EE 360P Homework #1

3. Create a TACC User ID and submit it along with Homework questions.

* ama, vod

4. Show that any of the following modifications to Peterson's algorithm makes it incorrect:

a) A process in Peterson's algorithm sets the turn variable to itself instead of setting it to the other

process.

* Assume the scenario
  + P0: wantCS[1] = true;
  + P0: turn = 0;
  + P0: while(wantCS[1] && (turn == 1)) -> False, waits to be true
  + P1: wantCS[0] = true;
  + P1: turn = 1; -> after this is set, P0 enters CS
  + P1: while(wantCS[0] && (turn == 0)) -> False, waits to be true
  + P0: releaseCS(0); -> wantCS[0] = false;
* P1 will be stuck waiting for its turn to be set, and therefore removing the Starvation-freedom property of Peterson’s algorithm

b) A process sets the turn variable before setting the wantCS variable.

* Mutual exclusion breaks:
  + 1. P1: turn = 0;
  + 1. P0: turn = 1;
  + 2. P1: wantCS[0] = true;
  + 2. P0: wantCS[1] = true;
  + 3. P1: while(wantCS[0] && (turn == 0)) -> True
  + 3. P0: while(wantCS[1] && (turn == 1)) -> True
  + 4. P0, P1: Enters critical section

5. Peterson's algorithm uses a multi-write variable turn. Modify the algorithm to use two variables turn0 and turn1 instead of turn such that P0 does not write to turn1 and P1 does not write to turn0.

class PetersonAlgorithmMod implements Lock {

boolean wantCS[] = {false, false};

int t0 = 1;

int t1 = 0;

public void requestCS(int i) {

int j = 1 – i;

if(i == 0){t0 = t1;}

else{t1 = t0;}

wantCS[i] = true;

if(i == 0){

while(wantCS[j] && (t0 == 1)){t0 = t1;}

}

else{

while(wantCS[j] && (t1 == 0)){t1 = t0;}

}

}

public void releaseCS(int i) {

if(i == 0){t0 = 1;}

else{t1 = 0;}

wantCS[i] = false;

}

}

6. Show that the bakery algorithm does not work in the absence of choosing variables.

* The priority order for the processes will be incorrect as it violates mutual exclusion.
* Consider two processes that both assign their choosing variable where they would both be assigned the same number where process P1 with a higher PID happens to be in the process of choosing its number slightly after process P2 with a lower PID.
* P1 chooses the max number, and since P2 is not assigned yet, P1 will then go to CS.
* Similarly, P2 chooses the same max number as P1.
* Since P1 has a lower PID, it also enters CS, therefore violates mutex.