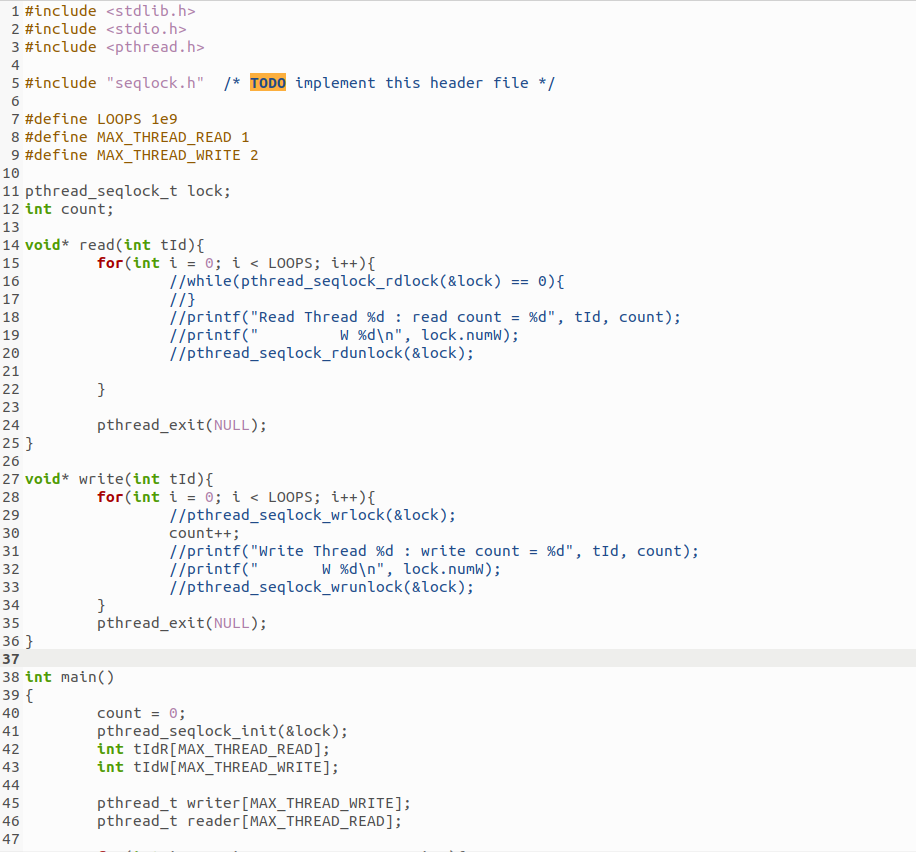
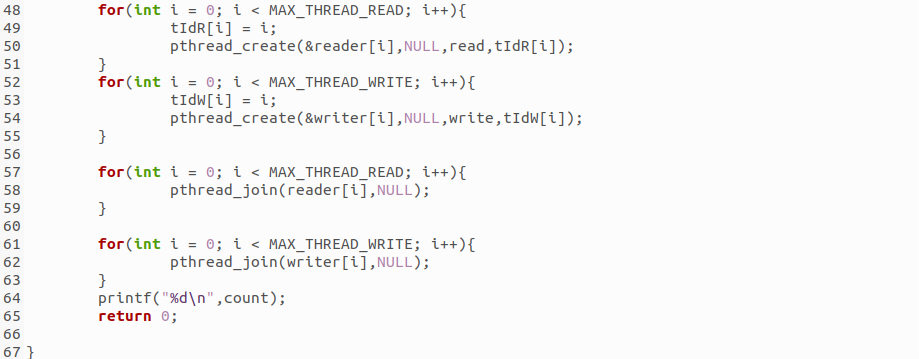
Writer – reader Problem : Khi một process có nhiều thread ( hoặc là khi có nhiều process) cùng modify (hay còn gọi là các writer) đến cùng một vùng nhớ (the same base data) sẽ xảy ra tình huống trong cùng một thời điểm có nhiều hơn một writer trong critical section, làm cho vùng nhớ đó bị lỗi và không ra được kết quả mong muốn. Do đó, với writer – reader problem thì writer là phần có khả năng làm hỏng dữ liệu, gây ra vấn đề, sẽ khiến cho reader đọc ra kết quả không đúng.

