Sinhgad Technical Education Society’s

RMD Sinhgad Technical Institutes Campus

<u>RMD Sinhgad School of Engineering, Warje, Pune-58</u>

Department of Information Technology

LAB Manual

314446 : Operating Systems Lab

Faculty of Science & Technology

Savitribai Phule Pune University,Pune,

Maharashtra, India

Curriculum For

Third Year of Information Technology

(2019 Course)

(With effect from AY 2021-22)

Savitribai Phule Pune University Third Year of Information Technology (2019 course) (With effect from Academic Year 2021-22)														
Semester-V														
Course Code	Course Name	Teaching Scheme (Hours/ week)			Examination Scheme and Marks						Credit Scheme			
		Theory	Practical	Tutorial	Mid-Sem	End-Sem	Term work	Practical	Oral	Total	Lecture	Practical	Tutorial	Total
314441	Theory of Computation	03	-	-	30	70	-	-	-	100	3	-	-	3
314442	Operating Systems	03	-	-	30	70	-	-	-	100	3	-	-	3
314443	Machine Learning	03	-	-	30	70	-	-	-	100	3	-	-	3
314444	Human Computer Interaction	03	-	-	30	70	-	-	-	100	3	-	-	3
314445	Elective-I	03	-	-	30	70	-	-	-	100	3	-	-	3
314446	Operating Systems Lab	-	04	-	-	-	25	25	-	50	-	2	-	2
314447	Human Computer Interaction- Lab	-	02	-	-	-		-	50	50	-	1		1
314448	Laboratory Practice-I	-	04	-	-	-	25	25		50	-	2	-	2
314449	Seminar	-	01	-	-	-	50	-	-	50	-	1	-	1
314450	Audit Course 5	-	-	-	-	-	-	-	-	-	-	-	-	-
										Total Credit	15	06	-	21
	Total	15	11	-	150	350	100	50	50	700	15	06	-	21

Abbreviations: TH: Theory, TW: Term Work, PR: Practical , OR: Oral ,TUT: Tutorial

Elective-I:	Audit Course 5:
314445A- Design and Analysis of Algorithm	314450A-Banking and Insurance
314445B- Advanced Database and Management System	314450B-Startup Ecosystems
314445C- Design Thinking	314450C- Foreign Language–(Japanese Language- III)
314445D- Internet of Things	

Laboratory Practice-I:

Assignment from Machine Learning and Elective I

Note: Students of T.E. (Information Technology) can opt any one of the audit course from the list of audit courses prescribed by BoS (Information Technology)

Savitribai Phule Pune University, Pune

Third Year Information Technology (2019 Course)

314446 : Operating Systems Lab

Teaching Scheme:	Credit Scheme:	Examination Scheme:
Practical (PR) : 4 hrs/week	02 Credits	PR: 25 Marks TW: 25 Marks

Prerequisites:

1. C Programming
2. Fundamentals of Data Structure

Course Objectives:

1. To introduce and learn Linux commands required for administration.
2. To learn shell programming concepts and applications.
3. To demonstrate the functioning of OS basic building blocks like processes, threads under the LINUX.
4. To demonstrate the functioning of OS concepts in user space like concurrency control (process synchronization, mutual exclusion), CPU Scheduling, Memory Management and Disk Scheduling in LINUX.
5. To demonstrate the functioning of Inter Process Communication under LINUX.
6. To study the functioning of OS concepts in kernel space like embedding the system call in any LINUX kernel.

Course Outcomes:

On completion of the course, students will be able to–

CO1: Apply the basics of Linux commands.

CO2: Build shell scripts for various applications.

CO3: Implement basic building blocks like processes, threads under the Linux.

CO4: Develop various system programs for the functioning of OS concepts in user space like concurrency control, CPU Scheduling, Memory Management and Disk Scheduling in Linux.

CO5: Develop system programs for Inter Process Communication in Linux.

Guidelines for Instructor's Manual

1. The faculty member should prepare the laboratory manual for all the experiments and it shouldbe made available to students and laboratory instructor/Assistant.

Guidelines for Student's Lab Journal

1. Student should submit term work in the form of handwritten journal based on specified list of assignments.
2. Practical Examination will be based on the term work.
3. Candidate is expected to know the theory involved in the experiment.
4. The practical examination should be conducted if and only if the journal of the candidate is complete in all aspects.

Guidelines for Lab /TW Assessment

1. Examiners will assess the term work based on performance of students considering the parameters such as timely conduction of practical assignment, methodology adopted for implementation of practical assignment, timely submission of assignment in the form of handwritten write-up along with results of implemented assignment, attendance etc.
2. Examiners will judge the understanding of the practical performed in the examination by asking some questions related to the theory & implementation of the experiments he/she has carried out.
3. Appropriate knowledge of usage of software and hardware related to respective laboratory should be checked by the concerned faculty member.

Guidelines for Laboratory Conduction

As a conscious effort and little contribution towards Green IT and environment awareness, attaching printed papers of the program in journal may be avoided. There must be hand-written write-ups for every assignment in the journal. The DVD/CD containing student's programs should be attached to the journal by every student and same to be maintained by department/lab In-charge is highly encouraged. For reference one or two journals may be maintained with program prints at Laboratory.

List of Laboratory Assignments

Group A

Assignment No. 1 :

A. Study of Basic Linux Commands: echo, ls, read, cat, touch, test, loops, arithmetic comparison, conditional loops, grep, sed etc.

B. Write a program to implement an address book with options given below: a) Create address book. b) View address book. c) Insert a record. d) Delete a record. e) Modify a record. f) Exit

Assignment No. 2:

Process control system calls: The demonstration of FORK, EXECVE and WAIT system calls along with zombie and orphan states.

A. Implement the C program in which main program accepts the integers to be sorted. Main program uses the FORK system call to create a new process called a child process. Parent process sorts the integers using sorting algorithm and waits for child process using WAIT system call to sort the integers using any sorting algorithm. Also demonstrate zombie and orphan states.

B. Implement the C program in which main program accepts an array. Main program uses the FORK system call to create a new process called a child process. Parent process sorts an array and passes the sorted array to child process through the command line arguments of EXECVE system call. The child process uses EXECVE system call to load new program which display array in reverse order.

Assignment No. 3:

Implement the C program for CPU Scheduling Algorithms: Shortest Job First (Preemptive) and Round Robin with different arrival time.

Assignment No. 4:

A. Thread synchronization using counting semaphores. Application to demonstrate: producer-consumer problem with counting semaphores and mutex.
B. Thread synchronization and mutual exclusion using mutex. Application to demonstrate: Reader-Writer problem with reader priority.

Assignment No. 5:

Implement the C program for Deadlock Avoidance Algorithm: Bankers Algorithm.

Assignment No. 6:
Implement the C program for Page Replacement Algorithms: FCFS, LRU, and Optimal for frame size as minimum three.

Assignment No. 7:

Inter process communication in Linux using following.

A. FIFOS: Full duplex communication between two independent processes. First process accepts sentences and writes on one pipe to be read by second process and second process counts number of characters, number of words and number of lines in accepted sentences, writes this output in a text file and writes the contents of the file on second pipe to be read by first process and displays onstandard output.

B. Inter-process Communication using Shared Memory using System V. Application to demonstrate: Client and Server Programs in which server process creates a shared memory segment and writes the message to the shared memory segment. Client process reads the message from the shared memory segment and displays it to the screen.

Assignment No. 8: Implement the C program for Disk Scheduling Algorithms: SSTF, SCAN, C-Look considering the initial head position moving away from the spindle.

Study Assignment: Implement a new system call in the kernel space, add this new system call in the Linux kernel by the compilation of this kernel (any kernel source, any architecture and any Linux kernel distribution) and demonstrate the use of this embedded system call using C program in user space.

Reference Books:

1. Das, Sumitabha, UNIX Concepts and Applications, TMH, ISBN-10: 0070635463, ISBN-13: 978-0070635463, 4th Edition.
2. Kay Robbins and Steve Robbins, UNIX Systems Programming, Prentice Hall, ISBN-13: 978-0134424071, ISBN-10: 0134424077, 2nd Edition.
3. Mendel Cooper, Advanced Shell Scripting Guide, Linux Documentation Project, Public domain.
4. Yashwant Kanetkar, UNIX Shell Programming, BPB Publication.

Sinhgad Technical Education Society's

RMD Sinhgad School of Engineering,

Warje, Pune-58.

Sinhgad Institutes

CERTIFICATE

This is to certify that Mr./Ms. ……………………………………………….. of Class ***Third Year (T.E.)*** *Branch:- …………………..Div:-……………. Roll No…………Exam Seat No ……………………. has completed. All Practical Assignments in the subject* **Operating Systems Lab (314446)** *satisfactorily* in the department of Information Technology in the *academic year 2021-2022.*

Prof. D. A. Meshram	**Mrs. Sweta Kale**	**Dr. V. V. Dixit**
Subject In-charge	**Head IT**	**Principal**

LIST OF ASSIGNMENTS

Sr. No	Name of the Assignment	Page No	Date	Sign
1	**Assignment No. 1 :** **A.** Study of Basic Linux Commands: echo, ls, read, cat, touch, test, loops, arithmetic comparison, conditional loops, grep, sed etc. **B.** Write a program to implement an address book with options given below: a) Create address book. b) View address book. c) Insert a record. d) Delete a record. e) Modify a record. f) Exit			
2	**Assignment No. 2:** Process control system calls: The demonstration of FORK, EXECVE and WAIT system calls along with zombie and orphan states. **A.** Implement the C program in which main program accepts the integers to be sorted. Main program uses the FORK system call to create a new process called a child process. Parent process sorts the integers using sorting algorithm and waits for child process using WAIT system call to sort the integers using any sorting algorithm. Also demonstrate zombie and orphan states. **B.** Implement the C program in which main program accepts an array. Main program uses the FORK system call to create a new process called a child process. Parent process sorts an array and passes the sorted array to child process through the command line arguments of EXECVE system call. The child process uses EXECVE system call to load new program which display array in reverse order.			
3	**Assignment No. 3:** Implement the C program for CPU Scheduling Algorithms: Shortest Job First (Preemptive) and Round Robin with different arrival time.			
4	**Assignment No. 4:** **A.** Thread synchronization using counting semaphores. Application to demonstrate: producer- consumer problem with counting semaphores and mutex. **B.** Thread synchronization and mutual exclusion using mutex. Application to demonstrate: Reader- Writer problem with reader priority.			
5	**Assignment No. 5:** Implement the C program for Deadlock Avoidance Algorithm: Bankers Algorithm.			
6	**Assignment No. 6:** Implement the C program for Page Replacement Algorithms: FCFS,			

	LRU, and Optimal for frame size as minimum three.			
7	**Assignment No. 7:** Inter process communication in Linux using following. **A. FIFOS:** Full duplex communication between two independent processes. First process accepts sentences and writes on one pipe to be read by second process and second process counts number of characters, number of words and number of lines in accepted sentences, writes this output in a text file and writes the contents of the file on second pipe to be read by first process and displays onstandard output. **B. Inter-process Communication using Shared Memory using System V.** Application to demonstrate: Client and Server Programs in which server process creates a shared memory segment and writes the message to the shared memory segment. Client process reads the message from the shared memory segment and displays it to the screen.			
8	**Assignment No. 8:** Implement the C program for Disk Scheduling Algorithms: SSTF, SCAN, C-Look considering the initial head position moving away from the spindle.			
9	**Study Assignment:** Implement a new system call in the kernel space, add this new system call in theLinux kernel by the compilation of this kernel (any kernel source, any architecture and any Linux kernel distribution) and demonstrate the use of this embedded system call using C program in user space.			

