OPERATING SYSTEM

LAB REPORT

COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Department of Computer Science & Engineering

PREPARED BY:
KRISHNA KANT SINGH
ROLL:17
CSE(S6)

SUBMITTED TO
MISS JOSHNA JOSE
CSE department

Exp. No. 1

12 12 19

FAMILARIZATION OF SYSTEM CALL

SYSTEM CALL :

In computing a system call is the programmatic way in which a computer program requests a service from the kernel of the operating system it is Executed on. A System call is a way for programs to interact with the operating system A computer program makes a program system call when it makes a request to the operating system Kernals.

System Call provide services of the operating system to the user program via application program Interface (API). It provide an interface between a process and operating system to allow user level process to request services of the operating system

Type of system call:

There are mainly five types of system calls

* Process Control
* File manipulation
* Device Managment
* Information maintainance
* communication

Exp. No.

Working of System calls:

- first, the user application program sets up the argument for the system call
- After the arguments are well setup, the program executes the "system call" instruction
- This system causes an exception an event the causes the processor to jump to a new address and start executing the code there
- The instruction at the new address save your user program state, figure out the system call you want.

Types:

1) Process control:

Those system calls deals with processes such as process creation, process termination etc.

Types of Process control:

(1) Fork():

Fork system calls is used for creating a new process, which is called child process which runs concurrently with the process that makes the fork() calls. After a new

child process is created, both processes will execute the next instruction following the fork() system calls. It takes no parameter and returns an integer value.

The different values returned by fork()
Negative value, zero, positive value.

* Syntax:-
```
int main()
{
fork();
printf ("Hello world! /n");
return 0;
}
```

* Header file:
```
# include <stdio.h>
# include <sys/types.h>
# include <unistd.h>
```

Ex: No of times hello is printed
```
# include <stdio.h>
# include <sys/types.h>
int main()
{
Fork();
Fork();
Fork();
Printf ("hello", n);
return 0;
}
```

Exp. No.

(ii) Exit(): The exit() function causes normal process termination and the value of status is returned to the parent.

Syntax: #include <stdlib.h>
void Exit (int status)

Header file: #include <unistd.h>

Ex:
```
# include <stdio.h>
int main
{
Printf(" Hey!! welcome to establish world\n");
exit( Exit_Sucess);
```

(iii) wait(): A call to wait() blocks the calling process untill one of its child processes exits or a signal is received. After child process terminates, parent continues its Execution after wait system call instruction.

Syntax:
```
#include <stdio.h>
# include <sys/wait.h>
# include <unistd.h>
int main()
{
pid_t wait (int * stat_loc);
}
```

Exp. No.

Header: #include <sys/wait.h>

Ex:
```
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <unistd.h>
int main()
{
Pid_t cpid;
if (fork()== 0)
 exit(0);
else
cpid= wait (NULL)
 printf("Parent pid= %d /n", get pid());
 printf ("child pid= %d /n", cpid);
return 0;
}
```

2. FILE Manipulation: These system calls are responsible for file manipulation such as creating a file, reading a file, writing into a file etc.

Types of file manipulation:

i) open () :-

Exp. No.

Openxt → start a program on a new virtual terminals

Syntax: open xt [-c xt number] [-s] [-u] [-l] [-v] [--]
command command - options

Header file:- # include <sys/stat.h>
include <fcntl.h>

(4.1) Read(): Read is the system call used to read data into a buffer.

Syntax: size_t read (int fd, void *buf, size_t c

Header: #include <fcntl.h>

Ex:
```
#include <stdio.h>
# include <fcntl.h>
int main()
{
char c;
int fd1 = open ("foobar.txt", O_RDONLY,0);
int fd2 = open ("foobar.txt", O_RDONLY,0);
Read (fd1, &c,1);
Read (&fd2, &c,1);
printf ("c→ %c \n", c);
exit(0);
```

3) Write() :- Writes int bytes from buf to the file or socket associated with fd, int should not be greater than INT. MAX. If int is zero write() simply return 0 without attempting any other action

Syntax:

```
#include <fcntl.h>
Size_t write (int fd, void* buf, size_t int);
```

Header : #include <fcntl.h>

ex:

```
#include <stdio.h>
#include <fcntl.h>
int main (void)
{
int fd [2];
char buff1[12] = "hello world";
char buff2 [12];
Fd [0] = Open ("foobar.txt", O_RDWR);
fd [1] = Open ("foobar.txt", O-RDWR);
write (fd [0] - buf1, strlen (buf1));
write (1, buf2, read (fd [1], buf 2, 12))
close (fd[0]);
close (fd[1]);
return 0;
}
```

4) close :

Tells the operating system you are done with a file descriptor and close the file which fontod by fd.

Syntax: #include <fcntl.h>
int close (int fd);

Header: #include <fcntl.h>

ex:

```
#include <stdio.h>
#include <fcntl.h>
int main ()
{
int fd1 = open ("foo.text", O_RDONLY, 0);
close (fd1);
int fd2 = open ("baz.txt, O_RDONLY, 0);
printf ("fd2 = %d/n", fd2);
exit (0);
}
```

3) Device Managment :

These system calls are responsible for file manipulation such as reading from device buffers, writing into device buffer etc.

Types of Device Managment sc :

1) ioctl () :- The ioctl function is useful for implementing a device driver to set the configuration on the device.

Syntax: int ioctl (int fd, int request...

header: # include <sys/ioctl.h>

* Read (): Read is the system call used to read data into buffer

Syntax: Size-t read (int fd, void *buf, size_t

header: # include <fcntl.h>

* Write (): write int bytes from buf to the file or socket associated with fd, int should not be greater than INT_MAX. If int is zero. write() simply return() without attempting any other active.

Syntax: size_t write (int fd, void *buf, size_t int

Header : # include <fcntl.h>

4) Information Maintanance :- These system calls handle information and its transfer between the operating system and the

prog.

Types of Information Maintanance:

* getpid()
* alarm()
* sleep()

1. getpid() → get process identification

Syntax: #include <sys/types.h>
#include <unistd.h>
Pid_t getpid (void);
Pid_t getpid (void);

Header:
#include <sys/types.h>
#include <unistd.h>

2. Alarm() :- Set an alarm clock for delivery of a signal.

Syntax: #include< std.h>
unsigned int alarm (unsigned, int seconds)

Header: #include <unistd.h>

Return value: alarm() returns the number of seconds remaining until any previously scheduled alarm was due to be delivered or zero if there was no previously scheduled alarm.

ex: setting a alarm

```
# include <stdio.h>
# include <unistd.h>
void alarm_handler (int sig num)
{
printf ("BUZZ BUZZ \n");
}
int main()
{ alarm;
  Pause();
}
```

3. sleep(): sleep for the specified number of seconds.

Syntax: # include <unistd.h>
unsigned int sleep / unsigned int seconds)

header file: # include <unistd.h>

Return value: Zero if the requested time has elapse or the no of sound left to slep if the call was interrupted by a signal handler.

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

```
{ printf ("wait a second \n");
  sleep();
  printf ("OK");
}
```

5) Communication:

These system calls are useful for interprocess communication. They also deals with creating and deleting a communication connection.

Types of communication system calls:

1) Pipe(): A pipe is a connection b/w two process such that the standard output from one process becomes the standard input of other process. In Unix, pipes are useful for communication b/w related process.

Syntax: int pipe (int fds [2]);

header file: #include <unistd.h>

Return value: 0 on success, -1 on error

Ex:
```
#include <stdio.h>
# include <unistd.h>
char* msg = "hello,world #1");
int main()
```

```
{ char inbuff [MSG SIZE];
  int p[2], i;
if pipe (p<0);
exit (1);
write ( p[1], msg1, MSG SIZE);
for (i=0; i<1; i++)
{ read (p[0], inbut, MSG SIZE);
 printf ("%s", inbuf);
}
return 0; }
```

2) Shmget(): allocates a shared memory segment. It returns the identifier of the schared memory segment associated with the value of the argument key.

Syntax:
```
# include <sys/ ipc.h>
# include <sys/ shm.h>
int shmget (key_t key, size_t size, int shmget
```

Headerfile:
```
# include < sys/ ipc.h>
# include < sys/ shm.h>
int shmget ( key_t key, size_t, size int shmf
```

Exp. No.

Return value: Upon successful completion, shmget() shall return a non negative integer, namely a shared memory identifier, otherwise it return. -1 and set error 0 to indicate the error.

Ex: Shared memory for Reader process

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

Result: The program familiarization of system calls are verified.

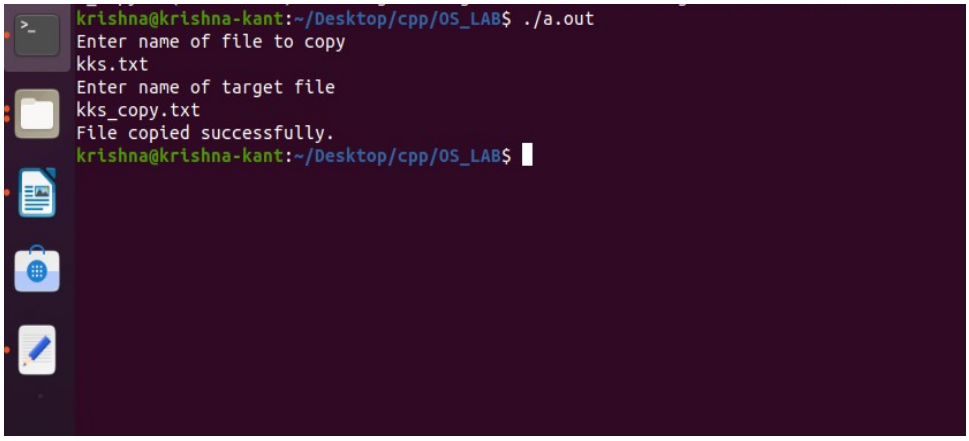
EXPERIMENT :2

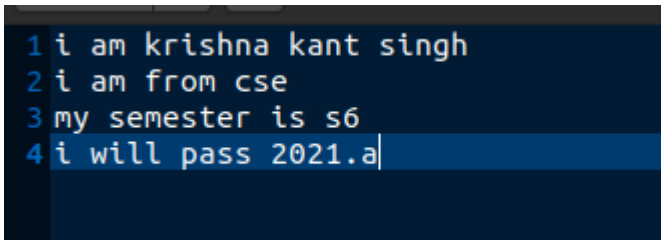
AIM: write a c program to copy content of a file.

