SRI SHANMUGHA COLLEGE OF ENGINEERING AND TECHNOLOGY

(APPROVED BY AICTE, NEW DELHI & AFFILIATED TO ANNAUNIVERSITYAND ACCREDITED BY NAAC & NBA(ECE,CSE,MECH)

Tiruchengode-Sankari main road, Pullipalayam, Morur (Po), Sankari (Tk), Salem (Dt) Pin: 637 304

RECORD NOTE BOOK

CS6481-OPERATING SYSTEM LABORATORY

NAME :

REG NO :

Submitted for the Anna University Practical Examinations

SHANMUGHA COLLEGE OF ENGINEERING AND

TECHNOLOGY

(APPROVED BY AICTE, NEW DELHI & AFFILIATED TO

ANNA UNIVERSITYAND ACCREDITED BY NAAC & NBA(ECE,CSE,MECH)

Tiruchengode-Sankari main road, Pullipalayam, Morur

(Po), Sankari (Tk), Salem (Dt) Pin: 637 304

RECORD NOTE BOOK

REG NO

Certified that this is a bonafide record of Practical work done by

Mr/Ms …… of the ……

Semester …….. Branch during the Academic year ……

in the ……. Laboratory

Staff-in-charge Head of the Department

Submitted for the University Practical

Examination held on……..

Internal Examiner External Examiner

CS8461-OPERATING SYSTEM LABORATORY

DEGREE / BRANCH**: B.E / CSE** YEAR / SEM**: II / IV**

TABLE OF CONTENTS

S.NO	DATE	NAME OF THE EXPERIMENT	PAGE NO	SIGNATURE

Ex.no: 01
Date : **Basic UNIX Commands**

AIM

To learn the basics UNIX commands.

COMMANDS:

I. File and Directory Related commands

1) pwd

This command prints the current working directory

2) ls

This command displays the list of files in the current working directory.

$ls –l Lists the files in the long format
$ls –t Lists in the order of last modification time
$ls –d Lists directory instead of contents
$ls -u Lists in order of last access time

3) cd

This command is used to change from the working directory to any other directory specified.

$cd directoryname

4) cd ..

This command is used to come out of the current working directory.

$cd ..

5) mkdir

This command helps us to make a directory.

$mkdir directoryname

6) rmdir

This command is used to remove a directory specified in the command line. It requires the specified directory to be empty before removing it.

$rmdir directoryname

7) cat

This command helps us to list the contents of a file we specify.

$cat [option][file]

cat > filename – This is used to create a new file.
cat >>filename – This is used to append the contents of the file

8) cp

This command helps us to create duplicate copies of ordinary files.

$cp source destination

9) mv

This command is used to move files.

$mv source destination

10) ln

This command is to establish an additional filename for the same ordinary file.

$ln firstname secondname

11) rm

This command is used to delete one or more files from the directory.

$rm [option] filename

$rm –i Asks the user if he wants to delete the file mentioned.

$rm –r Recursively delete the entire contents of the directory as well as the directory itself.

II) Process and status information commands

1) who

This command gives the details of who all have logged in to the UNIX system currently.

$ who

2) who am i

This command tells us as to when we had logged in and the system's name for the connection being used.

$who am i

3) date

This command displays the current date in different formats.

+%D mm/dd/yy +%w Day of the week
+%H Hr-00 to 23 +%a Abbr.Weekday
+%M Min-00 to 59 +%h Abbr.Month
+%S Sec-00 to 59 +%r Time in AM/PM
+%T HH:MM:SS +%y Last two digits of the year

4) echo

This command will display the text typed from the keyboard.

$echo

Eg: $echo Have a nice day
O/p Have a nice day

II) Text related commands

1. head

This command displays the initial part of the file. By default it displays first ten lines of the file.

$head [-count] [filename]

2. tail

This command displays the later part of the file. By default it displays last ten lines of the file.

$tail [-count] [filename]

3. wc

This command is used to count the number of lines, words or characters in a file.

$wc [-lwc] filename

4. find

The find command is used to locate files in a directory and in a subdirectory.

The –name option

This lists out the specific files in all directories beginning from the named directory. Wild cards can be used.

The –type option

This option is used to identify whether the name of files specified are ordinary files or directory files. If the name is a directory then use "-type d "and if it is a file then use "-type f".

The –mtime option

This option will allow us to find that file which has been modified before or after a specified time. The various options available are –mtime n(on a particular day),-mtime +n(before a particular day),-mtime –n(after a particular day)

The –exec option

This option is used to execute some commands on the files that are found by the find command.

IV) File Permission commands

1) chmod

Changes the file/directory permission mode: $ chmod 777 file1
Gives full permission to owner, group and others

$ chmod o-w file1
Removes write permission for others.

V) Useful Commands:

1) exit - Ends your work on the UNIX system.

2) Ctrl-l or clear

Clears the screen.

3) Ctrl-c

Stops the program currently running.

4) Ctrl-z

Pauses the currently running program.

5) man COMMAND

Looks up the UNIX command COMMAND in the online manual pages.

6) history

List all commands typed so far.

7) more FILE

Display the contents of FILE, pausing after each screenful.
There are several keys which control the output once a screenful has been printed.
<enter> Will advance the output one line at a time.
<space bar> Will advance the output by another full screenful.
"q" Will quit and return you to the UNIX prompt.

8) less FILE

"less" is a program similar to "more", but which allows backward movement in the file as well as forward movement.

9) lpr FILE

To print a UNIX text or PostScript file, type the following command at the system prompt:

Meta characters

Some special characters, called metacharacters may be used to specify multiple filenames. These characters substitute filenames or parts of filenames.

The "*"

This character is used to indicate any character(s)

$ cat ap*

This displays the contents of all files having a name starting with ap followed by any number of characters.

The "?" This character replaces any one character in the filename.

$ ls ?st -list all files starting with any character followed by st.

The [] These are used to specify range of characters.

$ ls [a-z]pple

Lists all files having names starting with any character from a to z.

Absolute path and relative path

Generally if a command is given it will affect only the current working directory. For example the following command will create a directory named curr in the current working directory.

$ mkdir curr

The directory can also be created else where in the file system using the absolute and relative path.If the path is given with respect to the root directory then it is called full path or absolute path

$ mkdir /home/it2006/it2k601/curr

The full path always start with the /, which represents the root directory.

If the path is given with respect to the current working directory or parent directory then it is called relative path.

$ mkdir ../curr

The above command will create a directory named curr in the parent directory.

$ mkdir ./first/curr

The above command will create a directory named curr inside first directory , where the directory first is located in the current working directory.

Note "." Represents current directory and ".." represents parent directory.

RESULT:

Thus the above Linux commands are executed and observed the results.

Ex.No:2.a

Date: **SHELL PROGRAMMING USING SYSTEM CALLS**

(FORK(), GETPID(), EXIT(),WAIT()

AIM:

To write a shell script using system calls forf(), getpid(), exit() and wait()

ALGORITHM:

1. Create the process using fork() system call.
2. Check the Process id. If process id=1,then display an error.
3. If process id=0 then display as child process.
4. Display its process id and parent process id.
5. Else display as parent process.
6. Display its process id and child process id

PROGRAM:

```
#include<stdio.h>
#include<process.h>
main()
{
int pid;
pid=fork();
if(pid<0)
{
printf("Error");
exit(0);
}
else if(pid==0)
{
printf("\n child process id is %d",getpid());
printf("\n Parent process id is %d",getppid());
}
else
{
wait(pid);
execl("/bin/date","date","null");
}
return 0;
}
```

OUTPUT:

Child process id is 18398

Parent process Id is 18398

Mon March 02

RESULT:

Thus the shell script using system calls fork(), getpid(), exit() and wait() was executed and output was verified.

