definition

shared_ptr <T>

Sharing ownership of an object through a pointer

deallocated < last remainning shared_ptr is destroyed (delete) < operator= or reset()

make_shared <T>

creates a shared pointer that manages a new object

i.e: shared_ptr <int> foo = make_shared <int> (10);

tcdrain()

used to block untill all output written to the object is transmitted

shared_lock()

Balance readers and writters. No one will have higher priority here.

lock_guard()

you don't need to call lock() and unlock() manually

- it is unsafe for a member function to return a pointer or reference

deadlock:

a situation where two or more threads are blocked forever, waiting for each other. (Nothing gonna happen)

race condition:

occurs when multiple concurrently access a shared data

Rather than using an instance of `std::mutex` for the synchronization, you use an instance of `boost::shared_mutex`. For the update operations, `std::lock_guard<boost::shared_mutex>` and `std::unique_lock<boost::shared_mutex>` can be used for the locking, in place of the corresponding `std::mutex` specializations. These ensure exclusive access, just as with `std::mutex`. Those threads that don't need to update the data structure can instead use `boost::shared_lock<boost::shared_mutex>`

Collaboration Graph Notation (CGN)

Fraction.cpp

Oct 7th recording
copy mutex
delete.

```
/*
 * Fraction.cpp
 *
 * Multiple threads accessing a simple fraction object
 *
 * 2019 C. Scratchley created to work with C++11/14
 *
 * Copyright 2019, School of Engineering Science, SFU, Canada
*/

#include <iostream>
#include <signal.h>
#include <memory>
#include <sched.h>      // sched_yield()
#include <unistd.h>     /* required for sleep() */
#include <stdlib.h>     /* abs() ? */
#include <thread>
#include <mutex>        // std :: mutex.h // mutual exclusion

using namespace std;

mutex consoleMutex;
#define COUT (lock_guard<mutex>(consoleMutex), std::cout)

class fraction
{
private: mutex myMutex;  //good
public:
    unsigned num;
    unsigned denom;

    fraction(unsigned numerator, unsigned denominator) {
        num = numerator;
        denom = denominator;

    };
    void report(char id) {
        unsigned numC = num;
        unsigned denomC = denom;

        cout << id << ": Numerator is " << numC << ". Denominator is " << denomC
             << ".  Floating representation is " << 1.0 * numC / denomC
             << ".  Integer representation is " << numC / denomC
             << endl;
    };
    void invert() {
        int diff = num - denom; // subtract num count from denom count
        if (diff >= 0) {
            sched_yield();
            num -= diff;        // remove from num
            denom += diff;      // and... insert into denom
        }
```

help to fix the code

where should we put a mutex?

chapter 3

use lock mutex to have only 1 thread running.

either public or private.

//bad, 2 threads may be working on 2 different vars, and mutex will prevent it.

to protect two lines data below

lock_guard guard{myMutex}

unwanted

only

// context switch here b/w threads

make floating type

// lock_guard guard(myMutex)

// 2/3 → 3/2

void invert(){
    mutex myMutex; // bad because each caller gets own mutex
    myMutex.lock();
}

// lock a mutex

// may cause crash. the program is changing data, jump to other funcs, can cause wrong data.

fraction.cpp

```
        else {
            unsigned adiff = abs(diff);
            denom -= adiff;     // remove from denom
            num += adiff;       // and... insert into num
        }
    // unlock the mutex mymutex.unlock();
    };
};

//fraction fraction1(2, 3);
fraction fraction1(3, 6);

void threadFunc(char id) {
//    while (true) {
    for (int i=0; i < 3000; i++) {
        //
        fraction1.invert();
        // possibly some code here with random execution time.
        fraction1.report(id);
//      sched_yield();
    };
};

int main(int argc, char *argv[]) {
    /*  Define behaviour for divide-by-zero */
    //  SIGFPE is for integer divide-by-zero too.
    //  Single Unix Specification defines SIGFPE as "Erroneous arithmetic operation."
    //  https://stackoverflow.com/questions/6121623/catching-exception-divide-by-zero
    std::shared_ptr<void(int)> handler(
        signal(SIGFPE, [](int signum) {throw std::logic_error("FPE"); }),
        [](__sighandler_t f) { signal(SIGFPE, f); });

    fraction1.report('P');

    thread threadA(threadFunc, 'A');
    thread threadB(threadFunc, 'B');
    // sleep(1);
    threadB.join();
    threadA.join();

    cout << "Primary thread ending" << std::endl;
    return EXIT_SUCCESS;
}
```

mutex ask thread 'B' to wait until 'A' unlock mutex

// washroom anology.

mutex lock → door lock

sched_yield → disability / senior priority.

← global object

'A', 'B', 'C' ← thread

// absolute priority higher than 0

// give up processor and allow another thread to use the processor

mutex Mymutex; // bad, mutex as data member, make every class has mutex.

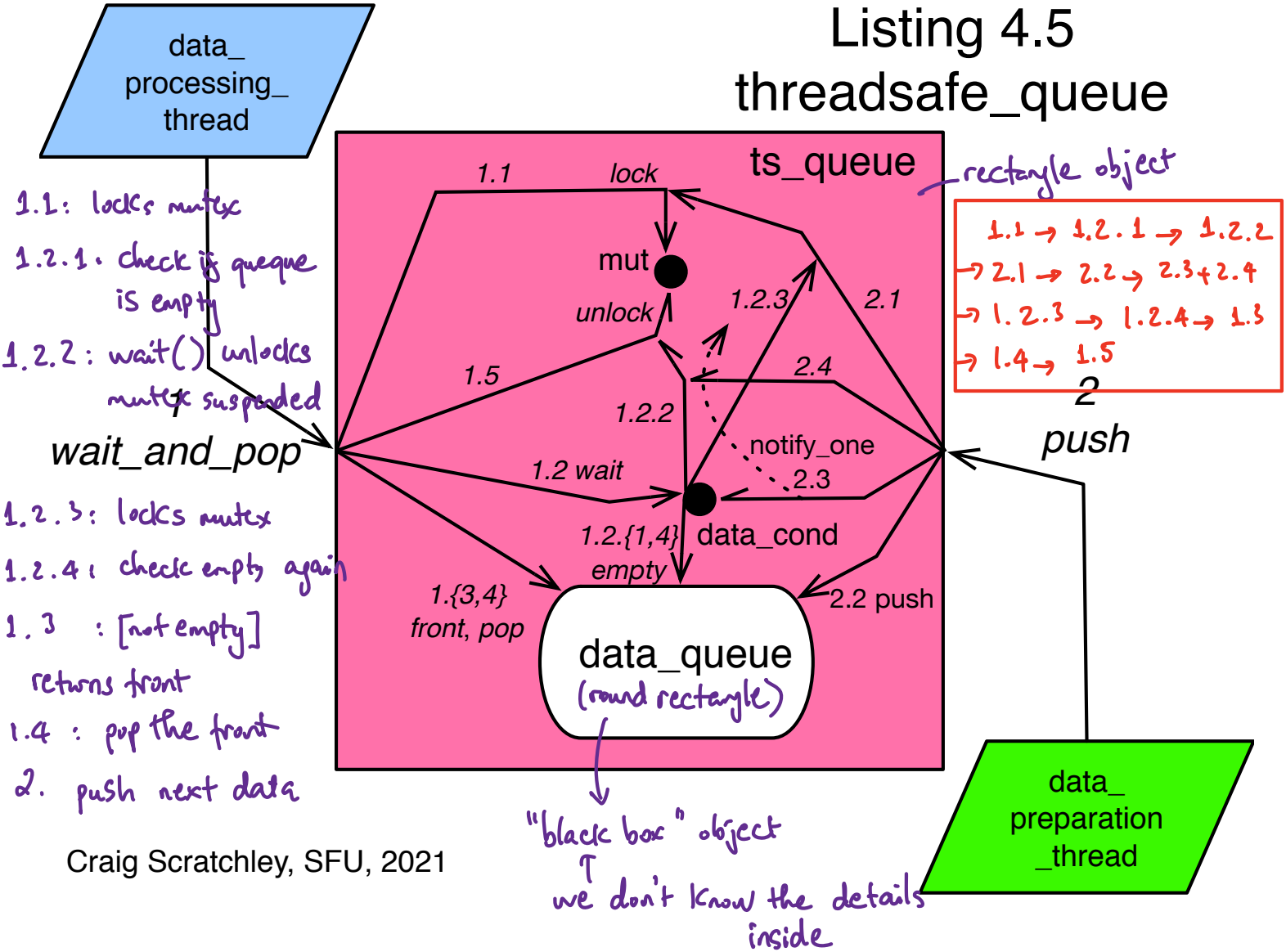
// 0

thread (file)

