OPERATING SYSTEMS RECORD

->Karedla Anantha Sashi Sekhar

->1601-14-733-091

->BE(3/4) CSE-2

FILE RELATED SYSTEM CALLS

1. Program to create a File

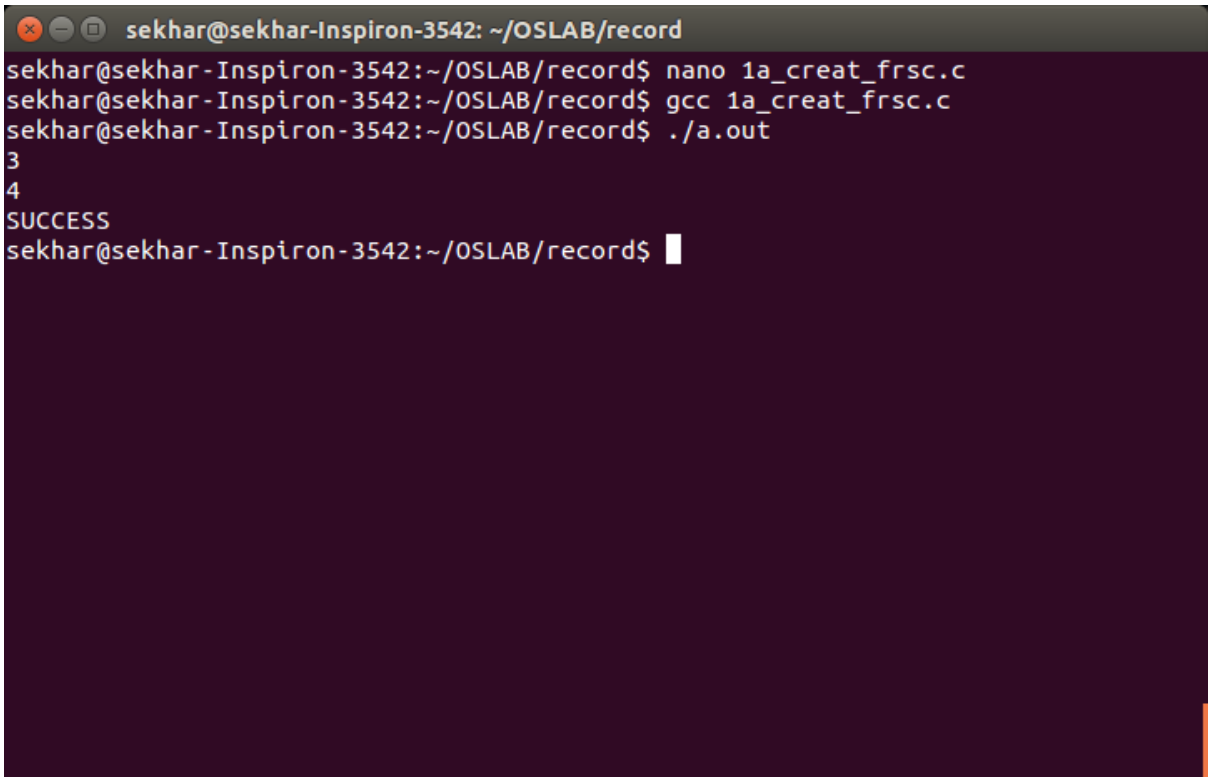
System calls required:
creat() system call :
open, creat - open and possibly create a file or device
Header files required:
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
Syntax:
int creat(const char *pathname, mode_t* mode)
Description:
Given a *pathname* for a file, open() returns a file descriptor, a small, nonnegative integer for use in subsequent system calls. The file descriptor returned by a successful call will be the lowest-numbered file descriptor not currently open for the process.
A call to open() creates a new *open file description*, an entry in the system-wide table of open files. This entry records the file offset and the file status flags. A file descriptor is a reference to one of these entries; this reference is unaffected if *pathname* is subsequently removed or modified to refer to a different file. The new open file description is initially not shared with any other process, but sharing may arise via ***fork****(2)*.
The argument *flags* must include one of the following *access modes*: O_RDONLY, O_WRONLY, or O_RDWR. These request opening the file read-only, write-only, or read/write, respectively.
Return Value:
creat() return the new file descriptor, or -1 if an error occurred.

close() system call :
close - close a file descriptor
Header files required:
#include <unistd.h>
Syntax:
int close(int fd);
Description:
close() closes a file descriptor, so that it no longer refers to any file and may be reused. Any record locks held on the file it was associated with, and owned by the process, are removed (regardless of the file descriptor that was used to obtain the lock).
If *fd* is the last file descriptor referring to the underlying open file description, the resources associated with the open file description are freed; if the descriptor was the last reference to a file which has been removed using *unlink(2)* the file is deleted.
Return Value:
close() returns zero on success. On error, -1 is returned.

Program:

```
#include<stdio.h>                /*header file for main function*/
#include<sys/types.h>
#include<sys/stat.h>  /*header files for creat() system call*/
#include<fcntl.h>
int main()
{
        int fd;             /*creating 2 file descriptors*/
        int fd1;
        fd=creat("first.txt",S_IREAD|S_IWRITE);       /*creating 2 files which */
        fd1=creat("second.txt",S_IREAD|S_IWRITE);     //returns file descriptors
        printf("%d\n",fd);
        printf("%d\n",fd1);
        if(fd==-1)                /*checking whether file descriptor is negative or not*/
                printf("ERROR\n");
        else
                printf("SUCCESS\n");
        close(fd);                          /*closing the file descriptors*/
        close(fd1);
}
```

OUTPUT:


sekhar@sekhar-Inspiron-3542: ~/OSLAB/record
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ nano 1a_creat_frsc.c
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ gcc 1a_creat_frsc.c
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ ./a.out
3
4
SUCCESS
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$

2. Program to write contents from file to console

System calls required:

open() system call:

open - open and possibly create a file or device

Header files required:

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

Syntax:

int open(const char *pathname, int flags);

int open(const char *pathname,int flags, mode_t mode);

Description:

Given a *pathname* for a file, open() returns a file descriptor, a small, nonnegative integer for use in subsequent system calls. The file descriptor returned by a successful call will be the lowest-numbered file descriptor not currently open for the process.

A call to open() creates a new *open file description*, an entry in the system-wide table of open files. This entry records the file offset and the file status flags. A file descriptor is a reference to one of these entries; this reference is unaffected if *pathname* is subsequently removed or modified to refer to a different file. The new open file description is initially not shared with any other process, but sharing may arise via ***fork****(2)*.

The argument *flags* must include one of the following *access modes*: O_RDONLY, O_WRONLY, or O_RDWR. These request opening the file read-only, write-only, or read/write, respectively.

Return Value:

open() return the new file descriptor, or -1 if an error occurred.

Read() system call:

read - read from a file descriptor

Header file required:

#include <unistd.h>

Syntax:

ssize_t read(int fd, void *buf, size_t count);

Description:

read() attempts to read up to *count* bytes from file descriptor *fd* into the buffer starting at *buf*.

Return Value:

On success, the number of bytes read is returned,On error, -1 is returned.

Write() system call:

write - write to a file descriptor

Header file required:

#include <unistd.h>

Syntax:

ssize_t write(int *fd*, const void **buf*, size_t *count*);

Description:

write() writes up to *count* bytes from the buffer pointed *buf* to the file referred to by the file descriptor *fd*.

Return Value:

On success, the number of bytes written is returned (zero indicates nothing was written). On error, -1 is returned.

If *count* is zero and *fd* refers to a regular file, then write() may return a failure status if one of the errors is detected. If no errors are detected, 0 will be returned without causing any other effect. If *count* is zero and *fd* refers to a file other than a regular file, the results are not specified.

Lseek() system call:

lseek - reposition read/write file offset

Header files required:

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

Syntax:

off_t lseek(int *fd*, off_t *offset*, int *whence*);

Description:

The lseek() function repositions the offset of the open file associated with the file descriptor *fd* to the argument *offset* according to the directive *whence* as follows:

SEEK_SET

The offset is set to *offset* bytes.

SEEK_CUR

The offset is set to its current location plus *offset* bytes.

SEEK_END

The offset is set to the size of the file plus *offset* bytes.

Return Value:

Upon successful completion, lseek() returns the resulting offset location as measured in bytes from the beginning of the file. On error, the value *(off_t) -1* is returned.

Exit() system call:

_exit, _Exit - terminate the calling process

Header files required:

#include <unistd.h>

#include <stdlib.h>

Syntax:

void _exit(int *status*);

void _Exit(int *status*);

Description:

The function _exit() terminates the calling process "immediately". Any open file descriptors belonging to the process are closed; any children of the process are inherited by process 1, *init*, and the process's parent is sent a SIGCHLD signal.

The value *status* is returned to the parent process as the process's exit status, and can be collected using one of the wait(2) family of calls.

The function _Exit() is equivalent to _exit().

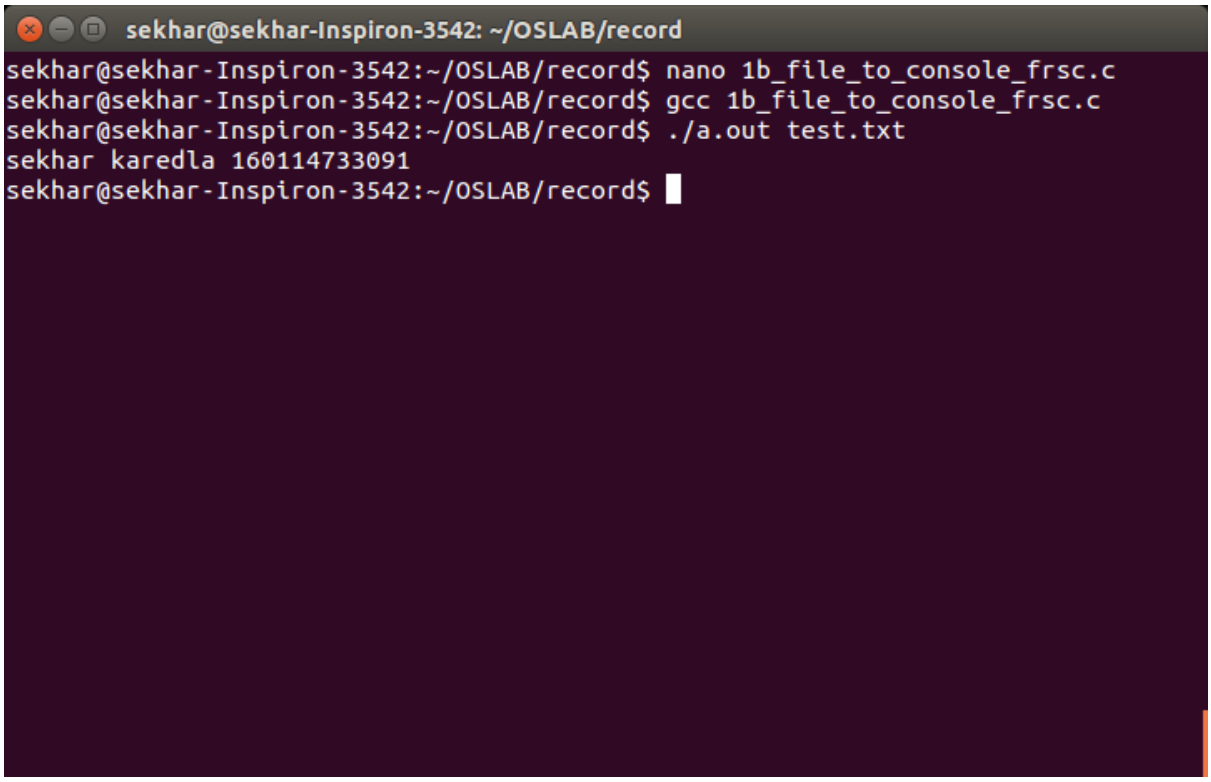
Return Value:

These functions do not return.

Program:

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
int main(int argc,char *argv[])
{
        int fd;
        int n_char=0;
        char buffer[1];
        fd=open(argv[1],O_RDONLY);
        if(fd==-1)
        {
                exit(-1);
        }
        while((n_char=read(fd,buffer,1))!=0)
        {
                //printf("%d",n_char);
                write(1,buffer,n_char);
        }
        return 0;
}
```

Output:


sekhar@sekhar-Inspiron-3542: ~/OSLAB/record
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ nano 1b_file_to_console_frsc.c
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ gcc 1b_file_to_console_frsc.c
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ ./a.out test.txt
sekhar karedla 160114733091
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$

3. Program to read from one file and write to another file

System calls required:

Creat(),open() system call

open, creat - open and possibly create a file or device

Header files required:

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

Syntax:

int creat(const char *pathname, mode_t mode);

int open(const char *pathname, int flags);

int open(const char *pathname,int flags, mode_t mode);

Description:

Given a *pathname* for a file, open() returns a file descriptor, a small, nonnegative integer for use in subsequent system calls. The file descriptor returned by a successful call will be the lowest-numbered file descriptor not currently open for the process.

A call to open() creates a new *open file description*, an entry in the system-wide table of open files. This entry records the file offset and the file status flags. A file descriptor is a reference to one of these entries; this reference is unaffected if *pathname* is subsequently removed or modified to refer to a different file. The new open file description is initially not shared with any other process, but sharing may arise via ***fork****(2)*.

The argument *flags* must include one of the following *access modes*: O_RDONLY, O_WRONLY, or O_RDWR. These request opening the file read-only, write-only, or read/write, respectively.

Return Value:

creat() and open() return the new file descriptor, or -1 if an error occurred.

Read() system call:

read - read from a file descriptor

Header file required:

#include <unistd.h>

Syntax:

ssize_t read(int fd, void *buf, size_t count);

Description:

read() attempts to read up to *count* bytes from file descriptor *fd* into the buffer starting at *buf*.

Return Value:

On success, the number of bytes read is returned,On error, -1 is returned.

Write() system call:

write - write to a file descriptor

Header file required:

#include <unistd.h>

Syntax:

ssize_t write(int *fd*, const void **buf*, size_t *count*);

Description:

write() writes up to *count* bytes from the buffer pointed *buf* to the file referred to by the file descriptor *fd*.

Return Value

On success, the number of bytes written is returned (zero indicates nothing was written). On error, -1 is returned.

If *count* is zero and *fd* refers to a regular file, then write() may return a failure status if one of the errors is detected. If no errors are detected, 0 will be returned without causing any other effect. If *count* is zero and *fd* refers to a file other than a regular file, the results are not specified.

Close(2) system call:

close - close a file descriptor

Header files required:

#include <unistd.h>

Syntax:

int close(int fd);

Description:

close() closes a file descriptor, so that it no longer refers to any file and may be reused. Any record locks held on the file it was associated with, and owned by the process, are removed (regardless of the file descriptor that was used to obtain the lock).

If *fd* is the last file descriptor referring to the underlying open file description, the resources associated with the open file description are freed; if the descriptor was the last reference to a file which has been removed using *unlink(2)* the file is deleted.

Return Value:

close() returns zero on success. On error, -1 is returned.

Exit() system call:

_exit, _Exit - terminate the calling process

Header files required:

#include <unistd.h>

#include <stdlib.h>

Syntax:

void _exit(int *status*);

void _Exit(int *status*);

Description

The function _exit() terminates the calling process "immediately". Any open file descriptors belonging to the process are closed; any children of the process are inherited by process 1, *init*, and the process's parent is sent a SIGCHLD signal.

The value *status* is returned to the parent process as the process's exit status, and can be collected using one of the wait(2) family of calls.

The function _Exit() is equivalent to _exit().

Return Value

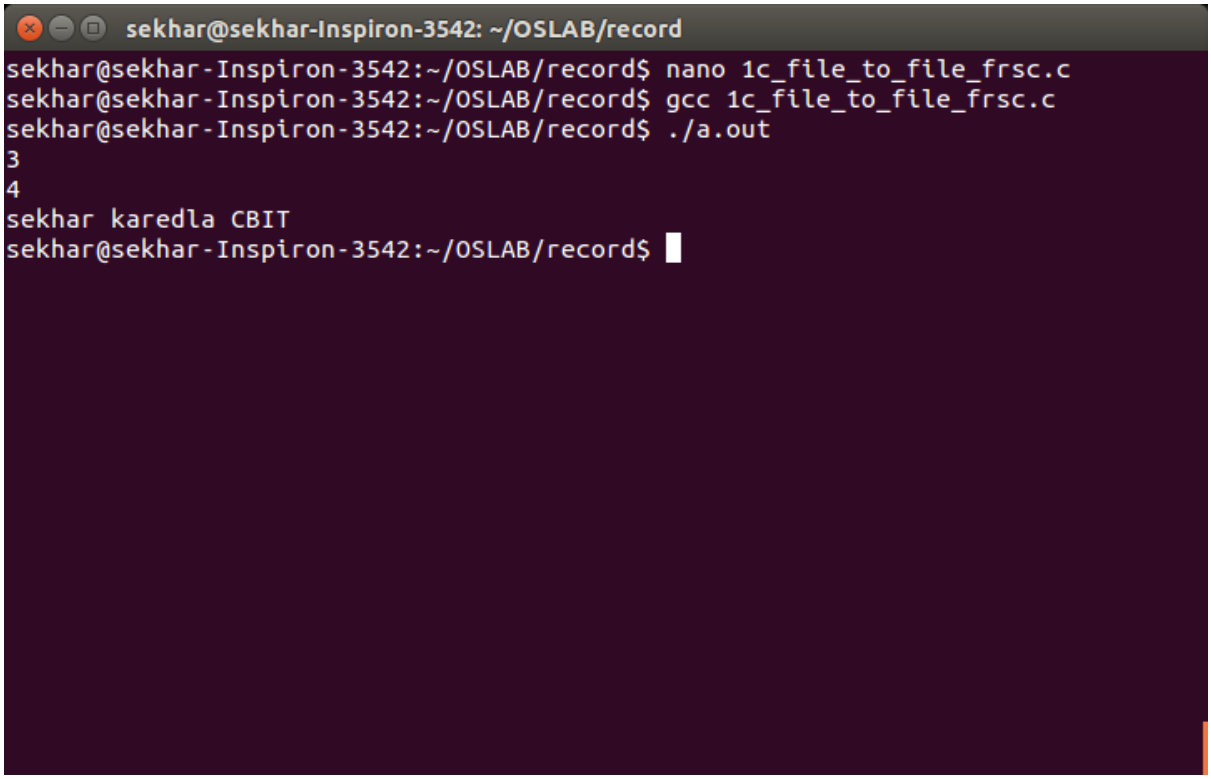
These functions do not return.

Program:

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
int main()
{
        int fd1,fd2;
        char ch[1];
        fd1=open("first.txt",O_RDONLY);
        printf("%d\n",fd1);
        fd2=creat("second.txt",S_IREAD|S_IWRITE);
        printf("%d\n",fd2);
        if(fd1<0||fd2<0)
        {
                printf("Error");
                exit(-1);
        }
        while((read(fd1,ch,1))>0)
                {
                write(fd2,ch,1);
                printf("%c",ch[0]);
                }
                close(fd1);
```

```
        close(fd2);

        return 0;

}
```

