CSCI 1730 Breakout Lab 11

Signal Handling and Error Handling

Learning Outcomes

- Design and implement programs that use system calls, signal handling, and run concurrently.
- Use the command line in a Unix environment to run processes and to pipe or redirect input or output.
- Get more experience with error handling and function pointers in C.

Problem / Exercise

WARNING: DO NOT use the fork system call for this lab!

The purpose of this lab is to get experience using the C programming language to handle Unix signals and errors. You will write a C program that sums up a list of double values that a user inputs or is redirected from a file, and the program will prompt or sleep until it receives a specific signal from kill.

Requirements

1. Read through this entire lab and chapter 10 in the Advanced Programming in the Unix Environment, Third Edition (Stevens and Rago) textbook.
2. Before you begin working on this lab, you will need to understand process ids, the kill command, and how to put your process to sleep (all concepts have been discussed in lecture classes). It is your responsibility to monitor your processes to ensure that they are not abusing resources on odin or the vcf nodes. Processes should be not be allowed to run for more than 90 seconds. If your account gets suspended or banned for abusing resources on our operating systems, then your grade might be penalized.
3. Your program should setup three signal handlers for kill -8 *pid*, kill -10 *pid*, and kill -12 *pid*, where *pid* is its current process id when it is executing.
4. Your program must output its pid as shown in the examples. The pid will vary each time the program is run. After displaying the pid, your program should print the statements as shown in the examples.
5. Your program should read in double inputs one line at a time from a user or redirected from a file. If input is from a user, then the program should continually prompt the user for a line of input until an aforementioned signal from kill is received. If input is redirected from a file, then your program should read in inputs one line at at time, and at the end of the input, it should go to sleep for 60 seconds. If the program encounters an aforementioned signal from kill while it is sleeping, then it should end as shown in the examples. If no aforementioned signal from kill is sent to your program after 60 seconds of sleep, then the program should also end as shown in an example.
6. Your program must handle all input errors. Each time it encounters an input error, it should print out the error message (to standard output) as shown in the examples, and the program should keep running normally afterwards.
7. Once a signal is handled from kill -8 *pid*, kill -10 *pid*, or kill -12 *pid*, then the program should output the last two messages (to standard output) as shown in the examples.
8. Your program CANNOT use any of the following: `break`, `goto`, `fork`, `exec` family of functions/calls, `system` function. The use of any of these for this assignment will result in a grade of zero on this assignment.
9. Your program must handle user input, errors, input/output redirected to/from standard input/output, and the aforementioned kill signals exactly as shown in the examples except for the pid, which will vary for each run.
10. Your program's I/O must adhere to the instructions in the Examples section.

Examples

After you finish implementing the `lab11.c`, the I/O from running `lab11.out` should be exactly what is shown below except that the pid will vary for each run. For these examples to work, you will need to send the aforementioned kill signals to your process while it is running on odin or a vcf node, which means you will probably need to use multiple ssh sessions (one session to run the example and a different session to kill the process while it is running). The `.txt` input files can be downloaded from eLC.

```
./lab11.out
Program started with pid = 34419.
Enter a list of doubles to sum,
and to end the program,
run one of the following Unix commands:
  kill -8 34419
  kill -10 34419
  kill -12 34419
5.5
2.5
1
q
Error:  please input a numeric value.
zoey
Error:  please input a numeric value.
zoey is awesome
Error:  please input a numeric value.
The sum is 9.000000.
Program ended by handling the signal from kill -8 34419.
```

```
./lab11.out < input1.txt
Program started with pid = 34726.
Enter a list of doubles to sum,
and to end the program,
run one of the following Unix commands:
  kill -8 34726
  kill -10 34726
  kill -12 34726
Error:  please input a numeric value.
The sum is 20.000000.
Program ended by handling the signal from kill -10 34726.
```

```
./lab11.out
Program started with pid = 37111.
Enter a list of doubles to sum,
and to end the program,
run one of the following Unix commands:
  kill -8 37111
  kill -10 37111
  kill -12 37111
4.5
10.25
1
2
3
The sum is 20.750000.
Program ended by handling the signal from kill -12 37111.
```

```
./lab11.out
Program started with pid = 37171.
Enter a list of doubles to sum,
and to end the program,
run one of the following Unix commands:
  kill -8 37171
  kill -10 37171
  kill -12 37171
Tyrion Lannister
Error:  please input a numeric value.
5.75
Zoey Stark
Error:  please input a numeric value.
1.75
abc
Error:  please input a numeric value.
def
Error:  please input a numeric value.
An entire line counts as one error
Error:  please input a numeric value.
2
quit
Error:  please input a numeric value.
The sum is 9.500000.
Program ended by handling the signal from kill -10 37171.

./lab11.out < input1.txt
Program started with pid = 3940.
Enter a list of doubles to sum,
and to end the program,
run one of the following Unix commands:
  kill -8 3940
  kill -10 3940
  kill -12 3940
Error:  please input a numeric value.
The sum is 20.000000.
Program ended after sleeping for 60 seconds.
```

Figure out why the example below does not show any output until the kill -12 command is successfully executed (or the 60 second sleep ends). Decide what Unix commands you need to use to find the pid for this example to end it with a signal from kill -12?

```
./lab11.out < input2.txt > output2.txt; cat output2.txt; cat output2.txt | wc -l;
Program started with pid = 12362.
Enter a list of doubles to sum,
and to end the program,
run one of the following Unix commands:
  kill -8 12362
  kill -10 12362
  kill -12 12362
Error:  please input a numeric value.
The sum is 30.500000.
Program ended by handling the signal from kill -12 12362.
10
```

