LABORATORY MANUAL

CSE325

Operating System Laboratory

Prepared By:

Pushpendra Kumar Pateriya
HoD, System Programming Domain
&
Dr. Allam Mohan
Assistant Professor
System Programming Domain
School of Computer Science & Engineering

Name of the Student: .....

Registration Number/Roll Number: .....

Section and Group: …….....

LOVELY PROFESSIONAL UNIVERSITY

General Guidelines for the students

- Completion of Lab manual exercises will be counted in the CA marks.
- Sit according to roll number and occupy the designated system only. Always use the same system throughout the semester.
- Reporting any issues related to the designated system to the teacher is the student's responsibility.
- Electronic devices should be used on a professional level.
- Do not change the system settings without seeking prior approval from the teacher.
- Self-practice is the key to understand concepts of this course.

Lab 1 - Introduction to Linux Commands

Aim

To study about various shells of different operating systems and commands to perform different functions and operations.

Learning Objective

Students will be able to work on Linux/Windows shell/command prompt using the commands.

Theory

Table 1: Commands

	Commands	
Description	Linux	Windows
Listing le and directories	ls	Dir
Long and Time Formatted List-Ing	ls -alt	Dir
Change directory to abc	cd abc	cd abc
Change to home directory	cd	-
Show current working directory	pwd	cd/chdir
Create a directory abc	mkdir abc	mkdir abc / md abc
Create a file file.txt	touch file.txt	notepad file.txt
Command is used to update the timestamp of a file	touch	
Command is used to delete files	rm	Del
Command is used to concatenate files and print on the monitor	cat	
Copy files from one directory to Another	cp	Copy
Move files from one directory to Another	mv	Move

Exercise

1. Explore the file system of a Window system and a Linux system, and write a prime difference.
2. Create a file in you Linux system, in your current user's home, named as file1.txt, Write your name and Registration number in the file using cat command. Now rename the file using mv command, the new name must be "your registratioinNo.txt".
3. Create a copy of the file you have created with your name. Now delete the original file.
4. Create a directory with your name and move all the files (using mv command) created by you in this directory.
5. Create multiple directories in single command.

Viva Questions

1. How Windows is different from Linux.
2. Differential internal and external commands in Linux.
3. Name any 3 Windows and Linux flavours.
4. List different file systems used in Windows and Linux.
5. What are the different types of files in Linux environment?

Learning Outcomes (What I have Learnt)

S.No.	Parameter	Marks Obtained	Max. Marks
1	Understanding of the student about the Procedure		15
2	Student is able to answer sample viva questions		15
3	Completion of Experiment		20
	Signature of Faculty with date		

Lab 2- Shell Programming

Aim

The aim of this laboratory is to introduce the shell script that offers the student with an interface to include a sequence of commands need to employ frequently for saving time.

Learning Objective

Students will be able to learn the basics of shell scripting to use variables, accept input from a user and perform tests and make decisions.

Description

A shell program, frequently called as a shell script, is basically a program composed of shell commands. Each command within the script is executed by the shell in sequence. Shell script files are created with editors such as vi and stored with the .sh extension. Set execute permission for shell script file using **chmod** command and execute with the **sh** or **bash** command in the terminal.

Shell Variables

User can include user-defined variables in a shell script program using the following format,

var_name=string

EX: day="Sunday"
In the above example, the variable “day” is assigned with the value "Sunday".

Standard input redirection

Mostly shell commands take input values from the standard input (keyboard). The input to these commands can also be redirected from the files using the symbol “<” (less-than character)
Ex: $wc

In the above example, **wc** command accepts the input from the standard input and displays the number of lines, words and characters.

Ex: $wc < test.txt
Here, the input to the **wc** command is redirected form the file “test.txt”

Standard output redirection

The shell command outputs basically directed to standard output. These outputs can also be redirected to files using the symbol “>” (greater-than character)
Ex: ls
In the above example, **ls** command displays the directory content on the standard output device (display).

Ex: $ls > test.txt

Here, the output of the ls command is redirected to the file with the name “test.txt”

Shell arithmetic

The shell enables the arithmetic expressions to be evaluated using the commands **let** or **expr**

Ex: Perform the sum of two numbers

```
x=2
y=3
let "a = $x + $y"
b=`expr $x + $y`
echo "Sum is $a"
echo "Sum is $b"
```

Flow control

The shell supports various commands to control the flow of execution in a program. The basic constructs which provide the flow control are,

- if, if-then, if-then-else, if-then-elif-then-else
- case

<u>if, if-then, if-then-else, if-then-elif-then-else</u>

The if command is fairly simple on the surface; it makes a decision based on the exit status of a command. The if command's syntax looks like this:

```
if <condition>
  then
            commands
 elif <condition>
  then
          commands
else
          commands
fi
```

<u>case construction</u>

The case command evaluates the given test expression and performs the matching operation against each case value to continue the execution of commands. The default condition (*) will be executed with no match is found.. The basic syntax of the case...esac statement is,

```
case  <test-value>  in
  value1)
    <commands>
    ;;
  value2)
    <commands>
    ;;
  value3)
    <commands>
    ;;
      *)  Default statement to be executed
    ;;
Esac
```

