CSE-376 (Spring 2020) Homework Assignment #4
(revision 3)
Due Thursday May 14, 2020, 11:59pm
(This assignment is worth 25-30% of your grade.)

* PURPOSE:

Put together much of the knowledge gained in this course, design, develop, and test code to demonstrate your low-level understanding of low-level Unix system calls. You will develop a client/server system for job submissions. You are free to use just about any system call here, whether we described it in class or not.

* DEMOS:

To expedite grading, and also allow you to show off your work, we'll have short demos (30-45 min) per group (or for individuals who work alone). Demos will be held over a period of 2-3 days following the due date. We will use Zoom and record the demos too. More info TBA (e.g., demo signup sheet, procedures).

* TEAMING:

For HW4, you may work alone or in groups of up to four people. Regardless of your group size, you will be graded the same, but at the end of the course, I take group sizes into account when assigning final course grades. See my grading policy online for details.

If you work in a team, you MUST fill out the following form by Thursday, 04/30/2020:

https://docs.google.com/forms/d/1Tn_njUOIftU59HHR7scvZ-hKdv5EUJEHZpFcw5m_zuo/edit

This is a Google form for you to select the CS ID of your team members (your CS ID is often the same as you SBU NetID). You only need to fill out the form if you work in a team (2-4 people): if you work alone, no need to fill the form. You will need to login to Google Forms with your CS ID to be able to fill out the form. Be sure to list the correct CS ID.

I will create a shared GIT repo for each team, so you will NOT be using your individual GIT repos. You must fill out the form by the deadline stated above: afterwards, I would not allow new teams to form or existing teams to split or change.

CHOOSE YOUR TEAM MEMBERS CAREFULLY. I will not accept later complaints about team members who "drop out" or don't contribute as much, or worse (cheating). That said, it is ok to borrow code from your homeworks 1-3 for this assignment, as long as you properly document it in your README.

Note that being the final homework assignment in the class, it is on purpose made more open ended than the previous assignments. I would expect you to demonstrate maturity, and come up with a clever, efficient design that you will describe in detail (in a design document that's part of your grade). You have more freedom in this assignment to come up with your own design and justify it. Recall that as time goes by this semester, I expect all of you to demonstrate an increased level of maturity and expertise in this class. This is your chance to shine and show off what you've learned so far in this class.

* TASK

This is a more open ended assignment, on purpose: consider it a "class project".

The goal is to create a "job submission" system: a client that can submit jobs to be executed with various criteria; and a server that will process these jobs asynchronously and report their status. We leave the design details up to you, but here are the minimum requirements (meaning you can add additional features as you see fit, and we'll consider them as extra credits):

SERVER: the server will have a number of configuration options that you can pass on the command line:

1. The maximum number of jobs that can be concurrently run. This is to ensure we don't blow up the system.
2. The server will track resources used by each running job (a process): time and CPU cycles consumed, memory used, etc. The server will be able to take action on a process that exceeds these caps: kill a job, stop it, use nice(2) to change it to a different priority, etc.
3. The server will be able to report back to the client the status of one or more jobs: list one or more jobs, resources they're using, are they running or not (and why not), and the output these jobs produced on stdout/stderr.

CLIENT: the client will be able to issue commands to the server to list one or more jobs, check their status, tell the server to suspend/kill a process or change its "nice" priority, retrieve status/error codes, etc.

The client could submit new jobs. New jobs will be described as a vector of argv[] and envp[] strings, along with an additional data structure that describes the restrictions on the job to run (max time to run, max resources to consume, priority, etc.).

CLIENT/SERVER INTERACTION:

The client and server should communicate via a Unix pipe (domain socket). This is a special file you create on the file system that has two communication channels. The server will create the pipe and listen for commands on it (e.g., "/home/jdoe/.hw3server_control"). The client could write(2) to the pipe and can read(2) back from it. When one side writes to a pipe, the other side can read back from it. Otherwise a reader trying to read from a pipe would be blocked waiting for the other side to write something. (Consider using poll(2) or select(2) like calls for the server.)

You will have to design a small protocol for communicating over the pipe. A good protocol starts with a short code that determines the type of operation to perform and the type of response that is returned. For example, the first byte can be 1 to mean "submit a new job", 2 to mean "list existing jobs", 3 to mean "kill an existing job", etc. Figure out how many commands you need and give them some unique command ID. For example, the client could submit the kill to an existing job: first byte has the value 3; next byte lists the job ID to kill. Or, for a more complex operation like submit a new job:

- first byte is 1 (command type number)

- next 4 bytes (an int) say how many bytes are in the whole message, to help the server verify the message size and malloc(3) enough space to hold it.
- next byte is the same as argc
- then come all bytes of argv, delimited by \0's
- then come envp[] array
- then comes a C struct that denotes limits: first value could be "max time", second could be "max memory", 3rd byte could be "priority value".

