*OPERATING SYSTEM LAB MANUAL*

|  |  |
| --- | --- |
| **Ex.No:1.a** | **BASICS OF UNIX COMMANDS** |
| **INTRODUCTION TO UNIX** |

## AIM:

To study about the basics of UNIX

## UNIX:

It is a multi-user operating system. Developed at AT & T Bell Industries, USA in 1969.

Ken Thomson along with Dennis Ritchie developed it from MULTICS (Multiplexed Information and Computing Service) OS.

By1980, UNIX had been completely rewritten using C language.

## LINUX:

It is similar to UNIX, which is created by Linus Torualds. All UNIX commands works in Linux. Linux is a open source software. The main feature of Linux is coexisting with other OS such as windows and UNIX.

## STRUCTURE OF A LINUXSYSTEM:

It consists of three parts.

1. UNIX kernel
2. Shells
3. Tools and Applications

## UNIX KERNEL:

Kernel is the core of the UNIX OS. It controls all tasks, schedule all Processes and carries out all the functions of OS.

Decides when one programs tops and another starts.

## SHELL:

Shell is the command interpreter in the UNIX OS. It accepts command from the user and analyses and interprets them

|  |  |
| --- | --- |
| **Ex.No:1.b** | **BASICS OF UNIX COMMANDS** |
| **BASIC UNIX COMMANDS** |

## AIM:

To study of Basic UNIX Commands and various UNIX editors such as vi, ed, ex and EMACS.

## CONTENT:

### Note: Syn->Syntax

1. **date**

–used to check the date and time Syn:$date

|  |  |  |  |
| --- | --- | --- | --- |
| Format | Purpose | Example | Result |
| +%m | To display only month | $date+%m | 06 |
| +%h | To display month name | $date+%h | June |
| +%d | To display day of month | $date+%d | O1 |
| +%y | To display last two digits of years | $date+%y | 09 |
| +%H | To display hours | $date+%H | 10 |
| +%M | To display minutes | $date+%M | 45 |
| +%S | To display seconds | $date+%S | 55 |

### cal

–used to display the calendar Syn:$cal 2 2009

### echo

–used to print the message on the screen.

Syn:$echo “text”

### ls

–used to list the files. Your files are kept in a directory.

Syn:$lsls–s

All files (include files with prefix) ls–l Lodetai (provide file statistics) ls–t Order by creation time

ls– u Sort by access time (or show when last accessed together with –l) ls–s Order by size

ls–r Reverse order

ls–f Mark directories with /,executable with\* , symbolic links with @, local sockets with =, named pipes(FIFOs)with

ls–s Show file size

ls– h“ Human Readable”, show file size in Kilo Bytes & Mega Bytes (h can be used together with –l or)

ls[a-m]\*List all the files whose name begin with alphabets From „a‟ to „m‟ ls[a]\*List all the files whose name begins with „a‟ or „A‟

Eg:$ls>my list Output of „ls‟ command is stored to disk file named „my list‟

### lp

–used to take printouts Syn:$lp filename

### man

–used to provide manual help on every UNIX commands.

Syn:$man unix command

$man cat

### who & whoami

–it displays data about all users who have logged into the system currently. The next command displays about current user only.

Syn:$who$whoami

### uptime

–tells you how long the computer has been running since its last reboot or power-off.

Syn:$uptime

### uname

–it displays the system information such as hardware platform, system name and processor, OS type.

Syn:$uname–a

### hostname

–displays and set system host name Syn:$ hostname

### bc

–stands for „best calculator‟

|  |  |  |  |
| --- | --- | --- | --- |
| $bc | $ bc | $ bc | $ bc |
| 10/2\*3 | scale =1 | ibase=2 | sqrt(196) |
| 15 | 2.25+1 | obase=16 | 14 quit |
|  | 3.35 | 11010011 |  |
|  | quit | 89275 |  |
|  |  | 1010 |  |
|  |  | Ā |  |
|  |  | Quit |  |
| $bc | $ bc-l |  |  |
| for(i=1;i<3;i=i+1)I | scale=2 |  |  |
| 1 | s(3.14) |  |  |
| 2 | 0 |  |  |
| 3 quit |  |  |  |

## FILE MANIPULATION COMMANDS

1. **cat**–this create, view and concatenate files.

