Principles of Operating Systems

CMSC 421 - Spring 2018

Homework 1

50 points total

Due by 11:59PM EST on Wednesday, March 7th

Part I

25 points

Please provide written responses to the following questions. Each response should be at least 5-7 sentences in length. Each question is worth 5 points.

1. Compare and contrast microkernel and monolithic kernel-based operating systems. Name one kernel that follows each of these models.
2. What is the purpose of a system call? Give at least 3 concrete examples of potential system calls in an OS and explain why each would be a system call.
3. Describe what a context switch is. How does context switching differ between processes and between threads within a single process?
4. Name two methods of IPC (interprocess communication) and discuss the pros and cons of each.
5. What is the difference between a program being executed by a single thread and one that is being executed by multiple threads? Compare and contrast task parallelism and data parallelism as it relates to multithreaded programs.

You must submit a PDF file of the answers to these questions on the Homework 1 assignment on Blackboard. You may prepare your responses in any word processing application of your choosing, but you must submit a PDF file. **Do not submit Microsoft Word/Apple Pages/LibreOffice documents with your responses. You must convert to a PDF file before submitting or you will lose points.**

Part II

25 points

Complete the following programming assigment and submit your code using the GitHub repository that you will have created for this assignment. This assignment must be completed in the C programming language (you can choose to use C89/C90/ANSI C, C99, or C11 as you see fit).

Throughout this semester, you will be building a shell for a Linux system as part of your homework assignments. While this shell will start out extremely basic in this assignment, you will be extending your shell program throughout the semester to give it more interesting capabilities. Please write the code this in mind as you proceed (extensibility is key). Also, keep in mind that you will need to use this code as a base for all future homework assignments — it is in your best interest to ensure that the base you write for this assignment works well and completely fulfills the requirements set herein.

For the first iteration of your shell program, you will only need to support a few very basic features of a full-fledged *nix shell. Specifically, you will need to have support for all of the following:

- Present the user with a prompt at which he or she can enter commands.
- Accept command input of arbitrary length.
- Parse command-line arguments from the user's input and pass them along to the program that the user requests to be started. The command to be called will be either specified as an absolute path to a program binary (like `/bin/ls`) or as the name of a program that resides in a directory in the user's `$PATH` environment variable (like `gcc`) — relative paths are not required to be supported (so, we will not test your code for this assignment by switching to the `/usr` directory and attempting to run `bin/gcc`, for instance). Also, you are not required to handle quoting or escaped characters at this point — you can assume that all spaces in the user's input delimit arguments.

You are not expected to support any of the following features, however these may or may not be added in a future homework assignment.

- Scripting
- Setting/retrieving environment variables
- Support for pipes (including stdin/stdout redirection)
- Built-in functionality that is often part of a *nix shell (such as implementations of common utilities like `cd`), other than what has been outlined above
- The ability to change directories or anything else of the like
- Quoting/escaping of characters in arguments/program names

Your shell program is not allowed to use any external libraries other than the system's C library. Do not try to use libraries like Readline. You will lose points for using external libraries to implement shell functionality! You are not allowed to use any of the following functions in the C library to implement your shell:

- system (insecure and can lead to major problems)
- getline
- scanf (this one is largely to save you trouble)
- fscanf (ditto)
- popen

On the other hand, the following list of functions may prove to be very useful to you, depending on how you you implement your shell:

- fgetc
- malloc
- realloc
- free
- strtok_r
- strchr
- isspace
- fork
- exec (a family of functions)
- fprintf

You are not allowed to implement any of your shell's functionality by calling on another shell to do the work. You must do the argument parsing and calling of programs in your own code!

When submitting your shell program, please be sure to include the source code of the shell program (in one or more C source code files), as well as a Makefile that can be used to build the shell. Your shell must be able to be built and run on a VM as has been set up for this course in your projects. Also, you should include a README file describing your approach to each of the requirements outlined above.

If you would like a template for use as a Makefile for your shell, I have provided one here.

To submit your project, you must first accept the assignment on GitHub. The link to do so is posted on the course Piazza page. Once you have done that, make sure that your project files (any source files and your Makefile) are in their own directory, then run the following commands in that directory (substituting the list of files you need to commit for `your_files_go_here` and your GitHub username for `username`, of course):

```
git init
git add your_files_go_here
git commit
git remote add origin git@github.com:UMBC-CMSC421-SP2018/shell-username.git
git push -u origin master

git tag hw1
git push origin --tags
```

Last modified Sunday, 25-Feb-2018 14:35:01 EST