COMPUTER LABORATORY MANUAL

Operating Systems
(CSC 211 L)
Version 2.0

Student Name:	
Class and Section:	
Roll Number:	
CGPA:	
Email ID:	

DEPARTMENT OF SOFTWARE ENGINEERING (DSE)
FOUNDATION UNIVERSITY RAWALPINDI CAMPUS (FURC)
www.fui.edu.pk
http://www.fui.edu.pk/FURC/

PREFACE

This lab manual has been prepared to facilitate the students of software engineering and computer science in studying operating system which is essential part of any computer science education. This field is undergoing rapid change, as computers are now prevalent in virtually every arena of day-to-day life from embedded devices in automobiles through the most sophisticated planning tools for governments and multinational firms. The lab sessions are designed to improve the abilities of the students by giving hands on experience. After completing the laboratory exercises, the students will be familiar with the practical issues of the different concepts explained in the course, as well as with the real equipment used nowadays in computer.

PREPARED BY

Lab manual is prepared by, Ms. Fauzia Khan and Asst. Prof. Umar Mahmud under the supervision of Head of Department Dr. Muhammad Shaheen.

GENERAL INSTRUCTIONS

a. Students are required to maintain the lab manual with them till the end of the semester.
b. All readings, answers to questions and illustrations must be solved on the place provided. If more space is required, then additional sheets may be attached. You may add screen print to the report by using the 'Print Screen' command on your keyboard to get a snapshot of the displayed output.
c. It is the responsibility of the student to have the manual graded before deadlines as given by the instructor.
d. Loss of manual will result in re submission of the complete manual.
e. Students are required to go through the experiment before coming to the lab session.
f. Students must bring the manual in each lab.
g. Keep the manual neat clean and presentable.
h. Plagiarism is strictly forbidden. No credit will be given if a lab session is plagiarised and no re-submission will be entertained.
i. Marks will be deducted for late submission.
j. You need to submit the report even if you have demonstrated the exercises to the lab instructor or shown them the lab report during the lab session.

VERSION HISTORY

Date	Update By	Details
February 2016	Umar Mahmud	Version 1.0. Initial draft prepared and experiments outlined.
February 2017	Fauzia Khan	Version 1.1. Improvements added in the manual
January 2020	Umar Mahmud	Version 2.0. New questions have been added. Labs restructured. Modules have been added. Formatting corrected. 14 experiments included.

MARKS

LAB #	Date Conducted	Lab Title	Max. Marks	Marks Obtained	Instructor Sign
1		INTRODUCTION TO LINUX	10		
2		SYSTEM MONITORING	10		
3		PROCESSES	10		
4		PROCESS SCHEDULING	10		
5		PRIORITY SCHEDULER	10		
6		THREAD MANAGEMENT	10		
7		THREADS IN C	10		
8		IPC USING PIPES	10		
9		PETERSON’S ALGORITHM	10		
10		SEMAPHORES	10		
11		MODULE PROGRAMMING	10		
12		MEMORY MANAGEMENT	10		
13		SOCKETS	10		
14		FILE SYSTEMS	10		
Grand Total					

LIST OF EXPERIMENTS

EXPERIMENT 1 – INTRODUCTION TO LINUX

OBJECTIVE

- Introduce the Linux environment and basic shell commands.
- Write, compile, and execute a “Hello World!” C program in Linux environment using gcc.

TIME REQUIRED	:	3 hrs
PROGRAMMING LANGUAGE	:	C
SOFTWARE REQUIRED	:	Ubuntu/Fedora, gcc/gc, Text Editor, Terminal
HARDWARE REQUIRED	:	Core i5 in Computer Labs

INTRODUCTION

Fedora/Ubuntu is installed on the lab computers. Fedora and Ubuntu are open-source Linux-based operating system alternative to Windows. We will learn the basic desktop environment and common shell commands. Please login using the credentials provided by the lab instructor. Change Desktop Back ground, look and feels etc.

LINUX SYSTEM

Linux System can be split into two parts:

- Shell
- Kernel

Formally, a Shell is interface between a user and a Linux operating system, i.e. user interacts with the Linux operating system through the shell.

Kernel is the core of Linux Operating System, which is operational if the computer system is running. The kernel is part of the Linux Operating system which consists of routines which interact with underlying hardware, and routines which include system calls handling, process management, scheduling, signals, the file system, and I/O to storage devices.

Unlike DOS, which permits you to organize your folders (directories) and files anyway you please, the Linux file system is organized into a standard directory structure.

/home	Users’ home directory
/etc	All system administrator commands, configuration files, and installation control files.
/bin	The core set of system commands and programs. Most systems cannot boot (initially start) without executing some of the commands in this directory.
/dev	The device files used to access system peripherals (for example, your terminal can be accessed from /dev/tty).
/lib	The standard set of programming libraries required by Linux programs.
/tmp	Temporary files created and used by many Linux programs.
/var	Log files, spool files etc.

/root	The root user’s home directory.
/usr/bin	Common commands and programs.
/usr/doc	Documentation
/usr/games	Games
/usr/include	Header files
/usr/info	Online documentation
/usr/man	Manual pages (help)
/usr/share	Shared information

EXERCISE 1.1 **[2]**

Open **Text Editor**. It can be opened by clicking ‘Show Applications’ on lower left corner and searching or browsing. Type in your name, date of birth, CGPA and your favourite quote and save on Linux drive. Show the contents as a screen shot here: -

What file extension did you use? Does Linux store file as **.txt** by default?

Where did you store your file? Type complete path here: -

Right click and view properties. Write all properties here: -

TASK 1.1

Open **Terminal**

In this section, we learn some basic Linux commands e.g., ls, cd, mkdir etc.

DIRECTORY COMMANDS

ls	**List the file in the directory, just like dir command in DOS.**
	Options
	-a Display all the files, and subdirectories, including hidden files.
	-l Display detailed information about each file, and directory.
	-r Display files in the reverse order.
mkdir directory-name	**Creates a new directory.**

Directory-name specifies the name of the new directory. If the name doesn't begin with a slash, the new directory is created as a subdirectory of the current working directory. If the name begins with a slash, the name defines the path from the root directory to the new directory.

$cd /

Try to use the following command first because this will bring you back to your home directory

```
$ cd shehreyar@shehreyar- virtualBox :/$ mkdir books
```

This command will create a new directory under the home directory.

What does **/ls** do? Screenshot and paste outcome here:

TASK 1.2:

Open **Text Editor**

Type the following source code

```
#include<stdio.h>
main()
{
   printf("Hello  World\n");
}
```

Save the file as HelloWorld.c in /Documents. Did the colours change after saving or before?

Open **Terminal** and navigate to your file.

Type **ls,** screen shot and paste outcome here: -

Type `gcc  -o  HelloWorld  Hello.c.`

Type **ls,** screenshot and paste here: -

What is the difference?

Execute by typing `./HelloWorld`

Screenshot and show outcome here:

EXERCISE 1.2 **[8]**

Rewrite code that shows your name, CGPA, date of birth and your favourite quote formatted adequately in C. Show code and outcome here: -

RESOURCES:

https://www3.ntu.edu.sg/home/ehchua/programming/cpp/gcc_make.html

https://en.wikipedia.org/wiki/Linux

https://www.youtube.com/watch?v=IVquJh3DXUA

https://www.youtube.com/watch?v=oLjN6jAg-sY

EXPERIMENT 2 – SYSTEM MONITORING

OBJECTIVE

- Learn the basic CLI and GUI tools to monitor system resources.
- Identify system calls used by programs written in C language.
- Learn to write simple batch/shell scripts.

TIME REQUIRED	:	3 hrs
PROGRAMMING LANGUAGE	:	C
SOFTWARE REQUIRED	:	Ubuntu/Fedora, gcc/gc, Text Editor, Terminal
HARDWARE REQUIRED	:	Core i5 in Computer Labs

SYSTEM MONITORING

A **system monitor** is a hardware or software component used to monitor system resources and performance in a computer system. Among the management issues regarding use of system monitoring tools are resource usage and privacy.

TASK 2.1

Navigate to **System Monitor**

How many cores are on your Desktop/Laptop? ____________

What is the total RAM and how much is free? ____

What is the Sending and Receiving Rate of your network? ___________________________________

Screenshot Processes Tab and paste here: -

TASK 2.2:

Open **Terminal.**

Type in the following commands and show outcomes:

To identify the available CPU, memory, and disk resources, we can used the following commands:

`cat /proc/cpuinfo` (read the CPU information)

`cat /proc/meminfo` (read the memory (RAM) information)

`df -h` (find out secondary storage (hard-disk) information) `top` is a command line program provides a real-time view of the processes running in the system. It provides system summary and the list of tasks managing by Linux kernel. The program is useful to identify the processes running with CPU and memory utilization. Launch a terminal and execute top command. You can press q to exit from top program.

Does the outcome of top match with the outcome of with System Monitor? ______________________

TASK 2.3

`strace` is a tool that helps to run specified command and traces its interaction with operating system. We can run any program using strace and identify the system calls it makes.

Launch a terminal and run `strace ls`

Try to read the output generated by the program and identify the system calls.

