arrival Time for Process p2 =  1
 Enter the process name: p3
 Enter The BurstTime for Process p3 = 8
 Enter the Arrival Time for Process p3 =  2
Average Turn around time=9.67 ms
 AverageWaiting Time=3.67 ms
                  GANTT CHART
-----------------------------------------
|      p1      |      p2      |      p3      |
-----------------------------------------
0             4             10             18
-----------------------------------------
```

RESULT:

Thus the program to implement the FCFS (First Come First Serve) scheduling Algorithm was written, executed and the output was verified successfully.

Ex.No.: 3d

PRIORITY

AIM:

To write the program to perform priority scheduling.

ALGORITHM:

```
Step 1: Start the program.
Step 2: Get the number of processes, their burst time and priority.
Step 3: Initialize the waiting time for process 1 is 0.
Step 4: Based upon the priority processes are arranged.
Step 5: The waiting time and turnaround time for other processes are calculated as
        twait=twait+wait[i];
        totl=totl+tta;
Step 6: The waiting time and turnaround time for all the processes are summed and then the
         average waiting time and turnaround time are calculated.
        wavg=twait/n;
        tavg=totl/n;
Step 7: The average waiting time and turnaround time are displayed.
Step 8: Stop the program.
```

PROGRAM:

```
#include<stdio.h>
main()
{
int prs[10],prty[10];
int n,i,j,twait=0,tta=0,tot=0,totl=0,temp,temp1;
int wait[10],bst[10],p[10];
float wavg,tavg;
printf("\n\t\tPRIORITY SCHEDULING");
printf("\n enter the no of process:");
scanf("%d",&n);
for(i=1;i<=n;i++)
```

```
{
        printf("\n enter the process name: P");
        scanf("%d",&prs[i]);

        printf("\n enter the burstTime");
        scanf("%d",&bst[i]);

        printf("\n enter the priority");
        scanf("%d",&prty[i]);
}
for(i=1;i<=n;i++)
{
        for(j=i+1;j<=n;j++)
        {
                if(prty[i]>=prty[j])
                {
                temp=bst[i];
                bst[i]=bst[j];
                bst[j]=temp;
                temp1=prs[i];
                prs[i]=prs[j];
                prs[j]=temp1;
                }
        }
}
printf ("\n\n\n sorted process with priority");
printf("\n-------------------------------------");
printf("\n Processname \t burst Time \n");
printf("\n-------------------------------------\n");
for(i=1;i<=n;i++)
{
```

```
printf("\tp%d \t %d \n\n",prs[i],bst[i]);
}
printf("\n\n");
printf("\n------------------------------------------------------------");
printf("\n Processor\tBursttime\tTurnaroundtime\tWaitingtime\n");
printf("\n------------------------------------------------------------");
for(i=1;i<=n;i++)
{
tta=tta+bst[i];
wait[i]=tta-bst[i];
printf("\n\tp%d\t%d\t\t%d\t\t%d",prs[i],bst[i],tta,wait[i]);
twait=twait+wait[i];
totl=totl+tta;
}
wavg=twait/n;
tavg=totl/n;
printf("\n\n\n\t\t***GRANTT CHART***\n");
printf("\n_______________________________________________________\n");
for(i=1;i<=n;i++)
printf("|\tp%d\t",prs[i]);
printf("|\t\n");
printf("\n_______________________________________________________\n");
printf("\n");
for(i=1;i<=n;i++)
printf("%d\t\t",wait[i]);
printf("%d",wait[n]+bst[n]);
printf("\n");
printf("\n\n Total burst time is :%d",tta);
printf("\n\n Total turnaround time :%d",totl);
printf("\n\n The average turnaround time:%f",tavg);
printf("\n\n Total waiting time :%d",twait);
printf("\n\n The avg waiting time :%f",wavg);
}
```

OUTPUT:

```
[2cse@localhost ~]$ cc pri.c
[2cse@localhost ~]$ ./a.out

             PRIORITY SCHEDULING
 enter the no of process:5


 enter the process name: P1
 enter the burstTime5
 enter the priority2
 enter the process name: P2
 enter the burstTime6
 enter the priority4
 enter the process name: P3
 enter the burstTime2
 enter the priority3
 enter the process name: P4
 enter the burstTime8
 enter the priority1
 enter the process name: P5
 enter the burstTime7
 enter the priority5
sorted process with priority
------------------------------------
 Processname     burst Time
-----------------------------------
       p4      8
```

```
        p1      5
        p3      2
        p2      6
        p5      7

----------------------------------------------------------
 Processor    Bursttime    Turnaroundtime  Waitingtime
----------------------------------------------------------
        p4         8              8              0
        p1         5              13             8
        p3         2              15             13
        p2         6              21             15
        p5          7             28             21
              ***GRANTT CHART***
__________________________________________________________
|     p4     |     p1     |     p3     |     p2     |     p5|
__________________________________________________________
0           8           13           15           21        28

