PART A

1) **Learn the use of basic UNIX commands -**
   a. To access information using date, history, man, who, whoami, ttptime, finger,cal'
   b. To display contents of files using cat, vi, more, head, tail, grep, cmp, wc
   c. To manage files using cat, cp, ls, mv,rm, chmod, find
   d. Process utilities using ps, pid, ppid, tty, time, kill, exit
   e. Directory handling utilities using cd, mkdir, rmdir, mv, pwd

```
#!/bin/bash
HISTFILE =~/.bash.history
set -o history

while true
do
        echo " 1. Display Date 2. Previously executed commands"
        echo " 3. User manual 4. List of Users"
        echo " 5. Current User 6. Current user running time"
        echo " 7. Information of current user 8. Calender"
       echo " 9.Display files content using cat"
       echo " 10. Display files content using vi"
       echo " 11. Display large files content using more"
       echo " 12. Display first few lines using head"
       echo " 13. Display last few lines using tail"
       echo " 14. Search Particular word in a file"
       echo " 15. Compare 2 files"
       echo " 16. Word count of a file"
       echo " 17. Copy file content"
       echo "18. List files and Diectories"
       echo " 19. Rename or move file content"
       echo " 20. Remove file"
```

```
 echo " 21. Change permission"
 echo " 22. Find file"
 echo " 23. Active process with pid and ppid"
 echo " 24. Name of terminal 25. Time taken by a process"
 echo " 26. Send signals to running processes 27. exit"
 echo " 28. Create a new directory"
 echo " 29. Change a new directory"
 echo " 30. Remove the directory"
 echo " 31. Print the present directory"
 echo "Enter Choice"
     read ch
case $ch in
     1) date ;;
     2) history 5 ;;
     3) man cat ;;
     4) who ;;
     5) whoami ;;
     6) uptime ;;
     7) finger ;;
     8) cal ;;
     9) echo "Enter a files in current folder"
          read f
         if [ -f $f ]
         then
              cat $f
             exit
        else
             echo "File does not exits"
         fi;;
    10) echo " create file and enter content in vi editor"
```

```
                    echo "Enter file name"
                   read f
                  vi $f;;
         11) echo " Enter the file name"
                read f
               cat $f | more ;;
       12) echo " Enter the file name"
read f
cat $f | head ;;
13) echo " Enter the file name"
        read f
        cat $f | tail ;;
14) echo "Enter the file name"
        read f
        echo " Enter the search word"
        read w
        grep $w $f ;;
15) echo " Enter 1st file name"
        read f1
        echo  " Enter 2nd file name"
        read f2
        cmp $f1 $f2 ;;
16) echo " Enter the file name"
        read f
        wc $f ;;
17) echo " Enter source filename"
        read s
        echo " Enter destination filename"
        read d
        cp $s $d ;;
```

```
18) ls ;;
19) echo " Enter sourse filename"
    read s
    echo " Enter destination filename"
    read d
    mv $s $d ;;
20) echo "Enter the filename to be deleted"
    read f
    rm $f ;;
21) echo " Enter the filename to change permission"
    read f
    chmod 777 $f ;;
22) echo " Enter the filename to find"
    read f
    find $f ;;
23) ps -eaf ;;
24) tty ;;
25) time sleep 3 ;;
26) kill -1 ;;
27) exit ;;
28) echo " Enter the name for new directory"
    read d
    mkdir $d ;;
29) echo " Enter the name of directory to change"
    read d
    cd $d ;;
30) echo " Enter the directory to remove"
    read d
    rmdir $d ;;
31) echo " Present working directory"
```

```
        pwd ;;
*) exit ;;
Esac
done
```

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

```
echo "enter the directory name"
read dir
if [ -d $dir ]
then
        ch $dir
        for file in *
        do
                if [ -f $file ]
                then
                        if [ -r $file -a -w $file -a -x $file ]
                        then
                                ls -ltr $file
                                echo "file permission are there"
                        else
                                echo "no permission"
                                chmod 777 $file
                                echo "permission given"
                                ls -ltr $file
                        fi
                fi
        done
fi
```

3. Write a shell script that accepts a list of file names as its arguments, count and reports the

occurrence of each word that is present in the last argument on other argument files.

```
if [ $# -lt 2 ]
then
        echo "enter more than one file name as argument"
        exit 1
fi
for i in `cat $@`

do
        echo "word= $i"
        echo "-------"
        grep -c "$i" $*
        echo "-------"
done
```

