DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LAB MANUAL

Academic Year: 2015-16 ODD SEMESTER

Programme (UG/PG)	**:**	**UG**
Semester	**:**	**v**
Course Code	**:**	**CS1035**
Course Title	**:**	**OPERATING SYSTEM LAB**

Prepared By

<B.SOWMIYA>

(<AP/O.G.>, Department of Computer Science and Engineering)

FACULTY OF ENGINEERING AND TECHNOLOGY
SRM UNIVERSITY
(Under section 3 of UGC Act, 1956)
SRM Nagar, Kattankulathur- 603203
Kancheepuram District

LIST OF EXPERIMENTS & SCHEDULE

Course Code: CS1035 Course Title: Operating System Lab

Exp. No.	Title	Week No.
1a,1b	Simulate the following CPU scheduling algorithms a)SJF, b) FCFS	1
1c, 1d	Simulate the following CPU scheduling algorithms c) Round Robin ,d) Priority	2
2a,2b	Simulate MVT AND MFT	3
3a,3b	Simulate all file allocation strategies a) Sequential b) Indexed	4
3c	Simulate all file allocation strategies c) Linked	5
4a,4b	Simulate all File Organization Techniques a) Single level directory b) Two level	6
4c	Simulate all File Organization Techniques c) Hierarchical	7
5	Simulate Bankers Algorithm for Dead Lock Avoidance	8
6	Simulate an Algorithm for Dead Lock Detection	9
7a,7b	Simulate all page replacement algorithms a) FIFO b) LRU	10
8,9	Simulate Shared memory and IPC & Simulate Paging Technique of memory management.	11
10	Implement Threading & Synchronization Applications	12

Course Coordinator **HOD**

HARDWARE AND SOFTWARE REQUIREMENTS

Windows		
	Run-Time Engine	**Development Environment**
Processor	Pentium III/Celeron 866 MHz or equivalent	Pentium 4/M or equivalent
RAM	256 MB	1 GB
Screen Resolution	1024 x 768 pixels	1024 x 768 pixels
OS	Windows 8.1/8/7/Vista (32-bit and 64-bit) Windows XP SP3 (32-bit) Windows Server 2012 R2 (64-bit) Windows Server 2008 R2 (64-bit) Windows Server 2003 R2 (32-bit)	Windows 8.1/8/7/Vista (32-bit and 64-bit) Windows XP SP3 (32-bit) Windows Server 2012 R2 (64-bit) Windows Server 2008 R2 (64-bit) Windows Server 2003 R2 (32-bit)
Disk Space	500 MB	5 GB (includes default drivers from NI Device Drivers DVD)
Mac OS X		
	Run-Time Engine	**Development Environment**
Processor	Intel-based processor	Intel-based processor
RAM	256 MB	2 GB
Screen Resolution	1024 x 768 pixels	1024 x 768 pixels
OS	OS X 10.7, 10.8, or 10.9	OS X 10.8 or 10.9
Disk Space	656 MB - 1.2 GB	1.2 GB for the complete installation (excluding drivers)
Languages	C,c++, java	C,c++,java

Internal Assessment Mark Split Up

Observation	:	25 Marks
Attendance	:	5 Marks
Record	:	10 Marks
Model Exam	:	20 Marks
TOTAL MARKS	:	60 Marks

SCHEDULING: FCFS

Ex. No: 1 a

OBJECTIVE:

A program to simulate the FCFS CPU scheduling algorithm

ALGORITHM:

Step 1: Create the number of process.

Step 2: Get the ID and Service time for each process.

Step 3: Initially, Waiting time of first process is zero and Total time for the first process is the starting time of that process.

Step 4: Calculate the Total time and Processing time for the remaining processes.

Step 5: Waiting time of one process is the Total time of the previous process.

Step 6: Total time of process is calculated by adding Waiting time and Service time.

Step 7: Total waiting time is calculated by adding the waiting time for lack process.

Step 8: Total turn around time is calculated by adding all total time of each process.

Step 9: Calculate Average waiting time by dividing the total waiting time by total number of process.

Step 10: Calculate Average turn around time by dividing the total time by the number of process.

Step 11: Display the result.

SOURCE CODE :