Ex.No:2.b

Date: **SHELL PROGRAMMING USING SYSTEM CALLS**

(OPENDIR(), READDIR())

AIM:

To write a shell script using system calls opendir(), readdir()

ALGORITHM:

1. Start
2. Get the directory name using DIR.
3. Open the directory using opendir() system call
4. If opendir=0 then display as can't open
5. Read the dir using readdir() system call
6. Close the directory using closedir() system call
7. Stop

PROGRAM:

```
#include<stdio.h>
#include<process.h>
#include<sys/types.h>
#include<dirent.h>
int main(int argc, char *argv[])
{
  DIR *dp;
struct dirent *dirp;
if(argc!=2)
{
  printf("Directory name is required \n");
  exit(1);
}
if((dp=opendir(argc[1]))==0)
{
  printf("Can't open %s \n",argv[1]);
  exit(1);
while((dirp=readdir(dp))!=0)
printf("%s \n",dirp->d_name);
closedir(dp);
exit(0);
}
return 0;
}
```

OUTPUT:

./a.out

One

Two

…

…

RESULT:

Thus the shell script using system calls open(), read() was executed and output was verified.

Ex.No:2.c

Date: **SHELL PROGRAMMING USING SYSTEM CALLS**

(STAT())

AIM:

To write a shell script to display the status of the file such as inode, size, mode, id etc

ALGORITHM:

1. create a file.
2. using the members of the stat system call display the status of the file such as inode,size,mode,id etc

PROGRAM:

```
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
void main()
{
struct stat s;
char fname[20];
printf (" enter the filename");
scanf("%s",fname);
stat(fname, &s);
printf("file inode\t%d\n", s.st_ino);
printf("file size \t%d\n",s.st_size);
printf("file mode \t%d\n", s.st_mode);
printf("userid \t%d\n",s.st_uid);
printf("groupid\t%d\n",s.st_gid);
}
```

OUTPUT:

```
Enter filename: aa
File size : 300
File mode : WR
User Id : CS005
Group Id : 5098
```

RESULT:

Thus the shell script to display the status of the file was executed and output was verified.

Ex.No:3

Date: **SIMULATION OF UNIX COMMANDS**

AIM:

To write a program for simulation of unix commands cp,ls and grep.

ALGORITHM:

1. Start the program.
2. Enter the options.
3. If the option is “1” call ls command and it display the files.
4. If the option is “2” call gerp command and match the given pattern with the read word and if it matches, display the line of occurance.
5. If the option is “3” call cp command.
6. If the option is “4” call rm command.
7. Stop the program.

PROGRAM:

```
#include<stdio.h>
main()
{
char c;
int n;
printf("1.list of files in the directory\n2.list the lines\n3.copy\n4.remove\n");
scanf("%d",&n);
switch(n)
{
case 1:
system("ls");
break;
case 2:
system("grepargclsgp.c");
break;
case 3:
system(" cplsgp.cgpls.c");
printf("the file is copied");
break;
case 4:
system("rmlsgp.c");
printf("the file is deleted");
break;
}
}
```

OUTPUT:

```
[it53@localhost ~]$ cc lsgp.c
[it53@localhost ~]$ ./a.out
1.list of files in the directory
2.list the lines
3.copy
4.remove
1
3q    bit.cftp.c   ospf1.c    praba.crarp.csmp.c        sort.sh
a.outforkexec.cio.c    praba1.sh  praba.sh  sample  sorting.sh   syscall.c
 [it53@localhost ~]$ ./a.out
1.list of files in the directory
2.list the lines
3.copy
4.remove
2
system("grepargclsgp.c");
[it53@localhost ~]$ ./a.out
1.list of files in the directory
2.list the lines
3.copy
4.remove
3
the file is copied[it53@localhost ~]$ ./a.out
1.list of files in the directory
2.list the lines
3.copy
4.remove
4
the file is deleted
```

RESULT:

Thus the program for simulation of unix commands ls, cp and grep was executed and verified.

Ex No:04

Date: **SHELL PROGRAMMING**

AIM

To write the following simple shell Programs

i) Finding biggest of two numbers
ii) Finding odd or even
iii) Finding Factorial of the given number

PROCEDURE:

1. Open the terminal window
2. Goto c shell
3. Type shell programs in any one following editors vi/emacs
4. Save the program using .sh extension and compile the program using shell
5. Note the output

PROGRAM:

i) Finding biggest of two numbers

Big.sh

```
echo "enter the number"
 read a b
if [ $a -gt $b ]
 then
echo "A is big"
else
 echo "B is big"
fi
```

OUTPUT:

Enter The Two Number: 23 67
B is Big.

ii) Finding Odd or Even

```
echo "enter the number"
 read num
echo "enter the number"
 read num
 if [ `expr $num % 2` -eq 0 ]
```

```
then
echo "number is even"
else
echo "number is odd"
fi
```

OUTPUT:

```
enter the number: 5
the number is odd.
```

iii) Finding Factorial of the given number

```
echo "enter the number"
read n
fact=1
i=1
while [ $i -le $n ]
do
fact=`expr $i \* $fact`
 i=`expr $i + 1`
 done
echo "the fcatorial number of $ni is $fact “
```

OUTPUT:

```
Enter the number : 4
The factorial of 4 is 24.
```

iv)Finding Fibonacci number of the given number

```
echo  “Enter the range:”
read n
a=0
b=1
echo  “Fibonacci series is”
echo $a
echo $b
i=2
while test $i -lt $n
do
        c=`expr $a + $b`
        echo $c
        i=`expr $i + 1`
        a=$b
        b=$c
done
```

OUTPUT:

```
Enter the range:
5
fibonacci series is
0
1
1
2
3
```

v) Finding Prime number of the given number

```
i=2
rem=1
echo -e "Enter a number: \c"
read num
    if [ $num -lt 2 ];
    then
    echo -e "$num is not prime\n"
    exit 0
    fi
    while [ $i -le `expr $num / 2` -a $rem -ne 0 ];
    do
            rem=`expr $num % $i`
            i=`expr $i + 1`
    done
    if [ $rem -ne 0 ]; then
            echo -e "$num is prime\n"
    else
            echo -e "$num is not prime\n"
    fi
```

OUTPUT:

```
Enter a number: 5
5 is prime

Enter a number: 22
22 is not prime
```

RESULT:

Thus the sample shell script for the Unix shell programming was written and output is verified

Ex No:5.a

Date: **IMPLEMENTATION OF FCFS**

AIM:

To write a C program for implementing First Come First Served algorithm.

ALGORITHM:

1. Establish the number of processes.
2. Establish process ID, arrival time and execution time for each process.
3. Initialize waiting time and the turnaround time to zero.
4. For each of the process do the following.
    a. Set the process waiting time to previous process turnaround time.
    b. Calculate current process turnaround time as sum of process execution time and process waiting time.
5. Display the Gantt chart.
6.Calculate the average waiting time by adding waiting time of all process and divide by the number of processes.
7. Calculate the average turnaround time by adding the turnaround time of each process and dividing it by the total number of processes.
8. Stop.