4. Write a shell script that accepts one or more file name as arguments and converts all of

them to uppercase, provided they exist in the current directory

```
if [ $# -lt 1 ]
then
        echo "no arguments"
        exit 1
else
        for file in $@
        do
                if [ ! -f $file ]
                then
                        exit 1
```

```
            else
                    echo $file | tr '[ a-z ]' '[ A-Z ]'
            fi
    done
fi
```

5. **Write grep commands to the following:**

a. To select the lines from a file that has exactly 2 characters.

b. To select the lines from a file that has more2 than one blank spaces.

```
while true
do
    echo "enter the file name"
    read filename
    echo "1. to select the lines from a file that has exactly 2 characters"
    echo "2. to select the lines from a file that hase more 2 than a blank spaces"
    echo "enter your choice"
    read ch
    case $ch in
    1) echo "lines that have only 2 characters are"
            grep -n ^..$ $filename ;;
    2) echo "lines that has more than 2 spaces are"
            grep '[[:space:]]\{2,\}' $filename > f1_result.txt
            cat f1_result.txt ;;
    *)exit;;
esac
done
```

6. **Write a shell script which accepts two file names as arguments. Compare the contents. If**

they are same, then delete the second file.

```
if [ $# -lt 1 ]
then
        echo "no arguments"
        exit 1
else
        if [ ! -f $1 -0 ! -f $2 ]
        then
                echo "file not existing"
                exit 1
        else
                if cmp $1 $2
                then
                        echo $1 and $2 have identical contents
                        rm $2
                        echo "second file is removed"
                else
                        echo $1 and $2 differ
                fi
        fi
fi
```

7. **Write a shell script**

a. to count number of lines that do not contain vowels.

b. to count number of characters, words, lines in a given file

```
while true
         do
                 echo "enter the filename"
                 read file
                 echo "1. to count number of lines in a file that do not contain vowels"
                 echo "2. to  count number of characters words lines in a given file"
```

```
          echo "enter your choice"
          read ch
          case $ch in
                1)echo "no: of lines that do not contain vowels"
                      grep -v '[aeiou]\+' $file ;;
                2)echo "to no: of characters"
                      wc -c $file
                      echo "no: of lines"
                      wc -l $file
                      echo "no: of words"
                      wc -w $file ;;
                *)exit;;
          esac
    done
```

8. **Write a shell script to list all the files in a given directory**

```
echo "enter directory name"
read dir
if [ -d $dir ]
then
    echo "list of files in the directory"
    ls -l $dir | egrep '^-'
else
    echo "enter proper directory name"
fi
```

9. Write a shell script to display list of users currently logged in.

```
Echo “Yourname:$(echo $USER)”
Echo “ current.date &time:$(date)”
Echo”currently logged on users $(who)
```

10. **Write a shell script to read three text files in the current directory and merge them into a**

single file and returns a file descriptor for the new file.

```
echo”enter first file names”
read f1
echo”enter second file names”
read f2
echo”enter third file names”
read f3
cat $f1 $f2 $f3 > file1.txt
ls – l file1.txt
cat file1.txt
```

PART B

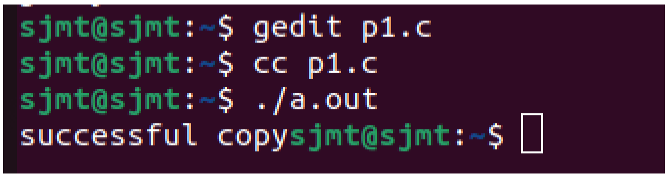
1. **Write a program to copy a file into another using system calls.**

```
#include<stdio.h>
#include<unistd.h>
#include<fcntl.h>
void main()
{
    char buf;
    int fd_one, fd_two;
    fd_one = open("p1.txt", O_RDONLY);
    if(fd_one==-1)
        {

            printf("error opening first_file\n");
            close(fd_one);
            return;
        }
fd_two=open("p4.txt",O_WRONLY|O_CREAT,S_IRUSR|S_IWUSR|S_IRGRP|S_IROTH);
        while(read(fd_one,&buf,1))
        {
                write(fd_two,&buf,1);
        }

printf("successful copy");
close(fd_one);
close(fd_two);
}
```

