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

BÁO CÁO HỌC PHẦN MỞ RỘNG

HỆ ĐIỀU HÀNH (CO201D)

CLASS: TN01

Instructor: Diệp Thanh Đăng

STT	Họ và tên	MSSV	Tên lớp	Tên ngành
1	Lê Công Vinh	2313912	TN01	Kỹ thuật máy tính

THÀNH PHỐ HỒ CHÍ MINH, THÁNG 12 NĂM 2025

Mục lục

1 Tổng quan

Báo cáo tập trung hiện thực và đối sánh hiệu năng của mười phương pháp đồng bộ hóa luồng trong C++, từ các cơ chế khóa truyền thống như Mutex, SpinLock, TicketLock đến các biến nguyên tử và kỹ thuật CAS tích hợp chiến thuật Exponential Backoff. Thông qua quy trình thực nghiệm với 1 000 000 thao tác tăng giá trị trên dải từ 1 đến 100 luồng để làm rõ đặc điểm vận hành và thời gian thực thi của từng phương pháp khi mức độ tranh chấp tài nguyên thay đổi.

2 Các phương pháp hiện thực

2.1 Nhóm giải pháp Lock-based

Nhóm phương pháp này vận hành theo nguyên tắc ngăn chặn tranh chấp từ gốc, đảm bảo tính nhất quán dữ liệu bằng cách thiết lập quyền truy cập độc quyền tại một thời điểm nhất định.

- **Mutex Counter**: Đặc điểm nổi bật nhất của Mutex là tính **an toàn và tiết kiệm tài nguyên CPU** khi phải chờ đợi lâu. Khi khóa đã bị chiếm giữ, luồng đang chờ sẽ được hệ điều hành sẽ giải phóng CPU cho tác vụ khác.
- **Semaphore Counter**: Phụ thuộc nặng nề vào sự điều phối của hệ điều hành để quản lý hàng đợi và đánh thức luồng.
- **SpinLock Counter**: Ngược lại với Mutex, SpinLock giữ luồng ở trạng thái **luôn sẵn sàng** bằng cách liên tục chạy vòng lặp kiểm tra điều kiện khóa. *Ưu điểm:* Loại bỏ hoàn toàn độ trễ của việc chuyển đổi ngữ cảnh, giúp phản ứng tức thì ngay khi khóa trống. *Nhược điểm:* Gây lãng phí tài nguyên cực lớn vì CPU phải hoạt động 100% công suất chỉ để "chờ đợi".
- **Ticket Lock Counter**: Trong khi SpinLock thông thường có thể khiến một luồng bị chết đói (*starvation*), Ticket Lock đảm bảo mọi luồng đều được phục vụ theo đúng trình tự thời gian đến. Đây là cơ chế có tính dự báo cao, giúp hiệu năng hệ thống ổn định và đồng đều, tránh được các điểm nghẽn bất ngờ khi có quá nhiều luồng cùng tranh giành tài nguyên.

2.2 Nhóm kỹ thuật Atomic-based

Nhóm phương pháp này khai thác trực tiếp các chỉ thị cấp thấp của phần cứng để thao tác trên dữ liệu chia sẻ mà không cần sử dụng cơ chế khóa (lock-free), giúp giảm thiểu tối đa chi phí đồng bộ hóa.

- **Atomic Relaxed Counter:** Đặc điểm nổi bật nhất của phương pháp này là **hiệu năng tối đa và chi phí phần cứng thấp nhất**. Nó hoạt động dựa trên nguyên tắc chỉ đảm bảo tính nguyên tử cho chính phép toán đang thực hiện, loại bỏ hoàn toàn các rào cản bộ nhớ dùng để đồng bộ thứ tự.
- **Atomic Sequential Consistency Counter:** Đây là cơ chế mang lại **mức độ an toàn và tính trật tự cao nhất** trong các mô hình bộ nhớ. Đặc điểm chính là sự cam kết về một global ordering, đảm bảo rằng mọi luồng trong hệ thống đều quan sát thấy chuỗi thay đổi trạng thái theo cùng một thứ tự duy nhất.

2.3 Nhóm thuật toán Compare-And-Swap

Đây là kỹ thuật nền tảng của lập trình không khóa, áp dụng mô hình optimistic concurrency control. Thay vì chặn truy cập, các luồng sẽ thực hiện tính toán cục bộ và chỉ cam kết thay đổi vào bộ nhớ chung nếu trạng thái hệ thống chưa bị luồng khác can thiệp trong quá trình đó.

