Programming Project #1
CS4352 Operating Systems
Spring 2020

Due Date: 3/10, 9:30 a.m., soft copy via Blackboard.
Late submissions are accepted till 3/17, 9:30 a.m., with 10% penalty each day.
No submissions accepted after 3/17, 9:30 a.m.

In this programming project, you are asked to try shell, shell programming, and system programming (in C programming language) on Linux platform. This assignment consists of two mandatory parts: part 1 and part 2, and one extra credit part, part 3. This assignment is designed to assist you for better understanding of operating systems including Unix/Linux shell and system programming, as well as enhancing programming skills and experiencing programming on a Unix-like environment.

Part 1. Shell Experiences and Programming

You are asked to try Unix/Linux shell in this part of project.

Requirements:

1. Run the *displayinfo.sh* and *greeting.sh* scripts provided and report the result.
2. Made two modifications to the *greeting.sh* script provided: 1) the script only shows a greeting message when it's morning, afternoon, or evening; please make a code modification to show a greeting message "Hello [your login name]!" when it's other than morning, afternoon, or evening; 2) please output the greeting message to a file called "greetingmsg", and if this file already exists, then deletes the file first, then output the message to this file.
3. Run the modified *greeting.sh* script on the Oak machine and report the result.

Part 2. System Programming and Process Management (in C programming language)

You are asked to develop basic system programming including process creation and termination on a Linux platform in this part of programming project. You are asked to create multiple children processes to work under one parent process. In theory, children processes can do their own work separately or cooperatively to accomplish a task. In this assignment, these children processes simply print out a "hello" message together with their PIDs (process IDs) and exit. You should use *fork()* and *wait()* system calls to implement this program.

Requirements:

1. Take the number of children processes as an argument when the parent process creates children processes. This argument should be passed through a command line argument.
2. The parent process creates children processes and should print out an error message if creation fails. The parent process should also wait for all children processes to finish and

then exit.
3. Each child process should print out a hello message together its PID and then exit.
4. Test with 2, 4, and 8 children processes.
5. Next, instead of creating multiple children processes of a parent process, you are asked to create a chain of processes. That is to say, the parent process will create one child and wait for it to finish, while the child creates its own child and wait for it as well, and so on. The last child created should print out the message and exit immediately so that its ancestors can finish too. Test your program with 2, 4, and 8 children processes again.
6. Develop a Makefile, similar to the one we will discuss in lecture to automate the compilation process.
7. Develop a shell script to automate the test process, i.e. to test with 2, 4, and 8 children processes for both versions of program automatically with this script.

Hints:

- System calls *getpid()* and *getppid()* return a process ID and the parent process ID, respectively.

Part 3. 20% Extra Credit: IPC (InterProcess Communication) Programming (in C programming language)

In Part 3, you are asked to develop a multithreaded program and a Makefile to automate the compilation on Linux platform. This assignment assists you for better understanding of processes and threads management, multithreaded programming, and enhancing programming skills and experience with programming on a Unix-like environment.

You are asked to implement a multithreaded producer-consumer problem with PThreads library in this programming assignment. The producer-consumer is a common problem that requires cooperating processes or threads. In this problem, a producer produces items and put into a shared buffer, then these items are consumed by consumers. Synchronization and mutual exclusion are required to solve the problem correctly as discussed in the lecture and described in the textbook (please see section 2.3 and corresponding lectures). Figure 2-32 in the textbook shows a solution with one producer and one consumer. In this part of assignment, you are asked to implement this problem with support of multiple producers and consumers threads with PThreads library on a Linux platform.

Requirements:

1. Ensure appropriate synchronization and mutual exclusion using PThreads mutex locks and conditional variables as discussed in the class and shown in Figure 2-32.
2. Support multiple producer threads and multiple consumer threads. The number of producer threads and consumer threads can be either hard coded in the program or provided through command line arguments.
3. Print out the producer/consumer actions, e.g., “producer *i* produced one item”, “consumer *j* consumed one item”, “producer *i* found the buffer is full and waiting for a consumer to consume”, and “consumer *j* found the buffer is empty and waiting for a producer to produce”.

