多线程 IO 方式在基于 VXI 总线的检测系统中的应用

吴华兴 范天慈 冯国强 西安空军工程大学工程学院(710038)

Abstract

To meet the real-time performance of A VXIbus-based test system,a multi-threaded I/O means is implemented in its program by properly sorting the I/O forms.And It also settled the communication and synchronization problem between threads.

Keywords:VXIbus,multi-thread,I/O means,application

摘 要

为满足基于 VXI 总线的某检测系统的实时性要求，在检测系统程序中通过合理的 I/O 形式归类和线程划分来具体实现了多线程的 I/O 方式,并解决了多线程的 I/O 方式带来的线程通信和同步问题。

关键词:VXI 总线,多线程,I/O 方式,应用程序

0 引言

基于 VXI 总线的各种系统目前已经广泛地应用在各个领域之中,尤其在工业测控中发挥了很大的作用,它们的应用大都建立在对大量数据和控制信号的 I/O 处理上。而在实时性要求较高的场合,通常需要充分发挥 VXI 总线本身的高速特性，这就对 VXI 总线系统的虚拟仪器软件开发时使用的 I/O 方式提出了很高的要求。在目前常用的多任务操作系统(如 Windows 系列操作系统)环境下,系统和其他应用程序通常要占去相当一部分 CPU 运行时间,VXI 系统的应用程序本身既要显示复杂的用户界面,随时响应用户的操作,又要对大量数据进行在线采集和处理，通常无法及时地执行部分 I/O 操作,以致难以准确监控、反映、记录系统设备的状态和变化过程。如何让 VXI 总线系统的应用程序在 Windows 系列操作系统环境下满足实时性要求,是 VXI 系统应用程序开发者必须解决的问题。

在 Windows 系列多任务操作系统下，一个应用程序除了自身的主线程外,还可以创建多个从线程,每个从线程都相对于主线程独立运行，操作系统通过为每个独立线程安排一些 CPU 时间,并以轮转方式为这些从线程提供时间片,由于 CPU 运行速度极快(纳秒级),使得这些线程好像都在同时运行,从而能极大地提高系统和应用程序的运行效率。对于需要花大量时间来进行 I/O 操作,又要保持响应用户输入的 VXI 系统应用程序来说,采用多线程的 I/O 方式(即在多个线程中同时进行 I/O 操作)是最佳的选择。下面即以基于 VXI 总线的某检测系统的应用程序为实例，详细介绍 VXI 总线系统的多线程 I/O 方式。

1 VXI 总线系统的 I/O 形式

基于 VXI 总线的某检测系统的结构如图 1 所示。用户通过键盘、鼠标和显示器与 PC 机进行交互,PC 机根据用户的输入控制三个 VXI 总线仪器：数字 I/O 模块 HPE1458、多路转换器模块 HPE1460 和万用表模块 HPE1412,再通过它们对多个外接被检测设备进行检测,将测得的有用信息返回到 VXI 总线上,最终由 PC 机显示给用户。对该系统来说，需要同时控制 HPE1460 和 HPE1412 从外设读入各种电压信号以检测多个电压源是否正常，需要通过 HPE1458 向外设的固定端口写控制信号或数据,还需要通过 HPE1458 从外设的固定端口读入数据和状态信号。

