DEPARTMENT OF INFORMATION TECHNOLOGY

LAB MANUAL

U23ITP41 OPERATING SYSTEMS LABORATORY

REGUALTION:2023

Prepared By **Approved By**

AP/ IT **HOD / IT**

LIST OF EXPERIMENTS

1. Basics of UNIX commands
2. Write programs using the following system calls of UNIX operating system fork, exec, getpid, exit, wait, close, stat, opendir, readdir
3. Write C programs to simulate UNIX commands like cp, ls, grep, etc.
4. Shell Programming
5. Write C programs to implement the various CPU Scheduling Algorithms
6. Implementation of Semaphores
7. Implementation of Shared memory and IPC
8. Bankers Algorithm for Deadlock Avoidance
9. Implementation of Deadlock Detection Algorithm
10. Write C program to implement Threading & Synchronization Applications
11. Implementation of the following Memory Allocation Methods for fixed partition
    a) First Fit b) Worst Fit c) Best Fit
12. Implementation of Paging Technique of Memory Management
13. Implementation of the following Page Replacement Algorithms
    a) FIFO b) LRU c) LFU
14. Implementation of the various File Organization Techniques
15. Implementation of the following File Allocation Strategies
    a) Sequential b) Indexed c) Linked

TOTAL: 60 PERIODS

BIBLIOGRAPHY

TEXT/REFERENCE BOOKS:

- Abraham Silberschatz, Peter Baer Galvin and Greg Gagne, "Operating System Concepts",9th Edition, John Wiley and Sons Inc.
- Operating System Concepts, 10th Edition Abraham Silberschatz, Greg Gagne, Peter B. Galvin. April 2018.
- Ramaz Elmasri, A. Gil Carrick, David Levine, "Operating Systems – A Spiral Approach", Tata McGraw Hill Edition, 2010.
- Achyut S.Godbole, Atul Kahate, "Operating Systems", Mc Graw Hill Education, 2016.
- Gary Nutt, "Operating Systems", Third Edition, Pearson Education, 2004.

Ex.No:1 **BASICS OF UNIX COMMAND**

Aim:

To study and execute the basic UNIX commands

Description:

Command	Example	Description
1. **ls**	ls ls -alF	Lists files in current directory List in long format
2. **cd**	cd tempdir cd .. cd ~dhyatt/web-docs	Change directory to tempdir Move back one directory Move into dhyatt's web-docs directory
3. **mkdir**	mkdir graphics	Make a directory called graphics
4. **rmdir**	rmdir emptydir	Remove directory (must be empty)
5. **cp**	cp file1 web-docs cp file1 file1.bak	Copy file into directory Make backup of file1
6. **rm**	rm file1.bak rm *.tmp	Remove or delete file Remove all file
7. **mv**	mv old.html new.html	Move or rename files
8. **more**	more index.html	Look at file, one page at a time
9. **lpr**	lpr index.html	Send file to printer
10. **man**	man ls	Online manual (help) about command

Files and Directories:

Command	Description
Cat	Display File Contents
Cd	Changes Directory to dirname
Chgrp	change file group
Chmod	Changing Permissions
Cp	Copy source file into destination

File	Determine file type
Find	Find files
Grep	Search files for regular expressions.
Head	Display first few lines of a file
Ln	Create softlink on oldname
Ls	Display information about file type.
Mkdir	Create a new directory dirname
More	Display data in paginated form.
Mv	Move (Rename) a oldname to newname.
Pwd	Print current working directory.
Rm	Remove (Delete) filename
Rmdir	Delete an existing directory provided it is empty.

Result

Thus the basic UNIX commands were studied

Ex: No: 2a **SYSTEM CALLS – FORK, WAIT AND EXIT**

Aim:

To write a program in C to implement the system calls fork, wait and exit.

Algorithm:

Step 1: Start the program.
Step 2: Declare the variables to store the status value and fork return value.
Step 3: Check the fork return value.
Step 4: If fork_return==0, then the child process is created.
Step 5: If fork_return== -1, then the creation of child process is failed.
Step 6: Display the status values.
Step 7: Stop the program.

Program:

```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
int main()
{
int status;
pid_t fork_return;
fork_return=fork();
if(fork_return==0)
{
printf("\n I'm the child!!");
exit(0);
}
else
{
wait(&status);
printf("\n I'm the parent!!");
printf("\n Child returned:%d\n",status);
}
return 0;
}
```

Output:

```
[exam127@redhat exam127]$ cc anufork.c
[exam127@redhat exam127]$ ./a.out

 I'm the child!!
 I'm the parent!!
 Child returned: 0
```

Result:

Thus the implementation of system calls fork exit and wait has been executed successfully

Ex: No: 2b **SYSTEM CALLS - GETPID**

Aim:

To write a program in C to implement the system calls getpid.

Algorithm:

Step 1: Start the process.
Step 2: Invoke the built in function getpid() which will return current process identification number.
Step 3: Display the process ID.
Step 4: Stop the program.

Program:

```
#include<stdio.h>
#include<unistd.h>
int main()
{
fork();
printf("\n Hello,I am the process with ID=%d\n",getpid());
return 0;
}
```

Output:

```
[exam127@redhat exam127]$ cc anugetpaid.c
[exam127@redhat exam127]$ ./a.out

 Hello, I am the process with ID=27995

 Hello, I am the process with ID=27994
```

Result:

Thus the implementation of system calls Getpid has been executed successfully

Ex: No: 2c **SYSTEM CALLS - OPEN DIR READ DIR CLOSE DIR**

Aim:

To write a program in C to implement the system call opendir (), readdir () and closedir ().

Algorithm:

Step 1: Start the program.
Step 2: The function opendir() is invoked to open the directory which contains name of the directory a parameter.
Step 3: The content of the directory can be read using the function readdir().
Step 4: It contains the name of the directory as a parameter.
Step 5: Display the content of the directory.
Step 6: The directory is closed using the function closedir().
Step 7: Stop the program.

Program:

```
#include<sys/types.h>
#include<dirent.h>
#include<sys/stat.h>
#include<stdio.h>
void traverse(char *fn)
{
DIR *dir;
struct dirent *entry;
char path[1025];
struct stat info;
printf("%s\n",fn);
if((dir=opendir(fn))==NULL)
printf("error");
else
{
while((entry=readdir(dir))!=NULL)
{
if((entry->d_name[0])!='.')
{
strcpy(path,fn);
strcat(path,"/");
strcat(path,(entry->d_name));
if(stat(path,&info)!=0)
printf("error");
else if(S_ISDIR(info.st_mode))
traverse(path);
}
}
closedir(dir);
}
}
main()
{
printf("directory structure");
```

```
traverse("/home/exam127/flower");
}
```

Output:

```
[exam127@redhat exam127]$ cc anuopen.c
[exam127@redhat exam127]$ ./a.out

directory structure/home/exam127/flower
/home/exam127/flower/lotus
/home/exam127/flower/rose
/home/exam127/flower/lilly
```

Result:

Thus the implementation of system calls Opendir(), Readdir() and Closedir() has been executed successfully

Ex: No: 2d **SYSTEM CALLS - STAT**

Aim:

To write a program in C to implement the system call stat.

Algorithm:

Step 1: Start the program.
Step 2: Enter the filename.
Step 3: The information about the given file such as size, number of links, file inode, file permission and symbolic link will be displayed by the member of structure stat.
Step 4: Stop the program.