Cxx → C++

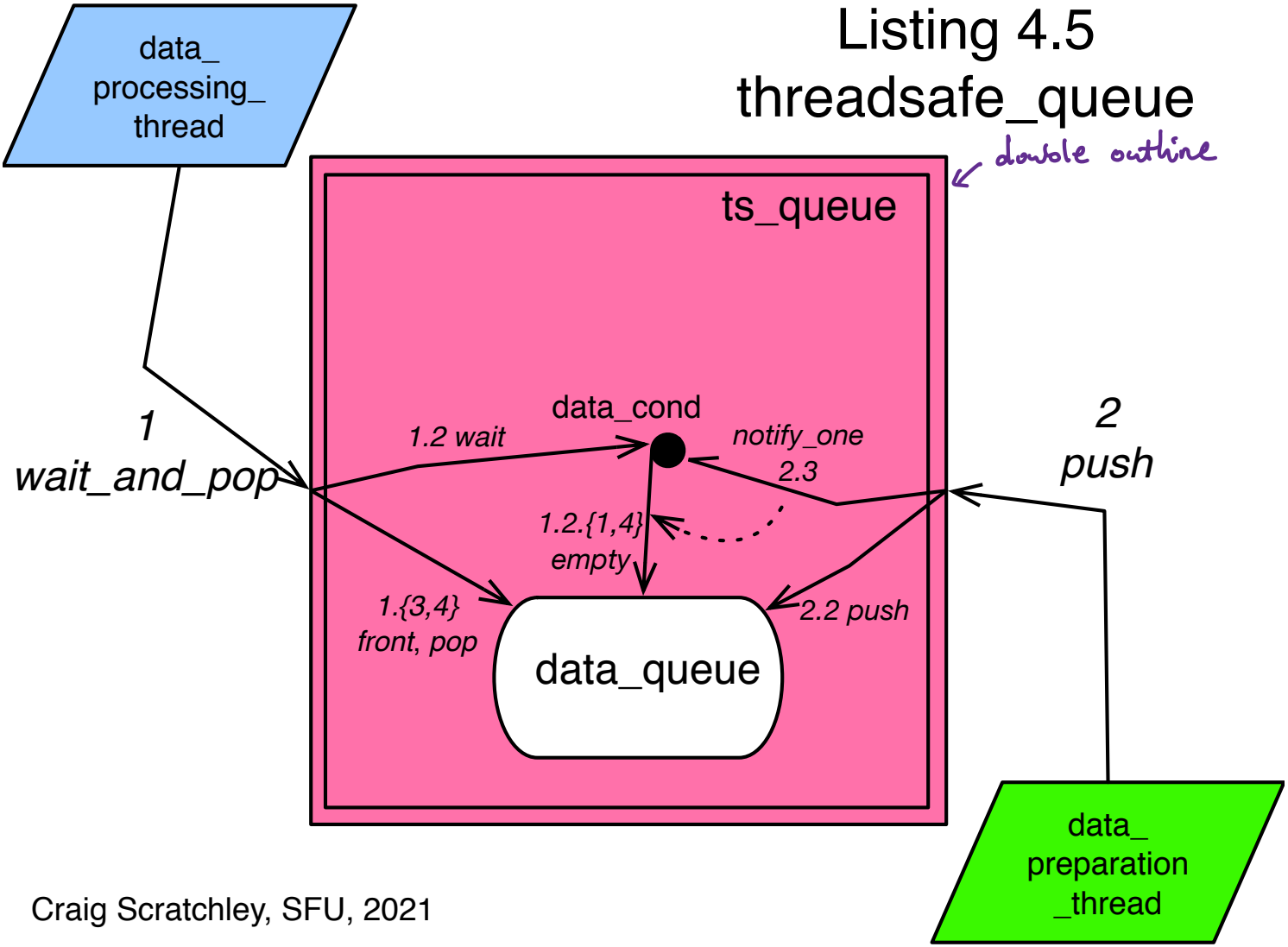
2threadTsQueueDot.pdf

Listing 4.5 threadsafe_queue



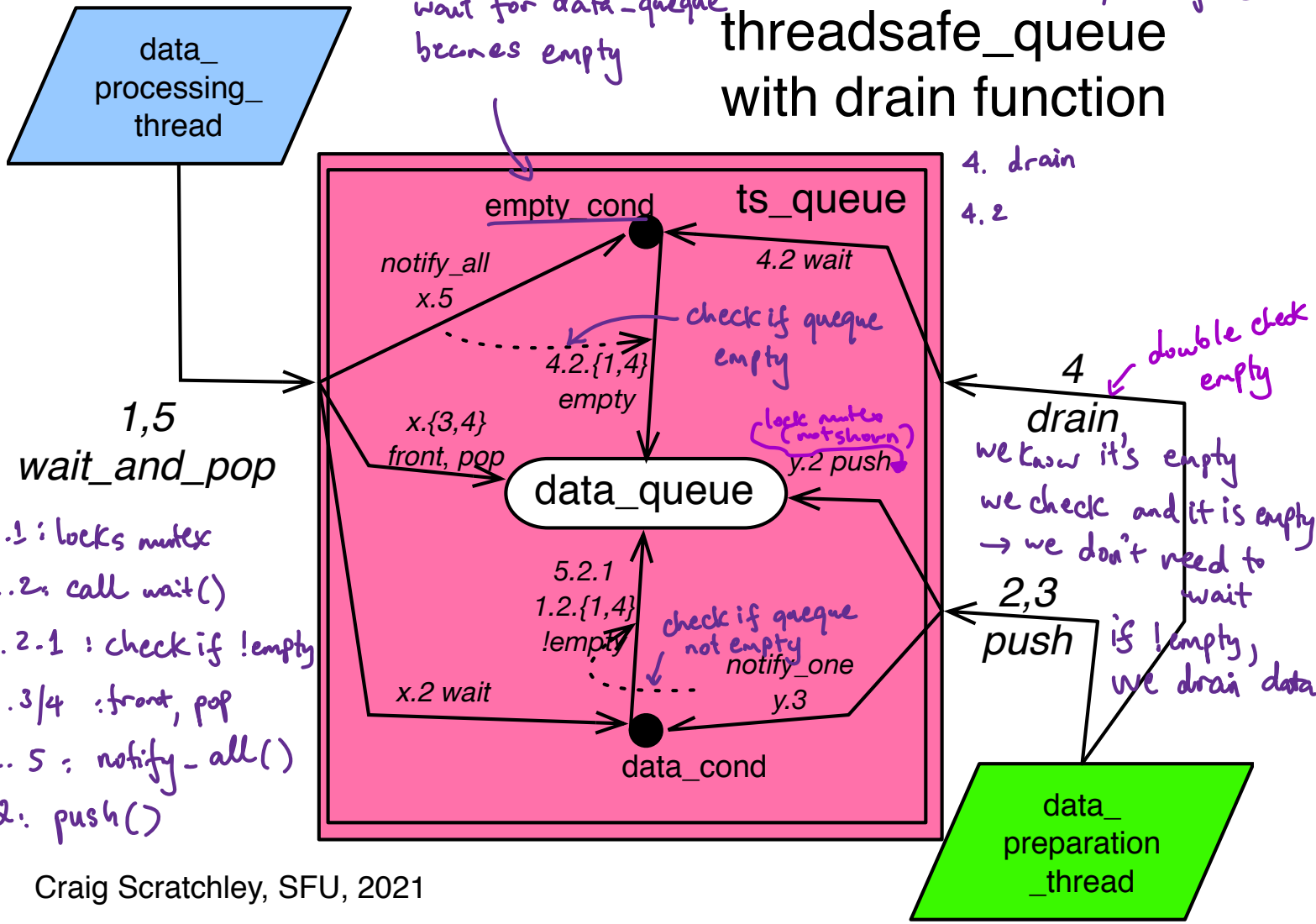
2threadTsQueueImpliedMutex.pdf

Listing 4.5 threadsafe_queue



threadsafe_queue w/ drain func

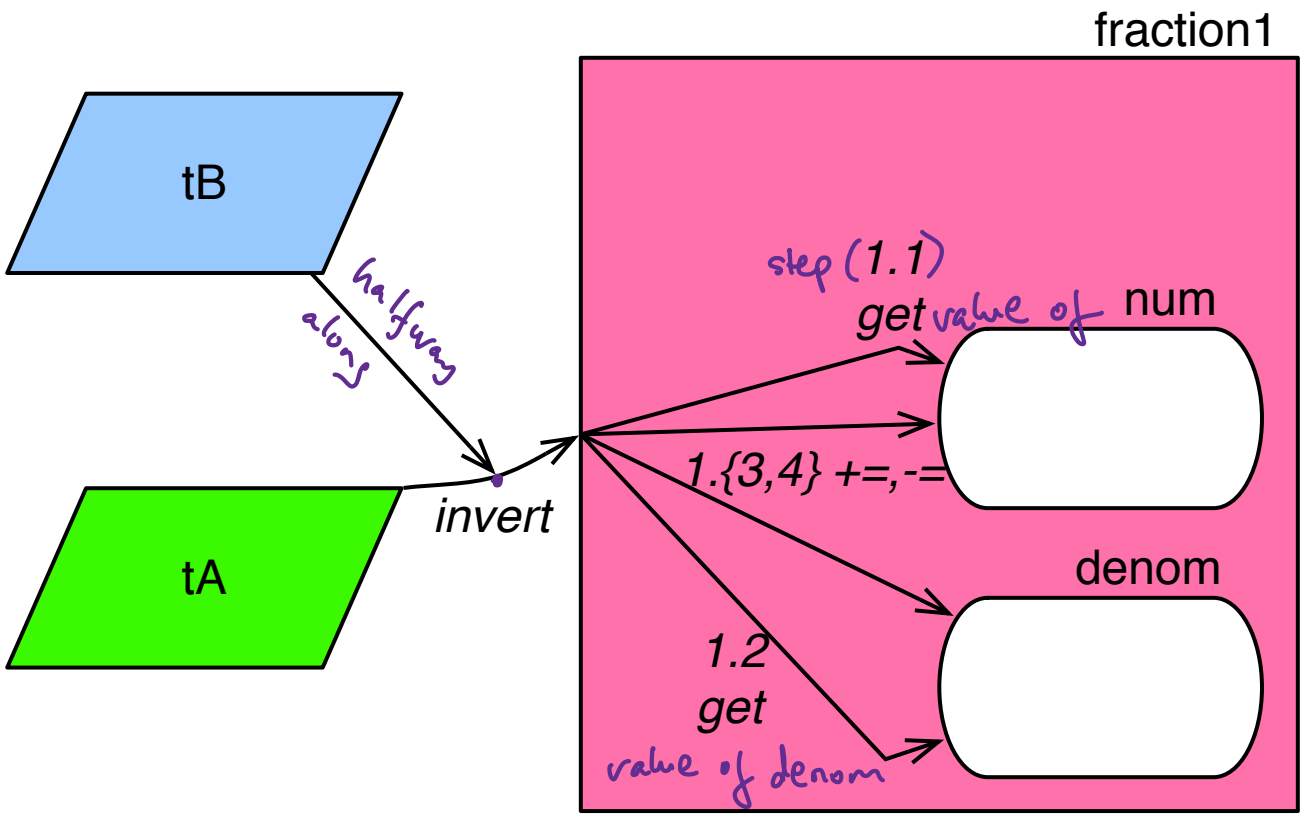
threadsafe_queue with drain function



1.1 : locks mutex
1.2 : call wait()
1.2.1 : check if !empty
1.3/4 : front, pop
1.5 : notify_all()
2. : push()

