Project 4 – Signaling with Multi-Process Programs

Problem:
This project requires a program development and an analysis of performance for the solution.

Summary:
The purpose of this project is to have processes responsible for servicing "signals" sent by other peer processes. The processes must execute concurrently, and performance of all process activities is to be monitored and analyzed.

Learning Objectives:
This project expands on our knowledge and experience of creating applications with multiple processes. The project also introduces signals as communication mechanisms and methods of sharing memory among processes. Synchronization in the form of protected access to shared variables is introduced.

The **Program** requires developing a solution using a **parent process**, four signal handling **processes**, three signal generating **processes** and a reporting **process** (total 9 processes).

For the **Program**, the parent process creates the remaining 8 child processes, controls execution time of the child processes and waits for their completion. The program is to use two signals for communication: SIGUSR1 and SIGUSR2. Each of the signal generating processes will execute in a loop, and during each loop repetition will randomly select either SIGUSR1 or SIGUSR2 to send to the processes in its group (its peers). Each repetition through the signal generating loop will have a random time delay ranging between .01 second and .1 second before moving to the next repetition. After generating a signal, the process will increment a shared 'signal sent' counter for that signal type (i.e., a counter globally available to all processes in the program that need access to the counter). That is, each signal type has its own counter for signal sent.

Each of the signal handling processes will process only one type of signal, either SIGUSR1 or SIGUSR2 and ignore servicing the other signal type. (Thus, two of the four processes are assigned to handle each type of signal). Each of the signal handling processes will also execute in a loop, waiting for its signal to arrive. When the signal arrives, the process will increment a shared 'signal received' counter for that signal type (i.e., a counter globally available to all processes in the program that need access to that counter). Each signal type has its own signal received counter. The process will then loop waiting for the next signal.

The reporting process will execute in a loop, also waiting for signals. It will respond to both types of signals. Each time it handles 10 signals, it will report the system time and the count of the number of occurrences of each signal type counter as indicated in each of the shared memory counters (at the time of the report). The reporting process should

also report the average time between receptions of each signal type during the interval in which the 10 signals are received. This will require the reporting process to remember the time of occurrence of each signal type.

Access to the global counters in each program requires the use of a critical section control. **You are to use a lock for the critical section control.** Use the lock to provide each process private access to each global counter.

Results to Report:
A. You should run the program for 30 seconds and examine the performance; that is, the number of signals of each type sent and received. Also examine the average time between signal receptions of each type by the reporting thread. Be sure to investigate and discuss any signal losses.
B. You should run the program for a total of 100,000 signals and look at the total time and counts for the signal types, as well as the average interval between signal receptions of each signal type by the reporting thread. (i.e., report the same information as in part A.)

Be sure to include a **test plan** for the program, and document the results of your testing such that you can clearly demonstrate that the program executes correctly.

You will develop your program using the supplied GitHub repository. You will submit your program code, testing documents and execution results through Canvas. Compress all of the files (do not include executables) into one file for submission. Use clear naming to help identify the project components that are submitted. Include a link to your GitHub repository as a comment to your Canvas submission.

There will be **weekly deliverables** and they are to be **submitted to Canvas** as well.

Week 1:

- Creation of the main process, the 3 signal generator processes, the 4 signal catcher processes and the reporter process. (one source for signal generator, one source for signal catcher).
- Design and implementation of interval clocks
- Create shared memory and attach to all processes

Week 2:

- Create shared counters and individual locks for each counter.
- Proper/safe access to send and receive counts
- Description of testing and documentation (did they include logs in processes, e.g.).

Week 3 [Complete Submission]

- Creation of proper functioning signal catcher routines
- Proper use of signal masks to enable/disable signal receptions

Document Production

  - Reporter receipt and time stamping of signals and intervals for process execution
  - Description and analysis/comparison of results from execution – including multiple executions and comparison of data.

Final executable and Demo