LAB #2: CLIENT-SERVER WITH MINIMAL SECURITY FEATURES
DUE DATE: March 3, 10:00 a.m.

TO BE DONE IN GROUPS OF TWO (MANDATORY)

1. General Description

In this laboratory, you will implement a simple UDP-based client-server system with limited security features. This laboratory has two objectives: (i) help you get familiarized with UNIX socket programming; (ii) prepare you to tackle the next laboratory assignment which involves the design of a more secure client server system.

2. File with sample binary files

We are providing the binary files of our implementation of the client and server. These files will be used as a reference to test your client and your server code. Two versions of the binaries are provided, one for SunOS and another for Linux. You need to ensure that your client and server programs work with both our Sun and Linux client and server implementations. Please download the file lab2.tar.gz from http://shay.ecn.purdue.edu/~ece495f/labs/lab2/lab2.tar.gz

3. Server – Client application:

You will implement a simple UDP client – server application, where a server transmits a file to a client after the client successfully logs in by using a password.

Requirements:
There are two pieces of code you have to implement in two different files -- a client and a server. Below is the protocol specification which will give you the details you need to implement these two programs. Also, we are giving you the packet format, which specifies the content of the messages the client and server will exchange. Your implementation must work correctly even when the client and server run on architectures with different endian formats.

Protocol Specification:

a. The client sends a JOIN_REQ packet to initiate communication with the server.
b. The server responds with a PASS_REQ packet, which is a password request to the user.
c. The client will send a PASS_RESP packet to the server which includes the password
d. The server will verify the password and in case the password is correct, the server will send a PASS_ACCEPT packet to the client.
e. In case the password is incorrect, the server sends a PASS_REQ packet again to the client. The PASS_REQ packet will be retransmitted at most three times.

After the third time, the server sends a REJECT message to the client. The client closes the session, and the server exits as well.

f. Once the server transmits the PASS_ACCEPT packet to the client, the server begins transmitting the file using DATA packets. The file is broken into several segments (depending on the size of the file), and each segment is transmitted using a DATA packet.
g. When the server completes sending the file, it will transmit a TERMINATE packet which marks the end of the file download. Included in this packet, there is a file digest (SHA1 digest) that the client will use to verify the integrity of the received file.

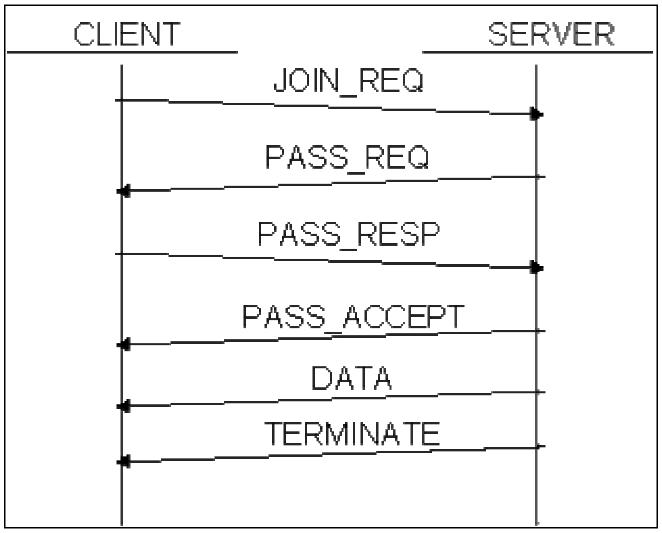

Figure #1: Functional Specification

Assumptions

You may make the following assumptions to simplify the design:

a. The server handles only one client at a time. There is no need to address issues associated with supporting multiple simultaneous clients. You will not need to use select/threads in this assignment.
b. Dealing with losses: We are using UDP-based data communication. While packet losses are possible with UDP, they are rare in a LAN environment such as the one where your code will be running on, and you will in all likelihood not encounter packet loss. There is no need for your code to implement any mechanism such as ACK/retransmission for reliable data delivery. However, in the event that a packet loss does occur, this will lead to gaps in data sequence numbers, or unexpected packets being received. In such cases, your code should exit gracefully.

Packet Formats

The picture shows the formats of the various packets. Some notes:

a. All packets have a 2 byte packet type, and 4 byte payload length.
b. The packet types are as follows: JOIN_REQ: 1, PASS_REQ: 2, PASS_RESP: 3, PASS_ACCEPT: 4, DATA: 5, TERMINATE: 6, REJECT: 7

c. For the JOIN_REQ, PASS_REQ, PASS_ACCEPT and REJECT messages, the payload length is 0. For the PASS_RESP message, the payload length is the length of the password, for the TERMINATE message, the payload length is the length of the SHA digest, and for the DATA packet, the payload length is the number of bytes of the data segment you are transmitting.
d. Note that the payload length of DATA does not include the packet ID.

