**Name: YiSHi Lu**

**Course: CS 344-715**

**Professor: Simina Fluture, PhD**

**Date: 03/21/2016**

Homework 3

**I. Prove that the conditions of the Critical Section problem are satisfied in Bakery’s Algorithm.**

**• Mutual Exclusion (you can use the 2 given assertions)**

**• No Starvation**

**• Progress Condition (no delay and no deadlock)**

**Code:**

**L1: While(true) {**

**L2: choosing[i] = 1;**

**L3: number[i] = 1 + max(number[1], …, number[N]);**

**L4: choosing[i] = 0;**

**L5: for (int j = 1; j <= N; j++) {**

**L6: while (choosing[j] = 1) {}**

**L7: while(number[j] <>0 and (number[j],j) < (number[i], i)) {}**

**}**

**L8: Critical Section**

**L9: number[i] = 0;**

**}**

**For Mutual Exclusion:**

Assume there are N processes, and only process Pi is in the CS.

By Assertion 2: if process i executes its Critical Section, and process k is in the bakery (k !=i) then ( number[i], i ) < (number[k], k ). This means Pi has the smallest (number[i], i) over all N processes; in other words it has the highest priority when it leaves from the Bakery, so other processes must block in the Bakery until Pi gets out.

By Assertion1: if processes i and k are in the bakery, and i entered the bakery before k entered the doorway, then number[i] < number[k]. This means when Pi finishes its execution in the CS and goes through the doorway or there are new processes just enter the doorway, they must have lower priority than other processes which are already in the Bakery; as a result, they must have lower priority than the process is in the CS, so they cannot enter the CS.

By these two assertions above, when a process is in the CS, then it has highest priority over all processes which are in the Bakery. Moreover, if there are some processes just enter the doorway, then these processes have lower priority than processes which are in the Bakery. As a result, these processes must have lower priority than the process which is in the CS. As a result, when a process is in the CS, no other process can enter the CS, so mutual exclusion is satisfied.

**For No Starvation:**

Assume there is N processes, and Pi with the smallest (number[i], i) is starved. Because Pi only can starve in the Bakery, it cannot change its number[i] (number[i] cannot be zero as well). Since there are a finite number of processes, and they are not starved, eventually all these processes will enter and exit the CS. After these processes exit the CS, they will go to entry section, and then they must get a larger number than Pi. Afterwards, if these processes want to starve Pi, they must access the CS again; but it is impossible, because L7: “while(number[i] != 0 and (number[i], i) < (number[k], k))” must be true for all of them. At this point, Pi will pass all the tests in the for loop and enter the CS. Therefore, that is a contradiction, so No starvation is satisfied.

**For No deadlock:**

Assume there is N processes, and no processes can access the CS, so there is a deadlock; and then all processes must be satisfied that L7: “while(number[j] != 0 and (number[j], j) < (number[i], i)) is true”, which means that number[j] < number[i] must be true, or number[j] == number[i] and j < i must be true for all processes. (For all processes, number[j] != 0 must be true because no one exits from the CS. Moreover, choosing also must be 0 for all processes because they are in Bakery).

For number[j] < number[i]. If the processes execute L3: “number[i] = 1 + max(number[1], …, number[N])” at different time, then they will get different value of number[i]. Therefore, there must be some process (Pk) with smallest number, so number[j] < number[k] is false to it. Therefore, this process (Pk) can access the CS.

For number[j] == number[i] and j < i. If the processes execute L3: “number[i] = 1 + max(number[1], …, number[N])” at same, then they may get same value of number[i]; as a result, number[j] == number[i] will be true to them. However, they are different processes, so their process ID must be different. Therefore, the process (Pk) with the smallest process ID can access the CS immediately because j < k is false to this process.

In short, there always exist some thread Pi with smallest pair (number[i], i), and it never has to wait for any other thread. Therefore, no deadlock satisfies.

**For No delay:**

Assume there is one process Pi is in the CS, and other processes are in the door way or Bakery. After Pi gets out from the CS, it must set number[i] to zero before goes to reminder. Therefore, one of the other processes (let’s say Pk) with smallest (number[k], k) can access the CS directly because number[i] != 0 or (number[j],j) < (number[k], k) must be false to this process. Therefore, if one process is in the reminder, it cannot delay other processes, then the other processes can access the CS all the time if the CS is empty, so No Delay is satisfied.

