ENSC 351: Real-time and Embedded Systems

Craig Scratchley, Fall 2021

Multipart Project Part #1

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

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

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

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

I have listed a good C++ book in the course syllabus. The book for ENSC 251 might also be helpful.

The first part of the project focuses on parts of the YMODEM-batch file transfer protocol.

Modify and complete the designated function members in the SenderY class in SenderY.cpp and in the PeerY class in PeerY.cpp. For this first part of the project, we are just simulating the sending of a couple files, including "/home/osboxes/hs_err_pid11506.log", using 16-bit CRC to detect errors and having all blocks hold 128 bytes of data. The protocol is described in the "ymodem.txt" file at

http://pauillac.inria.fr/~doligez/zmodem/ymodem.txt

and that file will be our primary reference. I've noticed a few ways to improve that file and so may create my own edited version of it and post it on Canvas. Other reference documents may be provided as well.

For the first part of the project, instead of sending the blocks and any other characters over a serial port or similar, the blocks and other characters should simply be written to an output file, "ymodemSenderData.dat". This file should only contain bytes that would normally be sent from the sender to the receiver. We are just imagining that a receiver exists and are assuming that it will correctly start the sender, positively acknowledge every block that it receives, and properly handle the termination of each transfer after it has received all the blocks for each file in the batch. A YMODEM sender implementation is allowed to send blocks that can hold either 128 bytes or 1024 bytes of data. At least for this first part of the project, we will generate only blocks that can hold just 128 bytes of data.

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

documentation on POSIX functions online and on the Xubuntu Virtual Machine that I created for the class. We will be using Linux this term, and I recommend that everybody use the Xubuntu VM that I have created.

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

Assume that the imaginary receiver sends a 'C' to begin the transfer of file information or file contents and always sends an ACK for each block. After the last block of a file is ACKed, send an EOT, which will be NAKed, and then repeat the EOT. So for a single file that needs 4 blocks and has no transmission errors an imaginary receiver requesting CRCs (and being provided information on the file) would send
'C', ACK, 'C', ACK, ACK, ACK, ACK, NAK, ACK, 'C', ACK

Except where indicated in the code, for now don't worry about references to the CAN byte in the specification, and don't worry about timing. Just assume that, for example, an ACK will be quickly received after each block is sent.

We plan to grade your code on both a little-endian system and a big-endian system. So be very careful when writing a CRC16 in Network Byte Order. More on that in class.

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

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

September 2021

https://youtu.be/ut-h5ZBYj0w

BBS: bullet board system

symbol rate

BaseName(): extract the file's name

Command:

man sb

man rb

man diff binary difference.

v bin diff _ [first file] _ [second file]

man -k compare

man 3 basename

↑ section 3 of basename's manual

* Add a second project to eclipse: (multi-projects)

- project (L-side) > white space > import (RClick) > general > existing project ... > select archive file > (select where zip file)

- RClick to project

  > build project

  Console (bottom)

  eclipse invokes linker → linker involkes executable file

  > Debug as > local C++ Application

  Debug prespective (Rside)

  Debugger console (bottom)

  (gdb) print mediumD ask for info

ENSC 351 part 1

```
int main() {
#ifdef __MINGW32__
   _fmode = _O_BINARY;  // needed for MinGW compiler which runs on MS Windows
#endif

   // for x86_64, output file will be in the Eclipse project.
   // for ppc/mips, output file will be in the home directory:  /home/osboxes
   const char* oFileName = "ymodemSenderData.dat";
   mode_t mode = S_IRUSR | S_IWUSR; // | S_IRGRP | S_IROTH;
   int mediumD = myCreat(oFileName, mode);
   if(mediumD == -1) {
      cout /* cerr */ << "Error opening medium file named: " << oFileName << endl;
      ErrorPrinter("creat(oFileName, mode)", __FILE__, __LINE__, errno);
      return -1;
   }

   vector<const char*> iFileNamesA = {"/doesNotExist.txt"};
   vector<const char*> iFileNamesB = {"/home/osboxes/.sudo_as_admin_successful", "/home/osboxes/hs_err_pid11506.log"};

   testSenderY(iFileNamesA, mediumD);          // file does not exist
   testSenderY(iFileNamesB, mediumD);          // empty file and normal text file

   if (-1 == myClose(mediumD)) {
      ErrorPrinter("close(mediumD)", __FILE__, __LINE__, errno);
      return -1;
   }
   else
      return 0;
}
```

