**Ex No: 1 UNIX-OS COMMANDS**

**Aim:**

To perform the various basic unix-os commands.

**I.WORKING WITH DIRECTORIES**

**Create a directory:**

[cse414@cc2linux ~]$ mkdir d1

[cse414@cc2linux ~]$ mkdir d2

[cse414@cc2linux ~]$ ls

d1 d2 latest new

**Change the working directory:**

[cse414@cc2linux ~]$ cd d1

[cse414@cc2linux d1]$ l

[cse414@cc2linux d1]$ cat > d

operating system.

networks.

micro processors.

[cse414@cc2linux d1]$ ls

d

[cse414@cc2linux d1]$ cd ..

[cse414@cc2linux ~]$ cd

Move file within directories:

[cse414@cc2linux ~]$ mv new d2

[cse414@cc2linux ~]$ cd d2 d3

[cse414@cc2linux d2]$ ls

New

**FILE PERMISSIONS: (u-user,g-group,o-others) (r-read,w-write,x-execute)**

**(octal notation for read-4, write-2 ,execute-1)**

Syntax: **chmod** *ugo filename*

[cplab2@localhost ~]$ chmod 761 College(user has permission to do all,group has only read and write permission, others has only execute permission)

[cplab2@localhost ~]$ ls -l College

-rwxrw---x 1 cplab2 cplab2 39 Feb 1 11:44 College

[cplab2@localhost ~]$ chmod 463 College

[cplab2@localhost ~]$ ls -l College

-r--rw--wx 1 cplab2 cplab2 39 Feb 1 11:44 College

**II.WORKING WITH FILE RELATED COMMANDS**

**Creating a file:**

[cse414@cc2linux ~]$ cat > basic

multitasking is the capability of the operating system to perform various tasks.

A single user can perform various tasks.

every user can have a login name and a password .

operating system is used to have interface between system and user.

[cse414@cc2linux ~]$ cat > commands

working with file related commands.

process and status information command

filter command

working with directories.

[cse414@cc2linux ~]$ cat > pipes

if we want to perform multiple tasks we can go pipes and filters.

a pipe is a mechanism which takes the output of a command

its input for the next command.

**Display a file:**

[cse414@cc2linux ~]$ cat commands

working with file related commands.

process and status information command

filter command

working with directories.

**Append a file:**

[cse414@cc2linux ~]$ cat >> pipes

filters are used to extract the lines, which contain a specific pattern to arrange the contents.

[cse414@cc2linux ~]$ cat pipes

if we want to perform multiple tasks we can go pipes and filters. a pipe is a mechanism which takes the output of a command

its input for the next command.

filters are used to extract the lines,which contain a specific pattern to arrange the contents.

**Concatenation of files:**

[cse414@cc2linux ~]$ cat basic commands > pipes

[cse414@cc2linux ~]$ cat pipes

multitasking is the capability of the operating system to peerform various tasks.

A single user can perform various tasks.

every user can have a login name and a password .

working with file related commands.

process and status information command

filter command

working with directories.

**Copy a file:**

[cse414@cc2linux ~]$ cp basic commands

[cse414@cc2linux ~]$ cat commands

multitasking is the capability of the operating system to perform various tasks.

A single user can perform various tasks.

every user can have a login name and a password .

**Move a file:**

[cse414@cc2linux ~]$ mv basic commands

[cse414@cc2linux ~]$ cat basic

cat: basic: No such file or directory

[cse414@cc2linux ~]$ cat commands

multitasking is the capability of the operating system to perform various tasks.

A single user can perform various tasks.

every user can have a login name and a password .

**Comparing files:**

[cse414@cc2linux ~]$ cmp commands pipes

cmp: EOF on commands

**Difference between files:**

[cse414@cc2linux ~]$ diff commands pipes

6a7,12

> working with file related commands.

> process and status information command

> filter command

> working with directories.

**List :**

[cse414@cc2linux ~]$ list ls

-bash: list: command not found

[cse414@cc2linux ~]$ ls

commands pipes

**Remove:**

[cse414@cc2linux ~]$ rm pipes

[cse414@cc2linux ~]$ ls

commands

[cse414@cc2linux ~]$ rm -i commands

rm: remove regular file `commands'? n

[cse414@cc2linux ~]$ rm -r commands

**More:**

[cse414@cc2linux ~]$ more new old

::::::::::::::

new

::::::::::::::

multitasking is the capability of the operating system to peerform various tasks

.A single user can perform various tasks.

every user can have a login name and a password .

os is used to have interface between system and user.

old: No such file or directory

**Additional file name:**

[cse414@cc2linux ~]$ ln new latest

[cse414@cc2linux ~]$ ls

latest new

**Present working directory:**

[cse414@cc2linux ~]$ pwd

/home/cse414

**III.PROCESS AND STATUS INFORMATION COMMAND**

**Date:**

[cse414@cc2linux ~]$ date

Sun Dec 31 23:55:10 IST 2000

[cse414@cc2linux ~]$ date +%d

31

[cse414@cc2linux ~]$ date +%m

12

[cse414@cc2linux ~]$ date +%h

Dec

[cse414@cc2linux ~]$ date +%y

00

[cse414@cc2linux ~]$ date +%R

23:55

[cse414@cc2linux ~]$ date +%T

23:55:38

**Calendar:**

[cse414@cc2linux ~]$ cal

December 2000

Su Mo Tu We Th Fr Sa

1 2

3 4 5 6 7 8 9

10 11 12 13 14 15 16

17 18 19 20 21 22 23

24 25 26 27 28 29 30

31

[cse414@cc2linux ~]$ cal 2015

2015

January February March

Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa

1 2 3 1 2 3 4 5 6 7 1 2 3 4 5 6 7

4 5 6 7 8 9 10 8 9 10 11 12 13 14 8 9 10 11 12 13 14

11 12 13 14 15 16 17 15 16 17 18 19 20 21 15 16 17 18 19 20 21

18 19 20 21 22 23 24 22 23 24 25 26 27 28 22 23 24 25 26 27 28

25 26 27 28 29 30 31 29 30 31

April May June

Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa

1 2 3 4 1 2 1 2 3 4 5 6

5 6 7 8 9 10 11 3 4 5 6 7 8 9 7 8 9 10 11 12 13

12 13 14 15 16 17 18 10 11 12 13 14 15 16 14 15 16 17 18 19 20

19 20 21 22 23 24 25 17 18 19 20 21 22 23 21 22 23 24 25 26 27

26 27 28 29 30 24 25 26 27 28 29 30 28 29 30

31

July August September

Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa

1 2 3 4 1 1 2 3 4 5

5 6 7 8 9 10 11 2 3 4 5 6 7 8 6 7 8 9 10 11 12

12 13 14 15 16 17 18 9 10 11 12 13 14 15 13 14 15 16 17 18 19

19 20 21 22 23 24 25 16 17 18 19 20 21 22 20 21 22 23 24 25 26

26 27 28 29 30 31 23 24 25 26 27 28 29 27 28 29 30

30 31

October November December

Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa

1 2 3 1 2 3 4 5 6 7 1 2 3 4 5

4 5 6 7 8 9 10 8 9 10 11 12 13 14 6 7 8 9 10 11 12

11 12 13 14 15 16 17 15 16 17 18 19 20 21 13 14 15 16 17 18 19

18 19 20 21 22 23 24 22 23 24 25 26 27 28 20 21 22 23 24 25 26

25 26 27 28 29 30 31 29 30 27 28 29 30 31

**Binary calculator:**

[cse414@cc2linux ~]$ bc

bc 1.06

5\*4

20

**Who :**

[cse414@cc2linux ~]$ who

cse319 pts/1 Dec 31 23:04 (132.147.162.8)

cse318 pts/2 Dec 31 23:51 (132.147.162.7)

cse212 pts/3 Dec 31 23:04 (132.147.162.9)

cse316 pts/4 Dec 31 23:24 (132.147.162.5)