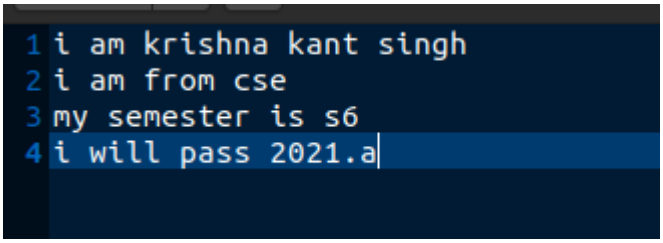
PROGRAM:

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

int main()
{
   char ch, source_file[20], target_file[20];
   FILE *source, *target;

   printf("Enter name of file to copy\n");
   gets(source_file);

   source = fopen(source_file, "r");

   if (source == NULL)
   {
      printf("Press any key to exit...\n");
      exit(EXIT_FAILURE);
   }

   printf("Enter name of target file\n");
   gets(target_file);

   target = fopen(target_file, "w");

   if (target == NULL)
   {
      fclose(source);
      printf("Press any key to exit...\n");
      exit(EXIT_FAILURE);
   }
```

```
    while ((ch = fgetc(source)) != EOF)
        fputc(ch, target);

    printf("File copied successfully.\n");

    fclose(source);
    fclose(target);

    return 0;
}
```







Kks.txt kks_copy.txt

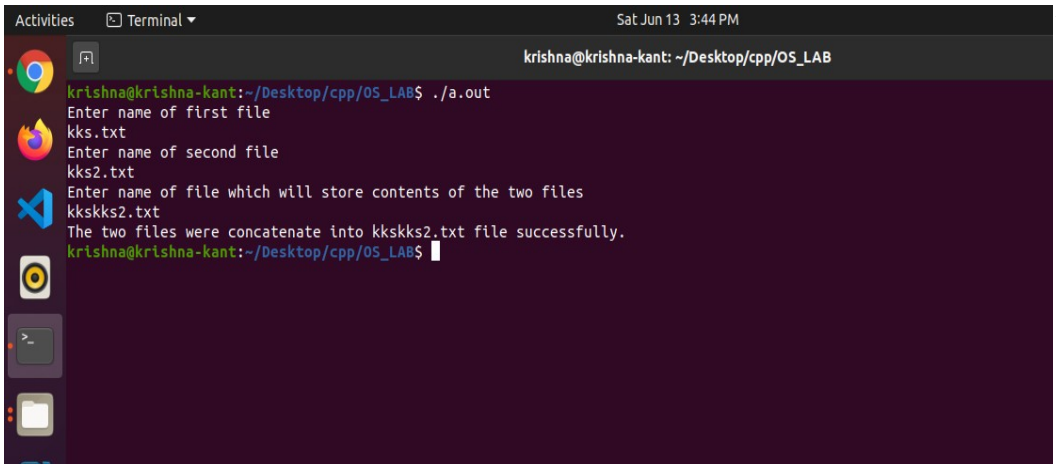
EXPERIMENT :3

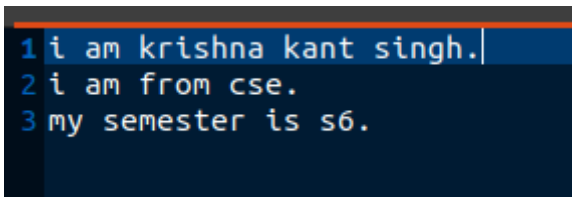
AIM: write a c program to concatenate two files.

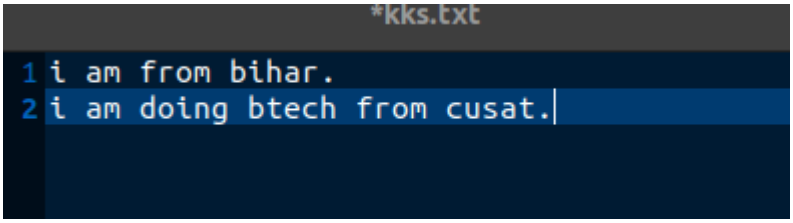
PROGRAM:

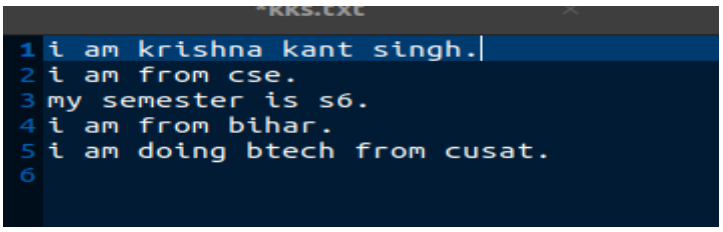
```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
  FILE *fs1, *fs2, *ft;

  char ch, file1[20], file2[20], file3[20];

  printf("Enter name of first file\n");
  gets(file1);

  printf("Enter name of second file\n");
  gets(file2);

  printf("Enter name of file which will store contents of the two files\n");
  gets(file3);

  fs1 = fopen(file1, "r");
  fs2 = fopen(file2, "r");

  if (fs1 == NULL || fs2 == NULL)
  {
    perror("Error ");
    printf("Press any key to exit...\n");
    exit(EXIT_FAILURE);
  }

  ft = fopen(file3, "w");

  if (ft == NULL)
  {
    perror("Error ");
```

```
    printf("Press any key to exit\n");
    exit(EXIT_FAILURE);
  }

  while ((ch = fgetc(fs1)) != EOF)
    fputc(ch,ft);

  while ((ch = fgetc(fs2)) != EOF)
    fputc(ch,ft);

  printf("The two files were concatenate into %s file successfully.\n",
file3);

  fclose(fs1);
  fclose(fs2);
  fclose(ft);

  return 0;
}
```







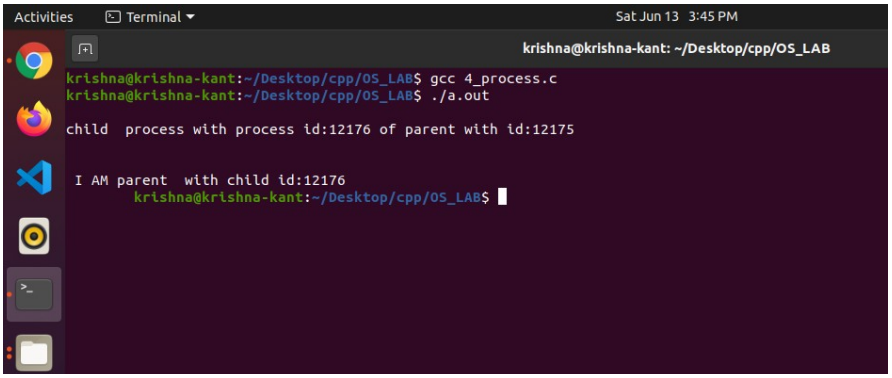


EXPERIMENT :4

AIM: write a c program to create a process.

PROGRAM:

```
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
int main()
{
int pid,cpid,ppid;
 ppid=getpid();
 pid=fork();
 cpid=getpid();
 if(!pid)
{
 printf("\n I AM parent  with child id:%d\n\t",cpid);
 sleep(1);
}
else
{
printf("\nchild  process with process id:%d of parent with id:%d\t\
n",pid,ppid);
printf("\n");
sleep(1);
}
return 0;
}
```



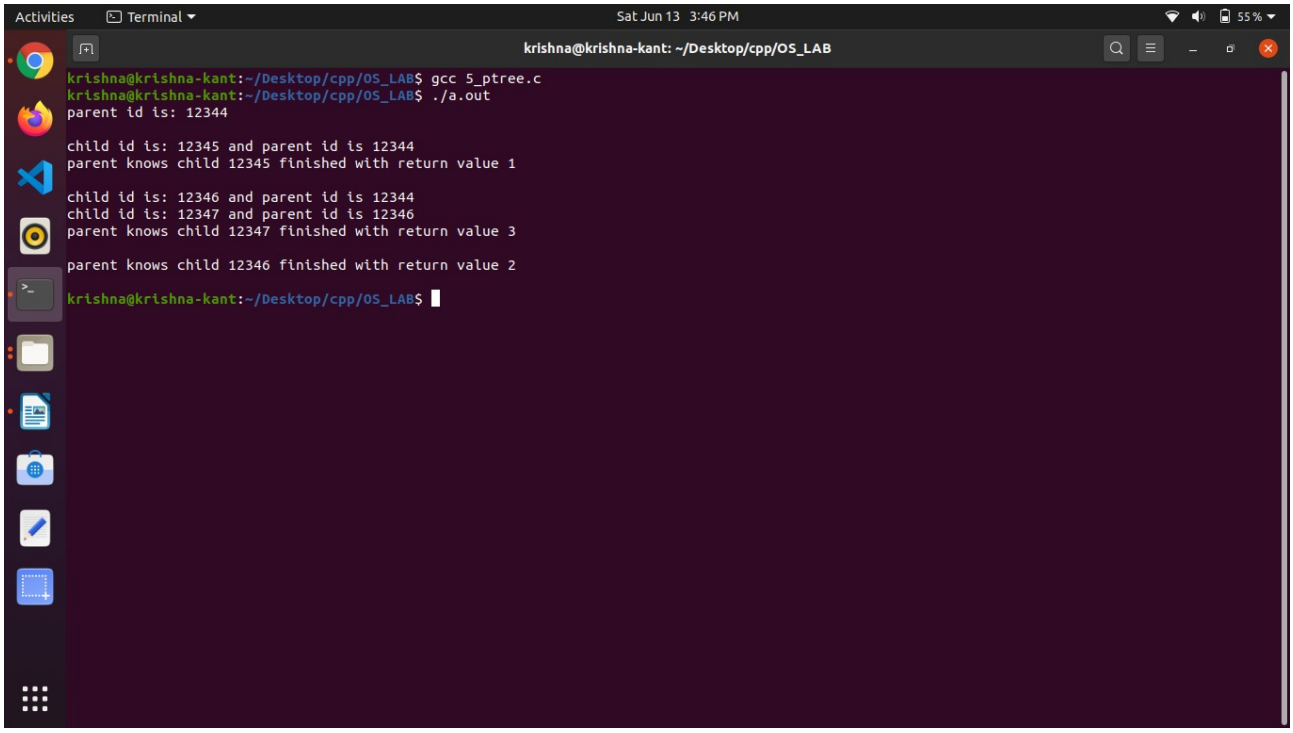
EXPERIMENT :5

AIM: write a c program to create a process tree.

PROGRAM:

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

int main() {

int status = 0;
printf("parent id is: %d\n\n", (int)getpid());
pid_t pid = fork();

if(pid == 0)
{
   printf("child id is: %d and parent id is %d\n",getpid(),getppid());
   exit(1);
}
else
{
   pid_t childPid = wait(&status);
   int childReturnValue = WEXITSTATUS(status);
   printf("parent knows child %d finished with return value %d\n\n", (int)
childPid, childReturnValue);

   pid_t pid = fork();
   if (pid == 0)
   {
      printf("child id is: %d and parent id is %d\n", getpid(), getppid());
      pid = fork();
      if (pid == 0)
      {
         printf("child id is: %d and parent id is %d\n",getpid(),getppid());
         exit(3);
      }
```

```
        else
        {
            pid_t childPid = wait(&status);
            int childReturnValue = WEXITSTATUS(status);
            printf("parent knows child %d finished with return value %d\n\n",
(int) childPid, childReturnValue);
        }
        exit(2);
    }
    else
    {
        pid_t childPid = wait(&status);
        int childReturnValue = WEXITSTATUS(status);
        printf("parent knows child %d finished with return value %d\n\n",
(int) childPid, childReturnValue);
    }
}

return 0;
}
```

