CSCE 3600: Systems Programming

Major Assignment 2 – Sockets & Synchronization

Due: 11:59 PM on Wednesday, December 6, 2017

COLLABORATION:

You should complete this assignment as a group assignment with the other members of your already formed group. *Each group should have 3 or 4 members, no more, no less.* Submit only ONE program per group. Also, make sure that you list the names of all group members who participated in this assignment in order for each to get credit.

PROGRAM DESCRIPTION:

In this assignment, you will write a complete C program to support a client/server and pseudo-client/client model) using Linux sockets using the **`apollo`** server instead of the standard Linux CSE machines (i.e., **`cse01`** – **`cse06`**). The program will consist of a server that will keep a running sum of the integral data being sent by two clients as follows:

- **Server**
  - The server will receive integer values from up to two clients (in any order) and add these values to its current total separate for each client, which the server then returns back to the originating client so that the client is aware of its most up-to-date total. Unlike *Minor 6*, the values sent in from the clients will be input manually from the keyboard for each client. The server can have, at most, 2 clients connected at one time, though this functionality (i.e., a client being able to send an integral value to the server and the server responding to this request) should still continue even when only 1 client is connected.
  - When the total sum for an individual client reaches a value between 1,024 and 49,151 (the range of ports registered for general use), the server will return this value back to the client as usual, but the server will also send the message "`PORT`" followed by this value to the *other* client, which will act as a trigger for both clients that the client whose value exceeded the port threshold will then act as a server for the *other* client to connect to (see *Clients* section below for details). Upon reaching this threshold value, the server shall reset the individual client's current total back to 0, even If only 1 client is connected at the time.
  - The server will provide important status updates of messages sent and received, such as connection events, each client's running total, and when the port threshold has been reached.
  - The server will continue to "listen" for clients and run indefinitely until manually terminated (using, for example, Ctrl-C). A client may disconnect and later re-connect to the server. Up to 2 clients may connect and disconnect at any time, but the server should still be able to respond to connected clients while listening for new clients (if 1 or no clients are connected). If 2 clients are already

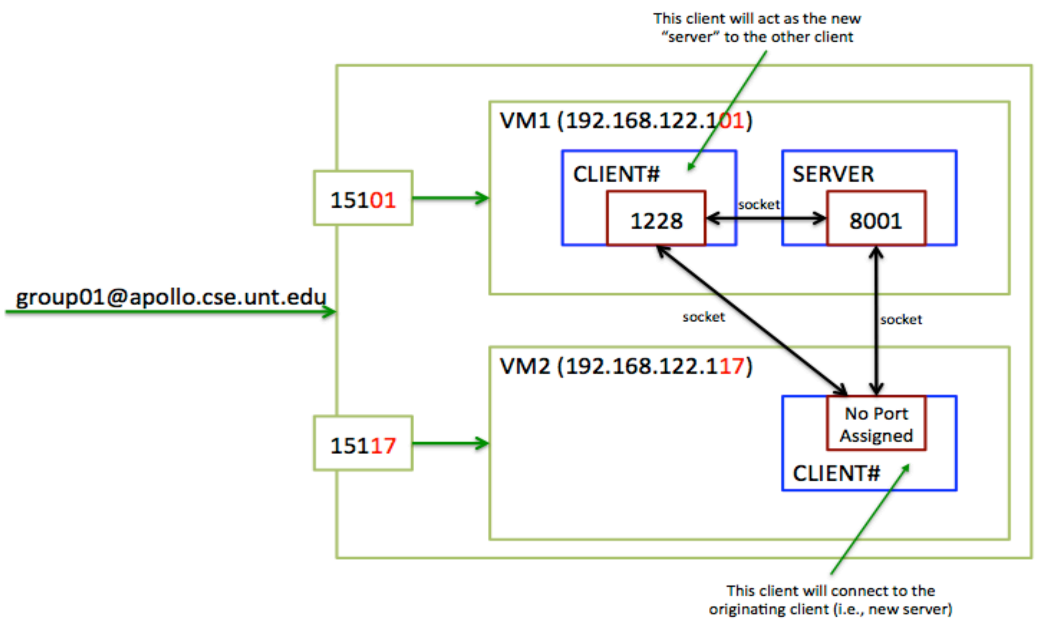
connected and another client attempts to connect, the server shall reject the connection (accepting the connection, realizing there are too many clients, then disconnecting is acceptable), but keep the other 2 connections active.

- **Clients** (note that there are up to 2 clients)
  - Once connected to the server, a user will manually enter integral values using the keyboard for each client that will be sent to the server. The server, in response to this integral value, will send the client a current individual total for that client so that the client is aware of the most up-to-date total.
  - Once the current individual total for a client reaches a value between 1,024 and 49,151 (the range of ports registered for general use), the "originating" client whose total has reached this threshold will then itself start acting as the "server" for the *other* client to connect to, using this total as the port number. The *other* client will then (1) disconnect from the server, (2) connect to the originating client (i.e., the new "client" server), (3) send its current total to the originating, and (4) disconnect from the originating client. The remaining "active" client should still be connected to the server and may continue sending its own integer values to the server.
  - If more than 2 clients attempt to connect to the server, the server shall reject the connection of the third client, which will then disconnect (if it did manage to connect) and then terminate the program. The other previously connected 2 clients shall remain active.
  - Each client will provide important status updates of messages sent and received, such as connection events, the client's running total and when a client-client connection is being established.
  - A client may voluntarily "quit" sending integer values to the server when the user manually enters a 0 for its integer value (whereupon the client will disconnect from the server).

