**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 06 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**

[**MỞ ĐẦU** 2](#_Toc102057905)

[**CHƯƠNG 1: TÌM HIỂU ĐỒNG BỘ HÓA** 3](#_Toc102057906)

[**1.** **Nhu cầu đồng bộ hóa (Synchronisation)** 3](#_Toc102057907)

[**2.** **Đồng bộ hóa là gì ?** 3](#_Toc102057908)

[**3.** **Vấn dề đồng bộ** 3](#_Toc102057909)

[**3.1.** **Các bài toán liên quan** 3](#_Toc102057910)

[**3.2.** **Đoạn găng là gì** 6](#_Toc102057911)

[**3.3.** **Giải pháp sử dụng biến cờ hiệu (Semaphore)** 7](#_Toc102057912)

[**CHƯƠNG 2: TÌM HIỂU VỀ SEMAPHORE** 7](#_Toc102057913)

[**1.** **Giới thiệu Edsger Wybe Dijkstra** 7](#_Toc102057914)

[**2.** **Kỹ thuật đèn báo (Semaphore)** 8](#_Toc102057915)

[**2.1.** **Khi tiến trình gọi P(S) và đèn báo S không dương** 8](#_Toc102057916)

[**2.2.** **Khi tiến trình đưa ra lời gọi V(S)** 8](#_Toc102057917)

[**3.** **Lí do nên sử dụng kỹ thuật đèn báo (Semaphore)** 9](#_Toc102057918)

# **MỞ ĐẦU**

# **CHƯƠNG 1: TÌM HIỂU ĐỒNG BỘ HÓA**

1. **Nhu cầu đồng bộ hóa (Synchronisation)**

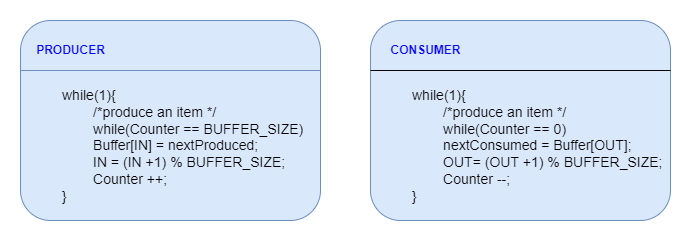
* Hệ thống máy tính hiện nay của chúng ta là hệ thống đa nhiệm do đó thì có nhiều tiến trình ở trong máy tính. Và tiến trình ở trong máy tính có 2 loại:
* Các tiến trình có nhu cầu liên lạc với nhau
* Các tiến trình không có nhu cầu liên lạc với nhau (các tiến trình làm việc theo lô)
* Hệ điều hành luôn cần cung cấp những cơ chế đồng bộ hóa để đảm bảo hoạt động đồng thời giữa các tiến trình mà các các tiến trình này hoạt động sẽ không làm sai lệch kết quả của tiến trình khác.
* Việc dẫn tới sự sai lệch các kết quả chính là sai lệch các giá trị trong vùng nhớ dùng chung
* Yêu cầu về độc quyền truy xuất (mutual exclusion) : tại một thời điểm, chỉ có một tiến trình được quyền truy xuất một tài nguyên không thể chia sẻ.
* Yêu cầu phối hợp (synchronization) : các tiến trình cần hợp tác với nhau để hoàn thành công việc. Ví dụ chương trình in sẽ xuất kí tự vào buffer, chương trình điều khiển máy in (printer driver) sẽ lấy kí tự trong buffer ra để in. Hai tiến trình này phải phối hợp với nhau để làm sao chương trình in không được xuất kí tự vào buffer khi buffer đầy mà phải chờ printer driver lấy bớt dữ liệu trong buffer ra.

1. **Đồng bộ hóa là gì ?**

* Đồng bộ các tiến trình là bảo đảm các tiến trình xử lý song song không tác động sai lệch đến nhau.