 Total burst time is :28
 Total turnaround time :85
 The average turnaround time:17.000000
 Total waiting time :57
 The avg waiting time :11.000000
```

RESULT:

Thus the program for priority scheduling was executed successfully.

Ex. No.: 4a

SEQUENTIAL FILE ALLOCATION

AIM:

Write a C Program to implement Sequential File Allocation method.

ALGORITHM:

Step 1: Start the program.

Step 2: Get the number of memory partition and their sizes.

Step 3: Get the number of processes and values of block size for each process.

Step 4: First fit algorithm searches the entire entire memory block until a hole which is big enough is encountered. It allocates that memory block for the requesting process.

Step 5: Best-fit algorithm searches the memory blocks for the smallest hole which can be allocated to requesting process and allocates if.

Step 6: Worst fit algorithm searches the memory blocks for the largest hole and allocates it to the process.

Step 7: Analyses all the three memory management techniques and display the best algorithm which utilizes the memory resources effectively and efficiently.

Step 8: Stop the program.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
main()
{
 int n,i,j,b[20],sb[20],t[20],x,c[20][20];
 clrscr();
 printf("Enter no.of files:");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
		printf("Enter no. of blocks occupied by file%d",i+1);
		scanf("%d",&b[i]);
```

```
                printf("Enter the starting block of file%d",i+1);
                scanf("%d",&sb[i]);
                t[i]=sb[i];
                for(j=0;j<b[i];j++)
                      c[i][j]=sb[i]++;
                }
 printf("Filename\tStart block\tlength\n");
 for(i=0;i<n;i++)
                printf("%d\t  %d \t%d\n",i+1,t[i],b[i]);
 printf("Enter file name:");
 scanf("%d",&x);
 printf("File name is:%d",x);
 printf("length is:%d",b[x-1]);
 printf("blocks occupied:");
 for(i=0;i<b[x-1];i++)
                printf("%4d",c[x-1][i]);
 getch();
}
```

OUTPUT:

```
Enter no.of files: 2
Enter no. of blocks occupied by file1 4
Enter the starting block of file1 2
 Enter no. of blocks occupied by file2 10
Enter the starting block of file2 5
Filename       Start block     length
 1                  2               4
2                   5               10
Enter file name: rajesh
 File name is:12803 length is:0blocks occupied.
```

RESULT:

Thus the program for Sequential File Allocation method was executed and verified successfully.

Ex. No.: 4b

INDEXED FILE ALLOCATION

AIM:

Write a C Program to implement Indexed File Allocation method.

ALGORITHM:

Step 1: Start.

Step 2: Let n be the size of the buffer

Step 3: check if there are any producer

Step 4: if yes check whether the buffer is full

Step 5: If no the producer item is stored in the buffer

Step 6: If the buffer is full the producer has to wait

Step 7: Check there is any cosumer. If yes check whether the buffer is empty

Step 8: If no the consumer consumes them from the buffer

Step 9: If the buffer is empty, the consumer has to wait.

Step 10: Repeat checking for the producer and consumer till required

Step 11: Terminate the process.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
main()
{
 int n,m[20],i,j,sb[20],s[20],b[20][20],x;
 clrscr();
 printf("Enter no. of files:");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {              printf("Enter starting block and size of file%d:",i+1);
                scanf("%d%d",&sb[i],&s[i]);
                printf("Enter blocks occupied by file%d:",i+1);
                scanf("%d",&m[i]);
                printf("enter blocks of file%d:",i+1);
                for(j=0;j<m[i];j++)
                     scanf("%d",&b[i][j]);
 } printf("\nFile\t index\tlength\n");
 for(i=0;i<n;i++)
 {              printf("%d\t%d\t%d\n",i+1,sb[i],m[i]);
 }printf("\nEnter file name:");
 scanf("%d",&x);
 printf("file name is:%d\n",x);
 i=x-1;
 printf("Index is:%d",sb[i]);
 printf("Block occupied are:");
 for(j=0;j<m[i];j++)
                printf("%3d",b[i][j]);
 getch();
}
```

OUTPUT:

```
Enter no. of files:2
 Enter starting block and size of file1: 2   5
Enter blocks occupied by file1:10
enter blocks of file1:3
2  5   4  6  7   2   6  4  7
Enter starting block and size of file2: 3   4
Enter blocks occupied by file2:5
enter blocks of file2: 2  3   4  5   6
File     index  length
 1       2       10
2       3       5
Enter file name: venkat
file name is:12803
Index is:0 Block occupied are:
```

RESULT:

Thus the program for Indexed File Allocation method was executed and verified successfully.

Ex. No.: 4c

LINKED FILE ALLOCATION

AIM:

Write a C Program to implement Linked File Allocation method.

ALGORITHM:

Step 1: Create a queue to hold all pages in memory

Step 2: When the page is required replace the page at the head of the queue

Step 3: Now the new page is inserted at the tail of the queue

Step 4: Create a stack

Step 5: When the page fault occurs replace page present at the bottom of the stack

Step 6: Stop the allocation.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
struct file
{
 char fname[10];
 int start,size,block[10];
}f[10];
main()
{
 int i,j,n;
 clrscr();
 printf("Enter no. of files:");
 scanf("%d",&n);
 for(i=0;i<n;i++)
  {
 printf("Enter file name:");
 scanf("%s",&f[i].fname);
 printf("Enter starting block:");
 scanf("%d",&f[i].start);
 f[i].block[0]=f[i].start;
 printf("Enter no.of blocks:");
 scanf("%d",&f[i].size);
 printf("Enter block numbers:");
 for(j=1;j<=f[i].size;j++)
 {            scanf("%d",&f[i].block[j]);
 }
 }
 printf("File\tstart\tsize\tblock\n");
 for(i=0;i<n;i++)
 {                printf("%s\t%d\t%d\t",f[i].fname,f[i].start,f[i].size);
              for(j=1;j<=f[i].size-1;j++)
                  printf("%d--->",f[i].block[j]);
              printf("%d",f[i].block[j]);
              printf("\n");
 }
 getch();
}
```

OUTPUT:

```
Enter no. of files:2
Enter file name:venkat
Enter starting block:20
Enter no.of blocks:6
Enter block numbers: 4
12
15
45
32
```

```
25
Enter file name:rajesh
Enter starting block:12
Enter no.of blocks:5
Enter block numbers:6
5
4
3
2
File    start   size    block
venkat  20      6       4--->12--->15--->45--->32--->25
rajesh  12      5       6--->5--->4--->3--->2
```