Program:

```
#include<unistd.h>
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
int main(int argc,char **argv)
{
if(argc!=2)
return 1;
struct stat fileStat;
if(stat(argv[1],&fileStat)<0)
return 1;
printf("Information for %s \n",argv[1]);
printf("_____");
printf("\n Files size:\t\t %d\n",fileStat.st_size);
printf("Number of links:\t\t %d\n",fileStat.st_nlink);
printf("file inode:\t\t%d\n",fileStat.st_ino);
printf("File PERMISSION:\t");
printf((S_ISDIR(fileStat.st_mode))?"d":"_");
printf((fileStat.st_mode &S_IRUSR)?"r":"_");
printf((fileStat.st_mode &S_IWUSR)?"w":"_");
printf((fileStat.st_mode &S_IXUSR)?"x":"_");
printf((fileStat.st_mode &S_IRGRP)?"r":"_");
printf((fileStat.st_mode &S_IWGRP)?"w":"_");
printf((fileStat.st_mode &S_IXGRP)?"x":"_");
printf((fileStat.st_mode &S_IROTH)?"r":"_");
printf((fileStat.st_mode &S_IWOTH)?"w":"_");
printf((fileStat.st_mode &S_IXOTH)?"x":"_");
printf("\n\n");
return 0;}
```

Output:

```
[exam127@redhat exam127]$ cc anustat.c
[exam127@redhat exam127]$ ./a.out anustat.c
Information for anustat.c
_____
 Files size:          943
Number of links:          1
file inode:          896255
```

File PERMISSION: rw_rw_r_____

[exam127@redhat exam127]$ cc anustat.c
[exam127@redhat exam127]$ ./a.out dad
Information for dad

Files size: 55
Number of links: 1
file inode: 896241
File PERMISSION: rw_rw_r_____

Result:

Thus the implementation of system calls STAT has been executed successfully

Ex: No: 2e **SYSTEM CALLS - EXECUTION**

Aim:

To write a program in C to implement the system call execution.

Algorithm:

Step 1: Start the program.
Step 2: Invoke the built in function execlp() which will execute the ls command for the particular user.
Step 3: Display all directories.
Step 4: Stop the program.

Program:

```
#include<stdio.h>
#include<sys/types.h>
int main()
{
printf("\n child process\n");
execlp("/bin/ls","ls",0);
sleep(5);
return 1;
}
```

Output:

```
[exam127@redhat exam127]$ cc child.c
[exam127@redhat exam127]$ ./a.out

 child process
add.c        anuopen.c  a.out    equal.c  fruits  number  ss      veg
anufork.c    anustat2.c child.c  equal.v  moon    numbers star
anugetpaid.c anustat.c  dad      flower   new     open.c  swap.c
```

Result:

Thus the implementation of system calls EXECUTION has been executed successfully

Ex: No: 2f **SYSTEM CALLS - READ**

Aim:

To write a program in C to implement the system call read.

Algorithm:

Step 1: Start the program.
Step 2: Read the content from file using read() function.
Step 3: If there is no content in the file, then stop the execution.
Step 4: Otherwise display the read content.
Step 5: Stop the program.

Program:

```
#include<unistd.h>
#include<fcntl.h>
int main()
{
char data[128];
write(2,"An error occured int the read.\n",31);
else
printf("Sucessfully read");
exit(0);
}
```

Output:

```
[exam127@redhat exam127]$ cc code.c
[exam127@redhat exam127]$. /a.out
Welcome
Sucessfully read.
```

Result:

Thus the implementation of system calls Read () has been executed successfully

Ex No:3 **SYSTEM CALLS - GREP AND LS COMMAND**

Aim :

To Write a program for GREP $ LS COMMAND in UNIX platform

Algorithm:

Step:1 Start the program
Step:2 Declare the variables.
Step 3:. Get your choice.
Step 4: if your choice is 1 means the grep command is executed.
Step 5:.if your choice is 2 means the LS command is executed.
Step 6: if your choice is 3means the exit the process.
Step 8: stop the program.

Program

```
#include<stdio.h>
main()
{
int i,ch;
system("clear");
s: printf("\n\n\n\t program to simulate UNIX using ls and grep command \n");
printf("\n\n\n 1. Display the extracted lines from the files with the grep");
printf("\n\n\n 2. Display lost list information about files and directories");
printf("\n\n\n 3.exit");
printf("\n\n enter your choice ");
scanf("%d",&ch);
if(ch==1)
{
system("grep printf fork.c");
goto s;
}
if(ch==2)
{

system("ls -l");
goto s;
}
else
{exit(0);
} }
```

Output

Program to simulate UNIX using is and grep command

1. Display the extracted lines from the files with the grep

2. Display lost list information about files and directories

3.Exit

Enter your choice

Printf(“rajesh\n”);
Printf(“rajesh\n”);
Printf (“rocky\n”);

1. Display the extracted lines from the files with the grep

2. Display lost list information about files and directories

3. exit
enter your choice 2
total 112

-rw-rw-r-- 1 gopi gopi 60 Feb 24 10:57 add1.sh

-rw-rw-r-- 1 gopi gopi 78 Feb 24 10:55 add.sh

-rw-rw-r-- gopi gopi 5446Mar 22 15:34a.out

-rw-rw-r-- 1 gopi gopi 15 Jan 11 15:27 execll.c

-rw-rw-r-- 1 gopi gopi 220 Dec 16 14:41 exit

Program to simulate UNIX using is and grep command

1.Display the extracted lines from the files with the grep

2.Display lost list information about files and directories
3.exit
enter your choice 3

4.cp

Use cp to copy files or directories. % cp foo foo.2
This makes a copy of the file foo. % cp ~/poems/jabber .date
Use this command to check the date and time. % date
Fri Nov 6 08:52:42 MST 2018

Result:

Thus the implementation of system calls GREP () and LS () command has been executed successfully

Ex.No:4 **SHELL PROGRAMMING**

Aim

To implement the shell programming

Description

The shell provides an interface to the UNIX system. It gathers input from user and executes programs based on that input. A shell is an environment in which we can run our commands, programs, and shell scripts. There are different flavors of shells, just as there are different flavors of operating systems. Shell Types are
The Bourne shell. If you are using a Bourne-type shell, the default prompt is the $ character.
The C shell. If you are using a C-type shell, the default prompt is the % character.

FACTORIAL OF N NUMBER

```
echo "Total no of factorial wants"
read fact

ans=1
counter=0
while [ $fact -ne $counter ]
do
      counter=`expr $counter + 1`
      ans=`expr $ans \* $counter`
done
echo "Total of factorial is $ans"
```

OUTPUT:

```
Enter the number: 5
The Factorial is: 120
echo "Enter a number: "
read num
i=2

while [ $i -lt $num ]
do
 if [ `expr $num % $i` -eq 0 ]
 then
    echo "$num is not a prime number"
    echo "Since it is divisible by $i"
    exit
 fi
 i=`expr $i + 1`
done

echo "$num is a prime number "
```

Output:

Enter the number:7
7 is prime number

Result

Thus the basic shell programming was implemented

Ex.No: 5a ROUND ROBIN SCHEDULING ALGORITHM

Aim:

To simulate the CPU scheduling algorithm round-robin.

Algorithm:

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue and time quantum (or) time slice

Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time

Step 4: Calculate the no. of time slices for each process where No. of time slice for process (n) = burst time process(n)/time slice

Step 5: If the burst time is less than the time slice then the no. of time slices =1.

Step 6: Consider the ready queue is a circular Q, calculate

(a) Waiting time for process (n) = waiting time of process(n-1)+ burst time of process(n-1) + the time difference in getting the CPU from process(n-1)

(b) Turnaround time for process(n) = waiting time of process(n) + burst time of process(n)+ the time difference in getting CPU from process(n).