### Creation:

Syn:$cat>filename

### Viewing:

Syn:$cat filename

### Add text to an existing file:

Syn:$cat>>filename

### Concatenate:

Syn:$catfile1file2>file3

$catfile1file2>>file3 (no over writing of file3)

1. **grep**–used to search a particular word or pattern related to that word from the file. Syn:$grep search word filename

Eg:$grep anu student

1. **rm**–deletes a file from the file system Syn:$rm filename
2. **touch**–used to create a blank file.

Syn:$touch file names

1. **cp**–copies the files or directories Syn:$cpsource file destination file Eg:$cp student stud
2. **mv**–to rename the file or directory syn:$mv old file new file

Eg:$mv–i student student list(-i prompt when overwrite)

1. **cut**–it cuts or pickup a given number of character or fields of the file. Syn:$cut<option><filename>

Eg: $cut –c filename

$cut–c1-10emp

$cut–f 3,6emp

$ cut –f 3-6 emp

-c cutting columns

-f cutting fields

1. **head**–displays10 lines from the head(top)of a given file Syn:$head filename

Eg:$head student

To display the top two lines:

Syn:$head-2student

1. **tail**–displays last 10 lines of the file Syn:$tail filename

Eg:$tail student

To display the bottom two lines;

Syn:$ tail -2 student

1. **chmod**–used to change the permissions of a file or directory. Syn:$ch mod category operation permission file Where, Category–is the user type

Operation–is used to assign or remove permission Permission–is the type of permission

File–are used to assign or remove permission all

Examples:

$chmodu-wx student

Removes write and execute permission for users

$ch modu+rw,g+rwstudent

Assigns read and write permission for users and groups

$chmodg=rwx student

Assigns absolute permission for groups of all read, write and execute permissions

1. **wc**–it counts the number of lines, words, character in a specified file(s) with the options as –l,-w,-c

|  |  |  |
| --- | --- | --- |
| Category | Operation | Permission |
| u– users g–group o– others | +assign  -remove  =assign absolutely | r– read w– write x-execute |

Syn: $wc –l filename

$wc –w filename

$wc–c filename

|  |  |
| --- | --- |
| **Ex.No:1.c** | **BASICS OF UNIX COMMANDS** |
| **UNIX EDITORS** |

## AIM:

To study of various UNIX editors such as vi, ed, ex and EMACS.

## CONCEPT:

Editor is a program that allows user to see a portions a file on the screen and modify characters and lines by simply typing at the current position. UNIX supports variety of Editors. They are:

ed ex vi EMACS

Vi- vi is stands for “visual”.vi is the most important and powerful editor.vi is a full screen editor that allows user to view and edit entire document at the same time.vi editor was written in the University of California, at Berkley by Bill Joy, who is one of the co-founder of Sun Microsystems.

### Features of vi:

It is easy to learn and has more powerful features.

Itworksgreatspeedandiscasesensitive.vihaspowerfulundofunctionsandhas3modes:

* 1. Command mode
  2. Insert mode
  3. Escape or ex mode

In command mode, no text is displayed on the screen.

In Insert mode, it permits user to edit insert or replace text. In escape mode, it displays commands at command line.

Moving the cursor with the help of h, l, k, j, I, etc

### EMACS Editor

Motion Commands:

M-> Move to end of file

M-< Move to beginning of file

C-v Move forward a screen M –v Move backward a screen C –n Move to next line C-p Move to previous line

C-a Move to the beginning of the line C-e Move to the end of the line

C-f Move forward a character C-b Move backward a character M-f Move forward a word

M-b Move backward a word

Deletion Commands:

DEL delete the previous character C -d delete the current character M -DEL delete the previous word

M-d delete the next word

C-x DEL deletes the previous sentence

M-k delete the rest of the current sentence

C-k deletes the rest of the current line

C-xu undo the lasted it change

Search and Replace in EMACS:

y Change the occurrence of the pattern

n Don‟t change the occurrence, but look for the other q Don‟t change. Leave query replace completely

! Change this occurrence and all others in the file

|  |  |
| --- | --- |
| **Ex.No:2** | Programs using the following system calls of UNIX operating system fork, exec, getpid, exit, wait, close, stat, opendir, readdir |

**AIM:**

To write C Programs using the following system calls of UNIX operating system fork, exec, getpid, exit, wait, close, stat, opendir, readdir.

