ROS信号量与互斥量

# 信号量概述

# 信号量概念

在操作系统中，信号量用以表示物理资源的实体，它是一个与队列有关的整型变量。实现时，信号量是一种变量类型，常常用一个记录型数据结构表示，它有两个分量：一个是信号量的值，另一个是信号量队列的队列指针。（摘自《操作系统教程》（第三版） 孙钟秀主编P167）

信号量(Semaphore)，有时被称为信号灯，是在多线程环境下使用的一种设施, 它负责协调各个线程, 以保证它们能够正确、合理的使用公共资源。

　　以一个停车场的运作为例。简单起见，假设停车场只有三个车位，一开始三个车位都是空的。这时如果同时来了五辆车，看门人允许其中三辆直接进入，然后放下车拦，剩下的车则必须在入口等待，此后来的车也都不得不在入口处等待。这时，有一辆车离开停车场，看门人得知后，打开车拦，放入外面的一辆进去，如果又离开两辆，则又可以放入两辆，如此往复。在这个停车场系统中，车位是公共资源，每辆车好比一个线程，看门人起的就是信号量的作用。

# 信号量分类

信号量按其用途可分为两种：

公用信号量：联系一组并发进程，相关的进程均可在此信号量上执行 P 和 V

操作。初值常常为 1，用于实现进程互斥。

私有信号量：联系一组并发进程，仅允许此信号量拥有的进程执行 P 操作，

而其他相关进程可在其上施行 V 操作。初值常常为 0 或正整数，多用于并发

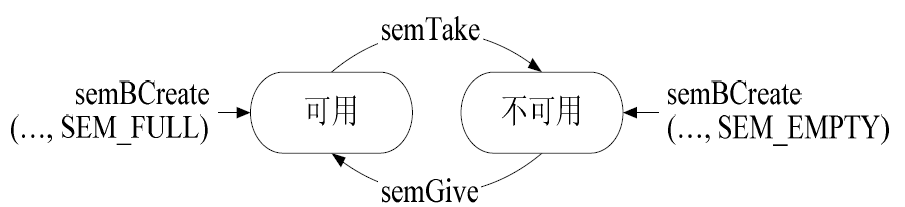
进程同步。

信号量按其取值可分为两种：

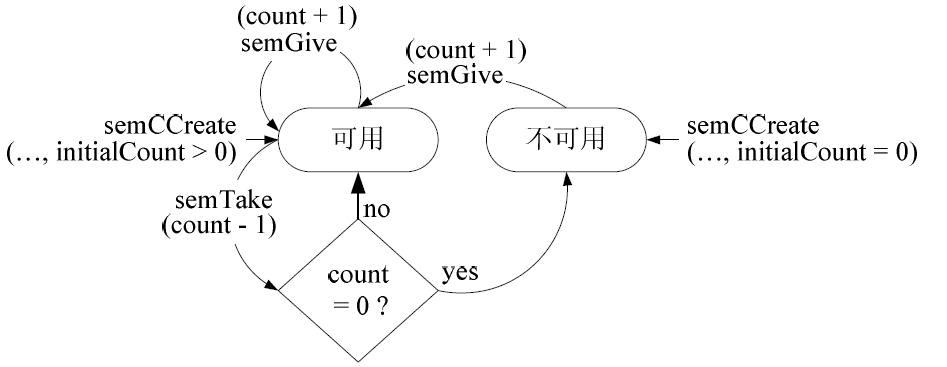
二元信号量：仅允许取值为 0 和 1，主要用于解决进程互斥问题。

一般信号量：允许取值为非负整数，主要用于解决进程同步问题。

二值信号量的状态图如下：



计数信号量的状态图如下：

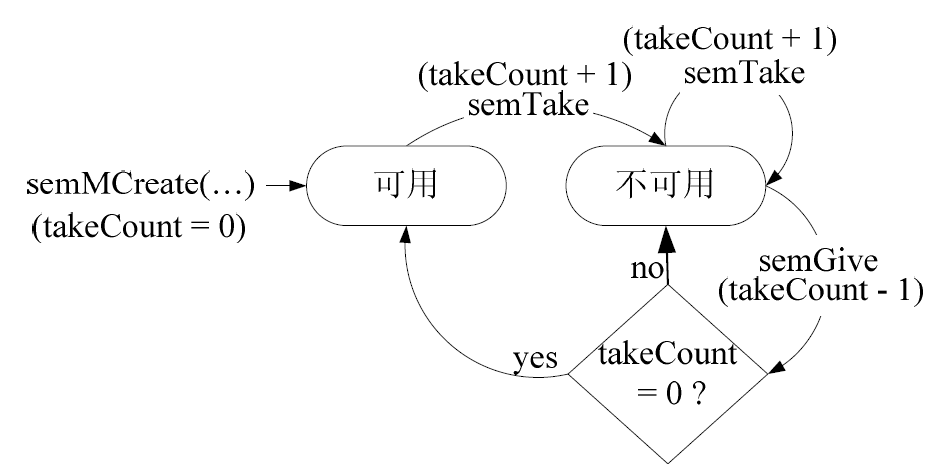


互斥锁(mutex) （互斥信号量）是一种比较简单的、用于实现线程间对资源互斥访问的机制。由于操作互斥锁的时间和空间开销都较低，因而较适合于高频度使用的关键共享数据和程序段。互斥锁可以有两种状态，即开锁(unlock)和关锁(lock)状态。相应地，可用两条命令(函数)对互斥锁进行操作。其中的关锁lock操作用于将 mutex关上，开锁操作 unlock则用于打开 mutex。 当一个线程需要读/写一个共享数据段时，线程首先应为该数据段所设置的 mutex 执行关锁命令。命令首先判别 mutex 的状态，如果它已处于关锁状态，则试图访问该数据段的线程将被阻塞；而如果 mutex 是处于开锁状态，则将 mutex 关上后便去读/写该数据段。在线程完成对数据的读/写后，必须再发出开锁命令将 mutex 打开，同时还须将阻塞在该互斥

