计算机科学2002Vol. 29№. 8

多线程技术及其在串口通信中的应用

The Technology of Multithreading and Application on Serial Communication

贾广雷　刘培玉　耿长欣

（山东师范大学计算机系　济南250014）

Abstract　In this article, we illustrate the concept of process and thread firstly, then discuss the application of multithreading in serial communication. We put forward and implement a model of serial communication based on multithreading. Finally we point out some problems to be solved.

Keywords　Process, Multithreading, Serial communication, Overlapped

1　引言

多线程技术能很好地解决并发多任务问题，提高资源的利用率，因此在计算机的许多研究领域都对其进行了探讨，从不同的角度实现了多线程。从大的方面讲，一是从计算机体系结构、多线程处理机硬件实现对多线程，即多线程计算机的研究。如 Stanford 的 DASH、MIT 的 Alewife 等；一是从软件角度实现多线程，即也可以在非多线程处理机上实现，这一方面主要表现在多线程编译系统、多线程操作系统成为系统软件的主流，如 IBM 的 OS/2、Sun 的 Solaris、Mircosoft 的 windows 系列等。

本文首先对多线程的概念进行了分析，然后结合实时串行通信中的不足和存在的问题，提出并实现了一个基于多线程的串口通信模型，在实际应用中取得了较好的效果。最后指出了多线程应用中需要解决的一些问题。

2　多线程技术

2.1　进程和线程

简单地说，进程就是程序的一次执行，具有动态性、并发性和独立性。进程是系统分配资源的基本单位，在自己的地址空间上运行，拥有各自独立的资源。一个进程包括代码、数据、堆栈、文件 I/O 和信号表等，具有动态性、并发性和独立性。进程间的关系如图1所示。

由图1可见，进程的上下文是很庞大的。在传统的操作系统中，进程也是 CPU 调度的基本单位，也叫单线程进程。随着计算机应用的深入，许多情况下需要处理并发多任务。传统上，由操作系统按照一定的策略调度各个进程进行处理的。这种方法有许多缺陷，随着问题的复杂化，越来越明显。主要表现在：

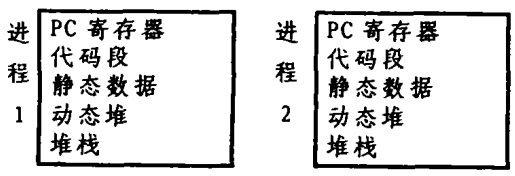

图1

(1)由于各个进程的上下文是相当庞大的，因此进程作为调度的基本单位，调度时进程间上下文切换引起的开销很大，调度越频繁、程序规模越大，这个问题就越突出；

(2)由于不同的进程有独立的地址空间和系统资源，一个进程创建一个子进程时，子进程一般通过复制来继承上下文，包括复制代码段、数据段、堆栈、进程表项、段表、页表等资源，这种开销也很大。而且由于子进程和父进程在逻辑上是共享一定资源的，因此也会造成资源的浪费；

(3)进程间的通信、同步时很多情况下须通过系统调用来完成，开销也很大的；

(4)因为单线程进程并发运行时所占资源很多，所以许多操作系统限制用户进程总数，如很多 UNIX 版本的典型值为40～100，这对许多并发应用是不够的。

为解决多进程调用在处理并发多任务的不足，提出了线程的概念。把进程中的执行代码与资源分离，则在一个地址空间中可执行多条指令流，每条执行流就是一个线程。线程目前还没有一个严格而统一的定义，简单地讲线程就是程序中的单一顺序执行流，如图2示。

由图2可见，线程间的上下文要轻得多，同一进程内的多个线程共享同一地址空间，动态堆、静态数据区及程序代码等为各线程所共享。进程作为独立的实体，为线程提供运行的资源并构成静态结构。线程为维护自己的控制流而保存寄存器和堆栈，线程是处理机调度的基本单位。

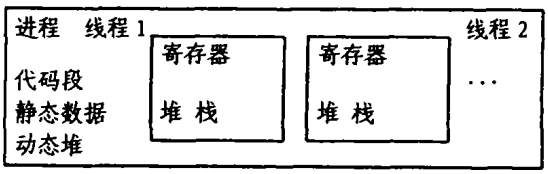

图2

通过以上的分析可知多线程技术的优点有：

(1)由于线程的上下文较轻，所以线程间的切换开销较小；

(2)同一进程的线程之间共享地址空间和系统资源，在创建的时候，不必复制庞大的上下文，减小了开销；

(3)线程间的通信和同步也比进程开销小；

(4)由于线程占有较少的资源，因此多线程操作系统中支持线程的总数要比单线程进程多很多，如 OS/2系统支持4096

贾广雷　硕士研究生，主要研究领域为智能卡 COS，数据加密。刘培玉　硕士生导师，教授，主要研究领域为数据库，网络安全。耿长欣　硕士研究生，主要研究内容为网络安全。

