**BỘ CÔNG THƯƠNG**

**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP THỰC PHẨM**

**THÀNH PHỐ HỒ CHÍ MINH**

**ĐỒ ÁN KẾT THÚC MÔN HỌC**

**Tìm hiểu phương pháp đồng bộ hóa tiến trình Semaphores - viết ứng dụng minh họa bài toán tiến trình sản xuất, tiêu thụ (Producer – Consumer).**

**GVBM: Phạm Tuấn Khiêm**

**Sinh viên thực hiện**:

* Từ Huệ Sơn 2001190791
* Lê Đức Tài 2001190794
* Trần Thành Tâm 2001190248
* Trương Văn Bình 2001210071
* Phạm Văn Long 2001210918

*TP. HỒ CHÍ MINH, tháng 01 năm 2022*

**Hệ điều hành**

**Đề tài: Tìm hiểu phương pháp đồng bộ hóa tiến trình**

**Semaphores - viết ứng dụng minh họa bài toán tiến**

**trình sản xuất, tiêu thụ (Producer – Consumer).**

|  |  |
| --- | --- |
| **Điểm** | **Lời phê của Giảng Viên** |

**Sinh viên thực hiện**:

|  |  |  |
| --- | --- | --- |
| **MSSV** | **HỌ TÊN** | **CÔNG VIỆC** |
| 2001190791 | Từ Huệ Sơn |  |
| 2001190794 | Lê Đức Tài |  |
| 2001190248 | Trần Thành Tâm |  |
| 2001210918 | Phạm Văn Long |  |
| 2001210071 | Trương Văn Bình |  |

Mục lục

[**Phần 1: Tìm hiểu về phương pháp đồng bộ hóa tiến trình Semaphores** 1](#_Toc100494035)

[**1.1.** **Khái niệm Semaphores** 1](#_Toc100494036)

[**1.2.** **Các khái niệm** 1](#_Toc100494037)

# **Phần 1: Tìm hiểu về phương pháp đồng bộ hóa tiến trình Semaphores**

## **Khái niệm Semaphores**

Semaphores là một cấu trúc dữ liệu, được dùng để đồng bộ tài nguyên và đồng bộ hoạt động.

Khi được dùng với mục đích đồng bộ tài nguyên, semaphore tương tự như một bộ các chìa khóa dự phòng. Nếu một thread lấy được một chiếc chìa khóa, thread đó được phép truy cập vào tài nguyên. Nhưng nếu không còn chiếc chìa khóa nào, thread đó phải đợi cho tới khi một thread khác trả lại chìa khóa dự phòng. Nhờ vậy, race condition sẽ bị ngăn chặn.

Semaphore gồm 2 thành phần chính: biến count và hàng đợi wait\_list. Linux kernel sử dụng cấu trúc [semaphore](https://elixir.bootlin.com/linux/v4.4.137/source/include/linux/semaphore.h#L16) để biểu diễn một semaphore.

## **Các khái niệm**

* **Tiến trình là gì ?**

Tiến trình - process là một khái niệm cơ bản trong bất kì một hệ điều hành nào. Một tiến trình có thể được định nghĩa là một thực thể chương trình đang được chạy trong hệ thống. Một web server chạy trong thiết bị là một tiến trình, hoặc một chương trình soạn thảo văn bản đang chạy trong thiết bị cũng là một tiến trình.

* **Tiểu trình là gì ?**

Tiểu trình - thread là một tiến trình nhỏ có thể được quản lý độc lập bởi một bộ lập lịch. Tất cả các luồng trong một chương trình đơn được chứa hợp lý trong một tiến trình. Nhân cấp phát một ngăn xếp và khối điều khiển luồng (TCB) cho mỗi luồng. Hệ điều hành chỉ lưu con trỏ ngăn xếp và trạng thái CPU tại thời điểm chuyển đổi giữa các luồng của cùng một tiến trình.

