CPE464 – UDP and SendtoErr()

Code must be submitted via the handin process on the Unix servers. The handin directory is **lab_udp**.

The purpose of this lab is to help prepare you for program #3. Program #3 is a file transfer program using UDP.

Using UDP and sendtoErr library

Before starting to code read this entire section to see what you are going to do (read and underline).

In this lab, you are going to work with the UDP code and sendtoErr library provided by Prof. Smith. This work will help you in program #3.

When you are done your rcopy/server will:

- rcopy (based on udpClient.c) must take in an error rate, read in user input, create an application level PDU, print that PDU and then send (using sendtoErr()) the PDU to the server

- server (based on udpServer) must also take an error rate, recvfrom() the PDU from the rcopy and print out the PDU that it received.

I. Initial Steps (just getting the UDP code working):

a. **Get the UDP code:** Download the UDP code from Canvas and untar the file.
   - Test the code to make sure it works

b. **Install sendtoErr() library:** see directions on Canvas in the program #3 section
   - You want to be in the directory with your UDP code before you install the library. The library makefile is called build464lib.mk and creates a library ('.a') file.

   - After downloading the library, untar it and build it according to the directions

c. **Verify the sendtoErr() library**. This library is called libcpe464_2_21.a (the 2_21 may change). To verify the library type: ar xv libcpe464_2_21.a

   This should give you a list of the object (.o) files in the .a (library) file.

d. Change the source code filenames to rcopy.c (from udpClient.c) and server.c (from udpServer.)

e. Modify the Makefile to use the code files rcopy.c/server.c and make the programs rcopy and server.

f. Below you are going to change the UDP code to use the sendtoErr() library.

STOP – at this point you should have a library called *libcpe464_2_21.a* in your directory
STOP – at this point you should also have executables called **rcopy** and **server**.

II. Overview of coding steps:

Just read this section… it gives you an overview of the next section and what you are going to do.

(The details of these steps are given in the next section, this is just an overview)

1. In file **separate** from your client and server code you will:
   a. Write a function to create an application level PDU (createPDU()) in the proper format for program #3. See below for more details. This includes using the in_cksum() function to generate the PDU checksum which then gets placed into your PDU.
   b. Write a function to print out a PDU (printPDU()).
2. You will modify rcopy.c to use your createPDU() function and to print out your PDU using printPDU(). This PDU is then sent to the server.
3. You will modify server.c to print out the received PDU using your printPDU() function.
4. You will modify server.c to send back the exact PDU it received from the client (rcopy.c). The client will then call printPDU() on the PDU received from the server.
5. You will modify your client and server to use the sendtoErr_init() and sendtoErr() functions. Including modifying your Makefile to link in the sendtoErr library.

You will use the **sendtoErr()** function that I provide in program #3 in place of the normal sendto() function. This function drops packets and flips bits but does not tell you. The sendtoErr() function's debugging output expects your packets to be in the format (e.g. PDU header) required by program #3.

III. <u>Detailed Coding Steps</u> (do these):

a. You should have already done this (above): Change the code and executable names from udpClient (and udpClient.c) to rcopy (rcopy.c) and udpServer (and udpServer.c) to server (server.c). You will also need to update the Makefile.

b. **Modify the rcopy.c and server.c code to take an error rate** (so the rate of dropping/corrupting packets) as a runtime parameter. This error rate will be between 0 and less than 1. The atof() function converts a string to a double. This error rate is needed for the sendtoErr library.

c. Example runs look like:

Bash$: server **.05** 44444

- This would have a 5% (i.e. .05) error rate
- Use atof() to convert the argv[1] to a float
- The optional server port number (e.g. 44444) follows the error rate

and

Bash$: rcopy **.1** localhost 44444[1]

- This would have a 10% (i.e. .1) error rate
- The server name/IP and port number follow the error rate

d. **Create the PDU:**

Write a function that will be used to create a PDU. After you create this PDU, it should contain the seq#, checksum, flag and payload. This is the format for PDUs in program #3**. This function must be in a separate file.**

