多线程

锁定

本词条由“科普中国”百科科学词条编写与应用工作项目 审核 。

多线程（英语：multithreading），是指从软件或者硬件上实现多个线程并发执行的技术。具有多线程能力的计算机因有硬件支持而能够在同一时间执行多于一个线程，进而提升整体处理性能。具有这种能力的系统包括对称多处理机、多核心处理器以及芯片级多处理（Chip-level multithreading）或同时多线程（Simultaneous multithreading）处理器。[1] 在一个程序中，这些独立运行的程序片段叫作“线程”（Thread），利用它编程的概念就叫作“多线程处理（Multithreading）”。具有多线程能力的计算机因有硬件支持而能够在同一时间执行多于一个线程（台湾译作“执行绪”），进而提升整体处理性能。

中文名	多线程	用　途	实现多个线程并发执行的技术
外文名	multithreading	对　象	计算机

目录

1 定义
2 用途
3 硬件支持
4 线程
5 优点
6 缺点
7 .NET Framework
8 Java

定义

在计算机编程中，一个基本的概念就是同时对多个任务加以控制。许多程序设计问题都要求程序能够停下手头的工作，改为处理其他一些问题，再返回主进程。可以通过多种途径达到这个目的。最开始的时候，那些掌握机器低级语言的程序员编写一些“中断服务例程”，主进程的暂停是通过硬件级的中断实现的。尽管这是一种有用的方法，但编出的程序很难移植，由此造成了另一类的代价高昂问题。中断对那些实时性很强的任务来说是很有必要的。但对于其他许多问题，只要求将问题划分进入独立运行的程序片断中，使整个程序能更迅速地响应用户的请求。

多线程

最开始，线程只是用于分配单个处理器的处理时间的一种工具。但假如操作系统本身支持多个处理器，那么每个线程都可分配给一个不同的处理器，真正进入“并行运算”状态。从程序设计语言的角度看，多线程操作最有价值的特性之一就是程序员不必关心到底使用了多少个处理器。程序在逻辑意义上被分割为数个线程；假如机器本身安装了多个处理器，那么程序会运行得更快，毋需作出任何特殊的调校。根据前面的论述，大家可能感觉线程处理非常简单。但必须注意一个问题：共享资源！如果有多个线程同时运行，而且它们试图访问相同的资源，就会遇到一个问题。举个例子来说，两个线程不能将信息同时发送给一台打印机。为解决这个问题，对那些可共享的资源来说（比如打印机），它们在使用期间必须进入锁定状态。所以一个线程可将资源锁定，在完成了它的任务后，再解开（释放）这个锁，使其他线程可以接着使用同样的资源。

多线程是为了同步完成多项任务，不是为了提高运行效率，而是为了提高资源使用效率来提高系统的效率。线程是在同一时间需要完成多项任务的时候实现的。

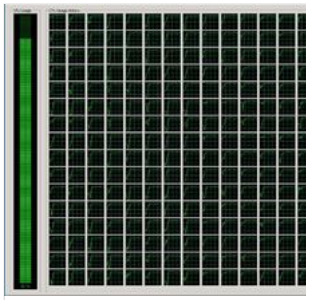
最简单的比喻多线程就像火车的每一节车厢，而进程则是火车。车厢离开火车是无法跑动的，同理火车也不可能只有一节车厢。多线程的出现就是为了提高效率。同时它的出现也带来了一些问题。

用途

在大多数研究领域内是要求线程调度程序要能够快速选择其中一个已就绪线程去运行，而不是一个一个运行而降低效率。所以要让调度程序去分辨线程的优先级是很重要的。而线程调度程序可能是以硬件、软件，或是软硬件并存的形式存在。

而另一个研究领域则是要研究何种事件（高速缓存失败、内部运行连续性、使用DMA等）会造成线程切换。

如果多线程的方案会复制所有软件可见的状态，包括特许的控制登录、TLB 等，那就能够让虚拟机去创造各式线程。这样子就允许在相同的处理器中每个线程跑各自的操作系统。换句话说，如果只有存储了用户模式的状态，就能够让相同的裸晶大小的芯片在一段时间内处理更多的线程。



多线程图册

硬件支持

多线程硬件支持的目标，即支持快速进行就绪态线程、执行态线程间的切换。为达成这个目标，需要硬件实现保存、恢复程序看得见的寄存器以及一些对程序执行有影响的控制寄存器（如程序计数器PC、程序状态寄存器SR）。从一个线程切换到另一个线程对硬件来讲意味着保存当前线程的一组寄存器的值，并恢复即将执行线程的一组寄存器的值。

