1. who

→ show who is logged on

→ Prints information about users who are currently logged in.

$ who -a [all info]

$ who -b [time of last system boot]

2. whoami - print effective user id.

→ Prints the username of the current user.

3. uname - print system information.

→ Prints certain system information. With no OPTION, same as -s.

$ uname -a
All system info in order.

$ uname -s
Print kernel name

$ uname -n
Print network node hostname.

$ uname -r
Print kernel release.

$ uname -v
Print kernel version.

$ uname -p
Print processor type.

$ uname -o
Print operating system.

4. man - an interface to system reference manuals.

→ Prints the user manual page of any command.

$ man -f [command-name]
Display short description from man page.
same as $ whatis [command-name].

$ man -k [command-name]
Display short descriptions of all matched commands.
same as $ apropos [commandname]

5. ls - list directory contents.

→ Lists information about all files (of current directory by default).

$ ls -a.

print all files including files starting with .

$ ls -h

print in human readable format. used with -l and -s to print sizes as 1K 234M 2G etc.

$ ls -l

print long list format (permissions, size, modification time etc)

$ ls -s

print the sizes of the each file.

$ ls -S

print files in sorted order by file size. largest first.

$ ls -t

print files in sorted order by modification time. newest first.

$ ls -1

list files in one line each.

6. pwd - present working directory.

→ prints name of current/working directory.

$ pwd has no arguments or options.

7. cd - change working directory

→ change the present working directory to specified path.

$ cd /
change to root directory.

$ cd ~
change to home directory.

$ cd ..
change to parent directory.

$ cd /home/user/test
changes to test in user directory from anywhere. (if exists)

$ cd sample
changes to sample directory in the present working directory. (if exists)

8. mkdir - make directories

→ create directories at given path.

$ mkdir test
$ mkdir test/test1 (X) [works only if test exists]
$ mkdir -p test/test1 [works also if test doesn't exist]
} Use -p for creating sub directories. if parent directory does not exist.

$ mkdir test1 test2 test3
multiple creation of direction.

$ mkdir -m=*** test
creates directory test with given permissions mode.

*** can be rrr, rwx, 777 etc.

9. rmdir - remove empty directories.

→ remove directories at given path if they are empty.

$ rmdir test

$ rmdir test1 test2

$ rmdir test/test1 [removes test1 but not test]

$rmdir -p test/test1 [removes test1 and then test also]

Use -p to remove parent directories as well.

10. rm - remove files or directories.

$ rm test.txt

$ rm test.c test1.txt

$ rm -d test

remove directory test (if empty). same as $ rmdir test

* $ rm -rd test

recursively delete test and all of its contents.

11. clear - clear the terminal screen.

12. nl - number lines of files.

→ print / display any content with line numbers added.

$ nl test.c

prints/displays content of test.c line by line with line numbers.

$ ls | nl

prints contents of directory line by line with line numbers.

13. cat – concatenate files and print to the output.

→ cat command can be used to display the content, append content or overwrite content of a file.

$ cat file.txt

displays contents of file.txt.

$ cat > file.txt

creates a file as file.txt if it did not exist already and gives space to enter text as input which overwrites the previous content of the file.

$ cat >> file.txt

same as > but appends instead of overwriting.

$ cat file1.txt file2.txt ....

display multiple files.

$ cat file1 file2 file3 ... > file_merged

multiple files file1, file2, file3, ... are written into file_merged.

$ cat -n file.txt

displays contents of file.txt with numbered lines.

$ tac file.txt

displays content in reverse order.

* Here "file.txt" is just an example. Any file containing text can be used.

14. cp – copy files or directories.

→ copy files or directories from source to destination

$ cp f1.txt /home/user/folder (or) $ cp f1.txt f2.txt

f1.txt: source file; /home/user/folder: destination directory

f1.txt: source file; f2.txt: destination file

(or)

$ cp -r folder1 folder3/folder2

-r: option. folder1: source directory; folder3/folder2: destination directory

Options:

-i → prompt before overwriting.

-l → hard link files instead of copying.

-n → donot overwrite an existing file.

-r → recursive - copy directories recursively.

-s → soft link / symbolic link files instead of copying.

-u → update – copy only if if source is newer than dest file or when the destination file is not present.

15. mv – move files or directories.

→ move files or directories from source to destination.

[same syntax as of cp] [same options: -i, -n, -u as of cp]

* mv command can be used to rename files or directories by moving them to a different destination name in same directory.

1. wc – print newline, word & byte counts of a file.

→ prints the no. of newlines, words and bytes present in the given file as argument or in a command output through pipe.

