Project 2 - Signal Handlers and User Level Threads

Introduction to Computer Systems - 14:332:434

April 12, 2017

Problem 1

In this problem, we utilized our Project 1D code as a skeleton for our program. Then, we removed all the code affiliated with pipes/shared memory. Finally, we programmed the rest with our acquired knowledge on signals and signal handlers.

We used various signals/signal handlers to pass information between child processes and their parent process. The following the table shows which signals we have overwritten to accomplish these calculations/communications.

Signal	*Function*
SIGBUS	*MIN1 Calculation*
SIGILL	*MIN2 Calculation*
SIGHUP	*MAX1 Calculation*
SIGSYS	*MAX2 Calculation*
SIGPWR	*SUM1 Calculation*
SIGTTOU	*SUM2 Calculation*
SIGCONT	*Send Value of MAX1 to Original Process*
SIGIO	*Send Value MAX2 to Original Process*
SIGURG	*Send Value of MIN1 to Original Process*
SIGXCPU	*Send Value of MIN2 to Original Process*
SIGPROF	*Send Value of SUM1 to Original Process*
SIGWINCH	*Send Value of SUM2 to Original Process*

Table 1

Part A

In this part, the original process calls on 3 signals corresponding to the first 3 functions, MAX1, MIN1, and SUM1 to calculate the appropriate operations on the first half of the array. These signals are performed in child processes sprouting from the original process. Furthermore, these processes call signals to the next three children processes MAX2, MIN2, and SUM2 to perform operations on the second half of the array. For all these signals, the values are placed in a sigqueue function which is handled by the signals to share their calculated values with the original process.

Part B

This part involved killing a process that was taking to long for the parent. The following signals were overwritten along with the signals in **Table 1** to accomplish this task.

Signal	*Function*
SIGXFSZ	*Sends SIGKILL to slow process*
SIGTRAP	*Send PID of MIN2*
SIGFPE	*Send PID of MAX2*
SIGSEGV	*Send PID of SUM2*

Table 2

Once the parent waits 3 seconds, it receives notices of which processes are finished using integer values to represent booleans. 0 represents not finished while 1 represents finished. When a process is not finished, a number of if statements are used to verify whether other children processes are finished. If there is a finished process, that child sends the PID of the unfinished process to be killed via sigqueue and the signal SIGXFSZ.

Q: What did you observe if more than one process attempts to terminate a given process?
A: According to the messages printed out to the terminal, multiple process will try to terminate it at once. However, one of them has to get there first and the rest will get a "No Such Process" error, which is a pretty trivial error in this situation.

Part C

The terminating signals were blocked in this program to allow the process to run fully without getting interrupted. This involved creating separate handlers which had overwritten the signals as summarized by the **Table 1** and the following list.

- SIGINT (CTRL + C)
- SIGQUIT (CTRL + \)
- SIGTSTP (CTRL + Z)
- SIGTERM (can be implemented by typing kill -s -SIGTERM "*PID NUMBER*") in another terminal window

Q: When can you do the latter and when can you not?
A: The user can attempt terminating the program using these signals during the second and third pause. The second pause corresponds to the children processes sleeping while the third pause corresponds to the parent process sleeping. In the first pause, you cannot perform these actions while the program is asking for the size of the array because this information is needed to execute the rest of the program.

In order to give the user is given an opportunity to terminate the program, we implemented functions to put our processes to sleep. When a user attempts to terminate the process, the signal number, signal name, and PID of the chosen process is printed to the console.