ANURAG ENGINEERING COLLEGE

(An Autonomous Institution)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Vision

- To generate Competent Professionals to become part of the Industry and Research Organizations at the National and International levels.

Mission

- To train the students to have in-depth knowledge of the subjects in the field of Computer Science and Engineering.
- To train the students with leadership qualities, team work skills, commitment and ethics thereby making them develop confidence for R & D activities and for placement in multinational and national.

PROGRAM EDUCATIONAL OBJECTIVES

PEO I : Excel in professional career and/or higher education by acquiring knowledge in mathematical, computing and engineering principles

PEO II : Be able to analyze the requirements of the software, understand the technical specifications, design and provide novel engineering solutions and efficient product designs.

PEO III : Adopt to professionalism, ethical attitude, communication skills, team work, lifelong learning in their profession.

PROGRAM SPECIFIC OUTCOMES

PSO 1: **Problem Solving Skills**: Ability to use mathematical abstraction, algorithmic design and appropriate data structures to solve real world problems using different programming paradigms.

PSO 2: **Professional Skills**: Ability to develop computing solutions for problems in multidisciplinary areas by applying software engineering principles.

PSO 3: **Successful Career and Entrepreneurship Skills**: Gain knowledge in diverse areas of computer science, and management skills for successful career, entrepreneurship and higher studies

PROGRAM OUTCOMES

PO 1: Gain an ability to apply knowledge of mathematics, science and engineering fundamentals appropriate to the discipline.

PO 2: Develop the competence to identify, analyze, formulate and solve engineering problems.

PO 3: Acquire an ability to design, implement, and evaluate a computer-based system, process, component, or program to meet desired needs with appropriate consideration for public health and safety, cultural, societal and environmental considerations.

PO 4: Are capable to design and conduct experiments, analyze and interpret data in the field of computer science and engineering.

PO 5: Gain expertise to use the techniques, skills and modern engineering tools with proficiency in basic area of computer science and engineering.

PO 6: An ability to analyze the local and global impact of computing on individuals, organizations, and society.

PO 7: Knowledge of contemporary issues.

PO 8: Sensitive to engage in activities with conscious social responsibility adhering to ethical values.

PO 9: An ability to function effectively individually and on teams, including diverse and multidisciplinary, to accomplish a common goal.

PO 10: An ability to articulate professional ideas clearly and precisely in making written and oral presentations.

PO 11: Recognition of the need for and an ability to engage in continuing professional development.

PO 12: An understanding of engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects.

CO-PO Mapping:

	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12
CO 1												
CO 2												
CO 3												
CO 4												
CO 5												

GENERAL LABORATORY INSTRUCTIONS

1. Students are advised to come to the laboratory at least 5 minutes before (to the starting time), those who come after 5 minutes will not be allowed into the lab.
2. Plan your task properly much before to the commencement, come prepared to the lab with the synopsis / program / experiment details.
3. Student should enter into the laboratory with:
   a) Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure, Program, Expected Output, etc.,) filled in for the lab session.
   b) Laboratory Record updated up to the last session experiments
   c) Proper Dress code and Identity card.
4. Sign in the laboratory login register and occupy the computer system allotted to you by the faculty.
5. Execute your task in the laboratory, and record the results / output in the lab observation note book, and get certified by the concerned faculty.
6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.
7. Computer labs are established with sophisticated and high end branded systems, which should be utilized properly.
8. Students must take the permission of the faculty in case of any urgency to go out; if anybody found loitering outside the lab / class without permission during working hours will be treated seriously and punished appropriately.
9. Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

ANURAG ENGINEERING COLLEGE

(An Autonomous Institution)

IV Year B.Tech. CSE - I Sem	L	T / P / D	C
	0	2	1

(CS703PC) LINUX PROGRAMMING LAB

PREREQUISITES:

1. Any programming language, operating systems and a parallel course on unix programming.

CO-REQUISITE :

1. A course on “Unix Programming”

COURSE OBJECTIVES:

- To provide the foundation of Unix programming..
- To understand the Unix utilities.
- Be able to work with Bourne again shell (bash).
- To provide exploration of file concepts.
- To understand the process, role of kernel in process management, signal generation and handling.