Sample program

The following shell program demonstrates the selection of a number with case statement.

```
num="one"

case "$num" in
  "one") echo "Number is 1."
  ;;
  "two") echo " Number is 2."
  ;;
  "three") echo " Number is 3."
  ;;
 *) echo " No Number."
  ;;
esac
```

Loops

Loops in shell scripting are used to execute a set of commands for a certain number of times. These loops will execute the commands repeatedly until a condition fulfils. The basic loops in shell scripting are,

- While loop
- For loop

while loop

The basic format for the while loop is,

```
while [expression]
do
  command-list
done
```

The commands in the while expression are executed to enter into the loop for executing the command-list.

Ex: The following example prints the numbers from 0 to 9.

```
num=0

while [ $num -lt 10 ]
do
    echo $num
    num=`expr $num + 1`
done
```

for loop

The **for** loop executes a set of commands for a specified number of times. The syntax

of for loop in shell scripting is presented as follows,

```
for var in list
do
   command-list
done
```

This for loop includes a number of items in the list and var is a looping variable. The for loop will execute the command-list for each item in the list.

Ex: The following example prints the numbers from 1 to 5.

```
list="one two three four five"
num=1
for i in $list
do
    echo $num
    num=`expr $num + 1`
done
```

Exercise

1. Write a shell script to produce a multiplication table.
2. Write a shell script program to implement a small calculator.
3. Write a shell script to display prime numbers up to the given limit

Viva Questions

1. What do you understand by a shell?
2. How can you define a Shell Variable?
3. What is the alternative to if-else if-else statements in bash?
4. How can we get input in shell script?
5. How set executable permission on a shell script file?

Learning Outcomes (What I have Learnt)

S.No.	Parameter	Marks Obtained	Max. Marks
1	Understanding of the student about the Procedure		15
2	Student is able to answer sample viva questions		15
3	Completion of Experiment		20
	Signature of Faculty with date		

Lab 3- File Manipulation System Calls

Aim
The objective of this laboratory is to introduce the working behind the copy(cp) command. Copy(cp) command uses system calls like open, read, write and lseek to copy the contents of one file to another and read what users enters and write the same in the file.

Learning Objective
Student will be able to learn the working of system calls and the working behind some most used command in Linux shell.

Description of File System calls

This section provides syntax of system calls used in this chapter. More about the system calls can be read using manual page/ help in Linux shell.

- open: to create a file and open the file in read/write and append mode.
- read: to open a file in read mode and read from file or console.
- write: to open a file in write mode and write on file or console.
- close: to close a file.
- lseek: to set the pointer inside the file to a position.

System calls for file management are structures as follows:

Syntax of system calls
open system call

int return = open(“file name",O_CREAT | O_RDONLY | O_WRONLY, 666);

read/write system call

int return = [read/write] (int fd, char buffer, bytes)

lseek system call
int return = lseek(int fd, int offset, whence);

Sample Programs

1. Program to create and open a file using system calls

```
#include<stdio.h>
#include<fcntl.h>
int main()
{
        int n, m;
        n=open(“new_file",O_RDONLY);
        printf(“file descriptor is %d \n",n);
        m=open(“new_file1",O_CREAT | O_WRONLY, 0777);
        printf(“file descriptor is %d \n", m);
}
```

2. Program to read from console and write to console

```
#include<stdio.h>
#include<fcntl.h>
int main()
{
        int n, m;
        char buffer[100];
        n = write(1,”Hello World", 11);
        printf(“No of bytes written %d \n", n);
        m = read( 0, buffer, 12);
        printf(“No of bytes read %d \n", m);
}
```

3. Program to append a file

```
#include<stdio.h>
#include<fcntl.h>
int main()
{
        int a, b, c, d;
        char buffer[100];
        a=open(“new_file2.txt",O_WRONLY|O_APPEND, 0777);
        printf(“file descriptor is %d \n", a);
        b=read(0, buffer, 10);
        c=write(a, buffer, b);
        printf(“value of b:%d , c: %d", b, c);
}
```

4. Program to read and write from and to a file

```
#include<stdio.h>
#include<fcntl.h>
int main()
{
        int a,b,c,d;
        char buffer[100];
        a=open(“new_file2",O_CREAT | O_WRONLY, 0777);
        printf(“file descriptor is %d \n", a);
        b = read(0, buffer, 10);
        c = write(a, buffer, b);
        printf(“the value of b: %d , c:%d \n", b, c);
        return 0;
}
```

Exercise

1. Write a program using system calls to copy half the content of the file to a newly created file.

(a) First half of the file

(b) Second half of the file

2. Write a program using system call to read from console until user enters '$' and print the same on a file.

3. Write a program using system call to read the contents of a file without using char array and display the contents on the console. (Do not use any built in functions like sizeof() and strlen())

Viva Question
1. Which system call is used to reposition the pointer in the file.
2. What is the return value of read system call.
3. What is the return value of open system call.
4. What is passed as the first argument of read/write system call.
5. If a file is to be appended in which mode it will be opened using open system call.
6. What will be used in the lseek system call is the file is to appended.
7. What is the return value of lseek system call.
8. Which system call can be used to find the size of file.
9. How can we find the size of a file using lseek system call.