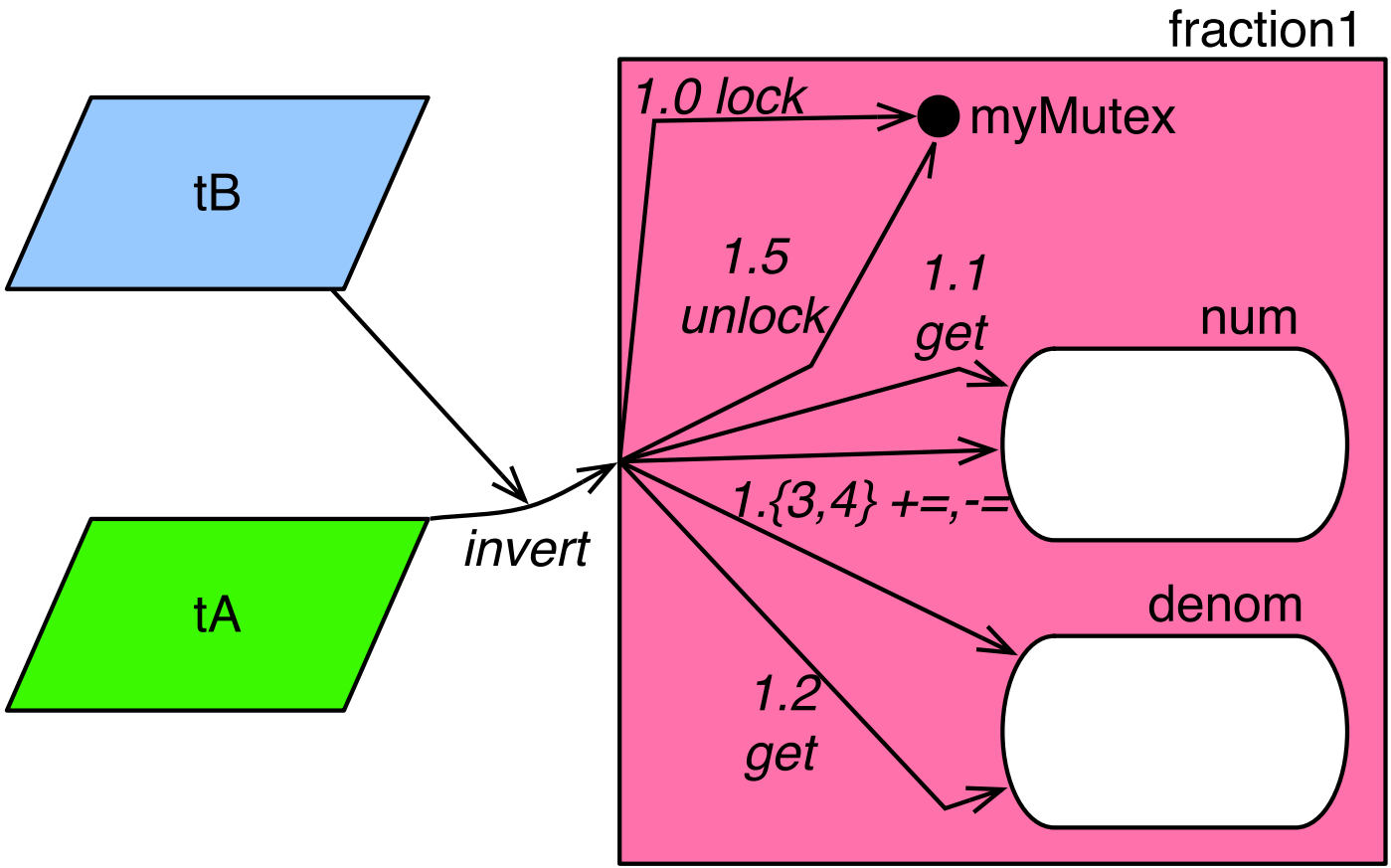
Craig Scratchley, SFU, 2020

two threads access a fraction object in an unsafe manner



Craig Scratchley, SFU, 2021

two threads access a fraction object protected by a mutex



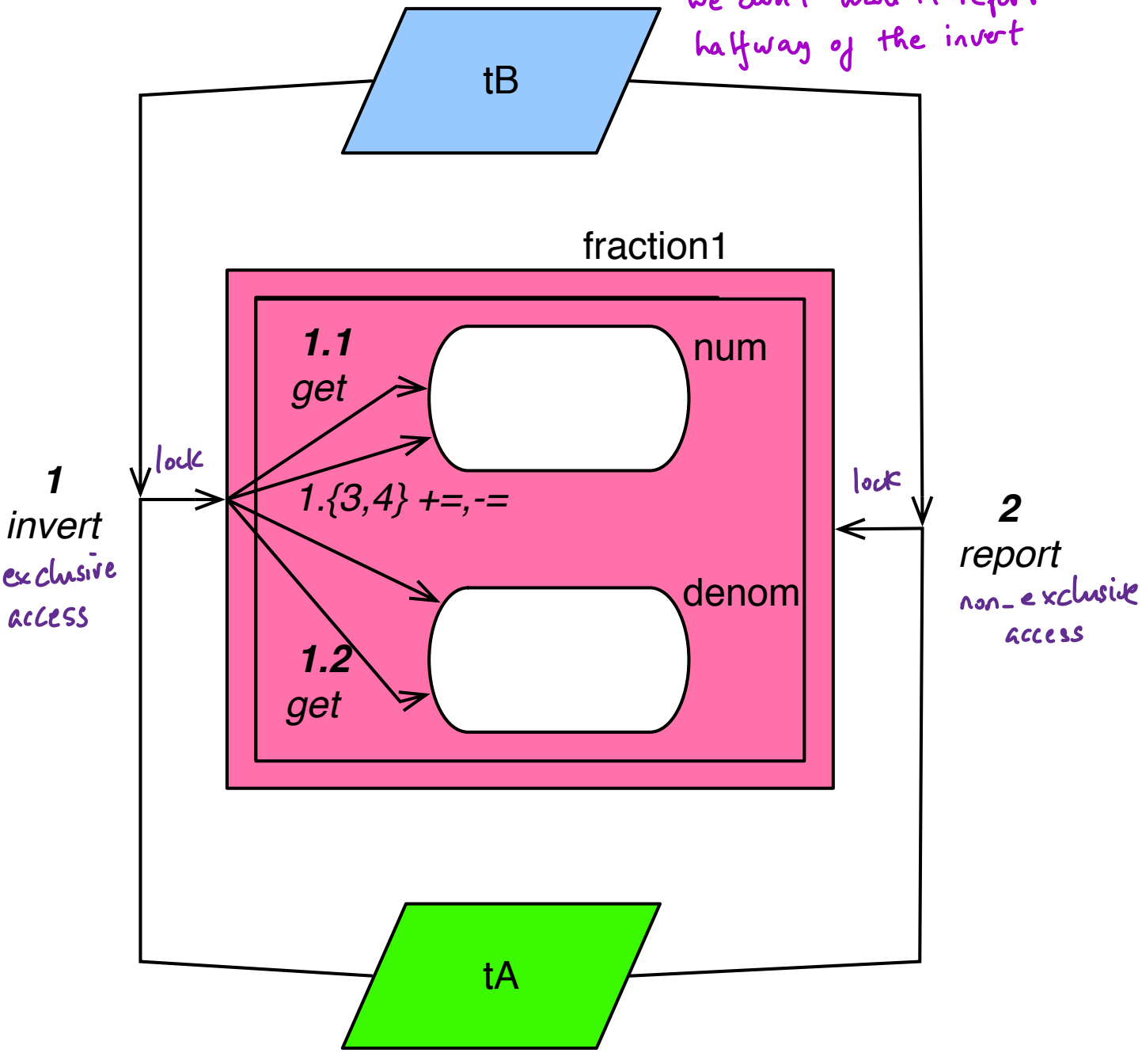
2threadfractionProtectedWReport

Craig Scratchley, SFU, 2020

two threads access a fraction object in a protected manner

more report() called than invert() → sharedMutex

we don't want it report halfway of the invert



2thread Protected

Craig Scratchley, SFU, 2020

two threads access a fraction object in a protected manner

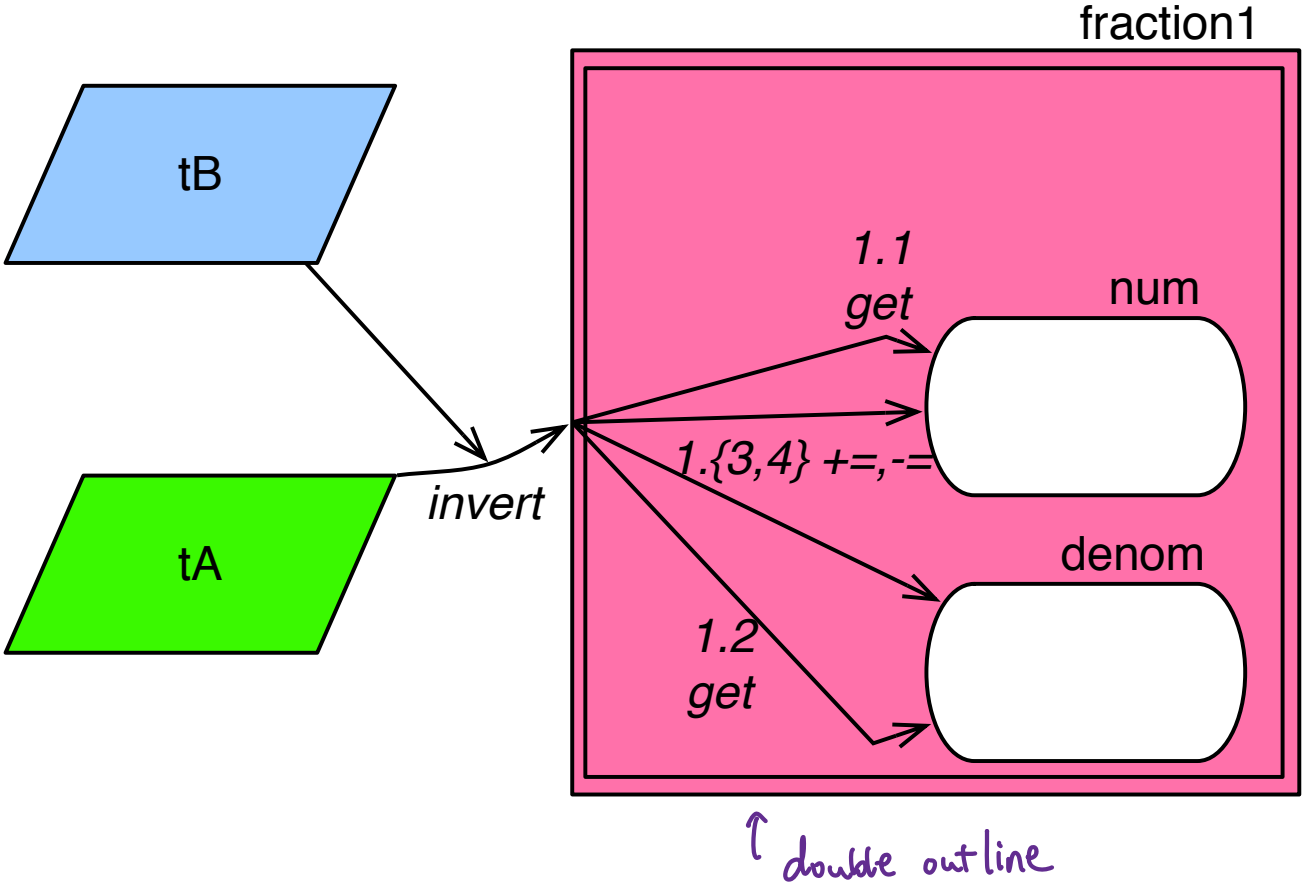
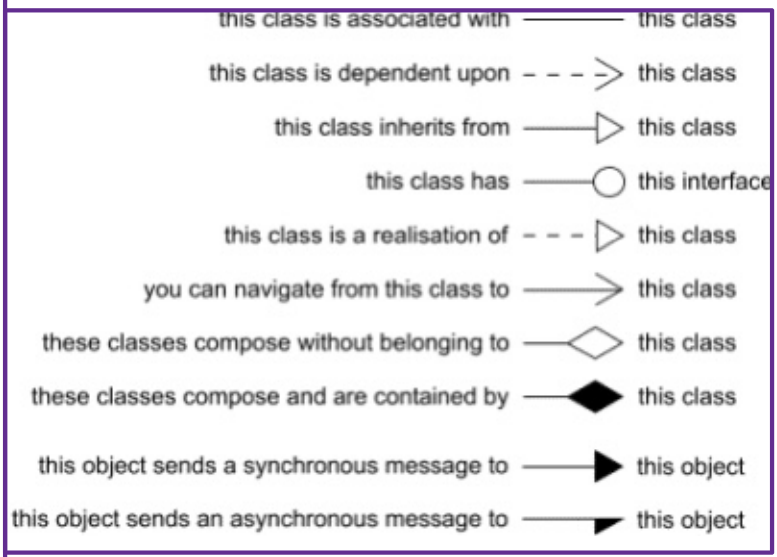
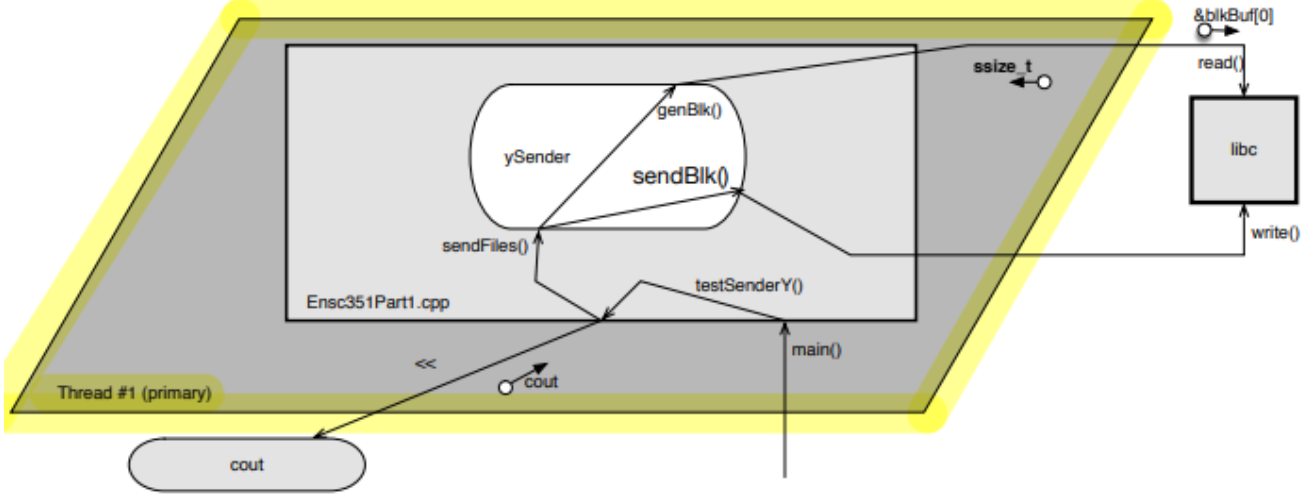


Figure 4. YMODEM Batch Transmission Session (2 files)

SENDER	RECEIVER
	"sb foo.c baz.c<CR>"
"sending in batch mode etc."	
	C (command:rb)
SOH 00 FF foo.c NUL[123] CRC CRC	
	ACK
	C
SOH 01 FE Data[128] CRC CRC	
	ACK
SOH 02 FC Data[128] CRC CRC	
	ACK
SOH 03 FB Data[100] CPMEOF[28] CRC CRC	
	ACK
EOT	
	NAK
EOT	
	ACK
	C
SOH 00 FF baz.c NUL[123] CRC CRC	
	ACK
	C
SOH 01 FB Data[100] CPMEOF[28] CRC CRC	
	ACK
EOT	
	NAK
EOT	
	ACK
	C
SOH 00 FF NUL[128] CRC CRC	
	ACK



A diagram for Part 1 of Multipart Project using Collaboration Graph Notation. The global object cout and primary thread have also been drawn.



