CS355 Hw 2

Due by the end of day **Sunday, 2/11**. Written problems due by **Monday 2/12**, in class.

Programming Assignment:

1. Threads

   (a) Create two matrices `A` and `B` of size `N` × `N` where `N` is a `#define`d constant. Create `N` × `N` threads each of which writes a random positive integer between 1 and 10 into a distinct location in matrix `A` (i.e. no two threads write into the same location in A) and another random positive integer betweeen 1 and 10 into the same location in matrix `B`.

   (b) Print out the two matrices `A` and `B`.

   (c) Now create `N` × `N` threads each of which computes a distinct element of the matrix `C` = `A` × `B`.

   (d) Print out matrix `C`.

   (e) Now compute the sum of elements in each row of `C`, and find the maximum of all row-sums. First initialize a single variable `MAX_ROW_SUM = 0`, this is the variable that should hold the maximum of the row-sums at the end of this step. Create `N` threads each of which computes the sum of a distinct row in `C`, and update `MAX_ROW_SUM` if necessary. To make things a bit more interesting, after reading but before updating `MAX_ROW_SUM`, let each thread sleep some random amount of time (few seconds).

   (f) Print the maximum row-sum.

   Note: Do NOT use any intermediate variables to store individual row-sums.

   (g) I hope that by now you clearly see that we were intentionally generating a race condition in (e). Using POSIX thread mutexes, add code to resolve the race condition (consult manual pages on `pthread_mutex_*`, i.e. `pthread_mutex_init(), pthread_mutex_lock()`, etc.

2. Simple Shell: design and implement a simple (blocking) shell *mysh*. The basic function of a shell is to accept lines of text as input and execute programs in response. The requirments are listed below:

   (a) Simple execution of commands in the foreground. These are the steps of the most basic shell function, which are looped:

      i. Print a prompt of your choice

      ii. Get the command line (as a string)

      iii. Parse the command line into command and arguments

      iv. Fork a child which executes the command with its arguments, wait for the child to terminate (shell will be blocked and not accepting other commands until child completes execution)

Relevant system calls are `fork()`, `wait()`, `exec()` families and `exit()`. Note that your shell should be able to perform standard path search according to the search path defined by the `PATH` environment variable. Hint: remember `execvp`.

(b) Built-in commands (the ones *mysh* recognizes, implements and does not pass onto `exec`)
   - exit: Exits *mysh* and returns to whatever shell you started *mysh* from.

(c) When parsing the command line, *mysh* should ignore all white spaces. You might find the Clib functions `isspace()` and `strtok()` useful. We are going to build on this shell, so please have future expansions in mind when you write your parser. I highly recommend you design your parser so that it will separate text into tokens based on a list of delimiters/symbols and whitespaces. Currently only whitespaces and newlines are active delimiters, but in the next assignment (and in any real Unix shell), additional symbols are delimiters too, for example, any of `&`, `;`, `|`, `>` or `<`.

(d) I highly recommend that you use the `readline` library to read the commandline from terminal. Many things will be much simpler this way, including the extra credit if you will be attempting it (see below). Read the manual pages on `readline` to find out more about the functionalities the library offers and how to use it.

(e) Any command that calls an executable should work flawlessly, e.g. `emacs`, `code`, `ps`, `cat`, `gcc`, `ls`, `sleep`, and many others. Experiment! Note that `cd` will NOT work, because it is not an executable, but a file-system built-in command that must be implemented from scratch. You will be doing that when we get to the final project.

(f) Thou shall not crash. Since the shell is at the front line of user interaction, and its quality reflects heavily on the usability of an OS, it is paramount that *mysh* does not crash under any circumstances. Granted it doesn't do much yet compared to a real shell and feel free to simply report error for all things you do not know how to handle. However, your shell must be able to check for and catch all the exceptions. Also remember to check for memory leaks.

(g) When in doubt, consult the man pages and do what Linux does.

(h) Extra Credit: Add support for history. The command `history` will display the command history list. Read the environment variable `HISTSIZE` to figure out how many commands to keep. If that is not set, default to 50. Double exclamation points (`!!`) will repeat the last line of input that was typed in. Exclamation point with a number (`!n`) will repeat the nth command, and (`!-n`) will repeat the nth most recent command. Again, if you are unfamiliar with how command history works, please experiment with Linux first.

Supplementary Questions: Do the following questions from Tanenbaum.

- pages 174-180, problem numbers 1, 11, 32, 54, 55 and 56.

Deliverables:

1. Electronic submissions of the programming assignments.
2. Hardcopy: Answers to the supplementary questions.