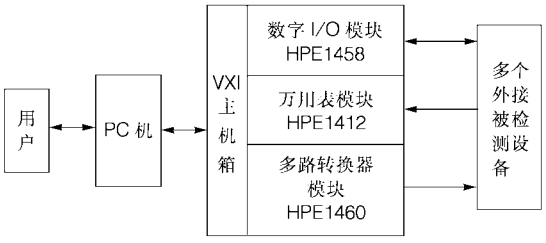

图 1 基于 VXI 总线的某检测系统的结构

因此将检测程序使用的 I/O 形式归结为三类：HPE1412 读、HPE1458 写和 HPE1458 读，并分别对应三个函数：Hpe1412Read、Hpe1458Write 和 Hpe1458Read。采用 Visual C++ 和 Agilent VISA (Virtual Instrument Software Architecture) I/O 库作为开发工具，编写函数 Hpe1412Read、Hpe1458Read 和 Hpe1458Write 的主要内容如下：

```
double Hpe1412Read(int nChannel)// HPE1412 读形式
{  //HPE1412 读外设的各种电源,nChannel 为电源的通道号
    double voltData;//最终测得的电压平均值存入该变量,作为函数返回值
    ViInt32 numRds;//实际测得的电压次数
    ViReal64 dc_rds[20];//多次采样测得的电压值需暂存于此,这里配置的采样次数为 20
    //viHPE1460 为代表 HPE1460 的内存对象,viHPE1412 为代表 HPE1412 的内存对象
    hpe1460_routClos(viHPE1460, nChannel);//选中通道 nChannel 对应的电压源
```

```
    hpe1412_read_Q (viHPE1412, dc_rds, &numRds );//从 HPE1412 取得电压值样本
    voltData = Average(dc_rds, numRds );//将所有样本值求平均,存入 voltData  return voltData;//返回最终测得的电压平均值
}
WORD Hpe1458Read(WORD addr)  // HPE1458 读形式
{   //HPE1458 读外接设备端口的操作,addr 为端口的地址
    WORD data;  //函数的返回值
    hpe1458_sourByte(viHPE1458, 2, 0x07);//写 HPE1458 控制端口 2
    hpe1458_sourWord (viHPE1458, 0, addr);//写 HPE1458 数据端口 0 和 1
    hpe1458_sourByte(viHPE1458, 2, 0x03);//viHPE1458 为代表 HPE1458 的内存对象
hpe1458_sourByte(viHPE1458, 2, 0x07);
hpe1458_sourByte(viHPE1458, 2, 0x06);
    hpe1458_sourByte(viHPE1458, 2, 0x04);
   hpe1458_measWord_Q(viHPE1458, 0, (ViPInt16) &data););//读 HPE1458 数据端口 0 和 1
    hpe1458_sourByte(viHPE1458, 2, 0x07);//
    return data;
}
void Hpe1458Write (WORD addr, WORD data)  // HPE1458 写形式
{   //HPE1458 写外接设备端口的操作,addr 为端口的地址,data 为要写入端口的数据
    hpe1458_sourByte(viHPE1458, 2, 0x07);//写 HPE1458 控制端口 2
    hpe1458_sourWord (viHPE1458, 0, addr);//写 HPE1458 数据端口 0 和 1
    hpe1458_sourByte(viHPE1458, 2, 0x03);//viHPE1458 为代表 HPE1458 的内存对象
    hpe1458_sourByte(viHPE1458, 2, 0x07);
    hpe1458_sourWord(viHPE1458, 0, data);
    hpe1458_sourByte(viHPE1458, 2, 0x06);
    hpe1458_sourByte(viHPE1458, 2, 0x02);
    hpe1458_sourByte(viHPE1458, 2, 0x06);
    hpe1458_sourByte(viHPE1458, 2, 0x07);
 }
```

2 使用多线程 I/O 方式的检测程序流程

对基于 VXI 总线的某检测系统的应用程序（以下简称检测程序）来说，涉及 VXI 系统 I/O 操作的主要步骤有：程序开始时对 VXI 系统的初始化；按工艺卡 1 到 8 项的要求对外接设备进行的操作，即有 8 个独立的步骤。而按这 8 项工艺卡进行的操作都要求对所有电压源进行监测（对设备操作的前提），都通过 HPE1458 向外设输出和从外设输入，仅在要处理的数据和端口上不相同。因此对按每项工艺进行的步骤作相同的线程划分，得到使用多线程 I/O 方式的检测程序流程，如图 2 所示。