used to write to .dat
// 0 is standard input
// descriptor
// -1 is error input
// send 2 CAN characters

```
#include "SenderY.h"

#include <iostream>
#include <stdint.h> // for uint8_t
#include <string.h> // for memset(),memcpy()
#include <errno.h>
#include <fcntl.h>      // for O_RDWR
#include <sys/stat.h>

class SenderY : public PeerY
{
      friend # include <string.h> // memset(),
      void testSenderY(std::vector<const char*> iFileNames, int mediumD)
      ;

public:
      SenderY(std::vector<const char*> iFileNames, int d);
      void statBlk(const char* fileName);
      //void sendBlk(blkT blkBuf);
      void sendBlk(uint8_t blkBuf[BLK_SZ_CRC]);
      void sendFiles();
    void cans();
      ssize_t bytesRd;  // The number of bytes last read from the input file.

private:
      std::vector<const char*> fileNames;
      unsigned fileNameIndex;
      uint8_t blkBuf[BLK_SZ_CRC];     // a  block
      // blkT blkBuf;    // A block // causes inability to debug this array.
      uint8_t blkNum;   // number of the current block to be acknowledged
      //void genBlk(blkT blkBuf);
      void genBlk(uint8_t blkBuf[BLK_SZ_CRC]);
      //void genStatBlk(blkT blkBuf, const char* fileName);
      void genStatBlk(uint8_t blkBuf[BLK_SZ_CRC], const char* fileName)
      ;
};

#endif
```

memcpy() copy filenam to the Buff

SenderY.h

SenderY.cpp

```
void SenderY::sendFiles()
{
//for (auto fileName : fileNames) {      for (unsigned fileNameIndex = 0 ; fileNameIndex
    // const char* fileName = fileNames[fileNameIndex];
    transferringFileD = myOpen(fileName, O_RDWR, 0);
    if(transferringFileD == -1) {
      // ******** fill in some code here to write 8 CAN characters **********
      cans();
      cout /* cerr */ << "Error opening input file named: " << fileName << endl;
      result = "OpenError";
      return;
    }
    else {
      cout << "Sender will send " << fileName << endl;

      // do the protocol, and simulate a receiver that positively acknowledges every
      // block that it receives.

      statBlk(fileName);

      // assume 'C' received from receiver to enable sending with CRC
      genBlk(blkBuf); // prepare 1st block
      while (bytesRd)
      {
        blkNum ++; // 1st block about to be sent or previous block was ACK'd

        sendBlk(blkBuf); // send block

        // assume sent block will be ACK'd
        genBlk(blkBuf); // prepare next block
        // assume sent block was ACK'd
      };
      // finish up the file transfer, assuming the receiver behaves normally and there are no transmission errors
      // ******** fill in some code here **********

      //(myClose(transferringFileD));
      if (-1 == myClose(transferringFileD))
        ErrorPrinter("myClose(transferringFileD)", __FILE__, __LINE__, errno);
    }
  }
  // indicate end of the batch.
  statBlk("");

  result = "Done";
}
```

network byte order ~ big edian

Figure 4. YMODEM Batch Transmission Session (2 files)

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

Figure 5. YMODEM Batch Transmission Session-1k Blocks

```
SENDER                                          RECEIVER
(command: ------------------------------------------"sb -k foo.*<CR>")
"sending in batch mode etc."
                                                C (command: "rb<CR>") C (command:rb
SOH 00 FF foo.c NUL[123] CRC CRC
                                                ACK
                                                C
STX 01 FD FE Data[1024] CRC CRC
                                                ACK
SOH 02 FC FD Data[128] CRC CRC
                                                ACK
SOH 03 FB FC Data[100] CPMEOF[28] CRC CRC
                                                ACK
EOT
                                                NAK
EOT
```

