







>>> COURSE MATERIAL <<<

INTRODUCTION TO OPERATING SYSTEM / Course ID 502047

BÀI TẬP CHƯƠNG 7 – BÀI TOÁN ĐỒNG BỘ

Hướng dẫn tự học, chỉ sử dụng để tham khảo và có thể thay đổi mà không báo trước.

	Programming Exercises	--3--6-----4567-9
	Exam multichoice questions	12-4567-----3---7--
	Discussion questions	1234-----01-3--6---
	Just for reference	-----2-----8-

7.1 Giải thích tại sao Windows và Linux thực hiện nhiều cơ chế khóa. Mô tả các trường hợp theo đó hệ thống sử dụng spinlocks, khóa mutex, semaphores và các biến điều kiện. Trong mỗi trường hợp, giải thích sự cần thiết của cơ chế khóa.

7.2 Windows cung cấp một công cụ đồng bộ hóa nhẹ được gọi là khóa mỏng (slim lock) bộ ghi - bộ đọc. Không như hầu hết các hiện thực khóa bộ ghi - bộ đọc là ưu tiên cho một phía là bộ ghi hoặc bộ đọc¹, hoặc có thể chờ đợi theo nguyên tắc FIFO², các khóa mỏng bộ ghi - bộ đọc không ưu tiên cho cả hai, và cũng không chờ các tiểu trình sắp xếp trong hàng chờ FIFO. Giải thích những lợi ích của việc cung cấp một công cụ đồng bộ hóa như vậy.

7.3 Mô tả những thay đổi nào là cần thiết cho các tiến trình sản xuất và tiêu thụ trong mô tả sau để có thể sử dụng khóa mutex thay cho semaphore nhị phân.

¹ Biến thể 1 và 2 của bài toán Writer-Reader.

² Biến thể 3 của bài toán Writer-Reader.

```

while (true) {
    wait(full);
    wait(mutex);

    . . .

    /* remove an item from buffer to next_consumed */

    . . .
    signal(mutex);
    signal(empty);

    . . .

    /* consume the item in next_consumed */

    . . .
}

```

Figure 7.2 The structure of the consumer process.

7.4 Mô tả sự tắc nghẽn có thể xảy ra với bài toán Triết gia ăn uống như thế nào?

7.5 Chỉ ra và giải thích sự khác biệt giữa các trạng thái được báo hiệu và không được báo hiệu với các đối tượng điều phối Windows.

7.6 Giả sử val là một biến số đơn nguyên trong hệ thống Linux. Giá trị của val sau khi hoàn thành các lệnh sau đây là bao nhiêu?

```

atomic set(&val, 10);
atomic sub(8, &val);
atomic inc(&val);
atomic inc(&val);
atomic add(6, &val);
atomic sub(3, &val);

```

7.7 Mô tả hai cấu trúc dữ liệu trong nhân trong đó tình trạng cạnh tranh là có khả năng xảy ra. Hãy chắc chắn bao gồm một mô tả về làm thế nào một tình trạng cạnh tranh có thể xảy ra.

7.8 Nhân Linux có một chính sách rằng một tiến trình không thể giữ một spinlock trong khi cố gắng để có được một semaphore. Giải thích tại sao chính sách này được đưa ra.

7.9 Thiết kế thuật toán cho một bộ quan sát có bộ nhớ đệm giới hạn trong đó bộ nhớ đệm (danh cho các gói tin) được nhúng bên trong chính bộ quan sát.

7.10 Loại trừ lẫn nhau nghiêm ngặt trong một bộ quan sát làm cho các bộ quan sát có bộ nhớ đệm giới hạn của Bài tập 7.14 chủ yếu phù hợp với các gói tin nhỏ.

a. Giải thích sự hợp lý của phát biểu trên.

b. Thiết kế một kiểu mẫu mới phù hợp với các gói tin lớn.

7.11 Thảo luận về sự đánh đổi giữa sự công bằng và thông lượng của các hoạt động của bài toán Bộ ghi - bộ đọc. Đề xuất một phương pháp để giải quyết bài toán này mà không gây ra sự cạn kiệt tài nguyên.

7.12 Giải thích tại sao lời gọi phương thức lock() trong Java ReentrantLock không được đặt trong mệnh đề try để xử lý ngoại lệ, tuy nhiên lệnh gọi phương thức unlock() được đặt trong mệnh đề cuối cùng.

7.13 Giải thích sự khác biệt giữa bộ nhớ giao dịch (transactional memory) hiện thực bởi phần mềm và hiện thực bởi phần cứng.

