进程、线程和纤程

赵海廷
(武汉工程职业技术学院,湖北　武汉:430080)

摘　要　多任务需要多任务操作系统的支持,而多任务的实现有赖于具体的编程来实现。多任务有进程、线程和纤程三种实现方法。文中综合介绍了进程、线程和纤程实现多任务的相关技术并给出一个示例。

关键词　进程　子进程　线程　纤程
中图分类号:TP361　　**文献标识码**: A　　**文章编号**:1671-3524(2004)04-0036-06

1　多任务操作系统的分类、多任务的实现方法

一般将操作系统定义为:操作系统是直接控制和管理计算机硬件、软件资源的最基本的系统软件,该软件可以方便用户充分、有效地利用这些资源,增加整个计算机的处理能力。

1.1　多任务操作系统的分类

随着微型计算机的发展,微型计算机的功能已经达到或超过早期的小型计算机乃至中型汁算机。而微型计算机的操作系统也由早期的单用户单任务发展为单用户多任务,甚至是多用户多任务。所谓的多任务,是指在一台计算机上能同时运行多个应用程序的能力。

在一台计算机上同时运行多个应用程序,二十年前只是大、中、小型计算机操作系统才具备的这种功能。随着微型计算机的发展,Windows 等操作系统的涌现,多任务操作系统已是当今的主流。目前,适应于微型计算机的多任务操作系统就有十余种。

多任务操作系统按系统结构可以分为内在式(Built-in)和帖附式(Bolt -on)。内在式是指在操作系统开始设计时就把多任务功能构筑在其中,这样的操作系统有 OS/2 和 UNIX 操作系统等。帖附式是指把多任务功能附加到一个单任务操作系统上而形成的,此类操作系统如 Microsoft 的 Windows 操作系统就是在 DOS 操作系统的基础上加入多任务功能而成的。

按操作系统调度方式的不同它们又可以分为抢先式(preemptave)和合作式(cooperative)。抢先式是指由系统内部时钟来决定 CPU 由一个任务转移去执行另一个任务的时刻,转移时刻是系统以某种分配策略预先确定的,应用程序无权干涉。任务间切换频繁,需要保存和恢复的信息量比较大。OS/2 和 UNIX 都是以抢先方式进行工作的。合作式是指几个应用程序联合动作,通过某种通信方式来分享 CPU。该类操作系统任务间切换安全、方便,切换时需要保存和恢复的信息量小。Wndows 和 Novell 公司的 Netwar 网络操作系统就是以合作方式工作的。

1.2　多任务的实现方法

以 Windows2000 和 WindowsNT 操作系统为例,其多任务的实现有三种方法。

其一是多进程。进程一般认为是应用程序的运行实例,它是应用程序的一次动态执行,它是资源的分配单位。系统为每个进程分配独立的 4GB 地址空间,其间包含有代码、数据和堆栈等等。每个进程所拥有的资源随着进程的产生而产生,也随着每个进程的中止而被撤销。依据进程的定义可以看出,所谓的多进程就

收稿日期:2004－08－20
作者简介:赵海廷,男(1948～),武汉工程职业技校学院信息工程系,副教授,研究方向:自学习和自适应技术.

是：相应的操作系统允许多个程序同时驻留在内存，也可以是同一个应用程序在内存中有多个副本，即同一个应用程序可以同时运行多次。

其二是多线程。线程一般认为是进程内部的一个可执行路径或线索，它是 CPU 调度的单位，是进程的一个执行单元。所谓的多线程就是：一个进程内可以有一个或多个可单独执行的单元，每个单元就是一个线程。

请注意：磁盘上的一个可执行文件仅仅是一个文件，当它启动以后就成了进程。进程仅仅是存在，它是呆板的，不做任何事情。一个进程中起作用的东西是线程。

其三是纤程。纤程是一种由应用程序自己负责调度的“轻量级”的线程。它运行于调度它的线程的上下文中，每个线程可以调度运行多个纤程。

2 进 程

进程(process)这个概念最早是由麻省理工学院的 J·H·Saltzer 于 1966 年提出的，并应用于 MULTICS 系统设计中。而 IBM 公司则称“进程”为任务(Task)，并首先在 CTSS/360 系统中实现。

