TOKEN RING BULLETIN BOARD

OVERVIEW

This assignment requires you to implement a simple bulletin board using UDP sockets for interprocess communication. The bulletin board lists messages that users post. The architecture of the bulletin board is based on a shared file that contains the list of messages. This file is accessible to hosts using the SSH server's Network File System for file sharing. However, access to the file is controlled by a token of a logical token ring that hosts running the bulletin board must join. Each host in the ring has access to the bulletin board only when it is in possession of the token. Otherwise, it must wait to receive the token from its sending neighbor in the token ring.

The access protocol is simple: when a host receives the token it can access the bulletin board to read messages or to append new messages to the shared file of the bulletin board. When the host has completed reading or modifying the shared file, it passes the token to its neighbor in the ring. If the host has nothing to read or write, it passes the token on immediately to the receiving neighbor in the token ring.

The user of the bulletin board should be prompted with a menu for reading and writing messages and to exit:

Action	Description
write	Appends a new message to the end of the message board.
read #	Read a particular message from the message board using a message sequence number. # is the sequence number of the message on the board.
list	Displays the range of valid sequence numbers of messages posted to the board.
exit	Closes the message board.

Exit requires that the user leaves the logical token ring.

MESSAGE FORMAT FOR THE BULLETIN BOARD

The shared file used by the bulletin board system contains a list of sequentially numbered messages starting at 1. Each message must be unique, i.e. the sequence number assigned to a message is different for each message. The message consists of a header, a footer, and a body. The header occupies the first line of the message, the footer the last line. Both follow a strict syntax described below. The message body is in free format and of any length except that the body cannot contain a valid header or footer message. You are not required to handle messages with any valid footer or header in it.

You may use variable or fixed-sized message bodies. The advantage of the fixed-sized message body is that your file operation can use seek operations to identify records in the file; otherwise, you will be required to scan from the start of the file to the end of the file to identify sequence numbers of messages and start and end of a single message.

	Format	Description
Header	<message n=*number*>	The XML tag on a single line to indicate the start of the message. Replace *number* with the actual number of the message.
Body	lines of text	multiple lines of text that may not start with the format of the header
Footer	</message>	The XML tag on a single line to indicate the end of the message.

An excerpt of sample messages stored in the bulletin board file is shown below:

<message n=1>
This is an interesting project.
I learned a great deal about interprocess communication and token rings.
</message>
<message n=2>
….

TOKEN PASSING IN RING

A host is only allowed to add to the bulletin board or read from it when it has the token to perform the write or read operation on the shared file. To avoid that a token is held by the host while the user enters a new bulletin board message on the keyboard, the host program needs to setup two separate threads. The main thread must handle token message passing among hosts. The bulletin board editing thread must handle user data entry. Note, only when the host has access to the token is it permitted to write a user-entered message to the bulletin board shared file or read from the shared file. The goal is for users to be able to use the message board freely without being hindered by the host waiting for the token. Users should be able to select a message they want to read or start writing a new message while the token continues to be received and passed through the peer.

ESTABLISHING AND EXITING FROM A RING

The ring is established at startup time but hosts may exit the token ring at any time. When hosts exit the ring the system must ensure no break in the flow of information within the token ring network. The ring is established by a separate server program that communicates with the hosts to inform them about their neighbors. Each host in the network has a neighbor from whom it receives a token and a neighbor to whom it sends the token. This means that for a token ring of N nodes there are N point-to-point communication paths forming the ring. We are assuming that messages travel in only one direction around the ring and token messages are not lost.

At startup, hosts contact the server to advertise their presence with their host address and port number for communication. When the server has received announcements from the expected number of hosts, it creates a logical ring and sends the hosts in the network information about their neighbors. The number of hosts in the token ring is specified as a startup parameter of the server. Once the ring is formed by the server, the server shuts down. One of the hosts in the ring network becomes the initiator and creates and sends the first token. You must decide how the hosts select their initiator in the ring. Note that the server is not involved in passing the token, administering the token, or deciding on the initiator.

To exit from the token ring, a host must advertise its desire to exit by sending a message with the token that lets the neighbors know about the host's intention to exit. The neighboring nodes will then bypass the host the next

time the token is passed through the network. It is up to you to decide how to handle border-line cases such as when two hosts desire to exit the ring simultaneously or when after an exit the ring consists of a single host only. Just be sure that those cases are handled appropriately so that the token continues to be passed in the ring.

THE PROGRAM

The bulletin board system consists of two programs: a `bbpeer` and a `bbserver`. The `bbpeer` program runs the bulletin board on a host machine that provides users with access to the bulletin board. The `bbserver` establishes the ring. Both programs use UDP datagrams for communication. The ring must consist of at least three hosts. You may assume that communication is reliable, i.e. tokens and announcements to the server are not lost.

`bbserver` *`numberHosts`*

The bulletin board server program reads the number of hosts (*numberHosts* must be replaced by an actual number) in the ring from the parameter in `main`. The server then waits for messages from the hosts that form the ring. When the server has received messages from all hosts, it creates the ring and informs each host about its two neighbors by sending port and host information of the neighbors. After the server creates the ring, it terminates.

`bbpeer` *`filenameBulletinBoard serverIP serverPort`*

The bulletin board peer program starts by dynamically binding a UDP socket to a port. The program then sends the server its host address and port number and then waits for the server's response. When the server provides the needed neighboring information, the program then undergoes an initiating process to select the host that generates the first token. You must choose an appropriate initiation process. Following the completion of the initiation process, the program displays the complete menu for the message board.

The peer program is started with the file name (*filenameBulletinBoard* is the name of the bulletin board file such as "myBulletinBoard.txt") of the shared bulletin board file and the host (*serverIP*) and port (*serverPort*) information of the server. The peer program is terminated by the user entering *exit* at the menu.

IMPLEMENTATION SUGGESTIONS

I suggest reading back through the lecture notes for details on ring networks discussed at the start of the semester. You should also consider examining the reading material and the sample program posted on *eLearning* as well as the textbook that discusses UDP programming. Read these pages thoroughly before starting this project. For your implementation you need to use the following system calls:

socket()	**gethostbyname()**	**gethostname()**	**bind()**
sendto()	**recvfrom()**	**close()**	**fopen()**
fflush()	**fclose()**	**fseek()**	**fprintf()**
fscanf()			

The last 6 system calls are needed to read and write from the shared bulletin board file.

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 client and server programs, C) a protocol document, and D) a README file (if applicable) to the corresponding dropbox in *eLearning*.

The protocol document must describe the protocols used by the system to make it work. At the minimum the document must describe the message exchange between `bbpeer` and `bbserver`, how the system undergoes the initiator process to select the host that creates the first token and how `bbpeer`'s exit from the ring without breaking communication. 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. I 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 (SSH1 through SSH5). I 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 I 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 1st by 11:00 pm. Late submissions will not be accepted and I will not accept any emailed solutions. The syllabus contains details in regards to late submissions. Groups that submit this project by February 22nd at 11:00 pm will receive 6 bonus points. However, any group that chooses to submit their solutions before the earlier deadline cannot resubmit and updated version at a later time.