EXERCISE 2.1 **[4]**

Execute `strace` for the code created by you in last experiment. (there must be two codes)

Type commands here:

Show outcomes here:

TASK 2.4

Type the following code and compile this program: -

```
    /*
    The    code    is    taken    from    http://www.daniweb.com/software-
    development/c/code/216411/reading-a-file-line-by-line */
    #include  <stdio.h>
    int main ( void ) {
         static const char filename[] = "file.txt"; FILE *file = fopen
         (filename, "r" );
         if ( file != NULL ) {
              char line [ 128 ];
                        /* or other suitable maximum line size */
              while ( fgets ( line, sizeof line, file ) != NULL ) {
                        /* read a line */
                   fputs ( line, stdout ); /* write the line */
              }
              fclose ( file );
```

```
        }
        else{
             perror ( filename ); /* why didn't the file open? */
        }
        return 0;
}
```

Execute the code. What is the outcome?

Now strace the executable and show outcome here: -

SIMPLE BATCH/SHELL SCRIPT:

In Linux we have a command interpreter known as shell. In this section, we practice writing a very simple shell script. The objective is to write set of commands in a file and run the file to understand the batch execution interface.

TASK 2.5

Create a File named **hello.sh.**

Type in the following code:

```
#!/bin/sh
echo  "Hello!  Lets  Execute
ls."
ls
```

Now type `sh hello.sh` and press enter.

What was the outcome? What did you understand?

EXERCISE 2.2 **[6]**

Search for any file-based code in C for Linux. Download it. Where did you download it from?

Compile and Execute code and show outcome here:

`strace` your code and show outcome here:

RESOURCES:

https://www3.ntu.edu.sg/home/ehchua/programming/cpp/gcc_make.html

https://en.wikipedia.org/wiki/Linux

https://www.youtube.com/watch?v=IVquJh3DXUA

https://www.youtube.com/watch?v=oLjN6jAg-sY

EXPERIMENT 3 – PROCESSES

OBJECTIVE

Learn to create processes using `fork()` system call.

TIME REQUIRED	:	3 hrs
PROGRAMMING LANGUAGE	:	C/C++
SOFTWARE REQUIRED	:	Ubuntu/Fedora, gcc/gc, Text Editor, Terminal, Windows, Dev
HARDWARE REQUIRED	:	Core i5 in Computer Labs

PROCESS CREATION

A process is a program in execution. The process which creates another process is called parent process. The process which is created is called child process. We can identify process by their unique key called process identifier or pid (integer number). In Linux we can use fork() system call to create processes. By this system call new process is created containing copy of parent process. Both process (parent and child) continue executing instructions after fork(). The return number of fork() for new (child) process will be 0 (zero), whereas for parent process value will be nonzero positive process identifier. If fork() fails, it return a negative number. In this section, we create a simple program using fork() to create child process using

TASK 3.1

Using `getpid()`: This function returns the pid of the current program. Use the following code and write the output.

```
int main(){
     int pid;
     pid = getpid();
     printf(“Process ID is %d\n”, pid);
     return 0;
}
```

TASK 3.2

What is the outcome of the following program?

```
int main(){
     long i;
     printf(“Process ID is %d\n”, getpid());
     for(i=0; i<=400;i++){
          printf(“i is %d\n”, i);
     }
     return 0;
}
```

TASK 3.3

Using `getppid()`: This function returns the pid of the parent process.

```
int main(){
     int ppid;
     ppid = getppid();
     printf(“Parent Process ID is %d\n”, ppid);
     return 0;
}
```

What is the outcome?

TASK 3.4

Using `fork()`: fork command in Linux creates a new process by duplicating the calling process. The new process, referred to as the **child**, is an exact duplicate of the calling process, referred to as the **parent**. What is the outcome of the following program?

```
int main(){
     fork();
     printf(“The PID is %d\n”, getpid());
     return 0;
}
```

TASK 3.5

What is the outcome of the following program?

```
int main(){
     int pid;
     pid = fork();
     if(pid==0){
          printf(“I am child, my process ID is %d\n”, getpid());
          printf(“The  parent process ID is   %d\n”, getppid());
     }
     else{
          printf(“I am parent, my process ID is   %d\n”, getpid());
          printf(“The parent process ID is   %d\n”, getppid());
     }
     return 0;
}
```

TASK 3.6

To see if the pid is same as shown in the system, Open System Monitor. Check to see if the pid is same. Use the following code

```
int main(){
	int pid,i;
	pid = fork();
	if(pid==0){
		printf(“I am child, my process ID is   %d\n”, getpid());
		printf(“The  parent  process ID is   %d\n”, getppid());
	}
	else{
		printf(“I am parent, my process ID is   %d\n”, getpid());
		printf(“The parent process ID is   %d\n”, getppid());
	}
	scanf(“%d”,&i);   //so that program halts for user input
	return 0;
}
```

Show screenshots here: -

TASK 3.7

What is the outcome of this program?

```
/**
 * This program forks a separate process using the fork()/exec()
system calls.
 * * Figure 3.10*
 * @author Gagne, Galvin, Silberschatz Operating System Concepts  -
Seventh Edition
 * Copyright John Wiley & Sons - 2005. */
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
int main(){
pid_t pid;
     /* fork a child process */
     pid = fork();
     if (pid < 0) { /* error occurred */
          fprintf(stderr, "Fork Failed\n");
          exit(-1);
     }
     else if (pid == 0) { /* child process */
          printf("I am the child %d\n",pid);
          execlp("/bin/ls","ls",NULL);
     }
     else { /* parent process */
          /* parent will wait for the child to complete */
          printf("I am the parent %d\n",pid);
          wait(NULL);

          printf("Child Complete\n");
          exit(0);
     }
}
```

EXERCISE 3.1 **[2]**

Explain how was the execution carried out in program in last code? You may use a flow model to describe it.

TASK 3.8

Using `Exec`: The *fork* system call creates a new process but that process contains, and is executing, exactly the same code that the parent process has. More often than not, we'd like to run a new program.

Example of Execve(): The *execve* system call replaces the current process with a new program. Type the following command in Terminal and show the output.

```
ls -aF /
```

Now execute the following code

```
/* execve: run a program */
#include <stdlib.h>  /* needed to define exit() */
#include <unistd.h>  /* needed to define getpid() */
#include <stdio.h>   /* needed for printf() */
int main(int argc, char **argv) {
        char *args[] = {"ls", "-aF", "/", 0};
              /* each element represents a command line argument */
        char *env[] = { 0 };
              /* leave the environment list null */
```

```
            printf("About to run /bin/ls\n");
            execve("/bin/ls", args, env);
            perror("execve");  /* if we get here, execve failed */
            exit(1);
     }
```
What is the outcome?

TASK 3.9

Using Execlp

execlp, which allows you to specify all the arguments as parameters to the function. Note that the first parameter is the command. The second parameter is the first argument in the argument list that is passed to the program (argv[0]). These are often the same but don't have to be. The last parameter must be a null pointer.

```
     /* execlp: run a program using execlp */
     #include <stdlib.h> /* needed to define exit() */
     #include <unistd.h> /* needed to define getpid() */
     #include <stdio.h>  /* needed for printf() */
     int main(int argc, char **argv) {
          printf("About to run ls\n");
          execlp("ls", "ls", "-aF", "/", (char*)0);
          perror("execlp");     /* if we get here, execlp failed */
          exit(1);
     }
```
What is the output?

TASK 3.10

Using `fork()` and `exec()`: The fork system call creates a new process. The execve system call overwrites a process with a new program. A process forks itself and the child process execs a new program, which overlays the one in the current process.

```
/* forkexec: create a new process. */
/*  The child runs "ls -aF /". The parent wakes up after 5 seconds
*/
#include <stdlib.h>  /* needed to define exit() */
#include <unistd.h>  /* needed for fork() and getpid() */
#include <stdio.h>   /* needed for printf() */
int main(int argc, char **argv) {
     void runit(void);
     int pid; /* process ID */
     switch (pid = fork()) {
     case 0:         /* a fork returns 0 to the child */
          runit();
          break;
     default: /* a fork returns a pid to the parent */
          sleep(5);  /* sleep for 5 seconds */
          printf("I'm still here!\n");
          break;
     case -1: /* something went wrong */
          perror("fork");
          exit(1);
     }
     exit(0);
}
void runit(void) {
     printf("About to run ls\n");
     execlp("ls", "ls", "-aF", "/", (char*)0);
     perror("execlp");     /* if we get here, execlp failed */
     exit(1);
}
```

What is the outcome of the program?

EXERCISE 3.2 **[2]**

What do you understand from last code?