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
main()
{
char pn[10][10],t[10];
int arr[10],bur[10],star[10],finish[10],tat[10],wt[10],i,j,n,temp; int totwt=0,tottat=0;
//clrscr();
printf("Enter the number of processes:"); scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter the Process Name, Arrival Time & Burst Time:");
scanf("%s%d%d",&pn[i],&arr[i],&bur[i]);
}
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
if(arr[i]<arr[j])
{
temp=arr[i];
arr[i]=arr[j];
arr[j]=temp;
temp=bur[i];
bur[i]=bur[j];
bur[j]=temp;
strcpy(t,pn[i]);
strcpy(pn[i],pn[j]);
strcpy(pn[j],t);
}
}}
for(i=0;i<n;i++)
{
if(i==0)
star[i]=arr[i]; elsestar[i]=finish[i-1];
wt[i]=star[i]-arr[i];finish[i]=star[i]+bur[i];tat[i]=finish[i]-arr[i];
}
printf("\nPName Arrtime Burtime WaitTime Start TAT Finish"); for(i=0;i<n;i++)
{
printf("\n%s\t%3d\t%3d\t%3d\t%3d\t%6d\t%6d",pn[i],arr[i],bur[i],wt[i],star[i],tat[i],finis
h[i]);
totwt+=wt[i];
tottat+=tat[i];
}
printf("\nAverage Waiting time:%f",(float)totwt/n); printf("\nAverage Turn Around
Time:%f",(float)tottat/n); getch();
return 0;
    }
```

SAMPLE INPUTS & OUTPUTS:

Input:
Enter the number of processes: 3
Enter the Process Name, Arrival Time & Burst Time: p1 2 4
Enter the Process Name, Arrival Time & Burst Time: p2 3 5
Enter the Process Name, Arrival Time & Burst Time: p3 1 6

Output:

PName	Arrtime	Burtime	WaitTime	Srart	TAT	Finish
p3	1	6	01	6	7	
p1	2	4	57	9	11	
p2	3	5	811	13	16	

Average Waiting Time: 4.3333333
Average Turn Around Time: 9.33333333

RESULT :

Thus the program to stimulate the FCFS scheduling algorithm is successfully executed.

SHORTEST JOB FIRST

Ex.No.1.b

OBJECTIVE:

A program to simulate the SJF CPU scheduling algorithm

ALGORITHM:

Step 1: Declare the array size.

Step 2: Get the number of elements to be inserted.

Step 3: Select the process which have shortest burst will execute first.

Step 4: If two process have same burst length then FCFS scheduling algorithm used.

Step 5: Make the average waiting the length of next process.

Step 6: Start with the first process from it's selection as above and let other process to be in queue.

Step 7: Calculate the total number of burst time.

Step 8: Display the values.

SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
#include<string.h> void main()
{
int et[20],at[10],n,i,j,temp,st[10],ft[10],wt[10],ta[10]; int totwt=0,totta=0;
float awt,ata;
char pn[10][10],t[10]; clrscr();
printf("Enter the number of process:"); scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter process name, arrival time & execution time:"); flushall();
scanf("%s%d%d",pn[i],&at[i],&et[i]);
}
for(i=0;i<n;i++)
for(j=0;j<n;j++)
{
if(et[i]<et[j])
{
temp=at[i];
at[i]=at[j];
at[j]=temp;
temp=et[i];
et[i]=et[j];
et[j]=temp;
strcpy(t,pn[i]);
strcpy(pn[i],pn[j]);
strcpy(pn[j],t);
}
}
for(i=0;i<n;i++)
{
if(i==0)
st[i]=at[i]; elsest[i]=ft[i-1];
wt[i]=st[i]-at[i];ft[i]=st[i]+et[i];ta[i]=ft[i]-at[i];totwt+=wt[i]; totta+=ta[i];
}
awt=(float)totwt/n;
ata=(float)totta/n;
printf("\nPname\tarrivaltime\texecutiontime\twaitingtime\ttatime");
for(i=0;i<n;i++)
printf("\n%s\t%5d\t\t%5d\t\t%5d\t\t%5d",pn[i],at[i],et[i],wt[i],ta[i]); printf("\nAverage
waiting time is:%f",awt);
printf("\nAverage turnaroundtime is:%f",ata); getch();
}
```

SAMPLE INPUT AND OUTPUT:

Input:

```
Enter the number of processes: 3
Enter the Process Name, Arrival Time & Burst Time: 1 4 6
Enter the Process Name, Arrival Time & Burst Time: 2 5 15
Enter the Process Name, Arrival Time & Burst Time: 3 6 11
```

Output:

```
Pname    arrivaltimeexecutiontime  waitingtime  tatime
1                 46               0            6
3                 611              4            15
2                 515              16           31
Average Waiting Time: 6.6667
Average Turn Around Time: 17.3333
```

RESULT:

Thus the program for implementing SJF scheduling algorithm was written and successfully executed.

PRIORITY SCHEDULING

Ex.No.1.c

OBJECTIVE:

A program to simulate the priority CPU scheduling algorithm

ALGORITHM:

STEP 1: Declare the array size.

STEP 2: Get the number of elements to be inserted.

STEP 3: Get the priority for each process and value

STEP 4: Start with the higher priority process from it's initial position let other process to be queue.

STEP 5: Calculate the total number of burst time.

STEP 6: Display the values

SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
#include<string.h> void main()
{
int et[20],at[10],n,i,j,temp,p[10],st[10],ft[10],wt[10],ta[10]; int totwt=0,totta=0;
float awt,ata;
char pn[10][10],t[10]; clrscr();
printf("Enter the number of process:"); scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter process name,arrivaltime,execution time & priority:"); flushall();
scanf("%s%d%d%d",pn[i],&at[i],&et[i],&p[i]);
}
for(i=0;i<n;i++)
for(j=0;j<n;j++)
{
if(p[i]<p[j])
{
temp=p[i];
p[i]=p[j];
p[j]=temp;
temp=at[i];
at[i]=at[j];
at[j]=temp;
temp=et[i];
et[i]=et[j];
et[j]=temp;
strcpy(t,pn[i]);
strcpy(pn[i],pn[j]);
strcpy(pn[j],t);
}
}
for(i=0;i<n;i++)
{
if(i==0)
{
st[i]=at[i];wt[i]=st[i]-at[i];ft[i]=st[i]+et[i];ta[i]=ft[i]-at[i];
}
else
{
st[i]=ft[i-1];wt[i]=st[i]-at[i];ft[i]=st[i]+et[i];ta[i]=ft[i]-at[i];
}
totwt+=wt[i];
totta+=ta[i];
}
awt=(float)totwt/n;
ata=(float)totta/n;
printf("\nPname\tarrivaltime\texecutiontime\tpriority\twaitingtime\ttatime");
```