$ wc file.txt

output format : newline_count word_count byte_count file_name.

$ wc file1.txt file2.txt file3.txt ...

same output format, each file counts are displayed in each line followed by total counts in last line.

$ wc -l file.txt

-l prints just no. of newlines in the file.

$ wc -m file.txt

-m prints just the no. of characters in the file.

$ wc -c file.txt

-c prints the byte count of the file.

$ wc -w file.txt

-w prints the word count of the file.

2. du – disk usage

→ print disk usage in KB for files and directories. in the present directory.

options :

-h → human readable format

-a → all files including directories recursively.

3. df - report file system disk space usage.

→ prints used, available and usage percentages of disk space for the file system

options:

-a → include pseudo, duplicate file systems.

-h → human readable format

4. uniq - report or omit repeated lines.

→ uniq command checks for duplicate adjacent lines and prints the file content removing/omitting duplicates.

[Not all repeated lines are considered as duplicates, only adjacent repeated lines are considered duplicates and removed.]

$ uniq file.txt

options:

-c → gives no. of times a line is duplicated.

-d → only prints repeated lines once.

-D → prints all duplicates.

-u → prints only unique lines.

-i → make comparision case-insensitive.

5. cmp – compare two files byte by byte.

→ compares each byte of both files and returns the byte value at which there is a difference (if they have any).

$ cmp file1 file2

Output format: file1 file2 differ: byte byte_value, line line_value

options:

-l → Displays all the differences found byte by byte.

-b → Displays the differing bytes and characters.

6. comm – compare two sorted files line by line.

→ compares two files and gives output as 3 columns where 1st column contains lines unique to file1
2nd column contains lines unique to file2
3rd column contains lines common to both.

$ comm file1 file2

7. diff – compare files line by line.

→ Displays the differences in the files by comparing line by line.

$ diff file1 file2

options:

-i → ignore case in comparision.

8. chmod – change mode of access for a file or directory.

→ chmod can be used to change the file/directory access permission for three types of users.

1. user (u)
2. group (g)
3. others (o).
4. all (a) or (ugo).

modes: r → read (4), w → write (2), x → execute (1).

1 → x only ; 2 → w only ; 3 → w, x ; 4 → r only

5 → r, x ; 6 → r, w ; 7 → r, w, x.

→ file access modes can be displayed using [$ ls -l].

```
- - - - - - - - - -
d r w x r w x r w x
↓   u     g     o
```

- → file.
d → directory.

Adding permissions : $ chmod u+rwx, g+rw, o+r file

$ chmod ugo+rw file or a+rw file

Removing permissions : $ chmod ugo-x file.

$ chmod go-wx file

Setting permissions using numbers : $ chmod 764 file.

9. ln - make links between files.

Hardlink: (default)

$ln file1 link1

Soft link (symbolic link):

$ln -s file1 link1

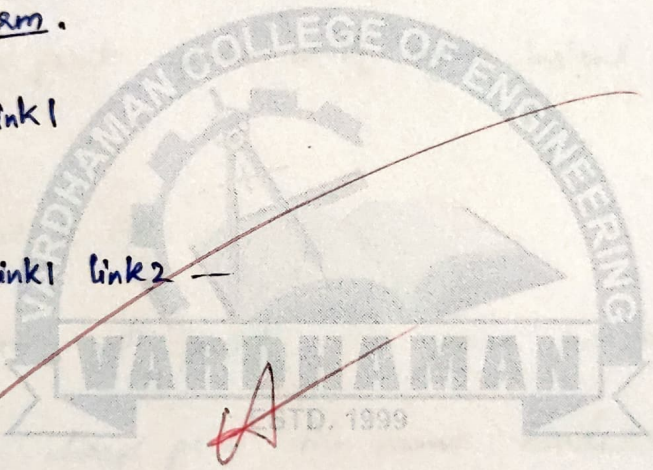
10. unlink - remove links from a file system.

similar to rm.

$ unlink link1

(or)

$ unlink link1 link2 —



1. head - output first part of files.

→ By default, prints first 10 lines of a file to output.

$ head file

$ head file1 file2

options:

-n → used to print given no. of lines instead of default = 10. lines.

$ head -n 5 file (or) $ head -5 file

-c → used to print given no. of bytes instead of default = 10 lines.

$ head -c 3 file

prints only first 3 bytes

-q → used to not print file name headers while printing multiple.

-v → used to always print file name headers.

2. tail - output last part of files.

→ By default, prints last 10 lines of a file to output.

→ head and tail similar except that head prints first part and tail prints last part.

→ All options of head apply to tail as well.

3. sort – sort lines of text

$ sort file.txt

options:

-n → numeric sort → compare numerical value.

