**HỆ ĐIỀU HÀNH**

**HƯỚNG DẪN**

**SEMAPHORE**

***Hướng dẫn làm bài:***

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 dùng chung 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.

Có hai loại semaphores POSIX - được đặt tên và vô danh. Như thuật ngữ cho thấy, semaphores được đặt tên có một tên, đó là định dạng / somename. Ký tự đầu tiên là dấu gạch chéo về phía trước, theo sau là một hoặc nhiều ký tự, không có ký tự nào là dấu gạch chéo. Đầu tiên chúng ta sẽ xem xét các semaphores được đặt tên và sau đó là semaphore vô danh.

Các chương trình sử dụng ngữ nghĩa POSIX cần được liên kết với thư viện pthread.

Một biến semaphore sẽ được gán đến các tài nguyên. Khi một tiến trình muốn sử dụng tài nguyên nó sẽ kiểm tra biến semaphore của tài nguyên này. Nếu giá trị của biến này khác 0, tài nguyên đang có sẳn. Nếu giá trị biến này bằng 0, tài nguyên đang được sử dụng

Thư viện và các lời gọi

#include <sys/types.h>

#include<sys/ipc.h>

#include<sys/sem.h>

sem\_open

sem\_post

sem\_wait

sem\_trywait

sem\_timedwait

sem\_getvalue

sem\_unlink

sem\_init

sem\_destroy

**Lời gọi sem\_open**

#include <fcntl.h>

#include <sys/stat.h>

#include <semaphore.h>

sem\_t \*sem\_open (const char \*name, int oflag); //form #1

sem\_t \*sem\_open (const char \*name, int oflag,

mode\_t mode, unsigned int value); //form #2

**sem\_open** là lời gọi để bắt đầu một semaphore. sem\_open mở một semaphore hiện có hoặc tạo một semaphore mới và mở nó cho các hoạt động tiếp theo. Tham số đầu tiên, name, là tên của semaphore, một chuỗi kí tự hằng số. oflag có thể chứa O\_CREAT, trong trường hợp đó, semaphore được tạo nếu nó chưa tồn tại. Nếu cả O\_CREAT và O\_EXCL được chỉ định, lời gọi sẽ báo lỗi nếu semaphore có tên được chỉ định đã tồn tại. Nếu tham số oflags có thiết lập O\_CREAT, hình thức thứ hai của sem\_open phải được sử dụng, có thêm 2 tham số: chế độ mode và giá trị nguyên không âm. Tham số mode chỉ định các quyền cho semaphore, được che dấu bằng umask cho tiến trình, tương tự như mode trong lời gọi hệ thống open() cho các tập tin. Tham số cuối cùng là giá trị khởi tạo cho semaphore. Nếu O\_CREAT được chỉ định trong oflag và semaphore đã tồn tại, cả tham số chế độ và giá trị đều bị bỏ qua.

sem\_open trả về một con trỏ tới semaphore khi thành công. Con trỏ này phải được sử dụng trong các cuộc gọi tiếp theo cho semaphore. Nếu cuộc gọi thất bại, sem\_open trả về SEM\_FAILED và errno được đặt lỗi tương ứng.

Trong Linux, các ngữ nghĩa POSIX được tạo trong thư mục /dev/shm. Các semaphores được đặt tên với một tiền tố, sem. theo sau là tên được truyền trong lệnh gọi sem\_open.

**Lời gọi sem\_post**

#include <semaphore.h>

int sem\_post (sem\_t \*sem);

sem\_post tăng semaphore. Nó cung cấp hoạt động V cho semaphore, tương tự signal(). Nó trả về 0 khi thành công và -1 khi lỗi.