- **CAS Strong Counter:** Đặc điểm chính của phương pháp này là tính tất định và sự ổn định cao. Nó cung cấp sự đảm bảo tuyệt đối rằng thao tác sẽ không bao giờ gặp phải "thất bại giả"(spurious failure)—nghĩa là phép gán chỉ thất bại khi và chỉ khi dữ liệu thực tế đã bị thay đổi bởi một luồng khác. Mặc dù giúp đơn giản hóa logic chương trình, cơ chế này có thể tạo ra chi phí phần cứng cao hơn trên một số kiến trúc vi xử lý do CPU phải thực hiện nhiều bước kiểm tra nội bộ để duy trì cam kết này.
- **CAS Weak Counter:** Đây là phiên bản được tối ưu hóa tối đa cho hiệu suất phần cứng. Đặc điểm nổi bật là nó cho phép xảy ra "thất bại giả"do nhiễu tín hiệu đường truyền hoặc ngắt hệ thống ngay cả khi dữ liệu chưa thay đổi. Tuy nhiên, nhờ ánh xạ trực tiếp tới các chỉ thị máy đơn giản và tự nhiên hơn của CPU, cơ chế này thường mang lại throughput cao hơn đáng kể trong các thuật toán sử dụng retry loops liên tục.

2.4 Nhóm cơ chế CAS tích hợp Exponential Backoff

Đây là chiến lược nâng cao nhằm giải quyết bài toán suy giảm hiệu năng do tranh chấp đường truyền khi mật độ luồng xử lý tăng cao.

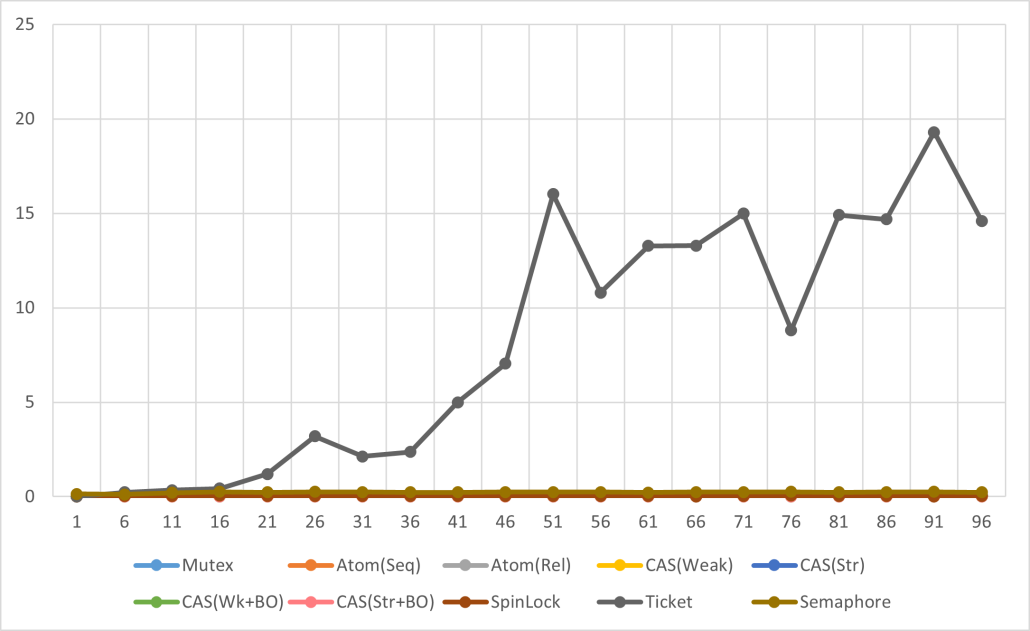
- **CAS Strong/Weak Backoff:** Phương pháp này tích hợp thuật toán exponential backoff để điều tiết tần suất truy cập. Thay vì liên tục chiếm dụng bus bộ nhớ để thử lại khi gặp xung đột, luồng sẽ tạm dừng hoạt động trong một khoảng thời gian tăng dần thích ứng với mức độ tắc nghẽn. Cơ chế này giúp giảm thiểu đáng kể lưu lượng đồng bộ cache, từ đó giải phóng băng thông cho luồng đang nắm giữ tài nguyên hoàn thành tác vụ nhanh hơn, giúp tăng cường khả năng mở rộng của toàn hệ thống.

