# Operating System Experiments Solution Set

**Q1.Explain the Ls and Mv Linux commands.**

Ans: i) **mv** stands for move. mv is used to move one or more files or directories from one place to another in a file system like UNIX. It has two distinct functions:

(i) It renames a file or folder.

(ii) It moves a group of files to a different directory.

No additional space is consumed on a disk during renaming. This command normally works silently means no prompt for confirmation.

**Syntax:**

mv [Option] source destination

ex. $ mv exp4.c exp5.c

ii) **ls** is a Linux shell command that lists directory contents of files and directories.

**Syntax:**

$ ls [options] [file|dir]

ex. $ ls

**Q2. Explain the cat and touch Linux commands.**

**Ans:** Basically, there are two different commands to create a file in the Linux system which is as follows:

i)**cat command:** It is used to create the file with content. This command displays the contents of one or more files without having to open the file for editing. The cat command also allows us to write some texts into a file.

**Syntax:**

$ cat >test.txt

Hello, hi this test.txt file 1.

ii)**touch command:** It is used to create a file without any content. The file created using touch command is empty. This command can be used when the user doesn’t have data to store at the time of file creation.

**Syntax:**

$ touch exp1.c

$ls

**Q3. Explain the CD and Mkdir Linux commands.**

**Ans:** i) **cd** command, also known as chdir (change directory), is a command-line shell command used to change the current working directory in various operating systems. It can be used in shell scripts and batch files.

**Syntax:**

$ pwd  // to know present current directory.

$ cd filename  // name of the directory, where you like to go.

$ cd  // to go back.

ii) **The mkdir**(make directory) command is used to create a new directory.

**Syntax:**

$ mkdir filename

**Q4. REPEATED**

**Q5. Explain vi editor and nano editor of linux.**

**Ans:** i) The **vi editor** is the most popular and commonly used Unix text editor. It is usually available in all Linux Distributions. It works in two modes, Command and Insert. Command mode takes the user commands, and the Insert mode is for editing text. You should know the commands to work on your file easily.

**Syntax:**

$ vi filename

ii) **Nano** is a simple, modeless, WYSIWYG command-line text editor included in most Linux installations. With a simple easy to use interface, it is a great choice for Linux beginners. he syntax files are stored in the /usr/share/nano directory and included by default in the /etc/nanorc configuration file.

**Syntax:**

$ nano filename

**Q6. Explain chmod, pwd Linux command.**

**Ans: i)**The Linux command **chmod** allows you to control exactly who is able to read, edit, or run your files. Chmod is an abbreviation for change mode; if you ever need to say it out loud, just pronounce it exactly as it looks: ch'-mod.

To use chmod, you need to know about access modes. Each file on a Linux system has nine access modes (or settings) that determine exactly who can do what to the file. Chmod is the command that lets you change these settings.

There are three classes of people:

**user (u)**

the person who created the file

**group (g)**

people in a selected group

**other (o)**

everyone else on the system

For each class of people there are three classes of permissions:

**read (r)**

ability to see the contents of the file

**write (w)**

ability to change the contents of the file

**execute (x)**

ability to execute the contents of the file

Controlling access with chmod

In order to control the access users may have to your file or directory, use the ‘change mode’ program, chmod.

The chmod command allows changing of permissions using the letters u, g, and o (user, group, and others) and r, w, and x (read, write, and execute). For example, to turn off others’ write permission you can issue the command:

```
chmod o-w filename
```

(you might translate “o-w” as “for others, take away write permission.”)

To turn write permission back on you would say:

chmod o+w filename

(similarly, “for others, add write permission.”)

You can group changes together with commas. For example, in order to make a file readable by the public but writable by your group, you might use the command:

chmod g+rw,o+r filename

To remove write permission from your group later on, you could issue the command:

chmod g-w filename

Another way to achieve the same result would be to use the command

chmod g=r filename

ii**) pwd** stands for Print Working Directory. It prints the path of the working directory, starting from the root. pwd is shell built-in command(pwd) or an actual binary(/bin/pwd). $PWD is an environment variable which stores the path of the current directory.