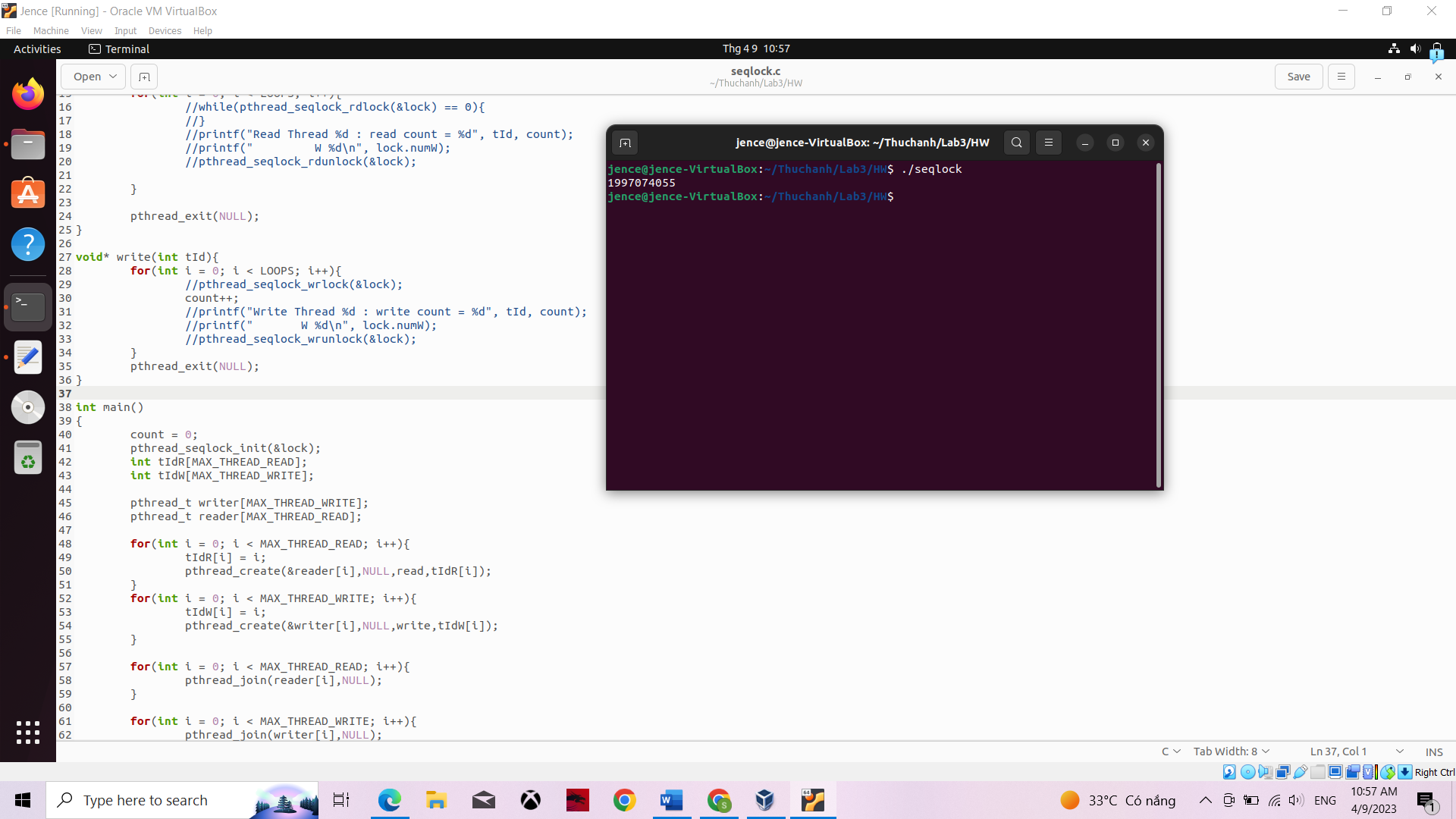
Sau đây là một đoạn chương trình mô tả chi tiết ví dụ về một process có nhiều writer thread cùng modify đến cùng một vùng nhớ :



Description : Process trên bao gồm :

+ 3 threads : 1 reader (vì số lần lặp rất lớn 1e9 nên reader ở đây tạm thời không cần define mà chỉ cần đọc kết quả cuối cùng) và 2 writer cùng truy cập đến một vùng nhớ mà ví dụ ở đây là biến *count*

+ LOOPS : tổng số lần mà 1 thread thực hiện read hay write *count*

Ở đây, để cho đơn giản, thao tác modify biến *count* là phép tăng lên 1 biến count đó *count++*. Với số lần lặp là 1e9 cho mỗi writer thì khi đọc kết quả cuối cùng, biến *count* sẽ có giá trị là 2e9 theo mong muốn. Tuy nhiên, theo *writer – reader problem* ở trên, biến *count* sẽ bị lỗi nếu như cả 2 writer cùng ở trong critical section (*count++)* tại cùng một thời điểm. Đây là kết quả nếu không có lock bảo vệ count :

