CPE464, Fall 2011 - Programming Assignment #2
rcopy/server programs using stop and wait

These programs are due **at 11:59 pm Monday October 17, 2011**. You will use an electronic hand-in process to submit your code. If your program is late you will lose 10% per day. The last day you may work on this to receive any grade is Friday October 21, 2011. After that you will receive a grade of 0.

In this part of the assignment you are to write a remote file copy command using **UDP**. To do this you will implement two programs in C (or C++). The first is **rcopy** (the client program). This program will take two file names, a remote-file name and a local-file name. The second program is the **server**, which will accept a request for a file from the rcopy program and transmit the file back. The client will receive the packets that make up the file and write them into the local file.

In order to make the file transfer interesting, there are three additional requirements (a 4th requirement for extra credit):

1) Both your server and client programs should call the sendtoErr() function. This function should be used for all communications between the client and server. This function will cause the server to drop some of the data packets and flip some bits. The probability that the server will drop a packet/flip a bit is a run time argument (error-percent). More on how to get this function below.

2) In order for the file transfer to complete successfully you will need to implement a stop-and-wait flow control algorithm. In this approach every data packet is acknowledged by the rcopy program prior to the server sending another packet. In order to recover from lost data you will set a timer (via the selection function). See below for more details.

3) To detect bit flips you will use the Internet Checksum algorithm. If an error is detected appropriate recover measures must be taken.

4) 10% extra credit if your server handles multiple connections simultaneously using pthreads (threads) or fork() (multiple processes). It must be able to serve files to multiple clients at the same time.

Note: Program #3 may build on this program. The main difference is that it will use a sliding windows protocol and selective reject or go-back-n ARQ. I would recommend you write your program with this in mind… use (short) functions, indent your code, make your variable names meaningful, take time to design up front…

Details:

1) **rcopy**: This program is responsible for taking the file names as command line arguments and communicating with the server program to request the from-remote-file. This program will then receive the file and store it to disk using the to-local-file name. The program will be run as:

> rcopy from-remote-file to-local-file buffer-size error-percent remote-machine remote-port

from-remote-file:	is the file on the remote machine to be copied
to-local-file:	is the file copied **into** on the local machine
buffer-size:	is the number of data bytes (from the file) transmitted in a packet
error-percent	is the percent of packets that will have errors (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 for files from the rcopy program and transmitting the file back. This program should never terminate (unless you kill it with a ctrl-c). It should continually process requests from the rcopy program.

   The server needs to handle error conditions such as non-existing files by passing back a flag to the rcopy program and then waiting for a new file request.

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

   > server error-percent

error-percent	is the percent of packets that will be dropped (floating point number)

Requirements:

1) Do not use any code off the web or from other students. You may make use code that I have given you this quarter.
2) I have given you sample UDP code in lecture. I would recommend starting with that code.
3) The buffer-size parameter in rcopy specifies how many bytes to read from the file and transmit in a single packet. All data packets should contain buffer-size amount of data, except for the last packet, which may contain less bytes. The value of this parameter will be between 500 and 1400 bytes. No error checking is required on the parameter… assume it is between 500 and 1400.
4) The buffer-size parameter should be sent from rcopy to the server at the beginning of a file transfer. The server uses this parameter to determine how much data in the file to send in each packet.
5) The **data** packet being sent from the server to the client must have the following format: Sequence number, Internet Checksum, optional header if you wish (your call, not required, some students find adding their own header helpful) and then the file data.
6) There is no format requirement for filename exchange packets or ACK packets. Sequence numbers are not required for these packets (ACK and filename exchange). You can use sequence numbers but they are not required.
7) When you send a **data** packet (a packet that contains data from the file) the sequence number MUST be in network order, the sequence number should start at 0 and grow every time a NEW

packet is transmitted. Retransmitted data packets should have the same sequence number as the original packet.

8) Name your makefile: Makefile. Your makefile should provide a clean target that deletes all of the .o and executable files. I recommend you use/modify the Makefile provided (see below).

9) The command names rcopy and server should be used. Also the run-time parameters should be in the order given. Since I will be using a script to make and execute your code, all executable names and parameters must be as listed. If your program fails to run with my script you will lose 20%.

10) In place of bind use my bindMod() function and in place of select() use my selectMod() function. See discussion at the end of this document.

11) In order to prevent the server program from blocking forever the selectMod(…) function must be used (selectMod() is my modified version of select, see the discussion on selectMod() at the end of this document). This function allows the server to "probe" the socket for a specified period of time. If no data is found on the socket the server should retransmit the packet. For this program use a timeout period fixed at 1 second.

12) You **must** create (and use) a function with the following prototype:

    int select_call(int socket, int seconds, int useconds).

    This function calls selectMod(…) and returns a 1 if the socket is ready for read prior to the time expiring and 0 if no data is available on the socket. The function must be in a separate file and linked into both the rcopy and server programs.

13) If the server retransmits a packet 5 times (so you have 5 consecutive losses) the server can assume the client is not reachable and close out the client connection.

14) If the client does not hear from the server for over 10 seconds the client should terminate with an error message. You may see this occur with very high error rates (> 40%).

15) If the remote file does not exist the rcopy program should print out an error message and terminate gracefully. The server should not terminate.

16) To use the sendtoErr(…) function you must first initialize it by calling sendtoErr_init(…). If you use fork() or pthreads you should call this function in each child process/thread.

17) Your rcopy/server programs need to be able to transfer binary files. (Be careful when using string functions such as strlen, strcpy.)

18) I have provided an object library with four functions (selectMod(), bindMod(), in_cksum(), and sendtoErr()). Notes:

A. To install this library – two steps:

   a. Get the files (if you are on a linux box e.g. vogon, unix1…) type:[1]

      i. wget http://users.csc.calpoly.edu/~networks/cpe464/lib.mk

   b. Make the library, type:

      make -f lib.mk

B. The lib.mk creates an object library that includes the functions selectMod(), bindMod(), in_cksum(), and sendtoErr(). The library is the file libcpe464.1.2.a

C. The Makefile (called Makefile) that comes with the library is already written to use the object library when compiling the rcopy and server programs.

D. The file cpe464.h is the header file for all of the functions in the object library. You must include this in your code files (e.g. #include "cpe464.h")

E. sendtoErr() function – The object library contains the sendtoErr() function. This function should be used in place of the normal call to sendto(). This function drops packets and flips bits prior to sending the packet. See the cpe464.h file (which you copied over in the step above) for details on using this function. Also, remember to call the sendtoErr_init() function.

F. selectMod() function – Use the selectMod() function in place of your calls to select(). This function just replaces the select function and will help us in testing your program. The call to selectMod() is identical to the call to select(). In your new select_call() function use the selectMod() call instead of the normal select call.

G. bindMod() function – Use the bindMod() function in place of your call to bind(). This function just replaces the bind() function and will help us in testing your program. The call to bindMod() is identical to the call to bind().

H. in_cksum() function – This library includes the in_cksum() function which calculates the Internet checksum. This is the same function you used in the trace program.

19) A run of these programs would look like:

On vogon:	**In another window on vogon:**
> server .01	> rcopy myfile1 myfile2 1000 .01 vogon 1234
Server is using port 1234	Error myfile1 does not exist on vogon
	> rcopy myfile3 myfile2 1000 .01 vogon 1234

[1] If you are not on a linux box then go to the following link and download the page into a file called lib.mk:

http://users.csc.calpoly.edu/~networks/cpe464/lib.mk