## PROGRAM FOR SYSTEM CALLS OF UNIX OPERATING SYSTEMS (OPENDIR, READDIR, CLOSEDIR)

**ALGORITHM:**

STEP 1: Start the program. STEP 2: Create struct dirent.

STEP 3: declare the variable buff and pointer dptr. STEP 4: Get the directory name.

STEP 5: Open the directory.

STEP 6: Read the contents in directory and print it. STEP 7: Close the directory.

# PROGRAM:

#include<stdio.h> #include<dirent.h> struct dirent \*dptr;

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

{

char buff[100]; DIR \*dirp;

printf(“\n\n ENTER 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);

}

1. **PROGRAM FOR SYSTEM CALLS OF UNIX OPERATING SYSTEM**

**(fork, getpid, exit)**

## ALGORITHM:

STEP 1: Start the program.

STEP 2: Declare the variables pid,pid1,pid2. STEP 3: Call fork() system call to create process. STEP 4: If pid==-1, exit.

STEP 5: Ifpid!=-1 , get the process id using getpid(). STEP 6: Print the process id.

STEP 7:Stop the program

# PROGRAM:

#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);

}

}

|  |  |
| --- | --- |
| **Ex.No:3** | C programs to simulate UNIX commands like cp, ls, grep. |

## AIM:

To write C programs to simulate UNIX commands like cp, ls, grep.

# Program for simulation of cp unix commands ALGORITHM:

STEP1: Start the program

STEP 2:Declare the variables ch, \*fp, sc=0 STEP3: Open the file in read mode

STEP 4: Get the character

STEP 5: If ch== “ “ then increment sc value by one STEP 6: Print no of spaces

STEP 7:Close the file

## PROGRAM:

#include<fcntl.h> #include<unistd.h> #include<stdio.h> main(int argc,char \*argv[])

{

FILE \*fp; char ch; int sc=0;

fp=fopen(argv[1],"r"); if(fp==NULL)

printf("unable to open a file",argv[1]); else

{

while(!feof(fp))

{

ch=fgetc(fp); if(ch==' ') sc++;

}

printf("no of spaces %d",sc); printf("\n");

fclose(fp);

}}

# PROGRAM FOR SIMULATION OF LS UNIX COMMANDS

**ALGORTIHM:**

STEP1 : Start the program

STEP2 : Open the directory with directory object dp STEP3 : Read the directory content and print it.

STEP4: Close the directory.

# PROGRAM:

#include<stdio.h> #include<dirent.h> main(int argc, char \*\*argv)

{

DIR \*dp;

struct dirent \*link; dp=opendir(argv[1]);

printf(“\n contents of the directory %s are \n”, argv[1]); while((link=readdir(dp))!=0)

printf(“%s”,link->d\_name); closedir(dp);

}

# OUTPUT:

|  |  |
| --- | --- |
|  | **CPU SCHEDULING ALGORITHMS** |
| **PRIORITY** |

**AIM:**

To write a C program for implementation of Priority scheduling algorithms.

## ALGORITHM:

Step 1: Inside the structure declare the variables.

Step 2: Declare the variable i,j as integer, totwtime and totttime is equal to zero. Step 3: Get the value of „n‟ assign p and allocate the memory.

Step 4: Inside the for loop get the value of burst time and priority. Step 5: Assign wtime as zero .

Step 6: Check p[i].pri is greater than p[j].pri .

Step 7: Calculate the total of burst time and waiting time and assign as turnaround time. Step 8: Stop the program.