CGN (part2)
CGN (part3)

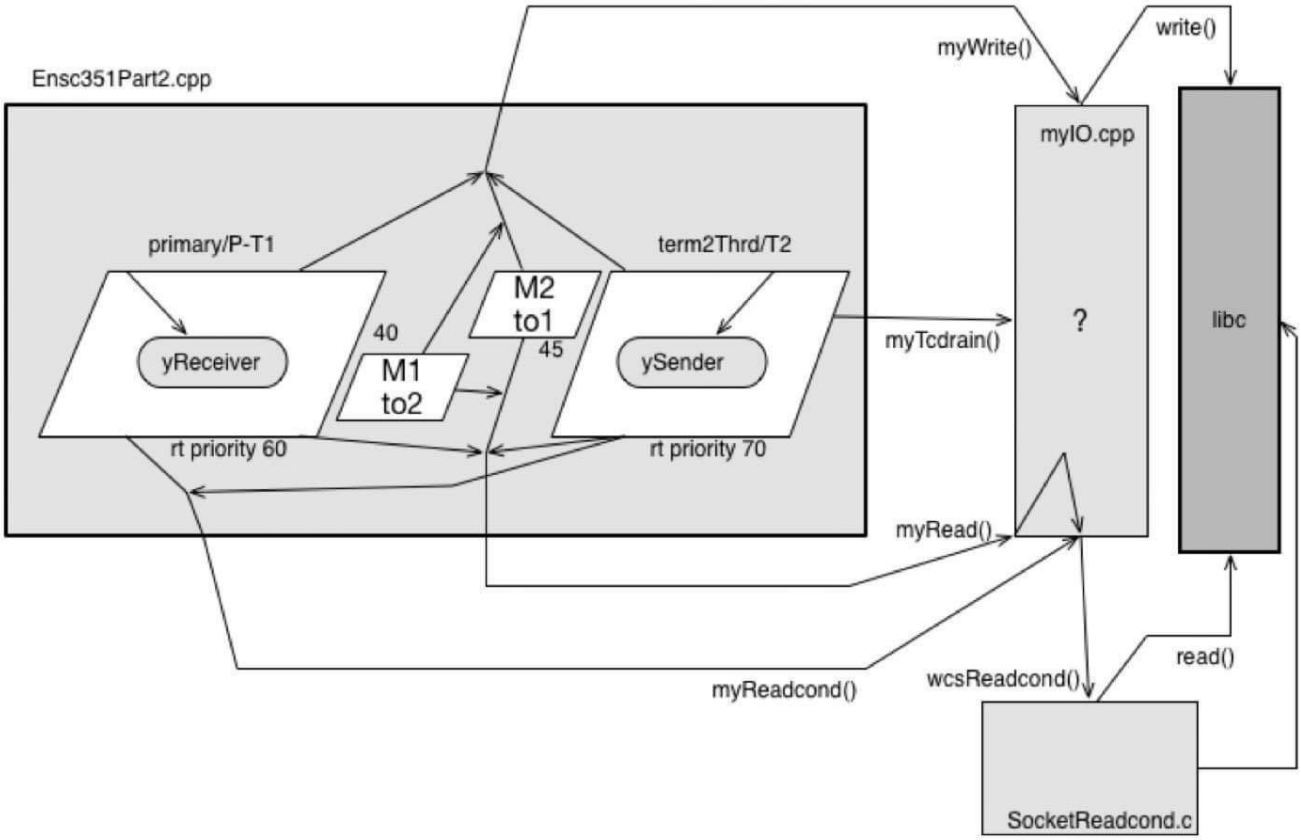
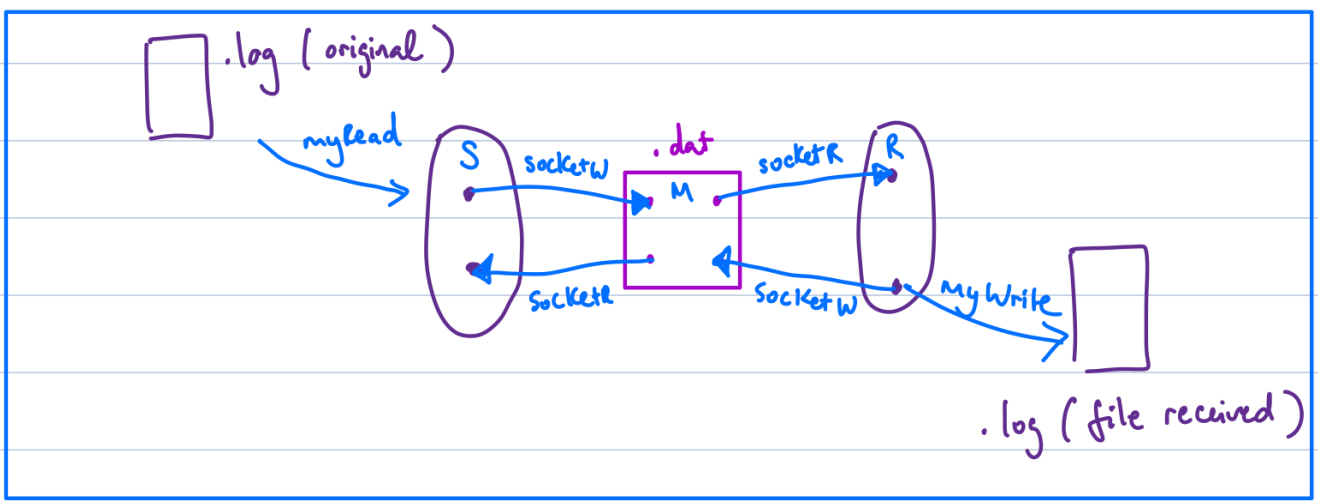
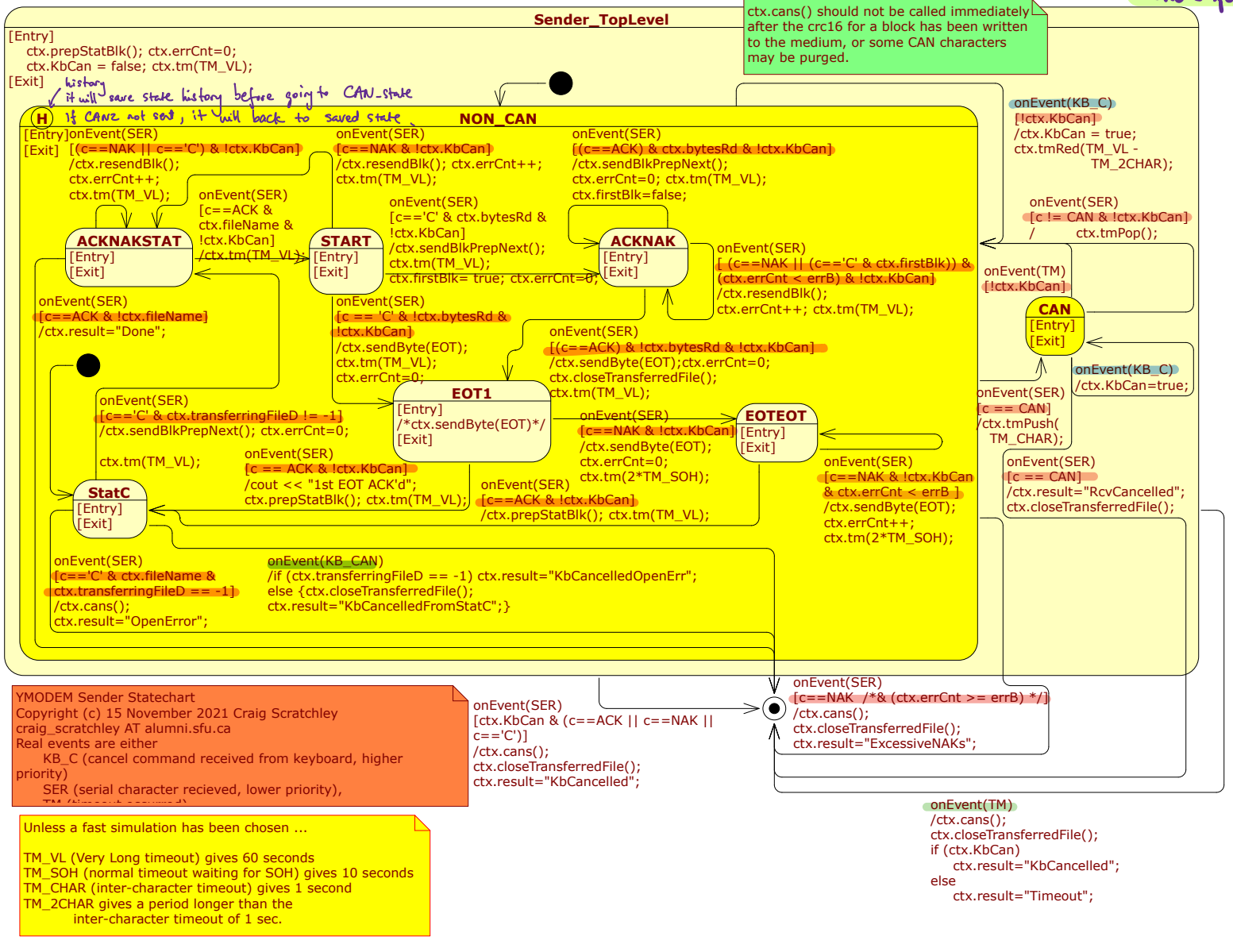
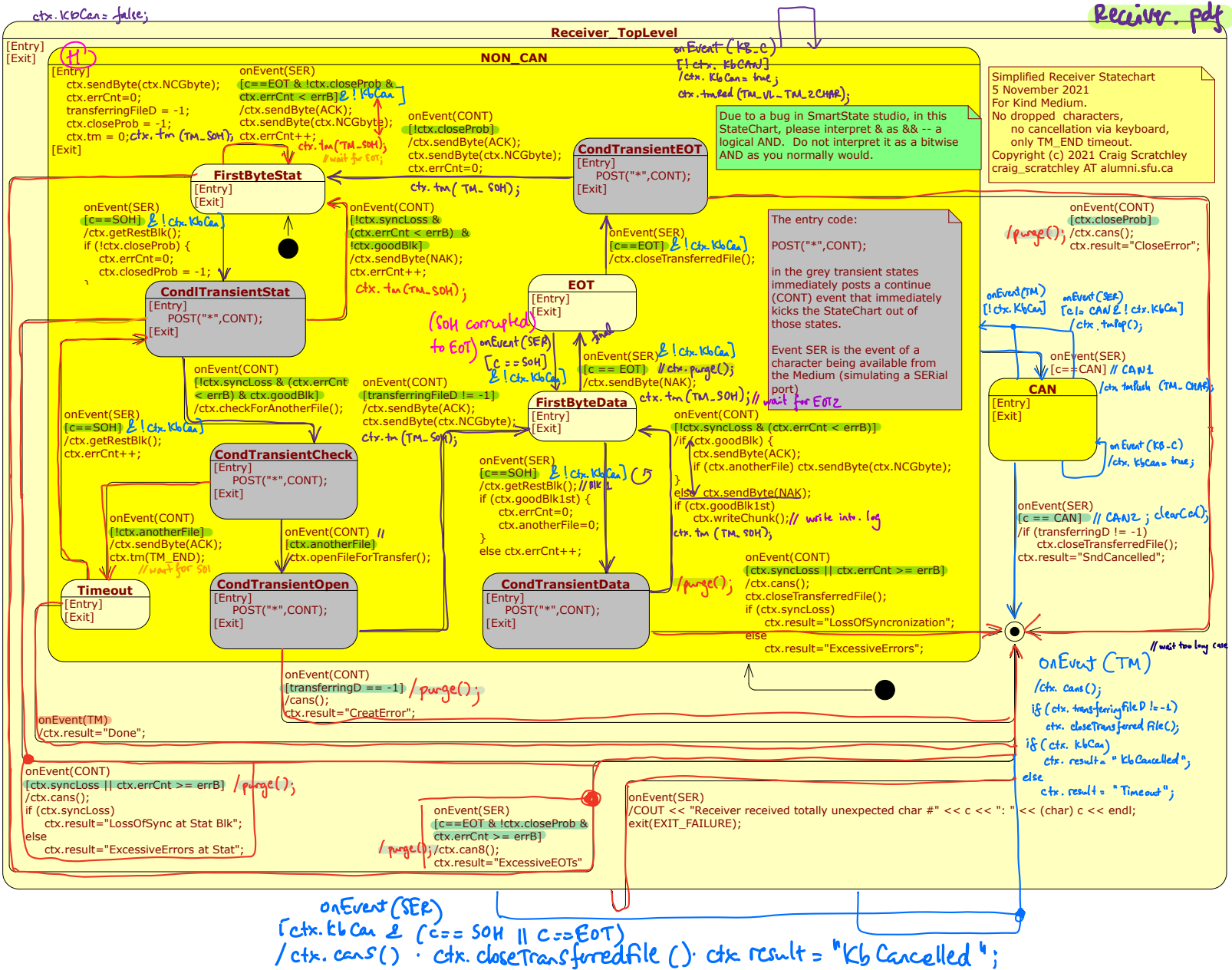


