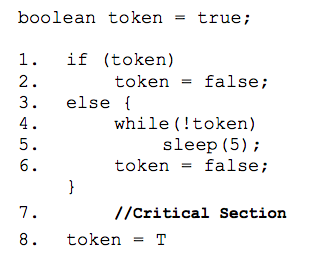
Keith Schmitt

Professor Dulimarta

CIS 452

Homework 5

1. *Consider the following 2-process solution to the Critical Section problem:*

**

*• Show that the algorithm fails the requirement(s) for Mutual Exclusion.*

**This algorithm fails Mutual Exclusion because a program can be interrupted at any time.**

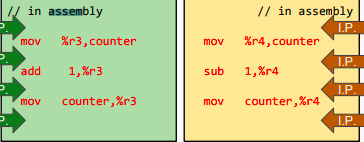
**Proof: Assume that Pi is at line 1, Pj is in the critical section. Because Pj is in the critical section, it should not also have Pi allowed into the critical section as well. Because if Pi is interrupted after executing line 2 token will be false, and say Pj continues to execute and sets token to T. This means that when Pj comes to the entry section again, Pj will be allowed to enter the critical section along with Pi, thus failing Mutual Exclusion.**

* 1. *Show that, if the wait() and signal() semaphore operations (either original or modern version) are not executed atomically, then mutual exclusion may be violated.*

**A wait or signal call by their simple implementation can be interrupted after the increment. Because in the assembly instruction, after the addi, they can be interrupted before the move command.**

**Increment assembly instructions with interruptible points**

**taken from lecture slides:**

****

* 1. *The first known correct software solution to the critical-section problem for two processes was developed by Dekker. The two processes, P0 and P1, share the following variables:*

*../../../../Dropbox/Screenshots/Screenshot%202018-02-19%2017.49.55.png*

*The structure of the two-process code for generic process Pi (i == 0 or 1) and (j==1 or 0) shown in Figure 5.25 (6.25?) in your textbook*

*. • Prove that the algorithm satisfies the requirement(s) for Progress*

**(**I couldn’t really find Figure 5.25 in the textbook, found a 6.25 but not 5.25 which is practically this question)

**This algorithm proves Progress because:**

**Case I: If we assume that Pi is interested but Pj is not. If that is the case, then we know that flag[i] is true while flag[j] would be false. Because flag[j] is false, we know that Pi will be allowed to proceed to go into the critical section.**

**Case II: If we assume that both Pi and Pj are interested and are in the entry section, then we know that flag[i] is true and flag[j] is true one of the processes will be blocked in the busy wait inside the entry section because turn was updated at the exit section of the other process. So if Pj was the last one to run, then turn would be i, allowing Pi to enter the critical section in this case.**

*5.39 Consider the deadlock situation that could occur in the Dining Philosophers problem when the philosophers obtain the chopsticks one at a time. Show how the four necessary conditions for deadlock indeed hold in this setting. Describe how deadlocks could be avoided by eliminating any one of the four necessary conditions.*

**Since For something to be in deadlock we need Mutual Exclusion (ME), Hold & Wait (HW), No Preemption(NP), and Circular Wait (CW) to all be present.**

**ME: It is mutually exclusive because each philosopher is forced to wait until it has both the left and right chopstick.**

**HW: It has Hold & Wait because it will grab one chopstick one at a time, holding that resource while trying to acquire the other.**

**NP: It has No Preemption because no Philosopher is going to forcibly take another philosopher’s chopstick.**

**CW: It has Circular Waiting because each philosopher can wait for their adjacent philosophers to use chopsticks. For example, philosopher0 will wait for philosopher1 to use the same chopstick and vice versa.**

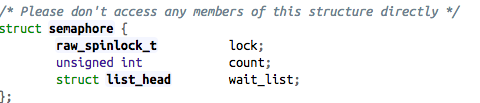
**Since Deadlock needs all of these it can be written as:**

**Which means the logically equivalent contrapositive:**

**Meaning that if we prove that one of these is not true then we don’t have a deadlock, otherwise we do have deadlock.**

*Practical: Find and report the definition (the actual code structure) of a semaphore in linux. Note that because of their signaling capability, semaphores are often considered a “communication” mechanism between processes.*

**Structure for Semaphore in Linux:**

****

**It has a lock which will lock the process from interrupts. A count for synchronization processes. And a wait\_list to see what processes can use this semaphore. Using these variables and some other functions, a semaphore is a good tool for interprocess communication.**

Extra Credit: a multiple-process solution to the Critical Section problem, known as the Bakery Algorithm, is posted on the course info page.

• What is the purpose of the 1st while loop in the algorithm?

**The purpose of the 1st while loop:**

**../../../../Dropbox/Screenshots/Screenshot%202018-02-20%2003.30.56.png**

**is that it will prevent further progress until another process determines the max of the number[] array.**

• Demonstrate what problem could occur if it was not included.

**If this was not included, then if a Process i (Pi) is in the for loop and a Process j (Pj), then when Pi is evaluating ../../../../Dropbox/Screenshots/Screenshot%202018-02-20%2003.37.43.png in the second Boolean expression of the second while, then it could include the value that Pj will want to use as well.**