2.1 进程和进程的状态

进程概念的引入是为了描述系统中的各种并发活动，但是由于并发活动的复杂性，人们强调的侧重各不相同，时至今日还没有统一的定义。在 1978 年全国操作系统会议上将进程定义为：进程是一个具有独立功能的程序关于某个数据集合的一次运行活动。

进程具有动态性和并发性两个重要的属性。

动态性：程序是有序指令的集合，是一个静止的概念。进程是程序的执行，是一个动态的概念。程序作为一种资源可以长期存在。进程是程序的一次执行，进程具有生命过程，即有自己的诞生、执行和消亡的生命周期。

并发性：几个进程可以同时存在于一个系统中，各个进程各自独立地以不可预知的速度向前推进。进程是系统进行资源分配和调度的一个独立单位。各个进程有着运行和等待交替的运作规律。需要注意：对于一个单 CPU 的计算机系统，任何时刻只能有一个进程占用 CPU 资源，其它进程只能等待。

进程有就绪(Ready)、运行(Runing)和阻塞(Blocked)三种基本状态，它们在一定的条件下可以相互转换。

2.2 进程的相关参数

所有的进程都有一个与之相关联的句柄。每一个进程除了具有句柄之外，还有命令行、环境变量等。

(1)进程句柄

加载一个进程都会被分配一个唯一的句柄，它们可以是每一个可执行文件或 DLL 文件。通常可以通过调用 CreateProcess()函数创建一个新的进程。当新的进程创建之后，会被加载到一个地址空间上。可执行文件加载的基地址由链接程序决定。Visual C++链接程序使用的默认地址是 $0_x00400000$。当然也可以通过 Microsoft 的链接程序来改变应用程序加载的基地址。一般文件加载的基地址不小于 $0_x0400000$。也可以通过 GetMocluleHandle()函数返回可执行文件或 DLL 文件加载到进程地址空间所用的句柄(基地址)。

当一个进程打算操作另外一个进程时，使用实例句柄显得非常方便。例如进程 A 打算操作进程 B，进程 A 就必须首先得到进程 B 的实例句柄，再实现相应的操作。

进程的终止却有多种：当进程的主线程返回时；当进程的所有线程运行终止时；当控制台进程按下 Ctrl+C 或 Ctrl+Break 组合键时；当计算机关闭或注销时；当线程调用了 ExitProcess()或 TermmateProcesS()函数时：进程都将终止。

(2)进程的命令行参数

在 WinMain()函数中，另一个参数是命令行。它是指向传递给程序进程的命令行参量地址的长指针。Windows 用户可以通过运行对话框改变程序的一些命令，达到预定的目的。

(3)进程的环境变量：

进程的环境变量包含有环境信息的字符串，比如驱动器、路径、文件名以及一些可以被系统使用的标记符。Wmdows 环境变量主要包含三个级别，即系统环境变量、用户环境变量和存在于 AUTOEXEC·BAT 中的环境变量。环境变量列出了文件搜索的路径、临时文件的目录、特殊操作和其它相关的信息。

2.3 子进程

所谓的子进程就是在一个进程所拥有的地址空间内再启动一个新的进程，这个新的进程相对于启动它的进程称为子进程，而启动子进程的进程称之为父进程。

在程序的开发过程中，经常会遇到需要执行另外一块代码的操作。完成此项操作可以是：函数调用，执行新的线程或者是使用新的进程。

调用函数时，源程序必须停止运行，等待被调用的函数执行完毕后再继续执行。函数调用适用于单任务操作系统的情况下。线程调用可以使子线程和主线程分别运行，但是当子线程和主线程需要交换数据时，线程的使用比较复杂。

子进程是一个比较好的方法。新的进程可能也需要对地址空间中数据进行改动。此时，最好让进程在它自己的地址空间中运行，并且只让它访问父进程地址空间中的数据，这样就能够保护与手头正在执行的任务无关的数据。在 Windows 系统中，进程之间共享数据的方法主要有动态数据交换(DDE)、OLE、管道、使用剪贴板和内存文件映射等。

