ENSC 351: Real-time and Embedded Systems

Craig Scratchley, Fall 2017

Multipart Project Part 2

Please continue working with a partner.

In part 1 of the project you programmed some code for the XMODEM protocol that generated blocks with checksums or CRC16's and responded to (imagined) ACKs and NAKs, and sometimes 'C'. In part 2 we will expand on that code and write additional code to receive blocks and hook in a simulator for a communication medium provided by a pair of telephone modems and the telephone system between them.

In the function *main()* of the provided code, I create a thread identified via *term2Thrd*. That created thread will represent a communication terminal program (terminal 2). The primary thread will serve as *term1* (terminal 1). Create another thread, identified via *mediumThrd*, for a "kind" medium.

We want to send a file using the original checksum or CRC16 variant of the XMODEM protocol from *term2Thrd* via the kind medium to the primary thread. That is, *term2Thrd* will generate blocks and send the blocks to the kind medium for forwarding on to the receiver running in the context of the primary thread. I am providing a template project, which contains elements of the part 1 solution, to use for this part 2 of the project.

The template code as I am giving it to you should compile, run, and actually successfully do an XMODEM file transfer. However, not much checking is done to see what bytes are actually being received on each end. Once the so-called kind medium is hooked up, the receiver will not correctly reconstruct the file being sent.

Your mission, should you choose to accept it, is to improve the code so that the file is successfully transferred even after the medium is hooked up. You will need to improve the XMODEM code so that characters like 'C', ACKs and NAKs are properly sent, received, and processed. In general, changes will need to be made where indicated by the markers "*****" in comments in the code. When writing the sender code, consider the StateChart that I provided for the simplified sender. Before writing the receiver code, draw a StateChart for the simplified receiver. You will need to hand in your simplified receiver StateChart at the time of your submission.

For now, don't worry about elements of the XMODEM protocol that are not needed for this part of the project. For example, don't worry about timeouts, user cancellation of a transmission, and purging at the receiver, etc. We will deal with them in later parts of the project.

The "kind medium" is kind but not error free. The kind medium complements certain bytes that pass from term2Thrd to terminal 1. The way the medium is written, the first byte of the blocks in the file we are transmitting should not get complemented. So this means that the SOH byte in our block headers will not be corrupted in this part of the project. That is essentially why we don't need to worry about timeouts in this part of the project and why we consider the medium to be kind. In future parts of the project we will have an "evil medium", which may alter the SOH byte and do other mean things.

The kind medium also introduces some errors when transferring data from terminal 1 to terminal 2. It changes every 10^{th} ACK from the XMODEM receiver into a NAK.

To communicate between the threads, we will use socketpairs according to the provided code and the following diagrams:

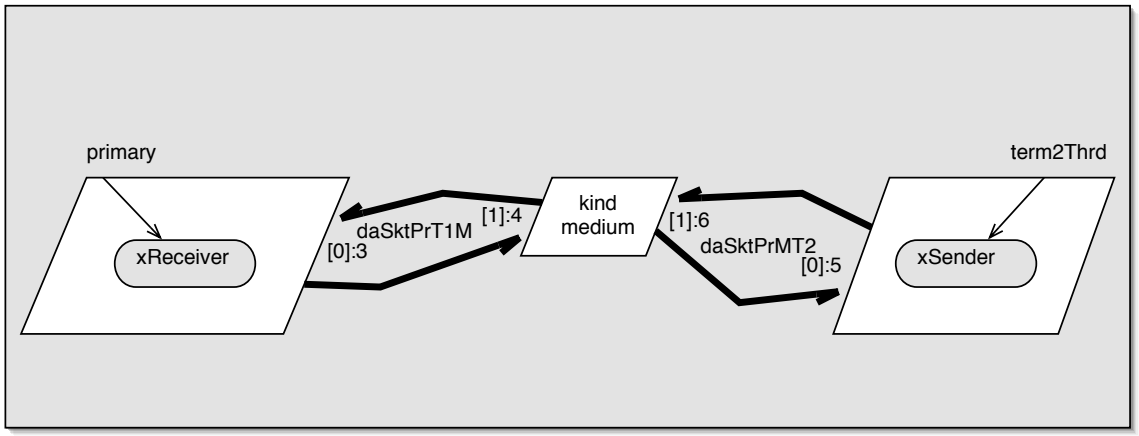

Figure 1: Collaboration graph showing asynchronous communication between threads provided by socketpairs

