Jaypee Institute of Information Technology, Noida
4th SemB.Tech CSE/IT
Even Sem 2017
OS Lab Exercise Sheet

Teacher In-charge for Lab:

Taj Alam (Lab Coordinator)
Prof. Nitin (Theory Coordinator)

Lab Plan

Lab Test-1→ 20 Marks (proposed Date: 6 February to 11 February)
Lab Test-2→ 20 Marks (proposed Date: 24 April to 29 April)
OS Project Synopsis→ 5 Marks (deadline: 30 January to 4 February)
Mid Project Evaluation → 10 Marks (proposed Date: 20 March to 25 March)
Final Project Evaluation → 15 Marks (proposed Date: 1 May to 6 May)
Lab Evaluation (2 in Number)→ 30 Marks (Random Evaluation during Practice Lab Session before Lab-test 1 and Lab-test 2)

Instructions for the Lab

1. All students have to maintain separate Record File for OS Lab. Hard copy of the exercise should be available during lab evaluations.
2. OS project will have three evaluations (Synopsis, Mid- evaluation and Final project + Report)
3. Deadline for submission should be strictly followed. Any absence during evaluations and tests will get Zero (Exception to the case for Serious Medical Reasons with Application being signed by Faculty in Charge of Lab to whom evaluation or test is given and submitted to Lab Coordinator).
4. Before leaving lab your system should be logged off from FEDORA.

Lab-1 (2-7 January 2017)

Refer Unix_Commands.PDF and execute all commands from it.

- **FILE COMMANDS**
- **DIRECTORY COMMANDS**
- **SYMOLIC LINKS**
- **TERMINAL COMMANDS**
- **HELP COMMANDS**
- **INFORMATION COMMANDS**
- **USEFUL CSHELL SYMBOLS**
- **PERMISSIONS AND FILE STORAGE (UNIX)**
- **PERMISSIONS AND FILE STORAGE (ANDREW)**
- **PROCESSES**
- **PRINTING**
- **ENVIRONMENT**
- **CUSTOMIZING**
- **NETWORKING**
- **X-APPLICATIONS**
- **UNIX FILTERS**

- Refer Enhancing Your UNIX skills.PDF and solve practical exercise from it.
- Read the Coding_Guidelines.PDF, VI Editor.PDF and Makefile.PDF in reference. Additional to the above exercise execute the following

1. Write Linux C Program for command line argument to print the output in following format.

```
Hello, welcome to Linux C programming!
No. of argument to the main Program: 5
Argument No. 0 : ./filename
Argument No. 1 : My
Argument No. 2 : name
Argument No. 3 : is
Argument No. 4 :Rahul
```

2. Write Linux C Program to initialize character, integer, String and float. Each data declaration will occupy an amount of memory in byte. Print the output in following pattern

```
Town name: Guildford population: 66773
County: Survey
Country: Great Britain
Location Latitude: 51.238598 longitude: -0.566257
Char = 1 byte int= 4 bytes float = 4 bytes
Memory used: 780 bytes
```

Hint: Use sizeof() library function

3. Write Linux C Program to print the entire environment variable on the screen.
4. Write Linux C Program to implement merge sort (Find the Sample from Help).

LAB-2 (9 -14 January, 2017)

Process Creation and Management

```
top       # view top consumers of memory and CPU (press 1 to see per-CPU statistics)
who       # Shows who is logged into system
w         # Shows which users are logged into system and what they are doing
ps        # Shows processes running by user
ps -e     # Shows all processes on system; try also '-a' and '-x' arguments
ps aux | grep<user_name> # Shows all processes of one user
ps ax --tree      # Shows the child-parent hierarchy of all processes
ps -o %t -p <pid> # Shows how long a particular process was running.
          # (E.g. 6-04:30:50 means 6 days 4 hours ...)
Ctrl z <enter>    # Suspend (put to sleep) a process
fg          # Resume (wake up) a suspended process and brings it into foreground
bg          # Resume (wake up) a suspended process but keeps it running
            # in the background.
Ctrl c      # Kills the process that is currently running in the foreground
kill <process-ID> # Kills a specific process
kill -9 <process-ID> # NOTICE: "kill -9" is a very violent approach.
           # It does not give the process any time to perform cleanup procedures.
kill -l     # List all of the signals that can be sent to a proccess
kill -s SIGSTOP <process-ID> # Suspend (put to sleep) a specific process
```

