Project 1: A Conferencing Application

Part I is due on February 22nd, 2017 23:59:59pm
Part II is due on March 1st, 2017 23:59:59pm

In this project, 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.

1 Part I: Using the select()[1] call

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.

70% of your Project 1 score will be based on this part.

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()` 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 file, `template.tgz`. After downloading, you can run `tar xzvf template.tgz` to extract files in the tar ball into a directory that contains the following files for the project:

`confclient.c` contains source code of conferencing client

`confserver.c` contains source code of conferencing server

`confutils.c` routines used by server and client

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

`Makefile` makefile to build confserver and confclient executables

`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".

How to submit (Part I)

To submit Part I of the project, you should first create a directory whose name is "your BU email ID"`-project1-part1`. For example, if your email ID is `jdoe@binghamton.edu`, you should create a directory called `jdoe-project1-part1`.

You should put the following files into this directory:

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. A `Readme` file describing the your implementation details and sample input/output.

Compress the directory (e.g., `tar czvf jdoe-project1-part1.tgz jdoe-project1-part1`) and submit the tarball (in `tar.gz` or `tgz` format) to Blackboard. Again, if your email ID is `jdoe@binghamton.edu`, you should name your submission: `jdoe-project1-part1.tar.gz` or `jdoe-project1-part1.tgz`. Failure to use the right naming scheme will result in a 5% point deduction.

Your project 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. If your code does not compile on these computers, you will receive no points.

2 Part II: A multi-threaded version

In Part I, the `confserver` handles multiple connections from clients using the select() call. In fact, this can also be implemented in a multi-threaded version, allowing each socket connection be handled by a separate thread.

In Part II, you are asked to implement a multi-threaded version of the `confserver` using the **pthreads** (POSIX threads) library. The `multi-threaded-confserver` is expected to produce the same effect as the `goodserver` we have provided in Part I. And it will be tested against the `goodclient` provided in Part I.

30% of your Project 1 score will be based on this part.

How to submit (Part II)

To submit Part II of the project, you should first create a directory whose name is "your BU email ID"`-project1-part2`. For example, if your email ID is `jdoe@binghamton.edu`, you should create a directory called `jdoe-project1-part2`.

You should put the following files into this directory:

1. All relative source files.
2. A `Makefile` to compile your source code into an executable, which should be named `multi-threaded-confserver`.
3. A `Readme` file describing the your implementation details and sample input/output.

Compress the directory (e.g., `tar czvf jdoe-project1-part2.tgz jdoe-project1-part2`) and submit the tarball (in `tar.gz` or `tgz` format) to Blackboard. Again, if your email ID is `jdoe@binghamton.edu`, you should name your submission: `jdoe-project1-part2.tar.gz` or `jdoe-project1-part2.tgz`. Failure to use the right naming scheme will result in a 5% point deduction.

Your project 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 CS Department computers. If your code does not compile on or cannot correctly run on the CS computers, you will receive no points.