新的进程可以完全脱离父进程，同时它也可以像函数调用一样，只有当子进程执行完毕之后，父进程再继续执行。

3 线 程

3.1 线程和线程的状态

(1)线程的定义

线程的概念与子程序的概念类似，是一个可独立执行的子程序。一般线程定义是进程式内的可以执行的单元，它是系统分配 CPU 时间资源的基本单元。一个应用程序可以创建多个线程，生成多个不同的执行流，并同时运行这些线程。线程机制使得系统具有多任务的功能，这样用户就可以同时运行多个应用程序，而每个应用程序中又可以同时运行多个线程，这些线程并发地运行在同一个进程之中。一个进程至少拥有一个线程，即主线程。主线程终止，进程也终止。主线程以函数地址形式(通常为 Main 或 WinMam 函数的地址)被启动代码提供给操作系统。也可以根据需要创建其它线程，每个线程都共享创建它们的父进程的内存空间、全局变量和系统资源。

(2)线程与进程的差别

进程由两个部分组成：一个是进程内核对象，一个是地址空间。同样，线程也由两个部分组成：一个是线程的内核对象，操作系统用它来管理线程；线程的内核对象同时也是系统用来存放线程相关信息的地方。一个是线程堆栈，它用于维护线程在执行代码时所需要的有关函数参数和局部变量。

进程是死板的。进程不执行任何操作，它只是线程的容器。线程是“动态”的，线程总是在某个进程环境中创建，只是在该进程中具有它的整个生存期。线程代码的执行和对数据的操作是在它的进程地址空间中完成的。因此，在一个进程环境中运行的多个线程将共享单个地址空间。这些线程能够执行相同的代码和操作相同的数据；还能共享内核对象句柄。

进程需要相对较大的地址空间，这些空间用于存放系统资源及其相关的信息；而线程只有一个内核对象和一个堆栈，所需要保存的信息也比较少。

(3)线程的状态

一个新建的线程在它的生命期中有五种基本状态。

新建 一个新生的线程对象处于新建状态，它有了相应的内存空间和其它资源并被初始化。

就绪 处于新建状态的线程被启动后，将进入线程队列排队等待 CPU 时间片，此线程已具备运行的条件。在此线程获得 CPU 资源时，就可以脱离创建它的主线程而独立开始自己的生命周期。而原来处于阻塞

状态的线程被解除阻塞也将进入就绪状态。

运行 就绪状态的线程被调度获得CPU 处理器资源时，就进入运行状态。每个线程对象都有一个重要的 run()方法。在线程对象被调度执行时，它将自动调用本对象的 run()方法，从第一个语句开始执行这个线程的操作和功能。

阻塞 一个正在执行的线程在某些特殊情况下，将进入阻塞状态，例如被人为地挂起。在阻塞状态时，线程不能进入排队队列。只有引起阻塞的原因被解除后，线程才能进入就绪状态，进入就绪状态才能进入线程排队队列等待CPU 的资源，以便于从原来中止处继续运行状态。

消亡 线程消亡的原因有二：其一是正常运行的线程完成了它的全部工作而退出；其二是线程被强迫地终止执行。

线程的各个状态之间的转换及线程生命周期的演进，由系统运行的状态、同时存在的其它线程以及线程本身的算法所共同决定。在创建和使用线程时要注意利用线程的方法进行控制。

3.2 Visual C++对多线程编程的支持

VC++支持多线程应用程序的开发，应用程序的每个线程，都是一个 CWinThread 对象，VC++将线程划分为两类：工作者线程(worker thread)和用户界面线程(user-interfacethread)，这两种类型都基于CWinThread，并且都要调用 AfxBeginThread 来创建。

如果线程能自生自灭并做某些事情，例如后台计算，而不需要与用户交互，这类线程称为工作者线程。工作者线程没有消息循环，不处理窗口消息，用于后台执行任务。它也是基于 CWinThread 的，但是不用显式地创建 CWinThread 对象，这是因为 AfxBeginThread 调用已经创建好了对象，这类线程是最常用的类型。

如果要处理用户的输入并响应用户产生的事件和消息，就要创建一个用户界面线程，它是通过自己的消息泵获取从系统接收到的消息。MFC 将为该线程增加一个消息循环和消息泵，以便能够处理接收到的消息。这个消息循环不同于应用程序主线程(CWinApp 及其派生类对象)的消息循环。事实上，主线程本身就是一个用户界面线程，这是因为 CWinApp 派生于 CWinThread。正如 MFC 从 CWinThread 派生出CWinApp一样，用户可以从 CWinThread 派生出自己的类来实现用户界面线程。

