嵌入式多任务操作系统中的任务间通信策略

胡修林, 杨 刚, 张蕴玉

(华中科技大学电子与信息工程系, 湖北 武汉 430074)

摘 要: 简要介绍了一种嵌入式操作系统 DeltaOS 及嵌入式图形用户界面 DeltaGUI 的结构和特点, 并着重分析了在 DeltaOS 中任务间通信的可靠方法。

关键词: 嵌入式系统; 图形用户界面; DeltaOS; DeltaGUI

中图分类号: TN919.2 文献标识码: A 文章编号: 1003-7241(2004)07-0039-04

Method of Communication for Tasks in the Embedded Multitask Operating System

HU Xiu-lin, YANG Gang, ZHANG Yun-yu

(Dpart. Of Electronics&Information Engineering, Huazhong University of Science&Technology, Wuhan, 430074, China)

Abstract: This paper presents the structure and characteristic of a embedded operating system DeltaOS and a embedded graphical user interface DeltaGUI. The reliability of the communication method for tasks in DeltaOS is analyzed specially.

Key words: Embedded system; Graphical user interface; DeltaOS; DeltaGUI

1 引言

所谓嵌入式系统是指用于特定用途的软硬件紧密结合的计算机系统。嵌入式系统早已渗透到我们生活的各个领域, 无论是手机、PDA 等消费类电子产品, 还是飞机、航天器的控制核心, 无不与嵌入式系统息息相关。DeltaOS 是中国自主知识产权的嵌入式实时多任务操作系统(RTOS), 具有高可靠性、强实时性等优点。DeltaGUI 是嵌入式系统中的图形用户界面, 运行在实时操作系统内核之上, 为需要友好用户界面的嵌入式应用程序提供支持。本文结合笔者在开发一款无线通信终端设备的过程中对 DeltaOS 和 DeltaGUI 的应用, 讨论了在 DeltaOS 中任务间通信的基本方法, 并重点分析了在 DeltaOS 中普通任务与特殊任务——GUI(图形用户界面)的通信策略。

2 实时多任务操作系统 DeltaOS 简介

DeltaOS 是北京科银京成技术有限公司开发的自主知识产权的嵌入式实时多任务操作系统。DeltaOS 的强实时内核 DeltaCORE 提供了一个多任务环境, 使应用程序可由一些相对独立的任务构成, 并且提供了任务间通信的多种手段。DeltaCORE 提供的功能包括任务管理、同步与通信、内存管理、中断管理、时间管理等。

DeltaCORE 支持最多 65535 个应用任务, 255 个任务优先级并支持动态任务优先级。DeltaCORE 采用基于优先级的可强占式调度, 对同优先级的任务还可采用时间片轮转调度。DeltaCORE 具有快速并且确定的上下文切换时间。为满足多任务应用程序的需要, DeltaCORE 提供了丰富的任务间同步和通信机制, 主要有消息队列、信号量、异步信号, 事件等。DeltaCORE 的体系结构如图 1[1] 所示。

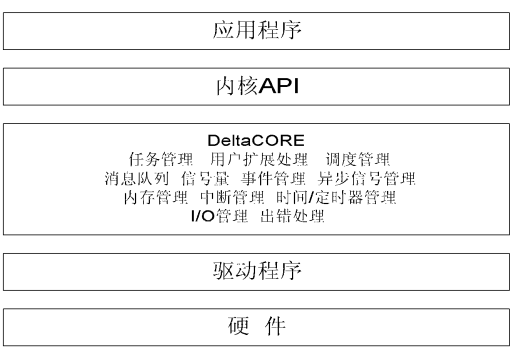

图 1 DeltaCORE 体系结构

收稿日期: 2004-02-13

3 DeltaGUI 体系结构及特点

DeltaGUI 是嵌入式系统中的图形用户界面，运行在实时操作系统内核之上，为需要友好用户界面的嵌入式应用程序提供支持。DeltaGUI 并不是一个操作系统，它仅仅是扩展操作系统人机交互功能的一个模块，位于操作系统核心之上。DeltaGUI 与目标环境的关系如图 2[2] 所示。在实际应用中，DeltaGUI 与用户基于 DeltaGUI 开发的 GUI 应用程序是作为多任务系统中的一个任务存在的，通常将这个任务命名为 GUI。这个任务受操作系统调度，也可以与系统中的其他任务进行通信，同时这个任务也具有自身的一些特点。