1. Write a program that fork once. Find out what happens when fork returns? Also find out what the parent gets as the return value of the fork and what the child gets as the return value of the same call.

2. Write a program that, forks to create a child process. The child exits with an exit status of 10 and the parent waits for the death of the child. Display the PID & PPID of both processes and let the parent display the child's exit status. (Use fork, exit, wait)

3. Create two executables ("main", "pgm1") from two different C source code files: PGM1.c and PGM2.c. These programs should behave as follows:
   a) The program "main" should have 4 different versions to do the following:
      a. Call fork() and Call exec("PGM1") after fork in the child process
      b. Call exec("PGM1") before fork()
      c. Call exec("PGM1") after fork() without checking for return value
      d. Call exec("PGM1") after fork() in the parent process
   b) It does not matter what program PGM2 does, it can have any set of statements

4. Write a program that uses an "exec()" system call to execute a program given to it through command line. For example if the name of your executable is "*myexecutable*" then if you invoke as follows: "*myexecutable mycmd a b*" then "*mycmd*" will be executed with command line parameters "a" & "b" by the program "*myexecutable*" (Note: "*mycmd*" may be any standard unix program or a program created by you)

5. Execute the following code

```
/* processdemo.c */

/* Process demonstration program. Note there are no shared variables */
#include <stdio.h>
#include <stdlib.h>
constclock_t MAXDELAY = 2000000;
int x = 50; /* a global variable */
void delay(clock_t ticks) { /* a "busy" delay */
clock_t start = clock();
do
; while (clock() < start + ticks);
}
voidadjustX(char * legend, inti) {
while (1) /* loop forever */
{ printf("%s: %i\n", legend, x);
x += i;
delay(rand()%MAXDELAY);
}
}
main()
{ int c;
srand(time(NULL));
printf("creating new process:\n");
c = fork();
printf("process %i created\n", c);
if (c==0)
adjustX("child", 1); /* child process */
else
adjustX("parent", -1); /* parent process */
}
```

(a) Examine the source code and try to work out what the output from the program will be.

(b) Compile the program with e.g. gcc processdemo.c -o processdemo

(c) Run the program. Is the output what you expected?

(d) While the program is running, use psxl(in another terminal window) and identify the processes for processdemo . Which process is the parent and which is the child? How can you tell?

(e) Use the *kill* command to stop one of the clones. What happens to the other clone?

(f) Does it make any difference if the parent is killed before the child?

LAB 3 (16- 21 January, 2017)

Process and IPC

1. Write a program where the parent process writes in the pipe and child process reads from the pipe continuously. To send data to the child the parent takes it from the user. When parent process receives the string "exit" from the user, it sends "exit" on the pipe, closes the write end of the pipe and exits. Upon receiving "exit", the client (child) terminates.

HINT:

- Interprocess communication in Unix may be accomplished at a low level using pipes. A Unix pipe is a circular buffer maintained by the operating system. Typically, one or more processes write data into the buffer; another process reads the data from the pipe.
- While Unix handles the internals of pipe communication, programs communicate through pipes at a slightly more conceptual level. More specifically, the use of pipes involves four main steps

1. An array of two integers is declared.
2. The array is initialized by the pipe procedure. With this initialization, the first array element (subscript 0) provides an appropriate file descriptor for reading, while the second array element (subscript 1) provides an appropriate file descriptor for writing.
3. After the fork() operation, both the reading and writing ends of the pipe are available to both processes. Traditionally, however, pipes are available for one-way communication only. Thus, the reading end of the pipe should be closed off in the writing process, and the writing end of the pipe should be closed off in the reading process.
4. Data are moved byte-by-byte through a pipe, using read and write system calls.
    - The write call requires three parameters, the output file number, a base address (e.g., a character string or an array of characters), and the number of characters to be written.
    - The read call requires three parameters, the input file number, a base address (e.g., a character string), and a number of bytes. Reading retrieves the next number of byes -up to the size of the buffer.