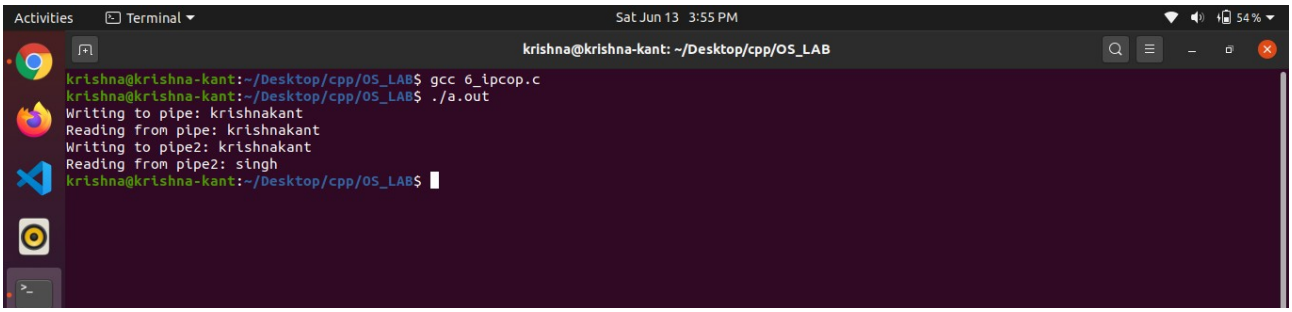


EXPERIMENT :6

AIM: write a c program to communicate using ipc one way.

PROGRAM:

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

int main() {
   int pipefds[2];
   int returnstatus;
   char writemessages[2][20]={"krishnakant", "singh"};
   char readmessage[20];
   returnstatus = pipe(pipefds);

   if (returnstatus == -1) {
      printf("Unable to create pipe\n");
      return 1;
   }

   printf("Writing to pipe: %s\n", writemessages[0]);
   write(pipefds[1], writemessages[0], sizeof(writemessages[0]));
   read(pipefds[0], readmessage, sizeof(readmessage));
   printf("Reading from pipe: %s\n", readmessage);
   printf("Writing to pipe2: %s\n", writemessages[0]);
   write(pipefds[1], writemessages[1], sizeof(writemessages[0]));
   read(pipefds[0], readmessage, sizeof(readmessage));
   printf("Reading from pipe2: %s\n", readmessage);
   return 0;
}
```



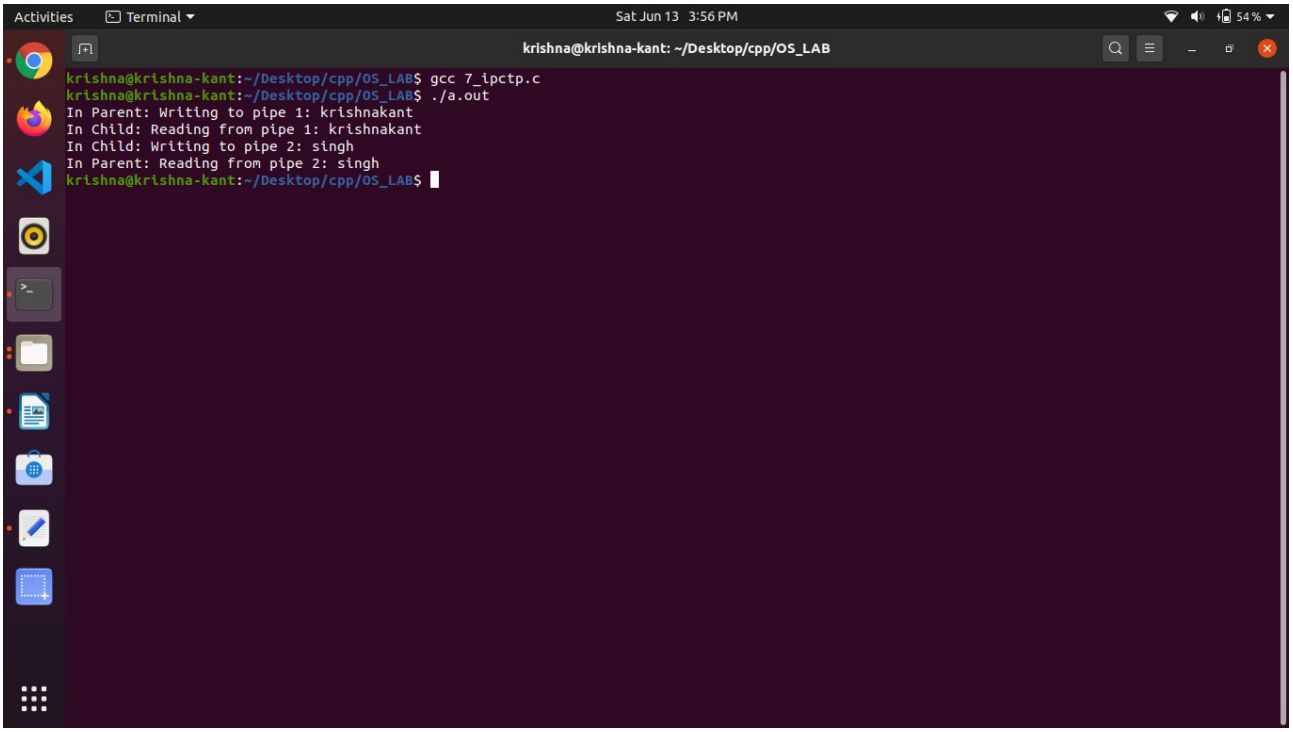
EXPERIMENT :7

AIM: write a c program to communicate using ipc two way.

PROGRAM:

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

int main() {
   int pipefds1[2], pipefds2[2];
   int returnstatus1, returnstatus2;
   int pid;
   char pipe1writemessage[20] = "krishnakant";
   char pipe2writemessage[20] = "singh";
   char readmessage[20];
   returnstatus1 = pipe(pipefds1);

   if (returnstatus1 == -1) {
      printf("Unable to create pipe 1 \n");
      return 1;
   }
   returnstatus2 = pipe(pipefds2);
   if (returnstatus2 == -1) {
      printf("Unable to create pipe 2 \n");
      return 1;
   }
   pid = fork();
   if (pid != 0)  {
      close(pipefds1[0]);
      close(pipefds2[1]);
      printf("In Parent: Writing to pipe 1: %s\n", pipe1writemessage);
      write(pipefds1[1], pipe1writemessage, sizeof(pipe1writemessage));
      read(pipefds2[0], readmessage, sizeof(readmessage));
      printf("In Parent: Reading from pipe 2: %s\n", readmessage);
   } else {
      close(pipefds1[1]);
      close(pipefds2[0]);
      read(pipefds1[0], readmessage, sizeof(readmessage));
      printf("In Child: Reading from pipe 1: %s\n", readmessage);
```

```
    printf("In Child: Writing to pipe 2: %s\n", pipe2writemessage);
    write(pipefds2[1], pipe2writemessage, sizeof(pipe2writemessage));
  }
  return 0;
}
```

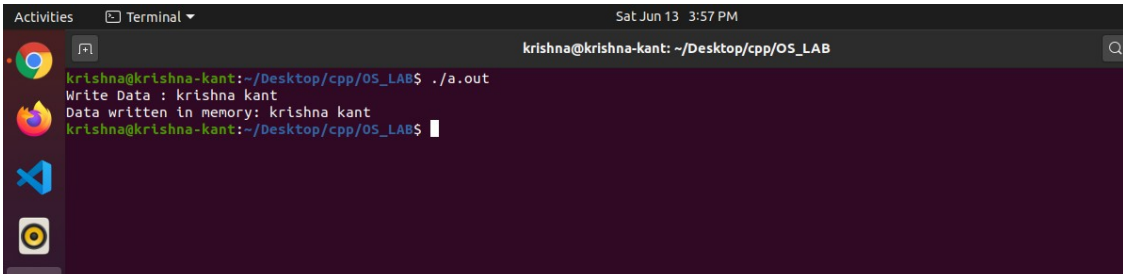


EXPERIMENT :8

AIM: write a c program to communicate using ipc shared memory.

