# Windows 95 多线程串口通信应用程序设计

刘　伟　刘光斌　余志勇　（西安第二炮兵工程学院　710025）

**摘要**：介绍了 Windows 95 线程多任务的特点和调度原理，给出了应用 Windows 95 线程多任务编制串口通信应用程序的方法。

**关键词**：串口通信　线程多任务　程序设计

## 一、概述

在远程监控系统、诊断维护和管理自动化中，一般都涉及到异步串行通信，并且对异步串行通信的实时性、吞吐量要求都很高。对此人们提出了许多解决方案，如在 Dos 环境下基于中断方式的通信编程和在 Windows 环境下基于单线程的消息处理机制的通信编程。由于在 Dos 环境下基于中断方式的通信应用程序需要直接与硬件打交道，并且是单一任务系统，这对提高应用程序的多任务处理能力和吞吐量时就显得无能为力。文献[1]较详细的说明了这种方法存在的难点和编程时应注意的要点。在 Windows 环境下基于单线程的消息处理机制的通信应用程序在程序的可移植性、实时性上较 Dos 通信应用程序有了很大的改善，但由于这种通信应用程序是基于单线程的消息处理机制的程序，使得其在提高实时性、增大吞吐量方面上的效果不是很理想，尤其在实现抢先式多任务时更是显得无能为力。利用 Windows 95 多线程抢先式任务的特点编制通信应用程序，可以较好的解决这些问题，并且为以后移植到多 CPU 操作系统奠定了基础。

本文主要讨论利用 Windows 95 多线程抢先式多任务的特点编制通信程序时的方法，并给出了与其有关的 API 函数。

## 二、Windows 95 多线程简介

Windows 95 操作系统支持两种类型的多任务：基于进程的协同式多任务和基于多线程的抢先式多任务。基于进程的协同式多任务，就是指两个或两个以上的进程可同时运行，而基于多线程的抢先式多任务是 Windows 95 操作系统以后提出的新概念。在这种多任务中，线程是进程内的一个执行路线，它是 Windows 95 的唯一执行单位，是 Windows 95 为程序分配 CPU 时间的实体。每个进程除了系统自动为它生成的主线程外，还可以创建多个线程，并由各个线程来协同完成指定操作。利用多线程可以执行某些实时性或随机性很强的操作，从而使应用程序对 CPU 的利用率大大提高，这对于加快信息处理速度，提高通信程序的实时性和增大吞吐量是非常有益的。

Windows 95 操作系统以优先级为基础安排所有的活动线程，这一点有点类似 Dos 应用程序中的中断调度机制。线程的优先级范围从最低的 0 直到最高的 31，其调度机制如图 1 所示。操作系统在给多个线程分配 CPU 时，将通过它本身的原始调度程序来评价各活动线程的优先级，以时间片（quantum）为执行单位来执行优先级最高的活动线程，挂起优先级低的活动线程。如果有优先级相同的活动线程，原始调度程序则按轮转方式向它们提供时间片。当优先级最高的各活动线程运行完毕后，操作系统才开始给优先级次高的各活动线程分配 CPU 时间，以此类推，直到运行完所有活动线程。对于非活动线程（比如处于挂起状态的线程），则不参与这种分配，并且 CPU 的时间片永不会分给它。一旦这些非活动线程被激活，将立即参与 CPU 时间的分配。

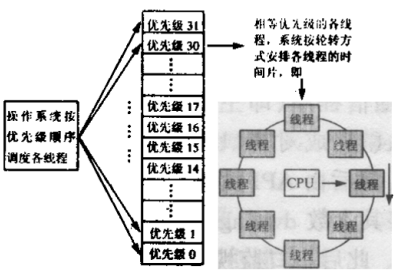


图 1　多线程的调度机制

值得注意的是，如果一个优先级低的活动线程正在

运行，而此时系统确定有一个更高优先级的线程准备运行，则系统立刻将较低优先级的线程挂起(即便它处于时间片中间)，把 CPU 分配给那个拥有满时间片的、具有更高优先级的线程。也就是说，较高优先级的线程总是抢先较低的线程，而不管较低优先级的线程正在执行什么，这也正是多线程抢先式多任务叫法的由来。

## 三、应用线程多任务编制串口通信程序

各系统之间的串口通信要求串口通信程序做到实时性高、吞吐量大，必要时能进行抢先式调度运行，以便能迅速处理重要的突发事件。这些要求在采用 Dos 环境下的中断方式编程和 Windows 环境下的单线程的消息处理机制方式编程时，并不能完美地解决，而利用线程多任务的特点，则可以较完美地解决这些问题。应用线程多任务编制串口通信程序时，需要考虑如何在串口通信程序里建立线程及如何协调好串口通信程序里各线程之间的关系。