**Finger command:**

[cse414@cc2linux ~]$ finger

Login Name Tty Idle Login Time Office Office Phone

Cse303 cse303 pts/22 Dec 31 23:10 (132.147.162.11)

Cse322 cse322 pts/3 1 Dec 31 23:04 (132.147.162.9)

Cse324 cse324 pts/8 1 Dec 31 23:06 (132.147.162.10)

cse325 cse325 pts/13 Dec 31 23:19 (132.147.162.1)

cse326 cse326 pts/11 Dec 31 23:55 (132.147.162.2)

cse332 cse332 pts/9 Dec 31 23:06 (132.147.162.3)

cse335 cse335 pts/7 1 Dec 31 23:05 (132.147.162.4)

cse338 cse338 pts/4 Dec 31 23:24 (132.147.162.5)

cse340 cse340 pts/10 2 Dec 31 23:32 (132.147.162.27)

cse346 cse346 pts/2 Dec 31 23:51 (132.147.162.7)

**Who am i:**

[cse414@cc2linux ~]$ who am i

cse414 pts/11 Dec 31 23:55 (132.147.162.2)

**Man:**

[cse414@cc2linux ~]$ man date

DATE(1) User Commands DATE(1)

NAME

date - print or set the system date and time

SYNOPSIS

date [OPTION]... [+FORMAT]

date [-u|--utc|--universal] [MMDDhhmm[[CC]YY][.ss]]

DESCRIPTION

Display the current time in the given FORMAT, or set the system

date.

-d, --date=STRING

display time described by STRING, not ânowâ

-f, --file=DATEFILE

like --date once for each line of DATEFILE

[1]+ Stopped man date

**Word count:**

[cse414@cc2linux ~]$ wc new

wc new

6 85 557 new

[cse414@cc2linux ~]$ wc -w new

85 new

[cse414@cc2linux ~]$ wc -c new

557 new

[cse414@cc2linux ~]$ wc -l new

6 new

**Echo:**

[cse414@cc2linux ~]$ echo

**Read:**

[cse414@cc2linux ~]$ read a

5

[cse414@cc2linux ~]$ echo $a

5

**Terminal line:**

[cse414@cc2linux ~]$ tty

/dev/pts/11

**Adding line number to file:**

[cse414@cc2linux ~]$ nl new

1 multitasking is the capability of the operating system to peerform various tasks.A single user can perform various tasks.

2 every user can have a login name and a password .so, accessing another users data is impossible without permission.

**Which :**

[cse414@cc2linux ~]$ which date

/bin/date

[cse414@cc2linux ~]$ which new

/usr/bin/which: no new in (/usr/kerberos/bin:/usr/local/bin:/bin:/usr/bin:/usr

/X11R6/bin:/home/cse414/bin)

**IV.FILTER COMMAND**

**Sorting:**

[cse414@cc2linux ~]$ cat new

multitasking is the capability of the operating system to perform various tasks.

A single user can perform various tasks.

every user can have a login name and a password .

[cse414@cc2linux ~]$ sort new

every user can have a login name and a password .

multitasking is the capability of the operating system to perform various tasks.

A single user can perform various tasks.

[cse414@cc2linux ~]$ sort -r new

multitasking is the capability of the operating system to perform various tasks.

A single user can perform various tasks.

every user can have a login name and a password .

[cse414@cc2linux ~]$ sort -u new

every user can have a login name and a password .

multitasking is the capability of the operating system to perform various tasks.

A single user can perform various tasks.

[cse414@cc2linux ~]$ sort -n new

communication is between different terminals.

every user can have a login name and a password .

multitasking is the capability of the operating system to perform various tasks

.A single user can perform various tasks.

**Head:**

[cse414@cc2linux ~]$ head new

multitasking is the capability of the operating system to perform various tasks.

A single user can perform various tasks.

every user can have a login name and a password .

[cse414@cc2linux ~]$ head -2 new

multitasking is the capability of the operating system to perform various tasks.A single user can perform various tasks.

every user can have a login name and a password .so, accessing another users data is impossible without permission.

**Tail:**

[cse414@cc2linux ~]$ tail new

multitasking is the capability of the operating system to perform various tasks.

A single user can perform various tasks.

every user can have a login name and a password .

[cse414@cc2linux ~]$ tail -2 new

Necessary ingredients like conditional and control structures and variables.

communication is between different terminals.

**Cut:**

[cse414@cc2linux ~]$ cut -c1-3 new

mul

eve

UNI

os

UNI

com

[cse414@cc2linux ~]$ cut -r1-3 new

cut: invalid option -- r

Try `cut --help' for more information.

[cse414@cc2linux ~]$ ls

latest new

**Paste:**

[cse414@cc2linux ~]$ paste new latest

multitasking is the capability of the operating system to perform various tasks.A single user can perform various tasks. multitasking is the capability of the operating system to perform various tasks.A single user can perform various tasks.

**Grep:**

[cse414@cc2linux ~]$ grep "between" new

os is used to have interface between system and user.

communication is between different terminals.

**Paste:**

[cse414@cc2linux ~]$ paste -d "&" new latest

multitasking is the capability of the operating system to perform various tasks.A single user can perform various tasks.&multitasking is the capability of the operating system

.**&** os is used to have interface between system and user.&os is used to have interface between system and user

**Common:**

[cse414@cc2linux ~]$ comm new latest

multitasking is the capability of the operating system to peerform various tasks.A single user can perform various tasks.

every user can have a login name and a password .so, accessing another users data is impossible without permission.

os is used to have interface between system and user.

communication is between different terminals.

**Pipes:**

[cse414@cc2linux ~]$ who | wc -l

22

**ExNo: 2 Programs using various system calls of UNIX operating system.**

**Aim:**

To Write a programs using the following system calls of UNIX operating system fork, exec, getpid, exit, wait, close, stat, opendir, readdir.

**Algorithm:**

1. Start the program.
2. Read the input from the command line.
3. Use fork() system call to create process, getppid() system call used to get the parent process ID and getpid() system call used to get the current process ID
4. execvp() system call used to execute that command given on that command line argument
5. execlp() system call used to execute specified command.
6. Open the directory at specified in command line input.
7. Display the directory contents.
8. Stop the program.

**Program to implement opendir, readdir, closedir, etc.. SYSTEM CALLS**

#include<stdio.h>

#include<dirent.h>

struct dirent \*dptr;

int main(int argc, char \*argv[])

{

char buff[100];

DIR \*dirp;

printf(“\n\n ENTER THE DIRECTORY NAME:”);

scanf(“%s”, buff);

if((dirp=opendir(buff))==NULL)

{

printf(“The given directory does not exist”);

exit(1);

}

while(dptr=readdir(dirp))

{

printf(“%s\n”,dptr->d\_name);

}

closedir(dirp);}

**OUTPUT:**

[kalai@localhost ~]$ cc sys1.c

[kalai@localhost ~]$ ./a.out

ENTER THE DIRECTORY NAME:kanna

.

