ENSC 351: Real-time and Embedded Systems

Craig Scratchley, Fall 2017

Multipart Project Part #1

Details on how and when to submit this part of the project will be given sometime in the next week. I will probably give out the next part of the project before this part is due, so don't delay in getting started on this part.

Unless I have instructed you or agreed with you otherwise, please choose one partner to work with. Choose your partner carefully. Try to choose a partner that will work with you on the various parts of the multipart project with as much care as your own degree of care. For purposes like this course, pair programming can be very useful.

http://en.wikipedia.org/wiki/Pair_programming

This part of the project does not require understanding real-time or embedded systems. It is a standard programming assignment and allows you to practice programming using relatively simple C and C++.

If you are beginning to forget the C++ you have learned, you will find the following website helpful:

http://www.icce.rug.nl/documents/cplusplus/

This document assumes knowledge of C language. One site for C language is:

http://www.eskimo.com/~scs/cclass/cclass.html

I have also listed a good C++ book in the course syllabus. The book for CMPT 128 should also be helpful. Some copies of these books should be in the library, or they can be purchased from a good bookstore.

The first part of the project focuses on parts of the XMODEM file transfer protocol.

Modify and complete the designated function members in the SenderX class in SenderX.cpp and in the PeerX class in PeerX.cpp. For this first part of the project, we are just simulating the sending of a file, "inputTextFile.txt", using both the originally specified checksum and the CRC variant of the XMODEM protocol. The protocol is described in the "xmodem-edited.txt" file, which will be our primary reference. The file "XMODEM-Clarified.txt" may also be useful to understand the original version of the protocol. Don't feel obliged to use the subtractions mentioned in the second .txt file if you think that simply ignoring overflows can do the trick for calculating a checksum.

The file xmodem-edited.txt is a version of the original specification that includes many edits that I have made in order to clarify the specification. It is interesting to read an older specification like the XMODEM spec. Note the reference to the (Intel) 8080 "CMA" instruction. Of you, who has ever used the 8080 "CMA" instruction! How many of you even know or remember that the 8080 microprocessor is an old one from Intel? (I sometimes mention the 8080 microprocessor in the ENSC 254 course.)

For the first part of the project, instead of sending the blocks and any other characters over a serial port or similar, the blocks and other characters should simply be written to an output file, "xmodem_sender_data.dat". This file should only contain bytes that would normally be sent from the

sender to the receiver. We are just imagining that a receiver exists and are assuming that it will correctly start the sender, positively acknowledge every block that it receives, and properly handle the termination of the transfer after it has received all the blocks for the entire file.

I have written a rather complete template for you to modify. You will only need to modify the member functions genBlk(), sendFile(), and crc16ns(), calling any additional functions that you might want. See the sendFile() member function as given to you for a simulation of the main part of the protocol in operation and for the use of the genBlk() member function (you *will* need to finish the sendFile() member function). Use the myCreat(), myOpen(), myRead(), myWrite(), and myClose() wrapper functions for the POSIX functions creat(), open(), read(), write(), and close() to create, open, read from, write to, and close files. You can find documentation on POSIX functions online.

The template already makes some use of the wrappers for these POSIX functions. On Microsoft-Windows-based computers these POSIX functions are directly available when using the MinGW compiler. Mac OS X and Linux support POSIX functions "out-of-the-box". Notice that in my template all direct or indirect calls to the POSIX functions open(), create(), read(), write(), and close() are checked for a return value like –1, indicating that an error occurred or something else went wrong during execution of the function. For any direct or indirect calls to write() and/or read() that you need to add to the template, you must handle a return value of -1 in a suitable way, and also handle an unexpected return value where reasonable to do so. Do not modify lines of code in my template unless modifications to those lines are indicated or you explain in a comment why you have decided to make such modifications. Modifications that you feel improve the code are encouraged but otherwise modifications are discouraged as, among other things, they may affect marking.

Assume that the imaginary receiver sends a 'C' or NAK to begin the transfer and always sends an ACK for each block. After the last block is ACKed, send an EOT, which will be NAKed, and then repeat the EOT. So for a file that needs 4 blocks an imaginary receiver requesting CRCs would send
'C', ACK, ACK, ACK, ACK, NAK, ACK

For this part of the project, don't worry about references to the CAN byte in the specification, and don't worry about timing. Just assume that, for example, an ACK will be quickly received after each block is sent.

As I understand is usually the case for ENSC courses, you are allowed to verbally discuss course projects with classmates. However, for each person, whether a classmate or not, who helps you with some part of your submission, you must acknowledge their help. Please use the indicated area at the top of the relevant files to do so. Any help that is not acknowledged constitutes academic dishonesty and can trigger university-specified penalties, which can include receiving a grade of FD for the course. Please consult the relevant university policies if you need to review them. I do report on academic dishonesty and have done so a number of times in the past.

Though verbal help is okay when acknowledged, it is not allowed to use code written by anyone other than yourself or your project partner while you are completing any part of the project. It is also not allowed to share your code or your partner's code or portions of such code with others. That means don't ask anyone for code or you could get them in trouble too. Any lines of code that may end up in online Piazza discussions for the course may be used, but should still be acknowledged. Obviously you should be judicious when posting lines of code to Piazza (or possibly Canvas). Don't post more than is necessary to ask or answer a question.