Output:



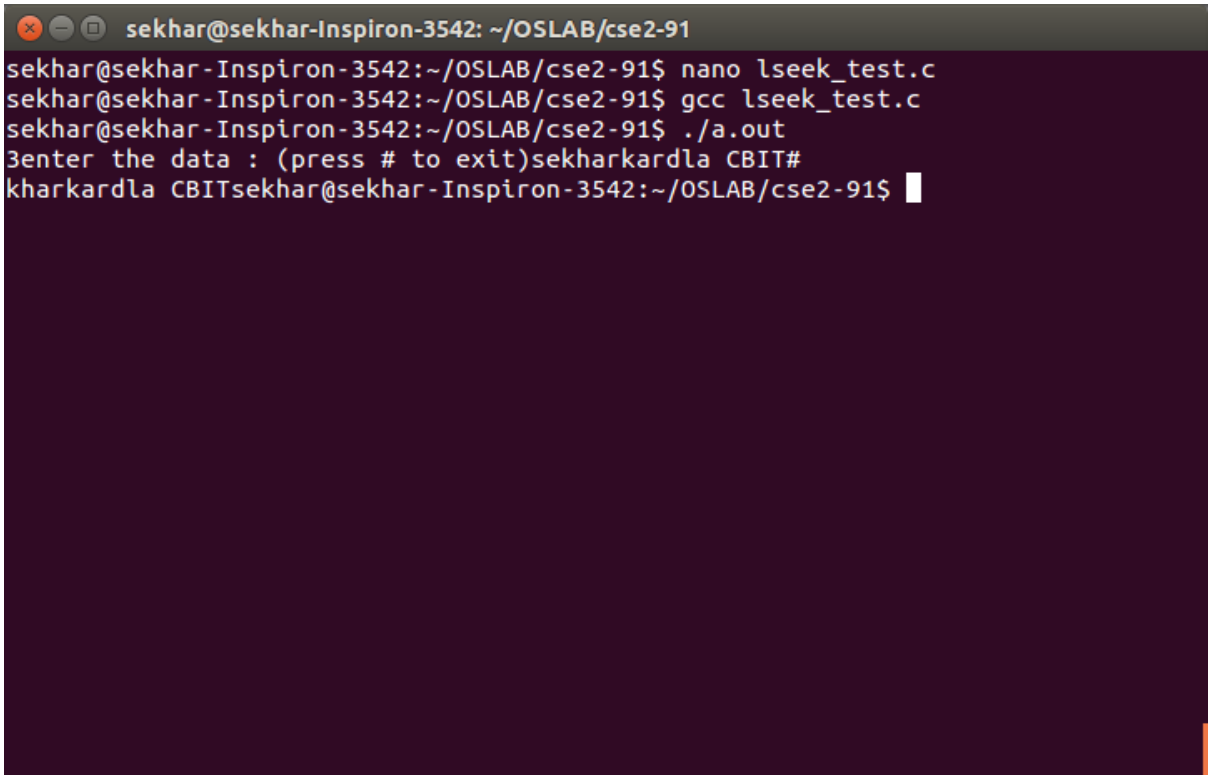
4.Program to show the working of lseek function.

Program:

```
#include<unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<stdio.h>
#include<stdlib.h>
int main()
{
        int fd1=open("1.txt",O_RDWR);
        printf("%d",fd1);
        char buffer[1];
        buffer[0]='1';
        printf("enter the data : (press # to exit)");
        do
        {
        scanf("%c",&buffer[0]);
        if(buffer[0]!='#')
        write(fd1,buffer,1);
        }while(buffer[0]!='#');
        close(fd1);
        int fd2=open("1.txt",O_RDWR);
        lseek(fd2,2*sizeof(char),0);int k;
        do
```

```
        {
        k=read(fd2,&buffer[0],1);
        if(k!=0)
        printf("%c",buffer[0]);
        }while(k!=0);
return 0;
}
```

Output:-



PROCESS RELATED SYSTEM CALLS

5. Program to demonstrate fork system call

System calls required:

fork() system call:

fork - create a child process

Header file required:

#include <unistd.h>

Syntax:

pid_t fork(void);

Description:

fork() creates a new process by duplicating the calling process. The new process, referred to as the *child*, is an exact duplicate of the calling process, referred to as the *parent*.

Return Value:

On success, the PID of the child process is returned in the parent, and 0 is returned in the child. On failure, -1 is returned in the parent, no child process is created.

open(2) system call:

open, creat - open and possibly create a file or device

Header files required:

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

Syntax:

int open(const char *pathname, int flags);

int open(const char *pathname,int flags, mode_t mode);

Description:

Given a *pathname* for a file, open() returns a file descriptor, a small, nonnegative integer for use in subsequent system calls. The file descriptor returned by a successful call will be the lowest-numbered file descriptor not currently open for the process.

A call to open() creates a new *open file description*, an entry in the system-wide table of open files. This entry records the file offset and the file status flags. A file descriptor is a reference to one of these entries; this reference is unaffected if *pathname* is subsequently removed or modified to refer to a different file. The new open file description is initially not shared with any other process, but sharing may arise via ***fork****(2)*.

The argument *flags* must include one of the following *access modes*: O_RDONLY, O_WRONLY, or O_RDWR. These request opening the file read-only, write-only, or read/write, respectively.

Return Value:

open() return the new file descriptor, or -1 if an error occurred.

Read() system call:

read - read from a file descriptor

Header file required:

#include <unistd.h>

Syntax:

ssize_t read(int fd, void *buf, size_t count);

Description:

read() attempts to read up to *count* bytes from file descriptor *fd* into the buffer starting at *buf*.

Return Value:

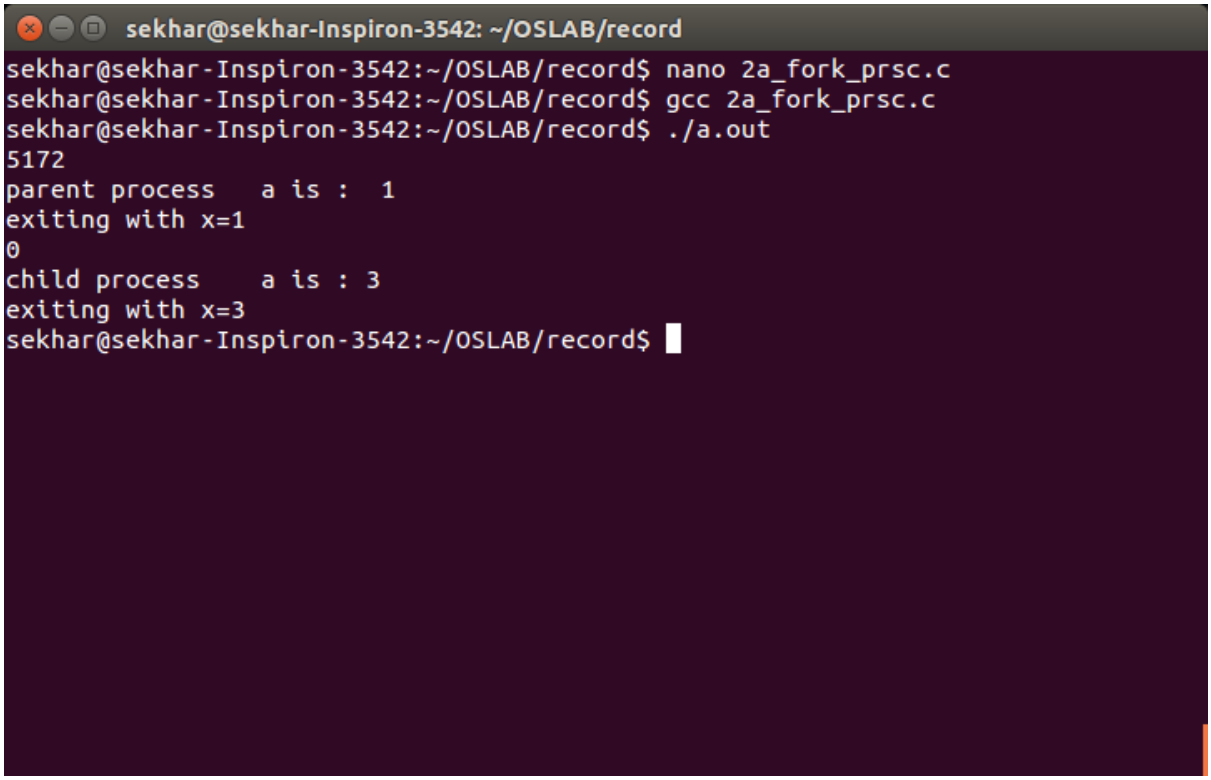
On success, the number of bytes read is returned,On error, -1 is returned.

Program:

```
#include<stdio.h>
#include<unistd.h>
int main()
{
	int a=2;
	pid_t pid;
	pid=fork();
	printf("%d\n",pid);
	if(pid<0)
	{
		printf("fork failed");
	}
	else if(pid==0)
	{
		printf("child process \t a is : ");
		printf("%d\n",++a);
	}
	else
	{
		printf("parent process \t a is :  ");
		printf("%d\n",--a);
	}
	printf("exiting with x=%d\n",a);
```

}

Output:


sekhar@sekhar-Inspiron-3542: ~/OSLAB/record
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ nano 2a_fork_prsc.c
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ gcc 2a_fork_prsc.c
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ ./a.out
5172
parent process a is : 1
exiting with x=1
0
child process a is : 3
exiting with x=3
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$

6. Program to demonstrate getpid(),getppid() system calls

System calls required:

fork() system call:

fork - create a child process

Header file required:

#include <unistd.h>

Syntax:

pid_t fork(void);

Description:

fork() creates a new process by duplicating the calling process. The new process, referred to as the *child*, is an exact duplicate of the calling process, referred to as the *parent*.

Return Value:

On success, the PID of the child process is returned in the parent, and 0 is returned in the child. On failure, -1 is returned in the parent, no child process is created.

getpid(),getppid() system calls:

getpid, getppid - get process identification

Header files required:

#include <**sys/types.h**>
#include <**unistd.h**>

Syntax:

pid_t getpid(void);
pid_t getppid(void);

Description:

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

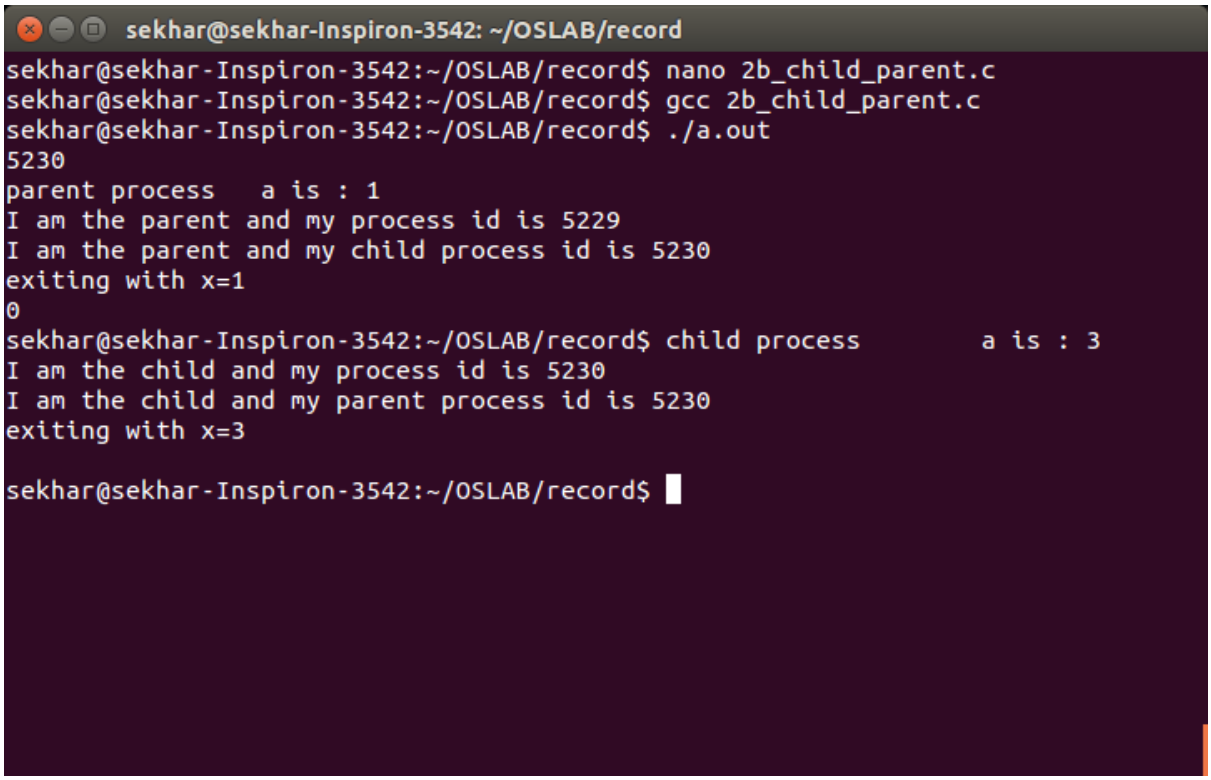
getppid() returns the process ID of the parent of the calling process.