Figure 2: Collaboration Graph showing function calls for socketpair communication



pair	paired	A	B
-1		✓	
-2		✓	✓
	-2		✓


Sender.pdf
Sender_TopLevel
[Entry]
ctx.prepStatBlk(); ctx.errCnt=0;
ctx.KbCan = false; ctx.tm(TM_VL);
[Exit]
history
it will save state history before going to CAN_state
H
If CANz not sent, it will back to saved state
NON_CAN
ctx.cans() should not be called immediately after the crc16 for a block has been written to the medium, or some CAN characters may be purged.
onEvent(KB_C)
[!ctx.KbCan]
/ctx.KbCan = true;
ctx.tmRed(TM_VL - TM_2CHAR);
onEvent(SER)
[c != CAN & !ctx.KbCan]
/ ctx.tmPop();
onEvent(TM)
[!ctx.KbCan]
CAN
[Entry]
[Exit]
onEvent(KB_C)
/ctx.KbCan=true;
onEvent(SER)
[c == CAN]
/ctx.tmPush(
TM_CHAR);
onEvent(SER)
[c == CAN]
/ctx.result="RcvCancelled";
ctx.closeTransferredFile();
[Entry]onEvent(SER)
[Exit] [(c==NAK || c=='C') & !ctx.KbCan]
/ctx.resendBlk();
ctx.errCnt++;
ctx.tm(TM_VL);
ACKNAKSTAT
[Entry]
[Exit]
onEvent(SER)
[c==ACK &
ctx.fileName &
!ctx.KbCan]
/ctx.tm(TM_VL);
onEvent(SER)
[c==NAK & !ctx.KbCan]
/ctx.resendBlk(); ctx.errCnt++;
ctx.tm(TM_VL);
START
[Entry]
[Exit]
onEvent(SER)
[c=='C' & ctx.bytesRd &
!ctx.KbCan]
/ctx.sendBlkPrepNext();
ctx.tm(TM_VL);
ctx.firstBlk= true; ctx.errCnt=0;
onEvent(SER)
[(c==ACK) & ctx.bytesRd & !ctx.KbCan]
/ctx.sendBlkPrepNext();
ctx.errCnt=0; ctx.tm(TM_VL);
ctx.firstBlk=false;
ACKNAK
[Entry]
[Exit]
onEvent(SER)
[(c==NAK || (c=='C' & ctx.firstBlk)) &
(ctx.errCnt < errB) & !ctx.KbCan]
/ctx.resendBlk();
ctx.errCnt++; ctx.tm(TM_VL);
onEvent(SER)
[c==ACK & !ctx.fileName]
/ctx.result="Done";
onEvent(SER)
[c == 'C' & !ctx.bytesRd &
!ctx.KbCan]
/ctx.sendByte(EOT);
ctx.tm(TM_VL);
ctx.errCnt=0;
EOT1
[Entry]
/*ctx.sendByte(EOT)*/
[Exit]
onEvent(SER)
[(c==ACK) & !ctx.bytesRd & !ctx.KbCan]
/ctx.sendByte(EOT);ctx.errCnt=0;
ctx.closeTransferredFile();
ctx.tm(TM_VL);
EOTEOT
[Entry]
[Exit]
onEvent(SER)
[c==NAK & !ctx.KbCan]
/ctx.sendByte(EOT);
ctx.errCnt=0;
ctx.tm(2*TM_SOH);
onEvent(SER)
[c==NAK & !ctx.KbCan
& ctx.errCnt < errB]
/ctx.sendByte(EOT);
ctx.errCnt++;
ctx.tm(2*TM_SOH);
onEvent(SER)
[c=='C' & ctx.transferringFileD != -1]
/ctx.sendBlkPrepNext(); ctx.errCnt=0;
ctx.tm(TM_VL);
onEvent(SER)
[c == ACK & !ctx.KbCan]
/cout << "1st EOT ACK'd";
ctx.prepStatBlk(); ctx.tm(TM_VL);
onEvent(SER)
[c==ACK & !ctx.KbCan]
/ctx.prepStatBlk(); ctx.tm(TM_VL);
StatC
[Entry]
[Exit]
onEvent(SER)
[c=='C' & ctx.fileName &
ctx.transferringFileD == -1]
/ctx.cans();
ctx.result="OpenError";
onEvent(KB_CAN)
/if (ctx.transferringFileD == -1) ctx.result="KbCancelledOpenErr";
else {ctx.closeTransferredFile();
ctx.result="KbCancelledFromStatC";}
YMODEM Sender Statechart
Copyright (c) 15 November 2021 Craig Scratchley
craig_scratchley AT alumni.sfu.ca
Real events are either
KB_C (cancel command received from keyboard, higher priority)
SER (serial character recieved, lower priority),
onEvent(SER)
[ctx.KbCan & (c==ACK || c==NAK ||
c=='C')]
/ctx.cans();
ctx.closeTransferredFile();
ctx.result="KbCancelled";
onEvent(SER)
[c==NAK /*& (ctx.errCnt >= errB) */]
/ctx.cans();
ctx.closeTransferredFile();
ctx.result="ExcessiveNAKs";
onEvent(TM)
/ctx.cans();
ctx.closeTransferredFile();
if (ctx.KbCan)
ctx.result="KbCancelled";
else
ctx.result="Timeout";
Unless a fast simulation has been chosen ...
TM_VL (Very Long timeout) gives 60 seconds
TM_SOH (normal timeout waiting for SOH) gives 10 seconds
TM_CHAR (inter-character timeout) gives 1 second
TM_2CHAR gives a period longer than the
inter-character timeout of 1 sec.


Receiver_TopLevel
NON_CAN
FirstByteStat
CondTransientEOT
CondTransientStat
EOT
FirstByteData
CondTransientCheck
CondTransientOpen
CondTransientData
Timeout
CAN
Simplified Receiver Statechart
5 November 2021
For Kind Medium.
No dropped characters,
no cancellation via keyboard,
only TM_END timeout.
Copyright (c) 2021 Craig Scratchley
craig_scratchley AT alumni.sfu.ca
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.
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)
Receiver.pdf

Receiver Y member fun() definition

For reference, below is a description of the public member functions and variables in class *ReceiverY*, the receiver context (an object of which is referenced as *ctx* in the receiver state chart actions and guard conditions). If you need to add additional functions and/or variables to this class, the *SenderY* class, or the *PeerY* parent class, please check with Craig first.

Function or Variable	**Description**
void cans()	Send CAN_LEN copies of CAN characters in a row to the peer, to inform it of the cancelling of a session.
void getRestBlk()	Call only after an SOH character has been received and posted to the receiver state chart. The function tries to read at least 132 more characters to form a complete block. The function will set or reset a Boolean variable, *goodBlk*. This variable will be made false if either • 132 bytes have not yet been received and another byte does not arrive within 1 second of the last byte (or within 1 second of the function being called). • if a good copy of the block has not already been received, 132 bytes (or more) are received and the block completed using the first 132 received bytes has something wrong with it, like the crc16 being incorrect. When it is released, see getRestBlk() in Part 2 solution for details. The function will also set or reset another Boolean member variable, *syncLoss*. *syncLoss* will only be set to true when *goodBlk* is set to true AND there is a fatal loss of syncronization as described in the YMODEM specification. The first time each block is received and is good, *goodBlk1st* will be set to true. This is an indication of when the data in a block should be written to disk. If *goodBlk1st* is false, or in any case if more than 132 bytes were received, then the *purge*() function (see below) will be called before returning from *getRestBlk*().
void writeChunk()	Write the data in a received block to disk.
void clearCan()	Read and discard up to (CAN_LEN – 2) contiguous CAN characters. Read characters one-by-one in a loop until either nothing is received over a 2-second period or a character other than CAN is received. If received, send a non-CAN character to the console.
void purge()	The *purge*() subroutine will read and discard characters until nothing is received over a 1-second period.
int errCnt /*in PeerY*/	A variable that counts a sequence of 'C' or NAK characters sent, 10-second timeouts, or ACK characters sent due to repeated blocks. The initial 'C' does not add to the count. The reception of a good block for the first time resets the count (see *goodBlk1st* below).
int closeProb	0 indicates that there was no problem closing the file just transferred. A positive value is the errno encountered when trying to close the file. -1 indicates that the program has not yet tried to close the current file (or there is no current file).
bool goodBlk	A Boolean variable that indicates whether the last block received was good (or deemed good) or whether it had problems (and was not deemed to be good).

PeerY member func() definition

bool goodBlk1st	A Boolean variable that indicates that a good copy of a block being sent has been received for the first time. It is an indication that the data in the block can be written to disk.
bool syncLoss	A Boolean variable that indicates whether or not a fatal loss of synchronization has been detected.

The *ReceiverY* class, like the *SenderY* class described below, is a class derived from *PeerY*. Here is a description of the public member functions and variables you can use in *PeerY*:

Function or Variable	**Description**
void sendByte(uint8_t)	Send a byte to the remote peer across the medium
void tm(int tmSeconds)	set a timeout time at an absolute time *tmSeconds* into the future. That is, determine an absolute time to be used for the next YMODEM timeout by adding *tmSeconds* to the current time.
void tmPush(int tmSeconds)	Store the current absolute timeout, and create a temporary absolute timeout *tmSeconds* into the future.
void tmPop()	Discard the temporary absolute timeout and revert to the stored absolute timeout
void tmRed(int secsToReduce)	Make the absolute timeout time earlier by *secsToReduce* seconds.
std::string result	Points to a string giving the "result" when the session ends.
int transferringFileD	Descriptor for file being read from or written to.
int errCnt	See description in derived classes for interpretation for each class.

Note that the *errCnt* variable described in the derived classes actually lives here in *PeerY*.

As written above, if you need to add additional functions and/or variables to this base class, the *SenderY* class mentioned on the next page, or the *ReceiverY* class, please check with Craig first. A solution, as will be provided by Craig, uses only the provided functions and variables.

For your information, here is a description of the public member functions and variables used in *SenderY*, the sender context class (an instance of which is referenced as *ctx* in the sender statechart actions and guard conditions):

Member Function or Variable	**Description**
void can8()	Send to the peer CAN_LEN copies of CAN characters in pairs, each pair separated by slightly over 1 second, to inform it of the canceling of a file transfer.
void sendBlkPrepNext()	Sends for the first time the block recently prepared (less the last byte), then updates the variable *bytesRd* and tries to prepare the next block. *bytesRd* is set to 0 and a block is not actually prepared if the input file is empty or the end of the input file was reached when the block recently prepared was filled in. Finally, after the rest of the block has been sent, dumps any received glitches and sends the last byte of block being sent (the last byte of crc16).
void resendBlk()	Resends the block that had been sent most recently or an empty zero-numbered "stat" block (less its last byte in both cases), and then dumps any received glitches and sends the last byte.
clearCan()	Read and discard contiguous CAN characters. Read up to (CAN_LEN – 2) characters one-by-one in a loop until either nothing is received over a 1-second period or a character other than CAN is received. If received, send a non-CAN character to the console.
int errCnt /*in PeerY*/	Counts the number of times that a block or EOT is resent.
ssize_t bytesRd	The number of bytes last read from the input file.
bool firstBlk	True if first block of file content data has not yet been acknowledged with an ACK.

Since we don't ask you, at this point, to test your Statechart by generating code from it and running the code, you can get full marks for your work as long as you have made a sincere attempt to handle in a reasonable way all the issues identified at the top of this document.

