CS 313 Project: Simulation of Simplex Stop-and-Wait with PAR and NAK

To be done individually, not in groups (10:00PM 4/17/2017)

Objective

To simulate data communication over a noisy simplex channel in the data link layer using a simplex stop-and-wait protocol with PAR and NAK

Description

This project implements the PAR error control protocol studied in the class. The two major programs are the sender and the receiver (feel free to modify LinkSender and LinkReceiver). The sender transmits a file through a noisy channel. The file is divided into "packets" to fit into individual frames. Errors are generated randomly in the CRC-checksummed frame. When the receiver receives a frame, it calculates the CRC checksum to detect the errors. If there is no error, a positive acknowledgment is sent to the sender. Otherwise, a negative acknowledgment is sent back to the sender and the frame has to be retransmitted. The communication continues until the file is transmitted completely. (Note that we unrealistically assume that no data will be lost and no error occurs on the acknowledgements. It is for the purpose of proof-of-concept and simplification.)

Frame Format

Two types of frames are used in our protocol. Their formats are described as follows:

1. Date frame (fixed size = 19 bytes), sent from sender to receiver

[SEQ | LEN | PAYLOAD | CHECKSUM]

- SEQ: sequence number; size = 1 byte; not really used, reserved for future use
- LEN: actual length of payload in bytes; size = 1 bytes; value is 0 to 16
    - LEN = 0 means "empty data" which is used to signal the termination of data transfer
- PAYLOAD: data portion of the frame; size = 16 bytes
- CHECKSUM: generated using CRC-8; size = 1byte

2. Acknowledgment frame, sent from receiver to sender

[ACK]

ACK: size = 1 byte; value = 1 for positive acknowledgment, value = 0 for negative acknowledgment

Sender Procedure

1. Reads a "packet" of 16 bytes (or whatever left) form the input file.
2. Puts the packet into the PAYLOAD field and completes the header information of the frame.
3. Calculates CRC checksum for the frame and places it in the CHECKSUM field of the frame.

4. Uses the error function to generate error or errors intentionally and then transmits the possibly damaged frame.
5. Receives the acknowledgment.
6. If positive then

   go to step 1

   else

   go to step 4 (if the frame is not "buffered" you might have to go to step 2)
7. The procedure in steps 1 to 6 terminates when the file is "transferred" completely.
8. The sender then sends an "empty data" to signal the end.
9. Close the input file.

Receiver Procedure

1. Receives the frame and calculates the checksum.
2. If “remainder” is 0 (i.e., no error) then

   writes the "packet" (i.e., frame's PAYLOAD field) to the output file

   sends a positive acknowledgment to sender

   else

   sends a negative acknowledgment to sender
3. The procedure in steps 1 to 2 terminates when "empty data" is received.
4. Close the output file.

(Don't forget to compare the two files, both content and size, after the program terminates to verify the correctness of your programs.)

Requirements

1. You are required to use the Link class provided by the instructor for the data communication.
2. Use the CRC-8 function provided by the instructor or implemented by yourself.
3. You are required to implement the protocol exactly as described, including the frame format and procedure.
4. Include a trace option. If trace is on, the sender will output the following message:

   Frame *x* transmitted, *status*

   and the receiver will output the following message:

   Frame *x* received, *status*

   where *x* is the sequence number and *status* is either "ok" or "error".

5. Error function uses a random number generator to determine whether to damage a frame.

   If a frame is to be damaged, the probability of 1 error is 0.75 and 2 errors is 0.25.

Every bit in a damaged frame is equally likely to be altered.

6. Input from user:

   Sender: input file name, probability that a frame will be damaged (i.e. error rate), trace on/off

   Receiver: name of output file, trace on/off

7. Test your programs on these input files: mission0.txt, mission1.txt, and mission2.txt, with error rates 0, 10%, 50%, and 90% for each test case. You are encouraged to include more test cases created by yourself.
8. Output displayed after transmission completed by sender:

   Total number of "packets" read

   Total number of frames transmitted

   Theoretical total number of frames transmitted (= Total number of "packets" read / (1 - error rate))

   Total number of frames damaged

   Maximum number of retransmission for any single frame

9. Both content and size of the received file must be identical to those of the file sent.
10. This project is to be done individually. Do not copy any other student's work or copy from any other source. However, after you are done, you are encouraged to test your programs with other students.

Submission

1. Compress your code, report, the executable files, and results of test runs into one single zip file. Submit the zip file to D2L Assignments. Submitting multiple files will not be accepted.
2. The report should include
   a. Instructions to run and test your program
   b. Analysis on the actual and theoretical total numbers of frames transmitted
   c. What you learned from this project
   d. Obstacles encountered during working on this project and how they are tackled
   e. Comments and suggestions
   f. Known bugs
3. The instructor may contact you regarding testing and grading your project.

Grading

- Correctness, robustness, documentation, and report. A correct receiver program might be used to test your sender program.

- Note that a successful file transfer means both content and size of the file being sent and the file being received should be identical.

<u>Suggested Milestones</u>

Week 1 (3/15 – 3/21): Make sure you understand the project and its requirements and can run the programs provided to you without problems. Figure out how to read from a file to byte arrays and write byte arrays to a file.

You are advised to write a simple program that can repeatedly read 16 bytes from a file into a byte array and then write the byte array to an output file until the entire file has been "copied". If you cannot make it work, there is no point to proceed to the next step.

Weeks 2-3 (3/22 – 4/4): Implement the protocol and test it using strings or a simple file without errors, and then with errors introduced.

Weeks 3-4 (3/29 – 4/11): Test on files. Add other requirements.

Weeks 5 (4/12 – 4/17): Wrap up. Write report.

Start as early as possible!

Note that this programming assignment deals with byte data, which might be unfamiliar to you.

FEEL FREE TO POST QUESTIONS TO D2L DISCUSSIONS!