Step 7: Calculate

(c) Average waiting time = Total waiting Time / Number of process

(d) Average Turnaround time = Total Turnaround Time / Number of process

Step 8: Stop the process

Program :

```
/* Round Robin Scheduling Algorithm*/
 #include<stdio.h>
main()
{
int s[10],p[10],n,i,j,w1=0,w[10],t[10], st[10],tq,tst=0;
int tt=0,tw=0;
float aw,at;
printf("Enter no.of processes \n");
scanf("%d",&n);
printf("\n Enter the time quantum \n");
scanf("%d",&tq);
printf("\n Enter the process & service time of each process separated by a space \n");
for(i=0;i<n;i++)
scanf("%d%d",&p[i],&s[i]);
for(i=0;i<n;i++)
{
st[i]=s[i];
tst=tst+s[i];
}
for(j=0;j<tst;j++)
for(i=0;i<n;i++)
{
if(s[i]>tq)
{
s[i]=s[i]-tq;
w1=w1+tq;
t[i]=w1;
w[i]=t[i]-st[i];
}
else if(s[i]!=0)
{
w1=w1+tq;
```

```
t[i]=w1;
w[i]=t[i]-st[i];
s[i]=s[i]-tq;
}
}
for(i=0;i<n;i++)
{
tw=tw+w[i];
tt=tt+t[i];
}
aw=tw/n;
at=tt/n;
printf("process\tst\twt\ttt \n");
for(i=0;i<n;i++)
printf("%d\t%d\t%d\t%d \n",p[i],st[i],w[i],t[i]);
printf("awt=%f\n",aw);
printf("att=%f\n",at);

}
```

OUTPUT:

```
Enter Number of process: 3
Enter the Time Quantum: 3
Enter the process and service time of each process separated by a space
1   3       2       6       3       9
Process     st      wt      tt
1           3       0       3
2           6       6       12
3           9       9       18
Awt=5.000000
Att=11.000000
```

Result:

Thus the Round Robin CPU scheduling algorithm was implemented and average waiting time, average turnaround time was computed.

Ex.No: 5b SHORTEST JOB FIRST SCHEDULING ALGORITHM

Aim:

To write a C program to simulate the shortest job first CPU scheduling algorithm.

Algorithm:

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue

Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time and process arrival time.

Step 4: Start the Ready Q according the shortest Burst time by sorting according to lowest to highest burst time.

Step 5: Set the waiting time of the first process as _0,, and its turnaround time as its burst time.

Step 6: Sort the processes names based on their Burt time and Arrival time

Step 7: For each process in the ready queue, calculate

a). Waiting timess(n)= waitingtime (n-1) + Burst time (n-1)-arrivaltime(n)

b). Turnaround time (n)= waiting time(n)+Burst time(n)

Step 8: Calculate and print the results

(a) Average waiting time = Total waiting Time / Number of process

(b) Average Turnaround time = Total Turnaround Time / Number of process

Step 9: Stop the process

Program :

```
/* SJF */
#include<stdio.h>
main()
{
int s[10],p[10],n,i,j,w1=0,w[10],t[10], st[10],tq,tst=0;
int tt=0,tw=0;
float aw,at;
clrscr();
printf("Enter no.of processes \n");
scanf("%d",&n);
printf("\n Enter the time quantum \n");
scanf("%d",&tq);
printf("\n Enter the process & service time of each process separated by a space \n");
for(i=0;i<n;i++)
scanf("%d%d",&p[i],&s[i]);
for(i=0;i<n;i++)
{
st[i]=s[i];
tst=tst+s[i];
}
for(j=0;j<tst;j++)
for(i=0;i<n;i++)
{
if(s[i]>tq)
{
s[i]=s[i]-tq;
w1=w1+tq;
t[i]=w1;
w[i]=t[i]-st[i];
}
else if(s[i]!=0)
{
```

```
w1=w1+tq;
t[i]=w1;
w[i]=t[i]-st[i];
s[i]=s[i]-tq;
}
}
for(i=0;i<n;i++)
{
tw=tw+w[i];
tt=tt+t[i];
}
aw=tw/n;
at=tt/n;
printf("process\tst\twt\ttt \n");
for(i=0;i<n;i++)
printf("%d\t%d\t%d\t%d \n",p[i],st[i],w[i],t[i]);
printf("awt=%f\n",aw);
printf("att=%f\n",at);
getch();
}
```

Output:

```
Enter number of process
3
 Enter the Burst Time of Process 04
 Enter the Burst Time of Process 13
 Enter the Burst Time of Process 25
 SHORTEST JOB FIRST SCHEDULING ALGORITHM
 PROCESS ID     BURST TIME     WAITING TIME    TURNAROUND TIME
   1         3               0           3
   0         4               3           7
   2         5               7          12
 AVERAGE WAITING TIME 3.33
 AVERAGE TURN AROUND TIME 7.33
```

Result:

Thus the Shortest Job First (SJF) CPU scheduling algorithm was implemented and average waiting time, average turnaround time was computed.

Ex.No: 5c FIRST COME FIRST SERVE SCHEDULING ALGORITHM

Aim:

To write a c program to simulate the cpu scheduling algorithm First Come First Serve (FCFS)

Algorithm:

Step 1: Start the process
Step 2: Accept the number of processes in the ready Queue
Step 3: For each process in the ready Q, assign the process name and the burst time
Step 4: Set the waiting of the first process as _0„and its burst time as its turnaround time
Step 5: for each process in the Ready Q calculate
a). Waiting timess(n)= waiting time (n-1) + Burst time (n-1)
b). Turnaround time (n)= waiting time(n)+Burst time(n)
Step 6: Calculate
(a) Average waiting time = Total waiting Time / Number of process
(b) Average Turnaround time = Total Turnaround Time / Number of process
Step 7: Stop the process

Program :

```
/* FCFS */
#include<stdio.h>
void main()
{
 int i,n,sum,wt,tat,twt,ttat;
 int t[10];
 float awt,atat;
 clrscr();
 printf("Enter number of processors:\n");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
  printf("\n Enter the Burst Time of the process %d",i+1);
  scanf("\n %d",&t[i]);
 }
 printf("\n\n FIRST COME FIRST SERVE SCHEDULING ALGORITHM \n");
 printf("\n Process ID \t Waiting Time \t Turn Around Time \n");
 printf("1 \t\t 0 \t\t %d \n",t[0]);
 sum=0;
 twt=0;
 ttat=t[0];
 for(i=1;i<n;i++)
 {
  sum+=t[i-1];
  wt=sum;
  tat=sum+t[i];
  twt=twt+wt;
  ttat=ttat+tat;
  printf("\n %d \t\t %d \t\t %d",i+1,wt,tat);
  printf("\n\n");
 }
  awt=(float)twt/n;
  atat=(float)ttat/n;
  printf("\n Average Waiting Time %4.2f",awt);
  printf("\n Average Turnaround Time %4.2f",atat);
  getch();
}
```

Output:

```
Enter number of processors:
3
Enter the Burst Time of the process 1: 2
Enter the Burst Time of the process 2: 5
Enter the Burst Time of the process 3:  4
FIRST COME FIRST SERVE SCHEDULING ALGORITHM
Process ID      Waiting Time    Turn Around Time
        1            0                2
        2            2                7
        3            7                11
Average Waiting Time 3.00
Average Turnaround Time 6.67
```

Result:

Thus the FCFS CPU scheduling algorithm was implemented and average waiting time, average turnaround time was computed.

Ex.No: 5d PRIORITY SCHEDULING ALGORITHM

Aim:

To write a c program to simulate the priority CPU scheduling algorithm.

