BÀI TẬP TRÊN LỚP

MÔN HỌC: HỆ PHÂN TÁN

CHƯƠNG 4: ĐỒNG BỘ HÓA

HỌ TÊN SV: TRẦN TRUNG PHONG MSSV: 20210676

MÃ LỚP: 149501 MÃ HỌC PHẦN: IT4611

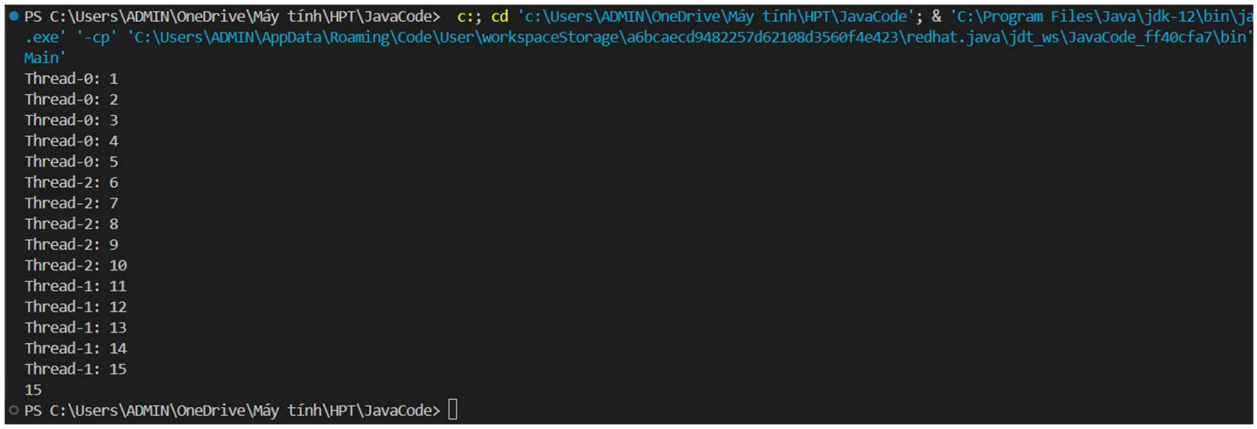
Câu hỏi 1: Chạy chương trình trên vài lần. Bạn nhận thấy điều gì? Giải thích!

```
Thread-0: 7
Thread-1: 3
Thread-1: 8
Thread-1: 9
Thread-1: 10
Thread-1: 11
Thread-2: 1
Thread-2: 12
Thread-2: 13
Thread-2: 14
Thread-2: 15
15
PS C:\Users\ADMIN\OneDrive\Máy tính\HPT\JavaCode>
```

- Đặt lệnh in ra console mỗi khi gọi hàm exploit(), sau một số lần gọi sẽ thấy xuất hiện trường hợp như trên. Các thread chạy không theo thứ tự và kết quả cũng tăng không theo một thứ tự thời gian.

- 3 thread đều truy cập vào resource cùng thời điểm (cụ thể là thời điểm có giá trị là 2) và bỏ qua thời điểm đầu tiên làm cho kết quả cuối cùng là 2867 thay vì là 3000. Tại thời điểm 2 đã xảy ra sự kiện dùng chung tài nguyên trong khi thời điểm ban đầu không thread nào dùng tài nguyên vì đang phải chờ đợi.

