

SHIV NADAR

— UNIVERSITY —
CHENNAI

BONAFIDE CERTIFICATE

Certified that this is the bonafide record of the practical work done in the

..... Laboratory by

Name

Register Number

SemesterClass & Sec.

Branch

SHIV NADAR UNIVERSITY Chennai

During the Academic year

Faculty

Head of the Department

Submitted for the.....Practical Examination held at
SNU CHENNAI on.....

Internal Examiner

External Examiner

INDEX

Name : Reg. No.

Sem : Class & Sec :

[illegible]

Ex. No: 1	Basic Linux commands
04/01/2023	

Aim:

To simulate basic linux commands and explore all different options that are available.

Command:

Date:

```
root@LAPTOP-FHHEGJQ5:~# date
Thu Dec 29 08:57:35 IST 2022
```

Cal:

```
root@LAPTOP-FHHEGJQ5:~# cal
    December 2022
Su Mo Tu We Th Fr Sa
                1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
```

Echo:

```
root@LAPTOP-FHHEGJQ5:~# echo ajay
ajay
```

Ls:

```
root@LAPTOP-FHHEGJQ5:~# ls -l
total 28
drwxr-xr-x 1 root root 512 Sep 13 2021 Ajay
-rwxr-xr-x 1 root root 16704 Nov  5 01:01 a.out
-rw-r--r-- 1 root root 261 Nov  5 01:01 bisquit.c
drwxr-xr-x 1 root root 512 Sep 28 15:03 dsa6
-rw-r--r-- 1 root root 817 Jun 18 2022 hw.c
-rw-r--r-- 1 root root 375 Jul 25 15:09 key.c
-rw----- 1 root root  6 Nov 24 2021 nano.save
-rw-r--r-- 1 root root 1398 Jan 15 2022 opengl.c
-rw-r--r-- 1 root root 166 Oct 13 2021 pattern.c
-rw-r--r-- 1 root root 71 Aug 24 19:32 test.c
```

Lp: line printer

Man :

```
STDIO(3)                                Linux Programmer's Manual                                STDIO(3)

NAME
    stdio - standard input/output library functions

SYNOPSIS
    #include <stdio.h>

    FILE *stdin;
    FILE *stdout;
    FILE *stderr;

DESCRIPTION
    The standard I/O library provides a simple and efficient buffered stream I/O interface. Input and output is mapped into logical data streams and the physical I/O characteristics are concealed. The functions and macros are listed below; more information is available from the individual man pages.

    A stream is associated with an external file (which may be a physical device) by opening a file, which may involve creating a new file. Creating an existing file causes its former contents to be discarded. If a file can support positioning requests (such as a disk file, as opposed to a terminal), then a file position indicator associated with the stream is positioned at the start of the file (byte zero), unless the file is opened with append mode. If append mode is used, it is unspecified whether the position indicator will be placed at the start or the end of the file. The position indicator is maintained by subsequent reads, writes and positioning requests. All input occurs as if the characters were read by successive calls to the fgetc(3) function; all output takes place as if all characters were written by successive calls to the fputc(3) function.

    A file is disassociated from a stream by closing the file. Output streams are flushed (any unwritten buffer contents are transferred to the host environment) before the stream is disassociated from the file. The value of a pointer to a FILE object is indeterminate after a file is closed (garbage).
```

Who/whoami:

```
root@LAPTOP-FHHEGJQ5:~# who
root@LAPTOP-FHHEGJQ5:~# whoami
root
```

Uptime:

```
root@LAPTOP-FHHEGJQ5:~# uptime
09:21:12 up 23 min,  0 users,  load average: 0.52, 0.58, 0.59
```

Uname:

```
root@LAPTOP-FHHEGJQ5:~# uname
Linux
```

Hostname :

```
root@LAPTOP-FHHEGJQ5:~# hostname
LAPTOP-FHHEGJQ5
```

Bc:(basic calculator)

```
root@LAPTOP-FHHEGJQ5:~# bc
bc 1.07.1
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006, 2008, 2012-2017 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
readline: /etc/inputrc: line 69: number: unknown variable name
1*3
3
34.53*323.424
11167.830
```

File Handling Commands:

Cat:

```
d diam vel. Nulla malesuada pellentesque elit eget gravida cum sociis natoque penatibus. Facilisis sed odio morbi quis  
commodo. Ut sem viverra aliquet eget sit amet tellus cras. Condimentum mattis pellentesque id nibh tortor id aliquet. A  
condimentum vitae sapien pellentesque habitant morbi.  
  
Consectetur a erat nam at. Sit amet est placerat in egestas. Volutpat diam ut venenatis tellus in metus vulputate. Ali  
am faucibus purus in massa tempor. Et odio pellentesque diam volutpat commodo sed egestas. Lectus proin nibh nisl cond  
entum. Sed arcu non odio euismod lacinia. Tortor vitae purus faucibus ornare suspendisse sed nisi lacus sed. Arcu odio  
t sem nulla pharetra diam sit. Vestibulum sed arcu non odio. Sodales ut eu sem integer.  
  
Proin fermentum leo vel orci porta non pulvinar neque laoreet. Lacus luctus accumsan tortor posuere. Aenean pharetra m  
na ac placerat vestibulum. Bibendum enim facilisis gravida neque convallis. Ultrices gravida dictum fusce ut placerat  
ci nulla. Integer quis auctor elit sed vulputate mi sit amet mauris. Pellentesque pulvinar pellentesque habitant morbi  
ristique senectus et netus et. Suspendisse sed nisi lacus sed viverra tellus in hac. In dictum non consectetur a erat  
e at lectus urna. At volutpat diam ut venenatis tellus in metus vulputate eu. Tincidunt augue interdum velit euismod i  
pellentesque massa. Tellus molestie nunc non blandit massa enim nec dui nunc. Malesuada bibendum arcu vitae elementum  
abitur vitae nunc sed.  
  
Mauris a diam maecenas sed enim ut sem viverra aliquet. Neque viverra justo nec ultrices dui. Est placerat in egestas  
at imperdiet sed. Tortor at auctor urna nunc id. Dis parturient montes nascetur ridiculus mus mauris. Interdum velit l  
reet id donec ultrices tincidunt arcu non. Risus in hendrerit gravida rutrum quisque non. Enim ut sem viverra aliquet  
et sit amet tellus. Lobortis feugiat vivamus at augue. Mi in nulla posuere sollicitudin aliquam ultrices.  
  
Quisque non tellus orci ac. Eros in cursus turpis massa tincidunt dui. Sagittis id consectetur purus ut faucibus pulvi  
r. Sit amet purus gravida quis. Libero nunc consequat interdum varius sit. Scelerisque varius morbi enim nunc. Vitae t  
tor condimentum lacinia quis vel eros. Sollicitudin nibh sit amet commodo nulla. Ullamcorper morbi tincidunt ornare ma  
a eget egestas purus viverra accumsan. Risus commodo viverra maecenas accumsan lacus vel facilisis volutpat. Sed felix  
get velit aliquet sagittis id consectetur purus. Luctus accumsan tortor posuere ac ut consequat. Arcu non sodales neque  
sodales. Lacus vel facilisis volutpat est velit egestas dui. Rhoncus dolor purus non enim praesent. Quis risus sed vul
```

2.grep:

```
Report bugs to: bug-grep@gnu.org  
GNU grep home page: <http://www.gnu.org/software/grep/>  
General help using GNU software: <https://www.gnu.org/gethelp/>  
root@LAPTOP-FHHEGJQ5:~# grep -c "lorem" test.txt  
1  
root@LAPTOP-FHHEGJQ5:~# grep -l "lorem"  
^C  
root@LAPTOP-FHHEGJQ5:~# grep -l "lorem" -l  
^C  
root@LAPTOP-FHHEGJQ5:~# grep -l "lorem" *  
grep: Ajay: Is a directory  
grep: dsa6: Is a directory  
test.txt
```

3. rm:

```
root@LAPTOP-FHHEGJQ5:~# touch dummy.txt  
root@LAPTOP-FHHEGJQ5:~# rm dummy.txt
```

4.touch

```
root@LAPTOP-FHHEGJQ5:~# touch dummy.txt
```

5.cp

```
root@LAPTOP-FHHEGJQ5:~# cp test.txt mnt  
root@LAPTOP-FHHEGJQ5:~#
```

6.mv

```
root@LAPTOP-FHHEGJQ5:~# mv test.txt mnt
```

7.cut

```
root@LAPTOP-FHHEGJQ5:/# cut -c 1 demo.c
#
i
{
```

8.head

```
root@LAPTOP-FHHEGJQ5:/# head -c 10 demo.c
#include <root@LAPTOP-FHHEGJQ5:/#
```

9.tail

```
root@LAPTOP-FHHEGJQ5:/# tail -c 12 demo.c
return 0;
}
```

10.chmod

```
root@LAPTOP-FHHEGJQ5:/# chmod u=r demo.c
root@LAPTOP-FHHEGJQ5:/#
```

11.wc(word count s);

```
root@LAPTOP-FHHEGJQ5:/# wc -w demo.c
61 demo.c
```

Result:

Thus the above commands were simulated and their options were experimented

Ex. No: 2	IMPLEMENTATION OF SYSTEM CALLS
09/01/2023	

Aim:

To write C Program to implement the following system calls

- Open ()
- Read ()
- Write ()
- Wait ()
- Exec ()
- Fork ()
- Sleep ()
- Getpid ()
- Lseek ()

Descriptions:

1.Open():

Header:

```
#include <fcntl.h>
```

Syntax:

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

Description:

->The open() system call opens the file specified by pathname. If the specified file does not exist, it may optionally (if O_CREAT is specified in flags) be created by open().

The return value of open() is a file descriptor, a small,

nonnegative integer that is an index to an entry in the process's table of open file descriptors. The file descriptor is used in subsequent system calls (read(2), write(2), lseek(2), fcntl(2), etc.) to refer to the open file. The file descriptor returned by a successful call will be the lowest-numbered file descriptor not currently open for the process

2.read():

Header:

```
#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.

On files that support seeking, the read operation commences at

the file offset, and the file offset is incremented by the number

of bytes read. If the file offset is at or past the end of file, no bytes are read, and read() returns zero.

If count is zero, read() may detect the errors described

below. In the absence of any errors, or if read() does not

check for

errors, a read() with a count of 0 returns zero and has no other effects.

3.write():

Header:

```
#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 starting at buffer to the file referred to by the file descriptor fd.

The number of bytes written may be less than count if, for example, there is insufficient space on the underlying physical medium, or the RLIMIT_FSIZE resource limit is encountered or the call was interrupted by a signal handler after having written less than count bytes.

4.wait():

Header:

```
#include <sys/wait.h>
```

Syntax:

```
pid_t wait(int *wstatus);
```

Description:

The wait() system call suspends execution of the calling thread until one of its children terminates.

these system calls are used to wait for state changes in a child of the calling process, and obtain information about the child whose state has changed. A state change is considered to be: the child terminated; the child was stopped by a signal; or the child was resumed by a signal. In the case of a terminated child, performing a wait allows the system to release the resources associated with the child; if a wait is not performed,

then the terminated child remains in a "zombie" state

5.exec():

Header:

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

Syntax:

```
int execl(const char *pathname, const char *arg, ...  
/*, (char *) NULL */);
```

Description:

The exec() family of functions replaces the current process image with a new process image. The functions described in this manual page are layered on top of execve(2). (See the manual page for execve(2) for further details about the replacement of the current process image.)

The initial argument for these functions is the name of a file that is to be executed.

The functions can be grouped based on the letters following the

"exec" prefix.

6.Fork():

Header :

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

Syntax:

```
pid_t fork(void);
```

Description:

fork() creates a new process by duplicating the calling process.

The new process is referred to as the child process. The calling process is referred to as the parent process.

The child process and the parent process run in separate memory spaces. At the time of fork() both memory spaces have the same content. Memory writes, file mappings (mmap(2)), and unmappings (munmap(2)) performed by one of the processes do not affect the other.