RESULT:

Thus the program for Linked File Allocation method was executed and verified successfully

Ex. No.: 5

SIMULATE MVT AND MFT

MVT:

Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int m=0,m1=0,m2=0,p,count=0,i;
clrscr();
printf("Enter the memory capacity:");
scanf("%d",&m);
printf("Enter the no of processes:");
scanf("%d",&p);
for(i=0;i<p;i++)
{
printf("\nEnter memory req for process%d: ",i+1);
scanf("%d",&m1);
count=count+m1;
if(m1<=m)
{
if(count==m)
{
printf("There is no further memory remaining:");
}
else
{printf("The memory allocated for process%d is: %d ",i+1,m);
m2=m-m1;
printf("\nRemaining memory is: %d",m2);
m=m2;
}           }
else
{
printf("Memory is not allocated for process%d",i+1);          }
printf("\nExternal fragmentation for this process is:%d",m2);
}               getch();
}
```

OUTPUT:

```
Input:
Enter the memory capacity: 80
Enter no of processes: 2
Enter memory req for process1: 23
Output:
The memory allocated for process1 is: 80
Remaining memory is: 57
External fragmentation for this process is: 57
Enter memory req for process2: 52
The memory allocated for process2 is: 57
Remaining memory is: 5
External fragmentation for this process is: 5
```

MFT:

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void main()
{int m,p,s,p1;
int m1[4],i,f,f1=0,f2=0,fra1,fra2,s1;   clrscr();
printf("Enter the memory size:");
scanf("%d",&m);
printf("Enter the no of partitions:");
scanf("%d",&p);
s=m/p;
printf("Each partn size is:%d",s);
printf("\nEnter the no of processes:");
scanf("%d",&p1);
for(i=0;i<p1;i++)
{printf("\nEnter the memory req for process%d:",i+1);
scanf("%d",&m1[i]);
if(m1[i]<=s)    {printf("\nProcess is allocated in partition%d",i+1);
fra1=s-m1[i];
printf("\nInternal fragmentation for process is:%d",fra1);
f1=f1+fra1;
}else
{printf("\nProcess not allocated in partition%d",i+1);
s1=m1[i]-s;                 fra2=s-s1;
f2=f2+fra2;
printf("\nExternal fragmentation for partition is:%d",fra2);
}               }
printf("\nProcess\tmemory\tallocatedmemory");
for(i=0;i<p1;i++)
printf("\n%5d\t%5d\t%5d",i+1,s,m1[i]);
f=f1+f2;
printf("\nThe tot no of fragmentation is:%d",f);        getch();
}
```

Output:

```
Input:          Enter the memory size: 80
Enter the no of partitions: 4
Each partition size: 20
Enter the number of processes: 2
Enter the memory req for process1: 18
Output:    Process1 is allocated in partn1
Internal fragmentation for process1 is: 2
Enter the memory req for process2: 22
Process2 is not allocated in partn2
External fragmentation for process2 is: 18
Process memory allocated
1 20 18
2 20 22
The tot no of fragmentation is: 20
```

Result:

Thus the program was executed successfully.

Ex. No.: 6a

DEAD LOCK AVOIDANCE

AIM

To write a C program to implement bankers algorithm for dead lock avoidance

ALGORITHM:

Step 1: Start the Program

Step 2: Obtain the required data through char and int data types.

Step 3: Enter the filename, index block.

Step 4: Print the file name index loop.

Step 5: File is allocated to the unused index blocks

Step 6: This is allocated to the unused linked allocation.

Step 7: Stop the execution

PROGRAM

```
#include <stdio.h>
int curr[5][5], maxclaim[5][5], avl[5];
int alloc[5] = {0, 0, 0, 0, 0};
int maxres[5], running[5], safe=0;
int count = 0, i, j, exec, r, p, k = 1;
int main(void)
{
   printf("\nEnter the number of processes: ");
   scanf("%d", &p);
   for (i = 0; i < p; i++) {
      running[i] = 1;
      count++;
   }
   printf("\nEnter the number of resources: ");
   scanf("%d", &r);
```

```
printf("\nEnter Claim Vector:");
for (i = 0; i < r; i++) {
    scanf("%d", &maxres[i]);
}
printf("\nEnter Allocated Resource Table:\n");
for (i = 0; i < p; i++) {
    for(j = 0; j < r; j++) {
        scanf("%d", &curr[i][j]);
    }
}
printf("\nEnter Maximum Claim Table:\n");
for (i = 0; i < p; i++) {
    for(j = 0; j < r; j++) {
        scanf("%d", &maxclaim[i][j]);
    }
}
printf("\nThe Claim Vector is: ");
for (i = 0; i < r; i++) {
    printf("\t%d", maxres[i]);
}
printf("\nThe Allocated Resource Table:\n");
for (i = 0; i < p; i++) {
    for (j = 0; j < r; j++) {
        printf("\t%d", curr[i][j]);
    }
    printf("\n");
}
printf("\nThe Maximum Claim Table:\n");
for (i = 0; i < p; i++) {
    for (j = 0; j < r; j++) {
        printf("\t%d", maxclaim[i][j]);
    }
    printf("\n");
}
for (i = 0; i < p; i++) {
    for (j = 0; j < r; j++) {
        alloc[j] += curr[i][j];
    }
}
printf("\nAllocated resources:");
for (i = 0; i < r; i++) {
    printf("\t%d", alloc[i]);
}
for (i = 0; i < r; i++) {
    avl[i] = maxres[i] - alloc[i];
}
printf("\nAvailable resources:");
for (i = 0; i < r; i++) {
    printf("\t%d", avl[i]);
}
printf("\n");
//Main procedure goes below to check for unsafe state.
while (count != 0) {
    safe = 0;
    for (i = 0; i < p; i++) {
        if (running[i]) {
            exec = 1;
            for (j = 0; j < r; j++) {
                if (maxclaim[i][j] - curr[i][j] > avl[j]) {
                    exec = 0;
                    break;
                }
                }
                if (exec) {
                    printf("\nProcess%d is executing\n", i + 1);
                    running[i] = 0;
                    count--;
                    safe = 1;
                    for (j = 0; j < r; j++) {
                        avl[j] += curr[i][j];
                    }
                    break;
                }
            }
        }
        if (!safe) {
            printf("\nThe processes are in unsafe state.\n");
            break;
        } else {
            printf("\nThe process is in safe state");
            printf("\nAvailable vector:");
            for (i = 0; i < r; i++) {
                printf("\t%d", avl[i]);
            }
            printf("\n");
        }
    }
    return 0;
}
```