-r → reverse → reverse the result of comparision.

-c → check → checks if the text is already sorted. Will not sort.

→ Numbers will appear before letters.

→ Lowercase letters will appear before Uppercase letters.

4. cut – cut/remove sections from each line of files.

options:

-b → select only these bytes.

$ cut -b 1-3,5,7 file

-c → select only these characters.

$ cut -c 1,2-5,7 file.

-d → use given character as delimiter instead of default: TAB.

[we should use -f option to mention which fields need to be displayed while using a custom delimiter.]

$ cut -d " " -f 1,3-5,8 file.

. suppose file has "s1 hi hello s2 here who are you".

output: s1 hello s2 here you.

5. paste – join files parallely line by line with each file seperated by tab (by default).

$ paste file1 file2 file3

output:

line1_of_file_1	line1 of file 2	line1_of_file3
line2 of file1	line2 of file2	line 2 of file 3
⋮		

options:

-d – delimiter ⇒ $ paste -d "|" file1 file2

prints | b/w the lines of different files instead of tab.

-s – serial → $ paste -s file1 file2

output:

line1 of file1	line2 of file1	_ ___
line1 of file2	line2 of file2	_ _

prints serially instead of parallely

6. join – join lines of two files on a common field

$ join file1 file2

$ join file1 file2 > file3

options:

-i → ignore case

7. grep – Global Regular Expression Print

→ searches through a file for a specific sequence of characters and returns all the lines containing it.

$ grep "string" file

options:

-c → prints only count of lines that match

-h → display matched lines, but do not display filenames.

-i → ignore case.

-n → display matched lines with their line number.

-v → invert match → display unmatched lines.

-o → display only the matched sequence. or string.

-f → keep the patterns in a file ⇒ $grep -f pattern.txt file

8. egrep – extended regular expression.

→ faster than grep.

→ can search multiple patterns.

$ egrep 'pattern1 | pattern2 | pattern3' file

9. fgrep – fixed strings ; instead of regular expressions.

→ fastest approach as no substitution of regular expressions.

$ fgrep "string" file.

Week – 4

4.a – Process Management calls – fork(), wait()

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int main()
{
    pid_t pid;
    pid = fork();
    if(pid<0)
    {
        printf("fork() failed to create a child\n");
        return -1;
    }
    else if(pid>0)
    {
        printf("The ID of parent process is %d\n",getpid());
        printf("\nParent process running...\n");
        printf("Parent process is pausing...\n");
        wait(NULL);
        printf("\nParent process started running again...\n");
        printf("End of Parent process.\n");
    }
    else
    {
        printf("\nChild process running...\n");
        printf("The ID of child process is %d\n",getpid());
        printf("The ID of parent of this child process is %d\n",getppid());
        execlp("/usr/bin/uname","uname",NULL);
    }
    return 0;
}
```

Output:

The ID of parent process is 469

Parent process running...
Parent process is pausing...

Child process running...
The ID of child process is 470
The ID of parent of this child process is 469
Linux

Parent process started running again...
End of Parent process.

Week – 5

5.a – Two way communication using pipes

```c
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>

void main()
{
    int n, p1[2], p2[2];
    pid_t pid;
    char msg[100];
    if(pipe(p1) == -1)
    {
        printf("pipe failed\n");
        return;
    }
    if(pipe(p2) == -1)
    {
        printf("pipe failed\n");
        return;
    }
    pid = fork();
    if(pid<0)
    {
        printf("pipe failed\n");
        return;
    }
    if(pid>0)
    {
        close(p2[1]);
        close(p1[0]);
        printf("Enter the msg from Parent (Server) to Child (Client) : ");
        scanf("%[^\n]s",msg);
        write(p1[1],msg,strlen(msg)+1);
        sleep(5);
        read(p2[0],msg,100);
        printf("The msg given by Child (Client) ---> Parent is \"%s\"\n",msg);
    }
    else
    {
        close(p2[0]);
        close(p1[1]);
        sleep(3);
        read(p1[0],msg,100);
        printf("The msg given by Parent (Server) ---> Child is \"%s\"\n",msg);
        sleep(3);
        printf("\nEnter the msg from Child (Client) to Parent (Server) : ");
        scanf("%[^\n]s", msg);
        write(p2[1],msg,strlen(msg)+1);
    }
}
```

Output:

Enter the msg from Parent (Server) to Child (Client) : Hello Child
The msg given by Parent (Server) ---> Child is "Hello Child"

Enter the msg from Child (Client) to Parent (Server) : Hi Parent
The msg given by Child (Client) ---> Parent is "Hi Parent"

5.b – Process Communication using FIFO