Algorithm:

Step 1: Start the process
Step 2: Accept the number of processes in the ready Queue
Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time
Step 4: Sort the ready queue according to the priority number.
Step 5: Set the waiting of the first process as _0,, and its burst time as its turnaround time
Step 6: Arrange the processes based on process priority
Step 7:For each process in the Ready Q calculate
Step 8: for each process in the Ready Q calculate
a). Waiting timess(n)= waiting time (n-1) + Burst time (n-1)
b). Turnaround time (n)= waiting time(n)+Burst time(n)
Step 9: Calculate
(a) Average waiting time = Total waiting Time / Number of process
(b) Average Turnaround time = Total Turnaround Time / Number of process
Print the results in an order.
Step 10: Stop the process

Program :

```
/* Priority */
#include<stdio.h>
main()
{
int i,j,bt[10],n,pt[10],wt[10],tt[10],t,k,l,w1=0,t1=0,b=0,p=0;
float at,aw;
clrscr();
printf("enter no of jobs");
scanf("%d",&n);
printf("enter burst time");
for(i=0;i<n;i++)
{
scanf("%d",&b);
bt[i]=b;
}
printf("enter priority values");
for(i=0;i<n;i++)
{
scanf("%d",&p);
pt[i]=p;
}
for(i=1;i<n;i++)
for(j=0;j<n-i;j++)
if(pt[j]<pt[j+1])
{
t=pt[j];
pt[j]=pt[j+1];
pt[j+1]=t;
k=bt[j];
bt[j]=bt[j+1];
bt[j+1]=k;
}
wt[0]=0;
```

```
for(i=0;i<n;i++)
{
wt[i+1]=bt[i]+wt[i];
tt[i]=bt[i]+wt[i];
w1=w1+wt[i];
t1=t1+tt[i];
}
aw=w1/n;
at=t1/n;
printf("\nbt\tprority\twt\ttt\n");
for(i=0;i<n;i++)
printf("%d\t%d\t%d\t%d\n",bt[i],pt[i],wt[i],tt[i]);
printf("aw=%f\nat=%f",aw,at);
getch();
}
```

Output

```
Enter no of jobs: 3
Enter burst time:  10 11 12
Enter priority values: 3 2 1
Bt      priority        wt      tt
10      3               0       10
11      2               10      21
12      1               21      33
Aw=10.000000
At=21.000000
```

Result:

Thus the Priority CPU scheduling algorithm was implemented and average waiting time, average turnaround time was computed.

Ex.No: 6 IMPLEMENT SEMAPHORE - DINING PHILOSOPHER

Aim:

To implement dining philosopher problem using semaphores.

Algorithm :

Step 1: There are N philosophers meeting around a table, eating spaghetti and talking about philosophy.
Step 2: There are only N forks available such that only one fork between each philosopher.
Step 3: There are only 5 philosophers and each one requires 2 forks to eat.
Step 4: A solution to the problem is to ensure that at most number of philosophers can eat Spaghetti at once.

Program :

```
#include<stdio.h>
 #define n 4
 int compltedPhilo = 0,i;
struct fork{
   int taken;
}ForkAvil[n];
 struct philosp{
   int left;
   int right;
}Philostatus[n];
void goForDinner(int philID){
int otherFork = philID-1; //same like threads concept here cases implemented
   if(Philostatus[philID].left==10 && Philostatus[philID].right==10)
      printf("Philosopher %d completed his dinner\n",philID+1);
   //if already completed dinner
   else if(Philostatus[philID].left==1 && Philostatus[philID].right==1){
         //if just taken two forks
         printf("Philosopher %d completed his dinner\n",philID+1);
          Philostatus[philID].left = Philostatus[philID].right = 10; //remembering that he completed dinner by
assigning value 10
          if(otherFork== -1)
            otherFork=(n-1);
          ForkAvil[philID].taken = ForkAvil[otherFork].taken = 0; //releasing forks
         printf("Philosopher %d released fork %d and fork %d\n",philID+1,philID+1,otherFork+1);
         compltedPhilo++;
      }
      else if(Philostatus[philID].left==1 && Philostatus[philID].right==0){ //left already taken, trying for
right fork
            if(philID==(n-1)){
               if(ForkAvil[philID].taken==0){ //KEY POINT OF THIS PROBLEM, THAT LAST
PHILOSOPHER TRYING IN reverse DIRECTION
                  ForkAvil[philID].taken = Philostatus[philID].right = 1;
                  printf("Fork %d taken by philosopher %d\n",philID+1,philID+1);
               }else{
                  printf("Philosopher %d is waiting for fork %d\n",philID+1,philID+1);
               }
            }else{ //except last philosopher case
               int dupphilID = philID;
               philID-=1;

               if(philID== -1)
                  philID=(n-1);

               if(ForkAvil[philID].taken == 0){
```

```
                ForkAvil[philID].taken = Philostatus[dupphilID].right = 1;
                printf("Fork %d taken by Philosopher %d\n",philID+1,dupphilID+1);
            }else{
                printf("Philosopher %d is waiting for Fork %d\n",dupphilID+1,philID+1);
            }
          }
        }
        else if(Philostatus[philID].left==0){ //nothing taken yet
            if(philID==(n-1)){
                if(ForkAvil[philID-1].taken==0){ //KEY POINT OF THIS PROBLEM, THAT LAST
PHILOSOPHER TRYING IN reverse DIRECTION
                    ForkAvil[philID-1].taken = Philostatus[philID].left = 1;
                    printf("Fork %d taken by philosopher %d\n",philID,philID+1);
                }else{
                    printf("Philosopher %d is waiting for fork %d\n",philID+1,philID);
                }
            }else{ //except last philosopher case
                if(ForkAvil[philID].taken == 0){
                    ForkAvil[philID].taken = Philostatus[philID].left = 1;
                    printf("Fork %d taken by Philosopher %d\n",philID+1,philID+1);
                }else{
                    printf("Philosopher %d is waiting for Fork %d\n",philID+1,philID+1);
                }
            }
      }else{}
}

int main(){
   for(i=0;i<n;i++)
      ForkAvil[i].taken=Philostatus[i].left=Philostatus[i].right=0;

   while(compltedPhilo<n){
      /* Observe here carefully, while loop will run until all philosophers complete dinner
      Actually problem of deadlock occur only thy try to take at same time
      This for loop will say that they are trying at same time. And remaining status will print by go for dinner
function
      */
      for(i=0;i<n;i++)
         goForDinner(i);
      printf("\nTill now num of philosophers completed dinner are %d\n\n",compltedPhilo);
   }

   return 0;
}
```

Output

```
Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 3 completed his dinner
Philosopher 3 released fork 3 and fork 2
Fork 3 taken by philosopher 4

Till now num of philosophers completed dinner are 3

Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 3 completed his dinner
Fork 4 taken by philosopher 4

Till now num of philosophers completed dinner are 3

Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 3 completed his dinner
Philosopher 4 completed his dinner
Philosopher 4 released fork 4 and fork 3

Till now num of philosophers completed dinner are 4
```

Result

Thus the program for demonstrating Dining-Philosopher problem was implemented

Ex. No: 7 **Implementation of Shared memory and IPC**

Date:

Aim:
To develop a client-server application program, this uses shared memory using IPC

Algorithm:

Client :

1. Define the key to be 5600
2. Attach the client to the shared memory created by the server.
3. Read the content from the shared memory.
4. Display the content on the screen.
5. stop

Server:

6. Define shared memory size of 30 bytes
7. Define the key to be 5600
8. Create a shared memory using shmget () system calls and gets the shared memory id in variable shmid.
9. Attach the shared memory to server data space
10. Get the content to be placed in the shared memory from the user of the server.
11. Write the content in the shared memory, which will read out by the client.
12. stop

Program :

Shared memory and IPC

SHARED MEMORY

```
//SHMServer.C
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <stdlib.h>
#define MAXSIZE     27
void die(char *s)
{
   perror(s);
   exit(1);
}
int main()
{
   char c;
   int shmid;
   key_t key;
   char *shm, *s;
   key = 5678;
   if ((shmid = shmget(key, MAXSIZE, IPC_CREAT | 0666)) < 0)
      die("shmget");
   if ((shm = shmat(shmid, NULL, 0)) == (char *) -1)
      die("shmat");
   s = shm;
   for (c = 'a'; c <= 'z'; c++)
      *s++ = c;
   while (*shm != '*')
      sleep(1);
   exit(0);
}
```

//SHMClient.C

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <stdlib.h>
#define MAXSIZE     27
void die(char *s)
{
   perror(s);
   exit(1);
}
int main()
{
   int shmid;
   key_t key;
   char *shm, *s;
   key = 5678;
   if ((shmid = shmget(key, MAXSIZE, 0666)) < 0)
      die("shmget");
   if ((shm = shmat(shmid, NULL, 0)) == (char *) -1)
      die("shmat");
   for (s = shm; *s != '\0'; s++)
      putchar(*s);
   putchar('\n');
   *shm = '*';
   exit(0);
}
```

Output:

