KHOA CÔNG NGHỆ THÔNG TIN
BỘ MÔN MẠNG MÁY TÍNH VÀ TT DỮ LIỆU

Hướng dẫn Lab 8.2 – Semaphore không tên.

Semaphores được sử dụng để đồng bộ các tiến trình và tiểu trình. Semaphores được kết hợp với Hàng đợi thông điệp và Bộ nhớ chia sẻ trong các IPC cơ sở trong các hệ thống Unix. Có hai loại semaphores, semaphores System-V truyền thống và semaphores POSIX. Trong bài học này, semaphores POSIX sẽ được giới thiệu.

Với các giải pháp đồng bộ khác, tiểu trình chờ đợi vẫn tiêu thụ CPU để chạy kiểm tra trạng thái khoá – được gọi là "busy waiting", dẫn đến sự nặng nề cho hệ thống. Với semaphore, tiểu trình thất bại trong việc yêu cầu tài nguyên này, sẽ bị đóng băng và sẽ được hệ điều hành đánh thức khi tài nguyên sẵn sàng trở lại. Trong suốt khoảng thời gian đó, tiểu trình không tiêu tốn CPU.

Có hai loại semaphores POSIX - được đặt tên và không tên. LAB 8.2 này sẽ nói về semaphore không tên, được sử dụng nội bộ giữa các tiểu trình HOẶC tiến trình cha – con.

Mục tiêu	**Lý thuyết liên quan**	**Tài nguyên**
Semaphore không tên Thao tác trên semaphore Áp dụng cho bài toán P-C	Ch6.6.1 Semaphore Usage	https://github.com/Trantin84/LAB_IntroOS (mã nguồn ví dụ). Sử dụng image Ubuntu 16 / 18

Yêu cầu sinh viên: Hiểu lý thuyết về semaphore. Áp dụng vào các bài toán đồng bộ.

Đánh giá sinh viên: Hỏi đáp các vấn đề lý thuyết. Kỹ năng thực hành. Bài tập.

Yêu cầu nộp bài: các tập tin mã nguồn .c và tập tin khả thực thi .out của các Yêu cầu trong buổi thực hành và Bài tập cuối hướng dẫn trong thời gian cho phép của giảng viên.

Preferences

[1] Abraham Silberschatz, Peter B. Galvin, Greg Gagne, [2018], Operating System Concepts, 10th edition, John Wiley & Sons, New Jersey.

Programming Problems of Chapter 6.

[2] Linux manual page, [2021], on *sem_init(3),*

Access https://man7.org/linux/man-pages/man3/sem_init.3.html

Yêu cầu 1: Đồng bộ bài toán Producer – Consumer bằng semaphore.

Hướng dẫn: Tải về tập tin ***LAB_IntroOS/LAB_8/task2_1_PC_sem.c***, biên dịch và thực thi.

- Dòng 6 là thư viện cần thiết.

- Dòng 34 là khai báo sem.

- Dòng 38 khởi tạo semaphore với giá trị 1 (đối số thứ ba). Trong TH này, semaphore tương được gọi là semaphore nhị phân, tương đương với khoá mutex, và mục đích là kiểm soát vùng nguy cơ.

  Với đối số thứ hai, ở đây sử dụng giá trị 0 nên được chia sẻ giữa các tiểu trình và đã được khai báo toàn cục ở dòng 34. Nếu giá trị này khác 0, semaphore được hiểu là sử dụng giữa các tiến trình cha – con và semaphore khi đó cần lưu trữ ở: vùng nhớ chia sẻ (Xem LAB 5).

- Dòng 58 là huỷ bỏ semaphore sau khi dùng xong.

- Dòng 72 (tương ứng dòng 89) là ra tín hiệu chờ vào vùng nguy cơ, nếu giá trị sem > 0 tiểu trình sẽ đi vào; ngược lại (sem <= 0) tiểu trình sẽ bị đóng băng.

- Dòng 76 là vùng nguy cơ của Producer, trong khi dòng 93 là vùng nguy cơ của Consumer.