OUTPUT

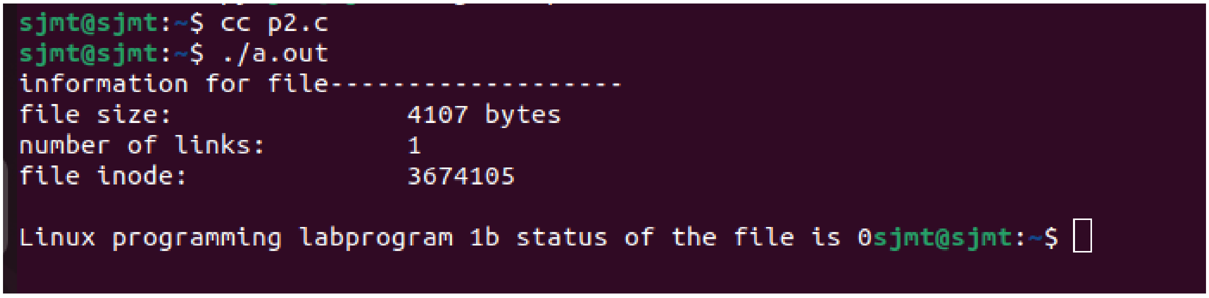


```
sjmt@sjmt:~$ gedit p1.c
sjmt@sjmt:~$ cc p1.c
sjmt@sjmt:~$ ./a.out
successful copysjmt@sjmt:~$
```

2. **Write a program using system call: create, open, write, close, stat, fstat, lseek.**

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<unistd.h>
#define BUFSIZE 512
char buf1[]="Linux programming lab";
char buf2[]="program 1b";
char buf[ BUFSIZE ];
struct stat buf3;
int main ()
{
        int fd,status;
        if ((fd=creat ("t1.txt",0666))<0)
        {
                perror ("creation error");
                exit(1);
        }
        if(write(fd,buf1,sizeof(buf1))<0)
        {
            perror ("writing error");
            exit(2);
        }
        if (lseek(fd,4096,SEEK_SET)<0)
        {
            perror ("positioning error");
            exit(3);
        }
        if (write(fd,buf2,sizeof(buf2))<0)
        {
            perror("writing error");
            exit(2);
        }
        if(stat("t1.txt",&buf3)<0)
            return 1;
        printf ("information for file");
        printf ("------------------\n");
        printf ("file size:\t\t %ld bytes \n",buf3.st_size);
        printf ("number of links:\t %ld \n",buf3.st_nlink);
        printf ("file inode:\t\t %ld \n",buf3.st_ino);
        fd = open ("t1.txt",O_RDONLY);
        status=fstat(fd,&buf3);
        printf ("\n status of the file is %d",status);
        if (fd==-1)
        {
            printf ("error opening file \n");
            exit (1);
        }
        else
            while (read(fd,&buf,1))
            {
                write(1,&buf,1);
            }
            close(fd);
    }
```

OUTPUT



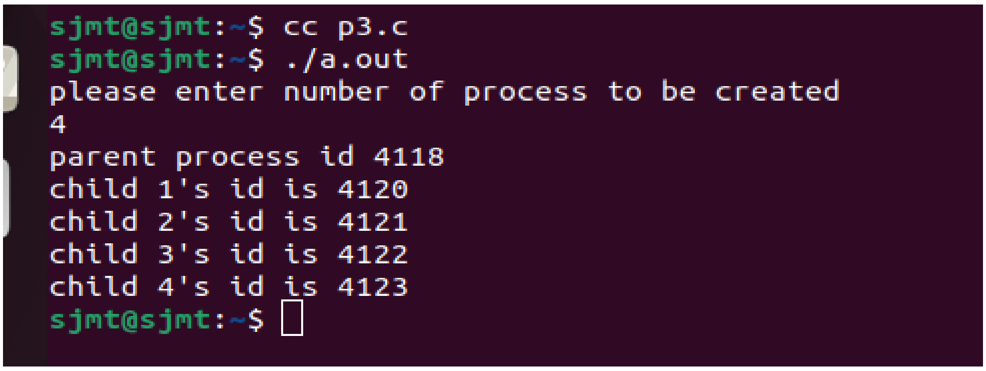
```
sjmt@sjmt:~$ cc p2.c
sjmt@sjmt:~$ ./a.out
information for file------------------
file size:                4107 bytes
number of links:          1
file inode:               3674105

