LABORATORY MANUAL

Course Code: CSE325
Course Title: Operating System Laboratory

Prepared By

Pushpendra Kumar Pateriya
HoD, System Programming Domain
School of Computer Science & Engineering
&
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

List of Experiments:

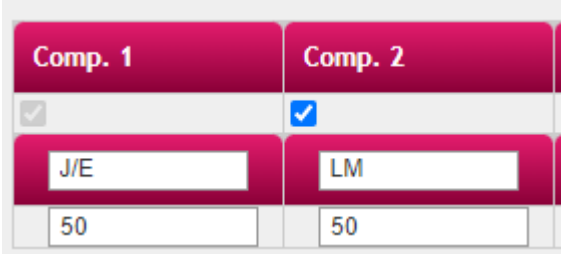
Experiment No.	Practical Description	Page No.	Student performed practical on Date (Need to be filled by student)	Techer checked the practical on Date (Need to be filled by teacher)
1	Introduction to Linux Commands	4		
2	Shell Programming	7		
3	File Manipulation using System Calls	12		
4	Directory Manipulation using System Calls	16		
5	Processes Management using System Calls	19		
6	Creation of Multithreaded Processes using Pthread Library	23		
7	Process Synchronization using Semaphores/Mutex	27		
8	Inter Process Communication using Pipes/Shared Memory	33		

General Guidelines for the students

- Your Lab manual exercises will be included in your Continuous Assessment (CA) marks.
- Please sit in your assigned seats according to your roll number and use the Lab PCs accordingly. Stick to the same system for all lab exercises throughout the semester.
- If you have any issues with the system you're assigned, please inform the lab instructor or lab technician.
- You can only use the computer systems in the lab when a teacher or lab technician is present.
- Don't change any system settings without getting approval from the teacher first.
- Eating inside the lab is not allowed.
- Make sure to practice on your own; practicing is important for understanding the concepts of this course.
- If you have any feedback or suggestions for each experiment, you can share them using this link: https://forms.gle/2PPp3ThtdXY2M1FWA.

General Guidelines for the teachers:

1. The teacher will check students' lab manual regularly, sign the manual and ensure the progressive learning of the students.
2. The teacher should create practical components by following the path on UMS: **UMS Navigation >> Learning Management System (LMS) >> Practical Components**.
3. Create two components (**1. J/E: Job Execution, 2. LM: Lab Manual Completion**) of 50-50 marks each.



4. There will be total 4 continuous assessment practical (CAP) conducted during the semester (100 marks each).
5. Best 3 out of 4 will be considered in grade calculation by the end of the semester.

Experient 1 – Introduction to Linux Commands

Aim

To study important Shell commands to perform various operations. Students will also do a comparative study of Linux and Windows commands.

Learning Objective

Students will learn and practice different Shell commands to work on the Linux shell prompt. They will gain the ability to perform a wide range of tasks using these commands.

Theory

Table 1: Comparative Study of Linux and Windows Commands

	Commands	
Description	**Linux**	**Windows**
Listing files	Ls	Dir
Long and Time Formatted Listing	ls -al	Dir
Change present working directory to XYZ	cd XYZ	cd XYZ
Change to home directory	Cd	Windows file system does not have concept of home directory.
Display the path of present working directory	pwd	Windows prompt displays the complete path of the working directory.
Create a directory named as XYZ	mkdir XYZ	md XYZ / mkdir XYZ
Create a file named as myFile.txt	touch myFile.txt	Copy con myFile.txt / notepad myFile.txt
Update the timestamp of a file. Consider the file name is XYZ.	touch XYZ	-
deletion of files, consider file name as XYZ	rm XYZ	del XYZ
Concatenation of files contents and printing the concatenated content on the monitor. Use file name f1 f2 and f3.	cat f1 f2 f3	-
Copy files from one directory to Another	cp fileName DestinationPath	copy fileName DestinationPath
Move files from one location to another	mv fileName DestinationPath	move fileName DestinationPath

Video Reference:

https://youtube.com/playlist?list=PLWjmN065fOfGdAZrlP6316HVHh8jlvefD

Lab Exercise(s)

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

Viva Questions

1. How Windows is different from Linux.
2. Differentiate 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	Max Marks	Obtained Marks
1	Conceptual/procedural understanding of the student	15	
2	Student is able to answer sample viva questions	15	
3	Completion of Lab exercises Signature of Faculty with date	20	