Since this part of the project will not require you to test your work, it should take less time than recent parts of the project.

```
#endif

using namespace std;

//Unnamed namespace
namespace{

    class socketInfoClass;

    typedef shared_ptr<socketInfoClass> socketInfoClassSp;
    map<int, socketInfoClassSp> desInfoMap {
        {-2, make_shared<socketInfoClass>(-1)}, // marker for a
descriptor whose pair is closed
        {0, nullptr}, // init for stdin,
        {1, nullptr}, //          stdout,
        {2, nullptr}  //          stderr
    };

    //  A shared mutex used to protect desInfoMap so only a
single thread can modify the map at a time.
    //  This also means that only one call to functions like
mySocketpair() or myClose() can make progress at a time.
    //  This mutex is also used to prevent a paired socket from
being closed at the beginning of a myWrite or myTcdrain function.
    shared_mutex mapMutex;

    class socketInfoClass {
        unsigned totalWritten = 0;// byte written to a buffer
        unsigned maxTotalCanRead = 0;// max total bytes can be read
        condition_variable cvDrain;    by a thread in a socket
#ifndef WCSREADCOND
        condition_variable cvRead;
#endif
#ifdef CIRCBUF
        CircBuf<char> circBuffer;
        bool connectionReset = false;
#endif
        mutex socketInfoMutex;
    public:
        int pair;   // Cannot be private because myWrite and
myTcdrain using it.
                    // -1 when descriptor closed, -2 when paired
descriptor is closed
```

avoid risk when combine code w/ partner whose also have same name class. So only our class can access this unnamed namespace.

myIO.cpp

// funcs belong to unnamed namespace can only be used in this file but not others. Keep it private!!

this map is not thread safe → need to lock mutex

1' used to blk reading activity

2'

3'

myIO.cpp

```
int draining()
{ // operating on object for paired descriptor of original des
    unique_lock<mutex> socketLk(socketInfoMutex);

    //  paired descriptor could have been closed before we
constructed socketLk
    //  once the reader decides the drainer should wakeup, it
should wakeup
    if (pair >= 0 && totalWritten > maxTotalCanRead)
        cvDrain.wait(socketLk); // what about spurious wakeup?
//      cvDrain.wait(socketLk, [this]{return pair < 0 ||
totalWritten <= maxTotalCanRead;});
                    these 2 condition code can replace w/ comment line
    if (pair == -2) {
        errno = EBADF; // check errno
        return -1;
    }
    return 0;
}
```

i.e.: If one lock in thread 2, one in thread 3 — if you can't lock one, the other also can't
You can't lock each other, but need another thread 4 to lock them

```
        unique_lock<mutex> socketLk(socketInfoMutex,
defer_lock);
        unique_lock<mutex> condPairlk(des_pair->socketInfoMutex, defer_lock);
        lock(condPairlk, socketLk); // lock both mutex at the same time,
        pair = -1; // this is first socket in the pair to be
closed
int mySocketpair(int domain, int type, int protocol, int des[2])
{
    lock_guard<shared_mutex> desInfoLk(mapMutex);
    int returnVal = socketpair(domain, type, protocol, des);
    if(returnVal != -1) {
```

move inside if()

what happen if we switch the order of these 2 lines?
socketpair is thread safe (which not modify our Map, so it can be put above mutex → holds mutex for shorter time → more efficient so other thread donnot wait too long

```
        desInfoMap.emplace(des[0],
make_shared<socketInfoClass>(des[1]));
        desInfoMap.emplace(des[1],
make_shared<socketInfoClass>(des[0]));
    }
    return returnVal;
}
```

// check if put mutex at the beginning gives any error?

Order of execution of code in Ensc351Part3-test.cpp and some output and details of the solution

threadT41 (priority 50)	threadT32 (priority 70 to 40 to 80 to 40)	threadT42 (priority 60)	daSktPr[0] - 3			daSktPr[1] - 4		
			tW/mTCR	pair	cvDrn	tW/mTCR	pair	cvDrn/cvR
			-	-	-	-	-	-

	mySocketpair(AF_LOCAL, SOCK_STREAM, 0, daSktPr);
	mySocketpair(AF_LOCAL, SOCK_STREAM, 0, daSktPr1);
	mySocketpair(AF_LOCAL, SOCK_STREAM, 0, daSktPr2);
			0/0	4		0/0	3	/

	PE_NOT(myWrite(daSktPr[0], "abcd", 4), 4);
	posixThread threadT42(60, threadT42Func);
	PE(myTcdrain(daSktPr[0])); // blocked
			0/0
			0/0
				4
				4
						4/0
						4/0
							3
							3
								/
								T32/

		PE_NOT(myWrite(daSktPr[1], "ijkl", 5), 5);
		posixThread threadT41(50, threadT41Func);
		/* X */ myTcdrain(daSktPr[1]);//blocked
			5/0
			5/0
				4
				4
					
					T42
						4/0
						4/0
							3
							3
								T32/
								T32/

/* A */ myReadcond(daSktPr[1], Ba, 20, 12, 0, 0); // blocked			5	4	T42	4/20	3	/T41
	setSchedPrio(40); PE_NOT(myWrite(daSktPr[0], "123456789", 10), 10);		5/0	4	T42	14/20	3	/

/* finished: statement A: result was 14 Ba: abcd123456789 */
myReadcond(daSktPr[1], Ba, 20, 0, 0, 0); // returned 0;
myWrite(daSktPr[1], "Will not be read", 17);
/* B */ myReadcond(daSktPr[1], Ba, 20, 12, 0, 0); // blocked
			5/0
			5/0
			22/0
			22/0
				4
				4
				4
				4
					T42
					T42
					T42
					T42
						0/0
						0/0
						0/0
						0/20
							3
							3
							3
							3
								/
								/
								/
								/T41

	setSchedPrio(80);
	PE_NOT (myWrite(daSktPr[0], "xyz", 4), 4);
	PE(myTcdrain(daSktPr[0]));
	PE(myClose(daSktPr[0])); // returned 0
	setSchedPrio(40);
			22/0
			
			22/0
				4
				
				-1
					T42
					
						4/20
						
						4/20
							3
							
							-2
								/T41
								
								/

		/* finished: statement X: result was 0*/
		myTcdrain(daSktPr[1]);
		threadT41.join();
			22/0	-1		4/20	-2	/

/* finished: statement B: result was 4 Ba: xyz */
myReadcond(daSktPr[1], Ba, 20, 1, 0, 0)); // errno 104 "Reset"
myWrite(daSktPr1[1], "Will not be read", 17)
/* C */ myReadcond(daSktPr1[1], Ba, 20, 1, 0, 0); // bl'ked								

	PE(myClose(daSktPr1[0]));							

/* finished: C: res was -1 errno 104: Connection reset by peer */
myReadcond(daSktPr1[1], Ba, 20, 1, 0, 0)); // will return 0
myWrite(daSktPr[1], "Added", 6); // ret -1 errno 32: Broken pipe
/* D */ myRead(daSktPr2[1], Ba, 20); // blocked								

	PE_NOT(myWrite(daSktPr2[0], "mno", 4), 4);							
/* finished: statement D: result was 4 Ba: mno */ /* E */ myRead(daSktPr2[1], Ba, 20);								
	PE(myClose(daSktPr2[0]));							

/* finished: statement E: result was 0*/
myClose(daSktPr2[1]); // returned 0
myClose(daSktPr1[1]); // returned 0
myClose(daSktPr[1]); // returned 0
myClose(daSktPr[1]); // ret -1 errno 9[lt]: Bad file descriptor
myRead(daSktPr[1], Ba, 20); // ret -1 errno 9
// ...
/* end of threadT41 */								

		/* end of threadT42 */						
	threadT42.join();							

tw: total written
mTCR: maximum total can be read

threadtable part3 sol

which thread is blocked on cvDrain

socket 0

socket 1

read 4 char from thread T32 → return to Tcdrain

write 4 chars to socket 0

create thread w/ priority 60

unblock

take 8 more bytes (read 4 bytes)

not have myreadcond

write 10 more bytes (include NULL)

close socket 0, unblock 2 threads in socket 1

no more data coming to socket 0

already had myReadCond → not blkd

max

min

unblock

don't bother me till you get 12 bytes

we closed, so only 4 byte read.

error due to mywrite "will not be read" above

no myread to read

Ensc351Part3test.cpp
& output

```
/* Ensc351Part3-test.cpp -- October 21 -- Copyright 2021 Craig Scratchley */

/* This program can be used to test your changes to myIO.cpp
 *
 * Put this project in the same workspace as your Ensc351 library project,
 * and build it.
 *
 * With 3 created threads for a total of 4 threads, the output that I get
with my solution
 * is in the file "output-fromSolution".
 *
 */

#include <sys/socket.h>
#include <stdlib.h>             // for exit()
#include <sched.h>
#include "posixThread.hpp"
#include "VNPE.h"
#include "myIO.h"

//#define OUTPUT_TO_FILE

#ifdef OUTPUT_TO_FILE
    #include <fstream>
    std::ofstream COUT("output");
    // When RUNning the Release configuration, output will appear in the
Eclipse project.
    // When using the "viaConnection" launcher, output will appear at path
/home/osboxes/output
#else
    #include "AtomicCOUT.h"
#endif


#define REPORT0(S) COUT << threadName << ": " << #S << "; statement will now
be started\n";  \
    S; \
    COUT << threadName << ": " << #S << "; statement has now finished\n";
#define REPORT1(FC) {COUT << threadName << ": " << #FC << " will now be
called\n";  \
    int RV = FC; \
    COUT << threadName << ": " << #FC << " result was " << RV; \
    if (RV == -1) COUT << " errno " << errno << ": " << strerror(errno); \
    COUT << "\n"; }
#define REPORT2(FC) {COUT << threadName << ": " << #FC << " will now be
called\n";  \
    int RV = FC; \
    COUT << threadName << ": " << #FC << " result was " << RV; \
    if (RV == -1) COUT << " errno " << errno << ": " << strerror(errno); \
    else if (RV > 0) COUT << " Ba: " << Ba; \
    COUT << "\n";}

using namespace std;
using namespace pthreadSupport;

static int daSktPr[2];    // Descriptor Array for Socket Pair
static int daSktPr2[2];    // Descriptor Array for 2nd Socket Pair
```

ENSC351Part3test.cpp & output

```
static int daSktPr1[2];    // Descriptor Array for another Socket Pair
cpu_set_t cpu_set;
int myCpu=0;

void threadT41Func(void) // starts at priority 50
{
    char    Ba[20];
    const char* threadName = "T41";
    PE_0(pthread_setname_np(pthread_self(), threadName));
    //
    // Blank lines below indicate that the statement above the blank line
    // will finish after one or more other threads in the process have made
progress.
    REPORT2(myReadcond(daSktPr[1], Ba, 20, 12, 0, 0));   // will block until
myWrite of 10 characters

    REPORT2(myReadcond(daSktPr[1], Ba, 20, 0, 0, 0));
    REPORT1(myWrite(daSktPr[1], "Will not be read", 17));
    REPORT2(myReadcond(daSktPr[1], Ba, 20, 12, 0, 0));   // will block until
myClose(daSktPr[0])

    REPORT2(myReadcond(daSktPr[1], Ba, 20, 1, 0, 0));   // will return -1
with error 104, Connection reset by peer
    REPORT1(myWrite(daSktPr1[1], "Will not be read", 17));
    REPORT2(myReadcond(daSktPr1[1], Ba, 20, 1, 0, 0));   // will block until
myClose(daSktPr1[0])

    REPORT2(myReadcond(daSktPr1[1], Ba, 20, 1, 0, 0));   // will return 0
    REPORT1(myWrite(daSktPr[1], "Added", 6));
    REPORT2(myRead(daSktPr2[1], Ba, 20));

    REPORT2(myRead(daSktPr2[1], Ba, 20));

    REPORT1(myClose(daSktPr2[1]));
    REPORT1(myClose(daSktPr1[1]));
    REPORT1(myClose(daSktPr[1]));
    REPORT1(myClose(daSktPr[1]));
    REPORT2(myRead(daSktPr[1], Ba, 20));
    REPORT2(myReadcond(daSktPr[1], Ba, 20, 0, 0, 0));
    REPORT1(myWrite(daSktPr[1], Ba, 20));
    REPORT1(myTcdrain(daSktPr[1]));
}

void threadT42Func(void) // starts at priority 60
{
    const char* threadName = "T42";
    PE_0(pthread_setname_np(pthread_self(), threadName));
    //
    REPORT1(PE_NOT(myWrite(daSktPr[1], "ijkl", 5), 5));
    REPORT0(posixThread threadT41(50, threadT41Func));
    REPORT1(myTcdrain(daSktPr[1])); // will block until myClose(daSktPr[0])

    REPORT1(myTcdrain(daSktPr[1]));
    REPORT0(threadT41.join());

    // output happens at this time from the above REPORT0
}
```

ENSC351Part3Test.cpp
& output

```
void threadT32Func(void) // priority 70 -> priority 40 -> priority 80 ->
priority 40
{
    const char* threadName = "T32";
    PE_0(pthread_setname_np(pthread_self(), threadName));
    //
    REPORT1(mySocketpair(AF_LOCAL, SOCK_STREAM, 0, daSktPr));
    REPORT1(mySocketpair(AF_LOCAL, SOCK_STREAM, 0, daSktPr1));
    REPORT1(mySocketpair(AF_LOCAL, SOCK_STREAM, 0, daSktPr2));
    //
    REPORT1(PE_NOT(myWrite(daSktPr[0], "abcd", 4), 4));
    REPORT0(posixThread threadT42(60, threadT42Func));
    REPORT1(myTcdrain(daSktPr[0])); // will block until 1st myReadcond(...,
12, ...);

    REPORT1(setSchedPrio(40));
    REPORT1(PE_NOT(myWrite(daSktPr[0], "123456789", 10), 10)); // don't
forget nul termination character

    REPORT1(setSchedPrio(80));
    REPORT1(PE_NOT(myWrite(daSktPr[0], "xyz", 4), 4));
    REPORT1(PE(myTcdrain(daSktPr[0])));
    REPORT1(myClose(daSktPr[0]));
    REPORT1(setSchedPrio(40));

    sched_yield(); // I noticed once the above priority change was not
immediately respected
    REPORT1(myClose(daSktPr1[0]));

    REPORT1(PE_NOT(myWrite(daSktPr2[0], "mno", 4), 4));

    REPORT1(myClose(daSktPr2[0]));

    REPORT0(threadT42.join());
}

int main() {
    CPU_SET(myCpu, &cpu_set);
    const char* threadName = "Pri";
    PE_0(pthread_setname_np(pthread_self(), threadName));
    PE(sched_setaffinity(0, sizeof(cpu_set), &cpu_set)); // set processor
affinity for current thread

    // Pre-allocate some memory for the process.
    // Seems to make priorities work better, at least
    // when using gdb.
    // For some programs, the amount of memory allocated
    // here should perhaps be greater.
    void* ptr = malloc(20000);
    free(ptr);

    try{
        sched_param sch;
        int policy = -1;
        int primaryPriority = 90;
```