7.14 Bài tập 3.20 yêu cầu bạn thiết kế chương trình quản lý PID phân bổ một ID duy nhất cho mỗi tiến trình. Bài tập 4.28 yêu cầu sửa đổi giải pháp ở Bài tập 3.20 bằng cách viết chương trình tạo ra một số tiểu trình để yêu cầu và cấp phát các ID của các tiến trình. Sử dụng khóa mutex, sửa đổi giải pháp của Bài tập 4.28 bằng cách đảm bảo rằng cấu trúc dữ liệu được sử dụng để thể hiện tính khả dụng của các ID định danh tiến trình là an toàn, không vi phạm điều kiện cạnh tranh.

7.15 Trong Bài tập 4.27, bạn đã viết một chương trình để tạo chuỗi Fibonacci. Chương trình yêu cầu tiến trình cha chờ đợi tiến trình con thực hiện thực thi trước khi in ra các giá trị được tính toán. Nếu chúng ta cho phép tiến trình cha truy cập vào các số Fibonacci ngay khi chúng được tính toán bởi tiến trình con, thì thay vì chờ đợi tiến trình con kết thúc, thì những thay đổi cần thiết cho giải pháp vừa nêu là gì? Hiện thực giải pháp sửa đổi đó.

7.16 Chương trình `stack-ptr.c` (có sẵn trong thư viện mã nguồn của môn học) chứa việc hiện thực ngăn xếp bằng danh sách được liên kết. Một ví dụ về việc sử dụng nó như sau:

```
StackNode *top = NULL;
push(5, &top);
push(10, &top);
push(15, &top);
int value = pop(&top);
value = pop(&top);
value = pop(&top);
```

Chương trình này tồn tại điều kiện cạnh tranh và không phù hợp với môi trường tính toán đồng thời. Sử dụng khóa mutex Pthreads (được mô tả trong Phần 7.3.1) để loại bỏ điều kiện cạnh tranh.

7.17 Bài tập 4.24 đã yêu cầu thiết kế một chương trình đa luồng ước tính giá trị π bằng kỹ thuật Monte Carlo. Trong bài tập đó, một tiểu trình duy nhất tạo các điểm ngẫu nhiên, lưu trữ kết quả trong một biến toàn cục. Khi tiểu trình đó hoàn tất, tiểu trình cha thực hiện phép tính ước lượng giá trị của π . Sửa đổi chương trình đó để tạo một số tiểu trình, mỗi tiểu trình tạo ra các điểm ngẫu nhiên và xác định xem các điểm có nằm trong vòng tròn hay không. Mỗi tiểu trình sẽ phải cập nhật tổng số điểm nằm trong và ngoài vòng tròn (là các biến toàn cục). Sử dụng khóa mutex để đảm bảo điều kiện cạnh tranh khi các tiểu trình đồng thời cập nhật giá trị toàn cục đã mô tả.

7.18 Bài tập 4.25 yêu cầu thiết kế chương trình bằng OpenMP ước tính giá trị π bằng kỹ thuật Monte Carlo. Kiểm tra lại giải pháp đó để tìm kiếm bất kỳ điều kiện cạnh tranh nào có thể tồn tại. Nếu có điều kiện cạnh tranh, hãy loại trừ nó bằng chiến lược được nêu trong Phần 7.5.2.

7.19 Rào chắn là một công cụ để đồng bộ hoạt động của một số lượng tiểu trình. Khi một tiểu trình thực thi đến một điểm rào cản, nó không thể tiếp tục cho đến khi tất cả các tiểu trình khác cũng đạt đến điểm này. Khi tiểu trình cuối cùng đạt đến điểm rào cản, tất cả các tiểu trình được giải thoát và có thể tiếp tục thực hiện đồng thời.

Giả sử ràng rào cản được khởi tạo cho N - số lượng tiểu trình phải chờ tại điểm rào cản:

```
init (N);
```

Mỗi tiểu trình sau đó thực hiện một số công việc cho đến khi nó thực thi đến điểm rào cản:

```
/* some calculating */
barrier point();
/* some calculating */
```

Sử dụng các công cụ đồng bộ hóa POSIX hoặc Java được mô tả trong chương này, xây dựng một rào cản thực hiện API sau:

```
int init(int n) //Khởi tạo rào cản với kích thước cụ thể.
int barrier point(void) //Xác định điểm cản. Tất cả các tiểu trình được
giải phóng khỏi rào cản khi tiểu trình cuối cùng đạt đến điểm này.
```

Giá trị trả về của mỗi hàm được sử dụng để xác định các lỗi, nếu có. Mỗi hàm sẽ trả về 0 nếu hoạt động bình thường và sẽ trả về -1 nếu xảy ra lỗi.