3 Đánh giá kết quả

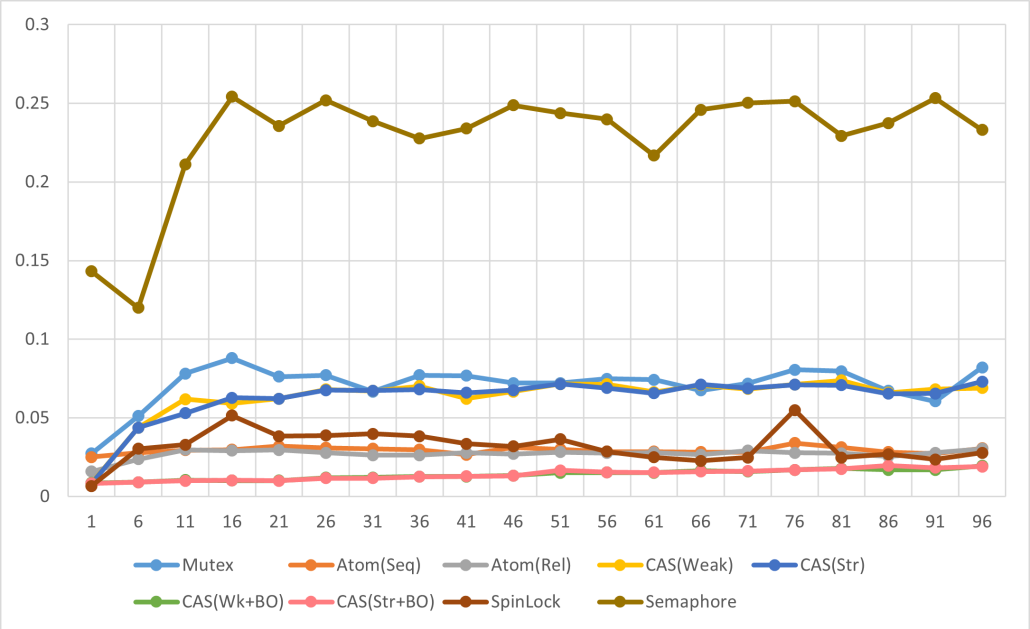
Sau khi tiến hành thực nghiệm với 1 000 000 lần tăng giá trị trên hệ thống, các số liệu thu được đã phản ánh rất rõ bản chất và giới hạn của từng cơ chế đồng bộ hóa khi đối mặt với áp lực tranh chấp gia tăng. Dưới đây là bảng tổng hợp kết quả thời gian thực thi (giây) theo số lượng luồng:

Threads	Mutex	Atom(Seq)	Atom(Rel)	CAS(Weak)	CAS(Str)	CAS(Wk+BO)	CAS(Str+BO)	SpinLock	Ticket	Semaphore
1	0.0274	0.025	0.0159	0.0085	0.0083	0.0087	0.0084	0.0067	0.007	0.1432
6	0.0512	0.0278	0.0237	0.0437	0.0438	0.009	0.009	0.0304	0.2264	0.1199
11	0.0782	0.0294	0.0297	0.0618	0.053	0.0104	0.01	0.0329	0.3466	0.2112
16	0.088	0.0298	0.0291	0.0591	0.0627	0.0102	0.0103	0.0515	0.427	0.2542
21	0.0762	0.032	0.0296	0.0621	0.0621	0.0102	0.01	0.0383	1.2014	0.2357
26	0.0771	0.0309	0.0278	0.0681	0.0676	0.012	0.0117	0.0388	3.192	0.2519
31	0.0667	0.0303	0.0263	0.0671	0.0673	0.0121	0.0117	0.0399	2.1302	0.2387
36	0.0771	0.0297	0.0263	0.0701	0.0681	0.0127	0.0125	0.0384	2.369	0.2277
41	0.0768	0.0266	0.0278	0.0621	0.066	0.0127	0.0128	0.0335	4.9944	0.2339
46	0.0722	0.0313	0.027	0.0667	0.0676	0.0133	0.0131	0.0318	7.0449	0.2488
51	0.0721	0.03	0.0282	0.0716	0.0715	0.0151	0.0167	0.0364	16.0141	0.2438
56	0.0749	0.0285	0.0276	0.0714	0.069	0.0152	0.0154	0.0286	10.8053	0.2399
61	0.0743	0.0285	0.0279	0.0664	0.0656	0.0151	0.0153	0.0249	13.2762	0.2168
66	0.0675	0.0282	0.0266	0.0707	0.0712	0.0165	0.0158	0.0227	13.2848	0.2459
71	0.0718	0.0283	0.0291	0.0683	0.0688	0.0159	0.0162	0.0248	14.9874	0.2502
76	0.0806	0.0339	0.0278	0.0712	0.0711	0.0169	0.017	0.0549	8.8155	0.2513
81	0.0797	0.0313	0.0274	0.0738	0.0708	0.0178	0.0176	0.0247	14.9077	0.2293
86	0.0671	0.0283	0.0255	0.0659	0.0653	0.0169	0.0196	0.0268	14.6838	0.2375
91	0.0605	0.0273	0.0277	0.0682	0.0655	0.0169	0.0183	0.0236	19.2949	0.2532
96	0.0821	0.0307	0.0303	0.069	0.073	0.0195	0.019	0.0278	14.5954	0.2331