```
for(i=0;i<n;i++)
printf("\n%s\t%5d\t\t%5d\t\t%5d\t\t%5d\t\t%5d",pn[i],at[i],et[i],p[i],wt[i],ta[i]);
printf("\nAverage waiting time is:%f",awt);
printf("\nAverage turnaroundtime is:%f",ata); getch();
}
```

SAMPLE INPUT AND OUTPUT:
Input:

Enter the number of processes: 3
Enter the Process Name, Arrival Time, execution time & priority: 1 2 3 1
Enter the Process Name, Arrival Time, execution time & priority: 2 4 5 2
Enter the Process Name, Arrival Time, execution time & priority: 3 5 6 3

Output:

Pname	arrivaltime	executiontime	priority	waitingtimetatime
1	2	3	1	03
2	4	5	2	16
3	5	6	3	511

Average Waiting Time: 2.0000
Average Turn Around Time: 6.6667

RESULT:

Thus the program for implementing Priority scheduling algorithm was written and successfully executed.

ROUND ROBIN

Ex.No.1.d

OBJECTIVE:

A program to simulate the Round Robin CPU scheduling algorithm.

ALGORITHM:

STEP 1: Declare the array size.

STEP 2: Get the number of elements to be inserted.

STEP 3: Get the value.

STEP 4: Set the time sharing system with preemption.

STEP 5: Define quantum is defined from 10 to 100ms.

STEP 6: Declare the queue as a circular.

STEP 7: Make the CPU scheduler goes around the ready queue allocating CPU to each process for the time interval specified.

STEP 8: Make the CPU scheduler picks the first process and sets time to interrupt after quantum expired dispatches the process.

STEP 9: If the process has burst less than the time quantum than the process releases the CPU.

SOURCE CODE:

```
#include<stdio.h>
#include<conio.h> void main()
{
int et[30],ts,n,i,x=0,tot=0; char pn[10][10];
clrscr();
printf("Enter the no of processes:"); scanf("%d",&n);
printf("Enter the time quantum:"); scanf("%d",&ts); for(i=0;i<n;i++)
{
printf("enter process name & estimated time:"); scanf("%s %d",pn[i],&et[i]);
}
printf("The processes are:"); for(i=0;i<n;i++)
printf("process %d: %s\n",i+1,pn[i]); for(i=0;i<n;i++)
tot=tot+et[i];
while(x!=tot)
{
for(i=0;i<n;i++)
{
if(et[i]>ts)
{
x=x+ts;
printf("\n %s -> %d",pn[i],ts);
et[i]=et[i]-ts;
}
else if((et[i]<=ts)&&et[i]!=0)
{
x=x+et[i];
printf("\n %s -> %d",pn[i],et[i]); et[i]=0;}
}
}
printf("\n Total Estimated Time:%d",x); getch();
}
```

SAMPLE INPUT AND OUTPUT:

Input:

Enter the no of processes: 2

Enter the time quantum: 3

Enter the process name & estimated time: p1 12

Enter the process name & estimated time: p2 15

Output:p1 -> 3 p2 -> 3 p1 -> 3 p2 -> 3 p1 -> 3 p2 -> 3 p1 -> 3 p2 -> 3 p2 -> 3

Total Estimated Time: 27

RESULT:

Thus the program for implementing Round Robin scheduling algorithm was written and successfully executed.

1. **What is fragmentation?**
2. **What is internal fragmentation?**
3. **What is external fragmentation?**
4. **There are four processes stored in memory with the sizes p1= 20kb, p2=30kb, p3=65kb, p4=5okb. The total memory size is 200kb. Divide the total memory space into four partitions.**
    i) **Calculate internal & external fragmentation using MFT?**
    ii) **Calculate external fragmentation using MVT?**

(These Questions should be dictated to students at the end of the lab hours. The students must write the answers for those questions in their Observation Notebooks before they come to next lab.)

MUTIPROGRAMMING VARIABLE TASK

Ex.No. 2a.

OBJECTIVE:

To program MVT using c programming

ALGORITHM:

Step 1: Declare the necessary variables.

Step 2: Get the memory capacity and number of processes.

Step 3: Get the memory required for each process.

Step 4: If the needed memory is available for the particular process it will be allocated and the remaining memory availability will be calculated.

Step 5: If not it has to tell no further memory remaining and the process will not be allocated with memory.

Step 6: Then the external fragmentation of each process must be calculated.

SOURCE CODE:

