# 一、信号量的概念

信号量（信号灯）本质上是一个计数器，用于协调多个进程（包括但不限于父子进程）对共享数据对象的读/写。它不以传送数据为目的，主要是用来保护共享资源（共享内存、消息队列、socket连接池、数据库连接池等），保证共享资源在一个时刻只有一个进程独享。

信号量是一个特殊的变量，只允许进程对它进行等待信号和发送信号操作。最简单的信号量是取值0和1的二元信号量，这是信号量最常见的形式。

通用信号量（可以取多个正整数值）和信号量集方面的知识比较复杂，应用场景也比较少。

本文只介绍二元信号量。

# 二、相关函数

Linux中提供了一组函数用于操作信号量，程序中需要包含以下头文件：

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/sem.h>

## 1、semget函数

semget函数用来获取或创建信号量，它的原型如下：

int semget(key\_t key, int nsems, int semflg);

1）参数key是信号量的键值，typedef unsigned int key\_t，是信号量在系统中的编号，不同信号量的编号不能相同，这一点由程序员保证。key用十六进制表示比较好。

2）参数nsems是创建信号量集中信号量的个数，该参数只在创建信号量集时有效，这里固定填1。

3）参数sem\_flags是一组标志，如果希望信号量不存在时创建一个新的信号量，可以和值IPC\_CREAT做按位或操作。如果没有设置IPC\_CREAT标志并且信号量不存在，就会返错误（errno的值为2，No such file or directory）。

4）如果semget函数成功，返回信号量集的标识；失败返回-1，错误原因存于error中。

示例：

1）获取键值为0x5000的信号量，如果该信号量不存在，就创建它，代码如下：

int semid=semget(0x5000,1,0640|IPC\_CREAT);

2）获取键值为0x5000的信号量，如果该信号量不存在，返回-1，errno的值被设置为2，代码如下：

int semid= semget(0x5000,1,0640)；

## 2、semctl函数

该函数用来控制信号量（常用于设置信号量的初始值和销毁信号量），它的原型如下：

int semctl(int semid, int sem\_num, int command, ...);

1）参数semid是由semget函数返回的信号量标识。

2）参数sem\_num是信号量集数组上的下标，表示某一个信号量，填0。

3）参数cmd是对信号量操作的命令种类，常用的有以下两个：

**IPC\_RMID**：销毁信号量，不需要第四个参数；

**SETVAL**：初始化信号量的值（信号量成功创建后，需要设置初始值），这个值由第四个参数决定。第四参数是一个自定义的共同体，如下：

// 用于信号灯操作的共同体。

union semun

{

int val;

struct semid\_ds \*buf;

unsigned short \*arry;

};

4）如果semctl函数调用失败返回-1；如果成功，返回值比较复杂，暂时不关心它。

示例：

1）销毁信号量。

semctl(semid,0,IPC\_RMID);

2）初始化信号量的值为1，信号量可用。

union semun sem\_union;

sem\_union.val = 1;

semctl(semid,0,SETVAL,sem\_union);

## 3、semop函数

该函数有两个功能：1）等待信号量的值变为1，如果等待成功，立即把信号量的值置为0，这个过程也称之为等待锁；2）把信号量的值置为1，这个过程也称之为释放锁。

int semop(int semid, struct sembuf \*sops, unsigned nsops);

1）参数semid是由semget函数返回的信号量标识。

2）参数nsops是操作信号量的个数，即sops结构变量的个数，设置它的为1（只对一个信号量的操作）。

3）参数sops是一个结构体，如下：

struct sembuf

{

short sem\_num; // 信号量集的个数，单个信号量设置为0。

short sem\_op; // 信号量在本次操作中需要改变的数据：-1-等待操作；1-发送操作。

short sem\_flg; // 把此标志设置为SEM\_UNDO，操作系统将跟踪这个信号量。

// 如果当前进程退出时没有释放信号量，操作系统将释放信号量，避免资源被死锁。

};

示例：

1）等待信号量的值变为1，如果等待成功，立即把信号量的值置为0；

struct sembuf sem\_b;

sem\_b.sem\_num = 0;

sem\_b.sem\_op = -1;

sem\_b.sem\_flg = SEM\_UNDO;

semop(sem\_id, &sem\_b, 1);

2）把信号量的值置为1。

struct sembuf sem\_b;

sem\_b.sem\_num = 0;

sem\_b.sem\_op = 1;

sem\_b.sem\_flg = SEM\_UNDO;

semop(sem\_id, &sem\_b, 1);

# 三、示例程序

为了便于理解，我把信号量的操作封装成CSEM类，称之为信号灯，类似互斥锁，包括初始化信号灯、等待信号灯、挂出信号灯和销毁信号灯。

## 1、示例（book259.cpp）

/\*

\* 程序名：book259.cpp，此程序用于演示信号量的使用方法。

\*/

#include <stdio.h>

#include <string.h>

#include <unistd.h>

#include <errno.h>

#include <sys/ipc.h>

#include <sys/sem.h>

class CSEM

{

private:

union semun // 用于信号灯操作的共同体。

{

int val;

struct semid\_ds \*buf;

unsigned short \*arry;