```
[gokul@localhost ~]$ ipcs -m

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x0000162e 98307      gokul      666        27         1

[gokul@localhost ~]$ cc shmclient1.c
[gokul@localhost ~]$ ./a.out
abcdefghijklmnopqrstuvwxyz
```

Result :

Thus the above program executed successfully.

Ex. No: 8 BANKER'S ALGORITHM FOR DEADLOCK AVOIDANCE

Aim:

To Simulate 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 it‟s possible to allocate.
Step 6: If it is possible then the system is in safe state.
Step 7: Else system is not in safety state.
Step 8: If the new request comes then check that the system is in safety or not if we allow the request.
Step 9: Stop the program.

Program :

```
#include<stdio.h>
 main()
{
int r[1][10],av[1][10];
int all[10][10],max[10][10],ne[10][10],w[10],safe[10];
int i=0,j=0,k=0,l=0,np=0,nr=0,count=0,cnt=0;
printf("enter the number of processes in a system");
scanf("%d",&np);
printf("enter the number of resources in a system");
scanf("%d",&nr);
for(i=1;i<=nr;i++){
printf("\n enter the number of instances of resource R%d " ,i);
scanf("%d",&r[0][i]);
av[0][i]=r[0][i];
}
for(i=1;i<=np;i++)
for(j=1;j<=nr;j++)
all[i][j]=ne[i][j]=max[i][j]=w[i]=0;
printf(" \nEnter the allocation matrix");
for(i=1;i<=np;i++)
{ printf("\n");
for(j=1;j<=nr;j++)
{
scanf("%d",&all[i][j]);
av[0][j]=av[0][j]-all[i][j];
}
}
printf("\nEnter the maximum matrix");
for(i=1;i<=np;i++)
{ printf("\n");
for(j=1;j<=nr;j++)
{
scanf("%d",&max[i][j]);
}
}
for(i=1;i<=np;i++)
{
for(j=1;j<=nr;j++)
{
ne[i][j]=max[i][j]-all[i][j];
```

```
}
}
for(i=1;i<=np;i++)
{
printf("pocess P%d",i);
for(j=1;j<=nr;j++)
{
printf("\n allocated %d\t",all[i][j]);
printf("maximum %d\t",max[i][j]);
printf("need %d\t",ne[i][j]);
}
printf("\n_____\n");
}
printf("\nAvailability");
for(i=1;i<=nr;i++)
printf("R%d %d\t",i,av[0][i]);

printf("\n_____");
printf("\n safe sequence");
for(count=1;count<=np;count++)
{
for(i=1;i<=np;i++)
{ cnt=0;
for(j=1;j<=nr;j++)
{
if(ne[i][j]<=av[0][j] && w[i]==0)
cnt++;
}
if(cnt==nr)
{
k++;
safe[k]=i;
for(l=1;l<=nr;l++)
av[0][l]=av[0][l]+all[i][l];
printf("\n P%d ",safe[k]);
printf("\t Availability");
for(l=1;l<=nr;l++)
printf("R%d %d\t",l,av[0][l]);
w[i]=1;
}
}
}
}
```

Output:

```
Enter the number of resources in a system 4
Enter the number of instances of resource R1 2
Enter the number of instances of resource R2 2
Enter the number of instances of resource R3 2
Enter the number of instances of resource R4 2
Enter the allocation matrix
1 0 1 1 0

1 1 0 0 0
0 0 0 1 0
0 0 0 0 0
```

Enter the maximum matrix
1 0 1 0 0
1 0 0 0 0
0 0 0 0 1
pocess P1
allocated 1 maximum 0 need -1
allocated 0 maximum 0 need 0
allocated 1 maximum 0 need -1
allocated 1 maximum 0 need -1

---

pocess P2
allocated 0 maximum 1 need 1
allocated 1 maximum 0 need -1
allocated 1 maximum 1 need 0
allocated 0 maximum 0 need 0

---

pocess P3
allocated 0 maximum 0 need 0
allocated 0 maximum 1 need 1
allocated 0 maximum 0 need 0
allocated 0 maximum 0 need 0

---

pocess P4
allocated 0 maximum 0 need 0
allocated 1 maximum 0 need -1
allocated 0 maximum 0 need 0
allocated 0 maximum 0 need 0

AvailabilityR1 1 R2 0 R3 0 R4 1

---

safe sequence
P1 AvailabilityR1 2 R2 0 R3 1 R4 2
P2 AvailabilityR1 2 R2 1 R3 2 R4 2
P3 AvailabilityR1 2 R2 1 R3 2 R4 2
P4 AvailabilityR1 2 R2 2 R3 2 R4 2

Result:

Thus the banker‟s algorithm for deadlock avoidance was simulated.

Ex. No: 9 IMPLEMENT AN ALGORITHM FOR DEAD LOCK DETECTION

Aim:

To implement an algorithm for deadlock detection.

Algorithm :

Simply detects the existence of a Cycle:

1. Start at any vertex finds all its immediate neighbors.
2. From each of these find all immediate neighbors, etc.
3. Until a vertex repeats (there is a cycle) or one cannot continue (there is no cycle).
4. Stop.

On a copy of the graph:

1. See if any Processes NEEDs can all be satisfied.
2. If so satisfy the needs with holds and remove that Process and all the Resources it holds from the graph.
3. If any Process are left Repeat step a
4. If all Processes are finally removed by this procedure there is no Deadlock in the original graph, if not there is.
5. Stop.

Program :

```
#include<stdio.h>
static int mark[20];
int i,j,np,nr;

int main()
{
int alloc[10][10],request[10][10],avail[10],r[10],w[10];

printf("\nEnter the no of process: ");
scanf("%d",&np);
printf("\nEnter the no of resources: ");
scanf("%d",&nr);
for(i=0;i<nr;i++)
{
printf("\nTotal Amount of the Resource R%d: ",i+1);
scanf("%d",&r[i]);
}
printf("\nEnter the request matrix:");
for(i=0;i<np;i++)
for(j=0;j<nr;j++)
scanf("%d",&request[i][j]);

printf("\nEnter the allocation matrix:");
for(i=0;i<np;i++)
for(j=0;j<nr;j++)
scanf("%d",&alloc[i][j]);
/*Available Resource calculation*/
for(j=0;j<nr;j++)
{
avail[j]=r[j];
for(i=0;i<np;i++)
{
avail[j]-=alloc[i][j];

}
}
```

```
//marking processes with zero allocation
for(i=0;i<np;i++)
{
int count=0;
 for(j=0;j<nr;j++)
   {
     if(alloc[i][j]==0)
       count++;
     else
       break;
   }
 if(count==nr)
 mark[i]=1;
}
// initialize W with avail
for(j=0;j<nr;j++)
   w[j]=avail[j];
//mark processes with request less than or equal to W
for(i=0;i<np;i++)
{
int canbeprocessed=0;
 if(mark[i]!=1)
{
  for(j=0;j<nr;j++)
   {
     if(request[i][j]<=w[j])
     canbeprocessed=1;
     else
        {
        canbeprocessed=0;
        break;
         }
    }
if(canbeprocessed)
{
mark[i]=1;

for(j=0;j<nr;j++)
w[j]+=alloc[i][j];
}
}
}
//checking for unmarked processes
int deadlock=0;
for(i=0;i<np;i++)
if(mark[i]!=1)
deadlock=1;
if(deadlock)
printf("\n Deadlock detected");
else
printf("\n No Deadlock possible");
}
```

Output:

```
Enter the no of process: 4
Enter the no of resources: 5

Total Amount of the Resource R1: 2
Total Amount of the Resource R2: 1
Total Amount of the Resource R3: 1
Total Amount of the Resource R4: 2
Total Amount of the Resource R5: 1

