Jake Leverett - jsl3356

Tanner Hudnall - tch2368

# HW #1 Theory

1/31/22

3a) Let there be 2 Processes, Process 0 and 1. Let Process 1 be in the critical section already. If a process 0 requests to be in the CS and turns the variable to itself, turn = 0, then it does not busy wait while Process 1 is in the CS as the condition (want[S[1] ∧ turn = 1) results to false. This put both Process 1 and 0 in the CS at the same time violating the rule of mutual exclusion.

3b) Given 2 Processes, Process 0 and Process 1

If Process 0 requests CS first and first sets turn to Process 1, then Process 1 requests CS and sets turn to Process 0 and want[S[1] = true. With scheduling it is possible Process 1 can now check the busy wait conditions before Process 0 sets its want variable, so Process 1 sees that it is P0's turn but doesn't want CS so it enters the CS. Process 0 now sets want[S[0] = true, then checks busy wait conditions and sees that want[S[1] = true but it is Process 0's turn, so it also enters the CS. This violates the rule of mutual exclusion and makes Peterson's algorithm incorrect.

4) To prove that Peterson's algorithm does not result in starvation, we must prove 3 cases of trying to request the critical section

Given 2 processes: Process 0 and Process 1

Case 1: If process 0 wants to enter the critical section and the CS is empty. This would result in the turn being in favor of process 1 however since the CS is empty then want[CS[1] = false resulting in the busy wait loop terminating and Process 0 is able to enter the CS as normal, not resulting in Starvation.

Case 2: If Process 0 wants to enter the CS and the CS is held by process 1. This results in the turn being in favor of process 1 and want[CS[1] = true. Process 0 must wait for Process 1 to exit the critical section for want[CS[1] = false. Assuming that process 1 will eventually exit the CS then Process 0 is free to enter the CS, avoiding Starvation.

Case 3: If Process 0 and Process 1 request access to the CS at the same time. This results in both want[CS[0] = true and want[CS[1] = true. However, since turn is an integer it is not possible for it to be 0 and 1 at the same time. This ensures that only one process makes it past the busy wait and is able to enter the CS. This leaves the other process to be in busy wait while waiting for the other to exit the CS. This is similar to Case 2 and we have already proved that this does not result in starvation.

We have proved all 3 possible cases of trying to request the CS and have proved the each do not result in starvation. Therefore, Peterson's algorithm is free from starvation.