PROGRAM:

```
#include <stdio.h>
struct process
{
int pid;
int btime;
int wtime;
int ttime;
} p[10];

main()
{
int i,j,k,n,ttur,twat;
float awat,atur;
getch();

printf("Enter no. of process : ");
scanf("%d", &n);
for(i=0; i<n; i++)
{
printf("Burst time for process P%d (in ms) : ",(i+1));
```

```
scanf("%d", &p[i].btime);
p[i].pid = i+1;
}

p[0].wtime = 0;
for(i=0; i<n; i++)
{
p[i+1].wtime = p[i].wtime + p[i].btime;
p[i].ttime = p[i].wtime + p[i].btime;
}
ttur = twat = 0;
for(i=0; i<n; i++)
{
ttur += p[i].ttime;
twat += p[i].wtime;
}
awat = (float)twat / n;
atur = (float)ttur / n;

printf("\nFCFS Scheduling\n\n");
for(i=0; i<28; i++)
printf("-");
printf("\nProcess B-Time T-Time W-Time\n");
for(i=0; i<28; i++)
printf("-");

for(i=0; i<n; i++)
printf("\nP%d\t%4d\t%3d\t%2d", p[i].pid,p[i].btime,p[i].ttime,p[i].wtime);
printf("\n"); for(i=0; i<28; i++)
printf("-");

printf("\n\nAverage waiting time: %5.2fms", awat);
printf("\nAverage turn around time : %5.2fms\n", atur);

printf("\n\nGANTT Chart\n");
printf("-");
for(i=0; i<(p[n-1].ttime + 2*n); i++)
printf("-");
printf("\n");
printf("|"); for(i=0; i<n; i++)
{
k = p[i].btime/2; for(j=0; j<k; j++)
printf(" "); printf("P%d",p[i].pid); for(j=k+1; j<p[i].btime; j++)
printf(" ");
printf("|");
}
```

```
printf("\n");
printf("-");
for(i=0; i<(p[n-1].ttime + 2*n); i++) printf("-");
printf("\n");
printf("0"); for(i=0; i<n; i++)
{
for(j=0; j<p[i].btime; j++) printf(" ");
printf("%2d",p[i].ttime);
}
return 0;
}
```

OUTPUT:

Enter no.of process : 3
Burst time for process P1(in ms) : 2
Burst time for process P1(in ms) : 8
Burst time for process P1(in ms) : 5

FCFS Scheduling:

Process	B-Time	T-Time	W-Time
P1	2	2	0
P2	8	10	2
P3	5	15	10

Average waiting time: 4.00 ms
Average turn around time : 9.00 ms

GANTT Chart

P1	P2	P3

0 2 10 15

RESULT:

Thus a C program for FCFS CPU scheduling algorithm was written and its output was verified.

Ex No:5.b

Date: **IMPLEMENTATION OF SJF**

AIM:

To write a C program for implementing Shortest Job First algorithm.

ALGORITHM:

1. Establish the number of processes.
2. Establish process ID, arrival time and execution time for each process.
3. Initialize waiting time and the turnaround time to zero.
4. For each of the process do the following.
    a. Set the process waiting time to previous process turnaround time.
    b. Calculate current process turnaround time as sum of process execution time and process waiting time.
5. Display the Gantt chart.
6. Calculate the average waiting time by adding waiting time of all process and divide by the number of processes.
7. Calculate the average turnaround time by adding the turnaround time of each process and dividing it by the total number of processes.
8. Stop.

PROGRAM:

```
#include <stdio.h>
struct process
{
int pid; int btime;
int wtime;
int ttime;
} p[10], temp;

main()
{
int i,j,k,n,ttur,twat;
float awat,atur;

printf("Enter no. of process : ");
scanf("%d", &n);
for(i=0; i<n; i++)
{
printf("Burst time for process P%d (in ms) : ",(i+1));
scanf("%d", &p[i].btime);
p[i].pid = i+1;
}
```

```
for(i=0; i<n-1; i++)
{
for(j=i+1; j<n; j++)
{
if((p[i].btime > p[j].btime) ||(p[i].btime == p[j].btime && p[i].pid > p[j].pid))
{
temp = p[i]; p[i] = p[j]; p[j] = temp;
}}}
p[0].wtime = 0; for(i=0; i<n; i++)
{
p[i+1].wtime = p[i].wtime + p[i].btime; p[i].ttime = p[i].wtime + p[i].btime;
}
ttur = twat = 0;
 for(i=0; i<n; i++)
{
ttur += p[i].ttime; twat += p[i].wtime;
}
awat = (float)twat / n;
atur = (float)ttur / n;
printf("\n        SJF Scheduling\n\n"); for(i=0; i<28; i++)
printf("-");
printf("\nProcess B-Time T-Time W-Time\n"); for(i=0; i<28; i++)
printf("-"); for(i=0; i<n; i++)
printf("\n        P%-4d\t%4d\t%3d\t%2d", p[i].pid,p[i].btime,p[i].ttime,p[i].wtime);
printf("\n"); for(i=0; i<28; i++)
printf("-");
printf("\n\nAverage waiting time       : %5.2fms", awat);
printf("\nAverage turn around time : %5.2fms\n", atur);
printf("\n\nGANTT Chart\n"); printf("-");
for(i=0; i<(p[n-1].ttime + 2*n); i++) printf("-");
printf("\n|"); for(i=0; i<n; i++)
{
k = p[i].btime/2; for(j=0; j<k; j++)
printf(" "); printf("P%d",p[i].pid); for(j=k+1; j<p[i].btime; j++)
printf(" ");
printf("|");
}
printf("\n-");
for(i=0; i<(p[n-1].ttime + 2*n); i++) printf("-");
printf("\n0"); for(i=0; i<n; i++)
{
for(j=0; j<p[i].btime; j++) printf(" ");
printf("%2d",p[i].ttime);
}
return 0;
}
```

OUTPUT:

Enter no.of process : 3
Burst time for process P1(in ms) : 4
Burst time for process P1(in ms) : 6
Burst time for process P1(in ms) : 8

SJF Scheduling:

Process	B-Time	T-Time	W-Time
P1	3	3	0
P2	4	7	3
P3	6	13	7

Average waiting time: 3.33 ms
Average turn around time :7.67 ms

GANTT Chart

P1	P2	P3
0	3	7	13

RESULT:

Thus a C program for SJF CPU scheduling algorithm was written and its output was verified.

Ex No:5.c

Date: **IMPLEMENTATION OF PRIORITY SCHEDULING**

AIM:

To write a C program for implementing Priority Scheduling Algorithm

ALGORITHM:

1. Get the number of Processes.
2. Get also the burst time and the priority for each process from the user.
3. Sort the burst time of the processes according to the process priority in ascending order.
4. For each process the waiting time is equivalent to the CPU time of the previous process.
5. Calculate current process turn around time as sum of process execution time and process waiting time.
6. The ratio of waiting time of all the processes to the number process will give the average waiting time.
7. Calculate the average turnaround time by adding turnaround time of all process and dividing it by the number of processes.
8. Display the output.

PROGRAM:

```
#include <stdio.h>
struct process
{
int pid;
int btime;
int pri;
int wtime;
int ttime;
} p[10], temp;

main()
{
int i,j,k,n,ttur,twat;
float awat,atur;
clrscr();
printf("Enter no. of process : ");
scanf("%d", &n);
for(i=0; i<n; i++)
{
printf("Burst time for process P%d (in ms) : ", (i+1));
scanf("%d", &p[i].btime);
printf("Priority for process P%d : ", (i+1));
```

```
scanf("%d", &p[i].pri);
p[i].pid = i+1;
}
for(i=0; i<n-1; i++)
{
for(j=i+1; j<n; j++)
{
if((p[i].pri > p[j].pri) ||(p[i].pri == p[j].pri && p[i].pid > p[j].pid) )
{
temp = p[i]; p[i] = p[j]; p[j] = temp;
}
}
}
p[0].wtime = 0; for(i=0; i<n; i++)
{
p[i+1].wtime = p[i].wtime + p[i].btime; p[i].ttime = p[i].wtime + p[i].btime;
}

ttur = twat = 0; for(i=0; i<n; i++)
{
ttur += p[i].ttime; twat += p[i].wtime;
}
awat = (float)twat / n; atur = (float)ttur / n;
printf("\n\t Priority Scheduling\n\n"); for(i=0; i<38; i++)
printf("-");
printf("\nProcess B-Time Priority T-Time W-Time\n");
for(i=0; i<38; i++)
printf("-");
for (i=0; i<n; i++)
printf("\n P%-4d\t%4d\t%3d\t%4d\t%4d", p[i].pid,p[i].btime,p[i].pri,p[i].ttime,p[i].wtime);
printf("\n");
for(i=0; i<38; i++)
printf("-");
printf("\n\nAverage waiting time: %5.2fms", awat);
printf("\nAverage turn around time : %5.2fms\n", atur);
printf("\n\nGANTT Chart\n");
printf("-");
for(i=0; i<(p[n-1].ttime + 2*n); i++)
printf("-");
printf("\n|");
for(i=0; i<n; i++)
{
k = p[i].btime/2;
for(j=0; j<k; j++)
printf(" ");
printf("P%d",p[i].pid);
```

```
for(j=k+1; j<p[i].btime; j++)
printf(" ");
printf("|");
}
printf("\n-");
for(i=0; i<(p[n-1].ttime + 2*n); i++)
printf("-");
printf("\n0");
for(i=0; i<n; i++)
{
for(j=0; j<p[i].btime; j++)
printf(" ");
printf("%2d",p[i].ttime);
}
getch();
return 0;
}
```