7. sleep():

Header:

```
#include <unistd.h>          Syntax:  
unsigned int sleep(unsigned int seconds);
```

Description:

sleep() causes the calling thread to sleep either until the number of real-time seconds specified in seconds have elapsed or until a signal arrives which is not ignored.

8.getpid():

```
Header:                               #include <unistd.h>          Syntax:  
pid_t getpid(void);
```

Description:

getpid() returns the process ID (PID) of the calling

process. (This is often used by routines that generate

unique temporary filenames.)

9.lseek():

Header:

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

Syntax:

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

Description:

lseek() repositions the file offset of the open file description

associated with the file descriptor fd to the argument offset

Programs:

1 Open:

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

int main(){
    int k=open("test.txt",O_RDONLY);
    char buffer[100];
    read(k,buffer,100);
    write(1,buffer,100);
}
```

Output:

```
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103# ./a.out
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore
```

2. Read():

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

int main(){
    char buffer[15];
    read(1,buffer,10);
    printf("%s\n",buffer);
}
```

Output:

```
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103# ./a.out
Hello
Hello
```

3.write():

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

```
int main(){
```

```
char bufferr[30]="hi there  hello world\n";
```

```
write(1,bufferr,30);
```

```
}
```

Output:

```
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103# ./a.out
hi there  hello world
```

4.wait():

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

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

```
#include <sys/wait.h>
```

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

```
int main(){
```

```
    int status;
```

```
    if(fork()==0)
```

```
    {
```

```
        printf("Exiting..");
```

```
        exit(1);
```

```
    }else{
```

```
        wait(&status);
```

```
    }
```

```
    printf("Exit status %d",WEXITSTATUS(status));
```

```
}
```

Output:

```
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103# ./a.out
Exiting..Exit status 1root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103#
```

5.Exec():

```
#include <unistd.h>

int main(){
char* path="/bin/ls";
char* arg[]={path,"-la",NULL};
execl(path,"-la",NULL);
execv(path,arg);
}
```

Output:

```
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103# gcc exec.c
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103# ./a.out
InterProcRecv.c  exec.c      fork2.c          oslabex1.pdf    pid.c   sleep.c   write.c
InterProcSend.c  fork.c      open.c          oslabex2.pdf    read.c  test.txt
a.out            fork.out    oslabex1.docx   oslabex2.rar    seek.c  wait.c
```

6. Fork():

```
#include <stdio.h>
#include <sys/types.h>

#include <unistd.h>

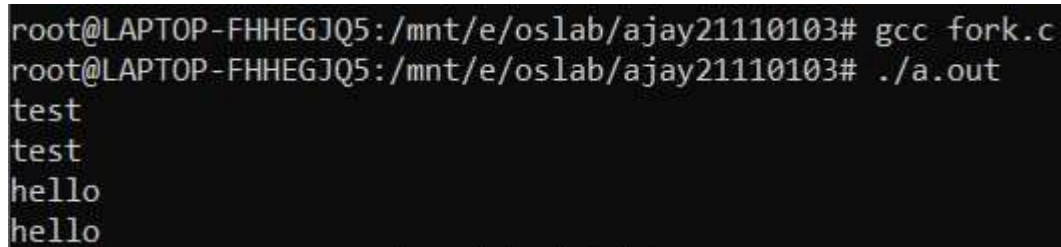
void test(){
fork();
//fork();

//fork();

printf("test\n");
```

```
}  
  
int main(){  
test();  
//fork()&&fork()||fork();  
//fork();  
//fork();  
  
//fork();  
  
printf("hello\n");  
}
```

Output:



```
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103# gcc fork.c  
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103# ./a.out  
test  
test  
hello  
hello
```

Fork():

```
#include <sys/types.h>  
#include <stdio.h>  
#include <unistd.h>  
  
#include <sys/wait.h>  
  
int main(){  
printf("Current processid: %d", (int) getpid());  
pid_t a=fork();  
printf(" After fork :%d\n", (int)a);  
if(a<0){  
fprintf(stderr, "Error \n");
```

```
}

else if (a==0){
printf("Child Process Created\n");
}
else{
printf("Forking not done yet\n");
wait(NULL);
printf("Child created\n");
}
}
```

Output:



```
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103# gcc fork2.c
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103# ./a.out
Current processid: 577 After fork :578
Current processid: 577 After fork :0
Forking not done yet
Child Process Created
Child created
```

7.sleep():

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

int main(){
printf("Hi there:\n");
sleep(3);
printf("hello\n");
}
```

Output:


```
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103# gcc sleep.c
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103# ./a.out
Hi there:
hello
```

8.getpid():

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

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

```
int main(){
if(fork()==0){
printf("Parent pid: %d\n",getpid());
printf("Child pid: %d\n",getppid());
}
}
```

Output:

```
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103# gcc pid.c
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103# ./a.out
Parent pid: 606
Child pid: 1
```

9.lseek():

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

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

```
#include <fcntl.h>
```

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

```
int main(){
int buf[40];
int fd=open("test.txt",O_RDWR);
read(fd,buf,40);
write(1,buf,40);
lseek(fd,15,SEEK_SET);
```

```
write(1, "\n", 1);  
read(fd, buf, 40);  
write(1, buf, 40);  
}
```

Output:

```
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103# ./a.out  
Lorem ipsum dolor sit amet, consectetur  
or sit amet, consectetur adipiscing elitroot@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103#
```

Test.txt:

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
```

Result:

Thus the above linux system calls were implemented in c

Ex. No: 3	Implementation of IPC using shared memory
17/01/2023	

Aim:

To Implement Interprocess communication using shared memory.

Description:

1.shm_open():

Header:

```
#include <sys/mman.h>

#include <sys/stat.h>    /* For mode constants */
#include <fcntl.h>       /* For O_* constants */
```

Syntax:

```
int shm_open(const char *name, int oflag, mode_t mode);
```

Description:

shm_open() creates and opens a new, or opens an existing, POSIX shared memory object. A POSIX shared memory object is in effect a handle which can be used by unrelated processes to mmap(2) the same region of shared memory. The shm_unlink() function performs the converse operation, removing an object previously created by shm_open().

2.shmget():

Header:

```
#include <sys/shm.h>
```

Syntax:

```
int shmget(key_t key, size_t size, int shmflg);
```

Description:

shmget() returns the identifier of the System V shared memory segment associated with the value of the argument key. It may be used either to obtain the identifier of a previously created

shared memory segment (when shmflg is zero and key does not have the value IPC_PRIVATE), or to create a new set.

3.shmat():

Header:

```
#include <sys/shm.h>
```

Syntax:

```
void *shmat(int shmid, const void *shmaddr, int shmflg);
```

Description:

The shmat() function attaches the shared memory segment associated with the shared memory identifier specified by shmid to the address space of the calling process. The segment is attached at the address specified by one of the following

criteria:

Code:

InterProcSend.c

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <stdlib.h>
//typedef struct values{
//    char* something;
//    int val;
//}val;

int main(){
char* SharedMemName="Sender";
char* message="Hi There!!\n";

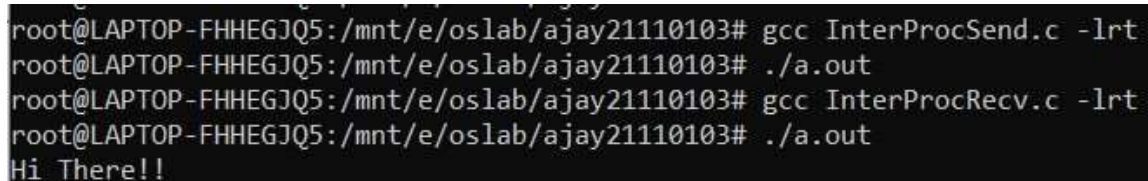
int shmFD;
void* ptr;
shmFD=shm_open(SharedMemName,O_CREAT|O_RDWR,0666);
if(shmFD==-1){
    write(1,"[+][shm_open]Failed To Create Shared Memory",40);
    return -1;
}
ftruncate(shmFD,4096);
```

```
ptr=mmap(0,4096,PROT_WRITE,MAP_SHARED,shmFD,0);
sprintf(ptr,"%s",message);
ptr+=strlen(message);
}
```

InterProcRecv.c:

```
#include <stdio.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <unistd.h>
int main(){
    char buf[10];
    char* SendShm="Sender";
    int shmFD;
    shmFD=shm_open(SendShm,O_RDONLY,0666);
    void* ptr=mmap(0,4096,PROT_READ,MAP_SHARED,shmFD,0);
    printf("%s",(char*)ptr);
    //read(shmFD,buf,10);
    //write(1,buf,10);
    shm_unlink("Sender");
}
```

OUTPUT:



```
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103# gcc InterProcSend.c -lrt
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103# ./a.out
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103# gcc InterProcRecv.c -lrt
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103# ./a.out
Hi There!!
```

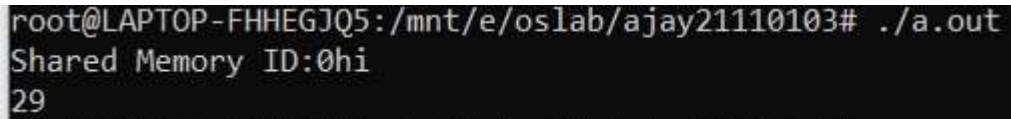
2.IPCS1.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <sys/shm.h>
typedef struct test_shm{
    char* something;
    int value;
}t_s;

int main(){
    key_t key=1234;
    int id=shmget(key,1024,IPC_CREAT|0644);
    printf("Shared Memory ID:%d",id);
    t_s*ptr=shmat(id,NULL,0);
    t_s *entry;
    entry=(t_s*)malloc(sizeof(t_s));
    entry->something="hi";
    entry->value=29;
```

```
memcpy(ptr,entry,sizeof(t_s));  
printf("%s\n%d\n",ptr->something,ptr->value);  
}
```

Output:



```
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103# ./a.out  
Shared Memory ID:0hi  
29
```

Result:

Thus Inter Process Communication was Established using shm.h library functions (shm_get,shmat).

Ex. No: 4	Implementation of IPC using Pipes And Creation Of Zombie and Orphan Process
24/01/2023	

Aim:

1. To Write a c program to implement Interprocess Communication using Pipes.
2. To Write a program to create Zombie process
3. To Write a Program to create orphan process

Description :

Pipes:

Conceptually, a pipe is a connection between two processes, such that the standard output from one process becomes the standard input of the other process. In UNIX Operating System, Pipes are useful for communication between related processes(inter-process communication).

Zombie Process:

A process which has finished the execution but still has entry in the process table to report to its parent process is known as a zombie process. A child process always first becomes a zombie before being removed from the process table.

Orphan Process:

An orphan process is a computer process whose parent process has finished or terminated, though it remains running itself.

1. write():

Header:

```
#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 starting at buffer to the file referred to by the file descriptor fd.

The number of bytes written may be less than count if, for example, there is insufficient space on the underlying physical medium, or the RLIMIT_FSIZE resource limit is encountered

or the call was interrupted by a signal handler after having written less than count bytes

2. read():

Header:

```
#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.

On files that support seeking, the read operation commences at the file offset, and the file offset is incremented by the number of bytes read. If the file offset is at or past the end of file, no bytes are read, and read() returns zero.

OS Lab Exercise 2 -Implementation Of System Calls -

B.Ajay(21110103) If count is zero, read() may detect the errors described below. In the absence of any errors, or if read() does not check for errors, a read() with a count of 0 returns zero and has no other effect

3. sleep(time):

Header:

```
#include
```

```
<unistd.h>
```

Syntax:

```
unsigned int sleep(unsigned int seconds);
```

Description:

sleep() causes the calling thread to sleep either until the number of real-time seconds specified in seconds have elapsed or until a signal arrives which is not ignored.

