Engi 8893 / 9869 Assignment 3

Due April 8, 2009.

Please submit questions Q0 and Q1 on paper. Solutions must be typed or equivalently legible. Code should use pseudocode similar to that in the text. Explain your solutions clearly using pseudocode and prose as appropriate.

Q0. A file is held on four geographically distributed servers. Each server and each client reads messages from a unique channel. Call the channels fileClientCh[c], serverCh[s], and managerClientCh. Each server must be able to respond to the following messages from file clients. In each case, c is the file client's identifier. Each file client may communicate with any of the four servers. For example, a file client may open the file on server 0, but read it on server 1.

- open(c) — The server should respond with ackOpen(len) where len is the file's current length. Or if the file is already open, it should reply with nackOpen().
- read(c, i, j) — If the client does not have the file open, the server should reply with notOpen(). If $\neg(0 \leq i \leq j \leq \text{len}(\mathit{file}))$, then the server replies with badArg(). Otherwise it replies with ackRead($string$) where $string = \mathit{file}[i, ..j]$.
- write($c, i, j, insertion$) — If the client does not have the file open, the server should reply with notOpen(). If $\neg(0 \leq i \leq j \leq len)$, then the server replies with badArg(). Oterwise it updates the file with a new value which is

  $$\mathit{file}[0, ..i] + \mathit{insertion} + \mathit{file}[j, ..]$$

  and the server replies with an ackWrite(len) message, where len is the new length.
- close(c) — If the client does not have the file open, it should reply with notOpen(). Otherwise the file becomes is closed and the server replies with ackClose().

However servers may go down for maintainence. When a server is down, it does not send messages to other servers and replies to all messages (except goUp) with an immediate down() message.

A manager client will send the following message to servers

- goDown() — The server replies with a ackDown() message and switches to the 'down' state.
- goUp() — The server replies with an ackUp() message, and reumes normal communications.

Design the system so that 2 servers may be down with no loss of functionality. (File clients may need to hop around to find a working server.)

Servers communicate with each other using a protocol that you design.

- Describe your approach and each server's state.
- Describe the messages of your server-to-server protocol.
- Give the algorithm that each server follows.

Q1. Consider a module that consists of a private state variable s of type S and a public set of operations of the form

---

procedure $op_i(T\ a)$ result $r : U$ {

$\langle \mathbf{await}(B_i)\ s := f_i(s, a); r := g_i(s, a); \rangle$ }

That is, each operation waits until some assertion, B_i, is true and then deterministically changes the state and computes a result. Suppose that we wish to replicate the state across multiple helper processes which communicate by broadcasting. Each helper will have its own copy of the state and be capable of computing all the f and g functions. Show how to implement the operations on the state so that all helpers execute the operations in the same order. Operations should not be unduly delayed.

Hint: Model your solution on the distributed semaphores in section 9.5.2 of MPDP.

Q2. Complete the following smv module.[1] Verify all properties.

```
module main(req1,req2,req3,ack1,ack2,ack3)
{
    input req1,req2,req3 : boolean;
    output ack1,ack2,ack3 : boolean;

    ...

    mutex : assert G ~(ack1 & ack2 | ack1 & ack3 | ack2 & ack3);
    service : assert G ((req1 | req2 | req3) -> (ack1 | ack2 | ack3));
    no_waste1 : assert G (ack1 -> req1);
    no_waste2 : assert G (ack2 -> req2);
    no_waste3 : assert G (ack3 -> req3);
    no_starve1 : assert G F (~req1 | ack1);
    no_starve2 : assert G F (~req2 | ack2);
    no_starve3 : assert G F (~req3 | ack3);
}
```

For extra challenge. Try to verify

```
fast1 : assert G ( req1 & X req1 & X X req1 -> ack1 | X ack1 | X X ack1 ) ;
fast2 : assert G ( req2 & X req2 & X X req2 -> ack2 | X ack2 | X X ack2 ) ;
fast3 : assert G ( req3 & X req3 & X X req3 -> ack3 | X ack3 | X X ack3 ) ;
```

Websubmit as `arb3.smv` to the assignment 3 queue of course 8893.

[1] SMV is available from http://embedded.eecs.berkeley.edu/Alumni/kenmcmil/smv/ . If requested, I will ask the CCAE to install SMV on a small number of computers in the labs.