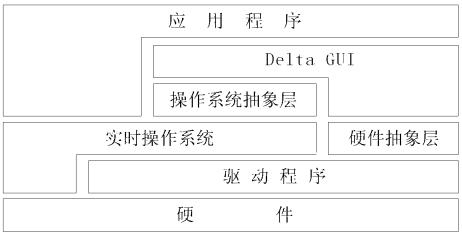

图 2 DeltaGUI 与目标环境的关系

DeltaGUI 也不是一个应用程序，它以类库的形式提供了一系列 Windows 风格的控件，使用这些控件能够开发出类 Windows 风格的图形用户界面，但是 DeltaGUI 自身并不为终端用户提供任何用于交互和显示的信息[2]。

DeltaGUI 采用了消息驱动的体系。DeltaGUI 在初始化时就创建了一个 FIFO 的消息队列，DeltaGUI 应用程序的运行就是不断地从这个消息队列中取消息并对消息进行处理的过程。当消息队列为空时，DeltaGUI 会不停地调用一个名为 DIdleFunction() 的函数，直到又有新的消息送入 DeltaGUI 的消息队列。DeltaGUI 应用程序的执行流程如图 3 所示。

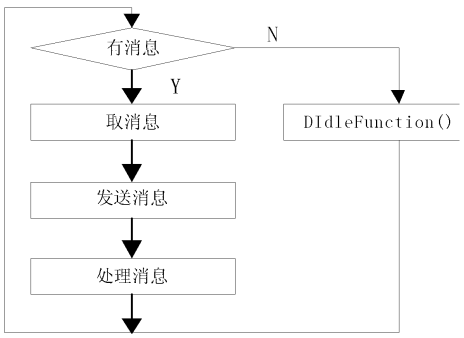

图 3 Delta GUI 应用程序执行流程

4 任务间通信的实现

在多任务的实时系统中，一项工作的完成往往需要通过多个任务或多个任务与多个中断处理程序共同完成，他们之间必须协调动作，互相配合，这就不可避免地涉及到任务之间的信息交换。提供任务间通信的手段也成为实时多任务操作系统的重要功能之一。

4.1 DeltaOS 中任务间通信的一般方法

DeltaOS 的实时内核 DeltaCORE 提供的任务间通信的最基本的方法是消息队列。消息队列用于存放任务和中断服务程序发出的消息。通常，进入消息队列的消息被任务按 FIFO 方式接收，但紧急消息将被置于消息队列的头部，首先被任务接收。

利用 DeltaCORE 提供的 API(应用程序接口)，可以方便地使用消息队列实现任务间的通信。下面针对两个任务 task1、task2 通信的程序实现，说明了使用 DeltaCORE 提供的 API 进行任务间通信的基本步骤。

第一步：建立一个消息队列。要使用消息队列，首先要建立一个消息队列。在应用中，一般都是在初始化任务中建立应用程序中所要用到的消息队列。

```
delta _ task Init() // Init()是 DeltaCORE 的初始化任务
{ ……// 其他操作
//构建消息队列的名字
Queue _ name[ 1] = delta _ build _ name(' Q', ' 1', ' ', ' ');
//创建一个名为"Q1"的消息队列
ret = delta _ message _ queue _ create(Queue _ name[ 1], 32,
600, DELTA _ DEFAULT _ ATTRIBUTES,
&Queue _ id[ 1] );
……//其他操作
}
```