2. Solve the arithmetic of ADD, SUB, MUL and DIV by forking child process for each. Make sure your parent is waiting for all child process to complete their task.

3. Write a program which goes to sleep and is woken up by its child.

HINT: There are 2 ways to solve this problem.

- Wait() – this has some limitations, have a look at them.
  - Ref: http://linux.die.net/man/3/wait
- Using signals –pause()
  - Ref: http://linux.die.net/man/7/signal
  - Ref: http://linux.die.net/man/2/pause
  - Ref: http://www.thegeekstuff.com/2012/03/linux-signals-fundamentals/
  - Ref: http://www.thegeekstuff.com/2011/02/send-signal-to-process/

4. Spawn a child process running a new program. PROGRAM is the name of the program to run; the path will be searched for this program. ARG_LIST is a NULL-terminated list of character strings to be passed as the program's argument list. Return the process ID of the spawned process.

HINT :The exec functions replace the program running in a process with another program. When a program calls an exec function, that process immediately ceases executing that program and begins executing a new program from the beginning, assuming that the exec call doesn't encounter an error.Functions that contain the letter *p* in their names (execvp and execlp) accept aprogram name and search for a program by that name in the current executionpath;Functions that contain the letter v in their names (execv, execvp, and execve)accept the argument list for the new program as a NULL-terminated array ofpointers to strings.

5. Write a program which acts like a timer. It asks the user for a specific time, allows the user to execute other programs while it waits and pops up with an alert on finishing the time.

HINT: Look at how the sleep() and wait() commands work. (can also be done using other IPC techniques e.g. signals)
Ref: http://linux.die.net/man/3/sleep

LAB-4 (23-28 January, 2017)

Process Scheduling Algorithms

1. Write C program in Linux operating systems for implementing the first come first serve scheduling algorithm.

   HINT: Read the no. of process from user, Read burst time for all the processes; apply FCFS Scheduling, print PID, Burst time and GANTT Chart
2. Write a C program for implementing the priority scheduling using Linux operating system.

   HINT: Read no. of process from user, Read burst time and priority for all processes. Apply Priority Scheduling algorithm, Print PID, Burst Time, Priority of process and Gantt Chart.
3. Write a Linux C Program for the Implementation of Shortest Job First Scheduling Algorithm.

HINT: Read the number of process from user, Read the burst time for all process, Print: Process, Waiting time, Turn-around time, also print the average waiting time and the average turnaround time in sec.

4. Write a Linux C Program for the Implementation of Round Robin Scheduling Algorithm .

   HINT: Create few processes. Read the process name and apply burst time of 20 ns for all the process, apply Round Robin Scheduling, print: Process name, Remaining Time, Total consumed Time.

5. For the above scheduling algorithm, write menu driven Linux c program and give average waiting time and the average turnaround time in sec.

LAB-5 and Project Synopsis Submission (5 Marks) (30 January-4 February, 2017)

Submit the Project Synopsis hard copy with following details:

1. Title of project
2. Problem statement
3. Project Description

Threads Creation

1. Write a program where 5 threads are created and each thread prints the thread Id and message "thread x executing".

2. Write a program where 2 threads are created and each thread prints information like (Name, Hall No., employee ID , branch).

3. Write a program where 2 threads communicate using a single global variable "balance" and initialized to 1000..Thread 1 deposits amount = 50 for for 50 times and prints the balance amount and thread 2 withdrawals amount=20 for 20 times and prints the final balance .Execution of thread 1 and thread 2 should not interleave.

