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

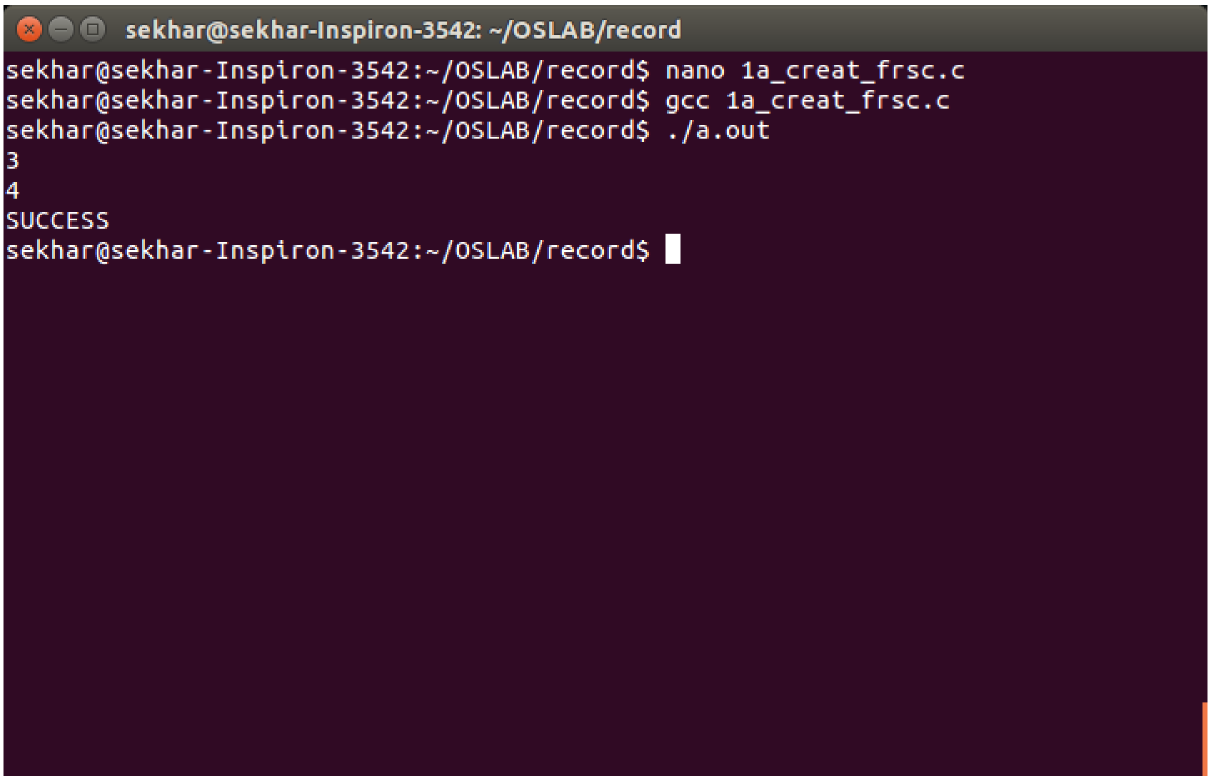
System calls required:
creat() system call :
open, creat - open and possibly create a file or device
Header files required:
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
Syntax:
int creat(const char *pathname, mode_t* mode)
Description:
Given a *pathname* for a file, open() returns 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$ 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().

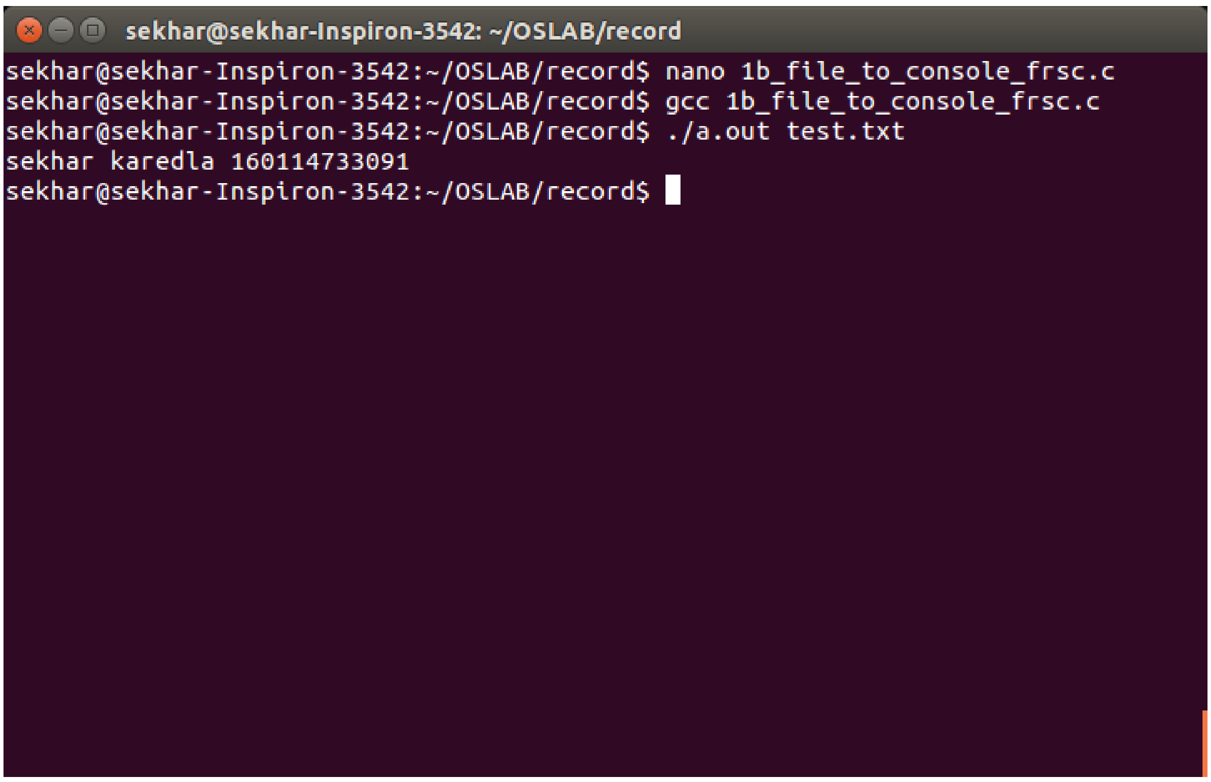
Return Value:

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

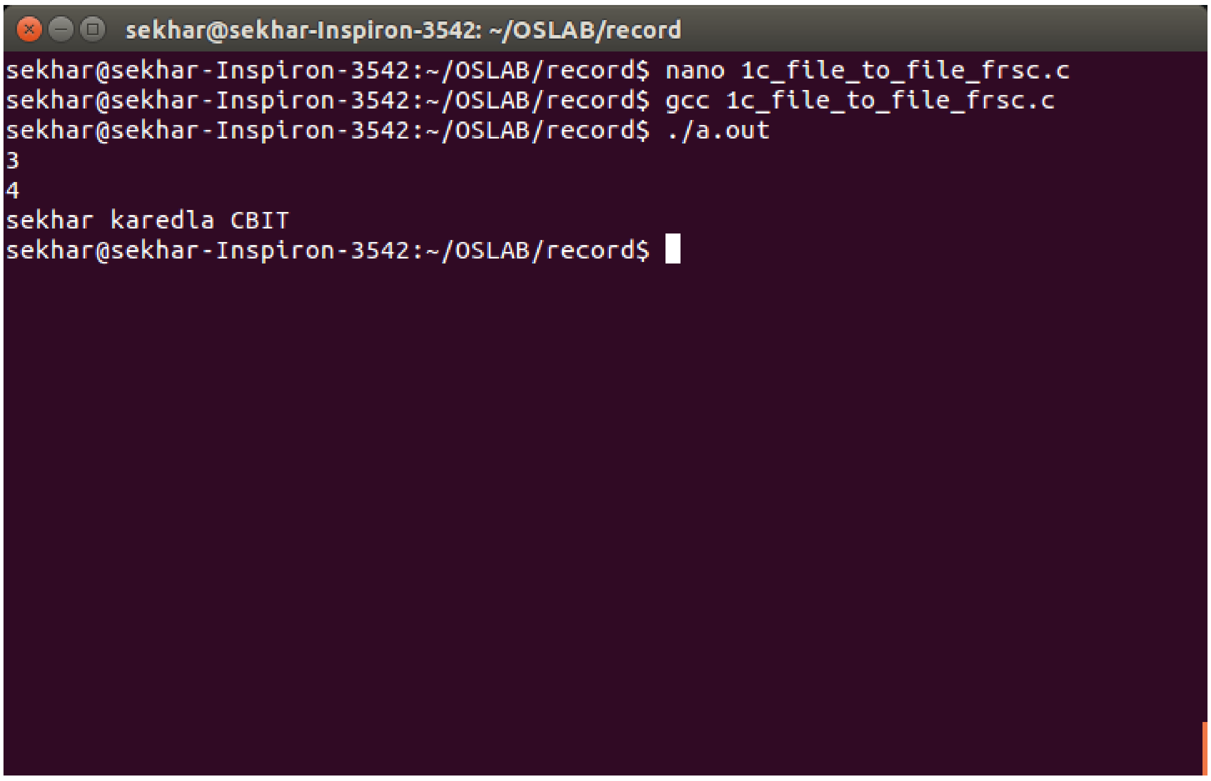
These functions do not return.

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:



```
sekhar@sekhar-Inspiron-3542: ~/OSLAB/record
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ nano 1c_file_to_file_frsc.c
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ gcc 1c_file_to_file_frsc.c
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ ./a.out
3
4
sekhar karedla CBIT
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$
```

PROCESS RELATED SYSTEM CALLS

4. 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*.

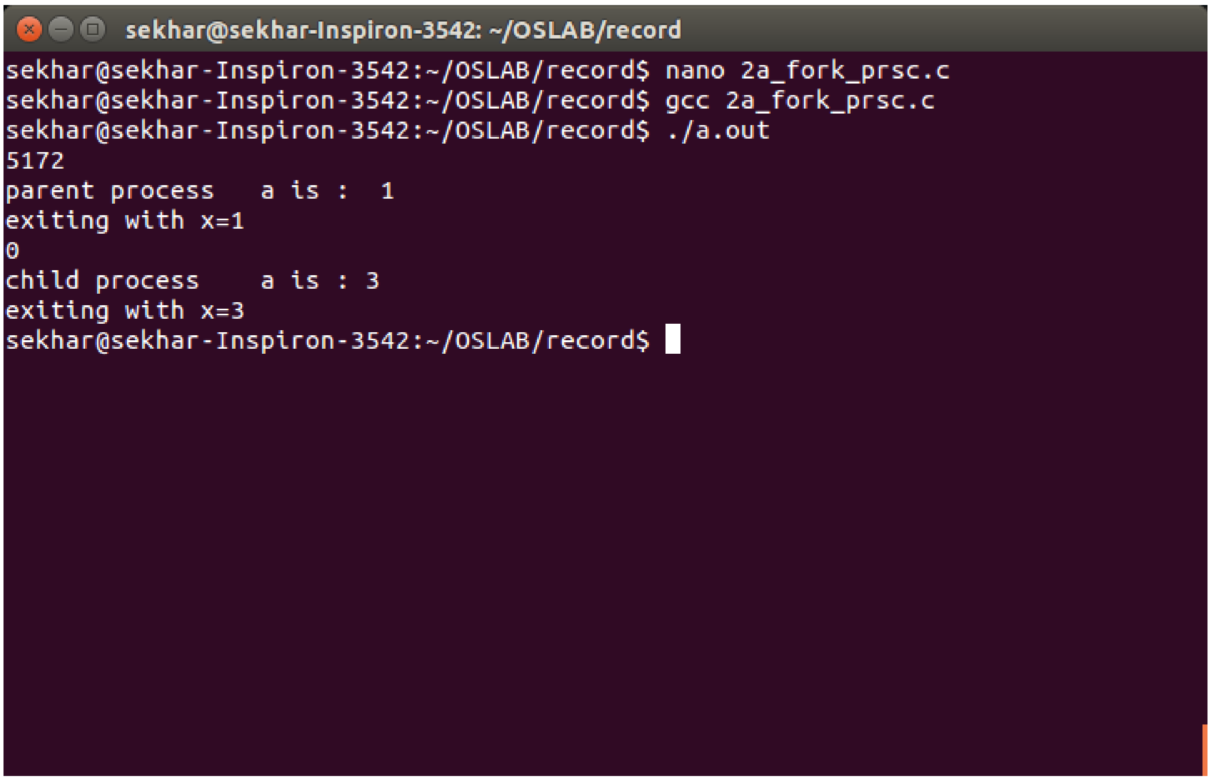
Return Value:

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$
```

5. 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:

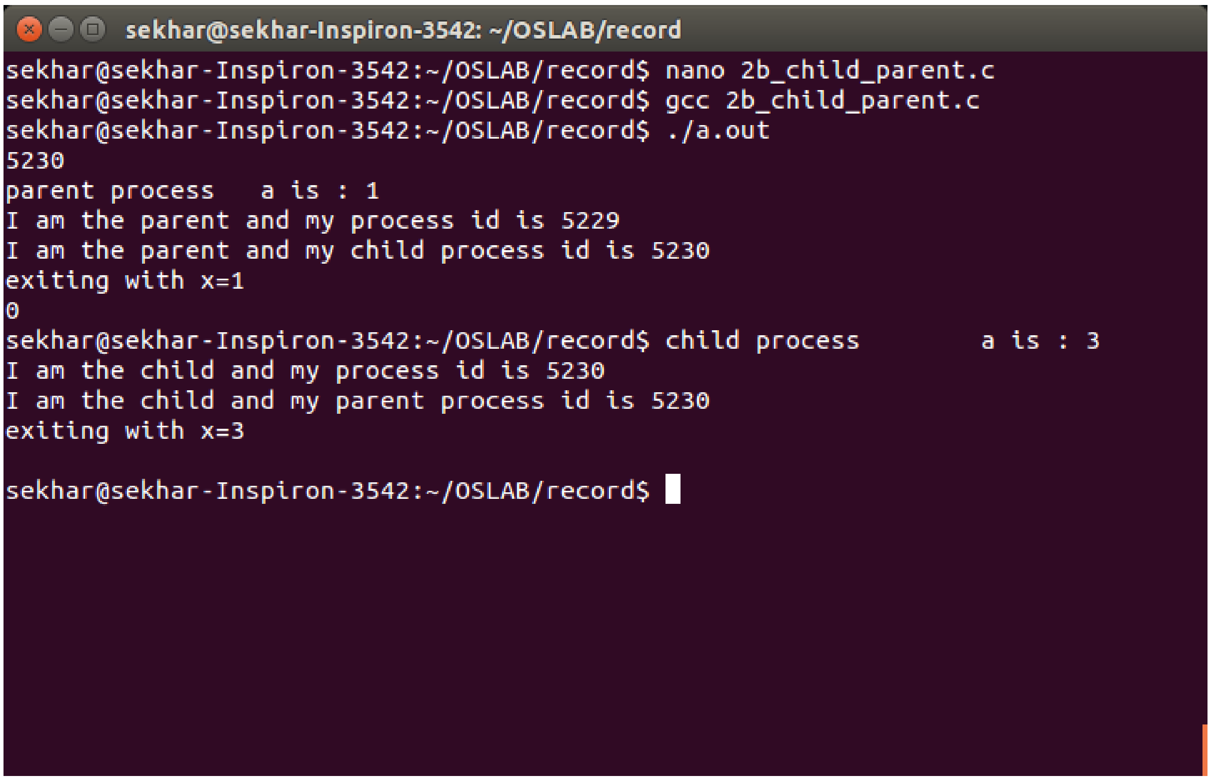
getpid() returns the process ID of the calling process. (This is often used by routines that generate unique temporary filenames.)

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:



sekhar@sekhar-Inspiron-3542: ~/OSLAB/record

```
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ nano 2b_child_parent.c
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ gcc 2b_child_parent.c
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ ./a.out
5230
parent process   a is : 1
I am the parent and my process id is 5229
I am the parent and my child process id is 5230
exiting with x=1
0
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ child process        a is : 3
I am the child and my process id is 5230
I am the child and my parent process id is 5230
exiting with x=3

sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ 
```

6. 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.)

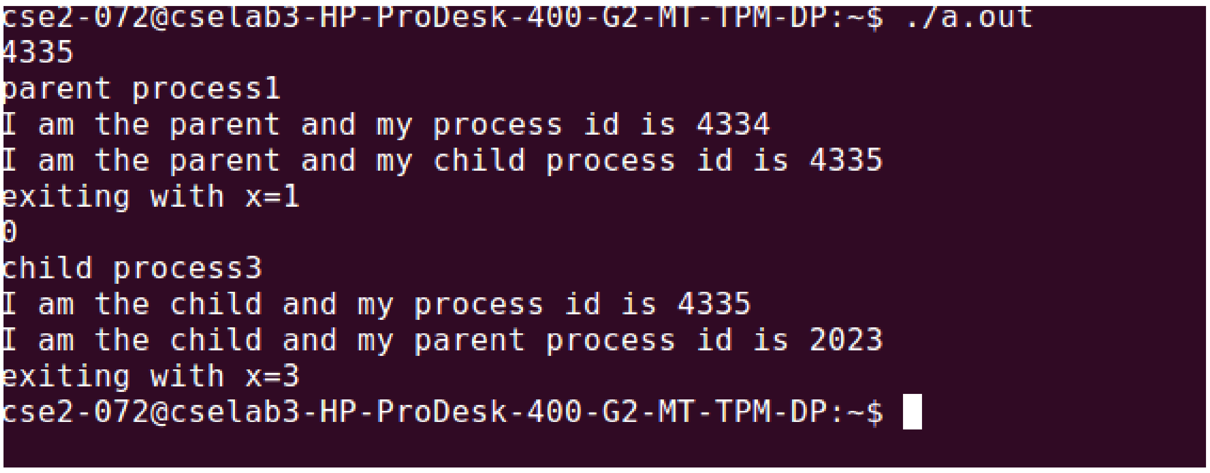
getppid() returns the process ID of the parent of the calling process.

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:



```
cse2-072@cselab3-HP-ProDesk-400-G2-MT-TPM-DP:~$ ./a.out
4335
parent process1
I am the parent and my process id is 4334
I am the parent and my child process id is 4335
exiting with x=1
0
child process3
I am the child and my process id is 4335
I am the child and my parent process id is 2023
exiting with x=3
cse2-072@cselab3-HP-ProDesk-400-G2-MT-TPM-DP:~$
```

7. 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.

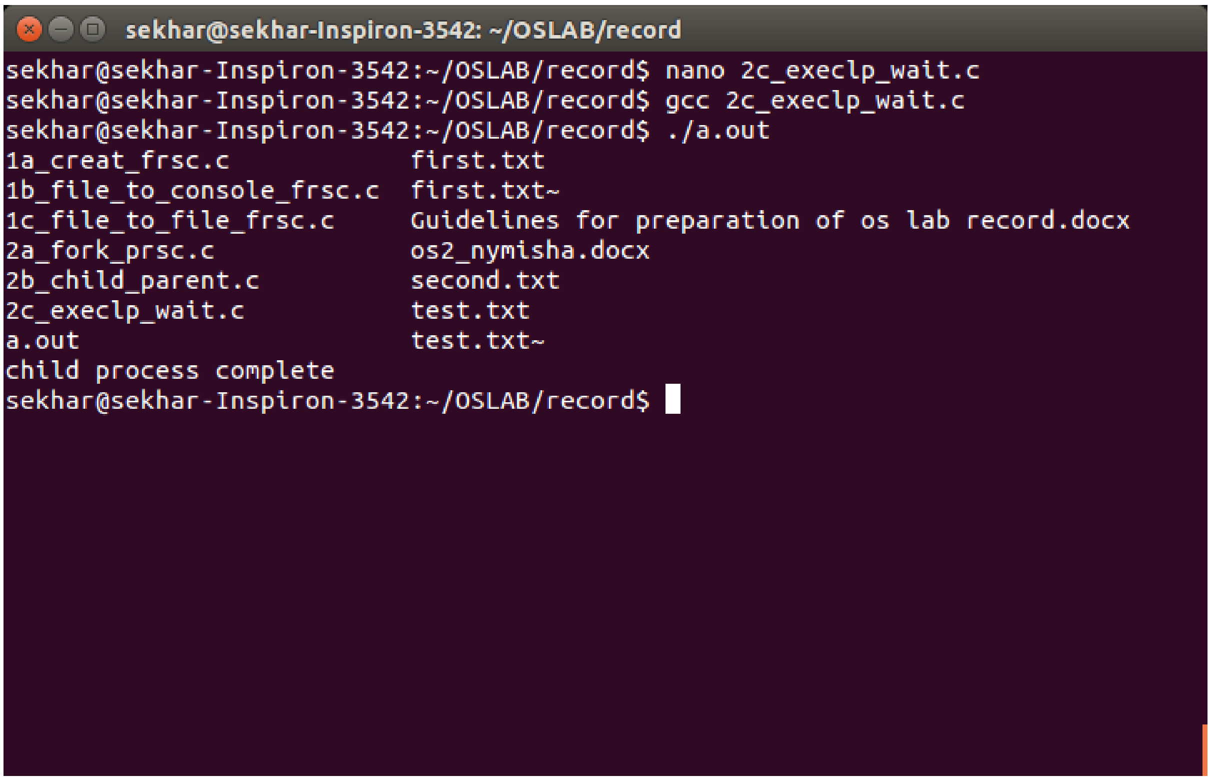
Return Value

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:



sekhar@sekhar-Inspiron-3542: ~/OSLAB/record

```
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ nano 2c_execlp_wait.c
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ gcc 2c_execlp_wait.c
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ ./a.out
1a_creat_frsc.c           first.txt
1b_file_to_console_frsc.c  first.txt~
1c_file_to_file_frsc.c     Guidelines for preparation of os lab record.docx
2a_fork_prsc.c             os2_nymisha.docx
2b_child_parent.c          second.txt
2c_execlp_wait.c           test.txt
a.out                      test.txt~
child process complete
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ 
```

CPU SCHEDULING

8. 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

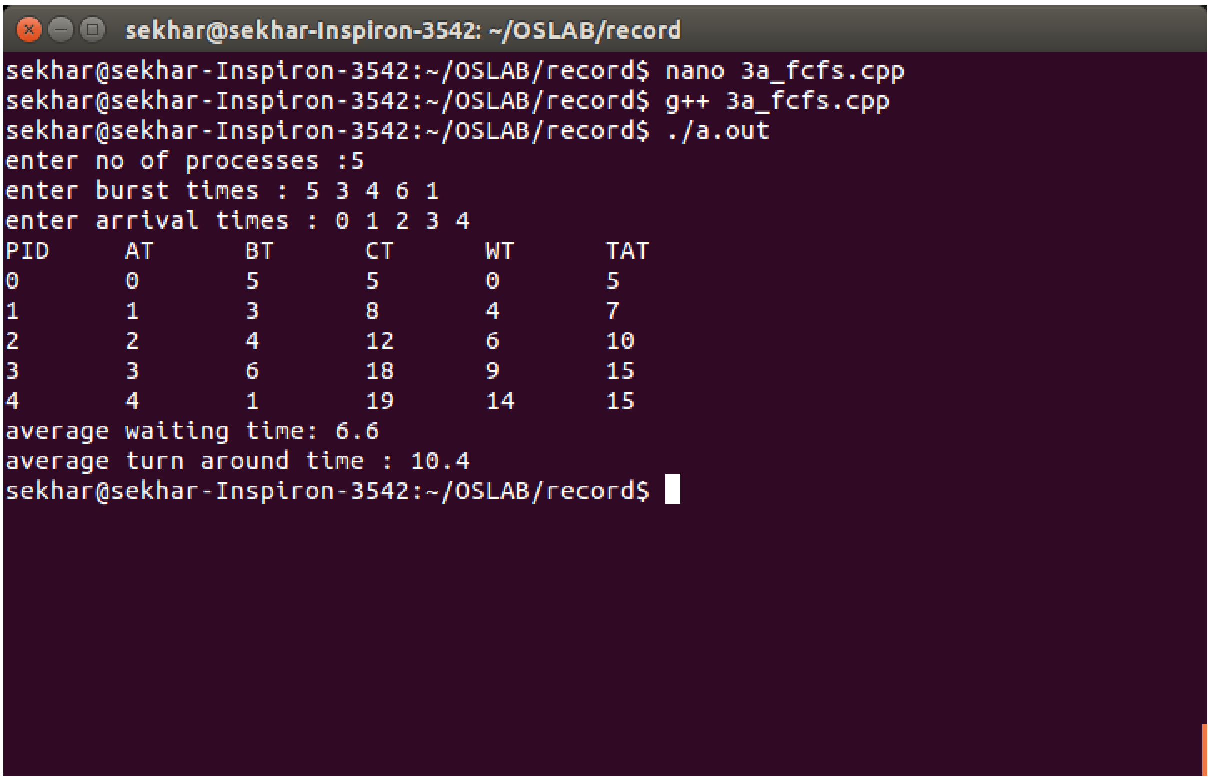
Program:

```
#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$ 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$
```

9. 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

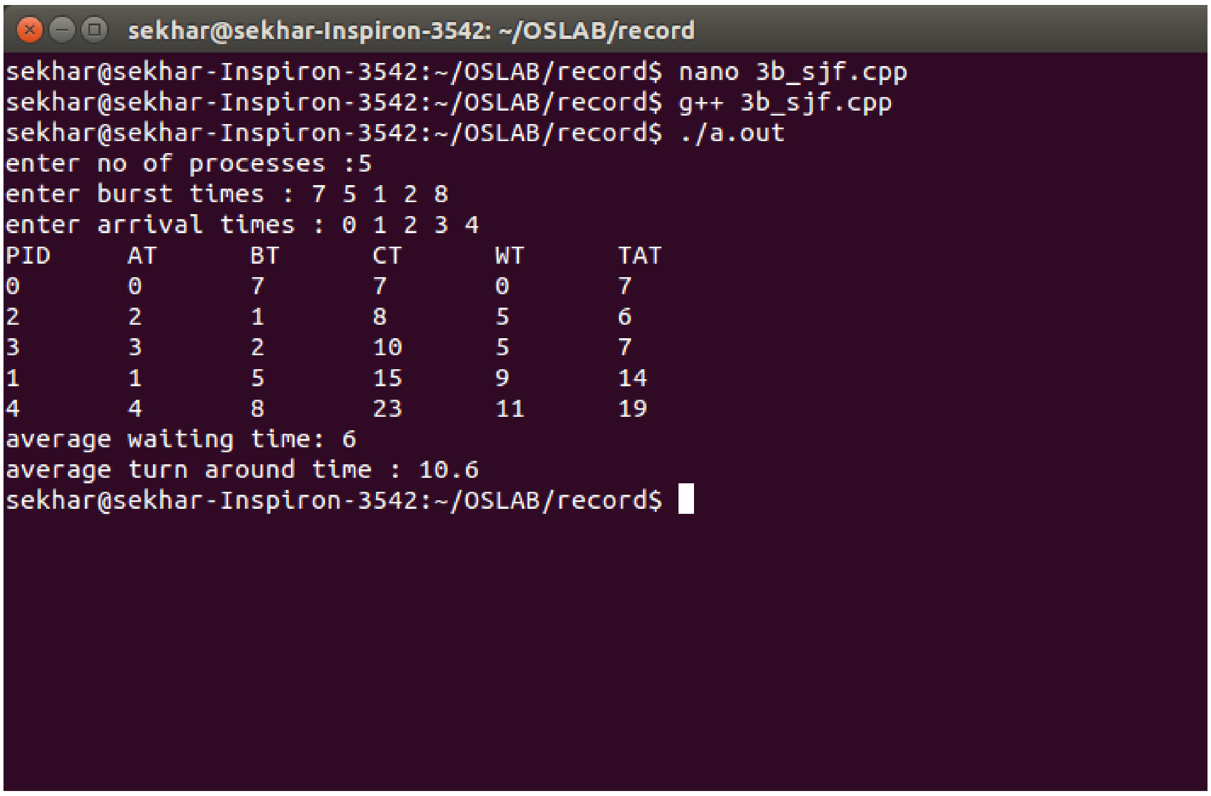
7. Display the values

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$ 
```

10. 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.

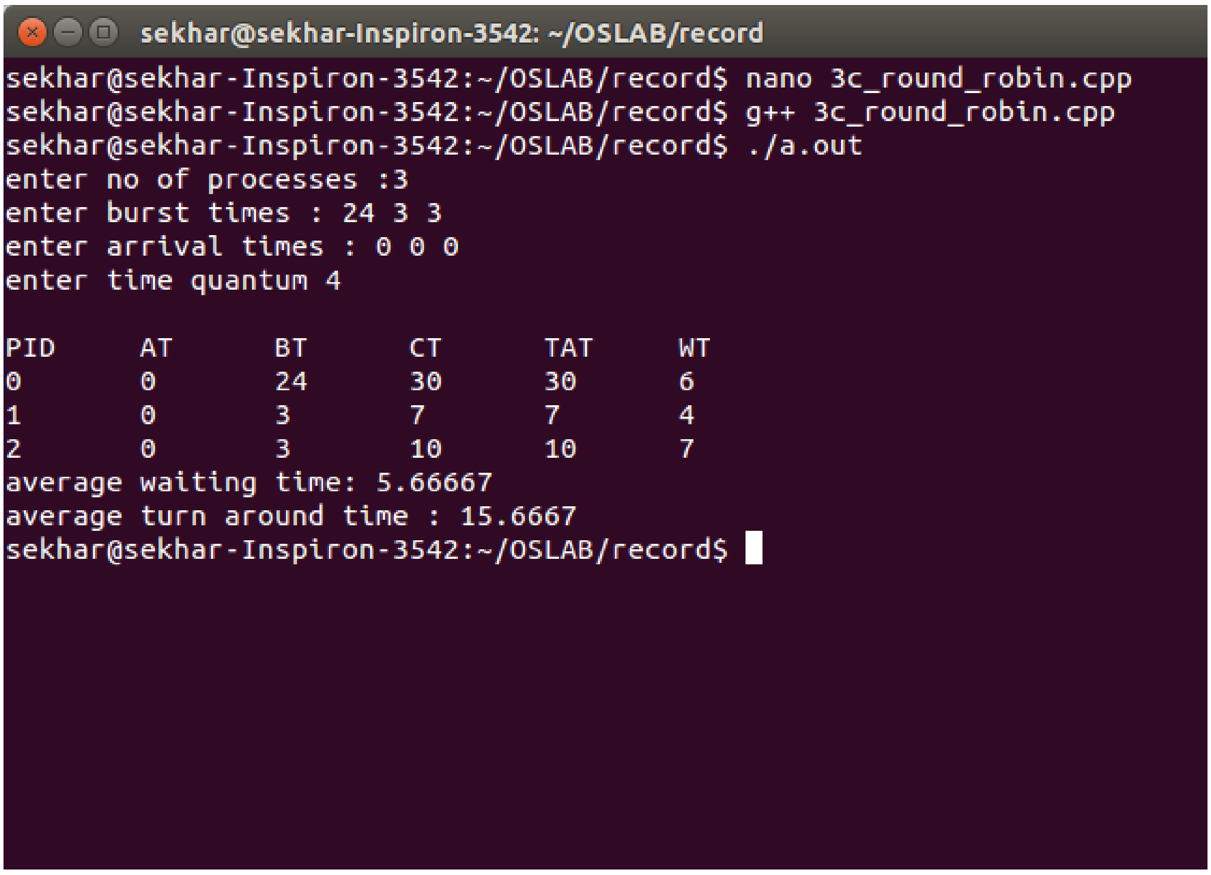
Program:

```
#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:



sekhar@sekhar-Inspiron-3542: ~/OSLAB/record

