Programming Assignment #1

ELEN E6761 - Computer Communication Networks — Due 9/12
Fall 2000 — Prof. Rubenstein

This programming assignment will cover some of the basics of socket programming, as well as give you a little bit of understanding of how web browsers interface with networks at the socket level. To do this assignment, you will need access to a web browser that allows configuration of a *proxy*. For instance, using Netscape, you can configure the proxy by clicking the following links Edit → Preferences → (Little arrow next to Advanced) → Proxies. Choose manual proxy configuration and then click the View button. You will want to change the HTTP Proxy (left column is address, right column is port). The address should match the IP address of the machine you are going to run your proxy on (echo $HOST to see the name). You can choose any port you want, but avoid well-known port numbers (see RFC 1700 for a list, or just choose something in the range of 10000-65535).

You should submit copies of your code, and the output generated. These submissions should be performed on-line. Please check the course web page for details. The answers to the questions can be submitted on-line or at the start of class. You should submit your solutions by the start of class (4:10 EST, 9/12).

Part 1

You will design a simple proxy server. Your server should expect to receive an HTTP request. It should then transmit a response back to the location from where the request came. The response should include some information of the request, as well as some indication that it passed through the proxy. For example, one obvious (and easy) solution would be to return a text message saying, "The proxy received the following text: XXX", where XXX is the text of the message received. If this is done correctly, your web browser will display the raw text received. If you want to do something more creative (e.g., if you know some more about HTTP and actually want to return a web page of some sort), please do so.

Your proxy needs to perform the following steps:

1. Listen on a given port (the one you told your web browser to forward requests to) for a request. HTTP requests are issued using the TCP protocol (i.e., the request expects to reach a socket with its protocol option set to SOCK_STREAM). This means that communication is connection-oriented.
   (a) create a socket of type SOCK_STREAM
   (b) bind the socket to the port that you configured the web browser to forward its requests to
   (c) listen for an incoming request
2. Accept an incoming connection when it arrives onto a socket (recall from class that the connection is established on a new socket).
3. Receive the data being transmitted off the socket.
4. Compute a response
5. Send the response out on the same socket (we are taking advantage of the duplex capabilities of the socket and the bi-directional nature of the connection).
6. Close the socket that performed the data transfer. Either close the other socket as well, or have the program loop so that the proxy will accept other requests
7. Run your proxy and initiate a request in your browser (configured to use the proxy) for some web page.
8. When you are done with your assignment, don't forget to reconfigure your web browser so that it no longer goes through the proxy or you will have trouble accessing the web.

Part 2

In Part 1, the socket provided connection-oriented, reliable delivery of data. In this next part, we look at some of the challenges if we simulate raw IP delivery (using the protocol option set to SOCK_DGRAM). To do this part, you can download the software that will send an HTTP request, and receive and display your proxy's response. The code is available for download from the materials/handouts web-page for the course.

The code should compile cleanly using `gcc` on linux and Solaris platforms (on linux, `gcc 6761-http-udp-client.c -o 6761-client` should work, on Solaris, use `gcc 6761-http-udp-client.c -o 6761-client -lnsl -lsocket`). To execute the code, type `6761-client proxy-addr portnum` where `host-addr` and `portnum` are respectively the address and port that your proxy will be installed on. The code sends an HTTP request via UDP to the indicated address/port about once a second.

In this part, you should design a proxy that receives the client's UDP request and returns information to the client that is similar to what you returned in Part 1. Because the communication is performed via the UDP protocol (SOCK_DGRAM) and is connectionless, the procedure differs slightly:

1. Set up a port to receive client datagram (UDP) requests.
   (a) create a socket
   (b) bind the socket to the port the proxy will forward requests to
   (c) (Note: there is no listen step due to the connectionless nature)
2. Receive a packet off the socket, and obtain the address from which the packet came (i.e., use `recvfrom`)[1]
3. Compute a response
4. Send the response out on the same socket (again we are taking advantage of the bi-directional nature of the socket).
5. Close the socket.
6. Terminate the client program (hit `ctrl-c`).

Questions

Please provide a short (1 or 2 line) response to the following questions (10% of the grade)

1. Why does the `accept` call create a new socket rather than continue using the old one?
2. Bearing in mind your answer to question 1, describe how your code would have to be modified to take advantage of the fact that a new socket was created.
3. You are told to perform a bind (step 1(b)) in both Parts 1 and 2. Was the bind necessary in either of the parts? Why or why not?
4. In Part 2, your proxy's transmission to the sender is unreliable (socket type SOCK_DGRAM does not guarantee delivery of the packet). Given the current behavior of the client (it keeps requesting the same information), does your code need to be modified to guarantee that each object requested is eventually delivered? Why or why not? What if the client changed what it was requesting every 5 requests? What if it went back and forth between two different requests?
5. Suppose your proxy is required to periodically pre-fetch a web-page (i.e., obtain the web page even though no client requested it). Assume the routine `fetch_web_page` is provided to you. Give a rough sketch of how you would modify your code. (Hint: are your sockets blocking?)

[1]IMPORTANT: even though the last parameter, `fromlen`, is passed in as a pointer, it is a value/result parameter, and you must set its value properly before making the call (i.e., use `sizeof(struct sockaddr_in)`). A common bug is to forget to set this value, and sometimes the program behaves unpredictably as a result.