WEEK 1

1. Write a shell script that accepts a file name, starting and ending numbers as argumentsand displays all the lines between the given line numbers.

2.Write a shell script that deletes all lines containing the specified word in one more filessupplied as arguments to it.

a. To delete first character

b. Deletes last second character in every line.

c. First word and second word goes to second word and first word in every line.

WEEK 2

3. Write a shell script that displays a list of all files in the current directory to which the user has read,write and execute permissions.

4. Write a shell script that receives any number of file names as arguments checks if every argument supplied is a file as its arguments, counts and reports the occurrence of each word that is present in the first argument file on other argument files.

5. Write a shell script that accepts a list of file names as its arguments, counts and reports the occurrence of each word that is present in the first argument file on other argument files.

WEEK 3

6. Write a shell script to list all of the directory files in a directory.

7. Write a shell script to find factorial of a given number.

WEEK 4

8. Implement in C the following Unix commands and System calls.

a. cat b. ls c. mv.

a. Implement in C the cat Unix command using system calls

b. Implement in C the following ls Unix command using system calls

c. Implement in C the Unix command mv using system calls

9. Write a C program to emulate the Unix ls – l command.

WEEK 5

10. Write a C program that takes one or more file or directory names as command line input and reports the following information on the file.

1. file type
2. number of links
3. read, write and execute permissions
4. time of last access

WEEK 6

11. Write a C program that redirects a standard output to a file. Ex: ls>f1.

12. Write a C program to create a child process and allow the parent to display "parent" and the child to display "child" on the screen.

WEEK 7

13. Write a C program to create a zombie process.

14. Write a C program that illustrates how an orphan is created.

WEEK 8

15. Write a C program that illustrates the following.

a) Creating a message queue.
b) Writing to a message queue.
c) Reading from a message queue.

WEEK 9

16. Write a C program that illustrates inter process communication using shared memory system calls.

WEEK 10

17. Write a C program that implements a producer-consumer system with two processes (using semaphores)

WEEK 11

18. Write a C program that illustrates file locking using semaphores.

WEEK 12

19. Write a C program that counts the number of blanks in a text file using standard I/O

WEEK 13

20. Write a C program that illustrates communication between two unrelated processes using named pipe.

COURSE OUTCOMES:

1. Will be able to describe and use the LINUX operating system.
2. Will be able to describe and use the fundamental LINUX system tools and utilities.
3. We will able to describe and write shell scripts in order to perform basic shell programming. Will be able to describe and understand the LINUX file system.

1	Write a shell script that accepts a file name, starting and ending numbers as arguments and displays all the lines between the given line numbers.

```
file=$1
a=$2
b=$3
x=`expr $a + 1`
y=`expr $b - $a - 1`
cat $file| tail -n +$x| head -$y

$ cat>f1.txt
    1
    2
    3
    4
    5
    6

$ Sh prgname.sh f1.txt 3 5
Output:
 4
```

2	Write a shell script that deletes all lines containing the specified word in one or more files supplied as arguments to it.

```
echo "enter any word"
read w
for i in $*
do
cat $i| grep -v $w > hi
echo "after deleting from file $i"
cat hi
done

$ cat>F2.txt
hi
hello
how r u

$ cat>f3.txt
hi
good mng

$sh prgname.sh f2.txt f3.txt
enter any word
hi

Output:
after deleting from file f2.txt
hello
how r u
after deleting from file f3.txt
good mng
```

2	a) To delete first character

```
sed 's/^.//' f1.txt

$ cat>f1.txt
hi
hello

$ sh programname.sh f1.txt

Output:
i
ello
```

