通用线程库的设计与实现①

洪承煜, 许自龙, 亢永敢, 杨尚琴

(中国石化石油物探技术研究院, 南京 211103)

摘 要: 随着采集、处理、解释一体化地震勘探软件的发展, 构建跨硬件、操作系统的云计算平台成为必要; 而云计算平台中, 通用线程库成为开发有大计算量算法的地震勘探软件的关键. 通过对线程模型和同步机制的分析, 分类整理出线程的属性、线程的控制、线程执行流的构建和同步的属性及操作; 最后在这些通用的组件基础上, 通过对各种本地线程库的封装, 实现基于任意本地线程库的跨平台通用线程库.

关键词: 通用线程库; 跨平台; 多核; NTPL; Windows Thread

Design and Implementation of the Common Thread Library

HONG Cheng-Yu, XU Zi-Long, KANG Yong-Gan, YANG Shang-Qin

(SINOPEC Geophysical Research Institute, Nanjing 211103, China)

Abstract: With the development of the integration of seismic exploration software in the acquisition, processing, explaining, builds cloud computing platform of a cross-hardware, operating system that becomes necessary; and in cloud computing platforms, the common thread library becomes the key to the development of a large amount of computation algorithm of seismic software. In the paper, the analysis of the threading model and synchronization mechanisms, sorting out the property of the thread, thread control, thread flow and synchronization properties and operation; finally, on the basis of these common components, to package a variety of local thread library, so that implement cross-platform common thread library on basis of any native threads library.

Key words: the common thread library; cross-platform; multi-core; NTPL; windows thread

在地震勘探软件中, 高性能计算上的需求越来越大[1], 特别是在地震采集、处理、解释一体化的软件中, 建立一个通用的多核线程库, 既可以让系统开发人员提供更高效、更前沿的本地线程库服务, 又可以让物探方法集成人员更加简易地使用到处理器的性能.

在各种现代操作系统中, 线程是程序执行流的最小单元[2], 合理使用好线程, 可以最大程度地发挥多核处理器的性能. 但是线程的掌握和使用有很多操作系统级的概念, 这给线程使用者带来了复杂度; 而且不同操作系统的本地线程库及同一操作系统的不同本地线程库都具有自己的一些 API, 这些 API 的接口特性总有一些差异, 这也给多线程程序的跨平台移植, 以及在云计算平台中的应用带来了困难. 根据上面的分析可得, 建立本地线程库封装层, 抽象出线程及同步的基本操作是构建通用线程库的首要条件. 在构建了抽象的线程及同步的基本操作后, 这只是帮助线程使用者解决了跨平台问题, 还有很多的线程管理[3]和其它高级的同步原语需要在这些基本操作的基础上进行合理的组合应用, 才能真正意义上实现通用线程库.

简之, 本文在封装了 NTPL 和 Windows Thread 跨平台接口的基础上, 实现了线程管理和同步操作的通用线程库. 在其它硬件和操作系统平台上, 可以通过简单的封装本地线程库实现移植.

1 总体框架设计

通用线程库框架主要围绕以下两个要点进行设计.

① 收稿时间:2012-02-13;收到修改稿时间:2012-03-22

1) 对各种本地线程库的封装. 这一层的设计目标: 线程、同步原语的各种操作的抽象接口定义, 达到隔离各种本地线程库的依赖. 设计原则: 不损失操作的性能;

表 1　线程主要功能的接口定义

功能		接口定义
线程创建		INT CreateThread (ThreadHandle , ExecFunc , ThreadAttr)
线程终止		INT ExitThread (ThreadExitCode , ExecFunc) INT TerminateThread(ThreadHandle , ThreadExitCode , ClearFunc)
等待线程终止		INT WaitThreadEnd(ThreadHandle , ThreadExitCode*)
线程标识获取/比较		ThreadHandle GetThreadHandle() BOOL CompareTH (ThreadHandle , ThreadHandle)
线程属性		INT InitThreadAttr(ThreadAttr) INT DestoryThreadAttr(ThreadAttr) INT SetThreadAttr(ThreadAttr) INT GetThreadAttr(ThreadAttr)
线程同步	互斥锁	INT CreateMutex(Mutex , SyncAttr)　INT DestoryMutex(Mutex) INT LockMutex (Mutex)　INT TryLockMutex(Mutex) INT LockTimeOutMutex(Mutex)　INT UnLockMutex(Mutex)
	条件变量	INT CreateCond(Cond , SyncAttr)　INT DestoryCond(Cond) INT Wait (Cond)　INT WaitTiemOut(Cond) INT Signal(Cond)　INT Broadcast(Cond)
同步属性		INT InitThreadAttr(SyncAttr) INT DestoryThreadAttr(SyncAttr) INT SetSyncAttr(SyncAttr)　INT GetSyncAttr(SyncAttr)