### 1. 串口通信程序里的线程剖析

串口通信程序一般由三大部分组成：前端人机交互部分、中间处理部分及后台的串口操作部分。要使串口通信程序做到实时性强、吞吐量大，理想的处理机制是并行处理这些事件，即能同时处理后台对串口的操作、中间的数据处理过程及前端的人机交互，利用 Windows 95 操作系统中的多线程就能很好地具备这种能力。基于多线程的串口通信程序至少应具有如下几个线程：主线程、串口监视线程、读事件线程及写事件线程。主线程是串口通信程序的管理者，用来进行人机交互的操作及协调好各线程运行；串口监视线程的职责是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，主线程在接到串口监视线程发送来的消息后，立即调用相应的线程进行处理。在应用多线程编制串口通信程序时，其中很关键的问题是如何建立串口通信线程及协调好各线程的运行。

串口通信程序(即主线程)首先通过 API 函数 SetCommState()完成对串口的初始化操作，初始化参数由 DCB 给出，然后由 API 函数 CreatThread()建立串口监视线程，并将其参数 dwFlags 设置为 0，即使串口监视线程立即执行。此后串口监视线程将在后台对串口进行实时监视，在监视到预定事件时，立即向主线程发出相应的消息，如接收到数据就发送接收到数据消息 M2(WM-COMMRECV)，而串口监视线程在发送此消息后就执行后面的程序代码，对串口继续进行监视，做到了发送消息和监视串口两不误，提高了实时性。与此同时，主线程可以完成人机交互操作、数据的处理以及协调好各线程的运行，比如根据串口监视线程发来的消息作出相应的处理，假设此时串口监视线程发送了一条接收到数据消息，主线程就调用读事件线程，让其读取串口接收缓冲区中的数据。主线程还可同时进行其他工作，如接收或处理键盘、鼠标一类的消息，数据显示等。在完成对串口的操作后，主线程就调用 API 函数 ExitThread()删除串口监视线程，并终止运行串口通信程序，其过程见图 2。

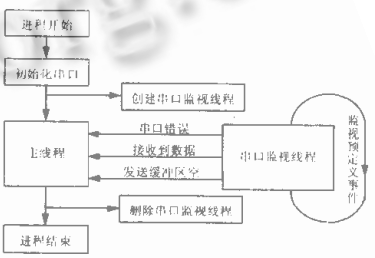


图 2　串口通信程序框图

串口监视线程是串口通信程序的核心，它具有如下几个功能：实时监视预定义的事件，给主线程发送预定义的消息。其处理过程如图 3 所示。

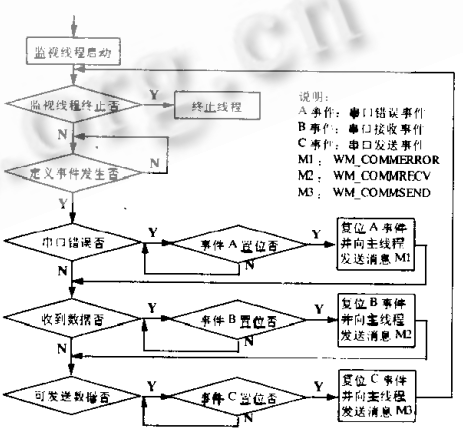


图 3　串口监视线程的实现

首先，串口监视线程通过 API 函数 SetCommMask()设置串口事件掩模，然后判断监视线程是否终止，如果终止，则退出监视线程，不然则调用 API 函数 WaitCommEvent()监视预定事件的发生。一旦预定事件之一发

生,则按如下顺序进行处理:

(1)判断串口是否发生了串口错误事件,"是"则判断此时主线程是否置位了自定义的串口错误事件(Signaled),"是"则立即复位该事件(Unsignaled),并向主线程发送串口错误消息,接着进行下一步,"否"则一直等待该事件置位;如果没发生串口错误事件则直接进行下一步;

(2) 判断串口是否接收到数据,"是"则判断此时主线程是否置位了自定义的串口接收事件,"否"则一直等待该事件置位,"是"则立即复位该事件并向主线程发送串口接收数据消息,然后进行下一步;如果没有发生串口接收事件则直接进行下一步;

(3) 判断串口是否可以发送数据,"是"则判断此时主线程是否置位了自定义的串口发送事件,"是"则立即复位该事件并向主线程发送串口可发送数据消息,再进行下一步;"否"则一直等待该事件置位;如果不需要发送数据,则直接进行下一步;

(4) 判断此时串口监视线程是否终止,"是"则立即退出该线程,"否"则调用 API 函数 WaitCommEvent()监视预定事件发生,一旦发生则返回到第一步进行处理。

