Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

6.826 – Principles of Computer Systems

Handout 54 December 6, 1993

Material for the Final Exam

This handout contains some preliminary material for the final exam. You should read it over and understand what is being proposed *before* the exam.

Note: the final exam is on Thursday, December 16, at 9 am. It will take place in two rooms, 26-204 and 26-210. The rooms are next to each other; come to either one and we will move people as necessary to balance the load.

The exam will be open-book. You should bring all the handouts from this term with you; some of the problems on the exam may refer to particular handouts, and you will find it easier to answer them if you have the handouts with you.

Some of the problems on the final exam will revisit the material from Problem Set 9 on Mellor-Crummey and Scott's mutex implementation. You might want to review that material before the exam.

The remainder of this handout describes a proposed modification to the Paxos protocol; the exam will include some questions about it.

1 A Faster Consensus Algorithm?

Having read Lampson's paper on "Hints for Computer System Design", Ben Bitdiddle proposes to modify Lamport's Paxos consensus algorithm to optimize the common case in which there is a single leader and consensus is achieved in a single ballot. In particular, Ben proposes to combine the querying and polling phases of Paxos: rather than querying the voters for old votes before choosing a value, a leader chooses a value (optimistically assuming that it will be a good one) and sends a "query-and-poll" message containing the ballot number and the value to the voters. The voters respond with their votes in previous ballots together with their vote on this value. If the value chosen was an anchored value and yes votes for this ballot are received from a majority of voters, the leader determines that consensus has been reached and sends the outcome to the voters. A leader can also be pessimistic and query before polling; this would make sense if the leader knows that there have been previous ballots and wants to find out what happened before trying to choose a value.

More specifically, Ben suggests the following changes to the Paxos implementation in Handout 42:

- Change the type `K` to add two new kinds of messages, `qp` (for "query-poll" message) and `allVotes` (for the combined response to a query and poll message).

- Change the type `Status` to add a new status `optimistic` for the leader.
- Add leader actions as follows:

```
[] % New ballot; optimistically guess value; query and poll together.
   VAR i, d' | i > n.i /\ d' IN Allowed() =>
        n.i := i; d := d'; theirVs := {}; status := optimistic

[] % Send qp message.
   status = optimistic => Send(qp, n, d)

[] % Receive an allVotes message.
   status = optimistic => VAR m := Receive(allVotes) | m.n = n =>
       theirVs := theirVs + m.x AS VS

[] % Send outcome message if a majority has voted 'yes' and value
   % is anchored (when being optimistic).
   status = optimistic
      /\ P.majority({p' | (n, p', d  ) IN theirVs})
      /\ IsAnchored(d,n,theirVs)
       => Send(outcome, n, d)
```

- Modify leader actions as follows:

```
[] % Send outcome message if a majority has voted 'yes' and went
   % through querying phase before polling.
   % Note: this does not check that the value is anchored before
   % sending the outcome, since a leader whose status is polling
   % chose the value based on votes received in the querying phase.
   status = polling /\ P.majority({p' | (n, p', d  ) IN theirVs})
       => Send(outcome, n, d)
```

- Add voter actions as follows:

```
[] % Receive qp message.
   VAR m := Receive(qp) |
        myVs := myVs + {n' | n' < m.n /\ VotesN(myVs, n') = {} | (n',p,nil)};
        BEGIN VotesN(myVs, m.n) = {} => myVs := myVs ++ (m.n, p, m.x AS d)
        [*] SKIP END;
        Send(allVotes, m.n, myVs)
```

- Add the following function to test whether a value is anchored:

```
FUNC IsAnchored(d,n,vs) -> Bool =
    RET ( ALL n' | n' < n =>
            Dead(vs, n')
            \/ (Val(vs, n') = d /\ Live(vs, n')) )

% True only if n cannot be dead given the votes in vs.
FUNC Live(vs,n) -> Bool =
    RET P.majority({p | (n, p, Val(vs,n)) IN vs}) /\ Val(vs,n) # nil
```