OPERATING SYSTEM LAB MANUAL

LAB MANUAL

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

BALAJI INSTITUTE OF TECHNOLOGY & SCIENCE

Affiliated to JNTU Hyderabad, Approved by AICTE New Delhi

Accredited by NAAC & Certified by ISO 9001-2015

Laknepally (V) Narsampet, Warangal - 506331

List of Programs:

1. Write C programs to simulate the following CPU scheduling algorithms:
a) Round Robin b) SJF

a) Round Robin CPU scheduling algorithm

```
#include<stdio.h>
 main()
{
int et[30],ts,n,i,x=0,tot=0;
char pn[10][10];
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();
}
```

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

B. SJF CPU scheduling algorithm

```
#include<stdio.h>
#include<string.h>
 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];
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];
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\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();
}
```

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	arrivaltime	executiontime	waitingtime	tatime
1	4	6	0	6
3	6	11	4	15
2	5	15	16	31

Average Waiting Time: 6.6667

Average Turn Around Time: 17.3333

2. Write C programs to simulate the following CPU scheduling algorithms:
a) Priority b) FCFS

a) Priority CPU scheduling algorithm

```
#include<stdio.h>
#include<string.h>
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];
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();
}
```

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	waitingtime	tatime
1	2	3	1	0	3
2	4	5	2	1	6
3	5	6	3	5	11

Average Waiting Time: 2.0000

Average Turn Around Time: 6.6667

b) FCFS CPU scheduling algorithm

```
#include<stdio.h>
main()
{
char pn[10][10];
int arr[10],bur[10],star[10],finish[10],tat[10],wt[10],i,n;
int totwt=0,tottat=0;
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++)
{
if(i==0)
{
star[i]=arr[i];
wt[i]=star[i]-arr[i];
finish[i]=star[i]+bur[i];
tat[i]=finish[i]-arr[i];
}
else
{
star[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    Start    TAT    Finish");
for(i=0;i<n;i++)
{
printf("\n%s\t%6d\t\t%6d\t%6d\t%6d\t%6d",pn[i],arr[i],bur[i],star[i],tat[i],finish[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();
}
```

OUTPUT:

Input:

Enter the number of processes: 3

Enter the Process Name, Arrival Time & Burst Time: 1 2 3

Enter the Process Name, Arrival Time & Burst Time: 2 5 6

Enter the Process Name, Arrival Time & Burst Time: 3 6 7

Output:

PName	Arrtime	Burtime	Srart	TAT	Finish
1	2	3	2	3	5
2	5	6	5	6	4
3	6	7	6	7	10

Average Waiting Time: 3.333

Average Turn Around Time: 7.000

3. Write C programs to simulate the following File organization techniques:
a) Single level directory b) Two level c) Hierarchical

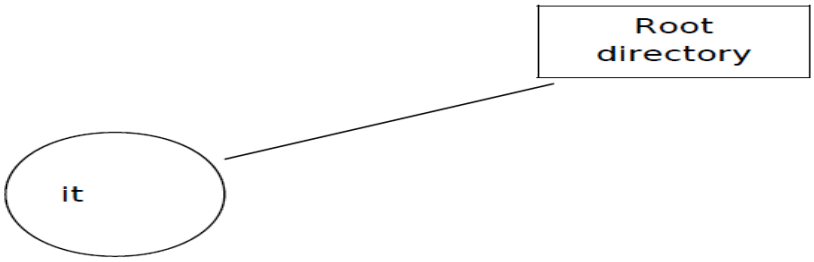
a)Single Level Directory

```
#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);
printf("enter number of files");
scanf("&d",&count);
if(i<count)
// for(i=0;i<count;i++)
{
cleardevice();
setbkcolor(GREEN);
printf("enter %d file 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);
i++;
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[i]);
}
}
getch();
```

}

output:-

```
Enter number of files: 2
Enter file 1 name: it
```



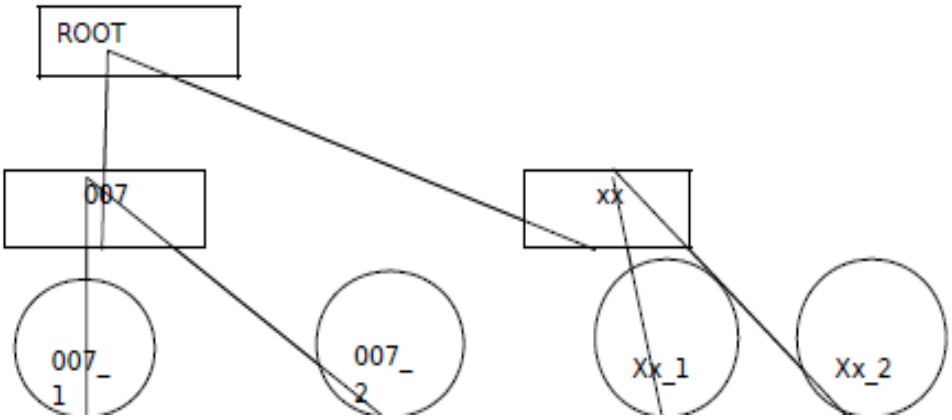
B. Two Level Directory:-

```
#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,"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 the name of ir file name %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;i<5;i++)
(*root)->link[i]=NULL;
if((*root)->ftype==1)
{
if(lev==0 || lev==1)
{
if((*root)->level==0)
```

```
printf("how many users");
else
printf(" how 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;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]);
}
}

}
```