Learning Outcomes (What I have Learnt)

S.No.	Parameter	Marks Obtained	Max. Marks
1	Understanding of the student about the Procedure		15
2	Student is able to answer sample viva questions		15
3	Completion of Experiment		20
	Signature of Faculty with date		

Lab 4- Directory System Calls

Aim

The objective of this laboratory is to introduce the working behind the command mkdir and ls. These commands use system calls like mkdir, opendir, readdir, to copy the contents of one file to another and read what users enters and write the same in the file.

Learning Objective

Student will be able to learn the working of directory system calls and the working behind some most used command in Linux shell.

Theory

This section provides syntax of system calls used in this chapter. More about the system calls can be read using manual page/ help in Linux shell.

- mkdir: to create directories with append mode.
- opendir: to open a directory stream returning pointer to directory stream.
- readdir: to return a pointer to the next directory entry.
- rmdir: to remove directory (only if empty).

System calls for file management are structures as follows:
Syntax for directory system call

mkdir system call

int mkdir(“pathname/dirname”, mode t mode);
int return = mkdir("directory name",0666);

opendir system call

DIR *opendir("dir name");

readdir system call

struct dirent *readdir(DIR *dirname);
rmdir system call
int rmdir("pathname/dirname");

Dirent Structure

Dirent structure can be read using man readdir as LINUX manual page.

```
struct dirent
{
        Ino_t d_ino; /* inode number */
        off_ t d_off; /* off_set to the next dirent */
        unsigned short d_reclen; /* length of this record */
        unsigned char d_type; /* type of file; not supported by all _le system types */
        char d_name[256]; /* filename */
};
```

Outline of Programs

1. Program to use directory system call and print the contents of the directory

```
#include<stdio.h>
#include<dirent.h>
int main()
{
        DIR *dp;
        struct dirent *dptr;
        int b = mkdir(“Dir1", 0777);
        dp=opendir(“Dir1");
        while(NULL !=(dptr = readdir(dp)))
        {
                printf(“\%s \n", dptr ->d_name);
                printf(\%d \n",dptr->d_type);
        }
return 0;
}
```

Exercise

1. Write a program using directory system calls make a directory on desktop and create a file inside the directory and list the contents of the directory.
2. Write a program using directory and file manipulation system calls to copy the contents of one directory in a newly created directory.

Learning Outcomes (What I have Learnt)

S.No.	Parameter	Marks Obtained	Max. Marks
1	Understanding of the student about the Procedure		15
2	Student is able to answer sample viva questions		15
3	Completion of Experiment		20
	Signature of Faculty with date		

Lab 5 - Operation on Processes

Aim

To introduce the concept of creating a new child process, performing operations on processes and working with orphan and zombie processes.

Learning Objective

Students will be able to learn the creation of a process using fork() call

Theory

This section provides information about various operations on processes.

- fork(): to create a child process.
- wait(): to momentarily stop the parent process and execute the child pro-cess rst.
- exec(): to replace the current executing process with a new process.

Operation on processes is structured as follows:

Outline of Programs

1. Program to create a child process using fork

```
#include<stdio.h>

#include<sys/types.h>

#include<stdlib.h>

int main()

{
        int pid;
        pid=getpid();
        printf("current process pid is %d",pid);
        printf("forking a child process \n");
        pid=fork();
        if(pid==0)
        {
                printf("child process id: %d and its parent id: %d", getpid(),
        getppid());
        }
else
{
        printf("parent process id %d",getpid());
}
        return 0;
```

```
}
```

2. Program to create an orphan process

```
#include<stdio.h>
#include<sys/types.h>
#include<stdlib.h>
int main()
{
        int pid;
        pid=getpid();
        printf(“current process pid is %d", pid);
        printf(“forking a child process \n");
        pid=fork();
        if(pid==0)
{
        printf(“child process is sleeping");
        sleep(10);
        printf(“orphan child parent id: %d", getppid());
}
else
{
        printf(“parent process completed");
}
        return 0;
}
```

3. Program to create a zombie process

```
#include<stdio.h>
#include<sys/types.h>
#include<stdlib.h>
int main()
{
        pid_t a;
        a=fork();
        if(a>0)
        {
                sleep(20);
                printf(“PID of Parent %d", getpid());
        }
else
{
        printf(”PID of CHILD %d", getpid());
        exit(0);
}
}
```

Exercise

1. Write a program using system calls for operation on process to simulate that n fork calls create $(2^n - 1)$ child processes.

1. Write a program using systems for operations on processes to create a hierarchy of processes $P_1 \rightarrow P_2 \rightarrow P_3$. Also print the id and parent id for each process.

2. Write a program using system calls for operation on processes to create a hierarchy of processes as given in figure 1. Also, simulate process p4 as orphan and P5 as zombie.