新增这些功能的硬件有以下优势：

- 线程切换能够在一个 CPU 周期内完成(有些硬件甚至没有开销，上个周期在运行线程A，下个周期就已在运行线程B)。
- 每个线程看起来就像是独自运行的，即没有与其他线程共享硬件资源。对操作系统来说，通常每个线程都被视做独占一个处理器，这样将简化系统软件的设计（尤其是对于支持多线程的操作系统）。

为了在各个线程间有效率的进行切换，每个线程需要保存自己的一组寄存器集（register set）。有些硬件设计成每个处理器核心具有两组寄存器文件，以实现在多个线程间快速切换。

线程

定义

英文：Thread

每个正在系统上运行的程序都是一个进程。每个进程包含一到多个线程。进程也可能是整个程序或者是部分程序的动态执行。线程是一组指令的集合，或者是程序的特殊段，它可以在程序里独立执行。也可以把它理解为代码运行的上下文。所以线程基本上是轻量级的进程，它负责在单个程序里执行多任务。通常由操作系统负责多个线程的调度和执行。

线程是程序中一个单一的顺序控制流程.在单个程序中同时运行多个线程完成不同的工作,称为多线程.

线程和进程的区别在于,子进程和父进程有不同的代码和数据空间,而多个线程则共享数据空间,每个线程有自己的执行堆栈和程序计数器为其执行上下文.多线程主要是为了节约CPU时间,发挥利用,根据具体情况而定. 线程的运行中需要使用计算机的内存资源和CPU。

优点

·使用线程可以把占据时间长的程序中的任务放到后台去处理

·用户界面可以更加吸引人，这样比如用户点击了一个按钮去触发某些事件的处理，可以弹出一个进度条来显示处理的进度

·程序的运行速度可能加快

·在一些等待的任务实现上如用户输入、文件读写和网络收发数据等，线程就比较有用了。在这种情况下可以释放一些珍贵的资源如内存占用等等。

.多线程技术在IOS软件开发中也有举足轻重的位置。

.线程应用的好处还有很多，就不一一说明了

缺点

·如果有大量的线程,会影响性能,因为操作系统需要在它们之间切换。

·更多的线程需要更多的内存空间。

·线程可能会给程序带来更多“bug”，因此要小心使用。

·线程的中止需要考虑其对程序运行的影响。

·通常块模型数据是在多个线程间共享的，需要防止线程死锁情况的发生。

[2]

一些线程模型的背景

可以重点讨论一下在Win32环境中常用的一些模型。

·单线程模型

在这种线程模型中，一个进程中只能有一个线程，剩下的进程必须等待当前的线程执行完。这种模型的缺点在于系统完成一个很小的任务都必须占用很长的时间。

·块线程模型（单线程多块模型STA）

这种模型里，一个程序里可能会包含多个执行的线程。在这里，每个线程被分为进程里一个单独的块。每个进程可以含有多个块，可以共享多个块中的数据。程序规定了每个块中线程的执行时间。所有的请求通过Windows消息队列进行串行化，这样保证了每个时刻只能访问一个块，因而只有一个单独的进程可以在某一个时刻得到执行。这种模型比单线程模型的好处在于，可以

响应同一时刻的多个用户请求的任务而不只是单个用户请求。但它的性能还不是很好，因为它使用了串行化的线程模型，任务是一个接一个得到执行的。

·多线程块模型（自由线程块模型）

多线程块模型（MTA）在每个进程里只有一个块而不是多个块。这单个块控制着多个线程而不是单个线程。这里不需要消息队列，因为所有的线程都是相同的块的一个部分，并且可以共享。这样的程序比单线程模型和STA的执行速度都要快，因为降低了系统的负载，因而可以优化来减少系统idle的时间。这些应用程序一般比较复杂，因为程序员必须提供线程同步以保证线程不会并发的请求相同的资源，因而导致竞争情况的发生。这里有必要提供一个锁机制。但是这样也许会导致系统死锁的发生。

进程和线程都是操作系统的概念。进程是应用程序的执行实例，每个进程是由私有的虚拟地址空间、代码、数据和其它各种系统资源组成，进程在运行过程中创建的资源随着进程的终止而被销毁，所使用的系统资源在进程终止时被释放或关闭。