**Syntax:**

$ pwd

**Q7. Explain the Man and help Linux commands.**

**Ans**: i)**man** command in Linux is used to display the user manual of any command that we can run on the terminal. It provides a detailed view of the command which includes NAME, SYNOPSIS, DESCRIPTION, OPTIONS, EXIT STATUS, RETURN VALUES, ERRORS, FILES, VERSIONS, EXAMPLES, AUTHORS.

**Syntax:**

$ man [COMMAND NAME]

**1. No Option:** It displays the whole manual of the command.

**Syntax :**

$ man [COMMAND NAME]

Example:

$ man printf

**2. Section-num:** Since a manual is divided into multiple sections so this option is used to display only a specific section of a manual.

**Syntax:**

$ man [SECTION-NUM] [COMMAND NAME]

Example:

$ man 2 intro

**3. -f option:** One may not be able to remember the sections in which a command is present. So this option gives the section in which the given command is present.

**Syntax:**

$ man -f [COMMAND NAME]

Example:

$ man -f ls

**4. -a option:** This option helps us to display all the available intro manual pages in succession.

**Syntax:**

$ man -a [COMMAND NAME]

Example:

$ man -a intro

**5. -k option:** This option searches the given command as a regular expression in all the manuals and it returns the manual pages with the section number in which it is found.

**Syntax:**

$ man -k [COMMAND NAME]

Example:

$ man -k cd

**6. -w option:** This option returns the location in which the manual page of a given command is present.

Syntax:

$ man -w [COMMAND NAME]

Example:

$ man -w ls

**7. -I option:** It considers the command as case sensitive.

Syntax:

$ man -I [COMMAND NAME]

Example:

$ man -I printf

ii) The **help** command is the simplest way to get information regarding a built-in shell command. It helps you fetch information from the shell's internal documentation. It takes a text string as the command line argument and looks for the provided string in the shell's documents.

Help command itself offers three options:

**-d:** display only a brief description of the specified command.

**-m:** organize the available information just as the man command does.

**-s:** display the command syntax of the specified command.

**Syntax:**

$ help -s pwd

$ help -s cd

$ help -s help

**Q8. Explain the ps and top Linux commands.**

**Ans:i)** ps (processes status) is a native Unix/Linux utility for viewing information concerning a selection of running processes on a system: it reads this information from the virtual files in the /proc filesystem.

**Syntax:**

ps [options]

Options for ps Command:

**1.Simple process selection:** Shows the processes for the current shell –

[root@rhel7 ~]# ps

**2. View Processes:** View all the running processes use either of the following option with ps:

**i) View Processes not associated with a terminal:** View all processes except both session leaders and processes not associated with a terminal.

[root@rhel7 ~]# ps -a

**ii)View all the processes except session leaders:**

[root@rhel7 ~]# ps -d

**iii)View all processes associated with this terminal:**

[root@rhel7 ~]# ps -T

**iv)View all the running processes:**

[root@rhel7 ~]# ps -r

ii) The **top** (table of processes) command shows a real-time view of running processes in Linux and displays kernel-managed tasks. The command also provides a system information summary that shows resource utilization, including CPU and memory usage. In this tutorial, you will learn to use the top command in Linux.

**Syntax:**

1) **Shows top command syntax:**
   top -h

**2) Batch Mode: Send output from top to file or any other programs.**

top -b

**3) Secure Mode: Use top in Secure mode.**

top -s

**4) Command Line: The below command starts top with last closed state.**

top -c

**5) Delay time: It tells delay time between screen updates.**

top -d seconds.tenths

**Q9. Explain rm and cp linux command.**

**Ans: i) cp** stands for copy. This command is used to copy files or group of files or directory. It creates an exact image of a file on a disk with different file name. cp command require at least two filenames in its arguments.

**Syntax:**

$ cp file1name file2name

ii) **rm** stands for remove here. rm command is used to remove objects such as files, directories, symbolic links and so on from the file system like UNIX. To be more precise, rm removes references to objects from the filesystem, where those objects might have had multiple references (for example, a file with two different names). By default, it does not remove directories.

