Jacob Rook

COP 4635

Project 2

April 25, 2018

# Token Ring Bulletin Board Program Protocol

**Summary:**

The Token Ring Bulletin Board Program is a peer to peer network that requires a server to link the peers into a ring. Once the peers are placed in the ring, the server terminates and does not assist the peers in the ring further.

The peers pass a token to be able to access a local file to read from and write to. Whenever a peer has the token, they are allowed to write to the local bulletin board file, read specific messages from the local bulletin board file, list all of the messages from the local bulletin board file, or exit the ring. Note, the peers are only allowed to exit when they have the token.

After the ring of peers has been initiated, new peers can join the ring by contacting one of the peers in the ring and the joining peer will receive the address of their next peer in the ring from one of the existing peers. It is also worth noting that message passing only go in one direction.

**Server Start-up:**

Obviously, the server will need to start up before the peers try to contact the server. The server program takes two command line arguments, the server's port number and the number of peers to accept before sending off the peer's next peer socket address. The command should look like:

./bbserver <portNum> <numberHosts>

The server will bind a socket with the given port address then wait for peers to send the server a message to join a ring.

The peers will continue to contact the server until the number of peers the server needs to start the ring contact the server. Once the specified number of peers have contacted the server, the server stops accepting joining peers and sends the peers the socket address of their next peer in the ring.

**Peer Start-up:**

This implementation of the Token Ring Bulletin Board program only allows process to process communication, meaning that the server and all of the peers need to be on the same host. Since the server and peers will be on the same host, the IP addresses of the hosts and peers are

not needed for communication. Also, the peers can share the same local bulletin board file. The general command to start a peer is:

./bbpeer [-new] <PortNum> <HostPort> <FilenameBulletinBoard>

The PortNum command line argument is the integer port number of the server or peer of an existing ring, the HostPort command line argument is the integer port number the host would like to have bound to the host's socket, and the FilenameBulletinBoard command line argument is the name of the bulletin board file that will be used to read and write to.

Peers can either contact a server that is starting a new ring or a peer of an existing ring. Either way, the message that they receive will be the same, i.e. the socket address of their next peer in the ring. The –new option distinguishes between a peer wanting to contact a server and a peer of an existing ring.

If a peer wants to contact a server, the user includes the –new option. If a peer wants to contact a peer of an existing ring, the user does not include the –new option.

**Peer-to-Server Ring Set-up:**

After the server has been started, the peers that want to start a ring will start to send messages to the server. The messages that the peers send to the server do not exactly matter because the server just creates an array of socket addresses from the socket addresses of the peer that sent the message. However, the server will print the message to the stdout.

Whenever the number of hosts specified by the user has sent the server a message, the server will send a struct sockaddr_in variable with that peer's next peer in the ring. The receiving peer will read the server's message into their own struct sockaddr_in variable to use for themselves.

The order of the ring is determined by the order in which the peers send messages to the server. The first peer to contact the server will be sent the socket address of the second peer to contact the server, the second peer to contact the server will be sent the socket address of the third peer to contact the server, so on and so forth up until the second to last peer to join. The last peer to contact the server will be sent the socket address of the first peer to contact the server to complete the ring.

After the last socket address is sent, the server shuts down, not aiding the peers anymore.

**Peer-to-Peer Communication:**

Peers communicate with each other by sending predefined message structures, namely struct message_t. The message_t structure and message_Header_t structure are:

```
struct message_t{

	struct message_Header_t header;
	char messageBody[MESSAGE_SIZE + 1];

};
```

```
struct message_Header_t{
	int token;
	int action;
};
```

The messageBody[] variable is used to store any data that the peers want to send to one another and the header is used to identify what the message is about. The header has two integer variables: token and action.

The token variable identifies the state of the token. The token only has three states: initiation, passing, or not passing. All three states have an integer value associated with the state for the peers to be able to identify which state is in the message.

The initiation state of the token is for when the peers are trying to determine which peer will create the first token. The passing state of the token is for when the previous peer in the ring is passing the token to the peer receiving the message. And, the not passing state of the token is for when the previous peer needs to send a message with an action.

There are two rules for message passing: tokens cannot be passed with actions and messages received with invalid header values are discarded.

The action variable of the header is used to determine which action the sending peer wants the receiving peer to do. There are three valid actions a peer can send another peer: join request, exit notification, and no action. Like the token, these actions have integer values associated with them for the peers to be able to identify what the action is.

A join request is for when a peer wants to join an existing ring, an exit notification is for when one of the peers in the ring wants to exit the ring, and the no action is for when a peer is passing a token.

**Ring Initiation:**

After the peers receive their next peer to send messages to in the ring from the server, the peers enter into an initiation phase to determine which peer will create the first token and the bulletin board file. The initiator is the peer with the lowest port number. To find which peer has the lowest port number, all of the peers send a message to their next peer in the ring with the token variable set to the initiation state and the messageBody including the host's port number.

When the peers receive the message with the token set to the initiation state, the peer checks to see if the port number in the message is smaller or equal to the host's port number. If the port number in the message is smaller, the peer forwards the message, knowing that it is not the peer and waits to be notified that the initiator has been found. If the port number in the message is equal to the host's port number, then the peer knows that it has to be the peer with the smallest port number, therefore it has to be the imitator. If the port number in the message is larger than the host's port number, then the peer ignores the message because the peer with the port number in the message cannot be the initiator.

Once the initiator determines that it is the initiator, it will create the bulletin board file and send its next peer the token to notify the peer that the initiator was found. The peer receiving

the token will know that the initiator was found, and will pass on a message notifying the next peer that the initiator was found. Each peer will keep notifying the next until it reaches the initiator at which time the initiation process stops.

**Peer Joining Existing Ring:**

To the joining peer, the process of the peer joining an existing ring is very similar to a peer joining a new ring. The only real difference is the user is not asked for a message because the joining peer must send a message structure to the peer in the ring so that peer can know what the peer wants. The joining peer sends a message structure with the joining action and the messageBody including the integer value of the joining peer's port number. Then, the joining peer waits for a response with its next peer's socket address.

**Peer Exiting Ring:**

A peer wanting to exit a ring must wait for it to receive the token then send out an exit notification with its own port number and the port number of its next peer in the ring in the messageBody. The peer waiting for the token to exit relieves the problem of two peers wanting to exit at the same time. Only one peer can exit at a given time if the peer has to wait for the token.

The exiting peer knows that it is safe to exit the ring when it receives the same exit notification that it originally sent out. However, until the exiting peer receives its original message, the exiting peer must still receive and respond to the other messages that it gets.

The only messages of concern that the exiting peer should be able to receive are join requests and exit notifications, the only other valid messages would be no actions, a passing token, or a token initiation all of which need to be discarded because the sending peer must have made an error.

If the exiting peer receives a join request from a peer, the exiting peer will forward the join request to its next peer to handle. If the exiting peer receives an exit notification, the exiting peer compares the exit notification with the one that it sent. If the exit notification is the same the peer exits the ring and terminates, otherwise, the exiting peer ignores the exit notification because one of the other peers must have made a mistake somewhere.

When a peer that is not exiting receives an exit notification, the peer checks to see if the exiting peer's next peer port number is equal to the host's next peer port number.

If the port numbers are the same, the host knows that its next peer is the one that is exiting and sets its new next peer socket address to the exiting peer's next peer to skip over the exiting peer then forwards the exit notification to the exiting peer to let that peer know that it is safe to exit.

If the port numbers are not the same, the peer just forwards the exit notification to its next peer.