本文所示的串口监视线程只考虑了对串口错误、接收到数据及可发送数据事件的监视,它也可监视其他事件,如对 MODEM 事件的监视,这只要通过 API 函数 SetCommMask()设置相应事件掩模,并将其对应处理过程加在第(4)步之前即可。

2.串口监视线程与主线程的通信

主线程和子线程之间存在着信息的交换,如消息的发布、接收发送数据等等,这些都涉及到线程之间的同步问题,特别在主线程包括多个子线程,且各子线程之间也存在着这种交换时,各线程之间的同步问题就显得更为重要,这个问题如果解决不好,将会使程序不能正常工作,甚至会导致系统崩溃。各线程之间的同步常采用的方式有信号灯(semaphore)、哑信号灯(mutex semaphore)、事件(Event)及临界区(critical section object)等。根据本串口通信程序的实际情况,采用了事件同步方式,其作用机理为:在串口监视线程建立后,调用 API 函数 CreateEvent()创建如下几个事件对象:串口错误事件、串口接收事件及串口发送事件,并设置为有信号状态(Signaled)和手工复位方式。在监视到预定事件后,用 API 函数 WaitForSingalObject()来等待相应事件是否置位,如果已置位则调用 API 函数 ResetEvent()复位该事件,并向主线程发送预定消息。主线程调用相应线程处理完该消息后,就调用 API 函数 SetEvent()重新置位该事件,以便串口监视线程可以再向主线程发送同样的消息。这种机理也就是所谓的"检查并设置(Test and Set)"。

3.多线程串口通信程序的 API 函数实现

应用多线程编制串口通信程序的方法很多,现在流行的许多开发软件如 VC++,VB,Delphi 都具有这种功能,这里仅给出用 API 函数实现时几个主要的 API 函数原形:

(1) 与串口相关的 API 函数

HANDLE CreateFile( //打开串口

LPCTSTR lpFileName, // 要打开的串口名

DWORD dwDesiredAccess, // 操作方式,这里置为 GENERIC-READ and GENERIC-WRITE

DWORD dwShareMode, // 共享模式,这里置为 0

LPSECURITY-ATTRIBUTES lpSecurityAttributes, //Windows 95 中必须为 NULL

DWORD dwCreationDistribution,// 创建方式,这里置为 OPEN-ALWAYS

DWORD dwFlagsAndAttributes,// 文档属性,这里置为 FILE-FLAG-OVERLAPPED

HANDLE hTemplateFile); // Windows 95 中必须设为 NULL

BOOL SetCommState( //初始化串口

HANDLE hFile, // 串口句柄,由 CreateFile()函数返回

LPDCB lpDCB); // 初始化参数

BOOL SetCommMask( //设置串口事件掩模

HANDLE hFile, // 串口句柄,由 CreateFile()函数返回

DWORD dwEvtMask); // 被监视的事件

BOOL WaitCommEvent( //监视通信事件

HANDLE hFile, // 串口句柄,由 CreateFile()函数返回

LPDWORD lpEvtMask, //指向存放当前发生事件的变量

LPOVERLAPPED lpOverlapped,); // 指向一个重叠结构

(2) 与线程相关的 API 函数

HANDLE CreateThread( //创建线程

LPSECURITY-ATTRIBUTES lpsa, //在 Windows95 中必须设为 NULL

DWORD dwStack, //线程堆栈大小,一般为 0 表示与进程有相同大小的 Stack

LPTHREAD-START-ROUTINE lpFunc, //线程函数指针,线程的入口点

LPVOID lpParam, //传递给线程函数的参数,可用于任何目的

DWORD dwFlags, //线程运行状态,为 0 则线程立刻执行

LPDWORD lpdwID);//与线程相关的标识符

其中的线程函数必须有如下的原形:

DWORD ThreadFunction(LPVOID lpParam);

BOOL TerminateThread( //终止线程

HANDLE hThread, //CreateThread 的返回值

DWORD dwStatus); //终止时线程的状态

HANDLE CreateEvent( //创建事件对象

LPSECURITY-ATTRIBUTES lpEventAttributes, //在 Windows 95 中必须设为 NULL

BOOL bManualReset, // 复位方式,手工或自动复位

BOOL bInitialState, // 事件初始状态

LPCTSTR lpName); // 指向事件名称的指针

关于其他 API 函数的原形及使用方法,可查看 VC++,VB,Delphi 的联机帮助文件或相关书籍。

## 参考文献


[1] 张家毅:实时监测系统 C 语言实现的几个要点,自动化与仪器仪表,1998.4

[2] 朱正伟:一种基于 PC 的新型过程监测软件,自动化仪表,1998.7

[3] 谢庆国:Windows 作为实时控制环境的探讨.计算机工程及应用杂志,1998.8


(来稿时间:1998 年 11 月)