TASK 3.11

Creating a Process in Windows

```
/**
 * This program creates a separate process using the
CreateProcess() system call. Figure 3.12
 * @author Gagne, Galvin, Silberschatz Operating System Concepts  -
Seventh Edition
 * Copyright John Wiley & Sons - 2005.
 */
#include <windows.h>
#include <stdio.h>
int main( VOID ){
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    ZeroMemory( &si, sizeof(si) );
    si.cb = sizeof(si);
    ZeroMemory( &pi, sizeof(pi) );     // Start the child process.
    if( !CreateProcess( NULL,
                    // No module name (use command line).
        "C:\\WINDOWS\\system32\\mspaint.exe", // Command line.
        NULL,             // Process handle not inheritable.
        NULL,             // Thread handle not inheritable.
```

```
            FALSE,          // Set handle inheritance to FALSE.
            0,              // No creation flags.
            NULL,           // Use parent's environment block.
            NULL,           // Use parent's starting directory.
            &si,           // Pointer to STARTUPINFO structure.
            &pi )          // Pointer to PROCESS_INFORMATION structure.
        )
         {
            printf( "CreateProcess failed (%d).\n", GetLastError() );
            return -1;
         }
         // Wait until child process exits.
         WaitForSingleObject( pi.hProcess, INFINITE );
         // Close process and thread handles.
         CloseHandle( pi.hProcess );
         CloseHandle( pi.hThread );
     }
```
What is the Outcome?

EXERCISE 3.3 **[6]**

Which OS gives you a better interface for creating and executing child programs and why?

RESOURCES

http://manpages.ubuntu.com/manpages/lucid/man2/fork.2.html

http://www.cs.rutgers.edu/~pxk/416/notes/c-tutorials/exec.html

http://www.cs.rutgers.edu/~pxk/416/notes/c-tutorials/forkexec.html

http://www.advancedlinuxprogramming.com/alp-folder/alp-ch03-processes.pdf

EXPERIMENT 4 – PROCESS SCHEDULING

OBJECTIVES

- Learning process scheduling
- Comparing among FCFS and SJF scheduling algorithms through a simulation (pre-emptive and non-pre-emptive)

TIME REQUIRED	:	3 hrs
PROGRAMMING LANGUAGE	:	C/C++/Java
SOFTWARE REQUIRED	:	Ubuntu/Fedora, gcc/gc, Windows, Dev, NetBeans
HARDWARE REQUIRED	:	Core i5 in Computer Labs

PROCESS SCHEDULERS:

Process schedulers are responsible for selecting which process to execute so that overall performance is increased.

PERFORMANCE METRICS:

Metrics include throughput, CPU overhead, response time, average wait time etc.

FIRST COME FIRST SERVED (FCFS) SCHEDULER:

This scheduler implements a FIFO Queue and schedules all process on their order of arrival.

http://cs.uttyler.edu/Faculty/Rainwater/COSC3355/Animations/fcfs.htm

SHORTEST JOB FIRST (SJF) SCHEDULER:

This scheduler priorities processes based on their CPU usage time.

http://sharingmythreeyears.blogspot.com/

COMPARISON OF SCHEDULERS:

Implement a queue of length 5 to hold processes for FCFS and SJF both and for different algorithms observe the throughput at each second. To observe throughput, you have to mark at each instant how many processes have been executed for each scheduler.

Assume that all processes arrive at initial time and burst time is available for each process.

Process ID	Burst Time (Seconds)
P1	5
P2	8
P3	11
P4	4
P5	2

Process ID	Burst Time (Seconds)
P1	5
P2	10
P3	15
P4	20
P5	25

Process ID	Burst Time (Seconds)
P1	25
P2	20
P3	15
P4	10
P5	5

EXERCISE 4.1 **[2]**

Given the states in schedule using FCFS and SJF both techniques and plot throughput on a graph. The horizontal specifies the time and the vertical marks the total number of processes completed so far.

EXERCISE 4.2 **[3]**

Calculate average wait time for each algorithm and for all three states on Points 10, 11 and 12. Average wait time is the sum of waiting times by each process divided by the total number of processes

EXERCISE 4.3 **[5]**

What is your conclusion based on the throughput and average wait time as well as the type of input?

RESOURCES

http://www.cosmolearning.com/video-lectures/concurrency-processes-threads-and-address-spaces-7406/

http://cs.uttyler.edu/Faculty/Rainwater/COSC3355/Animations/fcfs.htm

EXPERIMENT 5 – PRIORITY SCHEDULER

OBJECTIVES

- Defining data word and data byte variable
- Identification of Flag register
- Type of flag register

TIME REQUIRED	:	3 hrs
PROGRAMMING LANGUAGE	:	C/C++/Java
SOFTWARE REQUIRED	:	Ubuntu/Fedora, gcc/gc, Windows, Dev, NetBeans
HARDWARE REQUIRED	:	Core i5 in Computer Labs

PROCESS SCHEDULERS:

Process schedulers are responsible for selecting which process to execute so that overall performance is increased.

PERFORMANCE METRICS:

Metrics include throughput, CPU overhead, response time, average wait time etc.

PRIORITY SCHEDULER:

In this scheduler each process is assigned a priority. Process with highest priority is to be executed first and so on. Processes with same priority are executed on first come first served basis. Priority can be decided based on memory requirements, time requirements or any other resource requirement.

ROUND ROBIN (RR) SCHEDULER:

Each process is provided a fix time to execute, it is called a quantum. Once a process is executed for a given time period, it is pre-empted and other process executes for a given time period. Context switching is used to save states of pre-empted processes.

PRIORITY SCHEDULING ALGORITHM

A Process Scheduler schedules different processes to be assigned to the CPU based on scheduling algorithms. There are six popular process scheduling algorithms:-

- First-Come, First-Served (FCFS) Scheduling
- Shortest-Job-Next (SJN) Scheduling
- Priority Scheduling
- Shortest Remaining Time
- Round Robin(RR) Scheduling
- Multiple-Level Queues Scheduling

These algorithms are either non-preemptive or preemptive. Non-preemptive algorithms are designed so that once a process enters the running state, it cannot be preempted until it completes its allotted time, whereas the preemptive scheduling is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state.

EXERCISE 5.1: **[5]**

Processes	**Arrival Time**	**Burst Time**	**Priority**
P1	0	4	2 (L)
P2	1	2	4
P3	2	3	6
P4	3	5	10
P5	4	1	8
P6	5	4	12 (H)
P7	6	6	9

What is completion time, turnaround time, waiting time of each process

Calculate average waiting time and turnaround time?

EXERCISE 5.2: **[5]**

Processes	Arrival Time	Burst Time	Priority
P1	0	4	2 (L)
P2	1	2	4
P3	2	3	6
P4	3	5	10
P5	4	1	8
P6	5	4	12 (H)
P7	6	6	9

What is completion time, turnaround time, waiting time of each process

Calculate average waiting time and turnaround time?

RESOURCES

https://www.tutorialspoint.com/operating_system/os_process_scheduling_algorithms.htm

https://www.tutorialspoint.com/operating_system/os_process_scheduling_algorithms.htm

https://www.studytonight.com/operating-system/priority-scheduling

https://www.youtube.com/watch?v=lw0XyasWxDk

EXPERIMENT 6 – THREAD MANAGEMENT

OBJECTIVES

- Using Java Threads
- Thread Synchronization

TIME REQUIRED	:	3 hrs
PROGRAMMING LANGUAGE	:	C/C++/Java
SOFTWARE REQUIRED	:	Ubuntu/Fedora, gcc/gc, Windows, Dev, NetBeans
HARDWARE REQUIRED	:	Core i5 in Computer Labs

MULTITHREADING:

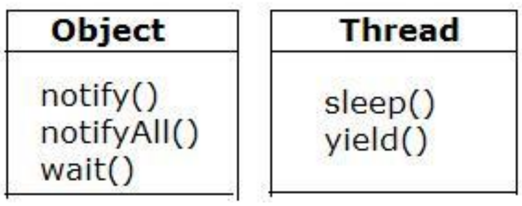
Multithreading refers to two or more tasks executing concurrently within a single program. A thread is an independent path of execution within a program. Many threads can run concurrently within a program.

CONTEXT SWITCH USING THREADS:

Switching among threads is much cheaper than to switch processes as threads share the same process area.

JAVA THREADS:

Every thread in Java is created and controlled by the **java.lang.Thread class**. A Java program can have many threads, and these threads can run concurrently, either asynchronously or synchronously. Following shows the methods in Object and Thread Classes of Java.



THREAD CREATION BY IMPLEMENTING THE RUNNABLE INTERFACE:

```
public interface Runnable {
       void run();
}
```

One way to create a thread in java is to implement the Runnable Interface and then instantiate an object of the class. We need to override the run() method into our class which is the only method that needs to be implemented. The run() method contains the logic of the thread.

- A class implements the Runnable interface, providing the run() method that will be executed by the thread. An object of this class is a Runnable object.
- An object of Thread class is created by passing a Runnable object as argument to the Thread constructor. The Thread object now has a Runnable object that implements the run() method.
- The start() method is invoked on the Thread object created in the previous step. The start() method returns immediately after a thread has been spawned.

The thread ends when the run() method ends, either by normal completion or by throwing an uncaught exception.

THREAD CREATION BY EXTENDING THE THREAD CLASS:

The procedure for creating threads based on extending the Thread is as follows:

- A class extending the Thread class overrides the run() method from the Thread class to define the code executed by the thread.
- This subclass may call a Thread constructor explicitly in its constructors to initialize the thread, using the super() call.
- The start() method inherited from the Thread class is invoked on the object of the class to make the thread eligible for running.