PROGRAM:

```
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <unistd.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;
}
```



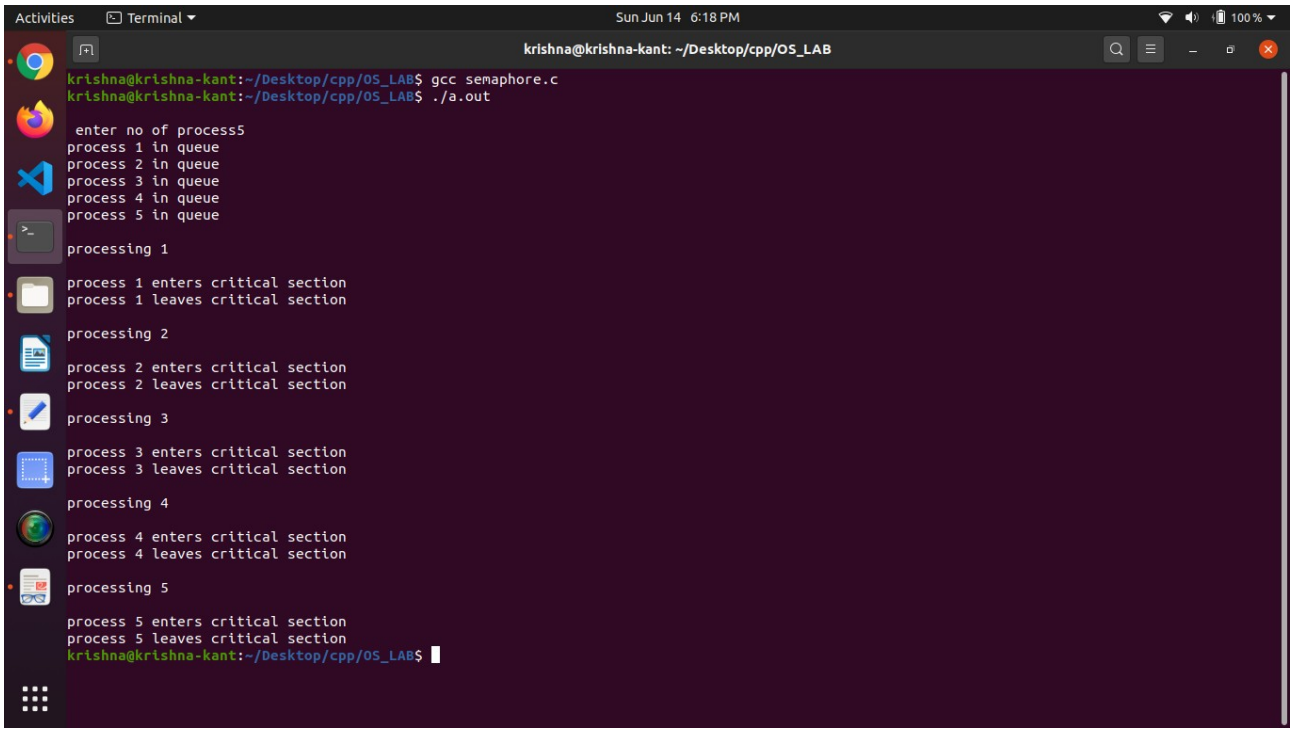
EXPERIMENT :9

AIM: write a c program to implementation of semaphore.

PROGRAM:

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>


int main()
{
   int i,a=0,h=2,n;
   printf("\n enter no of process");
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
      for(;a<n;a++)

       {
          printf("process %d in queue \n",a+1);


       }
   a++;
   if(h>1)
   {
      if(i+2<=n+1)
      {
         printf("\nprocessing %d\n",i+1);
         sleep(2);
      }
      printf("\nprocess %d enters critical section\n",i+1);
      sleep(2);
      printf("process %d leaves critical section \n",i+1);
      sleep(2);
   }

   n+1;
  }
```

}


Activities
Terminal
Sun Jun 14 6:18 PM
100 %
krishna@krishna-kant: ~/Desktop/cpp/OS_LAB
krishna@krishna-kant:~/Desktop/cpp/OS_LAB$ gcc semaphore.c
krishna@krishna-kant:~/Desktop/cpp/OS_LAB$ ./a.out
enter no of process5
process 1 in queue
process 2 in queue
process 3 in queue
process 4 in queue
process 5 in queue
processing 1
process 1 enters critical section
process 1 leaves critical section
processing 2
process 2 enters critical section
process 2 leaves critical section
processing 3
process 3 enters critical section
process 3 leaves critical section
processing 4
process 4 enters critical section
process 4 leaves critical section
processing 5
process 5 enters critical section
process 5 leaves critical section
krishna@krishna-kant:~/Desktop/cpp/OS_LAB$

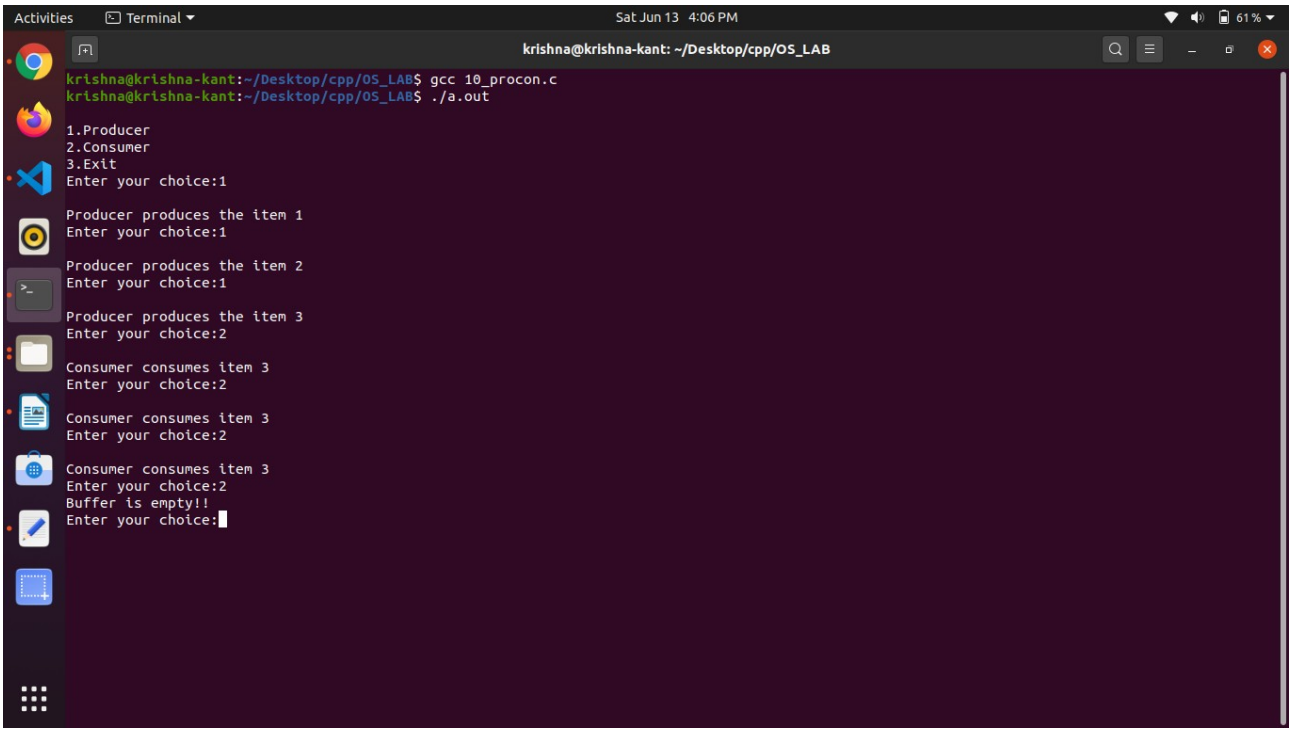
EXPERIMENT :10

AIM: write a c program to implementation of producer – consumer problem.

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int mutex=1,full=0,empty=3,x=0;
int main()
{
	int n;
	void producer();
	void consumer();
	int wait(int);
	int signal(int);
	printf("\n1.Producer\n2.Consumer\n3.Exit");
	while(1)
	{
		printf("\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;
		}
	}
```

```
        return 0;
}

int wait(int s)
{
        return (--s);
}

int signal(int s)
{
        return(++s);
}

void producer()
{
        mutex=wait(mutex);
        full=signal(full);
        empty=wait(empty);
        x++;
        printf("\nProducer produces the item %d",x);
        mutex=signal(mutex);
}

void consumer()
{
        mutex=wait(mutex);
        full=wait(full);
        empty=signal(empty);
        printf("\nConsumer consumes item %d",x);
        3-x;
        mutex=signal(mutex);
}
```


Activities
Terminal
Sat Jun 13 4:06 PM
krishna@krishna-kant: ~/Desktop/cpp/OS_LAB
krishna@krishna-kant:~/Desktop/cpp/OS_LAB$ gcc 10_procon.c
krishna@krishna-kant:~/Desktop/cpp/OS_LAB$ ./a.out
1.Producer
2.Consumer
3.Exit
Enter your choice:1
Producer produces the item 1
Enter your choice:1
Producer produces the item 2
Enter your choice:1
Producer produces the item 3
Enter your choice:2
Consumer consumes item 3
Enter your choice:2
Consumer consumes item 3
Enter your choice:2
Consumer consumes item 3
Enter your choice:2
Buffer is empty!!
Enter your choice:

EXPERIMENT :11

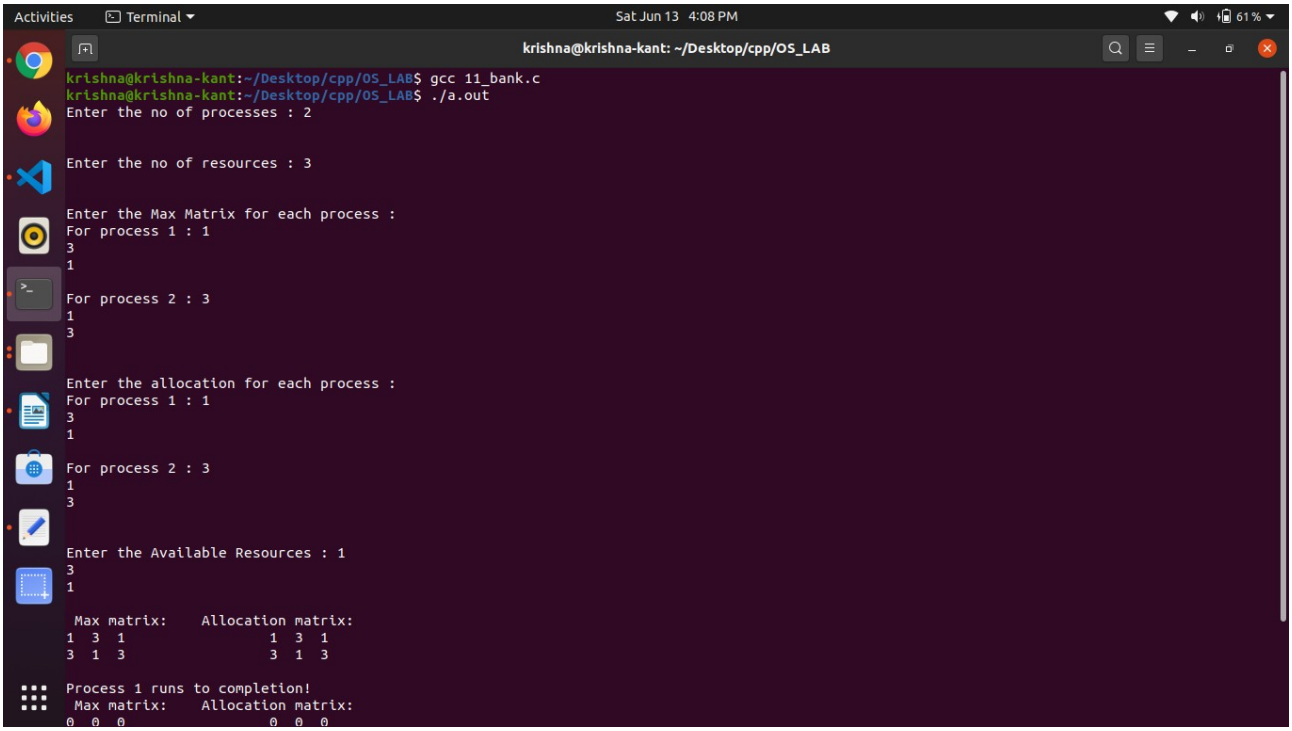
AIM: write a c program to implementation of banker’s algorithm.