OUTPUT:

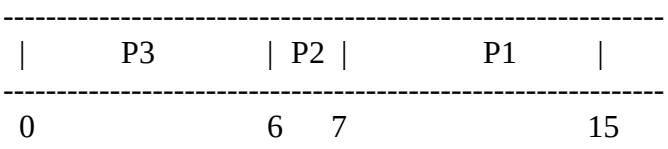
```
Enter no.of process : 3
Burst time for process P1(in ms) : 8
Priority for process P1 : 4
Burst time for process P2(in ms) : 1
Priority for process P2 : 3
Burst time for process P3(in ms) : 6
Priority for process P3 : 2
```

Priority Scheduling:

Process	B-Time	Priority	T-Time	W-Time
P3	6	2	6	0
P2	1	3	7	6
P1	8	4	15	7

Average waiting time: 4.33 ms
Average turn around time :9.33 ms

GANTT Chart



RESULT:

Thus a C program for Priority CPU scheduling algorithm was written and its output was verified.

Ex No: 5.d

Date: **IMPLEMENTATION OF ROUND ROBIN SCHEDULING**

AIM:

To write a C program for implementing Round Robin Scheduling Algorithm

ALGORITHM:

1. Get the number of Processes.
2. Get also the burst time and the priority for each process from the user.
3. Get time slice value.
4. For each process the waiting time is equivalent to the CPU time of the previous process.
5. Calculate current process turnaround time as sum of process execution time and process waiting time.
6. The ratio of waiting time of all the processes to the number of process will give the average waiting time.
7. Calculate the average turnaround time by adding turnaround time of all process and dividing it by the number of processes.
8. Display the output.

PROGRAM:

```
#include <stdio.h>
main()
{
int i,x=-1,k[10],m=0,n,t,s=0;
int a[50],temp,b[50],p[10],bur[10],bur1[10];
int   wat[10],tur[10],ttur=0,twat=0,j=0;
float awat,atur;
printf("Enter no. of process : ");
scanf("%d", &n);
for(i=0; i<n; i++)
{
printf("Burst time for process P%d : ", (i+1));
scanf("%d", &bur[i]);
bur1[i] = bur[i];
}
printf("Enter the time slice (in ms) : ");
scanf("%d", &t);
for(i=0; i<n; i++)
{
b[i] = bur[i] / t;
if((bur[i]%t) != 0)
b[i] += 1;
```

```
m += b[i];
}
printf("\n\t\tRound Robin Scheduling\n");
printf("\nGANTT Chart\n"); for(i=0; i<m; i++)
printf("");
printf("\n");
a[0] = 0;
while(j < m)
{
if(x == n-1)
x = 0;
else
x++;
if(bur[x] >= t)
{
bur[x] -= t;
a[j+1] = a[j] + t;
 if(b[x] == 1)
{
p[s] = x;
k[s] = a[j+1]; s++;
}
j++;
b[x] -= 1;
printf("P%d|", x+1);
}
else if(bur[x] != 0)
{
a[j+1] = a[j] + bur[x]; bur[x] = 0;
if(b[x] == 1)
{
p[s] = x;
k[s] = a[j+1]; s++;
}
j++;
b[x] -= 1;
printf("P%d|",x+1);
}
}
printf("\n");
for(i=0;i<m;i++)
printf("");
printf("\n");
for(j=0; j<=m; j++)
printf("%d\t", a[j]);
```

```
for(i=0; i<n; i++)
{
for(j=i+1; j<n; j++)
{
if(p[i] > p[j])
{
temp = p[i];
p[i] = p[j];
p[j] = temp;
temp = k[i];
k[i] = k[j];
k[j] = temp;
}
}
}
 for(i=0; i<n; i++)
{
wat[i] = k[i] - bur1[i];
tur[i] = k[i];
}
for(i=0; i<n; i++)
{
ttur += tur[i];
twat += wat[i];
}
printf("\n\n");
for(i=0; i<30; i++)
printf("-");
printf("\nProcess\tBurst\tTrnd\tWait\n");
for(i=0; i<30; i++)
printf("-");
for (i=0; i<n; i++)
printf("\nP%-4d\t%4d\t%4d\t%4d", p[i]+1, bur1[i], tur[i],wat[i]);
printf("\n"); for(i=0; i<30; i++)
printf("-");
awat = (float)twat / n;
atur = (float)ttur / n;
printf("\n\nAverage waiting time: %.2f ms", awat);
printf("\nAverage turn around time : %.2f ms\n", atur);
return 0;
}
```

OUTPUT:

```
Enter no.of process : 3
Burst time for process P1(in ms) : 4
Burst time for process P1(in ms) : 3
Burst time for process P1(in ms) : 7
Enter the time slice (in ms)     : 2
```

Round Robin Scheduling:

Process	B-Time	T-Time	W-Time
P1	4	8	4
P2	3	9	6
P3	7	14	7

Average waiting time: 5.67 ms
Average turn around time :10.33 ms

GANTT Chart

P1	P2	P3	P1	P2	P3	P3	P3

0 2 4 6 8 9 11 13 14

RESULT:

Thus a C program for SJF CPU scheduling algorithm was written and its output was verified.

Ex.No: 06

Date: **IMPLEMENTATION OF SEMAPHORES**

AIM:

To Write a C program for the implementation of Semaphores

ALGORITHM:

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

PROGRAM :

```
#define BUFFERSIZE 10
int mutex,n,empty,full=0,item,item1;
int buffer[20];
int in=0,out=0,mutex=1;
void wait(int s)
{
while(s<0)
{
printf("\nCannot add an item\n");
exit(0);
}
s--;
}
void signal(int s)
{
s++;
}
void producer()
{
do
```

```
{
wait (empty);
wait(mutex);
printf("\nEnter an item:");
scanf("%d",&item);
buffer[in]=item;
in=in+1;
signal(mutex);
signal(full);
}
while(in<n);
}
void consumer()
{
do
{
wait(full);
wait(mutex);
item1=buffer[out];
printf("\nConsumed item =%d",item1);
out=out+1;
signal(mutex);
signal(empty);
}
while(out<n);
}
void main()
{
printf("Enter the value of n:");
scanf("%d ",&n);
empty=n;
while(in<n)
producer();
while(in!=out)
consumer();
}
```

OUTPUT:

```
$ cc prco.c
$ a.out
Enter the value of n :3
Enter the item:2
Enter the item:5
Enter the item:9
consumed item=2
consumed item=5
consumed item=9
```

RESULT:

Thus a C program for Semaphore was written and its output was verified.