**Syntax:**

$ rm filename or $ rm directory.

**Q10. Explain the who and whoami Linux commands**.

**Ans**: i)**whoami** command is used both in Unix Operating System and as well as in Windows Operating System.

It is basically the concatenation of the strings “who”,”am”,”i” as whoami.

It displays the username of the current user when this command is invoked.

It is similar as running the id command with the options -un.

**Syntax:**

$ whoami

**ii)**The Linux "**who**" command lets you display the users currently logged in to your UNIX or Linux operating system. Whenever a user needs to know about how many users are using or are logged-in into a particular Linux-based operating system, he/she can use the "who" command to get that information.

**Syntax:**

$ who

**Q11. Explain the ping and uname Linux commands.**

**Ans:** i)**PING** (Packet Internet Groper) command is used to check the network connectivity between host and server/host. This command takes as input the IP address or the URL and sends a data packet to the specified address with the message “PING” and get a response from the server/host this time is recorded which is called latency.

**Syntax:**

ping www.geeksforgeeks.org

ii) **Uname** command is used to display basic information about the operating system and hardware. With options, Uname prints kernel details, and system architecture. Uname is the short name for 'UNIX name'. Uname command works on all Linux and Unix like operating systems.

**Syntax:**

$ uname

-a option: It prints all the system information in the following order: Kernel name, network node hostname, kernel release date, kernel version, machine hardware name, hardware platform, operating system.

**Syntax:** $ uname -a

-s option: It prints the kernel name.

**Syntax:** $uname -s

-n option: It prints the hostname of the network node(current computer).

**Syntax:** $uname -n

-r option: It prints the kernel release date.

**Syntax:** $uname -r

-v option: It prints the version of the current kernel.

**Syntax:** $uname -v

-m option: It prints the machine hardware name.

**Syntax:** $uname -m

p option: It prints the type of the processor.

**Syntax:** $uname -p

**Q12. Explain the du and df Linux commands.**

**Ans: du command,** short for disk usage, is used to estimate file space usage.

The du command can be used to track the files and directories which are consuming excessive amount of space on hard disk drive.

Syntax: $du

ii) The **df** command (short for disk free), is used to display information related to file systems about total space and available space. If no file name is given, it displays the space available on all currently mounted file systems.

Syntax: $ df

**Q13. REPEATED**

**Q14. Explain the grep and sort Linux commands.**

**Ans:** i) The **grep** filter searches a file for a particular pattern of characters, and displays all lines that contain that pattern. The pattern that is searched in the file is referred to as the regular expression (grep stands for global search for regular expression and print out).

**Syntax:**

**1. Case insensitive search :** The -i option enables to search for a string case insensitively in the given file. It matches the words like “UNIX”, “Unix”, “unix”.

$grep -i "UNix" geekfile.txt

**Output:**

unix is great os. unix is opensource. unix is free os.

Unix linux which one you choose.

uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.

**2. Displaying the count of number of matches : We can find the number of lines that matches the given string/pattern**

$grep -c "unix" geekfile.txt

**Output:**

2

**3. Display the file names that matches the pattern : We can just display the files that contains the given string/pattern.**

$grep -l "unix" *

**or**

$grep -l "unix" f1.txt f2.txt f3.xt f4.txt

**Output:**

geekfile.txt

**4. Checking for the whole words in a file : By default, grep matches the given string/pattern even if it is found as a substring in a file. The -w option to grep makes it match only the whole words.**

$ grep -w "unix" geekfile.txt

**Output:**

unix is great os. unix is opensource. unix is free os.

uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.

**5. Displaying only the matched pattern : By default, grep displays the entire line which has the matched string. We can make the grep to display only the matched string by using the -o option.**

$ grep -o "unix" geekfile.txt

**Output:**

unix

unix

unix

unix

unix

unix

**6. Show line number while displaying the output using grep -n : To show the line number of file with the line matched.**

$ grep -n "unix" geekfile.txt

**Output:**

1:unix is great os. unix is opensource. unix is free os.

4:uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a powerful.

**7. Inverting the pattern match : You can display the lines that are not matched with the specified search string pattern using the -v option.**

$ grep -v "unix" geekfile.txt

**Output:**

learn operating system.

Unix linux which one you choose.

ii) **SORT** command is used to sort a file, arranging the records in a particular order. By default, the sort command sorts file assuming the contents are ASCII. Using options in the sort command can also be used to sort numerically.

**Syntax:**

$ cat > filename

$ sort filename

**Q15. Write shell script to find OS version and kernel version.**

**Ans: i) OS version:** Using os-release file available in Linux's etc directory.

We can check the Linux Operating System (OS) info by running the below command

**Syntax:**

~$ cat /etc/os-release

ii) **kernel version:**

The kernel is the essential center of a computer operating system (OS). It is the core that provides basic services for all other parts of the OS. It is the main layer between the OS and hardware, and it helps with process and memory management, file systems, device control and networking**.**

**Syntax:** $ uname -r

**Q16. Write shell script to find current logged in user and log name.**

**Ans:** i**) To find current logged in user:** The **w** command shows information about the Linux users currently on the server, and their running processes. The first line displays, in this order:

The current time ( 22:11:17 )

How long the Linux server has been running (18 days)

How many users are currently logged on Linux (2 users)

The system load averages for the past 1, 5, and 15 minutes (1.01, 1.04, 1.05)

**Syntax:** $ w

ii) You can display or print the name of the current user (also know as calling user) using **logname** command.

**Syntax:** $ logname

**Q17. Write shell script to find home directory and current working directory.**

**Ans:i) Home Directory:** This is the simplest. Just use the cd command and no further options.

**Syntax:** $ cd /home

~ also represents the user's home directory. Therefore, you can use this command to cd into the home directory.

$ cd ~

ii) **) pwd** stands for Print Working Directory. It prints the path of the working directory, starting from the root. pwd is shell built-in command(pwd) or an actual binary(/bin/pwd). $PWD is an environment variable which stores the path of the current directory.

**Syntax:** $ pwd

**Q18. Write shell script to find top 5 processes in descending order.**

**Ans: i)**

**Q19. Write shell script to display addition of two numbers.**

**Ans: i)**

```
#!/bin/bash
# Calculate the sum of two integers with pre initialize values
# in a shell script

a=10
b=20

sum=$(( $a + $b ))

echo "Sum is: $sum"
```

**Q.20 Write shell script to display current date and time.**

**Ans**: You can use date command on Linux shell script to get current Date and Time. The date command is the part of the Linux Coreutils package.

**Syntax:** $ date

Or

$ currentDate=’date’

$ echo $currentDate

**Q21. Write shell script to find “entered number is one digit or two digit”.**

**Ans:**

**Q22. Write shell script to concatenate the inputted string during runtime.**

**Ans:**

**Q23. Write shell script to display substring from Main string (e.g consider string is "Welcome to the world of technology" from this display substring from position 6 to 12)**

**Ans:**

**Q24. Write shell script to display area of rectangle.**

**Ans:**
```
read -p "Enter a length: " length
read -p "Enter a width: " width
area=$[$length*$width]
echo "The area of the rectangle is $area"
```

**Q25. Explain the fork() and getppid() system calls.**

**Ans:** System call fork() is used to create processes. It takes no arguments and returns a process ID. The purpose of fork() is to create a new process, which becomes the child process of the caller. After a new child process is created, both processes will execute the next instruction following the fork() system call. Therefore, we have to distinguish the parent from the child. This can be done by testing the returned value of fork():

If fork() returns a negative value, the creation of a child process was unsuccessful.

fork() returns a zero to the newly created child process.

fork() returns a positive value, the process ID of the child process, to the parent. The returned process ID is of type pid_t defined in sys/types.h. Normally, the process ID is an integer. Moreover, a process can use function getpid() to retrieve the process ID assigned to this process.