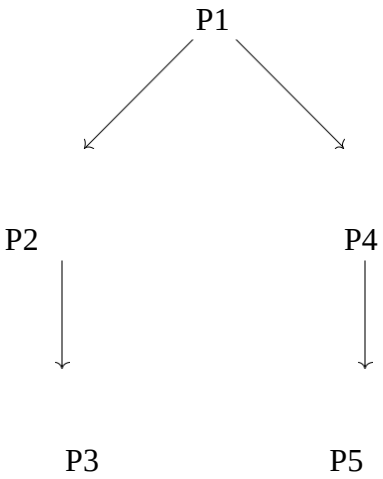


Figure 1: Hierarchy of Processes

3. Write a program using system calls for operation on processes to create a hierarchy of processes 2. Also, simulate P4 as an orphan process and P7 as a zombie.

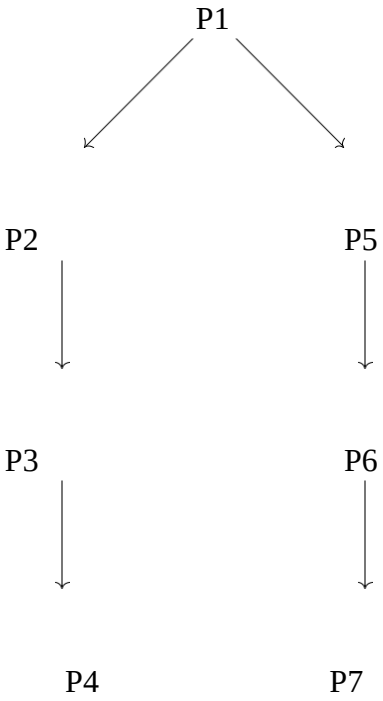


Figure 2: Hierarchy of Processes

Viva Questions

1. What is the difference between orphan and zombie processes?
2. How many child processes will be created if three consecutive fork statements are used in a main function.
3. In case of p = fork(), which process will be executed is p > 0.
4. In case of p = fork(), which process will be executed is p < 0.
5. Which system call causes the parent process to stop until the child completes the execution.
6. What is the function of an execl system call?

Learning Outcomes (What I have Learnt)

S.No.	Parameter	Marks Obtained	Max. Marks
1	Understanding of the student about the Procedure		15
2	Student is able to answer sample viva questions		15
3	Completion of Experiment		20
	Signature of Faculty with date		

Lab 6- Thread Creation using Pthread

Aim

Introduce the operations on threads, which include initialization, creation, join and exit functions of thread using pthread library.

Learning Objective

Students will be able to learn the concept of threads.

Theory

This section provides syntax of thread functions. More about the system calls can be read using manual page/ help in Linux shell.

- pthread_init(): to initialize a thread.
- pthread_create(): to create a thread.
- pthread_exit(): to exit a thread function with return value.
- pthread_join(): to join a thread in the main function and retrieve the value returned by thread function.

Syntax for pthread_create()

int a = pthread_create(pthread t *thread, pthread attr t *attr ,void *(*start routine), void(*));

Outline of Programs

1. Program to create a thread with NULL atributes

```
#include<stdio.h>
#include<pthread.h>
char *a;
void *func()
{
        printf(“In thread function \n");
        pthread_exit(“Exit thread function \n”);
}
int main()
{
        pthread_t thread1;
        void * a;
        printf(“In main thread \n");
        pthread create(&thread1, NULL, func, NULL);
        pthread join(thread1, &a);
        printf(“%s \n", a);
}
```

2. Program to pass message from main function to threads

```
#include<pthread.h>
#include<stdlib.h>
void *myfunc(void  *myvar);
int main (int argc, char *argv[])
{
        pthread t thread1, thread2;
        char *msg1= "first thread";
        char *msg2= "second thread";
        int ret1, ret2;
        ret1 = pthread_create(&thread1, NULL, myfunc,(void *) msg1);
        ret2 = pthread_create(&thread2, NULL, myfunc,(void *) msg2);
        printf("Main function after pthread_create \n");
        pthread_join(thread1, NULL);
        pthread_join(thread2, NULL);
        printf("first thread ret1= %d \n", ret1);
        printf("first thread ret1= %d \n", ret1);
        return 0;
}

void *myfunc (void *myvar)
{
        char *msg;
        msg= (char*) myvar;
        int i;
        for (i=0; i<10; i++)
        {
                printf("%s %d \n", msg, i);
                sleep(2);
        }
        return NULL;
}
```

3. Program to return a value from thread function to main function

```
#include <stdio.h>
#include <pthread.h>
int *a;
struct arg_struct
{
        int arg1;
        int arg2;
        int arg3;
};
void *print_the_arguments(void *arg)
{
        struct arg_struct *ar = (struct arg_struct *)arg;
        scanf("%d", &ar -> arg3);
```

```
        scanf("%d",&ar->arg2);
        int c=ar->arg2 + ar->arg3;
        pthread_exit(c);
}
int main()
{
        pthread_t some_thread;
        struct arg_struct args;
        args.arg1 = 5;
        args.arg2 = 7;
        void *a;
        pthread_create(&some thread, NULL, &print_the_arguments,
&args);
        pthread_join(some_thread, &a); /* Wait until thread is nished */
        printf("%d \n", a);
}
```