..

flower

**Program to Implement Fork, getpid, Exit, Etc…. SYSTEM CALLS**

#include<stdio.h>

#include<unistd.h>

main()

{

int pid,pid1,pid2;

pid=fork();

if(pid==-1)

{

printf(“ERROR IN PROCESS CREATION \n”);

exit(1);

}

if(pid!=0)

{

pid1=getpid();

printf(“\n THE PARENT PROCESS ID IS: %d\n”, pid1);

}

else

{

pid2=getpid();

printf(“\n THE CHILD PROCESS ID IS: %d\n”, pid2);

}

}

**OUTPUT:**

[kalai@localhost ~]$ cc sys2.c

[kalai@localhost ~]$ ./a.out

THE PARENT PROCESS ID IS: 8062

THE CHILD PROCESS ID IS: 8063

**Program to Implement open, read SYSTEM CALLS**

**1.OPEN SYSTEM CALL**

**PROGRAM:**

#include<stdio.h>

#include<stdlib.h>

int main()

{

int fd;

if((fd=open("prime.sh"))==-1)

{

perror("cannot open the file.dat");

exit(0);

}

else

printf("\n FILE OPENED SUCCESSSFULLY");

return 0;

}

**OUTPUT:**

[kalai@localhost ~]$ cc opdir.c

[kalai@localhost ~]$ ./a.out

FILE OPENED SUCCESSSFULLY

**2.READ SYSTEM CALL**

#include<stdio.h>

#include<stdlib.h>

main()

{

char b[20];

int fd,xr;

if((fd=open("flowers.sh",0))==-1)

{

printf("cannot open file");

exit(1);

}

do

{

xr=read(fd,b,20);

b[xr]='\0';

printf("%s",b);

}

while(xr==20);

close(fd);}

**OUTPUT:**

[kalai@localhost ~]$ cc sysread1.c

[kalai@localhost ~]$ ./a.out

rose

jasmine

lotus

**Ex No: 3 Implement the CPU scheduling algorithms**

**3A. RoundRobin**

**Aim:**To Write a c program to implement round robin CPU scheduling algorithms

# Algorithm for Round Robin

1. Start the process.
2. Accept the number of processes in the ready Queue and time quantum (or) time slice.
3. For each process in the ready Q, assign the process id and accept the CPU burst time
4. Calculate the no. of time slices for each process where

No. of time slice for process(n) = burst time process(n)/time slice

1. If the burst time is less than the time slice then the no. of time slices =1.
2. Consider the ready queue is a circular Q, calculate
3. 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)
4. Turn around time for process(n) = waiting time of process(n) + burst time of process(n)+ the time difference in getting CPU from process(n).
5. Calculate
6. Average waiting time = Total waiting Time / Number of process
7. Average Turnaround time = Total Turnaround Time / Number of process
8. Stop the process

**PROGRAM:**

#include <stdio.h>

main()

{

int i,f[10],bt[10],p[10],cbt[10],j,k,wt[10],tat[10];

int n,t,q,flag;

float atat,awt;

printf("\n\t\t Round Robin Scheduling \n\t");

printf("\n\t No. Of Process : ");

scanf(" %d",&n);

printf("\n\t Process Burst Time ");

for(i=1;i<=n;i++)

{

printf("\n\t P%d \t ",i);

scanf(" %d",&bt[i]);

cbt[i]=bt[i];

p[i]=i;

}

printf("\n\t Time Quantum : ");

scanf(" %d",&q);

for(i=1;i<=n;i++)

{

wt[i]=0;

tat[i]=0;

f[i]=0;

}

do

{

for(i=1;i<=n;i++)

{

if(f[i]==0)

{

if(bt[i]<=q)

{

f[i]=1;

t=bt[i];

}

else

t=q;

bt[i]=bt[i]-t;

for(j=1;j<=n;j++)

{

if((j==i)||(f[j]==1))

continue;

wt[j]=wt[j]+t;

}//for j

}//if finish

}//for i

flag=0;

for(i=1;i<=n;i++)

if(f[i]==0)

flag=1;

}while(flag==1);

atat=awt=0;

for(i=1;i<=n;i++)

{

tat[i]=wt[i]+cbt[i];

atat+=tat[i];

awt+=wt[i];

}

printf("\n\t Process Burst-Time Wait-Time Turnaround-Time\n");

for(i=1;i<=n;i++)

printf("\n\t P%d \t %d \t\t %d \t\t%d\n",p[i],cbt[i],wt[i],tat[i]);

printf("\n\t Average Turn Around Time : %f\n",atat/n);

printf("\n\t Average Waiting Time : %f\n",awt/n);

return 0;

}

**OUTPUT:**

[pec292@localhost ~]$ cc robin.c

[pec292@localhost ~]$ ./a.out

Round Robin Scheduling

No. Of Process : 3

Process Burst Time

P1 5

P2 8

P3 2

Time Quantum : 3

Process Burst-Time Wait-Time Turnaround-Time

P1 5 5 10

P2 8 7 15

P3 2 6 8

Average Turn Around Time : 11.000000

Average Waiting Time : 6.000000

**3B.Shortest Job First**

**Aim:**Write a c program to implement shortest job first CPU scheduling algorithms.

# Algorithm for SJF

1. Start the process
2. Accept the number of processes in the ready Queue
3. For each process in the ready Q, assign the process id and accept the CPU burst time
4. Start the Ready Q according the shortest Burst time by sorting according to lowest to

highest burst time.

1. Set the waiting time of the first process as ‘0’ and its turnaround time as its burst time.
2. For each process in the ready queue, calculate
3. Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n-1)
4. Turn around time for Process(n)= waiting time of Process(n)+ Burst time for process(n)
5. Calculate
6. Average waiting time = Total waiting Time / Number of process
7. Average Turnaround time = Total Turnaround Time / Number of process
8. Stop the process.

**PROGRAM:**

#include <stdio.h>

main()

{

int i,bt[10],p[10],cp[10],j,k,wt[10],tat[10];

int n,t;

float atat,awt;

printf("\n\t\t Shortest Job First\n\t");

printf("\n\t No. Of Process : ");

scanf(" %d",&n);

printf(" \n\t Process Burst Time\n");

for(i=1;i<=n;i++)

{

printf("\t P%d\t",i);

scanf(" %d",&bt[i]);

p[i]=i;

}

for(i=1;i<=n;i++)

{

for(j=1;j<=n;j++)

{

if(bt[i]<bt[j])

{

t=bt[i];

bt[i]=bt[j];

bt[j]=t;

t=p[i];

p[i]=p[j];

p[j]=t;

}

}

}

printf("\n\t Executed In Following Order\n");

printf("\n\t Process Burst Time \n");

for(i=1;i<=n;i++)

printf("\n\t P%d \t %d",p[i],bt[i]);

wt[1]=0;

tat[1]=bt[1];

atat=tat[1];

awt=0;

for(i=2;i<=n;i++)

{

wt[i]=wt[i-1]+bt[i-1];

tat[i]=wt[i]+bt[i];

awt+=wt[i];

atat+=tat[i];

}

printf("\n\t Process Burst-Time Wait-Time Turnaround-Time\n");

for(i=1;i<=n;i++)

printf("\n\t P%d \t %d \t\t %d \t\t%d\n",p[i],bt[i],wt[i],tat[i]);

printf("\n\t Average Turn Around Time : %f \n",atat/n);

printf("\n\t Average Waiting Time : %f \n",awt/n);

return 0;