The getppid() function returns the parent process ID of the calling process. The getpgid() function returns the process group ID of the process whose process ID is equal to pid, or the process group ID of the calling process, if pid is equal to 0.

**Q26. Explain the read() and write() file system calls.**

**Ans:** The **read** system call interface is standardized by the POSIX specification. Data from a file is read by calling the read function: ssize_t read(int fd, void *buf, size_t count); The value returned is the number of bytes read (zero indicates end of file) and the file position is advanced by this number.

The **write** is one of the most basic routines provided by a Unix-like operating system kernel. It writes data from a buffer declared by the user to a given device, such as a file. This is the primary way to output data from a program by directly using a system call.

**Q27. Explain the getpid() and getppid() system calls.**

**Ans:** Both getppid() and getpid() are inbuilt functions defined in unistd.h library.

**getppid():** returns the process ID of the parent of the calling process. If the calling process was created by the fork() function and the parent process still exists at the time of the getppid function call, this function returns the process ID of the parent process. Otherwise, this function returns a value of 1 which is the process id for init process.

**Syntax:**

pid_t getppid(void);

**getpid():** returns the process ID of the calling process. This is often used by routines that generate unique temporary filenames.

**Syntax:**

pid_t getpid(void);

**Q28. Explain the open() and close() system calls.**

**Ans:** The **open()** system call opens the file specified by pathname. If the specified file does not exist, it may optionally (if O_CREAT is specified in flags) be created by open().

**Close():** It is used to end file system access. When this system call is invoked, it signifies that the program no longer requires the file, and the buffers are flushed, the file information is altered, and the file resources are de-allocated as a result.

**29. Write a program to demonstrate FIFO process scheduling algorithm.**

```
#include<stdio.h>

int main()

{

   int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;

   printf("Enter total number of processes(maximum 20):");

   scanf("%d",&n);
```

```
printf("nEnter Process Burst Timen");
for(i=0;i<n;i++)
{
   printf("P[%d]:",i+1);
   scanf("%d",&bt[i]);
}
wt[0]=0;
for(i=1;i<n;i++)
{
   wt[i]=0;
   for(j=0;j<i;j++)
      wt[i]+=bt[j];
}
printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time");
for(i=0;i<n;i++)
{
   tat[i]=bt[i]+wt[i];
   avwt+=wt[i];
   avtat+=tat[i];
   printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);
}
avwt/=i;
avtat/=i;
printf("\n\nAverage Waiting Time:%d",avwt);
printf("\nAverage Turnaround Time:%d",avtat);
return 0;
```

```
}
```

**30. Write a program to demonstrate SJF process scheduling algorithm.**

```
#include<stdio.h>
int main()
{
    int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
    float avg_wt,avg_tat;
    printf("Enter number of process:");
    scanf("%d",&n);
    printf("nEnter Burst Time:n");
    for(i=0;i<n;i++)
    {
        printf("p[%d]:",i+1);
        scanf("%d",&bt[i]);
        p[i]=i+1;
    }
   //sorting of burst times
    for(i=0;i<n;i++)
    {
        pos=i;
        for(j=i+1;j<n;j++)
        {
            if(bt[j]<bt[pos])
                pos=j;
        }
```

```
    temp=bt[i];
    bt[i]=bt[pos];
    bt[pos]=temp;
     temp=p[i];
    p[i]=p[pos];
    p[pos]=temp;
}
wt[0]=0;
for(i=1;i<n;i++)
{
    wt[i]=0;
    for(j=0;j<i;j++)
        wt[i]+=bt[j];
       total+=wt[i];
}
avg_wt=(float)total/n;
total=0;
printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
for(i=0;i<n;i++)
{
    tat[i]=bt[i]+wt[i];
    total+=tat[i];
    printf("\np[%d]\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
}
avg_tat=(float)total/n;
printf("nnAverage Waiting Time=%f",avg_wt);
```

```
    printf("nAverage Turnaround Time=%fn",avg_tat);
    RETURN 0;
}
```

