Programming Assignment 2: Reliable transfer file

DUE Friday Oct. 20, 11.59.59 PM.

1 Overview

This assignment is intended to familiarize you with writing packet structures, network signals, and reliable communication protocols. You will implement a pair of sender and receiver programs which allow the sender to reliably send a file to a receiver using UDP. As UDP is an unreliable service, data can be lost or received out-of-order, thus the receiver will need to signal packet lost, while the sender will need to retransmit the lost packets.

NOTE: The name of the file stored by the receiver should be the name of the file sent by the sender concatenated with the process id of the receiver. (Hint: see getpid function.)

1.1 Part 1 - Stop-and-Wait

In the first part of the project, you will implement a very simple reliable protocol, Stop-and-Wait like the one described in class in slide 21 from Lecture Datalink. In Stop-and-Wait, the sender transmits a packet and then waits for the receiver to acknowledge that packet before sending the next one. If the acknowledgement (ACK) is received then the sender will send the next packet and so on till the entire file is sent. After the file was sent both the sender and receiver should close the connection. The receiver needs to store on the disk the received file. Each packet needs to be written on the disk when received, do not copy the entire file in memory and write it at the end, this will not be considered a correct solution. Note that the ACK can also be lost, in this case the sender will observe a timeout and will resend the packet. Receiver will discard duplicate packets.

1.2 Part 2 - Go-Back-N

In the second part of the project you will implement the Go-Back-N scheme which is a more simpler version of a sliding window algorithm. At a high level, the scheme allows the sender to send more than 1 packet, a window of packets before waiting for an acknowledgment. The receiver process keeps track of the sequence number of the next packet it expects to receive, and sends that number with every ACK it sends. The receiver will discard any packet that does not have the exact sequence number it expects (either a duplicate frame it already acknowledged, or an out-of-order packet it expects to receive later) and will resend an ACK for the last correct in-order packet. Once the sender has sent all of the packets in its window, it will detect that all of the packets since the first lost packet are outstanding, and will go back to the sequence number of the last ACK it received from the receiver process and fill its window starting with that frame and continue the process over again. More details are also here

`https://en.wikipedia.org/wiki/Go-Back-N_ARQ`

As in Part 1, the sender needs to send a file, the receiver needs to save the file on disk by writing each packet on disk, do not store the entire file in memory, before writing it to the disk, this will not be accepted as a correct solution.

2 Language

You need to write your code in C, and ensure that your code compiles and runs on unmodified CCIS Linux machines on the command line. Do not use libraries that are not installed by default on the CCIS Linux machines. Similarly, your code must compile and run on the command line. You may use IDEs (e.g. Eclipse) during development, but do not turn in your IDE project without a Makefile. Make sure you code has no dependencies on your IDE. You can use either multi-threading or events-based programming (i.e. select function).

3 Protocol

You can use any port you want since you are writing both the sender and the receiver. Your protocol will use three type of messages: INIT which is the first message sent by the sender and specifies the name and length of the file, and the protocol used (1 means Stop-and-Wait, > 1 size of window in Go-Back-N), DATA which carries data, and ACK which carries acknowledgments.

Hints; decide the format of your packets before implementing. For example you might want to use a byte for the type of the message, include a sequence number on the DATA, etc. Be efficient in your header design, do not use strings.

4 Your program

Your receiver program must execute on the command line using the following command.

```
$ ./receiver <-m mode> <-p port> -h <hostname>
```

Your sender program must execute on the command line using the following command.

```
$ ./sender <-m mode> <-p port> <-h hostname> <-f filename>
```

The *-m mode* parameter will specify if the sender and receiver are using Stop-and-Wait or Go-Back-N. If mode is 1, your protocol is using Stop-and-Wait, if mode > 1 then the protocol is Go-Back-N with N = mode. Your receiver should reject the connection from the sender if the protocol mode of the sender does not match the one from the receiver, i.e. the receiver and sender should have the same mode value. Remember that the sender will send its mode value to the receives in the INIT message.

5 Other Considerations

Your programs must verify the validity of messages by strictly checking their format. If a received message is not as expected, such as an incorrect field or wrong message type, you must assert an error and terminate your program. You should be strict; if the returned message does not exactly conform to the specification above, you should assert an error. Remember that network-facing code should be written defensively.

6 Submitting Your Project

Follow post @39 from piazza, with name of the project ”project2”.

To turn-in your project, you should submit your (thoroughly documented) code along with three other files:

- A Makefile that compiles your code.
- A plain-text (no Word or PDF) README file. In this file, you should briefly describe your high-level approach, any challenges you faced, and an overview of how you tested your code.

Your README, Makefile, and source code, etc. should all be placed in a directory on a ccs account before submitting. You may submit as many times as you wish; only the last submission will be graded, and the time of the last submission will determine whether your assignment is late.

7 Grading

This project is worth 100 points, 50 points first part and 50 points second part. All student code will be scanned by plagiarism detection software to ensure that students are not copying code from the Internet or each other.

In order to get full credit you will have to also explain how you tested your code, what scenarios did you consider, how do you know your code works correctly. You can include any documentation or code, state machine diagrams, wireshark screen terminals, testing code.

8 Extra Credit

Implement a sliding window algorithm where the receiver also maintains a window and accepts packets out of order. This is also known as Selective Repeat.

`https://en.wikipedia.org/wiki/Sliding_window_protocol`

This extra credit is worth 50 points.

9 Additional resources

You may find the following resources helpful

- Socket programming: `http://beej.us/guide/bgnet/`
- Unix programming links: `http://www.cse.buffalo.edu/~milun/unix.programming.html`
- C/C++ programming link: `http://www.cplusplus.com/`