Exercise 1 - Socket Programming

CMPE-570/670 Data & Computer Communications – 2191 Fall

Due Date: 11/11/2019

Description

In this exercise you are asked to design and develop a custom concurrent FTP Server named TigerS and its associated FTP client named TigerC. Once TigerS starts on a host, it will listen for incoming requests from multiple TigerC clients. You are requested to implement three basic commands for TigerS and TigerC:

1. **tconnect <TigerS IP Address> <User> <Password>**: This command takes the IP address of the TigerS, and the client user name and a password as parameters. If the provided user name and password are correct, then the client is allowed to connect TigerS, otherwise an error message is returned.
2. **tget <File Name>:** The command expects the name of the file to download from TigerS as parameter. If the file exists, it is downloaded from the server, otherwise an error message is displayed.
3. **tput <File Name>**: The tput command takes the name of the file to upload and uploads it to the TigerS host.

What to submit?

Please prepare a professionally written report that will include the following, and submit it to "mycourses assignments – SocketProgramming" together with your implementation:

1. How did you implement concurrency? Explain and show your work.
2. Show that your server can serve at least 100 TigerC clients concurrently and provide a sample run.
3. What is the maximum number of clients that your server can serve concurrently, when both TigerS and TigerC is run on your laptop or PC? Justify your answer.
4. Provide a sample run to demonstrate the execution of each command listed above. Your program is expected to inform the user with either an error or status message after each command.

Remarks

- Please articulate any assumption you made for this assignment if necessary. You may use C++ or C for your implementation.
- You are expected to use POSIX compliant sockets and threading libraries in your preferred language. You are not allowed to use a higher level (advanced) framework or sockets library.
- For the tconnect command, you may create a file to store the authorized users and passwords. Also you may use default user directories on TigerS (and TigerC) machines, for the tget and tput commands.

Supporting Materials

In order to be successful with this project, several topics must be researched, and their underlying concepts understood. This can be done at the student's leisure, but the time constraint of the project should be kept in mind. Leave enough time for implementation and test, as use of a new API is a learning experience and should be treated as such.

POSIX Compliance

To learn more about POSIX compliant functions for use in the application, refer to https://pubs.opengroup.org/onlinepubs/9699919799.2018edition/ . When a specific function or structure is required, search for the applicable term using the search in the upper left of the page and narrow in on the correct one.

For example, the sockets constructor can be found by searching for the keyword "socket". This search returns several pages, one of them pointing out the header at "sys/socket.h", and another the socket constructor function along with the arguments in its signature.

This type of research may be necessary to discover critical functionality for use in the project. Couple it with the example of sockets communication given below to achieve the functionality required.

Sockets Basics

- socket – Constructor takes params socket family (AF_INET), socket type (SOCK_STREAM) and a protocol descriptor. It returns an integer, which is either an integral socket ID or -1 indicating an invalid socket.
- struct sockaddr – Data members include sa_len a uint 8_t, sa_family_t sa_family and a 14 byte char array sa_data. Can be used to typecast specific sockaddr structures (_in, _in6) to more general structures, and memcpy is used to create the more specific structures from it.
- struct sockaddr_in – Data members include socket family (AF_INET), address (a 32-bit IPv4 address in network byte order) and port for the server (a 16-bit TCP or UDP port number network byte ordered).
- bind – Takes a socket ID, a pointer to a sockaddr struct and the size of the struct (using sizeof) and returns a 0 on success or -1 on error. Assigns a local protocol address to a socket.
- listen – Takes a socket ID, and a blacklog argument (queue of completed connections). Called by a TCP server, and converts an active unconnected socket into a passive socket, indicating the kernel should accept incoming connection requests directed to the socket.
- accept – Takes a socket ID, a pointer to a sockaddr struct, and the size of the struct. Returns a non-negative fd if OK, otherwise -1. Called by a TCP server to return the next completed connection from the front of the completed connection queue. If queue empty, process sleeps.
- close – The normal Unix close function is also used to close a socket and terminate a TCP connection. It takes a socket ID and returns a 0 on success or a -1 on error.
- write – Takes an fd, a buffer and max buffer size. Writes data to the socket from the buffer.

- connect – The connect function is used by a TCP client to establish a connection with a TCP server. It takes a socket ID, pointer to a sockaddr struct and size of the struct. It returns 0 on OK and -1 on error.
- read – Takes an fd, buffer address and max buffer size. Reads data from the socket and copies it to the buffer.

