**一、函数说明**

**1、pthread\_create**

pthread\_create是UNIX环境创建线程函数

头文件

　　#include<pthread.h>

函数声明

　　int pthread\_create(pthread\_t\*restrict tidp,const pthread\_attr\_t \*restrict\_attr,void\*（\*start\_rtn)(void\*),void \*restrict arg);

返回值

　　若成功则返回0，否则返回出错编号

　　返回成功时，由tidp指向的内存单元被设置为新创建线程的线程ID。attr参数用于制定各种不同的线程属性。新创建的线程从start\_rtn函数的地址开始运行，该函数只有一个万能指针参数arg，如果需要向start\_rtn函数传递的参数不止一个，那么需要把这些参数放到一个结构中，然后把这个结构的地址作为arg的参数传入。

　　linux下用C开发多线程程序，Linux系统下的多线程遵循POSIX线程接口，称为pthread。

　　由 restrict 修饰的指针是最初唯一对指针所指向的对象进行存取的方法，仅当第二个指针基于第一个时，才能对对象进行存取。对对象的存取都限定于基于由 restrict 修饰的指针表达式中。 由 restrict 修饰的指针主要用于函数形参，或指向由 malloc() 分配的内存空间。restrict 数据类型不改变程序的语义。 编译器能通过作出 restrict 修饰的指针是存取对象的唯一方法的假设，更好地优化某些类型的例程。

参数

　　第一个参数为指向线程标识符的指针。

　　第二个参数用来设置线程属性。

　　第三个参数是线程运行函数的起始地址。

　　最后一个参数是运行函数的参数。

　　另外，在编译时注意加上-lpthread参数，以调用静态链接库。因为pthread并非Linux系统的默认库

**2、pthread\_join()**

函数pthread\_join用来等待一个线程的结束,线程间同步的操作。

头文件 ： #include <pthread.h>

函数定义： int pthread\_join(pthread\_t thread, void \*\*retval);

描述 : pthread\_join()函数，以阻塞的方式等待thread指定的线程结束。当函数返回时，被等待线程的资源被收回。如果线程已经结束，那么该函数会立即返回。并且thread指定的线程必须是joinable的。

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。

**pthread\_join的应用：**pthread\_join使一个线程等待另一个线程结束。

代码中如果没有pthread\_join主线程会很快结束从而使整个进程结束，从而使创建的线程没有机会开始执行就结束了。加入pthread\_join后，主线程会一直等待直到等待的线程结束自己才结束，使创建的线程有机会执行。

所有线程都有一个线程号，也就是Thread ID。其类型为pthread\_t。通过调用pthread\_self()函数可以获得自身的线程号。

**3、sem\_init()**

信号量用sem\_init函数创建的，下面是它的说明：

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

功能：初始化信号量

返回值：创建成功返回0，失败返回-1

参数sem：指向信号量结构的一个指针

参数pshared：不为０时此信号量在进程间共享，为0时当前进程的所有线程共享

参数value：信号量的初始值

第一个参数:信号量名

看一个例子，比如有两个线程都要往打印机上打东西，但是同一时刻只能打一个。

第二个参数: 那么首先用sem\_init初始化一个信号量，注意pshared表示允许几个进程共享该信号量，一般设0用于进程内的多线程共享，要看是否支持进程共享，请查看下你的系统的man手册。

第三个参数value表示可用的资源的数目，即信号灯的数目，咱们这儿只有1个打印机所以设成1。

然后线程调用sem\_wait取获取这个信号灯，第一个线程一看，有1个，他就拿到了，然后可以继续后继操作，此时信号灯自动减1，变成0个。那么第二个线程调用sem\_wait时就会阻塞在这儿了。

第一个线程完成打印后，调用sem\_post释放信号灯，信号灯数目变成1，将会唤醒等待的第二个线程，然后第二个线程接着打印。

最后当所有任务完成后，主线程调用sem\_destroy释放这个信号量。

**4、sem\_post()，sem\_wait()，sem\_destroy()**

函数sem\_post( sem\_t \*sem )用来增加信号量的值。当有线程阻塞在这个信号量上时，调用这个函数会使其中的一个线程不在阻塞，选择机制同样是由线程的调度策略决定的。

