ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

Operating Systems (CO201D)

Hiện thực và Đánh giá Hiệu năng các Cơ chế Đồng bộ cho Bộ đếm Đồng thời

GVHD: TS. Diệp Thanh Đăng

Lớp TN01

Nguyễn Tô Quốc Việt – 2313898

TP. Hồ Chí Minh, Ngày 19 tháng 12 năm 2025

Mục lục

1 Giới thiệu

Trong các hệ thống tính toán hiện đại dựa trên nền tảng bộ nhớ chia sẻ (shared-memory platform), việc đảm bảo tính toàn vẹn dữ liệu khi có nhiều luồng (threads) cùng truy cập vào một tài nguyên chung là một thách thức lớn. Bài tập này tập trung vào việc hiện thực hóa và đánh giá các cơ chế đồng bộ hóa khác nhau thông qua bài toán bộ đếm đồng thời (concurrent counter). Mục tiêu chính bao gồm việc kiểm chứng tính đúng đắn của các giải pháp và thực hiện nghiên cứu chuyên sâu về hiệu năng cũng như khả năng mở rộng (scalability) của chúng khi số lượng luồng thay đổi.

2 Nội dung thực hiện

Dựa trên yêu cầu của đề bài, các loại bộ đếm đồng thời đã được triển khai bằng cách sử dụng các kỹ thuật đồng bộ hóa khác nhau trong C++.

2.1 Cấu trúc giao diện chung (Interface)

Tất cả các loại bộ đếm đều kế thừa từ lớp cơ sở trừu tượng `ICounter` để đảm bảo tính nhất quán. Giao diện này bao gồm các phương thức thuần ảo: `increment()`, `get()`, và `reset()`. Việc sử dụng tính đa hình giúp hệ thống kiểm tra (testbench) có thể vận hành đồng nhất trên mọi đối tượng bộ đếm.

2.2 Hiện thực các cơ chế đồng bộ

- **MutexCounter**: Sử dụng `std::mutex` kết hợp với `std::lock_guard` để quản lý vùng găng (critical section). Đây là cơ chế khóa dựa trên sự hỗ trợ của hệ điều hành, đảm bảo loại trừ tương hỗ hoàn toàn.
- **AtomicCounter**: Tận dụng các chỉ thị nguyên tử cấp phần cứng thông qua `std::atomic`. Phương thức `increment()` sử dụng hàm `fetch_add` với chế độ bộ nhớ `memory_order_relaxed` để đạt hiệu năng tối ưu nhất trên các kiến trúc đa nhân.
- **CASCounter (Compare-And-Swap)**: Sử dụng vòng lặp với hàm `compare_exchange_weak`. Để giải quyết vấn đề tranh chấp cao, giải pháp này đã tích hợp kỹ thuật **Exponential Backoff**. Khi một nỗ lực cập nhật thất bại, luồng sẽ tạm dừng một khoảng thời gian ngắn (tính bằng nanoseconds) trước khi thử lại, giúp giảm tải cho đường truyền dữ liệu (bus contention).

Mã nguồn bài tập:

Toàn bộ mã nguồn, các tệp tiêu đề và lịch sử phát triển của dự án được lưu trữ và quản lý tại repository công khai: github.com/NguyenToQuocViet/os-extended-concurrent-counters

3 Kết quả và Đánh giá

3.1 Kiểm chứng tính đúng đắn (Correctness Verification)

Hệ thống kiểm tra được thiết lập với 4 luồng, mỗi luồng thực hiện 100.000 lần tăng. Kết quả thực nghiệm cho thấy cả 3 cơ chế Mutex, Atomic và CAS đều đạt được giá trị cuối cùng là 400.000. Điều này xác nhận rằng các cơ chế đồng bộ đã được hiện thực chính xác và loại bỏ hoàn toàn hiện tượng tranh chấp dữ liệu (Race Condition).

3.2 Phân tích hiệu năng và khả năng mở rộng

Dưới đây là bảng dữ liệu thu thập được từ thực nghiệm trên hệ thống máy tính: CPU i5-12400f (6c-12t), RAM 32GB (DDR4-3200MHz), với 1.000.000 lượt lặp mỗi luồng.

