Assignment 2

240-426

Unix Network Programming

Write a pair of programs, a server and a client. They should communicate using two TCP (SOCK_STREAM) sockets.

Use IPv6 if IPv6 is available, otherwise use IPv4. The programs should dynamically determine which protocol to use, this should not be a compile time decision.

The server should accept connections upon a well known port. That is, you choose a port number, the server waits for connections upon that port. The client simply *knows* the port number it should connect to. That is, this port number can be compiled into both the server and the client.

Upon receiving a connection, the server should await and execute commands as sent to it by the client.

Before each command, one newline character (\n or decimal 10, 0xA) will be transmitted as TCP **Urgent** data. Detection of that command marker will cause the server to immediately terminate its previous command, whether complete or not, and receive and obey the following command. That is, as soon as the server notices that urgent data is coming, it should abandon its current command (if any) and obtain and commence the following command as soon as possible.

There are four (4) defined commands. Each is transmitted as a single line of text, terminated by a newline character (\n, decimal 10, 0xA). Commands may require additional data, which will follow the newline that terminates the command, and continue until (but not including) the **Urgent** newline character that precedes the next command.

The four commands are

- **CONNECT** *nnnn*

  This will cause the server to establish a new connection (secondary connection) to port *nnnn* at the client which sent the command. Only one such additional connection is permitted at a time – a CONNECT command while a secondary connection already exists will cause the previous connection to be closed before the new connection is opened. There is no additional data for this command, and no response to the client, other than that the new connection is created. If *nnnn* is zero (has a zero value), then no new connection is created, but any existing connection is still closed.

- **LOOP**

  This will cause the server to take subsequent data from the primary connection, until a new command is detected, and send it to the client. If a secondary connection exists, send the data to the client over the secondary connection. Otherwise, if there is no existing secondary connection, send the data to the client over the primary connection.

- **FILTER**

  If no secondary connection exists, this command does nothing. If there is a secondary connection, This command causes the server to read data from the secondary connection. The server should also read patterns, one at a time, as they arrive, from the primary connection. A pattern is simply a line of text. The most recent pattern read by the server is its current pattern, previous patterns are forgotten. A pattern is considered read only when the newline character that terminates it has been received, the newline character is not part of the pattern. Until then, the previous pattern continues to be used, if there is one.

Data from the secondary connection is compared to the pattern, line by line (where each line is terminated by newline character, which remains part of the data line). If the pattern string is present somewhere in the data line, the data line is transmitted back to the client. If the pattern does not appear, the data line from the client secondary connection is ignored, and the next line obtained. If there is no data pattern, it should be assumed to not match.

The library function **strstr** *can be used to determine if the pattern occurs in a data line, there is no need to write your own code to do this matching.*

This continues until a new command is received.

- **EXIT**

  The server will terminate the secondary connection, if one exists, then terminate the primary connection, and then exit. There is no additional data for this command.

Data on the primary connection which is not a command, and not being used as data to an existing command, should simply be ignored.

The client process should connect to the server (the server host name should be a parameter to the client program), using IPv6 if possible, or IPv4 if the server has no IPv6 addresses, or IPv6 fails. This should be a primary connection to the server.

The client process should then read from its standard input. It should also be prepared to handle interrupt and quit signals (SIGINT and SIGQUIT).

When it receives a SIGQUIT it should send an EXIT command to the server (sending the **Urgent** newline character first of course), wait for any existing connections to terminate, and then exit itself.

When it receives a SIGINT, it should delete any pending data intended for the server that had not yet been transmitted. The client should then pick one of the three commands the server understands (one of the four, excluding exit) send that command to the server, and inform the user (via standard output) which command has been selected.

If the command chosen was **CONNECT** the client should establish a socket, bind it to a local address, and transmit the port number used to the server. It should then wait for the incoming connection, and inform the user once that has been established. Approximately 1 in each 5 **CONNECT** commands transmitted should be sent as **CONNECT 0**, in which case the client should not establish a new socket.

Note that sending a **CONNECT** command from the client should not cause an existing secondary connection to be closed, the server should close the connection when the **CONNECT** command is processed there. Until then, the client should continue to process the existing connection.

If the command chosen was **FILTER**, and there is an existing secondary connection, the client should open a data file (the name of the file to open can be compiled into the program, or be passed to the client as a command line parameter). The file should then be transmitted to the server over the secondary connection, but no more than one data line a second should be sent (that is, transmit one line, then send no more for at least one second, then transmit the next line). If the end of file on the data file is reached before the command is terminated (by the next received SIGINT) rewind (or close and open again) the file, and transmit it all again.

If there is no existing secondary connection, the client should simply transmit the **FILTER** command.

Any data received from the server over the primary connection should be transmitted to the user, with each line preceded by the two characters **P>**

Any data received from the server over the secondary connection should be transmitted to the user, with each line preceded by the two characters **S>**

Any data entered by the user (via standard input) should be transmitted to the server over the primary connection.

Be sure to allow for all combinations of timing of data on the various connections, signals, etc. All I/O should take care not to block any other operations the process may need to perform (or not for long enough to be noticed).

Assignment due by March 1, 2004 (2547).

Do write beautiful code – nice indentation, consistent style, intelligent use of comments, ...

Send all files via e-mail (in a tar file, or as individual attachments) to *kre@munnari.OZ.AU* (or *kre@five-dots.coe.psu.ac.th*).

Each student's work is expected to be their own - you can seek help from friends, but not copies of their code, nor ask them to write code for you. That is, you can seek to learn how to do something that is causing you problems - but you must then actually do it yourself.