OUTPUT

```
----------------
Enter the number of processes:5
Enter the number of resources:4
Enter Claim Vector:8 5 9 7
Enter Allocated Resource Table:
4 0 0 3
2 0 1 1
0 1 2 1
0 2 1 0
1 0 3 0
Enter Maximum Claim Table:
5 1 0 5
3 2 1 4
0 2 5 2
1 5 3 0
3 0 3 3
The Claim Vector is:   8       5       9       7
The Allocated Resource Table:
            4   0       0       3
            2   0       1       1
            0   1       2       1
            0   2       1       0
            1   0       3       0
The  Maximum Claim Table:
            5   1       0       5
            3   2       1       4
            0   2       5       2
            1   5       3       0
            3   0       3       3
 Allocated resources:  7       3       7       5
 Available resources:  1      2      2      2
Process1 is executing
 The process is in safe state
 Available vector:     5      2      2      5
Process2 is executing
 The process is in safe state
 Available vector:     7      2      3      6
Process3 is executing
 The process is in safe state
 Available vector:     7      3      5      7
Process4 is executing
 The process is in safe state
 Available vector:     7      5      6      7
Process5 is executing
 The process is in safe state
 Available vector:     8      5      9      7
```

RESULT

Thus the program was executed successfully.

Ex. No.: 7

DEADLOCK DETECTION

AIM

To write a C program to implement Deadlock Detection algorithm

ALGORITHM:

Step 1: Start the Program

Step 2: Obtain the required data through char and in data types.

Step 3: Enter the filename, index block.

Step 4: Print the file name index loop.

Step 5: File is allocated to the unused index blocks.

Step 6: This is allocated to the unused linked allocation.

Step 7: Stop the execution

PROGRAM

```
 #include <stdio.h>
#include <conio.h>
void main()
{
int found,flag,l,p[4][5],tp,tr,c[4][5],i,j,k=1,m[5],r[5],a[5],temp[5],sum=0; clrscr();
printf("Enter total no of processes"); scanf("%d",&tp);
printf("Enter total no of resources"); scanf("%d",&tr);
printf("Enter claim (Max. Need) matrix\n"); for(i=1;i<=tp;i++)
{
printf("process %d:\n",i); for(j=1;j<=tr;j++) scanf("%d",&c[i][j]);
}
printf("Enter allocation matrix\n"); for(i=1;i<=tp;i++)
{
printf("process %d:\n",i); for(j=1;j<=tr;j++) scanf("%d",&p[i][j]);
}
```

```
printf("Enter resource vector (Total resources):\n"); for(i=1;i<=tr;i++)
{
scanf("%d",&r[i]);
}
printf("Enter availability vector (available resources):\n"); for(i=1;i<=tr;i++)
{
scanf("%d",&a[i]);
temp[i]=a[i];
}
for(i=1;i<=tp;i++)
{
sum=0;
for(j=1;j<=tr;j++)
{
sum+=p[i][j];
}
if(sum==0)
{
m[k]=i;
k++;
}
}
for(i=1;i<=tp;i++)
{
for(l=1;l<k;l++)
if(i!=m[l])
{
flag=1;
for(j=1;j<=tr;j++)
if(c[i][j]<temp[j])
{
```

```
flag=0;
break;
}
}
if(flag==1)
{
m[k]=i;
k++;
for(j=1;j<=tr;j++)
temp[j]+=p[i][j];
}
}
printf("deadlock causing processes are:"); for(j=1;j<=tp;j++)
{
found=0;
for(i=1;i<k;i++)
{
if(j==m[i])
found=1;
}
if(found==0)
printf("%d\t",j);
}
getch();
}
```

OUTPUT:

```
Enter total no. of processes : 4
Enter total no. of resources : 5
Enter claim (Max. Need) matrix : 0 1 0 0 1 0 0 1 0 1 0 0 0 0 1 1 0 1 0 1
Enter allocation matrix : 1 0 1 1 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0
Enter resource vector (Total resources) : 2 1 1 2 1
Enter availability vector (available resources) : 0 0 0 0 1
deadlock causing processes are : 2 3
```

RESULT:

Thus the program was executed successfully.