**Lời gọi sem\_wait / sem\_trywait / sem\_timedwait**

#include <semaphore.h>

int sem\_wait (sem\_t \*sem);

sem\_timedwait

struct timespec {

time\_t tv\_sec; /\* Seconds \*/

long tv\_nsec; /\* Nanoseconds [0 .. 999999999] \*/

};

**sem\_wait** giảm các semaphore được chỉ đến bởi \*sem. Nếu giá trị semaphore là khác không, sự sụt giảm xảy ra ngay lập tức. Nếu giá trị semaphore bằng 0, các lời gọi bị chặn cho đến khi semaphore trở nên lớn hơn 0 và việc giảm dần được thực hiện. sem\_wait trả về 0 khi thành công và -1 khi lỗi. Trong trường hợp có lỗi, giá trị semaphore được giữ nguyên và errno được đặt thành số lỗi thích hợp. sem\_wait cung cấp lệnh gọi cho hoạt động P cho semaphore.

**sem\_trywait** giống như lời gọi sem\_wait, ngoại trừ: nếu giá trị semaphore bằng 0, nó không chặn mà trả về ngay lập tức với errno được đặt thành EAGAIN.

**sem\_timedwait** cũng giống như cuộc gọi sem\_wait, ngoại trừ: có bộ đếm thời gian được chỉ định với con trỏ, abs\_timeout. Nếu giá trị semaphore lớn hơn 0, nó sẽ bị giảm và giá trị thời gian chờ được trỏ bởi abs\_timeout không được sử dụng. Trong trường hợp đó, lời gọi hoạt động giống như lời gọi sem\_wait. Nếu giá trị semaphore bằng 0, các lời gọi bị chặn, thời lượng chặn tối đa là thời gian cho đến khi bộ hẹn giờ tắt. Nếu giá trị semaphore trở nên lớn hơn 0 trong khoảng thời gian chặn, semaphore bị giảm ngay lập tức và lời gọi được đánh thức trở lại. Nếu không, bộ hẹn giờ sẽ tắt và cuộc gọi trở lại với errno được đặt thành ETIMEDOUT. Bộ định thời được chỉ định trong struct timespec{};

**sem\_getvalue**

#include <semaphore.h>

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

**sem\_getvalue** nhận giá trị của semaphore được chỉ bởi sem. Giá trị được trả về trong số nguyên được chỉ bởi sval. Nó trả về 0 khi thành công và -1 khi lỗi, với lỗi không cho biết lỗi thực tế.

sem\_unlink

#include <semaphore.h>

int sem\_unlink (const char \*name);

**sem\_unlink** xoá semaphore có tên đang liên kết với name.

**POSIX Unnamed Semaphore calls**

Trong các đề cập ở trên, các semaphores là cục bộ của một quá trình; chúng chỉ được sử dụng bởi chủ đề của nó. Không có quá trình khác sử dụng chúng. Vì vậy, có vẻ như lãng phí nỗ lực để có tên semaphore trên toàn hệ thống và sử dụng các cuộc gọi như sem\_open. Có những semaphores không tên POSIX có thể làm những gì chúng ta cần một cách đơn giản và hiệu quả hơn nhiều. Đầu tiên, hệ thống gọi,

sem\_init

#include <semaphore.h>

int sem\_init (sem\_t \*sem, int pshared, unsigned int value);

**sem\_init** tương đương với **sem\_open** và dùng cho các semaphores không tên. Cần định nghĩa một biến kiểu sem\_t và chuyển đối số con trỏ của nó là sem trong lời gọi. Hoặc, có thể định nghĩa một con trỏ và cấp phát bộ nhớ động bằng cách sử dụng lời gọi malloc. sem\_init khởi tạo semaphore được trỏ bởi sem với giá trị số nguyên value. Đối số thứ hai được pshared cho biết semaphore này được dùng chia sẻ giữa các tiểu tình của một tiến trình hay giữa các tiến trình. Nếu pshared có giá trị 0, semaphore được chia sẻ giữa các tiểu trình của một tiến trình. Semaphore nên được đặt tại một nơi mà mọi tiểu trình có thể tìm thấy. Nếu pshared có giá trị khác 0, semaphore được chia sẻ bởi các tiến trình. Trong trường hợp đó, semaphore phải được đặt trong một phân đoạn bộ nhớ dùng chung được gắn với các tiến trình liên quan.