线程是进程内部的一个执行单元。系统创建好进程后，实际上就启动执行了该进程的主执行线程，主执行线程以函数地址形式，比如说main或WinMain函数，将程序的启动点提供给Windows系统。主执行线程终止了，进程也就随之终止。

每一个进程至少有一个主执行线程，它无需由用户去主动创建，是由系统自动创建的。用户根据需要在应用程序中创建其它线程，多个线程并发地运行于同一个进程中。一个进程中的所有线程都在该进程的虚拟地址空间中，共同使用这些虚拟地址空间、全局变量和系统资源，所以线程间的通讯非常方便，多线程技术的应用也较为广泛。多线程可以实现并行处理，避免了某项任务长时间占用CPU时间。要说明的一点是，到2015年为止，大多数的计算机都是单处理器（CPU）的，为了运行所有这些线程，操作系统为每个独立线程安排一些CPU时间，操作系统以轮换方式向线程提供时间片，这就给人一种假象，好象这些线程都在同时运行。由此可见，如果两个非常活跃的线程为了抢夺对CPU的控制权，在线程切换时会消耗很多的CPU资源，反而会降低系统的性能。这一点在多线程编程时应该注意。C++ 11 标准中，STL类库也实现了多线程的类std::thread，使得多线程编程更加方便。

NET Framework

在本质上和结构来说，.NET是一个多线程的环境。有两种主要的多线程方法是.NET所提倡的：使用ThreadStart来开始你自己的进程，直接的（使用ThreadPool.QueueUserWorkItem）或者间接的（比如Stream.BeginRead，或者调用BeginInvoke）使用ThreadPool类。一般来说，你可以"手动"为长时间运行的任务创建一个新的线程，另外对于短时间运行的任务尤其是经常需要开始的那些，进程池是一个非常好的选择。进程池可以同时运行多个任务，还可以使用框架类。对于资源紧缺需要进行同步的情况来说，它可以限制某一时刻只允许一个线程访问资源。这种情况可以视为给线程实现了锁机制。线程的基类是System.Threading。所有线程通过CLI来进行管理。

·创建线程：

创建一个新的Thread对象的实例。Thread的构造函数接受一个参数：

Thread DummyThread = new Thread(new ThreadStart(dummyFunction));

·执行线程：

使用Threading命名空间里的start方法来运行线程：

DummyThread.Start ();

·组合线程：

经常会出现需要组合多个线程的情况，就是当某个线程需要其他线程的结束来完成自己的任务。假设DummyThread必须等待DummyPriorityThread来完成自己的任务，只需要这样做：

DummyPriorityThread.Join() ;

·暂停线程：

使得线程暂停给定的秒

DummyPriorityThread.Sleep(<Time in Second>);

·中止线程：

如果需要中止线程可以使用如下的代码：

DummyPriorityThread.Abort();

·同步

经常会遇到需要在线程间进行同步的情况，下面的代码给出了一些方法：

```
usingSystem;
usingSystem.Threading;

namespaceSynchronizationThreadsExample
{
classSynchronizationThreadsExample
```

```
{
privateintcounter=0;
staticvoidMain()
{
SynchronizationThreadsExampleSTE=newSynchronizationThreadsExample();
STE.ThreadFunction();
}

publicvoidThreadFunction()
{
ThreadDummyThread=newThread(newThreadStart(SomeFunction));
DummyThread.IsBackground=true;
DummyThread.Start();
Console.WriteLine("Startedthread");
ThreadDummyPriorityThread=newThread(newThreadStart(SomeFunction));
DummyPriorityThread.IsBackground=true;
//="SecondThread";
DummyPriorityThread.Start();
Console.WriteLine("Startedthread");
DummyThread.Join();
DummyPriorityThread.Join();
}

publicvoidSomeFunction()
{
try
{
while(counter<10)
{
inttempCounter=counter;
tempCounter++;
Thread.Sleep(1);
counter=tempCounter;
Console.WriteLine("Thread.SomeFunction:"+Thread.+counter);
}
}
catch(ThreadInterruptedExceptionEx)
{
Console.WriteLine("Exceptioninthread"+Thread.);
}
finally
{
Console.WriteLine("ThreadExiting."+Thread);
}
}

}
}
```

·使用Interlock

C#提供了一个特殊的类叫做interlocked，就是提供了锁机制的实现，可以加入如下的代码实现锁机制：

Interlocked.SomeFunction (ref counter);