我们无论采用那一种途径实现多线程，编程人员可以控制的操作有：定义用户线程的操作，就是定义用户线程的 run()方法，在适当的时候建立用户线程实例。

3.3 线程的控制与启动

一个工作者线程的实现比较简单：只需要编写线程控制函数和启动线程即可。线程控制函数为：UINT My Controlling Function(LPVOIDpParam)。而线程启动函数为：

CWinThread * AfxBeginThread(AFX -THREADPROC pfnThreadProc，LPVOID pParam，intnPriority = THREAD-PRIORITY-NORMAL，UINT nStackStze = O，DWORD dwCreatFlags = 0，LPSECURITY-ATTRIBUTES lpSecurityAttrs=NULL)；

与工作者线程相比，用户界面线程的实现有显著的不同。它要求从 MFC 派生用户界面线程类，要完成InitInstance()、ExatInstance()和 Run()函数的重载，再启动线程。

4 Windows 系统下的纤程

4.1 纤程

纤程是一个必须由应用程序手动调度的执行单元，它运行于线程的上下文中。Micrsoff 公司为什么给Windows2000 增加纤程？原因是便于 Windows 系统能够非常方便地将现有的 UNIX 服务器应用程序移植到 Windows 系统中。UINX 服务器应用程序属于单线程应用程序，但是它能够为多个客户服务。也就是说，UNIX 应用程序的开发人员已经创建了他们自己的线程结构库，他们能够使用这些线程结构库来仿真纯线程。该线程能够创建多个堆栈，保存某些 CPU 寄存器，并且在它们之间切换，以便为客户机请求提供服务。

显然，若要取得最佳的性能，这些 UNIX 应用程序必须重新设计，仿真的线程库应该用 Windows 提供的

纯线程来代替。然而,这种重新设计需要花费数月乃至更长的时间才能完成,因此许多公司首先将它们现有的 UNIX 代码移植到 Windows 系统中,这样就能够将某些应用软件推向 Windows 市场。

在将 UNIX 代码移植到 Windows 系统中时,一些系统间的不兼容问题将发生,尤其是 Windows 系统管理线程的内存堆栈的方法要比简单地分配内存复杂得多。Windows 内存堆栈开始时物理地址存储器的容量比较小,根据需要再逐步扩大。由于机构化异常处理机制的原因,代码的移植就更加复杂。

4.2 纤程的使用

为了能够更快和更正确地将它们的代码移植到 Windows 系统中,Microsoft 公司在 Windows2000 操作系统中增加了纤程。每一个线程可以调度多个纤程。一般来讲,纤程和设计得很好的多线程应用程序相比并没有很大的优势。不过,使用纤程可以更容易地操作应用程序,而这些应用程序管理着自己的线程。

从系统的观点来看,纤程对创建它的线程有认同感。换句话来说,如果纤程要访问线程本地存储器的话,它将访问创建它的那个线程的纤程的本地存储器。如果纤程调用了 ExaThread 函数,则创建纤程的线程将退出。不过,和线程相比,纤程并不具有与其完全相同的状态信息。纤程维护的是一个堆栈和其中保存的纤程创建数据。

纤程并没有抢先调度机制,用户可以在纤程之间切换。系统调度线程运行,如果正在运行的纤程的线程被抢先调度,则当前在线程中运行的纤程将被抢先调度。

4.3 与纤程操作有关的函数

只有纤程才能执行其它纤程。如果有线程需要执行一个纤程,它必须调用函数 Convert Thread To Fiber 来创建一个用于保存纤程状态信息的数据区。存储的纤程状态信息包括传递给 Convert Thread To Fiber() 函数作为参数 lpParameter 的纤程数据。函数原型是:LPVOID Convert Thread To Fiber(LPVOID lpParameter);其中的参数是新纤程的纤程数据。该函数为纤程的执行环境分配相应的内存(约 200 字节)。

当对纤程的执行环境进行分配和初始化后,就可以将执行环境的地址与线程关联起来。该线程被转换成一个纤程,而纤程则在该线程上运行。ConvertThreadToFiber()函数实际上返回纤程的执行环境的内存地址。虽然必须在晚些时候使用该地址,但是决不应该自己对该执行环境数据进行读写操作,因为必要时纤程函数会对该结果的内容进行操作。如果纤程(线程)返回或调用 ExatThread()函数,则纤程和线程都会终止运行。除非打算创建更多的纤程以便于在同一个线程上运行,否则没有理由将线程转换为纤程。