```c
//FIFO Communication - create fifo - FIFO_create.c
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
int main()
{
    int fd, res;
    char msg[100];
    res = mkfifo("fifo_1",0666);
    if(res == -1){
        printf("FIFO fifo_1 not created\n");
        return 0;
    }
    fd = open("fifo_1", O_WRONLY);
    if(fd == -1){
        printf("FIFO fifo_1 not opened\n");
        return 0;
    }
    printf("Enter the message to FIFO fifo_1 : ");
    scanf("%s", msg);
    write(fd, msg, sizeof(msg));
    close(fd);
    sleep(2);
    printf("\nMessage written to FIFO fifo_1 is : %s\n", msg);
    unlink("fifo_1");
    return 0;
}
```

```c
//FIFO Communication - access fifo - FIFO_access.c
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#define size 1024
int main()
{
    int fd;
    char buf[size];
    fd = open("fifo_1", O_RDONLY);
    if(fd == -1)
    {
        printf("FIFO fifo_1 not opened\n");
        return 0;
    }
    read(fd, buf, size);
    printf("\nMessage received from FIFO fifo_1 is : %s\n",buf);
    close(fd);
    return 0;
}
```

Output:

```
chsc_pop@pop-os: ~
chsc_pop@pop-os:~$ gcc FIFO_create.c
chsc_pop@pop-os:~$ ./a.out
Enter the message to FIFO fifo_1 : Hi@fifo

Message written to FIFO fifo_1 is : Hi@fifo
```

```
chsc_pop@pop-os: ~
chsc_pop@pop-os:~$ gcc FIFO_access.c
chsc_pop@pop-os:~$ ./a.out

Message received from FIFO fifo_1 is : Hi@fifo
```

Week – 6

IPC - Shared Memory

```c
//Shared memory - create Process – shm_create.c
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define SHMSZ 27
void main()
{
    char c='.', *shm, *s, str[100];
    int i, shmid;
    key_t key = 5678;

    if ((shmid = shmget(key, SHMSZ, IPC_CREAT | 0666)) < 0)
    {
        printf("Shared Memory cannot be created");
        exit(1);
    }

    if((shm = shmat(shmid, NULL, 0)) == (char *) -1)
    {
        printf("Shared memory cannot be attached");
        exit(1);
    }
    printf("Enter a message : ");
    scanf("%[^\n]s", str);
    for(i = 0; i < strlen(str); i++)
    {
        *shm++ = str[i];
    }
    *shm = '\0';
}
```

```c
//Shared memory - Access Process – shm_access.c
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <stdlib.h>
#define SHMSZ 27
void main()
{
    int shmid;
    key_t key = 5678;
    char *shm, *s;

    if ((shmid = shmget(key, SHMSZ, 0666)) < 0)
    {
        printf("shared memory cannot be opened");
        exit(1);
    }

    if ((shm = shmat(shmid, NULL, 0)) == (char *) -1)
    {
        perror("Can not be attached to shared memory");
        exit(1);
    }
    printf("Message received through shared memory is :\n");
    for (s = shm; *s != '\0'; s++)
        putchar(*s);
    putchar('\n');
}
```

Input given to shm_create.c:

Enter a message : Hello, This message is sent from shm_create.c

Output given by shm_access.c:

Message received through shared memory is :
Hello, This message is sent from shm_create.c

Week – 7

IPC - Message Queue

```c
// Message Queue - Send Message - mq_sendmsg.c
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#define SIZE 128

typedef struct msgbuf
{
    long mtype;
    char mtext[SIZE];
} message_buf;
int main()
{
    int msqid;
    int msgflg = IPC_CREAT|0666;
    key_t key;
    message_buf sbuf;
    size_t buf_length;
    char msg[100];
    key = 1234;
    if ((msqid = msgget(key, msgflg )) < 0)
    {
        printf("msgget-msg queue can not be created \n");
        return 1;
    }
    printf("\nMessage Queue created with ID : %d.\n", msqid);
    sbuf.mtype = 1;
    printf("\nEnter the message to be sent : ");
    scanf("%[^\n]s",msg);
    strcpy(sbuf.mtext, msg);
    buf_length = strlen(sbuf.mtext) + 1 ;
    if (msgsnd(msqid, &sbuf, buf_length, IPC_NOWAIT) < 0)
        return 1;
    else
        printf("\nMessage \"%s\" is sent to the Message Queue.\n", sbuf.mtext);
    return 1;
}
```

