ENSC 351: Real-time and Embedded Systems

Craig Scratchley, Fall 2021

Multipart Project Part 3

(Figures to be updated from xmodem to ymodem.)

Please continue working with a partner. You will have next week and the week after, including 3 weekends, to work on this part of the project.

In part 2 of the project you wrote code to both send and receive YMODEM blocks and hooked in a simulator for a communication medium provided by a pair of telephone modems and the telephone system between them. The medium corrupted the data in some ways, but did not as configured inject spurious characters from simulated noise on the simulated telephone line. In particular, we had not enabled the ability for the medium to send an extra ACK character to the YMODEM sender while a block was being transmitted. We will enable that ability in Part 3.

As I will discuss again in class, just before the last byte in a block is transmitted to the YMODEM receiver, the YMODEM sender should dump any "glitch" characters that may have arrived at the sender due to noise on the telephone line. The sender dumps characters by calling myReadcond() specifying that the read should timeout immediately. If any characters are available, they are read in and discarded. If no characters are available, the myReadcond() call times out immediately (it returns the value 0 immediately).

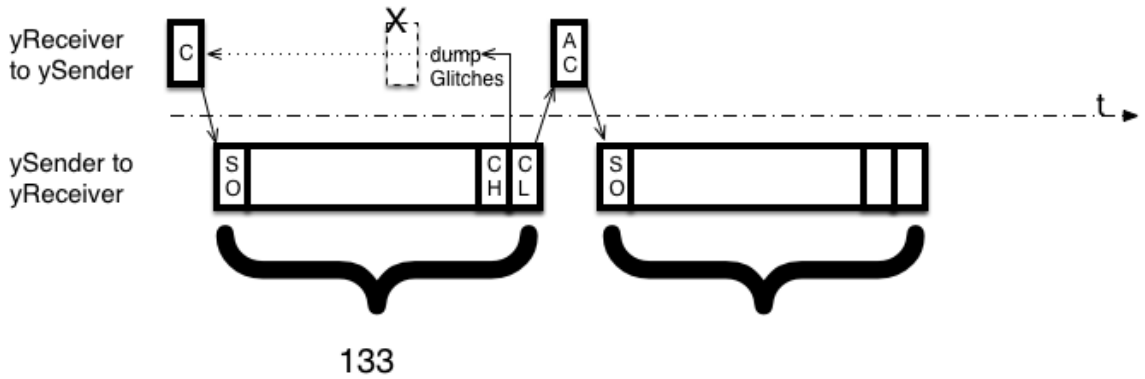

Figure 1: Glitches are dumped just before the last byte in a block is transmitted

How does the YMODEM sender know that all previously written characters have actually been transmitted to the modem and that now is the time for it to dump glitch characters? Well, in the code, we are calling myTcdrain() for the sender to know. For a terminal (i.e. console) device like a serial port, tcdrain() blocks the calling thread until all previously written characters have actually been sent. However, tcdrain() is not supported out-of-the-box for sockets, which we are using in Part 2 to simulate serial ports. In fact, our myTcdrain() currently does nothing but return 0.

You can enable the sending of extra ACKs in the medium by removing near the top of Medium.cpp the comment characters at the beginning of a line to yield this line:

#define SEND_EXTRA_ACKS

If you try this once you have finished Part 2, you will see that troubles currently arise transferring a file with the YMODEM protocol.

So, your job is to make modifications in the myIO.cpp file to get myTcdrain() working, similar to how it works for terminal devices, but when using sockets in socketpairs for communication. I recommend that you use condition variables, as discussed in Chapter 4, in order to achieve this. For our immediate needs in Part 2, we have only one thread using each socket (two threads using each socketpair – one thread per socket). However, try to make your code relatively general, where multiple threads might be affecting a socket in a socketpair. For example, thread A might write data to a socket and then block on a call to myTcdrain(), and then a thread B might block on a call to myTcdrain() for the same socket, and then a thread C might come along and read from the paired socket all the data written by thread A. At that point both the threads blocked on myTcdrain() should be unblocked and allowed to return from their respective calls to myTcdrain(). If you want, you can assume that there will be only one (blocked) reading thread at a time, so if one thread is blocked reading from a socket, no other thread will come along and also try to read from that same socket. Let us know if you try to relax that assumption. I will provide a testing project to you all to help in testing your modifications to myIO.cpp.

As long as there is memory available, the myIO.cpp file should work with as many descriptors as needed. We will be reasonable with our testing, but don't put your own arbitrary limit on the number of descriptors.

Please ask on Piazza if you have any questions or need any clarifications.

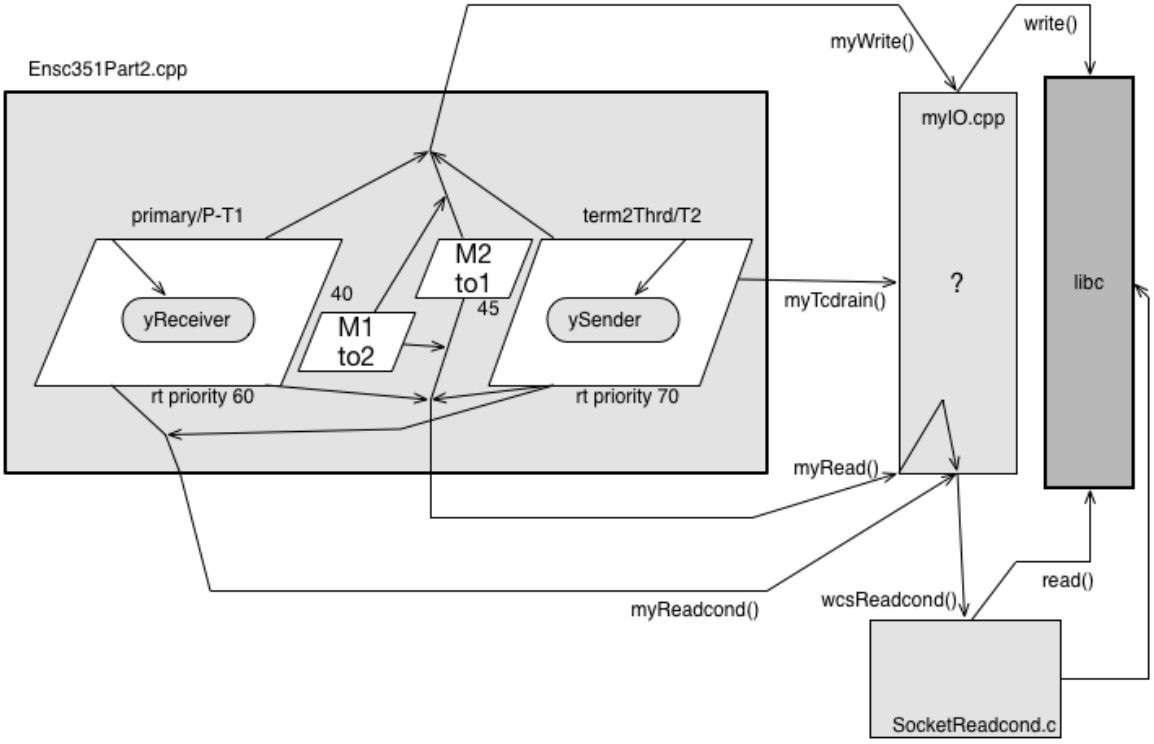

Figure 2: Collaboration Graph showing function calls for socketpair communication