·使用锁

这是为了锁定代码关键区域以进行同步，锁定代码如下：

lock (this){ Some statements ;}

·使用Monitor

当有需要进行线程管理的时候可以使用：

Monitor.Enter(this);

其他也有一些方法进行管理，这里就不一一提及了。

线程的缺点

线程自然也有缺点，以下列出了一些：

·如果有大量的线程，会影响性能，因为操作系统需要在他们之间切换；

·更多的线程需要更多的内存空间

·线程会给程序带来更多的bug，因此要小心使用

·线程的中止需要考虑其对程序运行的影响

·通常块模型数据是在多个线程间共享的，需要一个合适的锁系统替换掉数据共享

Java

Java对多线程的支持是非常强大的，他屏蔽掉了许多的技术细节，让我们可以轻松的开发多线程的应用程序。

Java里面实现多线程，有2个方法

继承 **Thread**类

```
classMyThreadextendsThread{

publicvoidrun(){
//这里写上线程的内容
}


publicstaticvoidmain(String[]args){
//使用这个方法启动一个线程
(newMyThread()).start();
}

}
```

实现 **Runnable**接口

```
classMyThreadimplementsRunnable{

publicvoidrun(){
//这里写上线程的内容
}


publicstaticvoidmain(String[]args){
//使用这个方法启动一个线程
(newThread(newMyThread())).start();
}

}
```

一般鼓励使用第二种方法，因为Java里面只允许单一继承，但允许实现多个接口。第二个方法更加灵活。

C++ 11

ISO C++ 11 标准在STL中提供了std::thread类，因此多线程变得非常容易。

```
#include<thread>

usingnamespacestd;

voidthreadFunc(){
//这里写上线程的内容
}
intmain(){
threadt(threadFunc);
//启动线程
t.join();
//等待线程运行完毕
return0;
}
```

一个采用了多线程技术的应用程序可以更好地利用系统资源。其主要优势在于充分利用了CPU的空闲时间片，可以用尽可能少的时间来对用户的要求做出响应，使得进程的整体运行效率得到较大提高，同时增强了应用程序的灵活性。更为重要的是，由于同一进程的所有线程是共享同一内存，所以不需要特殊的数据传送机制，不需要建立共享存储区或共享文件，从而使得不同任务之间的协调操作与运行、数据的交互、资源的分配等问题更加易于解决。 [3]

线程同步

在多线程应用中，考虑不同线程之间的数据同步和防止死锁。当两个或多个线程之间同时等待对方释放资源的时候就会形成线程之间的死锁。为了防止死锁的发生，需要通过同步来实现线程安全。在Visual Basic中提供了三种方法来完成线程的同步。在Java中可用**synchronized**关键字。

代码域同步

使用Monitor类可以同步静态/实例化的方法的全部代码或者部分代码段。

手工同步

可以使用不同的同步类（诸如WaitHandle, Mutex, ReaderWriterLock, ManualResetEvent, AutoResetEvent 和Interlocked等）创建自己的同步机制。这种同步方式要求你自己手动的为不同的域和方法同步，这种同步方式也可以用于进程间的同步和解除由于对共享资源的等待而造成的死锁。

上下文同步

使用SynchronizationAttribute为ContextBoundObject对象创建简单的，自动的同步。这种同步方式仅用于实例化的方法和域的同步。所有在同一个上下文域的对象共享同一个锁。 [4]

虽然多线程能给大家带来好处，但是也有不少问题需要解决。例如，对于像磁盘驱动器这样独占性系统资源，由于线程可以执行进程的任何代码段，且线程的运行是由系统调度自动完成的，具有一定的不确定性，因此就有可能出现两个线程同时对磁盘驱动器进行操作，从而出现操作错误；又例如，对于银行系统的计算机来说，可能使用一个线程来更新其用户数据库，而用另外一个线程来读取数据库以响应储户的需要，极有可能读数据库的线程读取的是未完全更新的数据库，因为可能在读的时候只有一部分数据被更新过。使隶属于同一进程的各线程协调一致地工作称为线程的同步。下面我们只介绍最常用的四种线程同步方式：

临界区（critical section）

事件（event）

互斥量（mutex）

信号量（semaphore）

通过这些类，可以比较容易地做到线程同步。

HT定义

超线程（HT）是英特尔所研发的一种技术，于2002年发布。超线程的英文是**HT**技术，全名为**H**yper-**T**hreading，中文又名超线程。超线程技术原先只应用于Xeon处理器中，当时称为Super-Threading。之后陆续应用在Pentium 4中，将技术主流化。早期代号为Jackson。