Experient 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
```

Video Reference:

https://youtube.com/playlist?list=PLWjmN065fOfGdAZrlP6316HVHh8jlvefD

Lab Exercise(s)

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	Max Marks	Obtained Marks
1	Conceptual/procedural understanding of the student	15	
2	Student is able to answer sample viva questions	15	
3	Completion of Lab exercises Signature of Faculty with date	20	

Experient 3- File Manipulation using 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);

   return 0;
```

```
}
```

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("Number of bytes written: %d \n", n);

   m = read(0, buffer, 12);
   printf("Number of bytes read: %d \n", m);

   return 0;
}
```

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);

   return 0;
}
```

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;
}
```

Video Reference:

https://youtube.com/playlist?list=PLWjmN065fOfGdAZrlP6316HVHh8jlvefD

Lab Exercise(s)

1. Create a program using system calls to copy either the first half or the second half of a file into a new file.

2. Develop a program using system calls to read input from the console until the user inputs '$', and then save the input into a file.

3. Design a program using system calls to read the contents of a file without using a char array and display the contents directly on the console. Please refrain from using built-in functions like sizeof() and strlen().

Viva Question

1. Which system call is used to change the position of the file pointer?
2. What does the read system call return?
3. What value does the open system call return?
4. What is passed as the first argument of the read/write system call?
5. If a file is to be appended, in which mode should it be opened using the open system call?
6. How can we use the lseek system call to handle file appending?
7. What value does the lseek system call return?
8. Which system call can be used to determine the size of a file?
9. How can we find the size of a file using the lseek system call?

Learning Outcomes (What I have Learnt)

S. No.	Parameter	Max Marks	Obtained Marks
1	Conceptual/procedural understanding of the student	15	
2	Student is able to answer sample viva questions	15	
3	Completion of Lab exercises Signature of Faculty with date	20	

Experient 4- Directory Manipulation using 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 file 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;
}
```

Video Reference:

https://youtube.com/playlist?list=PLWjmN065fOfGdAZrlP6316HVHh8jlvefD

Lab Exercise(s)

1. Create a program using directory system calls to make a new directory on the desktop, then create a file inside that directory, and finally, list the contents of the directory.

2. Develop a program using directory and file manipulation system calls to copy the contents of one directory into a newly created directory.

Learning Outcomes (What I have Learnt)

S. No.	Parameter	Max Marks	Obtained Marks
1	Conceptual/procedural understanding of the student	15	
2	Student is able to answer sample viva questions	15	
3	Completion of Lab exercises Signature of Faculty with date	20	

Experient 5 – Processes Management using System Calls

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 process.
- 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);
}
}
```

Video Reference:

https://youtube.com/playlist?list=PLWjmN065fOfGdAZrlP6316HVHh8jlvefD

Lab Exercise(s)

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.

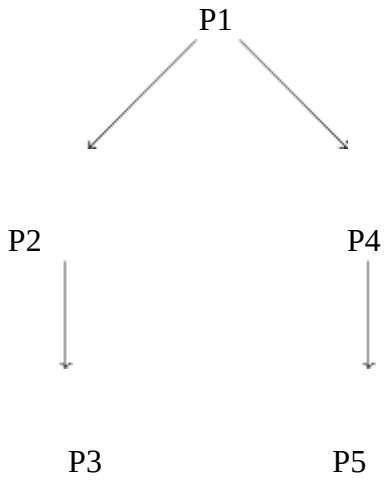
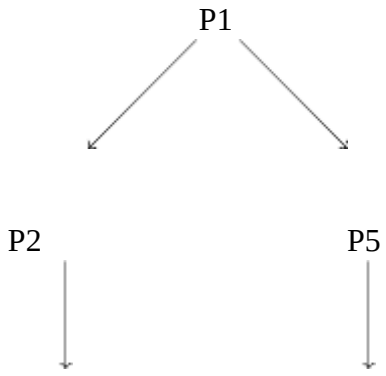


Figure 1: Hierarchy of Processes

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.



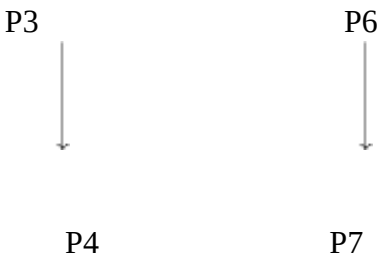


Figure 2: Hierarchy of Processes

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	Max Marks	Obtained Marks
1	Conceptual/procedural understanding of the student	15	
2	Student is able to answer sample viva questions	15	
3	Completion of Lab exercises Signature of Faculty with date	20	

Experient 6- Creation of Multithreaded Processes using Pthread Library

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 *result;

   printf("In main thread \n");

   pthread_create(&thread1, NULL, func, NULL);
```

```
    pthread_join(thread1, &result);

    printf("%s \n", (char *)result);

    return 0;
}
```