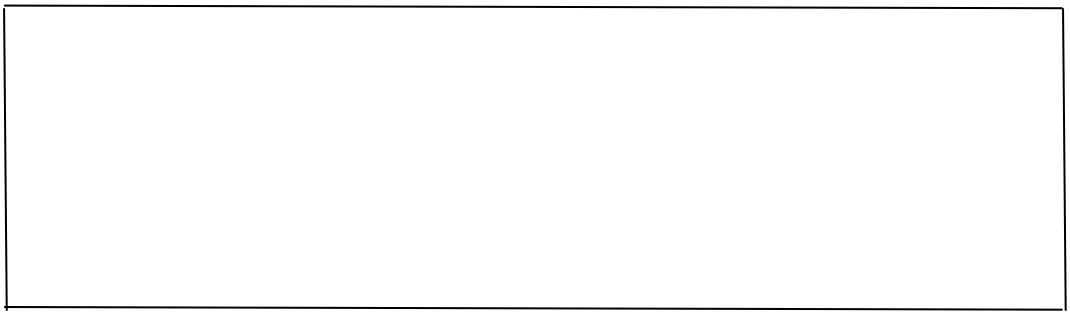
Exercise

1. Write a program using pthread to concatenate the strings, where multiple strings are passed to thread function.
2. Write a program using pthread to find the length of string, where strings are passed to thread function.
3. Write a program that performs statistical operations of calculating the average, maximum and minimum for a set of numbers. Create three threads where each performs their respective operations.
4. Write a multithreaded program where an array of integers is passed globally and is divided into two smaller lists and given as input to two threads. The thread will sort their half of the list and will pass the sorted list to a third thread which merges and sorts the list. The final sorted list is printed by the parent thread.

Viva Questions

1. Provide two examples in which multithreaded process provide an advantage over single threaded solution.
2. What resources are used when a thread is created.
3. What is the syntax for creating a thread.
4. What is the use of pthread_join().
5. What is the use of pthread_exit().
6. Which pthread function passes the value from thread function to main function.

Learning Outcomes(What I have Learnt)



S.No.	Parameter	Marks Obtained	Max. Marks
1	Understanding of the student about the Procedure		15
2	Student is able to answer sample viva questions		15
3	Completion of Experiment		20
	Signature of Faculty with date		

Lab 7- Process Synchronization

Aim

The objective of this laboratory is to introduce the concept of synchronization.

Learning Objective

Student will be able to learn the concept of mutex and semaphores using pthread library. some classical problems of process synchronization will be simulated and solved.

Theory

This section provides syntax of mutex and semaphore functions. More about the system calls can be read using manual page/ help in Linux shell.

- pthread_mutex_t: to initialize a mutex variable.
- pthread_mutex_lock(): to apply lock using mutex variable.
- pthread_mutex_unlock(): to release the lock using mutex variable.
- sem_t: to declare a semaphore variable.
- sem_init(): to initialize semaphore variable which takes three parameters:
    - Semaphore variable
    - Number of processes sharing semaphore variable
    - Maximum value of semaphore variable

- sem_wait(): to decrement the value of semaphore variable by 1. It also blocks other processes.
- sem_post(): to increment the value of semaphore variable by 1. It also sends a signal to wakeup the blocked process.

Oultine of the Programs

1. Program to simulate a race condition

```
#include<stdio.h> #include<pthread.h> int shared=5; void
*func1()
{
        int local;
        /*critical section */
        local=shared;
        local=local+1;
        sleep(5); /* causes a context switch */
        shared=local;
        /*critical section */
        printf("shared in func1: %d \n",shared);
        pthread_exit(NULL);
}
```

```
void *func2()
{
        int local;
        /*critical section */
        local=shared;
        local=local-1;
        shared=local;
        /*critical section */
        printf("shared in func2: %d \n",shared);
        pthread_exit(NULL);
}
int main()
{
        pthread_t t1, t2;
        pthread_create(&t1, NULL, func1, NULL);
        pthread_create(&t2, NULL, func2, NULL);
        pthread_join(t1, NULL);
        pthread_join(t2, NULL);
}
```

2. Program to remove race condition using mutex

```
#include<stdio.h>
#include<pthread.h>
int shared=5;
pthread_mutex_t l; /*mutex variable l*/
void *func1()
{                   `       `
        int local;
        /*critical section */
        pthread_mutex_lock(&l); /*applying lock using l(initally false)*/
        local=shared;
        local=local+1;
        sleep(5); /* causes a context switch */
        shared=local;
        pthread_mutex_unlock(&l) /*releasing lock using l*/
        /*critical section */
        printf("shared in func1: %d \n", shared);
        pthread_exit(NULL);
}
void *func2()
{
        int local;
        /*critical section */
        pthread_mutex_lock(&l); /*If acquired by func1 l will return true*/
        local=shared;
        local=local-1;
        shared=local;
        pthread_mutex_unlock(&l); /*releasing lock using l*/
        /*critical section */
```

```
        printf("shared in func2: %d \n", shared);
        pthread_exit(NULL);
}
int main()
{
        pthread_t t1,t2;
        pthread_create(&t1,NULL,func1,NULL);
        pthread_create(&t2,NULL,func2,NULL);
        pthread_join(t1,NULL);
        pthread_join(t2,NULL);
}
```