The example below contains input and output from lab11.out and from other parts of the OS. Figure out what parts belong to lab11.out and what parts do not belong to lab11.out. Before running this example, figure out what the inputs and outputs from various sources are doing.

```
./lab11.out < input3.txt
Program started with pid = 41863.
```

```
Enter a list of doubles to sum,
and to end the program,
run one of the following Unix commands:
  kill -8 41863
  kill -10 41863
  kill -12 41863
Error:  please input a numeric value.
Error:  please input a numeric value.
^Z
[1]+  Stopped                 ./lab11.out < input3.txt
csci-1730@csci-odin$ kill -10 41863; fg;
./lab11.out < input3.txt
The sum is 202.650000.
Program ended by handling the signal from kill -10 41863.
```

1 C Program

1.1 General Requirements

1. Place the files for this assignment in a directory called `lab11`.
2. Your program's source file(s) must compile on odin or a vcf node using the provided Makefile and using the required compiler for this course (`gcc version 11.2.0`).
3. Place your C source code for this assignment in a file named `lab11.c`. See the Examples section to see what your C program should output. Compile your C program using the provided Makefile for this assignment. Do NOT modify any commands, targets, or dependencies that are associated with compiling source code in the provided Makefile for this assignment. If there are any warnings or errors from the compiler, then you should fix them before you submit this assignment to us for grading.
4. Your program cannot contain a memory leak when executed. You are responsible for using valgrind to test your program to ensure it does not contain a memory leak, and valgrind should be run for every test case for your program. A program that contains one or more memory leaks (definitely, indirectly, or possibly lost) identified by valgrind will lose points.
5. The only include directives that you are authorized to use in your source code for this assignment are listed below.

   ```
   #include <signal.h>
   #include <stdio.h>
   #include <string.h>
   #include <stdlib.h>
   #include <stdbool.h>
   #include <unistd.h>
   ```

   You CANNOT use any other include directives anywhere in your source code.
6. All written and verbal instructions stated by the teaching staff (lecture instructor, lab instructor(s), etc.) must be followed for this assignment. Failure to follow instructions may result in losing points.

1.2 Coding Style Requirements

1. All functions must be commented. Comments must include a brief description of what the function does, its input(s), its output, and any assumptions associated with calling it. If your function has a prototype and an implementation separated into different parts of your source code, then you only need to put the comments for that function above its prototype (there is no need to comment both the prototype and implementation; commenting the prototype is sufficient).
2. All structs, unions, and enums must be commented.
3. All global variables and static variables must be commented.

4. All identifiers must be named well to denote their functionality. Badly named identifiers are not allowed. For example, identifiers like a, aaa, b, bbb, bbbb are bad names for identifiers.

5. Every line of source code must be indented properly and consistently.

1.3 README.txt File Requirements

Make sure to include a `README.txt` file (use a .txt file extension) that includes the following information presented in a reasonably formatted way:

- Your First and Last Name (as they appear on eLC) and your 810/811#
- Instructions on how to compile and run your program. Since we are using a Makefile for this assignment, then these instructions should pretty simple based on the provided Makefile.

2 Submission

Submit your files before the due date and due time stated on eLC. Submit your files on eLC under the Assignment named Lab 11. Submit only the following files.

1. All source files required for this lab: lab11.c
2. A README.txt file filled out correctly

Do not submit any compiled code. Also, do not submit any zipped files or directories. Do not submit a Makefile. We will use our own copy of the provided Makefile to compile your program. We only need the files mentioned above with the file extensions aforementioned.

3 Breakout Lab Attendance: 3 points

Breakout lab attendance is required for this assignment, and it is worth three points. Students must physically attend the entire period of the breakout lab section that they registered for during the week of this assignment to earn these three points for attendance. If you complete the assignment before the end of your breakout lab period during the week of this assignment, then you are still required to attend the entire breakout lab period to earn attendance points, and you may use this time for independent study for the next exam in this course.

4 Grading: 7 points

If your program does not compile on odin or a vcf node using the provided Makefile and using the required compiler for this course (`gcc version 11.2.0`), then you'll receive a grade of zero on this part of the assignment. Otherwise, your program will be graded using the criteria below.

Program runs correctly with various test cases on odin	7 points
README.txt file missing or incorrect in some way	-1 point
Not submitting to the correct Assignment on eLC	-1 point
Late penalty for submitting 0 hours to 24 hours late	-2 points
One or more compiler warnings	-2 points
Not adhering to one or more coding style requirements	-2 points
Valgrind identifies a memory leak (definitely, indirectly, or possibly lost)	-2 points
Penalty for using a `break`, `goto`, `fork`, `exec*`, or `system` function	-7 points
Not submitting this assignment before its late period expires	-7 points
Penalty for not following instructions (invalid I/O, etc.)	Penalty decided by grader

You must test, test, and retest your code to make sure it compiles and runs correctly on odin or a vcf node. This means that you need to create many examples on your own (that are different than the aforementioned examples) to ensure you have a correctly working program. Your program's I/O must adhere to the instructions given in the Examples section. You may NOT assume input will be valid for this assignment.