CPE464 - Programming Assignment #3 – Selective Reject
Due Monday March 5, 2018

This program is due on **Monday March 5th at 11:59 pm.** If your program is late you will lose 15% per day. The last day you can turn in this assignment for any credit is Thursday March 8th at 11:59 pm.

Extra Credit: If you turn in the program before 11:59 on Friday March 2th you will receive 20% extra credit (You must get the multithreaded/multiprocess part of the program done to receive the extra credit.)

In this assignment you are to write a remote file copy command using **UDP**. To do this you will implement two programs in C (or C++). These are **rcopy** (the client program) and **server**. The **rcopy** program takes two filenames, a local-filename and a remote-filename, as parameters. The transfer of the files is to download the remote file, from the server to rcopy, and store the file as the local-filename on the machine running rcopy. The **server** receives the requested remote-filename from rcopy and then transmits the data that makes up this file to rcopy.

This program will use **a sliding window flow control algorithm using selective-reject ARQ.** This program will use **UDP** and will use the Internet checksum function. You will need to break the file up into packets (datagrams) and append a header to each packet as described below.

In order to induce errors you will **not** use the normal sendto(…) function. Instead you will use a sendtoErr(…) function. This function will periodically drop packets and flip bits in your data You may **not** use the normal sendto(…) anywhere in either (rcopy/server) program.[1]

1) rcopy: This program is responsible for taking the two filenames as command line arguments and communicating with the server program to download the remote file to the local machine (machine running rcopy). The rcopy program will be run as:

   rcopy ***local-file remote-file window-size buffer-size error-percent remote-machine remote-port***

   where:

local-file:	is the name of the file created by rcopy
remote-file:	is the name of file the server sends to rcopy
window-size	is the size of the window in PACKETS (i.e. number of packets in window)
buffer-size:	is the number of data bytes (from the file) transmitted in a data packet[2]
error-percent	is the percent of packets that are in error (floating point number)
remote-machine:	is the remote machine running the server
remote-port:	is the port number of the server application

2) server: This program is responsible for accepting requests from rcopy to download files. This program should never terminate (unless you kill it with a ctrl-c). It should continually process requests to download files to the rcopy clients. To receive full credit the server must process multiple clients simultaneously.

[1] Just to be clear, the remote file is being downloaded from the server to rcopy. During this data transfer the server will utilize a selective reject sliding window protocol to control the transmission and recover from errors.

[2] This buffer-size is the number of bytes you should **read** from disk and send in each packet. This is not the packet size. The packet will be bigger (since it needs to include at least a header.) The last packet of file data may be smaller and file name exchange packets (or similar control packets) can be no bigger than needed to communicate the required information.

The server needs to handle error conditions such as an error opening the output file by passing back an error packet (e.g. error flag) to the rcopy program.

The server will receive the **window-size** from rcopy.

The server should output its port number to be used by the rcopy program. The server program is run as:

server *error-percent optional-port-number*

where:

error-percent	is the percent of packets that are in error (floating point number)
optional-port-number	allows the user to specify the port number to be used by the server (If this parameter is not present you should pass a 0 (zero) as the port number to bind().)

Requirements:

1) Do not use any code off the web or from other students. Do not look at any other code that provides a solution to this problem or parts of this problem. The work you turn in must be your entirely your own. You may make use of the code I handed out in lecture, any code I provided and the Internet Checksum code.

2) Both rcopy and server should call the sendtoErr(…) function for all communications between rcopy and server (e.g. for connection setup, data, RR's). You should use the sendtoErr function for all transfers including the file name. You should **not** use the normal sendto(…) function in your program.

3) You must provide a README file that includes your first and last name and your lab section time (9am, noon, 3pm).

4) Connection setup: You must establish communications between rcopy and the server prior to sending the filename from the server. Therefore, there must have been a successful sending of a packet (flag = 1) sent from rcopy to the server and a packet (flag = 2) from server to rcopy prior to sending any real information (so prior to sending the filename)[3].

5) If the server is not able to open the remote-file it must send back an error to rcopy. rcopy should print out the error message: Error during file open of *remote-file* on server. This could happen if the file does not exist on the server or you do not have read access to the file.

6) Remove or comment out any debugging print statements before you handin your assignment.

7) If you resend a packet **10 times** you can assume the other side is down and terminate gracefully.

8) Your solution for sending the filename should not permanently hang a server process/thread nor should it cause rcopy to terminate just because of a few lost packets. Your solution needs to handle the loss of the filename and the loss of the filename acknowledgement up to **10 times**. This means rcopy should transmit the filename up to 10 times before quitting.

[3] Since you are using UDP you are manually doing the connection setup – UPD is connectionless so you are not using connect() and accept() like you would with TCP. So you are doing a connection setup (flag = 1, flag =2 exchange) prior to using your connection between rcopy and server to send any data (including prior to sending the filename)

9) When sending file data and your window closes on server you should select for 1-second waiting on RRs. If you are sending data and your window is open you should process RRs but you should NOT block on select (use a non-blocking select... so…while your window is open, send one packet, see if any RRs are available, if so process them, repeat the loop).

10) When sending file data, if your window closes on server (meaning you have sent an entire windows worth of data) and then your 1-second select times out on the server, you can break this deadlock by resending the lowest packet (and only the lowest packet) in the window. If rcopy receives a packet lower than expected it should respond with something (an RR or SREJ depending on circumstances) that will reopen the window on server.

11) The maximum filename length is 100 characters. You should check the filename lengths prior to starting your file transfer. Print out an appropriate error message if a filename is too long and terminate rcopy.