Enter the request matrix:0 1 0 0 1
0 0 1 0 1
0 0 0 0 1
1 0 1 0 1

Enter the allocation matrix:1 0 1 1 0
1 1 0 0 0
0 0 0 1 0
0 0 0 0 0

 Deadlock detected
```

Result:

Thus the deadlock detection algorithm was implemented.

Ex. No: 10 IMPLEMENT THREADING & SYNCHRONIZATION APPLICATIONS

Aim:

To implement threading & synchronization applications using c

Algorithm :

1. Start the program
2. Read the Input
3. Allocate the memory
4. Process the input
5. Checking error
6. Print result

Program :

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

Output :

```
        gokul@localhost ~]$ cc filename.c -lpthread
Job 1 has started
Job 2 has started
Job 2 has finished
Job 2 has finished
```

Result

Thus the threading and synchronization concept was implemented

Ex: No: 11a IMPLEMENTATION OF FIRST FIT ALGORITHM

Aim:

To write a program to implement first fit algorithm for memory management.

Algorithm:

Step1: Start the process.
Step 2: Declare the size.
Step 3: Get the number of processes to be inserted.
Step 4: Allocate the first hole that is big enough for searching.
Step 5: Start the beginning of the set of holes.
Step 6: If not start at the hole, which is sharing the previous first fit search end.
Step 7: Compare the hole.
Step 8: If large enough, then stop searching in the procedure.
Step 9: Display the values.
Step 10: Terminate the process.

Program:

```
#include<stdio.h>
#include<process.h>
void main()
{
	int a[20],p[20],i,j,n,m;
	clrscr();
	printf("Enter no of blocks:\n");
	scanf("%d",&n);
	for(i=0;i<n;i++)
	{
		printf("\nEnter the %dst block size:",i);
		scanf("%d",&a[i]);
	}
	printf("\nEnter the no of process:");
	scanf("%d",&m);
	for(i=0;i<m;i++)
	{
		printf("\nEnter the size of %dst process:",i);
		scanf("%d",&p[i]);
	}
	for(i=0;i<n;i++)
	{
		for(j=0;j<m;j++)
		{
			if(p[j]<=a[i])
			{
				printf("The process %d allocated %d\n",j,i);
				p[j]=10000;
				break;
			}
		}
	}
	for(j=0;j<m;j++)
	{
		if(p[j]!=10000)
		{
			printf("\n The process is not allocated \n",j);
		}
```

```
        }
getch();
}
```

Output:

```
Enter no of blocks:
3
Enter the 0st block size:70
Enter the 1st block size:50
Enter the 2st block size:90

Enter the no of process:2

Enter the size of 0st process:50
Enter the size of 1st process:80

The process 0 allocated 0
The process 1 allocated 2
```

Result

Thus the first fit algorithm was implemented successfully.

Ex: No: 11b **IMPLEMENTATION OF WORST FIT ALGORITHM**

Aim:

To write a program to implement worst fit algorithm for memory management.

Algorithm:

Step-1: Read the number of processes and number of available memory blocks.

Step-2: Next read the processes‟ memory requirements.

Step-3: Initialize the sizes of each memory block.

Step-4: For first fit algorithm, select the blocks in given order that are greater than or equal to that of the process‟ requirements.

Step-5: For best fit, sort the memory blocks in ascending order. Then choose the suitable memory block for each process.

Step-6: For worst fit, sort the memory blocks in descending order. Then choose the suitable memory block for each process.

Step-7: The internal fragmentation is computed by adding the remaining memory available in the allocated memory blocks.

Step-8: The external fragmentation is computed by adding the unallocated memory blocks.

Program:

```
#include<stdio.h>
#include<conio.h>
#define max 25
void main()
{
 int frag[max],b[max],f[max],i,j,nb,nf,temp,highest=0;
static int bf[max],ff[max];
clrscr();
printf("\n\tMemory Management Scheme - Worst Fit");
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
 scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++) {printf("Block %d:",i);scanf("%d",&b[i]);}
 printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++) {printf("File %d:",i);scanf("%d",&f[i]);}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
 if(bf[j]!=1)    //if bf[j] is not allocated
{
  temp=b[j]-f[i];
if(temp>=0)
if(highest<temp)
{
 ff[i]=j;
highest=temp;}}}
frag[i]=highest;
bf[ff[i]]=1;
highest=0;
}
 printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
 for(i=1;i<=nf;i++)
 printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
 getch();
}
```

Output:

```
Enter the no of blocks: 3
Enter the no of files: 2
Enter the size of blocks
Block 1:5
Block 2: 2
Block 3:7
Enter the size of the files:
File 1:1
File 2:4
```

File no	File size	Block no	Block size	Fragment
1	1	3	7	6
2	4	1	5	1

Result

Thus the Worst fit algorithm was implemented successfully.

Ex: No: 11c IMPLEMENTATION OF BEST FIT ALGORITHM

Aim:

To write a program to implement best fit algorithm for memory management.

Algorithm:

Step1: Start the process.
Step 2: Declare the size.
Step 3: Get the number of processes to be inserted.
Step 4: Allocate the best hole that is small enough for searching.
Step 5: Start at the best of the set of holes.
Step 6: If not start at the hole, which is sharing the previous best fit search end.
Step 7: Compare the hole.
Step 8: If small enough, then stop searching in the procedure.
Step 9: Display the values.
Step 10: Terminate the process.

Program

```
#include<stdio.h>
#include<process.h>
void main()
{
	int a[20],p[20],i,j,n,m,temp,b[20],temp1,temp2,c[20];
	clrscr();
	printf("Enter the no of blocks:\n");
	scanf("%d",&n);
	for(i=0;i<n;i++)
	{
		printf("Enter the %dst block size:",i);
		scanf("%d",&a[i]);
		b[i]=i;
	}
	printf("Enter the no of process:");
	scanf("%d",&m);
	for(i=0;i<m;i++)
	{
		printf("Enter the size of %dst process:",i);
		scanf("%d",&p[i]);
		c[i]=i;
	}
	for(i=0;i<n;i++)
	{
		for(j=0;j<m;j++)
	{
		if(a[i]<a[j])
		{
			temp=a[i];
			temp1=b[i];
			a[i]=a[j];
			b[i]=b[j];
			a[j]=temp;
			b[j]=temp1;
		}
		if(p[i]<p[j])
		{
			temp=p[i];
			temp2=c[i];
```

```
                        p[i]=p[j];
                        c[i]=c[j];
                        p[j]=temp;
                        c[j]=temp2;
                }
        }
}
        for(i=0;i<n;i++)
        {
                for(j=0;j<m;j++)
                {
                        if(p[j]<=a[i])
                        {
                                printf("\n The process %d allocated to block %d\n",c[j],b[i]);
                                p[j]=10000;
                                break;
                        }
                }
        }
        for(j=0;j<m;j++)
        {
                if(p[j]!=10000)
                {
                        printf("The process %d is not allocated:",j);
                }
        }
getch();
}
```

Output:

```
Enter the no of blocks:
3
Enter the 0st block size:50
Enter the 1st block size:70
Enter the 2st block size:90
Enter the no of process:2
Enter the size of 0st process:60
Enter the size of 1st process:80

 The process 0 allocated to block 1

 The process 1 allocated to block 2
```

Result

Thus the Best fit algorithm was implemented successfully.

Ex.No: 12 IMPLEMENT PAGING TECHNIQUE OF MEMORY MANAGEMENT

Aim:

To implement simple paging technique.

Algorithm:

1. Start the process
2. Declare the size with respect to page length
3. Check the need of replacement from the page to memory
4. Check the need of replacement from old page to new page in memory
5. Forma queue to hold all pages
6. Insert the page require memory into the queue
7. Check for bad replacement and page fault
8. Get the number of processes to be inserted
9. Display the values
10. Stop the process

Program :