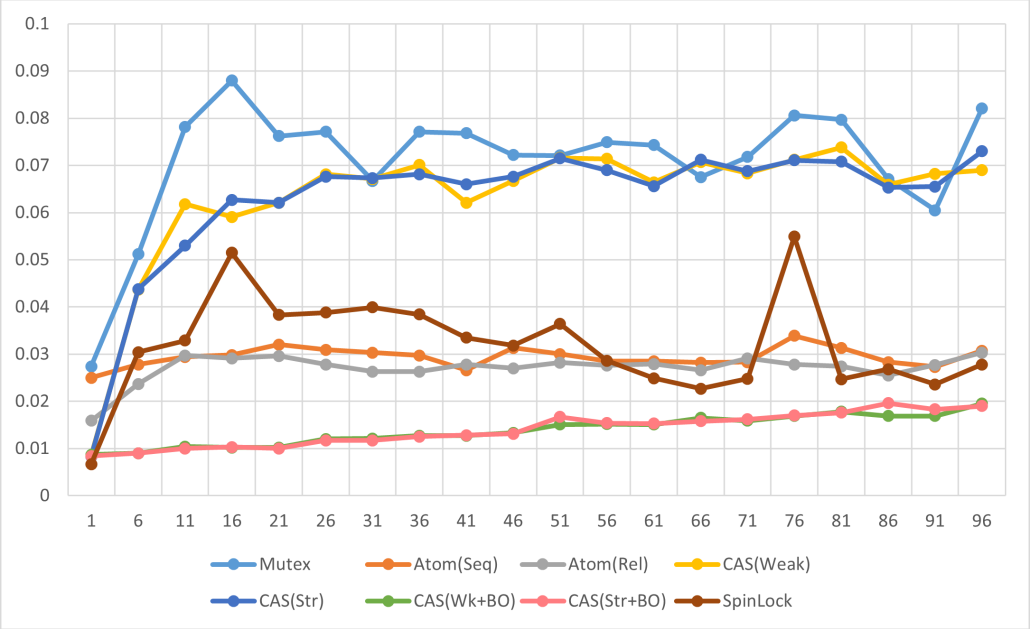
Bảng 1: Kết quả Benchmark thời gian thực thi của các cơ chế (đơn vị: giây)



Hình 1: Biểu đồ so sánh hiệu năng của tất cả cơ chế



Hình 2: Biểu đồ so sánh hiệu năng của tất cả cơ chế trừ Ticker Lock



Hình 3: Biểu đồ so sánh hiệu năng của tất cả cơ chế trừ Ticker Lock và Semaphore

Khi quan sát kỹ vào bảng số liệu và các biểu đồ thực nghiệm, chúng tôi rút ra những nhận xét quan trọng sau:

- **Mutex:** Biểu đồ cho thấy Mutex duy trì độ ổn định rất cao, dao động trong khoảng $0.06s - 0.08s$ bất chấp số lượng luồng tăng lên. Mặc dù không đạt tốc độ nhanh nhất do context switch của hệ điều hành, nhưng đây là phương án an toàn nhất, đảm bảo tính công bằng và tránh lãng phí CPU khi luồng phải chờ đợi lâu.

