TOKEN RING BULLETIN BOARD

OVERVIEW

This assignment requires you to implement a simple bulletin board using UDP sockets for inter-process 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. A list of actions and their description is given below.

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 number 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 body 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 as well as the start and end marker of messages.

	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 or footer
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 inter-process communication and token rings.
</message>
<message n=2>
….

TOKEN PASSING IN RING

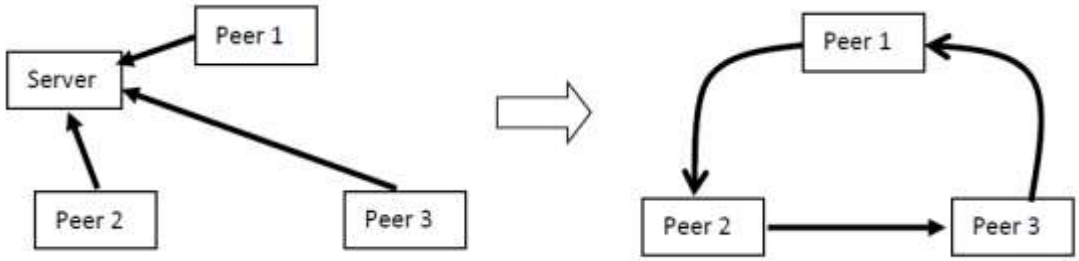
A host is only allowed to add to the bulletin board or read from it when it has the token to perform the write or read operation on the shared file. To avoid that a token is held by the host while the user enters a new bulletin board message on the keyboard, the host program needs to setup two separate 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, JOINING, AND EXITING FROM A RING

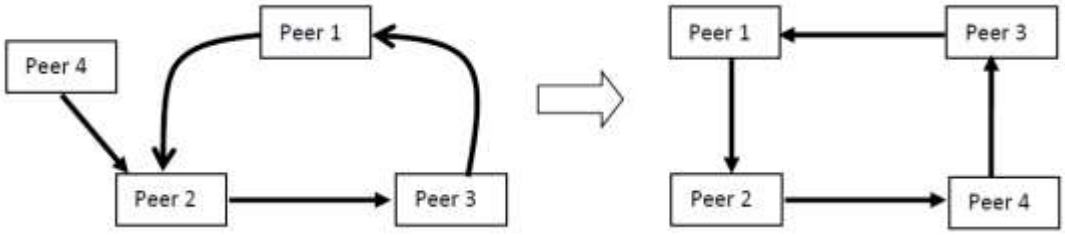
The ring is established at startup time but hosts may exit the token ring at any time. When hosts exit the ring the system must ensure no break in the flow of information within the token ring network. The ring is established by a separate server program that communicates with the hosts to inform them 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. In the ring, messages travel in only one direction around the ring and although tokens may get lost in physical token rings you may assume that the token messages for this project will not get 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 among themselves their initiator in the ring. Keep in mind that messages in the ring, whether

they are tokens or not, can only travel in one direction. Note that the server is not involved in passing the token, administering the token, or deciding on the initiator. The figure below illustrates the process of peers contacting the server to create a new ring.



To join an existing token ring, a host must send a join request message to one of the peers in the ring, to request to be inserted into the ring. The peer that receives a message must be able to recognize the request and distinguish the message from the new peer with a token message that it receives from a neighboring peer. Upon receiving the request to join, the peer must then be inserted into the ring. The new peer must be told about its next neighbor in the network, i.e. the neighbor to which the host needs to send the token. Should the peer that receives the join request message also receive a token while the new peer has not yet been inserted, it must hold the token and send it on after the new peer has joined. Note that as the new peer is inserted, it must comply with the token-ring policy that messages cannot be send in both directions of the ring. The figure below illustrations the insertion process. Peers can only join a ring that has been fully established, i.e. peers pass tokens in the ring.



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 requests to join are not lost.

bbserver portNum numberHosts

The bulletin board server program reads the port number (portNum) and number of hosts (numberHosts) in the ring from the argument list in main. The server binds a datagram socket to the specified port and then waits for messages from the hosts that form the logical ring. When the server has received messages from all participating

hosts, it creates the ring and informs each host about its next neighbor (the neighbor to which the host needs to send the token) by sending port and IP address of the host's neighbor. After the server creates the ring, it terminates.

bbpeer [-new] portNum hostIP hostPort filenameBulletinBoard

The bulletin board peer program starts by parsing the argument list in main extracting information about the peer's port number, the IP address and port number of the server or a peer in an existing token ring, and the name of the shared bulletin board file. Next, the peer binds a UDP socket to the specified port number. If the peer is joining a new ring (the user runs the peer with option –new), the program sends the server (bbserver) a join request to join the ring. The server and peer follow the protocol for establishing a new ring as described above. If the peer is joining an existing ring (the user does not provide –new on the command line), the host and port number specify a peer's IP address and port number in the existing ring. In this case, the new peer sends a join request to the existing peer to join the ring following the protocol described above. The peer uses the specified file name (filenameBulletinBoard) to read and write to the shared bulletin board file. 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 provided sample program on UDP programming. Read these pages thoroughly before starting this project. For your implementation you need to use the following system calls to create a socket, bind it, send and receive messages and perform file IO for reading and writing to the bulletin board file:

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

DELIVERABLES & EVALUATION

Your project submission should follow the instructions below. Any submissions that do not follow the stated requirements will not be graded.