TASK 6.1:

Type in the following code. Execute and show outcome:

```
/**
 * @(#)ThreadExample.java
 * @author Engr. Umar Mahmud
 * @version 1.00 2014/4/25
 */
 public class ThreadExample extends Thread {
          //To create producer and consumer as threads
     public ThreadExample(String threadName){
          //Constuctor
          super(threadName);
               //Call to constructor of Thread class
     }
     public void run(){
               System.out.println("I  am  a  thread  and  my  name  is
               "+Thread.currentThread().getName());
     }
     public static void main(String[] args) {
               ThreadExample  t  =  new  ThreadExample("Thread  01");
               //Create a thread object
               t.start();     //execute run() method
    }
}
```

EXERCISE 6.1 **[3]**

Create another thread object and execute it. Show code and outcome here:

TASK 6.2:

Execute the following code for producer and consumer. Execute and show outcome:

```
/**
 * @(#)ProducerConsumerUsingThreads.java
 * ProducerConsumerUsingThreads application
 * @author Engr. Umar Mahmud
 * @version 1.00 2013/12/20
 */
public class ProducerConsumerUsingThreads extends Thread {
     private int x = 0;       //Shared variable
     private int maxLimit = 5;//Maximum Limit to simulate a buffer
     public ProducerConsumerUsingThreads(String th readName) {
          //Constuctor
        super(threadName);    //Call to constructor of Thread class
     }
     public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.println(Thread.currentThread().getName());
            if(Thread.currentThread().getName().startsWith
                ("Producer")) {            //In a producer
                if (x < maxLimit) {
                    x++;
                } else {
                    System.out.println("\nProducer blocked");
                    yield();  //Give way to another thread
                }
            }
            else   if   (Thread.currentThread().getName().startsWith
               ("Consumer")) {            //In a Consumer
                if (x > 0) {
                    x--;
                } else {
                    System.out.println("\nConsumer blocked");
                    yield();  //Give way to another thread
                }
            } else {     //default case
                System.out.println("Some Error");
            }
       }
    }
```

```
        public static void main(String[] args) {
            ProducerConsumerUsingThreads        p         =          new
              ProducerConsumerUsingThreads("Producer");
            ProducerConsumerUsingThreads        c         =          new
              ProducerConsumerUsingThreads("Consumer");
            p.start();
            c.start();
        }
    }
```

Re-execute the code and identify if the output trace is same as before.

Again re-execute the code and identify if the output trace is same as before.

What is your conclusion?

EXCERSISE 6.2 **[7]**

Create two threads one to read from a file and the other to write into another file. **Show the output and code** for a given text file.

RESOURCES

http://www.youtube.com/watch?v=KxTRsvgqoVQ

http://www.youtube.com/watch?v=D0u2USIonds

http://www.youtube.com/watch?v=CKFq-WTZZc8

http://docs.oracle.com/javase/6/docs/api/java/lang/Thread.html [MUST VISIT]

http://www.javabeginner.com/learn-java/java-threads-tutorial

EXPERIMENT 7 – THREADS IN C

OBJECTIVE

Learn to threads for doing multiple function or parallel processing.

TIME REQUIRED	:	3 hrs
PROGRAMMING LANGUAGE	:	C/C++/Java
SOFTWARE REQUIRED	:	Ubuntu/Fedora, gcc/gc, Windows, Dev, NetBeans
HARDWARE REQUIRED	:	Core i5 in Computer Labs

THREADS:

Application can have multiple processes and process have many threads. Thread is smallest unit of execution to which processor allocates time. It consists of:

- Program Counter (contains the address of next instruction to be executed)
- Stack
- Set of Registers
- Unique ID

However, a thread itself is not a program. It cannot run on its own but runs with in a program.

TASK 7.1:

How to write simple multi-threaded program in C.

```
#include <stdio.h>
#include<pthread.h>
void * show(void * u){
     printf(“new thread”);
}
int main(){
     pthread_t tid;
     pthread_create(&tid,NULL,&show,NULL);
     printf(“main thread”);
     pthread_join(tid,NULL);
     return 0;
}
```

For Compilation and execution:

```
gcc  -o out1 task1.c   -lpthread
./out1
```

Execute code and show outcome here:

Re-execute 3 more times and show outcomes. Did the outcomes change? Why?

TASK 7.2:

```
#include <stdio.h>
#include<pthread.h>
pthread_t thread[2];
static void *funtion1(void *_){
     int i;
     for (i=0;i<=10;i++){
     printf("\nthread1 says%d",i);
     sleep(1);
     }
}
static void *funtion2(void *_){
     int i;
     for (i=0;i<=5;i++){
     printf("\nthread2 says%d",i);
     sleep(5);
     }
}
int main(){
pthread_create(&thread[0],NULL,function1,NULL);
pthread_create(&thread[1],NULL,function2,NULL);
pthread_exit(NULL);
return  1;
}
```

For Compilation and execution:

```
gcc  -o out2  task2.c   -lpthread
./out2
```

Execute Code and show outcome here:

EXERCISE 7.1: **[8]**

Create four more threads that can prints the following: -

- Thread 1 prints positive number from 1 to 20
- Thread 2 prints negative numbers from 1 to 20
- Thread 3 prints even numbers from 1 to 20
- Thread 4 prints odd numbers from 1 to 20

Show code and outcome here:

EXERCISE 7.2: **[2]**

Compare pthreads and Java threads

RESOURCES

https://www.geeksforgeeks.org/multithreading-c-2/

https://www.tutorialspoint.com/multithreading-in-c

EXPERIMENT 8 – IPC USING PIPES

OBJECTIVES

Learn to use pipes for inter process communication.

TIME REQUIRED	:	3 hrs
PROGRAMMING LANGUAGE	:	C/C++/Java
SOFTWARE REQUIRED	:	Ubuntu/Fedora, gcc/gc, Windows, Dev, NetBeans
HARDWARE REQUIRED	:	Core i5 in Computer Labs

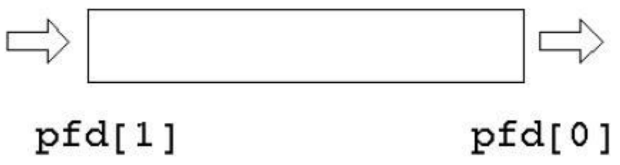
INTER PROCESS COMMUNICATION USING SIMPLE PIPES

Inter- process Communication (IPC) is set of ways to communicate between processes or threads. Pipe is one of the ways to communicate. Pipe is special file; the child inherits the pipe from its parent process. Process writes to one end of the pipe (write-end), other reads from the other end of pipe (read-end). Its unidirectional (one-way communication). For two-way communication we use two pipes.

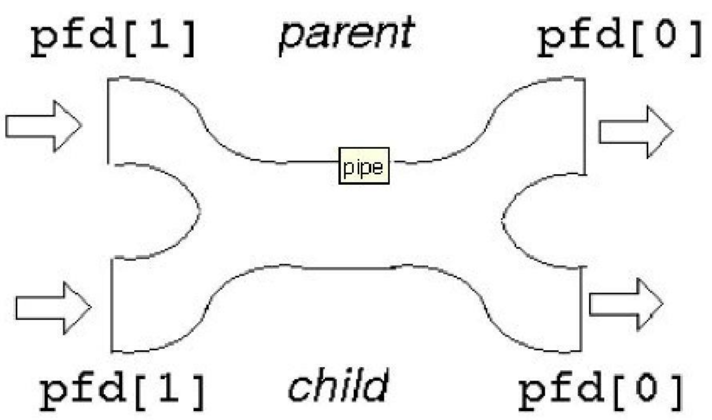
```
int pipe( int fd[2]);
```

Two file descriptors are returned through the argument: fd[0] is open for reading fd[1] is open for writing. Typically, a process creates the pipeline, then uses "fork" to create a child process. Each process now has a copy of the file descriptor array; one process writes data into pipeline, while the other reads from it.

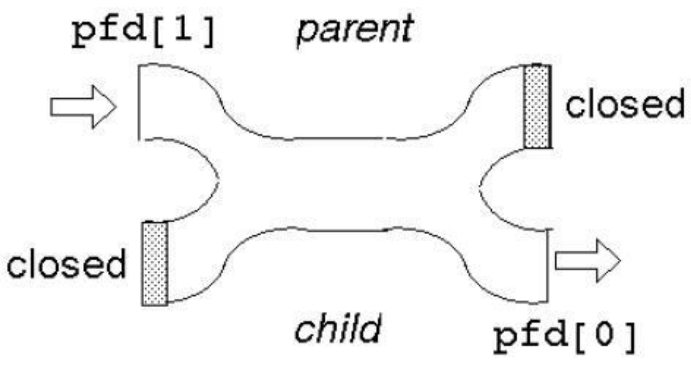
Before fork



After fork



This gives two read ends and two write ends. The read end of the pipe will not be closed until both read ends are closed, and the write end will not be closed until both the write ends are closed. Either process can write into the pipe, or either can read from it. Which process will get what is not known? For predictable behaviour, one of the processes must close its read end, and the other must close its write end. Then it will become a simple pipeline again.



