LAB 6: SYNCHRONIZATION

Trần Quốc Thái - 2010616
Võ Hùng -2013375
Đào Đức Thiện -1713287
Bùi Hoàng Minh - 2010410

Question 1: Listing possible outcomes we could get and pointing out in which situations those outcomes are produced. Then propose some methods that the bank could apply to avoid unexpected results.

Cụ thể tài khoản ngân hàng là tài nguyên được chia sẻ dùng chung cho 2 quá trình, ở đây là quá trình người chồng rút tiền và người vợ thêm tiền vào. Bài toán tương tự mô hình của bài toán producer - consumer.

Gọi tài khoản dùng chung là V, hành động rút tiền của người chồng là A, hành động thêm tiền vào của người vợ là B, khi đó các khả năng có thể xảy ra:
+ A, B xảy ra xen kẽ, hoặc liên tục không cùng lúc sử dụng V tại 1 thời điểm, ở đây các quá trình này diễn ra bình thường.
+ Khi A và B cùng lúc sử dụng V tại 1 thời điểm, xảy ra sự tranh chấp tài nguyên, dẫn đến sai sót, sai lệch hoặc lỗi trong quá trình thực hiện A, B, dẫn đến trạng thái tài khoản ngân hàng V có sự sai sót, có thể nhiều hơn hoặc ít hơn so với lý thuyết.

Để giải quyết tranh chấp trên, ta có thể sử dụng kỹ thuật mutex lock, hoặc semaphore để xử lý các quá trình đồng bộ trên.

```
static volatile int money;
pthread_mutex_t lock;

void *in() {
      while (true) {
            Check_if_deposit() {
                  pthread_mutex_lock(&lock);
                  Read_amount();
                  money += deposit(amount);
                  pthread_mutex_unlock(&lock);
            }
      }
}
void *out() {
      while (true) {
            Check_if_deposit() {
                  pthread_mutex_lock(&lock);
                  Read_amount();
                  money -= withdraw(amount);
                  pthread_mutex_unlock(&lock);
            }
      }
}
```

```
int main() {
      pthread_t t1,t2;
      pthread_create(&p1, NULL, in,);
      pthread_create(&p2, NULL, out,);
}
```

Question 2: How race conditions can be controlled in high level programming languages like Java or Python.

Trong Python, race condition có thể được giải quyết bằng việc sử dụng lớp Lock của thư viện threading. Lớp Lock được hiện thực bằng cách dùng Semaphore của hệ điều hành và gồm 2 phương thức chính:

- acquire(): khóa
- release(): mở khóa

```
lock.acquire()
//code
lock.release()
```

Trong java, java cung cấp một cách để tạo đa luồng và đồng bộ hóa các thread nó tạo ra bằng cách sử dụng synchronized blocks.

- Sử dụng phương thức *.join() của thread để đảm bảo việc thread có thể chạy cho đến khi kết thúc phần việc của mình mà không bị xung đột với các thread khác
- Một điều có thể xảy ra khi sử dụng đa luồng trong java đó là xung đột phương thức (2 hay nhiều luồng có thể gọi đến 1 phương thức cùng một thời điểm tạo ra kết quả cuối bị sai) . Để giải quyết việc đó, ta có thể dùng động bộ hóa phương thức chỉ cho phép 1 phương thức chỉ có thể truy cập bằng 1 luồng trong cùng 1 thời điểm .
- Syntax để đồng bộ hóa phương thức :

```
public synchronized void my_func(){
      //code hear...
}
```