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((void *)c);
}

int main() {
    pthread_t some_thread;
    struct arg_struct args;
    args.arg1 = 5;
    args.arg2 = 7;

    void *result;
    pthread_create(&some_thread, NULL, &print_the_arguments, &args);
    pthread_join(some_thread, &result); /* Wait until the thread is finished */
    printf("%d \n", (int)result);

    return 0;
}
```

Video Reference:

https://youtube.com/playlist?list=PLWjmN065fOfGdAZrlP6316HVHh8jlvefD

Lab Exercise(s)

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	Max Marks	Obtained Marks
1	Conceptual/procedural understanding of the student	15	
2	Student is able to answer sample viva questions	15	
3	Completion of Lab exercises Signature of Faculty with date	20	

Experient 7- Process Synchronization using Semaphores/Mutex

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);

    return 0;
}
```

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 (initially 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_mutex_init(&l, NULL); /* Initialize the mutex variable l */
    pthread_create(&t1, NULL, func1, NULL);
    pthread_create(&t2, NULL, func2, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    pthread_mutex_destroy(&l); /* Destroy the mutex variable l */

    return 0;
}
```

3. Program to remove race condition using semaphores

```
#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>


int shared = 5;

sem_t s; /* Semaphore variable s */


void *func1() {

    int local;

    /* Critical section */

    sem_wait(&s); /* Applying lock using s (initially 1) */

    local = shared;

    local = local + 1;

    sleep(5); /* Causes a context switch */

    shared = local;
```

```
    sem_post(&s); /* Releasing lock using s */
    /* 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, s will return true */
    local = shared;
    local = local - 1;
    shared = local;
    sem_post(&s); /* Releasing lock using s */
    /* 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 with value 1 */
    pthread_create(&t1, NULL, func1, NULL);
    pthread_create(&t2, NULL, func2, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    sem_destroy(&s); /* Destroying the semaphore */
```

```
    return 0;
}
```

Video Reference:

https://youtube.com/playlist?list=PLWjmN065fOfGdAZrlP6316HVHh8jlvefD

Lab Exercise(s)

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	Max Marks	Obtained Marks
1	Conceptual/procedural understanding of the student	15	
2	Student is able to answer sample viva questions	15	
3	Completion of Lab exercises Signature of Faculty with date	20	

Experient 8- Inter Process Communication using Pipes/Shared Memory
Aim

The aim of this laboratory is to introduce the Inter-process 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.

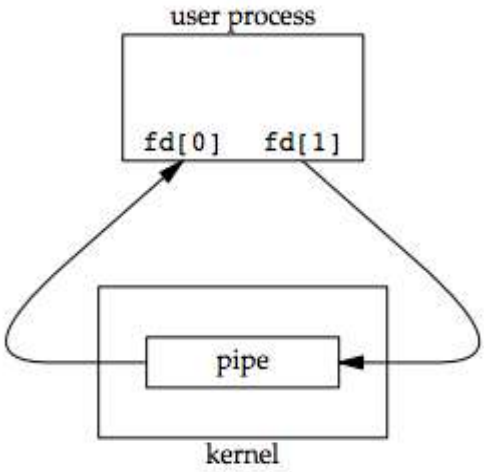
Pipes

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.

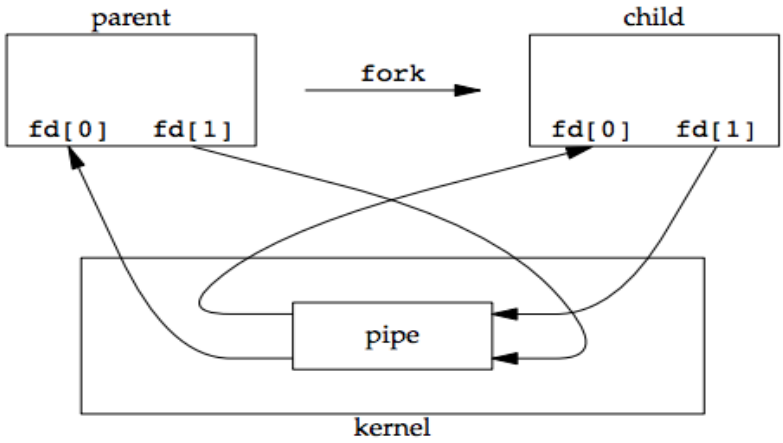
```
#include <unistd.h>

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.