TASK 8.1

Create a file named mypipe.c. Type in the following code:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
int main(void) {
     int fd[2]; pid_t childpid;
     char string[] = "Hello, world!\n"; char readbuffer[80];
     int result = pipe(fd); if (result < 0 ) {
          printf("Error Creating Pipe"); return 1;
     }
     childpid     =     fork();
     if(childpid == -1) {
          printf("Error in fork()"); return 1;
     }
     if(childpid == 0) {
          close(fd[0]);
               /* Send "string" through the output side of pipe */
          printf ("child writing\n");
          write(fd[1], string, sizeof(string)); return 0;
     }
     else {
          close(fd[1]);
          printf ("parent reading\n");
```

```
                  /* Read in a string from the pipe */
              read(fd[0],       readbuffer,       sizeof(readbuffer));
              printf("Received string: %s", readbuffer);
         }
         return 0;
     }
```

Compile, execute and show outcome her: -

EXERCISE 8.1: **[10]**

Update the mypipe .c program to make it bidirectional. Parent process must send an acknowledgment message to the Child. Hint! You may need to create two Pips.

RESOURCES

https://www.geeksforgeeks.org/c-program-demonstrate-fork-and-pipe/

https://www.geeksforgeeks.org/pipe-system-call/

EXPERIMENT 9 – PETERSON'S ALGORITHM

OBJECTIVES

- Learning Critical Section
- Learning how to implement Peterson's algorithm

TIME REQUIRED	:	3 hrs
PROGRAMMING LANGUAGE	:	C/C++/Java
SOFTWARE REQUIRED	:	Ubuntu/Fedora, gcc/gc, Windows, Dev, NetBeans
HARDWARE REQUIRED	:	Core i5 in Computer Labs

PRODUCER-CONSUMER PROBLEM:

In computing, the **producer–consumer problem** (also known as the **bounded-buffer problem**) is a classic example of a multi-process synchronization problem. The problem describes two processes, the producer and the consumer, who share a common, fixed-size buffer used as a queue. The producer's job is to generate data, put it into the buffer, and start again. At the same time, the consumer is consuming the data (i.e., removing it from the buffer), one piece at a time. The problem is to make sure that the producer won't try to add data into the buffer if it's full and that the consumer won't try to remove data from an empty buffer.

TASK 9.1:

Create two processes (producer and consumer) or functions or threads that access the same shared variable. One process increments the variable while the other decrements it. The maximum value of the variable is 5 and the producer cannot produce anymore items if the variable is 5. The consumer cannot consume if the variable is zero.

EXERCISE 9.1 **[2]**

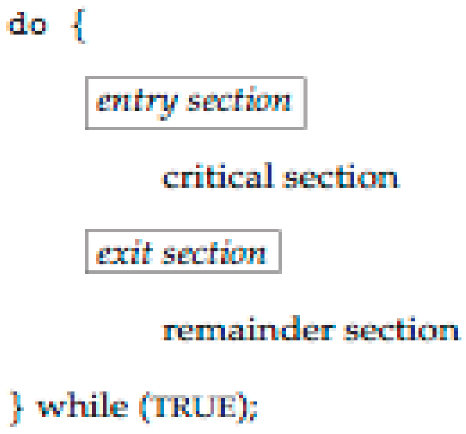
In the main function write a sequence of calls that simulate the producer consumer problem. Output the normal case, the case where the consumer should be blocked and the case where the producer should be blocked. Show the trace only.

EXERCISE 9.2 **[2]**

How does race condition occur in the cases of point 5? Explain and trace the case.

CRITICAL SECTION:

Access to a shared item is considered critical and is carried out in critical section only.



```
do {
      entry section
            critical section
      exit section
            remainder section
} while (TRUE);
```

PETERSON'S SOLUTION:

It provides a solution to the Critical Section.

Two processes share two variables:

int **turn**;

boolean **flag[2];**

//The variable **turn** indicates whose turn it is to enter the critical section

//The **flag** array is used to indicate if a process is ready to enter the critical section. //**flag[i]** = true implies that process **P$_i$** is ready!

ALGORITHM FOR PROCESS P$_I$:

```
do {
          flag[i] = True;
          turn = j;
          while (flag[j] && turn = j){
                              //busy waiting
                         }
          //go to critical section i.e., access the shared item
          flag[i] = FALSE;
          // go to remainder section
     } while (TRUE);
```

EXERCISE 9.3 **[6]**

Implement and show trace of Peterson's Algorithm for two processes. Show code and output here.

RESOURCES

https://en.wikipedia.org/wiki/Producer%E2%80%93consumer_problem

https://en.wikipedia.org/wiki/Peterson%27s_algorithm

http://yab-jab.blogspot.com/2012/03/petersons-algorithm-mutex-for-2.html

EXPERIMENT 10 – SEMAPHORES

OBJECTIVES

- Solving critical section problem using semaphores.
- Semaphore implementation.

TIME REQUIRED	:	3 hrs
PROGRAMMING LANGUAGE	:	C/C++/Java
SOFTWARE REQUIRED	:	Ubuntu/Fedora, gcc/gc, Windows, Dev, NetBeans
HARDWARE REQUIRED	:	Core i5 in Computer Labs

SEMAPHORES

In computer science, a semaphore is a variable or abstract data type used to control access to a common resource by multiple processes in a concurrent system such as a multitasking operating system. A semaphore is simply a variable. This variable is used to solve critical section problems and to achieve process synchronization in the multi-processing environment. A trivial semaphore is a plain variable that is changed (for example, incremented or decremented, or toggled) depending on programmer-defined conditions.

A useful way to think of a semaphore as used in the real-world system is as a record of how many units of a particular resource are available, coupled with operations to adjust that record safely (i.e., to avoid race conditions) as units are required or become free, and, if necessary, wait until a unit of the resource becomes available. Semaphores are a useful tool in the prevention of race conditions; however, their use is by no means a guarantee that a program is free from these problems. Semaphores which allow an arbitrary resource count are called counting semaphores, while semaphores which are restricted to the values 0 and 1 (or locked/unlocked, unavailable/available) are called binary semaphores and are used to implement locks.

The semaphore concept was invented by Dutch computer scientist Edsger Dijkstra in 1962 or 1963, when Dijkstra and his team were developing an operating system for the Electrologica X8. That system eventually became known as THE multiprogramming system.

An integer value used for signalling among processes. Only three operations may be performed on a semaphore, all of which are atomic:

- Initialize,
- Decrement (Wait)
- Increment. (Signal)

```
wait (S){
     while (S <= 0){
          ; // wait
            }
        S--;
 }
```

```
signal
(S) {

S++;
}
```

When one process modifies semaphore value, no other process can simultaneously modify that semaphore. When a process releases resource, it performs signal. When count goes to 0, all resources are being used. After that, process that wish to use a resource will block until count becomes greater than 0.

COUNTING SEMAPHORE:

Integer value can range over an unrestricted domain. Counting semaphores can be used to control access to a given resource with finite number of instances.

BINARY SEMAPHORE:

Integer value can range only between 0 and 1. They are simpler to implement. Binary semaphores can be used to deal with critical section problem.

```
Semaphore mutex;    //  initialized to 1
do {
     wait (mutex);
         // Critical Section
     signal (mutex);
          // remainder section
} while (TRUE);
```

```
struct semaphore {
     int count;
     queueType queue;
};
void semWait(semaphore s)
{
     s.count--;
     if (s.count < 0) {
          /* place this process in s.queue */;
          /* block this process */;
     }
}
void semSignal(semaphore s)
{
     s.count++;
     if (s.count <= 0) {
          /* remove a process P from s.queue */;
          /* place process P on ready list */;
     }
}
```

TASK 10.1 :

Create a semaphore class that implements a semaphore for mutual exclusion using threads. Also implement a producer thread and a consumer thread that acquire the use of a shared resource through the shared semaphore. Both the producer and consumer should implement the wait and signal method. Show code here:

EXERCISE 10.1 **[8]**

Show the trace of a simulation highlighting the following cases: -

- Normal execution.
- Producer is blocked.
- Consumer is blocked.

EXERCISE 10.2 **[2]**

What is your understanding of synchronization through semaphores based on this experiment?

RESOURCES

http://en.wikipedia.org/wiki/Semaphore_(programming)

http://elvis.rowan.edu/~hartley/ConcProgJava/Semaphores/bbou.java

http://www.cs.loyola.edu/~jglenn/702/F2007/Examples/Producer_Consumer_Semaphores/pc_semaphore.html

EXPERIMENT 11 – MODULE PROGRAMMING

OBJECTIVES

- Module programming
- Developing modules in Ubuntu
- Hello World module in Ubuntu

TIME REQUIRED	:	3 hrs
PROGRAMMING LANGUAGE	:	C/C++/Java
SOFTWARE REQUIRED	:	Ubuntu/Fedora, gcc/gc, Windows, Dev, NetBeans
HARDWARE REQUIRED	:	Core i5 in Computer Labs

MODULES:

In computing, a loadable kernel module (or LKM) is an object file that contains code to extend the running kernel, or so-called base kernel, of an operating system. LKMs are typically used to add support for new hardware, file-systems, or for adding system calls. When the functionality provided by an LKM is no longer required, it can be unloaded in order to free memory and other resources.

Modules are programs that execute in the OS shell or Ring 0.

TASK 11.1:

In this experiment the simplest hello world module will be created. The steps are as follows: -

- Make a folder on the desktop for this lab. Make sure that the name of the folder does not have a space character in it. Write the name of the folder here _____________________.
- Create a text file using a text editor with the name **hello-1.c.** You may create any file name but the extension should be **.c.**
- Write the following code in the text file. Text file is edited using the Text Edit tool in Linux

