COP 4635 SYS & NET II - PROJECT 4

OVERVIEW

This assignment requires you to implement the 1-bit Sliding Window Protocol *rdt 3.0* (Alternating Bit protocol) as described in class. You must use UDP message passing. Each process can have only one UDP socket. Base your program on the UDP sender and receiver you implemented in project 2.

This project requires three programs written in ANSI C. One program should model the sender, another should model the receiver described in *rdt3.0*, and a third program should act as a proxy simulating the network. All three programs should execute on different machines. The `main()` function of each program should be a driver program. This means that all variables should be declared in `main()` and passed to the appropriate functions. You may not have global variables! The sender is waiting for 3 events in *rdt3.0*: data to send, timer to go off, and acks from the network. For the sender to obtain data to send from the layer above (e.g., user), your program may simply wait on input from the keyboard. To handle the timer event and the arrival of an acknowledgement event, you must use the `select()` command. The proxy program acts as a middle-layer, transporting messages between sender and receiver, essentially simulating the network.

The time to send and receive a packet in our network is short. You will need to make use of the `sleep()` function to test whether your timer is working. Please read the manual page for `select()`. The manual page for `select()` has the necessary information for the project. I have uploaded a handout in *eLearning* for this week's unit with a more detailed description of `select()`.

Protocol rdt3.0 is robust in the presence of errors and lost messages. Sequence numbers and acks are required as specified by the protocol. The sequence numbers will either have a value of 0 or 1, and the only buffering required is of the last message sent. To demonstrate that your program is robust the receiver should deliver data to standard out. The data for the sender to send must be obtained by prompting the user for data from standard in. The data you typed in as input and the data delivered should be the same. The data must be sent in multiple segments to the recipient based on the segment's size. For this project, use a segment size of 10 bytes (header & payload). This means that if the complete message to be delivered is 50 bytes long, then you will have more than 5 segments that need to be sent to the recipient. In tests, we will have messages that will be longer than 10 bytes.

This project is divided into two parts A and B.

PART A: IMPLEMENTING RDT3.0

In this part, you must correctly implement the protocol in both the sender and receiver program. The `main()` in each program should be only a driver program meaning that variables should be declared local in main and passed to the appropriate functions as parameters. For this assignment, you may not have global variables!

The receiver and proxy should dynamically bind to a port and "announce" the port number on the screen. When the proxy is executed, the host name and port number of the receiver should be passed in as parameters to `main()`. Likewise, when the sender is executed, the host name and port number of the proxy should be passed in as parameters to `main()`. With this information, all three programs can setup a communication channel for sending messages back and forth. The figure below illustrates the communication channel involving a Sender,

Receiver, and a Proxy that serves as a message transporter between Sender and Receiver. Note that dynamic port binding is achieved with system call `bind()` and the bound port number is found with `getsockname()`.

PART B: SIMULATING LOST OR DELAYED MESSAGES

In our network, it is highly unlikely timeouts will occur or messages will be lost. Recall that a timeout occurs if a message is delayed or lost. Your system must simulate lost messages using the Proxy that delivers data to the Sender and Receiver. Random drop or delay of messages is easily accomplished by using random numbers to decide when to drop or delay a message. The Proxy's send function that is sending a packet to the receiver must use a random number to determine whether to call `sendto()` or not, to simulate a lost message on the network. The Proxy also simulates slow message transport within the network by delaying transport of a message to its destination using a fixed time delay. This time delay should be between 1.5 and 2 times the waiting time for the sender to resend messages. To accurately simulate slow messages, the process sending the slow message should be able to continue execution. This means that simply calling `sleep()` to delay sending a message will block the process preventing it to accept any messages or sending new messages. Instead, consider creating a new thread within the Proxy that will be tasked with sending the message with some delay using `sleep()`. You must simulate the probability of a lost or delayed message as some percentage such as 60%. The two separate parameters for specifying the delayed and lost message probabilities must be accepted as an input parameter in `main()` to the Proxy.

On our internal network, it is unlikely that your packets will be corrupt. You will need to simulate damaged data using a randomly chosen tag to indicate that your packets are either free of error or corrupt – no checksum computation is needed. Corrupt messages must also occur at a set percentage rate as above. Similarly, the percentage of corrupted messages must be accepted as an input parameter in `main()` to the Proxy, which handles the simulation of damaged data.

Your implementation of rdt3.0 should be as robust as possible in the presence of lost and damaged data. Remember both data and ack messages can be lost, delayed, or corrupted! You must simulate data and ack messages being lost delayed or corrupted.

For passing parameters for proxy and sender you must follow the following format:

```
sender [proxyHostname][proxyPort]
```

```
proxy [rcvHostname][rcvPort][lostPercent][delayedPercent][errorPercent]
```

Provide percentages for lost, delayed, and error packages as integers in the range of 0 to 100, not as a fraction of 0.0 to 1.0. For example, a value of 60 as percentage parameter would represent 60%.

IMPLEMENTATION SUGGESTIONS

For your implementation consider the following system calls:

socket()	**gethostbyname()**	**gethostname()**	**bind()**
sendto()	**recvfrom()**	**close()**	**select ()**

DELIVERABLES & EVALUATION

Submit your complete solution as a single zip file (only zip files will be accepted) containing A) source code, B) a single makefile to compile the three different programs, C) a program document, and D) a README file (if applicable) to the corresponding dropbox in *eLearning*.

The program document must describe briefly the respective algorithms of the sender, receiver, and proxy and the format of messages send between the three peers. The document must also describe the protocols used by the system to make it work. The README file should only be included if you submit a partial solution. In that case, the README file must describe the work you did complete.

You must follow the *Project Submission Instructions* posted in *eLearning* under Course Materials. The submission requirements are part of the grading for this assignment. If you do not follow the requirement, 5 points will be deducted from your project grade. Keep in mind that documentation of source code is an essential part of programming. If you do not include comments in your source code, points will be deducted. We also require you to refactor your code to make it more manageable and to avoid memory leaks. Points will be deducted if you don't refactor your code or if we encounter memory leaks in your program.

Your solution needs to compile and run in the computing environment provided on the CS department's Linux servers. Graders will upload your solution to a server and compile and test it running several undisclosed test cases. Therefore, to receive full credit for your work it is highly recommend that you test & evaluate your solution on the servers to make sure that graders will be able to run your programs and test them. You may use any of the 5 SSH servers available to you for programming and testing and evaluation. Use *ssh.cs.uwf.edu* to log into any of the 5 servers.

GRADING

This project is worth 100 points in total. A grade sheet posted in *eLearning* outlines the distribution of the points and grading criteria. Keep in mind that there will be a substantial deduction if your code does not compile.

DUE DATE

The project is due March 20th by 3:00 pm before the start of the class. Late submissions will not be accepted. See syllabus for details on late submission.