函数 CreateFiber()用于从一个现有的纤程中创建新的纤程。该函数的原型为:

LPVOID CreateFiber(DWORD dwStackSize,LPFIBER-START-ROUTINE

LpStartAddress,LPVOID lpParameter);

其中第一个参数是指定初始化堆栈的大小,第二个参数指向纤程函数的起始地址,第三个参数是纤程数据。这个起始地址通常是一个用户提供的函数,它需要一个称之为纤程数据的 IpParameter 参数。如果纤程函数返回了,运行该纤程的线程也就退出了。纤程函数 SwitchToFiber()可以执行任何由 CreateFiber()函数创建的纤程。该函数只有一个参数 LPVOID IpFiber,它是上次调用 Convert ThreadToFiber()或 CreateFiber()函数时返回的纤程执行环境的内存地址。该地址告诉该函数要对哪个纤程进行调度。

SwitchToFiber()函数是纤程获得 CPU 时间的唯一途径。由于程序代码必须在相应的时间显式调用 SwitchToFiber()函数,因此对纤程的调度可以实施全面的控制。纤程运行所依赖的线程始终都可以由操作系统终止其运行。当线程被调度时,当前选定的纤程开始运行,而其它纤程则不能运行,除非显式地调用 SwitchToFiber()函数。

纤程通过调用 GetFiberData 宏可以检索到纤程数据,通过调用 GetCurrentFiber 宏可以检索到纤程的地址。还可以调用 DeleteFiber()函数清除与纤程有关的数据。

4.4 纤程编程举例

```
#define -WIN32-WINNT 0x0500
#include"windows.h"
#include"iostream.h"
#include"winbase.h"
```

```
strutt FiberData{void * pfiberl, * pfiber2, * pfiber3;};
void stdcall fiber2(void * )
{cout<<"进入到纤程 2"<<endl;int result=0;
FiberData * p=(FiberData * )GetFiberData();SwitchToFiber(p->pfiber3);
cout<<"返回到纤程 2"<<end1;for(int i=0;i<=100;i++)result+=i;
SwitchToFiber(p->pfiber3);cout<<"1 到 100 的累加和是"<<result<<endl;
cout<<"切换到 main 纤程"<<end1;SwitchToFiber(p->pfiber1);}
vold -stdcall fibet3(void * )
{FiberData * p=(FiberData * )GetFiberData();cout<<"第一次进入纤程 3"<<endl;
SwitchToFiber(p->pfibet2);cout<<"又一次进入纤程 2"<<end1;
SwitchToFiber(p->pfiber2);}
int main(int argc,char * argv[])"
{FiberData fd;cout<<"转换线程为纤程"<<endl;
fd.pfiberl=ConverThreadToFiber(&fd); cout<<"创建纤程 2"<<end1;
fd.pfiber2=CreateFiber(0,fiber2,&fd);cout<<"创建纤程 3"<<end1;
fd.pfiber3=CreateFiber(0,fiber3,&fd);cut<<"切换到纤程 2"<<end1;
SwitchToFiber(fd.pfiber2);cout<<"返回到 main 纤程"<<end1;
return 0;
}
```

5 结束语

目前，线程函数的编写比较普遍，纤程则是随着 Windows 操作系统而推出的，在具体的编程经验等诸多方面还比较缺乏。但是可以相信，随着 Windows 操作系统的普及，在纤程函数编程方面也将有所突破。

参考文献

[1] 张昆苍编著.操作系统原理 DOS 篇[M]. 北京：清华大学出版社 2001.

[2] 方可燕主编 VisualC++6.0 实战与精通[M]. 北京：清华大学出版社，2001.

[3] 求实科技.编著 Windows 系统编程[M]. 北京：人民邮电出版社 2002.

[4] 范辉主编. VisualC++6.0 程序设计简明教程[M].北京：高等教育出版社，2001.

[5] 寥 俊、段爱民译 Windows32 位编程指南[M]. 北京：清华大学出版社 1996.

Process，Thread and Fiber

Zhao Haiting

Abstract：The article introduces multitask implementation with operating systern supporting，implementing multitask of multiprocess、multithread、Fiber under windows.

Key words：process， subprocess，thread，fiber

（责任编辑：栗 晓）