```
#include <linux/module.h>  /* Needed by all modules */
#include <linux/kernel.h>  /* Needed for KERN_INFO */
int init_module(void){
     printk(KERN_INFO "Sky Fall\n");
        // A non 0 return means init_module failed
        // module can't be loaded.
        return 0;
}
void cleanup_module(void){
     printk(KERN_INFO "Quantum of Solace.\n");
}
```

- Do not compile the code yet.

- Kernel modules must have at least two functions: a "start" (initialization) function called init_module() which is called when the module is loaded (insmod) into the kernel, and an "end" (cleanup) function called cleanup_module() which is called just before it is removed (rmmod).
- printk() is not meant to communicate information to the user. It happens to be a logging mechanism for the kernel, and is used to log information or give warnings. To view the log system log has to be seen.
- Open terminal and navigate to the working directory
- Type **lsmod** to list the modules loaded in the kernel.

TASK 11.2

Now the module will be compiled by creating a make file. The steps are as under: -

- Create a new file using the text editor and store it with the name **Makefile** in the working directory. Do not use any extensions while storing the file.
- Type the following commands in the Makefile

```
obj-m += hello-1.o
all:
 make -C /lib/modules/$(shell uname -r)/build M=$(shell pwd)
modules
clean:
 make -C /lib/modules/$(shell uname -r)/build M=$(shell pwd) clean
```

- The first line should show the name of your file with .o extension after the += characters.
- Do not forget to put a tab character before make in both instances of the keyword make.
- In the terminal type **sudo make.** If asked for the password type in the password.
- What is the outcome of this step (if correctly executed)?

- Show the output as well as the names of the files created in the working directory.

- Use **modinfo hello-1.ko** to see what kind of information it is. Show the output here.

TASK 11.3

To load the module in kernel type **sudo insmod ./hello-1.ko** and remember to rename hello-1 if you have different file name. Type **lsmod** and see if the module is loaded. Show the screen capture here.

TASK 11.4

To view log type **dmesg**. Show the output here.

TASK 11.5

To remove module type **sudo rmmod hello-1** and remember to rename hello-1 if you have different file name.

Type **lsmod** and see if the module is unloaded.

Type **dmesg** again and show the output here.

TASK 11.6

Write the following code in a new file. Name the file **hello-2.c**

```
#include <linux/module.h>  /* Needed by all modules */
#include <linux/kernel.h>  /* Needed for KERN_INFO */
#include <linux/init.h>    /* Needed for the macros */
static int __init hello_2_init(void){
printk(KERN_INFO "Hello, world 2\n");
 return 0;
}
static void __exit hello_2_exit(void){
 printk(KERN_INFO "Goodbye, world 2\n");
}
module_init(hello_2_init);
module_exit(hello_2_exit);
```

Remember that there are two underscores before init and exit keywords. Modify the make file as follows:

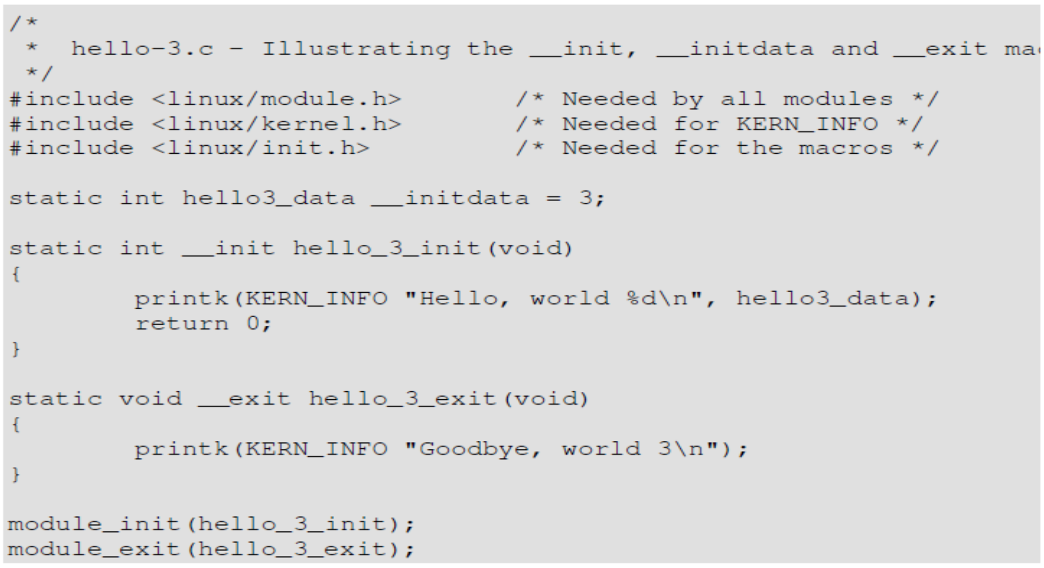
```
obj-m += hello-1.o
obj-m += hello-2.o
all:
 make -C /lib/modules/$(shell uname -r)/build M=$(shell pwd)
     modules
clean:
 make -C /lib/modules/$(shell uname -r)/build M=$(shell pwd) clean
```

Type **sudo make** to compile new module. Check the module directory by typing **lsmod.** Insert modules by typing **sudo insmod ./hello-1.ko** and **sudo insmod ./hello-2.ko** and remember to rename filenames if you have different file name. Check the log by typing **dmesg**. Show the output here.

Remove the modules by typing **sudo rmmod hello-1** and then **sudo rmmod hello-2.** Show the output of **dmesg**

TASK 11.7

The __init macro causes the init function to be discarded and its memory freed once the init function finishes for built–in drivers, but not loadable modules. If you think about when the init function is invoked, this makes perfect sense. There is also an __initdata which works similarly to __init but for init variables rather than functions.Type the following code and name it **hello-3.c**



```
/*
 *  hello-3.c - Illustrating the __init, __initdata and __exit ma
 */
#include <linux/module.h>       /* Needed by all modules */
#include <linux/kernel.h>       /* Needed for KERN_INFO */
#include <linux/init.h>         /* Needed for the macros */

static int hello3_data __initdata = 3;

static int __init hello_3_init(void)
{
        printk(KERN_INFO "Hello, world %d\n", hello3_data);
        return 0;
}

static void __exit hello_3_exit(void)
{
        printk(KERN_INFO "Goodbye, world 3\n");
}