锁上的一个线程唤醒，其它的线程仍被阻塞在等待mutex打开的队列上。

另外，为了减少线程被阻塞的机会，在有的系统中还提供了一种用于 mutex 上的操作命令 Trylock。当一个线程在利用 Trylock 命令去访问 mutex 时，若 mutex 处于开锁状态，Trylock 将返回一个指示成功的状态码；反之，若 mutex 处于关锁状态，则 Trylock 并不会阻塞该线程，而只是返回一个指示操作失败的状态码。

互斥信号量的状态图：



# 信号量横向比较

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 动作\系统 | win32 | POSIX(Linux) | VxWorks | ROS |
| 创建 | CreateSemaphore | sem\_init  sem\_open | semBCreate  semCCreate | ROS\_SemCreate |
| 等待 | WaitForSingleObject | sem \_wait | semTake | ROS\_SemP |
| 释放 | ReleaseMutex | sem \_post | semGive | ROS\_SemV |
| 试图等待 | WaitForSingleObject | sem \_trywait | semGive  (NO\_WAIT) | ROS\_SemP  （NO\_WAIT） |
| 销毁 | CloseHandle | sem\_destroy  sem\_unlink | semDelete | ROS\_SemDelete |

信号量封装的不同点：

1、POSIX接口提供的带超时的信号量等待接口受Linux版本影响，而VxWorks的semTak即可形式默认支持等待超时。

int sem\_timedwait(sem\_t \*sem, const struct timespec \*abs\_timeout);

# 互斥锁横向比较

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 动作  系统 | win32 | POSIX(Linux) | VxWorks | ROS |
| 创建 | CreateMutex | pthread\_mutex\_init | semMCreate | ROS\_SemCreate |
| 加锁 | WaitForSingleObject | pthread\_mutex\_lock | semTake | ROS\_SemP |
| 解锁 | ReleaseMutex | pthread\_mutex\_unlock | semGive | ROS\_SemV |
| 销毁 | CloseHandle | pthread\_mutex\_destroy | semDelete | ROS\_SemDelete |

二值信号量与互斥量两者的相同点是，都是用于互斥。互斥信号量信号量是二进制信号量的子集。不同点如下：

1、互斥信号量必须是同一个任务申请和释放，其他任务释放无效。 二进制信号量，一个任务申请成功后，可以由另一个任务释放。

2、同一个任务可以递归申请。

3、提供解决优先级翻转的协议支持。

  二进制信号量实现任务互斥举例：

有一台打印机资源，abc三个任务共享，当a取得使用权后，为了防止其他任务错误地释放了信号量()，必须将打印机房的门关起来(进入临界段)，用完后，释放信号量，再把门打开(出临界段)，其他任务再进去打印。(而互斥型信号量由于必须由取得信号量的那个任务释放，故不会出现其他任务错误地释放了信号量的情况出现，故不需要有临界段。)

# VxWorks与POSIX同步互斥

通过阅读VxWorks操作系统文档和POSIX接口文档，发现VxWorks系统信号量功能与三个POSIX标准接口（信号量、互斥锁、条件变量）等效。得出此结论的原因如下：

VxWorks系统信号量包括二值信号量、计数信号量、互斥信号量三种实现，除创建接口外，P、V、删除操作接口一致。其中互斥信号量带有删除安全、允许递归、优先级继承等特性。而相对VxWorks系统信号量接口的定义，POSIX接口定义复杂且定义接口的功能单纯。

* POSIX信号量接口分有名（用于进程间和线程间通信）和无名信号量（线程间通信），是二值信号量和计数信号量的实现，且该接口的指定超时的sem\_timedwait接口不是所有Linux的版本支持。VxWorks 单个semTake信号量获取接口完成了POSIX的sem\_timedwait与sem\_wait两个接口的功能。
* POSIX互斥锁只能用于线程互斥，操作只有pthread\_mutex\_lock和pthread\_mutex\_unlock，用于临界区的短暂保护，为了克服死锁和线程同步长时间等待等问题，而推出条件变量作为补充。互斥锁和条件变量的组合的功能（这是当前ROS 5.2实现方案）与Vxworks信号量等待超时功能相同。

为了生动说明互斥锁和条件变量的相互功能，举例如下：

正在进行的篮球比赛中，篮球好比临界资源，互斥锁好比正在带球队员，队员间的传球好比条件变量，条件变量与互斥锁成对出现，条件变量完成线程间长时间等待而互斥锁完成临界区短时间的保护。

# POSIX互斥同步机制的ROS封装方案

由于POSIX定义的同步互斥机制与VxWorks系统提供的机制接口封装相差很大，而ROS实现的信号量接口与VxWorks系统的接口相同，为了避免基于ROS开发的应用改动，现推出ROS封装POSIX接口的解决方案：

继续沿用ROS5.2当前的实现，使用互斥锁和条件变量模拟VxWorks系统信号量的操作。ROS信号量不使用POSIX信号量封装

产品接口中有很多递归调用的信号量的模块，此信号量接口需要依赖互斥锁的机制，封装互斥锁为ROS的互斥信号量用ROS\_SEM\_TYPE\_MUTEX创建。（由于POSIX接口的限制，互斥锁没有等待超时，ROS封装ROS\_SemP中指定的超时会失效）。

暂时不提供读写锁的封装。