基于 RTX 的实时数据处理系统①

崔亚军[1,2], 赵 奎[2], 王鸿亮[2], 王俊霖[3]

[1](中国科学院大学, 北京 100049)
[2](中国科学院 沈阳计算技术研究所, 沈阳 110168)
[3](大连理工大学 软件学院, 大连 116024)

摘 要: 为了解决远程设备数据处理不及时的问题, 提出了一种基于RTX实时操作系统的设备数据处理系统. 该系统由 RTX 实时数据处理进程和人机交互进程两部分组成, 其中 RTX 实时数据处理进程负责接收远程设备端的数据并进行实时的处理, 将处理过的数据经共享内存和同步事件对象等通信方式, 传输到Win32的人机交互进程进行显示和存储数据, 同时用户可以通过控制指令修改远程设备的状态. 实验证明, 该系统解决了设备数据处理不及时的问题, 保证了设备管控的实时性和稳定性, 能够满足对远程设备进行实时管控的要求.

关键词: 数据处理; 实时系统; RTX; 进程通信

Real-Time Data Processing System Based on RTX

CUI Ya-Jun[1,2], ZHAO Kui[2], WANG Hong-Liang[2], WANG Jun-Lin[3]

[1](University of Chinese Academy of Sciences, Beijing 100049, China)
[2](Shenyang Institute of Computing Technology, Chinese Academy of Sciences, Shenyang 110168, China)
[3](School of Software Technology, Dalian University of Technology, Dalian 116024, China)

Abstract: In order to ensure the real-time data processing of remote device, a real-time processing system based on RTX is proposed, which is composed of RTX real-time process and man-machine interaction. The RTX real-time process gets data from remote device and processes the real-time data. Then, it displays and storages data by transferring the processed data to a Win32 user interface layer by sharing memory and synchronizing event object. Simultaneously, the user could alter remote device upon commands from man-machine interaction. Experiments have shown that the system can reduce the latency of data processing, ensure the real-time and stability of equipment control, and satisfy the real-time control requirements for remote device.

Key words: data processing; real-time system; RTX; process communication

1 引言

随着设备管控技术的发展, 以实时数据处理为基础的管控技术成为当前工业实时设备管控的关键. 目前, 虽然通用操作系统的处理能力、速度和效率等已经能够满足大部分工业生产的需要, 然而, 在实时设备管控领域, 系统不仅需要好的处理能力和效率, 还要保证设备数据处理的实时性. 为了满足这一要求, 本文采用基于实时操作系统(RTOS)的实时设备数据处理方案[1]. RTOS 能够保证数据处理请求在极短的时间内得到相应处理, 具有低成本、高灵活性、高实时性及丰富配套资源等优点[2]. 目前常用的 RTOS 有 VxWorks、RTX 和 RtLinux 等, 其中 RTX 是 Windows 下硬实时的操作系统[3]. 为了充分利用 Windows 下丰富的资源, 本文提出一种基于 RTX 的实时数据处理系统, 实现设备数据的接收、处理、显示以及人机交互等功能, 以满足系统的实时性、可靠性和安全性.

2 RTX概述

2.1 RTX 相关概念

IntervalZero 公司研发的 RTX 系统是 Windows 环

① 基金项目:国家水体污染控制与治理科技重大专项(2012ZX07505003)
收稿时间:2016-04-26;收到修改稿时间:2016-06-21 [doi:10.15888/j.cnki.csa.005534]

境下纯软件实现的硬实时操作系统[4]，其拥有采用抢占式优先级的高效任务管理器，同时可以支持大部分C/C++语言函数库和 Win32 API. RTX 通过对 Windows 硬件抽象层(HAL)进行扩展来实现独立的内核驱动模型，并增加与 Win32 子系统并存的实时子系统(RTSS)来保证系统的实时性, RTX 的体系架构如图 1 所示. 在 RTX 环境下进行实时系统的开发，有以下几点优势：1)能够保证程序的实时性；2)实时接口 RtApi 的使用与 Win32 环境类似，便于程序的开发；3)RTX 支持使用 C/C++的集成开发环境进行程序开发[5].

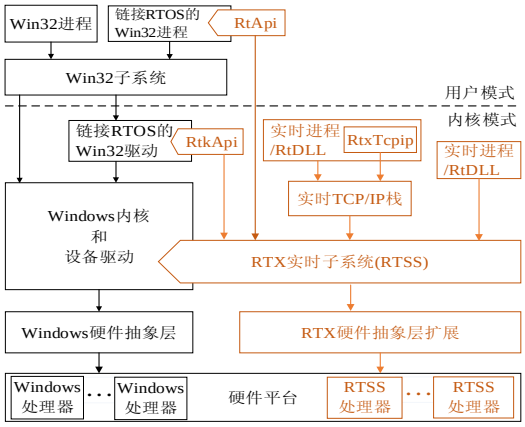

图 1 RTX 和 Windows 运行的架构图

2.2 RTX 的实时性分析

系统实时性的评价，主要从系统处理不确定事件时所消耗的线程切换延迟、中断响应延迟和定时器延迟等方面进行[6]. 本文利用 RTX64 和 Intel i3 CPU 的 PC 进行 RTX 的性能测试，并将测试结果与 Windows 7 进行对比，具体情况如表 1 所示.

