ENSC 351: Real-time and Embedded Systems

Craig Scratchley, Fall 2021

Multipart Project Part 5

you can't lock the mutex w/ interrupt service routine.

Please continue working with a partner.

Part 2 of the multipart project included programming some code for the YMODEM protocol that allowed an YMODEM file transfer to complete in some situations – it could handle, for example, some bytes of the data in a block being corrupted, but could not handle dropped characters or some other sorts and combinations of the transmission errors that can occur. In Part 5 we will expand on code derived from the Part 2, 3, and 4 solutions by handling the case when the medium becomes more evil, and will in either direction corrupt or drop bytes being transmitted and also inject glitch bytes onto the medium.

We have introduced a new thread for switching console input and aggregating console output. See the "kvm" thread in the diagram. It doesn't support a mouse, so maybe we should have just called it the "kv" thread. And it aggregates console input, so perhaps we really should have chosen a better name. Oh well, kvm is the name I've used. KVM commands are:

~1<return> -- write future keyboard data to terminal 1
~2<return> -- write future keyboard data to terminal 2
~q!<return> -- quit the entire program

Lines of input that do not start with one of these commands are forwarded to terminal 1 or 2 depending on which of these is currently selected to receive keyboard data. Terminal 2 is initially selected to receive keyboard data. The terminals, in turn, have their own commands:

&s [<filepath>]<return>
-- send file with specified path (or use the default filepath)

&r <return>
-- receive file

&c<return>
-- cancel transmission in progress

When received by a terminal, lines of input that do not start with one of these commands are transmitted to the other terminal through the medium. See the diagram "Simulator in Part 5".

StateCharts

Because of the expanded requirements in this part of the course project, including the ability described above to cancel a transmission in progress, you will need StateCharts that handle all features of the YMODEM protocol. Following the StateChart that you prepared for Part 4, I am providing you with the solution complete StateCharts. (As I believe I have mentioned, SmartState unfortunately does not always generate 100% correct code, particularly with regard to entry and exit code, and so I have provided you with StateCharts that I have tested as properly working around such problems).

Medium

For data going from TERM2 to TERM1, the Part 5 (evil) medium will corrupt (complement) the 264th byte and every 790 bytes thereafter. Also, it will drop (not forward) the 659th byte received from TERM2 and every 790 bytes thereafter. Every second time the medium corrupts a byte going in this direction, it will also inject 25 or so glitch bytes at the end of the bytes provided by the medium's read() function call.

For data going from TERM1 to TERM2, the medium will corrupt (complement) every 6th byte it gets from TERM1 and drop every 7th byte it gets from TERM1 (a byte like the 42nd is dropped and so does not need to be corrupted too). Also, before actually forwarding every 4rd byte (i.e. don't count dropped bytes), this medium will send to TERM2 a glitch byte. The first glitch byte sent will have the numeric value 0, and each subsequent glitch byte will increment the value by 1. Lastly, the medium writes to TERM2 an extra ACK after every 50th byte it gets from TERM1. The extra ACK is written not immediately, but rather is written to TERM2 after the next byte is written to TERM1.

I am providing the medium code to you, so you don't need to code this. Just before writing any characters to either terminal, the medium will write the characters to a special output file called /tmp/ymodemData.dat

Sample console input/output should look something like the following (complementing, dropping, and insertion of glitch bytes, etc. is just to give you the idea, this is not what you should actually get). The italic lines in *green* are typed into the Eclipse console via your keyboard. Comments are in **bold blue**.

```
KVM FUNCTION BEGINS (INPUT ROUTED TO terminal 2)
&s              or...   &s /etc/protocols (/etc/protocols is the default for fast simulations)
TERM 2: Will request sending of '/etc/protocols'
~1
KVM SWITCHING TO terminal 1
&r
TERM 1: Will request receiving (some debug info will probably be displayed after this)
TERM1: yReceiver result was: Done
TERM2: ySender result was: Done
~2
&s
TERM 2: Will request sending of '/etc/protocols'
~1
KVM SWITCHING TO terminal 1
&r
TERM 1: Will request receiving to '/etc/protocols'
TERM1: xReceiver result was: Done
TERM2: xSender result was: Done
Hi, how are you?
Hi, hw arGe yCu?
~2
KVM SWITCHING TO terminal 2
I'm fine, thanks. Please send me a file again.
I'm fne, thanks. Plese seCd me a file again.
~q!
KVM TERMINATING
TERM 1: inD Closed
TERM 2: inD Closed
Medium thread: TERM1's socket closed, Medium terminating
```

Note that in the above sample, all debugging output has been turned off. Some debugging output can be turned on by "#define"ing REPORT_INFO in Medium.cpp, ReceiverY.cpp, and/or SenderY.cpp. Medium.cpp is in the Ensc351Part5 application project. ReceiverY.cpp and SenderY.cpp are in the

Ensc351ymodLib library project. Both projects are in a common .zip file that you can download from Canvas. I have enabled some debugging output in the code I am providing to you. If desired, StateChart debugging output can be turned on for the StateCharts by commenting out the setDebugLog(nullptr) call in the transferCommon() function in PeerY.cpp.

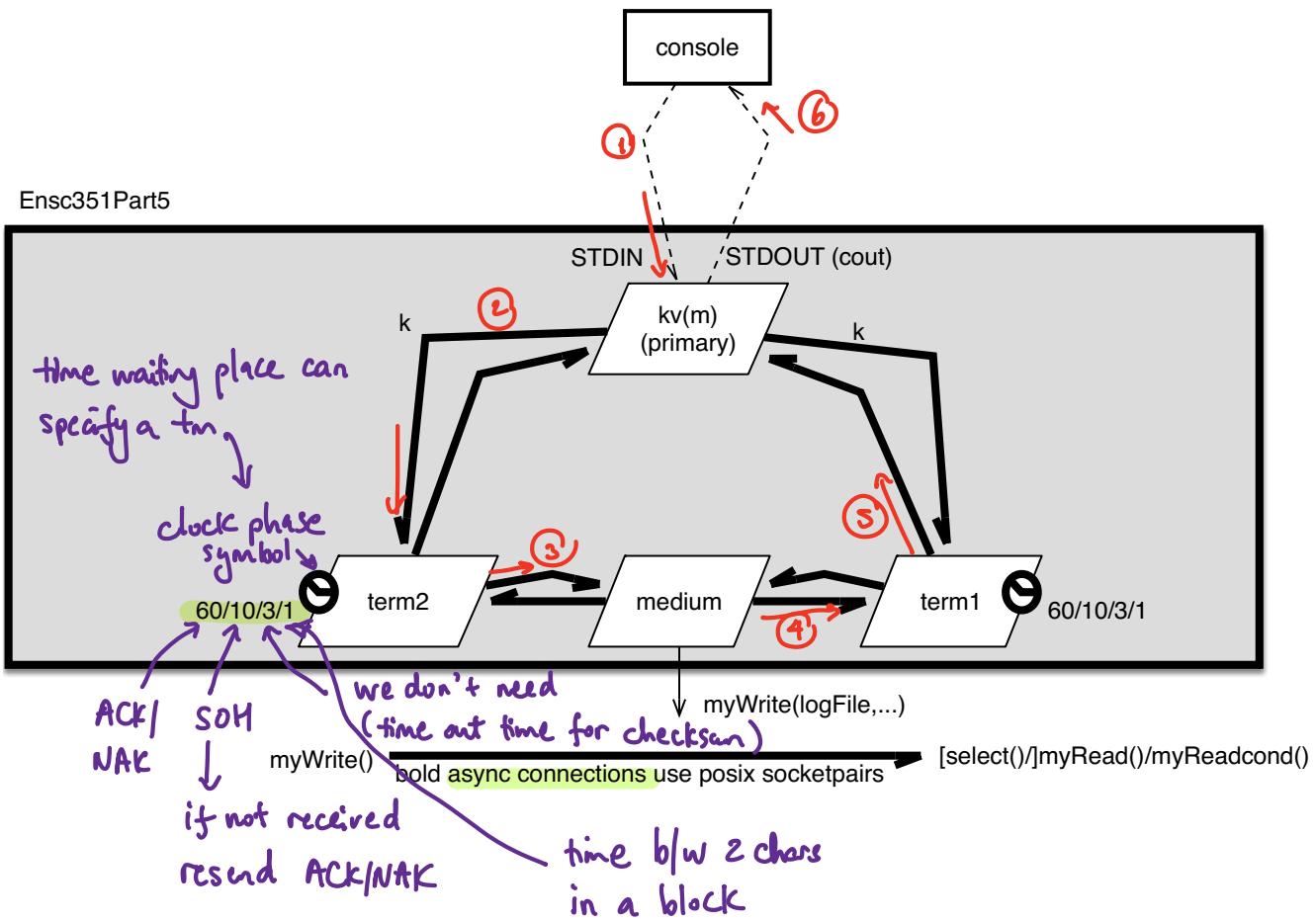
Because during file transfer the YMODEM sender and YMODEM receiver must now monitor for input from both the keyboard (via the KVM thread) and the medium, the select() function must be used somewhere in PeerY.cpp. Simple examples of the use of the select() function can be seen in a number of places in the code, for example in function *Terminal*() in the file terminal.cpp (in the library project). Note that in PeerY.cpp the select() call may need to timeout, so the last argument of the select() call there should not necessarily be NULL. Please see the documentation for the select() function such as in the man page for select().