表 3 线程句柄类型

```
typedef struct {
void *p; //指向真正的线程内核对象
//额外的信息，如被重用次数等
unsigned int unused;
} ThreadHandle;
```

表 4 线程属性类型

```
typedef struct {
void * p; //内核线程属性对象
int attrType; //线程属性的类型
void *attrValue; //线程属性值
} ThreadAttr;
```

表 5 同步属性类

```
typedef struct {
void * p; //内核同步属性对象
int attrType; //同步属性的类型
void *attrValue; //同步属性值
} SyncAttr;
```

2) 提供线程使用者统一的调用接口. 这一层首先提供线程的基本功能, 包括通过线程标识对线程状态的监测、线程创建. 其次是线程控制[4], 它包括: 开始运行线程、等待线程结束、线程进入睡眠、唤醒线程、获取和设置线程执行的优先级、设置和获取线程类型、线程退出资源释放和清理、线程终止, 而线程终止又包括自己终止、线程运行终止和被其它线程终止. 另一方面是同步原语, 这包括互斥量、条件变量、信号量、临界区的操作.

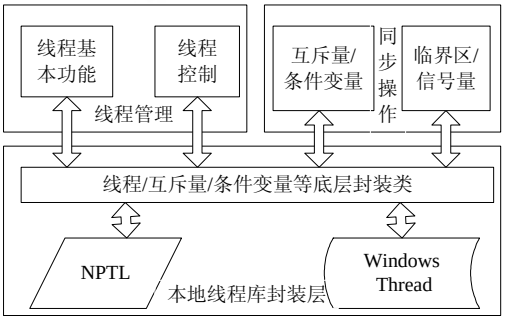

图 1　通用线程库框架

根据以上两个主要的设计要点, 通用线程框架可设计为本地线程库封装层、线程管理和同步操作三部分. 前一者实现各种本地线程库的隔离, 统一了线程、同步原语的操作接口; 后两者实现了线程和同步原语的抽象, 既为线程使用者封装了复杂的线程控制, 也为其使用者提供了统一的接口.

2　本地线程库封层装设计

按照 POSIX 线程库接口标准, 此处把线程分为以下七个主要功能部分, 并定义了其接口, 如表 1. 下面简单的给出各种参数的数据类型说明.

1)INT: 有符号 32 位整型; 2)ExecFunc: 线程流的执行函数, 类型为 void typedef *(*ExecFunc)(void *); (3)BOOL: TRUE 和 FALSE 两种值, 一般 TRUE 为成功, FALSE 为失败; 4)ClearFunc: 线程退出调用的清理函数, 类型为 void typedef *(* ClearFunc)(void *); 5)ThreadHandle: 线程句柄. 通过分析 NTPL、WinThread 等本地线程可得, 它们都有自己的内核对

象, 这是它们最重要的标识, 但是为以后更加复杂的多核线程使用, 所以在下面的结构体中使用了一个无符号整型作为扩展字段. 类型定义如表 3 所示; 6)ThreadAttr: 线程属性类型. 分析各种本地线程库可得, 线程属性的类型(attrType)主要有以下七种: ①线程类型属性, 它包括分离式和正常启动式两种属性值; ②线程栈最低地址和大小属性; ③线程栈大小属性④线程栈末尾的警戒缓冲区大小属性; ⑤线程可否取消属性, 这个属性的目的是控制线程可否被别的线程取消; 如果设置成不能取消 A 线程, 则 A 线程只能自己结束或者在进程退出时被进程结束; ⑥线程取消类型属性, 此属性值主要有延迟取消和立即(异步)取消, 延迟取消是默认的, 它会在下个取消点后取消线程; ⑦线程并发度属性, 它控制线程的并发度. 定义如表 4 所示; 7)SyncAttr: 同步属性类型. 通过各种本地线程库同步属性的总结, 大体有以下三种属性类型(attrType): ①互斥锁属性, 它包括不会自检死锁的属性值; 会自检死锁的属性值; 重复上锁, 不会引起死

