Homework 6: Readme.txt

Leader Election

Introduction: The following Java program implements the Leader Election Algorithm for . This algorithm is used to elect a leader among multiple processors in a system.

Leader Election Algorithm:

Each processor in our system will send a message with its identifier to its left neighbor and then wait for messages from its right neighbor. When it receives a message it checks the identifier in the message, if the identifier is greater than its own identifier, it forwards the message to the left; else it “swallows” the message and does not forward it. If a processor receives a message with its own identifier, it declares itself a leader and sends out a termination message to the left and terminates.

Files included in Project:

Buffer.java

Message.java

Processor.java

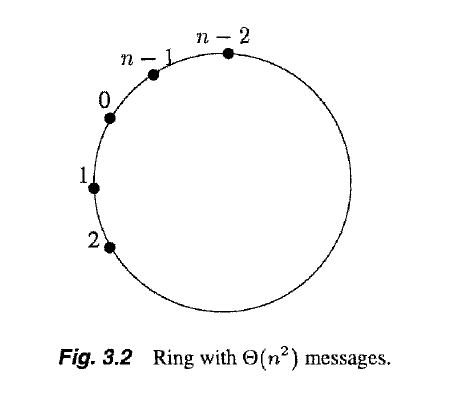
Main.java

Project Design:

Starting with the skeleton code given by Professor Tanjua for the DFS search we modified it so that we could implement the leader election Algorithm. We started with 5 processors in a system and placed them in a linked list to represent the connection and relationship between the processors. Each processor has an attribute of current Leader. Every processor will be observing the messages coming into its buffer and update will react accordingly. The comparison part in our update method determines what the processor will do next (either send your own id to the next processor or send the new message id it has received). The program will terminate when a termination message is sent. This occurs when a processor receives a message with its own identifier.

Time Analysis:

This algorithm never sends more than messages in any valid execution. The messages to determine the leader will be sent at most n times. The termination message will need to also be sent n times leading to the time of.



Tester Cases:

We tested our code by having each processor in the system have different id’s so that the leader would be different each time.