Output

```
Enter the name of the dir file ROOT
Ho many users :2
Enter the name of the dir file: 007
How many files  :2
Enter the name of the dir file007_1
Enter the name of the dir file007_2
Enter the name of the dir file  xx
How many files:  2
Enter the name of dir file xx_1
Enter the name of dir file xx_2
```



C. Hierarchical Directory Organization

```
#include<stdio.h>

#include<graphics.h>

structtree_element

{

charname[20];

intx,y,ftype,lx,rx,nc,level;

structtree_element*link[5];

};
```

```
typedefstructtree_elementnode;
voidmain()
{
intgd=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,intlev,char*dname,intlx,intrx,intx)
{
inti,gap;
if(*root==NULL)
{
(*root)=(node*)malloc(sizeof(node));
printf("Enternameofdir/file(under%s):",dname);
fflush(stdin);
gets((*root)->name);
printf("enter1forDir/2forfile:");
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("Noofsubdirectories/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)
{
inti;
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]);
}
}
```

OUTPUT

```
INPUT DATA
EnterNameofdir/file(underroot):ROOT
Enter1forDir/2forFile:1
Noofsubdirectories/files(forROOT):2
EnterNameofdir/file(underROOT):USER1
Enter1forDir/2forFile:1
Noofsubdirectories/files(forUSER1):1
EnterNameofdir/file(underUSER1):SUBDIR1
Enter1forDir/2forFile:1
Noofsubdirectories/files(forSUBDIR1):2
EnterNameofdir/file(underUSER1):JAVA
Enter1forDir/2forFile:1
Noofsubdirectories/files(forJAVA):0
EnterNameofdir/file(underSUBDIR1):VB
Enter1forDir/2forFile:1
Noofsubdirectories/files(forVB):0
EnterNameofdir/file(underROOT):USER2
Enter1forDir/2forFile:1
Noofsubdirectories/files(forUSER2):2
EnterNameofdir/file(underROOT):A
```

Enter1forDir/2forFile:2

EnterNameofdir/file(underUSER2):SUBDIR2

Enter1forDir/2forFile:1

Noofsubdirectories/files(forSUBDIR2):2

EnterNameofdir/file(underSUBDIR2):PPL

Enter1forDir/2forFile:1

Noofsubdirectories/files(forPPL):2

EnterNameofdir/file(underPPL):B

Enter1forDir/2forFile:2

EnterNameofdir/file(underPPL):C

Enter1forDir/2forFile:2

EnterNameofdir/file(underSUBDIR):AI

Enter1forDir/2forFile:1

Noofsubdirectories/files(forAI):2

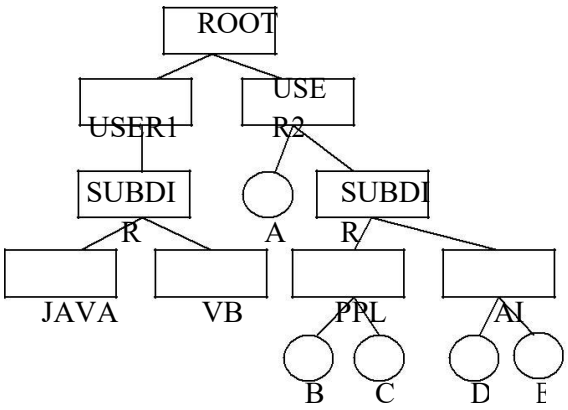
EnterNameofdir/file(underAI):D

Enter1forDir/2forFile:2

EnterNameofdir/file(underAI):E

Enter1forDir/2forFile:2

OUTPUT



4. Write C programs to simulate the following File allocation methods:
a)Contiguous b)Linked c)Indexed

a. Sequential file Allocation */

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
   int st[20],b[20],b1[20],ch,i,j,n,blocks[20][20],sz[20];
   char F[20][20],S[20];
   clrscr();
   printf("\n Enter no. of Files ::");
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
      printf("\n Enter file %d name ::",i+1);

      scanf("%s",&F[i]);
      printf("\n Enter file%d size(in kb)::",i+1);
      scanf("%d",&sz[i]);
      printf("\n Enter Starting block of %d::",i+1);
      scanf("%d",&st[i]);
      printf("\n Enter blocksize of File%d(in bytes)::",i+1);
      scanf("%d",&b[i]);
   }
   for(i=0;i<n;i++)
      b1[i]=(sz[i]*1024)/b[i];
   for(i=0;i<n;i++)
   {
      for(j=0;j<b1[i];j++)
         blocks[i][j]=st[i]+j;
   }
   do
   {
      printf("\nEnter the Filename ::");
      scanf("%s",S);
      for(i=0;i<n;i++)
      {
         if(strcmp(S,F[i])==0)
         {
            printf("\nFname\tStart\tNblocks\tBlocks\n");
            printf("\n----------------------------------------\n");
            printf("\n%s\t%d\t%d\t",F[i],st[i],b1[i]);
```

