**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**KHOA KỸ THUẬT MÁY TÍNH**

**TÀI LIỆU:**

**HƯỚNG DẪN THỰC HÀNH**

**HỆ ĐIỀU HÀNH**

**Nhóm biên soạn:**

- ThS. Phan Đình Duy

- ThS. Phạm Văn Phước

- ThS. Nguyễn Việt Quốc

- KS. Nguyễn Hữu Nhân

- KS. Lê Văn La

- KS. Trần Văn Quang

***Tháng 7 năm 2011***

**NỘI DUNG CÁC BÀI THỰC HÀNH**

**Phần 1: Lập trình trên Linux**

Bài 1: Hướng dẫn cài đặt Ubuntu và các lệnh cơ bản của shell

Bài 2: Cơ bản lập trình shell

**Phần 2: Thực hành hệ điều hành**

Bài 3: Quản lý tiến trình

Bài 4: Định thời CPU

Bài 5: Đồng bộ hóa tiến trình, tiểu trình

Bài 6: Quản lý bộ nhớ

**Phần 3: Bài tập lớn**

CÁC PHẦN MỀM THIẾT BỊ SỬ DỤNG TRONG MÔN THỰC HÀNH

- Phần mềm VMware

- Hệ điều hành Ubuntu

# 

# Đồng bộ hóa tiến trình và tiểu trình

## Mục Đích

Mục đích của bài thực hành này nhằm giới thiệu đến các sinh viên thư viện Semaphore và Mutex, một thư viên để đồng bộ hóa các tiến trình và tiểu trình chạy song song với nhau.

Những công cụ cần chuẩn bị cho bài lập trình:

* Hệ điều hành linux đã cài đặt GCC và G++ compilers (Gói build-essential package của linux)
* Cài đặt eclipse hoặc các C/C++ Integrate Development Environment.

## Semaphore

Trong khoa học máy tính, Đặc biệt là trong hệ điều hành, semaphore được biết đến như một **biến** được sử dụng để điều khiển sự truy xuất vào tài nguyên chung của các tiểu trình trong xử lý song song hoặc trong các môi trường đa người dùng.

Semaphore được xem như một danh sách các đơn vị còn trống của một tài nguyên cụ thể của máy tinh. Có 2 thao tác cơ bản trên một biến semaphore đó là yêu cầu tài nguyên và giải phóng tài nguyên,và nếu cần thiết nó còn có thể làm cờ để đợi cho đến khi tài nguyên được một tiểu trình khác giải phóng.

### Các function cơ bản cho Semaphore trên C

* Để include thư viên semaphore vào một chương trình C ta sử dụng

#include <semaphore.h>

Và khai báo thư viện pthread và rt khi biên dịch . Ví dụ

gcc -o filename filename.c -lpthread -lrt

* Để định nghĩa một semaphore ta dùng:

sem\_t sem\_name;

* Để khởi tạo một biến semaphore ta sử dụng hàm:
* int sem\_init(sem\_t \*sem, int pshared, unsigned int value);
* sem : là con trỏ đến địa chỉ của biến semaphore.
* Pshared là cờ để cho cài đặt cho việc chia sẻ biến semaphore giữa các với các fork()ed processes, tuy nhiên hiện tại LinuxThread không hỗ trợ Shared semaphores.
* Value là giá trị khởi tạo để gán cho semaphore
* Ví dụ:

Sem\_t sam;

Sem\_init(&sam,0,10);

* Đợi một semaphore:

Sem\_wait(&sem\_name);

* Nếu giá tri của semaphore là không dương, tiến trình sẽ bị block, một trong những tiến trình bị block sẽ mở khi có một process gọi sem\_post.
* Nếu giá trị của semaphore là dương, giá trị của semaphore sẽ giảm đi 1. Tiến trình tiếp tục được chạy.
* Tìm giá trị của semaphore:

int sem\_getvalue(sem\_t \*sem, int \*valp);

* Lấy giá trị của semaphore vào biến int có địa chỉ *valp.*
* Ví dụ:

int value;

sem\_getvalue(&sem\_name, &value);

printf("The value of the semaphors is %d\n", value);

