POSIX 信号量

信号量 (Semaphore)

- 信号量这一概念最早由 Dijkstra 提出（这位巨佬对计算机领域的贡献真的是数都数不过来），主要用于进程间的同步，是一个同步原语
- **简单的来看，信号量就是一个位于内核中的整数，其值永远不会小于 0。如果一个进程试图将信号量的值减少至小于 0，那么内核会阻塞该操作，直到信号量增长到允许执行该操作的程度**
- 从上面的概念上来看，信号量既可以使得一组进程/线程对同一个资源进行访问，也可以限定只有单个进程/线程对同一个资源进行访问，此时信号量的值只有 2 个: 0 和 1，那么也常被称为二元信号量
- Example
  - 有时候我们去吃海底捞，人多桌位少，那么当桌位坐满时，后面新来的只能在外边儿折千纸鹤（阻塞）
  - 有人吃完了，腾出来一张桌子，后面儿的人可以去就餐，并且你不需要时时刻刻的去看有没有人走了，服务员会在有空位的时候叫你（通知）

  桌位的数量其实就是一个信号量，每当有人落位，信号量就会减 1
- 我们可以把信号量比作互斥锁，只不过信号量支持多个进程/线程获取该信号量，互斥锁在同一时候只能有一个线程获取到

命名信号量

基本使用

POSIX 信号量其实有两种: 命名信号量和未命名信号量，这两个东西的唯一区别就是多个进程如何找到该信号量而已

- 命名信号量 — 通过唯一的名字来找到信号量
- 未命名信号量 — 只能在共享内存中共享

创建并打开一个信号量

```
sem_t *sem_open(const char *name, int oflag, ...
                /* mode_t mode, unsigned int value */);
```

name 必须以 '/' 开头，例如 "/my_sem"

oflag 表明我们到底是要创建新的信号量，还是打开一个既有的信号量

- sem_open("/mysem", O_CREAT) — 新建并打开
- sem_open("/mysem", 0) — 打开既有信号量

value 则是信号量初始化时的值，可以看到是一个无符号整型

关闭信号量

```
int sem_close(sem_t *sem);
```

当一个进程打开一个命名信号量时，系统会记录进程与信号量之间的关联关系。em_close() 函数会终止这种关联关系(即关闭信号量)，释放系统为该进程关联到该信号量之 上的所有资源，并递减引用该信号量的进程数

关闭一个信号量并不会删除这个信号量，而要删除信号量则需要使用 sem_unlink()

等待一个信号量

```
int sem_wait(sem_t *sem);
```

"等待"一词可能不太准确，我个人认为"减少一个信号量的值"可能会更贴切一些

如果信号量的当前值大于 0，那么 sem_wait() 会立即返回。如果信号量的当前值等于 0， 那么 sem_wait() 会阻塞直到信号量的值大于 0 为止，当信号量值大于 0 时该信号量值就被递减并且 sem_wait()会返回

sem_trywait() 和 sem_timedwait() 是等待一个信号量的非阻塞和带有超时时间的版本

发布一个信号量

```
int sem_post(sem_t *sem);
```

"发布"一词可能不太准确，我个人认为"增加一个信号量的值"可能会更贴切一些

递增 sem 所引用的信号量的值。如果在递增之前信号量的值为 0 的话，并且其他某个进程(或线程)正在因等待递减这个信号量而阻塞，那么该进程会被唤醒

和互斥锁的对比

权属

互斥锁是有权属的，也就是线程 A 不能去解锁线程 B 获取的锁 — (Golang 就可以，滑稽)

但是信号量是共享的，谁都可以来"掺和一脚"，一个线程能够递增一个被另一个线程递减的信号量，我们甚至可以在程序外部来做这件事情

异步信号安全

信号量是异步信号安全的，而 Pthreads 的一系列 API 均不是异步信号安全的