```
#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:");
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();
}
```

SAMPLE INPUT AND 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

RESULT:

Thus the program Multiprogramming variable task has been executed successfully.

MUTIPROGRAMMING FIXED TASK

Ex.No: 2b

OBJECTIVE:

To program MFT using c programming

ALGORITHM:

Step 1: Declare the necessary variables.

Step 2: Get the memory capacity and number of processes.

Step 3: calculate the number of partitions the total memory has to be divided.

Step 4: Get the required memory for each process if the required memory is available in that particular partition the process will be allocated to that partition and the internal fragmentation will be calculated.

Step 5: If the required memory not available then the process will not be allocated to that partition.

Step 6: Calculate the external fragmentation and total number of fragmentation.

SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
int main()
{
int m,p,s,p1;
int m1[4],i,f,f1=0,f2=0,fra1,fra2,s1,pos; 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);
pos=m;
for(i=0;i<p1;i++)
{
if(pos<s)
{
printf("\nThere is no further memory for process%d",i+1); m1[i]=0;
break;
}
else
{
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;
pos=pos-s;

}
else
{
printf("\nProcess not allocated in partition%d",i+1); s1=m1[i];
while(s1>s)
{
s1=s1-s;pos=pos-s;
}
pos=pos-s;fra2=s-s1;f2=f2+fra2;
printf("\nExternal Fragmentation for this process is:%d",fra2);
}
}
}
```

```
printf("\nProcess\tallocatedmemory");
for(i=0;i<p1;i++)
printf("\n%5d\t%5d",i+1,m1[i]);
f=f1+f2;
printf("\nThe tot no of fragmentation is:%d",f); getch();
return 0;
}
```

SAMPLE INPUT AND 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 Multiprogramming fixed task has been executed successfully.

1. **Explain about file allocation table.**
2. **What is sequential file allocation?**
3. **What is indexed file allocation?**
4. **What is linked file allocation?**
5. **Explain first fit, best fit and nearest fit strategies.**

(These Questions should be dictated to students at the end of the lab hours. The students must write the answers for those questions in their Observation Notebooks before they come to next lab.

SEQUENTIAL FILE ALLOCATION

Ex.No. 3a

OBJECTIVE:

To implement sequential file allocation technique.

ALGORITHM:

Step 1: Start the program.

Step 2: Get the number of files.

Step 3: Get the memory requirement of each file.

Step 4: Allocate the required locations to each in sequential order.

a). Randomly select a location from available location s1= random(100);

b). Check whether the required locations are free from the selected location.

c). Allocate and set flag=1 to the allocated locations.

Step 5: Print the results fileno, length , Blocks allocated.

Step 6: Stop the program.

SOURCE CODE:

```
#include<stdio.h>
main()
{
int f[50],i,st,j,len,c,k;
clrscr();
for(i=0;i<50;i++)
f[i]=0;
X:
printf("\n Enter the starting block & length of file");
scanf("%d%d",&st,&len);
for(j=st;j<(st+len);j++)
if(f[j]==0)
{
f[j]=1;
printf("\n%d->%d",j,f[j]);
}
else
{
printf("Block already allocated");
break;
}
if(j==(st+len))
printf("\n the file is allocated to disk");
printf("\n if u want to enter more files?(y-1/n-0)");
scanf("%d",&c);
if(c==1)
goto X;
else
exit();
getch();
}
```

Output:

Enter the starting block & length of file 4 10

4->1
5->1
6->1
7->1
8->1
9->1
10->1

11->1
12->1
13->1
The file is allocated to disk
If you want to enter more files? (Y-1/N-0)

RESULT :

Thus the program was executed successfully.

LINKED FILE ALLOCATION

Ex. No. 3.b

OBJECTIVE:

To write a C program to implement File Allocation concept using the technique Linked List Technique.

ALGORITHM:

Step 1: Start the Program

Step 2: Get the number of files.

Step 3: Allocate the required locations by selecting a location randomly

Step 4: Check whether the selected location is free.

Step 5: If the location is free allocate and set flag =1 to the allocated locations.

Step 6: Print the results file no, length, blocks allocated.

Step 7: Stop the execution

SOURCE CODE:

```
#include<stdio.h>
main()
{
int f[50],p,i,j,k,a,st,len,n,c;
clrscr();
for(i=0;i<50;i++)
f[i]=0;
printf("Enter how many blocks that are already allocated");
scanf("%d",&p);
printf("\nEnter the blocks no.s that are already allocated");
for(i=0;i<p;i++)
{
scanf("%d",&a);
f[a]=1;
}
X:
printf("Enter the starting index block & length");
scanf("%d%d",&st,&len);
k=len;
for(j=st;j<(k+st);j++)
{
if(f[j]==0)
{
f[j]=1;
printf("\n%d->%d",j,f[j]);
}
else
{
printf("\n %d->file is already allocated",j);
k++;
}
}
printf("\n If u want to enter one more file? (yes-1/no-0)");
scanf("%d",&c);
if(c==1)
goto X;
else
exit();
getch( );}
```

OUTPUT:

Enter how many blocks are already allocated 3
Enter the blocks no's that are already allocated 4 7 9
Enter the starting index block & length 3
7
3-> 1
4-> File is already allocated
5->1
6->1
7-> File is already allocated
8->1
9-> File is already allocated
10->1
11->1
12->1
If u want to enter one more file? (yes-1/no-0)

RESULT :

Thus the program was executed successfully

INDEXED FILE ALLOCATION

Ex. No. 3.c

OBJECTIVE:

To write a C program to implement File Allocation concept using the technique indexed allocation Technique

ALGORITHM:

Step 1: Start the Program

Step 2: Get the number of files.

Step 3: Get the memory requirement of each file.

