线程池的设计与实现

采用半同步半反应堆模式进行设计，主线程只负责项任务链表当中添加任务，相当于监听工作，工作线程只负责，处理任务。当工作线程空闲的时候，采用条件变量来使得工作线程出于“暂停”状态，当有任务到来的时候，主线程负责唤醒工作线程来处理业务。

1. 基础知识准备

条件变量：条件变量是线程可用的另一种同步机制，条件变量给多个线程提供了一个回合的场所。条件变量与互斥量一起使用，允许线程以无竞争的方式等待特定的条件发生，条件本身是油互斥量保护的。线程在改变条件前必须首先锁住互斥量，其他线程在获取互斥量之前不会察觉到这种变化，因为必须锁定互斥量以后才能计算条件。

条件变量使用之前必须首先进行初始化，pthread_cond_t 数据类型的条件变量可以使用两种方式进行初始化，可以把常量 PTHREAD_COND_INITIALIZER 赋给静态分配的条件变量，但是若果条件变量是动态分配的，可以使用 pthread_cond_init 函数进行初始化。

在释放底层内存空间之前，可以使用 pthread_mutex_destory 函数对条件变量进行去除初始化(deinitialize)

```
#include <pthread.h>
int pthread_cond_init(pthread_cond_t *restrict cond,\
                      Pthread_condattr_t *restrict attr);
Int pthread_cond_destory(pthread_cond_t *cond);
                                *两者的返回值都是：若成功则返回 0，否则返回错误编号.
```

除非需要创建一个非默认属性的条件变量，否则 pthread_cond_init 函数的 attr 参数可以设置为 NULL

使用 pthread_cond_wait 等待条件变为真，如果在给定的时间内条件不能满足，那么会生成一个出错码的返回变量。

```
int pthread_cond_wait(pthread_cond_t *restrict cond,\
                      pthread_mutex_t *restrict mutex);
int pthread_cond_timedwait(pthread_cond_t *restrict cond,\
                           pthread_mutex_t *restrict mutex,\
                      const struct timespec *restrict timeout);
成功返回 0，失败返回错误编号
```

传递给 phtread_cond_wait 的互斥量对条件进行保护，调用者把锁着的互斥量传递给函数。函数把调用线程放到等待条件的线程队列上，然后对互斥量解锁，这两个操作是原子操作。这样就关闭了条件检查和线程进入休眠状态等待条件改变这两个操作之间的时间通道，这样线程就不会错过条件的任何变化。pthread_cond_wait 返回时，互斥量再次被锁住。

Pthread_cond_timedwait 函数的工作方式与 pthread_cond_wait 函数相似只是多了一个 timeout。Timeout 值指定了等待的时间，它是通过 timespec 结构指定。

```
Struct timespec{
        time_t tv_sec;
        long   tv_nsec;  //纳秒
```

使用这个结构体时，需要指定愿意等待多长时间，时间值是一个绝对数而不是相对值。通过 clock_gettime(CLOCK_REALTIME,&abstime)获得当前时间。
然后在加上需要等待的时间。

```
          clock_gettime(CLOCK_REALTIME, &abstime);
           abstime.tv_sec+=2;
```

如果时间值到了但是条件还是没有出现，pthread_cond_timedwait 将重新获取互斥量然后返回错误 ETIMEDOUT。从 pthread_cond_wait 或者 pthread_cond_timedwait 调用成功是返回时，线程要重新计算条件，因为其他线程可能已经运行并改变了条件。

有两个函数可以用于通知线程条件已经满足。pthread_cond_signal 函数将唤醒等待条件的某个线程，而 pthread_cond_broadcast 函数将唤醒等待该条件的所有线程

```
    int pthread_cond_signal(pthread_cond_t *cond);
    int pthread_cond_broadcast(pthread_cond_t *cond);
```

2. 结构体设计

1. 线程池结构体

用于描述一个线程池，记录线程池的相关的状态，由主线程进行维护。

```
    typedef struct threadpool
    {
       condition_t ready;//条件变量,任务准备就绪或者线程池销毁通知
       task_t *first;      //
       task_t *end;
       int counter; //线程池中当前的线程数
       int idle;    //线程池中当前正在等待任务的线程数
       int max_threads;//线程池中允许最大的线程数
       int quit;       //销毁线程池的时候置 1
}threadpool_t;
```

2. 条件变量结构体封装由于条件变量要和一个互斥量进行同步使用，所以对两者进行了封装。

```
typedef struct condition
{
   pthread_mutex_t pmutex;
   pthread_cond_t pcond;
}condition_t;
```

3. 任务结构体

```
//任务结构体，将任务放在任务队列由线程池中的线程来执行
typedef struct task
{
    void *(*run)(void *arg);//任务回调函数
    void *arg;   //回调函数参数
    struct task *next;
}task_t;
```

3. 线程池逻辑图