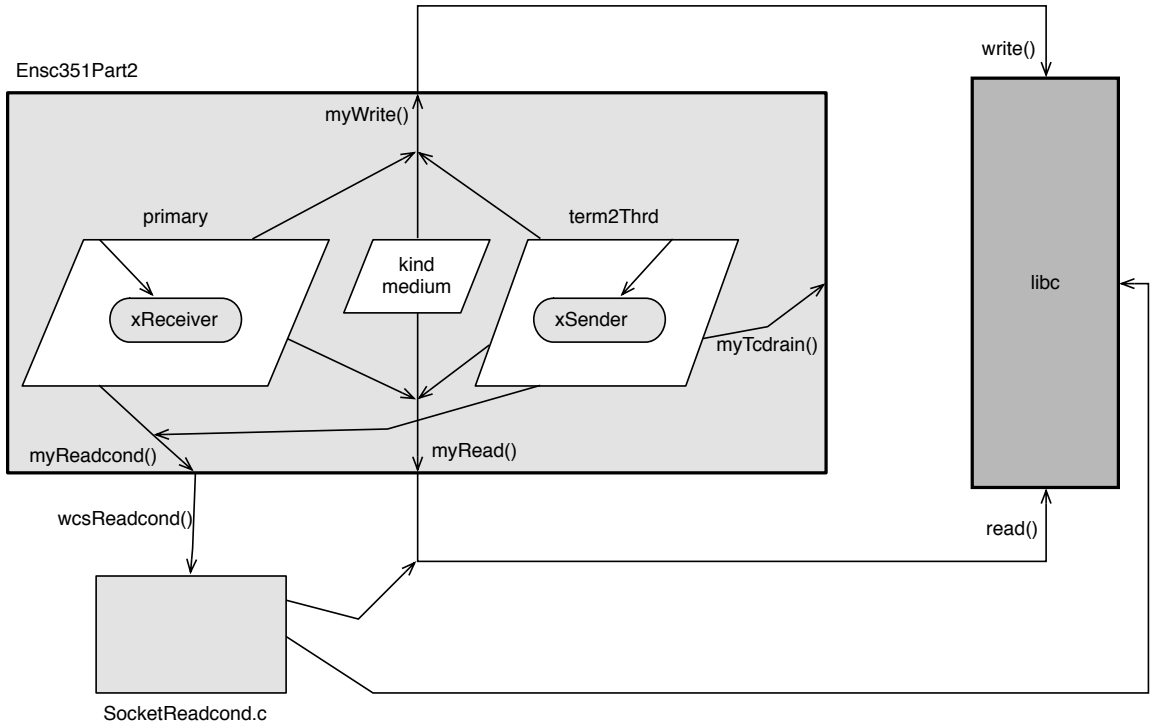

Figure 2: Collaboration Graph showing function calls for socketpair communication

Note that the template code writes to and reads from the socketpairs indirectly using functions *myWrite(), myRead()* and *myReadcond()* found in the file *myIO.cpp*. All calls to *myReadcond()*, which provides the functionality of the *readcond()* function ("READ from CONsole Device") available in the QNX Real-time Operating System, have the *time* and *timeout* arguments set to 0 for this part of the project. We will need and deal with timeouts later in the course. The supplied code also calls a function *myTcdrain(),* which will eventually use or simulate the *tcdrain()* function ("Terminal Control: DRAIN"). For this part of the project, myTcdrain() does nothing. myTcdrain() is used so that the XMODEM sender, for example, will in future parts of the project be able to wait for the medium to have received all the bytes of a block already written to the medium before sending the last byte of the block.

The XMODEM receiver should create an output file named “transferredFile” for the file being transferred. The kind medium should copy all data it sends in both directions to a special output file (“xmodemData.dat”). The input file should be “/etc/mailcap” from our Linux Virtual Machine).

You will have this week and next week, including both weekends, to work on this part of the multipart project. The exact time will be announced shortly on CourSys (courses.cs.sfu.ca).