}

**OUTPUT:**

[kalai@localhost ~]$ cc sjb.c

[kalai@localhost ~]$ ./a.out

Shortest Job First

No. Of Process : 3

Process Burst Time

P1 5

P2 2

P3 8

Executed In Following Order

Process Burst Time

P2 2

P1 5

P3 8

Process Burst-Time Wait-Time Turnaround-Time

P2 2 0 2

P1 5 2 7

P3 8 7 15

Average Turn Around Time : 8.000000

Average Waiting Time : 3.000000

**3C.first come first serve**

**Aim:**To Write a c program to implement first come first served CPU scheduling algorithms.

**Algorithm for FCFS scheduling:**

1. Start the process
2. Accept the number of processes in the ready Queue
3. For each process in the ready Q, assign the process id and accept the CPU burst time
4. Set the waiting of the first process as ‘0’ and its burst time as its turn around time
5. for each process in the Ready Q calculate
6. Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n-1)
7. Turn around time for Process(n)= waiting time of Process(n)+ Burst time for process(n)
8. Calculate
9. Average waiting time = Total waiting Time / Number of process
10. Average Turnaround time = Total Turnaround Time / Number of process
11. Stop the process

**PROGRAM:**

#include <stdio.h>

main()

{

int i,bt[10],p[10],cp[10],j,k,wt[10],tat[10];

int n,t;

float atat,awt;

printf("\n\t First In First Out");

printf("\n\t No. Of Process : ");

scanf(" %d",&n);

printf(" \n\t Process Burst Time\n");

for(i=1;i<=n;i++)

{

printf("\t P%d\t",i);

scanf(" %d",&bt[i]);

}

wt[1]=0;

tat[1]=bt[1];

atat=tat[1];

awt=0;

for(i=2;i<=n;i++)

{

wt[i]=wt[i-1]+bt[i-1];

tat[i]=wt[i]+bt[i];

awt+=wt[i];

atat+=tat[i];

}

printf("\n\t Process Burst-Time Wait-Time Turnaround-Time\n");

for(i=1;i<=n;i++)

printf("\n\t P%d \t %d \t\t %d \t\t%d\n",i,bt[i],wt[i],tat[i]);

printf("\n\t Average Turn Around Time : %f \n",atat/n);

printf("\n\t Average Waiting Time : %f \n",awt/n);

}

**OUTPUT:**

[pec292@localhost ~]$ cc firstcome.c

[pec292@localhost ~]$ ./a.out

First In First Out

No. Of Process : 3

Process Burst Time

P1 5

P2 3

P3 8

Process Burst-Time Wait-Time Turnaround-Time

P1 5 0 5

P2 3 5 8

P3 8 8 16

Average Turn Around Time : 9.666667

Average Waiting Time : 4.333333

**3D.Priority Scheduling**

**Aim:**To write a c program to implement Priority CPU scheduling algorithms.

**Algorithm for Priority Scheduling:**

1. Start the process
2. Accept the number of processes in the ready Queue
3. For each process in the ready Q, assign the process id and accept the CPU burst time
4. Sort the ready queue according to the priority number.
5. Set the waiting of the first process as ‘0’ and its burst time as its turn around time
6. For each process in the Ready Q calculate
7. Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n-1)
8. Turn around time for Process(n)= waiting time of Process(n)+ Burst time for process(n)
9. Calculate
10. Average waiting time = Total waiting Time / Number of process
11. Average Turnaround time = Total Turnaround Time / Number of process
12. Stop the process

**PROGRAM:**

#include <stdio.h>

main()

{

int i,bt[10],p[10],pr[10],cp[10],j,k,wt[10],tat[10];

int n,t;

float atat,awt;

printf("\n\t\t Priority Scheduling \n\t");

printf("\n\t No. Of Process : ");

scanf(" %d",&n);

for(i=1;i<=n;i++)

{

printf("\n\t For Process P%d : \n\tBurst Time : ",i);

scanf(" %d",&bt[i]);

printf("\t Priority : ");

scanf(" %d",&pr[i]);

p[i]=i;

}

for(i=1;i<=n;i++)

{

for(j=1;j<=n;j++)

{

if(pr[i]<pr[j])

{

t=bt[i];

bt[i]=bt[j];

bt[j]=t;

t=p[i];

p[i]=p[j];

p[j]=t;

t=pr[i];

pr[i]=pr[j];

pr[j]=t;

}

}

}

printf("\n\t Executed In Following Order\n");

printf("\n\t Process Burst Time Priority\n");

for(i=1;i<=n;i++)

printf("\n\t P%d \t %d \t %d",p[i],bt[i],pr[i]);

wt[1]=0;

tat[1]=bt[1];

atat=tat[1];

awt=0;

for(i=2;i<=n;i++)

{

wt[i]=wt[i-1]+bt[i-1];

tat[i]=wt[i]+bt[i];

awt+=wt[i];

atat+=tat[i];

}

printf("\n\t Process Burst-Time Wait-Time Turnaround-Time\n");

for(i=1;i<=n;i++)

printf("\n\t P%d \t %d \t\t %d \t\t%d\n",p[i],bt[i],wt[i],tat[i]);

printf("\n\t Average Turn Around Time : %f \n",atat/n);

printf("\n\t Average Waiting Time : %f \n",awt/n);

return 0;

}

**OUTPUT:**

[pec292@localhost ~]$ cc priority.c

[pec292@localhost ~]$ ./a.out

Priority Scheduling

No. Of Process : 3

For Process P1 :

Burst Time : 2

Priority : 1

For Process P2 :

Burst Time : 4

Priority : 3

For Process P3 :

Burst Time : 6

Priority : 2

Executed In Following Order

Process Burst Time Priority

P1 2 1

P3 6 2

P2 4 3

Process Burst-Time Wait-Time Turnaround-Time

P1 2 0 2

P3 6 2 8

P2 4 8 12

Average Turn Around Time : 7.333333

Average Waiting Time : 3.333333

**ExNo: 4 Implementation of Semaphores**

**Aim:** To write a c program to implement producer-consumer algorithm using semaphore.

**Algorithm for Producer – Consumer Problem**

1. Declare variable for producer & consumer.
2. Declare and define semaphore function for creation and destroy.
3. Declare and define semaphore function for creation and destroy
4. Define producer function.
5. Define consumer function
6. Call producer and consumer
7. Stop the execution

**PROGRAM:**

#include<stdio.h>

#include<stdlib.h>

 int mutex=1,full=0,empty=3,x=0;

 int main()

{

int n;

void producer();

void consumer();

int wait(int);

int signal(int);

printf("\n1.Producer\n2.Consumer\n3.Exit");

while(1)

{

printf("\nEnter your choice:");

scanf("%d",&n);

switch(n)

{

case 1: if((mutex==1)&&(empty!=0))

producer();

else

printf("Buffer is full!!");

break;

case 2: if((mutex==1)&&(full!=0))

consumer();

else

printf("Buffer is empty!!");

break;

case 3:

exit(0);

break;

}

}

return 0;

}

 int wait(int s)

{

return (--s);

}

int signal(int s)

{

return(++s);

}

 void producer()

{

mutex=wait(mutex);

full=signal(full);

empty=wait(empty);

x++;

printf("\nProducer produces the item %d",x);

mutex=signal(mutex);

}

 void consumer()

{

mutex=wait(mutex);

full=wait(full);

empty=signal(empty);

printf("\nConsumer consumes item %d",x);

x--;

mutex=signal(mutex);

}

**OUTPUT:**

-bash-4.1$ cc sema1.c

-bash-4.1$ ./a.out

1.Producer

2.Consumer

3.Exit

Enter your choice:1

Producer produces the item 1

Enter your choice:1

Producer produces the item 2

Enter your choice:1

Producer produces the item 3

Enter your choice:1

Buffer is full!!