2	b) Deletes last second character in every line.

```
echo $x | sed 's/.\(.\)$/\1/' file.txt

$cat>file.txt
Kiran
Sai
arun

$sh filename.sh

Output:
Kirn
Si
arn
```

2	c) First word and second word goes to second word and first word in every line.

```
awk '{print $2,$1}' shift.txt

$ cat>shift.txt

hi hello namashkar
good morning
afternoon good

$ sh 2swap.sh

Output:

hello hi
morning good
good afternoon
```

(or)

```
awk 'NF >=2{t=$2;$2=$1;$1=t}; {print}' shift1.txt

$cat>shift1.txt
hod sir
2nd year 1st year
4th year 3rd year

$ sh programname.sh

Output:

sir hod
year 2nd 1st year
year 4th 3rd year
```

3	Write a shell script that displays a list of all files in the current directory to which the user has read, write and execute permissions.

```
for i in *
do
if [ -s $i -a -f $i ]
then
if [ -r $i -a -w $i -a -x $i ]
then
echo $i
else
echo
fi
fi
done

$ sh prgname.sh


Output:
f1.txt
          f2.txt
```

4	Write a shell script that receives any number of file names as arguments checks if every argument supplied is a file as its arguments, counts and reports the occurrence of each word that is present in the first argument file on other argument files.

```
if [ $# -lt 2 ]
then
echo "invalid no of arguments"
exit
fi
str=`cat $1|tr '\n' ' '`
for a in $str
do
echo "word=$a"
count=0
s=0
for i in $*
do
if [ $i == $1 ]
then
continue
fi
s=`grep -c $a $i`
count=`expr $count + $s`
done
echo "count=$count"
done
```

```
$ cat>f4
abc
123
Xyz
$ cat>f5
bc
abc
123
xyz
xyz
xyz
123
$ cat>f6
Abc
123
xyx
123
xyz
567
123

$ sh prgname.sh f4 f5 f6

Output:
word=abc
count=2
word=123
count=5
word=xyz
count=4
```

5	Write a shell script that accepts a list of file names as its arguments, counts and reports the occurrence of each word that is present in the first argument file on other argument files.

```
for i in $*
do
if [ -f $i ]
then
echo "$i is a file"
echo "number of lines $i"
wc -l $i
fi
if [ -d $i ]
then
echo "$i is a directory"
fi
done

$cat>f1.txt
1
2
3
4
5
6

$ sh PRGNAME.sh f1

OUTPUT:

f1 is a file
number of lines f1
3 f1
          (or)


$ sh PRGNAME.sh “sub directory name”

OUTPUT:
“directory name” is a directory
```

6	Write a shell script to list all of the directory files in a directory

```
for i in *
do
if [ -d $i ]
then
echo $i
fi
done

$ sh prgname.sh

Output:
23
516
btech2016
lsdemo
```

7	Write a shell script to find factorial of a given number.

```
echo "enter any number"
read n
fact=1
while [ $n -gt 1 ]
do
fact=`expr $fact \* $n`
n=`expr $n - 1`
done
echo "fact of given num is: $fact "

$ sh prgname.sh

Output:
enter any number
5
fact of given num is: 120
```

8	a) Implement in C the cat Unix command using system calls

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/stat.h>
#include<fcntl.h>

int main(int argc,char **argv)
{
        int fd,n;
        char buf;
        fd=open(argv[1],O_RDONLY);

        while(n=read(fd,&buf,1)>0)
        {
printf("%c",buf);
        }

return(0);

}

$ Cat>f9.txt
Hi
Hello

$ cc prgname.c          //for compiling the program
$ ./a.out f9.txt        //for executing the program

Output:
hi
hello
```

8	b) Implement in C the following ls Unix command using system calls