4. close(int fd):

Header:

```
#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 (see fcntl(2)) 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).

Code:

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

int main(){

    char* write1=(char*)malloc(sizeof(char)*30);
    char* read1=(char*) malloc(sizeof(char)*30);
    pid_t pid;
    int fd[2];
    write1= "Hello World!!!";
    if(pipe(fd)==0){
        pid=fork();
        printf("Parent pid:%d\n",getppid());
        if(pid>0){
            printf("Child process created :%d\n",getpid());
            close(fd[0]);
            write(fd[1],write1,strlen(write1)+1);
            printf("Sent From %d to %d\n",getppid(),getpid());
            close(fd[1]);
        }
        else{
            close(fd[1]);
            read(fd[0],read1,30);
            printf("Read %s from %d\n",read1,getpid());
            close(fd[0]);
        }
    }
```

```
}  
}
```

OUTPUT:

```
lot-a1@snuce-HP-ProDesk-400-G7-Microtower-PC:~/ajay21110103$ gcc IPCPipe.c  
lot-a1@snuce-HP-ProDesk-400-G7-Microtower-PC:~/ajay21110103$ ./a.out  
Parent pid:4081  
Child process created :8314  
Sent From 4081 to 8314  
Parent pid:8314  
Read Hello World!!! from 8315  
lot-a1@snuce-HP-ProDesk-400-G7-Microtower-PC:~/ajay21110103$ █
```

Zombie.c:

```
#include <stdio.h>  
  
#include <unistd.h>  
  
#include <sys/types.h>  
  
#include <stdlib.h>  
  
int main(){  
pid_t pid=fork();  
printf("PID CHILD %d",getpid());  
if(pid>0){  
sleep(34);  
}else{  
exit(0);  
}  
}
```

OUTPUT:

```
lot-a1@snuce-HP-ProDesk-400-G7-Microtower-PC: ~  
top - 10:09:39 up 110S, 1 user, load average: 0.46, 0.64, 0.68  
Tasks: 1 total, 0 running, 0 sleeping, 0 stopped, 1 zombie  
Cpu(s): 4.9 us, 0.4 sy, 0.0 ni, 94.5 id, 0.1 wa, 0.0 ht, 0.0 st, 0.0 sr  
Mem Mem : 15777.2 total, 10602.6 free, 2500.1 used, 2594.5 buff/cache  
Mem Swap: 30518.0 total, 30518.0 free, 0.0 used, 12020.8 avail Mem  


| PID  | USER   | PR | NI | VIRT | RES | SHR | S | %CPU | %MEM | TIME+   | COMMAND |
|------|--------|----|----|------|-----|-----|---|------|------|---------|---------|
| 8594 | lot-a1 | 20 | 0  | 0    | 0   | 0   | Z | 0.0  | 0.0  | 0:00.00 | a.out   |

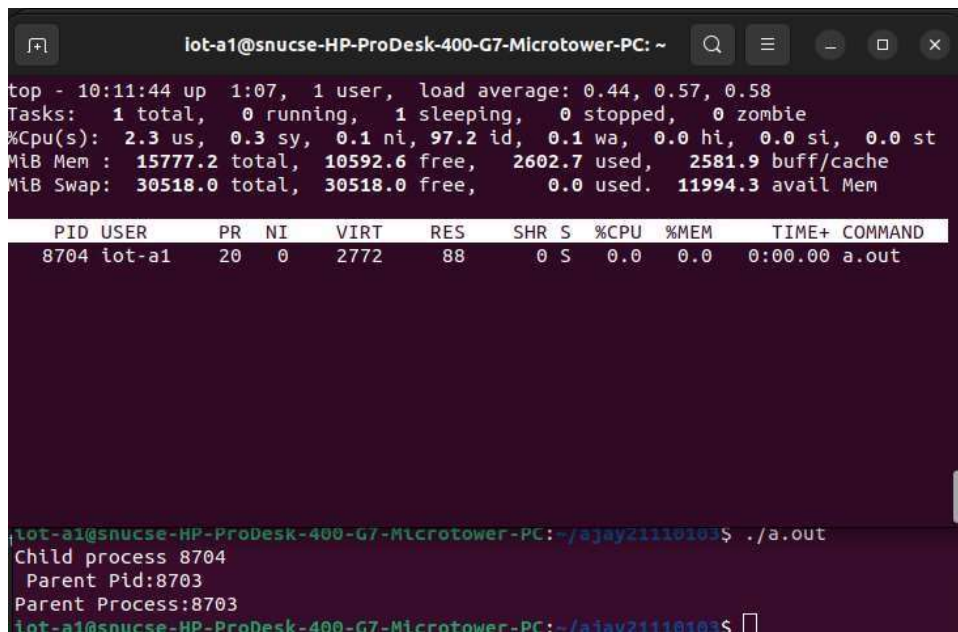
  
PID CHILD 8594PID CHILD 8555lot-a1@snuce-HP-ProDesk-400-G7-Microtower-PC: ~  
lot-a1@snuce-HP-ProDesk-400-G7-Microtower-PC: ~$ ./a.out  
PID CHILD 8508PID CHILD 8507lot-a1@snuce-HP-ProDesk-400-G7-Microtower-PC: ~$  
lot-a1@snuce-HP-ProDesk-400-G7-Microtower-PC: ~$ ./a.out  
PID CHILD 8594
```

Orphan.c:

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

```
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
int main(){
int p=fork();
if(p>0){
    sleep(10);
    printf("Parent Process:%d\n",getpid());
}
else {
    printf("Child process %d\n Parent Pid:%d\n",getpid(),getppid());
    sleep(30);
    exit(0);
}
}
```

OUTPUT (Orphan child still exist after parent terminates)

A terminal window titled 'lot-a1@snuce-HP-ProDesk-400-G7-Microtower-PC: ~' showing system statistics from the 'top' command and the output of a program. The 'top' command output shows system load averages (0.44, 0.57, 0.58), task counts (1 total, 0 running, 1 sleeping, 0 stopped, 0 zombie), CPU usage (2.3 us, 0.3 sy, 0.1 ni, 97.2 id, 0.1 wa, 0.0 hi, 0.0 si, 0.0 st), and memory usage (15777.2 total, 10592.6 free, 2602.7 used, 2581.9 buff/cache, 30518.0 total, 30518.0 free, 0.0 used, 11994.3 avail Mem). Below this, a table lists processes, with one entry for PID 8704, user 'lot-a1', priority 20, NI 0, VIRT 2772, RES 88, SHR 0, S 'S', %CPU 0.0, %MEM 0.0, TIME+ 0:00.00, and COMMAND 'a.out'. At the bottom, the terminal shows the command './a.out' being executed, resulting in the output: 'Child process 8704', 'Parent Pid:8703', and 'Parent Process:8703'.

```
lot-a1@snuce-HP-ProDesk-400-G7-Microtower-PC: ~
top - 10:11:44 up 1:07, 1 user, load average: 0.44, 0.57, 0.58
Tasks: 1 total, 0 running, 1 sleeping, 0 stopped, 0 zombie
%Cpu(s): 2.3 us, 0.3 sy, 0.1 ni, 97.2 id, 0.1 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 15777.2 total, 10592.6 free, 2602.7 used, 2581.9 buff/cache
MiB Swap: 30518.0 total, 30518.0 free, 0.0 used. 11994.3 avail Mem

  PID USER      PR  NI   VIRT   RES   SHR  S  %CPU  %MEM    TIME+  COMMAND
  8704 lot-a1    20   0    2772    88     0  S   0.0   0.0   0:00.00  a.out

lot-a1@snuce-HP-ProDesk-400-G7-Microtower-PC: ~/ajay21110103$ ./a.out
Child process 8704
Parent Pid:8703
Parent Process:8703
lot-a1@snuce-HP-ProDesk-400-G7-Microtower-PC: ~/ajay21110103$
```

Result:

Thus the IPC Communication was done through anonymous pipes and zombie, orphan process created using the appropriate syscalls.

Ex. No: 5	Implementation of CPU Scheduling Algorithms
30/01/2023	

Aim:

To simulate First Come First Serve Basis Scheduling Algorithm, Shortest Job First Algorithm and Shortest Remaining Time First Algorithm

Algorithm:

1.**FCFS:**

- 1.Maintain a 2d array with arrival time, burst time as columns
- 2.Declare a min function to find the index where the min arrival time is
- 3.Swap the index to the 1st Row and viceversa.
- 4.consider the swapped row till I as sorted
5. After n iterations we obtain a sorted order of PID int arrival time .
- 6.Compute the completion time for every iteration in the loop

2.**SJF :**

1. Maintain a 2d array with pid, arrival time, burst time as columns
- 2.Declare a function to check if all pid executed
3. Find the minimum arrival time in the 2d array
- 4.Based on its completion time compare other pid if they arrive within the completion time .
- 5.Next find the subsequent pid with smallest burst time.
- 6.While finding those keep track of all other parameters like TAT,CT etc...

3.**SRTF:**

- 1.In Shortest Remaining Time First first sort the 2d array based on their arrival time
2. Let the first process execute if there is another process that pre-empt the running process wherein its burst time is lesser than latter then preempt the latter executing process
Then reduce the burst time of latter by k units where k is the time it has finished executing
3. Keep track of new pid in an array.
- 4.Once all pids have been executed atleast once that guarantees that there exist no other process in the pipeline
- 5.Execute Remaining Process based off their shortest burst time.

Code:

FCFS.c

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

/*
Implementation of FCFS Scheduling
Works for Even Delay Arrival Process
Usage Whatever you begin with is your serialised ordered process ie.. first one is
P1
you can give whatever burst time and arrival time it should be good to go
Usage
no of proc,
arrival_time ,burst time
.
.
.
.
till no_proc

*/

void swap(int *a ,int *b){
    int tmp=*a;
    *a=*b;
    *b=tmp;
}

int min(int proc[][2],int start,int arrlen){
    int mins =proc[start][0];
    int i;
    int minidx;
```

```
for(i=start;i<arrlen;i++){
    if(proc[i][0]<=mins){
        minidx=i;
        mins=proc[i][0];
    }
}
return minidx;
}

int main(){
    int num,bt,at;
printf("Enter the number of Processes:");
    scanf("%d",&num);
    int proctable[num][2];
    int dup[num][2];
    for(int i=0;i<num;i++){

        scanf("%d %d",&at ,&bt);
        proctable[i][0]=at;
        proctable[i][1]=bt;
        dup[i][0]=at;
        dup[i][1]=bt;
    }
    int idx;
    int idxarr[num];
    int ct=0;
float c_avg=0.0,tat_avg=0.0,wt_avg=0.0;
    for (int i=0;i<num;i++){

        idx=min(proctable,i,num);
```

```
        idxarr[ct++]=idx;

        swap(&proctable[i][0],&proctable[idx][0]);
        swap(&proctable[i][1],&proctable[idx][1]);

    }

    int ct1=0,k,tat=0,wt=0;

    for(int i=0;i<num;i++){
        printf("%d %d\n",dup[idxarr[i]][0],dup[idxarr[i]][1]);
        if(dup[idxarr[i]][0]>ct){
            k=dup[idxarr[i]][0]-ct+dup[idxarr[i]][1];
            ct1+=k;
            c_avg+=(float)ct1;
        }
        else{
            k=dup[idxarr[i]][1];
            ct1+=k;
            c_avg+=(float)ct1;
        }
        tat=ct1-dup[idxarr[i]][0];
        wt=tat-dup[idxarr[i]][1];
        tat_avg+=(float)tat;
        wt_avg+=(float)wt;
        printf( "\nCompletion Time : %d\t Total Average Time: %d\t\n",ct1,tat,wt);
        swap(&dup[i][0],&dup[idxarr[i]][0]);
        swap(&dup[i][1],&dup[idxarr[i]][1]);
    }

    printf("Average Completion Time: %.2f\nAverage Total Arrival Time: %.2f\nAverage\nWaitingTime: %.2f\n",c_avg/(float)num,tat_avg/(float)num,wt_avg/(float)num);
}
```

Output:

```
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103/Exercise5# ./a.out
Enter the number of Processes:5
0 4
1 3
2 1
3 2
4 5
0 4

