VxWorks多任务编程中的异常研究

张伟 许瑞玲

摘要：阐述了嵌入式系统中异常防止与处理的重要性，介绍了VxWorks多任务编程中遇到的异常情况，分析了形成原因并提供了相应的防止与处理方法。

关键词：VxWorks 嵌入式系统 多任务 实时处理

1 引言

随着嵌入式系统复杂性的不断提高和嵌入式软件系统规模的不断庞大，在嵌入式软件系统中不可避免会存在这样或那样的错误和缺陷。目前对于嵌入式系统，现有的测试手段还无法穷尽程序的每个分支，以致这些缺陷和错误隐蔽得很深而不能在测试阶段发现，在多任务运行过程中，会出现一些异常的情况，导致任务不能正常运行或者对操作系统造成影响，防止这些异常情况的出现和出现后进行补救就有十分重要的意义。

2 VxWorks下的常见异常情况和分析

程序异常是指出现一些很少发生的或出乎意料的状态，通常显示了一个程序错误或要求一个必须提供的回应，不能满足这个回应经常造成程序功能削弱或无法执行，有时甚至导致整个系统瘫痪，异常主要包括:程序陷阱或者断电，除数为零，浮点数或者整数溢出，非法结构，总线出错，地址出错等等。一般来说，这些异常是由于程序的逻辑错误造成的，下面介绍常见的异常的情况。

首先，在应用中可能会出现多个任务调用同一段代码的情况，由于任务占用CPU是串行的，不会出现代码资源使用冲突。但是，不同优先级的任务同时调用同一段代码，则可能出现低优先级任务执行某一函数时被执行该函数的高优先级任务打断的情况，如果函数中要改写全局变量而没有使用互斥，就有可能导致错误的存取。例如在中断中调用内存分配或者释放函数，如果某个任务正在调用内存分配函数或者是内存释放函数，打断该任务时会造成异常，可能导致内存泄漏，甚至有可能会因在中断中异常而reboot。此外，如果多个任务共用的代码中有全局变量且使用目的不同，或者多个任务的代码中有全局变量同名的情况，则有可能造成变量使用中的错误。

另外，VxWorks中有模块（module）的概念，装载模块完成目标代码文件在内存中的链接，并可以将目标代码文件中的函数与全局变量加入符号表。符号表中的符号对C语言编写的函数以原来名字命名，对于C++语言的函数则是在后面加上形参的数据类型作为符号名。如func1( )的符号名为func1__FUNCv，最后的v表示void类型；func2(int)符号名为func2__FUNCi，func3(int，char)为func3__FUNCic，依此类推。代码的编译过程中并不对要使用的函数和变量进行检查。例如调用一个并不存在的函数编译并不报错，编译器认为此函数可能在操作系统内核中或者已经下载的目标文件中，但在目标文件下

载时会找不到要调用的函数。如果应用程序中使用了与操作系统内核同名的符号，则对操作系统某些 API 函数的调用将会失败。

再有，在 VxWorks 中，当一个任务被删除，其它任务不会得到通知，而且由于任务间的独立性，每一个任务可以无限制地删除其它任务。删除一个在临界区执行的任务可能引起意想不到的后果，造成保护资源的信号量不可用，可能导致资源处于破坏状态，也就导致了其他要访问该资源的所有任务无法得到满足。不同优先级的任务是通过抢占获得 CPU 使用权的，如果不选时间片轮转，相同优先级的任务之间也是抢占 CPU 的。任务就绪队列中正在运行的任务如果不主动放弃 CPU，则其它同优先级的任务不会得到运行，这样就有可能看到几个同优先级的任务状态同为 READY，但实际上只有一个任务在运行的现象。比如在一个任务中用 taskSpawn()函数创建一个同优先级或低优先级的任务，如果原任务一直占用 CPU，新任务就不会开始运行。另外，由于中断能够打断任务的运行，中断处理函数中执行的代码就要尽可能少地占用 CPU，并且中断中不能有获取信号量的操作。一旦处于等待之中，所有的任务均得不到运行，用户可能会有 CPU 不响应的错觉。如前所述，每一个任务都有自己的堆栈，任务创建时进行初始化。每个堆栈的大小是固定，但是任务运行过程中并不对堆栈的使用进行限制。由于 VxWorks 不对内存访问作限制，栈顶超越了原定的值后出现越界，这样操作系统中该任务堆栈以外的内存区域就可能被改写，会造成难以预料的结果，甚至可能造成任务的上下文区域被改写而任务消失。

