Caitlin Ross and Erika Mackin

Distributed Systems

Assignment 1

10/25/15

Distributed Calendar Application:

Implementing Wuu and Bernstein Algorithm

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 Wuu and Bernstein algorithm is implemented.

EventRecord.java: This class provides an object for each event record, as events are defined in the Wuu and Bernstein paper.

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.

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.

Before being prompted for the next action, the user will be notified of any appointments that were unable to be scheduled (because of conflicts with previously scheduled appointments on other nodes), so the user can reschedule if they wish.

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. Each process tracks it’s own appointments, and the appointments of the other processes that it currently knows of (though a processes view of the other calendars might be outdated). Each half-hour period is set to 0 if the time is available and 1 if there is an appointment scheduled for that time.

B) set of current appointments. Each appointment object consists of the information from the user, plus an appointment ID and appointment number.

Section 3 – Appointment Creation

When a process is creating a new appointment, it checks that it is free to schedule based on its views of all process calendars. If so, it schedules it by calling insert(), then send() for sending to other processes if necessary. Send(), insert(), and delete() follow the Wuu and Bernstein algorithm.

Section 4 – Receiving Appointments

When a process receives a set of appointments from another node (the set NP sent from each node according to the algorithm), the process first checks own calendar for conflicts with the new appointment. If no conflicts, creates and stores the new appointment object in the dictionary currentAppts, the timeslots in the calendar array are set to 1, and the node peforms the rest of the Wuu and Bernstein algorithm for receive().

If a conflict with a preexisting appointment is detected, the process will send a message back to the creator of the appointment notifying it that a cancellation of that appointment is required. The creator of the appointment will then notify any other processes involved (besides itself and the process that determined the conflict).

Section 5 - 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 by performing a delete, as described in Wuu and Bernstein. Then the time slots taken up by the old appointment are set back to 0 in the calendar array. If the old appointment involved multiple processes, a send(appt, delete) is performed for the relevant processes.

Section 6 - Failure and Recovery

To preserve the calendar in the event of a failure, the state of each node is saved and new event records are written to a text file on the disc after every insert, delete, send, or receive action. This textfile is completely written over each time. It keeps track of basic node information and the events/appointments in each of the sets (e.g. NP, PL, etc).

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: clock information, its calendar array, all current appointments, its log of event records, and the current times for which its info about other processes’ schedules is up to date.