```
            for(j=0;j<b1[i];j++)
               printf("%d->",blocks[i][j]);
         }

      }
      printf("\n---------------------------------------------\n");
      printf("\nDo U want to continue ::(Y:n)");
      scanf("%d",&ch);
      if(ch!=1)
         break;
   }while(1);
}
/*Input and Output;-

 Enter no. of Files ::2
 Enter file 1 name ::x.c
 Enter file1 size(in kb)::4
 Enter Starting block of 1::100
 Enter blocksize of File1(in bytes)::512
 Enter file 2 name ::y.c
 Enter file2 size(in kb)::2
 Enter Starting block of 2::500
 Enter blocksize of File2(in bytes)::256
   Enter the Filename ::y.c

Fname   Start   Nblocks Blocks
---------------------------------------------
y.c     500     8       500->501->502->503->504->505->506->507->
---------------------------------------------
Do U want to continue ::(Y:n) n             */
```

/*b. Index file Allocation Method   */

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int n;
void main()
{
```

```
    int b[20],b1[20],i,j,blocks[20][20],sz[20];
    char F[20][20],S[20],ch;
    clrscr();
    printf("\n Enter no. of Files ::");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
       printf("\n Enter file %d name ::",i+1);
       scanf("%s",&F[i]);
       printf("\n Enter file%d size(in kb)::",i+1);
       scanf("%d",&sz[i]);
       printf("\n Enter blocksize of File%d(in bytes)::",i+1);
       scanf("%d",&b[i]);
    }
    for(i=0;i<n;i++)
    {
       b1[i]=(sz[i]*1024)/b[i];
       printf("\n\nEnter blocks for file%d",i+1);
       for(j=0;j<b1[i];j++)
       {
          printf("\n Enter the %dblock ::",j+1);
          scanf("%d",&blocks[i][j]);
       }
    }
    do
    {
       printf("\nEnter the Filename ::");
       scanf("%s",&S);
       for(i=0;i<n;i++)
       {
          if(strcmp(F[i],S)==0)
          {
       printf("\nFname\tFsize\tBsize\tNblocks\tBlocks\n");
       printf("\n-----------------------------------------\n");
       printf("\n%s\t%d\t%d\t%d\t",F[i],sz[i],b[i],b1[i]);
       for(j=0;j<b1[i];j++)
        printf("%d->",blocks[i][j]);
          }
       }
       printf("\n-----------------------------------------\n");
       printf("\nDo U want to continue ::(Y:n)");
       scanf("%d",&ch);
    }while(ch!=0);
}
/*Input and Output;-

 Enter no. of Files ::2
```

```
Enter file 1 name ::x.c

Enter file1 size(in kb)::1

Enter blocksize of File1(in bytes)::512

Enter file 2 name ::y.c

Enter file2 size(in kb)::1

Enter blocksize of File2(in bytes)::512

Enter blocks for file1

Enter the 1block ::1000

Enter the 2block ::1001

Enter blocks for file2

Enter the 1block ::2000

Enter the 2block ::2001

Enter the Filename ::x.c

Fname   Fsize   Bsize   Nblocks Blocks
------------------------------------------------
x.c     1       512     2       1000->1001->
------------------------------------------------

Do U want to continue ::(Y:n)1

Enter the Filename ::y.c

Fname   Fsize   Bsize   Nblocks Blocks
------------------------------------------------
y.c     1       512     2       2000->2001->
------------------------------------------------

Do U want to continue ::(Y:n)0 */
```

c. Linked file Allocation Method */

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int n;
void main()
{
   int b[20],b1[20],i,j,blocks[20][20],sz[20];
   char F[20][20],S[20],ch;
   int sb[20],eb[20],x;
   clrscr();
   printf("\n Enter no. of Files ::");
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
      printf("\n Enter file %d name ::",i+1);
      scanf("%s",&F[i]);
      printf("\n Enter file%d size(in kb)::",i+1);
      scanf("%d",&sz[i]);
      printf("\n Enter blocksize of File%d(in bytes)::",i+1);
      scanf("%d",&b[i]);
   }
   for(i=0;i<n;i++)
   {
```