PROGRAM:

```
#include<stdio.h>
#include<unistd.h>
int main()
{
int Max[10][10], need[10][10], alloc[10][10], avail[10], completed[10],
safeSequence[10];
int p, r, i, j, process, count;
count = 0;

printf("Enter the no of processes : ");
scanf("%d", &p);

for(i = 0; i< p; i++)
	completed[i] = 0;

printf("\n\nEnter the no of resources : ");
scanf("%d", &r);

printf("\n\nEnter the Max Matrix for each process : ");
for(i = 0; i < p; i++)
{
	printf("\nFor process %d : ", i + 1);
	for(j = 0; j < r; j++)
		scanf("%d", &Max[i][j]);
}

printf("\n\nEnter the allocation for each process : ");
for(i = 0; i < p; i++)
{
	printf("\nFor process %d : ",i + 1);
	for(j = 0; j < r; j++)
		scanf("%d", &alloc[i][j]);
}

printf("\n\nEnter the Available Resources : ");
for(i = 0; i < r; i++)
```

```
            scanf("%d", &avail[i]);


        for(i = 0; i < p; i++)
            for(j = 0; j < r; j++)
                need[i][j] = Max[i][j] - alloc[i][j];

do
{
        printf("\n Max matrix:\tAllocation matrix:\n");
        for(i = 0; i < p; i++)
        {
            for( j = 0; j < r; j++)
                printf("%d  ", Max[i][j]);
            printf("\t\t");
            for( j = 0; j < r; j++)
                printf("%d  ", alloc[i][j]);
            printf("\n");
        }

        process = -1;

        for(i = 0; i < p; i++)
        {
            if(completed[i] == 0)
            {
                process = i ;
                for(j = 0; j < r; j++)
                {
                    if(avail[j] < need[i][j])
                    {
                        process = -1;
                        break;
                    }
                }
            }
            if(process != -1)
                break;
        }

        if(process != -1)
```

```
        {
                printf("\nProcess %d runs to completion!", process + 1);
                safeSequence[count] = process + 1;
                count++;
                for(j = 0; j < r; j++)
                {
                        avail[j] += alloc[process][j];
                        alloc[process][j] = 0;
                        Max[process][j] = 0;
                        completed[process] = 1;
                }
        }
}while(count != p && process != -1);

if(count == p)
{
        printf("\nThe system is in a safe state!!\n");
        printf("Safe Sequence : < ");
        for( i = 0; i < p; i++)
                        printf("%d  ", safeSequence[i]);
        printf(">\n");
}
else
        printf("\nThe system is in an unsafe state!!");
return 0;}
```



```
Enter the Max Matrix for each process :
For process 1 : 1
3
1

For process 2 : 3
1
3


Enter the allocation for each process :
For process 1 : 1
3
1

For process 2 : 3
1
3


Enter the Available Resources : 1
3
1

 Max matrix:    Allocation matrix:
1  3  1                 1  3  1
3  1  3                 3  1  3

Process 1 runs to completion!
 Max matrix:    Allocation matrix:
0  0  0                 0  0  0
3  1  3                 3  1  3

Process 2 runs to completion!
The system is in a safe state!!
Safe Sequence : < 1  2  >
krishna@krishna-kant:~/Desktop/cpp/OS_LAB$
```

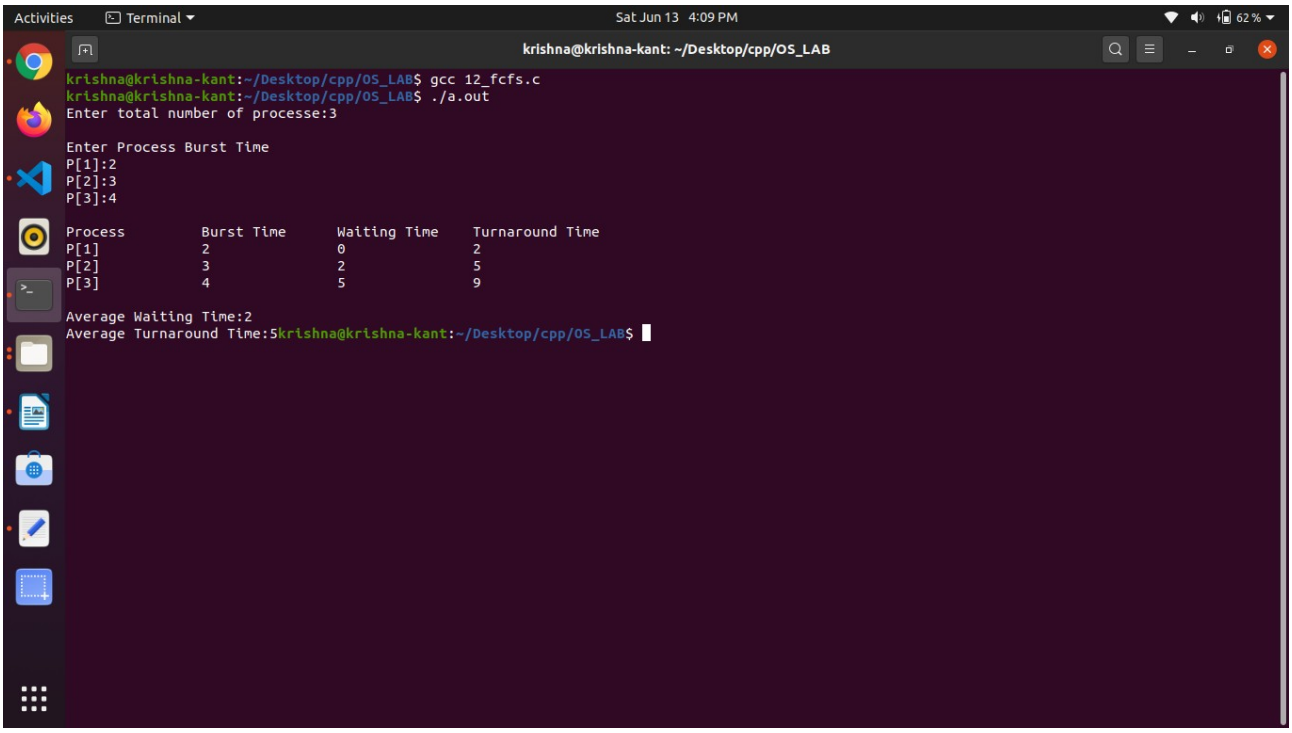
EXPERIMENT :12

AIM: write a c program to implementation of fcfs.

PROGRAM:

```
#include<stdio.h>

int main()
{
    int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;
    printf("Enter total number of processe:");
    scanf("%d",&n);

    printf("\nEnter Process Burst Time\n");
    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;
}
```



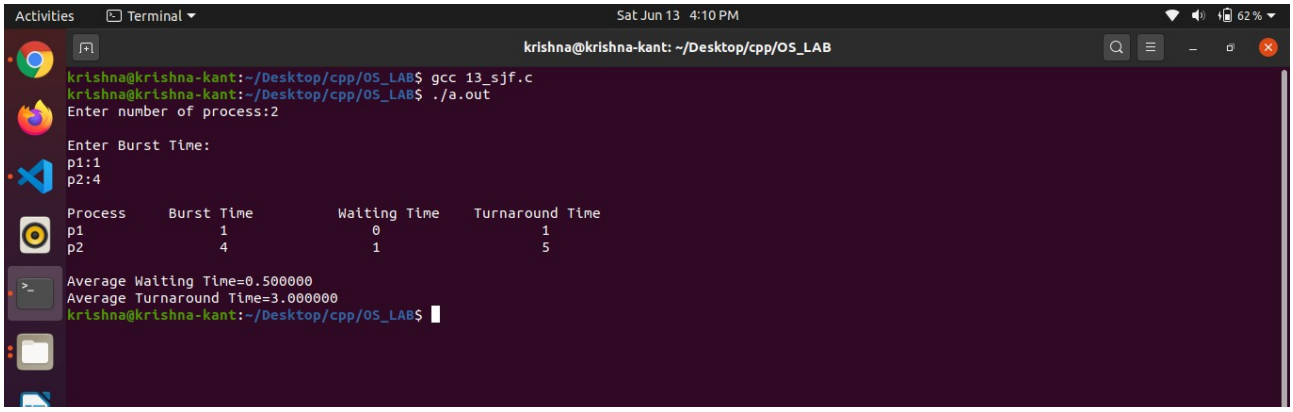
EXPERIMENT :13

AIM: write a c program to implementation of sjf.