```
#include<stdio.h>
#define max 25
main()
{
 int frag[max],b[max],f[max],i,j,nb,nf,temp;
 static int bf[max],ff[max];
 printf("\n\tMemory Management Scheme - First Fit");
 printf("\nEnter the number of blocks:");
 scanf("%d",&nb);
 printf("Enter the number of files:");
 scanf("%d",&nf);
 printf("\nEnter the size of the blocks:-\n");
 for(i=1;i<=nb;i++) {printf("Block %d:",i);scanf("%d",&b[i]);}
 printf("Enter the size of the files :-\n");
 for(i=1;i<=nf;i++) {printf("File %d:",i);scanf("%d",&f[i]);}

 for(i=1;i<=nf;i++)
 {
  for(j=1;j<=nb;j++)
  {
   if(bf[j]!=1)
   {
    temp=b[j]-f[i];
    if(temp>=0)
    {
     ff[i]=j;
     break;
    }
   }
  }
  frag[i]=temp;
  bf[ff[i]]=1;
 }
 printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
 for(i=1;i<=nf;i++)
 printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
}
```

Output:

Memory Management Scheme - First Fit

Enter the number of blocks:3
Enter the number of files:2
Enter the size of the blocks:-
Block 1:5
Block 2:2
Block 3:7
Enter the size of the files :-
File 1:1
File 2:4

File_no:	File_size :	Block_no:	Block_size:	Fragement
1	1	1	5	4
2	4	3	7	3

Result:

Thus the simple paging technique was implemented

Ex.No: 13a IMPLEMENT FIFO PAGE REPLACEMENT ALGORITHM

Aim:

To implement FIFO page replacement technique.

Algorithm:

1. Start the process
2. Declare the size with respect to page length
3. Check the need of replacement from the page to memory
4. Check the need of replacement from old page to new page in memory
5. Forma queue to hold all pages
6. Insert the page require memory into the queue
7. Check for bad replacement and page fault
8. Get the number of processes to be inserted
9. Display the values
10. Stop the process

Program

```
#include<stdio.h>
main()
{
        int i,j,n,a[50],frame[10],no,k,avail,count=0;
        printf("\n ENTER THE NUMBER OF PAGES:\n");
        scanf("%d",&n);
        printf("\n ENTER THE PAGE NUMBER :\n");
        for(i=1;i<=n;i++)
        scanf("%d",&a[i]);
        printf("\n ENTER THE NUMBER OF FRAMES :");
        scanf("%d",&no);
        for(i=0;i<no;i++)
        frame[i]= -1;
                j=0;
                printf("\tref string\t page frames\n");
for(i=1;i<=n;i++)
                {
                        printf("%d\t\t",a[i]);
                        avail=0;
                        for(k=0;k<no;k++)
if(frame[k]==a[i])
                                avail=1;
                        if (avail==0)
                        {
                                frame[j]=a[i];
                                j=(j+1)%no;
                                count++;
                                for(k=0;k<no;k++)
                                printf("%d\t",frame[k]);
        }
                        printf("\n");
        }
                printf("Page Fault Is %d",count);
                return 0;
}
```

Output:

ENTER THE NUMBER OF PAGES: 20

ENTER THE PAGE NUMBER : 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
ENTER THE NUMBER OF FRAMES :3

ref string	page frames		
7	7	-1	-1
0	7	0	-1
1	7	0	1
2	2	0	1
0			
3	2	3	1
0	2	3	0
4	4	3	0
2	4	2	0
3	4	2	3
0	0	2	3
3			
2			
1	0	1	3
2	0	1	2
0			
1			
7	7	1	2
0	7	0	2
1	7	0	1

Page Fault Is 15

Result:

Thus the implementation of FIFO page replacement algorithm was executed and output verified.

Ex.No: 13b IMPLEMENTATION OF LRU PAGE REPLACEMENT ALGORITHM

Aim:

To implement LRU page replacement technique.

Algorithm :

1. Start the process
2. Declare the size
3. Get the number of pages to be inserted
4. Get the value
5. Declare counter and stack
6. Select the least recently used page by counter value
7. Stack them according the selection.
8. Display the values
9. Stop the process

Program :

```
#include<stdio.h>
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
The no of page faults is 10
```

Result:

Thus the LRU page replacement algorithm was implemented and output verified

Ex.No: 13c IMPLEMENT LFU PAGE REPLACEMENT ALGORITHM

Aim:

To implement LFU page replacement technique**.**

Algorthim:

1. Start Program
2. Read Number Of Pages And Frames
3. Read Each Page Value
4. Search For Page In The Frames
5. If Not Available Allocate Free Frame
6. If No Frames Is Free Replace The Page With The Page That Is Lastly Used
7. Print Page Number Of Page Faults
8. Stop process.

Program:

```
#include<stdio.h>
int main()
{
  int f,p;
  int pages[50],frame[10],hit=0,count[50],time[50];
  int i,j,page,flag,least,minTime,temp;

  printf("Enter no of frames : ");
  scanf("%d",&f);
  printf("Enter no of pages : ");
  scanf("%d",&p);

  for(i=0;i<f;i++)
  {
    frame[i]=-1;
  }
  for(i=0;i<50;i++)
  {
    count[i]=0;
  }
  printf("Enter page no : \n");
  for(i=0;i<p;i++)
  {
    scanf("%d",&pages[i]);
  }
  printf("\n");
  for(i=0;i<p;i++)
  {
    count[pages[i]]++;
    time[pages[i]]=i;
    flag=1;
    least=frame[0];
    for(j=0;j<f;j++)
    {
      if(frame[j]==-1 || frame[j]==pages[i])
      {
        if(frame[j]!=-1)
        {
           hit++;
        }
```

```
            flag=0;
            frame[j]=pages[i];
            break;
          }
          if(count[least]>count[frame[j]])
          {
            least=frame[j];
          }
        }
       if(flag)
        {
          minTime=50;
          for(j=0;j<f;j++)
          {
            if(count[frame[j]]==count[least] && time[frame[j]]<minTime)
            {
              temp=j;
              minTime=time[frame[j]];
            }
          }
          count[frame[temp]]=0;
          frame[temp]=pages[i];
        }
        for(j=0;j<f;j++)
        {
          printf("%d ",frame[j]);
        }
        printf("\n");
      }
      printf("Page hit = %d",hit);
      return 0;
    }
```

Output:

```
Enter no of frames : 3
Enter no of pages : 1 4 7 8 5 2 3 6 0 9
Enter page no :
4 -1 -1
Page hit = 0
```

Result:

Thus the LFU page replacement algorithm was implemented

Ex.No. 14a IMPLEMENTATION OF FILE ORGANIZATION TECHNIQUES

Date:

Aim:

To implement Single level directory structure in C.

Algorithm:

1. Start
2. Declare the number, names and size of the directories and file names.
3. Get the values for the declared variables.
4. Display the files that are available in the directories.
5. Stop.

Program :

Single level directory

```
#include<stdio.h>
int main()
{
int master,s[20];
char f[20][20][20];
char d[20][20];
int i,j;
printf("enter number of directorios:");
scanf("%d",&master);
printf("enter names of directories:");
for(i=0;i<master;i++)
scanf("%s",&d[i]);
printf("enter size of directories:");
for(i=0;i<master;i++)
scanf("%d",&s[i]);
printf("enter the file names :");
for(i=0;i<master;i++)
for(j=0;j<s[i];j++)
scanf("%s",&f[i][j]);
printf("\n");
printf(" directory\tsize\tfilenames\n");
for(i=0;i<master;i++)
{
printf("%s\t\t%2d\t",d[i],s[i]);
for(j=0;j<s[i];j++)
printf("%s\n\t\t\t",f[i][j]);
printf("\n");
}
printf("\t\n");
}
```

Output

```
enter number of directorios:3
enter names of directories: at er org
enter size of directories:1 1 1
enter the file names :a s d
directory size filenames
at 1 a
er 1 s
org 1 d
```

Result :

Thus the above program executed successfully.

Ex.No. 14b. **TWO-LEVEL DIRECTORY STRUCTURE**

Aim:

To implement Two-level directory structure in C.

Algorithm:

1. Start
2. Declare the number, names and size of the directories and subdirectories and file names.
3. Get the values for the declared variables.
4. Display the files that are available in the directories and subdirectories.
5. Stop.

Program :

Two level directory