3. Program to remove race condition using semaphores

```
#include<stdio.h>
#include<pthread.h>
int shared=5;
sem_t s; /*mutex variable l*/
void *func1()
{
        int local;
        /*critical section */
        sem_wait(&s) /*applying lock using l(initally false)*/
        local=shared;
        local=local+1;
        sleep(5); /* causes a context switch */
        shared=local;
        sem_post(&s) /*releasing lock using l*/
        /*critical section */
        printf("shared in func1: %d \n",shared);
        pthread_exit(NULL);
}
void *func2()
{
        int local;
        /*critical section */
        sem_wait(&s) /*If acquired by func1 l will return true*/
        local=shared;
        local=local-1;
        shared=local;
        sem_post(&s); /*releasing lock using l*/
        /*critical section */
        printf("shared in func2: %d \n", shared);
        pthread_exit(NULL);
}
int main()
{
        pthread_t t1, t2;
        sem_init(&s,0,1); /*initializing semaphore*/
        pthread_create(&t1,NULL,func1,NULL);
        pthread_create(&t2,NULL,func2,NULL);
```

```
        pthread_join(t1,NULL);
        pthread_join(t2,NULL);
}
```

Exercise

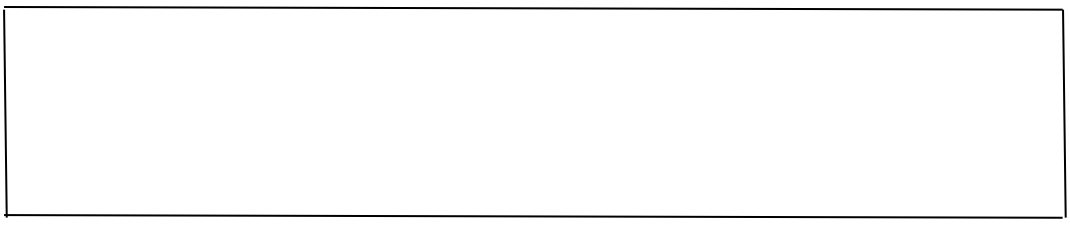
1. Implement the producer consumer problem using pthreads and mutex operations.
Test Cases:
(a) A producer only produces if buffer is empty and consumer only consumes if some content is in the buffer.
(b) A producer produces(writes) an item in the buffer and consumer consumes(deletes) the last produces item in the buffer.
(c) A producer produces(writes) on the last consumed(deleted) index of the buffer.

2. Implement the reader writer problem using semaphore and mutex operations to synchronize n readers active in reader section at a same time, and one writer active at a time.
Test Cases:
(a) If n readers are active no writer is allowed to write.
(b) If one writer is writing no other writer should be allowed to read or write on the shared variable.

Viva Questions

1. What is the difference between mutex and semaphore.
2. What should be the initial value of mutex and semaphore such that one process is allowed to enter in the critical section.
3. What is the return value when pthread create is successfully executed.
4. What is the sequence in which pthread_mutex operations should be used.
5. What should be the initial value for binary semaphore and counting semaphore?
6. If the initial value of semaphore is set to 3 in sem_init, how many processes can execute in the critical section at a given time unit.

Learning Outcomes (What I have Learnt)



S.No.	Parameter	Marks Obtained	Max. Marks
1	Understanding of the student about the Procedure		15
2	Student is able to answer sample viva questions		15
3	Completion of Experiment		20
	Signature of Faculty with date		

Lab 8- Inter Process Communication

Aim

The aim of this laboratory is to introduce the Interprocess communication (IPC) mechanism of operating system to allow the processes to communicate with each other.

Learning Objective

Students will be able to learn various IPC techniques to exchange the information between different processes in a system.

Description

Inter Process Communication (IPC) is a mechanism offered by the operating system two provide communication between two or more cooperating processes. This communication mechanism helps the processes to transfer or share data and coordinate activities. Inter process communication is supported by all UNIX systems. The IPC can be used between processes on a single computer system as well as on different systems. The examples of IPC methods include Pipes, FIFOs (named pipes), Shared memory, Message queues and Remote Procedure Call.

