# 实验1：进程同步

## 实验要求

在Linux环境下，创建一个包含n 个线程的控制进程。用这n 个线程来表示n个读者或写者。每个线程按相应测试数据文件的要求，进行读写操作。请用信号量机制分别实现读者优先和写者优先的读者-写者问题。

读者-写者问题的读写操作限制：

1）写-写互斥；

2）读-写互斥；

3）读-读允许；

读者优先的附加限制：如果一个读者申请进行读操作时已有另一读者正在进行读操作，则该读者可直接开始读操作。

写者优先的附加限制：如果一个读者申请进行读操作时已有另一写者在等待访问共享资源，则该读者必须等到没有写者处于等待状态后才能开始读操作。

运行结果显示要求：要求在每个线程创建、发出读写操作申请、开始读写操作和结束读

写操作时分别显示一行提示信息，以确信所有处理都遵守相应的读写操作限制。

## **测试数据文件格式**

测试数据文件包括n行测试数据，分别描述创建的n个线程是读者还是写者，以及读写操作的开始时间和持续时间。

每行测试数据包括四个字段，各字段间用空格分隔。

第一字段为一个正整数，表示线程序号。

第二字段表示相应线程角色，R 表示读者是，W 表示写者。

第三字段为一个正数，表示读写操作的开始时间。线程创建后，延时相应时间（单位为秒）后发出对共享资源的读写申请。

第四字段为一个正数，表示读写操作的持续时间。当线程读写申请成功后，开始对共享资源的读写操作，该操作持续相应时间后结束，并释放共享资源。

下面是一个测试数据文件的例子：

1 R 3 5  
  
2 W 4 5  
  
3 R 5 2  
  
4 R 6 5  
  
5 W 5.1 3

## **与实验相关的**API介绍

### 1. 线程创建相关

#### 线程创建

使用pthread\_create()函数。

int pthread\_create(pthread\_t \*thread, const pthread\_attr\_t\* attr, void \*(\*start\_routine) (void \*), void\* arg);

参数：

第一个参数：指向线程标示符pthread\_t的指针

第二个参数：设置线程的属性

第三个参数：线程运行函数的起始地址

第四个参数：运行函数的参数

#### 等待线程结束

等待函数pthread\_join()用来等待一个线程的结束。

int pthread\_join(pthread\_t thread, void \*\*retval);

参数 ：

thread: 线程标识符，即线程ID，标识唯一线程。

retval: 用户定义的指针，用来存储被等待线程的返回值 。

返回值 ： 0代表成功。 失败，返回的则是错误号。

#### 退出

pthread\_exit() 函数使线程退出并返回一个空指针类型的值，该值可以由pthread\_join()函数来获取。

#### sleep

函数原型：

unsigned int sleep (unsigned int seconds);

### 2. 多线程互斥锁相关

#### 初始化互斥锁

函数pthread\_mutex\_init()进行互斥锁的初始化。

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

pthread\_mutex\_init()函数是以动态方式创建互斥锁的，参数attr指定了新建互斥锁的属性。如果参数attr为NULL，则使用默认的互斥锁属性，默认属性为快速互斥锁。互斥锁的属性在创建锁的时候指定。

#### 销毁互斥锁

函数pthread\_mutex\_destroy()负责销毁互斥锁。

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

mutex 指向要销毁的互斥锁的指针。

互斥锁销毁函数在执行成功后返回 0，否则返回错误码。

#### 锁定

函数pthread\_mutex\_lock()负责锁定互斥锁。

当返回时，该互斥锁已被锁定。调用线程是该互斥锁的属主。如果该互斥锁已被另一个线程锁定和拥有，则调用线程将阻塞，直到该互斥锁变为可用为止。

#### 解锁

pthread\_mutex\_unlock() 与pthread\_mutex\_lock()成对存在，负责释放互斥锁。

### 3. 信号量相关

#### 创建

sem\_init()函数用于创建信号量，其原型如下：

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

该函数初始化由 sem 指向的信号对象，并给它一个初始的整数值 value。

pshared 控制信号量的类型，值为 0 代表该信号量用于多线程间的同步，值如果大于 0 表示可以共享，用于多个相关进程间的同步。

#### wait

sem\_wait() 是一个阻塞的函数，测试所指定信号量的值，它的操作是原子的。

函数原型：

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

若 sem value > 0，则该信号量值减去 1 并立即返回。

若sem value = 0，则阻塞直到 sem value > 0，此时立即减去 1，然后返回。

#### signal

sem\_post()把指定的信号量 sem 的值加 1，唤醒正在等待该信号量的任意线程。

函数原型：

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

#### 清理

sem\_destroy()函数用于对用完的信号量的清理。函数原型：

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

成功则返回 0，失败返回 -1。