Enter your choice:2

Consumer consumes item 3

Enter your choice:2

Consumer consumes item 2

Enter your choice:2

Consumer consumes item 1

Enter your choice:2

Buffer is empty!!

Enter your choice: 3

**ExNo: 5 Implementation of Shared Memory and IPC**

**5a.Inter-processcommunication**

**Aim:**To write a c program to implement inter process communication.

**Algorithm:**

1. Start the program
2. Read the input through command prompt
3. Send message using msgget() method
4. Receive the message using msgget() method
5. Display the result
6. Stop the program

**MSGSEND.C**

#include<sys/types.h>

#include<sys/ipc.h>

#include<sys/msg.h>

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

struct mymsg {

long mtype;

char data[50];

}message;

main() {

int qid,len;

qid=msgget((key\_t)107,IPC\_CREAT|0666);

printf("input message? ");

scanf("%s",message.data);

message.mtype=1;

len=strlen(message.data);

if(msgsnd(qid,&message,len,0) == -1) {

printf("MESSAGE FAILED");

exit(1);

}

else

printf("MESSAGE SENT");

}

**MSGRECEIVE.C**

#include<sys/types.h>

#include<sys/ipc.h>

#include<sys/msg.h>

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

struct mymsg {

long mtype;

char data[50];

}b;

int main()

{

int qid;

qid=msgget((key\_t)107,0666);

if(qid==-1)

{

printf("MESSAGE FAILED");

exit(0);

}

if(msgrcv(qid,&b,13,1,IPC\_NOWAIT) == -1)

printf("MESSAGE FAILED");

else

printf("MESSAGE RECIEVED %s \n",b.data);

}

**OUTPUT:**

[kalai@localhost ~]$ cc -o send msgsend.c

[kalai@localhost ~]$ ./send

input message? hi

MESSAGE SENT[kalai@localhost ~]$ cc -o rece msgreceive.c

[kalai@localhost ~]$ ./rece

MESSAGE RECIEVED welcome

**5b.Shared memory**

**Aim:**Write a c program to implement shared memory algoritham

**Algorithm for Shared Memory**

1. Create the shared memory for parent process using shmget()system call.

2. Now allow the parent process to write in shared memory using shmget pointer which is return

type of shmget()

3. Now across and attach the same shared memory to the child process

4. The data in the shared memory is read by the child process using the shmdt pointer

5. Now detach and reuse the shared memory.

**PROGRAM**

#include<stdio.h>

#include<stdlib.h>

#include <unistd.h>

#include<string.h>

#define MAXLINE 100

int main(void){

int n,i;

int fd[2];

pid\_t pid;

char line[MAXLINE],msg[MAXLINE];

if (pipe(fd) < 0)

{

printf("pipe error");

return 0;

}

if ((pid = fork()) < 0)

{

printf("fork error");

return 0;

}

else if (pid > 0)

{ /\* parent \*/

close(fd[0]);

for(i=0;i<5;i++)

{

printf("\ninput string? ");

scanf("%s",msg);

write(fd[1], msg , strlen(msg));

sleep(1);

}

}

else

{ /\* child \*/

close(fd[1]);

for(i=0;i<5;i++)

{

n = read(fd[0], line, MAXLINE);

write(STDOUT\_FILENO, line, n);

}

}

exit(0);

}

**OUTPUT:**

[kalai@localhost ~]$ cc share.c

[kalai@localhost ~]$ ./a.out

input string? hi

hi

input string? hello

hello

input string? 1

1

input string? 45

45

input string? @

@

**ExNo: 6 Implementation of Deadlock Detection**

**Aim:**To Write a c program to implement deadlock detection

**Algorithm :**

1. **Create the resource and process to find out the detection in the deadlock.**
2. **Determine the existence of the deadlock.**
3. **Identify the process and resource involved in the deadlock.**
4. **Check the resource availability for all the possibilities.**
5. **Temporarily prevent the resources from the deadlocked process.**
6. **Kill the process until the deadlock gets free.**
7. **Stop the process.**

**PROGRAM:**

#include<stdio.h>

int max[100][100];

int alloc[100][100];

int need[100][100];

int avail[100];

int n,r;

void input();

void show();

void cal();

int main()

{

int i,j;

printf("\*\*\*\*\*\*\*\*\*\* Deadlock Detection Algo \*\*\*\*\*\*\*\*\*\*\*\*\n");

input();

show();

cal();

return 0;

}

void input()

{

int i,j;

printf("Enter the no of Processes\t");

scanf("%d",&n);

printf("Enter the no of resource instances\t");

scanf("%d",&r);

printf("Enter the Max Matrix\n");

for(i=0;i<n;i++)

{

for(j=0;j<r;j++)

{

scanf("%d",&max[i][j]);

}

}

printf("Enter the Allocation Matrix\n");

for(i=0;i<n;i++)

{

for(j=0;j<r;j++)

{

scanf("%d",&alloc[i][j]);

} }

printf("Enter the available Resources\n");

for(j=0;j<r;j++)

{

scanf("%d",&avail[j]);

} }

void show()

{

int i,j;

printf("Process\t Allocation\t Max\t Available\t");

for(i=0;i<n;i++)

{

printf("\nP%d\t ",i+1);

for(j=0;j<r;j++)

{

printf("%d ",alloc[i][j]);

}

printf("\t");

for(j=0;j<r;j++)

{

printf("%d ",max[i][j]);

}

printf("\t");

if(i==0)

{

for(j=0;j<r;j++)

printf("%d ",avail[j]);

} }}

void cal()

{

int finish[100],temp,need[100][100],flag=1,k,c1=0;

int dead[100];

int safe[100];

int i,j;

for(i=0;i<n;i++)

{

finish[i]=0;

}

for(i=0;i<n;i++)

{ finish[i]=0;

}

for(i=0;i<n;i++)

{

for(j=0;j<r;j++)

{

need[i][j]=max[i][j]-alloc[i][j];

}

}

while(flag)

{

flag=0;

for(i=0;i<n;i++)

{

int c=0;

for(j=0;j<r;j++)

{

if((finish[i]==0)&&(need[i][j]<=avail[j]))

{

c++;

if(c==r)

{

for(k=0;k<r;k++)

{

avail[k]+=alloc[i][j];

finish[i]=1;

flag=1;

}

if(finish[i]==1)

{

i=n;

} } } } } }

j=0;

flag=0;

for(i=0;i<n;i++)

{

if(finish[i]==0)

{

dead[j]=i;

j++;

flag=1;

}

}

if(flag==1)

{

printf("\n\nSystem is in Deadlock and the Deadlock process are\n");

for(i=0;i<n;i++)

{

printf("P%d\t",dead[i]);

}

}

else

{

printf("\nNo Deadlock Occur");

}}

**OUTPUT:**

[kalai@localhost ~]$ cc deaddetect.c

[kalai@localhost ~]$ ./a.out

\*\*\*\*\*\*\*\*\*\* Deadlock Detection Algo \*\*\*\*\*\*\*\*\*\*\*\*

Enter the no of Processes 3

Enter the no of resource instances 3

Enter the Max Matrix

3 6 8

4 3 3

3 4 4

Enter the Allocation Matrix

3 3 3

2 0 3

1 2 4

Enter the available Resources

1 2 0

Process Allocation Max Available

P1 3 3 3 3 6 8 1 2 0

P2 2 0 3 4 3 3

P3 1 2 4 3 4 4

System is in Deadlock and the Deadlock process are

P0 P1 P2

**ExNo: 7 Implementation of Memory Allocation Methods**

**(a)First Fit (b) Best Fit (c) Worst Fit**

**7A. First fit**

**Aim:**To Write a c program to implement the memory allocation for first fit technique

**Algorithm:**

1. Start the program.
2. Initialize all memory blocks as free.
3. Start by picking each process and check if it can be assigned to current block.
4. If Size of the process <= Size of block then assign and check for the next process.
5. If not the keep checking the further blocks.