Starting from the code I am providing to you, add to and change the code and debug your program to get it working with the evil Part 5 medium described above. The program as I have given it to you uses a kind medium similar to Part 2 and can successfully send a file, at least in the direction from TERM2 to TERM1. To switch the behaviour of the medium in each direction, see the preprocessor definitions near the top of the Medium.h file. In terms of specific changes, you will need to add/change lines of code in function transferCommon() in PeerY.cpp and function purge() in the ReceiverY class. The number of lines of code that you need to add or change is actually quite limited, but the work should make you think and hopefully you will find it interesting.

Note that in the file socketReadcond.c in the ensc351 library project, as was the case with Part2 of the multipart project, I am providing you with a function *wcsReadcond*() that is similar to the QNX *readcond*() function, and that works with socket(pair)s too. We are accessing wcsReadcond() via the myReadcond() function in myIO.cpp. Read the documentation for readcond() to learn more about this function. A simple example of the use of myReadcond() is shown in the current version of the getRestBlk() function in ReceiverY.cpp. myIO.cpp is in the Ensc351ymodLib library project.

Details on exactly which files to submit will be provided separately on CourSys.

Simulator in Part 5



A **KVM switch** (with KVM being an abbreviation for "keyboard, video and mouse") is a hardware device that allows a user to control multiple computers from one or more[1] sets of keyboards, video monitors, and mice.

POSIX func (select()) check to see if data comes from either des.

C : C char
A : ACK
N : NAK or 'N'
E : EOT
! : CAN

(C) ← ( ) do w/ receiver
[ ] ← do w/ sender
{ } ← medium
< > ← medium in 1-direction

x : corrupted
+ : added / glitched / extra
_ : dropped char

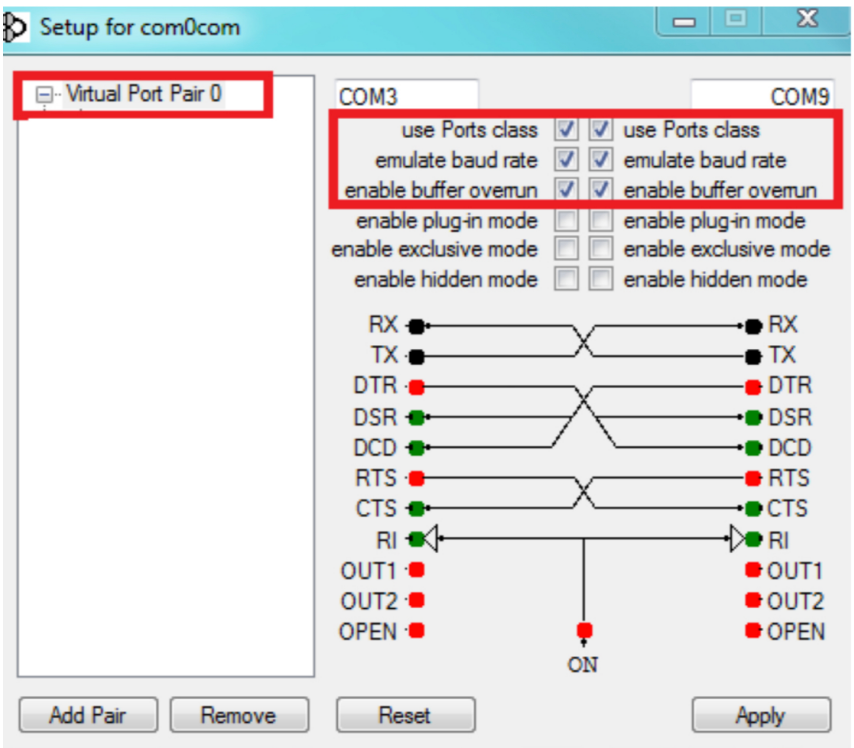
[w2] : sender written blk2
(bd2) : bad blk2
(f0) : good Blk 1st blk0
[d1] : dump 1 glitch
AxN : ACK corrupted to NAK
r16 : resend blk16
{+A} : extra ACK
<+35> : purge() to dump glitches
dumpglitches() used before CRC
{4+0} : glitches in the other direction.

getRestBlk() calls purge()

Hardware:

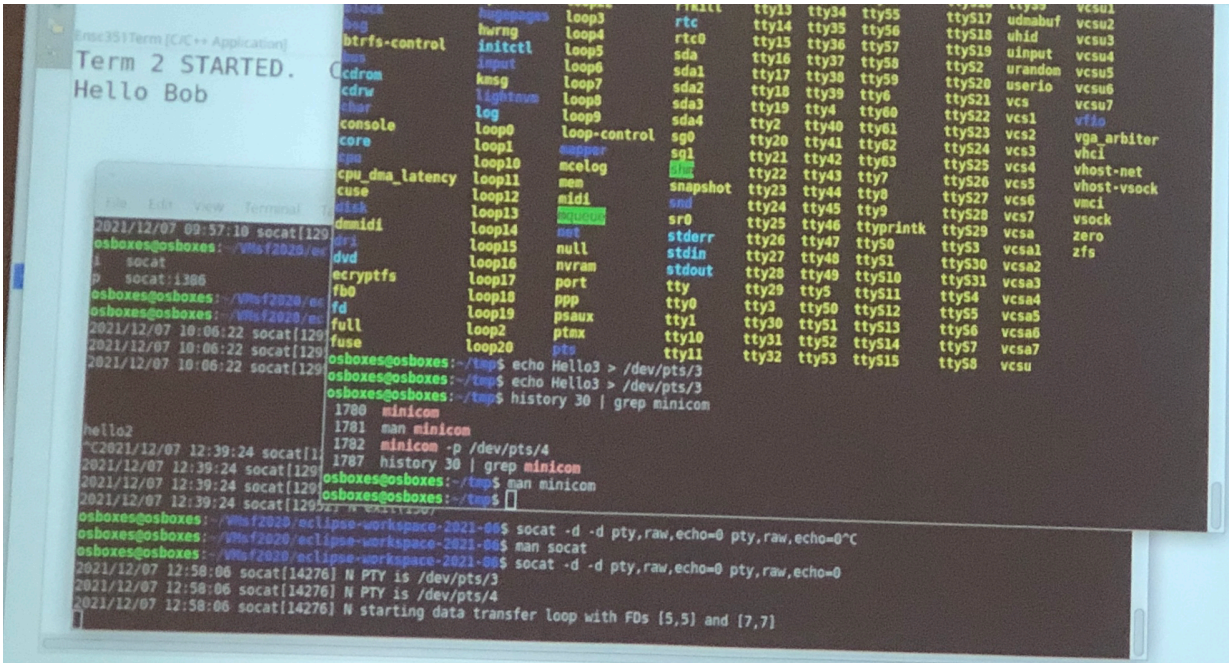
Serial Port 2 CNCA() → Virtual Machine

Serial Port 4 COM() → PC



cmd

echo Hello world	Hello world
pwd (print working directory)	/home/osboxes/tmp
ls	protocols
echo Hello myfile	myfile protocols
more myfile	Hello
echo Hello > /dev/pts/3	
ps	
which socat	
:	
: ← create one or multiple files	
cat myfile myfile2	
bg (restart a stopped background process)	



Ensc 351 Term. cpp.

output console:

Term2 STARTED. Connected to path

/dev/pts/3

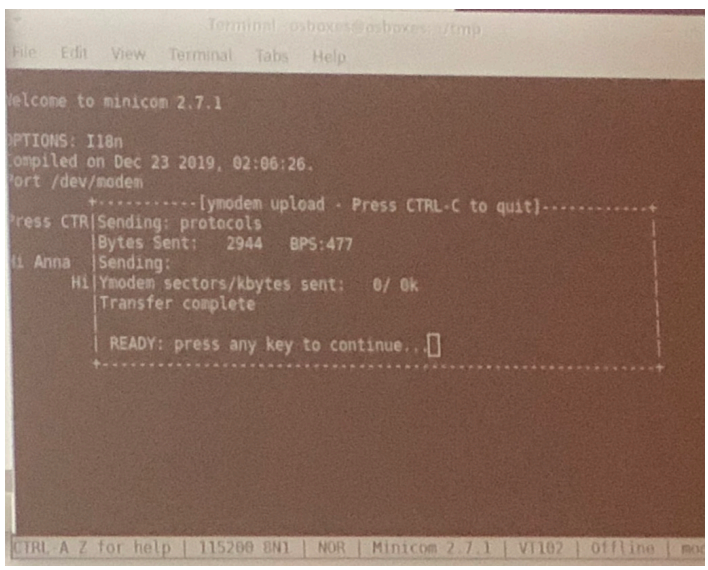
Hello

Goodbyte□□e Hi Anna

Hi again Anna

ls

terminal: osboxes@osboxes:~/tmp



remove buckets

rb > /dev/pts/4 < /dev/pts/4

Kvm.cpp

```
/*
 * Kvm.cpp
 *
 *      Author: wcs
 */

#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/select.h>
#include "Linemax.h"
#include "myIO.h"
#include "Kvm.h"
#include "VNPE.h"
#include "AtomicCOUT.h"

using namespace std;

//KVM input commands
#define KVM_TERM1_C    "~1\n"
#define KVM_TERM2_C    "~2\n"
#define KVM_QUIT_C     "~q!\n"

void Kvm(int d[2])
{
    // we could generalize to a kvm supporting an arbitrary number
of terminals.
    int term_num = TERM2; // initially terminal 2 selected
    COUT << "KVM FUNCTION BEGINS (INPUT ROUTED TO terminal " <<
(term_num + 1) << ")" << endl;

    fd_set set;
    FD_ZERO(&set);
    int max_fd_t = max(d[TERM1], d[TERM2]);
    int max_fd = max(max_fd_t, STDIN_FILENO)+1;

    char buf[LINEMAX];
    while(1) {

        FD_SET(STDIN_FILENO, &set);  //stdin
        FD_SET(d[TERM1], &set);
        FD_SET(d[TERM2], &set);
        int rv = PE(select(max_fd, &set, NULL, NULL, NULL));
```

```
      if( rv == 0 ) {
         CERR << "select() should not timeout" << endl;
         exit(EXIT_FAILURE);
      } else {
         if( FD_ISSET(STDIN_FILENO, &set) ) {
            //read the keyboard info into a buffer
            //replaces  cin>>buf;
            // should we do 'cin.getline(buf, LINEMAX_SAFE)' and
use cin.gcount()?
            int numBytesRead = PE(read(STDIN_FILENO, buf,
LINEMAX_SAFE));
            buf[numBytesRead] = 0;

            // best to first check for "~", and then check for
rest of command
            if( strcmp( buf, KVM_QUIT_C ) == 0) {
                           COUT << "KVM TERMINATING" << endl;
                           break;
            } else if( strcmp( buf, KVM_TERM1_C ) == 0) {
               COUT << "KVM SWITCHING TO terminal 1" << endl;
               term_num = TERM1;
            } else if( strcmp( buf, KVM_TERM2_C ) == 0) {
               COUT << "KVM SWITCHING TO terminal 2" << endl;
               term_num = TERM2;
            } else {
               //route keyboard input to selected terminal
               PE_NOT(myWrite(d[term_num], buf, numBytesRead),
numBytesRead); // strlen(buf)+1
            }
         }

         if ( FD_ISSET(d[TERM1], &set) ) {
            //kvm simply displays input sent from terminal 1
            int numBytesRead = PE(myRead(d[TERM1], buf,
LINEMAX_SAFE));
            buf[numBytesRead] = 0;
            COUT << buf << flush;
         }
         if ( FD_ISSET(d[TERM2], &set) ) {
            //kvm simply displays input sent from terminal 2
            int numBytesRead = PE(myRead(d[TERM2], buf,
LINEMAX_SAFE));
            buf[numBytesRead] = 0;
```

```
            COUT << buf << flush;
        }
    }
    }
    return;
}
```

```
// Solution to ENSC 351 Part 5.  Prepared by:
//      - Craig Scratchley, Simon Fraser University
//      - Zhenwang Yao

//#include <sys/types.h>
//#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <pthread.h>
#include <thread>
#include <iostream>

#include "myIO.h"
#include "Medium.h"
#include "Linemax.h"
#include "terminal.h"
#include "Kvm.h"
#include "VNPE.h"
#include "SocketReadcond.h"

using namespace std;

enum  {TERM_SIDE, OTHER_SIDE};

static int fdaSktPrTermMed[2][2];   //Socket Pairs between
terminals and Medium
static int fdaSktPrTermKvm[2][2];   //  "      " between
terminals and kvm

//kvm thread, handles all keyboard input and routes it to the
selected terminal
void kvmFunc() {
    int d[2];
    d[TERM2] = fdaSktPrTermKvm[TERM2][OTHER_SIDE];
    d[TERM1] = fdaSktPrTermKvm[TERM1][OTHER_SIDE];

    Kvm(d);
    PE(myClose(d[TERM1]));
    PE(myClose(d[TERM2]));
    return;
}
```

```
//terminal thread
//at least 2 terminal threads are required for a file transfer on
a single computer
void termFunc(int termNum)
{
    int mediumD, inD, outD;
    mediumD = fdaSktPrTermMed[termNum][TERM_SIDE];
    //mediumD = open("/dev/ser2", O_RDWR);
    inD = outD = fdaSktPrTermKvm[termNum][TERM_SIDE];
//     cout << "Term " << termNum << " THREAD STARTED" << endl;

    Terminal(termNum + 1, inD, outD, mediumD);
    PE(myClose(mediumD));
}

void mediumFunc(void)
{
    Medium medium(fdaSktPrTermMed[TERM1]
[OTHER_SIDE],fdaSktPrTermMed[TERM2][OTHER_SIDE],
"ymodemData.dat");
    medium.start();
}

int Ensc351Part5()
{
    cout.precision(2);
    // PE_0(pthread_setname_np(pthread_self(), "P-KVM")); // give
the primary thread (does kvm) a name

    // lower the priority of the primary thread to 4
//  PE_EOK(pthread_setschedprio(pthread_self(), 4));

    //Create and wire socket pairs
    // creating socket pair between terminal1 and Medium
    PE(mySocketpair(AF_LOCAL, SOCK_STREAM, 0,
fdaSktPrTermMed[TERM1]));

    // creating socket pair between terminal2 and Medium
    PE(mySocketpair(AF_LOCAL, SOCK_STREAM, 0,
fdaSktPrTermMed[TERM2]));

    // opening kvm-term2 socket pair
```

```
   PE(mySocketpair(AF_LOCAL, SOCK_STREAM, 0,
fdaSktPrTermKvm[TERM2]));

   // opening kvm-term1 socket pair
   PE(mySocketpair(AF_LOCAL, SOCK_STREAM, 0,
fdaSktPrTermKvm[TERM1]));

   //Create 3 threads

   thread term1Thrd(termFunc, TERM1);
   thread term2Thrd(termFunc, TERM2);

   // ***** create thread for medium *****
   thread mediumThrd(mediumFunc);

// PE_0(pthread_setname_np(pthread_self(), "P-K"));
   kvmFunc();

   term2Thrd.join();
   term1Thrd.join();
   // ***** join with thread for medium *****
   mediumThrd.join();

   return EXIT_SUCCESS;
}
```