PROGRAM:

```
#include<stdio.h>

void 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;
    }


    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;            //waiting time for first process will be 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=(float)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=(float)total/n;     //average turnaround time
    printf("\n\nAverage Waiting Time=%f",avg_wt);
    printf("\nAverage Turnaround Time=%f\n",avg_tat);
}
```



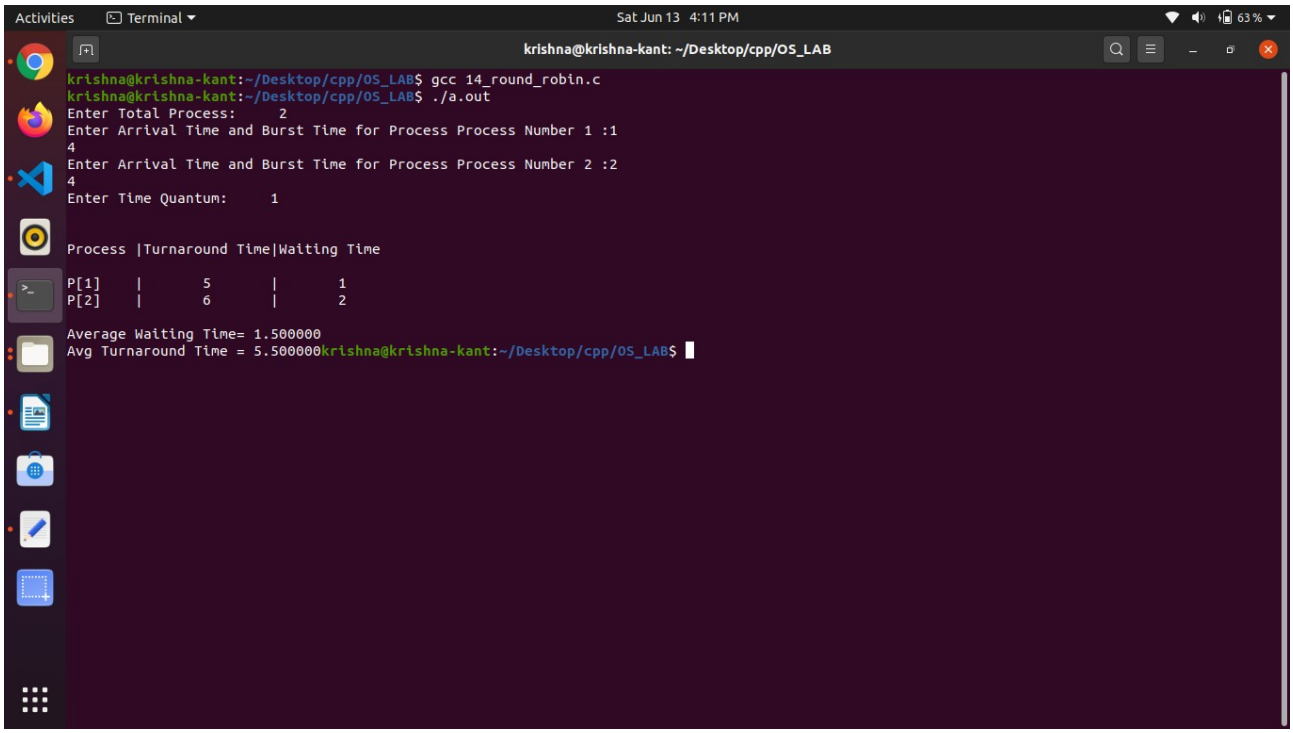
EXPERIMENT :14

AIM: write a c program to implementation of round robin.

PROGRAM:

```
#include<stdio.h>

int main()
{

  int count,j,n,time,remain,flag=0,time_quantum;
  int wait_time=0,turnaround_time=0,at[10],bt[10],rt[10];
  printf("Enter Total Process:\t ");
  scanf("%d",&n);
  remain=n;
  for(count=0;count<n;count++)
  {
    printf("Enter Arrival Time and Burst Time for Process Process Number
%d :",count+1);
    scanf("%d",&at[count]);
    scanf("%d",&bt[count]);
    rt[count]=bt[count];
  }
  printf("Enter Time Quantum:\t");
  scanf("%d",&time_quantum);
  printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");
  for(time=0,count=0;remain!=0;)
  {
    if(rt[count]<=time_quantum && rt[count]>0)
    {
      time+=rt[count];
      rt[count]=0;
      flag=1;
    }
    else if(rt[count]>0)
    {
      rt[count]-=time_quantum;
      time+=time_quantum;
    }
    if(rt[count]==0 && flag==1)
```

```
    {
      remain--;
      printf("P[%d]\t|\t%d\t|\t%d\n",count+1,time-at[count],time-at[count]-
bt[count]);
      wait_time+=time-at[count]-bt[count];
      turnaround_time+=time-at[count];
      flag=0;
    }
    if(count==n-1)
      count=0;
    else if(at[count+1]<=time)
      count++;
    else
      count=0;
  }
  printf("\nAverage Waiting Time= %f\n",wait_time*1.0/n);
  printf("Avg Turnaround Time = %f",turnaround_time*1.0/n);

  return 0;
}
```



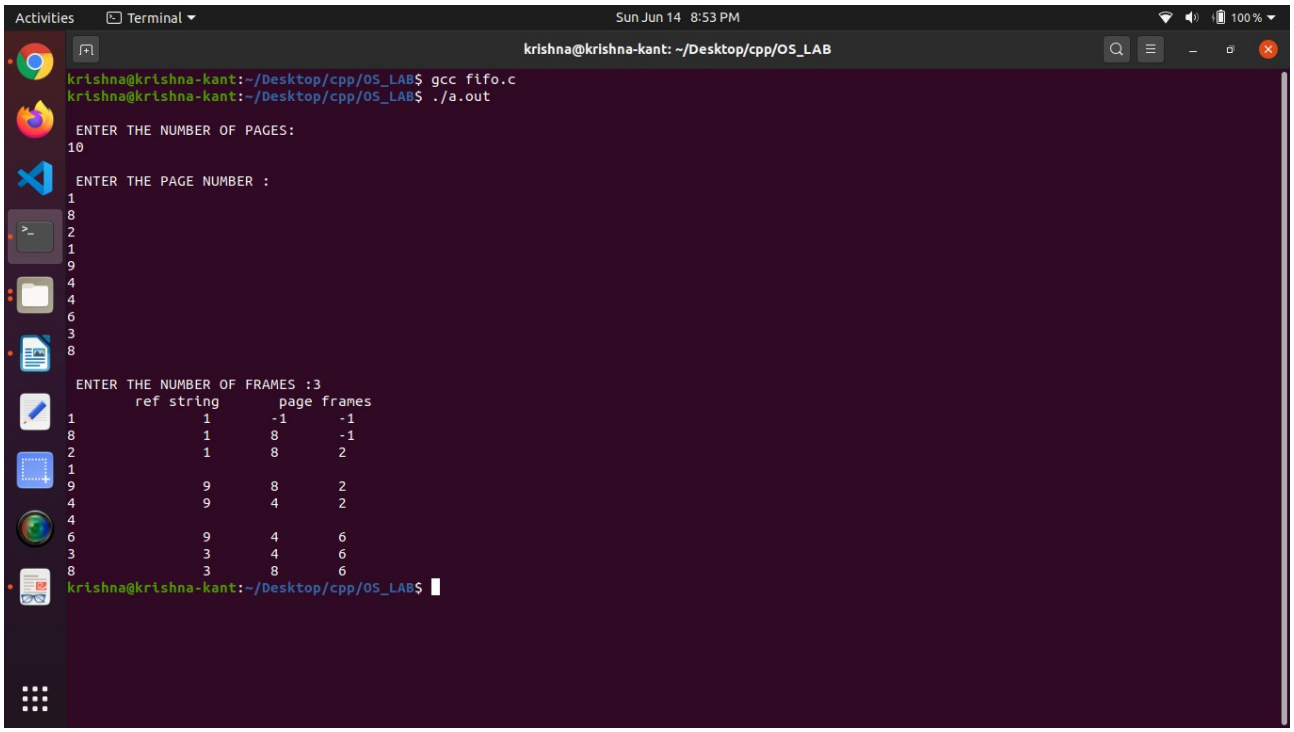
EXPERIMENT :15

AIM: write a c program to implementation of round robin.

PROGRAM:

```
#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;
```

}


Activities
Terminal
Sun Jun 14 8:53 PM
100 %
krishna@krishna-kant: ~/Desktop/cpp/OS_LAB
krishna@krishna-kant:~/Desktop/cpp/OS_LAB$ gcc fifo.c
krishna@krishna-kant:~/Desktop/cpp/OS_LAB$ ./a.out
ENTER THE NUMBER OF PAGES:
10
ENTER THE PAGE NUMBER :
1
8
2
1
9
4
4
6
3
8
ENTER THE NUMBER OF FRAMES :3
ref string page frames
1 1 -1 -1
8 1 8 -1
2 1 8 2
1
9 9 8 2
4 9 4 2
4
6 9 4 6
3 3 4 6
8 3 8 6
krishna@krishna-kant:~/Desktop/cpp/OS_LAB$

EXPERIMENT :16

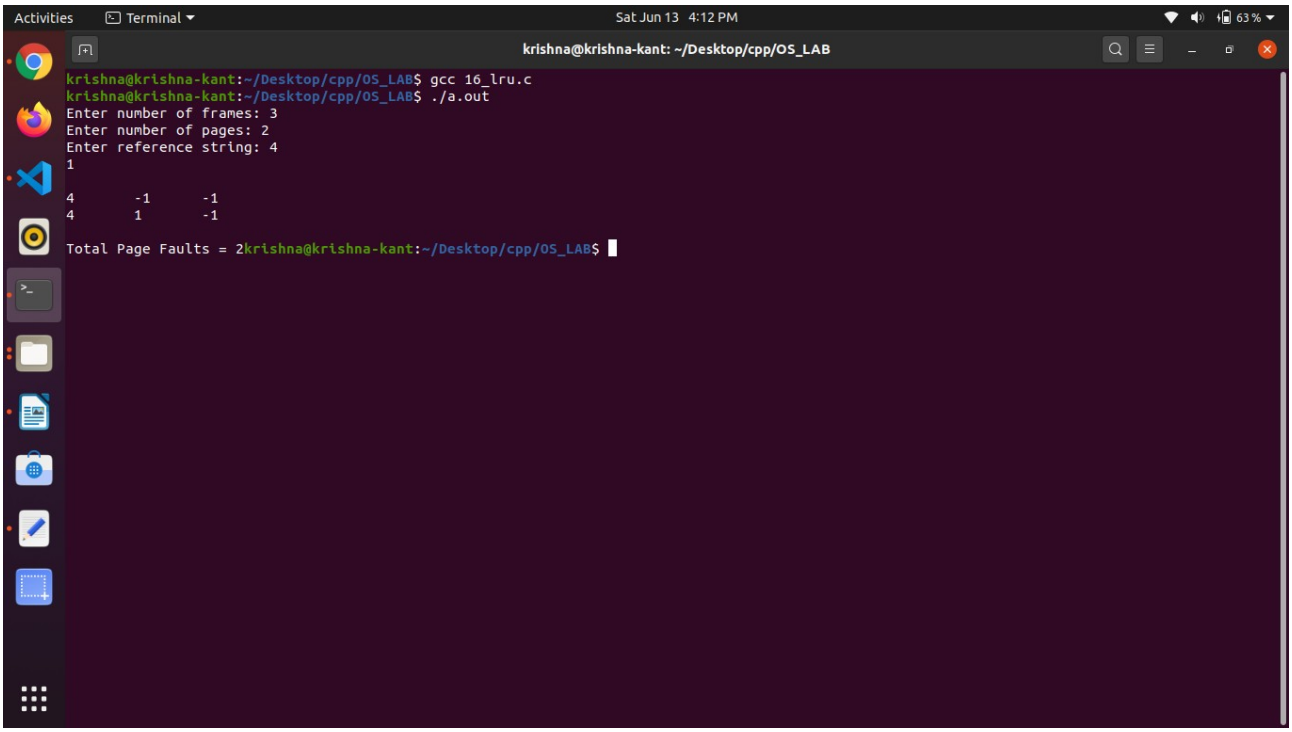
AIM: write a c program to implementation of lru.