Assignment No. 1

Title: **A.** Study of Basic Linux Commands: echo, ls, read, cat, touch, test, loops, arithmetic comparison, conditional loops, grep, sed etc.
B. Write a program to implement an address book with options given below: a) Create address book. b) View address book. c) Insert a record. d) Delete a record. e) Modify a record. f) Exit

PART A

Theory:

Study of Linux Commands:

echo: echo command in Linux is used to display line of text/string that are passed as an argument. This is a built-in command that is mostly used in shell scripts and batch files to output status text to the screen or a file.
Syntax:

```
echo [option] [string]
```

ls: The Linux ls command allows you to view a list of the files and folders in each directory. You can also use this command to display details of a file, such as the owner of the file and the permissions assigned to the file
Syntax:

```
ls [options] [paths]
```

read command: in Linux system is used to read from a file descriptor. This command read up the total number of bytes from the specified file descriptor into the buffer. If the number or count is zero then this command may detect the errors. But on success, it returns the number of bytes read. Zero indicates the end of the file. If some errors found then it returns -1.
Syntax:

```
Read
```

cat: Cat(concatenate) command is very frequently used in Linux. It reads data from the file and gives their content as output. It helps us to create, view, concatenate files. So let us see some frequently used cat commands.
Syntax:

1) To view a single file Command:

```
$cat filename.
```

2) To view multiple files Command:

```
$cat file1 file2
```

touch: It is used to create a file without any content. The file created using touch command is empty. This command can be used when the user does not have data to store at the time of file creation.
Syntax:

```
touch filename
```

test: On Unix-like operating systems, the **test** command checks file types and compares values.
Syntax:

```
test EXPRESSION
```

grep: The grep filter searches a file for a particular pattern of characters, and displays all lines that contain that pattern. The pattern that is searched in the file is referred to as the regular expression (grep stands for globally search for regular expression and print out).
Syntax:

```
grep [options] pattern [files]
```

sed: SED command in UNIX is stands for stream editor and it can perform lot's of function on file like, searching, find and replace, insertion or deletion. Though most common use of SED command in UNIX is for substitution or for find and replace.
Syntax:

```
sed OPTIONS... [SCRIPT] [INPUTFILE...]
```

Loops:
Looping Statements in Shell Scripting: There are total 3 looping statements which can be used in bash programming

1. while statement
2. for statement
3. until statement

To alter the flow of loop statements, two commands are used they are,

4. break
5. Continue

1. while loop:
Here command is evaluated and based on the result loop will executed, if command raise to false then loop will be terminated.
Syntax:

```
while command
Do
Statement to be executed
 Done
```

2. for loop:
The for loop operate on lists of items. It repeats a set of commands for every item in a list. Here var is the name of a variable and word1 to wordN are sequences of characters separated by spaces (words). Each time the for loop executes, the value of the variable var is set to the next word in the list of words, word1 to wordN.
Syntax:

```
for var in word1 word2 ...wordn do
Statement to be executed done
```

3. until statement:
The until loop is executed as many as times the condition/command evaluates to false. The loop terminates when the condition/command becomes true.
Syntax:

```
until command do
Statement to be executed until command is true done
```

Arithmetic Comparison: Linux or Unix operating system provides the bc command and expr command for doing arithmetic calculations. You can use these commands in bash or shell script also for evaluating arithmetic expressions
Syntax:

bc [-hlwsqv] [long-options] [file ...]

Options:

-h, {- -help } : Print the usage and exit
-i, {- -interactive}: Force interactive mode
-l, {- -mathlib } : Define the standard math library
-w, {- -warn}: Give warnings for extensions to POSIX bc
-s, {- -standard}: Process exactly the POSIX bc language
-q, {- -quiet}: Do not print the normal GNU bc welcome
-v, {- -version}: Print the version number and copyright and quit

PART B

Code:

```
#! /bin/bash
it=0
a=1
while [[ $op -lt 7 ]]
do
        echo enter the option
        echo "1 for create"
        echo "2 for add"
        echo "3 for display"
        echo "4 for search"
        echo "5 for delete"
        echo "6 for modify"
        echo "7 for exit"
        echo "enter u r choice" read op
word="$op"

case "$word" in
1)
        if [ "$op" == "1" ]
                then
                echo "Enter the name for the database"
                read db touch "$db"
        fi
;;

2)
        if [ "$op" == "2" ]
                then
                echo "in which database u want to add records" read db
                echo "enter the no. of records"
                read n
                while [ $it -lt $n ] do
                        echo "enter id:" read id1
                        echo "enter name:"
                        read nm pa1="^[A-Za-z]"
                        while [[ ! $add =~ $pa ]] do
                                echo "enter valid address:" read add
                                done
                        echo "enter address:" read add
                        pa="^[A-Za-z0-9]"
                        while [[ ! $add =~ $pa ]] do
                        echo "enter valid address:" read add
                        done
                        #echo $add
                        echo "enter phone no.:" read ph
                        pat="^[0-9]{10}$"
```

```
                    while [[ ! $ph =~ $pat ]] do
                    echo "please enter phone number as XXXXXXXXXX:" read ph
                    done
                    #echo $ph
                    echo "eter email:" read em
                    patem="^[a-z0-9._%-+]+@[a-z]+\.[a-z]{2,4}$"
                    while [[ ! $em =~ $patem ]] do
                    echo "please enter valid email address" read em
                    done
                    #echo $em
                    echo "$id1,$nm,$add,$ph,$em" >> "$db" it=`expr $it + 1`
                    echo "$it record entered" done
            fi
        ;;
 3)
        if [ "$op" == "3" ]
            then
            echo "enter name of database from where data to be display:" read db
            cat $db
        fi
        ;;
4)
        if [ "$op" == "4" ]
        then
            echo "enter name of database from where to search:" read db
            echo "enter email to be search:" read em1
            grep $em1 $db
            echo "record found" else
            echo "not found"
        fi
        ;;
5)
        if [ "$op" == "5" ] then
        echo "enter name of database:"
        read db
        echo "enter id:" read id1
        echo "enter line no. u want to delete:"
        read linenumber
        for line in `grep -n "$id1" $db` do
        number=`echo "$line" | cut -c1` #echo $number
        if [ $number == $linenumber ] then lineRemove="${linenumber}d" sed -i -e "$lineRemove"
        $db echo "record removed"
        fi #echo cat $db done
        fi
        ;;
6)
        if [ "$op" == "6" ]
        then
            echo "enter name of database:" read db
```

```
                echo "enter id:" read id1
                echo "enter line u want to modify:"
                read linenumber
                for line in `grep -n "$id1" "$db"` do
                number=`echo "$line" | cut -c1`
                if [ "$number" == "$linenumber" ] then
                echo "what would u like to change"
                echo "\"id,name,address,mobile,email\"" read edit
                linechange="${linenumber}s"
                sed -i -e "$linechange/.*/$edit/" $db echo record edited
                fi
                done
        fi
        ;;
7)
        echo "bye"
        ;;
        *) echo invalid input
esac
Done
```

Assignment No. 2:
Process control system calls: The demonstration of FORK, EXECVE and WAIT system calls along with zombie and orphan states.

A. Implement the C program in which main program accepts the integers to be sorted. Main program uses the FORK system call to create a new process called a child process. Parent process sorts the integers using sorting algorithm and waits for child process using WAIT system call to sort the integers using any sorting algorithm. Also demonstrate zombie and orphan states.

B. Implement the C program in which main program accepts an array. Main program uses the FORK system call to create a new process called a child process. Parent process sorts an array and passes the sorted array to child process through the command line arguments of EXECVE system call. The child process uses EXECVE system call to load new program which display array in reverse order.

Theory:

Fork() System Call:

System call fork() is used to create processes. It takes no arguments and returns a process ID. The purpose of fork() is to create a new process, which becomes the child process of the caller. After a new child process is created, both processes will execute the next instruction following the fork() system call. Therefore, we have to distinguish the parent from the child. This can be done by testing the returned value of fork():

• If fork() returns a negative value, the creation of a child process was unsuccessful.

• fork() returns a zero to the newly created child process.

• fork() returns a positive value, the process ID of the child process, to the parent. The returned process ID is of type pid_t defined in sys/types.h. Normally, the process ID is an integer. Moreover, a process can use function getpid() to retrieve the process ID assigned to this process.

Theref ore, after the system call to fork(), a simple test can tell which process is the child. Please note that Unix will make an exact copy of the parent's address space and give it to the child. Therefore, the parent and child processes have separate address spaces.

Theref ore, after the system call to fork(), a simple test can tell which process is the child. Note that Unix will make an exact copy of the parent's address space and give it to the child. Theref ore, the parent and child processes have separate address spaces.

Let us take an example:

```
int main() 1

{

printf("Before Forking");

fork();

printf("After Forking");

return 0;
```

If the call to fork() is executed successfully, Unix will

• Make two identical copies of address spaces, one for the parent and the other for the child.

• Both processes will start their execution at the next statement following the fork() call.

If we run this program, we might see the following on the screen: Before Forking

After Forking After Forking

Here printf() statement after fork() system call executed by parent as well as child process.

Both processes start their execution right after the system call fork(). Since both processes have identical but separate address spaces, those variables initialized before the fork() ca l ll have the same values in both address spaces. Since every process has its own address space, any modifications will be independent of the others. In other words, if the parent changes the value of its variable, the modification will only affect the variable in the parent process's address space. Other address spaces created by fork() calls will not be affected even though they have identical variable names.

Consider one simpler example, which distinguishes the parent from the child.