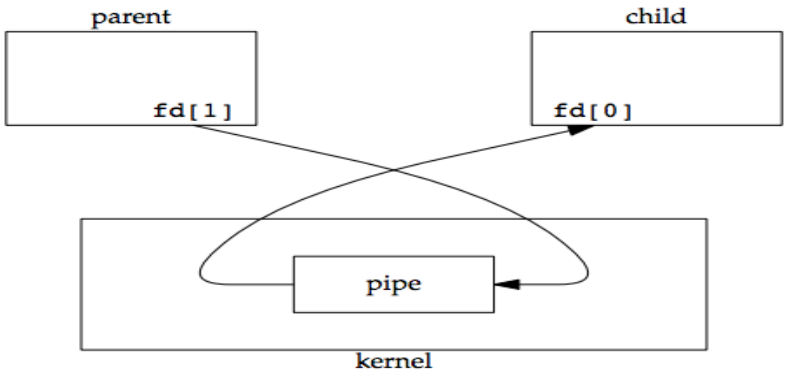


Ex: The pipe communication between the parent and the child processes.

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

1. 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;
}
```

2. 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>

int 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);
    }

    return 0;
}
```

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

3. 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 the shared memory segment
    printf("Process attached to the address of %p\n", shm);

    printf("Write the data to shared memory\n");
    read(0, buf, 100);
    strcpy(shm, buf);
    printf("The stored data in shared memory is: %s\n", (char *)shm);
    return 0;
}
```

Video Reference:

https://youtube.com/playlist?list=PLWjmN065fOfGdAZrlP6316HVHh8jlvefD

Lab Exercise(s)

1. Implement IPC using named pipes concept.

2. Implement IPC using message passing technique
3. Implement IPC using message queue technique

Viva Questions

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

Learning Outcomes (What I have Learnt)

S. No.	Parameter	Max Marks	Obtained Marks
1	Conceptual/procedural understanding of the student	15	
2	Student is able to answer sample viva questions	15	
3	Completion of Lab exercises Signature of Faculty with date	20	

Practice exercises organized by experiment

Introduction to Linux Commands

Exercise 1

1. List all the files and directories in your home directory.
2. Create a new directory called "test" in your home directory.
3. Change into the "test" directory you just created.
4. Create a new file called "example.txt" in the "test" directory.
5. Open the "example.txt" file and write "Hello, World!" in it.
6. Save and exit the file.
7. List all the files in the "test" directory.
8. Rename the "example.txt" file to "sample.txt".
9. List all the files in the "test" directory again to verify the file has been renamed.
10. Remove the "test" directory and all its contents.
11. List all the files and directories in the root directory.
12. Change into the "/etc" directory.
13. List all the files and directories in the "/etc" directory.

Exercise 2

1. Create a new directory called "lab_exercises" in your home directory.
2. Inside the "lab_exercises" directory, create a new file called "file1.txt" and write some text in it.
3. Create a copy of "file1.txt" and name it "file2.txt" using the cp command.
4. Verify that "file2.txt" is an exact copy of "file1.txt" by opening both files and comparing their contents.
5. Create a new directory called "backup" inside the "lab_exercises" directory.
6. Move "file1.txt" and "file2.txt" to the "backup" directory using the mv command.
7. Verify that both files have been moved to the "backup" directory by listing its contents.
8. Create a new file called "file3.txt" in the "lab_exercises" directory and write some text in it.
9. Create a new directory called "archive" inside the "lab_exercises" directory.
10. Move "file3.txt" to the "archive" directory and rename it to "file3_backup.txt" using the mv command.
11. Verify that "file3_backup.txt" has been moved to the "archive" directory and that its contents are the same as "file3.txt".
12. Create a new directory called "temp" inside the "lab_exercises" directory.
13. Create a new file called "file4.txt" in the "temp" directory and write some text in it.
14. Move "file4.txt" to the "lab_exercises" directory using the mv command.
15. Verify that "file4.txt" has been moved to the "lab_exercises" directory and that its contents are the same as before.

Shell Programming

Exercise 1

A. Write a script that displays "Hello, World!" when executed.

B. Modify the script to accept a command line argument and display "Hello, <argument>!" instead of "Hello, World!".

Exercise 2

Write a script that accepts two command line arguments and displays their sum.

Exercise 3

A. Write a script that accepts a directory name as a command line argument and displays the number of files in that directory.

B. Modify the script to display the number of files in the directory and all its subdirectories.

Exercise 4

A. Write a script that accepts a filename as a command line argument and displays the number of lines, words, and characters in that file.

B. Modify the script to accept multiple file names as command line arguments and display the number of lines, words, and characters in each file.

Exercise 5

A. Declare a variable called "name" and assign your name to it. Display the value of the variable using the echo command.

B. Declare a variable called "age" and assign your age to it. Display the value of the variable using the echo command.

C. Declare a variable called "color" and assign your favorite color to it. Display the value of the variable using the echo command.