以上程序段在系统中建立了一个名为"Q1"的消息队列。对消息队列 Q1 的基本描述如下：本消息队列的 ID 为 Queue _ id[1]，最多可以容纳 32 个消息，消息的最大长度是 600 个字节，该消息队列的属性为系统默认属性。

第二步：需要发送消息的任务将消息送入消息队列。这里设任务 task1 发送消息。

```
delta _ task task1()
{ ……// 其他操作
ret= delta _ message _ queue _ send( Queue _ id[ 1], buffer, 16
);
……//其他操作
}
```

任务 task1 向消息队列 Q1 中发送了一个长度为 16 个字节的消息。参数 buffer 为指向发送缓冲区的指针。这里要注意的是消息的长度不能超过建立消息队列时设置的消息最大长度。

第三步：由接收消息的任务从消息队列中取出消息。这里设任务 task2 接收消息。

```
delta _ task task2()
{ ……// 其他操作
ret = delta _ message _ queue _ receive(
Queue _ id[ 1], // 消息队列 ID
buffer, // 接收缓冲区
&size, // 消息大小
DELTA _ WAIT, //消息接收属性
```

```
DELTA _ NO _ TIMEOUT// 等待时间
);
……// 其他操作
}
```

任务 task2 从消息队列 Q1 中接收了一个消息。参数 buffer 为指向接收缓冲区的指针。参数 size 的值为消息的大小。消息接收的属性可以设置为 DELTA _ WAIT 或 DELTA _ NO _ WAIT, 这里为 DELTA _ WAIT。两种方式的区别如下。当指定的消息队列中无消息时, 若接收属性设置为 DELTA _ NO _ WAIT, 则接收消息的系统调用 delta _ message _ queue _ receive() 直接返回; 若接收属性设置为 DELTA _ WAIT, 则欲接收消息的任务被阻塞, 直到该消息队列中有消息或者设置的等待时限到期时该任务才会被唤醒。当消息的接收属性设为 DELTA _ WAIT 时可设置任务等待消息的最大时间。这里设为 DELTA _ NO _ TIMEOUT 表示当消息队列中无消息时, 任务 task2 将被阻塞, 并将永远等待, 直到成功接收到消息才转为就绪态。

以上是 Delta OS 中两个任务使用消息队列通信的基本步骤。系统中多个任务使用消息队列通信的原理与此相同。即首先建立消息队列, 然后由消息的发送方将消息送入消息队列, 最后由消息的接收方从消息队列中取出消息。

4.2 DeltaOS 中一般任务与 GUI 的通信

4.2.1 DeltaGUI 的基本工作原理

DeltaGUI 与用户基于 DeltaGUI 开发的 GUI 应用程序是作为多任务系统中的一个任务存在的(通常这个任务命名为 GUI)。同系统中的其他任务相比, GUI 这个任务有其自身的一些特点。DeltaGUI 采用了消息驱动的机制, DeltaGUI 应用程序的运行就是不断地从消息队列中取消息并对消息进行处理的过程。这个消息队列并不是用户使用 DeltaCORE 提供的 API 在应用程序中建立的, 而是在 DeltaGUI 在初始化时就创建好了的。用户的应用程序只能向这个消息队列中送入消息或者从消息队列中取出消息, 而不能对这个消息队列进行其他操作。对这个消息队列的维护以及消息的分派和路由都由 DeltaGUI 内部来完成。DeltaGUI 的消息也不同于普通任务的消息。DeltaGUI 的消息是一个固定格式的结构体, 只有这个结构体的对象才被认为是 DeltaGUI 的消息, 才可以被送入 DeltaGUI 的消息队列。因为 DeltaGUI 采用消息驱动的机制, 因此 DeltaGUI 的消息就具备了两个功能: 一个是触发接收消息的对象的操作, 另一个是传送数据给接收消息的对象。