检测程序启动运行后，将启用一个初始化线程，对系统中的 VXI 零槽控制模块、数字 I/O 模块 HPE1458、多路转换器模块 HPE1460 和万用表模块 HPE1412 进行初始化，并对外接设备中的多个端口赋初值。同时主程序（亦即主线程）将进行用户界面的初始化。待初始化线程和界面的初始化都成功完成后，进入显示工艺卡选择界面，并立即启动电源监测线程，使它作为全局线程，存在于每项工艺卡的操作中。当用户选择了某项工艺卡时，程序显示该项工艺卡对应的界面，启动该项工艺卡的从线程，此时主线程准备随时响应用户的输入来执行对 VXI 系统的 I/O 操作，而该项工艺卡的从线程则不间断地从外接设备中回读各种状态和数据。当该项工艺卡结束时，从线程亦结束，此时用户可以返回显示工艺卡选择界面，选择其他工艺卡进行操作，也可以结束主程序。

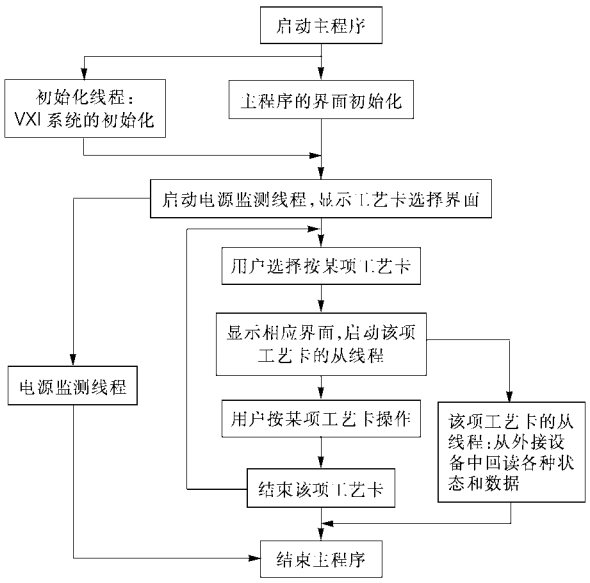

图 2 使用多线程 I/O 方式的检测程序流程

3 多线程 I/O 时的线程间通信与同步

3.1 多线程 I/O 时的线程间通信

在该检测程序中，所有从线程都需要由主程序（亦是主线程）适时启动，初始化线程结束时需要将结果通知主程序，电源监测线程需要将监测到的不正常情况通知主程序，各项工艺卡的从线程需要把从外接设备回读的各种状态和数据交给主线程处理及显示，电源监测线程和各项工艺卡的从线程都需要由主线程适时终止，这些都是程序对线程间的通信要求。因此，在检测程序中采用了 Windows 消息机制以实现线程间的通信。

以电源监测线程与主线程的通信为例。通过在程序中自定义一个 Windows 用户消息 VOLT_ABNORMAL，当电源监测线程发现有一个或多个电源不正常时，通过函数 PostMessage () 给主线程发送 VOLT_ABNORMAL 消息，并传送包含不正常电源类型的参数，由主线程处理。而主线程则通过发送 WM_QUIT 消息给电源监测线程通知其终止运行。这

样便可保证它们之间的及时通信。电源监测线程的主要代码如下：

```
#define VOLT_ABNORMAL (WM_USER+1001)//自定义的用户消息
UINT VoltMonitorThread(void* hMainDlg)//hMainDlg 为主线程窗口句柄
{   MSG msg;
    while(::GetMessage(&msg, NULL, 0, 0))//接到来自主线程的 WM_QUIT 消息时退出
    {   double volt[8];
        for(int i=0;i<8;i++)
            volt[i]=Hpe1412Read(VOLT_CHANNEL+i);//测得各通道对应电源的值
        ……//测得电压与正常范围的比较,结果存入变量 errorCode 中,省略
        if(errorCode != 0)  //存在电源不正常,给主线程发 VOLT_ABNORMAL 消息
        {   ::PostMessage((HWND) hMainDlg, VOLT_ABNORMAL,errorCode,NULL);
            break; //跳出循环,线程退出
        }
    }   return 0;
}
```

