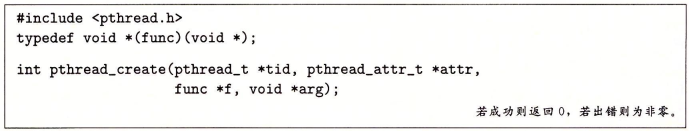


创建时预先派生指定数量的线程，然后去任务队列取添加进来的任务进行处理。

1、POSIX线程相关函数

**创建线程**



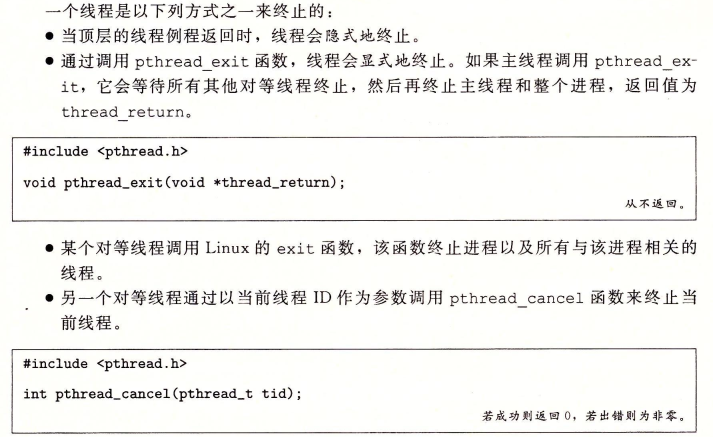
tid返回线程ID

attr修改线程属性

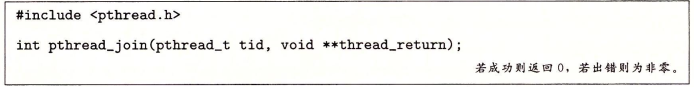
f在新线程中运行的函数

arg是输入变量

**终止线程**

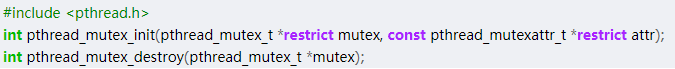


**回收线程资源**

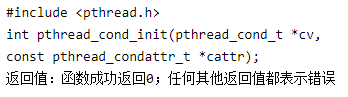


pthread\_join会阻塞，直到tid终止，将线程返回的通用（void \*）指针赋值为pthread\_return指向的位置，然后回收已终止线程占用的所有内存资源。

**互斥锁初始化**



**条件变量初始化**



以上两个初始化函数在第二个参数为NULL的时候使用默认属性。成功返回0.

2、数据结构

enum threadpool\_error\_t 枚举线程池错误

enum threadpool\_destroy\_flags\_t

enum threadpool\_shutdown\_t 枚举线程池关闭类型

struct threadpool\_task\_t 线程池任务结构

struct threadpool\_t 线程池

struct threadpool\_t {  
 pthread\_mutex\_t lock; //互斥锁  
 pthread\_cond\_t notify; //条件变量  
 pthread\_t \*threads; //线程池数组首指针  
 threadpool\_task\_t \*queue; //任务队列数组首指针  
 int thread\_count; //线程数量  
 int queue\_size; //任务数量  
 int head; //当前任务队列的头部  
 int tail; //当前任务队列的尾部  
 int count; //待运行的任务  
 int shutdown; //线程池状态是否关闭  
 int started; //正在运行的线程数  
};

3、对外函数

threadpool\_t \*threadpool\_create( //创建线程池对象

int thread\_count, //线程池长度

int queue\_size, //任务队列长度

int flags); //是否是新创建的线程池

1、分配线程池内存空间

2、初始化threadpool\_t的成员变量，包括为线程数组和任务队列数组分配空间并初始化

3、调用pthread\_create创建thread\_count个线程，并在pool->thread[i]中存储tid，线程中执行threadpool\_thread函数。

int threadpool\_add( //向线程池任务队列中添加一个新任务

threadpool\_t \*pool, //线程池指针

void (\*routine)(void \*), //新任务函数指针  
void \*arg, //新任务参数

int flags); //无用

1、计算新任务在任务队列中的存放位置

2、如果任务队列是满的或者线程池已经关闭，则不能添加新任务

3、添加新任务，并且更新任务队列的尾，以及待执行任务个数

int threadpool\_destroy( //销毁线程池

threadpool\_t \*pool, //线程池指针

int flags); //shutdown flags

1、如果线程池为空则返回错误

2、获取条件变量互斥锁，因为关闭的时候有些线程可能因为任务队列为空而阻塞在条件变量上，此时应用pthread\_cond\_broadcast唤醒所有的阻塞线程。

3、设置线程池的关闭方式，即shutdown属性

4、唤醒所有阻塞在条件变量的线程

5、用pthread\_join等待所有线程终结，并回收资源

4、辅助函数

static void \*threadpool\_thread(void \*threadpool);

//线程池中每个线程所执行的函数

1、每个线程需要不断的从任务队列中取任务执行，所以函数整体是一个死循环

2、获取条件变量互斥锁，如果任务队列中待执行任务为0，则阻塞在条件变量上。

3、判断shutdown属性，如果任务队列中任务为空并且shutdown通知关闭，退出死循环，并且用pthread\_exit退出线程

4、从任务队列的头部取一个任务执行，然后更新头指针和待执行任务数量。

5、释放条件变量互斥锁，并执行任务

int threadpool\_free(threadpool\_t \*pool);

//释放线程池内存

1、如果线程池不存在或者仍有运行的线程，则不能退出

2、调用free释放线程池数组和任务队列数组的空间

3、销毁互斥锁和条件变量

## 流程分析

1、线程池的创建与执行

threadpool\_create创建线程池时，首先分配线程池数组和任务队列数组的空间，然后调用pthread\_create创建线程，此时任务队列为空。

这样新创建的线程由于任务队列为空，所以都阻塞了。必须等待任务队列中有任务才能取执行。这部分实现使用条件变量。

当用threadpool\_add添加很多任务后，每个线程在取新任务时直接从任务队列头部取出并执行。由于需要不断的执行任务，所以threadpool\_thread采用一个死循环。

2、线程池的销毁

首先考虑到一个情况，就是在销毁的时候有可能任务队列已经为空，有若干线程阻塞在了条件变量上，首先将这些线程唤醒。

设置线程池的shutdown属性为真，这样每个线程在死循环中检测到shutdown为真后就退出死循环，然后用exit退出线程。

然后pthread\_join等待所有线程退出，并且释放资源

然后调用threadpool\_free释放其他资源