GUI 任务本身是一个死循环, 它不停地从 DeltaGUI 的消息队列中取出消息并进行处理。当 DeltaGUI 的消息队列为空, 即 DeltaGUI 空闲时, DeltaGUI 会不停地调用一个名为 DIdleFunction() 的函数, 直到又有新的消息送入 DeltaGUI 的消息队列。

4.2.2 一般任务发送消息给 GUI

在终端产品中, 应用程序的处理结果最终是要送给界面进行显示的。这实际上就是通过处理部分的任务发送消息给 GUI 任务来实现的。任务如何发送消息给 GUI 呢? 通常 DeltaGUI 的初学者想到的办法是在任务中直接定义一个 DeltaGUI 消息并将它送入 DeltaGUI 的消息队列, 过程如下:

```
delta _ task task1()
{ ……// 其他操作
DObject *pt = NULL; //定义一个 DeltaGUI 对象
DMessage NewMesg; // 定义一个 DeltaGUI 消息
//指定消息类别
NewMesg.wType = PM _ OTHERTASK;
//指定消息接收对象
NewMesg.iData = COMM _ WIN _ ID;
//把消息放入 DeltaGUI 消息队列
pt—> MessageQueue()—> Push( &NewMesg);
……//其他操作
}
```

任务 task1 企图发送一个消息给 ID 为 COMM _ WIN _ ID 的 DeltaGUI 对象, 表面看来没什么问题, 但是隐患就在这里产生了。任务 task1 要执行, 必然要获得 CPU, 这就要打断 GUI 任务的运行(假设系统中只有 task1 和 GUI 两个任务)。如果 GUI 任务在被打断时也正在向 DeltaGUI 的消息队列发送消息(消息 1), 那么消息就没有完全放入 DeltaGUI 的消息队列。由于在 task1 获得 CPU 后也向 DeltaGUI 的消息队列放入了一个消息(消息 2), 这就导致了消息 1 被截断, 结果在 DeltaGUI 的消息队列中存放的消息成了如下的格式。

……	消息2	消息1前半段

图 4 产生错误的消息列队

由于 DeltaGUI 在取消息时是按照固定的数据结构进行的, 这样当 DeltaGUI 再次从消息队列中取消息时就会产生错误, 并且该错误将一直延续下去, 最终导致 GUI 不能正常运行。

由以上分析可得出结论, 当一般任务要给 GUI 发消息时, 不能采用在任务中直接将消息送入 DeltaGUI 的消息队列这种简单的方法。根据笔者的开发经验, 一种可靠的方法是通过 DeltaGUI 的 DIdleFunction() 函数间接地将消息放入 DeltaGUI 的消息队列。当 DIdleFunction() 函数被调用时, DeltaGUI 的消息队列必然是空的, 因此可在该函数中将消息送入 DeltaGUI 的消息队列。

假设系统中只有两个任务 task1 和 GUI, 那么这种方法原理如下。首先 task1 和 GUI 要约定一个公共的变量, 当 task1 要发送消息给 GUI 时, 对该变量置位; 在 GUI 的 DIdleFunction() 函数中检查该变量, 若已置位, 则生成一个 DeltaGUI 的消息并送入 DeltaGUI 的消息队列, 然后将该变量复位即可。详细过程如下:

```
delta _ task task1()
{ ……// 其他操作
GUI _ receive = 1; // 对公共变量置位
```

```
……//其他操作
}
DIdleFunction()
{ ……//其他操作
    if(GUI_receive == 1)//检查公共变量
  {
    DObject *pt = NULL;
    DMessage NewMesg;
    NewMesg.wType = PM_OTHERTASK;
    NewMesg.iData = COMM_WIN_ID;
    //把消息放入 DeltaGUI 消息队列
    pt->MessageQueue()->Push(&NewMesg);
    GUI_receive = 0;//对公共变量复位
  }
……//其他操作
}
```