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

void ChildProcess(); /* child process prototype */ void ParentProcess(); /* parent process prototype
*/ int main()

{

pid_t pid; pid = fork(); if (pid == 0)

ChildProcess();

else ParentProcess(); return 0;

}
```

```
void ChildProcess()

{

}

void ParentProcess()

{

}
```

In this program, both processes print lines that indicate (1) whether the line is printed by the child or by the parent process, and (2) the value of variable I.

Part A

Implement the C program in which main program accepts the integers to be sorted. Main program uses the FORK system call to create a new process called a child process. Parent process sorts the integers using sorting algorithm and waits for child process using WAIT system call to sort the integers using any sorting algorithm. Also demonstrate zombie and orphan states.

Code:

```
# include<stdio.h>

# include <stdlib.h>

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

int split ( int[], int , int ); void quickSort(int* ,int, int);

void mergeSort(int arr[],int low,int mid,int high)

{

int i,j,k,l,b[20]; l=low;

i=low; j=mid+1;

while((l<=mid)&&(j<=high)){

if(arr[l]<=arr[j]){

b[i]=arr[l]; l++;

}

else{
```

```
b[i]=arr[j]; j++;

}

i++;

}

if(l>mid){

for(k=j;k<=high;k++){ b[i]=arr[k];

i++;

}

}

else{

for(k=l;k<=mid;k++){ b[i]=arr[k];

i++;

}

}

for(k=low;k<=high;k++)

{

arr[k]=b[k];

}

}

void partition(int arr[],int low,int high)

{

int mid;

if(low<high)

{

double temp;

mid=(low+high)/2; partition(arr,low,mid); partition(arr,mid+1,high); mergeSort(arr,low,mid,high);
```

```
}

}

void display(int a[],int size){ int i;

for(i=0;i<size;i++){ printf("%d\t\t",a[i]);

}

printf("\n");

}

int main()

{

int pid, child_pid; int size,i,status;

/* Input the Integers to be sorted */

printf("Enter the number of Integers to Sort::::\t"); scanf("%d",&size);

int a[size];

int pArr[size]; int cArr[size];

for(i=0;i<size;i++){

printf("Enter number %d:",(i+1)); scanf("%d",&a[i]);

pArr[i]=a[i];

cArr[i]=a[i];

}

/* Display the Enterd Integers */

printf("Your Entered Integers for Sorting\n"); display(a,size);

/* Process ID of the Parent */ pid=getpid();

printf("Current Process ID is : %d\n",pid);

/* Child Process Creation */

printf("[ Forking Child Process ... ] \n");

child_pid=fork(); /* This will Create Child Process and Returns Child's PID */
```

```
if( child_pid < 0){

/* Process Creation Failed ... */

printf("\nChild Process Creation Failed!!!!\n"); exit(-1);

}

else if( child_pid==0) {

/* Child Process */

printf("\nThe Child Process\n");

printf("\nchild process is %d",getpid());

printf("\nparent of child process is %d",getppid());

printf("Child is sorting the list of Integers by QUICK SORT::\n"); quickSort(cArr,0,size-1);

printf("The sorted List by Child::\n"); display(cArr,size);

printf("Child Process Completed ...\n"); sleep(10);

printf("\nparent of child process is %d",getppid());

}

else {

/* Parent Process */

printf("parent process %d started\n",getpid()); printf("Parent of parent is %d\n",getppid());

sleep(30);

printf("The Parent Process\n");

printf("Parent %d is sorting the list of Integers by MERGE SORT\n",pid); partition(pArr,0,size-1);

printf("The sorted List by Parent::\n"); display(pArr,size);

wait(&status);

printf("Parent Process Completed ...\n");

}

return 0;

}
```

```
int split ( int a[ ], int lower, int upper )

{

int i, p, q, t ;

p = lower + 1 ; q = upper ;

i = a[lower] ;

while ( q >= p )

{

while ( a[p] < i ) p++ ;

while ( a[q] > i )

q-- ;

if ( q > p )

{

t = a[p] ; a[p] = a[q] ; a[q] = t ;

}

}

t = a[lower] ; a[lower] = a[q] ; a[q] = t ;

return q ;

}

void quickSort(int a[],int lower, int upper){ int i ;

if ( upper > lower )

{

i = split ( a, lower, upper ) ; quickSort ( a, lower, i - 1 ) ; quickSort ( a, i + 1, upper ) ;

}

}
```

PART B

Implement the C program in which main program accepts an array. Main program uses the FORK system call to create a new process called a child process. Parent process sorts an array and passes the sorted array to child process through the command line arguments of EXECVE system call. The child process uses EXECVE system call to load new program which display array in reverse order.

Code:

```
/*main.c */ #include <stdio.h> #include <stdlib.h> #include <unistd.h> #include <string.h>

int main(int argc, char *argv[])

{

int val[10],ele; pid_t pid; char* cval[10];

char *newenviron[] = {NULL }; int i,j,n,temp;

printf("\nEnter the size for an array: "); scanf("%d",&n);

printf("\nEnter %d elements: ", n); for(i=0;i<n;i++)

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

printf("\nEntered elements are: "); for(i=0;i<n;i++)

printf("\t%d",val[i]);

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

{

for(j=0;j<n-1;j++)

{

if(val[j]>val[j+1])

{

temp=val[j]; val[j]=val[j+1]; val[j+1]=temp;

}

}

}
```

```
printf("\nSorted elements are: "); for(i=0;i<n;i++)

printf("\t%d",val[i]);

printf("\nEnter element to search: "); scanf("%d",&ele);

val[i] = ele;

for (i=0; i < n+1; i++)

{

char a[sizeof(int)];

snprintf(a, sizeof(int), "%d", val[i]);

cval[i] = malloc(sizeof(a)); strcpy(cval[i], a);

}

cval[i]=NULL;

pid=fork(); if(pid==0)

{

execve(argv[1], cval, newenviron); perror("Error in execve call...");

}

}

/*child.c */ #include <stdio.h> #include <stdlib.h> #include <string.h>

int main(int argc, char *argv[],char *en[])

{

int i,j,c,ele; int arr[argc];

for (j = 0; j < argc-1; j++)

{

int n=atoi(argv[j]); arr[j]=n;

}

ele=atoi(argv[j]); i=0;

j=argc-1; c=(i+j)/2;
```

```
while(arr[c]!=ele && i<=j)

{

if(ele > arr[c]) i = c+1;

else

j = c-1;

c = (i+j)/2;

}

if(i<=j)

printf("\nElement Found in the given Array...!!!\n"); else

printf("\nElement Not Found in the given Array...!!!\n");

}
```

Assignment 3

Title: Implement the C program for CPU Scheduling Algorithms: Shortest Job First (Preemptive) and Round Robin with different arrival time.

Program to implement Shortest Job First.

```
#include<stdio.h> void main()

{

int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp; float avg_wt,avg_tat;

printf("Enter number of process:"); scanf("%d",&n);

printf("\nEnter Burst Time:\n"); for(i=0;i<n;i++)

{

printf("p%d:",i+1);

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

p[i]=i+1; //contains process number

}

//sorting burst time in ascending order using selection sort

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

{

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

{

if(bt[j]<bt[pos]) pos=j;

}

temp=bt[i]; bt[i]=bt[pos]; bt[pos]=temp;

temp=p[i]; p[i]=p[pos]; p[pos]=temp;

}

wt[0]=0; //waiting time for first process will be zero

//calculate waiting time
```

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

{

wt[i]=0; for(j=0;j<i;j++)

wt[i]+=bt[j];

total+=wt[i];

}

avg_wt=(float)total/n; //average waiting time

total=0;

printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time"); for(i=0;i<n;i++)

{

tat[i]=bt[i]+wt[i]; //calculate turnaround time

total+=tat[i];

printf("\np%d\t\t %d\t\t %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);

}

avg_tat=(float)total/n; //average turnaround time printf("\n\nAverage Waiting Time=%f",avg_wt);
printf("\nAverage Turnaround Time=%f\n",avg_tat);
```

Program to implement Round Robin with different arrival time.

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

void main()

{

// initlialize the variable name

int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10]; float avg_wt, avg_tat;

printf(" Total number of process in the system: "); scanf("%d", &NOP);

y = NOP; // Assign the number of process to variable y

// Use for loop to enter the details of the process like Arrival time and the Burst Time for(i=0;
i<NOP; i++)
```

```
{

printf("\n Enter the Arrival and Burst time of the Process[%d]\n", i+1); printf(" Arrival time is: \t"); //
Accept arrival time

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

printf(" \nBurst time is: \t"); // Accept the Burst time scanf("%d", &bt[i]);

temp[i] = bt[i]; // store the burst time in temp array

}

// Accept the Time qunat

printf("Enter the Time Quantum for the process: \t"); scanf("%d", &quant);

// Display the process No, burst time, Turn Around Time and the waiting time printf("\n Process No
\t\t Burst Time \t\t TAT \t\t Waiting Time "); for(sum=0, i = 0; y!=0; )

{

if(temp[i] <= quant && temp[i] > 0) // define the conditions

{

sum = sum + temp[i]; temp[i] = 0; count=1;

}

else if(temp[i] > 0)

{

temp[i] = temp[i] - quant; sum = sum + quant;

}

if(temp[i]==0 && count==1)

{

y--; //decrement the process no.

printf("\nProcess No[%d] \t\t %d\t\t\t\t %d\t\t\t %d", i+1, bt[i], sum-at[i], sum- at[i]-bt[i]);

wt = wt+sum-at[i]-bt[i]; tat = tat+sum-at[i]; count =0;

}
if(i==NOP-1)

{

i=0;

}

else if(at[i+1]<=sum)

{

i++;

}

else

{

i=0;

}

}

// represents the average waiting time and Turn Around time avg_wt = wt * 1.0/NOP;

avg_tat = tat * 1.0/NOP;

printf("\n Average Turn Around Time: \t%f", avg_wt); printf("\n Average Waiting Time: \t%f",
avg_tat); getch();

}
```

Conclusion: We have studied program to implement Shortest Job First and Round Robin with different arrival time algorithm.

Assignment No. 4

Title: A. Thread synchronization using counting semaphores. Application to demonstrate: producer-consumer problem with counting semaphores and mutex.

B. Thread synchronization and mutual exclusion using mutex. Application to demonstrate: Reader-Writer problem with reader priority.

PART A

Code:

```
#include<stdio.h>
#include<semaphore.h>
#include<sys/types.h>
#include<pthread.h>
#include<unistd.h>
#include<stdlib.h>
#define BUFFER_SIZE 10

pthread_mutex_t mutex; sem_t empty,full;
int buffer[BUFFER_SIZE]; int counter;
pthread_t tid;
void *producer(); void *consumer();
void insert_item(int); int remove_item();

void initilize()
{
pthread_mutex_init(&mutex,NULL); sem_init(&full,0,0); sem_init(&empty,0,BUFFER_SIZE);
}

void *producer()
{
int item,wait_time; wait_time=rand()%5; sleep(wait_time)%5; item=rand()%10; sem_wait(&empty);
pthread_mutex_lock(&mutex);
printf("Producer produce %d\n\n",item); insert_item(item); pthread_mutex_unlock(&mutex);
sem_post(&full);
}

void *consumer()
{
int item,wait_time; wait_time=rand()%5; sleep(wait_time); sem_wait(&full);
pthread_mutex_lock(&mutex);
item=remove_item();
printf("Consumer consume %d\n\n",item); pthread_mutex_unlock(&mutex); sem_post(&empty);
}

void insert_item(int item)
{
buffer[counter++]=item;
}
```

```
int remove_item()
{
return buffer[--counter];
}


int main()
{
int n1,n2; int i;
printf("Enter number of Producers: "); scanf("%d",&n1);
printf("Enter number of Consumers: "); scanf("%d",&n2);
initilize();
for(i=0;i<n1;i++) pthread_create(&tid,NULL,producer,NULL); for(i=0;i<n2;i++)
pthread_create(&tid,NULL,consumer,NULL);
sleep(5);
exit(0);
}
```

Output:

```
Enter number of Producers: 7 Enter number of Consumers: 6 Producer produce 5
Consumer consume 5
Producer produce 3
Producer produce 6
Consumer consume 6
Producer produce 0
Consumer consume 0
Consumer consume 3
Producer produce 6
Consumer consume 6
Producer produce 2
Consumer consume 2
Producer produce 6
```

PART B

Thread synchronization and mutual exclusion using mutex. Application to demonstrate: Reader-Writer problem with reader priority.

Code:
```
#include<semaphore.h> #include<stdio.h> #include<pthread.h>
# include<bits/stdc++.h> using namespace std;
void *reader(void *); void *writer(void *);
int readcount=0,writecount=0,sh_var=5,bsize[5]; sem_t x,y,z,rsem,wsem;
pthread_t r[3],w[2];
```

```
void *reader(void *i)
{
cout << "\n ";
cout << "\n\n reader-" << i << " is reading";
sem_wait(&z); sem_wait(&rsem); sem_wait(&x); readcount++; if(readcount==1)
sem_wait(&wsem); sem_post(&x); sem_post(&rsem); sem_post(&z);
cout << "\nupdated value :" << sh_var; sem_wait(&x);
readcount--; if(readcount==0)
sem_post(&wsem); sem_post(&x);
}
void *writer(void *i)
{
cout << "\n\n writer-" << i << "is writing"; sem_wait(&y);
writecount++;
```

```
if(writecount==1) sem_wait(&rsem); sem_post(&y); sem_wait(&wsem);
sh_var=sh_var+5; sem_post(&wsem); sem_wait(&y); writecount--;
if(writecount==0) sem_post(&rsem); sem_post(&y);
}
int main()
{
sem_init(&x,0,1); sem_init(&wsem,0,1); sem_init(&y,0,1); sem_init(&z,0,1);
sem_init(&rsem,0,1);
pthread_create(&r[0],NULL,(void *)reader,(void *)0);
pthread_create(&w[0],NULL,(void *)writer,(void *)0);
pthread_create(&r[1],NULL,(void *)reader,(void *)1);
pthread_create(&r[2],NULL,(void *)reader,(void *)2);
pthread_create(&r[3],NULL,(void *)reader,(void *)3);
pthread_create(&w[1],NULL,(void *)writer,(void *)3);
pthread_create(&r[4],NULL,(void *)reader,(void *)4);
pthread_join(r[0],NULL); pthread_join(w[0],NULL); pthread_join(r[1],NULL);
pthread_join(r[2],NULL); pthread_join(r[3],NULL); pthread_join(w[1],NULL);
pthread_join(r[4],NULL);
return(0);
}
```

Output:

```
pthread_create(&w[1],NULL,(void *)writer,(void *)3);
pthread_create(&r[4],NULL,(void *)reader,(void *)4);
pthread_join(r[0],NULL); pthread_join(w[0],NULL); pthread_join(r[1],NULL);
pthread_join(r[2],NULL); pthread_join(r[3],NULL); pthread_join(w[1],NULL);
pthread_join(r[4],NULL);
return(0);
}
```

Output:

```
student@sh-4.4-desktop:~$ gcc rw1.c -lpthread
student@sh-4.4-desktop:~$ ./a.out
-------------------------
reader-0 is reading updated value : 5
writer-0 is writing
-------------------------
reader-1 is reading updated value : 10
-------------------------
reader-2 is reading updated value : 10
-------------------------
reader-3 is reading updated value : 10
writer-3 is writing
-------------------------
reader-4 is reading
```

Assignment 5

Title: Implement the C program for Deadlock Avoidance Algorithm: Bankers Algorithm.

Theory:

Deadlock

A deadlock is a state in which each member of a group waits for another member, including itself, to take action, such as sending a message or more commonly releasing a lock.

Deadlocks are a common problem in multiprocessing systems, parallel computing, and distributed systems, where software and hardware locks are used to arbitrate shared resources and implement process synchronization.

Deadlock characteristics:

1. Mutual Exclusion :One or more than one resource are non-shareable (Only one process can use at a time).
2. Hold and Wait: A process is holding at least one resource and waiting for resources.
3. No pre-emption: A resources cannot be taken from a process unless the process release the resource.
4. Circular wait: A set of processes are waiting for each other in circular form. Deadlock can be prevented by eliminating any of the above four conditions. Methods for handling deadlock:

1. Deadlock prevention or avoidance. I.e. Banker's Algorithm
2. Deadlock Detection and recovery.
3. Ignore the problem altogether.

Banker's Algorithm

The banker's algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an "s-state" check to test for possible activities, before deciding whether allocation should be allowed to continue.

Why Banker's algorithm is named so?

Banker's algorithm is named so because it is used in banking system to check whether loan can be sanctioned to a person or not. Suppose there are n number of account holders in a bank and the total sum of their money is S. If a person applies for a loan, then the bank first subtracts the loan amount from the total money that bank has and if the remaining amount is greater than S then only the loan is sanctioned. It is done because if all the account holders come to withdraw their money, then the bank can easily do it.

In other words, the bank would never allocate its money in such a way that it can no longer satisfy the needs of all its customers. The bank would try to be in safe state always. Following Data structures are used to implement the Banker's Algorithm:

Let 'n' be the number of processes in the system and 'm' be the number of resources types.

Available:

• It is a 1-d array of size 'm' indicating the number of available resources of each type.

• Available[j] = k means there are 'k' instances of resource type Rj Max:

• It is a 2-d array of size 'n*m' that defines the maximum demand of each process in a system.

• Max[i, j] = k means process Pi may request at most 'k' instances of resource type Rj.

Allocation:

• It is a 2-d array of size 'n*m' that defines the number of resources of each type currently allocated to each process.

• Allocation[i, j] = k means process Pi is currently allocated 'k' instances of resource type Rj

Need:

• It is a 2-d array of size 'n*m' that indicates the remaining resource need of each process.

• Need [i, j] = k means process Pi currently need 'k' instances of resource type Rj

• Need [i, j] = Max [i, j] – Allocation [i, j]
Banker's algorithm consists of Safety algorithm and Resource request algorithm
Safety Algorithm
The algorithm for finding out whether or not a system is in a safe state can be described as follows:
1) Let Work and Finish be vectors of length 'm' and 'n' respectively. Initialize: Work = Available
Finish[i] = false; for i=1, 2, 3, 4…. n
2) Find an i such that both
a) Finish[i] = false
b) Needi <= Work
if no such i exists goto step (4)
3) Work = Work + Allocation[i]
Finish[i] = true goto step (2)
4) if Finish [i] = true for all i then the system is in a safe state
Resource-Request Algorithm
Let Requesti be the request array for process Pi. Requesti [j] = k means process Pi wants k instances of resource type Rj. When a request for resources is made by process Pi, the following actions are taken:
1) If Requesti <= Needi
Goto step (2) ; otherwise, raise an error condition, since the process has exceeded its maximum claim.
2) If Requesti <= Available
Goto step (3); otherwise, Pi must wait, since the resources are not available.
3) Have the system pretend to have allocated the requested resources to process Pi by modifying the state as
follows:
Available = Available – Requesti Allocationi = Allocationi + Requesti Needi = Needi– Requesti

Code:
```
// Banker's Algorithm
#include <stdio.h>
int main()
{
// P0, P1, P2, P3, P4 are the Process names here
int n, m, i, j, k;
n = 5; // Number of processes m = 3; // Number of resources
int alloc[5][3] = { { 0, 1, 0 }, // P0 // Allocation Matrix
{ 2, 0, 0 }, // P1
{ 3, 0, 2 }, // P2
{ 2, 1, 1 }, // P3
{ 0, 0, 2 } }; // P4
int max[5][3] = { { 7, 5, 3 }, // P0 // MAX Matrix
{ 3, 2, 2 }, // P1
{ 9, 0, 2 }, // P2
{ 2, 2, 2 }, // P3
{ 4, 3, 3 } }; // P4
int avail[3] = { 3, 3, 2 }; // Available Resources
int f[n], ans[n], ind = 0; for (k = 0; k < n; k++) { f[k] = 0;
}
int need[n][m];
for (i = 0; i < n; i++) { for (j = 0; j < m; j++)
need[i][j] = max[i][j] - alloc[i][j];
}
int y = 0;
for (k = 0; k < 5; k++) { for (i = 0; i < n; i++) { if (f[i] == 0) {
int flag = 0;
for (j = 0; j < m; j++) { if (need[i][j] > avail[j]){ flag = 1;
break;
}
}
if (flag == 0) { ans[ind++] = i;
for (y = 0; y < m; y++) avail[y] += alloc[i][y]; f[i] = 1;
}
}
}
}
printf("Following is the SAFE Sequence\n"); for (i = 0; i < n - 1; i++)
printf(" P%d ->", ans[i]);
printf(" P%d", ans[n - 1]);
return (0);
// This code is contributed by Deep Baldha (CandyZack)
}
```
Conclusion:
Studied how to implement Deadlock Avoidance Algorithm using Bankers Algorithm.

Assignment 6

Title: Implement the C program for Page Replacement Algorithms: FCFS, LRU, and Optimal for frame size as minimum three.

Theory:

In an operating system that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when new page comes in.

Page Fault – A page fault happens when a running program accesses a memory page that is mapped into the virtual address space, but not loaded in physical memory.

Since actual physical memory is much smaller than virtual memory, page faults happen. In case of page fault, Operating System might have to replace one of the existing pages with the newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce the number of page faults.

Page Replacement Algorithms :

1. First In First Out (FIFO) –

This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

2. Optimal Page replacement –

In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

3. Least Recently Used –

In this algorithm page will be replaced which is least recently used.

Code:

1. /*FCFS*/