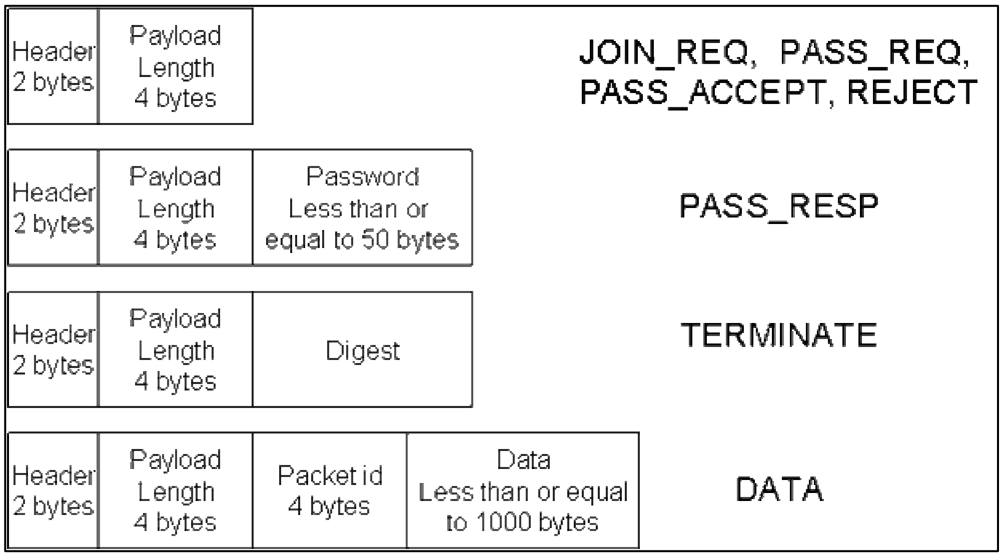

Figure #2: Packet Format

Command Line Arguments:
Your server and client code must be executed from the command line as follows:

./server <server port> <password> <input file>

The password corresponds to the correct password the client has to transmit in order for the server to consider a valid login. The input file is the path to the file that the server will send to the client.

./client <server name> <server port> <clientpwd1> <clientpwd2> <clientpwd3> <output file>

The three passwords correspond to the passwords used in each of the three attempts the client uses to login. Note that once a correct password is transmitted, no further login attempts are needed and the remaining password entries are ignored. The output file argument is the file name to assign to the file the server will send to the client.

Output messages:
You must print to the screen (STDOUT) the messages “OK” or “ABORT” depending on whether your application finishes correctly or terminate unexpectedly. Note that this information must be printed by both the client and the server programs.

a. Print the message “OK” in case your application finishes correctly. This is the case when the server is able to completely send the file to the client and the digest sent by the server matches the digest of the file received by the client.

b. Print the message "ABORT" in case an error occurs. Two examples of erroneous situations are: (i) the digest of the file the client receives differs from the digest sent by the server; (ii) the client or server receives an unexpected packet.

4. Deliverables

a. A physical copy of the file udpclient.c, and the file udpserv.c, submitted using the "turnin" command. Name your files <login1>.<login2>.client.c and <login1>.<login2>.server.c where login1 and login2 are login names of the two members of the group. For example: sanjay.rtorresg.client.c

   - Submission instructions

   - *Log into shay.ecn.purdue.edu*
   - *Move to the same directory as the file you are going to submit.*
   - *Execute: "turnin –c ece495f –p lab2 <login1>.<login2>.client.c"*
   - *Execute: "turnin –c ece495f –p lab2 <login1>.<login2>.server.c"*

b. Printout.

   Turn in a **typed** report with the following information:

   - C code implemented in parts 3.

5. Note on Grading

Points will be deducted if any of the deliverables in Section 4 is missing.
Further, points will be deducted if the program implemented in section 3

- Does not have a name as specified in section 4.a, (or)
- Does not compile (or)
- Does not produce the expected results. Please make sure your client (server) implementation works correctly with our server (client) implementation.

6. Some Socket Programming References

- http://www.ecst.csuchico.edu/~beej/guide/net/html/
- http://www.fortunecity.com/skyscraper/arpanet/6/cc.htm
- ftp://gaia.cs.umass.edu/cs653_1996/sock.ps
- UNIX Network Programming, W. Richard Stevens.
  It's available on the Engineering and Mathematical Science Libraries.

Please check the course web-page as we may update it with additional references