KHOA CÔNG NGHỆ THÔNG TIN
BỘ MÔN MẠNG MÁY TÍNH VÀ TT DỮ LIỆU

Hướng dẫn Lab 7.4 – Lệnh phần cứng

Khi mà các giải pháp phần mềm thất bại trong việc đồng bộ, vì chính các câu lệnh điều khiển Entry và Exit cũng gặp phải tình trạng cạnh tranh thì những giải pháp phần cứng cần được hiện thực và quá trình đồng bộ cần sự hỗ trợ của CPU, Hàng rào bộ nhớ là một trong những lệnh phần cứng. Hơn nữa, các thao tác cơ bản (+, -, =, …) cũng được hiện thực đơn nguyên ở mức phần cứng. Từ những nền tảng này, các công cụ phức tạp và hiệu quả hơn được các Hệ điều hành hiện thực (sẽ giới thiệu trong LAB 8).

Mục tiêu	**Lý thuyết liên quan**	**Tài nguyên**
Lệnh test_and_set()	Ch6.4.2 Hardware Instructions	https://github.com/Trantin84/LAB_IntroOS (mã nguồn ví dụ). Sử dụng image Ubuntu 16 / 18
Lệnh compare_and_swap()		
Lệnh đơn nguyên		

Yêu cầu sinh viên: Hiểu lý thuyết về lệnh phần cứng. Áp dụng cho các Yêu cầu và Bài tập.

Đánh giá sinh viên: Hỏi đáp các vấn đề lý thuyết. Kỹ năng thực hành. Bài tập.

Yêu cầu nộp bài: các tập tin mã nguồn .c và tập tin khả thực thi .out của các Yêu cầu trong buổi thực hành và Bài tập cuối hướng dẫn trong thời gian cho phép của giảng viên.

Preferences

[1] Abraham Silberschatz, Peter B. Galvin, Greg Gagne, [2018], Operating System Concepts, 10th edition, John Wiley & Sons, New Jersey.

Programming Problems of Chapter 3.

[2] Manual gcc of gnu, [2021], Built-in functions for atomic memory access,

Access https://gcc.gnu.org/onlinedocs/gcc-4.1.1/gcc/Atomic-Builtins.html

Yêu cầu 1: Một căn phòng được khoá bởi flag, một người sử dụng đầu tiên sẽ kiểm tra khoá, nếu khoá mở (test flag == false?) thì sẽ chiếm căn phòng và khoá lại (set flag = true) cho đến khi đi ra khỏi phòng thì mở khoá (set flag = false); ngược lại (test flag == true) thì anh ấy phải chờ.

Tình trạng cạnh tranh có thể xảy ra khi nhiều người cùng kiểm tra trạng thái của khoá và nhận trạng thái "false" đồng thời, cũng có nghĩa là họ sẽ cùng vào phòng, vi phạm điều kiện loại trừ tương hổ (mutual exclusion). Hành vi "kiểm tra và khoá lại" cần phải đơn nguyên, và đây là thao tác quan trọng nhất, nền tảng để làm nhiều lệnh đơn nguyên khác.

Hướng dẫn: Tải về tập tin ***LAB_IntroOS/LAB_7/task4_1_RoomTAS.c***, biên dịch và thực thi.

- Không cần thư viện, gcc đã bao gồm các lệnh này.

- Dòng 18 khai báo biến số cờ flag.

- Dòng 35 là kiểm tra cờ flag và đặt tín hiệu __ATOMIC_ACQUIRE nếu cờ chưa có tiểu trình chiếm hữu, thành công trả về 0.

- Dòng 40 là xoá trạng thái cờ.

18	`bool flag = false;`
35 (47)	`__atomic_test_and_set( & flag, __ATOMIC_ACQUIRE) == 0`
40 (52)	`__atomic_clear (&flag, __ATOMIC_RELEASE);`

Yêu cầu 2: (Tham khảo) Hiện thực lại Yêu cầu 1 với ở mức thư viện gcc atomic.

Hướng dẫn: Tải về tập tin ***LAB_IntroOS/LAB_7/task4_2_RoomLIB.c***, biên dịch và thực thi.