个线程。

进程是资源分配的基本单位，线程是CPU调度的基本单位。这种方式解决了单线程进程的不足，提高了系统资源的利用率和程序的执行效率。

2.2 多线程技术的实现

多线程在实现方式上可分为内核级多线程、用户级多线程和混合级多线程。

(1)内核级多线程(KLT，Kernel-level threads)：所有的线程都由操作系统内核来产生和管理，如WINDOWS NT、OS/2都是支持内核级多线程的操作系统；

(2)用户级多线程(ULT，User-level threads)：用线程库的方式来实现并调度线程，是语言级的线程提供机制，如Pthread等；

(3)混合级多线程：内核级和用户级同时提供对多线程的支持，用户线程库调度用户线程映射到内核，系统内核调度内核线程运行，是前面两种的综合。如Solaris2.x、Mach等。

3 多线程在串口通信中的应用

在工业监控、数据传输等许多领域，串口技术具有连接方便、可用于长距离通信等优点，在工业监控、数据传输等许多领域都有着广泛的应用，但控制起来较复杂。

基于WINDOWS3.X及以前的版本可编制出基于单线程进程和消息处理机制的通信处理程序，在处理多任务上有了很大的提高，也有了良好的图形界面，但也存在一些不足之处。单线程进程机制由于上下文庞大，通信开销大，而且系统是基于非抢占式多任务处理的，在处理抢占式多任务的时候也会遇到很大困难，不能很好地解决实时性的问题。如何开发出实时性良好、吞吐量大、人机交互友好的串口通信程序成为一个需解决的问题。根据我们以上的讨论，我们知道线程上下文较轻，通信开销也很小，我们可以把各个任务分配到多个线程上并行运行，可以很好地来满足多任务的需要。而且当前流行的WIN32系统采用抢占式多线程处理，可以给每一个线程规定一个优先级，如根据各个任务的时间紧迫性的高低给每一个任务定义一个数值来表示优先级的高低，调度时选择高的先执行，优先级相等时轮流执行，而且优先级低的线程运行的时间，优先级高的线程到达，前者让出CPU让后者使用。这种方法能很好地实现抢占式多任务的处理，提高数据传输的吞吐量和系统的执行效率。

基于多线程技术的一个串口通信的模型如图3所示。各个模块的描述如下：

(1)主线程：响应用户的输入，提供前端的人机交互界面；完成串口资源的打开、参数设置、关闭；其他线程的创建、关闭和协调运行；接收监听线程发来的信息，并调用相应的线程处理程序；

(2)串口监听线程：实时监视串口的状态，一旦发生预定的事件，就立即向主线程发送相应消息，请求主线程对其进行处理；

(3)输出处理线程：接收来自主线程的数据并将其写入串口；

(4)输出处理线程：从串口读出数据并传送给主线程处理；

(5)其他处理线程：响应一些预定事件并进行相应处理等；

(6)串口资源：提供一个或多个串口资源。

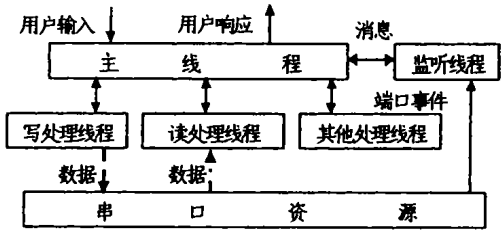

图3

这个模型采用多线程技术，使前端人机交互部分、中间处理部分和后台的串口操作部分并行处理。让耗时的I/O操作在后台运行，在大数据量通信的情况下对改善程序的响应速度相当有效的，并且对多个串口设备同时操作或对一个串口同时进行读写操作的处理也非常成功，提高了程序的响应速度和资源的利用率。