```
        b1[i]=(sz[i]*1024)/b[i];
        printf("\n Enter Starting block of file%d::",i+1);
        scanf("%d",&sb[i]);
        printf("\n Enter Ending block of file%d::",i+1);
        scanf("%d",&eb[i]);
        printf("\nEnter blocks for file%d::\n",i+1);
        for(j=0;j<b1[i]-2;)
        {
           printf("\n Enter the %dblock ::",j+1);
           scanf("%d",&x);
           if(x>sb[i]&&x<eb[i])
           {
              blocks[i][j]=x;
              j++;
           }
           else
              printf("\n Invalid block::");
        }
     }
     do
     {
        printf("\nEnter the Filename ::");
        scanf("%s",&S);
        for(i=0;i<n;i++)
        {
           if(strcmp(F[i],S)==0)
           {
              printf("\nFname\tFsize\tBsize\tNblocks\tBlocks\n");
              printf("\n-----------------------------------------------\n");
              printf("\n%s\t%d\t%d\t%d\t",F[i],sz[i],b[i],b1[i]);
              printf("%d->",sb[i]);
              for(j=0;j<b1[i]-2;j++)
                 printf("%d->",blocks[i][j]);
              printf("%d->",eb[i]);
           }
        }
        printf("\n-----------------------------------------------\n");
        printf("\nDo U want to continue (Y:n)::");
        scanf("%d",&ch);
     }while(ch!=0);
}
/*Input and Output;-

 Enter file1 size(in kb)::1

 Enter blocksize of File1(in bytes)::512

 Enter file 2 name ::sudee
```

Enter file2 size(in kb)::1

Enter blocksize of File2(in bytes)::1024

Enter Starting block of file::1100

Enter Ending block of file::1600

Enter blocks for file1::

Enter the 1block ::102

Enter the 2block ::104

Enter Starting block of file::2200

Enter Ending block of file::2500

Enter blocks for file2::

Enter the 1block ::201

Enter the Filename ::daya

```
Fname   Fsize   Bsize   Nblocks Blocks
-----------------------------------------------
daya    1       512     2       100->102->104->600->
-----------------------------------------------
```

Do U want to continue ::(Y:n)1

Enter the Filename ::sudee

```
Fname   Fsize   Bsize   Nblocks Blocks
-----------------------------------------------
sudee   1       1024    1       200->201->500->
-----------------------------------------------
```

Do U want to continue ::(Y:n)0

5.Write a C program to copy the contents of one file to another using system calls

```
#include <syscall.h>
#include <unistd.h>
#include <sys/types.h>
#include <fcntl.h>
#include <sys/uio.h>
#include <sys/stat.h>
#include <stdio.h>
int main(int argc,char * argv[])

{
	int fd;

	fd=open(argv[1],O_CREAT | O_RDONLY);
	if(fd==-1)
	{
		printf("error opening the file");
	}
	void *buf = (char*) malloc(120);
	int count=read(fd,buf,120);
	printf("count : %d",count);
	printf("%s",buf);
	close(fd);
	int f1;
	f1=open(argv[2],O_CREAT | O_WRONLY);
	if(f1==-1)
	{
		printf("error opening the file");
	}
	int c;
```

```
    while(count=read(fd,buf,120)>0)
    {
        c=write(f1,buf,120);
    }
    if(c==-1)
    {
        printf("error writing to the file");
    }
    close(f1);
}
```

6.Write a C program to simulate Bankers Algorithm for Dead Lock Avoidance

```
#include<stdio.h>
 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];
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]<0)
{
u=1;break;
}
}
if(u==0)
{
for(k=1;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 processes:3
Enter the no of resource classes:
5
Enter the total existed resource in each class:
3 3 5 4 5
Enter the allocated resources:
```

```
   0 1 0 2 0 0 3 0 2 2 1 1 0 0 2
Enter the process making the new request:
2
Enter the requested resource:5
5 6 7 2
Enter the process which are n blocked or running: process 2:
3 4
process 4:
5 5
Deadlock will occur
```

7.A program to simulate Bankers Algorithm for Deadlock Prevention.

```
#include<stdio.h>
main()
{
int cl[10][10],al[10][10],av[10],i,j,k,m,n,c,ne[10][10],flag=0;
printf("\nEnter the matrix");
scanf("%d %d",&m,&n);
printf("\nEnter the claim matrix");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
scanf("%d",&cl[i][j]);
}
}
printf("\nEnter allocated matrix");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
```

```
scanf("%d",&al[i][j]);
}
}
printf("\nThe need matrix");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
ne[i][j]=cl[i][j]-al[i][j];
printf("\t%d",ne[i][j]);
}
printf("\n");
}
printf("\nEnter avaliable matrix");
for(i=0;i<3;i++)
scanf("%d",av[i]);
printf("Claim matrix:\n");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
printf("\t%d",cl[i][j]);
}
printf("\n");
}
printf("\n allocated matrix:\n");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
printf("\t%d",al[i][j]);
}
printf("\n");
}
printf(" available matrix:\n");
for(i=0;i<3;i++)
{
printf("\t%d",av[i]);
}
for(k=0;k<m;k++)
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
if(av[j]>=ne[i][j])
flag=1;
else
break;
if(flag==1&& j==n-1)
goto a;
}
}
a: if(flag==0)
{
printf("unsafestate");
```