```
#include<stdio.h>
#include<conio.h>
struct st
{
char dname[10];
char sdname[10][10];
char fname[10][10][10];
int ds,sds[10];
}dir[10];
int main()
{
int i,j,k,n;
printf("enter number of directories:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter directory %d names:",i+1);
scanf("%s",&dir[i].dname);
printf("enter size of directories:");
scanf("%d",&dir[i].ds);
for(j=0;j<dir[i].ds;j++)
{
printf("enter subdirectory name and size:");
scanf("%s",&dir[i].sdname[j]);
scanf("%d",&dir[i].sds[j]);
for(k=0;k<dir[i].sds[j];k++)
{
printf("enter file name:");
scanf("%s",&dir[i].fname[j][k]);
}
}
}
printf("\ndirname\t\tsize\tsubdirname\tsize\tfiles");
printf("\n*****************************************\n");
for(i=0;i<n;i++)
{
printf("%s\t\t%d",dir[i].dname,dir[i].ds);
for(j=0;j<dir[i].ds;j++)
{
printf("\t%s\t\t%d\t",dir[i].sdname[j],dir[i].sds[j]);
for(k=0;k<dir[i].sds[j];k++)
printf("%s\t",dir[i].fname[j][k]);
printf("\n\t\t");
```

```
}
printf("\n");
}
}
```

Output

```
enter number of directories:3
enter directory 1 names:org
enter size of directories:1
enter subdirectory name and size: we 2
enter file name:a
enter file name: s
enter directory 2 names:com
enter size of directories:2
enter subdirectory name and size:er 1
enter file name:t
enter subdirectory name and size:et 1
enter file name:y
enter directory 3 names:io
enter size of directories:2
enter subdirectory name and size:ww 1
enter file name:t
enter subdirectory name and size:qq 1
enter file name:q
dirname size subdirname size files
************************************
org 1 we 2 a
s
com 2 er 1 t
et 1 y
io 2 ww 1 t
qq 1 q
```

Result :

Thus the above program executed successfully.

Ex.No.14.c. **HIERARCHICAL DIRECTORY STRUCTURE**

Aim:
To implement hierarchial directory structure in C.
Algorithm:
1. Start
2. Declare the number, names and size of the directories and subdirectories and file names.
3. Get the values for the declared variables.
4. Display the files that are available in the directories and subdirectories.
5. Stop.
Program :
Hierarchical

```
#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]); } } }
```

Output :

Result :

Thus the above program executed successfully.

Ex.No.14.d **DIRECTED ACYCLIC GRAPH (DAG)**

Aim:
To implement Directed Acyclic Graph in C.
Algorithm:
1. Start
2. Collect set of nodes 1, 2, …, n
3. Get the value of an edge (i,j) whenever i < j
4. Calculate N-choose-2 = n (n-1)/2 edges, but no cycles.
5. Stop.
Program :
DAG

```
#include<stdio.h>
#include<graphics.h>
#include<string.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;
typedef struct
{
char from[20];
char to[20];
}
link;
link L[10];
int nofl;
node * root;
void main()
{
int gd=DETECT,gm;
root=NULL;
clrscr();
create(&root,0,"root",0,639,320);
read_links();
initgraph(&gd,&gm,"c:\tc\BGI");
draw_link_lines();
display(root);
getch();
closegraph();
}
read_links()
{
int i;
printf("how many links");
scanf("%d",&nofl);
for(i=0;i<nofl;i++)
{
printf("File/dir:");
fflush(stdin);
gets(L[i].from); printf("user name:");
```

```
fflush(stdin); gets(L[i].to);
}
}
draw_link_lines()
{
int i,x1,y1,x2,y2; for(i=0;i<nofl;i++)
{
search(root,L[i].from,&x1,&y1);
search(root,L[i].to,&x2,&y2);
setcolor(LIGHTGREEN);
setlinestyle(3,0,1);
line(x1,y1,x2,y2);
setcolor(YELLOW);
setlinestyle(0,0,1);
}
}
search(node *root,char *s,int *x,int *y)
{
int i;
if(root!=NULL)
{
if(strcmpi(root->name,s)==0)
{
*x=root->x;
*y=root->y; return;
}
else
{
for(i=0;i<root->nc;i++)
search(root->link[i],s,x,y);
}
}
}
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;
} }
/* displays the constructed tree in graphics mode */
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 name of dir/file(under root):gk
enter 1 for dir/ 2 for file:11
how many links2
File/dir:1
user name:gokul
File/dir:xx
user name:krishnan

BGI Error: Graphics not initialized (use 'initgraph')
```

Result :

Thus the above program executed successfully.

Ex.No: 15a SEQUENTIAL FILE ALLOCATION STRATEGIES

Aim:

To implement the sequential file allocation strategies

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.

```
if(b[s1].flag==0)
  {
          for(j=s1;j<s1+p[i];j++) {
          if((b[j].flag)==0)
          count++;
         }
          if(count==p[i])
          break;
        }
```

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

```
          for(s=s1;s<(s1+p[i]);s++)
        {
          k[i][j]=s;
          j=j+1;
          b[s].bno=s;
          b[s].flag=1;
          }
```

Step 5: Print the results fileno, lenth ,Blocks allocated.
Step 6: Stop the program.

Program:

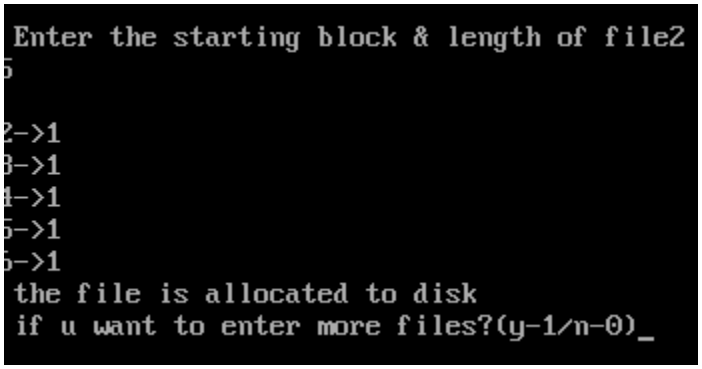
```
#include<stdio.h>
main()
{
int f[50],i,st,j,len,c,k;

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();

}
```

Output:



Results:

` Thus the file organization scheme was implemented

Ex.No: 15b **INDEX FILE ALLOCATION**

Aim:

To implement the Index file allocation technique.

Algorthim:

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 q= random(100);

a). Check whether the selected location is free .

b). If the location is free allocate and set flag=1 to the allocated locations. While allocating next location address to attach it to previous location

```
for(i=0;i<n;i++) {
for(j=0;j<s[i];j++) {
q=random(100);
if(b[q].flag==0)
b[q].flag=1;
b[q].fno=j;
r[i][j]=q;
if(j>0)  {
p=r[i][j-1];
b[p].next=q; }
    }
}
```

Step 5: Print the results fileno, lenth ,Blocks allocated.
Step 6: Stop the program

Program

```
#include<stdio.h>
int f[50],i,k,j,inde[50],n,c,count=0,p;
main()
{
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();  }
```

Output:

Enter how many block that are already allocated 3
Enter the block no.that are already al

Result:

` Thus the file organization scheme was implemented

Ex.No: 15c **LINKED FILE ALLOCATION METHOD**

Aim:

To implement the file allocation method using Linked method

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 q= random(100);
a). Check whether the selected location is free .
b). If the location is free allocate and set flag=1 to the allocated locations q=random(100); {
if(b[q].flag==0)
b[q].flag=1;
b[q].fno=j;
r[i][j]=q;
}
Step 5: Print the results fileno, lenth ,Blocks allocated.
Step 6: Stop the program.

Program

```
#include<stdio.h>
#include<conio.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 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 file organization scheme was implemented.