**Laboratory 2  
Due: 23 September (demo in lab and submit through Blackboard)**

This laboratory will result in your ability to build 2 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 another application. The “other” application in this case will be the command interpreter, i.e., command line.

## Steps:

Using the program *sigint.c* posted on Blackboard as an example to start from, you are to write a small program that is able to catch the interrupt SIGUSR1. Your program should:

1. Create a new project named “Lab2\_prog1” inside the GIT repository for your laboratory work.
2. Create your first program (prog1.c)
   1. Define a variable *usr1Happened* to indicate whether or not the program has received the signal.
      1. *usr1Happened* COULD be defined as integer or a char, as the only thing you are doing is to “trigger” the value.
      2. 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.
   2. 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.
   3. Install your signal using *sigaction( ),* using the signal type SIGUSR1
   4. Your program is to loop until the signal is received ( check the value of *usr1Happened* in your loop )
   5. Print your *pid* number on the console so that you will know it for testing when using the *kill* command from the command prompt.
   6. The output should look something like this:

|  |  |  |
| --- | --- | --- |
| # ./prog1  PID = 123645 : Running….  PID = 123645 : Received USR1.  PID = 123645: Exiting.  **Program output** |  | # kill –s SIGUSR1 123645  **Kill the child from another terminal** |

1. Create a new project called “Lab2\_prog2” and Modify the program from 2 to create a small C program (prog2.c) that will *fork( )* child processes.
   1. The parent should
      1. Print its pid.
      2. Read from the command line the number of children to fork and store it in an integer variable numChildren.
      3. *Fork the required number of children.*
      4. *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.*
   2. *Each child should*
      1. *Print its pid.*
      2. *Loop until it receives the USR1 signal as in the program from part 2. Then the child should:*
         1. *Print a “goodbye from pid = …” message.*
         2. *Finish and send a signal to its parent.*
   3. The output should look something like this:

|  |  |  |
| --- | --- | --- |
| # ./prog2  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 another terminal** |

1. Save and commit the changes to your repository.
2. Go to the working directory for your GIT repository (the one named with your Algonquin username) and zip it.
   1. If you open the “Window>Open Perspective>Other…>Git Repository Exploring”

You can see where it is physically stored

1. Submit your laboratory for grading by **demonstrating it in the laboratory** and submitting it online using Blackboard under the “Handing in Work” link, with the zip file as an attachment.

## Grading Scheme

Marked out of 30. 15 points for each of the programs.

Each program marked with the following rubric.

|  |  |
| --- | --- |
| **Criteria** | **Mark** |
| Superior capability. Submission meets or exceeds expected standards | 15 |
| Satisfactory capability, acceptable product/result | 10 |
| Marginal capability, substandard product/result | 5 |
| No capability, unacceptable product/result. Work not submitted | 0 |