```
}
if(flag==1)
{
flag=0;
for(i=0;i<m;i++)
{
for(j=0;j<n;i++)
{
av[j]+=al[i][j];
al[i][j]=1;
}
}
printf("\n safe state");
for(i=0;i<n;i++)
printf("\t available matrix:%d",av[i]);
}

getch();
}
```

OUTPUT:

```
Enter the matrix
2 2 2
Enter the claim matrix
2 2 2

Enter allocated matrix
2 2 2 2

The need matrix          0      0
                         0      0

Enter avaliable matrix
0 2 3
Claim matrix:
       2      2
       2      2

 allocated matrix:
       2      2
       2      2
 available matrix:
       0      0      0
```

8. Write C programs to simulate the following page replacement algorithms:
a) FIFO b) LRU c) LFU

FIFO Page Replacement Algorithm

```
#include<stdio.h>
int fr[3];
 main()
{
        void display();
        int i,j,page[12]={2,3,2,1,5,2,4,5,3,2,5,2};
        int flag1=0,flag2=0,pf=0,frsize=3,top=0;
        for(i=0;i<3;i++)
        {
        fr[i]=-1;
        }
        for(j=0;j<12;j++)
        {
        flag1=0;
        flag2=0;
        for(i=0;i<12;i++)
        {
        if(fr[i]==page[j])
```

```
{
flag1=1;
flag2=1;
break;
}
}
if(flag1==0)
{
for(i=0;i<frsize;i++)
{
if(fr[i]==-1)
{
fr[i]=page[j];
flag2=1;
break;
}
}
}
if(flag2==0)
{
fr[top]=page[j];
top++;
pf++;
if(top>=frsize)
top=0;
}
display();
}
printf("Number of page faults  : %d ",pf);
getch();
```

```
}
void display()
{
int i;
printf("\n");
for(i=0;i<3;i++)
printf("%d\t",fr[i]);
}
```

OUTPUT :

```
2 -1 -1

2  3 -1

2  3 -1

2  3  1

5  3  1

5  2  1

5  2  4

5  2  4

3  2  4

3  2  4
```

3 5 4

3 5 2

Number of page faults : 6

b) LRU Page Replacement Algorithm

```
#include<stdio.h>
int fr[3];
 main()
{
void display();
```

```
int p[12]={2,3,2,1,5,2,4,5,3,2,5,2},i,j,fs[3];
int index,k,l,flag1=0,flag2=0,pf=0,frsize=3;
for(i=0;i<3;i++)
{
fr[i]=-1;
}
for(j=0;j<12;j++)
{
flag1=0,flag2=0;
for(i=0;i<3;i++)
{
if(fr[i]==p[j])
{
flag1=1;
flag2=1;
break;
}
}
if(flag1==0)
{
for(i=0;i<3;i++)
{
if(fr[i]==-1)
{
fr[i]=p[j];
flag2=1;
break;
}
}
}
```

```
if(flag2==0)
{
for(i=0;i<3;i++)
fs[i]=0;
for(k=j-1,l=1;l<=frsize-1;l++,k--)
{
for(i=0;i<3;i++)
{
if(fr[i]==p[k])
fs[i]=1;
}
}
for(i=0;i<3;i++)
{
if(fs[i]==0)
index=i;
}
fr[index]=p[j];
pf++;
}
display();
}
printf("\n no of page faults :%d",pf);
getch();
}
void display()
{
int i;
printf("\n");
for(i=0;i<3;i++)
```

```
printf("\t%d",fr[i]);
}
```

OUTPUT :

```
2 -1 -1
2  3 -1
2  3 -1
2  3  1
2  5  1
2  5  1
2  5  4
2  5  4
3  5  4
3  5  2
3  5  2
3  5  2
```

no of page faults : 4

c). LFU Least Frequently Used Page Replacement Algorithm

```
#include<stdio.h>
void main()
{
int q[20],p[50],c=0,c1,d,f,i,j,k=0,n,r,t,b[20],c2[20];
printf("Enter no of pages:");
scanf("%d",&n);
printf("Enter the reference string:");
for(i=0;i<n;i++)
scanf("%d",&p[i]);
printf("Enter no of frames:");
scanf("%d",&f);
q[k]=p[k];
printf("\n\t%d\n",q[k]);
c++;
k++;
for(i=1;i<n;i++)
{
c1=0;
for(j=0;j<f;j++)
{
if(p[i]!=q[j])
c1++;
}
if(c1==f)
{
c++;
if(k<f)
{
q[k]=p[i];
k++;
for(j=0;j<k;j++)
printf("\t%d",q[j]);
printf("\n");
}
else
{
for(r=0;r<f;r++)
```

```
{
c2[r]=0;
for(j=i-1;j<n;j–)
{
if(q[r]!=p[j])
c2[r]++;
else
break;
}
}
for(r=0;r<f;r++)
b[r]=c2[r];
for(r=0;r<f;r++)
{
for(j=r;j<f;j++)
{
if(b[r]<b[j])
{
t=b[r];
b[r]=b[j];
b[j]=t;
}
}
}
for(r=0;r<f;r++)
{
if(c2[r]==b[0])
q[r]=p[i];
printf(“\t%d”,q[r]);
}
printf(“\n”);
}
}
}
printf(“\nThe no of page faults is %d”,c);
}