```
#include<stdio.h>
#include<stdlib.h>
#include<fcntl.h>
#include<dirent.h>
#include<unistd.h>

int main()
{

DIR *dp;
        struct dirent *d;
        char dirname[20];

printf("enter directory name");
scanf("%s",dirname);
dp=opendir(dirname);

if(dp==NULL)
{
printf("error");
exit(-1);
}

while(d=readdir(dp))
printf("%s\n",d->d_name);
return(0);
}

$ cc prgname.c
$ ./a.out
Output:
enter  directory subdirectory name
.
.  .
Sample1.txt
Sample2.txt
Sample3.txt
```

8	c) Implement in C the Unix command mv using system calls

```
#include<stdio.h>
#include<fcntl.h>
#include<unistd.h>
#include<sys/stat.h>
int main( int argc, char *argv[] )
{
int fd1,fd2,n;
fd1=open(argv[1],O_RDONLY);
fd2=creat(argv[2],S_IWUSR);
rename(argv[1],argv[2]);
unlink(argv[1]);
printf("file is moved");
return(0);
}

$ cat >f7.txt
hi
hello
how r u

$ cat>f8.txt

$ cc programname.c
$ ./a.out f7.txt f8.txt

Output:

file is moved
```

9	Write a C program to emulate the Unix ls – l command.

```
#include<dirent.h>
#include<stdio.h>
#include<unistd.h>
#include<sys/stat.h>
#include<sys/types.h>

int main(int argc, char **argv)
{

DIR *dp;
struct dirent *dirp;

if((dp=opendir(argv[1]))==NULL)

printf("cant open %s,argv[1]");

     while((dirp=readdir(dp))!=NULL)
     {
        struct stat fileStat;
        stat(dirp->d_name,&fileStat);

 printf(dirp->d_name);
printf("---------\n");

printf("File Size:\t\t%d bytes\n",fileStat.st_size);

printf("Number of links:\t%d\n",fileStat.st_nlink);

printf("FIle inode:\t\t%d\n",fileStat.st_ino);

printf("File Permissions:\t");

printf((S_ISDIR(fileStat.st_mode))?"d":"-");

printf((fileStat.st_mode&S_IRUSR)?"r":"-");
printf((fileStat.st_mode&S_IWUSR)?"w":"-");
printf((fileStat.st_mode&S_IXUSR)?"x":"-");


printf((fileStat.st_mode&S_IRGRP)?"r":"-");
printf((fileStat.st_mode&S_IWGRP)?"w":"-");
printf((fileStat.st_mode&S_IXGRP)?"x":"-");


printf((fileStat.st_mode&S_IROTH)?"r":"-");
printf((fileStat.st_mode&S_IWOTH)?"w":"-");
printf((fileStat.st_mode&S_IXOTH)?"x":"-");
printf("\n\n");

printf("the file %s a symbolic
link\n",(S_ISLNK(fileStat.st_mode))?"is":"is not");
}

return 0;
}
```

```
$ cc.programname.c
$ ./a.out

Output:
.---------


File Size:              4096 bytes
Number of links:        7
FIle inode:             251812
File Permissions:       drwxrwxrwx

the file is not a symbolic link
..---------
File Size:              36864 bytes
Number of links:        21
FIle inode:             99
File Permissions:       drwx------

the file is not a symbolic link
1.txt---------
File Size:              36864 bytes
Number of links:        21
FIle inode:             99
File Permissions:       drwx------

the file is not a symbolic link
2.txt---------
File Size:              36864 bytes
Number of links:        21
FIle inode:             99
File Permissions:       drwx------

the file is not a symbolic link
3.txt---------
File Size:              36864 bytes
Number of links:        21
FIle inode:             99
File Permissions:       drwx------

the file is not a symbolic link
```

10	Write a C program that takes one or more file or directory names as command line input and reports the following information on the file. 1. file type 2. number of links 3. read, write and execute permissions 4. time of last access