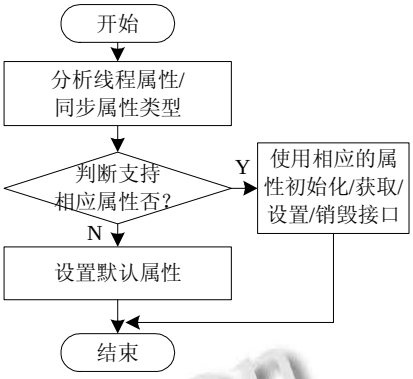

图 3 线程、同步属性操作流程图

锁的属性值, 但必须使用同样的解锁次数才会解锁互斥锁; ②互斥锁共享属性, 这样的互斥锁可以在进程之间共享使用; ③条件变量共享属性, 这样的条件变量可以在进程之间共享使用. 定义如表 4 所示. 8)ThreadExitCode: 线程退出码 typedef void * ThreadExitCode, 它代表线程退出码. 9)Mutex 和 Cond: 互斥锁和条件变量 typedef void * Mutex 和 typedef void * Cond, 它代表互斥锁和条件变量的内核对象[5].

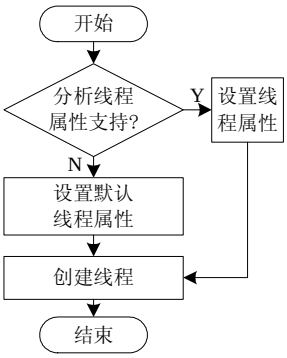

图 2 线程创建流程图

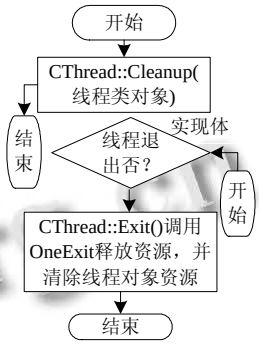

图 4 线程退出流程图

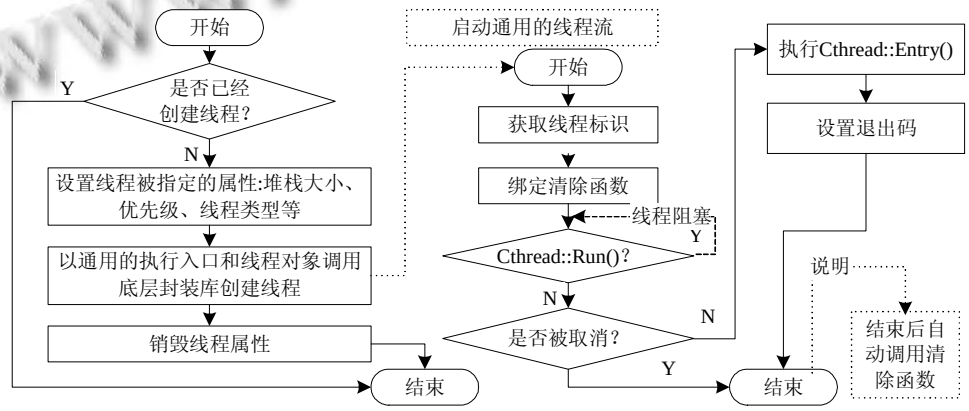

图 5 线程创建及线程流构建

有上面对接口数据类型的说明，下面简要的说明表 1 的接口设计及实现.

