Project 1: Conference Client and Server

Project Description

A conference server is an application program that enables communications between multiple conference clients. Clients connect to the server and all the connected clients receive all the messages sent by any of the clients. This group communication is enabled by the server. A client reads the message typed by the user and sends it to the server. It is the responsibility of the server to relay that message to all the other clients. The clients receive the message relayed by the server and display it on the screen.

You need to write two programs: server and client. The sockets are used for communication between the client and server. The functionality of each of these programs is described below.

server

usage : ./server

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 shown on the screen. Then *listen()* call is used to indicate that the server is now ready to receive connect requests.

The server waits 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 *select()* system call.

client

usage : ./client <*serverhost*> <*serverport*>

The client first creates a socket using the *socket()* system call. It then connects to the server using the *connect()* system call. The hostname and port of the server are passed as command line arguments.

One the connection is established; the client waits 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.

FAQ

*** What is in the folder?**

client.c contains source code of client
server.c contains source code of server
utils.c functions used by server and client
makefile make file to build server and client

execserver server executable
execclient client executable

All the codes and executables are located under folder project-1

*** What do we have to do?**

You have to supply the missing code in the files 'client.c', 'server.c' and 'utils.c'. The location of missing code is marked with "TODO:". Your task is to fill in the missing code.

*** How do I compile my programs?**

The 'makefile' is provided just for that purpose. To compile, run

```
make
```

It will create 'server' and 'client'.
(Alternatives: make server
make client
make server client
)

*** How do I test my programs?**

You are provided with the executables 'execserver' and 'execclient'. These

are the binaries of our implementation. You can run them and see how they behave. Your implementation is expected to produce similar effect.

*** How do I run these programs?**

The 'execserver' program takes no command line arguments. For example you can run it as follows.

```
./execserver
admin: started server on 'eecs325-VirtualBox' at '41117'
```

The 'execclient' program takes server host name and port as command line arguments. To connect to the above server, we can run

```
./execclient eecs325-VirtualBox 41117
admin: connected to server on 'eecs325-VirtualBox' at
'41117' thru '40804'
```

You run the server first and then many clients. Anything typed by a client would appear at all other clients. Note that port numbers will be dynamic.

*** How do I exit from these programs?**

In case of client, if you type Ctrl-D, it will exit. In case of server, you can just press Ctrl-C to kill it.

*** Do these programs run on any machine?**

They are compiled on the provided VM image, a linux machine. They may work on other machines with different kernel versions. TA will use the VM as the grading platform.

REMINDERS

- Submit your code to Canvas in a tarball file by running command in terminal:
  `tar -zcf [you-case-id]-proj-1.tar.gz project-1`

- If your code does not **compile** or could not **run**, you will not get credits.

- Document your code (by inserting comments in your code)
- DUE: 11:59 pm, Monday, April 1st