3.3.5 信号量（Semaphore）

1965 年，E.W.Dijkstra 提出了信号量的概念，之后信号量即成为操作系统实现互斥和同步的一种普遍机制。信号量是包含一个非负整型变量，并且带有两个原子操作 wait 和 signal。wait 还可以被称为 down、P 或 lock，signal 还可以被称为 up、V、unlock 或 post。

如果信号量的非负整形变量 S 大于零，wait 就将其减 1，如果 S 等于 0，wait 就将调用线程挂起。对于 signal 操作，如果有线程在信号量上阻塞(此时 S 等于 0)，signal 就会解除对某个等待线程的阻塞，使其从 wait 中返回，如果没有线程阻塞在信号量上，signal 就将 S 加 1。

由此可见，S 可以被理解为一种资源的数量，信号量即是通过控制这种资源的分配来实现互斥和同步的。如果把 S 设为 1，信号量即可实现互斥量的功能。如果 S 的值大于 1，那么信号量即可使多个线程并发运行。另外，信号量不仅允许使用者申请和释放资源，而且还允许使用者创造资源，这就赋予了信号量实现同步的功能。可见，信号量的功能要比互斥量丰富许多。

POSIX 信号量是一个 sem_t 类型的变量，但 POSIX 有两种信号量的实现机制：无名信号量和命名信号量。无名信号量可以用在共享内存的情况下，比如实现进程中各个线程之间的互斥和同步。命名信号量通常用于不共享内存的情况下，比如不共享内存的进程之间。

使用本节所介绍的函数和数据结构需要引用头文件 semaphore.h。

3.3.5.1 POSIX 无名信号量

在使用信号量之前，必须对其进行初始化。sem_init 函数初始化指定的信号量，它的形式为：

```
int sem_init(sem_t *sem, int pshared, unsigned value);
```

参数 sem 指向要初始化的信号量，参数 value 为信号量的初始值。参数 pshared 用于说明信号量的共享范围，如果 pshared 为 0，那么该信号量只能由初始化这个信号量的进程中的线程使用，如果 pshared 非零，任何可以访问到这个信号量的进程都可以使用这个信号量。如果成功，sem_init 返回 0，如果不成功，sem_init 返回-1 并设置 errno。下表列出了，sem_init 可能发生的错误和对应得错误码。

错误	原因
EINVAL	value 大于 SEM_VALUE_MAX
ENOSPC	初始化资源已经耗尽，或者信号量的数目超出了 SEM_NSEMS_MAX 的范围
EPERM	调用程序没有适当的特权

表 3.13 sem_init 可能发生的错误和对应得错误码

函数 sem_destroy 销毁一个指定的信号量，它的形式为：

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

参数 sem 为指向要销毁的信号量的指针。如果成功，sem_destroy 返回 0，如果不成功，sem_destroy 返回-1 并设置 errno。如果*sem 不是有效的信号量，sem_destroy 就将 errno 置为 EINVAL。

sem_post 函数实现对指定信号量的 signal 操作，它的形式为：

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

如果成功，sem_post 返回 0。如果不成功，sem_post 返回-1 并设置 errno。如果*sem 不是有效的信号量，sem_post 就将 errno 置为 EINVAL。

sem_wait 函数实现对指定信号量的 wait 操作。sem_trywait 函数与 sem_wait 类似，只是在试图对一个为零的信号量进行操作时，它不会阻塞调用线程，而是立即返回，类似于互斥量操作中的 pthread_mutex_trylock()。这两个函数的形式为：

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

如果成功，这两个函数返回 0，如果不成功，这些函数返回-1 并设置 errno。如果*sem 不是有效的信号量，sem_ wait 就将 errno 置为 EINVAL。如果 sem_trywait 在信号量为零时执行 wait 操作，则将 errno 置为 EAGAIN。

sem_post，sem_wait 和 sem_trywait 同样可用于命名信号量。

3.3.5.2 POSIX 命名信号量

之所以称为命名信号量，是因为它有一个名字、一个用户 ID、一个组 ID 和权限，这些是提供给不共享内存的那些进程使用命名信号量的接口。命名信号量的名字是一个遵守路径名构造规则的字符串。

sem_open 函数用于创建或打开一个命名信号量。它的形式为：

```
sem_t *sem_open(const char *name, int oflag);
```

参数 name 是一个标识信号量的字符串。参数 oflag 用来确定是创建信号量还是连接已有信号量。如果设置了 oflag 的 O_CREAT 比特位，则会创建一个新的信号量。

sem_close 函数用于关闭命名信号量。它的形式为：

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

参数 sem 是指向要关闭的信号量的指针。单个程序可以用 sem_close 函数关闭命名信号量，但是这样做并不能将信号量从系统中删除，因为命名信号量在单个程序的执行之外是具有持久性的。当进程调用_exit、exit、exec 或从 main 返回时，进程打开的命名信号量同样会被关闭。如果成功，sem_ close 返回 0，如果不成功，sem_ close 返回-1 并设置 errno。如果*sem 不是有效的信号量，sem_ close 就将 errno 置为 EINVAL。

sem_unlink 函数用于在所有进程关闭了命名信号量之后，将信号量从系统中删除。它的形式为：

```
int sem_unlink(const char *name);
```

参数 name 为要删除的命名信号量的名字。如果成功，sem_ unlink 返回 0，如果不成功，sem_ unlink 返回-1 并设置 errno。下表列出了，sem_ unlink 可能发生的错误和对应得错误码。

错误	原因
EACCES	权限不正确
ENAMETOOLONG	name 比 PATH_NAME 长，或者它有一个组件超出了 NAME_MAX 的范围
ENOENT	信号量不存在

表 3.14 sem_ unlink 可能发生的错误和对应得错误码

3.3.5.3 信号量应用实例

下面是一个无名信号量的应用实例，它完成的工作为：创建两个线程，这两个线程各自将自己的一个整型变量 i 从 1 递增到 100，并通过信号量控制递增的过程，即这两个整型变量的差不能超过 5。

代码 3.7 信号量应用实例

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#define UPBOUND 100
sem_t sem1;
sem_t sem2;

void *threadfunc1(void *pvoid)
{
    int i = 0;
    while(i < UPBOUND)
    {
        sem_wait(&sem1);
        i++;
```

```
        printf("The integer of child thread1 is %d now.\n", i);
        sem_post(&sem2);
    }
}

void *threadfunc2(void *pvoid)
{
    int i = 0;
    while(i < UPBOUND)
    {
    sem_wait(&sem2);
    i++;
    printf("The integer of child thread2 is %d now.\n", i);
    sem_post(&sem1);
}
return (NULL);
}
int main()
{
    pthread_t tid1, tid2;

    int i = sem_init(&sem1, 0, 5);
    printf("%d\n", i);
    sem_init(&sem2, 0, 5);
    printf("Semaphores are initialized.\n");

    pthread_create(&tid1, NULL, &threadfunc1, NULL);
    pthread_create(&tid2, NULL, &threadfunc2, NULL);
    printf("Threads are started.\n");

    pthread_join(tid1, NULL);
    pthread_join(tid2, NULL);
    printf("Threads finished.\n");

    sem_destroy(&sem1);
    sem_destroy(&sem2);
    printf("Semaphores are destoried.\n");

    printf("Main thread finished\n");
    return 0;
}
```