Caitlin Ross and Erika Mackin
Distributed Systems
Assignment 2
12/6/15

Distributed Calendar Application
Implementing Paxos

Section 0 - Code contents:

Application consists of 5 pieces of code:
Driver.java: This sets up the node object, creates a thread for listening for messages from other nodes, and provides the text-based UI.
Node.java: This object contains all of the information a node needs for keeping track of the calendar. This is the class where the Paxos algorithm and leader election are implemented.
LogEntry.java: Provides an object for each log entry—a full calendar and set of appointments. Every entry in the log is thus a complete schedule.
Appointment.java: This class tracks the information related to an appointment
Day.java: This is the enum class for providing day of the week information.
MessageType.java: This is the enum class for all the different types of messages used in Paxos and leader election.

To run from the command line, add the following arguments when starting the program (assuming use of the included jar file):
java –jar paxos-ross-mackin.jar <hostnames.txt> <nodeId> <numberNodes> <recover?>

hostnames.txt should be a list of the IP addresses for each node, one on each line, listed in order of node ID number (i.e. node 0's IP should be on the first line, and so on)
nodeID is the id number for the node being set up
numberNodes is the total number of nodes
recover determines whether the node should try to recover its state. 0 indicates no recovery, 1 allows for recovery from the nodestate.txt file.

Section 1 - Appointment Input

User input is done through the terminal by a series of text-input prompts. Users will be asked whether they wish to add or delete appointments, or to print the calendar.
If they wish to add an appointment, they will be asked to enter an appointment name , the day (either by spelling out the day, or entering the commonly-used three- or four-letter abbreviations), start and end times (in HHMM format) plus AM or PM, as well as the other participants' node numbers.
If they wish to delete an appointment, all the user need enter is the appointment ID. When selecting delete, the current appointments are listed with their ID numbers so the user can make a selection.

If a user tries to schedule an appointment that conflicts with their own most recent log entry, they will be notified immediately. If the appointment does not conflict with their most recent log entry, it will be sent on to the distinguished proposer. If it conflicts with the actual most recent log entry, the user will again be notified that the appointment could not be scheduled.
Due to issues with UDP messages (discussed in Known Issues below), we added a Dummy message option that can be invoked from the command line. This simply sends TCP and UDP messages back and forth to establish that messages are working.
Updates can also be selected from the command line. This allows a newly elected leader to use Paxos to ensure its log entries are up to date, in case the UDP issue discussed below causes it not to update when it's first elected.

Section 2 - Calendar makeup

The calendar is maintained by:
A) a three-dimensional array (#nodes x 7 x 48), of a 7-day period, broken up into half-hour sections. So it is a calendar of every node's appointment times.
B) set of current appointments. Each appointment object consists of the information from the user, plus an appointment ID and appointment number.
Every entry in the log is a new, complete calendar.

Section 3 – Appointment Creation

When a process is creating a new appointment, it checks that it is free to schedule based on its most recent entry in the log. If it does not conflict, it sends its list of appointments to the distinguished proposer. The proposer then compares the received set of appointments against the set of appointments in the leader's most recent log entry. If there is a conflict, the proposer notifies that user and sends back the most recent log entry as well so the user has more up-to-date information. If there is no conflict, the proposer starts Paxos on it.

Section 4 - Appointment Deletion

Given appointment ID of desired appointment to delete, the process will attempt to find the appointment in its list of current appointments. If the appointment is found, the appointment is removed from the current appointment list. This new list of appointments is again sent to the leader, who will perform Paxos on that new calendar.

Section 5 – Leader Election

If a node tries to send a proposed calendar to the distinguished proposer and finds that the proposer is down (which it can easily detect since TCP is used for this portion of the program), it will initiate leader election. The bully algorithm is used here. Once a new leader is chosen and all other live nodes have been notified, the leader runs the prepare promise portion of Paxos to fill in any gaps in its log, before accepting new calendars/log entries to run Paxos on.

Section 6 - Failure and Recovery

To preserve the calendar in the event of a failure, each time a node receives any

information about the log the list of appointments from each entry in the log is written to a text file. Any empty entries in the log will also be included.
To recover a crashed node, the 4th command line argument should be set to 1. This will cause to program to read from the nodestate.txt file to restore the state of the node before the crash. The process will regain all information re: its log and the highest proposal value it has ever used (to ensure it doesn't reuse any numbers).

Section 7 –Known Issues

We encountered a weird issue with UDP messages. There's no issues with TCP messages, but sometimes UDP appears to lose messages in patterns (as opposed to random packets being dropped). All traffic is allowed on our instances, so we're not sure why this is happening. Essentially when nodes are first started, they might send packets, but no packets ever make it to the receiving nodes. After sending enough packets, the situation appears to get better and looks more like what we expect (where many packets are received, but some get dropped). This is why we added the dummy message option, to have a quick way to send messages around. However, dummy UDP messages seem to always work, which is weird, because the same function does the sending for dummy messages as well as real messages.