## PROGRAM:

#include<stdio.h> #include<stdio.h> #include<stdlib.h> typedef struct

{

int pno; int pri; int pri; int btime;

int wtime;

}sp;

int main()

{

int i,j,n;

int tbm=0,totwtime=0,totttime=0; sp \*p,t;

printf("\n PRIORITY SCHEDULING.\n");

printf("\n enter the no of process. \n");

scanf("%d",&n); p=(sp\*)malloc(sizeof(sp));

printf("enter the burst time and priority:\n"); for(i=0;i<n;i++)

{

printf("process%d:”,i+1); scanf("%d%d",&p[i].btime,&p[i].pri);

p[i].pno=i+1;

p[i].wtime=0;

}

for(i=0;i<n-1;i++) for(j=i+1;j<n;j++)

{

if(p[i].pri>p[j].pri)

{

t=p[i]; p[i]=p[j]; p[j]=t;

}

}

printf("\n process\tbursttime\twaiting time\tturnaround time\n"); for(i=0;i<n;i++)

{

totwtime+=p[i].wtime=tbm; tbm+=p[i].btime; printf("\n%d\t\t%d",p[i].pno,p[i].btime);

printf("\t\t%d\t\t%d",p[i].wtime,p[i].wtime+p[i].btime);

}

totttime=tbm+totwtime;

printf("\n total waiting time:%d",totwtime);

printf("\n average waiting time:%f",(float)totwtime/n); printf("\n total turnaround time:%d",totttime); printf("\n avg turnaround time:%f",(float)totttime/n);

}

## OUTPUT:

|  |  |
| --- | --- |
|  | **CPU SCHEDULING ALGORITHMS** |
| **ROUND ROBIN SCHEDULING** |

**AIM:**

To write a C program for implementation of Round Robin scheduling algorithms.

## ALGORITHM:

Step 1: Inside the structure declare the variables.

Step 2: Declare the variable i,j as integer, totwtime and totttime is equal to zero. Step 3: Get the value of „n‟ assign p and allocate the memory.

Step 4: Inside the for loop get the value of burst time and priority and read the time quantum. Step 5: Assign wtime as zero.

Step 6: Check p[i].pri is greater than p[j].pri .

Step 7: Calculate the total of burst time and waiting time and assign as turnaround time. Step 8: Stop the program.

## PROGRAM:

#include<stdio.h> #include<stdlib.h> struct rr

{

int pno,btime,sbtime,wtime,lst;

}p[10];

int main()

{

int pp=-1,ts,flag,count,ptm=0,i,n,twt=0,totttime=0; printf("\n round robin scheduling ");

printf("enter no of processes:"); scanf("%d",&n);

printf("enter the time slice:"); scanf("%d",&ts); printf("enter the burst time"); for(i=0;i<n;i++)

{

printf("\n process%d\t",i+1); scanf("%d",&p[i].btime);

p[i].wtime=p[i].lst=0; p[i].pno=i+1; p[i].sbtime=p[i].btime;

}

printf("scheduling \n");

do

{

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

{

count=p[i].btime; if(count>0)

{

flag=-1; count=(count>=ts)?ts:count; printf("\n process %d",p[i].pno); printf("from%d",ptm); ptm+=count; printf("to%d",ptm);

p[i].btime-=count; if(pp!=i)

{

pp=i;

p[i].wtime+=ptm-p[i].lst-count; p[i].lst=ptm;

}

}

## OUTPUT:

|  |  |
| --- | --- |
|  | **CPU SCHEDULING ALGORITHMS** |
| **FCFS** |

**AIM:**

To write a C program for implementation of FCFS and SJF scheduling algorithms.

## ALGORITHM:

Step 1: Inside the structure declare the variables.

Step 2: Declare the variable i,j as integer,totwtime and totttime is equal to zero. Step 3: Get the value of „n‟ assign pid as I and get the value of p[i].btime.

Step 4: Assign p[0] wtime as zero and tot time as btime and inside the loop calculate wait time and turnaround time.

Step 5: Calculate total wait time and total turnaround time by dividing by total number of process.

Step 6: Print total wait time and total turnaround time. Step 7: Stop the program.

## PROGRAM:

#include<stdio.h> #include<stdlib.h> struct fcfs

{

int pid; int btime; int wtime; int ttime;

} p[10];

int main()

{

int i,n;

int towtwtime=0,totttime=0; printf("\n fcfs scheduling...\n"); printf("enter the no of process"); scanf("%d",&n); for(i=0;i<n;i++)

{

p[i].pid=1;

printf("\n burst time of the process”); scanf("%d",&p[i].btime);

}

p[0].wtime=0; p[0].ttime=p[0].btime; totttime+=p[i].ttime; for(i=0;i<n;i++)