Ex. No.: 8a

FIFO

AIM:

To write a c program to implement FIFO (First In First Out) page replacement algorithm

ALGORITHM:

Step 1: Start the process

Step 2: Declare the size with respect to page length

Step 3: Check the need of replacement from the page to memory

Step 4: Check the need of replacement from old page to new page in memory

Step 5: Forma queue to hold all pages

Step 6: Insert the page require memory into the queue

Step 7: Check for bad replacement and page fault

Step 8: Get the number of processes to be inserted

Step 9: Display the values

Step 10: Stop the process

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;
void main()
{
 clrscr();
 printf("\n \t\t\t FIFI PAGE REPLACEMENT ALGORITHM");
 printf("\n Enter no.of frames....");
 scanf("%d",&nof);
 printf("Enter number of reference string..\n");
 scanf("%d",&nor);
 printf("\n Enter the reference string..");
 for(i=0;i<nor;i++)
 scanf("%d",&ref[i]);
 printf("\nThe given reference string:");
 for(i=0;i<nor;i++)
 printf("%4d",ref[i]);
 for(i=1;i<=nof;i++)
 frm[i]=-1;
 printf("\n");
 for(i=0;i<nor;i++)
 {
  flag=0;
  printf("\n\t Reference np%d->\t",ref[i]);
  for(j=0;j<nof;j++)
  {
   if(frm[j]==ref[i])
   {
    flag=1;
    break;
   }}
    if(flag==0)
  {
   pf++;
   victim++;
   victim=victim%nof;
   frm[victim]=ref[i];
   for(j=0;j<nof;j++)
   printf("%4d",frm[j]);
  }
 }
 printf("\n\n\t\t No.of pages faults...%d",pf);
 getch();
}
```

OUTPUT:

```
                    FIFO PAGE REPLACEMENT ALGORITHM
Enter no.of frames....4
Enter number of reference string..
6
 Enter the reference string..
5 6 4 1 2 3
The given reference string:
......................................  5   6   4   1   2   3
        Reference np5->          5  -1  -1  -1
        Reference np6->          5   6  -1  -1
        Reference np4->          5   6   4  -1
        Reference np1->          5   6   4   1
        Reference np2->          2   6   4   1
        Reference np3->          2   3   4   1

              No.of pages faults...6
```

RESULT:

Thus the page replacement program was executed successfully.

Ex. No.: 8b

LRU

AIM:

To write a c program to implement LRU (Least Recently Used) page replacement algorithm

ALGORITHM:

Step 1: Start the process

Step 2: Declare the size

Step 3: Get the number of pages to be inserted

Step 4: Get the value

Step 5: Declare counter and stack

Step 6: Select the least recently used page by counter value

Step 7: Stack them according the selection.

Step 8: Display the values

Step 9: Stop the process

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;
int recent[10],lrucal[50],count=0;
int lruvictim();
void main()
{
 clrscr();
 printf("\n\t\t\t LRU PAGE REPLACEMENT ALGORITHM");
 printf("\n Enter no.of Frames....");
 scanf("%d",&nof);
 printf(" Enter no.of reference string..");
 scanf("%d",&nor);
 printf("\n Enter reference string..");
 for(i=0;i<nor;i++)
 scanf("%d",&ref[i]);
```

```
printf("\n\n\t\t LRU PAGE REPLACEMENT ALGORITHM ");
printf("\n\t The given reference string:");
printf("\n……………………………….. ");
for(i=0;i<nor;i++)
printf("%4d",ref[i]);
for(i=1;i<=nof;i++)
{
  frm[i]=-1;
  lrucal[i]=0;
}
for(i=0;i<10;i++)
 recent[i]=0;
 printf("\n");
 for(i=0;i<nor;i++)
 {
   flag=0;
   printf("\n\t Reference NO %d->\t",ref[i]);
   for(j=0;j<nof;j++)
   {
    if(frm[j]==ref[i])
     {
                 flag=1;
                 break;
     }
   }
   if(flag==0)
   {
     count++;
     if(count<=nof)
     victim++;
     else
```

```
       victim=lruvictim();
       pf++;
       frm[victim]=ref[i];
       for(j=0;j<nof;j++)
       printf("%4d",frm[j]);
     }
     recent[ref[i]]=i;
   }
   printf("\n\n\t No.of page faults...%d",pf);
   getch();
}
int lruvictim()
{
  int i,j,temp1,temp2;
  for(i=0;i<nof;i++)
  {
    temp1=frm[i];
    lrucal[i]=recent[temp1];
  }
  temp2=lrucal[0];
  for(j=1;j<nof;j++)
  {
    if(temp2>lrucal[j])
    temp2=lrucal[j];
  }
  for(i=0;i<nof;i++)
  if(ref[temp2]==frm[i])
  return i;
  return 0;
}
```

OUTPUT:

```
                    LRU PAGE REPLACEMENT ALGORITHM
Enter no.of Frames....3
 Enter no.of reference string..6
 Enter reference string..6 5 4 2 3 1
               LRU PAGE REPLACEMENT ALGORITHM
        The given reference string:
         …………………. 6   5   4   2   3   1

        Reference NO 6->        6  -1  -1
        Reference NO 5->        6   5  -1
        Reference NO 4->        6   5   4
        Reference NO 2->        2   5   4
        Reference NO 3->        2   3   4
        Reference NO 1->        2   3   1
        No.of page faults...6
```

RESULT:

Thus the LRU page replacement program was executed successfully.

Ex. No.: 8c

LFU

AIM:

To write a c program to implement LFU page replacement algorithm

ALGORITHM:

Step 1: Start the process

Step 2: Declare the size

Step 3: Get the number of pages to be inserted

Step 4: Get the value

Step 5: Declare counter and stack

Step 6: Select the least frequently used page by counter value

Step 7: Stack them according the selection.