函数sem\_wait( sem\_t \*sem )被用来阻塞当前线程直到信号量sem的值大于0，解除阻塞后将sem的值减一，表明公共资源经使用后减少。函数sem\_trywait ( sem\_t \*sem )是函数sem\_wait（）的非阻塞版本，它直接将信号量sem的值减一。

这两个函数控制着信号量的值，它们的定义如下所示：

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

功能：它的作用是从信号量的值减去一个“1”，但它永远会先等待该信号量为一个非零值（大于0）才开始做减法。（如果对一个值为0的信号量调用sem\_wait()，这个函数就会等待，直到有其它线程增加了信号量这个值使它不再是0为止，再进行减1操作。）

返回值：操作成功返回0，失败则返回-1且置errno

参数sem：指向信号量结构的一个指针

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

功能：它的作用来增加信号量的值。给信号量加1。

返回值：操作成功返回0，失败则返回-1且置errno

参数sem：指向信号量结构的一个指针

这两个函数都要用一个由sem\_init调用初始化的信号量对象的指针做参数。

最后一个信号量函数是sem\_destroy。这个函数的作用是在我们用完信号量对它进行清理。下面的定义：

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

功能：释放信号量自己占用的一切资源 （被注销的信号量sem要求：没有线程在等待该信号量了）

返回值：满足条件 成功返回0，否则返回-1且置errno为EBUSY

参数sem：指向信号量结构的一个指针

**5、pthread\_mutex\_init() pthread\_mutex\_destroy()**

linux下为了多线程同步，通常用到锁的概念。

posix下抽象了一个锁类型的结构：ptread\_mutex\_t。通过对该结构的操作，来判断资源是否可以访问。顾名思义，加锁(lock)后，别人就无法打开，只有当锁没有关闭(unlock)的时候才能访问资源。

即对象互斥锁的概念，来保证共享数据操作的完整性。每个对象都对应于一个可称为" 互斥锁" 的标记，这个标记用来保证在任一时刻，只能有一个线程访问该对象。

使用互斥锁（互斥）可以使线程按顺序执行。通常，互斥锁通过确保一次只有一个线程执行代码的临界段来同步多个线程。互斥锁还可以保护单线程代码。

要更改缺省的互斥锁属性，可以对属性对象进行声明和初始化。通常，互斥锁属性会设置在应用程序开头的某个位置，以便可以快速查找和轻松修改。

头文件：

#include <pthread.h>

函数原型：

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

pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

函数作用：

该函数用于C函数的多线程编程中，互斥锁的初始化。

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为空(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，在LinuxThreads实现中仅有一个锁类型属性，不同的锁类型在试图对一个已经被锁定的互斥锁加锁时表现不同。

pthread\_mutexattr\_init() 函数成功完成之后会返回零，其他任何返回值都表示出现了错误。

函数成功执行后，互斥锁被初始化为未锁住态。

互斥锁pthread\_mutex\_t的使用：

**互斥锁创建**

有两种方法创建互斥锁，静态方式和动态方式。POSIX定义了一个宏PTHREAD\_MUTEX\_INITIALIZER来静态初始化互斥锁，方法如下：

pthread\_mutex\_t mutex=PTHREAD\_MUTEX\_INITIALIZER;

在LinuxThreads实现中，pthread\_mutex\_t是一个结构，而PTHREAD\_MUTEX\_INITIALIZER则是一个结构常量。

动态方式是采用pthread\_mutex\_init()函数来初始化互斥锁，API定义如下：

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

其中mutexattr用于指定互斥锁属性（见下），如果为NULL则使用缺省属性。

**互斥锁销毁**

pthread\_mutex\_destroy()用于注销一个互斥锁，API定义如下：

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

销毁一个互斥锁即意味着释放它所占用的资源，且要求锁当前处于开放状态。由于在Linux中，互斥锁并不占用任何资源，因此LinuxThreads中的 pthread\_mutex\_destroy()除了检查锁状态以外（锁定状态则返回EBUSY）没有其他动作。

**二、网络资源参考**

1、【互斥锁与信号量】线程通信：互斥锁（mutex）与信号量（semaphore）\_mutex信号量-CSDN博客 https://blog.csdn.net/crr411422/article/details/131421923

2、Linux应用编程 | exec函数族-CSDN博客 https://blog.csdn.net/yychuyu/article/details/80173039