Kohl Johnson

CST-321

12-22-2024

Activity 3

# More Signals in Linux

**Theory of Operation:** The code works by having the parent process start with a message, write it to the shared buffer, send a signal to the child, and then moves into the next character in the message string. Once all characters have been sent to the buffer, the parent then sends the child a null character which indicates that the entire message has been sent (placed) into the buffer. The child process works by first waiting for a “wakeup” signal from the parent process. From there, it reads the character from the lower value in the buffer, prints it to console, and then restarts that process until the null character is received/read.

**Output View Part 1**

A computer screen shot of a computer screen

Description automatically generated

**Output View Part 2**

**A computer screen shot of a computer screen

Description automatically generated**

# More Signals and Mutexes in Linux

**Theory of Operation:** We start off by creating two threads, one being the “counter” thread and the other acts as a monitor. The counter thread will lock the counter (using a mutex), increment said counter, and then sleep for 1 second before unlocking the mutex to give the monitor thread a chance to “miss”. The monitor will attempt to lock the mutex every three seconds to read the value of the counter. If successful, it will print the value to the console, otherwise it will increment a separate counter (called misses) and restart the process.

**Output View**

**A screenshot of a computer

Description automatically generated**

# More Signals and Semaphores in Linux

**Theory of Operation:** The child process starts by simulating a very long and demanding (or hung) process to give the parent process a chance to check for a hung process. From there, it decides whether to kill the child process by calling a timer function. This function counts down from 10 before attempting to lock the semaphore. If successful, it returns “ok”, otherwise it returns 0. If 0 is returned, the parent process kills the active child, then “proves” it is killed by waiting for 5 seconds and checking for output from the child. From there, it attempts to secure the semaphore.

**Output Run A**

A screenshot of a computer

Description automatically generated

**Output Run B**

**A computer screen shot of a computer screen

Description automatically generated**