Program:

```
#include<stdio.h>
#include<unistd.h>
int main()
{
	int a=2;
	pid_t pid;
	pid=fork();
	printf("%d\n",pid);
	if(pid<0)
	{
		printf("Error");
	}
	else if(pid==0)
	{
		sleep(10);
		printf("child process \t a is : ");
		printf("%d\n",++a);
		printf("I am the child and my process id is %d\n",getpid());
		printf("I am the child and my parent process id is %d\n",getpid());
	}
```

```
        else
        {
                printf("parent process \t a is : ");
                printf("%d\n",--a);
                printf("I am the parent and my process id is %d\n",getpid());
                printf("I am the parent and my child process id is %d\n",pid);

        }
        printf("exiting with x=%d\n",a);
}
```

Output:



7. Program to demonstrate getpid(),getppid() system calls without sleep

System calls required:

fork() system call:

fork - create a child process

Header file required:

#include <unistd.h>

Syntax:

pid_t fork(void);

Description:

fork() creates a new process by duplicating the calling process. The new process, referred to as the *child*, is an exact duplicate of the calling process, referred to as the *parent*.

Return Value:

On success, the PID of the child process is returned in the parent, and 0 is returned in the child. On failure, -1 is returned in the parent, no child process is created.

getpid(),getppid() system calls:

getpid, getppid - get process identification

Header files required:

#include <**sys/types.h**>
#include <**unistd.h**>

Syntax:

pid_t getpid(void);
pid_t getppid(void);

Description:

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

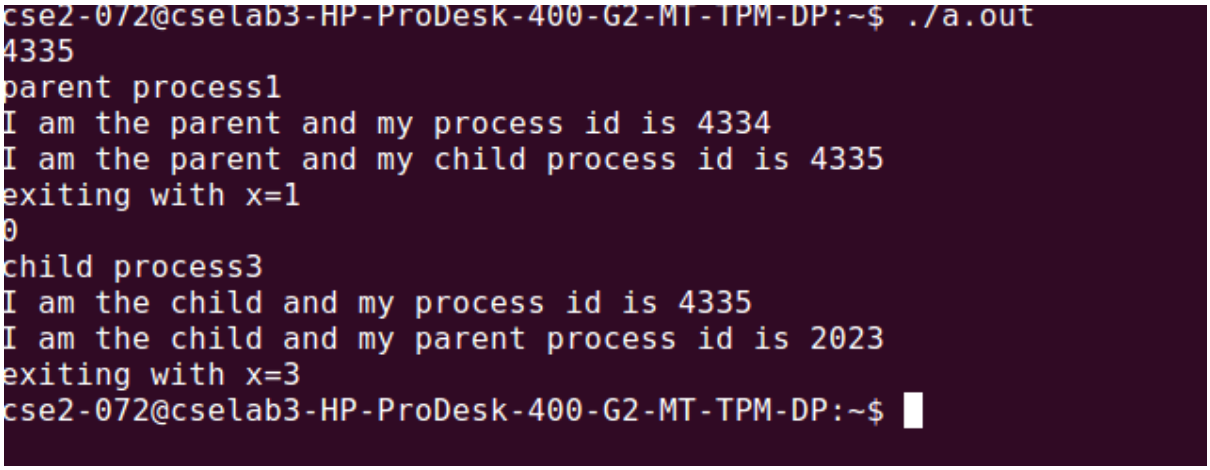
getppid() returns the process ID of the parent of the calling process.

Program:

```
#include<stdio.h>
#include<unistd.h>
int main()
{
        int a=2;
        pid_t pid;
        pid=fork();
        printf("%d\n",pid);
        if(pid<0)
        {
                printf("Error");
        }
        else if(pid==0)
        {
                printf("child process");
                printf("%d\n",++a);
                printf("I am the child and my process id is %d\n",getpid());
                printf("I am the child and my parent process id is %d\n",getppid());
        }
        else
        {
                printf("parent process");
                printf("%d\n",--a);
                printf("I am the parent and my process id is %d\n",getpid());
                printf("I am the parent and my child process id is %d\n",pid);
```

```
        }

        printf("exiting with x=%d\n",a);

}
```

Output:



8. Program to demonstrate execlp and wait system calls

System calls required:

Wait() system calls:

wait, waitpid, waitid - wait for process to change state

Header files required:

#include<**sys/types.h**>
#include <**sys/wait.h**>

Syntax:

pid_t wait(int **status*);

pid_t waitpid(pid_t *pid*, int **status*, int *options*);

int waitid(idtype_t *idtype*, id_t *id*, siginfo_t **infop*, int *options*);

Description:

The wait() system call suspends execution of the calling process until one of its children terminates. The call *wait(&status)* is equivalent to: waitpid(-1, &status, 0);

The waitpid() system call suspends execution of the calling process until a child specified by *pid* argument has changed state. By default, waitpid() waits only for terminated children.

Return value:

If *wait*() or *waitpid*() returns because the status of a child process is available, these functions shall return a value equal to the process ID of the child process for which *status* is reported. If *wait*() or *waitpid*() returns due to the delivery of a signal to the calling process, -1 shall be returned. If *waitpid*() was invoked with WNOHANG set in *options*, it has at least one child process specified by *pid* for which *status* is not available, and *status* is not available for any process specified by *pid*, 0 is returned. Otherwise, (pid_t)-1 shall be returned.

Execlp() system call:

execl, execlp, execle, execv, execvp, execvpe - execute a file

Header file required:

#include <**unistd.h**>

Syntax:

Int execl(const char *_path_, const char*_arg_,...);
int execlp(const char *_file_, const char *_arg_, ...);
int execle(const char *_path_, const char *_arg_,..., char * const _envp_[]);
int execv(const char *_path_, char *const _argv_[]);
int execvp(const char *_file_, char *const _argv_[]);
int execvpe(const char *_file_, char *const _argv_[],char *const _envp_[]);

Description:

The execlp(), execvp(), and execvpe() functions duplicate the actions of the shell in searching for an executable file if the specified filename does not contain a slash (/) character.

Return Value

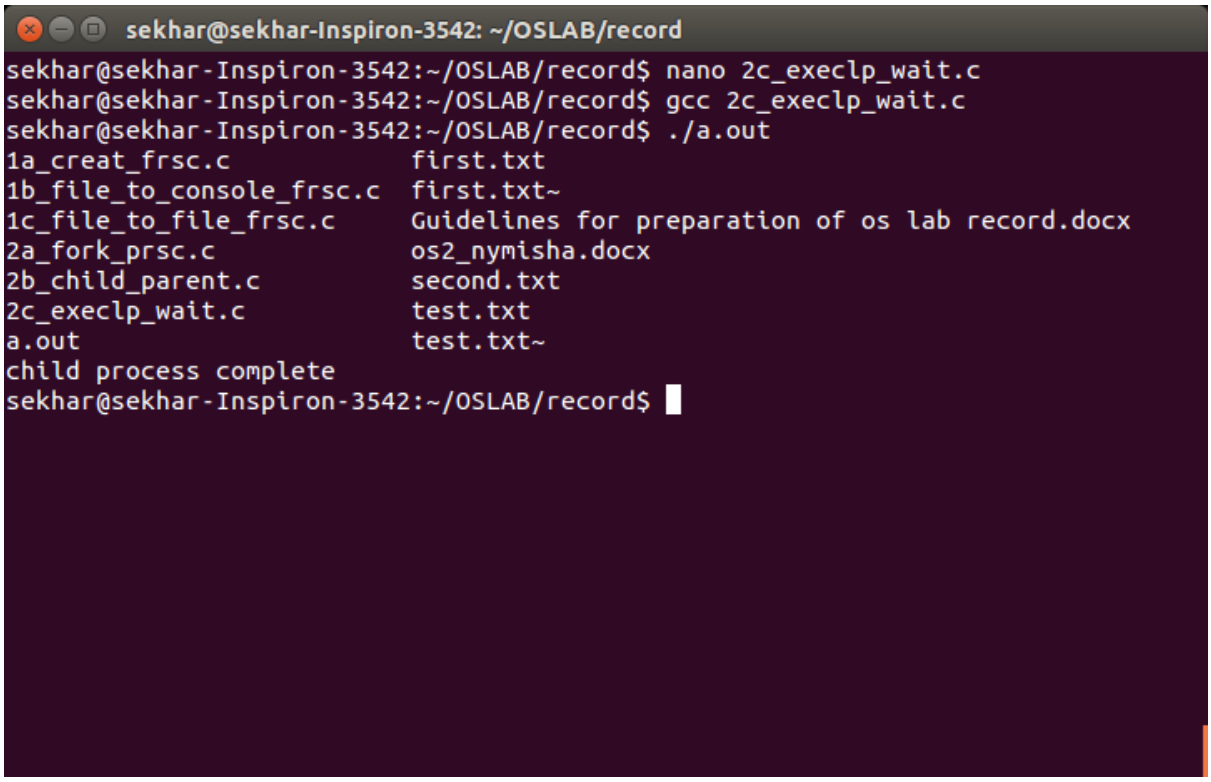
The exec() functions only return if an error has occurred. The return value is -1.

Program:

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<string.h>
#include<errno.h>
int main(int argc,char *argv[])
{
        int pid,childpid,status;
        pid=fork();
        if(pid<0)
        {
                fprintf(stderr,"fork failed");
                return 1;
        }
        else if(pid==0)
        {
                execlp("/bin/ls","ls",NULL);
                _exit(0);
        }
        else
        {
                wait(NULL);
                printf("child process complete\n");
```

```
        }

        return 0;

}
```

Output:



9.Program to show the working of execv():

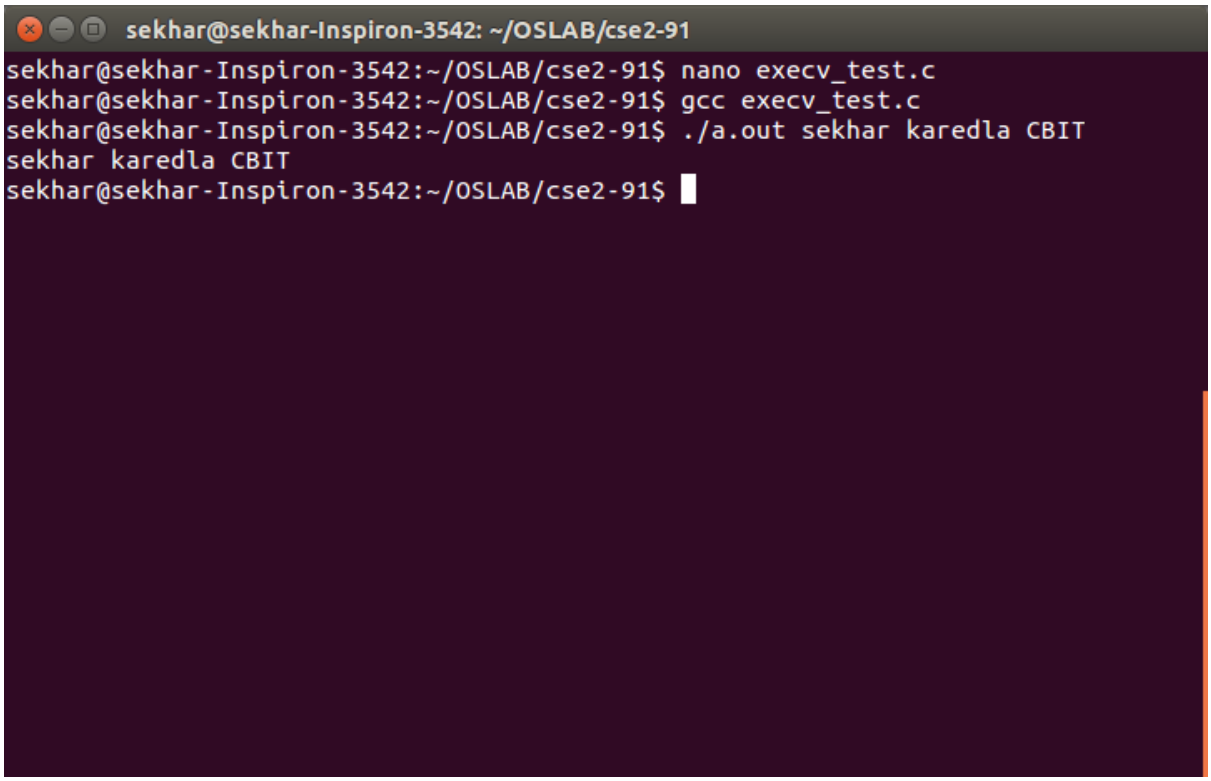
The execv(), execvp(), and execvpe() functions provide an array of pointers to null-terminated strings that represent the argument list available to the new program. The first argument, by convention, should point to the filename associated with the file being executed. The array of pointers must be terminated by a NULL pointer.

Program:-

```
#include<unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<stdio.h>
#include<stdlib.h>
int main(int argc,char *argv[])
{
        int pid;
        int status;
        pid=fork();
        if(pid<0)
        exit(-1);
        else if(pid==0)
        {

        execv("/bin/echo",argv);
        exit(1);
```

```
        }
        else
        {
        int childid=wait(&status);
        exit(2);
        }
return 0;
}
```

Output:



CPU SCHEDULING

10. FCFS(First Come First Serve)

Algorithm:

1. Start the process

2. Declare the array size

3. Get the number of processes to be inserted

4. Get the value

5. Start with the first process from it's initial position let other process to be in queue

6. Calculate the total number of burst time

7. Display the values

8. Stop the process

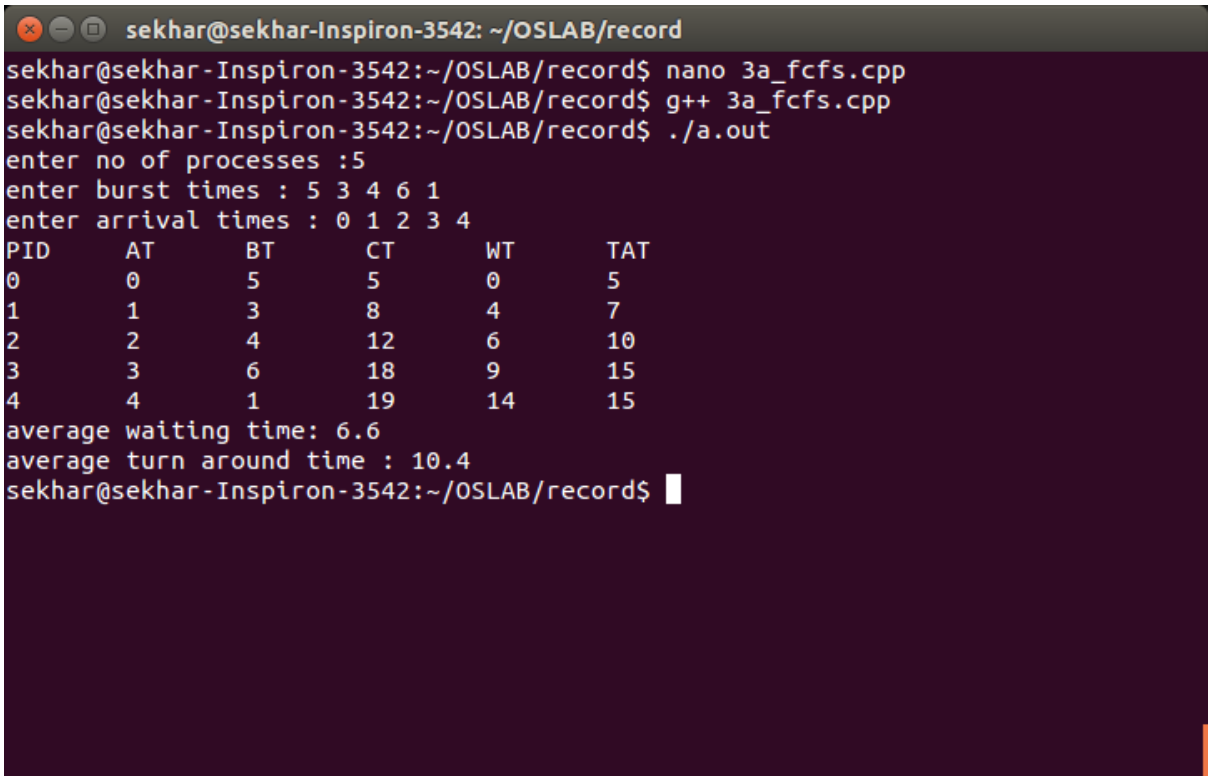
Program:

```
#include<iostream>

using namespace std;

int shortest(int at[],int done[],int n)

{

int k=0;

        for(int i=0;i<n;i++)

        {

                if(at[k]>at[i]&&done[i]!=1)

                k=i;

                else if(at[k]==at[i]&&done[i]!=1)

                k=i;

        }
```

```
return k;
}
int main()
{
	int n;
	cout<<"enter no of processes :";
	cin>>n;
	int *bt=new int[n];
	cout<<"enter burst times : ";
	for(int i=0;i<n;i++)
	cin>>bt[i];
	cout<<"enter arrival times : ";
	int *at=new int[n];
	for(int i=0;i<n;i++)
	cin>>at[i];
	int ct=0;
	int *wt=new int[n];
	int *tat=new int[n];
	cout<<"PID\tAT\tBT\tCT\tWT\tTAT\n";
	    int p=0;
	    int *done=new int[n];
	    for(int i=0;i<n;i++)
	    done[i]=0;
	while(p<n)
	{
				int i=shortest(at,done,n);
```

```
                    wt[i]=ct-at[i];
                    ct+=bt[i];
                    tat[i]=ct-at[i];

    cout<<i<<"\t"<<at[i]<<"\t"<<bt[i]<<"\t"<<ct<<"\t"<<wt[i]<<"\t"<<tat[i]<<endl;p++;
                    at[i]=99999;
                    done[i]=1;
    }
  float awt,atat;awt=atat=0.0;
  for(int i=0;i<n;i++)
  {
  awt+=wt[i];
  atat+=tat[i];
  }
  awt=float(awt/n);
  atat=float(atat/n);
  cout<<"average waiting time: "<<awt<<endl;
  cout<<"average turn around time : "<<atat<<endl;
}
```

Output:


sekhar@sekhar-Inspiron-3542: ~/OSLAB/record
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ nano 3a_fcfs.cpp
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ g++ 3a_fcfs.cpp
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ ./a.out
enter no of processes :5
enter burst times : 5 3 4 6 1
enter arrival times : 0 1 2 3 4
PID AT BT CT WT TAT
0 0 5 5 0 5
1 1 3 8 4 7
2 2 4 12 6 10
3 3 6 18 9 15
4 4 1 19 14 15
average waiting time: 6.6
average turn around time : 10.4
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$

11. SJF(Shortest Job First)

Algorithm:

1. Start the process
2. Declare the array size
3. Get the number of elements to be inserted
4. Select the process which have shortest burst will execute first
5. If two process have same burst length then FCFS scheduling algorithm used
6. Make the average waiting the length of next process
7. Start with the first process from it's selection as above and let other process to be in queue
6. Calculate the total number of burst time
7. Display the values
8. Stop the process

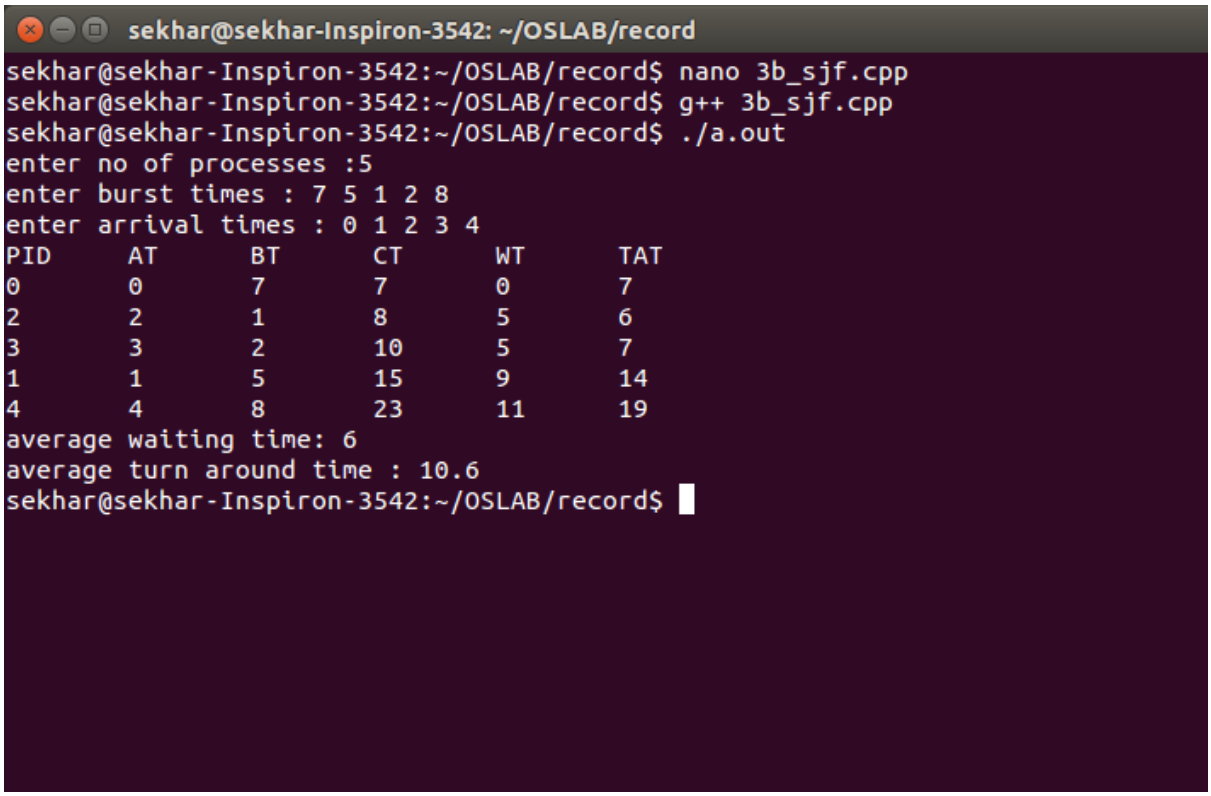
Program:

```
#include<iostream>
using namespace std;
int shortest(int bt[],int done[],int n,int ct,int at[])
{
int k=0;
        for(int i=0;i<n;i++)
        {
                if(bt[k]>bt[i]&&at[i]<=ct)
                k=i;
                else if(bt[k]==bt[i]&&i<k&&at[i]<=ct)
                k=i;
```

```
        }
return k;
}
int main()
{
        int n;
        cout<<"enter no of processes :";
        cin>>n;
        int *bt=new int[n];
        cout<<"enter burst times : ";
        for(int i=0;i<n;i++)
        cin>>bt[i];
        cout<<"enter arrival times : ";
        int *at=new int[n];
        for(int i=0;i<n;i++)
        cin>>at[i];
        int ct=0;
        int *wt=new int[n];
        int *tat=new int[n];
        cout<<"PID\tAT\tBT\tCT\tWT\tTAT\n";
        int p=0;
        int *done=new int[n];
        for(int i=0;i<n;i++)
        done[i]=0;
```

```
	while(p<n)
	{
		int k=shortest(bt,done,n,ct,at);
		wt[k]=ct-at[k];
		ct+=bt[k];
		tat[k]=ct-at[k];

	cout<<k<<"\t"<<at[k]<<"\t"<<bt[k]<<"\t"<<ct<<"\t"<<wt[k]<<"\t"<<tat[k]<<endl;
		bt[k]=999999;
		done[k]=1;p++;

	}
	float awt,atat;awt=atat=0.0;
	for(int i=0;i<n;i++)
	{
	awt+=wt[i];
	atat+=tat[i];
	}
	awt=float(awt/n);
	atat=float(atat/n);
	cout<<"average waiting time: "<<awt<<endl;
	cout<<"average turn around time : "<<atat<<endl;
}
```

Output:


sekhar@sekhar-Inspiron-3542: ~/OSLAB/record
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ nano 3b_sjf.cpp
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ g++ 3b_sjf.cpp
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ ./a.out
enter no of processes :5
enter burst times : 7 5 1 2 8
enter arrival times : 0 1 2 3 4
PID AT BT CT WT TAT
0 0 7 7 0 7
2 2 1 8 5 6
3 3 2 10 5 7
1 1 5 15 9 14
4 4 8 23 11 19
average waiting time: 6
average turn around time : 10.6
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$

12. Round Robin

Algorithm:

1. The queue structure in ready queue is of First In First Out (FIFO) type.

2. A fixed time is allotted to every process that arrives in the queue. This fixed time is known as time slice or time quantum.

3. The first process that arrives is selected and sent to the processor for execution. If it is not able to complete its execution within the time quantum provided, then an interrupt is generated using an automated timer.

4. The process is then stopped and is sent back at the end of the queue. However, the state is saved and context is thereby stored in memory. This helps the process to resume from the point where it was interrupted.

5. The scheduler selects another process from the ready queue and dispatches it to the processor for its execution. It is executed until the time Quantum does not exceed.

6. The same steps are repeated until all the process are finished.

The round robin algorithm is simple and the overhead in decision making is very low. It is the best scheduling algorithm for achieving better and evenly distributed response time.

Program:

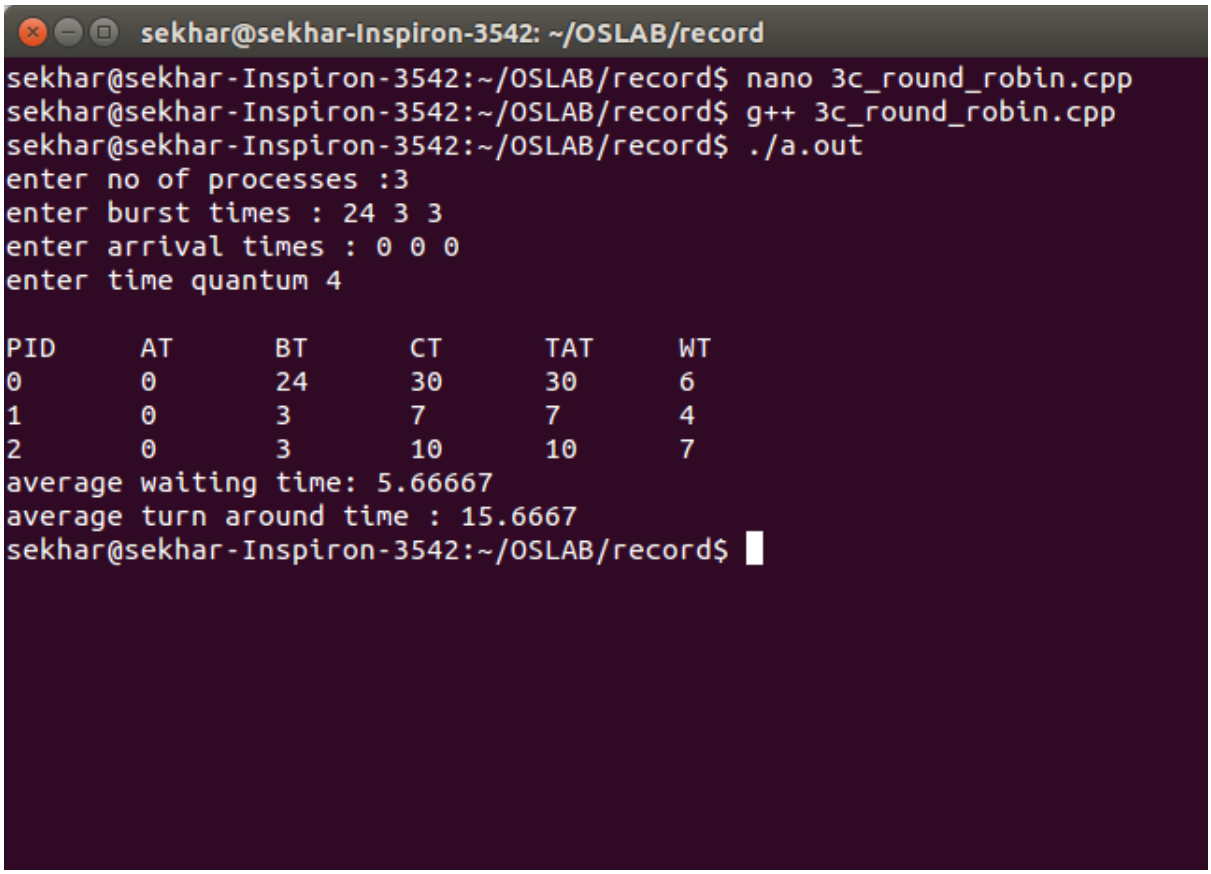
```
#include<iostream>

using namespace std;

int main()

{

        int n;

    cout<<"enter no of processes :";

    cin>>n;

    int *bt=new int[n];

    cout<<"enter burst times : ";

    for(int i=0;i<n;i++)
```

```
cin>>bt[i];
cout<<"enter arrival times : ";
int *at=new int[n];
for(int i=0;i<n;i++)
cin>>at[i];
int ct=0;
int *wt=new int[n];
int *tat=new int[n];
    int *cta=new int[n];
    int tq;
    cout<<"enter time quantum ";
    cin>>tq;
    int *temp=new int[n];
    for(int i=0;i<n;i++)
    temp[i]=bt[i];
    int flag=0;
    while(flag<n)
    {
            for(int i=0;i<n;i++)
            {
                    if(at[i]<=ct&&bt[i]!=0)
                    {
                    if(bt[i]<tq)
                    {
                    ct+=bt[i];
                    bt[i]=0;
```

```
                    cta[i]=ct;
                    tat[i]=cta[i]-at[i];
                    wt[i]=tat[i]-temp[i];
                    flag++;
                    }
                    else
                    {
                    ct+=tq;
                    bt[i]-=tq;
                    if(bt[i]==0)
                    {
                    flag++;
                    cta[i]=ct;
                    tat[i]=cta[i]-at[i];
                    wt[i]=tat[i]-temp[i];
                    }
                    }
                    }
          }
}
cout<<"\nPID\tAT\tBT\tCT\tTAT\tWT";
for(int i=0;i<n;i++)
{
cout<<endl<<i<<"\t"<<at[i]<<"\t"<<temp[i]<<"\t"<<cta[i]<<"\t"<<tat[i]<<"\t"<<wt[i];
}
cout<<endl;
```

```
        float awt,atat;awt=atat=0.0;
    for(int i=0;i<n;i++)
    {
    awt+=wt[i];
    atat+=tat[i];
    }
    awt=float(awt/n);
    atat=float(atat/n);
    cout<<"average waiting time: "<<awt<<endl;
    cout<<"average turn around time : "<<atat<<endl;
}
```

Output:



Bankers Algorithm

Deadlock Detection & Deadlock Avoidance

13. Resource Allocation Algorithm:

P puts request vector

P_i $Request_i$

1. if $Request_i$<$Need_i$ then goto 2 else error

2. if $Request_i$<Available then goto 3 else wait

3. Available=Available-$Request_i$

   $Allocation_i$=$Allocation_i$-$Request_i$

   $Need_i$=$Need_i$-$Request_i$

4. Check if this new state is safe and if safe sequence exists.

Program:

```
#include<iostream>
#include<stdlib.h>
#include<stdbool.h>
using namespace std;
void display(int **a,int n,int m)
{
        for(int i=0;i<n;i++)
        {
        cout<<endl;
        for(int j=0;j<m;j++)
        cout<<a[i][j]<<"  ";
```

```
            }
}
bool isSmall(int *a,int *b,int n)
{
      int flag=0;
      for(int i=0;i<n;i++)
      {
      if(a[i]>b[i])
      {
      flag=1;
      break;
      }
      }
      if(flag==0)
      return true;
      else
      return false;
}

int main()
{
            cout<<"\nenter no of variety of resources : ";
            int nr;
            cin>>nr;
            cout<<"enter the instances of resources : ";
            int *r=new int[nr];
```

```
    for(int i=0;i<nr;i++)
    cin>>r[i];
    cout<<"enter the no of processes : ";
    int p;
    cin>>p;
    cout<<"enter the allocation matrix : ";
    int **am=new int*[p];
    for(int i=0;i<p;i++)
    am[i]=new int[nr];
    for(int i=0;i<p;i++)
    for(int j=0;j<nr;j++)
    cin>>am[i][j];
    cout<<"enter the max matrix : ";
int **mm=new int*[p];
for(int i=0;i<p;i++)
mm[i]=new int[nr];
for(int i=0;i<p;i++)
for(int j=0;j<nr;j++)
cin>>mm[i][j];
int **nm=new int*[p];
for(int i=0;i<p;i++)
nm[i]=new int[nr];
for(int i=0;i<p;i++)
for(int j=0;j<nr;j++)
nm[i][j]=mm[i][j]-am[i][j];
    display(nm,p,nr);
```

```
int *avai=new int[nr];
int *temp=new int[nr];
cout<<"\nenter the process that is requesting :";
int rp;
cin>>rp;
cout<<"\nenter the request : ";
int *request=new int[nr];
for(int i=0;i<nr;i++)
cin>>request[i];
if(isSmall(request,nm[rp],nr)&&isSmall(request,avai,nr))
{
        for(int i=0;i<nr;i++)
        {
        avai[i]-=request[i];
        nm[rp][i]-=request[i];
        am[rp][i]+=request[i];
        }

}
for(int i=0;i<nr;i++)
{
int sum=0;
for(int j=0;j<p;j++)
{
sum+=am[j][i];
}
```

```
temp[i]=sum;
}
int *work=new int[nr];
for(int i=0;i<nr;i++)
{
avai[i]=r[i]-temp[i];
work[i]=avai[i];
}
bool *finish=new bool[p];
for(int i=0;i<p;i++)
finish[i]=false;
int trap=0;
int count=0;
int disp=1;
//cout<<"\nsafe sequence :";
while(trap<p)
{
        if(count>p*p)
        {cout<<"unsafe"<<endl;
                break;
        }

        for(int i=0;i<p;i++)
        {count++;
                if(finish[i]==false&&isSmall(nm[i],work,nr))
                {
```

```
                    if(disp==1)
                    cout<<"\nsafe sequence :";
                    disp++;
                    for(int j=0;j<nr;j++)
                    work[j]=work[j]+am[i][j];
                    trap++;
                    finish[i]=true;
                    cout<<i<<"  ";
                }

            }
        }

}
```

Output:

```
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ nano 4b_banker_request.cpp
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ g++ 4b_banker_request.cpp
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ ./a.out

enter no of variety of resources : 3
enter the instances of resources : 10 5 7
enter the no of processes : 5
enter the allocation matrix : 0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
enter the max matrix : 7 5 3
3 2 2
9 0 2
2 2 2
4 3 3

7  4  3
1  2  2
6  0  0
0  1  1
4  3  1
enter the process that is requesting :1

enter the request : 1 0 2

safe sequence :1  3  4  0  2  sekhar@sekhar-Inspiron-3542:~/OSLAB/record$
```

```
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ g++ 4b_banker_request.cpp
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ ./a.out

enter no of variety of resources : 3
enter the instances of resources : 7 2 6
enter the no of processes : 5
enter the allocation matrix :
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
enter the max matrix :
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3

7  4  3
1  2  2
6  0  0
0  1  1
4  3  1
enter the process that is requesting :2

enter the request : 2 0 2
unsafe
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ 3~
```

14. Safety Algorithm:

1. Work=Available

    Finish=false

2. find p1 such that

    Finish[i]=false

    $Need_i$<Work if no goto step 4

3. Work=Work+Allocation

    Finish[i]=true

    go to step 2

4. if Finish[i]=true for all i

    Then system is safe

Program:

```
#include<iostream>
#include<stdlib.h>
#include<stdbool.h>
using namespace std;
void display(int **a,int n,int m)
{
    for(int i=0;i<n;i++)
    {
    cout<<endl;
    for(int j=0;j<m;j++)
    cout<<a[i][j]<<"  ";
    }
}
```

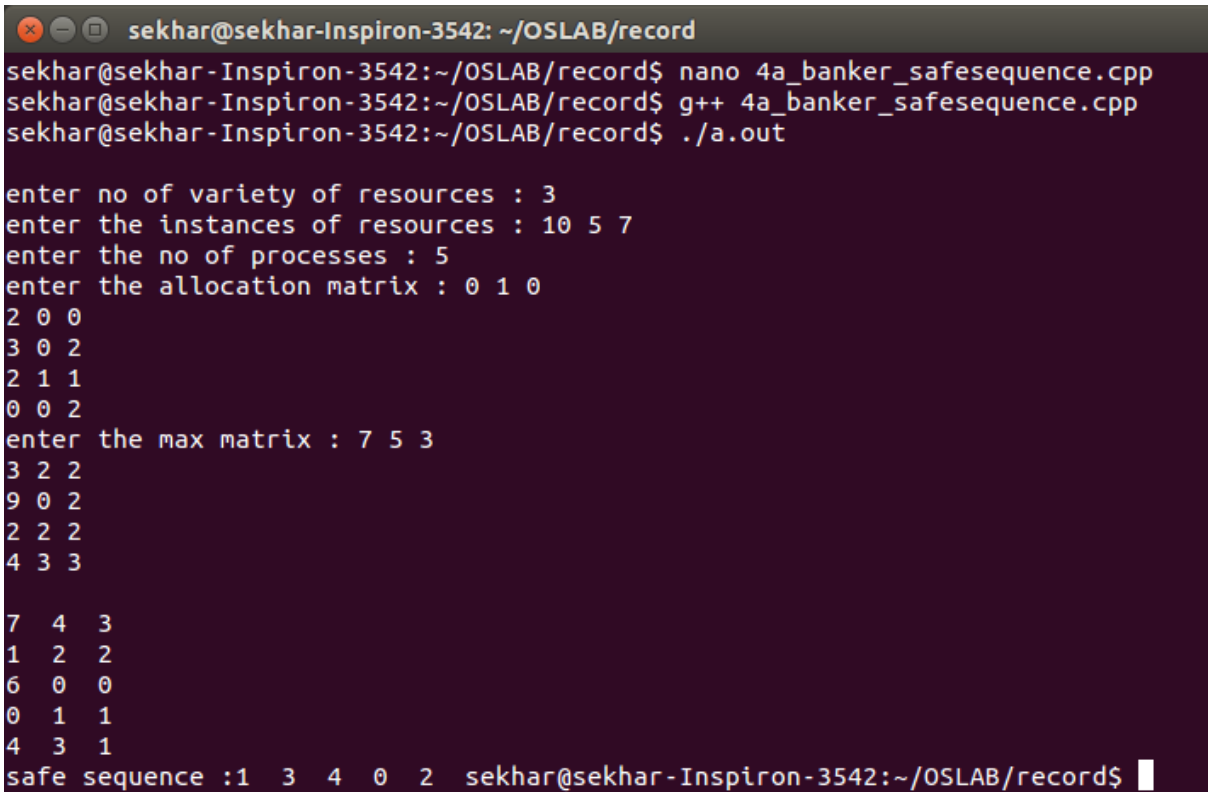
```
bool isSmall(int *a,int *b,int n)
{
	int flag=0;
	for(int i=0;i<n;i++)
	{
	if(a[i]>b[i])
	{
	flag=1;
	break;
	}
	}
	if(flag==0)
	return true;
	else
	return false;
}

int main()
{
		cout<<"\nenter no of variety of resources : ";
		int nr;
		cin>>nr;
		cout<<"enter the instances of resources : ";
		int *r=new int[nr];
		for(int i=0;i<nr;i++)
		cin>>r[i];
```

```
    cout<<"enter the no of processes : ";
    int p;
    cin>>p;
    cout<<"enter the allocation matrix : ";
    int **am=new int*[p];
    for(int i=0;i<p;i++)
    am[i]=new int[nr];
    for(int i=0;i<p;i++)
    for(int j=0;j<nr;j++)
    cin>>am[i][j];
    cout<<"enter the max matrix : ";
int **mm=new int*[p];
for(int i=0;i<p;i++)
mm[i]=new int[nr];
for(int i=0;i<p;i++)
for(int j=0;j<nr;j++)
cin>>mm[i][j];
int **nm=new int*[p];
for(int i=0;i<p;i++)
nm[i]=new int[nr];
for(int i=0;i<p;i++)
for(int j=0;j<nr;j++)
nm[i][j]=mm[i][j]-am[i][j];
    display(nm,p,nr);
    int *avai=new int[nr];
    int *temp=new int[nr];
```

```
for(int i=0;i<nr;i++)
{
int sum=0;
for(int j=0;j<p;j++)
{
sum+=am[j][i];
}
temp[i]=sum;
}
int *work=new int[nr];
for(int i=0;i<nr;i++)
{
avai[i]=r[i]-temp[i];
work[i]=avai[i];
}
bool *finish=new bool[p];
for(int i=0;i<p;i++)
finish[i]=false;
int trap=0;
cout<<"\nsafe sequence :";
while(trap<p)
{
        for(int i=0;i<p;i++)
        {
                if(finish[i]==false&&isSmall(nm[i],work,nr))
                {
```

```
                                for(int j=0;j<nr;j++)

                                work[j]=work[j]+am[i][j];

                                trap++;

                                finish[i]=true;

                                cout<<i<<" ";

                        }

                }

        }

}
```

Output:



PAGE REPLACEMENT ALGORITHMS

15. Program to implement page replacement using FIFO algorithm

Algorithm

1. Start the process

2. Declare the size with respect to page length

3. Check the need of replacement from the page to memory

4. Check the need of replacement from old page to new page in memory

5. Form a queue to hold all pages

6. Insert the page require memory into the queue

7. Check for bad replacement and page fault

8. Get the number of processes to be inserted

9. Display the values

10. Stop the process

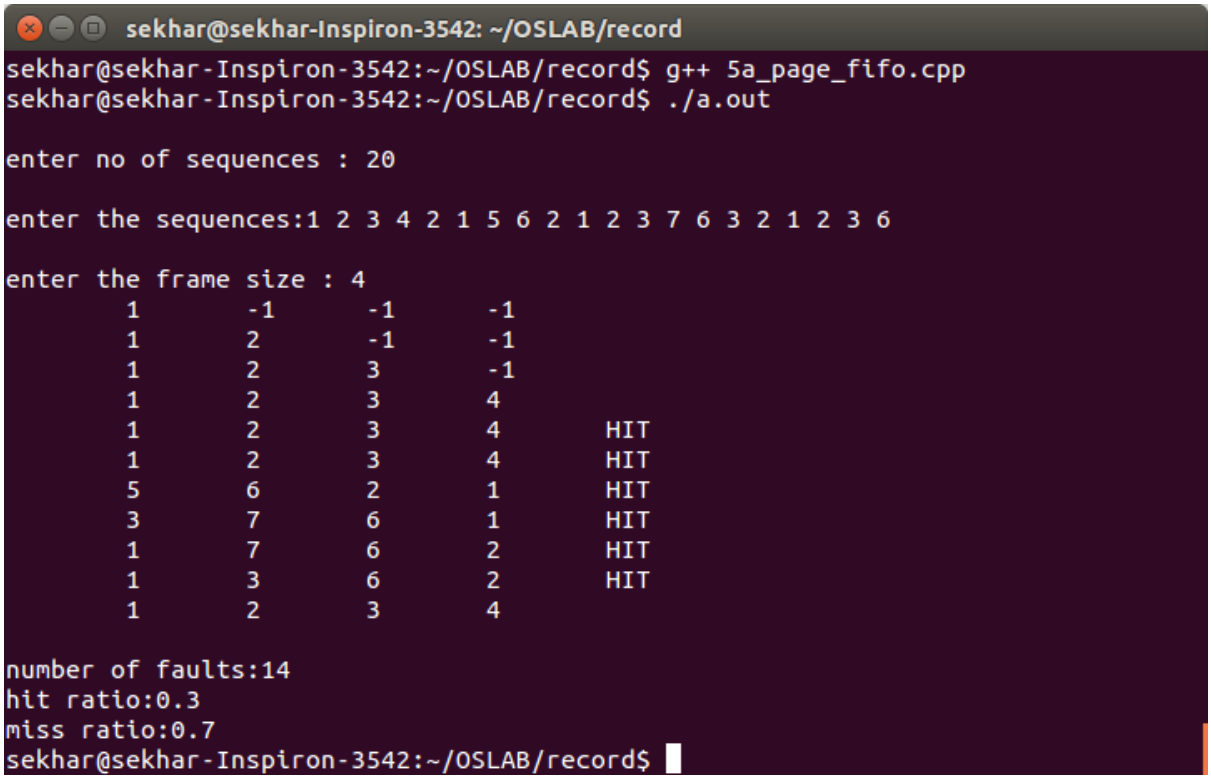
Program:

```
#include<iostream>
#include<stdbool.h>
using namespace std;
void display(int *s,int n,bool flag)
{
        for(int i=0;i<n;i++)
        {
        cout<<'\t'<<s[i];
        }
        if(flag)
```

```
        cout<<"\tHIT"<<endl;
        else
        cout<<endl;
}
int main()
{
        cout<<"\nenter no of sequences : ";
        int n;
        cin>>n;
        int *s=new int[n];
        cout<<"\nenter the sequences:";
        for(int i=0;i<n;i++)
        {
                cin>>s[i];
        }
        int f;
        cout<<"\nenter the frame size : ";
        cin>>f;
        int *fr=new int[f];
        for(int i=0;i<f;i++)
        fr[i]=-1;
        int size=0;int hit=0;int insert=0;int flag;
        for(int i=0;i<n;i++)
        {
                insert=0;flag=0;
                for(int j=0;j<f;j++)
```

```
            {
            if(fr[j]==s[i])
            {j=f;hit++;flag=1;display(fr,f,true);}
            if(fr[j]==-1)
            {fr[j]=s[i];insert=1;size++;j=f;display(fr,f,false);}
            }
            if(insert==1)
            continue;
            else if(flag==0)
            {
            fr[f-size]=s[i];
            size--;
            if(size==0)
            size=f;
            }
        }
        for(int i=0;i<f;i++)
        cout<<'\t'<<s[i];
        cout<<endl;
        cout<<"\nnumber of faults:"<<(n-hit);
        float temp=float(hit)/float(n);
        cout<<"\nhit ratio:"<<temp;
        cout<<"\nmiss ratio:"<<(1.0-temp)<<endl;

}
```

Output:


sekhar@sekhar-Inspiron-3542: ~/OSLAB/record
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ g++ 5a_page_fifo.cpp
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ ./a.out
enter no of sequences : 20
enter the sequences:1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6
enter the frame size : 4
1 -1 -1 -1
1 2 -1 -1
1 2 3 -1
1 2 3 4
1 2 3 4 HIT
1 2 3 4 HIT
5 6 2 1 HIT
3 7 6 1 HIT
1 7 6 2 HIT
1 3 6 2 HIT
1 2 3 4
number of faults:14
hit ratio:0.3
miss ratio:0.7
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$

16.Program to implement Page replacement algorithm using LRU algorithm.(Least Recently Used)

Program:-