- Dòng 77 (tương ứng dòng 94) là ra tín hiệu báo đã hoàn tất vùng nguy cơ (khi đó giá trị sem tăng 1 và sẽ đánh thức tiến trình đang bị đóng băng).

```
6         #include <semaphore.h>

34        sem_t mutex_sem;
38          sem_init( & mutex_sem, 0, 1)
58          sem_destroy( & mutex_sem)

72(89)      sem_wait( & mutex_sem)
76(93)           count++  / count--
77(94)      sem_post( & mutex_sem)
```

Yêu cầu 2: Giải quyết bài toán "Nông dân qua cầu Vermont" (xem phụ lục) bằng Counting semaphore. Để đơn giản, phiên bản đầu tiên chỉ xem xét tải trọng cầu = n (tối đa n người trên cầu) với n truyền vào qua lời gọi.

Hướng dẫn: Tải về tập tin ***LAB_IntroOS/LAB_8/task2_2_Bridge.c***, biên dịch và thực thi.

- Dòng 14 là số nông dân giả định.

- Dòng 18 đến 21 là hành vi của 1 nông dân: đến cầu, lên cầu, mất từ 3 đến 7 giây để đi qua, rời khỏi cầu.

- Dòng 29 là tạo ra các nông dân và dòng 30 giả định rằng nông dân tiếp theo đến sau 0 đến 2 giây.

```
14      #define MAX_FARMER 10

18        printf("\n%d Arriving bridge ...", v);
19        printf("\n%d Enter bridge ...", v);
20        sleep(rand() % 5 + 3);
21        printf("\n%d Leaving bridge ...", v);

29          pthread_create( )
30          sleep(rand() % 3);
```

Sau khi thực thi, chúng ta thấy rằng số nông dân trên cầu là tuỳ ý, giả sử chúng ta cần kiểm soát tải trọng cầu, hãy khai báo một counting semaphore và điều khiển như sau:

n là tải trọng cần kiểm soát.	`sem_init( & mutex_sem, 0, n)`
Trước khi "Enter" Sau khi "Leaving"	`sem_wait( & mutex_sem)` `sem_post( & mutex_sem)`

Hãy biên dịch lại và chạy chương trình vừa chỉnh sửa, thảo luận tải trọng cầu đã được kiểm soát hay chưa? Và vì sao?

Bài tập lập trình.

1. Xem bài toán Ước lượng số PI (***LAB_IntroOS/LAB_7/task1_2_PI.c***)
   Hãy thực hiện đồng bộ bằng semaphore, so sánh độ chính xác của số PI sau đó. Đồng thời đo thời gian chạy và cho biết sau khi đồng bộ, thời gian chạy đã tăng bao nhiêu %. Có thể cải tiến chương trình để giảm thiểu sự trả giá % thời gian chạy này không?

2. Xem xét bài toán “Nông dân qua cầu Vermont”, (tạm gọi bài tập này là phiên bản 2), chúng ta không kiểm soát tải trọng, mà giả định rằng người nông dân đến từ một trong hai hướng: Bắc và Nam, và không thể có 2 người cùng hướng đi lên cầu. Hãy thiết kế semaphore để giải quyết vấn đề này.
3. Xem xét bài toán “Nông dân qua cầu Vermont”, nếu chúng ta kiểm soát cả tải trọng lẫn chiều di chuyển, thì cần giải quyết như thế nào?

Phụ lục

Bài tập 8.30 Một cây cầu chỉ có một làn đường nối hai làng Vermont ở Bắc và Nam. Nông dân ở hai làng sử dụng cây cầu này để giao sản phẩm của họ đến thị trấn lân cận. Cây cầu có thể bị tắc nghẽn nếu một nông dân ở phía bắc và phía nam lên cầu cùng lúc, cùng thời điểm. (Nông dân Vermont bướng bỉnh nên sẽ không lùi bước). Sử dụng semaphores và / hoặc khóa mutex, thiết kế một thuật toán bằng mã giả để ngăn ngừa tắc nghẽn. Ban đầu, đừng quan tâm về vấn đề cạn kiệt tài nguyên (do nhiều nông dân từ một hướng liên tục đến cầu và phía bên kia phải chờ vô hạn định).