## 信号灯/信号量（semaphore）

概念：是不同进程间或一个给定进程内部不同线程间同步的机制。类似我们的

PV操作概念：

生产者和消费者场景

Ｐ(Ｓ) 含义如下:

if (信号量的值大于0) {

申请资源的任务继续运行；

信号量的值减一；

} else {

申请资源的任务阻塞；

}

V(Ｓ) 含义如下:

信号量的值加一；

if (有任务在等待资源) {

唤醒等待的任务，让其继续运行

}

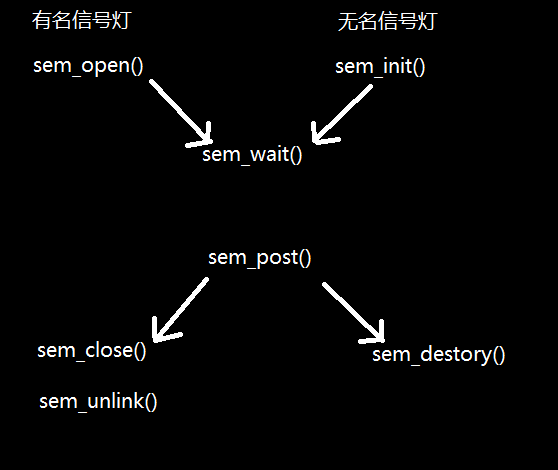
**三种信号灯：**

Posix 有名信号灯

Posix 无名信号灯 （linux只支持线程同步）

System V 信号灯

Posix 有名信号灯和无名信号灯使用：



有名信号灯打开：

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

sem\_t \*sem\_open(const char \*name, int oflag,mode\_t mode, unsigned int value);

参数：

name：name是给信号灯起的名字

oflag：打开方式，常用O\_CREAT

mode：文件权限。常用0666

value：信号量值。二元信号灯值为1，普通表示资源数目

信号灯文件位置：/dev/shm

有名信号灯关闭

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

有名信号灯的删除

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

无名信号灯初始化

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

参数：

sem：需要初始化的信号灯变量

shared: shared指定为0，表示信号量只能由初始化这个信号量的进程使用，不能在进程间使用,linux 不支持进程间同步。

Value:信号量的值

无名信号灯销毁

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

信号灯P操作

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

获取资源，如果信号量为0，表示这时没有相应资源空闲，那么调用线程就将挂起，直到有空闲资源可以获取

信号灯V操作

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

释放资源，如果没有线程阻塞在该sem上，表示没有线程等待该资源，这时该函数就对信号量的值进行增1操作，表示同类资源多增加了一个。如果至少有一个线程阻塞在该sem上，表示有线程等待资源，信号量为0，这时该函数保持信号量为0不变，并使某个阻塞在该sem上的线程从sem\_wait函数中返回

注意：编译posix信号灯需要加pthread动态库。

System V 信号灯使用：

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

功能：创建/打开信号灯

参数：key：ftok产生的key值（和信号灯关联的key值）

　　　　　　 nsems：信号灯集中包含的信号灯数目

　　　　　　 semflg：信号灯集的访问权限，通常为IPC\_CREAT |0666

返回值：成功：信号灯集ID ； 失败：-1

int semop ( int semid, struct sembuf \*opsptr, size\_t nops);

功能：对信号灯集合中的信号量进行P - V操作

参数：semid：信号灯集ID

　　　struct sembuf {

　　　　short sem\_num; // 要操作的信号灯的编号

　　　　short sem\_op; // 1 : 释放资源，V操作

// -1 : 分配资源，P操作

　　　　short sem\_flg; // 0（阻塞）,IPC\_NOWAIT, SEM\_UNDO

　　　};//对某一个信号灯的操作，如果同时对多个操作，则需要定义这种结构体数组

　　　nops: 要操作的信号灯的个数 ，1个

返回值：成功 ：0 ； 失败：-1

int semctl ( int semid, int semnum, int cmd…/\*union semun arg\*/);

功能：信号灯集合的控制（初始化/删除）

参数：semid：信号灯集ID

　　　semnum: 要操作的集合中的信号灯编号

　　　cmd：

　　　GETVAL：获取信号灯的值，返回值是获得值

　　　SETVAL：设置信号灯的值，需要用到第四个参数：共用体

　　　IPC\_RMID：从系统中删除信号灯集合

返回值：成功 0 ； 失败 -1