* Hủy một biến semaphore:

sem\_destroy(sem\_t \*sem);

* Hủy semaphore, không nên có tiểu trình nào sem\_wait nó nữa

### Ví dụ về sử dụng semaphore:

Giả sử có 2 process được thực thi song song bởi đoạn code như sau:

processA{

while (true)

na++;

}

processB{

while (true)

nb++;

}

Giá trị ban đầu của na và nb là 0. Sử dụng semaphore để đảm bảo rằng na<=nb (1);

Chúng ta biết rằng processA và processB được thực thi đồng thời, và không gì đảm bảo rằng processA và process sẽ được chạy ở cùng một tốc độ. Do đó, tốc độ tăng của na và nb là không giống nhau. Điều này dẫn đến nếu nb chạy chậm hơn na thì rang buộc (1) không còn được đảm bảo.

Để giải quyết điều này, chúng ta sử dụng 1 semaphore để đảm rằng nếu như na >= nb thì na phải chờ nb tăng lên trước.

*Sem\_t sam;*

*Sem\_init(&sam,0,0);*

processA{

while (true)

{

*Sem\_wait(&name);*

na++;

}

}

process{

while (true)

{

nb++;

*sem\_post(&sam);*

}

}

Theo solution trên, mỗi lần nb tăng them 1 đơn vị thì *sam*  sẽ tăng lên 1 đơn vị. điều đó kéo theo na được quyền tăng theo 1 đơn vị.

Trước mỗi lẫn na tăng thêm 1 đơn vị thì lệnh *Sem\_wait(&sam);*  được thực hiện. lệnh này sẽ trừ biến *sam*  đi một đơn vị. Cho đến khi *sam=0* thì na phải đợi nb được tăng thêm một đơn vị mới được tăng tiếp.

### Bài tập

**1.**Lập trình hiện thực hóa ví dụ 1.2.2 trên C

**2.** Giả sử có 2 process được thực thi song song bởi đoạn code như sau:

processA{

while (true)

na++;

}

process{

while (true)

nb++;

}

Giá trị ban đầu của na và nb là 0. Sử dụng semaphore để đảm bảo rằng nb<na<=nb+10 (2);

**3\*(optional).**Cho một mảng a được khai báo như một mảng số nguyên có thể chứa n phần tử, a được khai báo như một biến toàn cục.

Viết chương trình bao gồm 2 thread chạy song song

* Một thread làm nhiệm vụ sinh ra 1 số nguyên ngẫu nhiên sau đó bỏ vào a. Sau đó *đếm và xuất* ra số phần tử của a có được *ngay sau khi thêm vào.*
* thread còn lại lấy ra một phần tử trong a (phần tử bất kì, phụ thuộc vào người lập trình), Sau đó *đếm và xuất* ra số phần tử của a có được *ngay sau khi lấy ra,* nếu không có phần tử nào trong a thì xuất ra màng hình “ không có phần tử nào trong mảng a”.

Chạy thử và tìm ra lỗi khi chạy chương trình trên khi chưa được đồng bộ.

Thực hiện đồng bộ hóa với semaphore.

## Mutex

Mutex là một trường hợp đơn giản của semaphore khi mà giá trị của semaphore 0<=v<=1;

Mutex thường được sử dụng như sau:

* Khai báo và khởi tạo một biến mutex.
* Các thread xin phép khóa mutex để truy cập vào miền gang.
* Tại 1 thời điểm chỉ có một thread thành công trong việc khóa mutex, thread đó hiển nhiên được phép truy cập vào miền gang.
* Thread khóa mutex thực hiện một số thao tác trong miền găng.
* Thread khóa mutex nhả mutex. Sau khi nhả, các thread khác được quyền khóa mutex và sử dụng sử dụng miền găng.
* Hủy mutex

### Các function cơ bản cho Mutex trên C

Để sử dụng mutex ta cần phải include file pthread.h vào trước.

Sau khi include file, mutex có thể được sử dụng thông qua các function như sau:

* Để khai báo một mutex ta sử dung

pthread\_mutex\_t mutex;

thông thương mutex được khai báo như một biến toàn cục.