{

p[i].wtime=p[i-1].wtime+p[i-1].btim

p[i].ttime=p[i].wtime+p[i].btime; totttime+=p[i].ttime; towtwtime+=p[i].wtime;

}

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

{{

printf("\n waiting time for process”); printf("\n turn around time for process”); printf("\n");

}}

printf("\n total waiting time :%d", totwtime ); printf("\n average waiting time :%f",(float)totwtime/n); printf("\n total turn around time :%d",totttime);

printf("\n average turn around time: :%f",(float)totttime/n);

}

## OUTPUT:

|  |  |
| --- | --- |
|  | **CPU SCHEDULING ALGORITHMS** |
| **SJF SCHEDULING** |

**AIM:**

To write a C program for implementation of SJF scheduling algorithms.

## ALGORITHM:

Step 1: Inside the structure declare the variables.

Step 2: Declare the variable i,j as integer,totwtime and totttime is equal to zero. Step 3: Get the value of „n‟ assign pid as I and get the value of p[i].btime.

Step 4: Assign p[0] wtime as zero and tot time as btime and inside the loop calculate wait time and turnaround time.

Step 5: Calculate total wait time and total turnaround time by dividing by total number of process.

Step 6: Print total wait time and total turnaround time. Step 7: Stop the program.

## PROGRAM:

#include<stdio.h> #include<stdlib.h> typedef struct

{

int pid; int btime; int wtime;

}

sp;

int main()

{

int i,j,n,tbm=0,towtwtime=0,totttime sp\*p,t;

printf("\n sjf schaduling ..\n"); printf("enter the no of processor"); scanf("%d",&n); p=(sp\*)malloc(sizeof(sp)); printf("\n enter the burst time"); for(i=0;i<n;i++)

{

printf("\n process %d\t",i+1); scanf("%d",&p[i].btime); p[i].pid=i+1;

p[i].wtime=0;

}

for(i=0;i<n;i++) for(j=j+1,j<n;j++)

{

if(p[i].btime>p[j].btime)

{

t=p[i]; p[i]=p[j]; p[j]=t;

}}

printf("\n process scheduling\n"); printf("\n process \tburst time \t w for(i=0;i<n;i++)

{

towtwtime+=p[i].wtime=tbm; tbm+=p[i].btime; printf("\n%d\t\t%d",p[i].pid,p[i].bt printf("\t\t%d\t\t%d",p[i].wtime,p[i

}

totttime=tbm+towtwtime;

printf("\n total waiting time :%d", totwtime ); printf("\n average waiting time :%f",(float)totwtime/n); printf("\n total turn around time :%d",totttime);

printf("\n average turn around time: :%f",(float)totttime/n);

}

**OUTPUT:**

**RESULT:**

|  |  |
| --- | --- |
|  | **PAGE REPLACEMENT ALGORITHMS** |
| **FIFO** |

**AIM:**

To write a C program for implementation of FIFO page replacement algorithm.

## ALGORITHM:

Step 1: Start the program.

Step 2: Declare the necessary variables. Step 3: Enter the number of frames.

Step 4: Enter the reference string end with zero.

Step 5: FIFO page replacement selects the page that has been in memory the longest time and when the page must be replaced the oldest page is chosen.

Step 6: When a page is brought into memory, it is inserted at the tail of the queue. Step 7: Initially all the three frames are empty.

Step 8: The page fault range increases as the no of allocated frames also increases. Step 9: Print the total number of page faults.

Step 10: Stop the program.

## PROGRAM:

#include<stdio.h> int main()

{

int i=0,j=0,k=0,i1=0,m,n,rs[30],flag=1,p[30]; system("clear");

printf("FIFO page replacement algorithm. \\n");

printf("enter the no. of frames:"); scanf("%d",&n);

printf("enter the reference string:"); while(1)

{

scanf("%d",&rs[i]); if(rs[i]==0)

break; i++;

}

m=i; for(j=0;j<n;j++) p[j]=0;

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

{

flag=1; for(j=0;j<n;j++) if(p[j]==rs[i])

{

printf("data already in page \n");

flag=0; break;

}

if(flag==1)

{

p[i1]=rs[i]; i1++;

k++;

if(i1==n) i1=0;

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

{

printf("\n page %d:%d",j+1,p[j]); if(p[j]==rs[i])

printf("\*");

}

printf("\n\n");

}

}

printf("total no page faults=%d",k);

}

## OUTPUT:

|  |  |
| --- | --- |
|  | **PAGE REPLACEMENT ALGORITHMS** |
| **LRU** |

**AIM:**

To write a c program to implement LRU page replacement algorithm.