表 1 RTX 和 Windows 性能的对比

延迟(us) \ 系统		Windows	RTX
线程切换	最小值	0.3	0.1
	平均值	0.3	0.1
	最大值	1000+	0.5
中断响应	最小值	2.1	0.4
	平均值	2.6	0.4
	最大值	1000+	0.8
定时器延迟	最小值	91	0
	平均值	95	1.0
	最大值	1000+	8.0

由表 1 可以看出，Windows 系统的通用性能良好，但是最大延迟是实时系统所不能接收的，而 RTX 能够保证在任何情况下，系统的最大延迟都是极小的，因此 RTX 能保证在处理不确定事件时也具有实时性.

3 实时数据处理系统设计

3.1 系统需求分析

本文系统的设计旨在实现远程设备的实时管控，该系统主要包含两个模块：实时数据处理模块和人机交互模块. 系统的数据流程是远程设备通过 RtxTcpip 协议实时地把设备数据传输到该系统，并由数据处理模块进行接收和处理，然后将处理好的数据通过进程间通信传输给人机交互模块，由人机交互模块进行数据的显示和存储. 系统的控制流程是指首先由系统的人机交互模块发送控制指令，然后通过 RtxTcpip 协议将指令实时地传输给远程设备. 该系统的实时数据处理模块主要包括数据监听、数据接收、数据解析、数据检测和数据转储等子模块. 人机交互模块主要包括指令设置、数据存储、数据分析和数据可视化等子模块，详细的功能规划如图 2 所示.

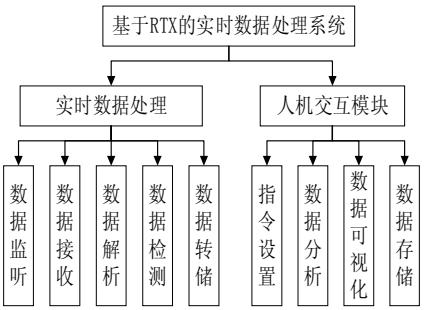

图 2 系统的功能模块图

3.2 系统整体架构的设计

系统架构是影响系统整体性能的关键因素之一，虽然 RTX 能够保证进程的实时性，但良好的架构能够减少任务调度对系统关键线程的影响[7]. 因此，在系统开发之前，有必要对系统整体的架构、各模块的运行机制以及内部的实现作详细地分析，尽可能设计出最优性能的系统，以达到相对理想的状态.

本文系统整体的结构如图 3 所示. 该系统的实时数据处理层在 RTX 环境下运行，包括实时的数据接收、解析和检测等功能，同时能够及时地进行异常中断处理，保证远程的设备数据传送到该系统时，数据能够得到实时地处理，而人机交互层在 Win32 环境下运行. 本系统利用 MFC 完成人机交互界面的设计，同时提供指令发送、数据可视化和数据存储的功能.

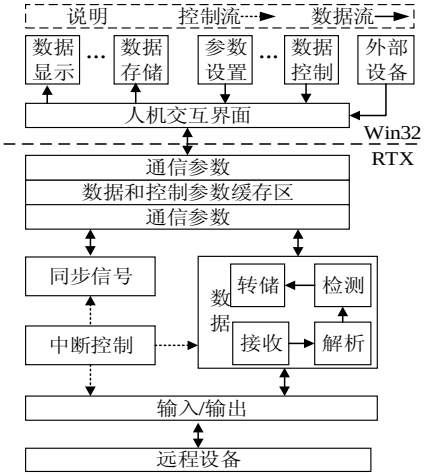

图 3　系统的整体架构图

3.3 系统的程序设计

整个系统程序流程的设计以RTX实时进程和人机交互进程之间的共享内存通信为中心. 系统的程序流程图如图 4 所示, 其中图 4 右侧为 RTX 实时进程, 图 4 左侧为人机交互进程.

