**HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY**
**OFFICE FOR INTERNATIONAL STUDY PROGRAMS**

# OPERATING SYSTEMS (LAB)

**Lab 6**

## SYNCHRONIZATION

| | |
|---|---|
| Lecturer: | Mr. Nguyễn Minh Trí |
| Student Name: | Lê Trung Hiếu |
| ID: | 1852365 |

*Ho Chi Minh City, June 1, 2020*

***Problem 1 : Race conditions are possible in many computer systems. Consider a banking system that maintains an account balance with two functions: deposit (amount) and withdraw (amount). These two functions are passed the amount that is to be deposited or withdrawn from the bank account balance. Assume that a married couple shares a bank account. Concurrently, the husband calls the withdraw() function and the wife calls deposit(). Write a short essay listing possible outcomes we could get and pointing out in which situations those outcomes are produced. Also, propose methods that the bank could apply to avoid unexpected results.***

First, for easier to understand, we can assume that the balance in the account is N.

At first glance, when the couple tries to change the bank account balance at the same time, a race condition will occur and it will lead to some unexpected results.

At a closer look, when the husband calls the ***withdraw(amount1)*** function and the wife calls ***deposit(amount2)*** concurrently (the expected balance then is ***N - amount1 + amount2***), there are three possible outcomes:

1. No race condition takes place. One of the couple has committed their action before another one takes place. The balance after that is ***N - amount1 + amount2***. (expected value)

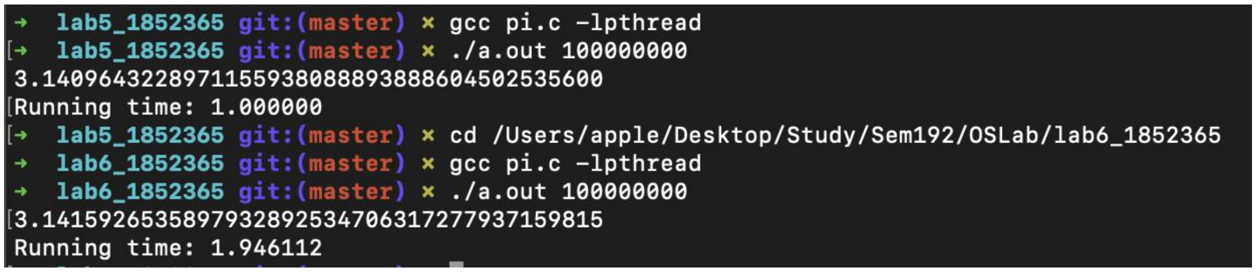
2. The local value of balance for the husband becomes N - amount1 after ***withdraw(amount1)*** is called. After that, the ***deposit(amount2)*** operation takes place and updates the shared value of balance to *N - amount1 + amount2* before the husband completes his action. The system then switches back to the husband and the value of the shared balance is set to ***N - amount1***. (unexpected value)

3. The local value of balance for the wife becomes N + amount2 after the ***deposit(amount2)*** is called. After that, the ***withdraw(amount1)*** operation takes place and updates the shared value of balance to *N + amount2 - amount1* before the wife completes her action. The system then switches back to the wife and the value of the shared balance is set to ***N + amount2***. (unexpected value)

To solve this problem, the can add a lock that protects each bank account. Now, before they can access or update an account balance, cash machines must first acquire the lock on that account.

***Problem 2 : In the Exercise 1 of Lab 5, we wrote a simple multi-thread program for calculating the value of pi using Monte-Carlo method. In this exercise, we also calculate pi using the same method but with a different implementation. We create a shared (global) count variable and let worker threads update on this variable in each of their iteration instead of on their own local count variable. To make sure the result is correct, remember to avoid race conditions on updates to the shared global variable by using mutex locks. Compare the performance of this approach with the previous one in Lab 5.***

Despite taking more time to compile, the one with mutex locks has more accurate value of pi than the remain.



```
➜  lab5_1852365 git:(master) ✗ gcc pi.c -lpthread
[➜  lab5_1852365 git:(master) ✗ ./a.out 100000000
3.140964322897115593808889388860450253560O
[Running time: 1.000000
[➜  lab5_1852365 git:(master) ✗ cd /Users/apple/Desktop/Study/Sem192/OSLab/lab6_1852365
➜  lab6_1852365 git:(master) ✗ gcc pi.c -lpthread
➜  lab6_1852365 git:(master) ✗ ./a.out 100000000
[3.1415926535897932892534706317277937159815
Running time: 1.946112
```