```
#include<stdio.h>
#include<sys/stat.h>
#include<time.h>
#include<fcntl.h>
#include<sys/types.h>

int main(int argc, char *argv[])
{
int i,j;
struct stat a;
for(i=1;i<argc;i++)
{
printf("%s:", argv[i]);
stat(argv[i],&a);

if(S_ISDIR(a.st_mode))
{
printf("is a directory");
}

else
{
printf("is a regular file\n");
}

printf("*********file properties******\n");
printf("Inode Number: %d\n",a.st_ino);
printf("UID:%o\n",a.st_uid);
printf("GID:%o\n", a.st_gid);
printf("no of links:%d\n",a.st_nlink);
printf("last acces time:%s",asctime(localtime(&a.st_atime)));
printf("permission flag:%o\n",a.st_mode%512);
printf("size in bytes:%d\n",a.st_size);
printf("blocks allocated:%d\n",a.st_blocks);
printf("last modification time:%s\n",ctime(&a.st_atime));
}
}
```

```
$ cc filename.c
$ ./a.out arg1 arg2 …

/* $ ./a.out mekala */

Output:

Mekala : is a directory*********file properties******
Inode Number: 251812
UID:1750
GID:1750
no of links:7
last access time: Mon Jul 22 09:33:37 2019
permission flag:777
size in bytes:4096
blocks allocated:16
last modification time:Mon Jul 22 09:33:37 2019
```

11	Write a C program that redirects a standard output to a file. Ex: ls>f1.

```
#include<stdio.h>
#include<fcntl.h>
#include<sys/stat.h>
#include<unistd.h>

main()
{
        int fd,fd2;
        fd=open("test",O_RDONLY,0777);
        close(1);
        fd2=creat("sam7.txt",O_RDONLY);
        printf("this is how we redirect standard output to a file");
}

$ cc. prgname.c
$ ./a.out

Output:
this is how we redirect standard output to a file
```

12	Write a C program to create a child process and allow the parent to display "parent" and the child to display "child" on the screen.

```
#include<sys/stat.h>
#include<sys/types.h>
#include<stdio.h>
int main()
{
pid_t pid;
pid=fork();

if(pid<0)
printf("error");
else if(pid==0)
{
printf("\n child");
printf("\nchild id %d\n",getpid());
printf(" child's parent id %d\n",getppid());
}

else
{
printf("\n parent");
printf("\n parent id %d\n",getpid());
printf("parent's parent id %d\n",getppid());
}

return 0;
}
$ cc programname.c
$ ./a.out

Output:

child
child id 13712
child's parent id 13711
parent
parent id 13711
parent's parent id 11960
```

13	Write a C program to create a zombie process.

```
#include<unistd.h>
#include<signal.h>
#include<sys/stat.h>
#include<sys/types.h>

main()
{
int i;
pid_t pid;
pid= fork();
if(pid==0)
exit(0);
else
{
sleep(60);
wait(&i);
}
return 0;
}

$ cc programname.c
$ ./a.out

Output:

child id 5733
parent id 5732
[1] 5732
[it@linux ~]$ ps -t
PID TTY STAT TIME COMMAND
3242 pts/1 Ss 0:00 bash
5732 pts/1 S 0:00 ./a.out
5733 pts/1 Z 0:00 [a.out] <defunct>
5734 pts/1 R+ 0:00 ps -t
```

14	Write a C program that illustrates how an orphan is created.

```
#include<stdio.h>
#include<signal.h>

main()
{
pid_t pid;
pid=fork();
if(pid==0)
{
printf("child has started %d\n",getpid());
printf("parent id %d\n",getppid());
sleep(30);
}
else
{
printf("parent has started %d\n",getpid());
kill(getpid(),SIGKILL);
}
printf("after fork");
}