APOLLO SERVER:

Each group will be assigned two VMs on the `apollo` server according to your group number on Blackboard. The server and one client will be run on one of the VMs, while another client will be run on the other VM. Note that client 1 and client 2 will consist of the same code handling both cases. The configuration for each team will be similar to the following diagram, although IP addresses and ports will vary for each team.

The figure shows the client on the remote machine connecting to the client on the same machine as the server, but it could be the other way around (i.e., the client on the same machine as the server connecting to the client on the remote machine), depending on which client exceeded the "port" threshold to trigger the new connection.


This client will act as the new "server" to the other client
VM1 (192.168.122.101)
CLIENT#
SERVER
15101
1228
socket
8001
group01@apollo.cse.unt.edu
socket
socket
VM2 (192.168.122.117)
No Port Assigned
15117
CLIENT#
This client will connect to the originating client (i.e., new server)

After connecting with VPN, each group will connect to the **apollo** server address, `groupxx@pollo.cse.unt.edu`, using their assigned group number and ports, which will be port forwarded to their specific VM. Then, once on the respective machine, you will invoke your programs for the server and clients in the following manner (using the above diagram as reference):

SERVER: (on A01)

`./server <svr_port>`

For example, `./server 8001`.

CLIENT 1: (on A01)

`./client <svr_host> <svr_port> <rem_ipaddr>`

For example, `./client A01 8001 192.168.122.117`.

CLIENT 2: (on A17)

`./client <svr_host> <svr_port> <rem_ipaddr>`

For example, `./client A01 8001 192.168.122.101`.

See the *Group Credentials* file associated with this project for specific VM and login information.

SAMPLE OUTPUT (user input shown in **bold**):

==> SERVER on A33

```
group33@A33:~$ ./svrMajor2
usage: ./svrMajor2 <svr_port>
group33@A33:~$ ./svrMajor2 9000
Waiting for Incoming Connections...
Client Connection Accepted
Client Handler Assigned
Client Connection Accepted
Client Handler Assigned
CLIENT 1:   200 - Total: 200
CLIENT 2:     5 - Total: 5
CLIENT 2:     5 - Total: 10
CLIENT 2:     5 - Total: 15
CLIENT 2:     5 - Total: 20
CLIENT 1:   400 - Total: 600
CLIENT 1:   800 - Total: 1400
Sending Client 1 Port Data to Client 2, Reset Total
CLIENT 2:     5 - Total: 25
Client 2 Disconnected, Reset Total
CLIENT 1:    50 - Total: 50
CLIENT 1:   100 - Total: 150
CLIENT 1:   200 - Total: 350
CLIENT 1:   400 - Total: 750
CLIENT 1:   800 - Total: 1550
Sending Client 1 Port Data to Client 2, Reset Total
Other Client Unavailable, Unable Send Port Data
CLIENT 1:   100 - Total: 100
Client 1 Disconnected, Reset Total
Client Connection Accepted
Client Handler Assigned
CLIENT 1:   800 - Total: 800
Client Connection Accepted
Client Handler Assigned
CLIENT 2:     5 - Total: 5
CLIENT 2:    10 - Total: 15
CLIENT 2:    15 - Total: 30
CLIENT 1:  2000 - Total: 2800
Sending Client 1 Port Data to Client 2, Reset Total
CLIENT 2:    30 - Total: 60
Client 2 Disconnected, Reset Total
CLIENT 1:   100 - Total: 100
Client 1 Disconnected, Reset Total
^C
```

==> CLIENT on A33

```
group33@A33:~$ ./cliMajor2 A33
usage: ./cliMajor2 <svr_host> <svr_port> <rem_ipaddr>
group33@A33:~$ ./cliMajor2 A33 9000 192.168.122.101
Connected
Enter CLIENT 2 Data: 5
SERVER Total: 5
Enter CLIENT 2 Data: 5
SERVER Total: 10
Enter CLIENT 2 Data: 5
SERVER Total: 15
Enter CLIENT 2 Data: 5
SERVER Total: 20
```

```
Enter CLIENT 2 Data: 5
SERVER Message: PORT 1400
SERVER Total: 25
CLIENT 2 Disconnecting...
Sending 25 to Other Client
CLIENT 2 Disconnecting...
group33@A33:~$ ./cliMajor2 A33 9000 192.168.122.101
Connected
Enter CLIENT 1 Data: 800
SERVER Total: 800
Enter CLIENT 1 Data: 2000
SERVER Total: 2800
Enter CLIENT 1 Data: Connection Accepted
Handler Assigned
Received 60 from Other Client
Other Client Disconnected
100
SERVER Total: 100
Enter CLIENT 1 Data: 0
CLIENT 1 Disconnecting...
```