Completion Time : 4      Total Average Time: 4      Waiting time: 0
1 3

Completion Time : 7      Total Average Time: 6      Waiting time: 3
2 1

Completion Time : 8      Total Average Time: 6      Waiting time: 5
3 2

Completion Time : 10     Total Average Time: 7      Waiting time: 5
4 5

Completion Time : 15     Total Average Time: 11     Waiting time: 6
Average Completion Time: 8.80
Average Total Arrival Time:6.80
Average WaitingTime:3.80
```

SJF.c

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

int is_vis(int *vis,int pid,int vislen){
    for(int i=0;i<vislen;i++){if(pid==vis[i]){return 1;}}
    return 0;
}

int main(){
    int num,bt,at,pid;
    printf("Enter the number of Processes:");
    scanf("%d",&num);
    int proctable[num][3];
    for(int i=0;i<num;i++){
```



```
scanf("%d %d %d",&pid,&at ,&bt);

proctable[i][0]=pid;
proctable[i][1]=at;
proctable[i][2]=bt;
}

int visited_pid[num],n;
int min=proctable[0][1],v=0,ct=0,k=0,minidx,tat,wt;
float ct_avg=0.0,tat_avg=0.0,wt_avg=0.0;

for(int j=0;j<num;j++){
    if(proctable[j][1]<=min&&
is_vis(visited_pid,proctable[j][0],v)==0){
        min=proctable[j][1];

        minidx=j;
    }
}

visited_pid[v++]=proctable[minidx][0];
ct+=proctable[minidx][2];
tat=ct-proctable[minidx][1];
wt=tat-proctable[minidx][2];
ct_avg+=(float)ct;
tat_avg+=(float)tat;
wt_avg+=(float)wt;

printf("PID:%d \t CT:%d\t TAT:%d\t
WT:%d\n",proctable[minidx][0],ct,tat,wt);

while(v<num){

    min=10000;

    for( n=0;n<num;n++){
```

```
if(proctable[n][1]<ct&&is_vis(visited_pid,proctable[n][0],v)==0 ){
    if(proctable[n][2]<min){
        min=proctable[n][2];
        minidx=n;
    }
}
visited_pid[v++]=proctable[minidx][0];
ct+=proctable[minidx][2];
tat=ct-proctable[minidx][1];
wt=tat-proctable[minidx][2];
ct_avg+=(float)ct;
tat_avg+=(float)tat;
wt_avg+=(float)wt;
printf("PID:%d \t CT:%d\t TAT:%d\t
WT:%d\n",proctable[minidx][0],ct,tat,wt);

}

printf("Average CT :%.2f \t Average TAT:%.2f \t Average WT:%.2f
\n",ct_avg/(float)num,tat_avg/(float)num,wt_avg/(float)num);

//for(int i=0;i<v;i++){
//    printf("\n%d\t",visited_pid[i]);
//}

}
```

Output:

```
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103/Exercise5# ./a.out
Enter the number of Processes:5
1 2 6
2 5 2
3 1 8
4 0 3
5 4 4
PID:4    CT:3    TAT:3    WT:0
PID:1    CT:9    TAT:7    WT:1
PID:2    CT:11   TAT:6    WT:4
PID:5    CT:15   TAT:11   WT:7
PID:3    CT:23   TAT:22   WT:14
Average CT :12.20      Average TAT:9.80      Average WT:5.20
```

3.SRTF.c

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

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

```
void swap(int *a ,int *b){
```

```
int tmp=*a;
```

```
*a=*b;
```

```
*b=tmp;
```

```
}
```

```
int is_vis(int *vis,int pid,int vislen){
```

```
for(int i=0;i<vislen;i++){if(pid==vis[i]){return 1;}}
```

```
return 0;
```

```
}
```

```
void sort(int proc[][3],int len){
```

```
for(int i=0;i<len;i++){
```

```
    for(int j=0;j<len-i-1;j++){
```

```
        if(proc[j][1]>proc[j+1][1]){
```

```
            swap(&proc[j][0],&proc[j+1][0]);
```

```
            swap(&proc[j][1],&proc[j+1][1]);
```

```
            swap(&proc[j][2],&proc[j+1][2]);
```

```
        }  
    }  
}  
  
}
```

```
int min(int proc[][3],int start,int arrlen){  
    int mins =proc[start][2];  
    int i;  
    int minidx;  
    for(i=start;i<arrlen;i++){  
        if(proc[i][2]<=mins){  
            minidx=i;  
            mins=proc[i][2];  
        }  
    }  
    return minidx;  
}
```

```
int main(){  
    int num,bt,at,pid;  
    printf("Enter the number of Processes:");  
    scanf("%d",&num);  
    int proctable[num][3],n,min1=proctable[0][1],minidx;  
    int dup[num][3];  
    int vis[num];  
    for(int i=0;i<num;i++){  
  
        scanf("%d %d %d",&pid,&at ,&bt);  
        dup[i][0]=pid;  
        dup[i][1]=at;  
        dup[i][2]=bt;
```

```
        proctable[i][0]=pid;
        proctable[i][1]=at;
        proctable[i][2]=bt;
    }
    sort(proctable,num);
    sort(dup,num);
    int proj_c=0;
    int v=0;
    int arr[num*num];
    int i=0,f=0,m;
    int ct=0,c_proj=0;
    int g=0;
    int zc=0;
    int ct1[num*num],mn;

    while(zc<num){
        if(proctable[i][2]==0){
            ct1[f++]=ct;
            zc++;
            i++;
            continue;
        }
        if(i==num){
            i=0;
            continue;
        }
        if(is_vis(vis,proctable[i][0],v)==1){
            ct1[f++]=ct;
            ct+=proctable[i][2];
            proctable[i][2]=0;

            arr[g++]=proctable[i][0];
            i++;
            continue;
        }
    }
```

```
}  
if(proctable[i+1][2]<proctable[i][2]){  
    ct+=proctable[i+1][1]-proctable[i][1];  
    proctable[i][2]--;  
    arr[g++]=proctable[i][0];  
    vis[v++]=proctable[i][0];  
}  
else{  
    ct1[f++]=ct;  
    ct+=proctable[i][2];  
    arr[g++]=proctable[i][0];  
    proctable[i][2]=0;  
}  
i++;  
}  
for(int i=0;i<g;i++){  
    printf("PID:%d \t CT :%d \t TAT:%d\t WT:%d \n",arr[i],ct1[i],ct1[i]-  
dup[arr[i]][1],(ct1[i]-dup[arr[i]][1])-dup[arr[i]][2]));  
}  
}
```

Output :

```
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103/Exercise5# gcc SRTF.c  
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103/Exercise5# ./a.out  
Enter the number of Processes:3  
1 0 8  
2 1 4  
3 2 2
```

PID:3	CT :11	TAT:11	WT:11
PID:1	CT :14	TAT:13	WT:9
PID:2	CT :14	TAT:12	WT:10

Result:

Thus the above given algorithms have been implemented and simulated in c and the output was verified .

Ex. No: 6	Implementation of CPU Scheduling Algorithms
06/02/2023	

Aim:

To Implement the following CPU Algorithms Using C

- 1.Round Robin
- 2.Priority

Alogorithm:

1.RoundRobin:

- 1.Get all inputs required
2. Maintain a time quantum and reduce BT with tq
- 3.Print GANTT Chart as we iterate
- 4.Maintain another list where the ct gets updated
- 5.Exit the loop once all BT=0
- 6.Print Average for the required Parameters

2.Priority (Non-Premptive):

- 1.Get all required inputs
- 2.iterate over all processes and choose the process with highest priority in the arrival queue
- 3.Once chosen Simultaneously print the completion time
- 4.tt is added with bt of existing instance of process running
- 5.Exit once we finish iterating the array
6. Print Average for the required Parameters Code:

Code:

RoundRobin.c :

```
#include <stdio.h>
int is_all_zero(int arr[][3],int len){
int flag=0;
for(int i=0;i<len;i++){
```

```
        if(arr[i][2]<=0){
            flag++;
        }
    }
    if(flag>=len){
        return 1;
    }
    return 0;
}
void swap(int *a ,int *b){
    int tmp=*a;
    *a=*b;
    *b=tmp;
}
void sort(int proc[][3],int len){
    int tmp_pid,tmp_at,tmp_bt;
    for(int i=0;i<len;i++){
        for(int j=0;j<len-i-1;j++){
            if(proc[j][1]>proc[j+1][1]){

                swap(&proc[j][0],&proc[j+1][0]);
                swap(&proc[j][1],&proc[j+1][1]);
                swap(&proc[j][2],&proc[j+1][2]);

            }
        }
    }
}

int main(){
    int pid,tq;
    printf("Enter NO of procs and time quantum: ");
    scanf("%d %d",&pid,&tq);
    int vis[pid*2],ctr=0;
    int et=0,at,bt,pc_no,i=0;
    int ct[pid];
    int proctable[pid][3],dup[pid][3];
    printf("Enter pid,At,BT\n");
    for(int i=0;i<pid;i++){
        scanf("%d %d %d",&pc_no,&at,&bt);
        proctable[i][0]=pc_no;
        proctable[i][1]=at;
        proctable[i][2]=bt;
        dup[i][0]=pc_no;
        dup[i][1]=at;
        dup[i][2]=bt;
    }
    printf("\ndone\n");
    sort(proctable,pid);
    sort(dup,pid);
    while (1){
        if(is_all_zero(proctable,pid)==1){
            break;
        }
    }
}
```



```
        if(i>=pid){
            i=0;
            continue;
        }
        if(proctable[i][2]<=0){
            i++;
            continue;
        }

        if(et+tq>=proctable[i][1]){
            vis[ctr++]=proctable[i][0];

            proctable[i][2]-=tq;
            et+=tq;
            ct[i]=et;
            i++;

        }
        else{
            i=0;
            vis[ctr++]=proctable[i][0];
            //printf("%d\t",proctable[i][0]);
            //et+=proctable[i][1];
            ct[i]=et;
            i++;
        }
    }

    float tat,wt;
    for(int i=0;i<pid;i++){
        printf("%d\t %d \t %d \n",proctable[i][0],ct[i]-proctable[i][1],ct[i]-
        dup[i][2]);
        tat+=ct[i]-proctable[i][1];
        wt+=ct[i]-dup[i][2];
    }
    printf("\nAverage TAT :%.2f\nAverage WT:%.2f",tat/(float)pid,wt/(float)pid);

}
```

Output:

```
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103/Exercise6# gcc RoundRobin.c
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103/Exercise6# ./a.out
Enter NO of procs and time quantum: 4 2
Enter pid,At,BT
1 0 5
2 1 4
3 2 2
4 4 1

done
1      14      9
2      11      8
3       4       4
4       4       7

Average TAT :8.25
Average WT:7.00root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103/Exercise6#
```

2.Priority(NonPreemptive):

```
#include <stdio.h>

int main()
{
    int pid,tq;
    printf("Enter NO of procs:");
    scanf("%d",&pid);
    int vis[pid],ctr=0;
    int et=0,at,bt,pc_no;
    int ct[pid],priority;
    int proctable[pid][4],dup[pid][3];
    printf("Enter pid,At,BT,priority\n");
    for(int i=0;i<pid;i++){
        scanf("%d %d %d %d",&pc_no,&at,&bt,&priority);
        proctable[i][0]=pc_no;
        proctable[i][1]=at;
        proctable[i][2]=bt;
        proctable[i][3]=priority;
    }
    printf("\ndone\n");
    int i=0,tt=0,j,j_pri;
    float tat=0,wt=0;
    for(int i=0;i<pid;i++){
        vis[i]=0;
```

```
}
for(i=0;i<pid;i++){
    j=-2,j_pri=123123412;
    for(int k=0;k<pid;k++){
        /*find k index using priority if not found and iff within total arr
time*/
        if(proctable[k][1]<=tt && !vis[k]){
            if(proctable[k][3]<j_pri){
                j_pri=proctable[k][3];
                j=k;
            }
        }
    }
    tt+=proctable[i][2];
    printf("%d %d %d %d\n",proctable[i][0],tt,tt-proctable[i][1],tt-
proctable[i][2]);
    tat+=tt-proctable[i][1];
    wt+=tt-proctable[i][2];
    vis[j]=1;
}
printf("Average TAT:%.2f\nAverage WT:%.2f\n",tat/(float)pid,wt/(float)pid);
}
```

Output:



```
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103/Exercise6# ./a.out
Enter NO of procs:5
Enter pid,At,BT,priority
1 0 4 1
2 0 3 2
3 6 7 1
4 11 4 3
5 12 2 2

done
1 4 4 0
2 7 7 4
3 14 8 7
4 18 7 14
5 20 8 18
Average TAT:6.80
Average WT:8.60
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103/Exercise6#
```

Result:

Thus the Above algorithms were simulated and implemented in C .