Step 4: Allocate the required locations by selecting a location randomly.

Step 5: Print the results file no,length, blocks allocated.

Step 7: Stop the execution.

SOURCE CODE:

```
#include<stdio.h>
int f[50],i,k,j,inde[50],n,c,count=0,p;
main()
{
clrscr();
for(i=0;i<50;i++)
f[i]=0;
x:
printf("enter index block\t");
scanf("%d",&p);
if(f[p]==0)
{
f[p]=1;
printf("enter no of files on index\t");
scanf("%d",&n);
}
else
{
printf("Block already allocated\n");
goto x;
}
for(i=0;i<n;i++)
scanf("%d",&inde[i]);
for(i=0;i<n;i++)
if(f[inde[i]]==1)
{
printf("Block already allocated");
goto x;
}
for(j=0;j<n;j++)
f[inde[j]]=1;
printf("\n allocated");
printf("\n file indexed");
for(k=0;k<n;k++)
printf("\n %d->%d:%d",p,inde[k],f[inde[k]]);
printf(" Enter 1 to enter more files and 0 to exit\t");
scanf("%d",&c);
if(c==1)
goto x;
else
exit();
getch();
```

}

OUTPUT:

Enter index block 9
Enter no of files on index 3
1 2 3
Allocated
File indexed
9-> 1:1
9-> 2:1
9->3:1
Enter 1 to enter more files and 0 to exit

RESULT :

Thus the program was executed successfully

1. What is File organization in Operating System?
2. List the various types of File organization techniques.
3. What is Single Level Directory?
4. Define Two Level Directories.
5. Brief on Hierarchical way of file organization.

(These Questions should be dictated to students at the end of the lab hours. The students must write the answers for those questions in their Observation Notebooks before they come to next lab.)

FILE ORGANIZATION TECHNIQUES
SINGLE LEVEL DIRECTORY

Ex.No. 4.a.

OBJECTIVE:

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

ALGORITHM:

Step 1: Start the Program

Step 2: Initialize values gd=DETECT,gm,count,i,j,mid,cir_x.

Step 3: Initialize graph function

Step 4: Set back ground color with setbkcolor();

Step 5: Read number of files in variable count.

Step 6: check i<count; mid=640/count;

Step 7: Stop the execution

SOURCE CODE:

```
#include<stdio.h>
 #include<conio.h>
 #include <stdlib.h>
#include<graphics.h>
 void main()
 {
 int gd=DETECT,gm,count,i,j,mid,cir_x;
char fname[10][20];
 clrscr();
 initgraph(&gd,&gm,"c:\tc\bgi");
 cleardevice();
 setbkcolor(GREEN);
 puts("Enter no of files do u have?");
scanf("%d",&count);
for(i=0;i< count;i++)
{
 cleardevice();
 setbkcolor(GREEN);
 printf("Enter file %d name",i+1);
 scanf("%s",fname[i]);
 setfillstyle(1,MAGENTA);
 mid=640/count;
cir_x=mid/3;
 bar3d(270,100,370,150,0,0);
 settextstyle(2,0,4);
settextjustify(1,1);
 outtextxy(320,125,"Root Directory");
 setcolor(BLUE);
for(j=0;j< =i;j++,cir_x+=mid)
{
 line(320,150,cir_x,250);
 fillellipse(cir_x,250,30,30);
outtextxy(cir_x,250,fname[j]);
 } getch();
} }
```

RESULT :

Thus the program was executed successfully

FILE ORGANIZATION TECHNIQUES

TWO LEVEL

Ex. No.4b.

OBJECTIVE:

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

ALGORITHM:

Step 1: Start the Program

Step 2: Initialize structure elements

Step 3: Start main function

Step 4: Set variables gd =DETECT, gm;

Step 5: Create structure using create (&root,0,”null”,0,639,320);

Step 6: initgraph(&gd,&gm,”c:\tc\bgi”);

Step 7: Stop the execution

SOURCE CODE:

```
#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 truct tree_element node;
 void main()
{
 int gd=DETECT,gm;
node *root;
 root=NULL;
 clrscr();
 create(&root,0,"null",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);
if(lev==0||lev==1) (*root)->ftype=1;
 else
 (*root)->ftype=2;
```

```
 (*root)->level=lev;
 (*root)->y=50+lev*50;
 (*root)->x=x;
 (*root)->lx=lx;
 (*root)->rx=rx;
for(i=0;ilink[i]=NULL;
 if((*root)->ftype==1)
 {
 if(lev==0||lev==1)
{
 if((*root)->level==0)
 printf("How many users");
 else
 printf("hoe many files");
 printf("(for%s):",(*root)->name);
 scanf("%d",&(*root)->nc);
 }
 else
(*root)->nc=0;
if((*root)->nc==0) gap=rx-lx;
 else
 gap=(rx-lx)/(*root)->nc;
 for(i=0;inc;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,roo>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]);
}}
}
```

RESULT :

Thus the program was executed successfully.

FILE ORGANIZATION TECHNIQUES
HIERARCHICAL

Ex.No. 4c.