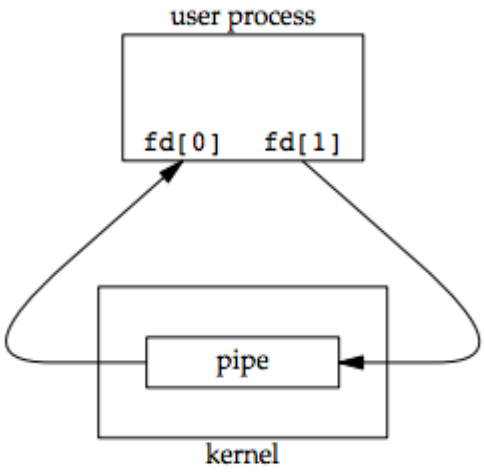
Pipes

Pipes are the oldest form of UNIX System IPC and are provided by all UNIX systems. Pipes only provides half duplex communication between processes that have a common ancestor.

A pipe is created using the pipe function.

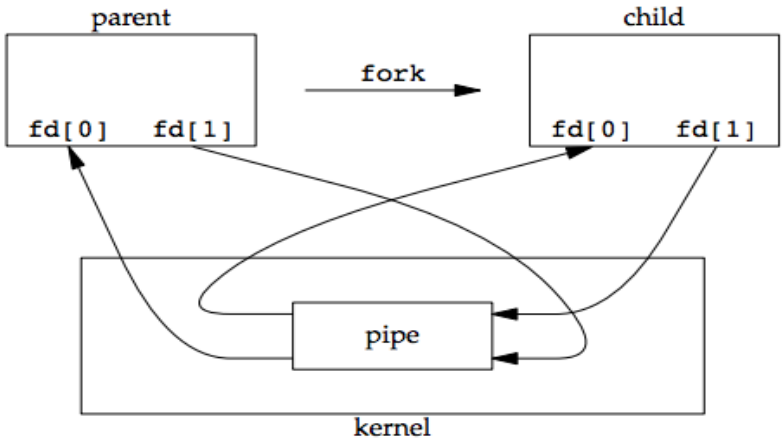
```c
#include <unistd.h>

int pipe(int fd[2]);

/* Returns: 0 if OK, -1 on error */
```

The pipe function returns two file descriptors through the fd argument. The fd[0] is a file descriptor that a process can use to read the data from the pipe, and fd[1] is a different file descriptor that a process in use to write data to the pipe.. The following figure shows the data flow between two ends of a half-duplex pipe in a process.

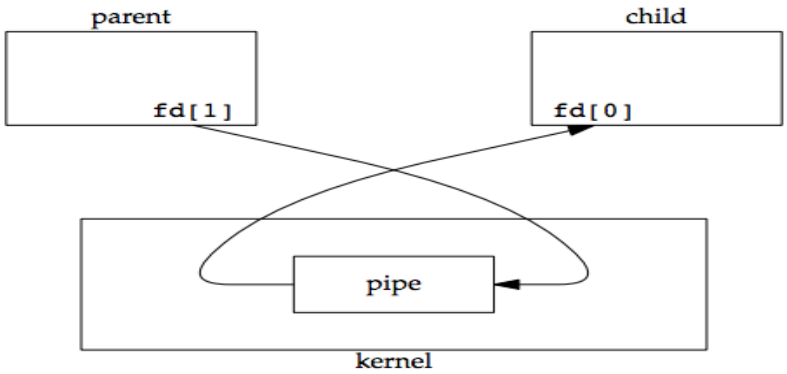


Ex: The pipe communication between the parent and the child processes.

The following figure shows the creation of a child process and IPC between parent and child process using pipe method.



In the pipe from the parent to the child, the parent closes the read end of the pipe (fd[0]), and the child closes the write end (fd[1]). The following figure demonstrates the resulting design of descriptors.



Sample program

- The following program illustrates the creation of a child process and data transfer using pipes.

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<string.h>
#include<stdlib.h>
int main()
{
	int fd[2],nb;
	pid_t cpid;
	char inf[]=" Welcome to LPU\n";
	char rbuff[50];
	pipe(fd);
	if((cpid=fork()) == -1)
	{
		printf(" Parent failed to create the child process");
		exit(1);
	}
	if(cpid == 0)
	{
		close(fd[0]);
		write(fd[1],inf,(strlen(inf)+1));
		printf("The information written in the pipe by child is: %s",inf);
		exit(0);
	}
	else
	{
		close(fd[1]);
		nb=read(fd[0],rbuff,sizeof(rbuff));
		printf("The information received by the Parent process from the pipe is:%s",rbuff);
	}
	return(0);
}
```

- The following program illustrates the creation of a child process and two way communication using pipes.

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


void main()
```

```
{
    int fd1[2],fd2[2],nb;
    pid_t cpid;
    char inf1[] = "Welcome to LPU";
    char inf2[] = "Thank you";
    char buff[100];
    pipe(fd1);
    pipe(fd2);
    if((cpid=fork())== -1)
    {
        printf("Parent failed to create the child process");
        exit(1);
    }
    if(cpid == 0)
    {
        close(fd1[0]);
        close(fd2[1]);
        write(fd1[1],inf1,strlen(inf1)+1);
        nb = read(fd2[0],buff,sizeof(buff));
        printf("\n The information received by the Child process is:%s\n",buff);
         exit(0);
    }
    else
    {
        close(fd1[1]);
        close(fd2[0]);
        write(fd2[1],inf2,strlen(inf2)+1);
        nb = read(fd1[0],buff,sizeof(buff));
        printf("\n The information received by the parent process is:%s\n",buff);
    }
  }
```

Shared Memory

Shared Memory is another important IPC mechanism where system memory is shared between two or more processes. Here, communication is done through this shared memory where modifications done by one process can be seen by another process. The operating system assigns a memory segment in the address space for several processes to read and write without involving the kernel during the data exchange. The basic steps involved in the shared memory communication are,

a. Requesting the operating system for a memory segment that can be shared between processes.
b. Associating the memory segment to the address space of the calling process.