- **SpinLock:** Cơ chế này có hiệu năng tốt hơn Mutex ở mức tải thấp ($0.03s$) nhưng thể hiện sự thiếu ổn định nghiêm trọng khi số luồng tăng (dao động mạnh từ $0.02s$ đến $0.05s$). Sự trồi sụt này là do SpinLock phụ thuộc hoàn toàn vào bộ lập lịch; nếu luồng đang giữ khóa bị ngắt (preempted), các luồng khác sẽ tiêu tốn tài nguyên CPU vô ích để chờ đợi.

- **Ticket Lock:** Đây là trường hợp có thời gian đáng chú ý nhất. Từ mức khởi điểm ấn tượng $0.007s$, thời gian thực thi bùng nổ theo hàm mũ, đạt đỉnh điểm hơn $19s$ tại 91 luồng. Cơ chế "xếp hàng"nghiêm ngặt (FIFO) đã tạo ra nút thắt cổ chai, nơi hàng loạt luồng phải chờ đợi tuần tự, làm tê liệt hoàn toàn hệ thống khi tranh chấp lên cao.

- **Semaphore:** Số liệu thực nghiệm cho thấy đây là cơ chế có chi phí quản lý cao nhất. Ngay từ 1 luồng, thời gian thực thi đã là $0.1432s$, chậm hơn đáng kể so với tất cả các phương pháp khác. Tuy nhiên, nó duy trì được sự ổn định (dao động quanh $0.21s - 0.25s$) khi số luồng tăng cao. Điều này chứng tỏ Semaphore quá "nặng nề"để sử dụng cho mục đích bảo vệ vùng găng nhỏ (như tăng biến đếm). Vai trò thực sự của nó phù hợp hơn cho việc điều phối tín hiệu hoặc quản lý tài nguyên gộp thay vì khóa loại trừ.

- **Atomic Relaxed & Sequential Consistency:** Cả hai phương pháp này đều thể hiện một đường thẳng hiệu năng "phẳng lì"tuyệt vời quanh mức $0.027s$. Sự ổn định này đến từ việc các chỉ thị được ánh xạ trực tiếp xuống phần cứng.

- **CAS Nguyên bản (Naive CAS):** Các biến thể CAS Weak và CAS Strong thông thường bộc lộ rõ điểm yếu khi đối mặt với tranh chấp. Thời gian thực thi tăng vọt gấp 10 lần (từ $0.008s$ lên $0.076s$) và bị bão hòa tại đó. Nguyên nhân là do hiện tượng bus saturatio khi hàng loạt luồng liên tục gửi yêu cầu cập nhật bộ nhớ thất bại.

- **CAS kết hợp Exponential Backoff:** Đây là điểm sáng nhất của thực nghiệm. Với chiến lược "lùi bước theo hàm mũ", thời gian thực thi được duy trì ở mức siêu thấp ($0.008s - 0.019s$) ngay cả khi chạy với 96 luồng.

Tổng kết chung: Kết quả thực nghiệm cho thấy cơ chế **CAS kết hợp Exponential Backoff** là giải pháp tối ưu nhất về hiệu năng, nhờ khả năng tận dụng tốc độ của nhóm **Atomic-based** đồng thời giảm thiểu tranh chấp bộ nhớ thông qua chiến thuật chờ đợi thông minh. Các phương pháp **Lock-based** truyền thống vẫn giữ được ưu thế về độ ổn định và an toàn, trong khi **Ticket Lock** cần được cân nhắc kỹ lưỡng khi triển khai ở quy mô luồng lớn để tránh gây ra hiện tượng nghẽn băng thông hoặc sụt giảm nghiêm trọng hiệu suất hệ thống.

4 Kết luận

Tổng kết lại, bài tập này không chỉ giúp củng cố kiến thức lý thuyết về các chỉ lệnh nguyên tử và các loại khóa trong hệ điều hành, mà còn cung cấp kỹ năng quan trọng trong việc lựa chọn công cụ đồng bộ hóa phù hợp cho từng bài toán cụ thể. Đối với các hệ thống hiện đại đòi hỏi hiệu năng cao, các giải pháp dựa trên chỉ lệnh nguyên tử kết hợp trì hoãn lũy thừa hiện là hướng đi tối ưu nhất để đạt được sự cân bằng giữa tính đúng đắn và tốc độ thực thi.

5 Source code

Link github: https://github.com/vinhle275/Operating_Systems-Extended_Part

Tài liệu

[1] Abraham Silberschatz, Peter Baer Galvin, và Greg Gagne. *Operating System Concepts*. Tái bản lần thứ 10. Hoboken, NJ: John Wiley & Sons, Inc., 2018.