```
#include<iostream>
using namespace std;
int main()
 {
        int nop,nof,page[20],i,count=0;
        cout<<"\n\tEnter the No. of Pages:";
        cin>>nop;  //Store the no of Pages
        cout<<"\n\t Enter the Reference String:";
        for(i=0;i<nop;i++)
        {
        cout<<"\t";
        cin>>page[i]; //Store the pages
        }

        cout<<"\n\t Enter the No of frames:-";
        cin>>nof;
        int frame[nof],fcount[nof];
        for(i=0;i<nof;i++)
        {
        frame[i]=-1; //Store the frames
        fcount[i]=0; //Track when the page is last used
        }
        i=0;
```

```
while(i<nop)
{
 int j=0,flag=0;
 while(j<nof)
 {
   if(page[i]==frame[j]){  //Checking whether page already exist in frames or not
   flag=1;
   fcount[j]=i+1;
   }
   j++;
 }
 j=0;
 cout<<"\n\t*************************************\n";
 cout<<"\t"<<page[i]<<"-->";
 if(flag==0)
 {
  int min=0,k=0;
   while(k<nof-1)
    {

      if(fcount[min]>fcount[k+1]) //Calculating the page which is least recently used
        min=k+1;
      k++;
    }
   frame[min]=page[i]; //Replacing it
   fcount[min]=i+1;    //Increasing the time
```

```
            count++;          //counting Page Fault



           while(j<nof)
           {
              cout<<"\t|"<<frame[j]<<"|";
              j++;
           }


            }
          i++;


        }
        cout<<"\n\t************************************\n";
        cout<<"\n\tPage Fault is:"<<count;
           float hitratio=float(count)/nop;
           float missratio=1.0-hitratio;
           cout<<"\n\thit ratio:"<<hitratio;
           cout<<"\n\tmiss ratio:"<<missratio<<endl;
        return 0;
}
```

Output:-

```
sekhar@sekhar-Inspiron-3542:~/OSLAB/cse2-91$ nano page_lru.cpp
sekhar@sekhar-Inspiron-3542:~/OSLAB/cse2-91$ g++ page_lru.cpp
sekhar@sekhar-Inspiron-3542:~/OSLAB/cse2-91$ ./a.out

        Enter the No. of Pages:20

         Enter the Reference String:    1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 1 2 3 6

         Enter the No of frames:-4

        ***************************************
        1-->    |1|     |-1|    |-1|    |-1|
        ***************************************
        2-->    |1|     |2|     |-1|    |-1|
        ***************************************
        3-->    |1|     |2|     |3|     |-1|
        ***************************************
        4-->    |1|     |2|     |3|     |4|
        ***************************************
        2-->
        ***************************************
        1-->
        ***************************************
        5-->    |1|     |2|     |5|     |4|
        ***************************************
        6-->    |1|     |2|     |5|     |6|
        ***************************************
        2-->
        ***************************************
        1-->
        ***************************************
        2-->
        ***************************************
        3-->    |1|     |2|     |3|     |6|
        ***************************************
        7-->    |1|     |2|     |3|     |7|
        ***************************************
        6-->    |6|     |2|     |3|     |7|
        ***************************************
        3-->
        ***************************************
        2-->
        ***************************************
        1-->    |6|     |2|     |3|     |1|
        ***************************************
        2-->
        ***************************************
        3-->
        ***************************************
        6-->
        ***************************************

        Page Fault is:10
        hit ratio:0.5
        miss ratio:0.5
sekhar@sekhar-Inspiron-3542:~/OSLAB/cse2-91$
```

ECHO SERVER USING PIPES

17. Echo server using pipes

System calls used:

Pipe():

Pipe,pipe2 – create pipe

Header files required:

#include<unistd.h>

Syntax:

int pipe(int pipefd[2]);

#define _GNU_SOURCE

Header files required:

#include<fcntl.h>

#include<unitsd.h>

Syntax:

int pipe2(int pipefd[2],int flags);

Description:

Pipe() creates a pipe,a unidirectional data channel that can be used for interprocess communication. The array pipefd is used to reform a file descriptor referring to the ends of the pipe.

Pipefd[0] refers to the read end of the pipe.

Pipefd[1] refers to the write end of the pipe.

If flag is 0,then pipe() is same as pipe()

Return value:

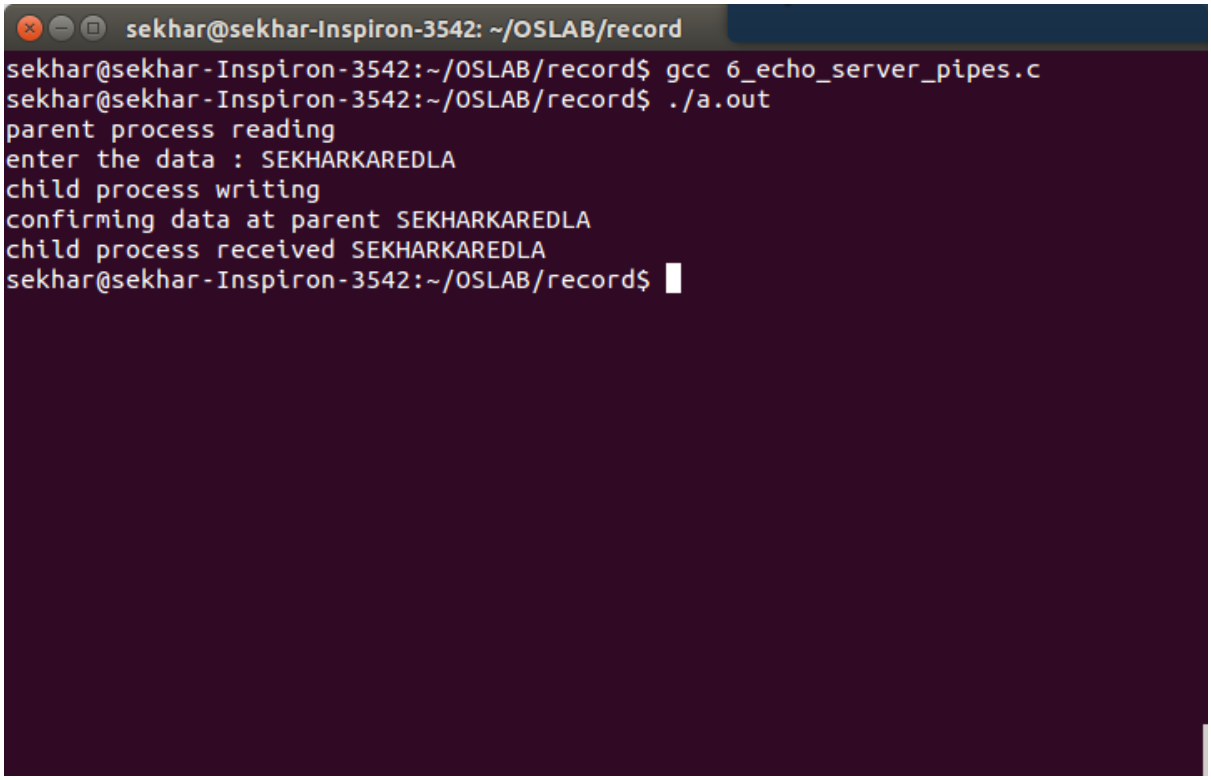
On success, zero is returned, on error -1 is returned.

Program:

```
#include<stdlib.h>
#include<stdio.h>
#include<unistd.h>
void main(int argc,char *argv[])
{
	int fd1[2],fd2[2];
	pipe(fd1);
	pipe(fd2);
	pid_t pid;
	char s[]="sekhar karedla";
	    char s1[100];
	pid=fork();
	char buf[100];
	if(pid<0)
	{
	printf("error");
	exit(-1);
	}
	else if(pid==0)
	{
	    printf("enter the data : ");
	    scanf("%s",s1);
	printf("child process writing\n");
	close(fd1[0]);
	write(fd1[1],s1,sizeof(s1));
```

```
        wait(NULL);
        close(fd2[1]);
        read(fd2[0],buf,sizeof(buf));
        printf("child process received %s\n",buf);
    exit(0);
    }
    else
    {
    //buf[0]=' ';
    close(fd1[1]);
    //wait(NULL);
    printf("parent process reading\n");
    read(fd1[0],buf,sizeof(buf));
    printf("confirming data at parent %s\n",buf);
        close(fd2[0]);
        write(fd2[1],buf,sizeof(buf));
        exit(0);
    }
}
```

Output:


sekhar@sekhar-Inspiron-3542: ~/OSLAB/record
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ gcc 6_echo_server_pipes.c
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ ./a.out
parent process reading
enter the data : SEKHARKAREDLA
child process writing
confirming data at parent SEKHARKAREDLA
child process received SEKHARKAREDLA
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$

Named Pipe

18.Program of reader and writer in Named pipe.

A named pipe works much like a regular pipe, but does have some noticeable differences.

- Named pipes exist as a device special file in the file system.
- Processes of different ancestry can share data through a named pipe.
- When all I/O is done by sharing processes, the named pipe remains in the file system for later use.

There are several ways of creating a named pipe. The first two can be done directly from the shell.

```
mknod MYFIFO p

mkfifo a=rw MYFIFO
```

The above two commands perform identical operations, with one exception. The mkfifo command provides a hook for altering the permissions on the FIFO file directly after creation. With mknod, a quick call to the chmod command will be necessary.

Program(Writer):-

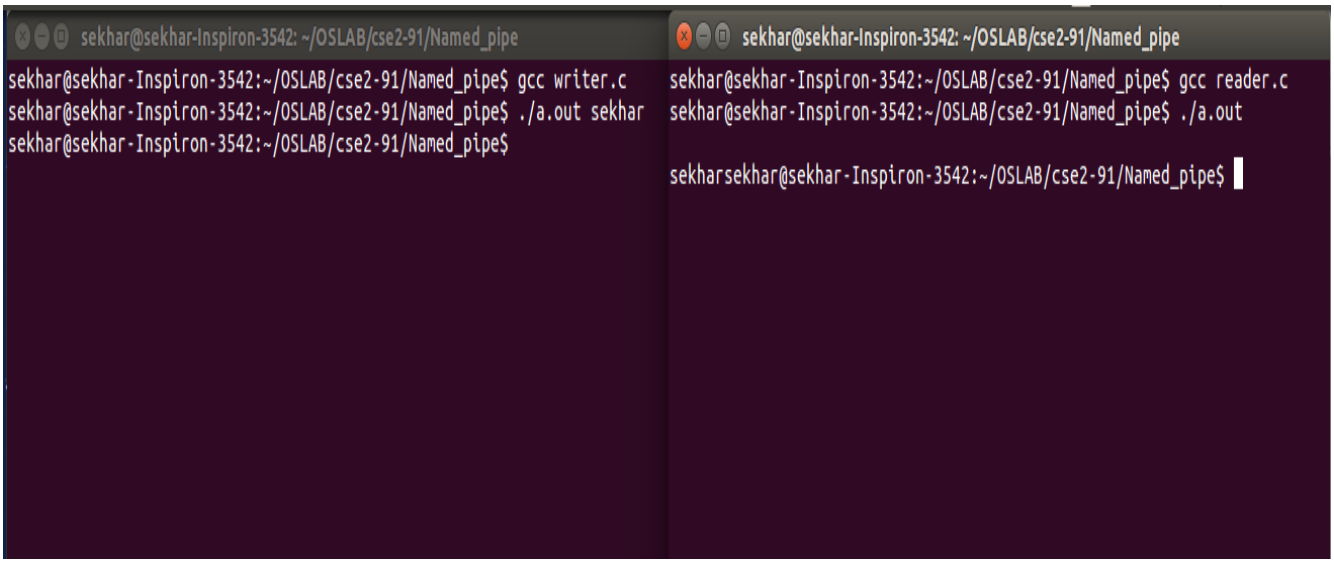
```
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<unistd.h>
#include<fcntl.h>
void main(int argc,char *argv[])
{
        char *s="/tmp/testfifo";
        int fd;
        mkfifo(s,0666);
        fd=open(s,O_WRONLY);
        write(fd,argv[1],sizeof(argv[1]));
        close(fd);
        unlink(s);
```

```
}
```

Program(Reader):-

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<unistd.h>
#include<fcntl.h>
#define BUF 1024
void main(int argc,char *argv[])
{
        char r[BUF];
        char *s="/tmp/testfifo";
        int fd;
        fd=open(s,O_RDONLY);
        read(fd,r,BUF);
        printf("\n%s",r);
        close(fd);
}
```

Output:-


sekhar@sekhar-Inspiron-3542: ~/OSLAB/cse2-91/Named_pipe
sekhar@sekhar-Inspiron-3542:~/OSLAB/cse2-91/Named_pipe$ gcc writer.c
sekhar@sekhar-Inspiron-3542:~/OSLAB/cse2-91/Named_pipe$ ./a.out sekhar
sekhar@sekhar-Inspiron-3542:~/OSLAB/cse2-91/Named_pipe$
sekhar@sekhar-Inspiron-3542: ~/OSLAB/cse2-91/Named_pipe
sekhar@sekhar-Inspiron-3542:~/OSLAB/cse2-91/Named_pipe$ gcc reader.c
sekhar@sekhar-Inspiron-3542:~/OSLAB/cse2-91/Named_pipe$ ./a.out
sekharsekhar@sekhar-Inspiron-3542:~/OSLAB/cse2-91/Named_pipe$

Shell Script

THEORY:-

->WHAT IS SHELL SCRIPTING?

Shells are interactive i.e shell accept command from users and execute them. User can also write series of commands to execute a specific task in a text file and execute them instead of giving commands in command prompt.

-> DECLARE A VARIABLE ?

```
a=0
b=1
```

->SYNTAX:-

if syntax:-

```
if test <condition>
then
        statements
elif test <condition>
then
        statements
else
        statements
fi
```

while syntax:-

```
while test <condition>
do
        statements
done
```

comparison of integers:-

n1 -eq *n2*	Integer *n1* is equal to integer *n2*.
n1 -ge *n2*	Integer *n1* is greater than or equal to integer *n2*.
n1 -gt *n2*	Integer *n1* is greater than integer *n2*.
n1 -le *n2*	Integer *n1* is less than integer *n2*.
n1 -lt *n2*	Integer *n1* is less than or equal to integer *n2*.
n1 -ne *n2*	Integer *n1* is not equal to integer *n2*.

switch case syntax:-

```
case item in
        part1)
                statements
                ;;
        part2)
                statements
                ;;
        part3)
                statements
                ;;
        part4)
                statements
                ;;
esac
```

until loop syntax:-

```
until test <condition>
do
        statements
done
```

for loop syntax:-

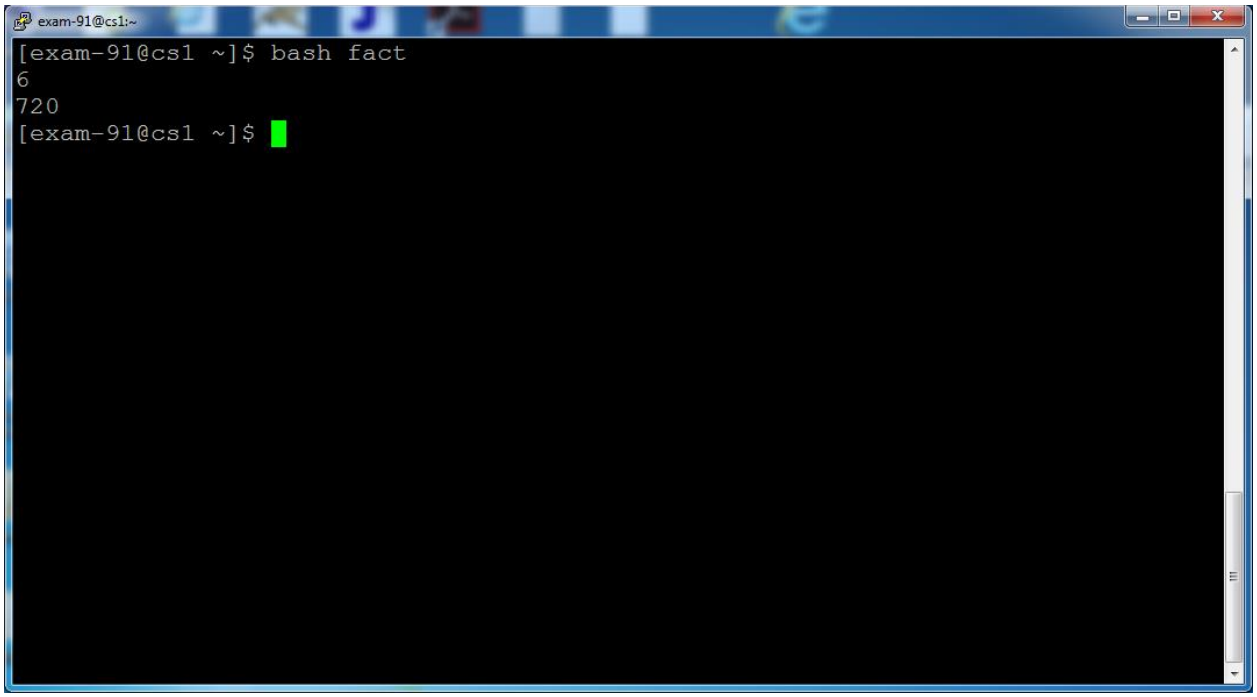
```
for((initialization;condition;modification))
do
        statements
done


ex:-
for((i=0;i<10;i++))
do
        echo "$i"
done
```

19.Program to find the factorial.