Ex. No: 7	Creation Of Threads
21/02/2023	

Aim:

To Create Threads Using the pthreads library in C

Algorithm:

- 1.Create a Runner Function that the threads run
2. Create Threads using pthread_create and pass the required parameters
- 3.Incase multiple threads are used use mutex lock for synchronisation
4. Multiple threads can be created by using thread_id in a array
5. Once the thread runs other threads are joined using pthread_join
6. Destroy mutex lock and clean up after thread finished execution

Description:

- 1.pthread_create - create a new thread

Syntax:

```
#include <pthread.h>

int pthread_create(pthread_t *restrict thread,
                  const pthread_attr_t
*restrict attr,          void
*(*start_routine)(void *),
void *restrict arg);
```

Description:

The **pthread_create()** function starts a new thread in the calling

process. The new thread starts execution by invoking `start_routine()`; `arg` is passed as the sole argument of `start_routine()`.

2. pthread_join:

Syntax:

pthread_join - join with a terminated thread

#include <pthread.h>

int pthread_join(pthread_t thread, void **retval); Description:

The **pthread_join()** function waits for the thread specified by *thread* to terminate. If that thread has already terminated, then **pthread_join()** returns immediately. The thread specified by *thread* must be joinable.

3. pthread_attr

Syntax:

#include <pthread.h>

**int pthread_attr_init(pthread_attr_t *attr); int
pthread_attr_destroy(pthread_attr_t *attr);**

Description:

The **pthread_attr_init()** function initializes the thread attributes object pointed to by *attr* with default attribute values. After this call, individual attributes of the object can be set using various related functions (listed under SEE ALSO), and then the object can be used in one or more [pthread_create\(3\)](#) calls that create threads.

4. pthread_exit:

Syntax:

**#include <pthread.h>
noreturn void pthread_exit(void *retval);**

Description:

The **pthread_exit()** function terminates the calling thread and returns a value via *retval* that (if the thread is joinable) is available to another thread in the same process that calls [pthread_join\(3\)](#).

5. pthread_mutex_:

Syntax:

```
#include <pthread.h>
```

```
int pthread_mutex_lock(pthread_mutex_t *mutex);    int  
pthread_mutex_trylock(pthread_mutex_t *mutex);    int  
pthread_mutex_unlock(pthread_mutex_t *mutex)
```

Description:

The mutex object referenced by *mutex* shall be locked by a call to

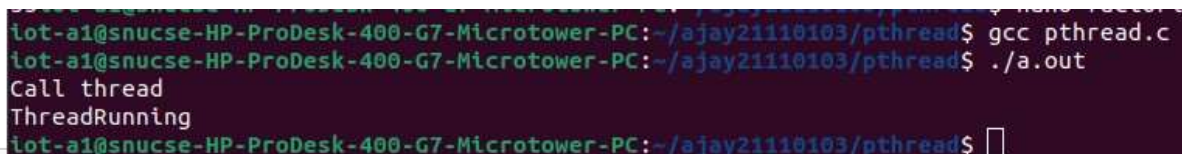
pthread_mutex_lock() that returns zero or **[EOWNERDEAD]**. If the mutex is already locked by another thread, the calling thread shall block until the mutex becomes available. This operation shall return with the mutex object referenced by *mutex* in the locked state with the calling thread as its owner.

Code:

pthread.c

```
#include <stdio.h>  
#include <pthread.h>  
void * runner(void *arg){  
    printf("ThreadRunning\n");  
    return NULL;  
}  
  
int main(){  
    pthread_t t1;  
    printf("Call thread\n");  
    pthread_create(&t1,NULL,runner,NULL);  
    pthread_join(t1,NULL);  
    return 0;  
}
```

Output:



```
lot-a1@snuce-HP-ProDesk-400-G7-Microtower-PC:~/ajay21110103/pthread$ gcc pthread.c  
lot-a1@snuce-HP-ProDesk-400-G7-Microtower-PC:~/ajay21110103/pthread$ ./a.out  
Call thread  
ThreadRunning  
lot-a1@snuce-HP-ProDesk-400-G7-Microtower-PC:~/ajay21110103/pthread$
```

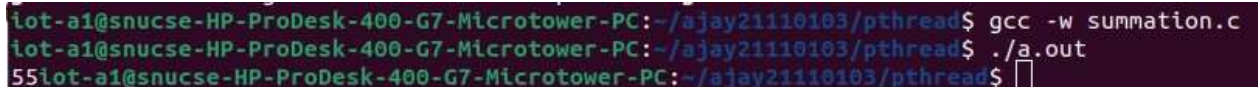
Summation Using Mutex Locks:

Summation.c

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

```
int s;
#define N 5
pthread_t n[N];
pthread_mutex_t msum;
int a[10];
int sum=0;
void* summation(void* param){
    long tid=(long)param;
    int s,e,tsum;
    s=(int)tid*2;
    e=s+2;
    for(int i=s;i<e;i++){
        tsum+=a[i];
    }
    pthread_mutex_lock(&msum);
    sum+=tsum;
    pthread_mutex_unlock(&msum);
    pthread_exit(0);
}
void main(){
    //a={1,2,3,4,5,6,7,8,9,10};
    for(int i=1;i<=10;i++){
        a[i-1]=i;
    }
    pthread_attr_t attr;
    pthread_mutex_init(&msum,NULL);
    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr,PTHREAD_CREATE_JOINABLE);
    for(int i=0;i<N;i++){
        pthread_create(&n[i],&attr,summation,(void*)i);
    }
    pthread_attr_destroy(&attr);
    for(int i=0;i<N;i++){
        pthread_join(n[i],NULL);
    }
    printf("%d",sum);
    pthread_mutex_destroy(&msum);
    pthread_exit(0);
}
```

Output:



```
lot-a1@snuce-HP-ProDesk-400-G7-Microtower-PC:~/ajay21110103/pthread$ gcc -w summation.c
lot-a1@snuce-HP-ProDesk-400-G7-Microtower-PC:~/ajay21110103/pthread$ ./a.out
55lot-a1@snuce-HP-ProDesk-400-G7-Microtower-PC:~/ajay21110103/pthread$
```

Result:

Thus, Threads were created using pthreads library and their usecases observed.

Ex. No: 8	Synchronization Applications
28/02/2023	

Aim:

To write a program to simulate

- 1.Producer Consumer Problem
- 2.Dining Philosopher's Problem

Using semaphores

Algorithm:

1.Producer Consumer Problem:

- 1.Declare a Producer ,consumer function that reads/write to the common buffer depending on BUFFER_LEN
- 2.Initialize buffer sem_t empty,full,buf indices,and item produced as global variable
- 3.In the produce function we produce first and then we wait and lock the mutex to modify the global buffer and its index
- 4.In the consumer function we wait if the buffer is empty and apply lock on mutex as we consume the item and modify index
- 5.Then we apply unlock on both functions
- 6.In the main function create producer,consumer variables as pthread variables
And initialize them using pthread_create ();
- 7.Once all Job has been executed use pthread_join to finish off remaining task by thread and free mutex memory

2.Dining Philosopher's Problem

1. The idea behind this problem is for a person to eat he needs 2 chopsticks which can only be availed only when the neighbour is thinking .This problem has to be solved without any of the philosopher starve/ create any deadlock .
- 2.In the main function we initialize chopstick mutexes and initialize philosopher threads and once the function finishes execution we use pthread_join,pthread_mutex_destroy

3. We define another function eatPhil that prints philosopher k is thinking then apply lock on left and right chopstick and eat then unlock and release both. Then we print who finished eating .

Description:

1. pthread_create - create a new thread

Syntax:

```
#include <pthread.h> int pthread_create(pthread_t *restrict thread, const
pthread_attr_t *restrict attr,
void *(*start_routine)(void *), void *restrict arg);
```

Description:

The pthread_create() function starts a new thread in the calling process. The new thread starts execution by invoking start_routine(); arg is passed as the sole argument of start_routine().

2. pthread_join:

Syntax:

```
pthread_join - join with a terminated thread
#include <pthread.h>
int pthread_join(pthread_t thread, void **retval);
```

Description:

The pthread_join() function waits for the thread specified by thread to terminate. If that thread has already terminated, then pthread_join() returns immediately. The thread specified by thread must be joinable.

3. semaphore.h:

```
#include <semaphore.h>
```

Description:

The <semaphore.h> header defines the sem_t type, used in performing semaphore operations. The semaphore may be implemented using a file descriptor, in which case applications are able to open up at least a total of OPEN_MAX files and semaphores.

4. sem_init():

Syntax:

```
int sem_init(sem_t *sem, int pshared, unsigned int value);
```

Description:

The `sem_init()` function is used to initialise the unnamed semaphore referred to by `sem`. The value of the initialised semaphore is `value`. Following a successful call to `sem_init()`, the semaphore may be used in subsequent calls to `sem_wait()`, `sem_trywait()`, `sem_post()`, and `sem_destroy()`. This semaphore remains usable until the semaphore is destroyed.

5.sem_post:

Syntax:

```
int sem_post(sem_t *sem);
```

Description

The `sem_post()` function unlocks the semaphore referenced by `sem` by performing a semaphore unlock operation on that semaphore.

6.sem_wait(sem t*sem)

Syntax:

```
int sem_wait(sem_t *sem)
```

Description:

The `sem_wait()` function locks the semaphore referenced by `sem` by performing a semaphore lock operation on that semaphore. If the semaphore value is currently zero, then the calling thread will not return from the call to `sem_wait()` until it either locks the semaphore or the call is interrupted by a signal. The `sem_trywait()` function locks the semaphore referenced by `sem` only if the semaphore is currently not locked; that is, if the semaphore value is currently positive. Otherwise, it does not lock the semaphore.