in code

```
SOH 00 FF NUL[128] CRC CRC
```

```
ACK
C

ACK
```

Figure 6. YMODEM Filename block transmitted by sz

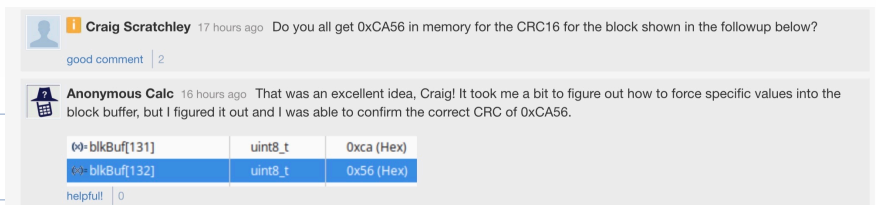
```
-rw-r--r--  6347 Jun 17 1984 20:34 bbcsched.txt
```

on . dat

```
00 0100FF62 62637363 6865642E 74787400   |...bbcsched.txt.|
10 36333437 20333331 34373432 35313320   |6347 3314742513 |
20 31303036 34340000 00000000 00000000   |100644..........|
30 00000000 00000000 00000000 00000000
40 00000000 00000000 00000000 00000000
50 00000000 00000000 00000000 00000000
60 00000000 00000000 00000000 00000000
70 00000000 00000000 00000000 00000000
80 000000CA 56
```

b b c s c

CRC-16



- Sender Y > genBlk

  uint16_t myCrc16ns;

  crc16ns (&myCrc16ns, &blkBuf[DATA_POS]);

  blkBuf[131] = myCrc16ns >> 8

  blkBuf[132] = myCrc16ns
- SOH 00 FF foo.c filesize NUL[..?..] CRC CRC
- NUL[123] fill up with "0" ?
- PeerY.cpp > crc16ns

  Do we need to worry about crc flg for now ?

  checksum

  How does the oldcrc work ?

  updcrc

what is sendfile (int) oldcrc >> 8 ?

↳ why shift needed?

- piazza!

SOH 00 FF foo.c NUL 100 NUL [128_5_J] CRC CRC

NUL → no more info coming

100 → filesize

128_5_J → fileNameSize

J → 100 presented in 3bytes

SenderY.cpp

```
/* generate a block (numbered 0) with filename and filesize */
//void SenderY::genStatBlk(blkT blkBuf, const char* fileName)
void SenderY::genStatBlk(uint8_t blkBuf[BLK_SZ_CRC], const char* fileName)
{
   // ********* additional code must be written ***********

      struct stat st;
      stat(fileName, &st);
      // unsigned fileSize = st.st_size;

   /* calculate and add CRC in network byte order */
   // ********* The next couple lines need to be changed ***********
   uint16_t myCrc16ns;
   crc16ns(&myCrc16ns, &blkBuf[0]);
}
```

there are at least 1 block 0th,
after reach 256th block,
it's back to block 0th

max fileName size on Linux ?

fileName > blkBuf (not worry in Part 1)

Variables tab (Rside) in Debug mode does not show global vars

Debugger console (bot): (gdb) print std::cout

MIPS Debugger:

Project tab (Rside) > RClick on project > properties > C/C++ Build (Rside) > Manage Configuration (L-side) > Select Mips > set active > ok > apply and close
Build project > Press F5 (Refresh) > look for binaryfile