1. **Vấn dề đồng bộ**
   1. **Các bài toán liên quan**

* **Bài toán : Producer – Consumer**

****

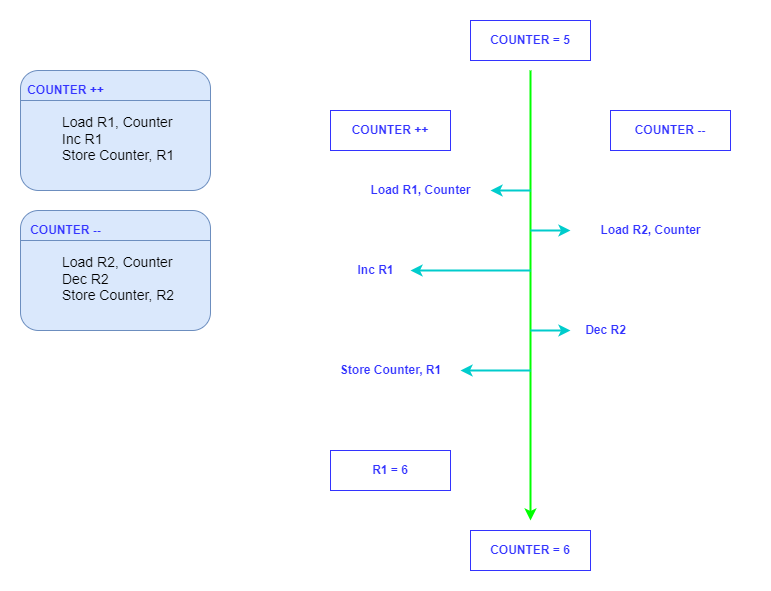
* **Producer (nhà sản xuất)**

Nếu như mà biến đếm (Counter) bằng **BUFFER\_SIZE** (là vùng nhớ mà ta khai báo) mà Counter đếm đã đầy không thể sản xuất được nữa thì nó sẽ dừng lại. Còn không bằng thì vòng lập while sai nó sẽ thoát ra và sẽ tạo **Buffer[IN] = nextProduced** (Buffer của con trỏ IN) để lưu trữ thêm sản phẩm được tạo ra (bởi vì con trỏ trỏ đến vùng nhớ trống trong bộ nhớ đệm) và con trỏ **IN = (IN+1) % BUFER\_SIZE** sẽ được tăng lên vùng nhớ tiếp theo và Counter sẽ tăng lên 1(**Counter ++**)

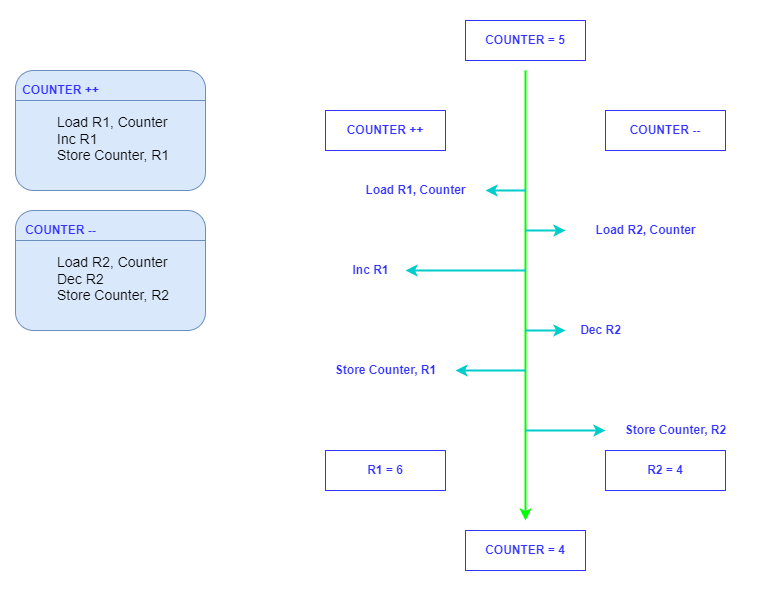
* **Consumer (Người tiêu thụ)**

Nếu Counter = 0 (có nghĩa là trong hàng đợi của nó không có sản phẩm nào thì làm sao người tiêu thụ có thể tiêu thụ được) thì kết thúc chờ while .Ngược lại, nếu Counter khác 0 (có nghĩa là có sản phẩm) thì sẽ sử dụng sản phẩm được lưu trữ trong Buffer mà con trỏ OUT trỏ vào **nextConsumed = Buffer[OUT]**. Sau khi trỏ vào Consumer lấy thì sẽ trỏ đến ô tiếp theo **OUT = (OUT+1) % BUFER\_SIZE** để lần sau Consumer lấy ô đó, mỗi lần Consumer lấy thì Counter sẽ giảm đi 1(**Counter --**)