**II. What will be the outcome of eliminating the choosing [] array from the Bakery’s Algorithm code? Explain. What condition is violated? Give the sequence that violates the condition.**

**Explanation:**

The reason for choosing is to prevent the while loop in Bakery from being entered when a process is setting its number (let’s process j with number[j]). Note that if the loop is entered and then process j reaches line: “number[i] = 1 + max(number[1], …, number[N])”, one of two situations arises. Either number[j] has the value 0 when the first test is executed, in which case another process (let’s say process i with number[i]) moves on to the next process, or number[j] has a non-zero value, in which case at some point number[j] will be greater than number[i] (since process i finished executing line: “number[i] = 1 + max(number[1], …, number[N])” before process j began). Either way, process i will enter the critical section before process j, and when process j reaches the while loop, it will loop at least until process i leaves the critical section. If there is no choosing [] array, then mutual exclusion is violated.

**Code:**

**L1: While(true) {**

**L2: number[i] = 1 + max(number[1], …, number[N]);**

**L3: for (int j = 1; j <= N; j++) {**

**L5: while(number[j] <>0 and (number[j],j) < (number[i], i)) {}**

**}**

**L6: Critical Section**

**L7: number[i] = 0;**

**}**

**Sequence:**

Suppose we have two processes just beginning; call them P1 and P2. Both reach L2: “number[i] = 1 + max(number[1], …, number[N])” at the same time. Now, we'll assume both read number[1] and number[2] before either addition takes place. Let P2 complete the line first, assigning 1 to number[2], but P1 blocks before the assignment (so number[1] is 0 at this point). Then P2 gets through the while loop at Bakery and enters the CS (because number[1] is 0 and (number[2],2) == (number[2], 2)). P1 unblocks, and assigns 1 to number[1] at L2: “number[i] = 1 + max(number[1], …, number[N])”.; then it proceeds to the while loop in Bakery. When it goes through that loop for j = 2, the condition on number[j] != 0 is true. Further, the condition on (number[j],j) < (number[i], i)) is false, so P1 enters the CS. Now P1 and P2 are both in the critical section, violating mutual exclusion.

P1 & P2: number[i] = 1 + max(number[1], …, number[N]); //P1 and P2 reach this line at same time, but both of them just read number[1] and number[2] before either addition takes place

P2:number[2] = 1 + max(number[1], …, number[N]); //finish the line, so number[2]=1

P2: for (int j = 1; j <= N; j++)

P2: while(number[1] != 0 and (number[1],1) < (number[2], 2)) {} //when j =1, number[1] != 0 is false, so skip

P2: while(number[2] != 0 and (number[2],2) < (number[2], 2)) {} //when j =2, number[2] != 0 is true, but (number[2],2) < (number[2],2) is false, so skip

P2: CS //P2 enters the CS

P1: number[1] = 1 + max(number[1], …, number[N]); //finish the line, and number[1]=1

P1: for (int j = 1; j <= N; j++)

P1: while(number[j] != 0 and (number[j],j) < (number[1], 1)) // when j =1, number[1] != 0 is true, but (number[1],1 ) < (number[1],1) is false so skip

P1: while(number[j] != 0 and (number[j],j) < (number[1], 1)) // when j =2, number[2] != 0 is true, but (number[2],2 ) < (number[1],1) is false so skip

P1: CS //P1 enter the CS as well

As a result, both of P0 and P1 enter the CS, mutual exclusion is not satisfied.

**III. Implement Mutual Exclusion for n processes using hardware support, and the following atomic instruction:**

**CSW (a, b, c) {**

**if ( a == b) {**

**b = c;**

**Sign = 1;**

**}**

**else {**

**a = b;**

**Sign = 0;**

**}**

**return sign**

**}**

**Answer:**

Initially set a = 0, b = 1, and c = 1

**The code1:**

While (true){

……

While(CSW(a, b, c) == 1){}

Critical Section;

a = 0;

……

}

**The code2:**

While (true){

……

While((a==1) || (CSW(a, b, c) == 1){}

Critical Section;

a = 0;

……

}

**Determine a, b, and c**

If we assign two identical number to b and c, and we assign a different number to a, then the atomic instruction should work by mutual exclusion. In my case, I choose a=0, and b=c=1.

**Proof mutual exclusion is satisfied (use code 1)**

Assume there is a processes P1, and P1 is in the CS. Because P1 is in CS, “a” must be 1. So if any other process want to enter the CS, they will block because a==b, and CSW(a, b, c) must return 1. After P1 gets out from the CS, it sets a = 0 immediately, and other processes are allowed to enter the CS; and the first process which entered CSW (a, b, c) will set “a” to 1 again. Therefore, there is only one process can access the CS at a time, Mutual Exclusion is satisfied.

**Solution of inefficiency (use code 2)**

In order to solve the inefficiency problem, we add “a==1” into the while statement as code 2 shows. When “a” is equal to 0, the process must check the CSW(a, b, c). However, if “a” is equal to 1 which means there is a process in the CS, the other process which attempts to enter the CS will block and does not need to check the CSW(a, b, c).