CS511 – Endterm – Topic 3 – December 9, 2020

Exercise 1 (*Message Passing in Erlang, 5 pts*)

The following program in Erlang implements the Producers/Consumers synchronization pattern using message passing. It is missing the implementation of `loopPC/4` which you are asked to complete. The arguments to `loopPC` are as follows:

- `Cs`: Number of consumers that started consuming
- `Ps`: Number of producers that started producing
- `MaxBufferSize`: Buffer size (constant, greater than 0)
- `OccupiedSlots`: Number of currently occupied slots in the buffer

Note: Multiple producers and consumers should be allowed.

Deliverable: file pc.erl

```
1  consumer(Id,Buffer) ->
2      timer:sleep(200),
3      io:fwrite("Consumer ~p trying to consume~n",[Id]),
4      Ref = make_ref(),
5      Buffer!{start_consume, self(), Ref},
6      receive
7          {ok_to_consume, Ref} ->
8              io:fwrite("Consumer ~p consuming~n",[Id]),
9              Buffer!{end_consume},
10             io:fwrite("Consumer ~p stopped consuming~n",[Id]),
11             consumer(Id,Buffer)
12     end.
13
14 producer(Id,Buffer) ->
15     timer:sleep(1000),
16     io:fwrite("Producer ~p trying to produce~n",[Id]),
17     Ref = make_ref(),
18     Buffer!{start_produce, self(), Ref},
19     receive
20         {ok_to_produce, Ref} ->
21             io:fwrite("Producer ~p producing~n",[Id]),
22             Buffer!{end_produce},
23             io:fwrite("Producer ~p stopped producing~n",[Id]),
24             producer(Id,Buffer)
25     end.
26
27 loopPC(Cs, Ps,MaxBufferSize,OccupiedSlots) ->
28        %% implement me
29
30 startPC() ->
31     Buffer = spawn (fun() -> loopPC(0,0,10,0) end),
32     [ spawn(fun() -> producer(Id,Buffer) end) || Id<- lists:seq(1,10)],
33     [ spawn(fun() -> consumer(Id,Buffer) end) || Id<- lists:seq(1,10)].
```

This page is intentionally left blank. You may use it for writing your answers.

Exercise 2 (*Spin, 5 pts*)

Consider the following solution seen in class to the "Ferry Problem". The solution here is coded in Promela. The code for `acquire` and `release` is omitted.

```
active proctype Ferry() {
  byte coast = 0; /* East = 0, West = 1 */
  int i;
  do
    ::
       for (i : 1..CAPACITY) {
         release(permissionToGetOn[coast]);
       };
       for (i : 1..CAPACITY) {
         acquire(permissionToSetSail);
        };
       /* move to other coast */
       coast = (coast+1) % 2;
       /* reached other coast */
       for (i:1..CAPACITY) {
         release(permissionToGetOff);
       }
       for (i: 1..CAPACITY) {
         acquire(permissionToReboard);
       }
  od
}

proctype PassengerAtCoast(byte coast) {
  acquire(permissionToGetOn[coast]);
  release(permissionToSetSail);
  /* waiting to arrive at other coast */
  acquire(permissionToGetOff);
  release(permissionToReboard);
}
```

Introduce assertions (and any variables you might need to formulate the conditions in the assertions) to check that this solution is correct in the following sense:

1. The ferry only leaves if it is full.
2. A passenger only gets off if the ferry has arrived at a coast (i.e. the ferry is not in transit).

Deliverables:

1. Item 1. File ferry1.pml (including any assertions) + output1.txt (output from jpsin indicating that there are no errors)
2. Item 2: File ferry2.pml + output1.txt (output from jpsin indicating that there are no errors)

This page is intentionally left blank. You may use it for writing your answers.