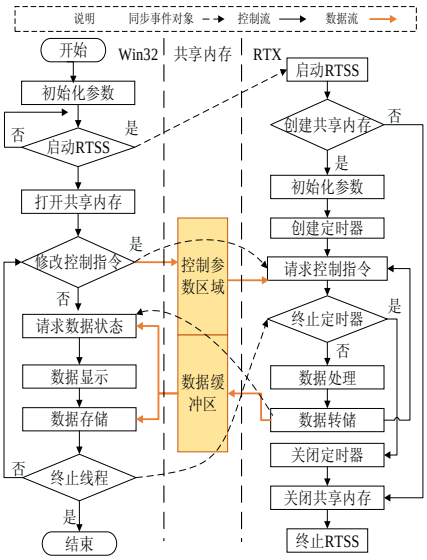

图 4　系统的程序流程图

RTX 实时进程主要包括实时主线程、数据接收线程和数据处理线程三部分. 实时处理线程负责将接收的设备数据进行实时处理, 并将结果通过共享内存的方式传输给人机交互层, 并根据人机交互的控制指令修改实时进程的状态. 人机交互进程主要由系统主线程、指令发送、数据显示和转储等多个线程组成. 指令发送线程负责发送用户对远程设备的控制指令, 数据显示线程负责对通过共享内存传来的数据进行及时可视化处理, 以向用户展示远程设备的当前状态, 同时利用数据存储线程将接收到的远程设备数据存储到系统指定的数据库中.

4　系统的实现

本文的系统开发是在 RTX64 实时环境下, 使用 C/C++语言在 Visual Studio 2013 平台进行的. 其中实时进程采用 RTX 提供的 RtApi 实时接口进行开发, 人机交互进程采用 MFC 类库进行开发.

4.1 实时进程的实现

4.1.1 实时数据接收的实现

高精度定时器是实时数据处理进程的核心, RTX 系统提供了三种类型的定时器: CLOCK_SYSTEM、CLOCK_FASTEST 和 CLOCK_3[8]. 本文采用了的时钟精度为 1us 的 CLOCK_FASTEST 类型的定时器. 在使用定时器时, 首先利用 RtCreateTimer()函数创建定时器, 然后通过 RtSetTimerRelative()函数设置定时器的时钟周期和重复间隔, 最后将数据接收执行代码放到定时器的回调函数中, 使其能够周期性的实时执行. 关键代码示例如下:

```
HANDLE hDataTimer=RtCreateTimer(NULL,0,
    DataFunc,NULL, RT_PRIORITY_MAX,
    CLOCK_FASTEST);
RtSetTimerRelative(hDataTimer,&nDataPeriod,
    &nDataPeriod);
void RTFCNDCL DataFunc(void* context){
    ...                          //数据接收代码段
    RtSetEvent(hDataParse);   //重置同步事件对象
}
```

4.1.2 实时数据解析和转储的实现

数据解析和转储线程主要包括以下几个方面. 首先数据解析线程通过 RtWaitForMultipleObjects()函数接收由数据接收线程发来的同步事件对象, 然后打开共享内存, 从中提取数据进行解析, 并对数据正确性的检测, 如果解析的数据符合标准的数据格式, 就把它交于数据转储线程进行转储到共享内存[9], 否则将错误信息封装到数据块的错误信息段, 并交由人机交互层处理. 最后由 RtSetEvent()函数重置与 Win32 通信的同步事件对象, 并通知人机交互层将共享内存中的数据及时进行显示和存储. 关键代码示例如下:

```
dwStatus=RtWaitForMultipleObjects(2,hEvent,
      FALSE,INFINITE);
if(WAIT_OBJECT_0+TERM_EVENT==dwStatus){
      RtSetEvent(hEvent[TERM_EVENT]);
      ExitProcess(0);
}
if(WAIT_OBJECT_0+EXECUTE_EVENT==dwStatus){
          if(memcmp(pData->prodData,pData->consData,
                sizeof(pData->prodData)) != 0){
      ...                          //数据转储代码段
      RtSetEvent(hEvent[TERM_EVENT]);
      ExitProcess(0);
      }
}
```

4.2 人机交互进程的实现

人机交互进程的实现是在Win32环境下使用MFC基础类库完成的，它与实时进程间的交互通过 RTX 的 RtApi 接口来完成[10]. 人机交互进程主要包括以下几个功能: 数据的显示、存储数据库、控制指令的发送. 其中，数据显示负责从实时进程中接收数据，并将数据显示到 MFC 界面，数据显示界面如图 5 所示.

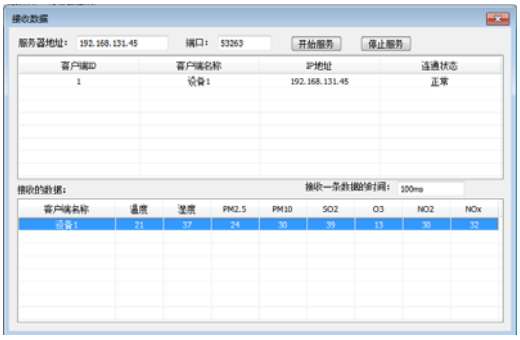

图 5 数据显示的界面图

存储数据功能负责将接收到的数据存储到数据库，为用户查询设备的历史信息提供便利. 控制指令的发送功能主要将控制指令对话框中的指令发送到远程设备，对远程设备进行控制.

