CS 6868: Homework 1

Submitted by: **Mahesh Gupta (CS12M018)**

---

1. As all the read and write operations are atomic, and A[i] values are assigned independently. The operations performed are

A[i] := 1
B[i] : = A[(i-1) mod N]

So, following observations can be made.

- As all the A[i] values are initialized independently and their values does not depend on the order of execution of corresponding process execution order, so all the A[i] values will be initialized with "A[i] = i" value.

- As the B[i] value depends on A[(i-1) mod N] value, which is written by other process, If a process Pi executed first then A[i] value is initialized and B[i] is initialized with random value, but it also means that B[i+1] value will be initialized with expected value. Hence, generalizing it, out of N processes at least (N-1)/2 processes will assign the A[(i-1) mod N] value to corresponding B[i]th element. And at most all the N processes will assign will the A[(i-1) mod N] value to corresponding B[i]th element.

2. For the puzzle two initial states are possible.

a. When the computer is initially turned off.

Initially, all the students can elect a leader among themselves who knows total number of students in the class. Now the protocol they can have among themselves is:

i. Any student other than the leader, if taken to the room and if computer is turned OFF and they have never turned it ON before they will turn it on. Otherwise they will leave the system as it.

ii. Leader student if taken to the room, if finds the computer to be turned ON then he turns it OFF and increments the count value. The count is used to keep track of at least how many students have visited the cabin.

Now when the count = N-1 (N-1 non leader students and 1 leader himself) then all the students have visited the cabin at least once and the leader can now claim that all students have visited at least once.

b. When the computers initial state is not known.

The procedure remains same as situation an exception some minor changes.

i. All students except the leader can turn on the system not once but twice if it is switched OFF.

ii. Leader also counts till 2*N-1 times computer being turned OFF.

In the second case, the computer can be initially turned ON or OFF. If the system is turned OFF initially then it becomes same as case a. and hence the solution works.

If the system is initially turned ON then no one except the leader can turn it OFF and counts it as one student. So, to do the payoff for this, if we count till N times to get the solution. But if system was initially turned OFF then count never reaches N value and we are stuck waiting for nth count. And program never terminates. Hence, to avoid this, we allow each student to add 2 to count and we count till 2*N-1. And hence the issue is automatically resolved.

3. Dining Philosopher Problem's Solution

Let all the philosophers be numbered starting from 1,2,..N starting from any philosopher and going in clockwise direction such that 1 and N are also sitting next to each other.

Now, philosopher 1 can pick up the two chopsticks placed next to him, similarly philosopher 3 can also pick up the chopstick and similarly every next 2nd philosopher in cyclic order picks up the chopstick if they can. (To pick up the chopstick, each philosopher initially picks up the right chopstick and only after picking the right one he picks the left one if right pickup was successful. If a clash occurs then it will be only to pick up the right chopstick as it is clockwise order.)

Next they all keep their spoon back to original place.

Now starting from 2nd philosopher, the process starts

And then from 3rd philosopher and so on..

In each step at least (N-1)/2 philosophers can eat. And then in next round another (N-1)/2 philosophers and so on..

Deadlock avoidance: Since in each step, at least one philosopher can pick up the chopstick if they can, and at a time either both the chopstick is picked up or none (as the order is right chopstick then left chopstick while it is cyclic order) and hence either philosopher can pick up both chopsticks or cannot pickup any. Hence there is no infinite waiting and hence deadlock does not exists.

Starvation avoidance: Since in each step, if one philosopher doesn't get the chance then in next turn he will surely get the chance and can then proceed. Also if a philosopher leaves the chopstick after finishing his turn then also eventually he will get it after certain finite number of steps. Hence starvation never occurs.

Thus, the solution provided is deadlock and starvation free solution.

4. The problem 2 is a situation quite similar to map reduce program execution on multi-processor systems. Multiple processes, which are unaware of each other, process the map operation and until they all have finished the map operation they cannot proceed for the reduce operation. In this event, only the central leader computer who distributed the work knows how many other processors are working on the same problem and keeps track of them.

I believe that as a part of concurrent programming course, we should learn about how to design an algorithm to handle concurrent processes and their execution when they all are working without other processes being aware of each other's presence. This gives system freedom to work as fault tolerant unit. Thus the problem 2 gives the idea of concurrent programming at a bigger scope where different processes are executing in different machines at a larger scope.