CIS3210 — Computer Networks

Assignment 3

Due November 26^{th} 2008 in/before class

This assignment asks you to create a UDP "client" that acquires a binary image from another UDP program ("server"). Your client will work with a provided remote server.

The goods

The server will send any accessible object of size not exceeding 500KB (one weakness of the design of the assignment).

The client requests one object from the set by sending a datagram (to the server) with the object's **URL**. This datagram (obviously of variable length) will have nothing else in its payload.

The server responds to the request by sending back a collection of datagrams containing the whole object. The client must be capable of handling non–text objects, such as **jpg** files.

The application protocol

The objects are too large to be sent in one datagram, hence many datagrams will be sent by the server. These datagrams will arrive out of order, if they arrive at all. In order to allow the client to reassemble the object, the server will send datagrams of unique lengths, following these conventions:

- Assume that the object is of size N with $\mathcal{N} > 15$. the server will send the $\mathcal{N}$ bytes in order, splitting them into datagrams of varying lengths.
- The first datagram will have a length $\mathcal{F}$ that is a power of 2 determined by the server and guaranteed $\mathcal{F} \leq 2^{10} = 1024$.
- The second datagram will have a length of $\mathcal{L}$ (the text below explains how $\mathcal{L}$ is arrived at).
- The third datagram will have a length of $\mathcal{R}$ where $\mathcal{R} < \mathcal{L}$ (again, explained below).

- The first three datagrams will be followed by $\mathcal{F}-\mathcal{L}-1$ datagrams which will have diminishing lengths: $\mathcal{F}-1, \mathcal{F}-2, ..., \mathcal{L}+1$. It is possible that $\mathcal{F}=\mathcal{L}+1$; if so, only the first three datagrams will be sent.

- Every datagram has a unique length. By tacit convention (followed by the server and the client) each datagram is identified by its length. The receiver does not know in advance the values of $\mathcal{F}$, $\mathcal{L}$ and $\mathcal{R}$, so these three datagrams can be referred to as 1024, 2, 1, respectively.

The value of $\mathcal{R}$ is such that the sum of lengths of all the datagrams equals $\mathcal{N}$.

The server will always send at least 3 datagrams. Note that for most values of $\mathcal{N}$ there is more than one legal way to split the object into datagrams, e.g. $35 = 16 + 15 + 4$ but also $35 = 8 + 3 + 2 + 7 + 6 + 5 + 4$.

The client will receive the datagrams sent by the server, not necessarily all of them, and not necessarily in the same order.

- If the client receives all the datagrams in the order in which they were sent, the object arrived in order and there is no need to rearrange it.

- If the client receives all the datagrams in a different order than the order in which they were sent, the client will have to sort them, thus reconstructing the original object.

- If datagrams are missing the client will ask the server to retransmit them by sending to the server a datagram of the form:

list length	$entry_1$	$entry_2$	... ...

  Each entry shows the length of the missing datagram sent in binary as a 2–byte short integer in **network order**. The list length (number of entries) is limited to 255.

  This datagram will be distinguished from regular requests because it with have its first byte equal NULL when decoded as a character.

Example

The server is about to send an object of 15910 bytes long. The server decides to send it in 16 datagrams with the first of length 1024.

The datagrams sent will have the following lengths (in order):

$$1024, 1010, 655, 1023, 1022, 1021, ..., ..., 1012, 1011$$

If the client received all the 16 datagrams, the communication is complete. Otherwise, the clients asks for repeats.

Suppose that the client receives 13 of the 16 datagrams with 655, 1021, and 1015 missing. It will send to the server an 8–byte datagram containing the following list of 2–byte short integers:

$$3, 1, 1021, 1015$$

Note that the "1" stands for the third datagram; the receiver does not know its length.

When the server receives a repetition request, it sends the datagrams one–by–one, in the order given in the list. Requests for datagrams that do not exist are quietly ignored.

If some of them fail to arrive, the client sends another repetition request, and so on.

Specifications

A few rules that must be followed:

- The object reconstructed by the client must be an exact copy of the object sent. You must be able to display it.
- The client will make a serious attempt to request missing datagrams and will continue until successful.
- Note that the server will ignore incorrect request formats.

The client will use a timeout mechanism to monitor the arrival of datagrams from the server. A repetition request is to be issued after a period of 30 seconds of silence.

Your task

Develop your client in steps (each step adds code to the previous one):

1. Create a client that manages to get the attention of the server. A trivial server can be copied from the web–I recommend an "echo" server that simply sends back whatever it received.
2. Upgrade your client to receive the datagrams sent by the trivial server.
3. Move to work with a simplified real server (I will post the code of a simplified version of it).
4. Make your client receive all the datagrams and reassemble them. This can be tried by running the server and the client on the same host (the chance of losing datagrams is remote in this scenario). Implement time-outs.
5. Make your client send repetition requests (spurious ones for testing) and use the repeated datagrams to reassemble the original object. Make sure to display the object also if it is a binary image.
6. Handle the case when repetition requests are lost.
7. Make your client work with the real server (which will be running on 131.104.49.193 toward the end of the season).

Deliverables

Your code will be written in C, C++or Java. It must use the Socket API function calls directly. All socket data traffic should be done by issuing `sendto/recvfrom` system calls (no wrappers).

You will submit on paper a printout of your code with a self–evaluation page attached (see appendix).

You will sign up for a demonstration of your code and successfully demonstrate that your self assessment is correct.

Self evaluation form

Name: _____ Student ID# _____ Total: [ ]

Step	Done correctly	Not done what is wrong?	Other Outline the problem in the space below
Code uses only sendto/recvfrom		If not, the grade = 0 skip the rest	
3			
4			
5			
6			
7			

Grading

You will receive marks for reaching milestones as listed above. Reaching a milestone ranked higher implies having satisfied lower–marked ones.

If you got this far:	3	4	5	6	7
You get this much:	4	8	12	16	20