Assignment 5 Part 1 Instructions

Assignment 5 Part 1: Native Socket Server

Github Classroom Start Instructions

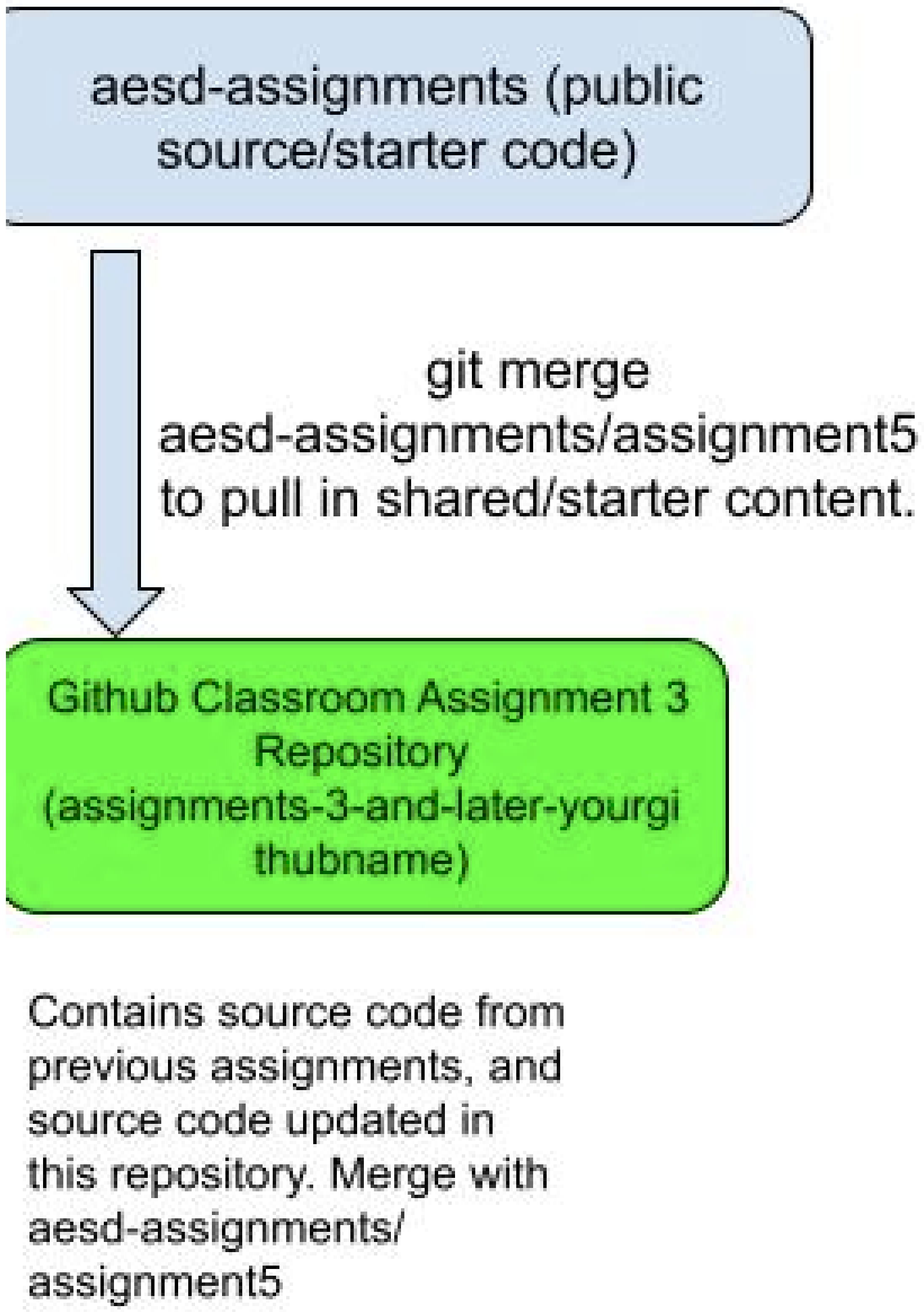
Execute the following instructions in your aesd-assignments assignments-3-and-later local repository folder:

1. **`git fetch assignments-base`**
   - This step assumes you've already created the remote using **`git remote add assignments-base git@github.com:cu-ecen-aeld/aesd-assignments.git`** in assignment 2. If you started from a new repo you'll need to re-run the git remote-add step before attempting to fetch.
2. **`git merge assignments-base/assignment5`**
3. **`git submodule update --init --recursive`**
   - Also make sure your aesd-assignments repository origin still points to your assignments-3-and-later repository using **`git remote get-url origin`**

Suggested Reading:

1. Video content, daemons and sockets
   - https://beej.us/guide/bgnet/html/
2. QEMU Documentation and network options
   - https://qemu.readthedocs.io/en/latest/system/devices/net.html
3. Buildroot Documentation:
   - https://buildroot.org/downloads/manual/manual.html#configure
4. Mastering Embedded Linux Programming Chapter 10: Starting Up
5. Init scripts documentation
   - http://man7.org/linux/man-pages/man8/start-stop-daemon.8.html

Repository Setup:


aesd-assignments (public source/starter code)
git merge aesd-assignments/assignment5 to pull in shared/starter content.
Github Classroom Assignment 3 Repository (assignments-3-and-later-yourgithubname)
Contains source code from previous assignments, and source code updated in this repository. Merge with aesd-assignments/ assignment5

Implementation:

1. Modify your `assignments-3-and-later` repository to add a new directory "server".

2. Create a socket based program with name **`aesdsocket`** in the "server" directory which:

a. Is compiled by the "all" and "default" target of a Makefile in the "server" directory and supports cross compilation, placing the executable file in the "server" directory and named **aesdsocket**.

b. Opens a stream socket bound to port 9000, failing and returning -1 if any of the socket connection steps fail.

c. Listens for and accepts a connection

d. Logs message to the syslog "Accepted connection from xxx" where XXXX is the IP address of the connected client.

e. Receives data over the connection and appends to file **`/var/tmp/aesdsocketdata`**, creating this file if it doesn't exist.

- Your implementation should use a newline to separate data packets received. In other words a packet is considered complete when a newline character is found in the input receive stream, and each newline should result in an append to the **`/var/tmp/aesdsocketdata`** file.
- You may assume the data stream does not include null characters (therefore can be processed using string handling functions).
- You may assume the length of the packet will be shorter than the available heap size. In other words, as long as you handle malloc() associated failures with error messages you may discard associated over-length packets.

f. Returns the full content of **`/var/tmp/aesdsocketdata`** to the client as soon as the received data packet completes.

- You may assume the total size of all packets sent (and therefore size of **`/var/tmp/aesdsocketdata`**) will be less than the size of the root filesystem, however you may **not** assume this total size of all packets sent will be less than the size of the available RAM for the process heap.

g. Logs message to the syslog "Closed connection from XXX" where XXX is the IP address of the connected client.

h. Restarts accepting connections from new clients forever in a loop until SIGINT or SIGTERM is received (see below).

i. Gracefully exits when SIGINT or SIGTERM is received, completing any open connection operations, closing any open sockets, and **deleting the file /var/tmp/aesdsocketdata**.

- Logs message to the syslog "Caught signal, exiting" when SIGINT or SIGTERM is received.

3. Install the netcat utility on your Ubuntu development system using **`sudo apt-get install netcat`**

4. Verify the sample test script "sockettest.sh" successfully completes against your native compiled application each time your application is closed and restarted. You can run this manually outside the ./full-test.sh script by:

- Starting your aesdsocket application
- Executing the **`sockettest.sh`** script from the assignment-autotest subdirectory.
- Stopping your aesdsocket application.

5. Modify your program to support a **`-d`** argument which runs the **`aesdsocket`** application as a daemon. When in daemon mode the program should fork **after** ensuring it can bind to port 9000.

- You can now verify that the **`./full-test.sh`** script from your aesd-assignments repository successfully verifies your socket application running as a daemon.

6. Tag the assignment with "assignment-<assignment number>-complete" once the final commit is pushed onto the respective repositories. The instructions to add a tag can be found here

Validation:

1. The **`full-test.sh`** script should complete successfully against your assignments-3-and-later repository, validating your socket implementation.