基于 uBASE-III 嵌入式程序设计的研究

肖巍[1]，文大化[2]

（1.长春师范学院 传媒科学学院，长春 130032；2.中国科学院长春光学精密机械与物理研究所，长春 130033）

摘 要：本文以 uBase-III 嵌入式操作系统为例，分析了嵌入式程序设计的特点，从内存管理、多任务并发设计和 RPC 三个方面对基于 uBase-III 上的程序设计进行了深入的研究，并给出了一定的实例分析。

关键词：uBase-III 嵌入式操作系统；内存管理；多任务；RPC

中图分类号： TP316 **文献标识码：**A **文章编号：**1672-9870（2010）04-0134-03

The Research of Programing on Embedded uBase-III Operating System

XIAO Wei[1], WEN Dahua[2]

(1.School of Media, Changchun Normal University, Changchun 130032; 2.Changchun Institute of Optics, Fine Mechanics and Physics, Changchun 130033)

Abstract: Based on uBase-III embedded operating system as an example, this paper analyzes the characteristics of embedded program design, from memory management, multitasking concarrent design and RPC based on three aspects of uBase-III procedure studied, and some example analyzed are given.

Key words: embedded uBase-III; memory management; multi-tasking; RPC

uBase-III 是一个嵌入式实时多任务操作系统，它主要由：多任务内核模块、内存管理模块、文件系统、设备管理模块、应用程序管理模块等组成。为了能够让用户快速编写出美观实用的应用程序，uBase-III 提供了一个 C++类库：UFC(uBase Foundation Class Library)，为了能够让设备能够同主机(PC 机)进行方便的信息交流，uBase-III 还提供了一套远程函数调用(RPC)，通过它可以让主机浏览设备上的信息，包括系统信息、文件信息等。

1 嵌入式程序内存管理

嵌入式设备的资源是极其有限的，因此，首先应该注意程序中的一些与资源假设相关的地方，一般来说最容易造成资源缺乏的是内存和 CPU。现在 PC 上的程序设计很少考虑到内存缺乏的问题，因为 PC 上有足够的内存支持程序运行，而且像 Windows 系统还具备虚拟内存的功能，在内存不足时还可以交换到硬盘上。而嵌入式设备一般只有几百 k 到几 M 字节的内存。嵌入式系统上的 CPU 资源也是有限的，因此，涉及到巨大运算量的程序，在 PC 上的表现同嵌入式设备上的表现就有很大的不同。

一个程序不运行的存放在设备的 Flash Rom 中，运行时被装载入 RAM 中。在 RAM 中的分布如图 1 所示。

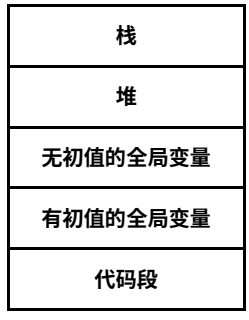

图 1 RAM 结构图

Fig.1 The structure of RAM

收稿日期：2010-05-16

作者简介：肖巍（1975-），男，工程师，主要从事图像处理与模式识别、自动识别技术、嵌入式系统方面的研究，Email：xiao_wei@foxmail.com。

“堆”是位于“栈”之下，应用程序变量区之上的那块内存。这块内存由 malloc 和 free 函数来管理，可以动态分配和回收。既然不能够在“栈”中声明大的数组，那么从“堆”中动态分配出一块内存就是唯一的选择，如：

```
void fun()
{
    char * pBuf；
    pBuf = (char * )malloc(1024*4)；
    if(pBuf != NULL){
    …
free(pBuf)；
    }
}
```

如果 malloc 分配不成功（可能是没有足够的内存），则返回的是 null 指针，使用前一定要做有效性检查（if(pBuf != NULL)…）。malloc 分配出来的内存，需要调用 free 进行释放，以免内存泄漏。