Concurrency Basics

Fork is the basic function used to spawn child processes from the main server process. This will create N number of child processes off the main, that will each serve a separate client. The function itself returns a process ID (pid).

The difference between an iterative and a concurrent server is that one will finish servicing the current client prior to taking the next client request. This is not true concurrency. See the example below to get a feel for how to use "fork" effectively to create a server that can handle the 100 clients that will be accessing it, but keep in mind that "fork" will only be a stepping stone to understanding true concurrency. The pthread library should be used in its stead to create lightweight child processes and manipulate them effectively.

Requirements Development

This exercise requires you to create a custom concurrent FTP Server and associated Client application. Furthermore, it needs to support three basic commands. Immediately, it should be evident that two sets of requirements are needed, one for the server, another for the Client. The two sets of requirements should be described using the identifiers SRV and CLN.

For the Server, we can immediately identify three basic requirements of the system. The Server is an IPv4 based service, using POSIX based sockets for portability; and thus, we can derive the following three system requirements:

Identifier	Requirement	Req/D.O.
SRV-1	The Server shall utilize an Internet Protocol based socket for inter-process communications.	Requirement
SRV-2	The Server shall listen for incoming connection requests.	Requirement
SRV-3	The Server shall allow for at least one active client connection at one time.	Requirement

The Client must compliment the Server, for which the following requirements can be derived:

Identifier	Requirement	Req/D.O.
CLN-1	The Client shall provide an ASCII based Command Line Interface to the user.	Requirement
CLN-2	The Client shall support an IP based connection to the Server.	Requirement
CLN-3	The Client shall support the uploading of local system files to the Server.	Requirement
CLN-4	The Client shall support the downloading of remote system files from the server to the local host.	Requirement

Reviewing the three commands listed (tconnect, tget, and tput), additional features can be described for both the Client and the Server. The Client requirements are pulled from the stated messages and formats:

Identifier	Requirement	Req/D.O.
CLN-5	The Client shall support a command <tconnect> for establishing a communications channel with the Server.	Requirement
CLN-6	The Client shall support a user-defined address for the peer Server in the tconnect command.	Requirement
CLN-7	The Client shall support a user-entered "identity" for Server access control via the tconnect command.	Requirement

Identifier	Requirement	Req/D.O.
CLN-8	The Client shall support a user-entered "password" for Server based authentication via the tconnect command.	Requirement
CLN-9	The Client shall support an authentication request primitive for transmission to the Server.	Requirement
CLN-10	The Client shall support the transmission of local files to the Server.	Requirement
CLN-11	The Client shall support the reception of remote files from the Server.	Requirement
CLN-12	The Client shall support a command <tget> via the CLI to initiate a download transaction with the Server.	Requirement
CLN-13	The <tget> command shall accept a file name parameter as the name of the remote file to download.	Requirement
CLN-14	The Client shall support a download request primitive for transmission to the Server.	Requirement
CLN-15	The Client shall support a command <tput> via the CLI to initiate an upload transaction with the Server.	Requirement
CLN-16	The <tput> command shall accept a file name parameter as the name of the local file to upload.	Requirement
CLN-17	The Client shall support an upload request primitive for transmission to the Server.	Requirement

The Client should also have requirements pertaining to error condition handling. For example, actions the Client must take when the Server reports a failed authentication, or if an incorrect number of arguments are provided when invoking the Client. Describing these will be left as an exercise to the student.

Server requirements for the session and authentication layer must support login, and both uploading and downloading of files. They should read as:

Identifier	Requirement	Req/D.O.
SRV-4	The Server shall support reception of an authentication primitive from a Client.	Requirement
SRV-5	The Server shall associate an authenticated session with a unique Client connection.	Requirement
SRV-6	The Server shall support an authentication acknowledgement primitive for transmission to a Client with valid credentials.	Requirement
SRV-7	The Server shall support an authentication negative acknowledgement primitive for transmission to a Client with invalid credentials.	Requirement
SRV-8	The Server shall support the transmission of local files to the Client.	Requirement
SRV-9	The Server shall support the reception of remote files from the Client.	Requirement
SRV-10	The Client shall support a command <tget> via the CLI to initiate a download transaction with the Server.	Requirement
SRV-11	The <tget> command shall accept a file name parameter as the name of the remote file to download.	Requirement
SRV-12	The Client shall support a command <tput> via the CLI to initiate an upload transaction with the Server.	Requirement
SRV-13	The <tput> command shall accept a file name parameter as the name of the local file to upload.	Requirement
SRV-14	The Server shall support a file download message primitive from a Client.	Requirement
SRV-15	The Server shall support a download file positive acknowledgement primitive for transmission to a Client.	Requirement
SRV-16	The Server shall support a download file negative acknowledgement primitive for transmission to a Client.	Requirement
SRV-17	The Server shall support a file upload message primitive from a Client.	Requirement
SRV-18	The Server shall support an upload file positive acknowledgement primitive for transmission to a Client.	Requirement
SRV-19	The Server shall support an upload file negative acknowledgement primitive for transmission to a Client.	Requirement

