Hello everyone, This video is to show my research of COMP3151 Assignment2. This research focus on a linear wait solution of concurrency and also a simple method to solve the problem of Lamport’s Concurrency Programming with java 11.

For Lamport’s Bakery Algorithm, it solved the mutual exclusion problem of size of n processes and meets the three key points mutual exclusion, deadlock freedom, fairness when processes are mutually exclusive. However, the Bakery algorithm also has some practical drawbacks.

first is that always exists process in the critical section, other process which is applying to enter the critical section will change itself to the maximum number of the array plus one , as the process is quickly, this number is unbounded size.

Second,

Many shared variables are used. Two shared arrays have 2N shared variable, which increases the amount of information exchanged between processes in a distributed environment.

What is more, the failure of any of the process will lead all other process stop, in other words, it is unreliable.

Therefore, this research will focus on an algorithm provided by *Boles/aw* ***K.*** *Szymanski*

In 1988 which is a simple solution to Lamport’s Concurrent algorithm problem with linear wait. The algorithm satisfy mutual exclusion, Strongest fairness and will not existed deadlock. There are two method to Implement the algorithum. First is use five distinct values which can record shared memory or store in each process. It also can implement using three boolean variables. Each thread which want to entry to the critical section will write at most 4 times the index of p.

This research will show that implement with sharing memory with five distinct value.

Suppose the five distinct value is 0, 1, 2, 3 and 4. When the process in state 0, denoting that the owner process is in the noncritical section. The process may not start or exit from critical section. Status 0 will not influence other process.

When the process in state 1, it indicating that the owner process wants to enter its critical section. If the Door\_in is open, the process have right to go to waiting room. If is close, it is possibly that other process is checking whether there are any process in prelog, when finding this process, it will open the door\_in and let it in. The other poiibility is that the waiting room is not allowed to come in because process in waiting room is entry to the critical section in order.

When the process in state 2, it showing that the process had been arrived in waitng room and open Door\_in and wait other process in state 1 to get throught the door\_in.

When the process in statue 3, it donotes that the owner process has just passed the door\_in, after that it will check whether exists some other process in state 1. If it is true, go to state 2, open the door\_in and wait other state 1 into waiting room. If is false, ensure door\_in close to prevent process come into waiting room and open door\_out, allow process in waiting room entry to the critical section in order.

When the process in state 4, as explanation above, it open the door\_out and process can entry critical section.

Now, the process go outside the waiting room, it may be clocked whether the process which has less index than itself also queue in critical section. Until all before process status equal 1 or 0, it can pass into the critical section.

After exiting from critical section, to avoid that some process which behind it not go outside the waiting room, some other progress run which not follow by the linear wait. Wait all process’s state equal 4 which behind it, and go to noncritical section,

Now let me introduce my code to implement the algorithm. This algorithm is rigorous, but in the process of implementation, it is necessary to traverse the list to detect the status of other processes. As the process size increases, when a program checks states[i] is suiable for the requestment, AFTER the subsequent process of checking, WE CANNOT EXSURE THAT whether some change happen or keep original. for example, at the same time, two value find all status is less than 3, both of them change their statue to 3 together, but we only allow that one process in state 3 to check the status and make decision of door\_in and door\_out, it will lead mistake. We use the Semaphore function of java. When the process wants to change the value or view the value, it needs to apply first. Only one program can access or change the value of the status at a time to avoid the above problems。

First is the monitor class, It record the number of process which in the system. In addition, I set a flag array which size is equal to the process size, it record the state for each process. At the same time, I set a Semaphore which only allow one process get. If want change the process, it need to get the semaphore.

Second is the process part. I use wait() and noticeall() function to implement the await part. If the process state is not suiable for the request, it will wait(). When some process change the value, it will noticeall process to wake up and check whether the new status data is suiable or not.

I use a random number to let process sleep to simulation all process not start together.

When a process go into critical section, I print all status equal 2 , 3 and 4 process, to prove that it doesnot exist the process in waiting list which the index is less than the process in p. And we can check all process in waiting list will go into the critical section by index order.

For a process to enter its critical section, the status of process will set to 4 and any lower index state need equal 0 or 1. If exists any lower index process state larger than 1, the process need to wait all of them exit from critical section and change statue to 0 or 1.

Process could set its state to 4 have two situation, first is directly from statement P21, second is indirectly through a wait in statement P22. Whatever situation is, it must exists that a process change state from 1 to 3 and donnot exist process with state 1. After that the process state change the state to 4, and wake up all of the process in waiting room to enter into critical section. During the period, Door\_in had been blocked, waiting room will not receive any state 1 process until all process in waiting room exit from critical section, the Door\_in open.

The state have four method, 1 to 3 3 to 2 3 to 4 2 to 4, if there are dealock , it means that the states will not changed. Suppose we have such waiting process.

First we need to except 3, because, when the states equal 3, the program will change it to 2 or 4. It is not satify the state value never change.

Suppose the number is 4, we cannot use 2, because if states 4 exist, the state 2 will change to state4, therefore, in this case, it only have state 4 and 1. But the process can go into the critical section which with the smallest index. It is not satify the deadlock.

Suppose the number is 1, as we cannot use 3 , 4, P11 will not blocked, and wil change state to 3. It is not satify.

The last possibility is 2, but when all other process state is 2, it donnot have state 1process. The process jump to P30, and change it to 4 not 2.

Acccording to the above, the final contradiction show thath the group of process waiting foverer donot exist.

Third is proof the algorithm is linear wait. We can see from P31, all process will according to the index order. Althogh a process in the end of the array get in first, it will wait all process in front of them finish.