Code:

Procons.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
int item=0;
#define buflen 1
int buf[buflen];
int in=0,out=0;
sem_t empty,full;
pthread_mutex_t mutex;
void produce(void *param){
    do{
        item++;
        sem_wait(&empty);
```

```
pthread_mutex_lock(&mutex);
buf[in]=item;

printf("Producer Produced : %d\n",buf[in]);
in=(in++)%buflen;
pthread_mutex_unlock(&mutex);
sem_post(&full);
}while(1);

}

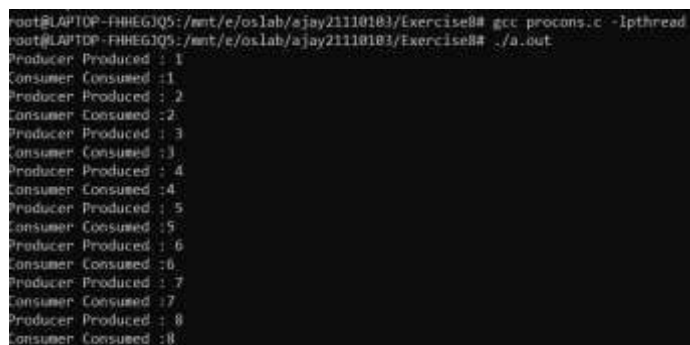
void consume(void *param){
    do{
        sem_wait(&full);
        pthread_mutex_lock(&mutex);

        printf("Consumer Consumed :%d\n",buf[out]);
        out=(out++)%buflen;
        pthread_mutex_unlock(&mutex);
        sem_post(&empty);
    }while(1);
}

int main(){
    pthread_t producer,consumer;
    sem_init (&empty,0,buflen);
    sem_init(&full,0,0);
    pthread_mutex_init(&mutex,NULL);
    pthread_create(&producer,NULL,(void*)produce,NULL);
    pthread_create(&consumer,NULL,(void*)consume,NULL);

    pthread_join(producer,NULL);
    pthread_join(consumer,NULL);
    pthread_mutex_destroy(&mutex);
    sem_destroy(&empty);
    sem_destroy(&full);
}
```

Output:



```
root@LAPTOP-FHHEGJQ5:/mnt/c/oslab/ajay21110103/Exercise8# gcc procons.c -lpthread
root@LAPTOP-FHHEGJQ5:/mnt/c/oslab/ajay21110103/Exercise8# ./a.out
Producer Produced : 1
Consumer Consumed :1
Producer Produced : 2
Consumer Consumed :2
Producer Produced : 3
Consumer Consumed :3
Producer Produced : 4
Consumer Consumed :4
Producer Produced : 5
Consumer Consumed :5
Producer Produced : 6
Consumer Consumed :6
Producer Produced : 7
Consumer Consumed :7
Producer Produced : 8
Consumer Consumed :8
```

DiningPhilosopher.c:

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <time.h>
#define no_philosopher 5
#define no_chopstick 5
pthread_t philosopher[no_philosopher];
pthread_mutex_t chopstick[no_chopstick];
void eatPhil(int k){
    printf("Philosopher %d ->Thinking\n",k );
    pthread_mutex_lock(&chopstick[k]);
    pthread_mutex_lock(&chopstick[(k+1)%no_philosopher]);
    printf("Philosopher %d -> Eating\n",k);
    sleep(1);
    pthread_mutex_unlock(&chopstick[k]);
    pthread_mutex_unlock(&chopstick[(k+1)%no_philosopher]);
    printf("Philosopher %d -> Ate\n",k);}

int main()
{
    for(int i=1;i<=no_chopstick;i++){
        pthread_mutex_init(&chopstick[i],NULL);
    }
    for(int i=1;i<=no_philosopher;i++){
        pthread_create(&philosopher[i],NULL,(void*)eatPhil,(int* )i);
    }
    for(int i=1;i<=no_philosopher;i++){
        pthread_join(philosopher[i],NULL);
    }
    for(int i=1;i<=no_chopstick;i++){
        pthread_mutex_destroy(&chopstick[i]);
    }

    return 0;
}
```

Output:



```
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103/Exercise8# ./a.out
Philosopher 3 ->Thinking
Philosopher 3 -> Eating
Philosopher 2 ->Thinking
Philosopher 4 ->Thinking
Philosopher 5 ->Thinking
Philosopher 5 -> Eating
Philosopher 1 ->Thinking
Philosopher 3 -> Ate
Philosopher 4 -> Eating
Philosopher 2 -> Eating
Philosopher 5 -> Ate
Philosopher 4 -> Ate
Philosopher 2 -> Ate
Philosopher 1 -> Eating
Philosopher 1 -> Ate
```

Result:

Thus the above programs were simulated in C using semaphores and pthreads

Ex. No: 9	Deadlock -Detection and Avoidance
07/03/2023	

Aim:

To write a program to

- 1.Avoid Deadlocks (Banker 's Algorithm)
- 2.Detect Deadlocks

Algorithm:

1. Deadlock Avoidance (Banker's Algorithm):

1. Accept all the required matrices that is allocation,max,available,given
2. Compute the need matrix using $\text{max}[i][j] - \text{alloc}[i][j]$
- 3.Keep a visited array to keep track of sequence
- 4.Traverse through need array if $\text{need}[i][j] \leq \text{available}[i][j]$ then we raise flag and we update our vis array then we mark the need array cost as inf to again traverse the matrix till we get all n process
- 5.If the above is not possible and result in deadlock print no safe state possible
6. Finally print safe state from visited array

2. Deadlock Detection:

- 1.Accept all the required matrices that is allocation,request,available
- 2.Traverse through required array if $\text{request}[i][j] \leq \text{available}[i][j]$ then we raise flag and we update our vis array then we mark the need array cost as inf to again traverse the matrix till we get all n process and add $\text{available}[i][*] + \text{alloc}[i][*]$ as the new available
- 3.if no flag raised during iteration that is no condition met then system is in deadlock
4. If not we print no deadlock detected after the whole iteration finished

Code:

BankersAlgo.c:

```
#include <stdio.h>
int main(){
int n_res,tmp,n_proc;
scanf("%d %d",&n_res,&n_proc);
int given[1][n_res],m=0;
int
allocation_matrix[n_proc][n_res],max[n_proc][n_res],available[n_proc][n_res],need[
n_proc][n_res];
int vis[n_proc];
for(int i=0;i<n_res;i++){
scanf("%d",&tmp);
given[0][i]=tmp;
}
printf("Enter The Allocation Matrix\n");
for(int i=0;i<n_proc;i++){
```

```
        for(int j=0;j<n_res;j++){
            scanf("%d",&tmp);
            allocation_matrix[i][j]=tmp;
        }
    }

    printf("Enter The Maximum Matrix\n");

    for(int i=0;i<n_proc;i++){
        for(int j=0;j<n_res;j++){
            scanf("%d",&tmp);
            max[i][j]=tmp;
        }
    }

    for(int i=0;i<n_proc;i++){
        for(int j=0;j<n_res;j++){
            need[i][j]=max[i][j]-allocation_matrix[i][j];
        }
    }

    for(int i=0;i<n_proc;i++){
        for(int j=0;j<n_res;j++){
            printf("%d\t",need[i][j]);
        }
        printf("\n");
    }

    for(int i=0;i<n_res;i++){
        scanf("%d",&tmp);
        available[0][i]=tmp;
    }
    int f1=0,f2=0,f3=0;
    int avl_cp[1][n_res];
    //for(int i=0;i<n_proc;i++){
    while(m<n_proc){
        for(int j=0;j<n_proc;j++){
            for(int k=0;k<n_res;k++){
                if(available[0][k]>=need[j][k] ){f1++;}
                else{break;}
            }
            if(f1==3){
                f1=0;
                f2=1;
                for(int p=0;p<n_res;p++){
                    need[j][p]=10203123;
                }
                for(int o=0;o<n_res;o++){
                    avl_cp[0][o]=available[0][o];
                }
                printf("%d\n",j);
                vis[m++]=j;
                for(int l=0;l<n_res;l++){
                    available[0][l]=avl_cp[0][l]+allocation_matrix[j][l];
                }
            }
        }
    }
}
```

```
    }
    if(f2!=1){
        printf("No Safe Sequence Possible\n");
        return -1;
    }
}
printf("Safe Sequence :<");
for(int i=0;i<n_proc;i++){printf("%d\t",vis[i]);}
printf(">\n");
return 1;
}
```

Output:

```
lot-a1@sncse-HP-ProDesk-400-G7-Microtower-PC:~/ajay21110103/deadlocks$ gcc bankersalgo.c
lot-a1@sncse-HP-ProDesk-400-G7-Microtower-PC:~/ajay21110103/deadlocks$ ./a.out
3 5
10 5 7
Enter The Allocation Matrix
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter The Maximum 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
3 3 2
Safe Sequence :<1      3      4      0      2      >
```

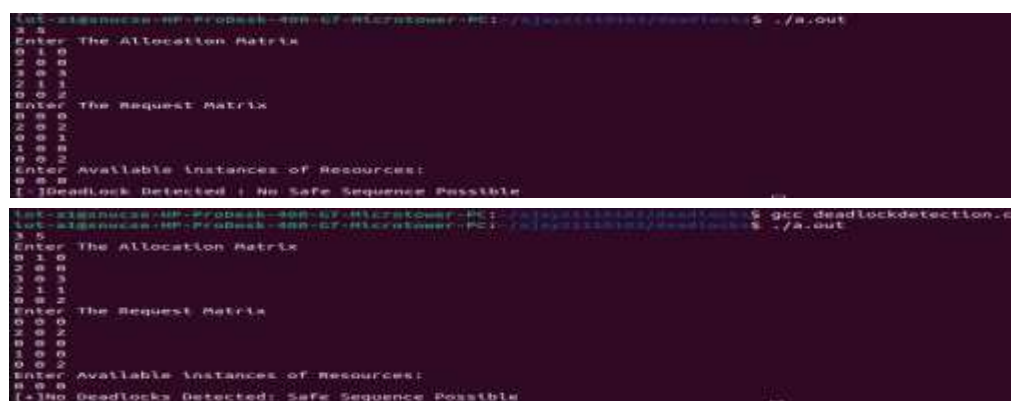
3. **DeadlockDetection.c**

```
#include <stdio.h>
int main(){
    int n_res,tmp,n_proc;
    scanf("%d %d",&n_res,&n_proc);
    int m=0;
    int
    allocation_matrix[n_proc][n_res],request[n_proc][n_res],available[n_proc][n_res];
    printf("Enter The Allocation Matrix\n");
    for(int i=0;i<n_proc;i++){
        for(int j=0;j<n_res;j++){
            scanf("%d",&tmp);
            allocation_matrix[i][j]=tmp;
        }
    }
    printf("Enter The Request Matrix\n");

    for(int i=0;i<n_proc;i++){
        for(int j=0;j<n_res;j++){
            scanf("%d",&tmp);
            request[i][j]=tmp;
        }
    }
}
```

```
printf("Enter Available instances of Resources: \n");
for(int i=0;i<n_res;i++){
    scanf("%d",&tmp);
    available[0][i]=tmp;
}
int f1=0,f2=0,f3=0;
int avl_cp[1][n_res];
while(m<n_proc){
    for(int j=0;j<n_proc;j++){
        for(int k=0;k<n_res;k++){
            if(available[0][k]>=request[j][k]){f1++;}
            else{break;}
        }
        if(f1==3){
            f1=0;
            f2=1;
            for(int p=0;p<n_res;p++){
                request[j][p]=10203123;
            }
            for(int o=0;o<n_res;o++){
                avl_cp[0][o]=available[0][o];
            }
            m++;
            for(int l=0;l<n_res;l++){
                available[0][l]=avl_cp[0][l]+allocation_matrix[j][l];
            }
        }
    }
    if(f2!=1){
        printf("[-]DeadLock Detected : No Safe Sequence Possible\n");
        return -1;
    }
    f2=0;
}
printf("[+]No Deadlocks Detected: Safe Sequence Possible\n");
return 1;
}
```

Output:



```
luc@siganosec-MP-ProDesk-489-GF-R1C7n2020P-PC1:~/ajay2101102/deadlock$ ./a.out
3.5
Enter The Allocation Matrix
0 1 0
2 0 0
3 0 3
2 1 1
0 0 2
Enter The Request Matrix
0 0 0
2 0 2
0 0 0
1 0 0
0 0 2
Enter Available instances of Resources:
0 0 0
[-]DeadLock Detected : No Safe Sequence Possible

luc@siganosec-MP-ProDesk-489-GF-R1C7n2020P-PC1:~/ajay2101102/deadlock$ gcc deadlockdetection.c
luc@siganosec-MP-ProDesk-489-GF-R1C7n2020P-PC1:~/ajay2101102/deadlock$ ./a.out
3.5
Enter The Allocation Matrix
0 1 0
2 0 0
3 0 3
2 1 1
0 0 2
Enter The Request Matrix
0 0 0
2 0 2
0 0 0
1 0 0
0 0 2
Enter Available instances of Resources:
0 0 0
[+]No Deadlocks Detected: Safe Sequence Possible
```

Result:

Thus the deadlock avoidance and detection algorithms were implemented and studied [

Ex. No: 10	Implementation of Dynamic Storage Allocation Schemes
14/03/2023	

Aim:

To study and implement Dynamic storage mapping strategies mainly

1.*First fit*

2.*Next Fit*

3.*Best Fit*

4.*Worst Fit*

Algorithm:

1.*First Fit:*

- 1.Initialize memory ,sequence array with the required number of blocks and sequences
- 2.After Initialization obtain the memory format /block size and store in the array
- 3.Iterate over the sequence array and the memory array from first and check if $\text{sequence}[i] \leq \text{memory}[j]$ if so set the flag as 1 and subtract $\text{memory}[j] -= \text{sequence}[i]$ and break inner loop
- 4.If flag is 0 then it means there is no fit for the request in the memory at the present time and we indicate the same
- 5.While inside the iteration we print the mapping accordingly

2.*Next Fit:*

- 1.Initialize memory ,sequence array with the required number of blocks and sequences
- 2.After Initialization obtain the memory format /block size and store in the array
- 3.Iterate over the sequence array and the memory array from where the previous sequence was mapped and check if $\text{sequence}[i] \leq \text{memory}[j]$ if so set the flag as 1 and subtract $\text{memory}[j] -= \text{sequence}[i]$ and break inner loop
- 4.If flag is 0 then it means there is no fit for the request in the memory at the present time and we indicate the same
- 5.While inside the iteration we print the mapping accordingly

3. *Best Fit:*

- 1.Initialize memory, sequence array with the required number of blocks and sequences
- 2.After Initialization obtain the memory format /block size and store in the array
- 3.Sort the array (memory sequence) and store it in another variable say j
- 4.Now Iterate over the sequence array and the memory array and check if $sequence[i] \leq memory[j[i]]$ if so set the flag as 1 and subtract $memory[j] -= sequence[i]$ and break inner loop
- 5.If flag is 0 then it means there is no fit for the request in the memory at the present time and we indicate the same
- 6.While inside the iteration we print the mapping accordingly

4.Worst Fit:

- 1.Initialize memory, sequence array with the required number of blocks and sequences
- 2.After Initialization obtain the memory format /block size and store in the array
- 3.Sort the array (memory sequence) and store it in another variable say j
- 4.Now Iterate over the sequence array and the memory array from reverse so as it is in descending order of space and check if $sequence[i] \leq memory[j[i]]$ if so set the flag as 1 and subtract $memory[j] -= sequence[i]$ and break inner loop
- 5.If flag is 0 then it means there is no fit for the request in the memory at the present time and we indicate the same
- 6.While inside the iteration we print the mapping accordingly

Code:

dynamicStorageAlloc.c:

```
#include <stdio.h>
#include <stdlib.h>
void FirstFit(int * mem,int n_blocks,int* seq,int n_seq){
    int count_alloc=0,flag=0;
    for (int i=0;i<n_seq;i++){
        for(int j=0;j<n_blocks;j++){
            if(*(seq+i)<=*(mem+j)){
                printf("%d request sequence allocated to %d
block\n",seq[i],j);
                *(mem+j)-=*(seq+i);
                flag=1;
                break;
            }
        }
        if(flag==0){
            printf("%d request sequence has to wait\n",*(seq+i));
        }
        flag=0;
    }
}
```

```
    }  
}  
  
void NextFit(int * mem,int n_blocks,int* seq,int n_seq){  
    int count_alloc=0,flag=0,c=0;  
    int mem_ptr=0;  
    for (int i=0;i<n_seq;i++){  
        c=0;  
        while(c<n_blocks){  
            if(*(seq+i)<=*(mem+mem_ptr)){  
                printf("%d request sequence allocated to %d  
block\n",seq[i],mem_ptr);  
                *(mem+mem_ptr)-=*(seq+i);  
                flag=1;break;  
            }  
            mem_ptr++;  
            c++;  
            if(mem_ptr>n_blocks){mem_ptr=0;}  
        }  
        if(flag==0){  
            printf("%d request sequence has to wait\n",*(seq+i));  
        }  
        flag=0;  
    }  
}
```

```
int * sort(int *mem_sq,int size){  
    int * arr=(int*)malloc(sizeof(int)*size);  
    int* mem_seq=(int*)malloc(sizeof(int)*size);  
    for(int i=0;i<size;i++){  
        *(mem_seq+i)=*(mem_sq+i);  
    }  
    int ctr=0,tmp;  
    for(int i=0;i<size;i++){  
        for(int j=i+1;j<size;j++){  
            if(*(mem_seq+i)>=*(mem_seq+j)){  
                /*(arr+(ctr))=i;  
                tmp=*(mem_seq+i);  
                *(mem_seq+i)=*(mem_seq+j);  
                *(mem_seq+j)=tmp;  
            }  
        }  
        ctr++;  
    }  
    return mem_seq;  
}
```

```
void BestFit(int *mem1 ,int n_blocks,int *seq,int n_seq){  
    int *mem=mem1;  
    int *arr=sort(mem1,n_blocks);  
    int c=0,flag=0;  
    int * arr1=(int*)malloc(sizeof(int)*n_blocks);  
    for(int i=0;i<n_blocks;i++){  
        for(int j=0;j<n_blocks;j++){
```

```
                if(*(mem+i)==*(arr+j)){
                    *(arr1+(c++))=j;break;
                }
            }
        }
        //for(int i=0;i<n_blocks;i++){
        //    printf("%d\t",mem[arr1[i]]);
        //}

        for(int i=0;i< n_seq;i++){
            for(int j=0;j<n_blocks;j++){
                if(*(seq+i)<=mem[*(arr1+j)]){
                    printf("%d request sequence allocated to %d
block\n",*(seq+i),*(arr1+(n_blocks-j)));
                    *(mem+*(arr1+j))-=(seq+i);
                    flag=1;
                    break;
                }
            }
            if(flag==0){
                printf("%d request sequence has to wait\n",*(seq+i));
            }
            flag=0;
        }
    }

void WorstFit(int *mem1 ,int n_blocks,int *seq,int n_seq){
    int *mem=mem1;
    int *arr=sort(mem1,n_blocks);
    int c=0,flag=0;
    int * arr1=(int*)malloc(sizeof(int)*n_blocks);

    for(int i=0;i<n_blocks;i++){
        for(int j=0;j<n_blocks;j++){
            if(*(mem+i)==*(arr+j)){
                *(arr1+(c++))=j;break;
            }
        }
    }
    // for(int i=0;i<n_blocks;i++){
    //     printf("%d\t",mem[arr1[i]]);
    // }
    //for(int j=0;j<n_blocks;j++){printf("--%d--\n",mem[j]);}
    //printf("====\n");
    for(int i=0;i< n_seq;i++){
        for(int j=n_blocks-1;j>=0;j--){
            if(*(seq+i)<=mem[*(arr1+j)]){
                printf("%d request sequence allocated to %d
block\n",*(seq+i),*(arr1+j));
                mem[arr1[j]]-=*(seq+i);
                flag=1;
                break;
            }
        }
    }
    if(flag==0){
        printf("%d request sequence has to wait\n",*(seq+i));
    }
}
```

```
        }
        flag=0;
    }
    //for(int j=0;j<n_blocks;j++){printf("--%d--\n",mem[arr1[j]]);}

}
void memSet(int *mem1,int*mem2,int size){
    for(int i=0;i<size;i++){
        *(mem1+i)=*(mem2+i);
    }

}

int main(){
    int n_blocks,n_seq,assign_blocks,seq1;
    scanf("%d %d",&n_blocks,&n_seq);
    int * mem=(int *)malloc(n_blocks*sizeof(int));
    int*mem1=(int*)malloc(sizeof(int)*n_blocks);
    int*mem2=(int*)malloc(sizeof(int)*n_blocks);
    int*mem3=(int*)malloc(sizeof(int)*n_blocks);
    int* seq=(int*)malloc(n_seq*sizeof(int));
    printf("Enter the memory block sequence\n");
    for(int i=0;i<n_blocks;i++){
        scanf("%d",&assign_blocks);
        *(mem+i)=assign_blocks;
    }
    memSet(mem1,mem,n_blocks);
    memSet(mem2,mem,n_blocks);
    memSet(mem3,mem,n_blocks);
    printf("Enter the memory request sequence\n");
    for(int i=0;i<n_seq;i++){
        scanf("%d",&seq1);
        *(seq+i)=seq1;
    }

    printf("First Fit: \n");
    FirstFit(mem,n_blocks,seq,n_seq);

    printf("Next Fit: \n");
    NextFit(mem1,n_blocks,seq,n_seq);

    printf("Best Fit: \n");
    BestFit(mem2,n_blocks,seq,n_seq);

    printf("Worst Fit: \n");
    WorstFit(mem3,n_blocks,seq,n_seq);

    //int *arr=sort(mem,n_blocks);
    //for(int i=0;i<n_blocks;i++){printf("%d\t",arr[i]);}

}
```

Output:

```
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103/Exercise10# gcc dynamicStorageAlloc.c
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103/Exercise10# ./a.out
5 4
Enter the memory block sequence
100 500 200 300 600
Enter the memory request sequence
212 417 112 426
First Fit:
212 request sequence allocated to 1 block
417 request sequence allocated to 4 block
112 request sequence allocated to 1 block
426 request sequence has to wait
Next Fit:
212 request sequence allocated to 1 block
417 request sequence allocated to 4 block
112 request sequence allocated to 4 block
426 request sequence has to wait
Best Fit:
212 request sequence allocated to 4 block
417 request sequence allocated to 2 block
112 request sequence allocated to 1 block
426 request sequence allocated to 3 block
Worst Fit:
212 request sequence allocated to 4 block
417 request sequence allocated to 1 block
112 request sequence allocated to 4 block
426 request sequence has to wait
```

Result:

Thus, the Above four Memory allocation Strategies were tested and implemented.

Ex. No: 11	Implementation of Page Replacement Algorithms
28/03/2023	

Aim:

To Implement Page Replacement policies like LRU, Optimal Replacement Policy, MFU, FIFO

Algorithm:

FIFO:

- 1.Iterate through the sequence and check if char found in page table
- 2.If not increment miss variable set the Fifo Pointer to the character
- 3.Print the total page faults

LFU:

- 1.Maintain a miss, in and index variable
- 2.Maintain a frequency table that tracks previous requests
- 3.Replace the one that was least used
- 4.Print the total page Faults

Optimal Replacement Policy:

- 1.Maintain a miss,index variable
- 2.Check for Future Occurrence for all the elements in the page table
- 3.Replace one that has the least of all future occurrences as the need arises.
- 4.Print the total page faults

MFU:

1. Maintain a miss,index variable
2. Maintain a frequency table that tracks previous requests
3. Replace the one that was most used as the need arises
4. Print the total page Faults

Code:

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

int isFound(char * arr,int length,char seq){
    for(int i=0;i<length;i++){
        if(seq==*(arr+i)){
```

```
        return 1;
    }
}
return 0;
}

void FIFO(char* table, char* arr, int n_pages, int length){
    //printf("%s", arr);
    int miss=0, ptr=0;
    for(int i=0; i<=length; i++){
        if(isFound(table, n_pages, arr[i])==0){
            miss++; //printf("55");
            *(table+((ptr++)%n_pages))=*(arr+i);
        }
        for(int k=0; k<n_pages; k++){
            printf("%c\t", table[k]);
        }
        printf("\n===== \n");
    }

    printf("\nMiss %d\n", miss);
}

int FindNextOccurrence(char* arr, int length, char charecter, int source){
    int val=0;
    //if(!isFound(table, n_pages, charecter)){
    //}
    for(int i=source; i<length; i++){
        val++;
        if(charecter==*(arr+i)){
            break;
        }
    }
    return val;
}

int min(int *arr, int n_pages){
    int max=*(arr+0), maxidx=0;
    for(int i=0; i<n_pages; i++){
        if(*(arr+i)>max){
            max=*(arr+i);
            maxidx=i;
        }
    }
    return maxidx;
}

void OPTIMAL_REPLACEMENT_POLICY(char* table, char* arr, int n_pages, int length){
    int miss=0;
    int tmp[n_pages];
    int max=0, idx=0;
    for(int i=0; i<length; i++){
        max=0;

        for(int k=0; k<n_pages; k++){
            printf("%c\t", table[k]);
        }
        printf("\n===== \n");
        if(isFound(table, n_pages, *(arr+i))==1){
            continue;
        }
    }
}
```



