**BÁO CÁO THỰC HÀNH**

**MÔN: HỆ PHÂN TÁN**

**CHƯƠNG 4: ĐỒNG BỘ HÓA**

**Họ và tên: Lê Thị Yên**

**MSSV: 20183861**

**Mã lớp: 128745**

# Triển khai đồng bộ các luồng trong một chương trình đa luồng sử dụng ngôn ngữ Java.

**Câu hỏi 1:** **Chạy chương trình trên vài lần. Bạn nhận thấy điều gì? Giải thích!**

* Chạy 3 lần:

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Nhận xét: Mỗi lần chạy sẽ có thể ra một kết quả nhỏ hơn hoặc bằng 3000. Với trường hợp chạy bất đồng bộ thì việc biến rsc thay đổi phụ thuộc vào việc truy cập biến rsc của các worker.

**Câu hỏi 2: Thay đổi đoạn mã trong chương trình chạy (phương thức main), thay đổi kiểu của 3 thực thể worker1-3 thành ThreadedWorkerWithSync. Bạn nhận thấy sự thay đổi gì ở đầu ra khi chạy chương trình đó khi so sánh với câu hỏi 1? Giải thích!**

A screenshot of a computer

Description automatically generated

* Do lớp ThreadedWorkerWithSync sử dụng cơ chế đồng bộ, nên tại một thời điểm, biến ***rsc*** chỉ có thể được sử dụng bởi một luồng, nên sau 1000 vòng lặp, mỗi luồng sẽ tăng ***rsc*** lên 1000 đơn vị. Và kết quả luôn là 3000 do có 3 luồng.

**Câu hỏi 3: Thay đổi đoạn code của chương trình chạy chính bằng cách thay thế kiểu của 3 thực thể worker1-3 thành ThreadedWorkerWithLock. Có khác nhau gì so với đầu ra của câu hỏi 1.** **Giải thích!**

Graphical user interface, text

Description automatically generated

* Không có sự khác nhau
* Do biến rsc là tài nguyên găng, khi có 1 thread sử dụng, các thread khác không thể sử dụng biến rsc và phải đợi biến rsc sử dụng xong.

# Lập trình song song với đoạn găng (ngôn ngữ C).

**Câu hỏi 4: Hoàn thiện file trên với một vòng lặp để tăng biến shared lên một đơn vị trong vòng 5 giây.**

Graphical user interface, text, application

Description automatically generated

**Text

Description automatically generated**

Tạo 1 file without-lock.c để mô phỏng một dịch vụ đơn giản của ngân hàng. Chạy với 5 luồng phụ:

Text

Description automatically generated

**Câu hỏi 5: Bây giờ hãy tăng giá trị số luồng và giá trị của số lần giao dịch NUM\_TRANS sau mỗi lần chạy chương trình cho đến khi nào bạn thấy sự khác nhau giữa giá trị Balance (giá trị còn lại trong tài khoản) và INIT\_BALANCE+credits-debits. Giải thích tại sao lại có sự khác nhau đó**

Tăng giá trị luồng lên 500 và giá trị số lần giao dịch NUM\_TRANS lên 200, ta được:

Text

Description automatically generated

Giải thích: Vì có nhiều luồng vào đoạn găng tại 1 thời điểm

**Câu hỏi 6: Hãy build và chạy chương trình này. Chạy lặp đi lặp lại đến bao giờ bạn thấy sự khác nhau giữa 2 giá trị Shared và Expect. Phân tích mã nguồn để hiểu vấn đề**

Text

Description automatically generated

Giải thích: do không đảm bảo các tác vụ sẽ được atomic (có nghĩa là việc nắm giữ lock sẽ bị ngắt)

**Câu hỏi 7: Bây giờ hãy thay đổi đoạn code của file without-lock.c bằng cách triển khai cơ chế mutex lock như trên (bạn có thể tạo file mới và đặt tên khác đi như mutex-lock-banking.c). Chạy chương trình nhiều lần và đánh giá đầu ra. Nó có cải thiện gì hơn so với naive-lock?**

Graphical user interface, text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

* Biên dịch và chạy chương trình nhiều lần:

Text

Description automatically generated

* Chạy với số luồng lớn cũng không có gì bất thường.
* Coarse Locking sẽ khoá chương trình bằng cách sử dụng duy nhất 1 lock cho toàn bộ đoạn găng. Cho nên việc truy cập vào tài nguyên găng sẽ được kiểm soát.

**Câu hỏi 8: so sánh và đo đạt thời gian để chứng minh là Fine Locking sẽ nhanh hơn Coarse Locking**

* File fine-locking-bank.c:

Text

Description automatically generated

* Kết quả khi chạy 2 chương trình mutex-lock-banking.c và fine-locking-bank.c với cùng số luồng phụ là 50000:

Text

Description automatically generated

Thời gian chạy của Fine locking chậm hơn Coarse locking

**Câu hỏi 9: chạy chương trình trên và bạn nhận thấy điều gì? Giải thích thông qua việc phân tích mã nguồn**

Cả 2 đoạn đều truy cập vào tài nguyên găng:

* 1 đoạn truy cập biến a
* 1 đoạn truy cập biến b
* Việc đợi chờ tài nguyên để sử dụng
* Chương trình bị treo