Luồng được triển khai theo ba cách khác nhau: luồng cấp nhân, luồng cấp người dùng, và luồng lai. Luồng có thể có ba trạng thái running (đang chạy), ready (sẵn sàng) và blocked (bị chặn); nó chỉ bao gồm trạng thái tính toán không phân bổ tài nguyên và trạng thái giao tiếp làm giảm chi phí chuyển đổi, khiến tăng cường sự tương tranh (song song) do đó tốc độ cũng tăng lên.

Đa luồng cũng đi kèm với nhiều vấn đề. Việc có nhiều luồng không tạo ra sự phức tạp, nhưng sự tương tác giữa chúng thì không như vậy.

Một luồng phải có thuộc tính ưu tiên khi có nhiều luồng đang hoạt động. Thời gian để nó thực thi tương ứng với các luồng đang hoạt động khác trong cùng tiến trình được chỉ định bởi mức độ ưu tiên của luồng.

* **Đồng bộ hóa là gì ?**

Đồng bộ hoá là việc phối hợp các sự kiện riêng lẻ để vận hành một hệ thống cùng lúc.

* **Đồng bộ hóa dữ liệu là gì ?**

Đồng bộ hoá là việc phối hợp các sự kiện riêng lẻ để vận hành một hệ thống cùng lúc.

|  |  |  |
| --- | --- | --- |
| **Các yếu tố so sánh** | **Semaphore** | **Monitor** |
| Cấu trúc cơ bản | Semaphores là một số nguyên S. Hay nói cách semaphore chỉ là một biến đệm, nhưng được bảo đảm cho việc truy xuất đa luồng  (thread-safe). | Monitor là một kiểu dữ liệu trừu tượng (abstract data khác, semaphore chỉ là một type). |
| Mức độ trừu tượng | Cấp độ trừu tượng thấp hơn Monitor. | Cấp độ trừu tượng cao hơn Semaphore.Monitor có thể được cài đặt bằng Semaphore. |
| Cách thức hoạt động | Giá trị của Semaphore S biểu thị số lượng tài nguyên có sẵn trong hệ thống (availableresources). | Monitor chứa các biến dùng chung và các phương thức thao tác lên các biến dùng chung đó. |
| Truy cập | Khi bất kỳ tiến trình nào cần truy cập vào tài nguyên dùng chung (shared resources), tiến trình đó sẽ phải gọi là wait() với semaphore S. Sau khi xong việc truy xuất tài nguyên dùng chung, nó sẽ trả lại tài nguyên rồi gọi là signal với semaphore S. | Khi bất kỳ tiến trình nào cần truy cập các biến dùng chung, tiến trình đó cần phải truy cập biến đó thông qua các phương thức trong Monitor. |
| Các biến điều kiện  (Condition Variable) | Semaphore không có biến điều kiện. | Monitor có các biến điều kiện. |
| An toàn | Semaphore cấp thấp hơn và yêu cầu người sử dụng cần thận trọng trong việc quản lý tài nguyên. | Monitor cấp cao hơn, tự động quản lý việc signal() và wait(), khiến cho việc sử dụng đơn giản và an toàn hơn. |

Cả semaphore và monitor đều được dùng như là một công cụ để đồng bộ hoá. Cả 2 đều có thể thực hiện công việc như nhau và thay thế được cho nhau dễ dàng.

Điểm khác biệt có thể thấy rõ nhất, như đã nói ở bảng trên, đó là monitor dễ sử dụng hơn semaphore. Semaphore giống như con trỏ trong C++, mạnh mẽ và low-level nhưng lại yêu cầu sự cẩn thận, chu đáo từ người sử dụng nó.

Nếu bạn wait() mà quên release() trên semaphore, bạn sẽ rất có thể gặp phải trường hợp deadlock, hoặc những bug khó phát hiện. Monitor, ngược lại, đã giúp bạn giải quyết vấn đề trên bằng cách gọi hàm thay vì phải tự kiểm soát signal() và wait().