Ex.No: 7a

Date: **IMPLEMENTATION OF SHARED MEMORY**

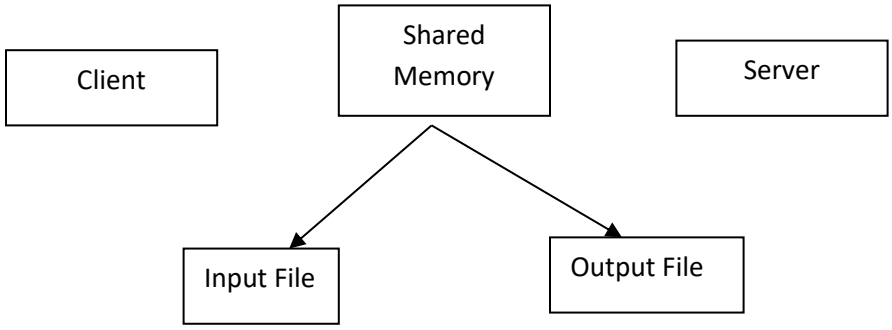
AIM:

Write a C program to implement two or more processes share the same area in the memory using shared memory.

DESCRIPTION:

Shared memory is a way of allowing two or more processes to share on the same area in memory. The processes have to coordinate the use of memory among themselves. For example,

If one process is reading into shared memory, another process must wait for the read to finish.



A Client Server relationship through shared memory has the following steps:

1. The server accesses a shared memory segment using a semaphore
2. The server reads the data from the input file into the shared memory segment
3. After the read finishes, the server signals the client using a semaphore
4. Finally, the client transfers the data from segment into output file

The following are some of the built in functions used in shared memory.

Shmget():

It is used to create a shared memory segment. It returns an identifier to the segment. It passes 3 parameters.

1. Key, which is the name given to the segment
2. Number of butes to reserve for the lock
3. Flag, which is either
    a. IPC_CREATE – which means that the shared memory is to be created if does not exist else it shows error.
    b. IPC_EXCL – which indicates error if the shared memory is already exists.

Shmmat():

To map the process to address the space of our process.

It passes 3 parameters.

1. Id of the shared memory and two 0's which returns a pointer to the shared memory segment.
2. It is used to map the segments to processes all we get in memory

Read():

It passes 3 parameters.

1. Standard Input (Keyboard)
2. The address of the segment
3. Number of bytes to be read

Write():

It passes 3 parameters.

1. Standard Output(Screen)
2. The address of the segment
3. Number of bytes to be read

PROGRAM:

a. Creating shared memory

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/shm.h>
main()
{
int shmid;
key_t key=0*20;
shmid=shmget(key,10,IPC_CREAT/0666);
if(shmid<0)
{
perror("shmget failed\n");
exit(1);
}
printf("success,shmid is %d\n",shmid);
}
```

OUTPUT:

Success, shmid is 65537

b. To change the bytes allocation

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/shm.h>
main()
{
int shmid;
key_t key=0*30;
shmid=shmget(key,10,IPC_CREAT/0666);
if(shmid<0)
{
perror("shmget failed\n");
exit(1);
}
printf("success,shmid is %d\n",shmid);
shmid=shmget(key,15,IPC_CREAT/0666);
if(shmid<0)
perror("\n size is increased, Hemce error\n");
else
printf("success,shmid is %d\n",shmid);
shmid=shmget(key,5,IPC_CREAT/0666);
if(shmid<0)
perror("error in reducing size\n");
else
printf("\nsuccess,shmid is%d:",shmid);
return 0;
}
```

OUTPUT:

```
Success, shmid is 13075
Success, shmid is 13075
Success, shmid is 13075
```

c. Reading and writing from shared memory

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/shm.h>
main()
{
int shmid,pid;
char*ptr;
shmid=shmget((key_t)0*45,20,IPC_CREAT/0666);
ptr=(char*)shmat(shmid,(char*)0,0);
```

```
pid=foke();
if(pid==0)
strcpy(ptr,"hello");
else
{
wait(0);
printf("parent process reads \n %s\n",ptr);
}
}
```

OUTPUT:

Parent process reads
Hello

d. Using read and write function

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/shm.h>
main()
{
int shmid,pid;
char*ptr;
shmid=shmget((key_t)0*60,20,IPC_CREAT/0666);
ptr=(char*)shamt(shmid,(char*)0,0);
pid=fork();
if(pid==0)
read(0,ptr,40);
else
{
wait(10);
printf("parent process reads content of shm\n");
write(1,ptr,40);
}
}
```

OUTPUT:

Hello Welcome to cse
Parent process reads contents of shm
Hello Welcome to cse

RESULT:

Thus the concept of shared memory was studied and the output was verified

Ex.No: 7b

Date: **IMPLEMENTATION OF INTER PROCESS COMMUNICATION**

AIM:

Write a C program to Implement inter process communication problem using semaphores

ALGORITHM:

Producer routine:

1. Get the element to be inserted into a buffer.
2. Lock the buffer to support mutual exclusion using semaphore Calculate the waiting time, average of time for each process.
3. Semaphore is a variable that has integer value which has two operations wait, signal
4. Put the element into buffer until buffer become FULL.
5. Unlock buffer

Consumer routine:

1. Before retrieving element from buffer locked it.
2. Retrieve the element from buffer until buffer become empty
3. Buffer is a FIFO queue, first come first served.
4. Unlock buffer

Disply routine:

1. Display the contents of buffer program.

PROGRAM:

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#define SIZE 10
struct process
{
char a[10];
}buffer[10];
int mutex=1,full=0,empty=SIZE,flag=0;
int signal(int);
int wait(int);
main()
{
int ch,i;
```

```
while(1)
{
printf("\nchoice");
printf("\n 1.producer routine");
printf("\n 2.Consumer routine");
printf("\n 3.Display the content of the buffer");
printf("\n 4. Exit");
printf("\n enter the choice");
scanf("%d",&ch);
switch(ch)
{
case 1:
empty=wait(empty);
mutex=wait(mutex);
if(flag==0)
{
printf("\n enter the element to be added");
scanf("%s",&buffer[full]);
printf("\n item is successfully added!");
full=signal(full);
}
else
{
printf("\n buffer is full");
flag=0;
}
mutex=signal(mutex);
break;
case 2:
full=wait(full);
mutex=wait(mutex);
if(flag==0)
{
for(i=0;i<SIZE;i++)
{
strcpy(buffer[i].a,buffer[i+1].a);
}
empty=signal(empty);
}
else
{
printf("\n no item in the buffer!");
flag=0;
}
mutex=signal(mutex);
break;
```

```
case 3:
if(full!=0)
{
for(i=0;i<full;i++)
{
printf("\n %s",buffer[i].a);
}
}
else
printf("Buffer is empty");
break;
case 4:
exit(0);
break;
default:printf("\n enter proper option");
}
}
}
int wait(s)
{
if(s==0)
{
flag=1;
}
else
{
s--;
}
return s;
}
int signal(s)
{
s++;
return s;
}
```

OUTPUT:

```
choice
1.producer routine 2.consumer routine 3.disply the contents of buffer 4.exit
 enter the choice:1
 enter the element to be added:34
 item is successfully added!

choice
1.producer routine 2.consumer routine 3.disply the contents of buffer 4.exit
 enter the choice:1
 enter the element to be added:56
 item is successfully added!

choice
1.producer routine 2.consumer routine 3.disply the contents of buffer 4.exit
 enter the choice:2
 item 34 is successfully is consumed!

choice
1.producer routine 2.consumer routine 3.disply the contents of buffer 4.exit
 enter the choice:3
 56

choice
1.producer routine 2.consumer routine 3.disply the contents of buffer 4.exit
 enter the choice:2
 item 56 is successfully is consumed!

choice
1.producer routine 2.consumer routine 3.disply the contents of buffer 4.exit
 enter the choice:2
 no item in the buffer!