==> CLIENT on A01

```
group33@A01:~$ ./cliMajor2
usage: ./cliMajor2 <svr_host> <svr_port> <rem_ipaddr>
group33@A01:~$ ./cliMajor2 A33 9000
usage: ./cliMajor2 <svr_host> <svr_port> <rem_ipaddr>
group33@A01:~$ ./cliMajor2 A33 9000 192.168.122.133
Connected
Enter CLIENT 1 Data: 200
SERVER Total: 200
Enter CLIENT 1 Data: 400
SERVER Total: 600
Enter CLIENT 1 Data: 800
SERVER Total: 1400
Enter CLIENT 1 Data: Connection Accepted
Handler Assigned
Received 25 from Other Client
Other Client Disconnected
50
SERVER Total: 50
Enter CLIENT 1 Data: 100
SERVER Total: 150
Enter CLIENT 1 Data: 200
SERVER Total: 350
Enter CLIENT 1 Data: 400
SERVER Total: 750
Enter CLIENT 1 Data: 800
SERVER Total: 1550
Enter CLIENT 1 Data: 100
SERVER Total: 100
Enter CLIENT 1 Data: 0
```

```
CLIENT 1 Disconnecting...
group33@A01:~$ ./cliMajor2 A33 9000 192.168.122.133
Connected
Enter CLIENT 2 Data: 5
SERVER Total: 5
Enter CLIENT 2 Data: 10
SERVER Total: 15
Enter CLIENT 2 Data: 15
SERVER Total: 30
Enter CLIENT 2 Data: 30
SERVER Message: PORT 2800
SERVER Total: 60
CLIENT 2 Disconnecting...
Sending 60 to Other Client
CLIENT 2 Disconnecting...
```

SAMPLE OUTPUT 2 (user input shown in **bold**):

The following SAMPLE OUTPUT shows what happens when more than 2 clients attempt to connect to the server at the same time.

==> SERVER on A33

```
group33@A33:~$ ./svrMajor2 9001
Waiting for Incoming Connections...
Client Connection Accepted
Client Handler Assigned
Client Connection Accepted
Client Handler Assigned
error: too many clients connected
CLIENT 2:   100 - Total: 100
CLIENT 1:     5 - Total: 5
Client 1 Disconnected, Reset Total
Client 2 Disconnected, Reset Total
^C
```

==> CLIENT on A33

```
group33@A33:~$ ./cliMajor2 A33 9001 192.168.122.101
Connected
Enter CLIENT 1 Data: 5
SERVER Total: 5
Enter CLIENT 1 Data: 0
CLIENT 1 Disconnecting...
```

==> CLIENT on A01

```
group33@A01:~$ ./cliMajor2 A33 9001 192.168.122.133
Connected
Enter CLIENT 2 Data: 100
SERVER Total: 100
Enter CLIENT 2 Data: 0
```

```
CLIENT 2 Disconnecting...
```

==> CLIENT on A01 (this is the third client attempting to connect)

```
group33@A01:~$ ./cliMajor2 A33 9001 192.168.122.133
SERVER: Too Many Clients Connected. Disconnecting...
```

REQUIREMENTS:

This assignment must be submitted via Blackboard with the following elements:

- Your two C program file named "**`svrMajor2.c`**" and "**`cliMajor2.c`**", without the quotes, for the server and client code, respectively. Your code should be well documented in terms of comments. For example, good comments in general consist of a header (with your name, course section, date, and brief description), comments for each variable, and commented blocks of code.
- A **`README`** file with some basic documentation about your code. This file should contain the following four components:
  - Your name(s).
  - Organization of the Project. Since there are multiple components in this project, you will describe how the work was organized and managed, including which team members were responsible for what components – there are different ways to do this, so your team needs to come up with the best way that works based on your team's strengths. Note that this may be used in assessment of grades for this project. Although the scope of this project is smaller than Major 1, there are still several key components, such as how the server can be responding to a client's requests (if only 1 client is connected), but also be listening for another client to connect, that needs to be considered in the design.
  - Design Overview: A few paragraphs describing the overall structure of your code and any important structures.
  - Complete Specification: Describe how you handled any ambiguities in the specification.
  - Known Bugs or Problems: A list of any features that you did not implement or that you know are not working correctly.
- A completed group assessment evaluation (given at a later date) for each team member.
- A `Makefile` for compiling your source code, including a clean directive.
- Your program will be graded based largely on whether it works correctly on the **`apollo`** machine (actually, your assigned VMs), so you should make sure that your program compiles and runs on this machine.

SUBMISSION:

- You will electronically submit all required components to the **Major Assignment 2** dropbox in Blackboard by the due date. Group submission will be enabled on Blackboard so that only one team member needs to submit the assignment.