Linux programming labprogram 1b status of the file is 0sjmt@sjmt:~$ 
```

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

```
#include<stdio.h>
#include<unistd.h>
#include<sys/time.h>
#include<sys/resource.h>
#include<stdio.h>
void main()
{
        int num;
        int procid[100]={0};
        int i;
        printf ("please enter number of process to be created \n");
        scanf("%d",&num);
        for(i=0;i<num;i++)
        {
                procid [i]=fork();
                switch (procid[i])
                {
                        case 0:while(1);
                        default:break;
                }
        }
        printf ("parent process id %u \n",getpid());
        for (i=0;i<num;i++)
        {
                printf ("child %d's id is %u \n",i+1,procid[i]);
        }
}
```

OUTPUT



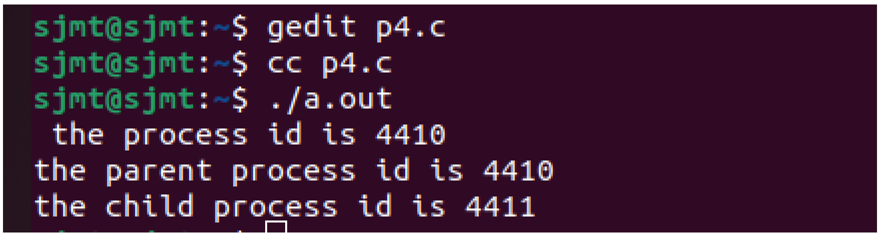
```
sjmt@sjmt:~$ cc p3.c
sjmt@sjmt:~$ ./a.out
please enter number of process to be created
4
parent process id 4118
child 1's id is 4120
child 2's id is 4121
child 3's id is 4122
child 4's id is 4123
sjmt@sjmt:~$ 
```

4. **Write a program to create a Zombie process**

```
#include<stdio.h>
#include <unistd.h>
int main()
```

```
{
pid_t ret_val;
printf(" the process id is %d\n",getpid());
ret_val = fork();
if (ret_val ==0)
{
printf("the child process id is %d\n",getpid());
sleep(20);
}
else
{
printf("the parent process id is %d\n",getpid());
sleep(30);
}
return 0;
}
```

OUTPUT



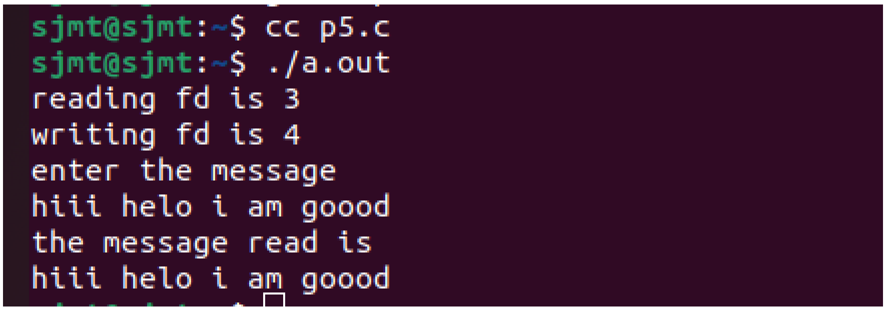
```
sjmt@sjmt:~$ gedit p4.c
sjmt@sjmt:~$ cc p4.c
sjmt@sjmt:~$ ./a.out
 the process id is 4410
the parent process id is 4410
the child process id is 4411
```

5. Write a program to implement inter process communication using pipes

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/time.h>
#include<sys/resource.h>
#include<sys/types.h>
#include<sys/wait.h>
int main()
{
    char msg[80],buf[80];
    int p[2],pid,i,byread;
    pipe(p);
    printf ("reading fd is %d \n",p[0]);
    printf ("writing fd is %d \n",p[1]);
    pid=fork();
    if(pid>0)
    {
        close (p[0]);
```

```
		printf ("enter the message \n");
		byread=read(0,msg,80);
		write (p[1],msg,byread);
		wait(NULL);
		exit(0);
	}
	else
	{
		close(p [1]);
		read(p[0],buf,80);
		printf("the message read is\n");
		printf("%s",buf);
		exit(0);
	}
}
```