terminal . cpp

```
// ENSC 351 Assignment 3.  2020  Prepared by:
//      - Craig Scratchley, Simon Fraser University
//        with help from:
//      - Zhenwang Yao
//     - Phoenix Yuan

#include <sstream>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/select.h>
#include "myIO.h"
#include "SenderY.h"
#include "ReceiverY.h"
#include "Linemax.h"
#include "VNPE.h"
#include "AtomicCOUT.h"

using namespace std;

//terminal input commands
#define SEND_C      "&s"
#define RECV_C      "&r"

#define TERM_QUIT_C    "&q!"

//function used by the terminal threads, process input from the
medium
// return true when terminal should terminate.
bool MediumReady(int mediumD, int outD)
{
   char bytesReceived[BUF_SZ];

   int numOfByteReceived = PE(myRead(mediumD, bytesReceived,
BUF_SZ));
   if (numOfByteReceived == 0) {
      COUT << "MediumReady finds Medium Descriptor Closed" <<
endl;
      return true;
   }
   PE_NOT(myWrite(outD, bytesReceived, numOfByteReceived),
numOfByteReceived);
   return false;
```

```
}

//function used by the terminal threads, process input from the
KeyBoard
// return true when terminal should terminate.
bool KbReady(int inD, int outD, int term, int mediumD)
{
   char bytesReceived[LINEMAX];
   //char bytesReceived[4];
   // should we make sure we get just a single line of input
   // should we separate command from file name?
   // If commands are 3 bytes long, then we can read 3 bytes for
command
   // We can introduce a (KVM) command to delay
   // when debugging, should we delay between blocks.
   int numOfByteReceived = PE(myRead(inD, bytesReceived,
LINEMAX_SAFE));
   //int numOfByteReceived = PE(myRead(inD, bytesReceived, 3));
   if (numOfByteReceived == 0) {
      COUT << "TERM " << term << ": inD Closed" << endl;
      return true;
   }
   bytesReceived[numOfByteReceived]=0; // is this needed?

   //grab command and possibly file name from input buffer
   char cmd[LINEMAX]; // longer than necessary?
   char fname[LINEMAX];
   char useCrcOption[LINEMAX]; // longer than necessary?
   int numItemsMatched = sscanf( bytesReceived, "%s %s %s", cmd,
fname, useCrcOption );
   if( numItemsMatched >= 1) {
      if (strcmp( cmd, SEND_C ) == 0) {
         //default filename
         if( numItemsMatched < 2 ) {// strlen(fname) == 0 ) {
#ifdef FAST_SIM
            strcpy(fname, "/etc/protocols");
#else
            strcpy(fname, "/etc/anacrontab");
#endif
         }
         CON_OUT(outD, "TERM " << term << ": Will request
sending of '" << fname << "'"<< endl);
          vector<const char*> iFileNames = {fname};
```

```
            SenderY ySender(iFileNames, mediumD, inD, outD);
            ySender.sendFiles();
            CON_OUT(outD, "\nTERM " << term << ": ySender result
was: " << ySender.result << endl);
            return false;
         } else if( strcmp( cmd, RECV_C ) == 0) {
            CON_OUT(outD, "TERM " << term << ": Will request
receiving."<< endl);
            ReceiverY yReceiver(mediumD, inD, outD);
            yReceiver.receiveFiles();
            CON_OUT(outD, "\nTERM " << term << ": yReceiver result
was: " << yReceiver.result << endl);
            return false;
         } else if( strcmp( cmd, TERM_QUIT_C ) == 0) {
            CON_OUT(outD, "TERM " << term << " TERMINATING" <<
endl);
            return true;
         }
      }
      int numOfBytesSent = 0;
      while((numOfBytesSent+=PE(myWrite(mediumD, bytesReceived +
numOfBytesSent, numOfByteReceived - numOfBytesSent))) <
numOfByteReceived)
         PE(myTcdrain(mediumD));
      return false;
}

void Terminal(int termNum, int inD, int outD, int mediumD)

{
      // empty any amount of data that might be previously buffered
      const int dumpBufSz = 20;
      char buf[dumpBufSz];
      int bytesRead;
      while (dumpBufSz == (bytesRead = PE(myReadcond(mediumD, buf,
dumpBufSz, 0, 0, 0))));
      // the above is dumpGlitches -- avoid this duplicated code.

      bool finished = false;

      fd_set set; // Posix lib , C language
      FD_ZERO(&set);
      do
```

```
    {
        FD_SET(mediumD, &set);
        FD_SET(inD, &set);
        int rv = PE(select( max(mediumD,
                            inD)+1, &set, NULL, NULL, NULL ));
        if( rv == 0 ) {
            // timeout occurred
            CERR << "This peer term (" << termNum << ") should not
timeout" << endl;
            exit (EXIT_FAILURE);
        } else {
            if( FD_ISSET( mediumD, &set ) ) {//check if mediumD is in set
                //route message from medium back to screen
                finished = MediumReady(mediumD, outD);
            };
            if( FD_ISSET( inD, &set ) ) { //check if input des is in set
                finished |= KbReady(inD, outD, termNum, mediumD);
            };
        }
    }  while(!finished);
    return;
}
```

tells how many des falls into active category, If not active, there is a tm

max number of des you can be working w/

input des

set of des you want to read from

when des report exception

we can pass a set into select()

