BÀI 26: ĐỒNG BỘ HÓA TRONG KERNEL

Spinlocks, Mutex & Atomic

🎯 Mục tiêu bài học

1. **Concurrency:** Tại sao lập trình Driver khó hơn App? (Đa nhân SMP, Ngắt chèn ngang).
2. **Race Condition:** Lỗi tranh chấp dữ liệu trong Kernel.
3. **Giải pháp:** Khi nào dùng Spinlock? Khi nào dùng Mutex?
4. **Atomic:** Các biến nguyên tử không thể chia cắt.

1. Các nguồn gây tranh chấp

Dữ liệu của Driver (biến toàn cục) có thể bị truy cập đồng thời bởi:

1. **Nhiều Process:** App A và App B cùng `write` vào driver.
2. **Interrupt Handler:** Đang xử lý process thì bị ngắt chèn ngang.
3. **SMP (Symmetric Multi-Processing):** Code chạy song song trên nhiều Core CPU.

2. Spinlock vs Mutex

Đặc điểm	Spinlock (Khóa quay)	Mutex (Khóa ngủ)
Cơ chế	Vòng lặp `while(locked);` liên tục kiểm tra (Busy wait).	Nếu khóa đóng, tiến trình đi ngủ (Sleep) nhường CPU.
Sử dụng trong	Interrupt Context (Vì ngắt không được ngủ).	Process Context (Được phép ngủ).
Thời gian giữ	Rất ngắn (Vài micro giây).	Dài (Có thể chứa I/O chậm).
Tài nguyên	Tốn CPU (quay vòng).	Tốn thời gian chuyển ngữ cảnh (Context Switch).

3. Sử dụng Spinlock

```
#include <linux/spinlock.h>

spinlock_t my_lock;
unsigned long flags;

spin_lock_init(&my_lock);

// Trong hàm ngắt hoặc nơi cần bảo vệ:
spin_lock_irqsave(&my_lock, flags);  // Khóa và tắt ngắt cục bộ
// --- CRITICAL SECTION ---
count++;
// ------------------------
spin_unlock_irqrestore(&my_lock, flags);  // Mở khóa và khôi phục ngắt
```

4. Atomic Variables (Biến nguyên tử)

Nếu chỉ cần cộng/trừ 1 biến số nguyên, dùng Atomic nhanh hơn dùng Lock.

```
#include <linux/atomic.h>

atomic_t my_counter = ATOMIC_INIT(0);

// Cộng 1 (Không bao giờ bị race condition)
atomic_inc(&my_counter);

// Đọc giá trị
int val = atomic_read(&my_counter);
```

🛠️ PHẦN THỰC HÀNH (LAB 26)

Race Condition Simulator

Yêu cầu

1. Viết driver có 1 biến toàn cục `counter`.
2. Viết 2 App User Space:
    - App A: Ghi lệnh tăng `counter` 1 triệu lần.
    - App B: Ghi lệnh tăng `counter` 1 triệu lần.
    - Chạy 2 App đồng thời (`./appA & ./appB`).
3. **Lần 1:** Không dùng Lock -> Kết quả < 2.000.000 (Sai).
4. **Lần 2:** Thêm `mutex` hoặc `spinlock` vào hàm `write` của Driver -> Kết quả = 2.000.000 (Đúng).

Q & A

Chúc mừng bạn hoàn thành Giai đoạn 4!

Chuẩn bị sang Giai đoạn 5: Project & Automation!