Similar to the Client, the Server must also have error condition requirements. Lastly, the design objective provided involves use of a password database for Client authentication. This database can be as simple as a file containing hard coded credentials:

Identifier	Requirement	Req/D.O.
SRV-20	The Server shall a User-Password database for Client authentication processing.	Design Objective

Command Line Interface Concepts

Requirement CLN-1 states that a CLI for the Client must be created. As review, CLIs parse three types of parameters: Arguments, Options and Flags. For this assignment, there is no need to parse anything but arguments. The CLI can consist entirely of simple string parsing of arguments.

Be advised that a common issue when parsing command line arguments is the failure to use a POSIX compliant stream I/O function that can detect a newline delimiter. It is suggested that the student utilize fgetc(), getline(), or other POSIX compliant functions.

The following *Here String* method of launching 100 clients should be utilized to verify program functionality. With small files, it should be noted that file transfers will occur very quickly, so to ensure concurrency of operation, files should be very large (100's of MB). Large file transfer will allow for overlap of operations on most current day OS's. ***The following bash script will be run by the instructor when validating programs after submission:***

```
#!/usr/bin/bash

n=1
last=100
while [ $n -lt $last]
do
  ./TigerC <<< "tconnect 127.0.0.1 user pass
  tget down$n.txt
  tput upload$n.txt
  exit" &
  n=$(($n+1))
done
```

Transport Layer

This was not specified in the project requirements at a lower level, so either TCP or UDP can be selected. Even though SRV-2 states that the server will listen for connection requests, this is a high-level requirement, and not explicit enough to map to a TCP or UDP implementation, however, authentication and reliable file transfer should guide the engineer to a specific transport protocol, and a method to associate authenticated clients with a session.

Error Correction

Error correction is a requirement throughout this exchange process, and it is essential to check every function call for an error return. POSIX compliant functions such as *socket, inet_pton, connect, read,* and *fputs* can cause the global "errno" variable to change, representing the type of error that occurred with a positive integer. The "errno" global is returned as the return value of the function in multithreaded applications, as opposed to being changed as a global value and accessed directly after a function call.

Server Client Basics

A Client generally functions to accept and parse command line arguments, create a socket, specify the server's IP address and port, establish connection with the server, read and display the server's reply, and terminate the grogram.

A Server also creates a socket, but then binds the Server's well-known port to the socket, converts the socket to a listening socket, accepts the client connection, sending a reply, and terminate the connection.

Sockets Helper Functions

- int inet_pton(int family, const char* strptr, void* addrptr); – Converts from dotted-decimal representation of IPv4 address to 32-bit binary (in_addr).
- const char* inet_ntop(int family, const void* addrptr, char* strptr, size_t len); – Converts from in_addr binary representation to dotted-decimal address notation.
- void bzero(void* dest, size_t nbytes); - Sets the specified number of bytes to 0 in the dest.
- uint16_t htons(uint16_t host16bitvalue);
- uint32_t htonl(uint32_t host32bitvalue); Both return: value in network byte order
- uint16_t ntohs(uint16_t host16bitvalue);
- uint16_t ntohl(uint32_t host32bitvalue); Both return: value in host byte order

Example Implementations

- See notes

Grading Rubric

Item	**Weight**
Build and Launch scripts are included and work	10
Server (standalone daemon) functions	10
Server supports single client	
- Supports user login	10
- Supports error free PUT and GET commands	10
- Supports error free transfer of binary or ASCII files	10
Server supports (up to 100) clients simultaneously	10
Basic socket functionality (TCP or UDP)	10
Sockets detection and error handling	5
Report as defined in the assignment	25
TOTAL	100