**Bài tập 3 - Đồng bộ hóa**

**(2312678 – Nguyễn Lê Bảo Long – CTK47A)**

**1) Xét giải pháp đồng bộ sau:**

Dùng biến **lock** để cho phép tiến trình vào CS. Khởi tạo lock=0

Giải pháp này có thỏa mãn các điều kiện của bài toán của bài toán CS?

**Process Pi**

do {

        while (lock != 0);

        lock=1;

        /\*critical section\*/

        lock = 0;

       /\*remainder section\*/

} while (1);

Giải pháp sử dụng biến lock được khởi tạo với giá trị 0 để kiểm soát việc vào vùng găng (Critical Section - CS) có thể được phân tích dựa trên ba điều kiện chính của bài toán vùng găng:

1. **Tính tương hỗ (Mutual Exclusion)**:
   * Trong giải pháp này, nếu một tiến trình Pi kiểm tra thấy lock == 0, nó sẽ đặt lock = 1 và đi vào vùng găng.
   * Khi tiến trình đang trong CS, nếu một tiến trình khác Pj cũng kiểm tra lock, nó sẽ thấy lock == 1, do đó bị chặn lại trong vòng while(lock != 0);.
   * Kết luận: Điều kiện tương hỗ **được đảm bảo**.
2. **Tiến trình tiến (Progress)**:
   * Nếu không có tiến trình nào trong CS và một số tiến trình muốn vào CS, một tiến trình phải được chọn để vào CS.
   * Giải pháp này không có bất kỳ cơ chế ưu tiên hoặc lựa chọn nào giữa các tiến trình đang chờ.
   * Nếu hai tiến trình cùng kiểm tra lock đồng thời và cả hai thấy lock == 0, cả hai có thể đặt lock = 1 gần như cùng lúc (trong hệ thống đa luồng hoặc hệ thống đa xử lý), dẫn đến cả hai có thể vào CS cùng lúc. Điều này phá vỡ điều kiện tương hỗ.
   * Ngoài ra, nếu một tiến trình bị treo sau khi kiểm tra lock == 0 nhưng trước khi đặt lock = 1, thì một tiến trình khác cũng có thể vào CS trước, tạo ra tình huống không công bằng.
   * Kết luận: Điều kiện tiến trình tiến **không được đảm bảo**.
3. **Chờ giới hạn (Bounded Waiting)**:
   * Nếu có một tiến trình muốn vào CS nhưng bị tiến trình khác chiếm giữ CS liên tục (vì không có cơ chế thay đổi thứ tự), tiến trình đó có thể bị chặn vô thời hạn (starvation).
   * Không có bất kỳ cơ chế nào để đảm bảo rằng một tiến trình đang chờ sẽ được vào CS sau một số lần cố gắng hữu hạn.
   * Kết luận: Điều kiện chờ giới hạn **không được đảm bảo**.

=> Giải pháp này **không phải** là một giải pháp đúng cho bài toán vùng găng vì nó không thỏa mãn điều kiện tiến trình tiến và chờ giới hạn.

2) Chứng minh giải pháp Peterson đáp ứng 03 điều kiện của bài toán CS?

**- Độc quyền (Mutual Exclusion)**

Điều này có nghĩa là chỉ một tiến trình có thể vào CS tại một thời điểm. Để chứng minh điều này, ta giả sử cả hai tiến trình P0­ và P1​ đều vào CS cùng một lúc. Khi đó, sẽ có một trong hai trường hợp sau:

* Nếu P0­ ​ vào CS trước P1 ​, thì khi P1 muốn vào CS, biến flag[0] sẽ là 1 và turn sẽ được đặt bằng 1 (theo logic của thuật toán), làm cho P1​ phải đợi cho đến khi P0­ ra khỏi CS. Ngược lại, nếu P1 vào CS trước, P0­ phải đợi.
* Nếu cả hai tiến trình đều muốn vào CS cùng lúc, thì chỉ có một tiến trình vào CS trước, vì **biến turn đảm bảo một tiến trình phải chờ đến lượt của mình**. Do đó, luôn chỉ có một tiến trình vào CS tại một thời điểm.

Vì vậy, **điều kiện độc quyền** được đảm bảo.

**- Tiến trình tiến lên (Progress)**

Điều này có nghĩa là nếu không có tiến trình nào trong CS và có ít nhất một tiến trình muốn vào CS, thì một trong các tiến trình đó sẽ vào CS.

Xét trường hợp nếu không có tiến trình nào trong CS. Khi đó, có một trong hai tiến trình P0­ ​ hoặc P1 sẽ đánh dấu flag[i] = 1 để thể hiện rằng nó muốn vào CS. Sau đó, tiến trình đó sẽ kiểm tra biến turn:

* Nếu turn != i (nghĩa là không phải lượt của tiến trình đó vào CS), tiến trình này sẽ tiếp tục chờ.
* Nếu turn == i, tiến trình đó sẽ vào CS ngay lập tức.

Nếu cả hai tiến trình đều muốn vào CS, thì thuật toán đảm bảo rằng một trong hai tiến trình sẽ vào CS (theo lượt) và tiến trình còn lại phải chờ. Điều này đảm bảo **tiến trình tiến lên** vì ít nhất một tiến trình sẽ vào CS nếu có tiến trình muốn vào.

**- Hạn chế tiến trình chờ lâu (Bounded Waiting)**

Điều này yêu cầu rằng không có tiến trình nào phải chờ quá lâu để vào CS. Ta sẽ chứng minh điều này bằng cách chỉ ra rằng mỗi tiến trình có một giới hạn số lần phải chờ.

Giả sử tiến trình P0­​ muốn vào CS, nhưng tiến trình P1 cũng muốn vào CS và không để P0­​ vào trước. Sau khi P0­​ kiểm tra flag[1] và nhận thấy rằng flag[1] == 1, P0­​ sẽ kiểm tra giá trị của turn. Nếu turn == 1, tiến trình P0­ sẽ phải chờ cho đến khi P1​ ra khỏi CS.

Tuy nhiên, mỗi lần tiến trình P1​ vào CS, tiến trình P0­ chỉ phải chờ một lượt. Mỗi tiến trình chỉ có thể bị chặn bởi tiến trình còn lại tối đa một lần trong mỗi vòng lặp, và thời gian chờ không thể kéo dài vô hạn.

Điều này đảm bảo rằng **không có tiến trình nào phải chờ quá lâu**. Do đó, **điều kiện hạn chế tiến trình chờ lâu** được thỏa mãn.

**Kết luận:**

Giải pháp **Peterson’s Algorithm** đáp ứng đầy đủ ba điều kiện của bài toán Critical Section:

1. **Độc quyền (Mutual Exclusion)**: Chỉ có một tiến trình vào CS tại một thời điểm.
2. **Tiến trình tiến lên (Progress)**: Nếu không có tiến trình nào trong CS và có ít nhất một tiến trình muốn vào CS, một tiến trình sẽ được phép vào.
3. **Hạn chế tiến trình chờ lâu (Bounded Waiting)**: Không có tiến trình nào phải chờ quá lâu để vào CS.

Top of Form

Bottom of Form