PROGRAM:

```
#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;
}
```


Activities
Terminal
Sat Jun 13 4:12 PM
63 %
krishna@krishna-kant: ~/Desktop/cpp/OS_LAB
krishna@krishna-kant:~/Desktop/cpp/OS_LAB$ gcc 16_lru.c
krishna@krishna-kant:~/Desktop/cpp/OS_LAB$ ./a.out
Enter number of frames: 3
Enter number of pages: 2
Enter reference string: 4
1
4 -1 -1
4 1 -1
Total Page Faults = 2krishna@krishna-kant:~/Desktop/cpp/OS_LAB$

EXPERIMENT :17

AIM: linux commands:-

a)grep

b)ls

PROGRAM:

A)grep:

Grep is an acronym that stands for **G**lobal **R**egular **E**xpression **P**rint.

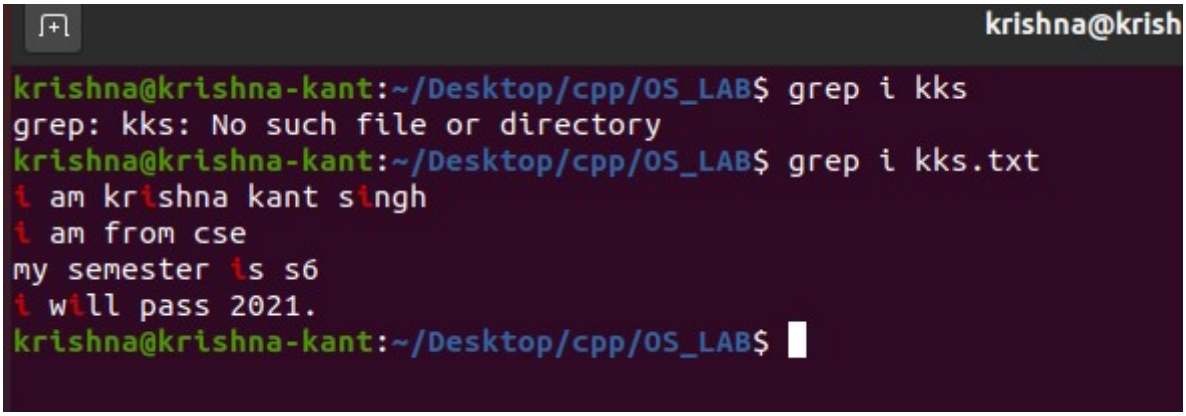
Grep is a Linux / Unix command-line tool used to search for a string of characters in a specified file. The text search pattern is called a regular expression. When it finds a match, it prints the line with the result. The grep command is handy when searching through large log files.

To Search a File

To print any line from a file that contains a specific pattern of characters, in our case **i** in the file **kks.txt** run the command:

```
grep i kks.txt
```

Grep will display every line where there is a match for the word **i** When executing this command, you do not get exact matches. Instead, the terminal prints the lines with words containing the string of characters you entered. Here is an example:



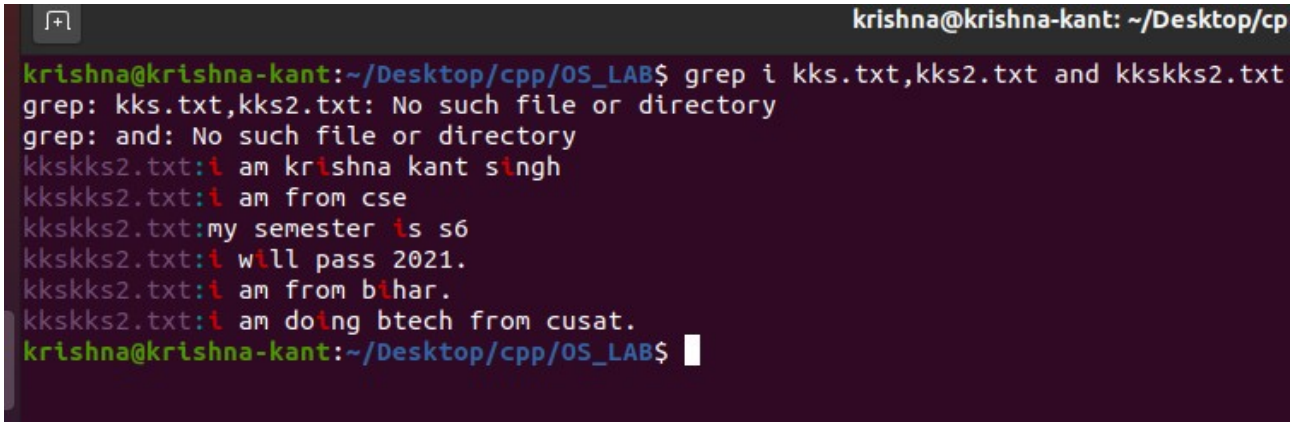
To Search Multiple Files

To search multiple files with the grep command, insert the filenames you want to search, separated with a space character.

In our case, the grep command to match the word `phoenix` in three files `kks.txt,kks2.txt` and `kkskks2.txt` looks like this example:

```
grep i kks.txt kks2.txt kkskks2.txt
```

The terminal prints the name of every file that contains the matching lines, and the actual lines that include the required string of characters.



Search All Files in Directory

To search all files in the current directory, use an asterisk instead of a filename at the end of a grep command.

In this example, we use **nix** as a search criterion:

```
grep nix *
```

The output shows the name of the file with **nix** and returns the entire line.

To Find Whole Words Only

Grep allows you to find and print the results for whole words only. To search for the word **phoenix** in all files in the current directory, append **–w** to the grep command.

```
grep -w phoenix *
```

This option only prints the lines with whole-word matches and the names of the files it found them in:

```
krishna@krishna-kant:~/Desktop/os$ grep -w i *
kks2.txt:i am from bihar.
kks2.txt:i am doing btech from cusat.
kks_copy.txt:i am krishna kant singh
kks_copy.txt:i am from cse
kks_copy.txt:i will pass 2021.
kkskks2.txt:i am krishna kant singh
kkskks2.txt:i am from cse
kkskks2.txt:i will pass 2021.
kkskks2.txt:i am from bihar.
kkskks2.txt:i am doing btech from cusat.
kks.txt:i am krishna kant singh
kks.txt:i am from cse
kks.txt:i will pass 2021.
krishna@krishna-kant:~/Desktop/os$
```

To Ignore Case in Grep Searches

As grep commands are case sensitive, one of the most useful operators for grep searches is **-i.**Instead of printing lowercase results only, the terminal displays both uppercase and lowercase results. The output includes lines with mixed case entries. An example of this command:

```
grep -i phoenix *
```

If we use the **-i operator** to search files in the current directory for **phoenix**, the output looks like this:

```
krishna@krishna-kant: ~/Desktop/os
krishna@krishna-kant:~/Desktop/os$ grep -i i *
kks2.txt:i am from bihar.
kks2.txt:i am doing btech from cusat.
kks_copy.txt:i am krishna kant singh
kks_copy.txt:i am from cse
kks_copy.txt:my semester is s6
kks_copy.txt:i will pass 2021.
kkskks2.txt:i am krishna kant singh
kkskks2.txt:i am from cse
kkskks2.txt:my semester is s6
kkskks2.txt:i will pass 2021.
kkskks2.txt:i am from bihar.
kkskks2.txt:i am doing btech from cusat.
kks.txt:i am krishna kant singh
kks.txt:i am from cse
kks.txt:my semester is s6
kks.txt:i will pass 2021.
krishna@krishna-kant:~/Desktop/os$
```

To Search Subdirectories

To include all subdirectories in a search, add the **-r** operator to the grep command.

```
grep -r phoenix *
```

This command prints the matches for all files in the current directory, subdirectories, and the exact path with the filename. In the example below, we also added the **-w** operator to show whole wordbut the output form is the same.

```
test@test-VirtualBox: ~
File  Edit  View  Search  Terminal  Help
test@test-VirtualBox:~$ grep -rw phoenix *
Desktop/files/sample2:phoenix number2
Desktop/files/sample:phoenix number1
Desktop/files/sample3:phoenix number3
test@test-VirtualBox:~$
```

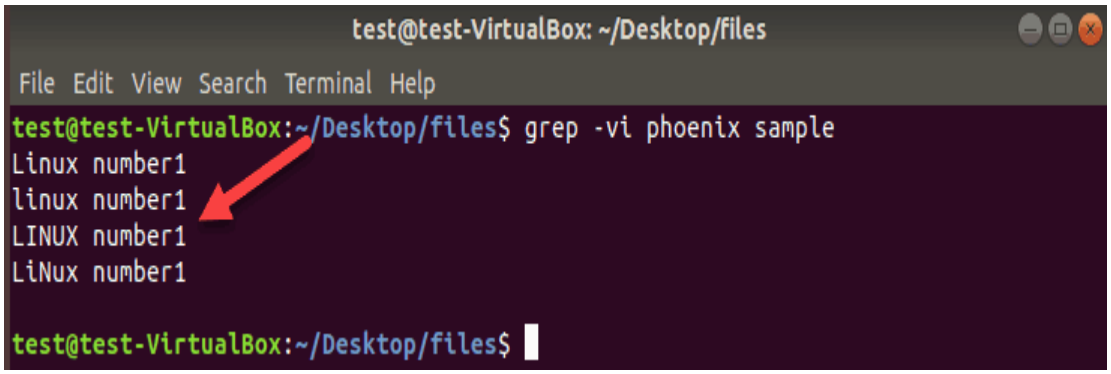
Inverse grep Search

You can use grep to print all lines that **do not** match a specific pattern of characters. To invert the search, append **-v** to a grep command.