5 实验测试及结果分析

本次性能测试实验的硬件配置为: CPU(Intel i3 3.60GHz)、内存(4.00GB); 软件环境为: Windows 7、RTX64 实时系统和 VS 2013 开发工具. 本实验测试的主要目的是对数据处理进程在实时环境下所消耗的时间与 Win32 环境所用时间进行对比，并分析本系统的实时性. 在实验测试中，采用高精度计数器完成时间的统计，通过 QueryPerformanceFrequency()函数和 QueryPerformanceCounter()函数分别获取计时器的频率和数值[11]，并利用两次获取的计数之差与计时器频率，计算出数据处理所花费的精确时间.

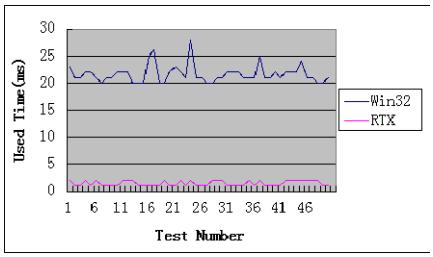

图 6 轻负载 RTX 与 Win32 性能的比较

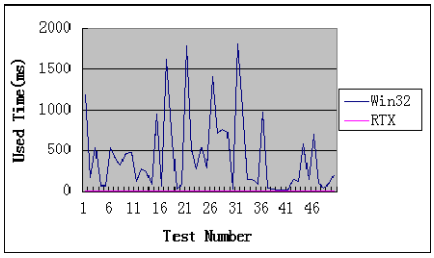

图 7 重负载 RTX 与 Win32 性能的比较

该数据处理系统在轻负载和重负载下分别进行了 50 次实验，实验结果如图 6 和图 7 所示. 在轻度负载的 Win32 环境下，数据处理时间在 22ms 左右，浮动相对较小; 在重负载的 Win32 环境下，数据处理时间浮动很大，接近秒级，因此不能达到业界对实时处理系统在 10ms 之内完成数据处理的要求. 而在 RTX 实时环境中，数据的处理时间不受 Windows 负载的影响，因此，无论在轻负载还是重负载下，实时数据处理都能够保证在 3ms 之内完成，具有很好的实时性，能够满足实时管控系统对数据处理的要求.

6 结语

本文设计的基于 RTX 的实时数据处理系统，充分利用了 RTX 的可扩展析性、实时性和高灵活性等特征，实现了抢占式优先级的高效任务管理和实时调度. 通过对数据处理系统的实时性测试，可以看出 RTX 环境为系统的实时性提供了保证，其设备数据处理的性能远远好于 Windows 环境，能够保证设备数据处理在 10ms 内完成. 因此，本文设计的数据处理系统能够很好地满足实时设备管控的需要，同时也为在 Windows

非实时环境下如何通过软硬件扩展进行实时系统的开发提供了一种有效的解决方法.

参考文献

1 White JA, Bettencourt J. Real-time experiment interface for biological control applications. 2010 Annual International Conference of the IEEE Engineering in Medicine and Biology. IEEE. 2010. 4160–4163.

2 IntervalZero lnc. RTX64 deployment guide. https://www.intervalzero.com/rtx-downloads/rtx64-downloads/rtx64-2014-downloads. [2015-10-13].

3 杨建,王建军,郭立红.基于 RTX 的某光电装备图像处理实时性研究.计算机测量与控制,2015,11(4):31–33.

4 马维斯.基于 RTX 的关节控制系统及测试平台研究[硕士学位论文].哈尔滨:哈尔滨工业大学,2014.

5 Guo C, Ni F, Zou T. Design of real-time and open control system for Chinese space manipulator joint on RTX. 2015 IEEE International Conference on Robotics and Biomimetics (ROBIO). IEEE. 2015. 2629–2634.

6 Lee YH, Kim D. Software architecture supporting integrate real-time systems. The Journal of Systems and Software, 2013, 65(4): 71–86.

7 彭健,包梦.基于RTX51 嵌入式操作系统的多路数据采集系统设计.工业仪表与自动化装置,2015,2(4):29–32.

8 雷毅,周徐昌.基于RTX 平台高速实时数据采集与处理平台设计.舰船电子工程,2010,6(7):89–96.

9 王伟,袁保君,吴佳楠.基于 RTX 的数据采集系统设计.测控技术,2012,8(10):78–81.

10 刘晓晶.基于 Windows 平台的实时扩展子系统(RTX)研究及其在雷达系统中的应用[硕士学位论文].南京:南京理工大学,2011.

11 任伟,陈韶千,王亮.基于 RTX 的网络延迟测试系统设计与实现.测控技术,2013,3(4):80–83.