```
    }
    for(int j=0;j<n_pages;j++){
        if(table[j]=='n'){
            //miss++;
            *(table+j)=*(arr+i);
            break;
        }
        //printf("jj%d\n",FindNextOccurrence(arr,length,table[j],i));
        if(FindNextOccurrence(arr,length,table[j],i)>max){
            max=FindNextOccurrence(arr,length,table[j],i);
            idx=j;
            //printf("\n%d\n",idx);
        }
    }
    miss++;
    table[idx]=*(arr+i);
}
printf("\n %d \n",miss);
}
int FreqPrevious(char* arr, int len ,char charecter,int source){
    int val=0;
    for(int i=source ;i>=0;i--){
        if(charecter==*(arr+i)){
            val++;
        }
    }
    return val;
}
void LFUPolicy(char* table,char*arr,int n_pages,int length){
    int miss=0;
    int min=100;
    int idx=0;
    for(int i=0;i<length;i++){
        min=100;
        for(int k=0;k<n_pages;k++){
            printf("%c\t",table[k]);
        }
        printf("\n===== \n");
        if(isFound(table,n_pages,*(arr+i))){
            continue;
        }
        for(int j=0;j<n_pages;j++){
            if(*(table+j)==-1){
                //miss++;
                *(table+j)=*(arr+i);
                break;
            }
            if(FreqPrevious(arr,length,table[j],i)<min){
                min=FreqPrevious(arr,length,table[j],i);
                idx=j;
            }
        }
        miss++;
        table[idx]=*(arr+i);
    }
}
```

```
        printf("\nMiss%d\n",miss);
    }
    void MFUPolicy(char* table,char*arr,int n_pages,int length){
        int miss=0;
        int max=0;
        int idx=0;
        for(int i=0;i<length;i++){
            max=0;
            if(isFound(table,n_pages,*(arr+i))){
                continue;
            }
            for(int j=0;j<n_pages;j++){
                if(*(table+j)==-1){
                    //miss++;
                    *(table+j)=*(arr+i);
                    break;
                }
                if(FreqPrevious(arr,length,table[j],i)>max){
                    max=FreqPrevious(arr,length,table[j],i);
                    idx=j;
                }
            }
            miss++;
            *(table+idx)=*(arr+i);
        }
        printf("\nMiss%d\n",miss);
    }
    int main(){
        int n,n_pages;
        scanf("%d %d",&n,&n_pages);
        char *table=(char*)malloc(n_pages*sizeof(char));
        char *a=(char*)malloc(n*sizeof(char));
        scanf(" %s",a);
        for(int i=0;i<n_pages;i++){*(table+i)='n';}
        //printf("%d",isFound(a,n,'3'));
        printf("FIFO \n");
        FIFO(table,a,n_pages,n);
        printf("Optimal Replacement Policy\n");
        OPTIMAL_REPLACEMENT_POLICY(table,a,n_pages,n);
        printf("MFU Policy\n");
        MFUPolicy(table,a,n_pages,n);
        printf("LFU Policy\n");
        LFUPolicy(table,a,n_pages,n);
        // printf("==%d",FindNextOccurrence(a,n,'7',3));
    }
}
```

Output:

```
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103/Exercise11# nano PageTablenew.c
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103/Exercise11# gcc PageTablenew.c
root@LAPTOP-FHHEGJQ5:/mnt/e/oslab/ajay21110103/Exercise11# ./a.out
20 3
70120304230321201701
FIFO
7      n      n
=====
7      0      n
=====
7      0      1
=====
2      0      1
=====
2      0      1
=====
2      3      1
=====
2      3      0
=====
4      3      0
=====
4      2      0
=====
4      2      3
=====
0      2      3
=====
0      2      3
=====
0      2      3
=====
0      1      3
=====
0      1      2
=====
0      1      2
=====
0      1      2
=====
7      1      2
=====
7      0      2
=====
7      0      1
=====
0      1
=====
Miss 16
```

```
Miss10
Optimal Replacement Policy
@ 1
=====
7 @ 1
=====
7 @ 1
=====
7 @ 1
=====
2 @ 1
=====
2 @ 1
=====
2 @ 3
=====
2 @ 3
=====
2 4 3
=====
2 4 3
=====
2 4 3
=====
2 @ 3
=====
2 @ 3
=====
2 @ 3
=====
2 @ 1
=====
2 @ 1
=====
2 @ 1
=====
2 @ 1
=====
7 @ 1
=====
7 @ 1
=====
7
MFU Policy
Miss10
LFU Policy
7 @ 1
=====
7 @ 1
=====
7 @ 1
=====
7 @ 1
=====
2 @ 1
=====
2 @ 1
=====
3 @ 1
=====
3 @ 1
=====
4 @ 1
=====
2 @ 1
=====
2 @ 3
=====
2 @ 3
=====
2 @ 3
=====
2 @ 3
=====
1 @ 3
=====
2 @ 3
=====
2 @ 3
=====
2 @ 1
=====
3 @ 7
=====
2 @ 7
=====
Miss10
```

Result:

Thus the Page Replacement Algorithms were implemented and their efficacies were studied.

Ex. No: 12	Implementation of Disk Scheduling Algorithms
28/03/2023	

Aim:

To implement various Disk Scheduling algorithms like FIFO, SCAN, SSTF, CSCAN

Algorithm:

1.FIFO:

- 1.Add seek times from the (disk arm position-arr[i]) FIFO Sequence
- 2.Print Seek time

2.SSTF:

- 1.For SSTF create a duplicate array and copy contents of original array
- 2.add another element to dup which is the diskpos ptr
- 3.Sort the new array
- 4.Add up seek times+=diskpos-arr[i]
- 5.Print the seek time

3.SCAN:

1. For SCAN create a duplicate array and copy contents of original array
2. add another element to dup which is the diskpos ptr
- 3.Sort the new array along with lb,ub,diskptr.
4. iterate from diskpos to left and add up seek times+=(disk arm position-arr[i] till lb
- 5.jump to arr[ub-1] that is the max req seq and iterate from max to disk pos and add seek time
- 6.Print the Seek time

4.CSCAN:

1. For SCAN create a duplicate array and copy contents of original array
2. add another element to dup which is the diskpos ptr
- 3.Sort the new array along with lb,ub,diskptr.
4. iterate from diskpos to left and add up seek times+=(disk arm position-arr[i] till min seq and we do not index lb or ub in this case.
- 5.jump to arr[ub-1] that is the max req seq and iterate from max to disk pos and add seek time

6.Print the Seek time

Code:

```
#include <stdio.h>
#include <stdlib.h>
void swap(int *a1,int*a2){
    int tmp=*a1;
    *a2=*a1;
    *a2=tmp;
}
int seqlen;
int FIFO(int arr[seqlen],int diskarmpos){
    int seek_time=0;
    for(int i=0;i<seqlen;i++){
        seek_time+=abs(diskarmpos-arr[i]);
        diskarmpos=arr[i];
    }
    return seek_time;
}
void sort(int *arr,int seqlen)
{
    int tmp;
    for(int i=0;i<seqlen;i++){
        for(int j=0;j<seqlen;j++){
            if(arr[i]<arr[j]){
                tmp=arr[i];
                arr[i]=arr[j];
                arr[j]=tmp;
            }
        }
    }
}
int CSCAN(int arr[seqlen],int lb,int ub,int diskarmpos){
int dup[seqlen+3],i,seek_time=0;
    for( i=0;i<seqlen;i++){
        dup[i]=arr[i];
    }
    int diskidx;
    dup[i]=diskarmpos;
    dup[i+1]=lb;
    dup[i+2]=ub;
    sort(dup,seqlen+3);
    for(int j=0;j<seqlen+1;j++){
        if(dup[j]==diskarmpos){
            diskidx=j;
            break;
        }
    }
    int cpdi=diskidx;
    diskidx--;
    // printf("%d==\n",diskidx);

    for(int i=diskidx;i>=0;i--){
        seek_time+=abs(diskarmpos-dup[i]);
        diskarmpos=dup[i];
    }
}
```

```
    }
    seek_time+=abs(diskarmpos-dup[seqlen+2]);
    diskarmpos=dup[seqlen+2];
    for(int j=seqlen+1;j>cpdi;j--){
//        printf("%d\n",dup[j]);
        seek_time+=abs(diskarmpos-dup[j]);
        diskarmpos=dup[j];
    }

    return seek_time;
}

int SCAN(int arr[seqlen],int lb,int ub,int diskarmpos ){
int dup[seqlen+3],i,seek_time=0;
    for( i=0;i<seqlen;i++){
        dup[i]=arr[i];
    }
    int diskidx;
    dup[i]=diskarmpos;
    dup[i+1]=lb;
    dup[i+2]=ub;
    sort(dup,seqlen+3);
    for(int j=0;j<seqlen+1;j++){
        if(dup[j]==diskarmpos){
            diskidx=j;
            break;
        }
    }
    int cpdi=diskidx;
    diskidx--;
//    printf("%d==\n",diskidx);

    for(int i=diskidx;i>=0;i--){
        seek_time+=abs(diskarmpos-dup[i]);
        diskarmpos=dup[i];
    }
    for(int j=cpdi+1;j<seqlen+2;j++){
        seek_time+=abs(diskarmpos-dup[j]);
        diskarmpos=dup[j];
//        printf("%d\n",dup[j]);
    }
    return seek_time;
}

int SSTF(int arr[seqlen],int diskarmpos){
    int dup[seqlen+1],i,seek_time=0;
    for( i=0;i<seqlen;i++){
        dup[i]=arr[i];
    }
    int diskidx;
    dup[i]=diskarmpos;
    sort(dup,seqlen+1);
    for(int j=0;j<seqlen+1;j++){
        if(dup[j]==diskarmpos){
            diskidx=j;
            break;
        }
    }
}
```

```
        int cpdi=diskidx;
        diskidx--;
//    printf("%d==\n",diskidx);
    for(int i=diskidx;i>=0;i--){
        seek_time+=abs(diskarmpos-dup[i]);
        diskarmpos=dup[i];
    }
    for(int j=cpdi+1;j<seqlen+1;j++){
        seek_time+=abs(diskarmpos-dup[j]);
        diskarmpos=dup[j];
    }

    return seek_time;
}

int main(){
    int diskarmpos,tmp;
    scanf("%d",&seqlen);
    int arr[seqlen];
    for(int i=0;i<seqlen;i++){
        scanf("%d",&tmp);
        arr[i]=tmp;
    }
    scanf("%d",&diskarmpos);

    printf(" SCAN_MODE | SEEK TIME\n" );

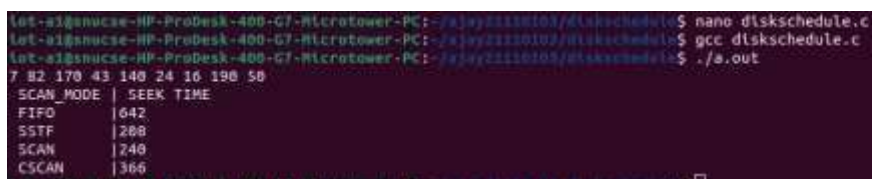
    printf(" FIFO      |%d\n",FIFO(arr,diskarmpos));

    printf(" SSTF      |%d\n",SSTF(arr,diskarmpos));

    printf(" SCAN      |%d\n",SCAN(arr,0,199,diskarmpos));
    printf(" CSCAN     |%d\n",CSCAN(arr,0,199,diskarmpos));

}
```

Output:



```
lat-aigsnucse-HP-ProDesk-400-G7-Microtower-PC:~/ajay21110102/diskchedule$ nano diskschedule.c
lat-aigsnucse-HP-ProDesk-400-G7-Microtower-PC:~/ajay21110102/diskchedule$ gcc diskschedule.c
lat-aigsnucse-HP-ProDesk-400-G7-Microtower-PC:~/ajay21110102/diskchedule$ ./a.out
7 82 170 43 140 24 16 190 50
SCAN_MODE | SEEK TIME
FIFO      |642
SSTF      |208
SCAN      |240
CSCAN     |366
```

Result:

Thus Out of many disk scheduling algorithms 4 of them were implemented and their Seek times were studied