- THÊM VÀO Dòng 16 là thư viện cần thiết.

- THAY ĐỔI Dòng 20 khai báo biến số cờ flag kiểu atomic.

- Dòng 39 là lời gọi TAS với ý nghĩa "Nếu cờ là false thì đặt là true và trả về trạng thái cờ trước đó – là false; còn cờ đang là true thì trả về true". Tham khảo sách [1] p. 266:

```
boolean test_and_set(boolean *target){
     boolean rv = *target;
     *target = true;
     return rv;
}
```

- Dòng 44 là xoá cờ, thiết lập giá trị false.

- Dòng 51 và 56 cũng tương tự như vậy.

```
16      #include <stdatomic.h>
20      atomic_bool flag = false;

39      if (atomic_flag_test_and_set(&flag) == 0)
44           atomic_flag_clear( &flag);

51        while (atomic_flag_test_and_set(&flag) == 1) {
56                atomic_flag_clear( &flag);
```

Yêu cầu 3: Nếu TAS thao tác trên cờ nhị phân để kiểm soát vùng nguy cơ, thì compare_and_swap() là một lệnh phần cứng để thao tác trên các biến số nguyên. Trong ví dụ này, một căn phòng trống (owner = 0) có thể được gán cho nhân viên nào đó (ID > 0) sử dụng, trong thời gian đó owner = ID và một nhân viên khác phải chờ nếu cần dùng phòng. Khi sử dụng xong, owner trở về giá trị 0 (và chỉ nhân viên đang mượn phòng mới có quyền thực hiện phép gán này).

Hướng dẫn: Tải về tập tin ***LAB_IntroOS/LAB_7/task4_3_Room.c***, biên dịch và thực thi.

- Dòng 18 là thư viện.

- Dòng 20 là đặt giá trị ban đầu (không ai sở hữu).

- Dòng 40 là lời gọi "Nếu giá trị hiện tại của owner là 0 thì đặt lại là 1; thành công trả về true".

- Dòng 45 là lời gọi "Nếu giá trị hiện tại của owner là 1 thì đặt lại là 0; thành công trả về true".

- Tương tự cho dòng 53 và 62 (owner đặt là 2 – giá trị gán cho tiểu trình còn lại).

Tham khảo [1] p. 267:

```
int compare_and_swap(int *value, int expected, int newvalue){
     int temp = *value;
     if (*value == expected)
     *value = newvalue;
     return temp;
}
```

```
20      int owner = 0; //share

40        if (__sync_bool_compare_and_swap( & owner, 0, 1) ==
        true) {
45        if (__sync_bool_compare_and_swap( & owner, 1, 0) ==
        true)

53        while (__sync_bool_compare_and_swap( & owner, 0, 2) ==
        false) {
```

62	`if (__sync_bool_compare_and_swap( & owner, 2, 0) == true)`

Bài tập lập trình.

Trong các yêu cầu trên, "căn phòng" là hiện thực của vùng tranh chấp giữa hai, hoặc nhiều tiểu trình.

1. Đồng bộ bài toán Producer – Consumer bằng các lệnh TAS.
   Gợi ý: vùng tranh chấp được bảo vệ bởi khoá "flag", để vào vùng tranh chấp, tiểu trình phải thực hiện lệnh TAS và nếu lệnh này trả về false, tiểu trình phải chờ đến khi trạng tháng TAS trả về là true. Sau khi ra khỏi vùng tranh chấp, tiểu trình cũng trả cờ lại bằng lời gọi `atomic_flag_clear()`.

2. Đồng bộ bài toán Producer – Consumer bằng các lệnh CAS.
   Gợi ý: vùng tranh chấp có thể sở hữu bởi Producer (1), Consumer (2) hoặc không ai cả (0). Một tiểu trình có thể vào vùng tranh chấp nếu trạng thái vùng này là 0, đồng thời chuyển sở hữu vùng tranh chấp cho nó. Sau khi ra khỏi vùng tranh chấp, tiểu trình cũng trả lại bằng cách trả trạng thái vùng trở về giá trị 0. Mặt khác, tiểu trình phải chờ nếu như vùng tranh chấp đang bị chiếm hữu (giá trị <> 0).