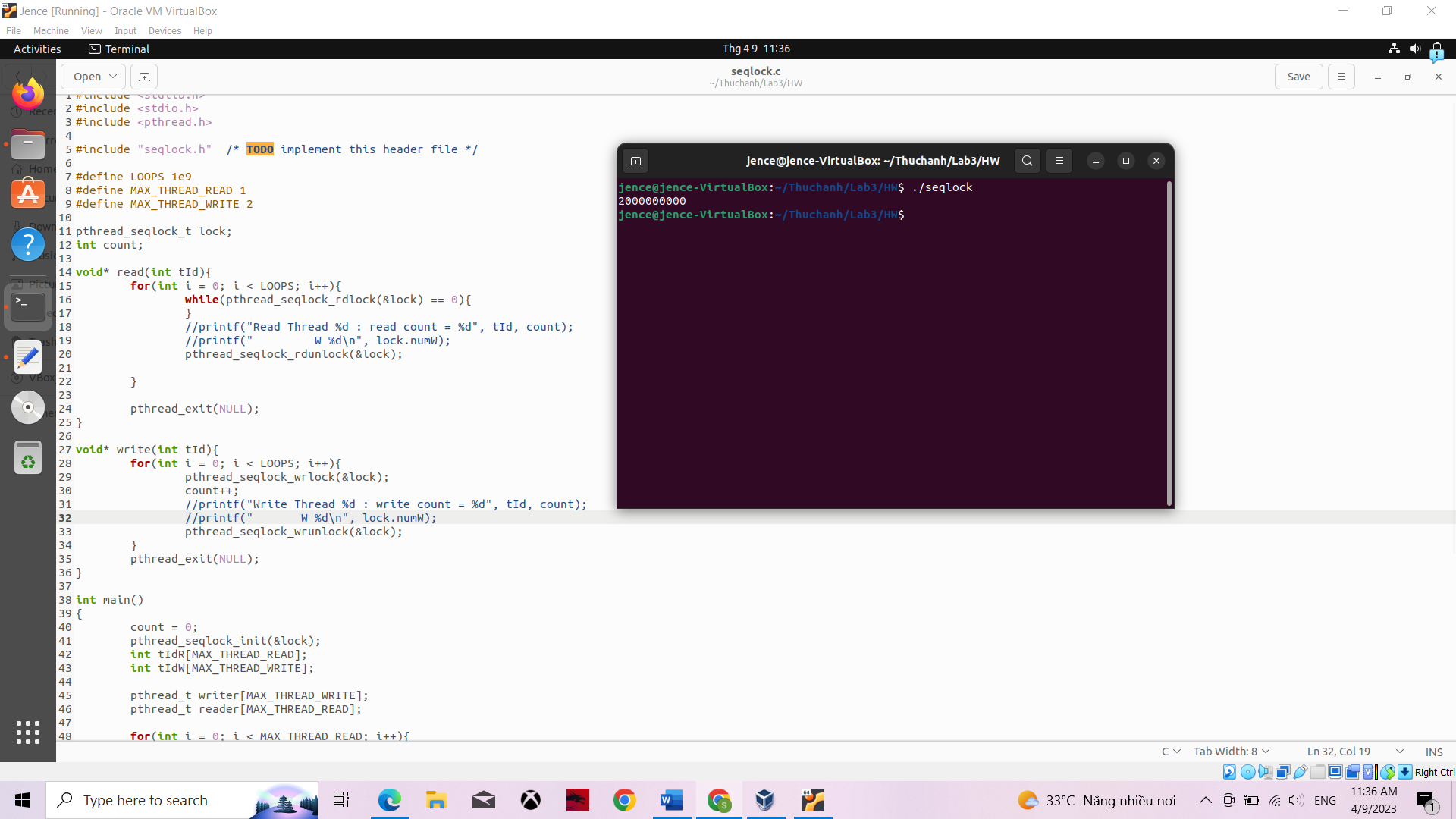
Như đã thấy, với 2 writer với số lần lặp là 1e9 thì kết quả mong muốn phải là 2e9 nhưng ở trên nếu không có lock bảo vệ, sẽ xảy ra tình huống  *count < 2e9*.

Mẫu chốt của vấn đề trên đó là phải sử lock để bảo vệ count, cụ thể hơn là trong cùng một thời điểm chỉ được phép có 1 writer trong critical section. Có hai cách để giải quyết vấn đề trên :

+ Read-write spinlock : sử dụng spinlock hoặc mutexlock để lock sao cho trong cùng một thời điểm, chỉ có một (nhiều) reader hoặc MỘT writer trong critical section, điều này sẽ giúp cho reader luôn đọc ra dữ liệu đúng và dữ liệu luôn được cập nhật đúng. Tuy nhiên, trong một số tình huống, nếu như reader chiếm dụng quá nhiều thời gian thì writer sẽ phải đợi rất lâu hay còn gọi là *writer starvation*. Mô hình giải quyết trên ưu tiên reader hơn writer (reader prefer).

+ Sequence lock : *writer starvation* trên thực tế là điều không nên, ví dụ khi người dùng cần cập nhật dữ liệu trong một hệ thống (game, SQL sever…) trong một khoảng thời gian ngắn thì việc phải chờ đợi reader quá lâu là không cần thiết. Do đó, *Sequence lock* với cơ chế tương tự như spinlock hay mutex những priority của writer cao hơn (writer prefer) :

* Nếu không có reader hay writer trong critical section => reader hay writer có thể vào mà không bị block
* Nếu có một writer trong critical section => không reader hay writer nào được phép vào, phải đợi hoàn thành => chỉ được phép có 1 writer trong critical section
* Nếu có reader trong critical section => các reader khác vẫn có thể vào. Tuy nhiên, nếu có một writer muốn vào để modify dữ liệu thì vẫn vào được mà không bị block => các reader hay writer vào sau đó sẽ bị block chờ writer hoàn thành ghi dữ liệu.

Sau đây là đoan chương trình trên nếu được sử dụng Sequence lock bảo vệ.

Kết quả mà writer cập nhật và reader đọc được chính xác 2e9 nếu các critical section của reader và writer có lock bảo vệ.