Số luồng	**Mutex (ms)**	**Atomic (ms)**	**CAS (ms)**
1	13	7	8
2	112	27	15
4	204	88	32
6	330	128	50
12	643	222	102
16	831	297	135
32	666	582	301
64	3161	1155	615
128	6393	2335	1320

Bảng 1: Kết quả Benchmark đo bằng thời gian thực thi (ms)

```
quocviet@Luminous:~/os-extended-concurrent-counters$ ./concurrent_counters
======================================
    PART 1: CORRECTNESS VERIFICATION
======================================
[PASS] Mutex Counter : 400000/400000
[PASS] Atomic Counter: 400000/400000
[PASS] CAS Counter   : 400000/400000

======================================
    PART 2: PERFORMANCE BENCHMARK
    (Iterations per thread: 1000000)
======================================
   Threads     Mutex (ms)    Atomic (ms)       CAS (ms)
------------------------------------------------------
         1             13              7              8
         2            112             27             15
         4            204             88             32
         6            330            128             50
        12            643            222            102
        16            831            297            135
        32            666            582            301
        64           3161           1155            615
       128           6393           2335           1320

Benchmark finished. Press Enter to exit.
```

Hình 1: Ảnh chụp thực tế kết quả Correctness và Benchmark trên Terminal

Phân tích các giai đoạn thực thi:

1. **Giai đoạn dưới ngưỡng giới hạn phần cứng (1 đến 12 luồng):** Trong phạm vi số luồng logic của CPU, thời gian thực thi tăng tương đối ổn định. `AtomicCounter` nhanh hơn `MutexCounter` đáng kể do không phải chịu chi phí quản lý trạng thái luồng (Sleep/Wakeup) từ Hệ điều hành. `CASCounter` với Exponential Backoff thể hiện ưu thế vượt trội nhất nhờ giảm thiểu tối đa sự tranh chấp tài nguyên ngay từ cấp độ phần cứng.

2. **Giai đoạn quá tải tài nguyên - Oversubscription (16 đến 128 luồng):** Đây là giai đoạn số lượng luồng phần mềm vượt xa số nhân vật lý.

   - **Mutex (6393ms tại 128 luồng):** Hiệu năng sụt giảm nghiêm trọng nhất. Khi 128 luồng cùng tranh giành một ổ khóa, chi phí cho việc chuyển đổi ngữ cảnh (Context Switching) và quản lý hàng đợi chờ khóa trong nhân Hệ điều hành trở thành nút thắt cổ chai chính.
   - **Atomic (2335ms tại 128 luồng):** Việc liên tục thực hiện lệnh `fetch_add` gây ra hiện tượng "nghẽn bus"(Bus Contention) dữ dội khi các lõi CPU liên tục phải đồng bộ hóa bộ nhớ đệm (Cache Coherency traffic).
   - **CAS với Backoff (1320ms tại 128 luồng):** Đây là cơ chế tối ưu nhất trong điều kiện khắc nghiệt. Thay vì nỗ lực thử lại (retry) liên tục và thất bại, kỹ thuật Exponential Backoff cho phép luồng tạm nghỉ, tạo khoảng trống cho các luồng khác hoàn tất nhiệm vụ. Điều này làm giảm lưu lượng giao tiếp trên bus và tận dụng tốt hơn các khe thời gian (time slices) mà hệ điều hành cấp phát.

Nhận xét chung:

Kết quả thực nghiệm minh chứng rằng việc lựa chọn cơ chế đồng bộ không chỉ phụ thuộc vào tính đúng đắn mà còn phụ thuộc vào quy mô hệ thống. `CASCounter` tích hợp Backoff không chỉ duy trì hiệu năng tốt ở mức tải thấp mà còn có khả năng chịu tải cực tốt khi hệ thống rơi vào trạng thái quá tải (Oversubscription).

4 Kết luận

Bài tập đã hoàn thành việc hiện thực và so sánh chi tiết các cơ chế đồng bộ hóa. Kết quả cho thấy các giải pháp không dùng khóa (Lock-free) như Atomic và CAS mang lại hiệu năng và khả năng mở rộng tốt hơn nhiều so với cơ chế dùng khóa (Lock-based) truyền thống. Việc kết hợp thêm kỹ thuật Exponential Backoff là một giải pháp tối ưu cho các hệ thống có độ tranh chấp cao.

——————————HẾT——————————