senderY.cpp

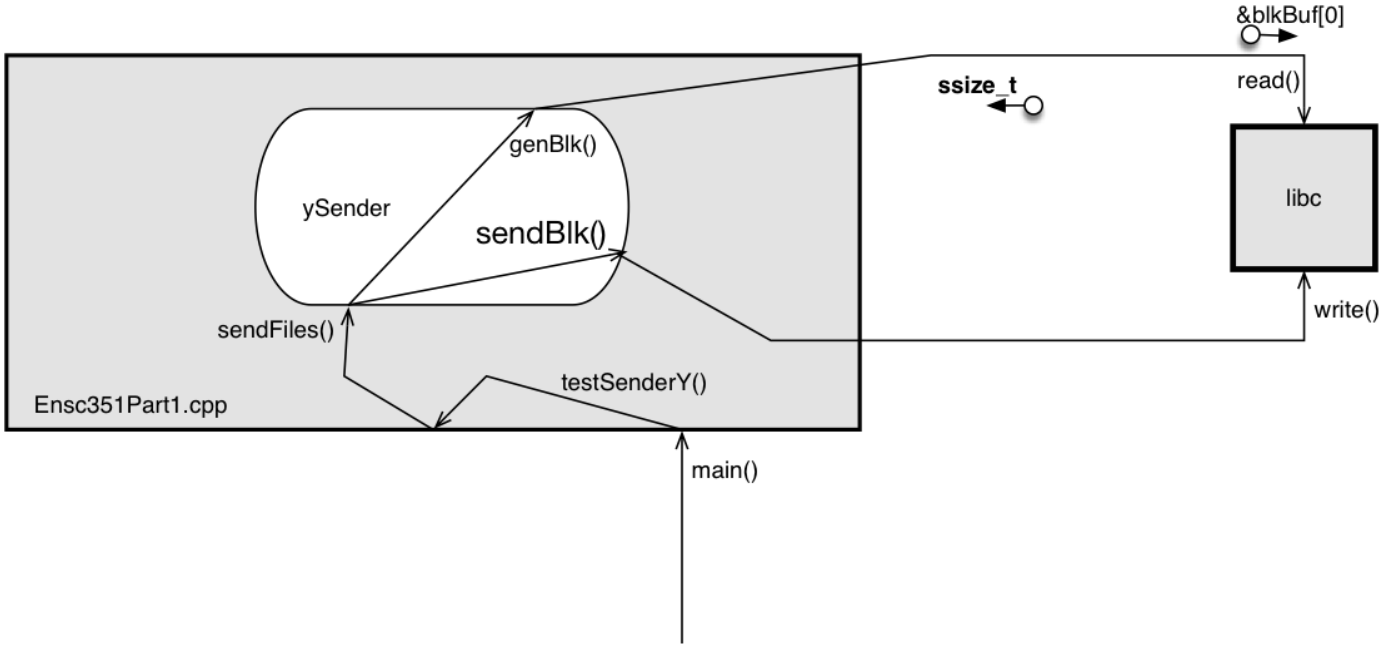
```
#include "SenderY.h"

#include <iostream>
#include <experimental/filesystem> // for C++14
#include <filesystem>
#include <stdio.h> // for snprintf()
#include <stdint.h> // for uint8_t
#include <string.h> // for memset(), and memcpy() or strncpy()
#include <errno.h>
#include <fcntl.h>     // for O_RDWR
#include <sys/stat.h>

#include "myIO.h"

using namespace std;
using namespace std::filesystem; // C++17
//using namespace experimental::filesystem; // C++14
```