$ cc programname.c
$ ./a.out

Output:

child has started 5866
parent id 5865
[1] 5865

[it@linux ~]$ parent has started 5865
ps -t
PID TTY STAT TIME COMMAND
3242 pts/1 Rs 0:00 bash
5866 pts/1 S 0:00 ./a.out
5884 pts/1 R+ 0:00 ps -t
[1]+ Killed ./a.out
[it@linux ~]$ after fork
```

15	Write a C program that illustrates the following. a) Creating a message queue. b) Writing to a message queue. c) Reading from a message queue.

mserver.c

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include<stdlib.h>
#include<string.h>

struct msg
{
long mtype;
char mtext[100];
} m;

int main()
{
key_t key;
int msgid;
key = ftok("msgq123", 230);
msgid = msgget(key, 0666 | IPC_CREAT);
m.mtype = 1;
printf("Write Data : ");
strcpy(m.mtext,"welcome");
msgsnd(msgid, &m, sizeof(m), 0);
printf("Data send is : %s \n", m.mtext);
return 0;
}
```

mclient.c

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>

struct msg
{
long mtype;
char mtext[100];
} m;

int main()
{
key_t key;
int msgid;
key = ftok("msgq123", 230);
msgid = msgget(key, 0666 | IPC_CREAT);
msgrcv(msgid, &m, sizeof(m), 1, 0);
printf("Data Received is : %s \n", m.mtext);
msgctl(msgid, IPC_RMID, NULL);
return 0;
}
```

Output:

```
$ cc mser.c
$ ./a.out

Write Data : Data send is : welcome

$ cc mcli.c
$ ./a.out

Data Received is : welcome
```

16	Write a C program that illustrates inter process communication using shared memory system calls.

sserver.c

```
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
int main()
{
key_t key = ftok("shmfile",65);
int shmid = shmget(key,1024,0666|IPC_CREAT);
char *str = (char*) shmat(shmid,(void*)0,0);
printf("Write Data : ");
gets(str);
printf("Data written in memory: %s\n",str);
shmdt(str);
return 0;
}
```

sclient.c

```
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
int main()
{
key_t key = ftok("shmfile",65);
int shmid = shmget(key,1024,0666|IPC_CREAT);
char *str = (char*) shmat(shmid,(void*)0,0);
printf("Data read from memory: %s\n",str);
shmdt(str);
shmctl(shmid,IPC_RMID,NULL);
return 0;
}
```

Output:

```
$ cc sserver.c
$ ./a.out

write Data:niranjan
data written in memory :niranjan

$ cc sclient.c
$ ./a.out
Data read from memory:niranjan
```

17	Write a C program that implements a producer-consumer system with two processes (using semaphores)

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/sem.h>
#include<time.h>
#include<unistd.h>
#define num_loops 2
#define rano_max 10

int main(int argc,char* argv[])
{
int sem_set_id;
int child_pid,i,sem_val;
struct sembuf sem_op;
int rc;
struct timespec delay;
sem_set_id=semget(IPC_PRIVATE,2,0600);

if(sem_set_id==-1)
{
printf("main:semget");
exit(1);
}

printf("semaphore set created,semaphore setid %d\n",sem_set_id);
child_pid=fork();

switch(child_pid)
{
case -1:
printf("fork");
exit(1);
case 0:

for(i=0;i<num_loops;i++)
{
sem_op.sem_num=0;
sem_op.sem_op=-1;
sem_op.sem_flg=0;
semop(sem_set_id,&sem_op,1);
printf("Consumer:%d\n",i);
fflush(stdout);
}

break;
default:

for(i=0;i<num_loops;i++)
{
printf("Producer:%d\n",i);
fflush(stdout);
sem_op.sem_num=0;
sem_op.sem_op=1;
sem_op.sem_flg=0;
semop(sem_set_id,&sem_op,1);
```

```
if(rand()>3*(rano_max ))
{
          delay.tv_sec=0;
delay.tv_nsec=10;
nanosleep(&delay,NULL);
}
}

break;
}
return 0;
}
Output:
semaphore set created
semaphore set id '327690'
producer: '0'
consumer:'0'
producer:'1'
consumer:'1'
```

