CMSC 417 Computer Networks **Spring 2010**

Programming Assignment 1

Assigned: February 6 *Due: Feb. 13, 11:59:59 PM.*

1 Background

The example server in Listing 1 is a server that handles one client at a time. If other clients connect to this server while it is serving someone else, they will have to wait for the server to finish with the current client. This type of server is called iterative. Iterative servers are able to handle one client at a time.

```
/*
 *  An example of a simple TCP server.
 *  It runs forever and does the following:
 *    1. sets up a listening socket for the new client
 *    2. Exchanges messages with client
 *    3. Terminates the connection at the end of the
 *        conversation
 */

int processClient( int clnt_socket )
{
  /*
   * Receive Client data
   */
  close(clnt_socket);
}

int main()
{
  /*
   *  We have to call bind(...) to bind socket with the
   *  local address, and then we have to call listen(...)
   *  ino order to listen for incoming client connections
   */
  for ( ;; )  /* Run Forever */
    {
      int clnt_len = sizeof(Triple_clnt);
      if ((clntSock = accept( servSock, (struct sockaddr *) &Triple_Addr,
                              &clnt_len)) < 0 )
        {
          /* Exit with Error: Accept Failed */
        }
      /* At this point clntSock is connected to a client */
        processClient( clntSock );
    }
  return 0;
}
```

Unix is a multitasking system and gives the ability to the network programmer to use tools such as processes and threads to allow a server to handle more than one client at a time. This family of servers is called concurrent.

The first way to implement a concurrent server is by using processes. The scheme is quite simple. The server accepts clients connections and then with the help of *fork(2)* creates new processes that deal with each of the clients. When the programmer calls *fork()* it creates a new identical process. The new process is called the child process and the process that initially invokes *fork()* it is called the parent process. Upon successful completion, *fork()* returns a value of 0 to the child process and the pid_t PID of the child to the parent process. In case of an error a value of -1 is returned to the parent and no child process is created. For more information about *fork* you can look the man pages. The book TCP/IP Sockets in C: A Practical Guide for Programmers presents an example of a forking server.

The second way to implement a concurrent server is by using threads. Creating a new process upon each client arrival is very costly. The operation system, has to duplicate the entire state of the parent process. Instead of paying this cost we can use threads, which decrease this cost by permitting concurrency inside the same process. However, in this case we should be extra carefull cause the newly created thread shares the same memory address space with its parent. On one hand, this reduces the cost but on the other hand could create race conditions between the parent process and the existing threads when they access common variables. You can find information about threads to the man pages as well as to the TCP/IP Sockets book where you can find an implementation of a TCP server with threads.

Lastly, the third way to implement a network server is by multiplexing. The implementation schemes we have seen so far have the main server to block on the *accept()* call, waiting for a new client connection. Instead of having a process or thread per request, the server can be implemented as a single process that "multiplexes" the incoming requests by handling any messages that may be waiting at a socket descriptor. Unix's *select()* function checks for pending I/O operations on a list of given file/socket descriptors and waits until a descriptor becomes available. When this happens, *select()* returns the available descriptor. *select()*-based servers are very efficient because they do not incur process- or thread-creation overhead and the programmer does not have to deal with synchronization and race condition issues. Your TCP/IP Sockets books presents a useful example of a select based server.

2 Description

In this assignment you will write four different implementations of the server program which will communicate using sockets with the client you implemented in the previous assignment. The four different implementations include the iterative, the forking based, the thread based and, the select based sever implementations. Try to make your sending/receiving functions as well as the message processing ones, as modular as possible in order not to have to implement them each time. We will provide you in a seperate file a list with valid and invalid Pythagorean triples from which your server will send randomly selected triple to the client. The implementation of the select based server should listen to at least 2 ports in order to understand and exploit the merits of this approach.

Correct thread implemenation will take extra credit.

Note that you cannot bind to a port below 1024 withoug having superuser (root) access. Thus, you are not allowed to use this port range in the linux-lab.

3 The Server Program

The command line argument for the server is given below. The server will take the port as an argument. However, for the case of the select based implementation you have to provide more command line arguments due to the fact that your server will listen to multiple ports. Your command line will look like this

```
server <port_A> [<port_B> ... <port_N>]
```

4 Requirments

The Pythogorean triple protocol described at the previous assignment. Your server is responsible for evaluating the messages that clients send. If a message doesn't follow the described protocol, the server should terminate the connection silently. Your server, should printing the messages that sends and receives. Also for any of the successful sessions should print the university ID that the client sends. Please, remember that a session is successful if and only if the client exchanges all the messages of the protocol and answers correctly about the validity of the given triple.

Please, be sure to comment out or even better clear your code for any debugging statements before you submit this assignment.

Expect that your implementations will be tested from scripts that will run simulataneus clients issuing multiple requests to your servers. Thus, do not test your implemention for the case of one client only.