```c
// Message Queue - Receive message - mq_receivemsg.c
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#include<stdio.h>
#include<stdlib.h>
#define SIZE 128
typedef struct msgbuf
{
    long mtype;
    char mtext[SIZE];
} message_buf;

int main()
{
    int msqid;
    key_t key;
    message_buf rbuf;
    key = 1234;
    if ((msqid = msgget(key, 0666)) < 0)
    {
        printf("msgget- can not open Message Queue");
        exit(1);
    }
    if (msgrcv(msqid, &rbuf, SIZE, 1, 0) < 0)
    {
        printf("msgrcv- error in receiving");
        exit(1);
    }
    printf("\n Message Received From the Queue : %s \n", rbuf.mtext);
    exit(0);
}
```

Output given by & Input given to mq_sendmsg.c:

Message Queue created with ID : 0.

Enter the message to be sent : Hello, message sent from mq_sendmsg.c

Output given by mq_receivemsg.c:

Message Recievied From the Queue : Hello, message sent from mq_sendmsg.c

Week – 8

8.a - Head Command Simulation

```c
// Head Command Simulation  //  execute as "./a.out [number_of_lines] [file_name]"
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
    FILE *f;
    char *line;
    size_t len = 0;
    int count = 0;
    if(argc < 3)
    {
        printf("Insufficient Arguments\n");
        return -1;
    }
    f = fopen(argv[2], "r");
    if(f == NULL)
    {
        printf("File cannot be opened\n");
        return -1;
    }
    while(getline(&line, &len, f) != -1)
    {
        count++;
        if(count > atoi(argv[1]))
            break;
        printf("%s", line);
        fflush(stdout);
    }
    fclose(f);
    return 0;
    /*
    // Head Command execution using execlp()
    FILE *f;
    char *line;
    int len = 0;
    int count = 0;
    if(argc < 2)
    {
        printf("Insufficient Arguments\n");
        return -1;
    }
    f = fopen(argv[1], "r");
    if(f == NULL)
    {
        printf("File cannot be opened\n");
        return -1;
    }
    execlp("/usr/bin/head","head","-5", argv[1], NULL);
    return 0;
    */
}
```

Content of test.txt:

Hello
Hi
How are you
I am fine

Execution in terminal:

```
$ gcc head.c
$ ./a.out 2 test.txt
```

Output:

Hello
Hi

8.b - Tail Command Simulation

```c
// Tail Command Simulation  //  execute as "./a.out [number_of_lines] [file_name]"
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
    FILE *f;
    char *line;
    int i, n, count = 0;
    size_t len = 0;
    if(argc < 3)
    {
        printf("Insufficient Arguments\n");
        return -1;
    }
    f = fopen(argv[2], "r");
    if(f == NULL)
    {
        printf("File cannot be opened\n");
        return -1;
    }
    while(getline(&line, &len, f) != -1)
    {
        count++;
    }
    printf("\nTotal no. of lines : %d\n", count);
    rewind(f);
    n = count - atoi(argv[1]);
    i = 0;

    while(getline(&line, &len, f) != -1)
    {
        i++;
        if(i > n)
            printf("%s", line);
    }
    fclose(f);
    return 0;
}
```

Content of test.txt:

Hello
Hi
How are you
I am fine

Execution in terminal:

```
$ gcc tail.c
$ ./a.out 3 test.txt
```

Output:

Total no. of lines : 4
Hi
How are you
I am fine

Week – 9

Shell Script program using control statements to check whether the given name is a file or a directory

```
echo "Enter file or dir name : "
read file
if [ -d $file ]
    then
    echo "Given name is a directory: $file"
    echo "The list of files in the directory are: "
    cd $file
    ls -l
    cd ..
elif [ -f $file ]
    then
    echo "Given name is a file: $file"
    echo "\nNo of lines in file are : "
    wc -l $file
    echo "\nThe contents of the file are : "
    cat $file
else
    echo "Given name is neither a file nor a directory"
fi
```

Execution for checking a file: $ sh week9.sh

Output:

Enter file or dir name :
sam.c
Given name is a file: sam.c

No of lines in file are :
5 sam.c

The contents of the file are :
#include <stdio.h>
void main()
{
   printf("Hello World");
}

Execution for checking a directory: $ sh week9.sh

Output:

Enter file or dir name :
sam_dir
Given name is a directory: sam_dir
The list of files in the directory are:
total 0
-rwxrwxrwx 1 ubuntu_wsl ubuntu_wsl 62 Feb 24 20:01 sam_dir_file.c

Week – 10

Shell Script program using control statements to find the factorial of a number

