CHƯƠNG 6: Đồng bộ hóa

Họ tên sinh viên: Phan Thành Đạt **MSSV:20173001**

Mã Lớp: 118636 **Mã học phần: IT4611**

Câu 1:

Chạy nhiều lần nhưng mỗi lần lại ra 1 kết quả khác nhau. Bởi vì cả 3 luồng đều ghi chung 1 biến resource mà không có cơ chế loại trừ nên trong 1 thời điểm có thể có nhiều hơn 1 luồng cùng ghi vào biến này và biến resource có thể chỉ được tăng lên 1 lần mặc dù có nhiều hơn 1 luồng cùng tăng biến đó.

Câu 2:

Sau khi thay đổi code, cho dù chạy nhiều lần thì vẫn ra 1 kết quả là 3000 bởi vì hàm **`synchronized`** đã thực hiện đồng bộ hóa biến resource bằng cách loại trừ, trong 1 thời điểm chỉ có 1 luồng được phép ghi vào biến đó.

Câu 3:

Sau khi thay đổi code, cho dù chạy nhiều lần thì kết quả vẫn ra giống nhau là 3000 bởi vì hàm

```
if (lock.tryLock(10, TimeUnit.SECONDS)) {
                    setRsc(getRsc() + 1);

                }
```

thực hiện khóa quá trình ghi vào biến resource, trong khoảng thời gian tối đa 10 giây thì chỉ có 1 luồng được phép truy cập (nếu trước 10 giây hàm thực hiện xong thì cũng tự động được mở khóa).

Câu 4:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <pthread.h>
#include <time.h>
int shared = 10;
void * fun(void * args){
 time_t start = time(NULL);
 time_t end = start+5; //run for 5 seconds
 while (time(NULL)<end){
      shared++;
 }
 return NULL;
}
```

```c
int main(){
    pthread_t thread_id;
    pthread_create(&thread_id, NULL, fun, NULL);
    pthread_join(thread_id, NULL);
    printf("shared: %d\n", shared);
    return 0;
}
```

Câu 5:

Khi tăng số luồng và thay đổi giá trị NUM_TRANS và ta chạy nhiều lần thì mỗi lần có thể sẽ ra kết quả khác nhau giữa Balance và INIT_BALANCE+credits-debits bởi vì có tài nguyên chia sẻ là các biến balance, credits và debits không được áp dụng bất cứ phương pháp loại trừ nào, nên có thể trong cùng 1 thời gian sẽ có nhiều hơn 1 luồng truy cập đồng thời và ghi vào biến balance nhưng biến credits lại được truy cập lần lượt (tức là biến balance thì chỉ được tăng 1 giá trị vì bị truy cập đồng thời còn biến credits thì lại được truy cập lần lượt và được tăng 2 giá trị) và ngược lại đối với biến debits, dẫn đến kết quả sai.

Câu 6:

Sự khác biệt giữa biến Shared và Expect ở đây là do mặc dù biến shared đã được loại trừ rồi nhưng biến lock lại chưa được áp dụng các phương pháp loại trừ, có thể trong cùng 1 thời gian có nhiều hơn 1 luồng cùng đọc và ghi vào biến lock, dẫn đến biến shared sẽ bị nhiều hơn 1 luồng truy cập để ghi và từ đó dẫn đến sai số.

Câu 7:

```c
#include <pthread.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#define INIT_BALANCE 50
#define NUM_TRANS 100
int balance = INIT_BALANCE;
int credits = 0;
int debits = 0;
pthread_mutex_t mutex;
void * transactions(void * args){
    int i,v;
    for(i=0;i<NUM_TRANS;i++){
        //choose a random value
        srand(time(NULL));
        v = rand() % NUM_TRANS;
        //randomnly choose to credit or debit
```