**PROGRAM:**

#include<stdio.h>

main()

{

int i,j,n,ns,jb,c,ss[20];

printf("\n Enter the no of slots ");

scanf("%d",&ns);

printf("\n Enter the %d slot ",ns);

for(i=0;i<ns;i++)

scanf("%d",&ss[i]);

do

{

printf("\n Enter the job size");

scanf("%d",&jb);

for(i=0;i<ns;i++)

{

if(jb<=ss[i])

{

printf("%d is allocated to slot %d of size %d",jb,i+1,ss[i]);

ss[i]=ss[i]-jb;

printf("\n Remaining space of slots %d is %d",i+1,ss[i]);

n=1;

break;

}

else

continue;

}

if(n!=1)

printf("%d is not allocated to any slots",jb);

n=0;

printf("\n Press 1 to continue");

scanf("%d",&c);

}

while(c==1);

}

**OUTPUT:**

[kalai@localhost ~]$ ./a.out

Enter the no of slots 3

Enter the 3 slot 12

2

34

Enter the job size12

12 is allocated to slot 1 of size 12

Remaining space of slots 1 is 0

Press 1 to continue 1

Enter the job size45

45 is not allocated to any slots

Press 1 to continue

**(b) Best Fit**

**Aim:**To write a c program to implement the memory allocation for best fit technique

**Algorithm:**

1. Start the program

2. Initialize all memory blocks as free

3. Start by picking each process and find the minimum block size that can be assigned to current process. If found then assign it to the current process.

4. If not then leave that process and keep checking the further processes.

**PROGRAM:**

#include<stdio.h>

main()

{

int a,b[50],i,j,temp,s;

printf("\n\n\tBEST FIT");

printf("\n Enter the number of unallocated space:");

scanf("%d",&a);

printf("\n Enter the unallocated space in KB\n");

for(i=1;i<=a;i++)

scanf("%d",&b[i]);

for(j=i+1;j<=a;j++)

if(b[i]>b[j])

{

temp=b[j];

b[j]=b[i];

b[i]=temp;

}

printf("\nEnter the space required for process");

scanf("%d",&s);

for(i=1;i<=a;i++)

{

if(s<b[i])

{

printf("\n The best fit for required space is %d\n",b[i]);

break;

}}}

**OUTPUT:**

[kalai@localhost ~]$ cc best.c

[kalai@localhost ~]$ ./a.out

BEST FIT

Enter the number of unallocated space:2

Enter the unallocated space in KB

100

200

Enter the space required for process 178

The best fit for required space is 200

**(c) Worst Fit**

**Aim:**To write a c program to implement the memory allocation for worst fit technique

**Algorithm:**

1.Start the program

2. Initialize all memory blocks as free

3. Start by picking each process and find the maximum block size that can be assigned to current process. If found then assign it to the current process.

4. If not then leave that process and keep checking the further processes.

**PROGRAM:**

#include<stdio.h>

main( )

{

int a[50],n,i,j,p,t;

printf("\n\t\tWORST FIT");

printf("\n Enter the unallocated space");

scanf("%d",&n);

printf("\nEnter the memory space\n");

for(i=1;i<=n;i++)

scanf("%d",&a[i]);

printf("\n Enter the memory required for the process:\n");

scanf("%d",&p);

for(i=1;i<n;i++)

for(j=i+1;j<=n;j++)

if(a[i]>a[j])

{

t=a[i];

a[i]=a[j];

a[j]=t;

}

printf("\nThe worst fit process %d\n",a[i]);

}

**OUTPUT:**

[kalai@localhost ~]$ cc worst.c

[kalai@localhost ~]$ ./a.out

WORST FIT

Enter the unallocated space3

Enter the memory space

12

10

4

Enter the memory required for the process:

5

The worst fit process 12

**ExNo: 8 Implementation of Paging Technique of Memory Management**

**Aim:** To write a c program to implement the Memory management policy- Paging.

**Algorithm:**

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

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

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

Step 4: Calculate the physical address using the following

Physical address = ( Frame number \* Frame size ) + offset

Step 5: Display the physical address.

Step 6: Stop the process

**PROGRAM:**

#include<stdio.h>

void main()

{

int np,nf,i,j,framenum,pagesize,vaddress,paddress,vpage,offset,pagetable[20];

printf("Inut no of pages in virtual memory?");

scanf("%d",&np);

printf("\nInput the page size?");

scanf("%d",&pagesize);

printf("\nInput no.of frames in physical memory?");

scanf("%d",&nf);

A:

printf("Input the virtual address?");

scanf("%d",&vaddress);

if(vaddress>=(pagesize\*np))

{

printf("\nInvalid virtual address...");

goto A;

}

vpage=vaddress/pagesize;

offset=vaddress%pagesize;

printf("\nInput the physical address?");

scanf("%d",&paddress);

for(i=0;i<nf;i++)

pagetable[i]=-1;

for(i=0;i<np;i++)

{

B:

printf("\nInput frame number for page %d?",i);

scanf("%d",&framenum);

if(framenum>nf)

{

printf("\nInvalid frame number...give valid frame number...");

goto B;

}

else

{

for(j=0;j<i;j++)

{

if(framenum==pagetable[j])

{

printf("\nFrame %d was already assigned to page %d",pagetable[j],j);

goto B;

}

}

}

pagetable[i]=framenum;

}

printf("\n===============Page Table=================");

printf("\nPage Number | Frame Number");

printf("\n==========================================");

for(i=0;i<np;i++)

printf("\n%5d\t%5d",i,pagetable[i]);

printf("\nVirtual address:\n");

printf("\nPage Number:%d,offset=%d",vpage,offset);

paddress=paddress+((pagetable[vpage]\*pagesize)+offset);

printf("\nThe Physical address of given virtual address is %d\n",paddress);

}

**OUTPUT:**

[kalai@localhost ~]$ cc paging.c

[kalai@localhost ~]$ ./a.out

Input no of pages in virtual memory? 4

Input the page size? 3

Input no.of frames in physical memory? 4

Input the virtual address? 10

Input the physical address? 3000

Input frame number for page 0? 3

Input frame number for page 1? 0

Input frame number for page 2? 1

Input frame number for page 3? 2

===============Page Table=================

Page Number | Frame Number

==========================================

0 3

1 0

2 1

3 2

Virtual address:

Page Number:3,offset=1

The Physical address of given virtual address is 3007

**Ex No: 9 Implementation of Page replacement Techniques**

**9a FIFO page replacement algorithm**

**Aim:** To write a c program to implement the FIFO page replacement algorithm.

**Algorithm**

1.Get length of the reference string, say l.

2.Get reference string and store it in an array, say rs.

3.Get number of frames, say nf.

4.Initalize frame array upto length nf to -1.

5.Initialize position of the oldest page, say j to 0.

6.Initialize no. of page faults, say count to 0.