**ALGORITHM:**

Step 1: Start the process Step 2: Declare the size

Step 3: Get the number of pages to be inserted Step 4: Get the value

Step 5: Declare counter and stack

Step 6: Select the least recently used page by counter value Step 7: Stack them according the selection.

Step 8: Display the values Step 9: Stop the process

## ROGRAM:

#include<stdio.h> main()

{

int q[20],p[50],c=0,c1,d,f,i,j,k=0,n,r,t,b[20],c2[20]; printf("Enter no of pages:");

scanf("%d",&n);

printf("Enter the reference string:"); for(i=0;i<n;i++) scanf("%d",&p[i]);

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

q[k]=p[k]; printf("\n\t%d\n",q[k]); c++;

k++;

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

{ c1=0;

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

{

if(p[i]!=q[j])

c1++;

}

if(c1==f)

{c++;

if(k<f)

{q[k]=p[i]; k++;

for(j=0;j<k;j++) printf("\t%d",q[j]); printf("\n");

}

else

{for(r=0;r<f;r++)

{c2[r]=0;

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

{if(q[r]!=p[j]) c2[r]++;

else break;

}}

for(r=0;r<f;r++) b[r]=c2[r]; for(r=0;r<f;r++)

{

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

{

if(b[r]<b[j])

{

t=b[r]; b[r]=b[j]; b[j]=t;

}}}

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

{

if(c2[r]==b[0])

q[r]=p[i]; printf("\t%d",q[r]);

}

printf("\n");

}}}

printf("\nThe no of page faults is %d",c);

}

|  |  |
| --- | --- |
|  | **PAGE REPLACEMENT ALGORITHMS** |
| **OPTIMAL** |

**#include<stdio.h>**

**int main()**

**{**

**int no\_of\_frames, no\_of\_pages, frames[10], pages[30], temp[10], flag1, flag2, flag3, i, j, k, pos, max, faults = 0;**

**printf("Enter number of frames: ");**

**scanf("%d", &no\_of\_frames);**

**printf("Enter number of pages: ");**

**scanf("%d", &no\_of\_pages);**

**printf("Enter page reference string: ");**

**for(i = 0; i < no\_of\_pages; ++i){**

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

**}**

**for(i = 0; i < no\_of\_frames; ++i){**

**frames[i] = -1;**

**}**

**for(i = 0; i < no\_of\_pages; ++i){**

**flag1 = flag2 = 0;**

**for(j = 0; j < no\_of\_frames; ++j){**

**if(frames[j] == pages[i]){**

**flag1 = flag2 = 1;**

**break;**

**}**

**}**

**if(flag1 == 0){**

**for(j = 0; j < no\_of\_frames; ++j){**

**if(frames[j] == -1){**

**faults++;**

**frames[j] = pages[i];**

**flag2 = 1;**

**break;**

**}**

**}**

**}**

**if(flag2 == 0){**

**flag3 =0;**

**for(j = 0; j < no\_of\_frames; ++j){**

**temp[j] = -1;**

**for(k = i + 1; k < no\_of\_pages; ++k){**

**if(frames[j] == pages[k]){**

**temp[j] = k;**

**break;**

**}**

**}**

**}**

**for(j = 0; j < no\_of\_frames; ++j){**

**if(temp[j] == -1){**

**pos = j;**

**flag3 = 1;**

**break;**

**}**

**}**

**if(flag3 ==0){**

**max = temp[0];**

**pos = 0;**

**for(j = 1; j < no\_of\_frames; ++j){**

**if(temp[j] > max){**

**max = temp[j];**

**pos = j;**

**}**

**}**

**}**

**frames[pos] = pages[i];**

**faults++;**

**}**

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

**for(j = 0; j < no\_of\_frames; ++j){**

**printf("%d\t", frames[j]);**

**}**

**}**

**printf("\n\nTotal Page Faults = %d", faults);**

**return 0;**

**}**