/*
```

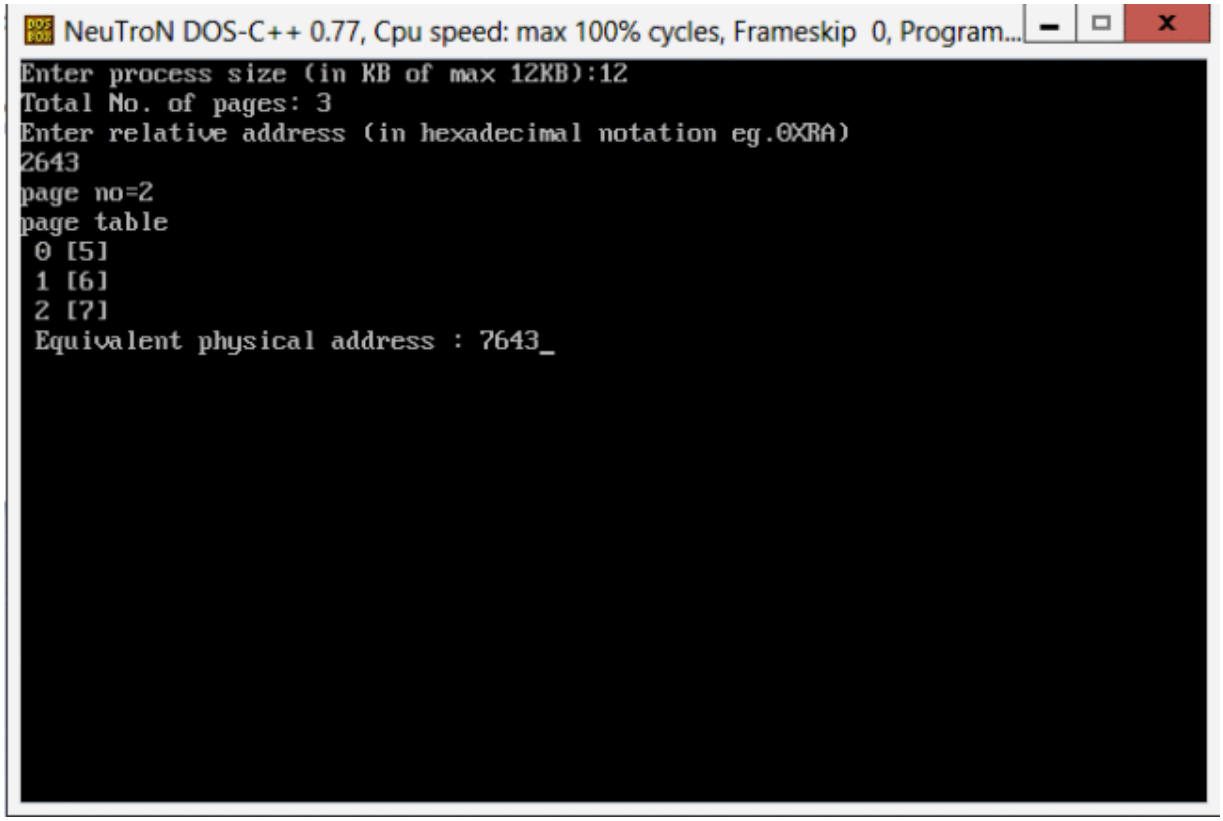
OUTPUT:

```
Enter no of pages:10
Enter the reference string:7 5 9 4 3 7 9 6 2 1
Enter no of frames:3
7
7 5
7 5 9
4 5 9
4 3 9
4 3 7
9 3 7
9 6 7
9 6 2
1 6 2
```

9.Write C programs to simulate the following techniques of memory management:
a) Paging **b) Segmentation**

Paging Technique

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
main()
{
int size,m,n,pgno,pagetable[3]={5,6,7},i,j,frameno;
double m1;
int ra=0,ofs;
clrscr();
printf("Enter process size (in KB of max 12KB):");/*reading memeory size*/
scanf("%d",&size);
m1=size/4;
n=ceil(m1);
printf("Total No. of pages: %d",n);
printf("\nEnter relative address (in hexadecimal notation eg.0XRA) \n");
//printf("The length of relative Address is : 16 bits \n\n The size of offset is :12 bits\n");
scanf("%d",&ra);
pgno=ra/1000; /*calculating physical address*/
ofs=ra%1000;
printf("page no=%d\n",pgno);
printf("page table");
for(i=0;i<n;i++)
printf("\n %d [%d]",i,pagetable[i]);
frameno=pagetable[pgno];
printf("\n Equivalent physical address : %d%d",frameno,ofs);
getch();
}
```