```
echo "Enter a number to find out it's factorial : "
read n
i=1
f=1
while [ $i -le $n ]
do
f=`expr $f \* $i`
i=`expr $i + 1`
done
echo "Factorial of $n is $f."
```

Execution: $ sh week10.sh

Output:

Enter a number to find out it's factorial:
6
Factorial of 6 is 720.

Week – 11

11.a - Implement FIFO Page replacement algorithm

```c
// FIFO Page Replacement Algorithm
#include<stdio.h>

void main()
{
    int i, j, k, m, n, pf=0, count=0, rs[30], mf[10];
    printf("\nEnter the length of the reference string : ");
    scanf("%d",&n);
    printf("\nEnter the reference string : ");
    for(i=0;i<n;i++)
        scanf("%d",&rs[i]);
    printf("\nEnter the number of frames : ");
    scanf("%d",&m);
    for(i=0;i<m;i++)
        mf[i] =-1;
    printf("\nThe FIFO page replacement process is \n");
    for(i=0; i<n ;i++)
    {
        for(k=0; k<m; k++)
        {
            if( mf[k] == rs[i])
                break;
        }
        if(k == m)
        {
            mf[count] = rs[i];
            count++;
            pf++;
        }
        for(j=0; j<m; j++)
            printf("\t%d",mf[j]);
        if(k == m)
            printf("\t\tPage Fault no. %d", pf);
        else
            printf("\t\tPage Hit");
        printf("\n");
        if(count == m)
            count = 0;
    }
    printf("\nUsing FIFO replacement,\nNo. of Page Faults --> %d\nPage Fault Ratio --> %d/%d\n", pf, pf, n);
}
```

Output:

Enter the length of the reference string : 10

Enter the reference string : 3 1 4 2 1 5 2 1 6 4

Enter the number of frames : 4

The FIFO page replacement process is

3	-1	-1	-1	Page Fault no. 1
3	1	-1	-1	Page Fault no. 2
3	1	4	-1	Page Fault no. 3
3	1	4	2	Page Fault no. 4
3	1	4	2	Page Hit
5	1	4	2	Page Fault no. 5
5	1	4	2	Page Hit
5	1	4	2	Page Hit
5	6	4	2	Page Fault no. 6
5	6	4	2	Page Hit

Using FIFO replacement,
No. of Page Faults --> 6
Page Fault Ratio --> 6/10

11.b - Implement LRU Page replacement algorithm

```c
#include<stdio.h>
void main()
{
    int i, j, k, n, min, rs[25], flag[25], f, pf=0, next=1, count[20], m[20];
    printf("Enter the length of reference string : ");
    scanf("%d",&n);
    printf("\nEnter the reference string : ");
    for(i=0;i<n;i++) {
        scanf("%d",&rs[i]);
        flag[i] = 0;
    }
    printf("\nEnter the number of frames : ");
    scanf("%d",&f);
    for(i=0;i<f;i++) {
        count[i] = 0;
        m[i] = -1;
    }
    printf("\nThe LRU page replacement process is \n" );
    for(i=0;i<n;i++)
    {
        for(j=0;j<f;j++)
        {
            if(m[j]==rs[i])
            {
                flag[i] = 1;
                count[j]= next;
                next++;
            }
        }
        if(flag[i] == 0)
        {
            if(i < f) {
                m[i] = rs[i]; count[i]= next; next++; }
            else
            {
                min = 0; for(j=1;j<f;j++)
                if(count[min] > count[j]) min = j;
                    m[min] = rs[i]; count[min] = next; next++;
            }
            pf++;
        }
        for(j=0; j<f; j++)
            printf("\t%d",m[j]);
        if(flag[i] == 0)
            printf("\t\tPage Fault no. %d", pf);
        else
            printf("\t\tPage Hit");
        printf("\n");
    }
    printf("\nUsing LRU replacement,\nNo. of Page Faults --> %d\nPage Fault Ratio --> %d/%d\n", pf, pf, n);
}
```

Output:

Enter the length of reference string : 10

Enter the reference string : 3 1 4 2 1 5 2 1 6 4

Enter the number of frames : 4

The LRU page replacement process is

3	-1	-1	-1	Page Fault no. 1
3	1	-1	-1	Page Fault no. 2
3	1	4	-1	Page Fault no. 3
3	1	4	2	Page Fault no. 4
3	1	4	2	Page Hit
5	1	4	2	Page Fault no. 5
5	1	4	2	Page Hit
5	1	4	2	Page Hit
5	1	6	2	Page Fault no. 6
4	1	6	2	Page Fault no. 7

Using LRU replacement,
No. of Page Faults --> 7
Page Fault Ratio --> 7/10

Bonus 11.c - Implement LIFO Page replacement algorithm