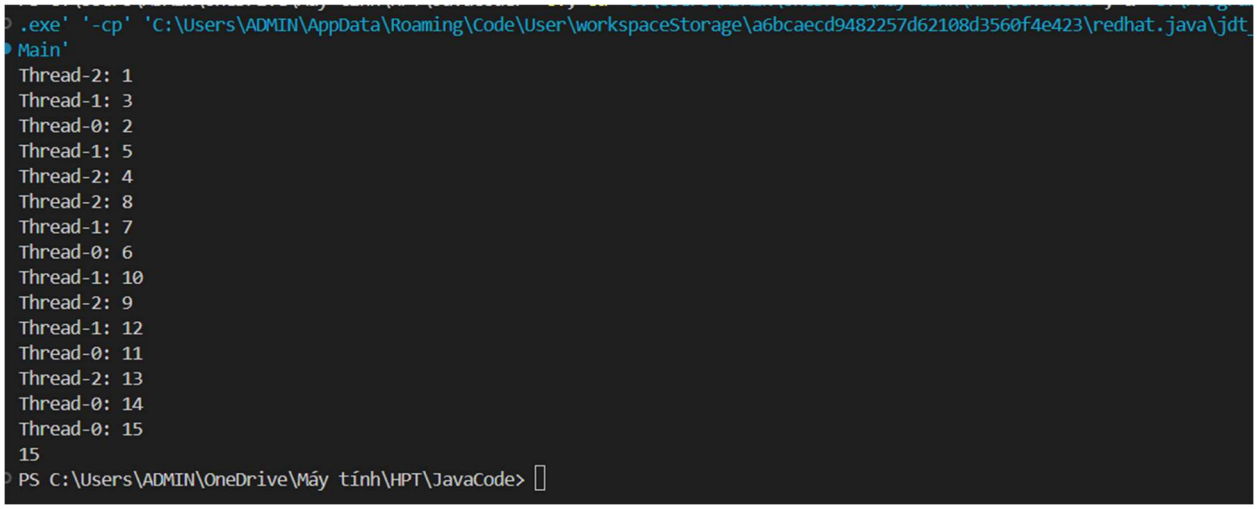
Câu hỏi 2: Thay đổi đoạn mã trong chương trình chạy (phương thức *main*), thay đổi kiểu của 3 thực thể worker1-3 thành *ThreadedWorkerWithSync*. Bạn nhận thấy sự thay đổi gì ở đầu ra khi chạy chương trình đó khi so sánh với câu hỏi 1? Giải thích!



Các luồng thực hiện đúng theo thứ tự và luồng sau chỉ thực hiện khi luồng trước đó đã kết thúc. Các kết quả được in ra tăng liên tục theo thời gian. Kết quả cuối cùng là 3000 và đúng với kết quả của thread chạy sau cùng.

→ Đặt vòng lặp vào trong khối synchronized khiến cho việc thực hiện vòng lặp kết thúc thì các luồng khác mới có thể truy cập vào resource().

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!



- Kết quả luôn luôn ra 3000, không xuất hiện trường hợp như câu 1.
- Thread-0 chạy đầu tiên nhưng kết thúc cuối cùng với giá trị 2867 nhưng kết quả cuối cùng nhận được vẫn là 3000.

→ Mỗi khi hàm exploit() được gọi, lock sẽ được kích hoạt trên resource khiến các thread khác không thể truy cập, sau khi truy cập xong (tăng giá trị lên 1) thì lock sẽ được unlock, lúc này các thread có thể truy cập resource, tuy nhiên ta thấy

thứ tự ngẫu nhiên hơn câu 2 vì ở đây chỉ lock khi gọi hàm exploit() chứ không lock toàn bộ vòng lặp, do đó các thread vẫn truy cập ngẫu nhiên trong toàn bộ quá trình thay vì đợi thread khác hoàn thành như ở câu 2.

Câu hỏi 4: Hoàn thiện file trên (điền vào phần **YOUR-CODE-HERE**) với một vòng lặp để tăng biến shared lên một đơn vị trong vòng 5 giây. (gợi ý: hàm time(NULL) sẽ trả về giá trị thời gian của hệ thống với đơn vị là giây).

```c
void *fun(void *args)
{
    time_t start = time(NULL);
    time_t end = start + 5;

    while (time(NULL) <= end)
    {
        continue;
    }
    shared++;

    return NULL;
}
```

```
PS C:\Users\ADMIN\OneDrive\Máy tính\HPT\c\output> & .\'simple.exe'
shared: 11
PS C:\Users\ADMIN\OneDrive\Máy tính\HPT\c\output> Measure-Command {.\simple}


Days              : 0
Hours             : 0
Minutes           : 0
Seconds           : 5
Milliseconds      : 134
Ticks             : 51341838
TotalDays         : 5.94234236111111E-05
TotalHours        : 0.00142616216666667
TotalMinutes      : 0.08556973
TotalSeconds      : 5.1341838
TotalMilliseconds : 5134.1838
```

Thời gian code thực thi khoảng 5.134s, tức là shared được tăng lên sau khoảng 5s.