To exclude all lines that contain **phoenix**, enter:

```
grep -v phoenix sample
```

The terminal prints all lines that do not contain the word used as a search criterion. Use **-i** to ignore case to exclude completely the word used for this search:



To Show Lines That Exactly Match a Search String

The grep command prints entire lines when it finds a match in a file. To print only those lines that completely match the search string, add the **-x** option.

```
grep -x “phoenix number3” *
```

The output shows only the lines with the exact match. If there are any other words or characters in the same line, the grep does not include it in the search results. Do not forget to use quotation marks whenever there is a space or a symbol in a search pattern.

Here is a comparison of the results without and with the **-x** operator in our grep command:

```
test@test-VirtualBox: ~/Desktop/files
File  Edit  View  Search  Terminal  Help
test@test-VirtualBox:~/Desktop/files$ grep "phoenix number3" *
sample3:phoenix number3
sample3:phoenix number3 test2
sample3:phoenix number3 test3
test@test-VirtualBox:~/Desktop/files$ grep -x "phoenix number3" *
sample3:phoenix number3
test@test-VirtualBox:~/Desktop/files$
```

To List Names of Matching Files

Sometimes, you only need to see the names of the files that contain a word or string of characters and exclude the actual lines. To print only the filenames that match your search, use the **-l** operator:

```
grep -l phoenix *
```

The output shows the exact filenames that contain **phoenix** in the current directory but does not print the lines with the corresponding word:

```
test@test-VirtualBox: ~/Desktop/files
File  Edit  View  Search  Terminal  Help
test@test-VirtualBox:~/Desktop/files$ grep -l phoenix *
sample
sample2
sample3
test@test-VirtualBox:~/Desktop/files$
```

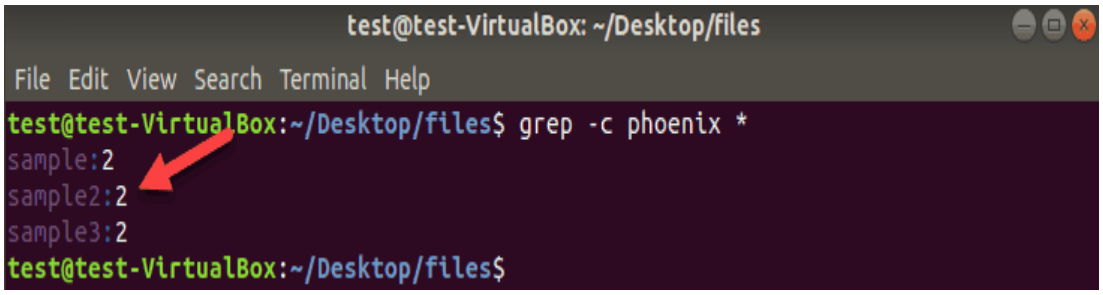
As a reminder, use the recursive search operator **-r** to include all subdirectories in your search.

To Count the Number of Matches

Grep can display the filenames and the count of lines where it finds a match for your word.

Use the **-c** operator to count the number of matches:

```
grep -c phoenix *
```



To Display the Number of Lines Before or After a Search String

Sometimes you need more content in search results to decide what is most relevant.

Use the following operators to add the desired lines before, after a match, or both:

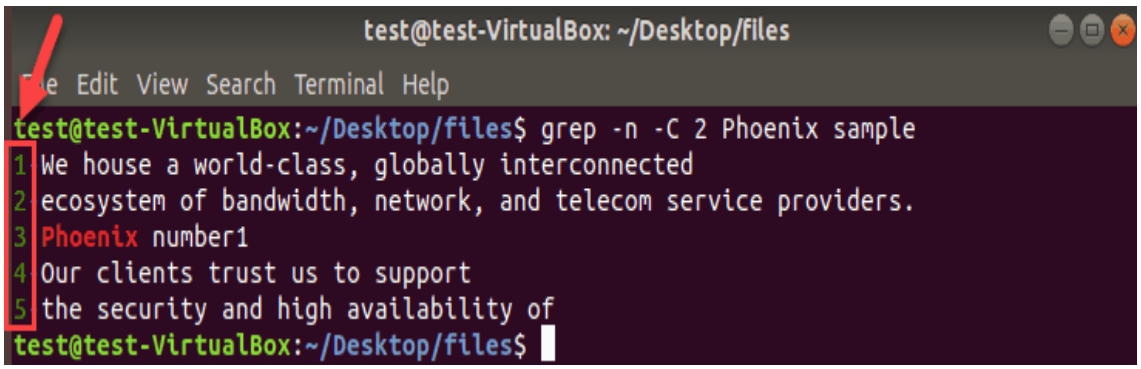
- Use **–A** and a number of lines to display after a match: `grep –A 3 phoenix sample` – this command prints three lines after the match.
- Use **–B** and a number of lines to display before a match: `grep –B 2 phoenix sample` – this command prints two lines before the match.
- Use **–C** and a number of lines to display before **and** after the match: `grep –C 2 phoenix sample` – this command prints two lines before and after the match.

To Display Line Numbers with grep Matches

When grep prints results with many matches, it comes handy to see the line numbers. Append the **-n** operator to any grep command to show the line numbers.

We will search for **Phoenix** in the current directory, show two lines before and after the matches along with their line numbers.

```
grep –n –C 2 Phoenix sample
```



Limit grep Output to a Fixed Number of Lines
Individual files, such as log files, can contain many matches for grep search patterns. Limit the number of lines in the grep output by adding the **-m** option and a number to the command.

```
grep –m2 Phoenix sample
```

In this case, the terminal prints the first two matches it finds in the **sample** file.

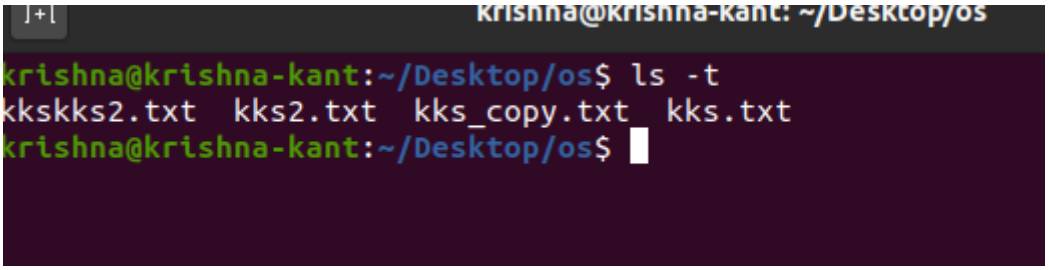
If you do not specify a file and search all files in a directory, the output prints the first two results from **every** file along with the filename that contains the matches.

B)ls command:

ls is a Linux shell command that lists directory contents of files and directories.Some practical examples of ls command are shown below.

1. Open Last Edited File Using ls -t

ls -t : It sorts the file by modification time, showing the last edited file first. head -1 picks up this first file.To open the last edited file in the current directory use the combination of ls and head commands as shown below.



2. Display One File Per Line Using ls -1

```
krishna@krishna-kant: ~/Desktop/os
krishna@krishna-kant:~/Desktop/os$ ls -1
kks2.txt
kks_copy.txt
kkskks2.txt
kks.txt
krishna@krishna-kant:~/Desktop/os$
```

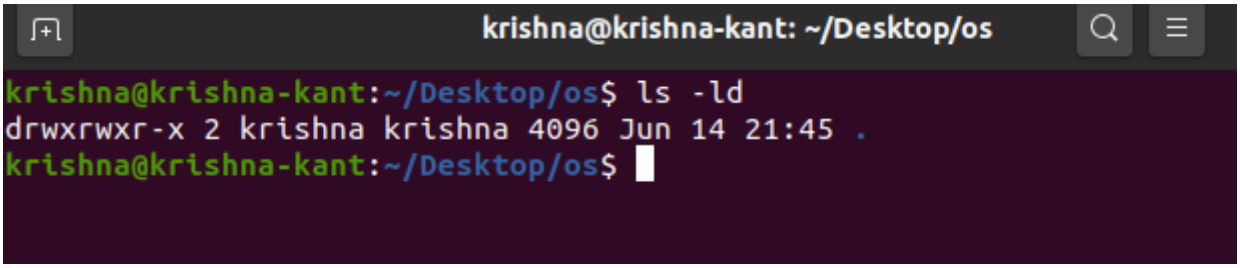
3. Display All Information About Files/Directories Using ls -l

$ ls -l : To show long listing information about the file/directory.

```
krishna@krishna-kant: ~/Desktop/os
krishna@krishna-kant:~/Desktop/os$ ls -l
total 16
-rw-rw-r-- 1 krishna krishna  47 Jun 13 15:43 kks2.txt
-rw-rw-r-- 1 krishna krishna  76 Jun 13 15:38 kks_copy.txt
-rw-rw-r-- 1 krishna krishna 123 Jun 13 15:43 kkskks2.txt
-rw-rw-r-- 1 krishna krishna  76 Jun 12 22:08 kks.txt
krishna@krishna-kant:~/Desktop/os$
```

4. Display File Size in Human Readable Format Using ls -lh

ls -lh (h stands for human readable form) : To display file size in easy to read format. i.e i.e M for MB, K for KB, G for GB.

```
krishna@krishna-kant: ~/Desktop/os
krishna@krishna-kant:~/Desktop/os$ ls -lh
total 16K
-rw-rw-r-- 1 krishna krishna  47 Jun 13 15:43 kks2.txt
-rw-rw-r-- 1 krishna krishna  76 Jun 13 15:38 kks_copy.txt
-rw-rw-r-- 1 krishna krishna 123 Jun 13 15:43 kkskks2.txt
-rw-rw-r-- 1 krishna krishna  76 Jun 12 22:08 kks.txt
krishna@krishna-kant:~/Desktop/os$
```

5. Display Directory Information Using ls -ld