**31. Write a program to demonstrate Priority based process scheduling algorithm.**

```
#include<stdio.h>
int main()
{
    int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg_wt,avg_tat;
    printf("Enter Total Number of Process:");
    scanf("%d",&n);
    printf("\nEnter Burst Time and Priority\n");
    for(i=0;i<n;i++)
    {
        printf("\nP[%d]\n",i+1);
        printf("Burst Time:");
        scanf("%d",&bt[i]);
        printf("Priority:");
        scanf("%d",&pr[i]);
        p[i]=i+1;           //contains process number
    }
    //sorting burst time, priority and process number in ascending order using selection sort
    for(i=0;i<n;i++)
    {
        pos=i;
        for(j=i+1;j<n;j++)
```

```
    {
        if(pr[j]<pr[pos])
            pos=j;
    }
    temp=pr[i];
    pr[i]=pr[pos];
    pr[pos]=temp;

    temp=bt[i];
    bt[i]=bt[pos];
    bt[pos]=temp;

    temp=p[i];
    p[i]=p[pos];
    p[pos]=temp;
}
wt[0]=0; //waiting time for first process is zero
//calculate waiting time
for(i=1;i<n;i++)
{
    wt[i]=0;
    for(j=0;j<i;j++)
        wt[i]+=bt[j];

    total+=wt[i];
}
```

```
    avg_wt=total/n;      //average waiting time
    total=0;
    printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i];     //calculate turnaround time
        total+=tat[i];
        printf("\nP[%d]\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
    }
    avg_tat=total/n;     //average turnaround time
    printf("\n\nAverage Waiting Time=%d",avg_wt);
    printf("\nAverage Turnaround Time=%d\n",avg_tat);
    return 0;
}
```

**32. Write a program to demonstrate Producer consumer problem with semaphore.**

```
#include<stdio.h>
#include<stdlib.h>
int mutex = 1;
int full = 0;
int empty = 10,x = 0;
void producer()
{
    --mutex;
    ++full;
    --empty;
```

```
    x++;
    printf("Producer produces" "item %d", x);
    ++mutex;
}
void consumer()
{
    --mutex;
    --full;
    ++empty;
    printf("Consumer consumes " "item %d", x);
    x--;
    ++mutex;
}
int main()
{
    int n,i;
    printf ("\n1. Press 1 for Producer" "\n2. Press 2 for Consumer" "\n3. Press 3 for Exit");
    for (i = 1; i > 0; i++)
    {
        printf ("\n\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;
    }
  }
}
```

**33. Write a program to demonstrate concept of deadlock avoidance through Banker's algorithm.**

```
#include<stdio.h>
#include<conio.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("********** Banker's Algo ************\n");
input();
show();
cal();
getch();
return 0;
}
void input()
{
int i,j;
printf("Enter the no of Processes\t");
```

```
scanf("%d",&n);
printf("Enter the no of resources 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 safe[100];
int i,j;
for(i=0;i<n;i++)
{
finish[i]=0;
}
//find need matrix
for(i=0;i<n;i++)
{
for(j=0;j<r;j++)

{
need[i][j]=max[i][j]-alloc[i][j];
}
}
printf("\n");
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;
}
printf("P%d->",i);
if(finish[i]==1)
{
i=n;
}
}
}
}
}
}
for(i=0;i<n;i++)
{
if(finish[i]==1)
{
c1++;
}
```

```
else
{
printf("P%d->",i);
}

}
if(c1==n)
{
printf("\n The system is in safe state");
}
else
{
printf("\n Process are in dead lock");
printf("\n System is in unsafe state");
}
}
```

**34. Write a program to demonstrate concept of FIFO page replacement police.**

```
#include<stdio.h>
int main()
{
int i,j,n,a[50],frame[10],no,k,avail,count=0;
printf("\n ENTER THE NUMBER OF PAGES:\n");
scanf("%d",&n);
printf("\n ENTER THE PAGE NUMBER :\n");
for(i=1;i<=n;i++)
```