**sem\_destroy**

#include <semaphore.h>

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

**sem\_destroy** xoá semaphore không tên đang liên kết với name.

**Ví dụ:**

// C program to demonstrate working of Semaphores

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

sem\_t mutex;

void\* thread(void\* arg)

{

//wait

sem\_wait(&mutex);

printf("\nEntered..\n");

//critical section

sleep(4);

//signal

printf("\nJust Exiting...\n");

sem\_post(&mutex);

}

int main()

{

sem\_init(&mutex, 0, 1);

pthread\_t t1,t2;

pthread\_create(&t1,NULL,thread,NULL);

// sleep(2);

pthread\_create(&t2,NULL,thread,NULL);

pthread\_join(t1,NULL);

pthread\_join(t2,NULL);

sem\_destroy(&mutex);

return 0;

}

**Giải thuật Monte Carlo để tính số pi**

#include <stdlib.h>

#include <stdio.h>

#include <math.h>

#include <string.h>

#define SEED 35791246

main(int argc, char\* argv)

{

int niter=0;

double x,y;

int i,count=0; /\* # of points in the 1st quadrant of unit circle \*/

double z;

double pi;

printf("Enter the number of iterations used to estimate pi: ");

scanf("%d",&niter);

/\* initialize random numbers \*/

srand(SEED);

count=0;

for ( i=0; i<niter; i++) {

x = (double)rand()/RAND\_MAX;

y = (double)rand()/RAND\_MAX;

z = x\*x+y\*y;

if (z<=1) count++;

}

pi=(double)count/niter\*4;

printf("# of trials= %d , estimate of pi is %g \n",niter,pi);

}

**Bài tập:**

**Bài 1:**

Tạo 2 tiểu trình con, một tiểu trình in ra các số lẻ từ 1 đến 11, một tiểu trình in ra số chẵn từ 2 đến 10. Hãy sử dụng semaphore sau cho màn hình in ra dãy số đúng theo thứ tự từ 1 đến 11.

**Bài 2:**

Một cách giá trị π khá thú vị là sử dụng kỹ thuật Monte Carlo, liên quan đến ngẫu nhiên. Kỹ thuật này hoạt động như sau: Giả sử bạn có một vòng tròn bán kính là 1 nội tiếp trong một hình vuông cạnh là 2, như thể hiện trong hình sau:



* Đầu tiên, tạo một chuỗi các điểm ngẫu nhiên dưới dạng tọa độ (x, y) đơn giản. Những điểm này phải nằm trong tọa độ Descartes bị ràng buộc hình vuông. Trong tổng số điểm ngẫu nhiên được tạo, một số sẽ xảy ra trong vòng tròn.
* Tiếp theo, ước tính π bằng cách thực hiện phép tính sau: π = 4 × (số điểm trong vòng tròn) / (tổng số điểm)

Hãy viết một phiên bản đa luồng của thuật toán này để tạo ra **một tiểu trình riêng biệt** sinh ra một số lượng điểm ngẫu nhiên; sau đó tính số lượng điểm nằm trong hình và lưu trữ kết quả đó trong một biến toàn cục. Khi tiểu trình này kết thúc, tiến trình cha sẽ tính toán và xuất giá trị ước tính của π. Hãy đánh giá độ chính xác của số π với số lượng điểm ngẫu nhiên được tạo ra. Theo nguyên tắc, số lượng điểm càng lớn, giá trị tính càng tiến gần π.

Áp dụng semaphore và ví dụ ở trên để giải quyết bài toán trên.

**Bài 3:**

Một hãng sản xuất xe ô tô có các bộ phận hoạt động song song:

Bộ phận sản xuất khung xe

void SXKhung(){

printf(“San xuat khung”);

}

Bộ phận sản xuất bánh xe

void SXBanhXe(){

printf(“San xuat banh xe”);

}

Bộ phận lắp ráp: Sau khi có đủ 1 khung và 4 bánh thì tiến hành lắp ráp.

void LapRapXe(){

printf(“Lap rap xe”);

}

Hãy đồng bộ hoạt động của các bộ phận trên theo nguyên tắc: tại mỗi thời điểm chỉ cho phép sản xuất 1 khung xe, cần chờ đủ 4 bánh xe để gắn vào khung xe hiện tại này trước khi sản xuất một khung xe khác.

Wait—

Post++;