```
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ nano 3c_round_robin.cpp
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ g++ 3c_round_robin.cpp
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ ./a.out
enter no of processes :3
enter burst times : 24 3 3
enter arrival times : 0 0 0
enter time quantum 4

PID     AT      BT      CT      TAT     WT
0       0       24      30      30      6
1       0       3       7       7       4
2       0       3       10      10      7
average waiting time: 5.66667
average turn around time : 15.6667
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ 
```

Bankers Algorithm

Deadlock Detection & Deadlock Avoidance

11. 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];
```

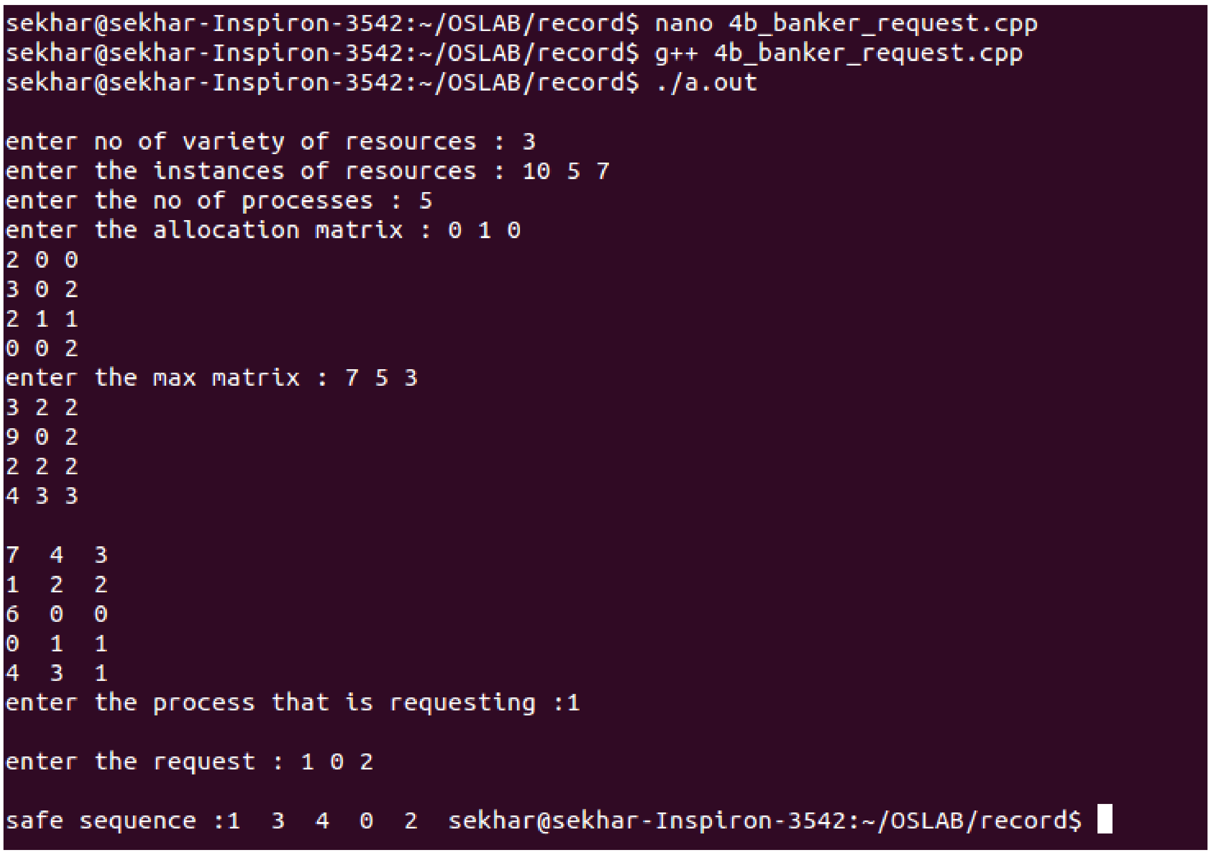
```
    for(int i=0;i<nr;i++)
    cin>>r[i];
    cout<<"enter the no of processes : ";
    int p;
    cin>>p;
    cout<<"enter the allocation matrix : ";
    int **am=new int*[p];
    for(int i=0;i<p;i++)
    am[i]=new int[nr];
    for(int i=0;i<p;i++)
    for(int j=0;j<nr;j++)
    cin>>am[i][j];
  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

}
```