OUTPUT



```
sjmt@sjmt:~$ cc p5.c
sjmt@sjmt:~$ ./a.out
reading fd is 3
writing fd is 4
enter the message
hiii helo i am goood
the message read is
hiii helo i am goood
```

6. Simulate the following CPU scheduling algorithms

a. Round Robin

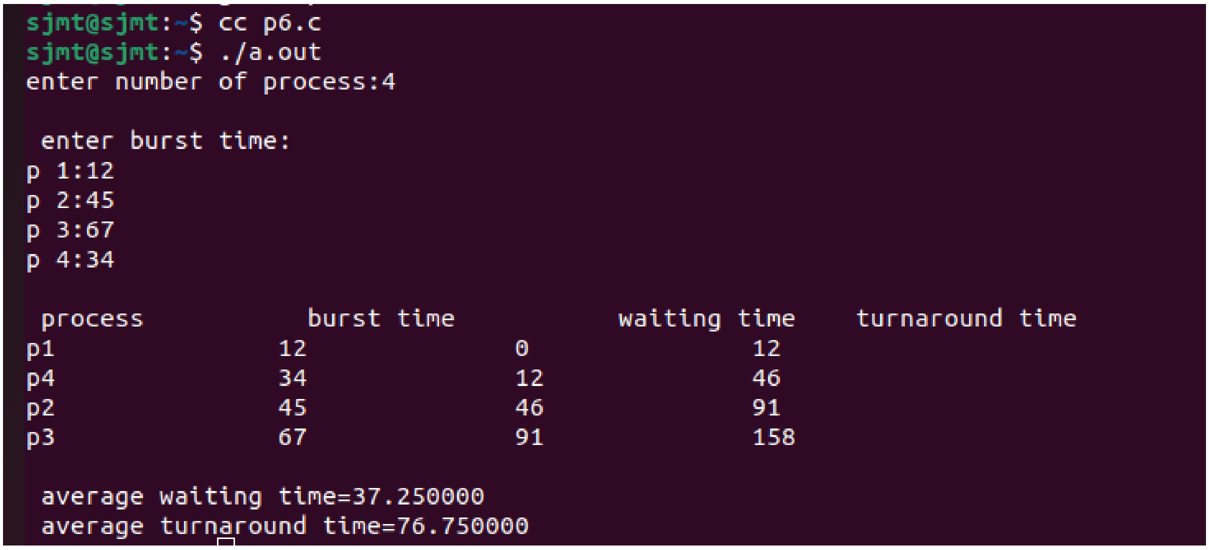
b. SJF

```
#include<stdio.h>
int main()
{
	int bt[20],p[20],wt[20],tat[20],i,j,n,tot_wt=0,tot_tat=0,pos,temp;
	float avg_wt,avg_tat;
	printf("enter number of process:");
	scanf("%d",&n);
	printf("\n enter burst time:\n");
	for(i=0;i<n;i++)
	{
		printf("p %d:",i+1);
		scanf("%d",&bt [i]);
		p[i]=i+1;   }
	for(i=0;i<n;i++)
	{
		for(j=i+1;j<n;j++)
```

```
          {
          if(bt[i]>bt[j])
          {
               temp=bt[i];
               bt[i]=bt[j];
               bt[j]=temp;

          temp=p[i];
          p[i]=p[j];
          p[j]=temp;
          }}}
     wt[i]=0;
     for(i=1;i<n;i++)
     {    wt[i]=0;
          for(j=0;j<i;j++)
               wt[i]+=bt[j];
          tot_wt+=wt[i];   }
     avg_wt=(float)tot_wt/n;
     printf("\n process\t   burst time    \twaiting time\tturnaround time");
     for(i=0;i<n;i++)
     {  tat[i]=wt[i]+bt[i];
          tot_tat+=tat[i];
          printf("\np%d\t\t %d \t\t %d \t\t %d",p[i],bt[i],wt[i],tat[i]);}
          avg_tat=(float)tot_tat/n;
          printf("\n\n average waiting time=%f",avg_wt);
          printf("\n average turnaround time=%f\n",avg_tat);
}
```