Step 8: Display the values

Step 9: Stop the process

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;
int recent[10],optcal[50],count=0;
int optvictim();
void main()
{
  clrscr();
  printf("\n OPTIMAL PAGE REPLACEMENT ALGORITHN");
  printf("\n................................");
  printf("\nEnter the no.of frames");
  scanf("%d",&nof);
  printf("Enter the no.of reference string");
  scanf("%d",&nor);
  printf("Enter the reference string");
  for(i=0;i<nor;i++)
```

```
    scanf("%d",&ref[i]);
clrscr();
printf("\n OPTIMAL PAGE REPLACEMENT ALGORITHM");
printf("\n...............................");
printf("\nThe given string");
printf("\n...................\n");
for(i=0;i<nor;i++)
    printf("%4d",ref[i]);
for(i=0;i<nof;i++)
{
    frm[i]=-1;
    optcal[i]=0;
}
for(i=0;i<10;i++)
    recent[i]=0;
printf("\n");
for(i=0;i<nor;i++)
{
   flag=0;
   printf("\n\tref no %d ->\t",ref[i]);
   for(j=0;j<nof;j++)
   {
               if(frm[j]==ref[i])
               {
                  flag=1;
                  break;
               }
   }
   if(flag==0)
   {
               count++;
```

```
                if(count<=nof)
                    victim++;
                else
                    victim=optvictim(i);
                pf++;
                frm[victim]=ref[i];
                for(j=0;j<nof;j++)
                   printf("%4d",frm[j]);
      }
   }
   printf("\n Number of page faults: %d",pf);
   getch();
}
int optvictim(int index)
{
   int i,j,temp,notfound;
   for(i=0;i<nof;i++)
   {
      notfound=1;
      for(j=index;j<nor;j++)
                if(frm[i]==ref[j])
                {
                    notfound=0;
                    optcal[i]=j;
                    break;
                }
      if(notfound==1)
                   return i;
   }
   temp=optcal[0];
   for(i=1;i<nof;i++)
```

```
    if(temp<optcal[i])
            temp=optcal[i];
  for(i=0;i<nof;i++)
    if(frm[temp]==frm[i])
            return i;
 return 0;
}
```

OUTPUT:

```
   OPTIMAL PAGE REPLACEMENT ALGORITHM
Enter no.of Frames....3
 Enter no.of reference string..6
 Enter reference string..6 5 4 2 3 1
          OPTIMAL PAGE REPLACEMENT ALGORITHM
     The given reference string:
     …………………. 6   5   4   2   3   1
     Reference NO 6->        6  -1  -1
     Reference NO 5->        6   5  -1
     Reference NO 4->        6   5   4
     Reference NO 2->        2   5   4
     Reference NO 3->        2   3   4
     Reference NO 1->        2   3   1
     No.of page faults...6
```

RESULT:

Thus the LFU page replacement program was executed successfully.

Ex. No.: 9a

SINGLE LEVEL DIRECTORY

AIM

To write a C program to implement File Organization concept using the technique Single level directory.

ALGORITHM:

Step 1: Start the Program

Step 2: Obtain the required data through char and int datatypes.

Step 3: Enter the filename, index block.

Step 4: Print the file name index loop.

Step 5: Fill is allocated to the unused index blocks

Step 6: This is allocated to the unused linked allocation.

Step 7: Stop the execution

PROGRAM

```
#include<stdio.h>
struct
{
char dname[10],fname[10][10];
int fcnt;
}dir;
void main()
{int i,ch;
char f[30];
clrscr();
dir.fcnt = 0;
printf("\nEnter name of directory -- ");
scanf("%s", dir.dname);
while(1)
{printf("\n\n1. Create File\t2. Delete File\t3. Search File \n
4. Display Files\t5. Exit\nEnter your choice -- ");
```

```
scanf("%d",&ch);
switch(ch)
{case 1: printf("\nEnter the name of the file -- ");
scanf("%s",dir.fname[dir.fcnt]);
dir.fcnt++;
break;
case 2: printf("\nEnter the name of the file -- ");
scanf("%s",f);
for(i=0;i<dir.fcnt;i++)
{if(strcmp(f, dir.fname[i])==0)
{               printf("File %s is deleted ",f);
strcpy(dir.fname[i],dir.fname[dir.fcnt-1]);
break;
}               }
if(i==dir.fcnt)
printf("File %s not found",f);
else
dir.fcnt--;
break;
case 3: printf("\nEnter the name of the file -- ");
scanf("%s",f);
for(i=0;i<dir.fcnt;i++)
{
if(strcmp(f, dir.fname[i])==0)
{
printf("File %s is found ", f);
break;
}               }
if(i==dir.fcnt)
printf("File %s not found",f);
break;
case 4: if(dir.fcnt==0)
printf("\nDirectory Empty");
else
{printf("\nThe Files are -- ");
for(i=0;i<dir.fcnt;i++)
printf("\t%s",dir.fname[i]);
}
break;
default: exit(0);
}               }
getch();
}
```

OUTPUT:

```
Enter name of directory -- CSE
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 1
Enter the name of the file -- A
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 1
Enter the name of the file -- B
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 1
Enter the name of the file -- C
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 4
The Files are -- A B C
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 3
Enter the name of the file – ABC
File ABC not found
```

```
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 2
Enter the name of the file – B
File B is deleted
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 5
```

RESULT:

Thus the program was executed successfully.

Ex. No.: 9b

TWO LEVEL DIRECTORY

AIM

To write a C program to implement File Organization concept using the technique two level directory.

ALGORITHM:

Step 1: Start the Program

Step 2: Obtain the required data through char and in datatypes.

Step 3: Enter the filename, index block.

Step 4: Print the file name index loop.

Step 5: File is allocated to the unused index blocks

Step 6: This is allocated to the unused linked allocation.

Step 7: Stop the execution

PROGRAM