1. Follow the submission requirements of the instructor as published on eLearning under the Content area.

2. You should submit all the following files for this assignment:

   a. source code files for bbpeer and bbserver, and supporting code files if any
   b. a single Makefile to compile the client and server programs,
   c. a protocol document
   d. a README file containing names of all project members, directions on compiling and running your files indicating usage and examples, and any other relevant details like an analysis if project wasn't completed
   e. Screen captures of your running program showing all windows involved, in two or three stages of communication.

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.

Your program will be evaluated according to the steps shown below. Notice that the instructor/GA will not fix your syntax errors. However, they will make grading quick!

1. Program compilation with Makefile. The options –g and –Wall must be enabled in the Makefile. See the sample Makefile that I uploaded in eLearning.
   - If errors occur during compilation, the instructor/GA will not fix your code to get it to compile. The project will be given zero points.
   - If warnings occur during compilation, there will be a deduction. The instructor/GA will test your code, though.
2. Program documentation and code structure.
   - The source code must be properly documented and the code must be structured to enhance readability of the code.
   - Each source code file must include a header that describes the purpose of the source code file, the name of the programmer(s), the date when the code was written, and the course for which the code was developed.
3. Perform several test-runs with input of the grader's own choosing. At a minimum, the test runs address the following questions.
   - Will the ring be established?
   - Will the token traverse through the ring and control access to the shared file?
   - Will the user be able to use the message board to write and read messages?
   - Will peers be able to exit the ring without breaking it?
   - Will new peers be able to join the ring without breaking it?
   - Will the ring keep working when only a single peer is left in the ring?

Keep in mind that documentation of source code is an essential part of computer programming. The better your code is documented, the better it can be maintained and reused. If you do not include comments in your source code, points will be deducted. You should 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 during testing.

DUE DATE

The project is due as indicated by the Dropbox for Project 1 in eLearning. Upload your complete solution to the dropbox and the shared drive. I will not accept submissions emailed to me or the GA. Upload ahead of time, as last-minute uploads may fail.

TESTING

Your solution needs to compile and run on the CS department's six SSH servers. I will compile and test your programs running several undisclosed test cases involving a minimum of four of the six SSH servers to run three peers and a

server that establishes the token ring. Therefore, to receive full credit for your work it is highly recommended that you test & evaluate your solution on the servers to make sure that I will be able to run your programs and test them successfully. You may use any of the five SSH servers (cs-ssh1.cs.uwf.edu, ..., cs-ssh6.cs.uwf.edu) available to you for programming, testing, and evaluation. For security reasons, most of the ports on the servers have been blocked from communication. Ports that are open for communication between the public servers' range in values from 60,000 to 60,099.

GRADING

This project is worth 100 points total. The rubric used for grading is included below. Keep in mind that there will be deductions if your code has memory leaks, crashes, or is otherwise, poorly documented or organized and there will be zero points for code that does not compile. The points will be given based on the following criteria:

Submission	Perfect	Deficient		
eLearning	5 points individual files have been uploaded	0 points files are missing		
shared drive	5 points individual files have been uploaded	0 points files are missing		
Compilation	**Perfect**	**Good**	**Attempted**	**Deficient**
Makefile	5 points make file works; includes clean rule	3 points missing clean rule	2 points missing rules; does not compile project	0 points make file is missing
compilation	10 points no errors, no warnings	7 points some warnings	3 points many warnings	0 points errors
Documentation & Program Structure	**Perfect**	**Good**	**Attempted**	**Deficient**
documentation & program structure	5 points follows documentation and code structure guidelines	3 points follows mostly documentation and code structure guidelines; minor deviations	2 points some documentation and/or code structure lacks consistency	0 points missing or insufficient documentation and/or code structure is poor; review sample code and guidelines
Server	**Perfect**	**Good**	**Attempted**	**Deficient**
creates a ring and sends neighboring information to the peers	10 points correct, completed	7 points correct, completed	3 points incomplete	0 points missing or does not compile
Peer	**Perfect**	**Good**	**Attempted**	**Deficient**
connects to server to establish ring	10 points correct, completed	7 points minor errors	3 points incomplete	0 points missing or does not compile
performs initiation process to select peer generating the first token	10 points correct, completed	7 points minor errors	3 points incomplete	0 points missing or does not compile
accesses message board correctly to read and write	10 points correct, completed	7 points minor errors	3 points incomplete	0 points missing or does

messages				not compile
uses multiple threads to handle message editing and token management separately	10 points correct, completed	7 points minor errors	3 points incomplete	0 points missing or does not compile
can exit from the ring at any time without breaking the ring	5 points correct, completed	4 points minor errors	1 points incomplete	0 points missing or does not compile
ring works even if a single peer is left in the ring	5 points correct, completed	4 points minor errors	1 point incomplete	0 points missing or does not compile
Protocol Document	**Perfect**	**Good**	**Attempted**	**Deficient**
has initiation, join, and exit procedure	5 points fully described	4 points mostly described	1 point critical information is missing	0 point incomplete or missing
has format of messages exchanged between peers and peers and the server	5 points fully described	4 points mostly described	1 point critical information is missing	0 point incomplete or missing

Not shown above are deductions for memory leaks and run-time issues. Up to 15 points will be deducted if your program has memory leaks or run-time issues such as crashes or endless loops when tested on the SSH server.