Program:-

```
read n

f=1

for((i=1;i<=n;i++))

do

    f=`expr $f \* $i`

done

echo "$f"
```

Output:-

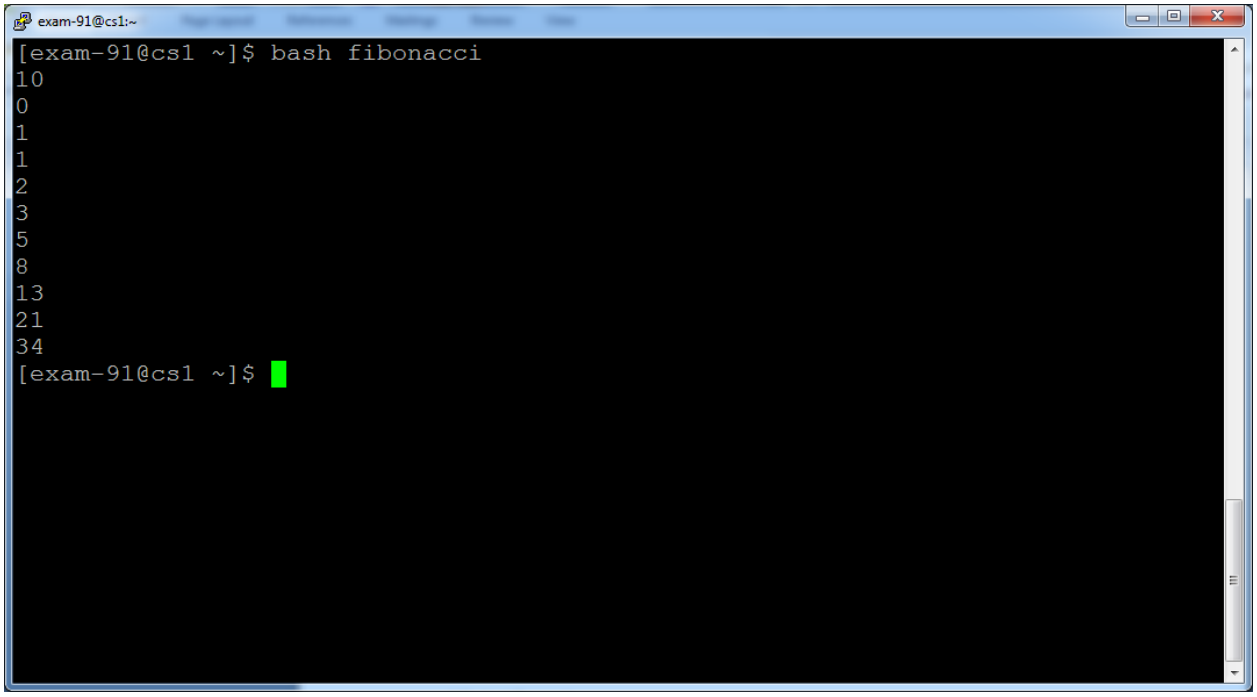


20.Program to print the Fibonacci Series.

Program:-

```
read n
a=0
b=1
c=0
for((i=1;i<=n;i++))
do
    echo "$c"
    a=$b
    b=$c
    c=`expr $a + $b`
done
```

OUTPUT:-

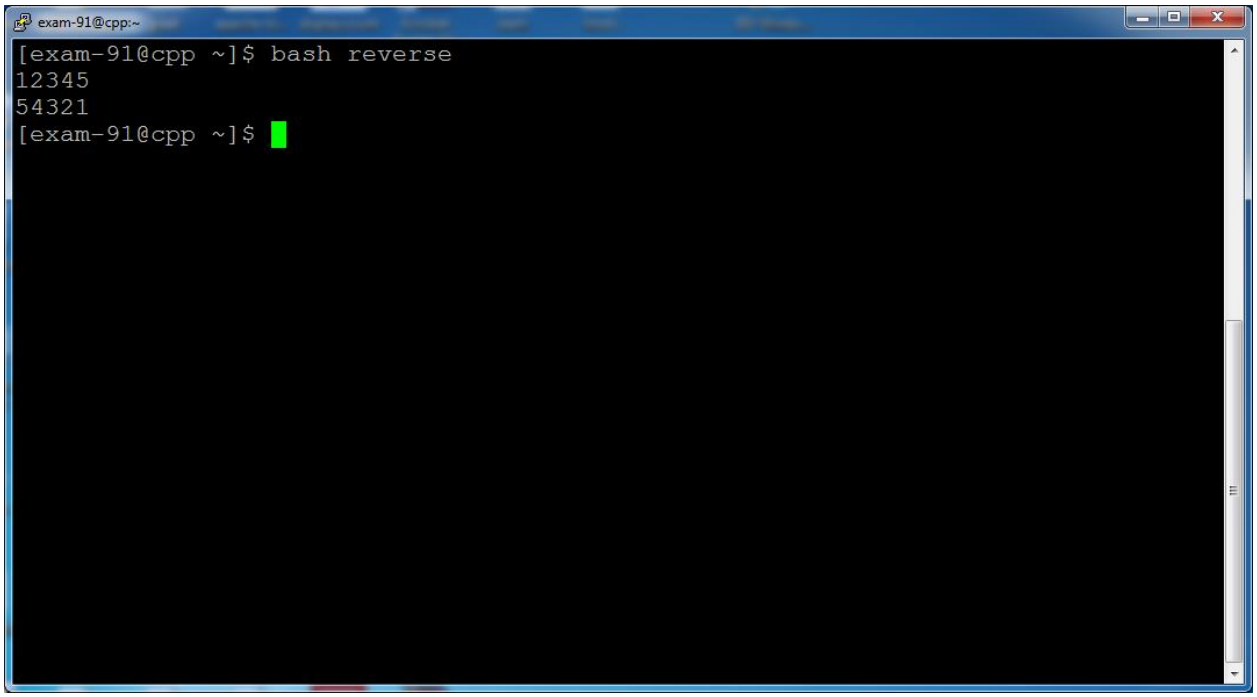


21.Program to find the Reverse of a number.

Program:-

```
read n

s=0

r=0

while test $n -ne 0

do

    r=`expr $n % 10`

    s=`expr $s \* 10`

    s=`expr $s + $r`

    n=`expr $n / 10`

done

echo "$s"
```

OUTPUT:-

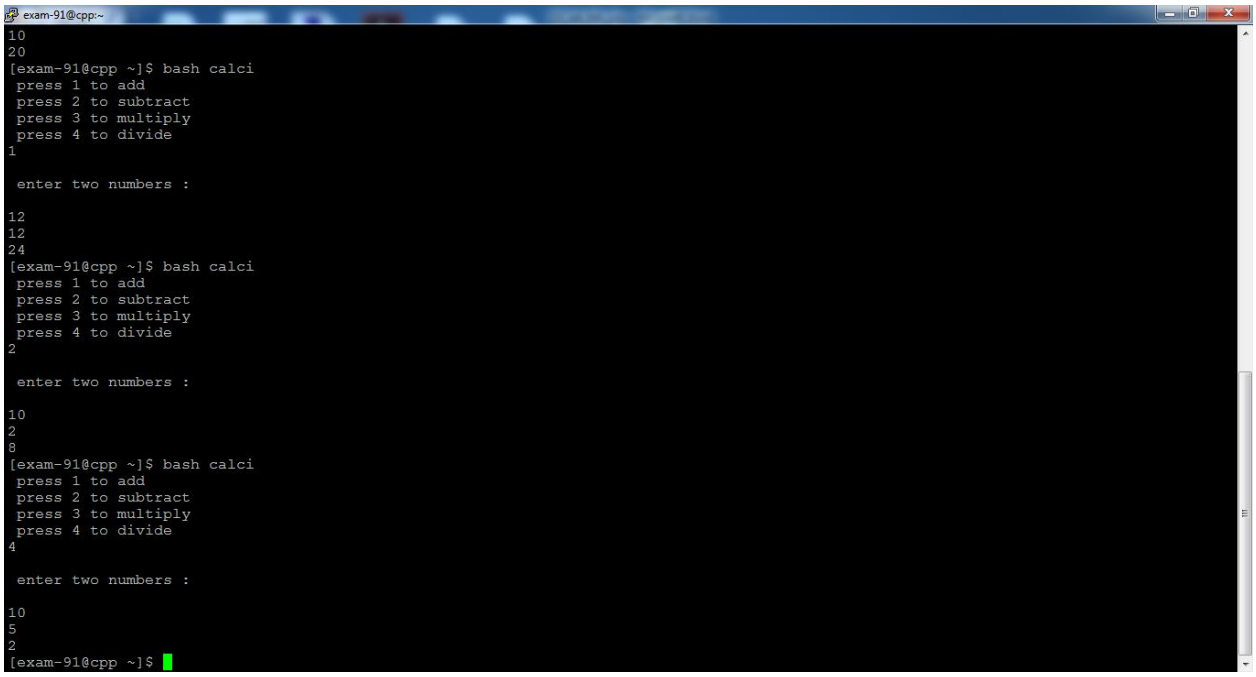


22.Program to simulate simple Calculator.

Program:-

```
cho -e " press 1 to add \n press 2 to subtract \n press 3 to multiply \n press 4 to divide"
read ch
echo -e "\n enter two numbers :\n"
read a
read b
if test $ch -eq 1
then
    expr $a + $b
elif test $ch -eq 2
then
    expr $a - $b
elif test $ch -eq 3
then
    expr $a \* $b
else
    expr $a / $b
fi
```

OUTPUT:-

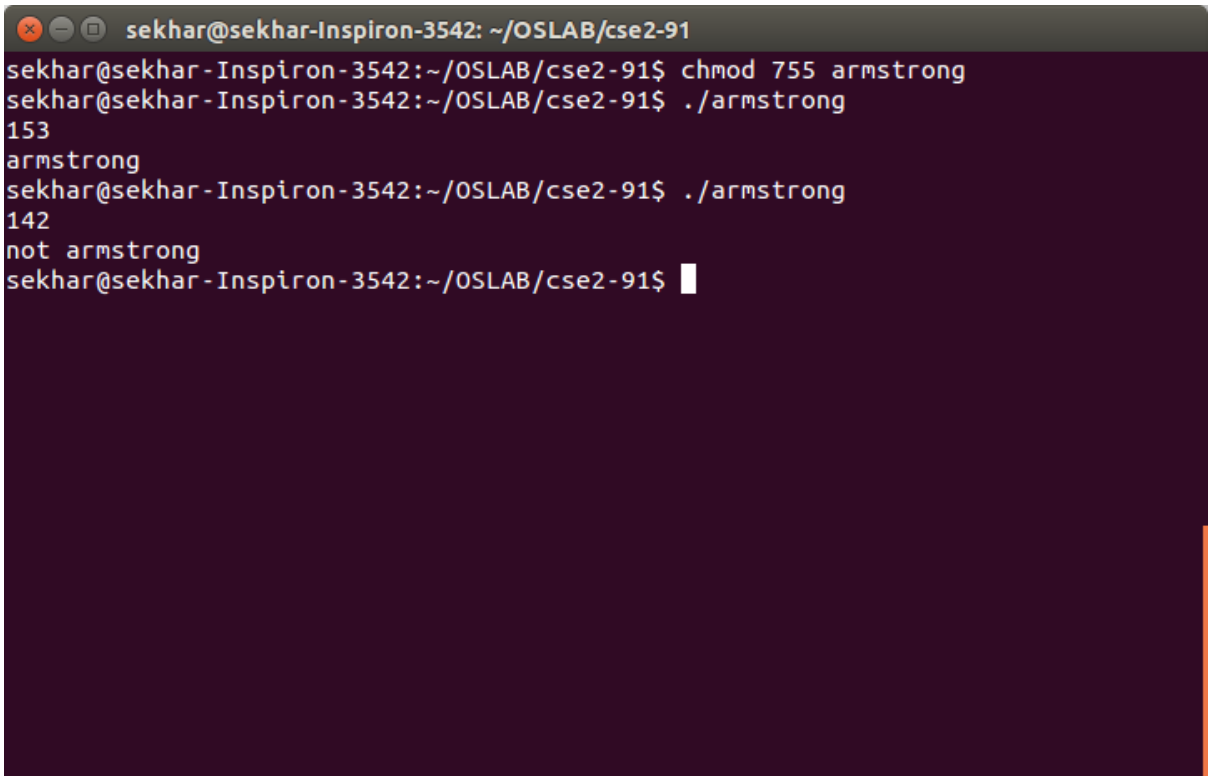

exam-91@cpp:~
10
20
[exam-91@cpp ~]$ bash calci
press 1 to add
press 2 to subtract
press 3 to multiply
press 4 to divide
1
enter two numbers :
12
12
24
[exam-91@cpp ~]$ bash calci
press 1 to add
press 2 to subtract
press 3 to multiply
press 4 to divide
2
enter two numbers :
10
2
8
[exam-91@cpp ~]$ bash calci
press 1 to add
press 2 to subtract
press 3 to multiply
press 4 to divide
4
enter two numbers :
10
5
2
[exam-91@cpp ~]$

23.Program to find whether a number is armstrong or not.

