CÔNG TY TNHH DEVLINUX
ĐÀO TẠO LẬP TRÌNH EMBEDDED CHUYÊN SÂU
Địa chỉ: LK05B No10 Khu Đồng Dưa, Quận Hà Đông,TP Hà Nội
ĐT: **0877123348** mail: **devlinux.edu@gmail.com**

Thao tác với Luồng – Thread

Bài tập 1: Tạo và Chạy Luồng Cơ bản

Bài tập này giới thiệu các thao tác nền tảng nhất với luồng: tạo một luồng thực thi mới từ luồng chính và chờ đợi luồng đó hoàn thành.

Yêu cầu:

1. Viết một chương trình C tạo ra hai luồng.
2. Mỗi luồng, khi bắt đầu thực thi, sẽ in ra một chuỗi thông báo kèm theo ID của nó (ví dụ: "Thread với ID <thread_id> đang chạy!").
3. Luồng chính phải sử dụng hàm pthread_join() để đợi cả hai luồng con kết thúc hoàn toàn trước khi chương trình chính kết thúc.

Câu hỏi phân tích:

- Giải thích vai trò và các tham số chính của hàm pthread_create() và pthread_join().
- Một luồng kết thúc khi nào?

Bài tập 2: Vấn đề "Race Condition" và Giải pháp Mutex

Bài tập này minh họa vấn đề "race condition" – một lỗi kinh điển trong lập trình đa luồng– và giới thiệu Mutex (Mutual Exclusion) như một giải pháp cơ bản để đảm bảo sự an toàn khi truy cập tài nguyên chia sẻ.

Yêu cầu:

1. Viết chương trình khai báo một biến toàn cục long long counter = 0;.
2. Tạo ra ba luồng. Mỗi luồng thực thi một vòng lặp để tăng giá trị của counter lên 1,000,000 lần.
3. Sử dụng một pthread_mutex_t để bảo vệ thao tác tăng counter. Mỗi luồng phải khóa (lock) mutex trước khi tăng và mở khóa (unlock) ngay sau đó.
4. Luồng chính dùng pthread_join() để đợi cả ba luồng hoàn thành, sau đó in ra giá trị cuối cùng của counter.

Câu hỏi phân tích:

- Tại sao cần phải sử dụng mutex trong bài toán này?
- Điều gì sẽ xảy ra nếu chúng ta bỏ qua việc sử dụng mutex? Giải thích tại sao kết quả cuối cùng có thể không chính xác và không ổn định.

Bài tập 3: Đồng bộ hóa với Condition Variables (Mô hình Producer-Consumer)

Bài tập này giải quyết bài toán "Nhà sản xuất - Người tiêu dùng" (Producer-Consumer) kinh điển. Nó giới thiệu cách sử dụng Condition Variables để một luồng có thể chờ đợi một điều kiện cụ thể xảy ra một cách hiệu quả, thay vì phải liên tục kiểm tra (busy-waiting).

Yêu cầu:

1. Xây dựng chương trình với một luồng Producer và một luồng Consumer.
2. Sử dụng một biến toàn cục data và một biến cờ data_ready để chia sẻ dữ liệu.
3. Producer: Lặp 10 lần, mỗi lần tạo một số ngẫu nhiên, đặt nó vào data, bật cờ data_ready, và sau đó báo hiệu (signal) cho Consumer.
4. Consumer: Chờ đợi (wait) cho đến khi được Producer báo hiệu. Sau khi được đánh thức, nó sẽ đọc và in giá trị từ data, tắt cờ data_ready.
5. Sử dụng một Mutex và một pthread_cond_t để đồng bộ hóa chính xác hai luồng này.

Bài tập 4: Tối ưu hóa Truy cập với Read-Write Lock

Trong các hệ thống có tần suất đọc dữ liệu cao hơn nhiều so với ghi, sử dụng Mutex có thể gây tắc nghẽn không cần thiết. Read-Write Lock là một cơ chế khóa chuyên biệt, cho phép nhiều luồng đọc đồng thời nhưng vẫn đảm bảo tính độc quyền cho luồng ghi.

Yêu cầu:

1. Viết chương trình mô phỏng một tài nguyên dữ liệu (một biến nguyên toàn cục).

2. Tạo ra 5 luồng đọc (Reader) và 2 luồng ghi (Writer).
3. Reader: Chỉ đọc và in ra giá trị của tài nguyên.
4. Writer: Tăng giá trị của tài nguyên lên 1.
5. Sử dụng pthread_rwlock_t để đồng bộ hóa:
    - Các Reader phải yêu cầu khóa đọc (pthread_rwlock_rdlock).
    - Các Writer phải yêu cầu khóa ghi (pthread_rwlock_wrlock).
6. Quan sát và in ra các hoạt động đọc/ghi để thấy rằng nhiều Reader có thể chạy song song, nhưng Writer phải chạy độc quyền.