ENSC351Part3test.cpp
& output

```
        getSchedParam(&policy, &sch);
        if (sch.__sched_priority < 98)
            cout << "**** If you are debugging, debugger is not running at a
high priority. ****\n" <<
                                " **** This could cause problems with debugging.
Consider debugging\n" <<
                                " **** with the proper debug launch configuration
****" << std::endl;
        cout << "Primary Thread was executing at policy " << policy << " and
priority " <<  sch.sched_priority << endl;
        sch.__sched_priority = primaryPriority;
        setSchedParam(SCHED_FIFO, sch); //SCHED_FIFO == 1, SCHED_RR == 2
        getSchedParam(&policy, &sch);
        cout << "Primary Thread now executing at policy (should be 1) " <<
policy << " and priority (should be " << primaryPriority << ") " <<
sch.sched_priority << endl;

        REPORT0(posixThread T32(SCHED_FIFO, 70, threadT32Func));
        REPORT0(T32.join());

        return 0;
    }
    catch (std::system_error& error){
        cout << "Error: " << error.code() << " - " << error.what() << endl;
        return error.code().value();
    }
    catch (...) { throw; }
}
```

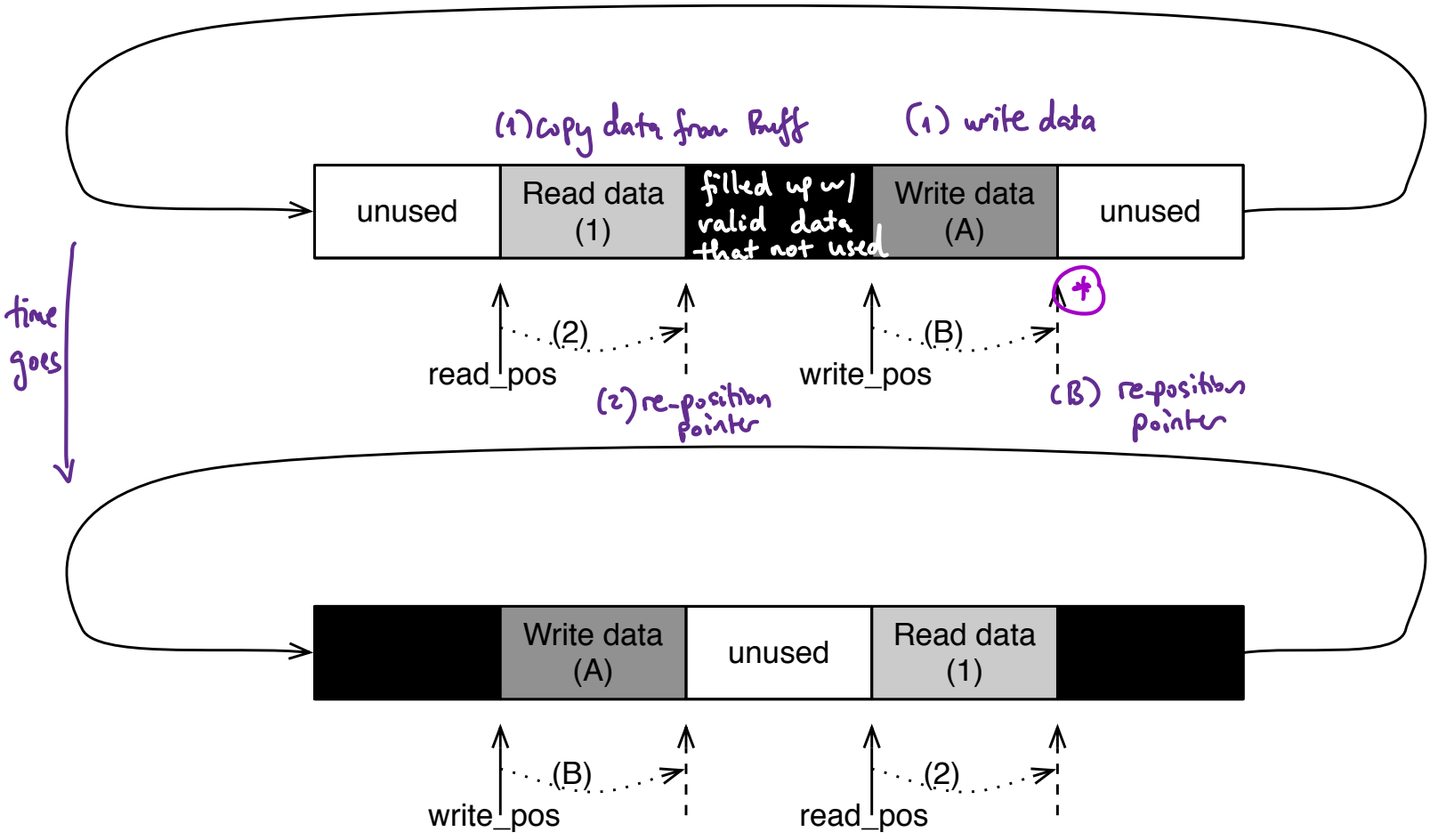
```
Pri: posixThread T32(SCHED_FIFO, 70, threadT32Func); statement will now be started
Pri: posixThread T32(SCHED_FIFO, 70, threadT32Func); statement has now finished
Pri: T32.join(); statement will now be started
T32: mySocketpair(AF_LOCAL, SOCK_STREAM, 0, daSktPr) will now be called
T32: mySocketpair(AF_LOCAL, SOCK_STREAM, 0, daSktPr) result was 0
T32: mySocketpair(AF_LOCAL, SOCK_STREAM, 0, daSktPr1) will now be called
T32: mySocketpair(AF_LOCAL, SOCK_STREAM, 0, daSktPr1) result was 0
T32: mySocketpair(AF_LOCAL, SOCK_STREAM, 0, daSktPr2) will now be called
T32: mySocketpair(AF_LOCAL, SOCK_STREAM, 0, daSktPr2) result was 0
T32: PE_NOT(myWrite(daSktPr[0], "abcd", 4), 4) will now be called
T32: PE_NOT(myWrite(daSktPr[0], "abcd", 4), 4) result was 4
T32: posixThread threadT42(60, threadT42Func); statement will now be started
T32: posixThread threadT42(60, threadT42Func); statement has now finished
T32: myTcdrain(daSktPr[0]) will now be called
T42: PE_NOT(myWrite(daSktPr[1], "ijkl", 5), 5) will now be called
T42: PE_NOT(myWrite(daSktPr[1], "ijkl", 5), 5) result was 5
T42: posixThread threadT41(50, threadT41Func); statement will now be started
T42: posixThread threadT41(50, threadT41Func); statement has now finished
T42: myTcdrain(daSktPr[1]) will now be called
T41: myReadcond(daSktPr[1], Ba, 20, 12, 0, 0) will now be called
T32: myTcdrain(daSktPr[0]) result was 0
T32: setSchedPrio(40) will now be called
T32: setSchedPrio(40) result was 0
T32: PE_NOT(myWrite(daSktPr[0], "123456789", 10), 10) will now be called
T41: myReadcond(daSktPr[1], Ba, 20, 12, 0, 0) result was 14 Ba: abcd123456789
T41: myReadcond(daSktPr[1], Ba, 20, 0, 0, 0) will now be called
T41: myReadcond(daSktPr[1], Ba, 20, 0, 0, 0) result was 0
T41: myWrite(daSktPr[1], "Will not be read", 17) will now be called
```

ENSC351Part3test.cpp
& output

```
T41: myWrite(daSktPr[1], "Will not be read", 17) result was 17
T41: myReadcond(daSktPr[1], Ba, 20, 12, 0, 0) will now be called
T32: PE_NOT(myWrite(daSktPr[0], "123456789", 10), 10) result was 10
T32: setSchedPrio(80) will now be called
T32: setSchedPrio(80) result was 0
T32: PE_NOT(myWrite(daSktPr[0], "xyz", 4), 4) will now be called
T32: PE_NOT(myWrite(daSktPr[0], "xyz", 4), 4) result was 4
T32: PE(myTcdrain(daSktPr[0])) will now be called
T32: PE(myTcdrain(daSktPr[0])) result was 0
T32: myClose(daSktPr[0]) will now be called
T32: myClose(daSktPr[0]) result was 0
T32: setSchedPrio(40) will now be called
T42: myTcdrain(daSktPr[1]) result was 0
T42: myTcdrain(daSktPr[1]) will now be called
T42: myTcdrain(daSktPr[1]) result was 0
T42: threadT41.join(); statement will now be started
T41: myReadcond(daSktPr[1], Ba, 20, 12, 0, 0) result was 4 Ba: xyz
T41: myReadcond(daSktPr[1], Ba, 20, 1, 0, 0) will now be called
T41: myReadcond(daSktPr[1], Ba, 20, 1, 0, 0) result was -1 errno 104: Connection
reset by peer
T41: myWrite(daSktPr1[1], "Will not be read", 17) will now be called
T41: myWrite(daSktPr1[1], "Will not be read", 17) result was 17
T41: myReadcond(daSktPr1[1], Ba, 20, 1, 0, 0) will now be called
T32: setSchedPrio(40) result was 0
T32: myClose(daSktPr1[0]) will now be called
T41: myReadcond(daSktPr1[1], Ba, 20, 1, 0, 0) result was -1 errno 104: Connection
reset by peer
T41: myReadcond(daSktPr1[1], Ba, 20, 1, 0, 0) will now be called
T41: myReadcond(daSktPr1[1], Ba, 20, 1, 0, 0) result was 0
T41: myWrite(daSktPr[1], "Added", 6) will now be called
T41: myWrite(daSktPr[1], "Added", 6) result was -1 errno 32: Broken pipe
T41: myRead(daSktPr2[1], Ba, 20) will now be called
T32: myClose(daSktPr1[0]) result was 0
T32: PE_NOT(myWrite(daSktPr2[0], "mno", 4), 4) will now be called
T41: myRead(daSktPr2[1], Ba, 20) result was 4 Ba: mno
T41: myRead(daSktPr2[1], Ba, 20) will now be called
T32: PE_NOT(myWrite(daSktPr2[0], "mno", 4), 4) result was 4
T32: myClose(daSktPr2[0]) will now be called
T41: myRead(daSktPr2[1], Ba, 20) result was 0
T41: myClose(daSktPr2[1]) will now be called
T41: myClose(daSktPr2[1]) result was 0
T41: myClose(daSktPr1[1]) will now be called
T41: myClose(daSktPr1[1]) result was 0
T41: myClose(daSktPr[1]) will now be called
T41: myClose(daSktPr[1]) result was 0
T41: myClose(daSktPr[1]) will now be called
T41: myClose(daSktPr[1]) result was -1 errno 9: Bad file descriptor
T41: myRead(daSktPr[1], Ba, 20) will now be called
T41: myRead(daSktPr[1], Ba, 20) result was -1 errno 9: Bad file descriptor
T41: myReadcond(daSktPr[1], Ba, 20, 0, 0, 0) will now be called
T41: myReadcond(daSktPr[1], Ba, 20, 0, 0, 0) result was -1 errno 9: Bad file
descriptor
T41: myWrite(daSktPr[1], Ba, 20) will now be called
T41: myWrite(daSktPr[1], Ba, 20) result was -1 errno 9: Bad file descriptor
T41: myTcdrain(daSktPr[1]) will now be called
T41: myTcdrain(daSktPr[1]) result was -1 errno 9: Bad file descriptor
T42: threadT41.join(); statement has now finished
T32: myClose(daSktPr2[0]) result was 0
T32: threadT42.join(); statement will now be started
T32: threadT42.join(); statement has now finished
Pri: T32.join(); statement has now finished
```

- write socketPair using circular Buffer

Operation of lock-free circular buffer



Circular Buffer

- you don't need mutex
- you can write and read at the same time.