我们用VC6.0实现了这一模块，并在实际应用中取得了较好的效果。在实现时，需注意的两个问题是对串口的操作和各线程间的同步。

对串口的操作流程一般是：打开、设置参数、使用串口、关闭串口，这可以由一系列WINDOWS API函数来实现，用VC++6.0实现部分代码如下：

```
m_hCom = CreateFile (m_sPort, GENERIC_READ | GENERIC_
WRITE,0,NULL,OPEN_EXISTING,FILE_ATTRIBUTE_NOR-
MAL|FILE_FLAG_OVERLAPPED,NULL);//打开串口
    SetupComm(m_hCom,4096,4096);//设置缓冲区
SetCommMask(m_hCom, EV_RXCHAR);//设置事件掩码
DCB dcb;
f(!GetCommState(m_hCom,&dcb))
      return false;
  dcb.fBinary=true;
  dcb.BaudRate=9600;//波特率
  ……
  return SetcommState(m_hCom,&dcb);//设置串口参数
```

对多线程的同步问题，现在的操作系统提供了许多安全、高级的线程同步控制方法。以WIN32系统为例，系统提供了一些同步对象来协调各线程的执行，同步对象有：事件对象(EventObject)，冲突区(CriticalSection)，信号量(Semaphore)和互斥(Mutex)。

(1)事件对象(Event Object) 用来标志某个事件是否发生，从而确定是否执行某个线程。当一个线程在执行某项任务之前，需要等待某一事件的发生的时候，使用事件对象非常方便。

(2)冲突区(Critical Section) 是一种保证在某一时刻只能有一个线程能够访问某种资源的方法。非常类似于令牌环形网中的令牌。受保护的资源用一个冲突区对象来表示，这个对象为所有线程共享，哪个线程占有该对象，哪个线程就可以访问受保护的数据，也只有拥有该对象的线程才可以访问该数据，并且只有当占有对象的线程释放对象，其他线程才可以抢占它。

(3)信号量(Semaphore) 是允许多个线程同时访问同一资源的方法。信号量的核心对象用于资源计数，维持着一个最小值为0的计数，为0时对象无信号，这时候除非有线程释放该资源，否则其他线程无法访问该资源。大于0时对象有信息，这时候其他线程可以访问该资源。当有一个新的线程夺得了资源，计数值就会减1；当某线程释放了资源，计数值就会加1。

(4)互斥对象(Mutex) 是用来控制共享资源的方法。每个线程要访问共享资源，都要首先获得互斥对象折拥有权。应

(下转第130页)

对象所提供的服务。MOM 允许客户机和服务器是不可用的。通过在 GIOP/IIOP 中引入"中间路由代理"来完成这一功能，这些代理主要完成存储-转发功能。当目标对象当前不可访问时，代理保存客户发送的消息。当客户和目标对象之间的连接由于异常原因被关闭时，代理也可以保留对象实现的应答。至于请求和应答的生命周期，则由相应的 QoS(Quality of Service)来控制。即允许客户和服务器在不同的时间里运行。客户发出的请求可以不在客户对象的的生命期内完成。这个模型在松散耦合的企业间情况下非常有用，这两方没有必要等着事项处理完成。

第三，在 CORBA 平台上，ORB 向客户方的应用屏蔽了许多与分布式计算有关的细节，如对象的定位、网络连结的建立和请求的发送等。使在实现 CORBA 规范时有很大的自由度，并使 CORBA 平台的易用性体现得非常充分。但是，过多地屏蔽实现细节使应用难以控制底层 ORB 所提供的服务的质量。CORBA 消息服务通过允许客户说明它所需的服务质量来弥补这一漏洞，如客户可以提出对消息分发、消息排队和消息优先级的要求。CORBA 的 QoS 策略是，让 QoS 可以在多个层面上发生作用：在 ORB 层，它影响所有的请求；在线程层，它影响该线程发出的所有请求；在对象引用层，它影响所有发给这一对象的请求。

第四，允许服务器决定何时从队列里检索出消息。服务器对象既可以在先进先出的基础上从队列里检索出消息，也可以依据某种优先权或负载均衡模式从队列里检索出消息。服务器也可以使用消息过滤器仍掉它们不想处理的消息，或可以把这些消息传送给其它的服务器。队列既可以是持久(记录在磁盘上)的，也可以是非持久的(在存储器里)。队列的类型取决于用户规定的服务质量。