Exercise 6

Declare a variable called "num1" and assign the value 10 to it. Declare a second variable called "num2" and assign the value 5 to it. Add the values of the two variables and display the result using the echo command.

Exercise 7

A. Declare a variable called "filename" and assign the value "sample.txt" to it. Use the variable to create a new file with that name using the touch command.

B. Declare a variable called "directory" and assign the value "myfolder" to it. Use the variable to create a new directory with that name using the mkdir command.

Exercise 8

Declare a variable called "files" and assign a list of filenames to it. Use a loop to display the contents of each file in the list using the cat command.

Exercise 9

A. Write a command that displays the contents of a file called "file1.txt" on the screen.

B. Use input redirection to create a new file called "file2.txt" with the contents of "file1.txt".

C. Write a command that appends the contents of "file1.txt" to the end of "file2.txt".

Exercise 10

A. Write a for loop that prints the numbers from 1 to 10 on the screen.

B. Modify the loop to print only the even numbers from 1 to 10.

Exercise 11

A. Write a loop that displays the names of all files in the current directory.

B. Modify the loop to display only the names of files with the extension ".txt".

Exercise 12

A. Write a case/esac statement that displays a message on the screen based on the value of a variable called "day". If the value is "Monday", the message should be "It's the start of the week". If the value is "Friday", the message should be "Thank goodness it's Friday!". For any other value, the message should be "Just another day".

B. Modify the case/esac statement to use a read command to read the value of "day" from the user.

Exercise 13

Write a case/esac statement that calculates the area of a geometric shape based on the user's input. If the input is "square", the statement should ask the user for the length of the side and display the area. If the input is "rectangle", the statement should ask the user for the length and width and display the area. If the input is "circle", the statement should ask the user

Exercise 14

A. Write an if statement that checks if a variable called "x" is greater than 10. If it is, display the message "x is greater than 10".

B. Modify the if statement to check if "x" is equal to 10 as well. If it is, display the message "x is equal to 10".

File Manipulation using System Calls

Exercise 1

Write a program in C that creates a file called "output.txt", writes the text "Hello, World!" to it, and then closes the file.

Exercise 2

Write a program in C that reads the contents of a file called "input.txt" and writes them to a new file called "output.txt". You should use system calls like open(), read(), and write().

Exercise 3

Write a program in C that reads a file called "input.txt" and counts the number of lines in the file. You should use system calls like open(), read(), and write().

Exercise 4

A. Write a C program that creates a file called "numbers.txt" and writes 100 integers to it, one integer per line.

B. Using the lseek system call, move the file pointer to the beginning of the file.

C. Read the first 10 integers from the file and print them to the console.

Exercise 5

Write a C program that prints the last 10 characters of a file named as "input.txt" on the screen. Use open, read, write and lseek system calls.

Exercise 6

Write a C program that prints half content of a file named as "input.txt" on the screen. Use open, read, write and lseek system calls. If there are 100 characters written in the file, your program should display the first 50 characters on the screen.

Directory Manipulation using System Calls

Exercise 1

A. Write a C program that opens a directory called "my_directory" and reads all the files and directories inside it.

B. For each file and directory, print its name and whether it is a file or a directory.

C. Count the number of files and directories inside the "my_directory" directory.

D. Close the directory.

Process Management using System Calls

Exercise 1
Write a C program that uses the fork system call to create a child process. In the child process, print the process ID (PID) and the parent process ID (PPID). In the parent process, print the PID and the child's PID.

Exercise 2
Write a C program that uses the fork system call to create a child process. In the child process, print a message indicating that it is the child process. In the parent process, print a message indicating that it is the parent process.

Exercise 3
Write a C program that uses the fork system call to create a child process. In the child process, create a file called "child.txt" and write the message "This is the child process" to it. In the parent process, create a file called "parent.txt" and write the message "This is the parent process" to it.