```
        pthread_mutex_lock(&mutex);
        if( rand()% 2){
            balance = balance + v;
            credits = credits + v;
        }else{
            balance = balance -v;
            debits = debits + v;
        }
        pthread_mutex_unlock(&mutex);
    }
    return 0;
}
int main(int argc, char * argv[]){
    int n_threads,i;
    pthread_t * threads;
    if(argc < 2){
        fprintf(stderr, "ERROR: Require number of threads\n");
        exit(1);
    }
    n_threads = atol(argv[1]);
    if(n_threads <= 0){
        fprintf(stderr, "ERROR: Invalivd value for number of threads\n");
        exit(1);
    }
    pthread_mutex_init(&mutex, NULL);
    threads = calloc(n_threads, sizeof(pthread_t));
    for(i=0;i<n_threads;i++){
        pthread_create(&threads[i], NULL, transactions, NULL);
    }
    for(i=0;i<n_threads;i++){
        pthread_join(threads[i], NULL);
    }
    printf("\tCredits:\t%d\n", credits);
    printf("\t Debits:\t%d\n\n", debits);
    printf("%d+%d-
%d=   \t%d\n", INIT_BALANCE,credits,debits,INIT_BALANCE+credits-debits);
    printf("\t Balance:\t%d\n", balance);
    free(threads);
    return 0;
}
```

Cho dù có chạy bao nhiêu lần đi chẳng nữa và với bao nhiêu luồng thì kết quả các lần đều ra giống nhau, bởi vì các tài nguyên được càng luồng dùng chung là biến balance, credits và debits

đã được hàm pthread_mutex_lock(&mutex); khóa lại, đảm bảo trong 1 thời điểm chỉ có duy nhất 1 luồng sử dụng.

Câu 8:

```c
#include <pthread.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#define INIT_BALANCE 50
#define NUM_TRANS 100
int balance = INIT_BALANCE;
int credits = 0;
int debits = 0;
pthread_mutex_t mutex;
pthread_mutex_t b_lock,c_lock,d_lock;
void * transactions(void * args){
    int i,v;
    for(i=0;i<NUM_TRANS;i++){
        //choose a random value
        srand(time(NULL));
        v = rand() % NUM_TRANS;
        //randomnly choose to credit or debit
        if( rand()% 2){
            // pthread_mutex_lock(&mutex);
            pthread_mutex_lock(&b_lock);
            balance = balance + v;
            pthread_mutex_unlock(&b_lock);
            pthread_mutex_lock(&c_lock);
            credits = credits + v;
            pthread_mutex_unlock(&c_lock);
            // pthread_mutex_unlock(&mutex);
        }else{
            // pthread_mutex_lock(&mutex);
            pthread_mutex_lock(&b_lock);
            balance = balance -v;
            pthread_mutex_unlock(&b_lock);
            pthread_mutex_lock(&d_lock);
            debits = debits + v;
            pthread_mutex_unlock(&d_lock);
            // pthread_mutex_unlock(&mutex);
        }
    }
    return 0;
```

```
}
int main(int argc, char * argv[]){
    int n_threads,i;
    pthread_t * threads;
    if(argc < 2){
        fprintf(stderr, "ERROR: Require number of threads\n");
        exit(1);
    }
    n_threads = atol(argv[1]);
    if(n_threads <= 0){
        fprintf(stderr, "ERROR: Invalivd value for number of threads\n");
        exit(1);
    }
    pthread_mutex_init(&mutex, NULL);
    pthread_mutex_init(&b_lock, NULL);
    pthread_mutex_init(&c_lock, NULL);
    pthread_mutex_init(&d_lock, NULL);
    threads = calloc(n_threads, sizeof(pthread_t));
    for(i=0;i<n_threads;i++){
        pthread_create(&threads[i], NULL, transactions, NULL);
    }
    for(i=0;i<n_threads;i++){
        pthread_join(threads[i], NULL);
    }
    printf("\tCredits:\t%d\n", credits);
    printf("\t Debits:\t%d\n\n", debits);
    printf("%d+%d-
%d=   \t%d\n", INIT_BALANCE,credits,debits,INIT_BALANCE+credits-debits);
    printf("\t Balance:\t%d\n", balance);
    free(threads);
    return 0;
}
```

Đoạn code sau khi tinh chỉnh sẽ chạy nhanh hơn nhiều do ta sử dụng **Fine Locking.** Luồng nào ghi vào biến dùng chung nào thì sẽ chỉ lock biến đó lại.

Câu 9:

Chương trình trên sẽ xảy ra bế tắc nếu luồng thứ nhất có được lock_a và luồng thứ 2 có được lock_b hoặc ngược lại bởi vì luồng thứ 2 sẽ đợi lock_a được mở và luồng thứ nhất sẽ đợi lock_b được mở để chạy, 2 luồng đều đợi nhau dẫn đến chương trình bị deadlock