```
 #include <stdio.h>
int main() {
int referenceString[10], pageFaults = 0, m, n, s, pages, frames; printf("\nEnter the number of
Pages:\t");
scanf("%d", & pages);
printf("\nEnter reference string values:\n"); int(m = 0; m < pages; m++) {
printf("Value No. [%d]:\t", m + 1);
scanf("%d", & referenceString[m]);
}
printf("\n What are the total number of frames:\t"); { scanf("%d", & frames);
}
inttemp[frames];
for (m = 0; m < frames; m++) {
temp[m] = -1;
}
for (m = 0; m < pages; m++) { s = 0;
for (n = 0; n < frames; n++) {
if (referenceString[m] == temp[n]) { s++;
pageFaults--;
}
}
pageFaults++;
if ((pageFaults <= frames) && (s == 0)) { temp[m] = referenceString[m];
} else if (s == 0) {
temp[(pageFaults - 1) % frames] = referenceString[m];
}
printf("\n");
for (n = 0; n < frames; n++) { printf("%d\t", temp[n]);
}
}
printf("\nTotal Page Faults:\t%d\n", pageFaults); return 0;
}
```

Output:

```
Enter the number of Pages: 5 Enter reference string values: Value No.[1]: 4
Value No.[2]: 1
Value No.[3]: 2
Value No.[4]: 4
Value No.[5]: 5
What are the total number of frames: 3 4 - 1 - 1
4 1 - 1
4 1 2
4 1 2
5 1 2
Total number of page faults: 4
```

2. /*LRU*/

```
#include<stdio.h>
int findLRU(int time[], int n) {
int i, minimum = time[0], pos = 0;
for (i = 1; i < n; ++i) {
if (time[i] < minimum) { minimum = time[i]; pos = i;
}
}
return pos;
}
int main() {
int no_of_frames, no_of_pages, frames[10], pages[30], counter = 0, time[10], flag1, flag2, i, j, pos,
faults = 0;
printf("Enter number of frames: "); scanf("%d", & no_of_frames); printf("Enter number of pages: ");
scanf("%d", & no_of_pages); printf("Enter reference string: "); for (i = 0; i < no_of_pages; ++i) {
scanf("%d", & pages[i]);
}
for (i = 0; i < no_of_frames; ++i) { frames[i] = -1;
}
for (i = 0; i < no_of_pages; ++i) { flag1 = flag2 = 0;
for (j = 0; j < no_of_frames; ++j) { if (frames[j] == pages[i]) { counter++;
time[j] = counter; flag1 = flag2 = 1;
break;
}
}
if (flag1 == 0) {
for (j = 0; j < no_of_frames; ++j) { if (frames[j] == -1) {
counter++; faults++;
frames[j] = pages[i]; time[j] = counter; flag2 = 1;
break;
}
}
}
if (flag2 == 0) {
pos = findLRU(time, no_of_frames); counter++;
faults++;
frames[pos] = pages[i];
time[pos] = counter;
}
printf("\n");
for (j = 0; j < no_of_frames; ++j) { printf("%d\t", frames[j]);
}
}
printf("\n\nTotal Page Faults = %d", faults);
return 0;
}
```

Output:

```
Enter number of frames: 3 Enter number of pages: 6
Enter reference string: 5 7 5 6 7 3
5 -1 -1
5 7 -1
5 7 -1
5 7 6
5 7 6
3 7 6
Total Page Faults = 4
```

3. /* Optimal Page Replacement */

```
#include<stdio.h>
int main() {
int no_of_frames, no_of_pages, frames[10], pages[30], temp[10], flag1, flag2, flag3, i, j, k, pos, max,
faults = 0;
printf("Enter number of frames: "); scanf("%d", & no_of_frames);
printf("Enter number of pages: "); scanf("%d", & no_of_pages);
printf("Enter page reference string: ");
for (i = 0; i < no_of_pages; ++i) { scanf("%d", & pages[i]);
}
for (i = 0; i < no_of_frames; ++i) {
frames[i] = -1;
}
for (i = 0; i < no_of_pages; ++i) { flag1 = flag2 = 0;
for (j = 0; j < no_of_frames; ++j) { if (frames[j] == pages[i]) {
flag1 = flag2 = 1; break;
}
}
if (flag1 == 0) {
```

```
for (j = 0; j < no_of_frames; ++j) { if (frames[j] == -1) {
faults++;
frames[j] = pages[i]; flag2 = 1;
break;
}
}
}
if (flag2 == 0) { flag3 = 0;
for (j = 0; j < no_of_frames; ++j) { temp[j] = -1;
for (k = i + 1; k < no_of_pages; ++k) { if (frames[j] == pages[k]) {
temp[j] = k; break;
}
}
}
for (j = 0; j < no_of_frames; ++j) { if (temp[j] == -1) {
pos = j; flag3 = 1; break;
}
}
if (flag3 == 0) { max = temp[0]; pos = 0;
for (j = 1; j < no_of_frames; ++j) { if (temp[j] > max) {
max = temp[j]; pos = j;
}
}
}
frames[pos] = pages[i]; faults++;
}
printf("\n");
for (j = 0; j < no_of_frames; ++j) { printf("%d\t", frames[j]);
}
}
printf("\n\nTotal Page Faults = %d", faults); return 0;
}
```
Output:

```
Enter number of frames: 3 Enter number of pages: 10
Enter page reference string: 2 3 4 2 1 3 7 5 4 3
2 -1 -1
2 3 -1
2 3 4
2 3 4
1 3 4
1 3 4
7 3 4
5 3 4
5 3 4
5 3 4
```
Conclusion:

Studied how to implement c program for Page Replacement Algorithms: FCFS, LRU, and Optimal for frame size as minimum three.

Assignment 7

Title: Inter process communication in Linux using following.

A. FIFOS: Full duplex communication between two independent processes. First process accepts sentences and writes on one pipe to be read by second process and second process counts number of characters, number of words and number of lines in accepted sentences, writes this output in a text file and writes the contents of the file on second pipe to be read by first process and displays on standard output.

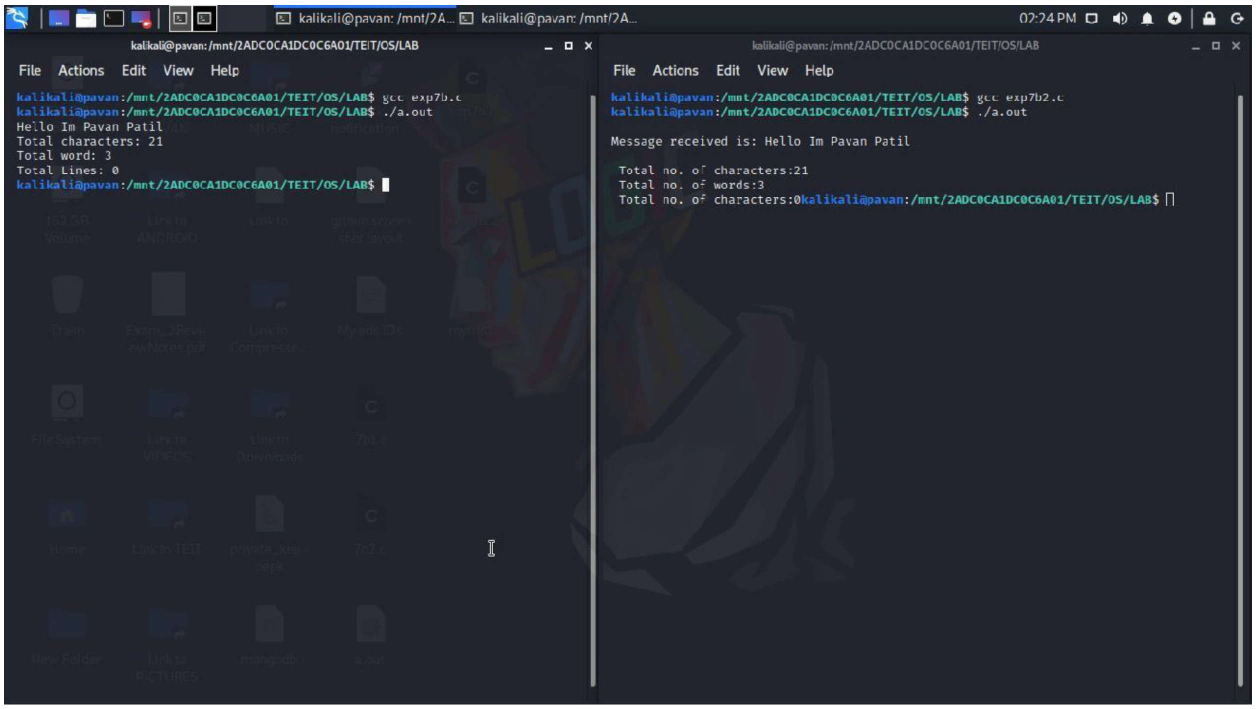
B. Inter-process Communication using Shared Memory using System V. Application to demonstrate: Client and Server Programs in which server process creates a shared memory segment and writes the message to the shared memory segment. Client process reads the message from the shared memory segment and displays it to the screen.

PART A

Code:

```
//FIFO 2
#include<stdio.h> #include<unistd.h> #include<sys/types.h> #include<sys/stat.h> #include<fcntl.h>
#define max_buf 100 int main() {
char * myfifo1 = "myfifo1", * myfifo2 = "myfifo2"; char buf[50];
FILE * fp;
int i = 0, words = 0, lines = 0; mkfifo(myfifo2, 0777);
int fd, fd1;
fd = open(myfifo1, O_RDWR); read(fd, buf, max_buf);
printf("\nMessage received is: %s", buf); while (buf[i] != '\0') {
while (buf[i] == ' ') { words++, i++;
}
if (buf[i] == '.' || buf[i] == '?' || buf[i] == '!') { lines++, i++;
} i++;
}
printf("\n Total no. of characters:%d", i); fp = fopen("abc.txt", "w+");
fprintf(fp, "Total characters=%d", i);
printf("\n Total no. of words:%d", words); fp = fopen("abc.txt", "w+");
fprintf(fp, "Total characters=%d", words);
printf("\n Total no. of characters:%d", lines); fp = fopen("abc.txt", "w+");
fprintf(fp, "Total no. of lines=%d", lines); fclose(fp);
unlink(myfifo1);
fd1 = open(myfifo2, O_RDWR); write(fd1, & i, sizeof(i));
write(fd1, & words, sizeof(words)); write(fd1, & lines, sizeof(lines)); close(fd1);
return 0;
}
```

Output:



```
kalikali@pavan:/mnt/2ADC0CA1DC0C6A01/TEIT/OS/LAB$ gcc exp7b.c
kalikali@pavan:/mnt/2ADC0CA1DC0C6A01/TEIT/OS/LAB$ ./a.out
Hello Im Pavan Patil
Total characters: 21
Total word: 3
Total Lines: 0
kalikali@pavan:/mnt/2ADC0CA1DC0C6A01/TEIT/OS/LAB$
```

