多线程编程

简介

线程（ thread ）是包含在进程内部的顺序执行流，是进程中的实际运作单位，也是操作系统能够进行调度的最小单位。一个进程中可以并发多条线程，每条线程并行执行不同的任务。

标准

POSIX Threads 就是这样一个规范的多线程标准接口。POSIX Threads（通常简称为 Pthreads ）定义了创建和操纵线程的一套 API 接口， 一般用于 Unix-like POSIX 系统中（如 FreeBSD、 GNU/Linux、 OpenBSD、 Mac OS 等系统）。

Pthreads **接口可以根据功能划分四个组：**

- 线程管理
- 互斥量
- 条件变量
- 同步

```
/**
 * @brief:获取自己的线程ID
 * @return:线程ID（pthread_t）
 */
pthread_t pthread_self(void);

/**
 * @brief:对比两个线程ID是否相等
 * @return:相等则返回非0值
 *         否则返回0
 */
int pthread_equal(pthread_t t1, pthread_t t2);

/**
 * @brief:创建线程
 * @Parma[1]:thread 用来指向新创建线程的ID
 * @Parma[2]:attr   线程各种属性的属性对象
 * @Parma[3]:start_routine 线程开始执行时，调用的函数的名字
 * @Parma[4]:arg    参数 start_routine 指定函数的参数
 * @return:创建不成功则返回非0的错误码
 *             EINVAL：attr参数无效
 *             EAGAIN：系统没有创建线程所需的资源
 *             EPERM ：调用程序没有适当的权限来设定调度策略或 attr 指定的参数
 *         创建成功则返回0
 */
int pthread_create(pthread_t *thread,
                   const pthread_attr_t *attr,
                   void *(*start_routine) (void *),
                   void *arg);
```

```
/**
 * @brief:对比两个线程ID是否相等
 * @Parma:retval 是一个 void 类型的指针，可以将线程的返回值当作 pthread_exit()的参数传入，这
 *        个值同样被 pthread_join()当作退出状态处理。
 */
void pthread_exit(void *retval);
```

线程的分类：

- 分离线程：退出时会释放它的资源的相乘
- 非分离线程：退出后不会立即释放资源，需要另一个线程为它调用pthread_join 函数或者进程退出时才会释放资源。

```
/**
 * @brief:将非分离线程设置为分离线程
 * @parma:线程ID
 * @retval:返回0表示成功
 *         失败返回错误码
 *             EINVAL：thread 参数所表示的线程不是可分离的线程
 *             ESRCH：没有找到 ID 为 thread 的线程
 */
int pthread_detach(pthread_t thread);

/**
 * @brief:对非分离线程进行连接，调用线程挂起
 * @parma[1]:thread 线程ID
 * @parma[2]:retval 为指向线程的返回值的指针提供一个位置， 这个返回值是目标线程
 *                  调用 pthread_exit()或者 return 后所返回的值，不需要可设为NULL
 * @retval:返回0表示成功
 *         失败返回错误码
 *             EINVAL：thread 参数所表示的线程不是可分离的线程
 *             ESRCH：没有找到 ID 为 thread 的线程
 */
int pthread_join(pthread_t thread, void **retval);
```

线程属性（pthread_create()**函数的第二个参数为**pthread_attr_t **类型， 用于设置线程的属性 ）**

线程的基本属性包括：栈大小、调度策略、线程状态

```
/**
 * @brief:将属性对象使用默认值进行初始化
 * @parma:指向属性对象的指针
 * @retval:返回0表示成功
 *         失败返回错误码
 */
int pthread_attr_init(pthread_attr_t *attr);
```

```
/**
 * @brief:销毁属性对象
 * @parma:指向属性对象的指针
 * @retval:返回0表示成功
 *         失败返回错误码
 */
int pthread_attr_destroy(pthread_attr_t *attr);
```