OUTPUT:



```
sjmt@sjmt:~$ cc p6.c
sjmt@sjmt:~$ ./a.out
enter number of process:4

 enter burst time:
p 1:12
p 2:45
p 3:67
p 4:34

 process           burst time           waiting time    turnaround time
p1                12               0                12
p4                34               12               46
p2                45               46               91
p3                67               91               158

 average waiting time=37.250000
 average turnaround time=76.750000
```

6.b SJF

```
#include<stdio.h>
int main()
{
```

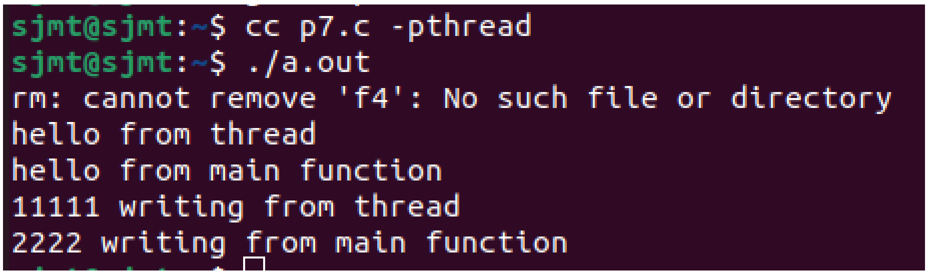
```
int i,n,total=0,x,counter=0,qt;
int tot_wt=0,tat=0,at[10],bt[10],burst_time[10];
float avg_wt,avg_tat;
printf("\n enter total number of processors\t");
scanf("%d",&n);
x=n;
for(i=0;i<n;i++)
{
    printf("\n enter details of process[%d]\n",i+1);
    printf("arrival time:\t");
    scanf("%d",&at[i]);
    printf("burst time:\t");
    scanf("%d",&burst_time[i]);
    bt[i]=burst_time[i];
}
printf("\n enter time quantum:\t");
scanf("%d",&qt);
printf("\n process ID \t\tburst time\t\t turnaround time \t\t waiting time\n");
for(total=0,i=0;x!=0;)
{
    if(bt[i]<=qt && bt[i]>0)
    {
        total=total+bt[i];
        bt[i]=0;
        counter=1;
    }
    else if(bt[i]>0)
    {
        bt[i]=bt[i]-qt;
        total=total+qt;
    }
    if(bt[i]==0 && counter==1)
    {
        x--;
        printf("\nprocess[%d]\t\t%d\t\t%d\t\t%d",i+1,burst_time[i],total-at[i],total-at[i]-burst_time[i]);
        tot_wt=tot_wt+total-at[i]-burst_time[i];
        tat=tat+total-at[i];
        counter=0;
    }
    if(i==n-1)
    {
        i=0;
    }
    else if(at[i+1]<=total)
```

```
        {
            i++;
        }
        else
        {
            i=0;
        }
    }
    avg_wt=tot_wt /n;
    avg_tat=tat /n;
    printf("\n\naverage waiting time:%f",avg_wt);
    printf("\navg turnaround time:%f\n",avg_tat);
    return 0;
}
```

7. Write a program that illustrates file locking using semaphores

```
#include<pthread.h>
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<string.h>
#include<semaphore.h>
sem_t semaphore;
FILE *fd;
void* routine(void* args);
int main(int argc,char*argv[])
{
    pthread_t threadID;
    sem_init(&semaphore,0,1);
    system("rm f4");
    fd=fopen("f4","a");
    pthread_create(&threadID,NULL,&routine,NULL);
    sleep(1);
    sem_wait(&semaphore);
    printf("hello from main function\n");
    fprintf(fd,"2222 writing from main function\n");
    sem_post(&semaphore);
    pthread_join(threadID,NULL);
    sem_destroy(&semaphore);
    fclose(fd);
    system("cat f4");
    return 0;
}
void* routine(void* args)
{
    char input;
    sem_wait(&semaphore);
    printf("hello from thread\n");
    fprintf(fd,"11111 writing from thread\n");
    sem_post(&semaphore);
}
```

OUTPUT:



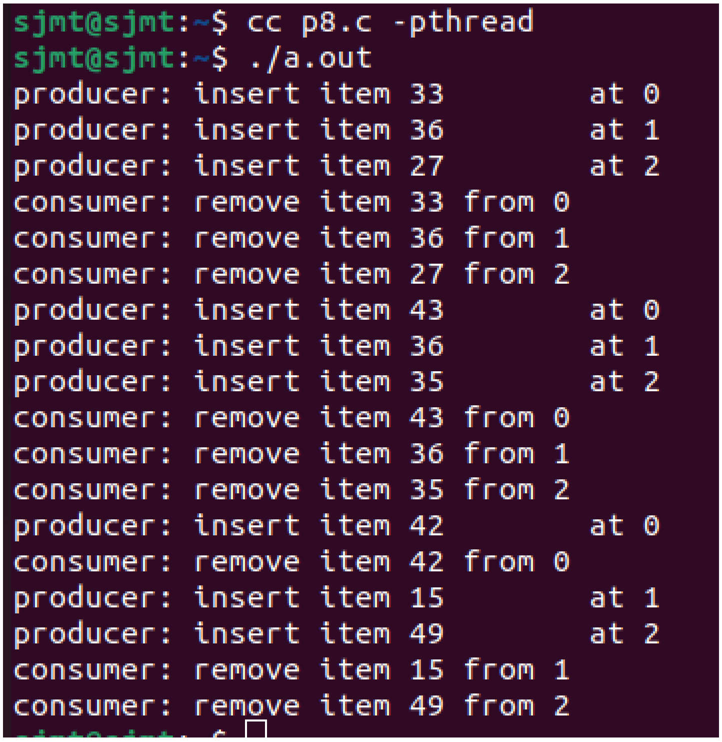
```
sjmt@sjmt:~$ cc p7.c -pthread
sjmt@sjmt:~$ ./a.out
rm: cannot remove 'f4': No such file or directory
hello from thread
hello from main function
11111 writing from thread
2222 writing from main function
```

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

```
#include<stdlib.h>
#include<unistd.h>
#include<string.h>
#include<semaphore.h>
#include<stdio.h> #include<pthread.h>
sem_t empty; sem_t full; int in=0; int out=0;
int buffer[3]; pthread_mutex_t mutex;
void *producer(void *pno)
{
int item; for(int i=0;i<3;i++)
{
item=rand()%50; sem_wait(&empty); pthread_mutex_lock(&mutex);
buffer[in]=item;
printf("producer: insert item %d        at %d\n",buffer[in],in); in=(in+1)%3;
pthread_mutex_unlock(&mutex);
sem_post(&full);
}
}
void *consumer(void *cno)
{ for(int i=0;i<3;i++)
{
sem_wait(&full); pthread_mutex_lock(&mutex); int item=buffer[out];
printf("consumer: remove item %d from %d\n",item,out);
out=(out+1)%3; pthread_mutex_unlock(&mutex);
sem_post(&empty);
}
}
```

```
int main()
{
pthread_t pro[3],con[3] pthread_mutex_init(&mutex,0);sem_init(&empty,0,3);
sem_init(&full,0,0); int a[3]={1,2,3};
for(int i=0;i<3;i++)
{
pthread_create(&pro[i],NULL,&producer,NULL);
}
for(int i=0;i<3;i++)
{
pthread_create(&con[i],NULL,&consumer,NULL);
}
for(int i=0;i<3;i++)
{
pthread_join(con[i],NULL);
}
pthread_mutex_destroy(&mutex); sem_destroy(&empty); sem_destroy(&full);
return 0;
}
```

OUTPUT:



```
sjmt@sjmt:~$ cc p8.c -pthread
sjmt@sjmt:~$ ./a.out
producer: insert item 33        at 0
producer: insert item 36        at 1
producer: insert item 27        at 2
consumer: remove item 33 from 0
consumer: remove item 36 from 1
consumer: remove item 27 from 2
producer: insert item 43        at 0
producer: insert item 36        at 1
producer: insert item 35        at 2
consumer: remove item 43 from 0
consumer: remove item 36 from 1
consumer: remove item 35 from 2
producer: insert item 42        at 0
consumer: remove item 42 from 0
producer: insert item 15        at 1
producer: insert item 49        at 2
consumer: remove item 15 from 1
consumer: remove item 49 from 2
```

