PROGRES - TME1

Sébastien Tixeuil - Sebastien.Tixeuil@lip6.fr

Note

Star exercises (indicated by *) should only be done if all other exercises have been completed. You don't have to do them to get a score of 20/20.

Exercise 1 - UDP Client / Server

We are trying to program in Python a PING mechanism between two machines connected to the Internet. The client sends a series of request messages to the server in the form of UDP packets of arbitrary content. The server responds to each request message with a UDP packet of arbitrary content addressed to the client that sent the request message. The client measures the time elapsed between its initial request message and the response message received from the server to evaluate the RTT with the server. Then, it takes the average of the measured RTTs and displays it to the user.

1. From the code examples seen in class for the UDP client and server, program the client / server PING mechanism in Python;
2. Test client and server programs on the same machine and on different machines;
3. Test the server and several clients making simultaneous requests;
4. Modify the server code so that it forgets to respond with a probability of 0.5 to a request message sent by a client. What's going on? Modify the client to restore normal operation. Test the modifications as in 2. and 3.

Exercise 2 - TCP Client / Server

We are trying to program in Python a simplified time server that responds to requests from clients by sending the current time on the local machine. In addition, we are looking to develop in Python a client that repeatedly makes calls to the time server to calculate the clock difference between the server and the client and display it on the screen.

1. From the code examples seen in progress for the client and the TCP server, program the clock difference calculation mechanism between the client and the server in Python;
2. Test client and server programs on the same machine and on different machines;
3. Test the server and several clients that are launched simultaneously.

Exercise 3 - Web server with TCP

We are trying to program in Python a simplified Web server which responds to requests formulated by clients (Web browser) as follows:

1. Creation of a connection socket upon request by a client;
2. Reception of the request (supposed to follow the HTTP protocol) on this connection socket;
3. Analysis of the request to determine the requested file;
4. Find the requested file in the server file system;
5. Create a response in HTTP format including the header and the content of the file;
6. Send the response on the connection socket, then close it.

In the event that the browser requests a file that is not present on the server, the server must return to the client an HTTP type 404 message. If the server is run on a machine that is already running a web server, it can be necessary to use a port number other than port 80.

1. Program the web server in Python;
2. Test the program with a web browser, on the same machine and on different machines;
3. Test the server and multiple clients making simultaneous requests.

Exercise 4* - Transferring Files with UDP

We are trying to program in Python a simplified file transfer mechanism with UDP that allows a client to request a file from a server, which in turn sends the file back to the client. We will suppose that the client's requests have the form: "`GET filename\r\n\r\n`", where "`GET`" is a keyword that specifies the type of the request, "`filename`" is to be replaced by the name of the file, and "`\r`" and "`\n`" are special characters that indicate the end of the file name. It is assumed that each UPD packet sent contains pieces of the requested file of size 256 bytes. It is therefore necessary to provide a mechanism to allow the transfer of files whose size exceeds 256 bytes. Specify the format of messages sent from the server to the client to send file fragments. Specify the format of messages sent by the client to request retransmission of unreceived messages. The server should do the following:

1. Receipt of the request (assumed to follow the requested format);
2. Analysis of the request to determine the requested file;
3. Find the requested file in the server file system;
4. Create a series of responses in the format expected by the client;
5. Send the set of responses to the client.

For his part, the client must perform the following operations:

1. Sending a request to the server in the requested format;
2. Receipt of responses from the server;
3. Data reordering; Possible request for data not received from the server;
4. Writing the file on the client.

The requested work is as follows:

1. Program the client and the server in Python;
2. Test client and server programs on the same machine and on different machines;
3. Test the server and several clients making simultaneous requests;
4. Modify the server code so that it forgets to send with a probability of 0.5 a packet corresponding to a request made by a client. What's going on ? Modify the client and server to restore normal operation. Test the modifications as in 2. and 3.