7.For each page in reference string in the given order, examine:

a.Check whether page exist in the frame array

b.If it does not exist then

i. Replace page in position j.

ii.Compute page replacement position as (j+1) modulus nf.

iii.Increment count by 1.

iv.Display pages in frame array.

8.Print count.

9. Stop

**PROGRAM:**

#include<stdio.h>

int main()

{

int refstr[20],nf,i,j,rscount,pgfault=0,frame[20],found,loc=0,victim=0;

printf("Input reference string count?");

scanf("%d",&rscount);

printf("enter the reference string\n");

for(i=0;i<rscount;i++)

scanf("%d",&refstr[i]);

printf("Input No. frames? ");

scanf("%d",&nf);

for(i=0;i<nf;i++)

frame[i]=-1;

for(i=0;i<rscount;i++)

{ found=0;

for(j=0;j<nf;j++)

if(refstr[i]==frame[j])

{

found=1;

break;

}

if(!found)

{

if(loc<nf)

frame[loc++]=refstr[i];

else

{

frame[victim]=refstr[i];

victim=(victim+1)%nf;

}

pgfault++;

}

printf("\nref.%d: ",refstr[i]);

if(found)

printf("\t---hit---");

else

for(j=0;j<nf;j++)

printf("%3d",frame[j]);

}

printf("\nthe number of page fault=%d",pgfault);

}

**OUTPUT:**

[kalai@localhost ~]$ cc fifo.c

[kalai@localhost ~]$ ./a.out

Input reference string count? 4

enter the reference string

2 3 4 5

Input No. frames? 3

ref.2: 2 -1 -1

ref.3: 2 3 -1

ref.4: 2 3 4

ref.5: 5 3 4

the number of page fault=4

**9b LRU page replacement**

**Aim:** To write a c program to implement the LRU page replacement algorithm

**Algorithm**

1.Get length of the reference string, say *len*.

2.Get reference string and store it in an array, say rs.

3.Get number of frames, say nf.

4.Create access array to store counter that indicates a measure of recent usage.

5.Create a function arrmin that returns position of minimum of the given array.

6. Initalizeframearrayupto length nf to -1.

7.Initialize position of the page replacement, say j to 0.

8.Initialize freq to 0 to track page frequency

9.Initialize no. of page faults, say count to 0.

10.For each page in reference string in the given order, examine:

a.Check whether page exist in the frame array.

b.If page exist in memory then

i.Store incremented freq for that page position in access array.

c.If page does not exist in memory then

i. Check for any empty frames.

ii.If there is an empty frame,

11. Assign that frame to the page

12. Store incremented freq for that page position in access array.

13. Increment count.

iii. If there is no free frame then

14.Determine page to be replaced using arrmin function.

15. Store incremented freq for that page position in access array.

16. Increment count.

Display pages in frame array.

11.Print count.

12.Stop

**PROGRAM:**

#include<stdio.h>

int main()

{

int refstr[20],count,nf,i,j,k,rscount,pgfault=0,frame[20],found,loc=0,victim=0,vframe[10];

printf("Input reference string count?");

scanf("%d",&rscount);

printf("enter the reference string\n");

for(i=0;i<rscount;i++)

scanf("%d",&refstr[i]);

printf("Input No. frames? ");

scanf("%d",&nf);

for(i=0;i<nf;i++)

frame[i]=-1;

for(i=0;i<rscount;i++)

{ found=0;

for(j=0;j<nf;j++)

if(refstr[i]==frame[j])

{

found=1;

break;

}

if(!found)

{

if(loc<nf)

frame[loc++]=refstr[i];

else

{

count=0;

for(j=0;j<nf;j++)

vframe[j]=frame[j];

for(j=i-1;j>0 && (count<nf-1);j--)

for(k=0;k<nf && (count<nf-1) ;k++)

if(count< nf-1 )

{

if(refstr[j]==vframe[k])

{

vframe[k]=-1;

count++;

}

}

for(j=0;j<nf;j++)

if(vframe[j]!=-1) /\*victim frame does not contain -1\*/

victim=j;

frame[victim]=refstr[i];

}

pgfault++;

}

printf("\nref.%d: ",refstr[i]);

if(found) printf("\t---hit---");

else

for(j=0;j<nf;j++)

printf("%3d",frame[j]);

}

printf("\nthe number of page fault=%d",pgfault);

}

**OUTPUT**:

[kalai@localhost ~]$ cc lru.c

[kalai@localhost ~]$ ./a.out

Input reference string count? 4

enter the reference string

2 3 4 5

Input No. frames? 3

ref.2: 2 -1 -1

ref.3: 2 3 -1

ref.4: 2 3 4

ref.5: 5 3 4

the number of page fault=4

**9C** OPTIMAL page replacement algorithm

**Aim:** To write a c program to implement the Optimal page replacement algorithm.

**Algorithm**

1.Get length of the reference string, say *len*.

2.Get reference string and store it in an array, say rs.

3.Get number of frames, say nf.

4.Create access array to store counter that indicates a measure of recent usage.

5.Create a function arrmin that returns position of minimum of the given array.

6. Initalizeframearrayupto length nf to -1.

7.Initialize position of the page replacement, say j to 0.

8.Initialize freq to 0 to track page frequency

9.Initialize no. of page faults, say count to 0.

10.For each page in reference string in the given order, examine:

a.Check whether page exist in the frame array.

b.If page exist in memory then

i.Store incremented freq for that page position in access array.

c.If page does not exist in memory then

i. Check for any empty frames.

ii.If there is an empty frame,

**PROGRAM:**

#include<stdio.h>

int main()

{

int refstr[20],count,nf,i,j,k,rscount,pgfault=0,frame[20],found,loc=0,victim=0,vframe[10];

printf("Input reference string count?");

scanf("%d",&rscount);

printf("enter the reference string\n");

for(i=0;i<rscount;i++)

scanf("%d",&refstr[i]);

printf("Input No. frames? ");

scanf("%d",&nf);

for(i=0;i<nf;i++)

frame[i]=-1;

for(i=0;i<rscount;i++)

{ found=0;

for(j=0;j<nf;j++)

if(refstr[i]==frame[j])

{

found=1;

break;

}

if(!found)

{

if(loc<nf)

frame[loc++]=refstr[i];

else

{

count=0;

for(j=0;j<nf;j++)

vframe[j]=frame[j];

for(j=i+1;j<rscount && (count<nf-1);j++)

for(k=0;k<nf && (count<nf-1) ;k++)

if(count< nf-1 )

{

if(refstr[j]==vframe[k])

{

vframe[k]=-1;

count++;

}

}

for(j=0;j<nf;j++)

if(vframe[j]!=-1) /\*victim frame does not contain -1\*/

victim=j;

frame[victim]=refstr[i];

}

pgfault++;

}

printf("\nref.%d: ",refstr[i]);

if(found) printf("\t---hit---");

else

for(j=0;j<nf;j++)

printf("%3d",frame[j]);

}

printf("\nthe number of page fault=%d",pgfault);

}

**OUTPUT:**

[kalai@localhost ~]$ cc optimal.c

[kalai@localhost ~]$ ./a.out

Input reference string count? 4

enter the reference string

2 3 4 5

Input No. frames? 3

ref.2: 2 -1 -1

ref.3: 2 3 -1

ref.4: 2 3 4

ref.5: 2 3 5

the number of page fault=4

**Ex No: 10 Implementation of File Organization Techniques**

**10A Single Level Directory Structure**

**Aim:** To write a c program to implement Single Level Directory file organization technique

**Algorithm for Single Level Directory**

1. Start the process
2. Initialize values gd=DETECT,gm,count,i,j,mid,cir\_x;
3. Initialize character array fname[10][20];
4. Read number of files in variable count.
5. if check i<count
6. for i=0 & i<count i increment;
7. End