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

```
}
```

**35. Write a program to demonstrate concept of LRU page replacement police.**

```
#include<stdio.h>
int findLRU(int time[], int n)
{
   int i, minimum = time[0], pos = 0;
   for(i = 1; i < n; ++i)
   {
      if(time[i] < minimum)
      {
         minimum = time[i];
         pos = i;
      }
   }
   return pos;
}
int main()
{
   int no_of_frames, no_of_pages, frames[10], pages[30], counter = 0, time[10], flag1, flag2,
i, j, pos, faults = 0;
   printf("Enter number of frames: ");
   scanf("%d", &no_of_frames);
   printf("Enter number of pages: ");
   scanf("%d", &no_of_pages);
```

```
printf("Enter 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])
      {
         counter++;
         time[j] = counter;
         flag1 = flag2 = 1;
         break;
       }
   }
   if(flag1 == 0)
   {
      for(j = 0; j < no_of_frames; ++j)
      {
```

```
            if(frames[j] == -1)
            {
                counter++;
                faults++;
                frames[j] = pages[i];
                time[j] = counter;
                flag2 = 1;
                break;
            }
        }
    }
    if(flag2 == 0)
    {
        pos = findLRU(time, no_of_frames);
        counter++;
        faults++;
        frames[pos] = pages[i];
        time[pos] = counter;
    }
    printf("\n");
    for(j = 0; j < no_of_frames; ++j)
    {
        printf("%d\t", frames[j]);
    }
}
printf("\n\nTotal Page Faults = %d", faults);
```

```
    return 0;
}
/*
Output

Enter number of frames: 3
Enter number of pages: 6
Enter reference string: 5 7 5 6 7 3

5 -1 -1
5 7 -1
5 7 -1
5 7 6
5 7 6
3 7 6

Total Page Faults = 4
*/
```

**36. Write a program to demonstrate concept of FCFS disk scheduling algorithm.**

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int RQ[100],i,n,TotalHeadMoment=0,initial;
    printf("Enter the number of Requests\n");
```

```
    scanf("%d",&n);
    printf("Enter the Requests sequence\n");
    for(i=0;i<n;i++)
     scanf("%d",&RQ[i]);
    printf("Enter initial head position\n");
    scanf("%d",&initial);
    for(i=0;i<n;i++)
    {
       TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
       initial=RQ[i];
    }
    printf("Total head moment is %d",TotalHeadMoment);
    return 0;
}
```

**37. Write a program to demonstrate concept of SCAN disk scheduling algorithm**

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;
    printf("Enter the number of Requests\n");
    scanf("%d",&n);
    printf("Enter the Requests sequence\n");
    for(i=0;i<n;i++)
     scanf("%d",&RQ[i]);
```

```
printf("Enter initial head position\n");
scanf("%d",&initial);
printf("Enter total disk size\n");
scanf("%d",&size);
printf("Enter the head movement direction for high 1 and for low 0\n");
scanf("%d",&move);
for(i=0;i<n;i++)
{
   for(j=0;j<n-i-1;j++)
   {
      if(RQ[j]>RQ[j+1])
      {
         int temp;
         temp=RQ[j];
         RQ[j]=RQ[j+1];
         RQ[j+1]=temp;
      }
   }
}
int index;
for(i=0;i<n;i++)
{
   if(initial<RQ[i])
   {
      index=i;
      break;
```

```
    }
}
// if movement is towards high value
if(move==1)
{
    for(i=index;i<n;i++)
    {
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
        initial=RQ[i];
    }
    //  last movement for max size
    TotalHeadMoment=TotalHeadMoment+abs(size-RQ[i-1]-1);
    initial = size-1;
    for(i=index-1;i>=0;i--)
    {
         TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
         initial=RQ[i];
    }
}
// if movement is towards low value
else
{
    for(i=index-1;i>=0;i--)
    {
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
        initial=RQ[i];
```

```
        }
        //  last movement for min size
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i+1]-0);
        initial =0;
        for(i=index;i<n;i++)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
            initial=RQ[i];
        }
    }
    printf("Total head movement is %d",TotalHeadMoment);
    return 0;
}
```