COSC 4330/6310—Operating System Fundamentals
Assignment #2 for spring 2017: A public key server
Now due on Wednesday, March 29 at 11:59:59 PM

Objective

You will be introduced to public-key cryptography and learn to use stream sockets.

Public-key cryptography

Public-key cryptography allows to people to exchange secret signed messages without having to exchange keys to a secure channel. To achieve that goal, it assigns to each user a public key K_p and a secret key K_s that satisfy the three following properties:

1. Messages encrypted with the public key K_p of a user can be only decrypted using the secret key K_s of that user.
2. Messages encrypted with the secret key K_s of a user can be only can be decrypted using the public key K_p of that user**.**
3. There is no possible algorithm that can compute in a reasonable amount of time the secret key K_s of a user knowing her public key K_p

Your programs

You are to write two programs:

1. A ***client program*** that will connect with your server and send it a single message requesting the public key of a user;
2. A ***server program*** that will read a list of pairs (user ID, public key) from a file, wait for connection requests from client processes and send them back either a string containing the requested public key or an empty string if the user is not in the table

The server program

Your server should start by prompting the user for a file name:

Enter a file name: keys.txt

This file will contain pairs of user IDs and public keys as in:

jorge@uh.edu a1b234c4567df123abc
eva@cs.uh.edu 5672c4567df123abc54

Both entries will never contain any spaces or tabs. It will then store the contents of the file into a table. Once it is done, it should prompt the user for a port to listen to as in:

Enter server port number: 2468

It should then create a stream **socket**, **bind** it to the specified port number, do a **listen( )** to specify a maximum number of queued connection requests and do an **accept( )** that will let it wait for connection requests.

Whenever it accepts a connection request, it will read in a string containing a user ID and reply by sending back the associated public key.

In addition, a special user request **"Terminate"** will force the process to terminate:

The client program

Your client should start by prompting the user for a server host name and a server port number as in:

Enter a server host name: program.cs.uh.edu
Enter server port number: 2468

Please note that the only correct answer to the first prompt is the name of the local host as provided by **gethostname( ).** Your client should ***reject*** any other entry and prompt for a new entry.

It will then create a stream **socket**, do a **connect( )** request to the specified server, and prompt for a user name as in:

Enter a user name: jorge@uh.edu

After that, the client will request a public key from the server and print it out as in:

The public key for user jorge@uh.edu is 25acd09…

unless the user entered the user ID **"Terminate."**

Hints

1. Please refer to "BSD Sockets: A Quick and Dirty Primer" on the Piazza resource page. It contains a general introduction to sockets. You can include any code from that document in your assignment.
2. Keep in mind that server and client processes read the messages byte by byte and have no way to know how many bytes they should read. You will need to come with a way to let the sender specify either the length of each message or its end using a non printable character.
3. Use a *single-threaded server* to keep things simple. You will not have to not worry about zombies and can safely ignore the **fireman()** call in the primer.
4. Neither user IDs nor public keys will ever contain spaces or tabs. You can expect user IDs to be short (think of an email address) and private keys a bit longer (but no more than 256 bytes).
5. Each client process will perform a single server request.
6. The server table should be able to store up to 1,024 pairs (userID, public key).

Updated last on Monday, March 20, 2017.