第五，允许客户提出无阻塞请求。MOM 允许对象提出不要阻塞客户执行线程的异步请求。这就意味着用户的客户机没有必要是多线程的。这样它们更易于编写和维持。

MOM 风格的通信使 CORBA 更灵活，更有耐力。

3. 怎样实现基于 CORBA 的 MOM

有下列几种方式实现 CORBA 和 MOM 结合：

·客户端和服务器端两端都可以通过一个服务质量属性规定消息接发形式。利用服务质量，客户和服务器都可以规定请求或响应的形式。QoS 可以包括以下属性：确认等级、存活时间、优先权、成本、可靠性、路由选择、事项处理支持和持久性等级。

·客户机可以在单个调用的基础上设置 QoS。客户机可以使用 current 准对象逐个调用地控制 QoS，管理器也可以设置一个缺省 QoS。在规定了 QoS 之后，客户机可以进行一个普通的 CORBA 调用。

·客户机规定一个回叫对象去处理响应。延迟响应被发送

(下转第144页)

(上接第149页)

用互斥对象编程的时候最复杂，但也是控制共享资源最为强大的方法，它不仅能够在一个进程内的线程之间实现资源的完全共享控制，而且可以在不同的进程的线程之间实现。

我们在本模块的实现中，可以用事件对象来实现各个线程的同步。在没有预定事件时，输出处理线程、输入处理线程和其他处理线程挂起以消耗尽量少的资源，监视线程检测到有预定事件时，用一事件对象通知主线程。请求相应的处理。主线程接收发送来的消息，自己处理或唤醒相应的线程处理程序，使信息得到实时处理。这种方法比用轮询法效率要高得多，编程也更为雅致。

用 VC++6.0实现的部分代码如下：

```
if ((m_hPostMsgEvent = CreateEvent (NULL, TRUE, TRUE,
    NULL))==NULL)
  return FALSE;
memset(&m_osRead, 0, sizeof(OVERLAPPED));
memset(&m_osWrite, 0, sizeof(OVERLAPPED));
if ((m_osRead.hEvent = CreateEvent (NULL, TRUE, FALSE,
    NULL))==NULL)//为重叠读创建事件对象
  return FALSE;
if ((m_osWrite.hEvent = CreateEvent (NULL, TRUE, FALSE,
    NULL))==NULL)// 为重叠写创建事件对象
  return FALSE;
  ClearCommError(pDoc->m_hCom,&dwErrorFlags,&ComStat);
  if(ComStat.cbInQue){
   WaitForSingleObject (pDoc-> m_hPostMsgEvent, INFINITE);
   // 等待 WM_COMMNOTIFY 结束
  ResetEvent(pDoc->m_hPostMsgEvent);
  PostMessage(pDoc->m_hWnd,WM_COMMNOTIFY, EV_RX-
CHAR,0);//通知视图
  continue;
}
  dwMask=0;
if(!WaitCommEvent(pDoc->m_hCom, &dwMask, &os)){
  if(GetLastError()==ERROR_IO_PENDING)
     GetOverlappedResult (pDoc-> m_hCom, &os, &dwTrans,
TRUE);// 无限等待重叠操作结果
    Else{
      CloseHandle(os.hEvent);
      return (UINT)-1;
    }
}
```

结束语 通过以上的讨论，我们可看到多线程技术能很好地解决各种逻辑并发性和物理并行性问题，改善系统的性能如吞吐量、计算速度、响应时间等，提高程序的执行效率和资源的利用率，同时也提高了程序的可读性和稳定性，在实际的开发中有着非常广泛的应用。多线程有着诸多的优点，但同时也带来了一些新的问题，这随着系统规模的增大而更加突出，主要表现在：

(1)多线程结构比原来要复杂，线程规模越大，程序设计的复杂性也越大，可维护性也越差；

(2)多个线程的调度和管理是要耗费系统资源的，线程越多、调度越频繁，耗费越大，甚至会降低系统实际可得到的性能。

从另一方面，线程规模太小又不足以体现多线程技术的优点，不能充分提高系统的资源利用率。如何解决多线程在设计、运行时的复杂性，以及在资源利用率、用户作业周转时间、系统吞吐量等诸多方面达到最佳，以确保在任务完成的基础上达到整体最优，是多线程技术应用研究中面临的一个新的课题。

参 考 文 献

1 张宏莉，田耕，胡铭曾. 多线程技术与并行运算，计算机科学，1999,26

2 李春华，徐明，周兴铭. 多线程的软件实现，计算机工程与科学，1999,21(4)

3 Denver A. Serial Communications in Win32 Microsoft Windows Developer Support

4 陈章龙，陈泽文 编著. IBM-PC 机软硬件接口及试验. 北京：人民邮电出版社