OBJECTIVE:

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

ALGORITHM:

Step 1: Start the Program

Step 2: Define structure and declare structure variables

Step 3: start main and declare variables

Step 4: Check a directory treestructure

Step 5: Display the directory tree in graphical mood

Step 6: Stop the execution

SOURCE CODE:

```
#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;inc;i++)
 {
 display(root->link[i]);
} }
 }
```

RESULT :

Thus the program was executed successfully

1. **Differentiate between Safe state and Unsafe state**
2. **What are the approaches have been used in deadlock Avoidance?**
3. **What are the restrictions does the Deadlock Avoidance have?**
4. **Define Claim Matrix, Allocation Matrix, Resource Vector**
5. **How to Calculate the Current Available resource in Banker's algorithm?**

(These Questions should be dictated to students at the end of the lab hours. The students must write the answers for those questions in their Observation Notebooks before they come to next lab.)

SIMULATE BANKERS ALGORITHM FOR DEADLOCK AVOIDANCE

Ex. No: 5

OBJECTIVE:

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

ALGORITHM:

Step 1: Start the Program

Step 2: Get the values of resources and processes.

Step 3: Get the avail value.

Step 4: After allocation find the need value.

Step 5: Check whether its possible to allocate. If possible it is safe state

Step 6: If the new request comes then check that the system is in safety or not if we allow the request.

Step 7: Stop the execution

SOURCE CODE:

```
#include<stdio.h>
 #include<conio.h>
 void main()
 {
 int n,r,i,j,k,p,u=0,s=0,m;
int block[10],run[10],active[10],newreq[10];
int max[10][10],resalloc[10][10],resreq[10][10];
int totalloc[10],totext[10],simalloc[10];
clrscr();
 printf("Enter the no of processes:");
scanf("%d",&n);
printf("Enter the no of resource classes:");
 scanf("%d",&r);
printf("Enter the total existed resource in each class:");
 for(k=1;k<=r;k++)
scanf("%d",&totext[k]);
 printf("Enter the allocated resources:");
 for(i=1;i<=n;i++)
for(k=1;k<=r;k++)
scanf("%d",&resalloc);
 printf("Enter the process making the new request:");
 scanf("%d",&p);
printf("Enter the requested resource:");
 for(k=1;k<=r;k++)
scanf("%d",&newreq[k]);
 printf("Enter the process which are n blocked or running:");
 for(i=1;i<=n;i++)
{
 if(i!=p)
 {
 printf("process %d:\n",i+1);
 scanf("%d%d",&block[i],&run[i]);
 } }
 block[p]=0;
 run[p]=0;
for(k=1;k<=r;k++)
 {
j=0;
for(i=1;i<=n;i++) {
 totalloc[k]=j+resalloc[i][k];
j=totalloc[k];
 } }
```

```
 for(i=1;i<=n;i++)
{
 if(block[i]==1||run[i]==1)
active[i]=1;
 else
 active[i]=0;
 }
for(k=1;k<=r;k++)
{
 resalloc[p][k]+=newreq[k];
 totalloc[k]+=newreq[k];
 }
 for(k=1;k<=r;k++)
 {
 if(totext[k]-totalloc[k]<=r;k++)
simalloc[k]=totalloc[k];
 for(s=1;s<=n;s++)
 for(i=1;i<=n;i++)
 {
 if(active[i]==1)
 {
 j=0;
for(k=1;k<=r;k++)
 {
 if((totext[k]-simalloc[k])<(max[i][k]-resalloc[i][k]))
{
J=1;break;
}}}
If(j==0)
{
active[i]=0;
 for(k=1;k<=r;k++)
simalloc[k]=resalloc[i][k];
} }
 m=0;
 for(k=1;k<=r;k++)
resreq[p][k]=newreq[k];
printf("Deadlock willn't occur");
 }
 Else
 {
 for(k=1;k<=r;k++)
{
 resalloc[p][k]=newreq[k];
```

```
 totalloc[k]=newreq[k];
 }
 printf("Deadlock will occur");
}
 getch();
 }
```

OUTPUT:

```
 Enter the no of resources: 4
 Enter the no of resource classes: 3
 Enter the total existed resources in each class: 3 2 2
 Enter the allocated resources: 1 0 0 5 1 1 2 1 1 0 0 2
Enter the process making the new request: 2
Enter the requested resource: 1 1 2
Enter the processes which are n blocked or running: Process 1: 1 2
Process 3: 1 0
Process 4: 1 0
Deadlock will occur
```

RESULT :

Thus the program was executed successfully

1. **1.How to recover once the Deadlock has been detected?**
2. **List the steps to illustrate the Deadlock Detection Algorithm.**
3. **3.What are the advantages to check each resource request?**
4. **4.How to fill the Allocation matrix?**
5. **5.How to identify whether the process exist deadlock or not?**

(These Questions should be dictated to students at the end of the lab hours. The students must write the answers for those questions in their Observation Notebooks before they come to next lab.)

SIMULATE AN ALGORITHM FOR DEAD LOCK DETECTION

Ex. No: 6

OBJECTIVE:

To write a C program to implement Deadlock Detection algorithm

ALGORITHM:

Step 1: Start the Program

Step 2: Get the values of resources and processes.

Step 3: Get the avail value..

Step 4: After allocation find the need value.

Step 5: Check whether its possible to allocate.

Step 6: If it is possible then the system is in safe state.