* Giả sử Producer chạy để lưu sản phẩm thứ 7 thì đến lượt Consumer thí nó sẽ tiêu thụ 1 sản phẩm thì Counter sẽ giảm từ 7 sang 6. Như vậy, Producer cứ sản xuất 1 mà Consumer lại tiêu thụ 1 thì nó sẽ gọi đến Counter ++ và Counter --. Counter chính là biến đếm để Producer biết được nó đã sản xuất đầy **BUFFER\_SIZE** hay chưa nếu đầy thì dừng lại không sản xuất được nữa. Và Consumer cũng vậy sẽ kiểm tra **BUFFER\_SIZE** đó hết hay chưa nếu hết thì nó không tiêu thụ nữa => Buffer không đổi
* Vấn đề đồng bộ ở đây liên quan đến Counter ++ trong Producer và Counter -- Consumer. Ở đây Counter ++ sẽ chạy làn ba lệnh là: load Counter vào thanh ghi R1 (**Load R1, Counter**), tăng giá trị trong thanh ghi R1 (**Inc R1**), lưu trữ thanh ghi R1 vào Counter (**Store Counter, R1**). Counter -- sẽ tương tự cũng gồm ba lệnh là: load Counter vào thanh ghi R2 (**Load R2, Counter**), giảm giá trị trong thanh ghi R2 (**Dec R2**), lưu trữ thanh ghi R2 vào Counter (**Store Counter, R2**)

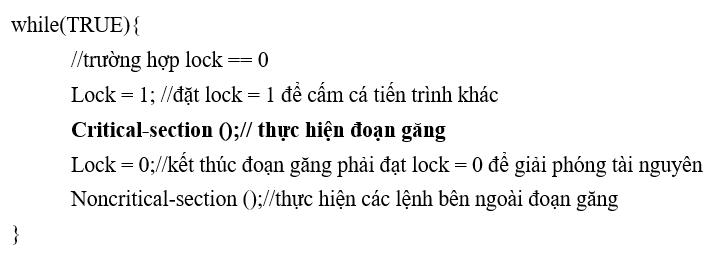


* Nếu Counter ++ được chạy trước thì nó sẽ bắt đầu chạy dòng lệnh **Load R1, Counter** (Counter = 5) thì R1 = 5 và hết thời gian thì sẽ đến lượt tiến trình Counter – thực hiện, thì nó sẽ chạy dòng lệnh đầu tiên là **load R2, Counter** thì R2 = 5 (bởi vì Counter là biến dung chung và Counter = 5). Giả sử đang sử dụng thuật toán Round Probin, mỗi tiến trình có lượng tử thời gian chiếm dụng q = 1. Counter -- hết thời gian thì Counter ++ thực hiện lệnh **Inc R1** (tăng R1) lúc này R1 = 6. Tiếp tới thì đến Counter -- thực hiện thì chạy câu lệnh **Dec R2** (giảm R2) khi đó R2 = 4. Sau đó, Counter ++ chạy cây lệnh cuối cùng là **Store Counter, R1** (lưu trữ R1 vào Counter) và R1 lúc này đang bằng 6 do đó nó sẽ lưu trữ vào biến dùng chung là Counter = 6.



Tiến trình Counter – vẫn còn 1 dòng lệnh chưa chạy và tiếp tục chạy tiếp lệnh **Store Counter, R2** (lưu trữ R2 vào Counter) thì Counter = 4. Giá trị ban đầu của Counter = 5 như vậy sẽ gây ra sai lệch giữa Producer và Consumer thì lúc này lượng sản phẩm được sản xuất và lượng tiêu thụ nó sẽ bị sai.

