## Programming Exercise F - POSIX Signals

### Assignment Summary

In this assignment, you shall create and register signal handlers to be used in a signal testing program. The program is divided into two parts: a test driver module and a signal handlers module. The test driver module is already complete. It contains the main function and other function definitions that execute code that intentionally results in the generation of signals such as alarm notification, floating point error, segmentation fault, and program abort. These signals, when not explicitly handled, cause the program to abnormally terminate.

Your assignment shall be to implement the ".c" portion of the signal handlers module. (The ".h" header file portion has already been completed for you.) In this signal handlers module, you shall create and register signal handlers that change the program's default action of abnormally terminating when any of these signals are generated.

Note that the test driver module not only contains the code to generate various signals, but also the functionality to test if your signal handlers are operating correctly.

### Signal Handlers

You shall implement a separate signal handler function for each of the seven signals listed below.

Accompanying each signal name in the list below is the description of what the signal handler shall do. For this assignment, any output message from a signal handler shall be sent to **standard error** only and shall be accomplished by means of a write() function. Each message shall start with the name of the specific signal enclosed in parentheses (i.e., (SIGABRT) ) followed by an applicable error or status message as described below.

* SIGABRT

Print a message stating that an assertion has failed and that the program will now end. Then call the **exit()** function to gracefully terminate the program.

* SIGALRM

Print a message stating that the allowable time has expired and that the program will now end. Then call the **exit()** function to gracefully terminate the program.

* SIGCHLD

Call the wait() function. When it returns a child process ID, print a message stating that the program has received a wait call from child process #X (where X is the child process ID). Then use the **return** statement to return control back to regular program processing.

* SIGFPE

Print a message stating that a critical arithmetic error has occurred and that the program will now end. Then call the **exit()** function to gracefully terminate the program.

* SIGINT

Print a message stating that the program has detected the entry of the <Control C> key. Then use the **return** statement to return control back to regular program processing.

* SIGSEGV

Print a message stating that a critical memory access error has occurred and that the program will now end. Then call the **exit()** function to gracefully terminate the program.

* SIGTERM

Print a message stating that the program has received a request to terminate and will now end. Then call the **exit()** function to gracefully terminate the program.

### Assignment Directions

1. Download the following three source code files from the Assignments page of the course website:
   * **program-driver.c** - This source code file contains a completed test driver module that you will use to test the operation of the code in your signal handlers module. In this file, only make the small changes described later in these directions.
   * **signal-handlers.h** - This source code header file contains the function prototype for the **registerSignalHandlers** function that the test driver calls to register the various signal handlers. Make no changes in this file.
   * **Student-module.c** - This source code file contains some functions that are already written and serve as examples of how you need to write the rest of the code.
2. Initially compile and link the program-driver.c program all by itself. In its downloaded condition it doesn't need any other source code files because of two lines that are commented out in the program: one is a #include for the signal-handlers.h file and the other is a call to the registerSignalHandlers function that registers the signal handlers. Read through the source code for this test driver program so that you understand its operations. Then run it several times to understand how to generate the various signals and to see what action is taken by the default signal handlers. Afterwards, uncomment the two lines described above so that the test driver module can be properly compiled and linked successfully with your signal handlers module.
3. Rename the student-module.c source code file to "signal-handlers-module.c". This file shall constitute your implementation part of the signal handlers module. (The signal-handlers.h file is the other part.) In this file place the following items in this order (Some of this code is already written):
   * A #include directive for the signal-handlers.h file
   * A function prototype for each of the seven signal handler functions that you will define later in this file. Give each function a name that reflects the specific signal that it handles.
   * Your implementation of the **registerSignalHandlers** function, whose prototype is supplied in the signal-handlers.h file. This function shall contain the code to register each of the **seven** signal handlers with its respective signal. This approach is used as a means to modularize the program into a test driver module and a signal handlers module (with each contained in its own file). Note that this register function is the **only** function from this file that is explicitly called by source code in the test driver module. Refer to the Robbins slides on signals for the sequence of source code statements required to register a signal handler.
   * A function definition for each of the **seven** signal handler functions whose prototypes you listed earlier in this file
4. Compile and link the test driver file and your signal-handlers-module.c into a single executable file. Use this program to test the operation of each of your signal handlers. Note that some of your signal handlers can only be tested by sending a signal to the process via the **kill** command from within another command shell.
5. Only submit your signal-handlers-module.c file to Blackboard. Do **not** submit the other two source code files because the instructor already has them.

### Implementation Constraints

* Follow the same coding standards given in previous assignments