- interupt service routine is not allowed to lock the mutex but thread is

of 2 re-position pointer (2) and (B)

(*) if you do (B) re-position pointer first, the reader may misunderstand and keep reading, up to "block" block, even up to "write data" block and end at (B): re-position pointer

351 Sample Midterm Exam

Enter "D100" on the computer answer sheet under "SECTION" and "0001" under "SPECIAL CODE". When you are finished with the exam, hand in these exam pages, the dataServer.c code, as well as the computer sheet.

Please refer to the distributed code "dataServer.c" for questions A, B, and 01 through 10. This code is ***based*** on the time1.c code from the Krten textbook, but has significant differences. There are two written questions (marks as specified) and 12 multiple-choice questions (1 mark each), for a total of 20 marks.

A. Consider a single-threaded client that has a loop in which it will request data from the server, and when it

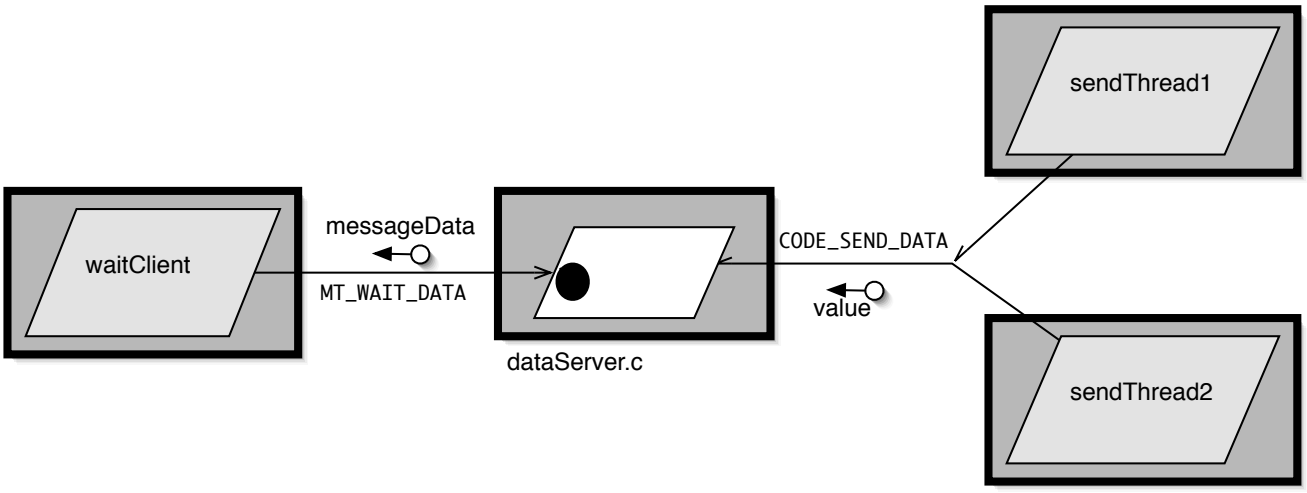
loop
3 marks)

```
for (;;) {
    clientMessageT msg, rmsg;
    msg.messageType = MT_WAIT_DATA;
    MsgSend(coid, &msg, sizeof(msg), &rmsg, sizeof(rmsg));
    // error handling could go here if MsgSend returned -1

    // if (rmsg.messageType == MT_OK)
        cout << rmsg.messageData << endl;
}
```

B. Draw using the collaboration graph notation the server, one data-requesting client, and two data-supplying threads (you may assume each is in a separate process if you wish). Indicate send blocking or reply blocking if either is to be expected. (5 marks)

CGN diagram for the midterm.



Answer each of the following multiple-choice questions. Circle your choice for each question on these pages and eventually fill in the computer answer sheets.

01. In function *main()*, the line "exit (EXIT_FAILURE)" does what?

a. Ends the primary thread and then immediately ends the process.
b. Ends the process but the primary thread still remains.
c. Ends the primary thread but the process still remains.
d. Calls the *exit()* function after which the program goes on to the *receiveMessages()* function.
e. Exits the block of code associated with the "if" statement.

For questions 2 through 10… In function *main()*, consider if the line

```
pthread_create (NULL, NULL, receiveMessages, NULL);
```

was put before the line

```
receiveMessages (NULL);
```

(i.e. "uncomment" the *pthread_create()* line)

02. How many threads would now be in the process:

a. 0
b. 1
c. 2
d. 3
e. more than 3

03. Which variable and/or data structures would **have** to be protected from concurrent access to ensure correct operation of the program:

a. msg
b. clients[0], clients[1], …, clients[MAX_CLIENT – 1]
c. chid
d. both b and c
e. none of the above

04. Of the following choices, with QNX what would be the most-efficient way to protect any variables or data structures needing protection:

a. use one or more mutexes
b. use one or more semaphores
c. use one or more barriers
d. use message passing
e. No variables or data structures need to be protected

05. If a semaphore named `sem4` and initialized to 1 was being used in the program to protect the consistency of one or more variables and/or data structures, what modified code would allow correct operation of these lines from function *gotAMessage()*:

```
for (i = 0; i < MAX_CLIENT; i++) {

    if (!clients [i].in_use) {

        // found one -- mark as in use, save rcvid
        clients [i].in_use = 1;
        clients [i].rcvid = rcvid;
        return;
    }
}
```

a.

```
sem_wait(&sem4);
for (i = 0; i < MAX_CLIENT; i++) {
    if (!clients [i].in_use) {
        clients [i].in_use = 1;
        clients [i].rcvid = rcvid;
        return;
    }
}
sem_post(&sem4);
```

b.

```
for (i = 0; i < MAX_CLIENT; i++) {
    sem_post(&sem4);
    if (!clients [i].in_use) {
        clients [i].in_use = 1;
        clients [i].rcvid = rcvid;
        return;
    }
    sem_wait(&sem4);
}
```

c.

```
for (i = 0; i < MAX_CLIENT; i++) {
    sem_post(&sem4);
    if (!clients [i].in_use) {
        clients [i].in_use = 1;
        clients [i].rcvid = rcvid;
        sem_wait(&sem4);
        return;
    }
    sem_wait(&sem4);
}
```

d.

```
for (i = 0; i < MAX_CLIENT; i++) {
    sem_wait(&sem4);
    if (!clients [i].in_use) {
        clients [i].in_use = 1;
        clients [i].rcvid = rcvid;
        sem_post(&sem4);
        return;
    }
    sem_post(&sem4);
}
```

e.

```
for (i = 0; i < MAX_CLIENT; i++) {
    sem_wait(&sem4);
    if (!clients [i].in_use) {
        sem_wait(&sem4);
        clients [i].in_use = 1;
        clients [i].rcvid = rcvid;
        sem_post(&sem4);
        return;
    }
    sem_post(&sem4);
}
```

06. Comparing the program resulting from a correct modification from question 5 (and any other needed modifications) with the original program (i.e. the program exactly as was given out with this exam), on average how much time would transpire using the modified program between receiving a synchronous (i.e. non-pulse) message and replying to it? Assume the programs are being run on a **uniprocessor** system.
    a. roughly half the time it would take the original program
    b. a little less time than it would take the original program
    c. exactly the same time it would take the original program
    d. a little more time than it would take original program
    e. roughly twice the time it would take the original program

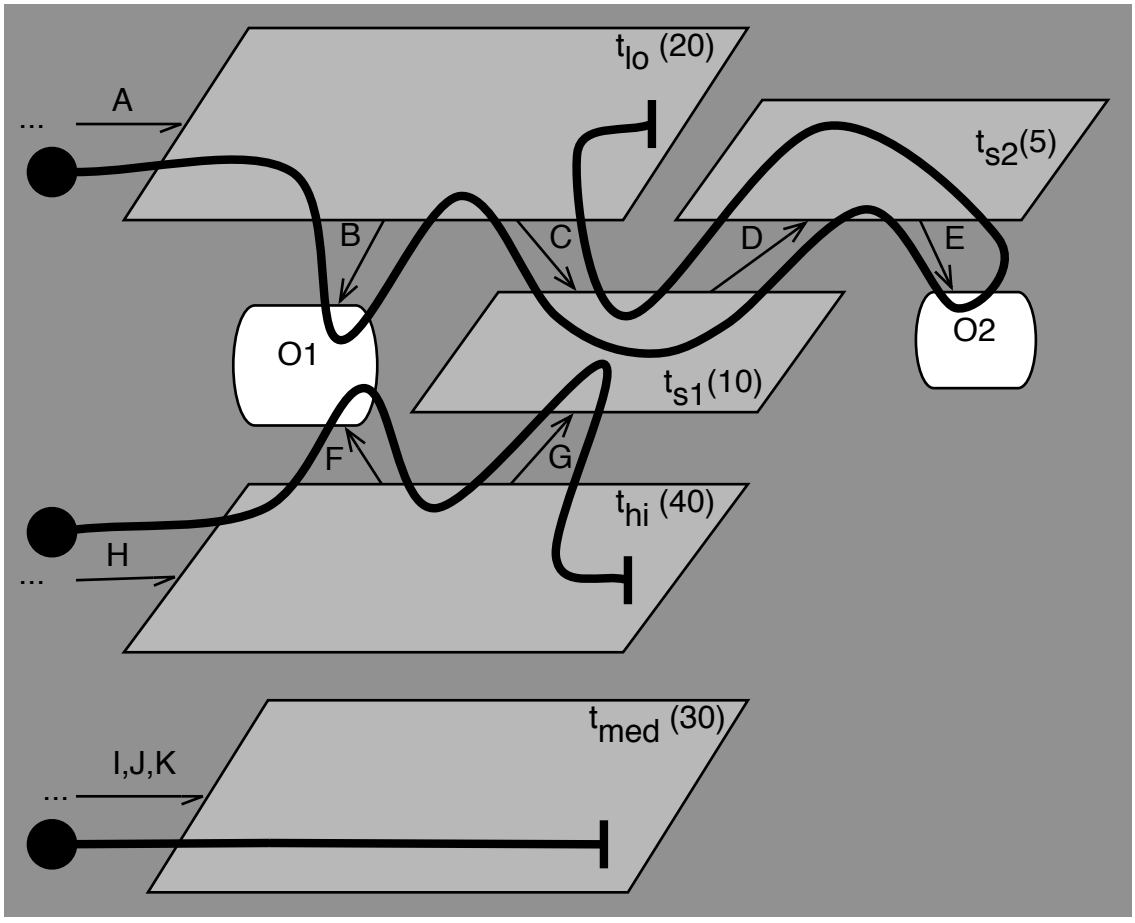
07. Using for both programs a **very lightly loaded 8-processor symmetric multiprocessor (SMP)** system, on average how much time would transpire using the modified program between receiving a synchronous message and replying to it?
    a. roughly 1/8 of the time it would take the original program
    b. roughly half the time it would take the original program
    c. roughly the same time it would take the original program
    d. roughly twice the time it would take the original program
    e. roughly 8 times the time it would take the original program

08. Considering FIFO and standard Round Robin scheduling for the resulting program …
    a. If the threads in the program use standard Round Robin scheduling, more context switching would typically result.
    b. The choice of FIFO or standard Round Robin would typically make no difference to this program.
    c. Round Robin scheduling would be the best choice for this program.
    d. both a and c are correct
    e. none of the above are correct

09. If we turn off priority inheritance and if the thread created by the `pthread_create()` call is given a lower priority than the primary thread …
    a. on a uniprocessor system the created thread would never be given the chance to start running
    b. on a lightly loaded symmetric multiprocessor system the created thread would never be given the chance to start running
    c. the created thread would receive about half the pulses and messages on a lightly loaded multiprocessor
    d. both a and b are correct
    e. none of the above are correct

10. Considering *pthread_join(), pthread_barrier_init(),* and *pthread_barrier_wait()* …
    a. One or more *pthread_join()* calls are needed for this program to run properly.
    b. One or more *pthread_barrier_init()* calls are needed for this program to run properly.
    c. One or more *pthread_barrier_wait()* calls are needed for this program to run properly.
    d. both b and c are correct.
    e. none of the above are correct

Refer to the following diagram for the remaining questions.



11. Assuming default QNX priority inheritance behaviour and that the priority of threads when created are shown in parentheses, what priority will thread t_{s1} be running at immediately after the sequence A,B,C,I,H,F

a. 10.
b. 20.
c. 30.
d. 40.
e. None of the above.

12. With the previous assumptions, what priority will thread t_{s1} be running at immediately after the sequence A,B,C,I,H,F,G

a. 10.
b. 20.
c. 30.
d. 40.
e. None of the above.