Step 7: Stop the execution

SOURCE 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("********** Deadlock Detection 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 resource 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 dead[100];
        int safe[100];
        int i,j;
        for(i=0;i<n;i++)
        {
                finish[i]=0;
        }
        //find need matrix
        for(i=0;i<n;i++)
        {
                for(j=0;j<r;j++)
                {
                        need[i][j]=max[i][j]-alloc[i][j];
                }
        }
        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("\nP%d",i);
 if(finish[i]==1)
   {
i=n;
 }}}}}}
              j=0;
              flag=0;
              for(i=0;i<n;i++)
              {
                        if(finish[i]==0)
                        {
                                  dead[j]=i;
                                  j++;
                                  flag=1;
                        }
              }
              if(flag==1)
              {
                        printf("\n\nSystem is in Deadlock and the Deadlock process are\n");
                        for(i=0;i<n;i++)
                        {
                                  printf("P%d\t",dead[i]);
                        }
              }
              else
              {
                        printf("\nNo Deadlock Occur");
              }
}
```

Output

```
Enter the no.  Of  processes 3
Enter the no of resources instances 3
Enter the max matrix
3 6 8
4 3 3
3 4 4
```

Enter the allocation matrix
3 3 3
2 0 3
1 2 4
Enter the available resources
1 2 0

Process	allocation	max	available
P1	3 3 3	3 6 8	1 2 0
P2	2 0 3	4 3 3	
P3	1 2 4	3 4 4	

System is in deadlock and deadlock process are
P0 p1 p2

RESULT :

Thus the program was executed successfully

1. **What is mean by frame locking?**
2. **Explain replacement policy.**
3. **Explain LRU policy.**
4. **How do FIFO works?**
5. **What is paging?**

(These Questions should be dictated to students at the end of the lab hours. The students must write the answers for those questions in their Observation Notebooks before they come to next lab.

SIMULATE ALL PAGE REPLACEMENT ALGORITHMS
A) FIFO

Ex. No: 7 a.

OBJECTIVE:

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

ALGORITHM:

Step 1: Start the Program

Step 2: Read number of pages n

Step 3: Read number of pages no

Step 4: Read page numbers into an arraya[i]

Step 5: Initialize aval[i]=o, to check page hit

Step 6: Print the results.

Step 7: Stop the Process.

SOURCE CODE:

```
#include<stdio.h>
#include<conio.h> void main()
{
int a[5],b[20],n,p=0,q=0,m=0,h,k,i,q1=1; char f='F';
clrscr();
printf("Enter the Number of Pages:"); scanf("%d",&n);
printf("Enter %d Page Numbers:",n); for(i=0;i<n;i++)
scanf("%d",&b[i]);
for(i=0;i<n;i++)
{if(p==0)
{
if(q>=3)
q=0;
a[q]=b[i];
q++;
if(q1<3)
{
q1=q;
}
}
printf("\n%d",b[i]);
printf("\t");
for(h=0;h<q1;h++)
printf("%d",a[h]);
if((p==0)&&(q<=3))
{
printf("-->%c",f);m++;
}
p=0;
for(k=0;k<q1;k++)
{
if(b[i+1]==a[k])
p=1;
}
}
printf("\nNo of faults:%d",m); getch();
}
```

SAMPLE INPUT AND OUTPUT:

Input:

Enter the Number of Pages: 12 Enter 12 Page Numbers:
2 3 2 1 5 2 4 5 3 2 5 2

Output:

2 2-> F
323-> F
223
1231-> F
5531-> F
2521-> F
4524-> F
5524
3 324-> F
2 324
5 354-> F
2 352-> F

No of faults: 9

RESULT:

Thus the program was executed successfully.

SIMULATE ALL PAGE REPLACEMENT ALGORITHMS
B) LRU

Exp. No: 7 b.

OBJECTIVE:

To write a C program to implement page replacement LRU (Least Recently Used**)** algorithm.

ALGORITHM:

Step 1: Start the Program

Step 2: Declare the size.

Step 3: Get the number of pages to be issued.

Step 4: Declare counter and stack

Step 5: Select the LRU page by counter value

Step 6: Stack them according the selection

Step 7: Stop the execution

SOURCE CODE:

```
#include<stdio.h>
#include<conio.h> void main()
{
int g=0,a[5],b[20],p=0,q=0,m=0,h,k,i,q1=1,j,u,n; char f='F';
clrscr();
printf("Enter the number of pages:"); scanf("%d",&n);
printf("Enter %d Page Numbers:",n); for(i=0;i<n;i++)
scanf("%d",&b[i]);
for(i=0;i<n;i++)
{if(p==0)
{
if(q>=3)
q=0;
a[q]=b[i];
q++;
if(q1<3)
{
q1=q;
//g=1;
}
}
printf("\n%d",b[i]);
printf("\t");
for(h=0;h<q1;h++)
printf("%d",a[h]);
if((p==0)&&(q<=3))
{
printf("-->%c",f);m++;
}
p=0;
g=0;
if(q1==3)
{
for(k=0;k<q1;k++)
```

