CSE 421/521: Assignment #0

Due on Tuesday, September 5, 2017

Dantu 11:00am

Karthik Dantu

Useful Tools

Becoming a system's programmer is often accelerated by the use of several popular tools. Some of them are required to work (`git, make`) on this class, and you are better off familiarizing yourself with them sooner than later. Others help productivity but are not essential. Most of them come installed on your VM. Here is a small list with useful links to learn about them.
Note: While we provide these, we are not here to teach you these tools. The TAs might be happy to help you with them if they have the time, but it is expected that you have working knowledge of the essential tools and/or learn it in your own time.

`bash`

bash is Bourne Again SHell. It is the one of the more popular shells on *nix systems. This course assumes familiarity with a shell such as bash. Other shells include tcsh (shell on several cse machines such as timberlake), and zsh. Below are some useful links to learn bash and bash scripting.

- A list of online bash tutorials
- The Bash Academy
- Bash scripting tutorial
- Bash guide for beginners - teaches you bash, scripting and command line tools such as sed, awk and regular expressions

`git`

Version control is very useful while designing and implementing large projects. It not only serves as backup, but also helps version updates for easy retrieval as well as coordinate with others when working in a team. Git is a distributed version control system that also serves this purpose. It is expected that you know the basic features of git. Inner working of it is also interesting in itself and potentially useful. Here are some useful links for more information:

- Git Basics
- Presentation on Git for advanced beginners
- Git inner workings

`make`

As your software project grows in size as well as number of files, it usually makes sense to have a build system that automatically compiles/links all the relevant pieces together. `make` is one such system. It is also intelligent enough to look at previous builds, examine the files that have changed, and only build the modified sources and regenerate the binary. It does so by traversing a user-specified dependency graph and examining time stamps to determine if the source was modified after the binary. `make` however has somewhat esoteric syntax, and can be confusing if you do not understand the details of how to specify a `Makefile`. Here are a few good tutorials:

- `make` Howto
- `make`: Quick Overview
- The `make` software manual

`gdb`

C programming is challenging, and writing large systems software is hard in particular. However, on *nix systems, the `gcc` compiler has good integration with the debugger `gdb`. Just compile your programs with `-g` flag (with debug symbols), and you can use `gdb` to do debugging activities such as stepping through a program, inspecting variables, injecting values, adding breakpoints to pause flow etc.

- Beej guide to `gdb` - gdb basics
- Comprehensive introduction to `gdb`
- Official `gdb` documentation

Additional Tools

- tmux
- `vim`
- emacs tour
- ctags with vim or emacs

Setting Up Your Development Environment

For all programming assignments, our preference is that you use a Linux system. Specifically, we would like you to use VirtualBox VM and the Ubuntu 16.04 VirtualBox image we provide. Setup instructions are below.
Note: You are welcome to use your Linux machine, custom distribution, or any other development platform of choice. However, **we will not support** any other platform other than the one we provide. If you use any other platform, you'll have to do solve your system administration/programming/debugging challenges yourself.
Please download Virtualbox (5.0 or higher) for your host OS from here. Then, download the linux image from here.

Assignment-0 Reading

Familiarizing yourself with the `pthreads` library would be useful for Assignment-0. Here are some links that might help:

- POSIX Threads Programming from LLNL
- pthread tutorial
- Multi-threaded programming

Problem 1

Write a C Program with the following requirements:

1. The program should have exactly 2 threads. You should use the standard pthreads library for thread creation and management. This includes the main thread. So you have to spaw 1 additional thread after the program starts.

Problem 1 continued on next page...

2. The main thread should print odd numbers on the console with 1 number on each line Example:
   1
   3
   5
   7
   and so on..

3. The second thread should print even numbers on the console, again with 1 number on each line Example:
   2
   4
   6
   8
   and so on.

4. The program should stop after printing 1000 numbers (from both threads)

5. Notice that the output is jumbled up and the numbers are not in order. In part 2, fix this.

Part - 2:

1. Now use thread synchronization mechanics (ex. mutex, semaphores and conditional variables etc.) to synchronize the output.

2. After synchronization the numbers printed by both threads should be in order.
   Hint: The main thread prints a odd number and "informs" the second thread about this. Meanwhile, the second thread will be waiting for this "event" and prints a number only after the "event" occurs. Vice-versa for the main thread - after printing an odd number it waits for an "event" from the second thread to make sure that the following even number is printed before printing the next odd number.

   There is no one way to do this assignment. It can be achieved by using any synchronization mechanisms available on Linux and pthreads (Ex. Semaphores, Locks, Conditional Variables, Pipes and Sockets)

   Bonus: Repeat the assignment using a different synchronization technique.