The required PDU format (for both the lab and program #3) is:

4-byte sequence number in **network order**	2-byte checksum	1-byte flag	Payload (up to 1400 bytes)

Your createPDU() function should look similar to:

```
// pduBuffer is the buffer you fill with your PDU header/payload
// sequenceNumber = 32 bit sequence number passed in in host order
// flag = the type of the PDU (e.g. flag = 3 means data)
// payload = payload (data) of the PDU (treat the payload as just bytes)
// dataLen = length of the payload (so # of bytes in data),
//           this is used to memcpy the data into the PDU
// returns the length of the created PDU

int createPDU(uint8_t * pduBuffer, uint32_t sequenceNumber, uint8_t flag, uint8_t * payload, int payloadLen)
{
    // create PDU code goes here

    return pduLength;
}
```

Your createPDU() function takes in a 32-bit sequence number, a 1-byte flag, a unit8_t * pointer to a buffer of data and an length (int) of the data buffer. This function puts the sequence number into **network order** prior to putting the number in the PDU. This function also uses the checksum function to calculate a checksum (crc) and put that into the header. This function creates a PDU in the format required by program 3 (seq#, crc, flag, payload). Put this function in a separate .c file and create a .h file to use with both your server and rcopy code.

Note – the checksum function (in_cksum()) is the same checksum function you used in program #1. The sendtoErr() library contains this function and it will be linked into your executable as part of the library. The prototype for this function is:

unsigned short in_cksum(unsigned short *addr, int len);

e. **Create a printPDU() function** that takes in a PDU, verifies the checksum and prints out the sequence number, flag, payload and payload length (note for initial testing the payload will be text). It should also print out an error message if the PDU is corrupted.

[1] FYI.... In program #3, the runtime parameters are different for rcopy.

Put this function in the same .c files as the createPDU() (and update your new .h)

void printPDU(unit8_t * aPDU, int pduLength);

f. **Modify rcopy.c to utilize your createPDU() and printPDU () functions**.

- Since this is just a example (test) program, for the sequence number just use a counter and increment it every time you create a new PDU.
- For this test program, the payload of the PDU is the message typed in by the user.
- For this test program, set the flag to a number between 0 and 255
- Use your **printPDU** to verify your code works!

g. Modify server.c to utilize the printPDU function.

- Note – if the sendtoErr() function drops the PDU when either the client or server tries to send the PDU, the client and server will hang. This is normal behavior. You can just ^c them to start again. In program #3 you will fix this!

h. Modify server.c to send back the received PDU to the client.

i. Modify rcopy.c to receive and print out the PDU sent back from the server.

j. Integrate the sendtoErr() library into the rcopy.c and server.c.

The purpose of the sendtoErr() library is to drop and corrupt packets. The sendtoErr_init() sets up the error rate and other paramaters for this fuction. They you will call sendtoErr() to send your PDUs using UDP. In program #3 you will write the code to recover the corrupted or lost PDUs.

i. Add a call to sendtoErr_init() in both rcopy's and server's main() functions. In this test program errorRate is the value passed in at run time (see earlier step). You also need to add "#include cpe464.h" in your .c file to use this function.

```
sendErr_init(errorRate, DROP_ON, FLIP_ON, DEBUG_ON, RSEED_OFF);
```

ii. Change all sendto() calls used by rcopy/sever to **sendtoErr()**. The recommended way of doing this is to change my safeSendto() function to call sendtoErr() instead of sendto(). This requires you to only change one line of code and works for all of your sends (assuming you are using safeSendto().

iii. Modify the Makefile to link in the sendtoErr library (remove two comments in the Makefile). It should look like this:

```
#uncomment next two lines if your using sendtoErr() library
LIBS += libcpe464.2.21.a -lstdc++ -ldl
CFLAGS += -D__LIBCPE464_
```

iv. NOTE – when you move your code from your home machine to the Unix servers you will need to rebuild the sendtoErr library. The .a file that is created on your home machine will not work on the Unix servers.