Yeukhon Wong 8066
CSC 332 Operating System
Fall 2012

Homework #1

Prove whether the following algorithm works for a two-process execution or not:

```
flag[0] = False
flag[1] = False
```

Process P0 (i=0)	Process P1 (i=1)

forever
  flag[i] = True
  while (flag[mod([i+1], 2) == True)
  do noop;
  <CS>
  flag[i] = False
  <non-CS>
	forever
	  flag[i] = True
	  while (flag[mod([i+1], 2) == True)
	  do noop;
	  <CS>
	  flag[i] = False
	  <non-CS>

In general, there are three cases to consider for a concurrent mutex algorithm

1. when one process gets to execute first than the other,
2. when one process is already in CS and the other is trying to enter CS,
3. when both intend to enter CS

Induction Hypothesis:

For concurrent mutex algorithm, no two can be in the CS simultaneously. Only one process can be using the CS at a time. The other one is excluded (hence, mutual excluded) from CS, and must wait for CS to be free again.

Base case:

Suppose, without the loss of generality, we pick P0 as the first process that get to run, and that P1is delayed for an infinite amount of time, such that P0 gets to run. Such trivial example means P0will always access CS because the loop invariant fails as `flag[mod([i+1], 2)` (where i is 0) will always return False, and the loop will exit immediately and proceed to CS without further a do. This trivial base case satisfies the hypothesis or the induction.

Now let's consider case #1:

If now P0 gets to run first, P0 will set its flag to True. At that point, P1's flag is still False, and so the loop invariant for P0 is False and exit. Then P0 proceed into CS. Meanwhile, P1 now sets its flag, but the loop invariant is True until P0 exits CS (at that point, P0 sets its flag back to False).

So case #1 holds.

Now let's consider case #2:

Well. If P0 is already in CS, from case 1, we've learned that P1 will wait P0 to finish and then proceed to enter CS when P0 is finished. So case 2 also holds.

Finally, let consider case #3:

When both intend to enter CS? If both want to access, and say both are now executing their own `flag[i] = True` , then we have a deadlock. This happens either on a multi-core (each process is on a different core), or if context switch from P0 to P1 when P0 has just finished setting the flag. In either case, both processes satisfy its own loop invariant. Both set the flag to `True`. **This is a deadlock**.

A deadlock contradicts our induction hypothesis because neither processes gain access to the CS, which means both are excluded from the CS. In this two-process system, we must have one process be able to enter CS if it wishes (even if it has to wait for the other to finish). But deadlock means forever waiting. Thus, proof by contradiction says this algorithm cannot work properly.