```
kalikali@pavan:/mnt/2ADC0CA1DC0C6A01/TEIT/OS/LAB$ gcc exp7b2.c
kalikali@pavan:/mnt/2ADC0CA1DC0C6A01/TEIT/OS/LAB$ ./a.out

Message received is: Hello Im Pavan Patil

 Total no. of characters:21
 Total no. of words:3
 Total no. of characters:0kalikali@pavan:/mnt/2ADC0CA1DC0C6A01/TEIT/OS/LAB$
```

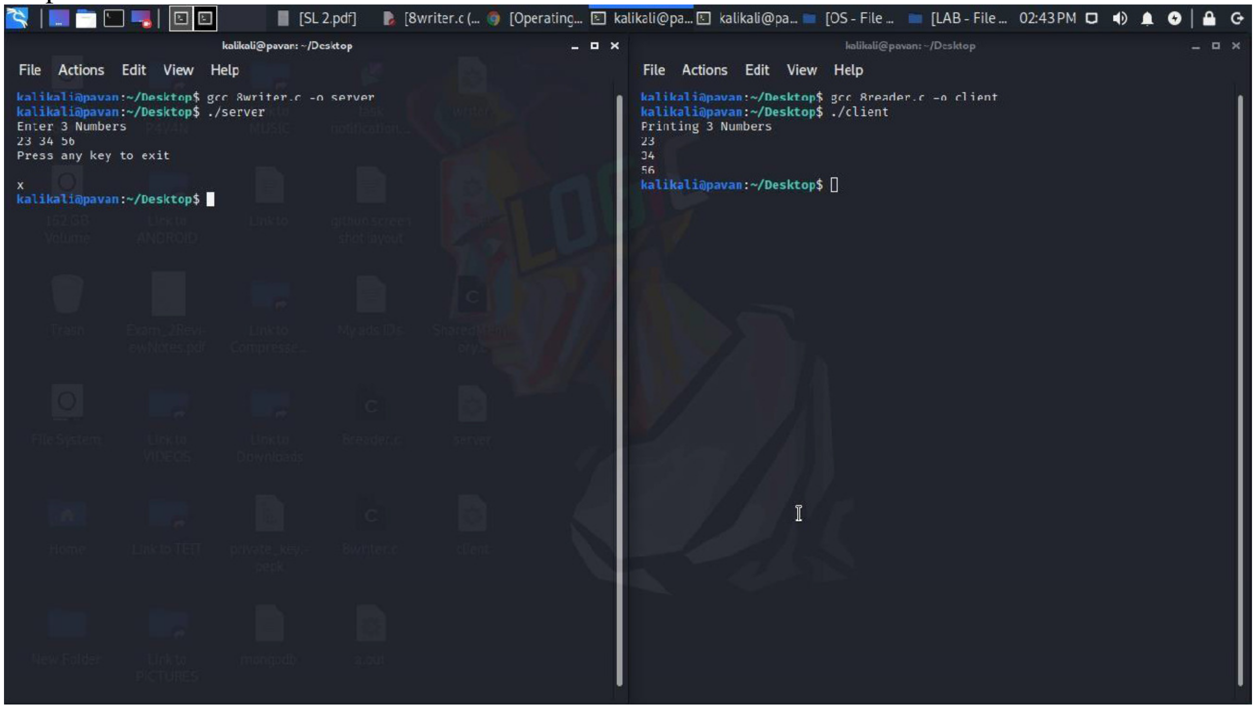
PART B

Code:

```
// reader.c #include <stdio.h>
#include <stdlib.h> #include "SharedMemory.c" int main() {
int shm_id, i;
if ((shm_id = shm_init()) == -1) {
perror("Error occured while initialising Shared Memory\n"); exit(-1);
}
SharedMemory * mSharedMemory = attach(shm_id);
if (mSharedMemory -> status == READ_BY_CLIENT) { printf("Server hasn't written value yet\n");
exit(-1);
}
printf("Printing %d Numbers\n", ARRAY_LENGTH); for (i = 0; i < ARRAY_LENGTH; i++) {
printf("%d\n", mSharedMemory -> array[i]);
}
mSharedMemory -> status = READ_BY_CLIENT; if (detach(mSharedMemory) == -1) {
perror("Error occured while detaching Shared memory\n"); exit(-1);
}
}
//Writer.cc #include <stdio.h> #include <stdlib.h>
#include "SharedMemory.c" int main() {
int shm_id, i;
if ((shm_id = shm_init()) == -1) {
perror("Error occured while initialising Shared Memory\n"); exit(-1);
}
SharedMemory * mSharedMemory = attach(shm_id);
if (mSharedMemory -> status == WRITTEN_BY_SERVER) { printf("Client hasn't read value
yet\n");
exit(-1);
}
printf("Enter %d Numbers\n", ARRAY_LENGTH); for (i = 0; i < ARRAY_LENGTH; i++) {
scanf("%d", & mSharedMemory -> array[i]);
}
mSharedMemory -> status = WRITTEN_BY_SERVER;
if (detach(mSharedMemory) == -1) {
perror("Error occured while detaching Shared memory\n"); exit(-1);
}
char c;
printf("Press any key to exit\n"); scanf(" %c", & c);
}

//sharedMemory.c
 #include <sys/ipc.h>
#include <sys/shm.h>
#define PROJECT_ID 209
#define READ_BY_CLIENT 0
#define WRITTEN_BY_SERVER 1
#define ARRAY_LENGTH 3
// Status holds either value READ_BY_CLIENT or WRITTEN_BY_SERVER
// Server writes into an array of ARRAY_LENGTH and the client reads this typedef struct
SharedMemory {
int status;
int array[ARRAY_LENGTH];
}
SharedMemory;
key_t getKey() {
return ftok(".", PROJECT_ID);
}
int shm_init() {
return shmget(getKey(), sizeof(SharedMemory), IPC_CREAT | 0666);
}
SharedMemory * attach(int shm_id) {
return (SharedMemory * ) shmat(shm_id, NULL, 0);
}
int detach(SharedMemory * shm) { return shmdt((void * ) shm);
}
```

Output:



```
kalikali@pavan:~/Desktop$ gcc 8writer.c -o server
kalikali@pavan:~/Desktop$ ./server
Enter 3 Numbers
23 34 56
Press any key to exit