des you want to write to

where you can indicate a tm

"set is a pair of des"

```
//
============================================================
==========
// File Name   : PeerY.cpp
// Version     : November, 2021
// Description : Starting point for ENSC 351 Project Part 5
// Original portions Copyright (c) 2021 Craig Scratchley  (wcs AT
sfu DOT ca)
//
============================================================
==========

#include "PeerY.h"

#include <sys/time.h>
#include <arpa/inet.h> // for htons() -- not available with MinGW

#include "VNPE.h"
#include "Linemax.h"
#include "myIO.h"
#include "AtomicCOUT.h"

using namespace std;
using namespace smartstate;

PeerY::
PeerY(int d, char left, char right, const char *smLogN, int
conInD, int conOutD)
:result("ResultNotSet"),
 errCnt(0),
// KbCan(false),
// transferringFileD(-1),  // will need to be updated
 mediumD(d),
 logLeft(left),
 logRight(right),
 smLogName(smLogN),
 consoleInId(conInD),
 consoleOutId(conOutD),
 reportInfo(false),
 absoluteTimeout(0),
 holdTimeout(0)
{
   struct timeval tvNow;
```

```
    PE(gettimeofday(&tvNow, NULL));
    sec_start = tvNow.tv_sec;
}

//Send a byte to the remote peer across the medium
void
PeerY::
sendByte(uint8_t byte)
{
    if (reportInfo) {
        //*** remove all but last of this block ***
        char displayByte;
         if (byte == NAK)
             displayByte = 'N';
         else if (byte == ACK)
             displayByte = 'A';
         else if (byte == EOT)
             displayByte = 'E';
         else
             displayByte = byte;
         COUT << logLeft << displayByte << logRight << flush;
    }
    PE_NOT(myWrite(mediumD, &byte, sizeof(byte)), sizeof(byte));
}

void
PeerY::
transferCommon(std::shared_ptr<StateMgr> mySM, bool
reportInfoParam)
{
    reportInfo = reportInfoParam;
    /*
    // use this code to send stateChart logging information to a
file.
    ofstream smLogFile; // need '#include <fstream>' above
    smLogFile.open(smLogName, ios::binary|ios::trunc);
    if(!smLogFile.is_open()) {
        CERR << "Error opening sender state chart log file named:
" << smLogName << endl;
        exit(EXIT_FAILURE);
    }
    mySM->setDebugLog(&smLogFile);
    // */
```

```
   // comment out the line below if you want to see logging
information which will,
   // by default, go to cout.
   mySM->setDebugLog(nullptr); // this will affect both peers.
Is this okay?

   mySM->start();

   /* ******** You may need to add code here ******** */

   struct timeval tv;

   while(mySM->isRunning()) {
      // ************* this loop is going to need more work
************
      tv.tv_sec=0;
      long long int now = elapsed_usecs();
       if (now >= absoluteTimeout) {
           //...
           mySM->postEvent(TM);
       } else {
           // ...
           /****/ {
               //read character from medium
            char byte;
            unsigned timeout = (absoluteTimeout - now) / 1000 /
100; // tenths of seconds
           if (PE(myReadcond(mediumD, &byte, 1, 1, timeout,
timeout))) {
                if (reportInfo) {
                    char displayByte;
                    if      (byte == NAK)
                        displayByte = 'N';
                    else if (byte == ACK)
                        displayByte = 'A';
                       else if (byte == SOH)
                           displayByte = 'S';
                       else if (byte == EOT)
                           displayByte = 'E';
                       else if (byte == CAN)
                           displayByte = '!';
                       else if (byte == 0)
```

abs → relative

ask for 1 byte. by 0 byte coming and we ran so goes to else {}

```
                            displayByte = '0';
                        else
                            displayByte = byte;

                        COUT << logLeft << 1.0*timeout/10 << ":" <<
(int)(unsigned char) byte << ":" << displayByte << logRight <<
flush;
                    }
                    mySM->postEvent(SER, byte);
                }
                else { // This won't be needed later because timeout
will occur after the select() function.
                    if (reportInfo)
                        COUT << logLeft << 1.0*timeout/10 <<
logRight << flush;
                    mySM->postEvent(TM); // see note 3 lines above.
                }
            }
        }
    }
//     smLogFile.close();
}

// returns microseconds elapsed since this peer was constructed
(within 1 second)
long long int
PeerY::
elapsed_usecs()
{
    struct timeval tvNow;
    PE(gettimeofday(&tvNow, NULL));
    /*_CSTD */ time_t   tv_sec = tvNow.tv_sec;
    return (tv_sec - sec_start) * (long long int) MILLION +
tvNow.tv_usec; // casting needed?
}

/*
set a timeout time at an absolute time timeoutUnits into
the future. That is, determine an absolute time to be used
for the next one or more XMODEM timeouts by adding
timeoutUnits to the elapsed time.
*/
void
```

chap 4.3.3: wait_until()

absolute vs. relative tm.
↑ used in func
Craig used
relative tm
you don't want to wait up to 60s just b/c a glitch. should wait for, says 30s

```
PeerY::
tm(int timeoutUnits)
{
    absoluteTimeout = elapsed_usecs() + timeoutUnits *
uSECS_PER_UNIT;
}
```

relative → abs

```
/* make the absolute timeout earlier by reductionUnits */
void
PeerY::
tmRed(int unitsToReduce)
{
    absoluteTimeout -= (unitsToReduce * uSECS_PER_UNIT);
}

/*
Store the current absolute timeout, and create a temporary
absolute timeout timeoutUnits into the future.
*/
void
PeerY::
tmPush(int timeoutUnits)
{
    holdTimeout = absoluteTimeout;
    absoluteTimeout = elapsed_usecs() + timeoutUnits *
uSECS_PER_UNIT;
}

/*
Discard the temporary absolute timeout and revert to the
stored absolute timeout
*/
void
PeerY::
tmPop()
{
    absoluteTimeout = holdTimeout;
}
```