18	Write a C program that illustrates file locking using semaphores.

```
#include <stdio.h>
#include <sys/file.h>
#include<string.h>
#include<stdlib.h>
#include <error.h>
#include <sys/sem.h>
#define MAXBUF 100
#define KEY 1216
#define SEQFILE "seq_file"

int semid,fd;
void my_lock(int);
void my_unlock(int);

union semun
{
int val;
struct semid_ds *buf;
short *array;
}arg;

int main()
{
int child, i,n, pid, seqno;
char buff[MAXBUF+1];
pid=getpid();

if((semid=semget(KEY, 1, IPC_CREAT | 0666))== -1)
{
perror("semget");
exit(1);
}

arg.val=1;
if(semctl(semid,0,SETVAL,arg)<0)
perror("semctl");

if((fd=open(SEQFILE,2))<0)
{
perror("open");
exit(1);
}

pid=getpid();

for(i=0;i<=5;i++)
{
my_lock(fd);
lseek(fd,01,0);
if((n=read(fd,buff,MAXBUF))<0)
{
perror("read");
exit(1);
}
printf("pid:%d\n, Seq no:%d\n", pid, seqno);
seqno++;
sprintf(buff,"%dn", seqno);
```

```
n=strlen(buff);
lseek(fd,01,0);
if(write(fd,buff,n)!=n)
{
perror("write");
exit(1);
}
sleep(1);
my_unlock(fd);
}
}

void my_lock(int fd)
{
struct sembuf sbuf={0,-1,0};
        if(semop(semid,&sbuf,1)==0)
printf("Locking: Resource…..\n");
else
printf("Error in Lockn\n");
}

      void my_unlock(int fd)
{
struct sembuf sbuf={0, 1, 0};
if(semop(semid,&sbuf,1)==0)
printf("UnLocking: Resource…..\n");
else
printf("Error in Unlockn\n");
}

$ cc programname.c
 $ ./a.out
Output:


Locking: Resource…
pid:6982
, Seq no:0
UnLocking: Resource…
Locking: Resource…
pid:6982
, Seq no:1
UnLocking: Resource…
Locking: Resource…
pid:6982
, Seq no:2
UnLocking: Resource…
Locking: Resource…
pid:6982
, Seq no:3
UnLocking: Resource…
Locking: Resource…
pid:6982
, Seq no:4
UnLocking: Resource…
Locking: Resource…
pid:6982
, Seq no:5
UnLocking: Resource…
```

19	Write a C program that counts the number of blanks in a text file using standard I/O

```
#include<stdio.h>

int main()
{
int c,nl,nt,ns;
char ch;
FILE *fp;
nl=0;
nt=0;
ns=0;
fp=fopen("sample.txt","r");

while((ch=getc(fp))!=EOF)
{
if((ch=='\n'))
nl=nl+1;
if(ch=='\t')
nt=nt+1;
if(ch==' ')
ns=ns+1;
}
printf("Blanks: %d\n Tabs: %d\n New lines: %d\n",ns,nt,nl);
return 0;
}

$cat > sample.txt
Q 3 4 5 6
W e r 4
CTRL+D

$cc blanks.c
$./a.out

Output:
Blanks:5
Tabs:2
Newlines:2
```

20	Write a C program that illustrates communication between two unrelated processes using named pipe.

fserver.c

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

int main()
{
int fd;
mkfifo("myfifo",06666);
char str[50];
while(1)
{
fd=open("myfifo",O_WRONLY);
write(fd,"welcome",strlen("welcome"));
close(fd);
}
return 0;
}
```

fclient.c

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

int main()
{
int fd;
char str2[50];
fd=open("myfifo",O_RDONLY);
read(fd,str2,50);
printf("%s",str2);
close(fd);
return 0;
}
```

Output:

```
[anurag@localhost 585]$ cc fserver.c
[anurag@localhost 585]$ ./a.out&
[2] 7206
[anurag@localhost 585]$ cc fclient.c
[anurag@localhost 585]$ ./a.out
welcome[anurag@localhost 585]$
```