module_init(hello_3_init);
module_exit(hello_3_exit);
```

Load the module and show output here:

EXERCISE 11.1. **[10]**

What did you learn in this experiment?

RESOURCES

https://en.wikipedia.org/wiki/Loadable_kernel_module

www.tldp.org/LDP/lkmpg/2.6/lkmpg.pdf

http://www.tldp.org/LDP/lkmpg/2.6/html/index.html

EXPERIMENT 12 – MEMORY MANAGEMENT

OBJECTIVES

To write a program for first fit and best fit algorithm for memory management.

TIME REQUIRED	:	3 hrs
PROGRAMMING LANGUAGE	:	C/C++/Java
SOFTWARE REQUIRED	:	Ubuntu/Fedora, gcc/gc, Windows, Dev, NetBeans
HARDWARE REQUIRED	:	Core i5 in Computer Labs

MEMORY MANAGEMENT ALGORITHMS:

In an environment that supports dynamic memory allocation, the memory manager must keep a record of the usage of each allocate able block of memory. This record could be kept by using almost any data structure that implements linked lists. An obvious implementation is to define a free list of block descriptors, with each descriptor containing a pointer to the next descriptor, a pointer to the block, and the length of the block. The memory manager keeps a free list pointer and inserts entries into the list in some order conducive to its allocation strategy. Several strategies are used to allocate space to the processes that are competing for memory.

FIRST FIT:

Another strategy is first fit, which simply scans the free list until a large enough hole is found. Despite the name, first-fit is generally better than best-fit because it leads to less fragmentation. Small holes tend to accumulate near the beginning of the free list, making the memory allocator search farther and farther each time. **Solution:** Next Fit

BEST FIT:

The allocator places a process in the smallest block of unallocated memory in which it will fit. It requires an expensive search of the entire free list to find the best hole. More importantly, it leads to the creation of lots of little holes that are not big enough to satisfy any requests. This situation is called *fragmentation* and is a problem for all memory-management strategies, although it is particularly bad for best-fit. One way to avoid making little holes is to give the client a bigger block than it asked for. For example, we might round all requests up to the next larger multiple of 64 bytes. That doesn't make the fragmentation go away, it just hides it. Unusable space in the form of holes is called *external fragmentation*

EXERCISE 12.1 **[5]**

Start the program.

- Get the number of segments and size.
- Get the memory requirement and select the option.
- If the option is ‘2’ call first fit function.
- If the option is ‘1’ call best fit function.

- Otherwise exit.
- For first fit, allocate the process to first possible segment which is free.
- For best fit, do the following steps.
    - Sorts the segments according to their sizes.
    - Allocate the process to the segment which is equal to or slightly greater than the process size.
- Stop the program.

Implement the algorithm, show the outcome and code here.

FIFO:

The frames are empty in the beginning and initially no page fault occurs, so it is set to minus one. When a page fault occurs the page reference string is brought into memory. The operating system keeps track of all the pages in memory, herby keeping track of the most recently arrived and the oldest one. If the page in the page reference string is not in memory, the page fault is incremented, and the oldest page is replaced. If the page in the page reference string is in the memory, take the next page without calculating the page fault. Take the next page in the page reference string and check if the page is already present in the memory or not. Repeat the process until all the pages are referred and calculate the page fault for all those pages in the page reference string for the number of available frames.

LRU:

A good approximation to the optimal algorithm is based on the observation that pages that have been heavily used in the last few instructions will probably be heavily used again in the next few. Conversely, pages that have not been used for ages will probably remain unused for a long time. This idea suggests a realizable algorithm: when a page fault occurs, throw out the page that has been unused for the longest time. This strategy is called LRU (Least Recently Used) paging.

EXERCISE 12.2 **[5]**

- Start the program
- Obtain the number of sequences, number of frames and sequence string from the user
- Now when a page is not in the frame comes, increment the number of page fault and remove the page that come in the first in FIFO algorithm
- In LRU algorithm, when a page fault occurs, the page which most recently used is removed
- Display the number of faults
- Stop the program

Implement the code and show the outcome with code here

RESOURCES

https://en.wikipedia.org/wiki/Memory_management

EXPERIMENT 13 – SOCKETS

OBJECTIVE:

- Socket programming
- Communication using sockets

TIME REQUIRED	:	3 hrs
PROGRAMMING LANGUAGE	:	C/C++/Java
SOFTWARE REQUIRED	:	Ubuntu/Fedora, gcc/gc, Windows, Dev, NetBeans
HARDWARE REQUIRED	:	Core i5 in Computer Labs

SOCKET-DEFINITION:

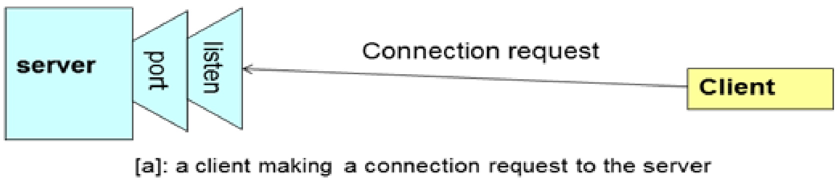
A socket is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent.

SERVER:

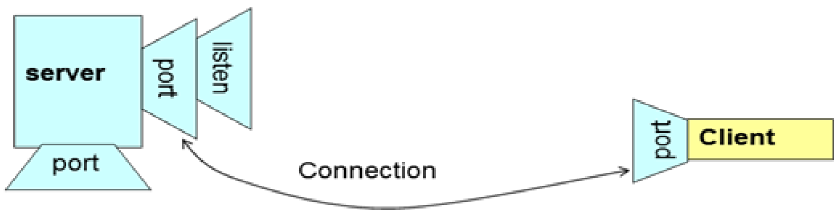
A server runs on a specific computer and has a socket that is bound to a specific port number. The server just waits, listening to the socket for a client to make a connection request.

CLIENT:

The client knows the hostname of the machine on which the server (has a socket) is running and the port number on which the server is listening. The client also needs to identify itself to the server so it binds to a local port number that it will use during this connection.



[a]: a client making a connection request to the server



[b]: session established with temporary ports used for two way communication.

SOCKET AND SERVERSOCKET:

Socket and ServerSocket classes are used to implement socket programming. ServerSocket has constructors that create new ServerSocket objects, methods that listen for connections on a specified port, and methods that return a Socket object when a connection is made so that you can send and receive data. A new ServerSocket is created on a particular port using a ServerSocket( ) constructor. In following activity student will be able to find out open port on Local machine.

TASK 13.1

Code the following server socket:

```
    public ServerSocket(int port) throws IOException, BindException
    //   (LocalPortScanner.java)
    import java.net.*;
    import java.io.*;
    public class LocalPortScanner{
        public static void main(String args[]) throws IOException{
            for(int port=1024;port<=65535;port++){
                try{
                    ServerSocket server=new ServerSocket(port);
                }
                catch(IOException e){
                    System.out.println("Open ports "+port);
                }
            }
        }
    }
```

IMPLEMENTATION OF ECHO CLIENT-SERVER APPLICATION:

This Activity implements a client, EchoClient that connects to the Echo server. The Echo server simply receives data from its client and echoes it back. The Echo server is a service built into most operating systems. EchoClient creates a socket and gets a connection to the Echo server. It reads input from the user on the standard input stream, and then forwards that text to the Echo server by writing the text to the socket. The server echoes the input back through the socket to the client. The client program reads and displays the data passed back to it from the server.

TASK 13.2

Code the following client socket:

```
    //(EchoServer.java)
    import java.io.*;
    import java.net.*;
    public class EchoServer{
        public EchoServer(int portnum){
            try{
                server = new ServerSocket(portnum);
            }
            catch (Exception err){
                System.out.println(err);
            }
        }
```

```
     public void serve(){
          try{
               while (true){
               Socket client = server.accept();
               BufferedReader    r    =    new    BufferedReader(new
                    InputStreamReader (client.getInputStream()));
               PrintWriter            w            =            new
                    PrintWriter(client.getOutputStream(), true);
               w.println("Welcome to EchoServer. 'bye' to close.");
               String line;
               do{
                    line = r.readLine();   //read from Cleint
                    if ( line != null )
                         w.println("Got: "+ line);
               }
               while ( !line.trim().equals("bye") );
                    client.close();
               }
          }
          catch (Exception err){
               System.err.println(err);
          }
     }
     public static void main(String[] args){
          EchoServer s = new EchoServer(9999);
          s.serve();
     }
     private ServerSocket server;
}

//(EchoClient.java)
import java.io.*;
import java.net.*;
public class EchoClient{
     public static void main(String[] args){
          try{
          Socket s = new Socket("127.0.0.1", 9999);
          BufferedReader     r     =     new     BufferedReader(new
               InputStreamReader (s.getInputStream()));
          PrintWriter   w   =   new   PrintWriter(s.getOutputStream(),
               true);
```

```
                BufferedReader    con    =    new    BufferedReader(new
                     InputStreamReader(System.in));
                String line;
                do{
                     line = r.readLine();  //Read from Server
                     if ( line != null )
                          System.out.println(line); //print if not null
                     line = con.readLine(); //Read from user
                     w.println(line);       //send to server
                }
                while ( !line.trim().equals("bye") );
           }
           catch (Exception err){
                System.err.println(err);
           }
     }
}
```

EXERCISE 13.1. **[2]**

Execute the code and show the outcome here.

EXERCISE 13.2. **[8]**

Write a code that reads a file from the client and sends to the server. The server then writes the file at the server end. Show code and output here.

RESOURCES

http://www.oracle.com/technetwork/java/socket-140484.html

http://www.tutorialspoint.com/java/java_networking.htm

EXPERIMENT 14 – FILE SYSTEMS

OBJECTIVE:

- File system information
- Reading file systems information in java
- Listing files in a directory

TIME REQUIRED	:	3 hrs
PROGRAMMING LANGUAGE	:	C/C++/Java
SOFTWARE REQUIRED	:	Ubuntu/Fedora, gcc/gc, Windows, Dev, NetBeans
HARDWARE REQUIRED	:	Core i5 in Computer Labs

FILE SYSTEMS:

In computing, a **file system** (or **filesystem**) is used to control how data is stored and retrieved. Without a file system, information placed in a storage area would be one large body of data with no way to tell where one piece of information stops and the next begins. By separating the data into individual pieces, and giving each piece a name, the information is easily separated and identified. Taking its name from the way paper-based information systems are named, each piece of data is called a "file". The structure and logic rules used to manage the groups of information and their names is called a "file system".

There are many kinds of file systems. Each one has different structure and logic, properties of speed, flexibility, security, size and more. Some file systems have been designed to be used for specific applications. For example, the ISO 9660 file system is designed specifically for optical disks.

File systems can be used on many kinds of storage devices. Each storage device uses a different kind of media. The most common storage device in use today is a hard drive whose media is a disc that has been coated with a magnetic film. The film has ones and zeros 'written' on it sending electrical pulses to a magnetic "read-write" head. Other media that are used are magnetic tape, optical disc, and flash memory. In some cases, the computer's main memory (RAM) is used to create a temporary file system for short term use.

TASK 14.1:

To get the file systems details type in the following code in NetBeans.

