

ECE361 Computer Networks, Spring 2014

Lab 2: File Transfer

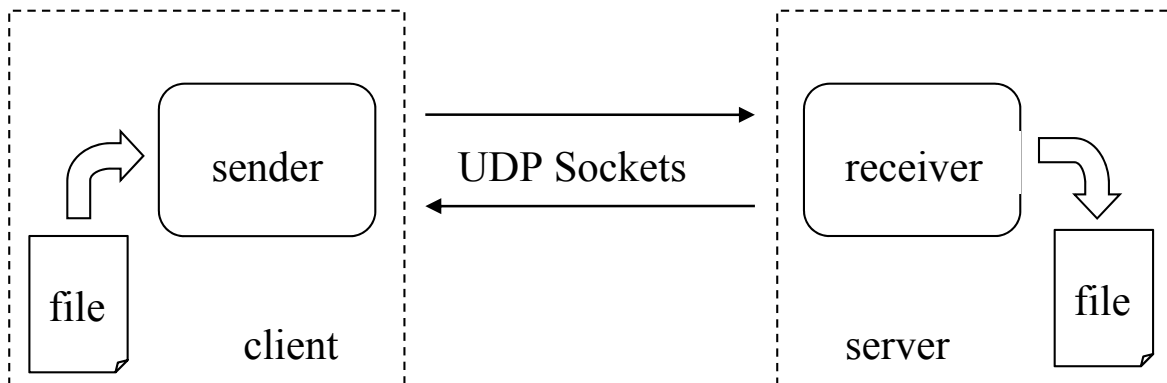
(Deadline: February 9, 9:00 pm)

Objective

This assignment will give you some hands-on experience with socket programming. You will use UNIX sockets to implement simple client and server programs which interact with each other to accomplish a file transfer in a connectionless manner. Before starting this assignment, you should read Section 2.4 “The Berkeley API” of your textbook. For further information on network socket programming, you may read *Beej's Guide to Network Programming* posted on the course website.

Lab Assignment

In this assignment, you need to implement a server that opens a socket and listens for incoming data transfer at a particular port number. You also need to implement a client that reads a binary file from the file system and transfers the file to the server. When the server receives the client's data, it writes the data to a file.



Packet Format:

All packets sent between the client and server **must** have the following structure:

```
struct packet {
    unsigned int total_frag;
    unsigned int frag_no;
    unsigned int size;
    char* filename;
    char filedata[1000];
}
```

The `total_frag` field indicates the total number of *fragments* of the file. Each packet contains one fragment. The `frag_no` field indicates the sequence number of the fragment, starting from 1. The `size` field should be set to the size of the data. Therefore, it should be in the range of 0 to 1000. All members of the packet should be sent as a **single string**, each field separated by a colon. For instance:

```
total_frag = 3
frag_no = 2
size = 10
filename = "foobar.txt"
filedata = "lo World!\n"
```

Your packet should look like this:

```
packet = "3:2:10:foobar.txt:lo World!\n"
```

Please remember that while the beginning of the packet is in fact just plain text, the data portion of the packet may in fact contain *binary* data. This means that you should not use string manipulation functions available in C for the data field or for the whole packet. Only the first part of the packet before data is really a string.

The reason you cannot use string functions is because string functions assume that the data ends with the null character. This character however, may appear *within* the data of the packet. If you were to use `strcpy` on a packet with binary data, some of your data may get lost and your program will not function correctly. You should test your program on both binary data (an image file for instance) as well as a text file. In general, if your program works for binary data, it will work for a text file.

Client Program (deliver.c):

You should implement a send program, called `deliver.c`, in C on a UNIX system. Its execution command should have the following structure:

```
deliver <server address> <server port number> <client listen  
port> <file name>
```

Upon execution, the client program should read data from the file specified and send it to the server using a UDP socket. If a file is larger than 1000 bytes, the file needs to be fragmented into smaller packets with maximum size 1000 before transmission. **You should implement some sort of acknowledgement in order to guarantee correct receipt of the file.** For this assignment, you may use a simple stop-and-wait style acknowledgement. The client should open a UDP socket to listen for acknowledgements from the server.

Server Program (server.c):

You should also implement a server program, called `server.c`, in C on a UNIX system. Its execution command should have the following structure:

```
server <UDP listen port>
```

Upon execution, the server should open a UDP socket and listen at the specified port number. Upon receiving the first packet in a sequence (i.e. `frag_no = 1`), the program should read the file name from the packet and create a corresponding file stream on the local file system. Data read from packets should then be written to this file stream. If the EOF packet is received, the file stream should be closed.

You may wish to use ACK and NACK packets to control data flow from the sender. You will have to carefully coordinate between the client and the server in order to guarantee correct file transfer.

Makefile

You should also prepare a makefile that generates the executable file `deliver` from `deliver.c` and the executable file `server` from `server.c`.

Execution Example

Assuming you have a file named `source.jpg` on `ug201` which you wish to send to `ug202`:

On the host `ug202`:

```
server 9993
```

On the host `ug201`:

```
deliver ug202.eecg.utoronto.ca 9993 9876 source.jpg
```

Remember that your two programs need to be in separate folders as the file cannot be copied onto itself. You can verify correct operation of your code by performing a binary `diff` on the source and destination file.

Deliverables:

The following should be available for the lab evaluation:

- The client program (`deliver.c`)
- The server program (`server.c`)
- Makefile to compile your program
- Any extra header files or source code necessary for correct operation of your code.

Submission Procedure:

For electronic submission, only one person in the group should submit the file. You have to create a tar ball (`a2.tar.gz`) with all the files needed to compile and run your programs.

The following command can be used to tar your files:

```
tar -czvf a2.tar.gz <project directory>
```

where the project directory contains your source code, headers, and Makefiles.

Use the following command on the eecg UNIX system to submit your code:

```
submitece361s 2 a2.tar.gz
```

You can perform the electronic submission any number of times before the actual deadline. A resubmission of a file with the same name simply overwrites the old version. To see a list of what you have submitted, use the command

```
submitece361s -l 2
```