**Nyeinchan Kyaw**

CS 340

03/13/2012

**Homework Assignment 3**

**A. Consider the 2nd attempt (from the lecture notes). Is the “No Starvation” condition satisfied?**

**Hint: in your proof you might want to check if there is a particular execution sequence by which a process might be able to use the CS over and over, while the other process is starving in the while loop.**

Second Attempt

Each process should have its own key to the CS so that if one process is executing outside the CS the other one is able to get in to its CS.

Replace the turn variable with a shared global variable initialized to false.

flag is initialized to 'false'

P0 P1

while(true) { while(true){

while (flag[1]) do no-op; while (flag[0]) do no-op;

flag[0] = true; flag[1] = true;

CS CS

flag[0] = false; flag[1] = false;

remainder section; } remainder section; }

First, in order to prove “No starvation” condition satisfied, we have to prove that no process will be postponed indefinitely long(forever). In second attempt, both flags are initialized to ‘false’. Thus, in second while loop, both P0 and P0 doesn’t go into loop since both of the flags are false. Then, in P0, flag[0] is set to “true”. Also, in P1, flag[1] is set to“true”. Then, both can enter CS at the same time. This situation violates “Mutual Exclusion” condition because P0 is not blocked when P1 attempts to enter CS. Likewise, P1 is not blocked when P0 attempts to enter CS. PO and P1 catch-up with each other in CS. After both P0 and P1 exit CS, flag[0] and flag[1] are set to be ‘false’ then both P0 and P1 finish processing the remainder section. There can be a sequence that a process P0 takes a little bit longer than P1 or a process P1 takes a bit longer than P0. We can say that there is a starvation when we have a specific sequence by which a process will busy wait forever. In this second attempt, there is possible situation by which P0 will wait forever while P1 uses the CS over and over, likewise, there is also possible situation by which P1 will wait forever while P0 uses the CS over. Therefore, “No Starvation” condition is not satisfied.

**B. Prove that the Peterson Solution is correct by showing that all 3 conditions for a correct solution to the Critical Section Problem are respected.**

**Hint: you can use the textbook comments but your proof should be clearer and more detailed.**

Peterson's solution :

turn = 0;

flag[0] = false;

flag[1] = false;

P0 P1

while(true){ while(true){

flag[0] = true; flag[1] = true;

turn =1; turn = 0;

while (flag[1] and turn ==1) do no-op; while (flag[0] and turn ==0) do no-op;

CS CS

flag[0] = false; flag[1] = false;

remainder section; } remainder section;}

Mutual exclusion:

Assume by contradiction that both processes are in the CS. This mean both flag[0] is true and flag[1] is set to ‘true’. If p0 is in the CS, turn must be set to 0. Also, if P1 is in the CS, turn must be set to 1. However, turn cannot be set to 1 and 0 at the same time. In Peterson’s solution, turn can only be either 0 or 1 before a process enters CS. If turn==1, then process P1 can enter its critical condition. If turn==0, then process P0 can enter its critical condition. When flag[1] is true and turn==1, P1 is ready to enter CS and P0 will busy wait. When flag[0] is true and turn==0, P0 is ready to enter CS and P1 will busy wait. In Peterson’s solution, only one process either P0 or P1 can be in the CS at a given time. To make sure only one process can enter CS, we use flag and turn. We have to check both flag and turn because flag[0] and flag[1] can be set to true at the same time but turn can be either 1 or 0.Therefore, the assumption is incorrect and “Mutual exclusion” condition is satisfied.

No starvation:

Assume that P0 attempts to enter CS but flag[1] is true and turn==1. Therefore, P0 cannot enter CS and has to stay in while loop.When P1 exits CS or P1 is not ready to enter CS, flag[1] is set to ‘false’ so that P0 has a chance to enter its critical condition. Even if P1 is fast and attempts to enter CS again after it exits its CS before P0 enters CS, it won’t get through because P0 is busy waiting; turn is set to 1 and flag[0] is set to ‘true’ already. Thus, P1 can’t enter CS again and again. Once it gets out, P0 will enter its CS immediately because flag[1] is false and turn is 0. There is no possible situation by which one of the processes (either P0 or P1) will wait forever while the other process uses the CS over and over. Therefore, “No starvation” condition is satisfied.

Progress Condition:

No Delay:

Assume that P0 exits CS, flag[0] will be set to ‘false’ and it will be in a remainder section. That means CS is now available (empty). When flag[0] is false and turn is set to 0, P1 can enter CS (since CS is empty at the moment). Since P0 is in the remainder section, it will not prevent P1 from attempting to enter CS anymore. P1 will not be delayed because P0 is done with its critical section already. Thus, “No Delay” condition is satisfied.

No Deadlock:

Assume by contradiction that there is Deadlock. That means that both P0 and P1 will be blocked in the entry section while CS is empty. Consequently, flag[1] is set to ‘true’ and turn is set to 1. Also, flag[0] is set to ‘true’ and turn is set to 0. In that way, both P0 and P1 will get stuck in while loop and can’t enter CS while CS is empty and we can prove that there is a Deadlock. However, turn can only be set to 0 or 1 in Peterson’s solution and that will make one of the while loops condition unsatisfied. Let’s say turn is set to 0 and flag[1] and flag[0] are true, P0 will enter CS, and if turn is set to 1 and both flags set to ‘true’, P1 will enter CS.That violates Deadlock condition since only one process(either P0 or P1) will be in entry section waiting another process to finish executing its CS and there can’t not be both P0 and P1 in the entry section. Therefore, the assumption is incorrect and “No Deadlock” condition is satisfied.