3.2 多线程 I/O 时的线程间同步

采用多线程的 I/O 方式后，检测程序可能在执行到函数 Hpe1458Write（由各项工艺卡的主线程调用）的中间时转去执行函数 Hpe1458Read(由各项工艺卡的从线程调用)中的代码,使系统无法按正确的时间和逻辑顺序对 HPE1458 和外接设备进行 I/O 操作,并导致 HPE1458 的共享访问冲突，最终会使检测程序崩溃,整个系统无法正常运作;此外,当工艺卡从线程要往主线程正在访问的存储区写数据，或者主线程要读从线程正在更新的存储区时，都会导致共享存储区的访问冲突。此即线程间的同步问题。因此在检测程序中采用了临界区(CRITICAL_SECTION)这种线程同步对象,以解决各项工艺卡主、从线程间的同步问题。

下面以工艺卡 1 为例来说明如何解决主、从线程间的同步问题。在程序中定义一个全局的临界区对象 cs1,对应 HPE1458 的内存对象 viHPE1458,由工艺卡 1 的主线程初始化和删除，并在函数 Hpe1458Write 和 Hpe1458Read 首尾分别添加下面的代码：

```
{::EnterCriticalSection(&cs1);//独占 viHPE1458 的访问权
 ……//保持原来内容
 ::LeaveCriticalSection(&cs1);//释放 viHPE1458 的访问权,在 return 语句之前
}
```

再在程序中定义另一个全局的临界区对象 cs2,对应工艺卡 1 主、从线程的共享数据存储区 readPort90,由工艺卡 1 的主线程初始化和删除,并编写从线程的内容为下面形式：

```
CRITICAL_SECTION cs2;//使用临界区来同步对 readPort90 的访问
static WORD readPort90 [5];//存放工艺卡 1 相关端口状态、数据的全局变量
UINT Item1_Read_Thread(void* hRegInDlg)
{   WORD curValue[5];
    MSG msg;
   while(::GetMessage(&msg, NULL, 0, 0))//接到来自主线程的 WM_QUIT 消息时退出
    {   curValue[0]=Hpe1458Read(0x9001);//从外接设备的固定端口读入数据
        ……//类似上面读端口的操作,省略
        ……//读入数据与原来读得的值(在 readPort90 中)比较,结果存入变量
                changeCode(WORD 型变量)中,省略
        if(changeCode != 0) //端口的状态发生改变,给主线程发 STATE_CHANGE 消息
        {   ::PostMessage ((HWND) hRegInDlg, STATE_CHANGE,changeCode,NULL);
            EnterCriticalSection (&cs2);//独占全局变量 readPort90 的存取权
            for(int i=0;I<5;i++)
                readPort90[i]=curValue[i];//读入数据放入全局变量 readPort90
            ::LeaveCriticalSection (&cs2);//释放全局变量 readPort90 的存取权
        }
    }   return 0;
}
```

这样,工艺卡的主、从线程中始终只有一个能独占对 viHPE1458 和 readPort90 的访问权,从而避免了程序对 HPE1458 和共享数据存储区的访问冲突。

4 结束语

多线程 I/O 方式的基于 VXI 总线的某检测系统的应用程序,在 Windows98 和 Windows2000 操作系统下调试通过,能及时响应用户的操作,实时监控所有电压源的情况，同时显示多个外接设备的状态和变化情况,自动记录最终检测结果,达到了该系统的实时性要求。由此可见,多线程 I/O 方式是 VXI 系统应用程序开发者值得借鉴的方法。

参考文献

1 陈光禹.VXI 总线测试平台技术. 成都电子科技大学出版社,1996
2 David J.Kruglinski 著.希望图书创作室译.Visual C++ 6.0 技术内幕.北京希望电子出版社,1999
3 Agilent VISA User's Guide.Agilent Technologies,Inc. 2000
4 Microsoft MSDN Library Visual Studio 6.0,Microsoft Corp,1998

[收稿日期:2002.10.30]