Programming Assignment 1: A Simple Conferencing Application Using the select() Call

Due: September 25th, 2018 23:59:59pm

This assignment is worth 7% of your total score in this course. In this assignment, you will create a conferencing application that enables conversation between multiple conferencing clients. Clients connect to the server and all the connected clients receive all the messages transmitted by any of the clients. This group communication is facilitated by the conference server. A client reads the text typed by the user and sends it to the server. It is the responsibility of the server to relay that text to all the other clients. The clients receive the text relayed by the server and display it on the screen.

You need to write two programs: `confserver` and `confclient`, using sockets for communication between the client and server. The functionality of each of these programs is described below.

confserver usage: ./confserver

The server first creates a socket using the `socket()` system call. It then binds its address to the socket using the `bind()` system call. It **specifies the port number as 0** which allows the system to assign an unused port number for the server. The assigned port number is displayed on the screen. Then `listen()` call is used to indicate that the server is now ready to receive connect requests. The server's job is wait for either connect/disconnect requests or messages from clients. If it receives a connect request, it accepts the connection and adds it to the list of established connections which it has to monitor for messages. If it receives a disconnect request, it closes the connection and removes the client from the list of established connections. Otherwise, it forwards the message to all other clients. This monitoring of events from several connections can be performed by using the `select()`[1] system call.

confclient usage : ./confclient <servhost> <servport>

The client first creates a socket using the `socket()` system call. It then connects to the server using the `connect()` system call. The whereabouts of the server are passed as command line arguments. One the connection is established, the client's job is to wait for either user input from the keyboard or messages from the server. Inputs from the user are sent to the sever and the messages from the server are displayed on the screen. Once again, `select()` call is used for monitoring the input from keyboard and socket.

I have provided a template for you to work on this assignment. The template contains the following file for the assignment:

`confclient.c` contains source code of conferencing client

`confserver.c` contains source code of conferencing server

`confutils.c` routines used by server and client

`Makefile` makefile to build confserver and confclient executables

[1] http://beej.us/guide/bgnet/output/html/multipage/advanced.html#select

`goodserver` an example confserver executable compiled from our implementation

`goodclient` an example confclient executable compiled from our implementation

You have to supply the missing code in the files `confclient.c`, `confserver.c`, and `confutils.c`. The location of missing code is marked with "FILL HERE". Your task is to fill in the missing code.

You are supplied with executables `goodserver` and `goodclient`. These are the binaries of our implementation. You can run them and see how they behave. **Your implementation is expected to produce similar effect.**

The `goodserver` program takes no command line arguments. For example, to run the `goodserver` program:

```
$> ./goodserver
admin: started server on 'remote01.cs.binghamton.edu' at '44529'
```

The `goodclient` program takes server host name and port as command line arguments. For example, to connect to the above server, we can run:

```
$> ./goodclient remote01.cs.binghamton.edu 44529
server address: 128.226.180.163
admin: connected to server on 'remote01.cs.binghamton.edu' at '44529' thru '39182'
```

You run the server first (in the example above, on `remote01.cs.binghamton.edu`) and then many clients (possibly on different machines). Anything typed by any client would appear at all other clients.

To exit from the client programs, you can press Ctrl-D. In case of server, you can just press Ctrl-C to kill it.

Note: When accessing `remote.cs.binghamton.edu`, you are actually redirected to one of the 8 `REMOTE` machines ({remote00, remote01, ..., remote06, remote07}.cs.binghamton.edu) using DNS redirection. So to test your implementation, you need to make sure your client is contacting the server using the correct host name: "remoteXX.cs.binghamton.edu", not "remote.cs.binghamton.edu".

Github classroom

To access this assignment, first log into your Github account created with your BU email. Then go to: `https://classroom.github.com/a/92kWroz_`. After clicking this link, Github classroom will ask you to join the class roster by selecting your email from a list. Github classroom will automatically create a repository (e.g., if your Github username is jdoe, the repository will be named cs428-cs528-pa1-jdoe) and import starter code (the template that you work from) into the repository.

This repository is a private repository. Only you, course instructor, and teaching assistants are able to see this repository. To clone this repository to your local file system,

```
git clone https://github.com/Yao-Liu-CS428-CS528/cs428-cs528-pa1-username.git
```

Note that: i) `git` is already installed on CS department computers. To use it on your own computer, you must install it first; ii) you must replace "username" with your own Github username.

To add a file to the next commit, use the `git add` command. To commit your code, use the `git commit` command. Be sure to also push your commit to the Github repository after every commit using the `git push` command (e.g., `git push origin master`).

We expect each repository to have **at least three commits** (not including the two commits in the template code), with the first one and the last one **more than 48 hours apart**.

If you have never used `git` before, you can start with a cheatsheet prepared by Github[2]. More detailed references are also available online[3]. You can also come to the instructor and the teaching assistants' office hours.

How to submit

To submit, commit and push your latest code to the private Github repository Github classroom created. Your commit should contain the following files:

1. Your fully implemented `confclient.c`, `confserver.c`, and `confutils.c`.
2. A `Makefile` to compile your source code into two executables, which should be named `confclient` and `confserver`. (You can use the Makefile provided in the template.)
3. (Optional) A `Readme` file if there is anything you want the TA to be aware of when grading.
4. A `STATEMENT` file, containing the following statement followed by the student's full name:

   "I have done this assignment completely on my own. I have not copied it, nor have I given my solution to anyone else. I understand that if I am involved in plagiarism or cheating I will have to sign an official form that I have cheated and that this form will be stored in my official university record. I also understand that I will receive a grade of **0** for the involved assignment and my grade will be reduced by one level (e.g., from A to A- or from B+ to B) for my first offense, and that I will receive a grade of **"F" for the course** for any additional offense of any kind."

After pushing your final commit to the Github repository, please let us know by **submitting your commit hash to myCourses**. There are two ways to locate your commit hash: You can type `git log` in your local repository, which will output the commit history. Locate the commit you want to use as your final submission, record the hash associated with the commit. For example, the second commit in the template code has a hash value of "41d2cda799d6e08ecb27bef405418a90526f39a3". Alternatively, you can also go to the Github page of your repository, e.g., `https://github.com/Yao-Liu-CS428-CS528/cs428-cs528-pa1-username` and locate it on the webpage.

It is important that you submit your commit hash to myCourses. This helps us know your submission is ready for grading and which of your commits we should grade. We will not grade your assignment unless you have submitted the commit hash to myCourses before the assignment submission deadline

Your assignment will be graded on the CS Department computers `remote.cs.binghamton.edu`. It is your responsibility to make sure that your code compiles and runs correctly on these remoteXX computers.

Your assignment must be your original work. We will use MOSS[4] to detect plagiarism in the projects.

[2] `https://education.github.com/git-cheat-sheet-education.pdf`
[3] `https://git-scm.com/docs`
[4] `https://theory.stanford.edu/~aiken/moss/`