```
{
if(b[i+1]==a[k])
p=1;
}
for(j=0;j<q1;j++)
{
u=0;
k=i;while(k>=(i-1)&&(k>=0))
{
if(b[k]==a[j])
u++;k--;
}
if(u==0)
q=j;
}
}
else
{
for(k=0;k<q;k++)
{
if(b[i+1]==a[k])
p=1;
}
}
}
printf("\nNo of faults:%d",m); getch();
}
```

SAMPLE INPUT AND OUTPUT:

Input:

Enter the Number of Pages: 12 Enter 12 Page Numbers:
2 3 2 1 5 2 4 5 3 2 5 2

Output:

22-> F
323-> F
2 23
1231-> F
5251-> F
2251
4254-> F
5254

3354-> F
2352-> F
5352
2352

No of faults: 7

RESULT:

Thus the program was executed successfully.

1. **Why we need shared memory?**
2. **Why is relocation needed?**
3. **What the basic requirement of memory management?**
4. **What is IPC?**
5. **Why most programs are organized into modules?**

(These Questions should be dictated to students at the end of the lab hours. The students must write the answers for those questions in their Observation Notebooks before they come to next lab.)

SHARED MEMORY AND IPC

Ex. No. 8

OBJECTIVE:

To write a C program to implement shared memory and inter process communication.

ALGORITHM:

Step 1: Start the Program

Step 2: Obtain the required process id

Step 3: Increment the *ptr=*ptr+1;

Step 4: Print the process identifier.

Step 5: check the values of sem_num, sem_op, sem_flg.

Step 6: Stop the execution.

SOURCE CODE:

```
#include<stdio.h>
#include<sys/ipc.h>
 #include<sys/sem.h>
 int main()
 {
 int id,semid,count=0,i=1,j;
 int *ptr;
 id=shmget(8078,sizeof(int),IPC_CREAT|0666);
 ptr=(int *)shmat(id,NULL,0);
union semun
{
 int val;
 struct semid_ds *buf;
 ushort *array;
 }u;
 struct sembuf sem;
semid=semget(1011,1,IPC_CREAT|0666);
 ushort a[1]={1};
 u.array=a;
semctl(semid,0,SETALL,u);
while(1)
{
 sem.sem_num=0;
sem.sem_op=-1;
 sem.sem_flg=0;
semop(semid,&sem,1);
 *ptr=*ptr+1;
 printf("process id:%d countis :%d \n",getpid(),*ptr);
 for(j=1;j< =1000000;j++)
 {
 sem.sem_num=0;
sem.sem_op=+1;
 sem.sem_flg=0;
 semop(semid,&sem,1);
 }
 }
 shmdt(ptr);
 }
```

SAMPLE INPUT AND OUTPUT:

enter Process id 16338992
pprocess
10
process 1

RESULT:

Thus the program was executed successfully.

1. **What is virtual memory?**
2. **What is segmentation?**
3. **What is paging?**
4. **What are frames?**
5. **What is page table?**

(These Questions should be dictated to students at the end of the lab hours. The students must write the answers for those questions in their Observation Notebooks before they come to next lab.)

SIMULATE PAGING TECHNIQUE OF MEMORY MANAGEMENT

Ex. No. 9

OBJECTIVE:

To write a C program to implement the concept of Paging

ALGORITHM:

Step 1: Read all the necessary input from the keyboard.

Step 2: Pages – Logical memory is broken into fixed- sized blocks.

Step 3: Frames – physical memory is broken into fixed – sized blocks

Step 4: Calculate the physical address using the following Physical address=(frame number * Frame size)+ offset

Step 5: Display the physical address.

Step 6: Stop the execution

SOURCE CODE:

```
#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); sa=(int*)malloc(2*np); for(i=0;i<np;i++)
{
sa[i]=(int)malloc(ps);
printf("page%d\t address %u\n",i+1,sa[i]);
}
getch();
}
```

SAMPLE INPUT AND OUTPUT:

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 was executed successfully.

1. **What is a semaphore?**
2. **Explain about mutexes?**
3. **What are the different types of thread synchronization mechanisms?**
4. **Explain about multilevel queue scheduler thread operation?**
5. **What is SMP?**

(These Questions should be dictated to students at the end of the lab hours. The students must write the answers for those questions in their Observation Notebooks before they come to next lab.)

THREADING & SYNCHRONIZATION

Ex.No. 10

OBJECTIVE:

To write a C program to implement Threading & Synchronization

ALGORITHM:

Step 1: Start the Program

Step 2: Initialize the process thread array.

Step 3: Print the job started status.

Step 4: Print the job finished status.

Step 5: Start the main function

Step 6: Check for the process creation if not print error message.

Step 7: Stop the execution

SOURCE CODE:

```
#include<stdio.h>
 #include <string.h>
#include<pthread.h>
 #include <stdlib.h>
#include <unistd.h>
pthread_t tid[2];
 int counter;
 void* doSomeThing(void *arg)
{
unsigned long i = 0;
 counter += 1;
 printf("\n Job %d started\n", counter);
 for(i=0; i< (0xFFFFFFFF);i++);
printf("\n Job %d finished\n", counter);
return NULL;
}
 int main(void)
{
 int i = 0;
 int err;
while(i < 2)
 {
 err = pthread_create(&(tid[i]), NULL, &doSomeThing, NULL);
if (err != 0)
 printf ("\ncan't create thread :[%s]", strerror(err));
 i++;
 }
 pthread_join(tid[0], NULL);
pthread_join(tid[1], NULL);
 return 0;
 }
```

SAMPLE OUTPUT:

Job 1 started

Job 1finished

can't create thread :

RESULT:

Thus the program was executed successfully.