Functions of IPC Using Shared Memory

shmget() is used to create the shared memory segment. The syntax of the function is shown below.

```
#include <sys/ipc.h>
#include <sys/shm.h>
int shmget (key_t key, size_t size, int shmflg);
```

- The first parameter indicates the unique number recognizing the shared segment.
- The second parameter indicates the size of the shared segment (e.g., 1024 or 2048 byt es).
- The third parameter indicates the permissions on the shared segment.

shmat() is used to connect the shared segment with the process's address space. The syntax of the function is shown below.

```
#include <sys/types.h>
#include <sys/shm.h>
void *shmat(int shmid, const void *shmaddr, int shmflg);
```

- The first parameter is the identifier returned by the shmget() on success.
- The second parameter specifies the address to which the shared segment can be linked to a process.
- The third parameter value is '0' if the second parameter value is NULL.

shmdt(): This function is used to detach the shared segment from a process

shmctl(): This function allows the a process to control the shared memory segment.

Sample Program for IPC using Shared Memory

➢ The following program illustrates the creation of a shared memory segment and adding data into it.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/shm.h>
#include<string.h>
int main()
{
      void *shm;
```

```
char buf[200];
int shmid;
shmid=shmget((key_t)123, 2048, 0666|IPC_CREAT);          //creation of shared memory
segment
printf("The Key value of shared memory is %d\n",shmid);
shm=shmat(shmid,NULL,0);       // Attaching the process to shared memory segment
printf("Process attached to the address of %p\n",shm);
printf("Write the data to shared memory\n");
read(0,buff,100);
strcpy(shm,buf);
printf("The stored data in shared memory is : %s\n",(char *)shm);
}
```

- The following program illustrates the retrieving of data from the shared memory segment.

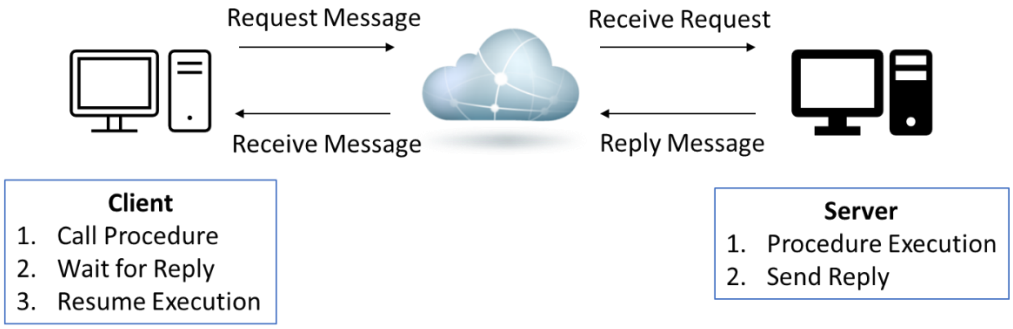
```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/shm.h>
#include<string.h>
int main()
{
    void *shm;
    char buf[200];
    int shmid;
    shmid=shmget((key_t)123, 2048, 0666);
    printf("The Key value of shared memory is %d\n",shmid);
    shm=shmat(shmid,NULL,0);
    printf("Process attached to the address of %p\n",shm);
    printf("The retrieved data from the shared memory is : %s\n",(char *)shm);
}
```

Remote Procedure Call (RPC)

RPC is an interprocess communication mechanism that is employed for client-server-based applications. The two processes may be within the same system, or in different systems with a network connection. The client starts the communication with a procedure or a function call and sends a request message to the server by setting the parameters in the message. The server starts the execution of the procedure after receiving the request message and sends a response

back to the client. The client is blocked while the server is executing the call and only continued execution after the server is completed.

The following diagram demonstrates the sequence of events in a remote procedure call-



Exercise

1. Implement IPC using named pipes concept.
2. Implement IPC using message passing technique
3. Implement IPC using message queue technique
4. Implement a program to perform IPC using RPC

Viva Questions

1. What is remote procedure call?
2. What is function of named pipe?
3. Can named pipes be bidirectional?
4. Differentiate shared memory and message passing
5. Which is faster shared memory or message queue?

Learning Outcomes (What I have Learnt)

S.No.	Parameter	Marks Obtained	Max. Marks
1	Understanding of the student about the Procedure		15
2	Student is able to answer sample viva questions		15
3	Completion of Experiment		20
	Signature of Faculty with date		