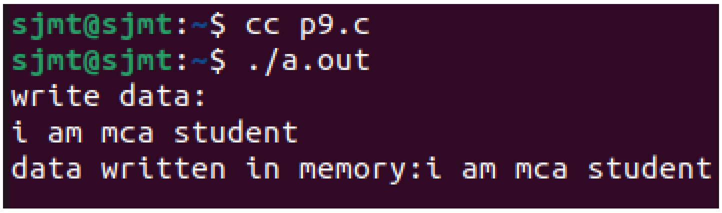
9. **Write a program that illustrates inter process communication using shared memory system calls.**

   First part of the program

```
#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: \n");
        fgets(str,100,stdin);
        printf("data written in memory:%s\n",str);
        shmdt(str);
        return 0;
}
```

<u>OUTPUT:</u>

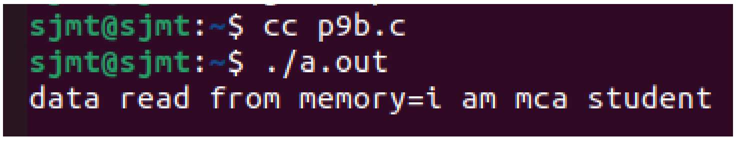


```
sjmt@sjmt:~$ cc p9.c
sjmt@sjmt:~$ ./a.out
write data:
i am mca student
data written in memory:i am mca student
```

9.Second part of the progrm

```
#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;
}
```

<u>OUTPUT:</u>



```
sjmt@sjmt:~$ cc p9b.c
sjmt@sjmt:~$ ./a.out
data read from memory=i am mca student
```

10. Write a program that illustrates the following:

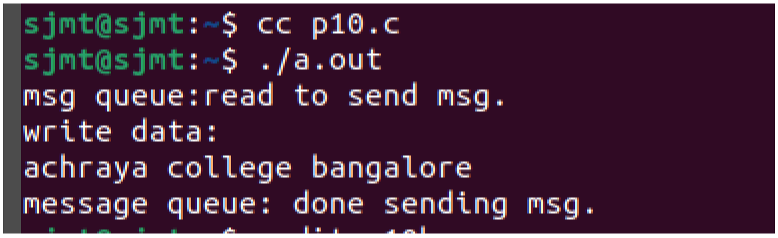
a. Creating message queue

b. Writing to a message queue

c. Reading from a message queue

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<errno.h>
#include<string.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#include<stdio.h>
char mtext[2000];
int main(void)
{
        int msgid;
        int len;
        key_t key=ftok("shmfile",65);
        msgid=msgget(key,0644|IPC_CREAT);
        printf("msg queue:read to send msg. \n");
        printf("write data: \n");
        fgets(mtext,100,stdin);
        len=strlen(mtext);
        msgsnd(msgid,&mtext,len,0);
        printf("message queue: done sending msg. \n");
        return 0;
}
```

OUTPUT:

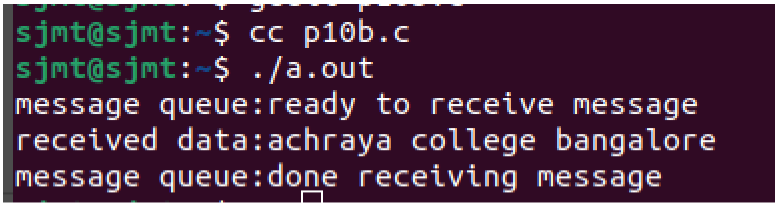


```
sjmt@sjmt:~$ cc p10.c
sjmt@sjmt:~$ ./a.out
msg queue:read to send msg.
write data:
achraya college bangalore
message queue: done sending msg.
```

10.b, c

```
#include<stdio.h>
#include<stdlib.h>
#include<errno.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#include<string.h>
char mtext[2000];
int main(void)
{
        int msgid;
        key_t key=ftok("shmfile",65);        // for IPC
        msgid=msgget(key,0644);  //get msg queue
        printf("message queue:ready to receive message\n");
        msgrcv(msgid,&mtext,sizeof(mtext),0,0);  //msg receive operation, readsthe msg
        from queue
        printf("received data:%s",mtext);
        printf("message queue:done receiving message\n");
        return 0;
}
```

OUTPUT:



```
sjmt@sjmt:~$ cc p10b.c
sjmt@sjmt:~$ ./a.out
message queue:ready to receive message
received data:achraya college bangalore
message queue:done receiving message
```