在 C++中，new 关键字也是用来从“堆”中分配出一块内存，用完后要用 delete 来删除。虽然 new 与 malloc 都是从堆中分配内存，但是不能够用 free 来释放由 new 分配出来的内存，也不能用 delete 来释放由 malloc 分配出来的内存。实际上，new 最终是调用 malloc 来进行内存分配的，只不过它额外做了一些初始化的工作。同样 delete 也是最终调用 free 来进行释放。建议在 C++程序中用 new 和 delete 来分配和释放内存，如：

```
void ShowMsg(const char *msg)
{
    CUMsgBox *box = new CUMsgBox (msg)；
    box->DoModal()；
    delete box；
}
```

但是，调用 malloc 或 new 从“堆”中分配出一块内存是一个比较复杂的过程，如果你的函数调用频率很高，对执行的效率要求很高，那么 malloc 和 new 可能会给程序造成较大的影响，在优化程序的时候要考虑用“栈”代替“堆”。

“栈”即我们通常说的“堆栈”，它是一个先进后出的队列，用于存放局部变量、函数参数等。管理栈的代码由编译器自动产生，应用程序并不需要关心一个局部变量是如何从栈上分配出来，函数参数是如何通过栈来传递的。但是，我们必须清楚的是，栈是自顶向下增长的，在嵌套的函数调用中，栈是累加的。例如：

```
void fun1()
{
    char v［100］；
}
void fun2()
{
    char v［100］；
    fun1()；
}
void fun3()
{
    char v［100］；
    fun2()；
}
```

在这个例子中，fun3()函数调用了 fun2()，fun2()函数调用了 fun1()，那么总共需要的栈的大小就是：fun3() + fun2() + fun1()。这个例子中，每个函数都声明了一个大小为 100 字节的数组，因此每个函数最少需要 100 字节的栈，所以总的栈至少需要：100 + 100+ 100 = 300 字节。一般来说，为了安全起见，栈的大小在估计的最大值再加上 2~10k。如：在多任务程序设计时，需要为每个任务设置一个堆栈尺寸，这个时候就需要估计任务需要的栈，然后再加上 2~10k。一旦程序的栈不够用，益出，导致把处于他的下方的“堆”破坏了，就有可能导致程序崩溃！减少堆栈溢出的有效方法是：

1．不要声明特别大的局部变量数组。如：

```
int fun()
{
    int buffer［1024*1024］；
}
```

就有可能导致堆栈溢出。因为 int buffer［1024*1024］需要 1024*1024*sizeof(int) = 4M 字节的堆栈！一般的嵌入式设备都难以提供如此大的运行栈。一般来说，不要声明大于 1k 的局部变量，而且注意函数调用嵌套时堆栈的累加特性。

2．少用递归算法，或者要严格控制递归的级数。递归算法通常是非常耗堆栈的，递归算法使用的堆栈 = 每一级使用的堆栈 X 递归的级数，以下是一个利用递归算法求阶乘的函数：

```
int Cal(int n)
{
```

```
    if(n > 1) return n*Cal(n - 1)；
    else return 1；
}
```

当求 10 阶乘时，需要至少 40 字节的堆栈。

2 多任务程序设计

有时需要多任务程序来完成一项特殊的任务，比如通讯程序中，就常常需要创建一个任务在后台接收数据。

uBase-III 的多任务程序设计要注意两个问题：

1.uBase-III 的“多任务”，是线程级的多任务，也就是说，所有的任务都运行在一个内存空间，共享全局的变量。

2.uBase-III 的任务调度完全基于优先级，一个任务对应一个优先级，同一个优先级只能允许有一个任务存在。uBase-III 共有 255 个优先级可用，用户的应用程序优先级被限定在 80-170 之间，即允许 90 个应用程序任务。其它优先级被用作系统的驱动程序和管理模块。

2.1 任务的同步与通信

多任务程序运行过程中，有可能出现程序的运行顺序不一致或者系统死锁，所以一套任务之间相互通信机制是不可缺少的。uBase-III 提供了一套用于任务之间相互同步的对象，包括：信号量、邮箱以及消息队列。

