COMP 360 — Programming Task 2 Extra Credit

Jeff Epstein

Due 20 Oct 2016

1 Introduction

This document offers you the opportunity to implement additional functionality, above and beyond the requirements of the programming task.

Extra credit will not be considered unless all mandatory features are implemented satisfactorily. The value of the extra credit depends the completeness and elegance of your solution. The maximum extra credit per assignment is 10 points.

You may complete any number of the following tasks. Credit will not be given for incomplete tasks. In order to claim extra credit on your assignment, you must state in your README file which task or tasks you've completed. In addition, it is essential that you provide detailed instructions (i.e. a concrete set of commands) on how to test your work. Your provided instructions should clearly demonstrate all the features of your work.

You are, of course, welcome to ask me for help in completing these tasks or if you have any questions about them.

2 Tasks

2.1 Cancel locks after crash

A `lock_get` RPC includes a string parameter specifying the identity of the requester, which is typically a client. However, a lock may also be requested by a server.

For this task, assume that a requester ID in the form *:xxxx* identifies a lock request by a server, where *xxxx* is a port number. That is, if the first character of a requester ID is a colon, then the remaining characters are assumed to be the TCP port on which the requesting server accepts RPCs.

Your task is to extend the heartbeat/lease mechanism in your application to automatically release locks held by crashed servers. The semantics of locks held by non-servers (that is, in cases where the requester ID does not have the form described above) remain unchanged. When such a lock is automatically released, an appropriate message (including the name of the lock and the endpoint of the server) should be emitted to the terminal by the view leader.

2.2 Deadlock detection

As in the previous task, assume that servers may request locks using the special form of requester ID.

Your task is to write an automatic deadlock detector. If it detects deadlock, the view leader should emit an appropriate message (including the lock names and servers endpoints involved) to the terminal. No other action should be taken.

2.3 Readers/writer locks

Extend the locking system to support multiple-reader, single-writer locks. The `lock_get` command should support an additional optional parameter indicating the type of lock. The parameter may be `reader` or `writer`. If the parameter is omitted, `writer` is the default, in which case the semantics of the lock remain as specified in the assignment (i.e. exclusive).

For example, this command would request a read lock named `lockname`:

```
./client.py lock_get lockname requester reader
```

The semantics of such a lock are as we discussed: any number of readers may concurrently hold a readers/writer lock if no writer holds the lock, but only a single writer may hold it at a time.

As part of this task, you must select an algorithm for prioritizing waiters (that is, deciding who gets the lock next after it is released). Your choice should try to be fair and to avoid starvation. You must document your choice of algorithm and justify your reasoning in choosing it.