12) You cannot call sleep or usleep anywhere in your program.

13) You must use a header in all of your packets. The header should have as a minimum a 32-bit sequence number in network order, the internet checksum and a one byte flag field. So, your packet format for ALL packets should be packet seq#, checksum, flag, data/Filename/whatever.

14) Regarding the format for the RR and SREJ packets (it is basically a normal header created by the sender of the RR/SREJ and then the seq number being RRed/SREJed.) This packet seq number must be in network order.
- **RR packet format:** Packet Seq number, checksum, flag, RR seq number (what you are RRing)[4]
- **SREJ packet format:** Packet Seq number, checksum, flag , SREJ seq number (what you are SREJing)

15) You must use the following flag values in your header. If there are other types of packets you want to send (meaning you need other flag values) that is ok. Please document any additional flag values in your readme file. You must use the following value:

a. Flag = 1 Call setup packet (rcopy to server)
b. Flag = 2 Call setup response packet (server to rcopy)
c. Flag = 3 Data packet
d. Flag = 4 (Not used this quarter)
e. Flag = 5 RR packet
f. Flag = 6 SREJ packet
g. Flag = 7 Packet contains the file name (rcopy to server)
h. Flags > 7 should be used for other things (like filename response, ending the connection or other control functions). You decide how to use these.

16) The program names, rcopy and server must be used. Also the run-time parameters should be in the order given. Name your makefile: Makefile.

17) Use datagram sockets for these programs (UDP). Your program must be written in C (C++). Also, while some protocol stacks do support UDP calls to connect() and accept() – you may not use these functions in your programs.

[4] The packet sequence number is the sequence number created by the sender of the RR/SREJ. Both rcopy and server maintain their own set of sequence numbers and this is the number put into the first 4 bytes of the header.

18) No process (e.g rcopy, server child/thread) should permanently block when the other side terminates. The sender of the data (eg. rcopy sending filename, server sending data) should try to resend packets 10 times after a 1second timeout. The receiver of the data (e.g. rcopy during the file data transfer) should use a 10 second timeout while waiting for data. This allows rcopy to terminate cleanly if it no longer can communicate with the server..

19) If the remote file on the server does not exist, the rcopy program should print out an error message (which includes the error string based on errno from the server) and terminate gracefully.

20) Regardless of problems with clients (e.g. ^c's), the main server process should not terminate. It should loop waiting to process other rcopy requests. The server should only terminate if it receives a ^c.

21) When sending file data, the only time your program should block (on select for 1 second) is when the window is closed (you have sent all the data you can.) Otherwise you should not block. Use a non-blocking select(…) (which means set your time value in your select call to 0 (zero)) to check for RR's when the window is open.

22) Regarding buffering packets for your window processing. You cannot buffer the entire file on either the server or rcopy. You can only maintain buffers approximately the size of your window. Your buffer must be a normal C array (it can be an array of structs but must be an array). You cannot use any data structures built into the language or supplied in 3rd party libraries. Your window management must be done by you. Also, you cannot move back and forward in the disk file instead of buffering. Any data held for window processing must be buffered in your array. Any code to process this buffer must be written by you.

23) You will not need to use the sliding window concepts for the filename exchange or after you send the last packet. Sliding windows only makes sense during the file data exchange.

24) When you are sending data and your window is open the flow should be: Send one data packet, non-blocking check for RRs/SREJ's and if RR's/SREJ's are available process all of them, goto send one data packet.

25) Your server must handle multiple clients at the same time via either **multithreaded (pthreads) or multiprocessing (fork).** (You will lose 25% if this does not work.)

26) Summary
    a. You can **only** resend lost/corrupted data. You cannot resend data that was received correctly unless you need to recover from a timeout (and then only 1 packet).
    b. You must send a SREJ on lost data (you cannot just wait until the other side times out. But you only need to send a SRJE once per lost packet – let your timeout recover from a lost SREJ.)
    c. If your window is closed and you timeout (so you have not heard about an entire window of data and have waited 1 second), send the lowest unacknowledged (so un-RRed) packet and wait for a response. The intent of this packet is to wake up (break the deadlock) the other side and hopefully open up your window. Continue this for 10 tries (wait 1 second, send lowest packet) or until you get an RR or SREJ.
    d. Do **not** set timers for every packet. That would be a major pain.
    e. Your blocking 1-second select(..) may only time out if:
        i. All RRs for a window are lost
        ii. or all data in a window is lost
        iii. or A SREJ packet is lost
        iv. You cannot timeout during the data transfer in any other situation

f. Server should process RRs/SREJs (without blocking) after every send(). In select() if you set the time value to 0 (zero) select() will check the socket for data but return immediately. See *man select*.

a. You can only call select() with a time value greater than 0 if the window is closed. If the window is open you can only call select() with a time value of 0 seconds, 0 microseconds.
b. On server, when the window is closed you should do a blocking select for a time of 1-second.
c. You cannot resend good data, if a packet arrives in the current window it is regarded as good data and should either be buffered or written to disk.

27) A run of these programs would look like:

On unix1:
> server .01
Server is using port 1234

On unix3:
> rcopy myfile1 myfile2 10 1000 .01 unix1 1234
Error opening local file: myfile1 – Error: Read-only filesystem[5].
>rcopy myfile3 myfile2 10 1000 .01 unix1 1234
Error during file open of myfile2 on server – Error: No such file or directory.
>rcopy myfile4 myfile5 10 1000 .01 unix1 1234
>

[5] You need to print out the real system error message, these are just examples. This requires the server to send back to rcopy the errno. See man errno and man strerror