4. Write a program where 2 threads operate on a global variable "account" initialized to 1000. There is a deposit function which deposits a given amount in this "account": int deposit(int amount)

   There is a withdrawal function which withdraws a given amount from the "account":
   int withdrawal(int amount)

   However there is a condition: withdrawal function should block the calling thread when the amount in the "account" is less than 1000, i.e. you can't withdraw if the "account" value is less than 1000. Threads calling the deposit function should indicate to the withdrawing threads when the amount is greater than 1000.

5. Write a thread program which demonstrates how to "wait" for thread completions by using the Pthread join routine. Since some implementations of Pthreads may not create threads in a joinable state, therefore explicitly created attribute in a joinable state so that they can be joined later. Created thread should perform the calculation of sum =sum + sin(i) + Tan(i), where i =0 to 10000. Print the out in the following manner

Main: creating thread 0
Main: creating thread 1
Thread 0 starting...
Main: creating thread 2
Thread 1 starting...
Main: creating thread 3
Thread 2 starting...
Thread 3 starting...
Thread 1 done. Result = -3.153838e+06
Thread 0 done. Result = -3.153838e+06
Main: completed join with thread 0 having a status of 0
Main: completed join with thread 1 having a status of 1
Thread 3 done. Result = -3.153838e+06
Thread 2 done. Result = -3.153838e+06
Main: completed join with thread 2 having a status of 2
Main: completed join with thread 3 having a status of 3
Main: program completed. Exiting.

6. Write a thread program which demonstrates Pthreads condition variables. The main thread creates three threads. Two of those threads increment a "count" variable, while the third thread watches the value of "count". When "count" reaches a predefined limit, the waiting thread is signaled by one of the incrementing threads. The waiting thread "awakens" and then modifies count. The program continues until the incrementing threads reach TCOUNT. The main program prints the final value of count.

Lab Test 1 (6-11 February, 2017, 20 Marks)

Course Lab1 to Lab 5

LAB-6 (20-25 February, 2017)

Synchronization

1. Given two character strings s1 and s2. Using C and pthread to write a parallel program to find out the number of substrings, in string s1, that are exactly the same as string s2. The strings are ended with '\0'. For example, suppose number_substring(s1, s2) implements the function, then number_substring ("abcdab", "ab") = 2, number_substring("aaa", "a")= 3, number_substring("abac", "bc") = 0. Suppose the size of s1 and s2 are n1 and n2, respectively, and p threads are used, we assume that n1 mod p = 0, and n2 < n1/p. Strings s1 and s2 are stored in a file named "strings.txt". String s1 is evenly partitioned

among p threads to concurrently search for matching with string s2. After a thread finishes its work and obtains the number of local matching, this local number is added into a global variable showing the total number of matched substrings in string s1. Finally this total number is printed out. The format of the strings.txt is like this (the first string is s1 and the second one is s2):
s1: Hello we are doing pthread testing with string.
s2:in

HINT: divide the s1 into two half and create two threads to search substring in this two half.

2. Demonstration of Race Condition in producer and consumer problem using thread implementation.

3. Write a program to implement producer consumer problem (Using POSIX semaphores)
Description: The producer-consumer problem (Also called the bounded-buffer problem.) illustrates the need for synchronization in systems where many processes share a resource. In the problem, two processes share a fixed-size buffer. One process (producer) produces information and puts it in the buffer, while the other process (consumer) consumes information from the buffer. These processes do not take turns accessing the buffer, they both work concurrently. Herein lies the problem. What happens if the producer tries to put an item into a full buffer? What happens if the consumer tries to take an item from an empty buffer? In order to synchronize these processes, we will block the producer when the buffer is full, and we will block the consumer when the buffer is empty. So the two processes, Producer and Consumer, should work as follows:
Procedure for doing the experiment:
   1. Declare variable for producer & consumer as pthread-t-tid produce tid consume.
   2 . Declare a structure to add items, semaphore variable set as struct.
   3 . Read number of items to be produced and consumed.
   4. Declare and define semaphore function for creation and destroy.
   5 . Define producer function.
   6 . Define consumer function.
   7 . Call producer and consumer.
   8. Stop the execution.

