CS 431 Lab #3
Serial Communication

Spring 2015

1 Overview

This is a 2-part lab split over 2 lab weeks. There will be two TA demos: You will demo Part 1 during lab on the week of Feb 18, 2015 and Part 2 during the week of Feb 25, 2015. Since Part 2 is substantially more difficult than Part 1, we recommend starting early on the second part.

In this lab you will implement robust serial communication between a server (PC/Linux) and a client (Flex board). You will write both the client and server, and test the reliability of transmission in the face of an adversarial middleman.

2 Communications Protocol

The server will send messages to the client using a known protocol defined below.

Size (bytes)	Value	Description
1	0x0	Start byte
2		16-bit CRC of message body
1		Length of message body: N (max 255 chars)
N		Message body

When the client successfully receives a message, it should respond with an ACK byte of 0x1. If it detects that the message has been corrupted, it should respond with a NACK byte of 0x0. Messages can be corrupted in the following ways:

1. An incoming message did not begin with the start byte.
2. Too many or too few bytes of message data were sent.
3. The locally computed 16-bit CRC of the message body does not match the one sent from the server.

The client and server will communicate using the following data settings: 9600 baud, 1 start bit, 8 data bits, no parity, and 1 stop bit. When setting the number of data bits in the client, read the datasheet carefully as it can be tricky.

3 Procedure

1. Before getting started, read sections 3.7, 4.10, 6.1-6.3, 6.7 in the CS 431 Laboratory Manual. NOTE: In addition to the tcsetattr flags listed in the lab manual for writing, you will also need CREAD in order to read from the serial port. See the man page for more details.

2. Create a *lab3* directory in your cs431 directory and change directories to it. Since this lab has both an Flex and Linux component, you will make two separate projects. Make a Makefile for the Linux portion, and a MPLAB project for the Flex portion as you did for the first two labs.

3. Demonstration versions of both the Lab 3 client and server programs that you now need to write are provided in compiled form. To upload the client to your Flex board, change directories to */emb1/lab/demo* and execute `make lab3_client`. To run the server program, execute `/emb1/lab/demo/lab3_server`. You can use the provided server binary to test your client, and the provided client binary to test your server during development.

4. Part 1: PC/Linux Server

   (a) You are given a skeletal header and source file for the server portion of this assignment. You will add code to this file to complete the server implementation. These files are available in */emb1/lab/lab3.h* and */emb1/lab/lab3_server.c*.

   (b) The function `int pc_crc16(const char *str, int length)` from */emb1/lab/pc_crc16.*{*h,c*} computes the server-side CRC of your message body. You can copy these files to your *lab3* directory and include them in *Makefile*. Unlike the embedded version, `pc_crc16()` accepts a string and its length and returns the 16-bit CRC of the entire string.

   (c) The troll program binary is provided for you in */emb1/lab/lab3_troll*. The troll is allowed to flip from 0 to 8 arbitrary bits in each byte sent, with a probability specified as a command line argument. You will send outgoing serial communications through the troll by invoking `lab3_troll` via a `popen()` call and writing through a stream to the process. The code to do this is given in the provided skeleton code.

   (d) Complete *~/lab3/lab3_server.c* such that it can send messages in the protocol defined above and receive acknowledgments from the client. The server should resend the message whenever a negative acknowledgment (NACK) is received, until a positive acknowledgment (ACK) is seen.

5. Part 2: Flex Client

   (a) It is good practice to put associated reusable code into new *.c* and *.h* files. In this lab, you are required to put the following three functions into *~/flexserial.c* and *~/flexserial.h*. Add `../flexserial.o` to the `OBJS` line in your MPLAB project for the file to be compiled and linked.

      i. `void uart2_init(uint16_t baud)`
      Perform any necessary initialization of the serial interface, baud rate, and data

protocol. You will setup the UART2 device to be used accessed via polling. Specfic information in regards to this is located in the lab manual in section TODO. There is also further documentation in section 17 of the dsPIC datasheet.

ii. `int uart2_putc(uint8_t c)`
Write a byte to the serial interface.

iii. `uint8_t uart2_getc()`
Read a byte from the serial interface.

(b) You can use the function `uint16_t crc_update(uint16_t crc, uint8_t data)` on the Flex board to compute the 16-bit CRC (cyclic redundancy check) of your message body for integrity verification. You will have to include `flex/crc16.h` to use this function. `uint16_t crc` should be initialized to 0, and repeatedly passed to `crc_update()` with each byte of message data to compute the CRC of the entire message body. Compare the locally computed CRC to the CRC sent by the server to verify data integrity.

(c) Since incoming data may be corrupted, you cannot guarantee that the message length you expect to receive is correct. Therefore, you should set a timer to trigger an interrupt 1 second after data is first seen to signal that no more incoming data is coming. If you successfully read the expected amount of data (or more), disable the timer to prevent it from firing. If it fires before you have received the data you expect, you know that data corruption has occurred.

(d) Write *∼/lab3/lab3_client.c* such that it can receive and acknowledge messages sent in the protocol defined above.

(e) Count the number of failed sends per message and display it along with the message body as shown in the demo client.

(f) Test your server and client together with various values for the troll's corruption probability. Using the verbose mode, ensure that ACKs are sent when data is not corrupted, and NACKs are sent in every case where corruption occurs.

4 Hints and Tricks

It can be difficult to debug the serial communication code and ensure that you are sending the correct data. An easy way to verify your data is to redirect your Linux file pointers to write to a file instead of or in addition to the serial port. Once you have this data, you can open it in a hex editor such as `hexedit` to analyze. This will show you the actual hexidecimal values in addition to the ascii characters, making it possible to check non-printing characters like `0x0`. In addition to the above, make sure that you are performing initialization, read, and writes on the UART2 device. All registers and bitmaps will be prefixed with U2.

5 Questions to Ponder

1. Data Validation via CRCs
   (a) Assuming that CRCs are randomly distributed, what is the probability that two messages will have the same CRC?
   (b) How would using only the least significant 8 bits of the CRC influence data robustness? Try making this change to your client and server, and testing the results.
   (c) For high speed networks, what is the trade off between short and long error detection schemes? For example: 16/32-bit CRC vs 128-bit MD5.
2. What are other potential solutions to the receive buffer overflow problem?