B. Implementation of Segmentation.

SOURCE CODE:
#include<stdio.h>

```
#include<conio.h>
struct list
{
int seg;
int base;
int limit;
struct list *next;
} *p;
void insert(struct list *q,int base,int limit,int seg)
{
if(p==NULL)
{
p=malloc(sizeof(Struct list));
p->limit=limit;
p->base=base;
p->seg=seg;
p->next=NULL;
}
else
{
while(q->next!=NULL)
{
Q=q->next;
Printf("yes")
}
q->next=malloc(sizeof(Struct list));
q->next ->limit=limit;
q->next ->base=base;
q->next ->seg=seg;
q->next ->next=NULL;
}
}
int find(struct list *q,int seg)
{
while(q->seg!=seg)
{
q=q->next;
}
return q->limit;
}
int search(struct list *q,int seg)
{
while(q->seg!=seg)
{
q=q->next;
}
return q->base;
}
```

```
main()
{
p=NULL;
int seg,offset,limit,base,c,s,physical;
printf(“Enter segment table/n”);
printf(“Enter -1 as segment value for termination\n”);
do
{
printf(“Enter segment number”);
scanf(“%d”,&seg);
if(seg!=-1)
{
printf(“Enter base value:”);
scanf(“%d”,&base);
printf(“Enter value for limit:”);
scanf(“%d”,&limit);
insert(p,base,lmit,seg);
}
}
while(seg!=-1)
printf(“Enter offset:”);
scanf(“%d”,&offset);
printf(“Enter bsegmentation number:”);
scanf(“%d”,&seg);
c=find(p,seg);
s=search(p,seg);
if(offset<c)
{
physical=s+offset;
printf(“Address in physical memory %d\n”,physical);
}
else
{
printf(“error”);
}
}
```

OUTPUT:

```
 $ cc seg.c
 $ ./a.out

Enter segment table
Enter -1 as segmentation value for termination
Enter segment number:1
Enter base value:2000
Enter value for limit:100
Enter segment number:2
Enter base value:2500
```

```
Enter value for limit:100
Enter segmentation number:-1
Enter offset:90
Enter segment number:2
Address in physical memory 2590

 $ ./a.out

Enter segment table
Enter -1 as segmentation value for termination
Enter segment number:1
Enter base value:2000
Enter value for limit:100
Enter segment number:2
Enter base value:2500
Enter value for limit:100
Enter segmentation number:-1
Enter offset:90
Enter segment number:1
Address in physical memory 2090
```

10.Write a C program to implement the ls | sort command.

```
/* ls command simulation - list.c */
#include <stdio.h>
#include <dirent.h>
main()
{
struct dirent  **namelist;
int n,i;
char pathname[100];
getcwd(pathname);
n = scandir(pathname, &namelist, 0, alphasort);
if(n < 0)
printf("Error\n");
else
for(i=0; i<n; i++) if(namelist[i]->d_name[0] != '.')
printf("%-20s", namelist[i]->d_name);
}
```

Output

```
$ gcc list.c -o list
$ ./list
```

cmdpipe.c consumer.c
a.out
dirlist.c ex6a.c ex6b.c
ex6c.c ex6d.c exec.c
fappend.c fcfs.c fcreate.c
fork.c fread.c hello
list list.c pri.c
producer.c rr.c simls.c
sjf.c stat.c wait.c

11. Write a C program to solve the Dining- Philosopher problem using semaphores.

```
#include<stdio.h>
#include<stdlib.h>
#include<semaphore.h>
#include<pthread.h>
#include<unistd.h>
#include<string.h>
#include<sys/types.h>
#include<sys/wait.h>
#define N 5
#define LEFT (i+4)%N

#define RIGHT (i+1)%N

#define THINKING 0
#define HUNGRY 1
#define EATING 2
sem_t spoon;
sem_t phil[N];
int state[N];
int phil_num[N]={0,1,2,3,4};
int fd[N][2];// file descriptors for pipes
pid_t pid, pids[N];// process ids
int i;int num;
void philosopher(int i);
```

