School of Advanced Technology

CST8244 Lab 3: Processes and Signals

Lab Objectives:

This lab will result in your ability to build two simple programs. The first demonstrates the handling of signals sent from another program, and the second creates child processes which each handle signals sent by a third application. The "third" application in this case will be the command interpreter, i.e., command-line.

Part A: Write an Interrupt Signal Handler

Using the program **sigint.c** (posted on Brightspace) as an example to start from, you are to write a small program that is able to catch the interrupt SIGUSR1.

1. Create a new Momentics workspace named: **cst8244_lab3.ws**
2. Create a new QNX Executable Project named: **partA_sighandler**
3. Create your first program (**partA_sighandler.c**)
    a. Create a local git repository; add your project to git and make your initial commit.
    b. Define a global variable **usr1Happened** to indicate whether or not the program has received the signal.
        i. **usr1Happened** COULD be defined as integer or a char, as the only thing you are doing is to "trigger" the value.
        ii. Nevertheless, to avoid uncertainty about interrupting access to a variable, you should use the **sig_atomic_t** data type to ensure that access is always atomic. Reading and writing this data type is guaranteed to happen in a single instruction, so there's no way for a handler to run "in the middle" of an access.
        iii. Additionally, declare the **usr1Happened** variable to be **volatile**.
        iv. Initialize **usr1Happened** to be "false".
    c. Write a signal handler for the interrupt SIGUSR1. The only thing your handler has to do, is to set your **usr1Happened** to 1 to indicate that the signal has happened.
    d. Install your signal using **sigaction( )**, using the signal type SIGUSR1.
    e. Your program is to loop until the signal is received (check the value of **usr1Happened** in your loop).
    f. Print your **pid** number on the console so that you will know it for testing when using the **kill** command from the command prompt.

g. The output should look something like this:

// run program from Momentics IDE
PID = 123645 : Running…
PID = 123645 : Received USR1.
PID = 123645: Exiting.
 	
Program output
		# kill –s SIGUSR1 123645
		
		Kill the child from Neutrino terminal

h. Add a README.txt file to your project. Please follow this template:

Title {give your lab a title}
Status
{Tell me the status of your project. Does your program meet all of the requirements of the lab specification? Does your program run, and more importantly, behave as expected? Does your program terminate unexpectantly due to a run-time error? Any missing requirements? Please…. just don't answer the questions: yes, no, etc. A small paragraph should be sufficient}
Known Issues
{Tell me of any known issues. From my point of view, it's better that you tell me versus I discover.}
Expected Grade
{Tell me your expected grade. I'll reply with one of three states: i) Agree, ii) Disagree – grade higher; Disagree – grade lower (with explanation)}

Part B: Signals and Processes

1. To the same Momentics workspace --- **cst8244_lab3.ws** --- create a new QNX Executable Project named: **partB_sigproc**
2. Create a new C/C++ program (**partB_sigproc.c**) that will **fork( )** child processes.
    a. The parent should
        i. Print its pid.
        ii. Read from the command-line the number of children to fork and store it in an integer variable **numChildren**.
        iii. Fork the required number of children.
        iv. Wait for all the children to finish (by counting the number that have finished), and then print out an appropriate message stating that all the children have finished.
    b. Each child should
        i. Print its pid.
        ii. Loop until it receives the USR1 signal as in the program from Part A. Then the child should:
            1. Print a received signal message.
            2. Print a goodbye message (child exiting) showing child's pid.
            3. Return to parent; the parent calls: **exit (EXIT_SUCCESS);**
    c. The output should look something like this:

//run program from Momentics IDE Enter the number of children: 1 PID = 123640: Parent running… PID = 123645: Child running… PID = 123645: Child received USR1. PID = 123645: Child exiting. PID = 123640: Children finished, parent exiting. **Program output**		# kill –s SIGUSR1 123645 **Kill the child(ren) from Neutrino terminal**

Deliverables:

Before making the zip-file deliverable (next item), please action:

- Format your source code to make it easier for me to read. Momentics IDE will do this for you: Source -> Format
- Verify your projects build cleanly. Please action: a) Project -> Clean… and b) Project -> Build All

Prepare a zip-file archive that contains the following items:

1. A "README.txt" file reporting the status of your lab. Follow this template:

   Title {give your work a title}

   Status
   {Tell me that status of your project. Does your program meet all of the requirements of the specification? Does your program run, more importantly, does your program behave as expected? Does your program terminate unexpectantly due to a run-time error? Any missing requirements? A small paragraph is sufficient.}

   Known Issues
   {Tell me of any known issues that you've encountered.}

   Expected Grade
   {Tell me your expected grade.}

2. Export your Momentics workspace as a zip-archive file.
   Momentics IDE provides a wizard to export your projects:
   File > Export… > General > Archive File > Next > Select All > Click "Browse…" > Save As:
   cst8244_lab3_*yourAlgonquinCollegeUsername*.zip > Save > Finish

3. For partA_sighandler, take a series of screenshots that capture the following scenarios:
   1. Launch *partA_sighandler* from Momentics IDE, and then run the command from the Neutrino terminal: kill -s SIGUSR1 <PID>
      where PID is the process ID of partA_sighandler
   2. Launch partA_sighandler from Momentics IDE, then run the command from the Neutrino terminal: kill -s SIGUSR1 <BogusPID>
      where BogusPID does not exist
   3. With partA_sighandler still running, enter the command: kill -s SIGUSR2 <PID>
      where PID is the process ID of partA_sighandler

4. For partB_sigproc, take a series of screenshots that capture the following scenarios:
   1. Create 3 children; kill each child, one at a time
   2. Create 2 children; kill the parent (no zombies please!)

   Your screenshots are to match mine; see Brightspace for the reference screenshots.

Upload and submit your zip-file to Brightspace before the due date.

Postlab:

Congratulations on completing the lab!

Remember to shut down the Neutrino RTOS (use the *shutdown* command), followed by shut down of VMware, and finally quitting VMware.