**Os2 Project**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Name** | **ID** | **Level** | **Department** |
| **1** | منه الله محمد عبدالعليم محمد | 202000937 | **3** | **IS** |
| **2** | ياسمين علي حسين علي ناصر | 202001055 | **3** | **IS** |
| **3** | اميره محمود عبدالله شاكر | 202000161 | **3** | **IS** |
| **4** | ندي محمود سيد علي | 202000995 | **3** | **IS** |
| **5** | عبير صلاح بيومي محمد | 201900472 | **4** | **IS** |
| **6** | رحمه حسن علي حسن | 202000314 | **3** | **IS** |
| **7** | منار محمود احمد جارحي | 202000932 | **3** | **IS** |

**The Dining Philosophers Problem**

**Introduction :**

The dining philosophers problem states that there are 5 philosophers sharing a circular table and they eat and think alternatively. There is a bowl of rice for each of the philosophers and 5 chopsticks. A philosopher needs both their right and left chopstick to eat. A hungry philosopher may only eat if there are both chopsticks available.Otherwise a philosopher puts down their chopstick and begin thinking again.

The dining philosopher is a classic synchronization problem as it demonstrates a large class of concurrency control problems.

Solution pseudocode:

The structure of Philosopheri:

while (true) {

doAction(" unActive");

FirstAccount.lock();

try {

SecondAccount.lock();

try {

doAction("can use two account- acess ")

}

} finally {

SecondAccount.unlock();

}

} finally {

FirstAccount.unlock();

}

**Examples of Deadlock:**

**deadlock** is a situation where the progress of a system is halted as each process is waiting to acquire a resource held by some other process

**Deadlock happen** when each of the Philosophers has acquired his left fork, but can't acquire his right fork, because his neighbor has already acquired it. This situation is commonly known as the circular wait and is one of the conditions that results in a deadlock and prevents the progress of the system.

**How did solve deadlock:**

The primary reason for a deadlock is the circular wait condition where each process waits upon a resource that’s being held by some other process. Hence, to avoid a deadlock situation we need to make sure that the circular wait condition is broken **by making** :All Philosophers reach for their left fork first, except one who first reaches for his right fork.

**Examples of starvation:**

if 2 adjacent philosophers are hungry at the same time. Since the test(i) is checking whether its left and right philosophers are hungry. If it finds that its adjacent one is also hungry. It is kind of a deadlock right? I mean both of them can never eat since their adjacent philosopher's are hungry right ?

**For example:**

 philosopher0 has leftHungry[0] == TRUE and their state is HUNGRY

, and philosopher1 calls pickUp() and their state is then HUNGRY too.

Even if philosopher1 must wait,

(because philosopher2 is EATING),

 philosopher0 sets leftHungry[0] == FALSE,

preventing any adjacent philosophers from simultaneously thinking each

other is hungry.

**How did solve starvation:**

Using fair lock on two accounts , Class called ReentrantLock

If thread asked for lock and was busy it wait until unlock and if another one came asked fot lock it wait until first one do it work

**Explanation for real world application and how did apply the problem:**

We have an account in the bank and we want to Withdraw and deposit

The first we want to withdraw from the account and then we should deposit to the account

How we can do that?

The first we withdraw the money after this process finite we can deposit the money all this done concurrently