Program:-

```
read n
k=$n
s=0
r=0
ar=0
while test $n -ne 0
do
        r=`expr $n % 10`
        s=`expr $r \* $r`
        s=`expr $s \* $r`
        ar=`expr $ar + $s`
        n=`expr $n / 10`
done
if test $ar -eq $k
then
        echo "armstrong"
else
        echo "not armstrong"
fi
```

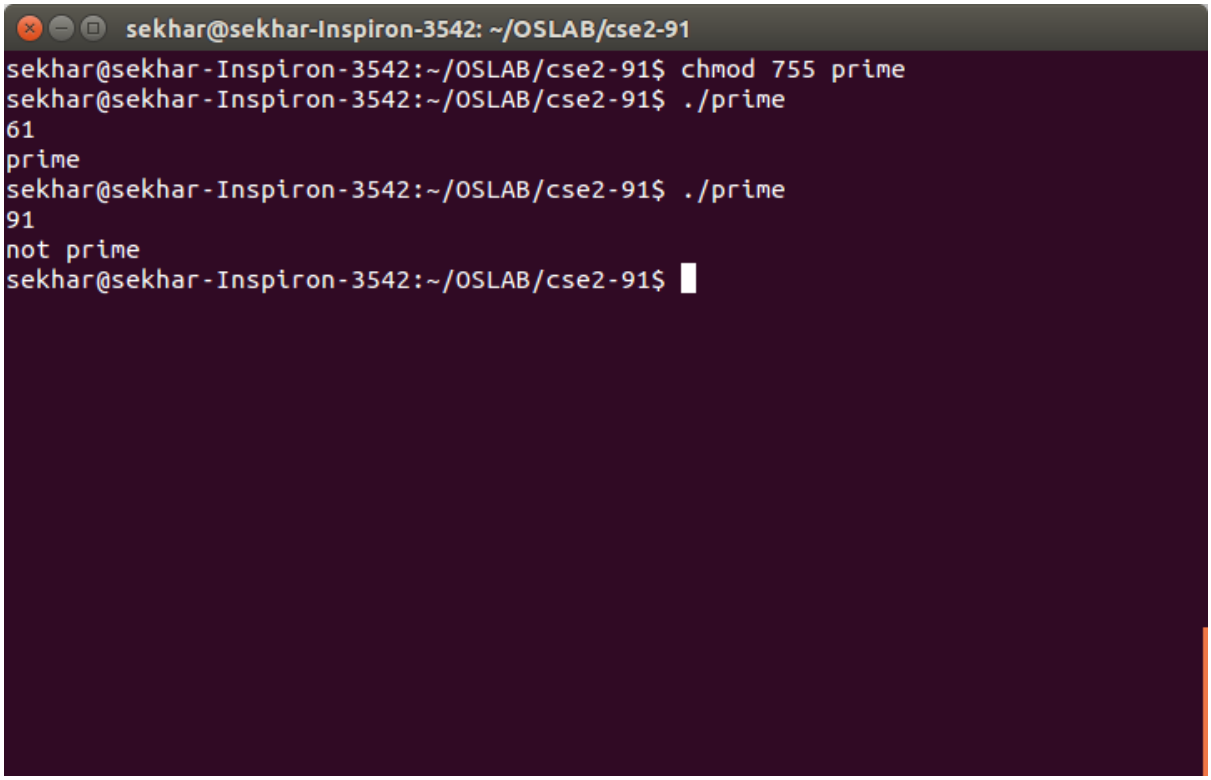
Output:-


sekhar@sekhar-Inspiron-3542: ~/OSLAB/cse2-91
sekhar@sekhar-Inspiron-3542:~/OSLAB/cse2-91$ chmod 755 armstrong
sekhar@sekhar-Inspiron-3542:~/OSLAB/cse2-91$ ./armstrong
153
armstrong
sekhar@sekhar-Inspiron-3542:~/OSLAB/cse2-91$ ./armstrong
142
not armstrong
sekhar@sekhar-Inspiron-3542:~/OSLAB/cse2-91$

24.Program to find whether a number is Prime or not.

```
read n
flag=1
for((i=2;i<n;i++))
do
        m=`expr $n % $i`
        if test $m -eq 0
        then
                flag=0
                #echo "$flag"
        fi
done
if test $flag -eq 0
then
        echo "not prime"
else
        echo "prime"
fi
```

Output:-


sekhar@sekhar-Inspiron-3542: ~/OSLAB/cse2-91
sekhar@sekhar-Inspiron-3542:~/OSLAB/cse2-91$ chmod 755 prime
sekhar@sekhar-Inspiron-3542:~/OSLAB/cse2-91$ ./prime
61
prime
sekhar@sekhar-Inspiron-3542:~/OSLAB/cse2-91$ ./prime
91
not prime
sekhar@sekhar-Inspiron-3542:~/OSLAB/cse2-91$

Threads

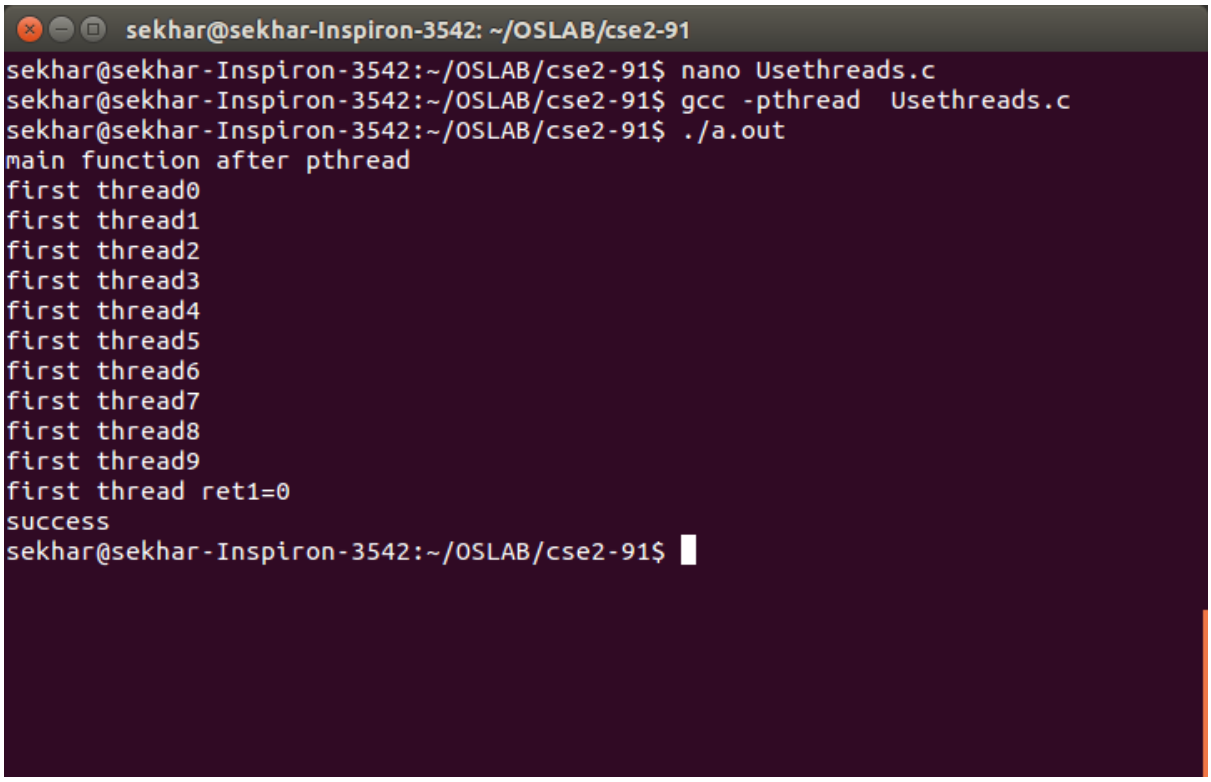
POSIX.1 specifies a set of interfaces (functions, header files) for threaded programming commonly known as POSIX threads, or Pthreads. A single process can contain multiple threads, all of which are executing the same program. These threads share the same global memory (data and heap segments), but each thread has its own stack (automatic variables).

25.Program to demonstrate the use of threads.

```
#include<stdio.h>
#include<string.h>
#include <pthread.h>
void *myfunc (void *mydata);
pthread_t theThread;
main()
{
pthread_t thread1;
char *my1="first thread";
int ret1;
ret1=pthread_create(&thread1,NULL,&myfunc,(void*)my1);
printf("main function after pthread\n");
pthread_join(thread1,NULL);
printf("first thread ret1=%d\n",ret1);
if(pthread_equal(thread1,theThread))
```

```
{
printf("success\n");
}
}

void *myfunc (void *mydata)
{
char *msg;
msg=(char*)mydata;
int i;
for(i=0;i<10;i++)
{
printf("%s%d\n",msg,i);
sleep(1);
theThread=pthread_self();
}
pthread_exit(NULL);
}
```

Output:-


sekhar@sekhar-Inspiron-3542: ~/OSLAB/cse2-91
sekhar@sekhar-Inspiron-3542:~/OSLAB/cse2-91$ nano Usethreads.c
sekhar@sekhar-Inspiron-3542:~/OSLAB/cse2-91$ gcc -pthread Usethreads.c
sekhar@sekhar-Inspiron-3542:~/OSLAB/cse2-91$ ./a.out
main function after pthread
first thread0
first thread1
first thread2
first thread3
first thread4
first thread5
first thread6
first thread7
first thread8
first thread9
first thread ret1=0
success
sekhar@sekhar-Inspiron-3542:~/OSLAB/cse2-91$

26.Another Program to demonstrate the use of threads.

Program:-

```
#include<stdio.h>
#include<string.h>
#include <pthread.h>
void *myfunc (void *mydata);
main()
{
pthread_t thread1,thread2;
char *my1="first thread";
char *my2="second thread";
int ret1,ret2;
ret1=pthread_create(&thread1,NULL,&myfunc,(void*)my1);
ret2=pthread_create(&thread2,NULL,&myfunc,(void*)my2);
printf("main function after pthread\n");
pthread_join(thread1,NULL);
pthread_join(thread2,NULL);
printf("first thread ret1=%d\n",ret1);
printf("second thread ret1=%d\n",ret2);
}

void *myfunc (void *mydata)
{
char *msg;
msg=(char*)mydata;
int i;
```

```
for(i=0;i<10;i++)

{

printf("%s%d\n",msg,i);

sleep(1);

}

return NULL;

}
```

Output:-

```
sekhar@sekhar-Inspiron-3542:~/OSLAB/cse2-91$ nano Usethreads2.c
sekhar@sekhar-Inspiron-3542:~/OSLAB/cse2-91$ gcc -pthread  Usethreads2.c
sekhar@sekhar-Inspiron-3542:~/OSLAB/cse2-91$ ./a.out
first thread0
second thread0
main function after pthread
first thread1
second thread1
first thread2
second thread2
second thread3
first thread3
first thread4
second thread4
first thread5
second thread5
second thread6
first thread6
second thread7
first thread7
second thread8
first thread8
second thread9
first thread9
first thread ret1=0
second thread ret1=0
sekhar@sekhar-Inspiron-3542:~/OSLAB/cse2-91$
```

27.Program to demonstrate the stat function.

Header Files:

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

```
struct stat {
    dev_t     st_dev;       /* ID of device containing file */
    ino_t     st_ino;       /* inode number */
    mode_t    st_mode;      /* file type and mode */
    nlink_t   st_nlink;     /* number of hard links */
    uid_t     st_uid;       /* user ID of owner */
    gid_t     st_gid;       /* group ID of owner */
    dev_t     st_rdev;      /* device ID (if special file) */
    off_t     st_size;      /* total size, in bytes */
    blksize_t st_blksize;   /* blocksize for filesystem I/O */
    blkcnt_t  st_blocks;    /* number of 512B blocks allocated */

    /* Since Linux 2.6, the kernel supports nanosecond
      precision for the following timestamp fields.
      For the details before Linux 2.6, see NOTES. */

    struct timespec st_atim;  /* time of last access */
    struct timespec st_mtim;  /* time of last modification */
    struct timespec st_ctim;  /* time of last status change */
```

```
#define st_atime st_atim.tv_sec      /* Backward compatibility */
#define st_mtime st_mtim.tv_sec
#define st_ctime st_ctim.tv_sec
};
```

RETURN VALUE

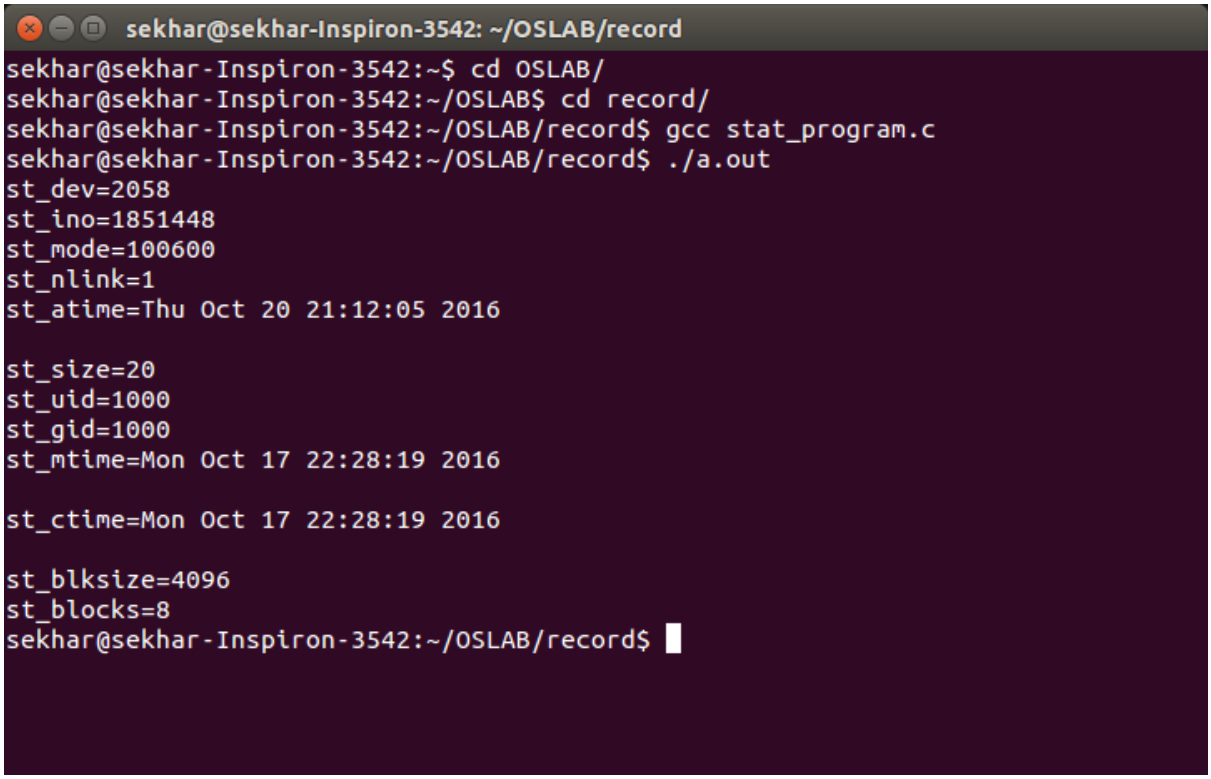
On success, zero is returned. On error, -1 is returned, and errno is set appropriately.

Program:-

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<unistd.h>
#include<string.h>
#include<time.h>
main()
{
        struct stat buf;
        int i;
        i=stat("file.txt",&buf);
        printf("st_dev=%ld\n",buf.st_dev);
        printf("st_ino=%ld\n",buf.st_ino);
        printf("st_mode=%o\n",buf.st_mode);
        printf("st_nlink=%ld\n",buf.st_nlink);
        printf("st_atime=%s\n",ctime(&buf.st_atime));
        printf("st_size=%ld\n",buf.st_size);
        printf("st_uid=%d\n",buf.st_uid);
        printf("st_gid=%d\n",buf.st_gid);
        printf("st_mtime=%s\n",ctime(&buf.st_mtime));
        printf("st_ctime=%s\n",ctime(&buf.st_ctime));
        printf("st_blksize=%ld\n",buf.st_blksize);
        printf("st_blocks=%ld\n",buf.st_blocks);
```

}

Output:-


sekhar@sekhar-Inspiron-3542: ~/OSLAB/record
sekhar@sekhar-Inspiron-3542:~$ cd OSLAB/
sekhar@sekhar-Inspiron-3542:~/OSLAB$ cd record/
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ gcc stat_program.c
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ ./a.out
st_dev=2058
st_ino=1851448
st_mode=100600
st_nlink=1
st_atime=Thu Oct 20 21:12:05 2016
st_size=20
st_uid=1000
st_gid=1000
st_mtime=Mon Oct 17 22:28:19 2016
st_ctime=Mon Oct 17 22:28:19 2016
st_blksize=4096
st_blocks=8
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$