```c
// LIFO Page Replacement Algorithm
#include<stdio.h>

void main()
{
    int i, j, k, m, n, pf=0, count=0, rs[30], mf[10];
    printf("\nEnter the length of the reference string : ");
    scanf("%d",&n);
    printf("\nEnter the reference string : ");
    for(i=0;i<n;i++)
        scanf("%d",&rs[i]);
    printf("\nEnter the number of frames : ");
    scanf("%d",&m);
    for(i=0;i<m;i++)
        mf[i] =-1;
    printf("\nThe LIFO page replacement process is \n");
    for(i=0; i<n ;i++)
    {
        for(k=0; k<m; k++)
        {
            if( mf[k] == rs[i])
                break;
        }
        if(k == m)
        {
            mf[count] = rs[i];
            if(count < m-1)
                count++;
            pf++;
        }
        for(j=0; j<m; j++)
            printf("\t%d",mf[j]);
        if(k == m)
            printf("\t\tPage Fault no. %d", pf);
        else
            printf("\t\tPage Hit");
        printf("\n");
    }
    printf("\nUsing LIFO replacement,\nNo. of Page Faults --> %d\nPage Fault Ratio --> %d/%d\n", pf, pf, n);
}
```

Output:

Enter the length of the reference string : 10

Enter the reference string : 3 1 4 2 1 5 2 1 6 4

Enter the number of frames : 4

The LIFO page replacement process is

3	-1	-1	-1	Page Fault no. 1
3	1	-1	-1	Page Fault no. 2
3	1	4	-1	Page Fault no. 3
3	1	4	2	Page Fault no. 4
3	1	4	2	Page Hit
3	1	4	5	Page Fault no. 5
3	1	4	2	Page Fault no. 6
3	1	4	2	Page Hit
3	1	4	6	Page Fault no. 7
3	1	4	6	Page Hit

Using LIFO replacement,
No. of Page Faults --> 7
Page Fault Ratio --> 7/10

Bonus 11.d - Implement MRU Page replacement algorithm

```c
// MRU Page Replacement Algorithm
#include<stdio.h>

void main()
{
    int i, j, k, m, n, pf=0, count=0, rs[30], mf[10];
    printf("\nEnter the length of the reference string : ");
    scanf("%d",&n);
    printf("\nEnter the reference string : ");
    for(i=0;i<n;i++)
        scanf("%d",&rs[i]);
    printf("\nEnter the number of frames : ");
    scanf("%d",&m);
    for(i=0;i<m;i++)
        mf[i] =-1;
    printf("\nThe MRU page replacement process is \n");
    for(i=0; i<n ;i++)
    {
        for(k=0; k<m; k++)
        {
            if( mf[k] == rs[i])
            {
                count = k;
                break;
            }
        }
        if(k == m)
        {
            mf[count] = rs[i];
            if(count < m-1 && i < m-1)
                count++;
            pf++;
        }
        for(j=0; j<m; j++)
            printf("\t%d",mf[j]);
        if(k == m)
            printf("\t\tPage Fault no. %d", pf);
        else
            printf("\t\tPage Hit");
        printf("\n");
    }
    printf("\nUsing MRU replacement,\nNo. of Page Faults --> %d\nPage Fault Ratio --> %d/%d\n", pf, pf, n);
}
```

Output:

Enter the length of the reference string : 10

Enter the reference string : 3 1 4 2 1 5 2 1 6 4

Enter the number of frames : 4

The MRU page replacement process is

3	-1	-1	-1	Page Fault no. 1
3	1	-1	-1	Page Fault no. 2
3	1	4	-1	Page Fault no. 3
3	1	4	2	Page Fault no. 4
3	1	4	2	Page Hit
3	5	4	2	Page Fault no. 5
3	5	4	2	Page Hit
3	5	4	1	Page Fault no. 6
3	5	4	6	Page Fault no. 7
3	5	4	6	Page Hit

Using MRU replacement,
No. of Page Faults --> 7
Page Fault Ratio --> 7/10

Week – 12

12.a - Implement FCFS Disk scheduling algorithm

```c
//FCFS (First Come First Serve) Disk Management Algorithm
#include<stdio.h>
#include <stdlib.h>
void main()
{
    int i, n, head, sum=0;

    printf("*** FCFS Disk Scheduling Algorithm ***\n\n");
    printf("Enter the size of the Disk Queue : ");
    scanf("%d", &n);
    int queue[n], t[n];

    printf("Enter the initial position of head : ");
    scanf("%d", &head);

    printf("Enter the Disk Queue elements : ");
    for(i=0; i<n; i++)
    {
        scanf("%d", &queue[i]);
        t[i] = abs(head - queue[i]);
        head = queue[i];
        sum += t[i];
    }
    printf("\nMovement of total cylinders = %d\n", sum);
}
```

Output:

*** FCFS Disk Scheduling Algorithm ***

Enter the size of the Disk Queue : 10
Enter the initial position of head : 7
Enter the Disk Queue elements : 4 97 28 65 1 54 89 99 43 12

Movement of total cylinders = 451

12.b - Implement SSTF Disk scheduling algorithm