};

int sem\_id; // 信号灯描述符。

public:

bool init(key\_t key); // 如果信号灯已存在，获取信号灯；如果信号灯不存在，则创建信号灯并初始化。

bool wait(); // 等待信号灯挂出。

bool post(); // 挂出信号灯。

bool destroy(); // 销毁信号灯。

};

int main(int argc, char \*argv[])

{

CSEM sem;

// 初始信号灯。

if (sem.init(0x5000)==false) { printf("sem.init failed.\n"); return -1; }

printf("sem.init ok\n");

// 等待信信号挂出，等待成功后，将持有锁。

if (sem.wait()==false) { printf("sem.wait failed.\n"); return -1; }

printf("sem.wait ok\n");

sleep(50); // 在sleep的过程中，运行其它的book259程序将等待锁。

// 挂出信号灯，释放锁。

if (sem.post()==false) { printf("sem.post failed.\n"); return -1; }

printf("sem.post ok\n");

// 销毁信号灯。

// if (sem.destroy()==false) { printf("sem.destroy failed.\n"); return -1; }

// printf("sem.destroy ok\n");

}

bool CSEM::init(key\_t key)

{

// 获取信号灯。

if ( (sem\_id=semget(key,1,0640)) == -1)

{

// 如果信号灯不存在，创建它。

if (errno==2)

{

if ( (sem\_id=semget(key,1,0640|IPC\_CREAT)) == -1) { perror("init 1 semget()"); return false; }

// 信号灯创建成功后，还需要把它初始化成可用的状态。

union semun sem\_union;

sem\_union.val = 1;

if (semctl(sem\_id,0,SETVAL,sem\_union) < 0) { perror("init semctl()"); return false; }

}

else

{ perror("init 2 semget()"); return false; }

}

return true;

}

bool CSEM::destroy()

{

if (semctl(sem\_id,0,IPC\_RMID) == -1) { perror("destroy semctl()"); return false; }

return true;

}

bool CSEM::wait()

{

struct sembuf sem\_b;

sem\_b.sem\_num = 0;

sem\_b.sem\_op = -1;

sem\_b.sem\_flg = SEM\_UNDO;

if (semop(sem\_id, &sem\_b, 1) == -1) { perror("wait semop()"); return false; }

return true;

}

bool CSEM::post()

{

struct sembuf sem\_b;

sem\_b.sem\_num = 0;

sem\_b.sem\_op = 1;

sem\_b.sem\_flg = SEM\_UNDO;

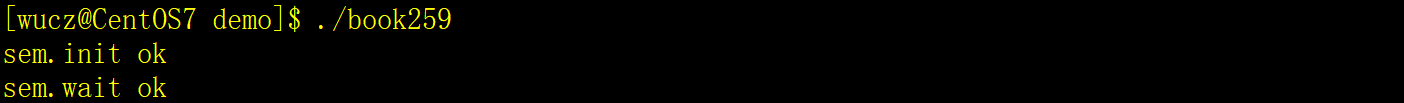
if (semop(sem\_id, &sem\_b, 1) == -1) { perror("post semop()"); return false; }

return true;

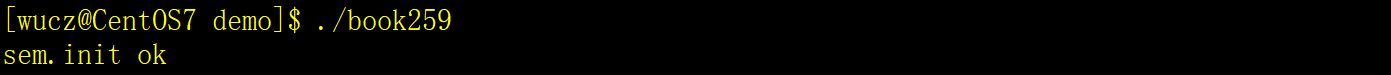
}

## 2、测试方法

第一步：运行book259程序，它会创建键值为5000的信号灯，并持有锁，如下：



第二步：立即再运行一个book259程序，它会获取键值为5000的信号灯，并等待锁，如下：

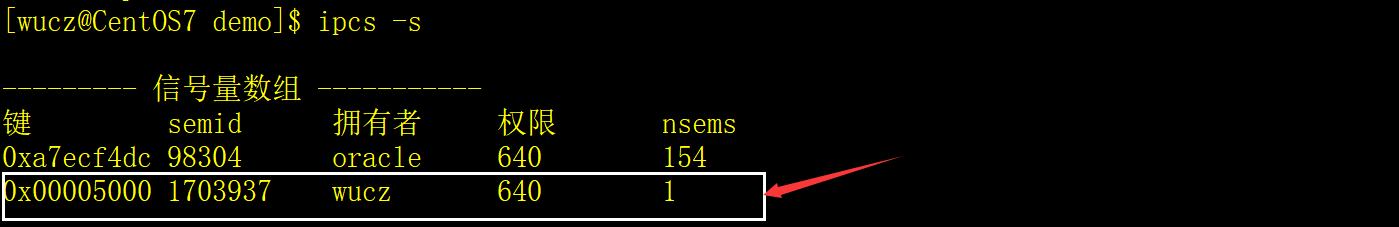


第三步，当第一次运行的book259程序sleep完50秒之后，释放锁，第二个运行book259的程序将获得锁；

第四步，可以启动更多的book259程序，它们将排队等待锁。

# 四、其它的操作命令

用ipcs -s可以查看系统的信号量，内容有键值（key），信号量编号（semid），创建者（owner），权限（perms），信号量数（nsems）。



用ipcrm sem 信号量编号，可以手工删除信号量，如下：

