Problem 1: Race condition 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 These two functions are passed the amount that is to be deposited or withdrawn from the bank account balance.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 husband and wife share a bank account. Concurrently, the husband calls the withdraw() function and the wife calls deposit(). Write a short essay listing possible outcomes are produced. Also, propose methods that bank could apply to avoid unexpected results.

Answer: Giả sử số tiền còn lại trong tài khoản là 200.000. Khi người chồng gọi function withdraw (100.000) và người vợ gọi function deposit(100.000), nếu như 2 giao dịch này được hệ thống ngân hàng xử lí tuần tự (xử lí function withdraw trước đến xử lí function deposit sau hoặc ngược lại), thì số tiền trong tài khoản sẽ không đổi (200.000). Tuy nhiên trong trường hợp hệ thống ngân hàng xử lí giao dịch đồng thời, sẽ có khả năng xảy ra lỗi. Người chồng gọi function withdraw, hệ thống xử lí giao dịch thành công nên số tiền là 100.000, bên cạnh đó người vợ gọi function deposit, hệ thống xử lí giao dịch với số tiền trong tài khoản lúc ban đầu (do chưa kết thúc giao dịch withdraw của người chồng nên số tiền không đổi) nên số tiền mới là 300.000. Khi 2 giao dịch kết thúc sẽ dẫn đến sự nhập nhằng trong số tiền => hệ thống sẽ xảy ra lỗi hoặc làm mất cũng như thêm tiền của tài khoản.

Method: Chỉ cho phép một người dùng giao dịch trên tài khoản của ngân hàng (nghĩa là khi người chồng đang thực hiện function withdraw, người vợ sẽ không thực hiện được function deposit và ngược lại). Như vậy ta sẽ đảm bảo số tiền trong tài khoản được cập nhật mới liên tục khi có giao dịch mới đến.

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. Compares the performance of this approach with the previous on in Lab 5.

Answer: So với việc khai báo một biến count cục bộ cho các threads và cập nhật biến count toàn cục sau khi thread kết thúc, việc để các thread cập nhật biến toàn cục trong mỗi vòng lặp sẽ tốn nhiều thời gian hơn do việc cập nhật biến toàn cục chỉ do một thread thực hiện, các thread còn lại sẽ phải chờ đến lượt.