需要说明的是, DeltaGUI 的消息格式是固定的, 仅仅依靠 DeltaGUI 的消息无法实现一次将较大量的数据传送给 GUI 的功能。当在实际应用中需要通过一个 DeltaGUI 的消息将较大量的数据传送给 GUI 时, 就需要采取一定的辅助手段。例如任务 task1 要发送数据给 GUI, 那么可定义一个公共数组, 由任务 task1 将要发送的数据写入这个数组, 然后 task1 发送消息给 DeltaGUI 的对象。这时 DeltaGUI 的消息所起的作用仅仅是触发接收消息的对象的操作, 消息本身并不携带任何信息。当 DeltaGUI 的对象收到消息后, 再从公共数组中读出数据, 进行后续处理。使用这种手段, 就实现了一个消息发送大量数据给 GUI 的功能。

4.2.3 GUI 发送消息给一般任务

在终端设备中, GUI(图形用户界面)的功能之一就是接受用户的操作, 并将用户的操作转换成数据, 交给系统中的其他任务去处理。这个功能就是通过 GUI 发送消息给其他任务来实现的。

GUI 发送消息给其他任务本质上就是 DeltaOS 中的两个普通任务进行通信。即在 GUI 中调用 DeltaCORE 提供的 API 将消息送入一个已经建立好的消息队列。这里唯一需要注意的就是选择合适发送时间。

为了不打断 GUI 的正常消息处理, 一般情况下也是在 DeltaGUI 的空闲时间, 即 DIdleFunction()函数中将消息送入消息队列。基本原理是: 在 GUI 内部约定一个公共变量, 当某一 GUI 的对象需要发送消息给其他任务时, 则在该任务的消息处理函数中将该变量置位, 同时将要发送的数据写入发送缓冲区。当 DeltaGUI 空闲时, DIdleFunction()函数执行, 在该函数中若检测到公共变量被置位, 则把消息放入消息队列中, 同时将公共变量复位。当欲接收消息的任务运行时, 直接从消息队列中取出消息即可。

假设 DeltaGUI 的一个窗口对象 MyWindow 欲发送消息给其他任务, 那么具体的过程为:

```
//窗口消息处理函数
SIGNED MyWindow::Message(const DMessage &mesg)
{ ……//其他操作
//数据写入发送缓冲区
buffer[0] = …;
……
buffer[i] = …;
send = 1; //对公共变量 send 置位
……//其他操作
}
```

相应地, DIdleFunction()函数应修改为:

```
DIdleFunction()
{ ……//其他操作
if( send == 1)// 检查公共变量
  {
send = 0;// 对公共变量复位
//消息送入消息队列
ret = delta_message_queue_send(
Queue_id[1], buffer, 40 );
  }
……//其他操作
}
```

由 4.2.2、4.2.3 两节的分析, 可将一般任务与 GUI 的通信归纳为下图:

图 5 一般任务与 GUI 的通信

5 结束语

DeltaOS 是中国自主知识产权的嵌入式操作系统。使用本文给出的方法能够可靠地实现 DeltaOS 下任务间的通信。该方法已经在基于 DeltaOS 和 DeltaGUI 开发的某卫星通信系统终端设备应用软件中的到了成功的应用。

6 参考文献:

[1] DeltaCORE2.1 编程手册[Z], 北京科银京成技术有限公司, 2002

[2] DeltaGUI2.1.2 编程手册[Z]. 北京科银京成技术有限公司, 2002

[3] 詹瑾瑜, 熊光泽, 孙明. 一种嵌入式 GUI 软件结构实现方案[J]. 电子科技大学学报, 2003, 32(1): 89-93

[4] 陈恩庆, 宋豫冀, 王忠勇. VxWorks 下图形用户界面的开发[J]. 微计算机信息, 2003, 19(3): 48-49

作者简介: 胡修林(1945-), 男, 河南滑县人, 华中科技大学电信系教授, 博导, 研究方向: 现代通信系统与通信网, 多媒体通信。