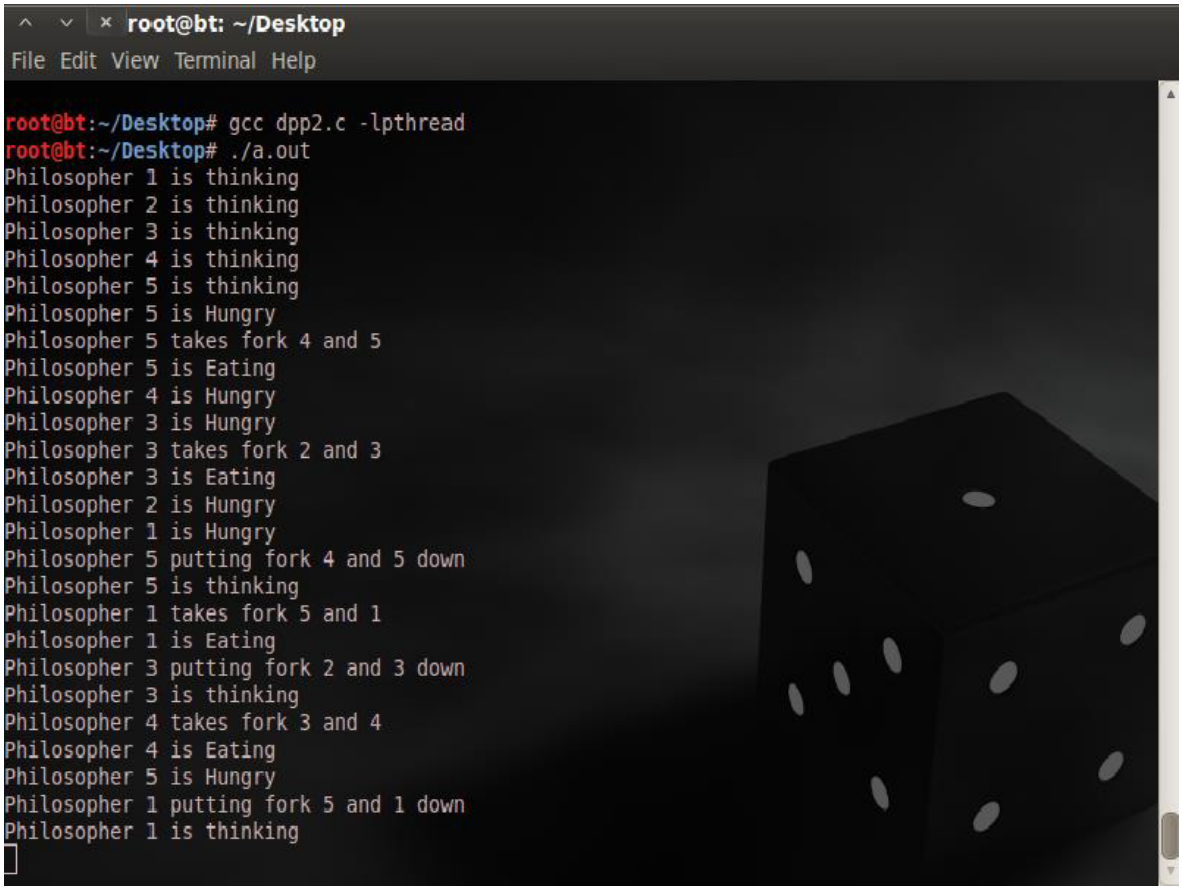
```
void test(int i);
void take_spoon(int i);
void put_spoon(int i);
char buffer[100];
int main(void)
{
for(i=0;i<N;++i)
{

pipe(fd[i]);
pids[i]= fork();

printf("i=%d\n",i);

printf("pids[i]=%d\n",pids[i]);
if(pids[i]==0)
{// child

dup2(fd[i][1],1);

close(fd[i][0]);

close(fd[i][1]);

philosopher(i);

exit(0);
}
else
if(pids[i]>0)
{
// parent

dup2(fd[i][0],0);

close(fd[i][0]);

close(fd[i][1]);
}
}// wait for child processes to end
for(i=0;i<N;++i)
waitpid(pids[i],NULL,0);
return 0;
}
void philosopher(int i)
{
while(1)
{

sleep(1);
```

```
take_spoon(i);

sleep(2);

put_spoon(i);

sleep(1);
}
}
void take_spoon(int i)
{

sem_wait(&spoon);

state[i]= HUNGRY;

printf("philosopher %d is hungry\n",i+1);

test(i);

sem_post(&spoon);

sem_wait(&phil[i]);
}
void put_spoon(int i)
{

sem_wait(&spoon);

state[i]= THINKING;

printf("philosopher %d puts down spoon %d and %d hin\n",i+1,LEFT+1,i+1);

printf("philosopher %d thinks\n",i+1);

test(LEFT);

test(RIGHT);

sem_post(&spoon);
}
void test(int i)
{
if( state[i]== HUNGRY && state[LEFT]!= EATING && state[RIGHT]!= EATING)
{

state[i]= EATING;

printf("philosopher %d takes spoon %d and %d\n",i+1,LEFT+1,i+1);
```

printf(“philosopher %d eats\n”,i+1);

sem_post(&phil[i]);
}
}

OUTPUT



12.Write C programs to implement ipc between two unrelated processes using named pipe

```
#include<stdio.h>
#include<stdlib.h>
#include<errno.h>
#include<unistd.h>
int main()
{
int pfds[2];
char buf[30];
if(pipe(pfds)==-1)
{
perror("pipe");
exit(1);
}
printf("writing to file descriptor #%d\n", pfds[1]);
write(pfds[1],"test",5);
printf("reading from file descriptor #%d\n ", pfds[0]);
read(pfds[0],buf,5);
printf("read\"%s\"\n" ,buf);
}
```

Output:

```
 $ vi pipes1.c
 $ cc pipes1.c
 $ ./a.out
writing to file descriptor #4
reading from file descriptor #3
```