c++ for file Name

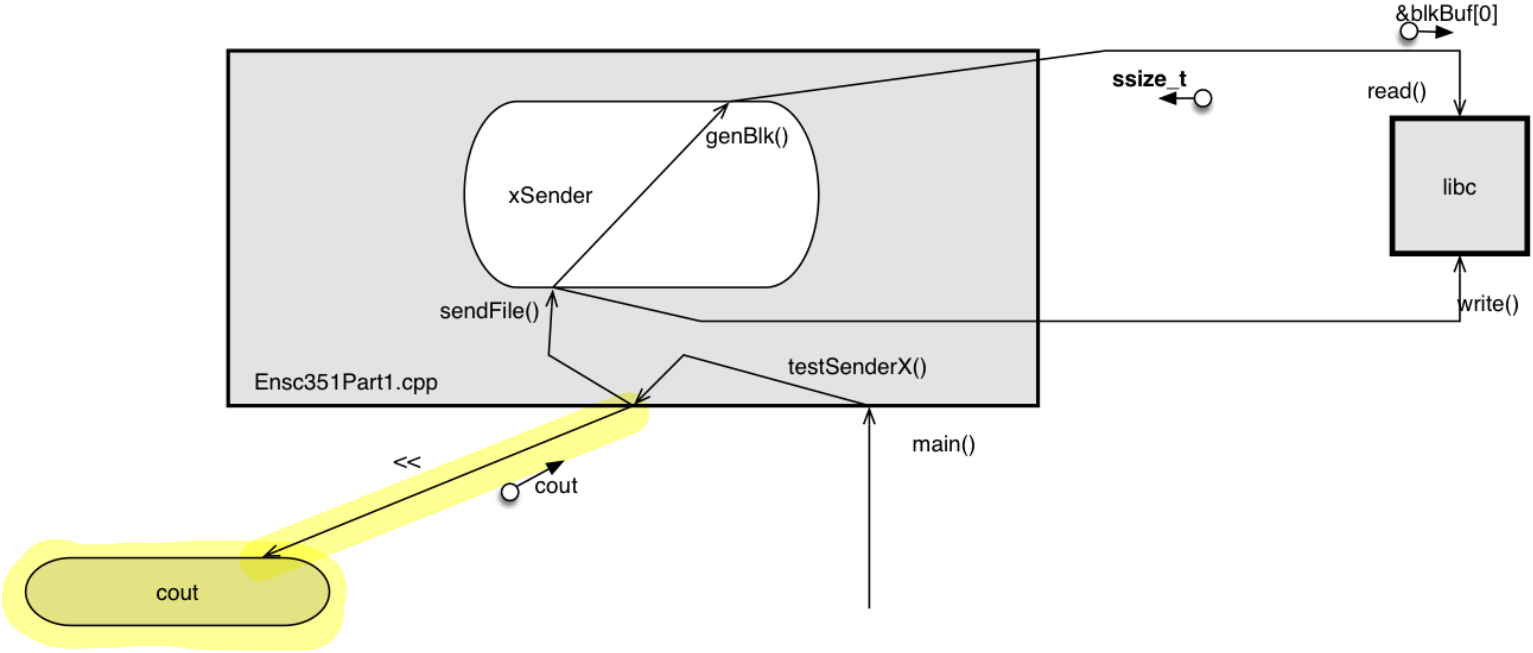
Add library:

Project tab (Rside) > RClick on project > properties > C/C++ General (R-side) > Code Analysis > Paths & Symbols

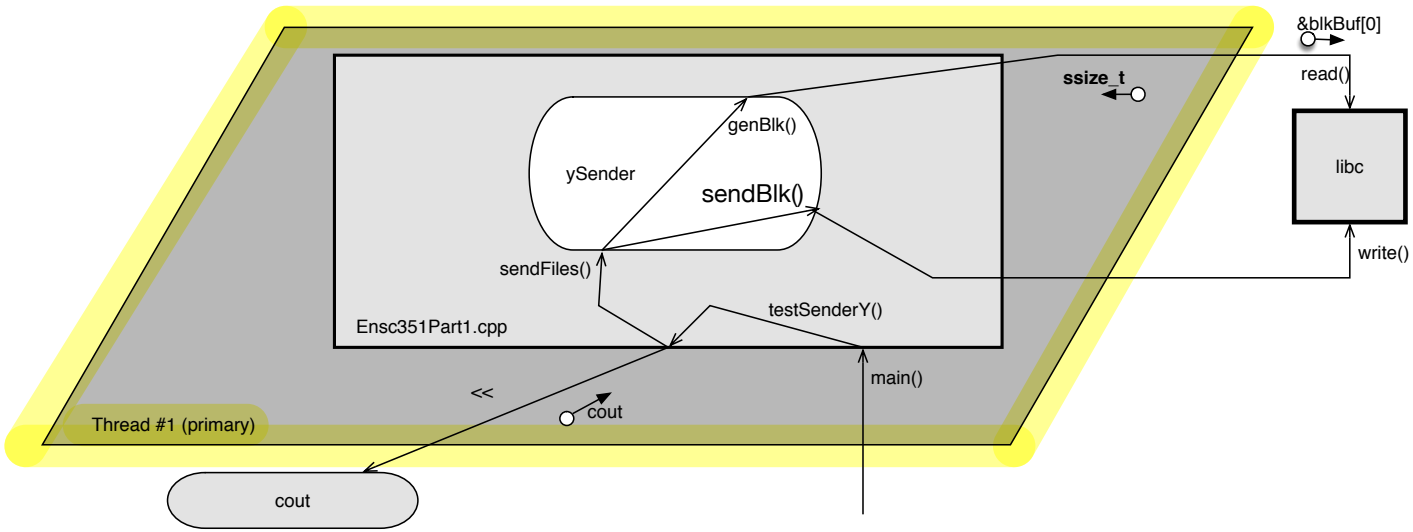
A diagram for Part 1's Ensc351Part1.cpp using Collaboration Graph Notation with data flows.