```
#include<stdio.h>
struct
{
char dname[10],fname[10][10];
int fcnt;
}dir[10];
void main()
{
int i,ch,dcnt,k;
char f[30], d[30];
clrscr();
dcnt=0;
while(1)
{
printf("\n\n1. Create Directory\t2. Create File\t3. Delete File");
printf("\n4. Search File\t\t5. Display\t6. Exit\t
Enter your choice -- ");
scanf("%d",&ch);
switch(ch)
{
case 1:
 printf("\nEnter name of directory -- ");
scanf("%s", dir[dcnt].dname);
dir[dcnt].fcnt=0;
dcnt++;
printf("Directory created");
break;
case 2:
printf("\nEnter name of the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
if(strcmp(d,dir[i].dname)==0)
{
printf("Enter name of the file -- ");
scanf("%s",dir[i].fname[dir[i].fcnt]);
dir[i].fcnt++;
printf("File created");
break;
}
if(i==dcnt)
printf("Directory %s not found",d);
break;
case 3:
 printf("\nEnter name of the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
```

```
{
if(strcmp(d,dir[i].dname)==0)
{
printf("Enter name of the file -- ");
scanf("%s",f);
for(k=0;k<dir[i].fcnt;k++)
{
if(strcmp(f, dir[i].fname[k])==0)
{
printf("File %s is deleted ",f);
dir[i].fcnt--;
strcpy(dir[i].fname[k],dir[i].fname[dir[i].fcnt]);
goto jmp;
}
}
printf("File %s not found",f);
goto jmp;
}
}
printf("Directory %s not found",d);
jmp : break;
case 4:
printf("\nEnter name of the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
{
if(strcmp(d,dir[i].dname)==0)
{
printf("Enter the name of the file -- ");
scanf("%s",f);
for(k=0;k<dir[i].fcnt;k++)
{
if(strcmp(f, dir[i].fname[k])==0)
{
printf("File %s is found ",f);
goto jmp1;
}
}
printf("File %s not found",f);
goto jmp1;
}
}
printf("Directory %s not found",d);
jmp1: break;
case 5:
 if(dcnt==0)
printf("\nNo Directory's ");
else
{
printf("\nDirectory\tFiles");
for(i=0;i<dcnt;i++)
{
printf("\n%s\t\t",dir[i].dname);
for(k=0;k<dir[i].fcnt;k++)
printf("\t%s",dir[i].fname[k]);
}
}
break;
default:exit(0);
}               }
getch();
}
```

OUTPUT:

```
1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice -- 1
Enter name of directory -- DIR1
Directory created
1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice -- 1
Enter name of directory -- DIR2
Directory created
1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice -- 2
Enter name of the directory – DIR1
Enter name of the file -- A1
File created
1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice -- 2
Enter name of the directory – DIR1
Enter name of the file -- A2
File created
1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice -- 2
Enter name of the directory – DIR2
Enter name of the file -- B1
File created
1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice -- 5
Directory Files
DIR1 A1 A2
DIR2 B1
1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice -- 4
```

```
Enter name of the directory – DIR
Directory not found
1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice -- 3
Enter name of the directory – DIR1
Enter name of the file -- A2
File A2 is deleted
1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice -- 6
```

RESULT:

Thus the program was executed successfully.

Ex. No.: 9c

HIERARCHICAL LEVEL DIRECTORY

AIM

To write a C program to implement File Organization concept using the technique hierarchical level directory.

ALGORITHM:

1 Start the Program

2 Obtain the required data through char and int datatypes.

3 Enter the filename,index block.

4 Print the file name index loop.

5 Fill is allocated to the unused index blocks

6 This is allocated to the unused linked allocation.

7 Stop the execution

PROGRAM

```
#include<stdio.h>
#include<graphics.h>
struct tree_element
{
char name[20];
int x, y, ftype, lx, rx, nc, level;
struct tree_element *link[5];
};
typedef struct tree_element node;
void main()
{
int gd=DETECT,gm;
node *root;
root=NULL;
clrscr();
```

```
create(&root,0,"root",0,639,320);
clrscr();
initgraph(&gd,&gm,"c:\tc\BGI");
display(root);
getch();
closegraph();
}
create(node **root,int lev,char *dname,int lx,int rx,int x)
{
int i, gap;
if(*root==NULL)
{
(*root)=(node *)malloc(sizeof(node));
printf("Enter name of dir/file(under %s) : ",dname);
fflush(stdin);
gets((*root)->name);
printf("enter 1 for Dir/2 for file :");
scanf("%d",&(*root)->ftype);
(*root)->level=lev;
(*root)->y=50+lev*50;
(*root)->x=x;
(*root)->lx=lx;
(*root)->rx=rx;
for(i=0;i<5;i++)
(*root)->link[i]=NULL;
if((*root)->ftype==1)
{
printf("No of sub directories/files(for %s):",(*root)->name); scanf("%d",&(*root)>nc);
if((*root)->nc==0)
gap=rx-lx;
else
gap=(rx-lx)/(*root)->nc;
for(i=0;i<(*root)->nc;i++)
create(&((*root)>link[i]),lev+1,(*root)>name,lx+gap*i,lx+gap*i+gap,
lx+gap*i+gap/2);
}
else
(*root)->nc=0;
}
}
display(node *root)
{
int i;
settextstyle(2,0,4);
settextjustify(1,1);
setfillstyle(1,BLUE);
setcolor(14);
if(root !=NULL)
{
for(i=0;i<root->nc;i++)
line(root->x,root->y,root->link[i]->x,root->link[i]->y);
if(root->ftype==1)
bar3d(root->x-20,root->y-10,root->x+20,root>y+10,0,0);
else
fillellipse(root->x,root->y,20,20);
outtextxy(root->x,root->y,root->name);
for(i=0;i<root->nc;i++)
display(root->link[i]);
}
}
```

INPUT

