<u>Câu 5:</u> (Đinh Công Thành – 2014487)

Tổng quan vấn đề:

Bài toán Reader-Writer là một trong những bài toán cổ điển về tính đồng thời. Nó liên quan đến một tình huống, khi một hệ thống tệp hoặc bất kỳ dữ liệu nào đang được đọc và ghi bởi nhiều luồng đồng thời. Nó là cần thiết để chương trình có thể ngăn việc đọc dữ liệu khi có ghi đè đang diễn ra và ngăn việc đọc làm gián đoạn dữ liệu không hợp lệ đang được sửa đổi. Do đó, điều quan trọng là 2 writer không nhập cấu trúc dữ liệu hoặc cơ sở dữ liệu để bắt đầu viết lại. Chúng ta cần đảm bảo rằng reader và writer thực thi mã khác nhau trước và sau phần quan trọng để có thể đáp ứng các ràng buộc sau:

1. Bất kỳ số lượng readers nào cũng có thể ở trong critical section đồng thời.
2. Writers phải có quyền truy cập độc quyền vào critical section: không writer nào có thể vào khi có một thread khác và trong khi writer ở đó, không một thread nào khác có thể vào critical section

Mô tả giải pháp:

Ngay từ đầu, điều quan trọng là phải khởi tạo các biến sẽ chịu trách nhiệm thực hiện. Em thiết lập một semaphore với biến bộ đếm được khởi tạo cho số readers hiện tại trong critical section. Semaphore sem_t mutex đó ở đó để ngăn các luồng khác sửa đổi biến bộ đếm. Semaphore sem_t wrt thứ hai sẽ ngăn không cho các writers khác tham gia thực thi khi bất kỳ ai ở đó. Semaphore sem_t wIn thứ ba được sử dụng như một cửa quay. Khi chủ đề writer nhập vào, nó sẽ gọi sem_wait trên wIn, vì vậy khi reader đến sau, nó sẽ tự ngủ, điều này cho phép writer không bị reader vượt qua và tránh bị starvation.

Khi writer thoát ra, nó biết rằng không có luồng nào đi vào quá trình thực thi vì nó là luồng duy nhất trong đó bắt đầu. Mã cho reader liên quan đến việc giữ số lượng reader, tăng và giảm nó, khi reader ra vào phòng. Người đầu tiên bước vào và người đầu tiên thoát ra phải gọi sem_wait và sem_post cho wrt để writer biết khi nào vào phòng. Khi reader tiến hành nhập lệnh, nó sẽ ngăn writer vào lệnh thực thi. Nếu reader đến khi phòng đang bận với writer, nó sẽ đợi, giống như writer thứ hai và sẽ có hàng đợi hình thành.

Semaphore thứ ba ở đó để đóng vai trò như một cửa quay. Nó được thêm vào cho reader và cho phép writer khóa và mở khóa nó sau khi hoàn tất, để những reader đã xếp hàng sau đó có thể tiến hành thực thi. Nếu một writer đến cùng với reader trong quá trình hành quyết, đầu tiên nó sẽ khóa cửa quay, sau đó sẽ tự ngủ vì căn phòng không trống. Reader tiếp theo đến nơi sẽ được đưa vào giấc ngủ ngay lập tức vì cửa quay khi đó đã được khóa, ưu tiên writer vào phòng trước. Sau khi tất cả reader rời khỏi quá trình thực thi, writer sẽ ngay lập tức vào, vì cửa quay đã bị đóng sau khi writer và hiện không reader nào có thể vào đó. Khi thực thi, writer thực hiện sem_post trên wIn và bỏ chặn thread tiếp theo là reader hoặc writer. Tuy nhiên, bây giờ nó cho thấy khả năng cho bất kỳ luồng nào (writer hoặc reader) tham gia thực thi.

<u>Câu 6:</u> (Đinh Công Thành – 2014487)

Tổng quan vấn đề:

Một sự vướng mắc của mutex lock là có thể dẫn đến deadlock không? Nó chắc chắn có thể. Chốt lại là điều thứ hai mà cha mẹ bạn nói với bạn về mutexes: nếu một thread đạt được A, sau đó đạt được B trước khi giải phóng A, và thread kia làm tương tự theo thứ tự ngược lại, các luồng có thể bị deadlock. Và chúng sẽ deadlock nếu hai thread đạt được đầu tiên được chọn từ các thread riêng biệt. Đây là mã của hai luồng và chú thích mô tả quá trình thực thi có vấn đề:

```
1  Mutex A,B;
2  void Thread() {
3      B.Readlock();
4      A.Writelock();
5      A.WriteUnlock();
6      B.ReadUnlock();
7  }
```

```
1  Mutex A,B;
2  void Thread() {
3      A.Writelock();
4      B.Readlock();
5      B.ReadUnlock();
6      A.WriteUnlock();
7  }
```

Mô tả giải pháp: Morris's solution

Nó sử dụng hai cửa quay để tạo ra hai phòng chờ trước critical section. Cơ chế hoạt động theo hai giai đoạn.

Trong giai đoạn đầu, cửa quay thứ nhất mở và cửa quay thứ hai đóng, do đó, các thread tích tụ trong phòng thứ hai. Trong giai đoạn thứ hai, cửa quay đầu tiên bị đóng, vì vậy không có thread mới nào có thể vào và cửa quay thứ hai đang mở, vì vậy các chủ đề hiện có có thể đi đến critical section.

Morris's algorithm

```
mutex.wait()
    room1 += 1
mutex.signal()

t1.wait()
    room2 += 1
    mutex.wait()
    room1 -= 1

    if room1 == 0:
        mutex.signal()
        t2.signal()
    else:
        mutex.signal()
        t1.signal()

t2.wait()
    room2 -= 1

    # critical section

    if room2 == 0:
        t1.signal()
    else:
        t2.signal()
```

Mutex nó được sử dụng để bảo vệ biến vùng critical section room 1 và thực hiện một phần quá trình t1.wait() chờ đợi sự xuất hiện của luồng . Câu lệnh chỉ ra rằng miễn là có luồng

trong đó, hãy cho phép luồng đó vào . Và tất cả các luồng đến đều bị "kẹt" (tức là tất cả đang chờ ở). Khi không có luồng mới nào nhập vào (nghĩa là không có luồng nào hiện tại), hãy đóng và mở. Room 2 "mở" tương ứng với một phần , và "đóng" room 1 thực sự sẽ không báo hiệu t1.room1 if room1 room1 room2 room1 room2 t2.wait() room1 room1 room1 room2 t2.signal()

t2.wait() phần này được sử dụng để room 2 xử lý từng luồng trong, vì nó t2 được khởi tạo thành 0 room 1 và room 2 luồng cuối cùng signal tới t2. Giá trị ban đầu của t2 là 0 room2, tại một thời điểm chỉ có một luồng hoạt động trên phần tới hạn.