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 biệt đó.

```
PS C:\Users\ADMIN\OneDrive\Máy tính\HPT\c\output> & .\'without-lock.exe' 20
         Credits:       0
         Debits:        130000

         50+0-130000=   -129950
         Balance:       -129950
PS C:\Users\ADMIN\OneDrive\Máy tính\HPT\c\output> Measure-Command {.\without-lock 20}


Days              : 0
Hours             : 0
Minutes           : 0
Seconds           : 0
Milliseconds      : 31
Ticks             : 314151
TotalDays         : 3.63600694444444E-07
TotalHours        : 8.72641666666667E-06
TotalMinutes      : 0.000523585
TotalSeconds      : 0.0314151
TotalMilliseconds : 31.4151
```

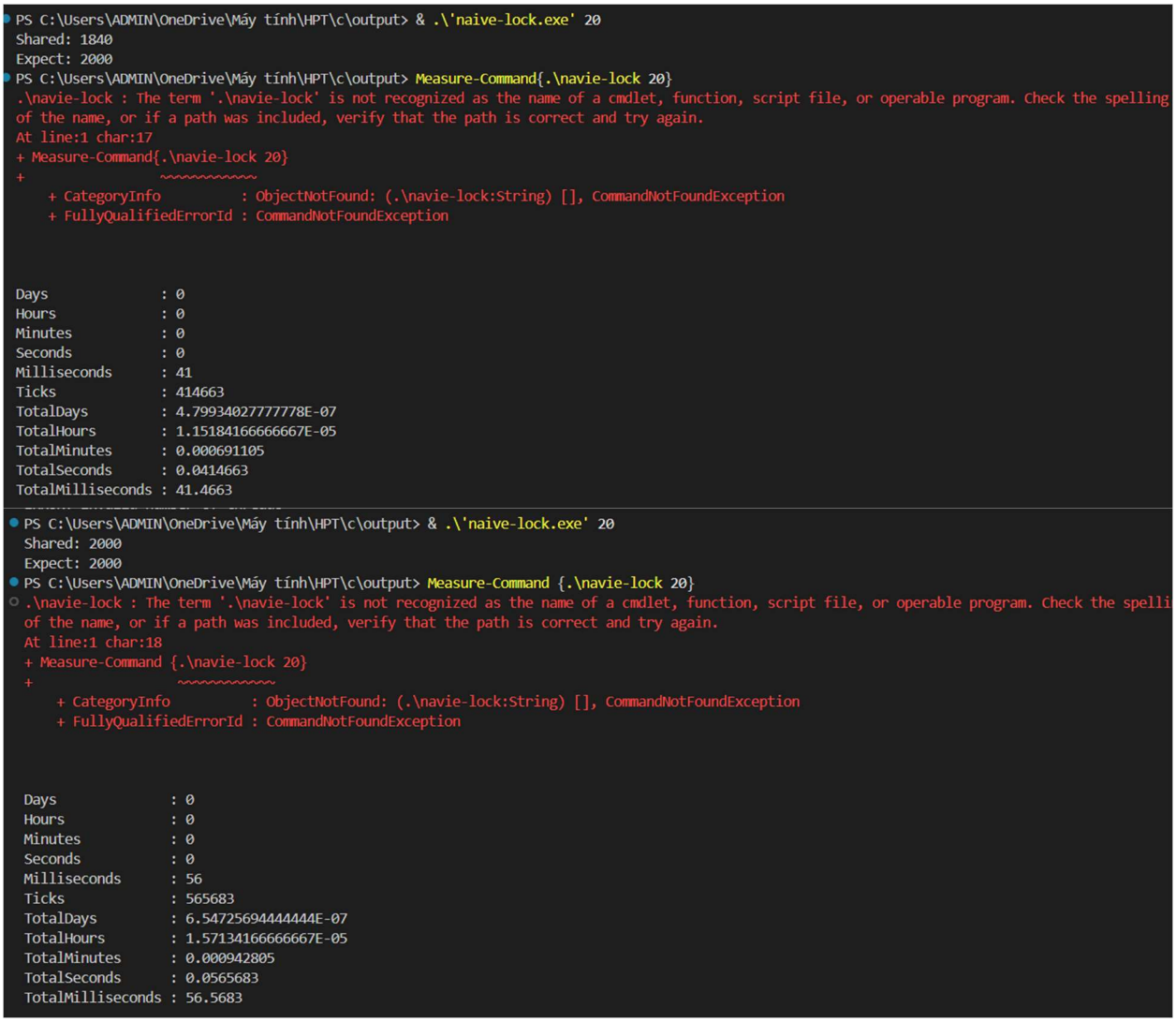
```
PS C:\Users\ADMIN\OneDrive\Máy tính\HPT\c\output> & .\'without-lock.exe' 20
         Credits:       0
         Debits:        18000

         50+0-18000=    -17950
         Balance:       -17950
PS C:\Users\ADMIN\OneDrive\Máy tính\HPT\c\output> Measure-Command {.\without-lock 20}


Days              : 0
Hours             : 0
Minutes           : 0
Seconds           : 0
Milliseconds      : 10
Ticks             : 100521
TotalDays         : 1.1634375E-07
TotalHours        : 2.79225E-06
TotalMinutes      : 0.000167535
TotalSeconds      : 0.0100521
TotalMilliseconds : 10.0521
```