Exercise 4
Write a C program that uses the fork system call to create a child process. In the child process, print the sum of the first 100 positive integers. In the parent process, print the sum of the first 1000 positive integers.

Creation of Multithreaded Processes using Pthread Library

Exercise 1
Write a C program that creates two threads using the Pthread library. Each thread should print its thread ID to the console.

Exercise 2
Write a C program that creates two threads using the Pthread library. One thread should generate a random number, print it to the console and another should check for a prime number.

Exercise 3
Write a C program that creates two threads using the Pthread library. One thread should print even numbers from 2 to 100, and the other thread should print odd numbers from 1 to 99.

Process Synchronization using Semaphores/Mutex

Exercise 1
Write a C program to create two threads that increment a shared variable using a mutex to synchronize access to the variable.

Exercise 2
Write a C program to create two processes that increment a shared variable using semaphores to synchronize access to the variable.

Exercise 3

Write a C program to create two processes that implement a producer-consumer model using semaphores to synchronize access to the shared buffer.

Inter Process Communication using Pipes/Shared Memory/RPC

Exercises on pipe(), dup()/dup2()

Exercise 1
Write a C program that creates a file called "file1.txt" and writes some text to it. Then, use the dup system call to create a duplicate file descriptor for the file. Finally, use the duplicate file descriptor to write some more text to the file.

Exercise 2
Write a C program that creates two pipes using the pipe system call. Then, fork a child process. In the parent process, use the dup2 system call to redirect the standard input to one end of the pipe, and the standard output to the other end of the pipe. In the child process, read from the standard input and write to the standard output.

Exercise 3
Write a C program that takes a file name as a command-line argument. Use the open, dup, and dup2 system calls to open the file, create two duplicate file descriptors for it, and redirect the standard input and standard output to those file descriptors. Then, read from the standard input and write to the standard output.

Exercise 4
Write a C program that creates a file called "file1.txt" and writes some text to it. Then, use the dup2 system call to create a duplicate file descriptor for the file, and close the original file descriptor. Finally, use the duplicate file descriptor to read the text from the file and write it to the standard output.

Exercises on mkfifo(): Named pipe

Exercise 1
Write a C program that creates a named pipe (FIFO) using the mkfifo system call. Then, fork a child process. In the parent process, write some data to the pipe using the write system call. In the child process, read the data from the pipe using the read system call.

Exercise 2
Write a C program that creates a named pipe called "mypipe" using the mkfifo system call. Then, use the open system call to open the pipe for writing. Write some data to the pipe using the write system call. Finally, use the cat command to read the data from the pipe.

Exercise 3
Write a C program that creates a named pipe called "mypipe" using the mkfifo system call. Then, use the open system call to open the pipe for reading. Read some data from the pipe using the read system call. Finally, use the echo command to write the data to the standard output.

Exercise 4

Write a C program that creates a named pipe called "mypipe" using the mkfifo system call. Then, fork a child process. In the parent process, use the open system call to open the pipe for writing. In the child process, use the open system call to open the pipe for reading. Then, write some data to the pipe in the parent process and read the data from the pipe in the child process.

Exercise 5
Write a program in C that creates a child process using fork(). The parent process should read a message from the user and send it to the child process using a pipe. The child process should then read the message from the pipe and print it to the console.

Shared Memory

Exercise 1
Implement a simple message passing system using shared memory in C. Create a parent process that forks a child process and communicates with it using the shared memory.
Use the shmget, shmat, and shmdt functions to create a shared memory segment, attach to it, and detach from it. The parent process should write a message to the shared memory segment, and the child process should read and print the message.