信号量是一个整数，通常表示系统中某一资源的数量，当信号量大于 0 时，表示该资源可用，对信号量的一次成功申请，将使信号量减 1，当信号量为 0 时，所有申请该资源的任务将被阻塞(即被挂起)，一旦某个占有该资源的任务释放资源后，所有被阻塞的任务中拥有最高优先级的任务将被唤醒，并能获得资源。

邮箱是一种通讯机制，邮箱内存放了一个指针，当该指针为空时，表示邮箱是空的，此时访问该邮箱的任务将被阻塞(即被挂起)，一旦某个任务向邮箱发送一条信息(即填写该指针)时，所有被阻塞的任务中拥有最高优先级的任务将被唤醒，并能获得指针信息，同时将邮箱再次清空。

消息队列与邮箱很类似，它由一个指针链表组成，任务通过访问消息队列，获取链表中的一个指针，同时该指针将从链表中删除。当链表为空时，访问该消息队列的任务将被阻塞(即被挂起)，直至其它任务向消息队列发送消息。

同步与通信对象的操作主要由以下函数完成：

UCreateEvent()	创建一个同步对象，返回对象的句柄
UDeleteEvent()	删除一个同步对象
UPendForEvent()	等待获得一个消息或资源
UPostToEvent()	发送一个消息或释放资源

2.2 任务挂起与恢复

正在运行中的任务可以被挂起和恢复，当任务被挂起后，将不会被调度，即不占用任何处理器资源。通过调用函数 USuspend、UResume，任务可以将自己或其他任务挂起和恢复。

任务还可以通过 USleep 函数使自己进入休眠状态，在休眠状态下的任务同样不会被调度。在指定的休眠时间结束后，系统将立即唤醒休眠中的任务使其进入就绪状态；也可以通过调用 UWakeup 函数将一个正在休眠的任务唤醒。

2.3 删除任务

当一个任务完成他的使命后，可以将其删除以回收系统资源。

需要注意的是，被删除的任务可能正在使用一些同步对象，由于系统的同步对象是全局对象，在删除一个任务时，该对象并不会受到影响，因此删除一个任务可能导致一些不希望发生的现象。

例如：主线程 A 创建了任务 B、C，同时创建了一个邮箱 M。任务 B 不断等待邮箱 M 中的消息，任务 C 不断向邮箱 M 发送消息。如图 2 所示。

图 2 多任务机制示意图

Fig.2 Schematic diagram of multi-tasking system

如果此时将任务 C 删除，任务 B 将不再获得消息，从而产生死锁。

3 RPC 程序设计

UBase-III RPC 是 PC 机与掌上机之间远程过程调用函数库，它包含通讯协议以及远程过程 RAPI，RPC 协议基于远程过程调用模型，类似于本地过程调用模型。在本地过程调用模型中，调用者把过程的参数放在特定的位置。接着，它把控制传给过程，过程的结果可以从特定的位置得到，调用者继续执行。远程过程调用模型也是类似的。控制在两

(下转第 143 页)

3　结论

利用溶胶－凝胶法制备出不同掺杂量的 Gd/TiO_2 纳米粉体，分别在不同温度下焙烧，考察焙烧温度对晶型的影响，可以看出 Gd/TiO_2 的由锐钛矿型向金红石型转变温度远高于纯TiO_2的转变温度，说明Gd^{3+}离子进入到TiO_2的晶体内，在一定程度上抑制了晶型的转变。观察了不同温度焙烧样品的形貌，随着焙烧温度的升高，粒子尺寸增大，并出现明显的聚集现象。通过对不同钆掺杂量的光催化剂对罗丹明B光催化效果的研究，得出最佳钆的掺杂量为1.0%，对于该降解反应催化剂的最佳加入量为0.5g/L。

参考文献

[1] Sugimoto T，Okada K，Itoh H. Synthetic of Uniform Spindle-Type Titania Particles by the Gel-Sol Method [J]. J. Colloid Interface Sci，1997，193 (1)：140-143.

[2] 张霞，赵岩，张彩碚，等. 低温水热合成异形 TiO2 纳米晶及其表征 [J]. 物理化学学报，2007，23 (6)：856-860.