medium. h

```
/*
 * Medium.h
 *
 *      Author: Craig Scratchley
 *      Copyright(c) 2013 (Fall) Craig Scratchley
 */

#ifndef MEDIUM_H_
#define MEDIUM_H_

//comment out "define USE_PART2A_R1_TO_S2"
// to use the final terminal 1->2 medium. It can drop chars,
glitch, etc.
#define USE_PART2A_R1_TO_S2

//comment out "define USE_PART2A_S2_TO_R1"
// to use the final terminal 2->1 medium. It can drop chars,
glitch, etc.
#define USE_PART2A_S2_TO_R1

class Medium {
public:
   Medium(int d1, int d2, const char *fname);
   virtual ~Medium();

   void start();

private:
   int Term1D;  // descriptor for Term1
   int Term2D;  // descriptor for Term2
   const char* logFileName;
   int logFileD;   // descriptor for log file

#ifndef USE_PART2A_S2_TO_R1
   unsigned int fromT2ByteCount;
   unsigned int corruptThreshold;
   unsigned int dropThreshold;
   bool fromT2Glitch;
#else
   int byteCount;
#endif

#ifndef USE_PART2A_R1_TO_S2
```

```
   unsigned char glitchCount;
   unsigned int fromT1ByteCount;
   unsigned int sentCount;
#else
   int ACKreceived;
   int ACKforwarded;
#endif
   bool sendExtraAck;

   bool MsgFromTerm1();
   bool MsgFromTerm2();
};

#endif /* MEDIUM_H_ */
```

```
/*
 * Medium.cpp
 *
 *      Author: Craig Scratchley
 *      Copyright(c) 2014 (Spring) Craig Scratchley
 */

#include <fcntl.h>
#include <unistd.h> // for write()
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include <sys/select.h>
#include "Medium.h"
#include "myIO.h"
#include "VNPE.h"
#include "AtomicCOUT.h"

#include "PeerY.h"

// Uncomment the line below to turn on debugging output from the
medium
#define REPORT_INFO

//This medium contains both kind and non-kind versions.
//  See Medium.h

#ifndef USE_PART2A_S2_TO_R1
#define T2toT1_CORRUPT_FIRST_BYTE   264
#define T2toT1_CORRUPT_BYTE         790//current algorithm only
works when
                                  // T2toT1_CORRUPT_BYTE is greater
than bufSz. Also applies to T2toT1_DROP_BYTE
#define T2toT1_DROP_FIRST_BYTE      659
#define T2toT1_DROP_BYTE         790

#define T2toT1_GLITCH_BYTES         35
#else
#define T2toT1_CORRUPT_BYTE          395
#endif

#ifndef USE_PART2A_R1_TO_S2
#define T1toT2_CORRUPT_BYTE      6 //4 //6
```

```
#define T1toT2_DROP_BYTE     7 //6 //7
#define T1toT2_GLITCH_BYTE       4 //2 //3
#define   T1toT2_SEND_GLITCH_ACK 50
#endif

using namespace std;

ssize_t mediumRead( int fildes, void* buf, size_t nbyte )
{
 ssize_t numOfByte = myRead(fildes, buf, nbyte );
 if (numOfByte == -1 && errno == 104) // errno 104 is "Connection
reset by peer"
  numOfByte = 0; // switch errno 104 to 0 bytes read
 return numOfByte;
}

Medium::Medium(int d1, int d2, const char *fname)
:Term1D(d1), Term2D(d2), logFileName(fname)
{
#ifndef USE_PART2A_S2_TO_R1
   fromT2ByteCount = 0;
   corruptThreshold = T2toT1_CORRUPT_FIRST_BYTE;
   dropThreshold = T2toT1_DROP_FIRST_BYTE;
   fromT2Glitch = false;
#else
   byteCount = 0;
#endif

#ifndef USE_PART2A_R1_TO_S2
   glitchCount = 0;
   fromT1ByteCount = 0;
   sentCount = 0;
#else
   ACKforwarded = 0;
   ACKreceived = 0;
#endif
   sendExtraAck = false;
   logFileD = -1;
   // mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
   //logFileD = PE2(creat(logFileName, mode), logFileName);
}

Medium::~Medium() {
```

```
}

bool Medium::MsgFromTerm2()
{
#ifndef USE_PART2A_S2_TO_R1
   const int mediumBufSz = 160; // allow for enough glitch bytes
   uint8_t bytesReceived[mediumBufSz];
   memset(bytesReceived, CAN, mediumBufSz); // initialize buffer,
so glitches will be deterministic
   int fromT2GlitchBytes=0;

   int numOfByteReceived = PE(myRead(Term2D, bytesReceived, 70));
   if (numOfByteReceived == 0) {
      COUT << "Medium thread: TERM2's socket closed, Medium
terminating" << endl;
      return true;
   }
   fromT2ByteCount += numOfByteReceived;

   // deal with corrupt byte
   if( fromT2ByteCount >= corruptThreshold ) {
      int byteToCorrupt = numOfByteReceived-1-(fromT2ByteCount-
corruptThreshold);
      bytesReceived[byteToCorrupt] =
~bytesReceived[byteToCorrupt];
#ifdef REPORT_INFO
      COUT << "<" << byteToCorrupt << "x" << fromT2ByteCount -
(numOfByteReceived - 1 - byteToCorrupt) << ">" << flush;
#endif
      corruptThreshold += T2toT1_CORRUPT_BYTE;
      if (fromT2Glitch) {
         fromT2GlitchBytes = T2toT1_GLITCH_BYTES;
#ifdef REPORT_INFO
      COUT << "<+" << T2toT1_GLITCH_BYTES << ">" << flush;
#endif
      }
      else
         fromT2GlitchBytes = 0;
      fromT2Glitch = !fromT2Glitch;
   }

   //deal with drop byte, if only 1 byte is being sent, then the
operation does not
```

```
   // matter, since 1 less byte will be sent in the end
   int drop = 0;
   if (fromT2ByteCount >= dropThreshold ) {
      int byteToDrop = numOfByteReceived-1-(fromT2ByteCount-
dropThreshold);
      memmove( bytesReceived + sizeof(char)*byteToDrop,
         bytesReceived + sizeof(char)*(byteToDrop+1),
(mediumBufSz-1-byteToDrop));
#ifdef REPORT_INFO
      COUT << "<" << byteToDrop << "-" << fromT2ByteCount -
(numOfByteReceived - 1 - byteToDrop) << ">" << flush;
#endif
      dropThreshold += T2toT1_DROP_BYTE;

      //send 1 less byte
      drop = 1;
   }
   // sometimes inject glitches at this time on Term2D
   int fromT2bytesToWrite = numOfByteReceived - drop +
fromT2GlitchBytes;

   PE_NOT(write(logFileD, bytesReceived, 1), 1);
   //Forward the bytes to RECEIVER,
   PE_NOT(myWrite(Term1D, bytesReceived, 1), 1);

   if (sendExtraAck) {
#ifdef REPORT_INFO
         COUT << "{" << "+A" << "}" << flush;
#endif
      uint8_t buffer = ACK;
      PE_NOT(write(logFileD, &buffer, 1), 1);
      //Forward the buffer to term2,
      PE_NOT(myWrite(Term2D, &buffer, 1), 1);

      sendExtraAck = false;
   }

   PE_NOT(write(logFileD, &bytesReceived[1],
fromT2bytesToWrite-1), fromT2bytesToWrite-1);
   //Forward the bytes to Terminal 1,
   PE_NOT(myWrite(Term1D, &bytesReceived[1],
fromT2bytesToWrite-1), fromT2bytesToWrite-1);
```