1)线程创建接口主要完成线程属性设置，并创建线程；它的实现流程如图 2. 2)线程终止有两种，一种是线程自己终止，要么是调用 ExitThread 接口，要么是 ExecFunc 函数返回；另一种是别的线程终止其它线程，这两中终止方式的实现流程为：简单的调用本地线程库中的自我终止接口即可. 从表 1 接口中可知，在调用终止接口时都可以指定线程退出码，这个退出码可以是线程退出后，在调用“等待线程终止”的线程中，依然有意义的代码，比如，一个全局的共享变量等[6]. 3)设计等待线程终止接口的目的是回收结束线程的系统资源，比如线程堆栈等. 调用等待线程终止接口，会阻塞当前线程，直到等待的指定线程被自己或其它线程终止. 但是如果设置了分离式线程属性后，则此接口会立即返回消息码，通知调用者线程可以自动释放资源. 所以当我们设置了分离式线程属性后，可以不用在主线程中调用此接口来达到系统资源回收的目的. 4)线程标识获取和比较接口，它用于线程的身份的确定. 它们的实现只需简单的调用本地线程库接口. 5)线程属性和同步属性接口，完成线程和同步属性的初始化、设置、获取、销毁，它们的实现流程如图 3. 6)同步接口分为互斥锁和条件变量，它们都有创建和销毁接口，互斥锁有上锁、解锁接口，而条件变量有等待接口，它们的实现都是调用相应的本地线程库接口实现.

通过这儿的本地线程库封装设计与实现，隔离了各种本地线程库的差异，为下一步的线程管理和同步原语的构建提供了基础.

3　线程管理和同步原语设计

这儿的线程管理是狭义上的本地线程库抽象功能的封装，它的目的是提供简单的线程使用机制[7]，让使用者脱离复杂的线程概念，就可以很熟练的使用线程来处理应用问题.

如表 2 所示，线程管理被主要分为七个功能大类，以下依次分析:

1) 类静态成员函数：它们是，CPU 个数获取，阻塞当前线程睡眠若干时间，把当前线程 CPU 时间片给其它线程，获取、设置线程并发度，获取当前线程标识符，获取主线程标识符等. 这些功能的实现只需简单地调用本地线程库的封装层接口.

表 2　线程管理类

class CThread { CThread(CThreadKind kind); virtual ~CThread();	// 2)

CThreadError Create(UINT stackSize);
virtual void *Entry() = 0;
CThreadError Run();
ThreadExitCode Wait();
void Exit(ThreadExitCode exitcode);
virtual void OnExit() { }
	// 3)

CThreadError Pause(); CThreadError Resume();	// 4)
CThreadError Delete(ThreadExitCode *rc); virtual bool TestDestroy(); CThreadError Kill();　　};	// 6)

2) 类的构造函数，首先指定将要创建的线程的类型(分离式或正常类型)，接着初始化线程的状态、线程句柄、退出码等；类的析构函数判断线程流是否还在运行，如果还在执行，会通知此线程执行流会崩溃.

3) 与线程开始和结束有关的 6 个接口，它们依次是：线程创建接口，我们需要实现的线程流执行接口，启动线程流执行的接口，等待线程流执行完成接口，线程退出时调用接口，线程在退出时自动调用的退出清理接口. 它们的主要实现如图 4、5 所示.

4) 线程的暂停和继续运行，通过互斥锁实现，它们之间必须在两个不同的线程中配合使用.

5) 状态包括线程运行的状态、退出状态等.

6) 这是线程终止功能，其中 Kill()是强制终止线程，通过设置立即取消线程功能实现. Delete()和 TestDestory()是两个通过互相配合达到结束线程目的的接口；控制线程调用 Delete()通知任务线程结束，并阻塞当前线程，直到任务线程结束，最后收回任务线程系统资源；任务线程调用 TestDestory()测试是否要求被退出，达到结束自己的执行流. 它们通过线程对象中的线程状态的设置来实现.

7)其它功能，它们有线程优先级的获取与设置，线程属性获取与设置等针对单个线程实例的接口，这些接口可根据需要，通过调用封装层接口即可实现.

在底层互斥锁和条件变量封装类的基础上，互斥锁和条件变量类可以简单的封装实现，而临近区可以使用互斥锁实现，信号量则可有互斥锁和条件变量类

（下转第 85 页）