4. Write a program to implement producer consumer problem (Using MUTEX semaphores).

5. Demonstrate dead lock in dining philosophers' problem.

LAB-7 (27 February – 4 March, 2017)

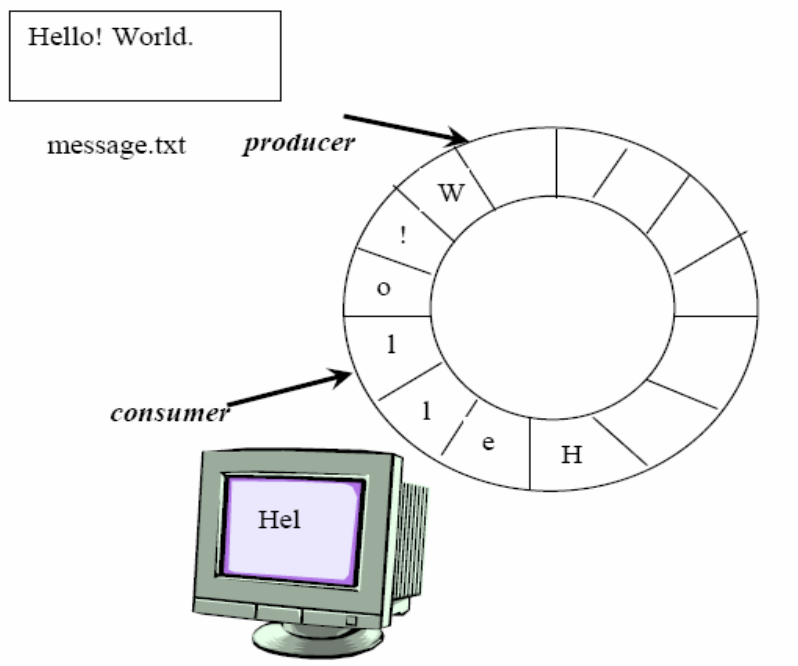
Synchronization

1. To avoid deadlock in dining philosophers' problem use a possible solution as the odd numbered philosophers grab the right and then the left. Implement this solution using pthread mutual exclusion lock.

2. Implementation of Boss/Worker Model: Here the idea is to have a single boss thread that creates work and several worker threads that process the work. Typically the boss thread creates a certain number of workers right away -even before any work has arrived. The worker threads form a thread pool and are all programmed to block immediately. When the boss thread generates some work, it arranges to have one worker thread unblock to handle it. When all workers be busy the boss thread might do one of the following by taking request from the user
    1. Queue up the work to be handled later as soon as a worker is free.
    2. Create more worker threads.
    3. Block until a worker is free to take the new work.

   If no work has arrived recently and there are an excessive number of worker threads in the thread pool, the boss thread might terminate a few of them to recover resources. In any case, since creating and terminate threads is relatively expensive (compared to, say, blocking on a mutex) it is generally better to avoid creating a thread for each unit of work produced.

3. Using condition variables to implement a producer-consumer algorithm. Define two threads: one producer and one consumer. The producer reads characters one by one from a string stored in a file named "string.txt", then writes sequentially these characters into a circular queue. Meanwhile, the consumer reads sequentially from the queue and prints them in the same order. The diagram illustrates the process. Upon completion of running the program, "Hello! World." is printed on the screen. In the program, use #define to specify the size of the queue. For example, #define QUEUE_SIZE 5. Make sure to test your program with different queue sizes, including 1.



4. Write a program to implement producer consumer problem (Using conditional semaphores and monitors)

5. The barber has one barber chair and a waiting room with a number of chairs in it. When the barber finishes cutting a customer's hair, he dismisses the customer and then goes to the waiting room to see if there are other customers waiting. If there are, he brings one of them back to the chair and cuts his hair. If there are no other customers waiting, he returns to his chair and sleeps in it.