特点

通过此技术，英特尔成为第一间公司实现在一个实体处理器中，提供两个逻辑线程。之后的Pentium D纵使不支援超线程技术，但就集成了两个实体核心，所以仍会见到两个逻辑线程。超线程的未来发展，是提升处理器的逻辑线程，英特尔有计划将8核心的处理器，加以配合超线程技术，使之成为16个逻辑线程的产品。

英特尔表示，超线程技术让（P4）处理器增加5%的裸晶面积，就可以换来15%~30%的效能提升。但实际上，在某些程式或未对多执行绪编译的程式而言，超线程反而会降低效能。除此之外，超线程技术亦要操作系统的配合，普通支援多处理器技术的系统亦未必能充分发挥该技术。例如Windows 2000，英特尔并不鼓励使用者在此系统中利用超线程。原先不支援多核心的Windows XPHome Edition却支援超线程技术。

AMDBulldozer“推土机”

据相关消息透露,在HotChips会议上,AMD宣布下一代代号为Bulldozer“推土机”的处理器架构将采用单核多线程技术(multi-threadingtechnology),类似于Intel著名的超线程技术.

AMD没有透露有关其多线程能力和更多的细节,只说推土机处理器将在2011年推出,支持单核多线程技术.不过,AMD的做法和Intel的 HT是不同的,更类似于Sun的同步多线程技术(SimultaneousMulti-Threading),由1个物理核心扩展到4个线程.“推土机扩展出的单核心多线程技术和Intel的超线程采用的是不同方式.”AMD的代表PatConway也证实了这一点. 有趣的是,早些时候AMD还表示暂不考虑SMT或其他多线程技术,并将它应用在当下的处理器中.然而,AMD也认同步多线程是未来处理器产品大幅提升性能的必要特征.

推土机是AMD下一代微架构的处理器,事实上,它将是AMD自2003年后第一次对处理器架构进行重大改变.新一代的处理器将提供远高于现代产品的高性能,同时也加入SSE5指令集.

首款推土机系列桌面处理器代号为Orochi,将会拥有超过4个以上的处理器核心,8M以上的缓存并支持DDR3内存,基于32nm工艺.服务器版处理器代号为Valencia和Interlagos,这两款处理器将会拥有6、8以及12个处理器核心.

AMD至今从未采用过同步多线程（SMT）也就是Intel所称的超线程技术。虽然这样的技术在当年的P4时代显得并无实际用途，但到了2015年为止，越发普及的多线程环境让超线程重新焕发了青春。

发展前景

截止到2014年，以应用环境来看，超线程技术可以让一些特定应用程序显著提速达10到15%。除了Intel的在Nehalem、Atom等中引入的超线程，无论IBM的Power系列，Sun的T1/T2/Rock系列等处理器架构都应用了类似的SMT同步多线程技术，用少量的晶体管带来大幅度的多线程性能提升。

一位AMD工程师日前向媒体坦诚，不支持单核多线程技术让Opteron处理器看起来性能比不上Intel的低端Xeon。据称，AMD内部高层已经承认，没有早早引入此类技术是一项技术选择上的失误。

不过，AMD副总裁兼服务器工作站业务总经理Patrick Patla接受采访时，并没有明确透露单核多线程技术的未来，而是继续重申已经公布的Opteron路线图：“如果你看一下我们路线图以及我们在多线程处理器市场的表现就会知道，我们相信每条线程都拥有完整的核心是目前的最佳选择。2010年，我们就会推出12核处理器，2011年16核。我们相信未来几年内我们就能够完善支持48或64线程环境，让我们来看看2012到2013年会带来些什么吧。”

既然2011年才是16核，那么2012到2013直接跳跃到48甚至64核似乎并不是那么正常。另外，Patrick Patla前面句句都在讲“核”，而到了后面又变成了“线程”，似乎就在暗示到时AMD可能会采纳单核多线程技术。

并行计算

参考资料

1. 多线程-维基百科 . 维基百科[引用日期2014-07-14]
2. 王建新 隋美丽. Labwindows/CVI虚拟仪器测试技术及工程应用. 北京：化学工业出版社，2011年9月：P316~P317
3. http://www.yesky.com/212/1743212.shtml
4. http://www.vckbase.com/document/viewdoc/?id=1708

词条标签： 中国电子学会， 软件