choice
1.producer routine 2.consumer routine 3.disply the contents of buffer 4.exit
 enter the choice:4
[sec@Telnet ~]$
```

RESULT:

Thus the inter process communication application executed successfully and the output was verified

Expt No: 08

Date: **IMPLEMENTATION OF BANKERS ALGORITHM**

AIM:

Write a C program to Implement Bankers Algorithm for Dead Lock Avoidance

ALGORITHM:

1. Start
2. Enter the number of resources and then according to this read the resources available to each resource type.
3. Enter the number of processes and according to this and the number of Resources read the maximum claim (row->process, column->resource)
4. Enter the current allocation table to 2D alloc [][].
5. Compute the availability of each resource by total unit of particular resource – column sum of that resource as index in alloc[][] .
6. If any process need some resource type some unit and the resources are available then the resources are allocated to that process.
7. After the process's processing , all the resources are released and hence consist units of resources are increased.
8. If all processes can process , then the state in safe else not in safe state.
9. Stop.

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;
clrscr();
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("Enter no. 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("Enter the allocation matrix");
for(i=1; i<=np; i++)
{
for(j=1; j<=nr; j++)
{
scanf("%d", &all[i][j]);
av[0][j] = av[0][j] - all[i][j];
}
}
printf("Enter the maximum matrix");
for(i=1; i<=np; i++)
{
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;
}}}
getch();
return 0;
}
```

OUTPUT:

```
Enter the no of processes  5
Enter the no of resources instance  3

Enter the Max Matrix
7  5  3
3  2  2
9  0  2
2  2  2
4  3  3

Enter the Allocation Matrix
0  1  0
2  0  0
3  0  2
2  1  1
0  0  2

Enter the available Resources
3  3  2

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

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

The system is in Safe state
```

RESULT

Thus the C program for implementation of Banker's algorithm using deadlock avoidance was executed and output was verified.

Ex. No: 9

Date: **IMPLEMENTATION OF DEADLOCK DETECTION ALGORITHM (RESOURCE ALLOCATION GRAPH)**

AIM:

Write a C program for implementing resource allocation graph which is used to detect the deadlock

ALGORITHM:

1. Mark each process that has a row in the Allocation matrix of all zeros.
2. Initialize a temporary vectorW to equal the Available vector.
3. Find an index*i* such that process*i* is currently unmarked and the*i* th row ofQ is less than or equal to W . That is,*Q ik … Wk*, for 1 … *k* … *m* . If no such row is found, terminate the algorithm.
4. If such a row is found, mark process*i* and add the corresponding row of the allocation matrix to W . That is, set*Wk = Wk* + A*ik*, for 1 … *k* … *m* . Return to step 3.
5. Stop the program.

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];
int deadlock=0;
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;
}
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];
}
}
}
for(i=0;i<np;i++)
if(mark[i]!=1)
deadlock=1;
if(deadlock)
printf("\n Deadlock detected");
else
printf("\n No Deadlock possible");
return 0;
}
```

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 C program for implementing resources allocation graph algorithm was verified and output was observed.

Ex.No: 10

Date: **IMPLEMENTATION OF THREADING & SYNCHRONIZATION APPLICATIONS**

AIM:

Write a C program to implement threading & synchronization applications

ALGORITHM:

1. Start the Program
2. Obtain the required data through char and in data types
3. Enter the filename, index block
4. Print the file index loop
5. File is allocated to the unused index blocks
6. This is allocated to the unused linked blocks
7. Stop the execution

PROGRAM:

```
#include <stdio.h>
#include <pthread.h>
#include<pthread.h>
#include<stdlib.h>
#include<stdlib.h>
#include<unistd.h>
pthread_ t tid[2]
int counter;
void *dosomething(void *arg)
{
unsigned long i=0;
counter+=1;
printf("\n Job %d started \n", counter);
for(i=0; i<(0xFFFFFFFF);i++)
printf("\n Job %d finished \n", counter);
return NULL;
}

int main(void)
{
int i=0;
int err;
while(i<2)
{
err=pthread_create(&(tid[i]),NULL,&dosomething,NULL);
if(err!=0)
printf("\n Can't create thread : [%s]", strerror(err));
```

```
i++;
}
pthread_join(tid[0],NULL);
pthread_join(tid[1],NULL);
return 0;
}
```

OUTPUT:

```
Job 1 started
Job 2 started
Job 1 finished
Job 2 finished
```

RESULT:

Thus the C program for implementation of Threading & Synchronization was written and output was verified.

Ex. No: 11.a

Date: **IMPLEMENTATION OF MEMORY ALLOCATION METHODS USING FIRST FIT STRATEGY**

AIM:

Write a C program to implement memory management technique using first fit strategy

ALGORITHM:

1. Start the program.
2. Read the number of process and number of blocks.
3. Get the size of each block and process requests.
4. Allocate the processes
5. If block size >= process size , allocate the process
6. Otherwise display the processes with the blocks that are allocated to a respective process
7. Stop the program.

PROGRAM:

```
#include<stdio.h>
void main()
{
int bsize[10], psize[10], bno, pno, flags[10], allocation[10], i, j;
for(i = 0; i < 10; i++)
{
flags[i] = 0;
allocation[i] = -1;
}
printf("Enter no. of blocks: ");
scanf("%d", &bno);
printf("\nEnter size of each block: ");
for(i = 0; i < bno; i++)
scanf("%d", &bsize[i]);
printf("\nEnter no. of processes: ");
scanf("%d", &pno);
printf("\nEnter size of each process: ");
for(i = 0; i < pno; i++)
scanf("%d", &psize[i]);
for(i = 0; i < pno; i++)        //allocation as per first fit
for(j = 0; j < bno; j++)
if(flags[j] == 0 && bsize[j] >= psize[i])
{
allocation[j] = i;
flags[j] = 1;
break;
```

```
}
//display allocation details
printf("\nBlock no.\tsize\t\tprocess no.\t\tsize");
for(i = 0; i < bno; i++)
{
printf("\n%d\t\t%d\t\t", i+1, bsize[i]);
if(flags[i] == 1)
printf("%d\t\t\t%d",allocation[i]+1,psize[allocation[i]]);
else
printf("Not allocated");
}
}
```

OUTPUT:

```
Enter no. of blocks : 3
Enter size of each blocks :
12
7
4
Enter no. of processes: 3
Enter size of each process:
7
4
9

Block no        size            process no      size
   1             12                1             7
   2              7                2             4
   3              4          Not allocated
```

```
Process exited with return value 3
Press any key to continue . . .
```

RESULT

Thus the C program for implementation of memory management technique using first fit strategy was written and output was verified.

Ex.No: 11.b

Date: **IMPLEMENTATION OF MEMORY ALLOCATION METHODS USING WORST FIT STRATEGY**

AIM:

Write a C program to implement memory management technique using worst fit strategy

ALGORITHM:

1. Start the program.
2. Input memory block with a size
3. Input process with size
4. Initialize by selecting each process to find the maximum block size that can be assigned to the current process
5. If the condition does not fulfill, they leave the process
6. If the condition is not fulfilled, then leave the process and check for the next process
7. Stop the program.

PROGRAM:

```
#include<stdio.h>
int main()
{
int fragments[10], blocks[10], files[10];
int m, n, number_of_blocks, number_of_files, temp, top = 0;
static int block_arr[10], file_arr[10];
printf("\nEnter the Total Number of Blocks:\t");
scanf("%d",&number_of_blocks);
printf("Enter the Total Number of Files:\t");
scanf("%d",&number_of_files);
printf("\nEnter the Size of the Blocks:\n");
for(m = 0; m < number_of_blocks; m++)
{
printf("Block No.[%d]:\t", m + 1);
scanf("%d", &blocks[m]);
}
printf("Enter the Size of the Files:\n");
for(m = 0; m < number_of_files; m++)
{
printf("File No.[%d]:\t", m + 1);
scanf("%d", &files[m]);
}
for(m = 0; m < number_of_files; m++)
{
```

```
for(n = 0; n < number_of_blocks; n++)
{
if(block_arr[n] != 1)
{
temp = blocks[n] - files[m];
if(temp >= 0)
{
if(top < temp)
{
file_arr[m] = n;
top = temp;
}
}
}
fragments[m] = top;
block_arr[file_arr[m]] = 1;
top = 0;
}
}
printf("\nFile Number\tFile Size\tBlock Number\tBlock Size\tFragment");
for(m = 0; m < number_of_files; m++)
{
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", m, files[m], file_arr[m], blocks[file_arr[m]],
fragments[m]);
}
printf("\n");
return 0;
}
```

OUTPUT:

```
Enter the total number of blocks:
5
Enter total number of processes
4

Enter the size of the blocks:
Block number [1]: 5
Block number [2]: 4
Block number [3]: 3
Block number [4]: 6
Block number [5]: 7

Enter the size of the processes:
Process number [1]: 1
Process number [2]: 3
Process number [3]: 5
Process number [4]: 3
```

Process Number	Process Size	Block Number	Block Size	Fragment
0	1	4	7	6
1	3	0	5	0
2	5	0	5	0
3	3	0	5	0

RESULT

Thus the C program for implementation of memory management technique using Worst fit strategy was written and output was verified.

Ex. No: 11.c

Date: **IMPLEMENTATION OF MEMORY ALLOCATION METHODS USING BEST FIT STRATEGY**

AIM:

Write a C program to Implement memory management technique using best fit strategy

ALGORITHM:

1. Input memory block with a size.
2. Get no. of Processes and no. of blocks.
3. After that get the size of each block and process requests.
4. Then select the best memory block that can be allocated using the above definition.
5, Display the processes with the blocks that are allocated to a respective process.
6. Value of Fragmentation is optional to display to keep track of wasted memory.
7. Stop.

PROGRAM:

```
#include<stdio.h>

void main()
{
int fragment[20],b[20],p[20],i,j,nb,np,temp,lowest=9999;
static int barray[20],parray[20];

printf("\n\t\t\tMemory Management Scheme - Best Fit");
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of processes:");
scanf("%d",&np);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
printf("Block no.%d:",i);
scanf("%d",&b[i]);
}
printf("\nEnter the size of the processes :-\n");
for(i=1;i<=np;i++)
{
printf("Process no.%d:",i);
scanf("%d",&p[i]);
}
for(i=1;i<=np;i++)
{
```

```
for(j=1;j<=nb;j++)
{
if(barray[j]!=1)
{
temp=b[j]-p[i];
if(temp>=0)
if(lowest>temp)
{
parray[i]=j;
lowest=temp;
}
}
}
fragment[i]=lowest;
barray[parray[i]]=1;
lowest=10000;
}
printf("\nProcess_no\tProcess_size\tBlock_no\tBlock_size\tFragment");
for(i=1;i<=np && parray[i]!=0;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,p[i],parray[i],b[parray[i]],fragment[i]);
}
```

OUTPUT:

Memory Management Scheme : Best Fit

Enter the total number of blocks:
3
Enter total number of processes
2

Enter the size of the blocks:
Block number [1]: 2
Block number [2]: 3
Block number [3]: 5

Enter the size of the processes:
Process number [1]: 2
Process number [2]: 6

Process Number	Process Size	Block Number	Block Size	Fragment
1	2	1	1	0

RESULT:

Thus the C program for implementation of memory management technique using Worst fit strategy was written and output was verified.

Expt No: 12

Date: **IMPLEMENTATION OF PAGING TECHNIQUE USING MEMORY MANAGEMENT**

AIM:

Write a C program to implement paging technique using memory management

ALGORITHM:

1. Start the program.
2. Read the base address, page size, number of pages and memory unit.
3. If the memory limit is less than the base address display the memory limit is less than limit.
4. Create the page table with the number of pages and page address.
5. Read the page number and displacement value.
6. If the page number and displacement value is valid, add the displacement value with the address corresponding to the page number and display the result.
7. Display the page is not found or displacement should be less than page size.
8. Stop the program.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
main()
{
int np,ps,i;
int *sa;
clrscr();
printf("Enter how many pages:\n");
scanf("%d",&np);
printf("Enter the page size: \n");
scanf("%d",&ps);
sa=(int*)malloc(2*np);
for(i=0;i<np;i++)
{
sa[i]=(int)malloc(ps);
printf("page%d\t address %u\n",i+1,sa[i]);
}
getch();
return 0;
}
```

OUTPUT:

```
Enter how many pages: 3
 Enter page size : 2
Page1 address : 1904
Page2 address : 1912
Page3 address : 1920
```

RESULT

Thus the C program for implementation using Paging with memory management technique was written and output was verified.

Expt No: 13

Date: **IMPLEMENTATION OF PAGE REPLACEMENT ALGORITHM (FIFO, LRU, LFU)**

AIM:

Write a C program to implement all page replacement algorithms
a) FIFO b) LRU c) LFU

ALGORITHM:

1. Get the reference string to be accessed in program execution.
2. Get the number of blocks.
3. Using FIFO page replacement, allocate memory frames for all pages in the reference string by selecting the first page that is placed in memoryblock for replacement, if FIFO() method is called.
4. Using OPTIMAL page replacement, allocate memory frames for all pages in the reference string by selecting the page that will not be placed in memory block for a long time for replacement, if OPTIMAL() method is called.
5. Using LRU page replacement, allocate memory frames for all pages in the reference string by selecting the page that is not recently used for replacement, if LRU() method is called.
6. Display the output.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int g=0,a[5],b[20],p=0,q=0,m=0,q1=1;
int h,k,i,j,u;
char f='F';
clrscr();
printf("Enter numbers:");
for(i=0;i<12;i++)
scanf("%d",&b[i]);
for(i=0;i<12;i++)
{
if(p==0)
{
if(q>=3)
q=0;
a[q]=b[i];
q++;
if(q1<3)
```

```
{
q1=q;
g=1;
}
}
printf("\n%d",b[i]);
printf("\t");
for(h=0;h<q1;h++)
printf("%d",a[h]);
if((p==0)&&(q1==3)&&(g!=1))
{
printf("-->%c",f);
m++;
}
p=0;
g=0;
if(q1==3)
{
for(k=0;k<q-1;k++)
{
if(b[i+1]==a[k])
p=1;
}
for(j=0;j<q1;j++)
{
u=0;
k=i;
while(k>(i-2)&&(k>=0))
{
if(b[k]==a[j])
u++;
k--;
}
if(u==0)
q=j;
}}
else
{
for(k=0;k<q;k++)
{
if(b[i+1]==a[k])
p=1;
}}}
printf("\nNo.of Faults=%d",m);
getch();
}
```

OUTPUT:

```
PAGE REPLACEMENT ALGORITHM
_____
ENTER TOTAL NUMBER OF PAGES : 20
ENTER 20 PAGE NUMBERS :
7
0
1
2
0
3
0
4
2
3
0
3
2
1
2
0
1
7
0
1
THE REFERENCE STRING :
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
1. FIFO ALGORITHM
2. OPTIMAL PAGE ALGORITHM
3. LRU PAGE ALGORITHM
4. EXIT
CHOICE : 1
_____
FIFO PAGE REPLACEMENT ALGORITHM
_____
ENTER THE NUMBER OF THE BLOCKS : 3
_____
REFERENCE STRING PAGE FRAME FAULT
_____
7 7 +
0 7 0 -
1 7 0 1 -
2 0 1 2 +
0 -
3 1 2 3 +
```

0 2 3 0 +
4 3 0 4 +
2 0 4 2 +
3 4 2 3 +
0 2 3 0 +
3 -
2 -
1 3 0 1 +
2 0 1 2 +
0 -
1 –
7 1 2 7 +
0 2 7 0 +
1 7 0 1 +

TOTAL NUMBER OF PAGES : 20 FAULT : 15
PAGE FAULT FREQUENCY : 0.750
WANT TO CONTINUE (y/n) : y
THE REFERENCE STRING :
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
1. FIFO ALGORITHM
2. OPTIMAL PAGE ALGORITHM
3. LRU PAGE ALGORITHM
4. EXIT
CHOICE : 2

OPTIMAL PAGE REPLACEMENT ALGORITHM

ENTER THE NUMBER OF THE BLOCKS : 3

REFERENCE STRING PAGE FRAME FAULT

---

7 7 +
0 7 0 -
1 7 0 1 -
2 2 0 1 +
0 -
3 2 0 3 +
0 -
4 2 4 3 +
2 -
3 -
0 2 0 3 +
3 -
2 -
1 2 0 1 +

2 -
0 -
1 -
7 7 0 1 +
0 -
1 -

TOTAL NUMBER OF PAGES : 20 FAULT : 9
PAGE FAULT FREQUENCY : 0.450
WANT TO CONTINUE (y/n) : y
THE REFERENCE STRING :
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
1. FIFO ALGORITHM
2. OPTIMAL PAGE ALGORITHM
3. LRU PAGE ALGORITHM
4. EXIT
CHOICE : 3

LRU PAGE REPLACEMENT ALGORITHM

ENTER THE NUMBER OF THE BLOCKS : 3

REFERENCE STRING PAGE FRAME FAULT

---

7 7 +
0 7 0 -
1 7 0 1 -
2 2 0 1 +
0 -
3 2 0 3 +
0 -
4 4 0 3 +
2 4 0 2 +
3 4 3 2 +
0 0 3 2 +
3 -
2 -
1 1 3 2 +
2 -
0 1 0 2 +
1 -
7 1 0 7 +
0 -
1 -

TOTAL NUMBER OF PAGES : 20 FAULT : 12

PAGE FAULT FREQUENCY : 0.600
WANT TO CONTINUE (y/n) : n

RESULT

Thus the C program for implementation of Page replacement Technique using FIFO, LRU and LFU was written and output was verified.

Expt No: 14.a

Date: **IMPLEMENTATION OF FILE ORGANIZATION TECHNIQUE USING SINGLE LEVEL DIRECTORY**

AIM:

Write a C program to implement file organization technique using single level directory.

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:

```
#include<stdio.h>
#include<conio.h>
main()
{
int master,s[20];
char f[20][20][20];
char d[20][20];
int i,j;
clrscr();
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");
printf("**************************************************\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");
getch();
return 0;
}
```

OUTPUT:

```
Enter the number of  the directory : 2
Enter name of the directory : aa      bb
Enter size of the directories :  2    3
Enter the file names :  a  b  c   d   e

Directory              Size                File Names
*******************************************
aa                      2                      a
                                               b

bb                      3                      c
                                               d
                                               e
```

RESULT

Thus the C program for implementation of file organization technique using single level directory was written and output was verified.

Expt No: 14.b

Date: **IMPLEMENTATION OF FILE ORGANIZATION TECHNIQUE USING TWO LEVEL DIRECTORY**

AIM:

Write a C program to implement file organization technique using two level directory.

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:

```
#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];
void main()
{
int i,j,k,n;
clrscr();
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");
}
}
}
```

OUTPUT:

```
Enter the number of  the directories : 2
Enter directory 1 name : hai
Enter size of the directories :  2

Enter the sub directory name and size:  a  2
Enter the file name : z
Enter the file name : y

Enter the sub directory name and size:  b  2
Enter the file name : x
Enter the file name : w


DirectoryName          Size          Sub-Dir          Size          Files
*************************************************************************
hai                    2             a                2             z    y
b                           2              x    w
```

RESULT

Thus the C program for implementation of file organization technique using two level directory was written and output was verified.

Ex,No: 15.a

Date: **IMPLEMENTATION OF FILE ALLOCATION STATEGIES USING SEQUENTIAL FILE ALLOCATION**

AIM:

Write a C program to implement file allocation strategies using sequential file allocation

ALGORITHM:

1. Start
2. Declare the starting block no. and the length of the file.
3. Get the Starting block no. and length of the file from the user.
4. Allocate files sequentially until end of the file.
5. Display the fragments of the file.
6. stop

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
main()
{
int n,i,j,b[20],sb[20],t[20],x,c[20][20];
clrscr();
printf("Enter no.of files:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter no. of blocks occupied by file%d",i+1);
scanf("%d",&b[i]);
printf("Enter the starting block of file%d",i+1);
scanf("%d",&sb[i]);
t[i]=sb[i];
for(j=0;j<b[i];j++)
c[i][j]=sb[i]++;
}
printf("Filename\tStart block\tlength\n");
for(i=0;i<n;i++)
printf("%d\t  %d \t%d\n",i+1,t[i],b[i]);
printf("Enter file name:");
scanf("%d",&x);
printf("File name is:%d",x);
printf("length is:%d",b[x-1]);
printf("blocks occupied:");
for(i=0;i<b[x-1];i++)
```

```
printf("%4d",c[x-1][i]);
getch();
}
```

OUTPUT:

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

RESULT

Thus the C program for implementation of file allocation strategies using sequential file allocation was written and output was verified.

Ex.No: 15.b

Date: **IMPLEMENTATION OF FILE ALLOCATION STATEGIES USING INDEXED FILE ALLOCATION**

AIM:

Write a C program to implement file allocation strategies using indexed file allocation

ALGORITHM:

1. Start
2. Declare the index block no. and total no.of files in a block
3. Get the index block no. and total no.of files in a block from the user.
4. Allocate files based on the index block no.
5. Arrange the files based on indexes which are created for each fragment of the file such that each and every similar indexed file is maintained by the primary index to provide the flow to file fragments.
6.stop

PROGRAM:

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

```
printf("Block occupied are:");
for(j=0;j<m[i];j++)
printf("%3d",b[i][j]);
getch();
return 0;
}
```

OUTPUT:

```
Enter no.of files : 2
Enter Starting block of file 1:3
Enter size of file 1: 2
Enter blocks of file 1:
5
6
Enter starting block of file 2: 1
Enter size of file 2: 3
Enter blocks of file 2:
1
2
3


File            Index           Length
  1               3               2
  2               1               3

Enter file name : 2
File name is : 2
Index is : 1

Block occupied are:
1       2       3
```

RESULT

Thus the C program for implementation of file allocation strategies using indexed file allocation was written and output was verified.

Ex. No: 15.c

Date: **IMPLEMENTATION OF FILE ALLOCATION STATEGIES USING LINKED FILE ALLOCATION**

AIM:

Write a C program to implement file allocation strategies using linked file allocation

ALGORITHM:

1. Create a queue to hold all pages in memory
2. When the page is required replace the page at the head of the queue
3. Now the new page is inserted at the tail of the queue
4. Create a stack
5.When the page fault occurs replace page present at the bottom of the stack
6. Stop the allocation.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
int f[50], p,i, st, len, j, c, k, a;
clrscr();
for(i=0;i<50;i++)
f[i]=0;
printf("Enter how many blocks already allocated: ");
scanf("%d",&p);
printf("Enter blocks already allocated: ");
for(i=0;i<p;i++)
{
scanf("%d",&a);
f[a]=1;
}
x: printf("Enter index starting block and length: ");
scanf("%d%d", &st,&len);
k=len;
if(f[st]==0)
{
for(j=st;j<(st+k);j++)
{
if(f[j]==0)
{
f[j]=1;
printf("%d-------->%d\n",j,f[j]);
```

```
}
else
{
printf("%d Block is already allocated \n",j);
k++;
}
}
}
else
printf("%d starting block is already allocated \n",st);
printf("Do you want to enter more file(Yes - 1/No - 0)");
scanf("%d", &c);
if(c==1)
goto x;
else
exit(0);
getch();
}
```

OUTPUT:

```
Enter how many blocks already allocated: 3
Enter blocks already allocated: 1 3 5
Enter index starting block and length: 2 2
2-------->1
3 Block is already allocated
4-------->1
Do you want to enter more file(Yes - 1/No - 0) 0
```

RESULT:

Thus the C program for implementation of file allocation strategies using Linked file allocation was written and output was verified.