4. You can fix the shared buffer with 8 items, and produce 64 items from each producer thread. Test with 1 producer thread and 1 consumer thread, 2 producer threads and 2 consumer threads, and 4 producer and 4 consumer threads.
5. Develop a Makefile to automate the compilation process.

Hints:

- Implement a producer routine and a consumer routine, in which, a mutex lock and a conditional variable used to synchronize accessing the shared buffer. Figure 2-32 in the textbook shows a solution with one producer and one consumer.
- In the main function, create multiple producer and consumer threads separately, and call for producer/consumer routine. Don't forget to join threads and deallocate data structures such as mutex locks and conditional variables created as shown in Figure 2-32.
- Use Reference Materials regarding Pthreads listed at the end of this project description to help with Pthreads APIs (all APIs you need are already shown in Figure 2--32).

Expected Submission:

You should submit a single tarball/zipped file through the Blackboard containing the following, and please name your submission file starting with LastName_FirstName_Project#1.

- Source codes for part 1, part 2, and part 3 (if you take this extra credit).
- Output files for the result of test cases for all parts.

How to report your results

- You can simply use bash redirection ('>') to produce log files like what we showed in class, and then submit the log files (plain text files)

How to submit your codes and results

- First, on Oak machine, you can create a tar ball to include all your source codes, logs, and other files, like one of below commands
  - `tar cvf mycs4352project.tar *`
  - `tar cvfz mycs4352project.tgz *`
- Then you can use "scp" or any SFTP client, e.g. FileZilla to download the tar/zipped file to your local machine, after that you can submit to Blackboard just like you submit your other homework solutions
  - E.g. you can download "pscp.exe" from the Putty website, then run a command like below to copy/download "mycs4352.tar" file to your local machine (you'll need to replace the path name "home/TTU/yonchen/cs4352/public_gitlab_repo/mycs4352.tar" with your own file name
  - `pscp.exe yonchen@oak.cs.ttu.edu:~/cs4352/public_gitlab_repo/mycs4352.tar .`
  - `pscp.exe yonchen@oak.cs.ttu.edu:/home/TTU/yonchen/cs4352/public_gitlab_repo/mycs4352.tar .`

Grading Criteria:

Please note that, if needed, we may request an in-person, 5-10 mins quick demo from you.

Part 1	Percentage %	Criteria
40%	50%	Correct shell programming modification, modified script can be run
	50%	Correctness of result
Part 2	**Percentage %**	**Criteria**
60%	10%	Inline comments to briefly describe your code
	30%	Correct use of system calls/library procedures of creating processes, check process ID
	20%	Correctness of result. Source code can be compiled and executed.
	20%	Successfully carry out specified test cases
	10%	Correctness and features of Makefile (minimum features of automating compilation and cleaning up)
	10%	Correct shell script to automate tests
Part 3	**Percentage %**	**Criteria**
20% (extra credit)	10%	Inline comments to briefly describe your code
	30%	Correct use of system calls/library procedures of creating threads, correct use of primitives for synchronization and mutual exclusion, e.g. mutex locks, conditional variables
	30%	Correctness of result. Source code can be compiled and executed.
	20%	Successfully carry out specified test cases
	10%	Correctness and features of Makefile (at least automate compilation and cleanup)

Source Code Samples

Please checkout source code samples with executing the following command on the Oak machine:

```
git clone https://discl.cs.ttu.edu/gitlab/yongchen/cs4352.git
```

Reference Materials:

- Linux Shell scripting: http://www.freeos.com/guides/lsst/
- Linux system programming:

Book: *Linux System Programming*

Online:
Tutorial for Beginners, http://www.ee.surrey.ac.uk/Teaching/Unix/
Advanced Linux Programming, http://www.advancedlinuxprogramming.com/alp-folder/advanced-linux-programming.pdf

Stack Overflow, http://stackoverflow.com/
www.unix.com
The Geek Stuff, http://www.thegeekstuff.com/

- PThreads**:**
  - POSIX Threads Programming, https://computing.llnl.gov/tutorials/pthreads/
  - Linux Tutorial: POSIX Threads, http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html
  - Complete Pthreads API: http://pubs.opengroup.org/onlinepubs/7908799/xsh/pthread.h.html

- Makefile**:**
  - http://www.gnu.org/software/make/manual/make.pdf or
  - http://www.gnu.org/software/make/manual/html_node/index.html