Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

6.826 – Principles of Computer Systems

Handout 56 December 8, 1993

Solutions to Problem Set #11

The maximum score for this problem set was 20 points.
Revised AMO2 spec to allow service to reply before committing actions.

```
MODULE AMO2 [
   V,                                         % Value
   S WITH { s0: () -> S },                    % State
   A WITH { meaning: A -> S -> (V,S) }        % Action
   ] =

TYPE SQA = SEQ A

VAR s := S$s0()                               % State of service.
    volatile: SQA := {}                       % Volatile actions not yet
                                              %   committed.
    pending: SET A := {}                      % Failed actions to be done.
    down: BOOL := false                       % Flag: true when system has
                                              %   failed and not finished
                                              %   recovering.

% Either do the action or raise failed.
% failed can be raised only if the system is down sometime during the
% execution of the operation.
APROC Do (a) -> V RAISES {failed} =
  VAR failure: BOOL |
  << failure := down >>;
  << VAR v,s' | (v,s') = a.meaning(Apply(s,volatile)) =>
                  volatile := volatile ++ a; RET v
    [] failure => pending := pending ++ {a}; RAISE failed >>

% Do a pending failed action.
THREAD DoPending() =
```

```
  << VAR a | a IN pending =>
        volatile := volatile ++ a; pending := pending - {a} >>

% Discard a pending failed action.
THREAD LosePending() =
  << VAR a | a IN pending =>
      pending := pending - {a} >>

% Commit volatile actions.
THREAD Commit() =
   DO << VAR v,s' | (v,s') = volatile.head.meaning(s) =>
           s := s'; volatile := volatile.tail >>

% Appears to happen whenever a node crashes or the network fails.
THREAD Fail() = DO << down := true >>; << down := false >> OD

% Lose volatile actions.
THREAD Crash() =
   DO << VAR sqa, a, sqa' | volatile = sqa ++ a + sqa' =>
           volatile := sqa + sqa' >>

% Apply actions in sqa to s1.
FUNC Apply(s1, sqa) -> S =
   sqa = {} => RET s1
  [*] VAR v,s' | (v,s') = sqa.head.meaning(s1) => RET Apply(s', sqa.tail)

END AMO2
```