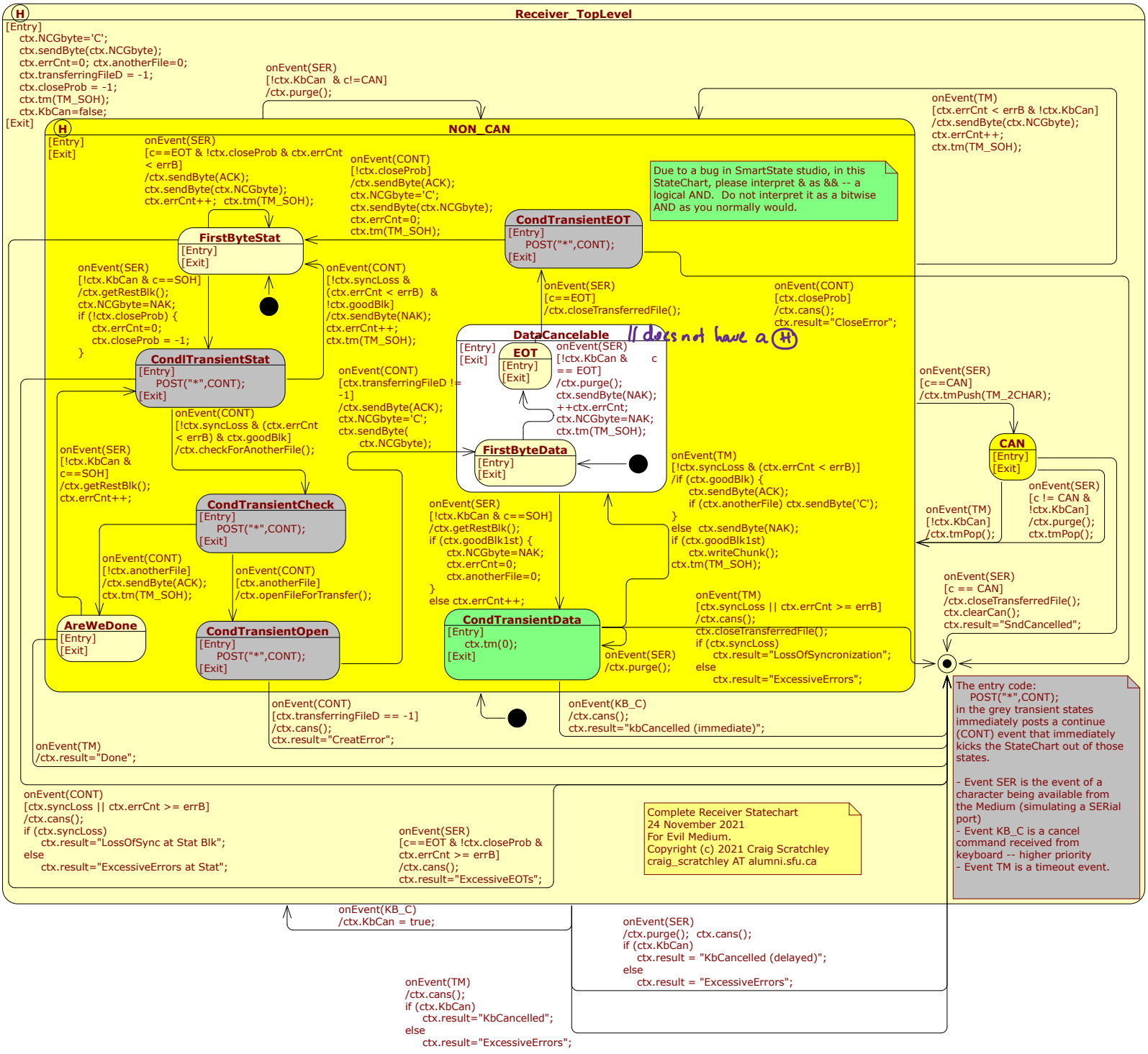
```
// PE_NOT(write(logFile, bytesReceived, fromT2bytesToWrite),
fromT2bytesToWrite);
// PE_NOT(myWrite(Term1D, bytesReceived, fromT2bytesToWrite),
fromT2bytesToWrite);

   return false;
#else // USE_PART2A_S2_TO_R1 is defined
    blkT bytesReceived; // ?
    int numOfBytesReceived;
    int byteToCorrupt;

    if (!(numOfBytesReceived = PE(mediumRead(Term2D,
bytesReceived, 1)))) {
        COUT << "Medium thread: TERM2's socket closed, Medium
terminating" << endl;
        return true;
    }
    byteCount += numOfBytesReceived;

    PE_NOT(myWrite(logFileD, bytesReceived, numOfBytesReceived),
numOfBytesReceived);
    //Forward the bytes to Term1 (usually RECEIVER),
    PE_NOT(myWrite(Term1D,   bytesReceived, numOfBytesReceived),
numOfBytesReceived);

    if(bytesReceived[0] == CAN) {
           numOfBytesReceived = PE_NOT(mediumRead(Term2D,
bytesReceived, CAN_LEN - 1), CAN_LEN - 1);
           // byteCount += numOfBytesReceived;
           PE_NOT(myWrite(logFileD, bytesReceived,
numOfBytesReceived), numOfBytesReceived);
           //Forward the bytes to Term1 (usually RECEIVER),
           PE_NOT(myWrite(Term1D,   bytesReceived,
numOfBytesReceived), numOfBytesReceived);
    }
    else if (bytesReceived[0] == SOH) {
        if (sendExtraAck) {
    #ifdef REPORT_INFO
                COUT << "{" << "+A" << "}" << flush;
    #endif
            uint8_t buffer = ACK;
           PE_NOT(myWrite(logFileD, &buffer, 1), 1);
           //Write the byte to term2,
```

```
            PE_NOT(myWrite(Term2D, &buffer, 1), 1);

            sendExtraAck = false;
        }

        numOfBytesReceived = PE(mediumRead(Term2D, bytesReceived,
(BLK_SZ_CRC) - numOfBytesReceived));

        byteCount += numOfBytesReceived;
        if (byteCount >= T2toT1_CORRUPT_BYTE) {
            byteCount = byteCount - T2toT1_CORRUPT_BYTE;  // how
large could byteCount end up? (CB - 1) + 133 - CB = 132
            byteToCorrupt = numOfBytesReceived - byteCount; //
how small could byteToCorrupt be?
            if (byteToCorrupt < numOfBytesReceived) {
                bytesReceived[byteToCorrupt] = (255 -
bytesReceived[byteToCorrupt]);
        #ifdef REPORT_INFO
                COUT << "<" << byteToCorrupt << "x>" << flush;
        #endif
            }
        }

        PE_NOT(myWrite(logFileD, &bytesReceived,
numOfBytesReceived), numOfBytesReceived);
        //Forward the bytes to Term1 (RECEIVER),
        PE_NOT(myWrite(Term1D, &bytesReceived,
numOfBytesReceived), numOfBytesReceived);
    }
    return false;
#endif
}

bool Medium::MsgFromTerm1()
{
#ifndef USE_PART2A_R1_TO_S2
   char byteReceived;

   int numOfByteReceived = PE(myRead(Term1D, &byteReceived,
sizeof(byteReceived)));
   if (numOfByteReceived == 0) {
      COUT << "Medium thread: TERM1's socket closed, Medium
terminating" << endl;
```

```
      return true;
   }
   fromT1ByteCount+= numOfByteReceived;

   if ( fromT1ByteCount % T1toT2_DROP_BYTE == 0 ) {
      // sends nothing
#ifdef REPORT_INFO
      COUT << "{" << (int)(unsigned char)byteReceived << "-" <<
fromT1ByteCount << "}" << flush;
#endif
   } else {
      sentCount++;
      if( sentCount % T1toT2_GLITCH_BYTE == 0 ) {
#ifdef REPORT_INFO
         COUT << "{" << sentCount << "+" << (int)glitchCount <<
"}" << flush;
#endif
         PE_NOT(write(logFileD,&glitchCount,
sizeof(glitchCount)), sizeof(byteReceived));
         PE_NOT(myWrite(Term2D, &glitchCount,
sizeof(glitchCount)), sizeof(byteReceived));
         glitchCount++;
      }
      if( fromT1ByteCount % T1toT2_CORRUPT_BYTE == 0 ) {
#ifdef REPORT_INFO
         COUT << "{" << (int)(unsigned char)byteReceived << "xN"
<< "}" << flush;
#endif
         byteReceived = NAK;
      }
      if( fromT1ByteCount % T1toT2_SEND_GLITCH_ACK == 0 ) {
         sendExtraAck = true;
      }
      // should we sometimes inject glitches at this time on
Term1D?
      PE_NOT(write(logFileD, &byteReceived,
sizeof(byteReceived)), sizeof(byteReceived));
      PE_NOT(myWrite(Term2D, &byteReceived,
sizeof(byteReceived)), sizeof(byteReceived));
   }
   return false;
#else // USE_PART2A_R1_TO_S2 is defined
   uint8_t buffer[CAN_LEN];
```

```
   int numOfByte = PE(myRead(Term1D, buffer, CAN_LEN));
   if (numOfByte == 0) {
      COUT << "Medium thread: TERM1's socket closed, Medium
terminating" << endl;
      return true;
   }

   /*note that we record the errors in ACK so that we can check
in the log file*/
   if(buffer[0]==ACK)
   {
      ACKreceived++;

      if((ACKreceived%9)==0)
      {
         ACKreceived = 0;
         buffer[0]=NAK;
#ifdef REPORT_INFO
         COUT << "{" << "AxN" << "}" << flush;
#endif
      }
      else/*actually forwarded ACKs*/
      {
         ACKforwarded++;

         if((ACKforwarded%6)==0)/*Note that this extra ACK is
not an ACK forwarded from receiver to the sender, so we don't
increment ACKforwarded*/
         {
            ACKforwarded = 0;
            sendExtraAck = true;
         }
      }
   }

   PE_NOT(write(logFileD, buffer, numOfByte), numOfByte);

   //Forward the buffer to term2,
   PE_NOT(myWrite(Term2D, buffer, numOfByte), numOfByte);
   return false;
#endif
}
```

```
void Medium::start()
{
   mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
   logFileD = PE2(creat(logFileName, mode), logFileName);
   fd_set cset;
   FD_ZERO(&cset);

   bool finished=false;
   while(!finished) {

      //note that the file descriptor bitmap must be reset every
time
      FD_SET(Term1D, &cset);
      FD_SET(Term2D, &cset);

      int rv = PE(select( max(Term2D,
                      Term1D)+1, &cset, NULL, NULL, NULL ));
      if( rv == 0 ) {
         // timeout occurred
         CERR << "The medium should not timeout" << endl;
         exit (EXIT_FAILURE);
      } else {
         if( FD_ISSET( Term1D, &cset ) ) {
            finished = MsgFromTerm1(); //Term1D,Term2D);
         }
         if( FD_ISSET( Term2D, &cset ) ) {
            finished = MsgFromTerm2(); //Term1D,Term2D);
         }
      }
   };
   PE(close(logFileD));
   PE(myClose(Term1D));
   PE(myClose(Term2D));
}
```


