Lab 6 Synchronization
Course: Operating Systems

Problem 1 (5 points): 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 These two functions are passed the amount that is to be deposited or withdrawn from the bank account balance.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 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 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.

Answer: Các trường hợp có thể xảy ra:

- Các quá trình deposit và withdraw diễn ra xen kẽ, không đồng thời, khi đó, các quá trình này diễn ra biền thường.
- Quá trình deposti và withdraw diễn ra cùng lúc, khi đó, xảy ra tranh chấp tài nguyên về khoảng tiền trong tài khoản, điều này dẫn đến sai lệch trong điểu chỉnh khoản tiền và có thể chênh lệch nhiều hơn hoặc ít đi so với lý thuyết.

Để giải quyết tình trạng này, ngân hàng có thể sử dụng các kỹ thuật semaphore, mutex lock hoặc conditional variable để khoá lại tranh chấp (số tiền dư trong tài khoản) khi thực hiện một giao dịch và chỉ mở khoá nó ra khi thực hiện giao dịch thành công, qua đó, không xảy ra race condition.

Problem 2 (5 points): 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.

Answer: So sánh về hiệu năng giữa 2 bài tính pi_multi-thread Lab 5 với Lab 6:

- Lab 5: Sử dụng một mảng các struct để lưu lại giá trị của các tính toán được thực hiện trên mỗi thread, đảm bảo không xảy ra tranh chấp dữ liệu giữa các thread, tốc độ xử lý tương đối nhanh, và có thể cải thiện nếu sử dụng biến kiểu long để tính toán trực tiếp. Tuy nhiên cần sử dụng rất nhiều bộ nhớ và tốc độ tuỳ thuộc vào khả năng tạo ô nhớ trống của hệ điều hành (ở trường hợp của người viết, nếu dùng struct sẽ chậm hơn cách ở Lab 6, nếu đổi thành mảng kiểu long sẽ nhanh hơn với trường hợp 10 000 000 000 số ngẫu nhiên được tạo).
- Lab 6: Sử dụng một biến toàn cục duy nhất để đếm, ít vùng nhớ hơn rất nhiều so với cách ở Lab 5, tuy nhiên cần sử dụng một biến mutex lock để đảm bảo đồng bộ không tranh chấp giữa các thread. Mỗi thread cần phải đợi đến khi biến đếm được nhàn rỗi, tuy nhiên, thời gian thực thi là nhanh hơn so với lab 5 với 10 000 000 000 số tạo ngẫu nhiên.

Với các con số ít hơn, thời gian thực thi của cả 2 cách có thể coi là tương đương.