**Operating System Lab (CSE 3242)**

1. Create child and parent process.
2. Effect of wait in child and parent process relationship.
3. Use exclp() to execute a separate exe file as child process.
4. Write a C program to create a main process named ‘parent\_process’ having 3 child processes without any grandchildren processes. Trace parent and child processes in the process tree. Show that child processes are doing addition, subtraction and multiplication on two variables initialized in the parent\_process
5. Write a program to create an orphan process.
6. Write a program to create a zombie process.
7. Write a C program to create a main process named ‘parent\_process’ having 3 child processes without any grandchildren processes. Child Processes’ names are child\_1, child\_2, child\_3. Trace the position in the process tree.
8. Write a C program to create a main process named ‘parent\_process’ having ‘n’ child processes without any grandchildren processes. Child Processes’ names are child\_1, child\_2, child\_3,......., child\_n. Trace the position in the process tree. Number of child processes (n) and name of child processes will be given in the CLI of Linux based systems.

Example:

$ ./parent\_process 3 child\_1 child\_2 child\_3

Hint: fork, exec, fopen, system

1. Write a C program for creating a multi-threaded process and check:
2. If one thread in the process calls fork() , does the new process duplicate all threads, or is the new process single-threaded?
3. If a thread invokes the exec() system call, does it replace the entire code of the process?
4. If exec() is called immediately after forking, will all threads be duplicated?
5. Write a C program to show how two threads can communicate by the help of ‘signal’.
6. Write a C program to show how data inconsistency arises in a multi-threaded process.
7. Write a multithreaded program that calculates various statistical values for a list of numbers. This program will be passed a series of numbers on the command line and will then create three separate worker threads. One thread will determine the average of the numbers, the second will determine the maximum value, and the third will determine the minimum value. For example, suppose your program is passed the integers

90 81 78 95 79 72 85

The program will report

A. The average value is 82

B. The minimum value is 72

C. The maximum value is 95

The variables representing the average, minimum, and maximum values will be stored globally. The worker threads will set these values, and the parent thread will output the values once the workers have exited.

1. The Fibonacci sequence is the series of numbers 0, 1, 1, 2, 3, 5, 8, ....

Formally, it can be expressed as:

fib0 = 0

fib1 = 1

fibn = fibn−1 + fibn−2

Write a multithreaded program that generates the Fibonacci sequence. This program should work as follows: On the command line, the user will enter the number of Fibonacci numbers that the program is to generate. The program will then create a separate thread that will generate the Fibonacci numbers, placing the sequence in data that can be shared by the threads (an array is probably the most convenient data structure). When the thread finishes execution, the parent thread will output the sequence generated by the child thread. Because the parent thread cannot begin outputting the Fibonacci sequence until the child thread finishes, the parent thread will have to wait for the child thread to finish.

1. Implement a server-client model to provide services to client processes running in machines different from the server processes. Explain what you experience when you:

* Server process being a single threaded process tries to provide services to multiple client processes.
* Server process being a multi-threaded process tries to provide services to multiple client processes.
* Server process being a single threaded process tries to provide services to multiple client processes with multiple child processes.

1. Write a multi-threaded program to solve a unique problem which is figured out by you and solved by you.
2. Check how multi-threading helps us to execute multiple processes at the same time. (a. Check global and local variable effect c. Check thread race problem)
3. Write a program to learn read() from keyboard file and write() into monitor file
4. Write a program to read from a normal file/conventional file and display to monitor using open() read() write() system calls.
5. Write a program to write a normal file/conventional file using open() read() write() system calls.
6. Write a program to create a named pipe using mkfifo() system call.
7. Write a program to write in a named pipe what is read from the keyboard using open() read() write() system calls.
8. Write a program to read from a named pipe and display to monitor using open() read() write() system calls.
9. Write a chat-room using a named pipe. (a. Start from one end and one by one message. b. Start from both ends and message simultaneously)
10. Investigate what would happened if you try to open a named pipe for reading multiple times.
11. Implement a unnamed pipe to send specific message (Say "Don't wait for me") from a child process to parent process to inform not to wait for the child process.
12. Investigate if two separate process try to read a named pipe first and then write to the pipe.
13. Investigate when race condition occurs in a multi-threaded program.
14. Find a solution to avoid race condition in a multi-threaded program.
15. Check how many core you have in your system. Create 4 more process than your core and use a for loop to understand how os schedule the process in different core. Each core should have three thread. Use get\_cpu() to read the assigned core and set\_cpu() to set the core for each thread.
16. If we lock the thread twice, without unlocking it, what would happen? Investigate it.
17. If you have two thread and you use two mutex for each thread, what would happen for each combination? Investigate it. Example:

mutex\_lock(key1)

mutex\_lock(key2)

##-- Critical Section --##

mutex\_lock(key1)

mutex\_lock(key2)

1. Inter-Process Communication (Shared Memory)
2. Signal
3. Stack Smashing
4. Implement FIFO, LRU and Optimal
5. Create server-client model for communication using socket programming in C.
6. Create server-client model for multi-client communication using socket programming in C.
7. Create server-client model for multi-client communication using socket programming and multi-threading in C.