CS 505 Programming Assignment 1

Prof. Ananth Grama

Due: February 18, 2011. 11:59pm

1 Task A: Implementing a synchronous bi-directional ring

In this task, you will implement a synchronous bi-directional ring, i.e. you will implement a communication substrate based on the ring that applications can use to send messages to each other, i.e. you have to write a **class** Ring with the following API: (Your code should throw a RingException when any of the operations in RingSubstrate fails – set the cause field appropriately.)

```
class RingSubstrate {
  public RingSubstrate(RingApp rApp);
  public Set<String> joinRing(String hostName) throws RingException;
  public void joinRing(String hostName1, String hostName2) throws RingException;
  public void leaveRing() throws RingException;
  public List<String> getHosts() throws RingException;
  public void sendAppMessageClockwise(String message) throws RingException;
  public void sendAppMessageCounterClockwise(String message) throws RingException;
  public void sendAppMessage(String message, String hostName) throws RingException;
}
class RingException extends Exception {
  public RingException(String cause) {
    super(cause);
    ...
  }
}
class RingApp {
  public boolean deliver(String message);
  ...
}
```

You can assume that there is exactly one application running on a physical host, and the application is "identified" by the fully qualified hostname e.g. xinu10.cs.purdue.edu. You can also assume that no more than one RingApp object uses a RingSubstrate object. In this task, you can assume that processes (application+substrate) do not fail. You are free to choose port numbers that your code uses for various messages. In the following, the current host shall be denoted by currHost.

1. **void** joinRing(String hostName1, String hostName2): The substrate shall communicate with its peers at hostName1 and hostName2 to insert currHost into the ring between hostName1 and hostName2.

2. Set<String> joinRing(String hostName): When this method is called at the substrate on currHost, it communicates with its peer at hostName, which returns to currHost the hostnames of 2 peers on the ring, willing to accept currHost as a neighbor. You are free to decide how the substrate at hostName chooses these 2 peers, but hostName must get their "permission" by communicating with them. These peers are returned by this method in a Set.

3. **void** leaveRing(): Gracefully disconnect currHost from the ring. The substrate shall notify the neighbors that currHost wants to leave the ring and wait for their acknowledgements before returning from this method.

4. List<String> getHosts() returns the *ordered* list of hosts in the ring. The ordered list returned can start in either direction.

5. **void** sendAppMessageClockwise(String message) sends a message into the ring in the "clockwise" direction. You can assume that the application RingApp implements a **boolean** deliver(String message) method, which RingSubstrate should call to "deliver" a message. This is why the constructor for RingSubstrate takes a RingApp object as an argument. You're free to decide which direction is "clockwise".

6. **void** sendAppMessageCounterClockwise(String message) sends a message into the ring in the "counter clockwise" direction.

7. **void** sendAppMessage(String message, String hostName) sends a message into the ring destined for `hostName`. The substrate shall deliver the message *only* to hostName.

2 Task B: Hirschberg-Sinclair leader election algorithm (HS algorithm)

In this task, you will implement the HS algorithm for leader election in bidirectional rings. You can assume that each node is uniquely identified by its fully qualified host name e.g. sslab02.cs.purdue.edu. You may use lexicographic ordering for comparing identifiers of nodes, i.e. java.lang.String.compareTo(...). You will add the following method to class RingSubstrate:

public String getLeader()**throws** RingException

The purpose of the leader, in this task, would be to maintain a consistent view of the ring. It is the responsibility of the (substrate at the) leader to periodically broadcast an *ordered* list of members in the ring (the list starts with the leader).

In this task, processes, i.e. (application+substrate) at hosts can fail (assume *crash-stop* failures). When a process fails, it will be the responsibility of the (substrate at the) leader to detect this failure, and reconstruct the ring. Test your code by causing your java program at one of the hosts to fail (e.g.,login to that host and type 'Ctrl-C' or 'kill -9 ...'). When a non-leader process in the ring fails, check to see if the leader reorganizes the ring by sending a message and/or calling getHosts(...) on all other hosts. *For simplicity, you can assume that once a process is elected leader, it doesn't fail.* You can use heartbeats for failure detection.

RingSubstrate should re-run the leader election algorithm when the current leader leaves the ring by calling disconnect(). Consequently, you have to modify the behavior of disconnect() in this task – when disconnect is called on currHost all other nodes in the ring are notified, and acknowledgements are awaited from all of them.

3 Submissions

Put all the files in your submission into a directory, and name it after (one of) the team members, i.e., if your name is John Smith, the directory name should be JohnSmith. Create two sub-directories, TaskA and TaskB. Please submit a README containing the names and emails of the both the team members. Please also submit a brief writeup of your design, either as a plaintext or as a pdf file. You have to use turnin to submit this assignment: turnin −c cs505 −p Programming1 <directory containing your files>

4 Resources

You can use any set of CS Linux machines, e.g., {sslab00−sslab24, xinu00−xinu21,mc01−mc16}@cs.purdue.edu. Try to test your code with rings containing atleast 7-8 hosts. You can implement the ring using Java's TCP

Sockets, but the use of higher level messaging APIs like Netty is prohibited.

5 Helpful Tips (i.e. you're not required to do this)

To deploy a distributed system, e.g. a ring on 10 hosts, a cumbersome way is to open 10 terminal windows, ssh into 10 hosts, and startup the program. This is unnecessary, because in the machines you'll be using, the filesystem is the same. A better way is to write a shell script which does this for you. Your script can take a list of hosts from a file, ssh into the hosts and start the program for you. The output can be stored in log files, automatically generated e.g. using HOSTNAME shell variable. For this, you also have to figure out how to ssh into a system without a password, see e.g., `http://bit.ly/5OSXoo`.

Other tools, which we have not used are parallel-ssh (`http://code.google.com/p/parallel-ssh/`) and gexec (`http://www.theether.org/gexec/`).