```
package fileallocationtableexample;
/*
 *   @http://stackoverflow.com/questions/15656139/get-partition-and-
volume-information?rq=1
 */
import java.io.File;
import javax.swing.filechooser.FileSystemView;
public class FileAllocationTableExample {
    public static void main(String[] args) {
        System.out.println("File     system     roots     returned
byFileSystemView.getFileSystemView():");
```

```
            FileSystemView fsv = FileSystemView.getFileSystemView();
            File[] roots = fsv.getRoots();
            for (int i = 0; i < roots.length; i++) {
                System.out.println("Root: " + roots[i]);
            }

            System.out.println("Home directory: " +
    fsv.getHomeDirectory());

            System.out.println("File system roots returned by
    File.listRoots():");
            File[] f = File.listRoots();
            for (int i = 0; i < f.length; i++) {
                System.out.println("Drive: " + f[i]);
                System.out.println("Display name: " +
    fsv.getSystemDisplayName(f[i]));
                System.out.println("Is drive: " + fsv.isDrive(f[i]));
                System.out.println("Is floppy: " +
    fsv.isFloppyDrive(f[i]));
                System.out.println("Readable: " + f[i].canRead());
                System.out.println("Writable: " + f[i].canWrite());
                System.out.println("Total space: " +
    f[i].getTotalSpace());
                System.out.println("Usable space: " +
    f[i].getUsableSpace());
            }
        }
}
```

Execute the Java code and show the output here: -

Re-execute the Java Code and connect your mobile phone as well as USB devices with your system. Also insert a DVD/CD in the ROM. Display output here: -

Exercise 14.1. **[2]**

What is the difference in the above outputs? What kind of information was shared?

TASK 14.2

Import the org.openide.filesystems.FileSystem class in a java code and execute the getDefault() function as shown below

```
package fileallocationtableexample;
import java.nio.file.FileSystems;
/**
 *@http://bits.netbeans.org/dev/javadoc/org-openide-
filesystems/org/openide/filesystems/FileSystem.html
 */
public class TestFileSystem {
    public static void main(String args[]){
        System.out.println(FileSystems.getDefault());
    }
}
```

What is the output? Show here.

TASK 14.3

Execute the following code to display the roots.

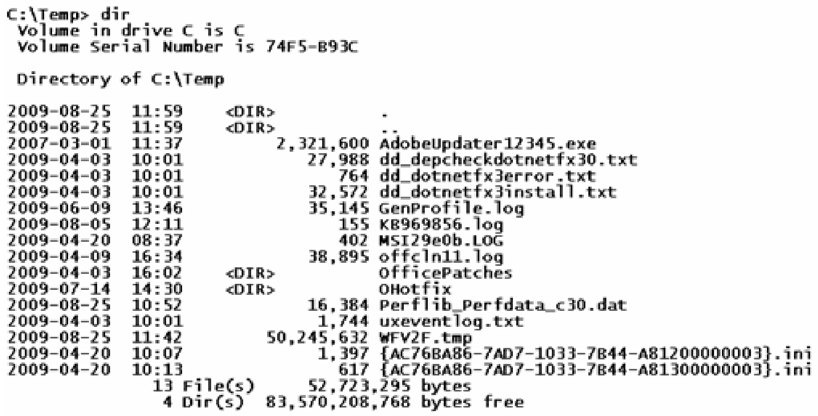
```
package fileallocationtableexample;
/**
* @http://www.tutorialspoint.com/java/io/file_listroots.htm
 */
import java.io.File;
public class FileRoots {
   public static void main(String[] args) {
      File[] paths;
        try{
         // returns pathnames for files and directory
         paths = File.listRoots();

         // for each pathname in pathname array
         for(File path:paths)
         {
            // prints file and directory paths
            System.out.println(path);
         }
      }catch(Exception e){
         // if any error occurs
         e.printStackTrace();
      }
   }
}
```

What is the output? Show here: -

DIRECTORY:

In computing, a **directory** is a file system cataloguing structure which contains references to other computer files, and possibly other directories. On many computers directories are known as **folders**, **catalogues** (used on the Apple II, the Commodore 128 and some other early home computers as a command for displaying disk contents - the file systems used by these DOS did not support hierarchal directories), or **drawers** to provide some relevancy to a workbench or the traditional office file cabinet. On Microsoft Windows, the terms *folder* and *directory* are used interchangeably. Files are organized by storing related files in the same directory. In a hierarchical file system (that is, one in which files and directories are organized in a manner that resembles a tree), a directory contained inside another directory is called a **subdirectory**. The terms **parent** and **child** are often used to describe the relationship between a subdirectory and the directory in which it is catalogued, the latter being the parent. The top-most directory in such a files system, which does not have a parent of its own, is called the **root** directory. Following figure shows a directory listing:



```
C:\Temp> dir
 Volume in drive C is C
 Volume Serial Number is 74F5-B93C

 Directory of C:\Temp

2009-08-25  11:59    <DIR>          .
2009-08-25  11:59    <DIR>          ..
2007-03-01  11:37         2,321,600 AdobeUpdater12345.exe
2009-04-03  10:01            27,988 dd_depcheckdotnetfx30.txt
2009-04-03  10:01               764 dd_dotnetfx3error.txt
2009-04-03  10:01            32,572 dd_dotnetfx3install.txt
2009-06-09  13:46            35,145 GenProfile.log
2009-08-05  12:11               155 KB969856.log
2009-04-20  08:37               402 MSI29e0b.LOG
2009-04-09  16:34            38,895 offcln11.log
2009-04-03  16:02    <DIR>          OfficePatches
2009-07-14  14:30    <DIR>          OHotfix
2009-08-25  10:52            16,384 Perflib_Perfdata_c30.dat
2009-04-03  10:01             1,744 uxeventlog.txt
2009-08-25  11:42        50,245,632 WFV2F.tmp
2009-04-20  10:07             1,397 {AC76BA86-7AD7-1033-7B44-A81200000003}.ini
2009-04-20  10:13               617 {AC76BA86-7AD7-1033-7B44-A81300000003}.ini
              13 File(s)     52,723,295 bytes
               4 Dir(s)  83,570,208,768 bytes free
```

TASK 14.4

The following code gives the directory listing using C++.

```
#include <fstream>
#include <iostream>
#include <string>
#include <dirent.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
using namespace std;
int main()
  {
  ifstream fin;
  string dir, filepath;
  int num;
  DIR *dp;
  struct dirent *dirp;
  struct stat filestat;

  cout << "dir to get files of: " << flush;
  getline( cin, dir );  // gets everything the user ENTERs

  dp = opendir( dir.c_str() );
  if (dp == NULL)
    {
    cout << "Error(" << errno << ") opening " << dir << endl;
    return errno;
    }
  while ((dirp = readdir( dp )))
    {
    filepath = dir + "/" + dirp->d_name;
    // If the file is a directory (or is in some way invalid) we'll
skip it
    if (stat( filepath.c_str(), &filestat )) continue;
    if (S_ISDIR( filestat.st_mode ))         continue;
    // read a single number from the file and display it
    fin.open( filepath.c_str() );
    if (fin >> num)
      cout << filepath << ": " << num << endl;
    fin.close();
```

```
        }
      closedir( dp );
      return 0;
     }
```

Execute this code and show the output here: -

TASK 14.5

The following code shows the directory listing of working directory.

```
package test;
import java.io.File;
public class ListFiles {
    public static void main(String[] args) {
        // Directory path here
        String path = ".";   //path name here
        String files;
        File folder = new File(path);
        File[] listOfFiles = folder.listFiles();
        for (int i = 0; i < listOfFiles.length; i++) {
            if (listOfFiles[i].isFile()) {
                files = listOfFiles[i].getName();
                System.out.println(files);
            }
        }
    }
}
```

Execute this code and show the output here: -

EXERCISE 14.2 **[8]**

Update both codes that shows the directory listing of a given directory in a column format showing the File Name, File Size, and File Rights (Read only, Write Only, Both) and Last Accessed Time and date. Show the output of both codes here and show the execution on machine as well.

RESOURCES

http://stackoverflow.com/questions/15656139/get-partition-and-volume-information?rq=1

http://en.wikipedia.org/wiki/File_system

http://bits.netbeans.org/dev/javadoc/org-openide-filesystems/org/openide/filesystems/FileSystem.html

http://www.tutorialspoint.com/java/io/file_listroots.htm

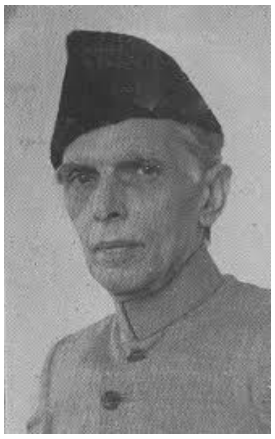
https://en.wikipedia.org/wiki/Directory_%28computing%29

https://en.wikipedia.org/wiki/File:DirectoryListing1.png

http://www.javaprogrammingforums.com/java-programming-tutorials/3-java-program-can-list-all-files-given-directory.html

http://www.cplusplus.com/forum/beginner/10292/

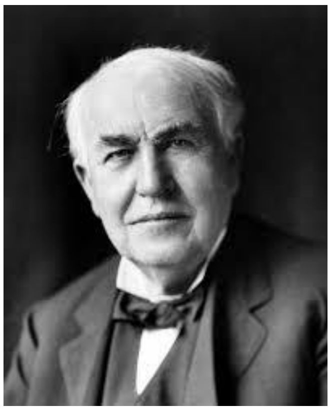
“I insist you to strive. Work, Work and only work for satisfaction with patience, humbleness and serve thy nation”.


Muhammad Ali Jinnah

“A scientist in his laboratory is not a mere technician: he is also a child confronting natural phenomena that impress him as though they were fairy tales”.

Marie Curie

“I have not failed. I've just found 10,000 ways that won't work”.


Thomas Alva Edison