CS355 Hw 1

Due by the end of day **Sunday, 2/4**. Written problems due by **Monday 2/5**, in class.

Hw1 is an exercise to get you to realize how much you can do with just the bare basics. Remember that less is more. For example, you don't need to know about the size of the files in 2 and you don't need malloc, not for this assignment.

Programming Assignment:

1. Linux signals

   - In lab 0, we saw how many OS kernel parameters are stored as pseudo files in the `/proc` directory. Begin by reading the manual pages for '`proc`' if necessary.
   - Write a program that will show the rate of interrupts. Each second it should print the number of interrupts (1) in the previous second, and (2) the total number of interrupts since boot time.
   - Write the program two ways:
     (a) Control the main looping with `alarm, signal` and `pause`. Name this version `hw1a`.
     (b) An alternative is to set a timer `ITIMER_REAL` to produce a signal every second, which is more accurate than using `alarm`. The timer can be set using `setitimer`. Read manual pages for more on how to use this (i.e. type `man setitimer` on command line), go over lab 1 notes and exercises as well. Name this version `hw1b`.
   - It must be possible to terminate the process by using `ctrl-c`.
   - Add a command line argument to allow custom interval, i.e. instead of every second I may specify every two-second printout by typing `'hw1b -s 2'`. You only need to implement this feature for hw1b. Remember to include proper error checking.
   - **Note:** Do not have the signal handler get the information and print it out. Instead, let the process itself get the information and print it out each time it wakes up.

2. Linux system calls

   - Using only system calls (that is, functions documented in section 2 of the Unix/Linux manual pages such as `fork()`, `execl()`, `execve()`, `open()`, `read()`, `write()`, `wait()`, etc., write a C program, `microcat`, that concatenates and prints files (taken as arguments) to standard output. If no arguments are given, it takes input from `stdin`. This is basically the Unix `cat` with no flags. Remember that `stdin` is file descriptor 0, and `stdout` is file descriptor 1.
   - Modify `microcat` so that if a signal (any signal) arrives it prints the message "Help! I think I've been shot!!!" before terminating.

- Important: You may use only UNIX system calls (those documented in section 2 of the UNIX man pages, plus any of `waitpid()`, `sleep()`, `signal()`, `kill()`, `exit()` and `select()` (which are not strictly system calls on some systems)) to implement this program. You may NOT use the standard library functions (higher-level functions documented in section 3 of the manual). You may use `string.h` functions, if needed, for commandline arguments parsing.
- Extra Credit:
  (a) If any file name is a single dash `-`, `microcat` replaces it by reading from `stdin`. Play with the Unix `cat` to get a feeling of how it works.
  (b) Add support for redirection to file, i.e. `>`. Again, play with the Unix `cat` to get a feeling of how it should work. Note that you will need to quote the redirection symbol in your input to prevent the shell from executing its own redirection (and thus not passing the input into your `main`). That is, you need to test redirection with command given as: `microcat f1.txt '>' f2.txt`.

Supplementary Questions: Do the following questions from Tanenbaum.

- pages 81-82, problem numbers 15, 17, 18, 20, 23 and 26.

Deliverables:

1. Electronic submission of the programming assignment. See below for submission instructions.
2. Hardcopy: Answers to the supplementary questions.

Electronic Submission Details: Your submission will handed in using the '`submit`' script provided on our Linux system. Note that your program will be graded based on how it runs on the department's Linux system, not how it runs on your PC. The submission must include the following items:

1. README: There must be a file called `README`, which contains any helpful information on how to compile and run your program. This is also the place you should make every effort to make granting credits for your work easy. Remember, if we can't figure out how it works, you won't receive credit for it!

This includes:

Your name:

How to compile: You should provide a `Makefile` and I should just have to type '`make`' to compile your program. You may leave this out/empty if that is indeed so. Any extra work I have to do get the program to compile must be clearly explained there.

How to run: Explain how to execute your program if it differs from specification

Known Bugs and Limitations: List any known bugs, deficiencies, or limitations with respect to the project specifications. Documented bugs will receive less deduction versus uncaught ones.

Summary of Features and EC completed: For projects with optional extra credits, or things you just felt like adding, you should summarize any extra work you did, or any features you implemented that might differ from specification, i.e. particially implemented features, and to which extent.

File directory: If you have multiple source files, (other than those created by the compiler), please briefly explain the purpose of each.

2. Makefile

3. Source Files: All the source files and header files.

DO NOT INCLUDE: Please delete all executable and object (.o) files prior to submission. '`make clean`' would be a useful target line to have in your `Makefile` for this purpose.

To submit, store everything (README, Makefile, source files) in a directory, say `hw1`. Then follow the directions here: `https://cs.brynmawr.edu/systems/submit_assignments.html`

If you discover an error in an earlier submission, you may repeat your submission, i.e. just run the script again.