目录

[1.线程互斥锁(mutex) 2](#_Toc200967666)

[1.1创建互斥锁 2](#_Toc200967667)

[1.2销毁互斥锁 2](#_Toc200967668)

[1.3获取互斥锁 2](#_Toc200967669)

[1.4尝试获取互斥锁 2](#_Toc200967670)

[1.5解锁 2](#_Toc200967671)

[2.线程读写锁(rwlock) 2](#_Toc200967672)

[2.1初始化读写锁 2](#_Toc200967673)

[2.2销毁读写锁 2](#_Toc200967674)

[2.3获取读锁 2](#_Toc200967675)

[2.4获取写锁 3](#_Toc200967676)

[2.5解锁 3](#_Toc200967677)

[2.6死锁 3](#_Toc200967678)

[3.线程条件变量(Condition Variables) 4](#_Toc200967679)

[3.1初始化条件变量 4](#_Toc200967680)

[3.2销毁条件变量 4](#_Toc200967681)

[3.3等待条件变量 4](#_Toc200967682)

[3.4发送信号 4](#_Toc200967683)

[4.信号量(Semaphore) 4](#_Toc200967684)

[4.1初始化信号量 5](#_Toc200967685)

[4.2销毁信号量 5](#_Toc200967686)

[4.3等待信号量(P 操作) 5](#_Toc200967687)

[4.4尝试等待信号量(非阻塞的 P 操作) 5](#_Toc200967688)

[4.5释放信号量(V 操作) 5](#_Toc200967689)

[4.6获取信号量的当前值 6](#_Toc200967690)

## 1.线程互斥锁(mutex)

### 1.1创建互斥锁

int pthread\_mutex\_init(pthread\_mutex\_t \*mutex, const pthread\_mutexattr\_t \*attr);

### 1.2销毁互斥锁

int pthread\_mutex\_destory(pthread\_mutex\_t \*mutex);

### 1.3获取互斥锁

int pthread\_mutex\_lock(pthread\_mutex\_t \*mutex);

### 1.4尝试获取互斥锁

int pthread\_mutex\_trylock(pthread\_mutex\_t \*mutex);

### 1.5解锁

int pthread\_mutex\_unlock(pthread\_mutex\_t \*mutex);

5个函数的返回值都是成功返回0，失败返回错误号

## 2.线程读写锁(rwlock)

存在读锁时，可以再加读锁，不能加写锁。存在写锁时，不能再加任何锁。

### 2.1初始化读写锁

pthread\_rwlock\_init()：初始化一个新的读写锁对象。

int pthread\_rwlock\_init(pthread\_rwlock\_t \*restrict rwlock, const pthread\_rwlockattr\_t \*restrict attr);

如果 attr 是 NULL，则使用默认属性。

### 2.2销毁读写锁

pthread\_rwlock\_destroy()：销毁一个读写锁对象，释放其占用的所有资源。

int pthread\_rwlock\_destroy(pthread\_rwlock\_t \*rwlock);

### 2.3获取读锁

pthread\_rwlock\_rdlock()：获取读锁，如果已经有写锁被其他线程持有，则该调用将会阻塞直到写锁被释放。

int pthread\_rwlock\_rdlock(pthread\_rwlock\_t \*rwlock);

pthread\_rwlock\_tryrdlock()：尝试获取读锁，但如果当前无法获取锁(例如已有写锁)，则立即返回而不是等待。

int pthread\_rwlock\_tryrdlock(pthread\_rwlock\_t \*rwlock);

### 2.4获取写锁

pthread\_rwlock\_wrlock()：获取写锁，如果有任何其他读锁或写锁存在，该调用将会阻塞直到所有锁都被释放。

int pthread\_rwlock\_wrlock(pthread\_rwlock\_t \*rwlock);

pthread\_rwlock\_trywrlock()：尝试获取写锁，但如果当前无法获取锁，则立即返回而不是等待。

int pthread\_rwlock\_trywrlock(pthread\_rwlock\_t \*rwlock);

### 2.5解锁

pthread\_rwlock\_unlock()：释放当前持有的读锁或写锁。

int pthread\_rwlock\_unlock(pthread\_rwlock\_t \*rwlock);

### 2.6死锁

死锁是指在计算机系统中，两个或多个进程(或线程)无限期地等待永远不会发生的条件的情况。

在这种情况下，每个涉及的进程或线程都在等待另一个进程或线程完成某项操作，而这些操作本身又依赖于当前进程或线程的状态，从而形成一个循环等待链，导致所有相关的进程或线程都无法继续执行。