x
kalikali@pavan:~/Desktop$
```

```
kalikali@pavan:~/Desktop$ gcc 8reader.c -o client
kalikali@pavan:~/Desktop$ ./client
Printing 3 Numbers
23
34
56
kalikali@pavan:~/Desktop$
```

Assignment 8

Title: Implement the C program for Disk Scheduling Algorithms: SSTF, SCAN, C-Look considering the initial head position moving away from the spindle.

Theory:

Disk scheduling is done by operating systems to schedule I/O requests arriving for the disk. Disk scheduling is also known as I/O scheduling.

Disk scheduling is important because:

• Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by the disk controller. Thus, other I/O requests need to wait in the waiting queue and need to be scheduled.

• Two or more request may be far from each other so can result in greater disk arm movement.

• Hard drives are one of the slowest parts of the computer system and thus need to be accessed in an efficient manner.

There are many Disk Scheduling Algorithms but before discussing them let’s have a quick look at some of the important terms:

• Seek Time: Seek time is the time taken to locate the disk arm to a specified track where the data is to be read or write. So the disk scheduling algorithm that gives minimum average seek time is better.

• Rotational Latency: Rotational Latency is the time taken by the desired sector of disk to rotate into a position so that it can access the read/write heads. So the disk scheduling algorithm that gives minimum rotational latency is better.

• Transfer Time: Transfer time is the time to transfer the data. It depends on the rotating speed of the disk and number of bytes to be transferred.

• Disk Access Time: Disk Access Time is:

Disk Access Time = Seek Time + Rotational Latency + Transfer Time

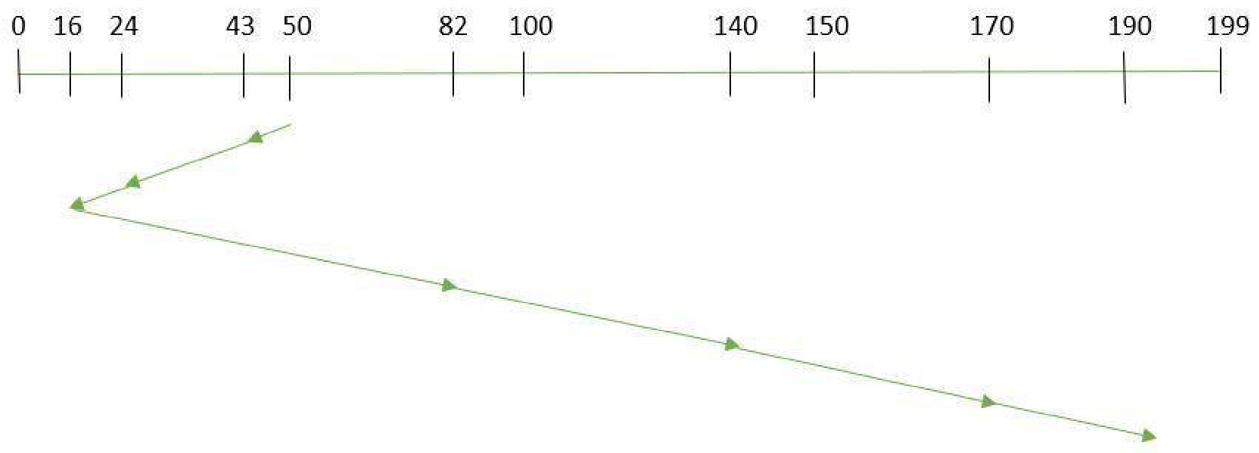
• Disk Response Time: Response Time is the average of time spent by a request waiting to perform its I/O operation. Average Response time is the response time of the all requests. Variance Response Time is measure of how individual request are serviced with respect to average response time. So the disk scheduling algorithm that gives minimum variance response time is better.

Disk scheduling Algorithm:

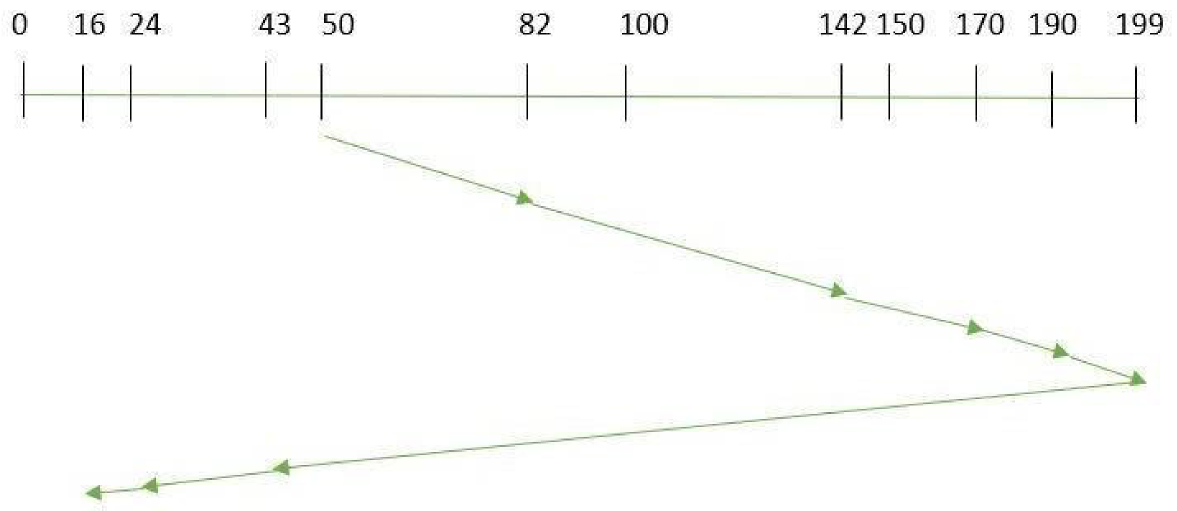
1. SSTF: In SSTF (Shortest Seek Time First), requests having shortest seek time are executed first. So, the seek time of every request is calculated in advance in the queue and then they are scheduled according to their calculated seek time. As a result, the request near the disk arm will get executed first. SSTF is certainly an improvement over FCFS as it decreases the average response time and increases the throughput of system.

Example:

Suppose the order of request is- (82,170,43,140,24,16,190) And current position of Read/Write head is : 50

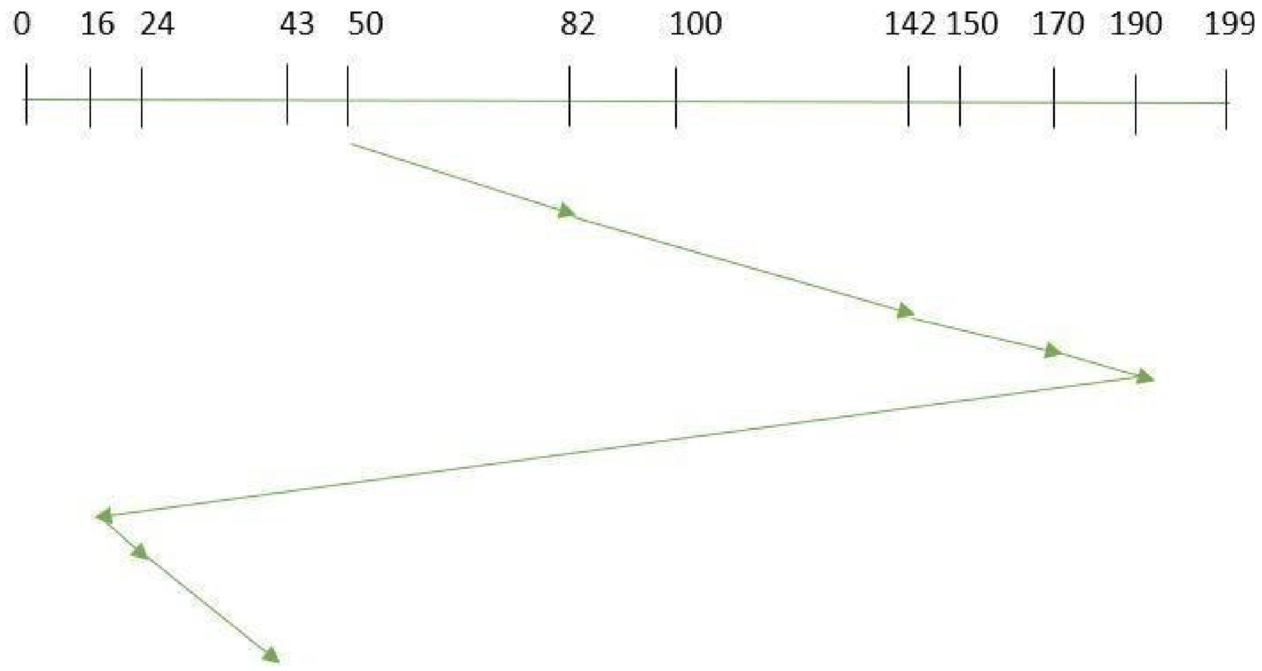


So, total seek time:
= (50-43) +(43-24) +(24-16) +(82-16) +(140-82) +(170-40) +(190-170)
=208
2. SCAN: In SCAN algorithm the disk arm moves into a particular direction and services the requests coming in its path and after reaching the end of disk, it reverses its direction and again services the request arriving in its path. So, this algorithm works as an elevator and hence also known as elevator algorithm. As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait.
Example:
Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move "Towards the larger value".



Therefore, the seek time is calculated as:
= (199-50) +(199-16)
=332
3. C-Look: As LOOK is like SCAN algorithm, in similar way, CLOOK is similar to CSCAN disk scheduling algorithm. In CLOOK, the disk arm despite going to the end goes only to the last request to be serviced in front of the head and then from there goes to the other end's last request. Thus, it also prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.
Example:

Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move.



So, the seek time is calculated as:
= (190-50) +(190-16) +(43-16)
=341

Code:

1. SSTF

```
 #include<stdio.h>
#include<stdlib.h>
 int main() {
int RQ[100], i, n, TotalHeadMoment = 0, initial, count = 0; printf("Enter the number of Requests\n");
scanf("%d", & n);
printf("Enter the Requests sequence\n"); for (i = 0; i < n; i++)
scanf("%d", & RQ[i]);
printf("Enter initial head position\n"); scanf("%d", & initial);
// logic for sstf disk scheduling
/* loop will execute until all process is completed*/ while (count != n) {
int min = 1000, d, index; for (i = 0; i < n; i++) {
d = abs(RQ[i] - initial); if (min > d) {
min = d; index = i;
}
}
TotalHeadMoment = TotalHeadMoment + min; initial = RQ[index];
// 1000 is for max
// you can use any number RQ[index] = 1000; count++;
}
printf("Total head movement is %d", TotalHeadMoment); return 0;
}
```

Output:

```
Enter the number of Request 8
Enter Request Sequence
95 180 34 119 11 123 62 64
Enter initial head Position 50
Total head movement is 236
```

2. SCAN

```
#include<conio.h>
#include<stdio.h>
int main() {
int i, j, sum = 0, n; int d[20];
int disk; //loc of head int temp, max;
int dloc; //loc of disk in array
clrscr();
printf("enter number of location\t"); scanf("%d", & n);
printf("enter position of head\t"); scanf("%d", & disk);
printf("enter elements of disk queue\n"); for (i = 0; i < n; i++) {
scanf("%d", & d[i]);
}
d[n] = disk; n = n + 1;
for (i = 0; i < n; i++) // sorting disk locations
{
for (j = i; j < n; j++) { if (d[i] > d[j]) { temp = d[i];
d[i] = d[j]; d[j] = temp;
}
}
}
max = d[n];
for (i = 0; i < n; i++) // to find loc of disc in array
{
if (disk == d[i]) { dloc = i;
break;
}
}
for (i = dloc; i >= 0; i--) { printf("%d -->", d[i]);
}
printf("0 -->");
for (i = dloc + 1; i < n; i++) { printf("%d-->", d[i]);
}
sum = disk + max;
printf("\nmovement of total cylinders %d", sum); getch();
return 0;
}
```

Output:

```
Enter no of location 8 Enter position of head 53 Enter elements of disk queue 98
183
37
```

```
122
14
124
65
67
53 -> 37 -> 14 -> 0 -> 65 -> 67 -> 98 -> 122 -> 124 -> 183 ->
Movement of total cylinders 236.
```

3. C-LOOK

```
 #include<stdio.h>
#include<stdlib.h>
int main() {
int RQ[100], i, j, n, TotalHeadMoment = 0, initial, size, move;
printf("Enter the number of Requests\n"); scanf("%d", & n);
printf("Enter the Requests sequence\n"); for (i = 0; i < n; i++)
scanf("%d", & RQ[i]);
printf("Enter initial head position\n"); scanf("%d", & initial);
printf("Enter total disk size\n"); scanf("%d", & size);
printf("Enter the head movement direction for high 1 and for low 0 \n");
scanf("%d", & move);
// logic for C-look disk scheduling
/*logic for sort the request array */ for (i = 0; i < n; i++) {
for (j = 0; j < n - i - 1; j++) {
if (RQ[j] > RQ[j + 1]) {
int temp; temp = RQ[j];
RQ[j] = RQ[j + 1];
RQ[j + 1] = temp;
}
}
}
int index;
for (i = 0; i < n; i++) { if (initial < RQ[i]) { index = i;
break;
}
}
// if movement is towards high value if (move == 1) {
for (i = index; i < n; i++) {
TotalHeadMoment = TotalHeadMoment + abs(RQ[i] - initial); initial = RQ[i];
}
for (i = 0; i < index; i++) {
TotalHeadMoment = TotalHeadMoment + abs(RQ[i] - initial); initial = RQ[i];
}
}
// if movement is towards low value else {
for (i = index - 1; i >= 0; i--) {
TotalHeadMoment = TotalHeadMoment + abs(RQ[i] - initial); initial = RQ[i];
}
for (i = n - 1; i >= index; i--) {
TotalHeadMoment = TotalHeadMoment + abs(RQ[i] - initial); initial = RQ[i];
}
}
printf("Total head movement is %d", TotalHeadMoment); return 0;
}
```

Output:

```
Enter the number of Request 8
Enter the Requests Sequence
95 180 34 119 11 123 62 64
Enter initial head position 50
Enter the head movement direction for high 1 and
for low 0
1
Total head movement is 322
```

Conclusion:
Studied the way to Implement the C program for Disk Scheduling Algorithms: SSTF, SCAN, C-Look considering the initial head position moving away from the spindle.

Study Assignment: kernelspace

Title:

Implement a new system call in the kernel space, add this new system call in the Linux kernel by the compilation of this kernel (any kernel source, any architecture and any Linux kernel distribution) and demonstrate the use of this embedded system call using C program in user space.

Theory:

1. Download the kernel source:

In your terminal type the following command:

wget https://www.kernel.org/pub/linux/kernel/v4.x/linux-4.17.4.tar.xz

wget command : GNU Wget is a free utility for non-interactive download of files from the Web.

2. Extract the kernel source code:

sudo tar -xvf linux-4.17.4.tar.xz -C/usr/src/

tar — Tar stores and extracts files from a tape or disk archive.

-x — extract files from an archive

-v — requested using the –verbose option, when extracting archives

-f — file archive; use archive file or device archive

-C — extract to the directory specified after it.(in this case /usr/src/)

Now, we'll change the directory to where the files are extracted: cd /usr/src/linux-4.17.

3. Define a new system call sys_hello( ):

Create a directory named hello/ and change the directory to hello/:

mkdir hello cd hello Create a file hello.c using your favourite text editor:

gedit hello.c

write the following code in the editor: #include <linux/kernel.h> asmlinkage long sys_hello(void)

{

printk("Hello world\n"); return 0;

}

printk prints to the kernel's log file.

1. Create a "Makefile" in the hello directory:

gedit Makefile

and add the following line to it:

obj-y := hello.o

This is to ensure that the hello.c file is compiled and included in the kernel source code.

4. Adding hello/ to the kernel's Makefile: Go back to the parent dir i.e. cd ../ and open "Makefile"

gedit Makefile

search for core-y in the document, you'll find this line as the second instance of your search:

core-y += kernel/ mm/ fs/ ipc/ security/ crypto/ block/

Add 'hello/' to the end of this line:

core-y += kernel/ mm/ fs/ ipc/ security/ crypto/ block/ hello/

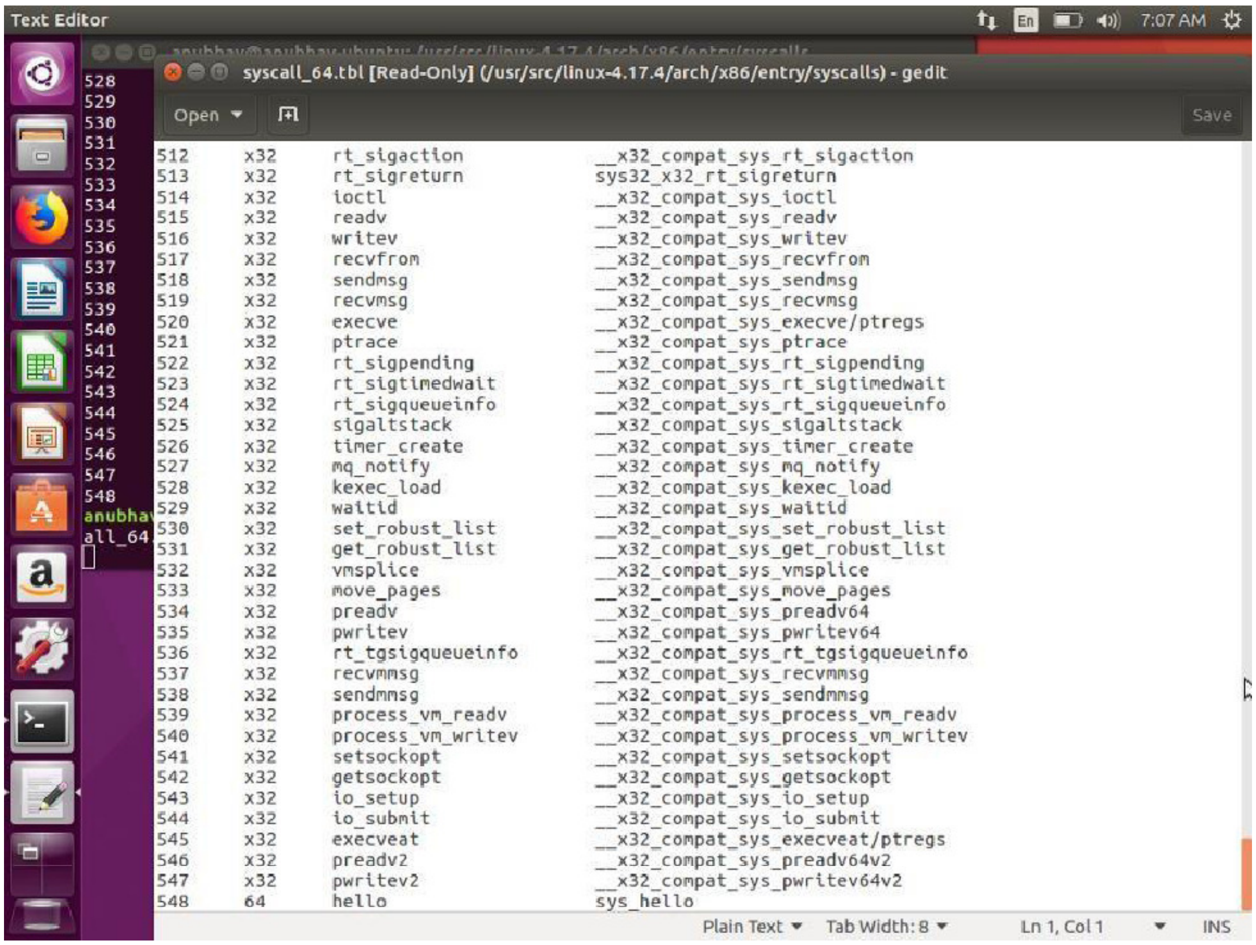
This is to tell the compiler that the source files of our new system call (sys_hello()) are in present in the hello directory.

5. Add the new system call to the system call table:

If you are on a 32-bit system you'll need to change 'syscall_32.tbl'. For 64-bit, change 'syscall_64.tbl'.

Run the following commands in your terminal from linux-4.17.4/ directory: cd arch/x86/entry/syscalls/ gedit syscall_64.tbl

You'll get a file like the following in your editor :



syscall_64.tbl [Read-Only] (/usr/src/linux-4.17.4/arch/x86/entry/syscalls) - gedit

```
512     x32     rt_sigaction            __x32_compat_sys_rt_sigaction
513     x32     rt_sigreturn            sys32_x32_rt_sigreturn
514     x32     ioctl                   __x32_compat_sys_ioctl
515     x32     readv                   __x32_compat_sys_readv
516     x32     writev                  __x32_compat_sys_writev
517     x32     recvfrom                __x32_compat_sys_recvfrom
518     x32     sendmsg                 __x32_compat_sys_sendmsg
519     x32     recvmsg                 __x32_compat_sys_recvmsg
520     x32     execve                  __x32_compat_sys_execve/ptregs
521     x32     ptrace                  __x32_compat_sys_ptrace
522     x32     rt_sigpending           __x32_compat_sys_rt_sigpending
523     x32     rt_sigtimedwait         __x32_compat_sys_rt_sigtimedwait
524     x32     rt_sigqueueinfo         __x32_compat_sys_rt_sigqueueinfo
525     x32     sigaltstack             __x32_compat_sys_sigaltstack
526     x32     timer_create            __x32_compat_sys_timer_create
527     x32     mq_notify               __x32_compat_sys_mq_notify
528     x32     kexec_load              __x32_compat_sys_kexec_load
529     x32     waitid                  __x32_compat_sys_waitid
530     x32     set_robust_list         __x32_compat_sys_set_robust_list
531     x32     get_robust_list         __x32_compat_sys_get_robust_list
532     x32     vmsplice                __x32_compat_sys_vmsplice
533     x32     move_pages              __x32_compat_sys_move_pages
534     x32     preadv                  __x32_compat_sys_preadv64
535     x32     pwritev                 __x32_compat_sys_pwritev64
536     x32     rt_tgsigqueueinfo       __x32_compat_sys_rt_tgsigqueueinfo
537     x32     recvmmsg                __x32_compat_sys_recvmmsg
538     x32     sendmmsg                __x32_compat_sys_sendmmsg
539     x32     process_vm_readv        __x32_compat_sys_process_vm_readv
540     x32     process_vm_writev       __x32_compat_sys_process_vm_writev
541     x32     setsockopt              __x32_compat_sys_setsockopt
542     x32     getsockopt              __x32_compat_sys_getsockopt
543     x32     io_setup                __x32_compat_sys_io_setup
544     x32     io_submit               __x32_compat_sys_io_submit
545     x32     execveat                __x32_compat_sys_execveat/ptregs
546     x32     preadv2                 __x32_compat_sys_preadv64v2
547     x32     pwritev2                __x32_compat_sys_pwritev64v2
548     64      hello                   sys_hello
```

Go to the last of the document and add a new line like so: 548 64 hello sys_hello
Save and exit.
6. Add new system call to the system call header file:
Go to the linux-4.17.4/ directory and type the following commands: cd include/linux/ gedit syscalls.h
Add the following line to the end of the document before the #endif statement: asmlinkage long sys_hello(void);
After this your file will look like so:

Save and exit.
This defines the prototype of the function of our system call. "asmlinkage" is a key word used to indicate that all parameters of the function would be available on the stack.
7. Compile the kernel:
Before starting to compile you need to install a few packages. Type the following commands in your terminal: sudo apt-get install gcc sudo apt-get install libncurses5-dev sudo apt-get install bison sudo apt-get install flex sudo apt-get install libssl-dev sudo apt-get install libelf-dev sudo apt-get update sudo apt-get upgrade

to configure your kernel use the following command in your linux-4.17.4/directory sudo make menuconfig
Once the above command is used to configure the Linux kernel, you will get a pop up window with the list of menus and you can select the items for the new configuration. If your unfamiliar with the configuration just check for the file systems menu and check whether “ext4” is chosen or not, if not select it and save the configuration.
Now to compile the kernel you can use the make command:
sudo make
8. Install / update Kernel:
Run the following command in your terminal:
sudo make modules_install install
It will create some files under /boot/ directory and it will automatically make a entry in your grub.cfg. To check whether it made correct entry, check the files under /boot/ directory . If you have followed the steps without any error you will find the following files in it in addition to others.
System.map-4.17.4 vmlinuz-4.17.4 initrd.img-4.17.4
Now to update the kernel in your system reboot the system . You can use the following command:
shutdown -r now
After rebooting you can verify the kernel version using the following command:
uname -r
It will display the kernel version like so:
4.17.4
9. Test system call:
Go to your home(~) directory using the following commands and create a userspace.c file.
cd ~ gedit userspace.c
Write the following code in this file:

```
#include <stdio.h> #include <linux/kernel.h> #include <sys/syscall.h> #include <unistd.h>
int main()
{
long int amma = syscall(548);
printf("System call sys_hello returned %ld\n", amma); return 0;
}
```

Now, compile and run the program: gcc userspace.c ./a.out
If all the steps are done correctly you’ll get an output like below: System call sys_hello returned 0
Now, to check the message of your kernel run the following command: dmesg This will display Hello world at the end of the kernel’s message.
The use of this embedded system call using C program in user space.
C provides optimized machine instructions for the given input, which increases the performance of the embedded system. Most of the high-level languages rely on libraries, hence they require more memory which is a major challenge in embedded systems ..... Since C does none of that, there is little to no overhead.