Chạy một vài lần ta thấy 2 giá trị trên sai lệch nhau khá nhiều lần. Do các luồng có khả năng truy cập tài nguyên vào cùng một thời điểm, mặt khác sử dụng hàm random để thay đổi giá trị nên có một thời điểm nào đó khiến ít nhất 2 luồng cùng truy cập vào tài nguyên và thay đổi giá trị theo cách khác nhau nên dẫn đến sai lệch giữa 2 giá trị trên.

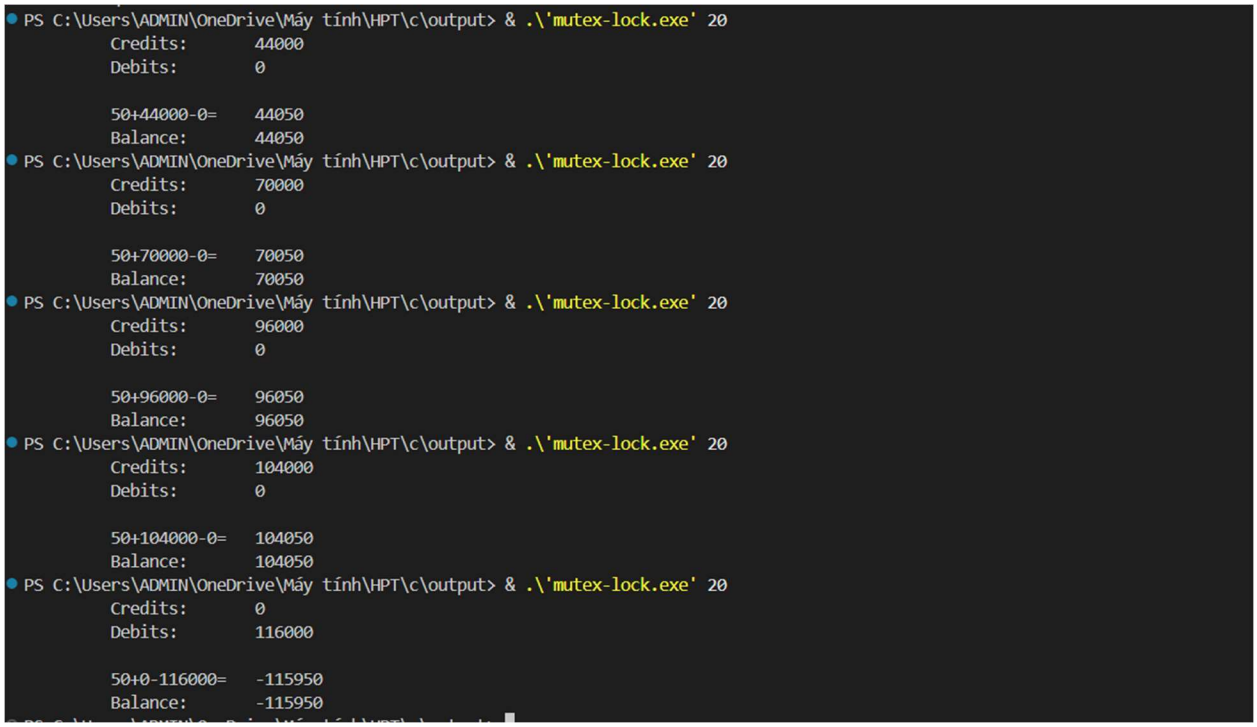
Câu hỏi 6: Hãy build và chạy chương trình này. Chạy lặp đi lặp lại đến bao giờ bạn thấy sự khác nhau giữa 2 giá trị *Shared* và *Expect*. Phân tích mã nguồn để hiểu vấn đề.