You then write(2) that data to the pipe. And if you've set things right on the server, the server will wake up and be able to read from the pipe: it can read one byte to determine the command type, and then read the rest incrementally or in one shot.

The client can then go into a read(2) loop waiting for a response: when the server, say, writes a return status reply, the client will be able to read and process/display it. For example, the server can write a reply message that looks like this:

- first byte is 1 (meaning a REPLY to a "submit job command")
- next byte can be an int stating whether the submitted job was received successfully (0) or failed (errno). Note this is NOT an indicator of the exit status of a job submitted.
- last byte would be optional: if the command succeeded, you can return a unique job ID assigned to this command.

Note that you will need a different client/server command (or two) to return the stderr and/or stdout output of a command that finished or was terminated. It would help if each submitted job would have a unique job ID, so you can send a request to "query status of job N".

For this assignment, assume all submitted programs are non-interactive: that is, they do not read from stdin. This will simplify the assignment. But also, in most job-submission systems, the work submitted is self-contained and does not depend on user interaction (e.g., submitting a weather simulation to an HPC cluster).

DESIGN DOCUMENT: please write a design document describing the requirements, design, and implementation of your system: you must describe your protocol and, client submission tool flags/options, key data structures, and system operation. In other words, make this design document sufficiently detailed that anyone reading it could re-implement your system from scratch. Use 11pt TimesRoman font, 1” margins, and number each page. Ensure your name(s) are listed at the start of the document. I expect the document to be at least 2 pages but no more than 6 pages. Feel free to include code snippets if it helps, figures you draw, etc. The design document will be worth at least 10 points (out of 100) in this assignment. You must submit your design doc as a file named “design.pdf” (all lower-case, PDF format only); if you author this design.pdf doc with something else, it is ok to also include the source file (latex, docx, etc.).

* TEST SCRIPTS

You know the drill. Write and include as many or as few test scripts/programs to show us that your features work. The easier it’d be for us to test your code, the happier the graders will be. Test scripts would also allow you to automate your demo.

* STYLE AND MORE:

Aside from testing the proper functionality of your code, we will also evaluate the quality of your code. Be sure to use a consistent style, well documented, and break your code into separate functions and/or source files as it makes sense. Your code should be properly commented (e.g., don't use /* C comments */ to turn off code, add comments to every major function and data structure, etc.).

To be sure your code is very clean, it must compile with "gcc -Wall -Werror" without any errors or warnings! If the various sources you use require common definitions, then do not duplicate the definitions. Make use of C's code-sharing facilities.

There is no README requirement for this assignment: all information should be in the design.pdf document that you must provide.

* SUBMISSION

You will need to submit all of your sources, headers, scripts, Makefiles, and README. Submission is accepted via GIT only! Do not submit regenerable files like binaries, *.o files, or any temp files -- only true "source" files.

PLEASE test your submission before submitting it, by checking it out in a separate directory, compiling it cleanly, and testing it again. DO NOT make the common mistake of writing code until the very last minute, and then trying to figure out how to use GIT and skipping the testing of what you submitted. You will lose valuable points if you do not get to submit on time or if your submission is incomplete!!!

(General GIT submission guidelines are available on the class website.)

* EXTRA CREDIT

[A] max 30 points.

Because this is a more open-ended assignment, there are no explicit extra credit features. However, feel free to add additional features as you feel it makes sense. You can ask me privately or publicly (e.g., piazza) and I'll let you know if I think if such features make sense (please don't ask me how much this or that extra feature would be worth). Think about useful features to someone using such a system, efficiency, and more. Impress us, document all your features, and demo them.

Study time based functions and getrusage(2).

[D] 6 points (max).

To incentivize you to submit your code earlier than the deadline, we'll give you 3 points if you submit your assignment at least 24 hours before the official deadline; and 3 points if you submit at least 48 hours before. Note that we count the LAST git-push to your repo as the last time you submitted the assignment, even if you made a very small change.

Good luck.

* Copyright Statement

* ChangeLog

- V1: initial draft.
- V2: TA review
- V3: clarify no need to support stdin