```c
//SSTF (Shortest Seek Time First) Disk Management Algorithm
#include <stdio.h>
#include <stdlib.h>
void main()
{
    int i, j, n, temp, head, st=0;

    printf("*** SSTF Disk Scheduling Algorithm ***\n\n");
    printf("Enter the size of the Disk Queue : ");
    scanf("%d", &n);
    int queue[n], t[n];

    printf("Enter the initial position of head : ");
    scanf("%d", &head);

    printf("Enter the Disk Queue elements : ");
    for(i=0;i<n;i++)
    {
        scanf("%d", &queue[i]);
        t[i] = abs(head - queue[i]);
    }

    for(i=0;i<n;i++)
    {
        for(j=i+1;j<n;j++)
        {
            if(t[i]>t[j])
            {
                temp=t[i];  t[i]=t[j];t[j]=temp;
                temp=queue[i];  queue[i]=queue[j];  queue[j]=temp;
            }
        }
    }
    for(i=1;i<n-1;i++)
    {
        st += abs(head-queue[i]);
        head = queue[i];
    }
    printf("\nTotal Seek Time is = %d",st);
    printf("\nAverage Seek Time is = %f\n",st/(float)n);
}
```

Output:

*** SSTF Disk Scheduling Algorithm ***

Enter the size of the Disk Queue : 10
Enter the initial position of head : 7
Enter the Disk Queue elements : 4 97 28 65 1 54 89 99 43 12

Total Seek Time is = 112
Average Seek Time is = 11.200000

Week – 13

Implement Semaphore form of IPC

```c
#include <pthread.h>
#include <semaphore.h>
#include <stdlib.h>
#include <stdio.h>
#define MaxItems 2
#define BufferSize 2
sem_t empty;
sem_t full;
int in = 0;
int out = 0;
int buffer[BufferSize];
pthread_mutex_t mutex;
void *producer(void *pno)
{
    int item;
    for(int i = 0; i < MaxItems; i++) {
        item = rand();
        sem_wait(&empty);
        pthread_mutex_lock(&mutex);
        buffer[in] = item;
        printf("Producer %d: Insert Item %d at %d\n", *((int *)pno),buffer[in],in);
        in = (in+1)%BufferSize;
        pthread_mutex_unlock(&mutex);
        sem_post(&full);
    }
}
void *consumer(void *cno)
{
    for(int i = 0; i < MaxItems; i++) {
        sem_wait(&full);
        pthread_mutex_lock(&mutex);
        int item = buffer[out];
        printf("Consumer %d: Remove Item %d from %d\n",*((int *)cno),item, out);
        out = (out+1)%BufferSize;
        pthread_mutex_unlock(&mutex);
        sem_post(&empty);
    }
}
int main()
{
    pthread_t pro[2],con[2];
    pthread_mutex_init(&mutex, NULL);
    sem_init(&empty,0,BufferSize);
    sem_init(&full,0,0);
    int a[2] = {1,2};
    for(int i = 0; i < 2; i++)
        pthread_create(&pro[i], NULL, (void *)producer, (void *)&a[i]);
    for(int i = 0; i < 2; i++)
        pthread_create(&con[i], NULL, (void *)consumer, (void *)&a[i]);
    for(int i = 0; i < 2; i++)
        pthread_join(pro[i], NULL);
    for(int i = 0; i < 2; i++)
        pthread_join(con[i], NULL);
    pthread_mutex_destroy(&mutex);
    sem_destroy(&empty);
    sem_destroy(&full);
    return 0;
}
```

Execution: $ gcc -pthread ipc_semaphore.c $ ./a.out

Output:

Producer 1: Insert Item 41 at 0
Producer 2: Insert Item 41 at 1
Consumer 1: Remove Item 41 from 0
Consumer 2: Remove Item 41 from 1
Producer 1: Insert Item 18467 at 0
Producer 2: Insert Item 18467 at 1
Consumer 2: Remove Item 18467 from 0
Consumer 1: Remove Item 18467 from 1

*Some text are coloured green in the Execution-Output areas to represent text entered by the user while execution as a command or as an input.