Kể cả khi có lock thì thỉnh thoảng vẫn sẽ có sai khác giữa 2 giá trị Shared và Expect.

→Có một thời điểm nào đó mà khi lock = 0, tức là đang unlock, lúc đó có một số thread đồng thời kiểm tra điều kiện của vòng lặp while và đều có thể truy cập được vào resource, khi đó sẽ xảy ra trường hợp tương tự như câu 1 và câu 5.

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?



Gần như không xảy ra trường hợp sai khác giữa kết quả mong muốn và kết quả thực tế.

Câu hỏi 8: So sánh và đo đạc thời gian để chứng minh là Fine Locking sẽ nhanh hơn Coarse Locking.

```
PS C:\Users\ADMIN\OneDrive\Máy tính\HPT\c\output> Measure-Command {.\fine-lock 20}


Days              : 0
Hours             : 0
Minutes           : 0
Seconds           : 0
Milliseconds      : 15
Ticks             : 152632
TotalDays         : 1.76657407407407E-07
TotalHours        : 4.23977777777778E-06
TotalMinutes      : 0.000254386666666667
TotalSeconds      : 0.0152632
TotalMilliseconds : 15.2632
PS C:\Users\ADMIN\OneDrive\Máy tính\HPT\c\output> Measure-Command {.\mutex-lock 20}


Days              : 0
Hours             : 0
Minutes           : 0
Seconds           : 0
Milliseconds      : 16
Ticks             : 163730
TotalDays         : 1.89502314814815E-07
TotalHours        : 4.54805555555556E-06
TotalMinutes      : 0.000272883333333333
TotalSeconds      : 0.016373
TotalMilliseconds : 16.373
```

Fine- lock nhanh hơn mutex-lock.

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

```
PS C:\Users\ADMIN\OneDrive\Máy tính\HPT\c\output> cd 'c:\Users\ADMIN\OneDrive\Máy tính\HPT\c\output'
PS C:\Users\ADMIN\OneDrive\Máy tính\HPT\c\output> & .\'deadlock.exe'
PS C:\Users\ADMIN\OneDrive\Máy tính\HPT\c\output> Measure-Command {.\deadlock}
PS C:\Users\ADMIN\OneDrive\Máy tính\HPT\c\output>
```

Sau hơn 10s, chương trình vẫn không in ra màn hình bất cứ gì (phải dùng Ctrl-C để thoát).

→ `thread_1` gọi `func1` khóa a trước, `thread_2` gọi `func2` lại khóa b trước, khi đó việc `thread_1` khóa b và `thread_2` khóa a thất bại và phải chờ lẫn nhau mở khóa còn lại, lúc này sẽ xảy ra `deadlock` và không có `thread` nào thực thi tiếp.