死锁的发生需要满足以下四个条件：

1.非共享资源：至少有一个资源必须以不可共享的方式使用，即一次只能由一个进程或线程使用。

2.占有并等待：一个进程已经占有了资源，并且正在等待被其他进程占用的资源

3.非抢夺资源：资源不能被强制从其他进程中夺走，只能由该进程自行释放资源。

4.循环等待：存在一系列等待资源的进程 {P1, P2, ..., PN}，P1 等待 P2 的资源，P2 等待 P3 的资源，...，PN 等待 P1 的资源，形成一个闭合的循环。

示例：

假设有两个线程 A 和 B，以及两个资源 R1 和 R2，线程 A 锁定了 R1，线程 B 锁定了 R2。

如果此时线程 A 试图锁定 R2 ，同时线程 B 试图锁定 R1 ，那么这两个线程就会陷入死锁状态，因为它们都将永远等待对方释放所需的资源。

## 3.线程条件变量(Condition Variables)

条件变量允许线程在某些条件未满足时等待，并在其他线程改变了该条件后得到通知并继续执行。

条件变量通常与互斥锁(mutex)一起使用，以确保对共享资源的访问是线程安全的。

### 3.1初始化条件变量

pthread\_cond\_init(pthread\_cond\_t \*cond, const pthread\_condattr\_t \*attr);

### 3.2销毁条件变量

pthread\_cond\_destroy(pthread\_cond\_t \*cond);

### 3.3等待条件变量

pthread\_cond\_wait(pthread\_cond\_t \*restrict cond, pthread\_mutex\_t \*restrict mutex);

解锁互斥锁并使当前线程进入等待状态，直到被另一个线程唤醒。函数执行的最后重新锁定互斥锁。

为什么要先解锁最后又锁上？

条件不满足，解开锁让其他线程对条件进行操作。操作完成收到提醒后重新上锁。

pthread\_cond\_timedwait(pthread\_cond\_t \*restrict cond,

pthread\_mutex\_t \*restrict mutex,

const struct timespec \*restrict abstime);

类似于 pthread\_cond\_wait()，但带有超时功能。如果指定的时间到了而没有收到信号，则该函数会自动返回。

### 3.4发送信号

pthread\_cond\_signal(pthread\_cond\_t \*cond);

唤醒至少一个正在等待指定条件变量的线程。如果有多个线程在等待同一个条件变量，则随机唤醒线程

pthread\_cond\_broadcast(pthread\_cond\_t \*cond);：唤醒所有正在等待指定条件变量的线程。

## 4.信号量(Semaphore)

二进制信号量(即互斥锁mutex)

用途：主要用于实现互斥，即确保同一时间只有一个线程可以访问某个资源。

行为：值只能是0或1。当信号量的值为1时，表示资源可用；当值为0时，表示资源已被占用。

计数信号量

用途：通常用来限制同时访问某一资源的线程或进程的数量。

行为：表示当前可用资源的数量。每当一个线程获取资源时，信号量的值减1；当一个线程释放资源时，信号量的值加1。

信号量通常提供以下两种基本操作：

P 操作(wait 或 down)：尝试**减少信号量**的值，即尝试占有资源

如果信号量的值大于0，则将其减1，并允许继续执行。

如果信号量的值等于0，则将调用线程阻塞，直到其他线程执行V操作使得信号量的值增加。

V 操作(signal 或 up)：**增加信号量**的值，即释放资源

如果有因P操作而被阻塞的线程，则唤醒其中一个等待的线程。

### 4.1初始化信号量

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

pshared 决定信号量是否可以在进程间共享(0表示不共享，非0表示共享。0时线程同步，非0时进程同步)

### 4.2销毁信号量

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

### 4.3等待信号量(P 操作)

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

获取资源，如果没有资源则 等待

如果信号量的值大于0，则减1并返回；如果值为0，则阻塞直至信号量变为正数。

### 4.4尝试等待信号量(非阻塞的 P 操作)

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

获取资源，如果没有资源则 直接返回

类似于 sem\_wait()，但如果信号量的值为0，则立即返回错误而不是阻塞。

### 4.5释放信号量(V 操作)

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

释放资源

增加信号量的值，并唤醒一个正在等待此信号量的线程(如果有)

### 4.6获取信号量的当前值

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