   Each customer, when he arrives, looks to see what the barber is doing. If the barber is sleeping, then the customer wakes him up and sits in the chair. If the barber is cutting hair, then the customer goes to the waiting room. If there is a free chair in the waiting room, the customer sits in it and waits his turn. If there is no free chair, then the customer leaves. Based on a naïve analysis, the above description should ensure that the shop functions correctly, with the barber cutting the hair of anyone who arrives until there are no more customers, and then sleeping until the next customer arrives.

   Solve the above problem using conditional semaphore.

LAB-8 (6 March to 11 March, 2017)

Deadlocks

1. Consider a system with: five processes, P0 → P4, three resource types, A, B, and C. Type A has 10 instances, B has 5 instances, C has 7 instances. At time T0 the following snapshot of the system is taken.

	Allocation	*Max*	*Available*
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

   Write a Linux C program to check weather system is in safe state or not. Also demonstrate the unsafe sequence by modifying any resource allocation.

2. Write a Linux C program to demonstrate that deadlock and starvation are opposite problems. Solution to deadlock problem causes more starvation and solution to the starvation problem can cause deadlock.

Mid Project Evaluation (20-25 March, 2017, 10 Marks)

Submit the Project Report till Date hard copy with following details:

1. Title of project
2. Problem statement
3. Project Description
4. Feasibility Study
5. DFD

LAB-9 (10-15 April, 2017)

File Management

1. Write a program to open 2 files (the names of the files should be taken as command line arguments) one in read only mode and the other in write only mode and then read all the characters from the first file and write them into the second file. Also check whether the opened file exists and if not then catch the exception by writing the code for the error also. What is it doing, which system utility does it resemble?

2. Write a program to open 2 files (the names of the files should be taken as command line arguments) one in read only mode and the other in write only mode and then read the characters from the first file and write them into the second file. However do not write all the characters in the second file; write the first one and then every n^{th} character ('*n*' can be any positive number here) from the the first file. Also check whether the opened file exists and if not then catch the exception by writing the code for the error also.

3. Write a program to open 2 files (the names of the files should be taken as command line arguments) one in read only mode and the other in write only mode and then read the contents of the first file and write them into the second file, word by word. However the words should be written in the reverse order, the first word of the first file should be the last word of second file and the last word of the first file should be the first word of the second file.

4. (a)Write a program to open a file in RDWR mode and read the characters from the file and append them in the same file.
   (b) For the above program in 2(a), read a string from the user and append the string into the opened file.

5. (a)Write a program to create 2 files using open system call and then write the string "open system call" in one file and "write system call" in the second file.
   (b) open 2 files, take the input from the user for the data and then using lseek() write the user data at the specified location at the beginning , at the end or at the location specified by the offset.

LAB-10 (17-22 April, 2017)

Disk Scheduling and Memory Management

1. Suppose that a disk drive has 5000 cylinders, numbered 0 to 4999. The drive is currently serving a request at cylinder 143. The queue of pending requests, in FIFO order, is

   86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130

   Starting from the current head position, write an Linux C program to calculate total distance for each of the following disk-scheduling algorithms.

(a) FCFS

(b) SSTF

(c) SCAN

(d) LOOK

(e) C-SCAN

2. Consider the following page reference string:

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6

Implement following page replacement algorithms, assuming one, two, three, four, five, size, or seven frames? Assume that all frames are initially empty.

i. LRU replacement

ii. FIFO replacement

iii. Optimal replacement

LAB TEST 2 (24-29 April, 2017, 20 Marks)

Course: Lab 6 to Lab 10

FINAL PROJECT VIVA (1-6 May, 2017, 15 Marks)

Submit the Complete Project Report hard copy with following details:

1. Title of project
2. Problem statement
3. Project Description
4. Feasibility Study
5. DFD
6. Implementation.
7. Result
8. Conclusion
9. References