A diagram for Part 1's Ensc351Part1.cpp using collaboration graph notation with data flows. The global object cout has also been drawn.

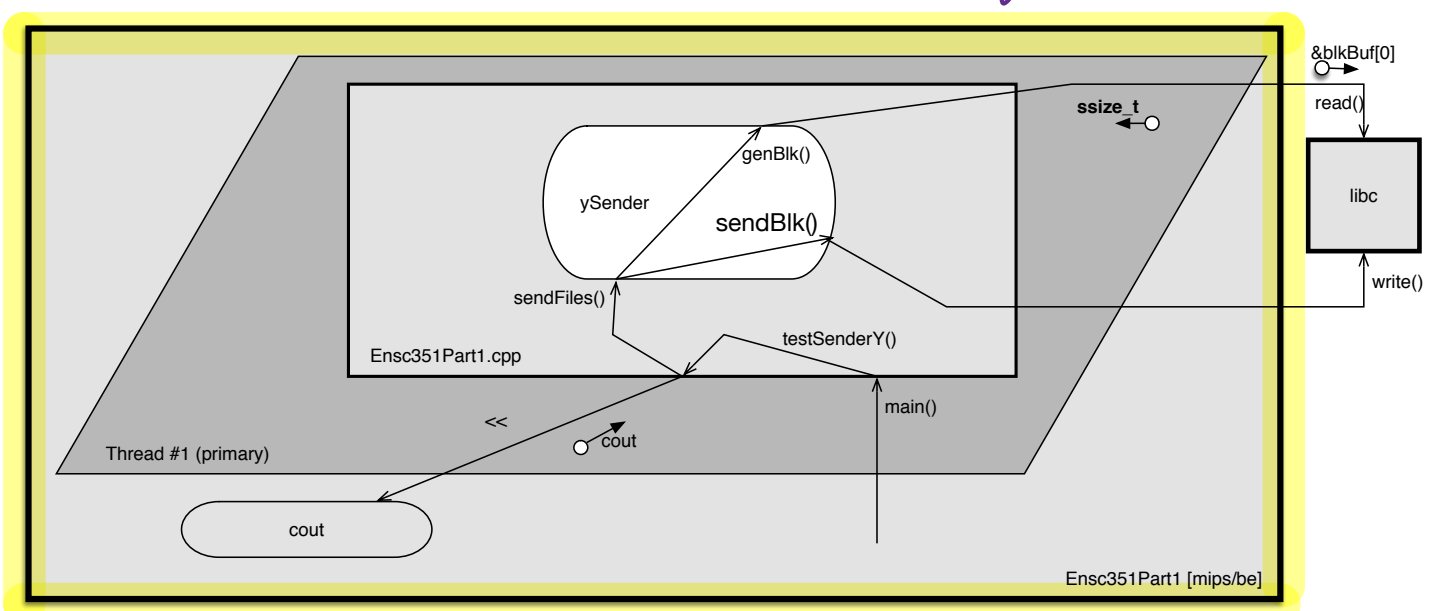


A diagram for Part 1 of Multipart Project using Collaboration Graph Notation. The global object cout and primary thread have also been drawn.



A diagram for the Ensc351Part1 executable using collaboration graph notation. The libc library is shown as being shared.

bold line shows a process



Bits or characters that fill up unused portions of a data structure, such as a field, packet or frame. Typically, padding is done at the end of the structure to fill it up with data, with the padding usually consisting of 1 bits, blank characters or null characters. See null and bit stuffing.

https://www.pcmag.com › Encyclopedia › P

Definition of padding | PCMag