**PROGRAM:**

#include<stdio.h>

main()

{int master, s[20];

char f[20][20][20];

char d[20][20];

int i,j;

printf("Enter number of directories.");

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 \t Size \t Filenames\n");

for(i=0;i<80;i++)

printf("\*");

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");

}

**OUTPUT:**

[kalai@localhost ~]$ cc onedir.c

[kalai@localhost ~]$ ./a.out

Enter number of directories. 1

Enter names of directories. CSE

Enter size of directories. 3

Enter the file names: C

JAVA

DATASTRUCTURE

Directory Size Filenames

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CSE 3 C

JAVA

DATASTRUCTURE

**10 B. Two Level Directory Structure**

**Aim:** To write a c program to implement two Level Directory file organization technique.

**Algorithm for Two Level Directory**

1. Start
2. Initialize structure and start main function.
3. Create structure.
4. End main function.
5. Initialize variables i,gap;
6. if check \*root==NULL(\*root)=(node\*)malloc(sizeof(node));
7. stop.

**PROGRAM:**

#include<stdio.h>

struct st

{char dname[10];

char sdname[10][10];

char fname[10][10][10];

int ds,sds[10];

}dir[10];

main()

{

int i,j,k,n;

printf("Enter no. 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 directory: ");

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 \t Size \t Subdirname \t Size \t Files \n");

for(i=0;i<80;i++)

printf("\*");

printf("\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:**

[kalai@localhost ~]$ cc twodir.c

[kalai@localhost ~]$ ./a.out

Enter no. of directories: 1

Enter directory 1 names: CSE

Enter size of directory: 1

Enter subdirectory name and size.IICSE

3

Enter file name: OS

Enter file name: SE

Enter file name: DAA

Dirname Size Subdirname Size Files

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CSE 1 IICSE 3 OS SE DAA

**Ex No:11 Implement all file allocation strategies**

**11A Sequential Allocation**

**Aim:** To write a c program to implement sequentialAllocation strategy.

**Algorithm**

1. Start the process
2. set each file occupies a set of contiguous blocks on the disk.
3. get – only starting location (block #) and length (number of blocks) are required.
4. stop the process.

**PROGRAM:**

#include <stdio.h>

main()

{

int i,j,a[200],n,t,b;

printf("\n\t Sequential Allocation");

printf("\n\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

printf("\n\t No. of Files : ");

scanf("%d",&n);

j=1;

for(i=1;i<=n;i++)

{

printf("\n\tNumber of Blocks - File %d : ",i);

scanf("%d",&b);

t=1;

while(t<=b)

{

a[j]=i;

j++;

t++;

}

}

printf("\n\t Sequential Allocation");

printf("\n\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

i=1;

j=1;

while(i<=n)

{

printf("\n\t Blocks Allocated for File - %d \n\t",i);

while(a[j]==i)

{

printf(" %d",j);

j++;

}

i++;

}

return 0;

}

**OUTPUT:**

[kalai@localhost ~]$ cc seq.c

[kalai@localhost ~]$ ./a.out

Sequential Allocation

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

No. of Files : 2

Number of Blocks - File 1 : 4

Number of Blocks - File 2 : 5

Sequential Allocation

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Blocks Allocated for File - 1

1 2 3 4

Blocks Allocated for File - 2

5 6 7 8 9

**1B Linked file allocation**

**Aim:** To write a c program to implement linked file allocation strategy.

**Algorithm**

1. Start the process.
2. get the starting address.
3. Free-space management system – no waste of space.
4. No random access.
5. Mapping the file.
6. Stop the process.

**PROGRAM:**

**PROGRAM:**

#include <stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*nxt;

}\*st=NULL;

int f[10],n;

void cre()

{

struct node \*t,\*nn,\*cn;

nn=(struct node \*) malloc (sizeof(struct node));

nn->nxt=NULL;

a1:scanf("%d", &nn->data);

t=st;

while(t!=NULL)

{

if(t->data==nn->data)

{

printf("\n\t Block is already occupied \n");

goto a1;

}

else

t=t->nxt;

}

if(st==NULL)

st=nn;

else

{

t=st;

while (t->nxt!=NULL)

t=t->nxt;

t->nxt=nn;

}

}

void display()

{

struct node \*nn;

int t,j,i=1;

printf("\n\t Linked List :");

nn=st;

while(nn!=NULL)

{

t=f[i];

printf("\n\t Contents of File %d :",i);

for(j=1;j<=t;j++)

{

printf(" %d -->",nn->data);

nn=nn->nxt;

}

printf(" NULL");

i++;

}

}

void main()

{

int i,b,j;

printf("\n\t Linked Allocation");

printf("\n\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

printf("\n\t Number of Files: ");

scanf("%d",&n);

for(i=1;i<=n;i++)

{

printf("\n\t Number of Blocks for File %d : ",i);

scanf("%d",&b);

f[i]=b;

for(j=1;j<=b;j++)

{

printf("\n\t Block %d: ",j);

cre();

}

}

display();

}

**OUTPUT:**

[kalai@localhost ~]$ cc link.c

[kalai@localhost ~]$ ./a.out

Linked Allocation

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Number of Files: 2

Number of Blocks for File 1 : 2

Block 1: 1

Block 2: 2

Number of Blocks for File 2 : 2

Block 1: 2

Block is already occupied 5

Block 2: 3

Linked List :

Contents of File 1 : 1 --> 2 --> NULL

Contents of File 2 : 5 --> 3 --> NULL[

**11C Index file allocation**

**Aim:** To write a c program to implement index file allocation strategy.

**Algorithm**

1. Create a file index .txt if it is null print unable to open file.
2. Get the number of files.
3. Print the block number and the file name.
4. Displays the block number and the file name.
5. Cat index.txt to view the display.

**PROGRAM:**

#include <stdio.h>

main()

{

int n,i,j,t,k,pos,b[10],a[100];

printf("\n\t Indexed Allocation \n");

printf("\n\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\n\t Enter Number of Files:");

scanf("%d",&n);

pos=0;

for(i=1;i<=n;i++)

{

printf("\n\t Number of Blocks for File %d :",i);

scanf("%d",&b[i]);

for(j=1;j<=b[i];j++)

{

a1: printf("\nIndex %d : ",j);

scanf("%d",&t);

if(pos!=0)

{

for(k=1;k<=pos;k++)

{

if(a[k]==t)

{

printf("\n\tIndex location is already Occupied ");

goto a1;

}

}

}

pos++;

a[pos]=t;

}

}

printf("\n\t File Length Indexes\n");

j=1;

for(i=1;i<=n;i++)

{

printf("\n\t %d \t\t %d \t",i,b[i]);

for(k=1;k<=b[i];k++)

{

printf(" %d",a[j]);

j++;

}

printf("\n");

}

return 0;

}

**OUTPUT:**

[kalai@localhost ~]$ cc index.c

[kalai@localhost ~]$ ./a.out

Indexed Allocation

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enter Number of Files: 2

Number of Blocks for File 1 : 2

Index 1 : 1

Index 2 : 3

Number of Blocks for File 2 : 3

Index 1 : 4

Index 2 : 9

Index 3 : 10

File Length Indexes

1 2 1 3

2 4 9 10