* 1. **Đoạn găng là gì**
* **Khái niệm đoạn găng (Miền găng)**
* Là đoạn mã lệnh có khả năng xảy ra mâu thuẫn khi truy xuất tài nguyên chung.
* **Hệ điều hành cần cài đặt các giả pháp đồng bộ để giải quyết vấn đền đọc quyền truy xuất**
* Quyết định tiến trình nào sẽ được vào đoạn găng (thực hiện các lệnh trong miền này)
* Khi 1 một tiến trình đang ở đoạn găng thì các tiến trình khác không thể vào đoạn găng
* **Critical session (Critical region đoạn găng)**
  1. **Giải pháp sử dụng biến cờ hiệu (Semaphore)**
* **Ý tưởng**
* Các tiến trình sẽ chia sẻ một biến chung đóng vai trò là cờ hiệu hay là một cái khóa (lock). Ban đầu biến này được khởi động là 0.
* Một tiến trình muốn vào đoạn găng, trước tiên giá trị của biến lock. Nếu lock = 0, tiến trình đặt lại giá trị cho lock = 1 và đi vào đoạn găng. Nếu lock đang nhận giá trị 1, tiến trình phải chờ khi có lock có giá trị là 0
* **Triển khai**



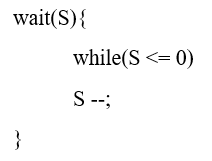
# **CHƯƠNG 2: TÌM HIỂU VỀ SEMAPHORE**

1. **Giới thiệu Edsger Wybe Dijkstra**

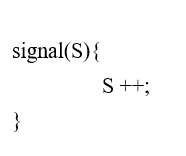
* Edsger Wybe Dijkstra 11 tháng 5 năm 1930 tại Rotterdam – 6 tháng 8 năm 2002 tại Nuenen), là nhà khoa học máy tính người Hà Lan. Ông được nhận giải thưởng Turing năm 1972 cho các đóng góp có tính chất nền tảng trong lĩnh vực ngôn ngữ lập trình. Không lâu trước khi chết, ông đã được nhận giải Bài báo ảnh hưởng lớn trong lĩnh vực tính toán phân tán của ACM dành cho bài báo đã khởi đầu cho ngành con Tự ổn định. Sau khi ông qua đời, giải thưởng thường niên này đã được đổi tên thành giải thưởng ACM Edsger W. Dijkstra.
* Edsger Wybe Dijkstra (người Hà Lan) phát minh ra khái niệm semaphore trong khoa học máy tính vào năm 1965. Semaphore được sử dụng lần đầu tiên trong cuốn sách “The operating system” của ông

1. **Kỹ thuật đèn báo (Semaphore)**

* Là một biến nguyên S, khởi tạo bằng khả năng phục vụ của tài nguyên nó điều độ
* Số tài nguyên có thể phục vụ tại một thời điểm (VD: 3 máy in)
* Số đơn vị tài nguyên có sẵn (VD: 10 chỗ trống trong buffer)
* Chỉ có thể thay đổi giá trị bởi 2 thao tác cơ bản P và V
* Thao tác P(S) (wait(S))



* Thao tác V(S) (signal(S))



* Các thao tác P(S) và V(S) xử lý không được tách rời
* Gồm 2 thao tác cơ bản
* Block(): ngừng tạm thời tiến trình đang thực hiện
* Wakeup(P): thực hiện tiếp tiến trình P dừng bởi lệnh block()
  1. **Khi tiến trình gọi P(S) và đèn báo S không dương**
* Tiến trình phải dừng bởi gọi tới câu lệnh block()
* Lệnh block() đặt tiến vào bảng hàng đợi gắn với đèn báo S
* Hệ thống lấy lại CPU giáo cho tiến trình khác (điều phối CPU)
* Tiến trình chuyển sang trạng thái chờ đợi (waiting)
* Tiến trình nằm trong hàng đợi đến khi tiến trình khác thực hiện thao tác V(S) trên cùng đèn báo
  1. **Khi tiến trình đưa ra lời gọi V(S)**
* Lấy một tiến trình trong hàng đợi ra (nếu có)
* Chuyển tiến trình lấy ra từ trạng thái chờ sang trạng thái sẵn sàng và đặt lên hàng đợi sẵn sàng gọi tới bởi wakeup(P)
* Tiến trình mới sẵn sàng có thể trưng dụng CPU từ tiến trình đang thực hiện nếu thuật toán điều phối CPU cho phép

1. **Lí do nên sử dụng kỹ thuật đèn báo (Semaphore)**

* Dễ dàng áp dụng cho các hệ thống phức tạp
* Không tồn tại hiện tượng chờ đợi tích cực
* Hiệu quả sử dụng phụ thuốc vào người dùng