[3] Sugimoto T，Zhou X，Muramatsu A. Synthesis of uniform anatase TiO_2 nanoparticles by gel-sol method：4. Shape control [J]. J. Colloid Interface Sci. 2003，259，53-61.

[4] 石建稳，郑经堂，胡燕，等. 铕掺杂对二氧化钛光催化降解甲基橙性能的影响 [J]. 稀土，2007，28 (2)：68-70.

[5] Yu Hua，Li Xin-jun，Zheng Shao-jian，et a1. Photocatalytie activity of Tio_2 thin film non-unifomfly doped by Ni [J]. Materials Chemistry and Physics，2006，97 (1)：59-63.

[6] 奚丽荷，朱忠其，江海军，等. 掺杂 CeO_2 的 TiO_2 光催化降解甲醛的研究 [J]. 功能材料，2007，38 (11)：1762-1765.

[7] 孙力军，侯象洋，郑雪萍，等. 掺杂 TiO_2 薄膜太阳光催化降解甲基橙的研究 [J]. 稀土，2005，26 (2)：59-61.

（上接第 136 页）

个进程中传递：调用者进程和服务器进程。调用者进程首先向服务器进程发出请求报文，然后等待回答。调用报文包括过程的参数，回答报文包括过程的结果。一旦接收到应答报文，就可获取过程的结果，调用者继续执行。

在服务器端，一个进程睡眠等待调用报文的到来。当有调用报文时，服务器进程提取过程的参数，计算出结果，发送应答报文，接着等待下一条报文。

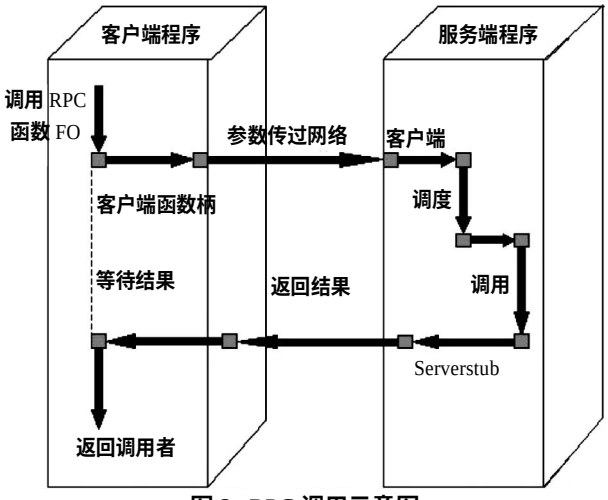

图 3　RPC 调用示意图

Fig.3 RPC call diagram

在这种模型中，在任一时刻最多只有一个进程时活跃的。因此 UBase-III RPC 协议对于并发模型是有限制的。

远程过程调用和本地过程调用有以下不同：

1 错误处理：远程过程调用中必须处理远程服务器和网络错误。

2 全局变量：由于服务器端不能读写客户端的地址空间，隐含参数不能作为全局变量传递。

3 效率：远程过程比本地过程慢。

4　结语

嵌入式实时操作系统在工业领域的应用日益广泛，通过对基于uBase-III嵌入式操作系统上程序设计中几个关键技术的研究，确保了以后在嵌入式系统的开发过程中可以避免一些问题、少走一些弯路，使系统的开发更加流畅，同时能够提高系统的稳定性和可靠性。

参考文献

[1] 夏梦宇. 基于嵌入式实时操作系统μC/OS-II 的多任务算法研究 [J]. 电子测试，2010 (3)：9-11.

[2] 曹晓燕，周岩，李欣颖. μC/OS-II 内核分析 [J]. 长春理工大学学报：自然科学版，2009，32 (1)：123-125.

[3] 刘永奎. 嵌入式实时多任务操作系统安全性的分析与研究 [J]. 现代电子技术，2008，31 (14)：25-26.

[4] 俞萍. 浅谈嵌入式系统 [J]. 光盘技术，2009 (1)：15-16.

[5] 郁发新. 常用嵌入式实时操作系统比较分析 [J]. 计算机应用，2006 (4)：761-765.