Receiver_TopLevel
[Entry]
ctx.NCGbyte='C';
ctx.sendByte(ctx.NCGbyte);
ctx.errCnt=0; ctx.anotherFile=0;
ctx.transferringFileD = -1;
ctx.closeProb = -1;
ctx.tm(TM_SOH);
ctx.KbCan=false;
[Exit]
onEvent(SER)
[!ctx.KbCan & c!=CAN]
/ctx.purge();
onEvent(TM)
[ctx.errCnt < errB & !ctx.KbCan]
/ctx.sendByte(ctx.NCGbyte);
ctx.errCnt++;
ctx.tm(TM_SOH);
NON_CAN
[Entry]
[Exit]
onEvent(SER)
[c==EOT & !ctx.closeProb & ctx.errCnt < errB]
/ctx.sendByte(ACK);
ctx.sendByte(ctx.NCGbyte);
ctx.errCnt++; ctx.tm(TM_SOH);
onEvent(CONT)
[!ctx.closeProb]
/ctx.sendByte(ACK);
ctx.NCGbyte='C';
ctx.sendByte(ctx.NCGbyte);
ctx.errCnt=0;
ctx.tm(TM_SOH);
Due to a bug in SmartState studio, in this StateChart, please interpret & as && -- a logical AND. Do not interpret it as a bitwise AND as you normally would.
CondTransientEOT
[Entry]
POST("*",CONT);
[Exit]
FirstByteStat
[Entry]
[Exit]
onEvent(SER)
[!ctx.KbCan & c==SOH]
/ctx.getRestBlk();
ctx.NCGbyte=NAK;
if (!ctx.closeProb) {
ctx.errCnt=0;
ctx.closeProb = -1;
}
onEvent(CONT)
[!ctx.syncLoss & (ctx.errCnt < errB) & !ctx.goodBlk]
/ctx.sendByte(NAK);
ctx.errCnt++;
ctx.tm(TM_SOH);
onEvent(SER)
[c==EOT]
/ctx.closeTransferredFile();
onEvent(CONT)
[ctx.closeProb]
/ctx.cans();
ctx.result="CloseError";
DataCancelable
// does not have a H
[Entry]
[Exit]
EOT
[Entry]
[Exit]
onEvent(SER)
[!ctx.KbCan & c == EOT]
/ctx.purge();
ctx.sendByte(NAK);
++ctx.errCnt;
ctx.NCGbyte=NAK;
ctx.tm(TM_SOH);
CondInTransientStat
[Entry]
POST("*",CONT);
[Exit]
onEvent(CONT)
[ctx.transferringFileD != -1]
/ctx.sendByte(ACK);
ctx.NCGbyte='C';
ctx.sendByte(
ctx.NCGbyte);
onEvent(CONT)
[!ctx.syncLoss & (ctx.errCnt < errB) & ctx.goodBlk]
/ctx.checkForAnotherFile();
onEvent(SER)
[!ctx.KbCan & c==SOH]
/ctx.getRestBlk();
ctx.errCnt++;
FirstByteData
[Entry]
[Exit]
onEvent(SER)
[c==CAN]
/ctx.tmPush(TM_2CHAR);
CAN
[Entry]
[Exit]
onEvent(SER)
[c != CAN & !ctx.KbCan]
/ctx.purge();
ctx.tmPop();
onEvent(TM)
[!ctx.KbCan]
/ctx.tmPop();
onEvent(TM)
[!ctx.syncLoss & (ctx.errCnt < errB)]
/if (ctx.goodBlk) {
ctx.sendByte(ACK);
if (ctx.anotherFile) ctx.sendByte('C');
}
else ctx.sendByte(NAK);
if (ctx.goodBlk1st)
ctx.writeChunk();
ctx.tm(TM_SOH);
CondTransientCheck
[Entry]
POST("*",CONT);
[Exit]
onEvent(SER)
[!ctx.KbCan & c==SOH]
/ctx.getRestBlk();
if (ctx.goodBlk1st) {
ctx.NCGbyte=NAK;
ctx.errCnt=0;
ctx.anotherFile=0;
}
else ctx.errCnt++;
onEvent(CONT)
[!ctx.anotherFile]
/ctx.sendByte(ACK);
ctx.tm(TM_SOH);
onEvent(CONT)
[ctx.anotherFile]
/ctx.openFileForTransfer();
AreWeDone
[Entry]
[Exit]
CondTransientOpen
[Entry]
POST("*",CONT);
[Exit]
CondTransientData
[Entry]
ctx.tm(0);
[Exit]
onEvent(SER)
/ctx.purge();
onEvent(TM)
[ctx.syncLoss || ctx.errCnt >= errB]
/ctx.cans();
ctx.closeTransferredFile();
if (ctx.syncLoss)
ctx.result="LossOfSyncronization";
else
ctx.result="ExcessiveErrors";
onEvent(SER)
[c == CAN]
/ctx.closeTransferredFile();
ctx.clearCan();
ctx.result="SndCancelled";
The entry code:
POST("*",CONT);
in the grey transient states immediately posts a continue (CONT) event that immediately kicks the StateChart out of those states.
- Event SER is the event of a character being available from the Medium (simulating a SERial port)
- Event KB_C is a cancel command received from keyboard -- higher priority
- Event TM is a timeout event.
onEvent(CONT)
[ctx.transferringFileD == -1]
/ctx.cans();
ctx.result="CreatError";
onEvent(KB_C)
/ctx.cans();
ctx.result="kbCancelled (immediate)";
onEvent(TM)
/ctx.result="Done";
onEvent(CONT)
[ctx.syncLoss || ctx.errCnt >= errB]
/ctx.cans();
if (ctx.syncLoss)
ctx.result="LossOfSync at Stat Blk";
else
ctx.result="ExcessiveErrors at Stat";
onEvent(SER)
[c==EOT & !ctx.closeProb & ctx.errCnt >= errB]
/ctx.cans();
ctx.result="ExcessiveEOTs";
Complete Receiver Statechart
24 November 2021
For Evil Medium.
Copyright (c) 2021 Craig Scratchley
craig_scratchley AT alumni.sfu.ca
onEvent(KB_C)
/ctx.KbCan = true;
onEvent(SER)
/ctx.purge(); ctx.cans();
if (ctx.KbCan)
ctx.result = "KbCancelled (delayed)";
else
ctx.result = "ExcessiveErrors";
onEvent(TM)
/ctx.cans();
if (ctx.KbCan)
ctx.result="KbCancelled";
else
ctx.result="ExcessiveErrors";