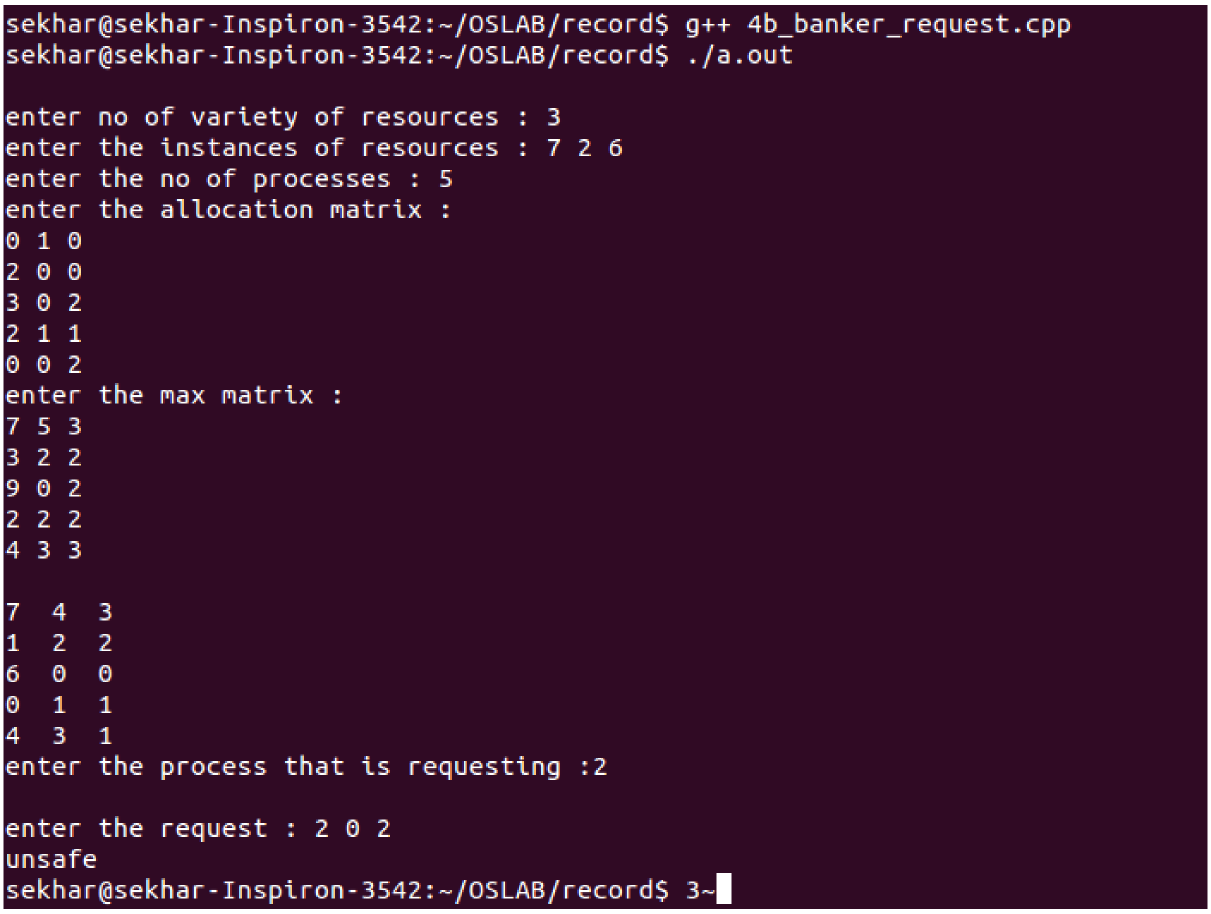
```
temp[i]=sum;
}
int *work=new int[nr];
for(int i=0;i<nr;i++)
{
avai[i]=r[i]-temp[i];
work[i]=avai[i];
}
bool *finish=new bool[p];
for(int i=0;i<p;i++)
finish[i]=false;
int trap=0;
int count=0;
int disp=1;
//cout<<"\nsafe sequence :";
while(trap<p)
{
        if(count>p*p)
        {cout<<"unsafe"<<endl;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

12. 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]<<"  ";
        }
}
```

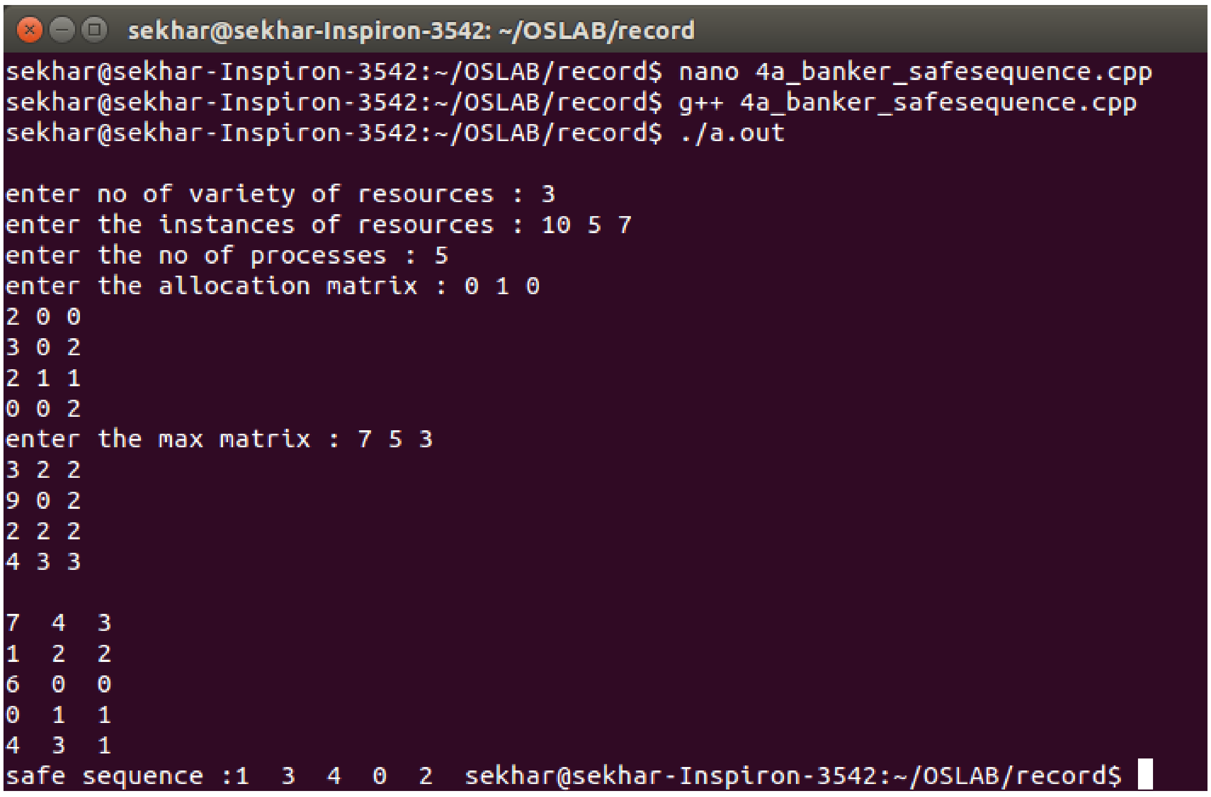
```
bool isSmall(int *a,int *b,int n)
{
    int flag=0;
    for(int i=0;i<n;i++)
    {
    if(a[i]>b[i])
    {
    flag=1;
    break;
    }
    }
    if(flag==0)
    return true;
    else
    return false;
}

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:



```
sekhar@sekhar-Inspiron-3542: ~/OSLAB/record
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ nano 4a_banker_safesequence.cpp
sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ g++ 4a_banker_safesequence.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
safe sequence :1   3   4   0   2   sekhar@sekhar-Inspiron-3542:~/OSLAB/record$ 
```

PAGE REPLACEMENT ALGORITHMS

12. 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

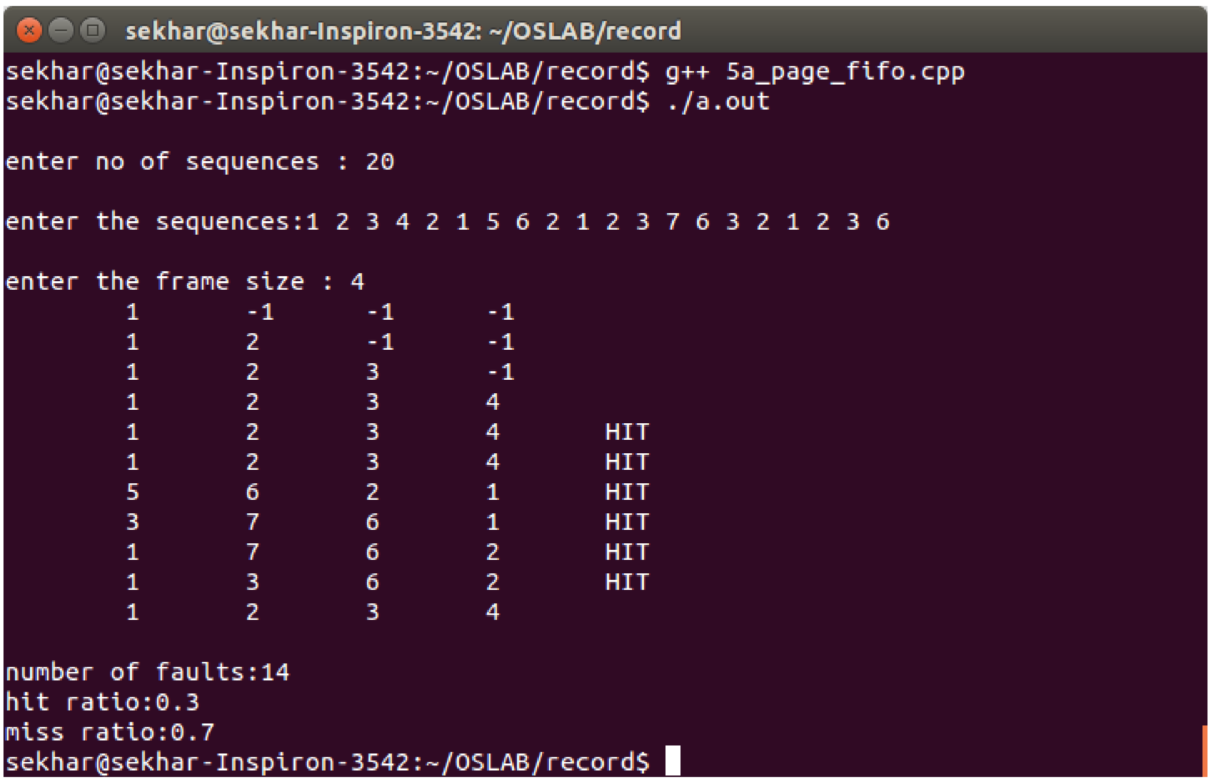
Program:

```
#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$ 
```

ECHO SERVER USING PIPES

14. 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:

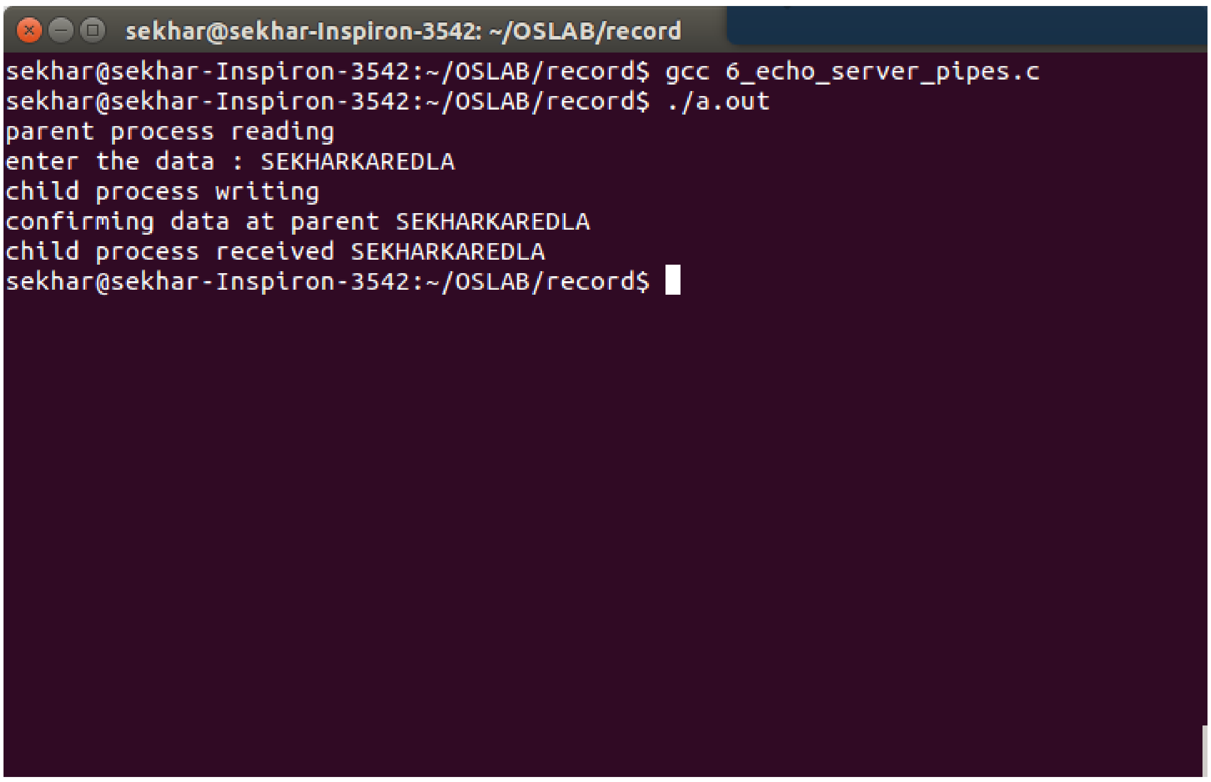
On success, zero is returned, on error -1 is returned.

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$ 
```