* Để khởi tạo một mutex ta sử dụng function:

int pthread\_mutex\_init(pthread\_mutex\_t \*mutex,const pthread\_mutexattr\_t \* attr);

Ở đó, mutex là con trỏ chỉ đến địa chỉ của biến mutex, attr là con trỏ chỉ đến nơi mà function có thể tìm thấy thuộc tính của mutex cần khởi tạo. con trỏ của attr có thể được để NULL khi ta sử dụng các cài đặt mặc định của Mutex.

* Để hủy một mutex, ta sử dụng:

[pthread\_mutex\_destroy](https://computing.llnl.gov/tutorials/pthreads/man/pthread_mutex_destroy.txt)(pthread\_mutex\_t \*mutex);

* Để khóa mutex ta sự dụng hàm:

pthread\_mutex\_lock (pthread\_mutex\_t \*mutex);

Trường hợp mutex đã bị khóa bởi một thread khác, hàm này sẽ block thread hiện tại lại và chờ cho đến khi mutex được nhả ra.

* Để mở khóa cho mutex ta sử dụng hàm:

pthread\_mutex\_unlock (pthread\_mutex\_t \*mutex);

sau khi mở khóa, các thread khác được quyền tranh chấp giành quyền khóa mutex lại.

### Bài tập

1. Chạy song song 2 process A và B được mô tả như sau:

int x=0;

process A()

{

while (1){

x=x+1;

if (x==20) { x = 0;}

print(x);

}

}

process B()

{

while (1){

x=x+1;

if (x==20) { x = 0;}

print(x);

}

}

Thực hiện chưa trình trên trên C và chỉ ra xem kêt quả thu được trên màn hình có gì không hợp. Chỉ ra điểm không hợp lý.

1. Thực hiện đồng bộ với mutex để chương trình (1) chạy đúng với ý đồ.
2. Biến ans được tính từ các biên x1,x2,x3,x4,x5,x6 như sau:

w=x1\*x2; (a)

v=x3\*x4; (b)

y=v\*x5;(b)

z=v\*x6;(c)

y=w\*y;(d)

z=w\*z;(e)

ans=y+z;(f)

Giả sử các câu lệnh từ (a)🡪(f) được nằm trên các thread chạy song song với nhau.

Hãy lập trình mô phỏng và đồng bộ chúng trên C để đảm bảo rằng lệnh (b),(c) được thực hiện chỉ sau khi v đã được tính, lệnh (d) được thực hiện chỉ sau khi w và y đã được tính, lệnh (f) được thực hiện chỉ sau khi y và z được được tính.

PHỤ LỤC

[Chương 1. Đồng bộ hóa tiến trình và tiểu trình 3](#_Toc410557192)

[1.1 Mục Đích 3](#_Toc410557193)

[1.2 Semaphore 3](#_Toc410557194)

[1.2.1 Các function cơ bản cho Semaphore trên C 3](#_Toc410557195)

[1.2.2 Ví dụ về sử dụng semaphore: 5](#_Toc410557196)

[1.2.3 Bài tập 6](#_Toc410557197)

[1.3 Mutex 7](#_Toc410557198)

[1.3.1 Các function cơ bản cho Mutex trên C 8](#_Toc410557199)

[1.3.2 Bài tập 9](#_Toc410557200)

**TÀI LIỆU THAM KHẢO**

1. <https://www.google.com.vn/url?sa=t&rct=j&q=&esrc=s&source=web&cd=3&cad=rja&uact=8&ved=0CDEQFjAC&url=https%3A%2F%2Fnguyenvanquangcse.files.wordpress.com%2F2014%2F09%2Fexcercise1.pdf&ei=CVHEVLrBHeOxmwXQh4GIDg&usg=AFQjCNFX6il7uy1TbHmcC4E8jzB1hXCCiw&bvm=bv.84349003,d.dGY>
2. <http://kb.datapool.vn/how-to-huong-dan-lam-quen-voi-cac-lenh-co-ban-tren-linux-unix/4/>
3. <https://computing.llnl.gov/tutorials/pthreads/#Mutexes>