```
Enter Name of dir/file(under root): ROOT
Enter 1 for Dir/2 for File: 1
No of subdirectories/files(for ROOT): 2
Enter Name of dir/file(under ROOT): USER1
Enter 1 for Dir/2 for File: 1
No of subdirectories/files(for USER1): 1
Enter Name of dir/file(under USER1): SUBDIR1
Enter 1 for Dir/2 for File: 1
No of subdirectories/files(for SUBDIR1): 2
Enter Name of dir/file(under USER1): JAVA
Enter 1 for Dir/2 for File: 1
No of subdirectories/files(for JAVA): 0
Enter Name of dir/file(under SUBDIR1): VB
Enter 1 for Dir/2 for File: 1
No of subdirectories/files(for VB): 0
Enter Name of dir/file(under ROOT): USER2
Enter 1 for Dir/2 for File: 1
No of subdirectories/files(for USER2): 2
Enter Name of dir/file(under ROOT): A
Enter 1 for Dir/2 for File: 2
Enter Name of dir/file(under USER2): SUBDIR2
Enter 1 for Dir/2 for File: 1
No of subdirectories/files(for SUBDIR2): 2
Enter Name of dir/file(under SUBDIR2): PPL
Enter 1 for Dir/2 for File: 1
No of subdirectories/files(for PPL): 2
Enter Name of dir/file(under PPL): B
Enter 1 for Dir/2 for File: 2
Enter Name of dir/file(under PPL): C
Enter 1 for Dir/2 for File: 2
Enter Name of dir/file(under SUBDIR): AI
Enter 1 for Dir/2 for File: 1
No of subdirectories/files(for AI): 2
Enter Name of dir/file(under AI): D
Enter 1 for Dir/2 for File: 2
Enter Name of dir/file(under AI): E
Enter 1 for Dir/2 for File: 2
```

OUTPUT

```
       ROOT
USER1          USER2
SUBDIR1       A     SUBDIR2
JAVA VB                  PPL        AI
                         B C        D E
```

RESULT:

Thus the program was executed successfully.

Ex. No.: 10

PAGING TECHNIQUE OF MEMORY MANAGEMEN

AIM:

To write a C program to implement the concept of Paging

ALGORITHM:

```
Step 1 The Semaphore mutex, full & empty are initialized.
Step 2 In the case of producer process
        i) Produce an item in to temporary variable.
        ii) If there is empty space in the buffer check the mutex value for enters into the
        critical section.
        iii) If the mutex value is 0, allow the producer to add value in the temporary variable
        to the buffer.
Step 3 In the case of consumer process
        i) It should wait if the buffer is empty
        ii) If there is any item in the buffer check for mutex value, if the mutex==0, remove
        item from buffer
        iii) Signal the mutex value and reduce the empty value by 1.
        iv) Consume the item.
Step 4 Print the result
```

PROGRAM

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
main()
{
 int np,ps,i;
 int *sa;
 clrscr();
 printf("Enter how many pages\n");
 scanf("%d",&np);
 printf("Enter the page size \n");
 scanf("%d",&ps);
 for(i=0;i<np;i++)
 { sa[i]=(int)malloc(ps);
 printf("Page%d\t Address %u\n",i+1,sa[i]);
 }
getch();
}
```

INPUT:

```
Enter how many pages: 5
Enter the page size: 4
```

OUTPUT:

```
Page1 Address: 1894
Page2 Address: 1902
Page3 Address: 1910
Page4 Address: 1918
Page5 Address: 1926
```

RESULT:

Thus the program is executed

Ex. No.: 11

SIMULATE SHARED MEMORY AND IPC

AIM:

To write a program for interprocess communication using shared memory.

ALGORITHM:

```
Step 1: Start the program
Step 2: Create the child process using fork()
Step 3: Create the shared memory for parent process using shmget() system call
Step 4: Now allow the parent process to write inn shared memory using shmpet pointer which is
        return type of shmat()
Step 5: Now across and attach the same shared memory to the child process
Step 6: The data in the shared memory is read by the child process using the shnot pointer
Step 7: Now, detach and rebase the shared memory
Step 8: Stop the program
```

PROGRAM:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/shm.h>
#include<sys/ipc.h>
int main()
{
int child,shmid,i;
char * shmptr;
child=fork();
if(!child)
{
shmid=shmget(2041,32,IPC_CREAT|0666);
shmptr=shmat(shmid,0,0);
printf("\n Parent writing\n");
for(i=0;i<10;i++)
{
```

```
            shmptr[i]=’a’+i;
putchar(shmptr[i]);
}
                printf(“\n\n %s”, shmptr);
                wait(NULL);
            }
            else
{
shmid=shmget(2041,32,0666);
            shmptr=shmat(shmid,0,0);
            printf(“\n Child is reading\n”);
for(i=0;i<10;i++)
putchar(shmptr[i]);
            shmdt(NULL);
            shmctl(shmid,IPC_RMID,NULL);
}
return 0;
}
```

OUTPUT:

```
[cse2@localhost ~]$ cc share.c
[cse2@localhost ~]$ ./a.out

Parent writing
abcdefghij
Child is reading
abcdefghij
```

RESULT:

Thus the interprocess communication using shared memory was successfully executed.

Ex. No.: 12

IMPLEMENT THREADING & SYNCHRONIZATION APPLICATIONS

AIM:

To implement Banking system involving Concurrency (Thread)

ALGORITHM:

1. Create a Bank Database

2. Create functions for adding a new user, depositing the amt, withdrawing the amt & viewing the customer

3. Create a thread for each functionality

4. Implement the concurrency control among the threads

5. Function add new user

a. Get the user details namely, Acno, Name & Bank Balance

b. Write the user details to the bank database

6. Function Deposit

a. Get the Acno & Amt

b. Update the Bank balance for the given Acno

7. Function Withdraw

a. Get the Acno & Amt

b. Update the Bank balance for the given Acno

8. Function View

a. Get the Acno

b. Display the account details