Cpts 483/580 Feb. 24, 2013

Update Ring Assignment

No class this Thursday.

Midterm grades - based on exam and the early homeworks.

Implement shared state with a server process.

```
server(State) ->
    receive
        {Pid, {get}} -> Pid ! State, server(State);
        {Pid, {set, NewState}} -> Pid ! State, server(NewState);
        {Pid, {update, UpdateFun}} -> Pid ! State,
                                      server(UpdateFun(State))
    end.

init() -> server([]).
```

So w' a user of this state server

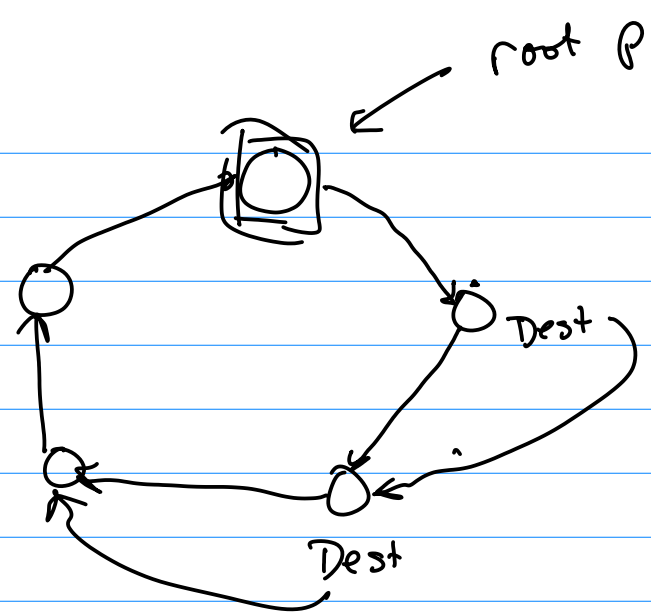
```
register (myInt, spawn (fun server:init/0)).
```

```
myInt ! {self(), {set, 0}},
receive
    Ans -> Ans
end
```

```
rpc (Server, Msg) ->
        Server ! {self(), Msg},
        receive
            Reply -> Reply
        end.
```

```
Ans = rpc (myInt, {set, 0}).
```

```
get (Server) -> rpc(Server, {get}).
set (Server, V) -> rpc (Server, {set, V}).
```



1 - msg - at a time
Send All msgs
then
recv All msgs

Send some
recv what's available
Send some more

○ processes

```
loop (Dest) →
    receive
        quit → Dest ! quit;
        M → Dest ! M, loop (Dest)
    end.
```

Suppose That's our implementation of The forwarder.
I have to build a lot of Them and connect Them
together by entering in Dest differently in each one.

{ • each process somehow creates and initializes its successor.
* • "master" process creates all of the forwarders and tells them
to whom they should send. }

X

- pass a parameter as part of spawning each process
- send the dest as a message to each spawned process

```
init(Dest) ->
    receive
        Dest -> loop(Dest)
    end.
```

```
param_batch_init(N) -> param_batch_init_loop(N, self()).

param_batch_init_loop(N, First) ->
    lists:foldl(fun spawnLoop/2, First, seq(1, N)).

spawnLoop(_, A) -> spawn(ring, loop, [A]).
```

What is the behavior of the root process?

Send some / receive some

```
mixed (M, N) ->
    Dest <- param_batch_init (N),
    drive (1, 1, M, Dest),
    Dest ! quit.


drive (ToSend, ToRecv, Limit, Dest) ->
    Timeout = if ToSend =< Limit -> 0;
                 true -> infinity
              end,
    receive
        ToRecv when ToRecv < Limit ->
                drive (ToSend, ToRecv+1, Limit, Dest);
        ToRecv -> {}
    after Timeout ->
            Dest ! ToSend, drive (ToSend+1, ToRecv, Limit, Dest)
    end.
```

A few hints:

use some output statements to make sure you are constructing the ring properly.

```
io:format("Process ~p sends to ~p~n", [self(), Dest])
```

You don't want to do this in final version of your code —

In root process, esp. if you are doing mixed — want to see interleaving of sending and receiving — use output statements in the receive and send branches of driver to print message numbers.

Again, not in the final code.