还有，在 VxWorks 中，当任务的指令执行中出现了指令非法、地址寻址错误、总线错、除数为 0 等情况时，就会出现 CPU 异常。比较常见的情况是，指针地址非法或者数组下标越界就有可能存取有效地址空间以外的地址而造成 CPU 异常。

可见，处理上述可能出现的异常显的十分必要，否则就会对整个系统造成不可预估的严重问题。

3 VxWorks 下的异常防止与处理

3.1 正确划分任务

对于功能联系比较紧密的各工作可以化为一个任务来运行。如果都以一个个任务来进行相互之间的消息通信，影响系统效率，不如采用任务中一个个独立模块完成。对于实时性要求比较高的任务，要以高优先级运行，以保证事件的实时响应。对于一个需要周期性执行的工作，应作为一个任务来运行，通过定时器以一定时间间隔激活任务。

3.2 防止任务异常

操作系统发生死锁、饥饿或者优先级翻转都会让任务处于异常状态。死锁是指多个任务因为等待进入对方占据的临界区而导致的不可自行恢复的运行终止。在程序设计过程中要注意对死锁的预防，一个是尽量使互斥资源在相同优先级任务中使用，必须在不同优先级任务中使用时，要注意对死锁的解锁处理。

饥饿是指优先级较低的任务长时间得不到系统资源而造成的任务长期无法运行。造成饥饿的主要原因时优先级较高的任务调度过于频繁或占用时间过长。合理的分配任务

的优先级和对较高优先级任务的合理调度是解决饥饿的主要策略。

任务的优先级翻转是指高优先级任务因等待低优先级任务占用的互斥资源而被较低优先级(高于低优先级但低于高优先级)的任务不断抢占的情况。有些实时多任务操作系统自身提供保护机制可对优先级翻转进行预防。在操作系统未提供保护的情况下，就需要在编程时注意避免优先级翻转的情况发生(如在同一优先级内使用互斥资源)，或采取相应的手段进行处理(如动态的进行优先级提升)。VxWorks 系统提供了自身的防止优先级翻转机制，它主要通过限制对系统调用的使用来实现。

3.3 共享代码与重入

在一个多任务环境中，函数的可重入性是十分重要的。可重入函数是一个可以被多个任务调用的过程，任务在调用时不必担心数据是否会出错。在写函数时要尽量使用局部变量(例如寄存器、堆栈中的变量)，对于要是用的全局变量要加以保护(例如采用关中断、信号量等)，这样构成的函数就一定是一个可重入的函数。

此外，编译器是否有可重入函数的库，与它所服务的操作系统有关，例如 DOS 下的 Borland C 和 Microsoft C/C++等就不具备可重入的函数库，这是因为 DOS 是一个单用户单任务的操作系统。为了确保每一个任务控制自己的私有变量，在一个可重入的 C 函数中，须将这样的变量申明为局部变量。C 编译器将这样的变量存放在调用栈上或寄存器里。

在 VxWorks 中，多个任务可调用同一个函数或函数库。VxWorks 系统利用动态连接工具使这样做相当方便，这种共享代码让系统更加高效、更加易于维护，如图 1 所示。

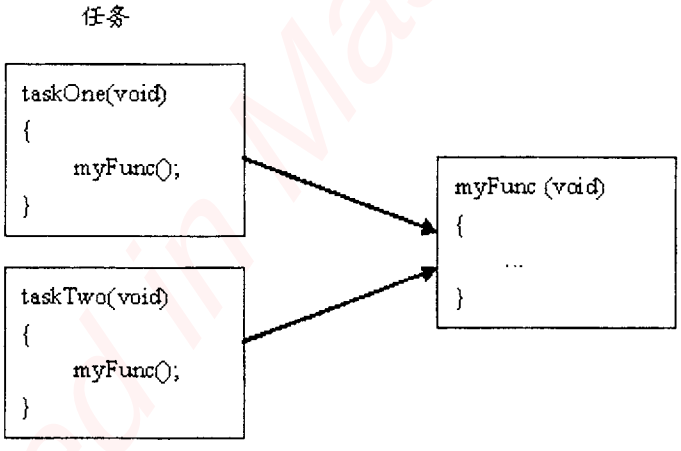

图 1 共享代码

VxWorks 系统主要采用如下几种可重入技术：

(1) 动态堆栈变量

许多子函数只是纯代码，除了动态堆栈变量外没有其他数据。调用称许的参数作为子函数的数据。这种子函数是完全可重入的，多个任务同时使用这种子函数，不会互相影响，因为它们各有自己的堆栈空间，如图 2 所示。

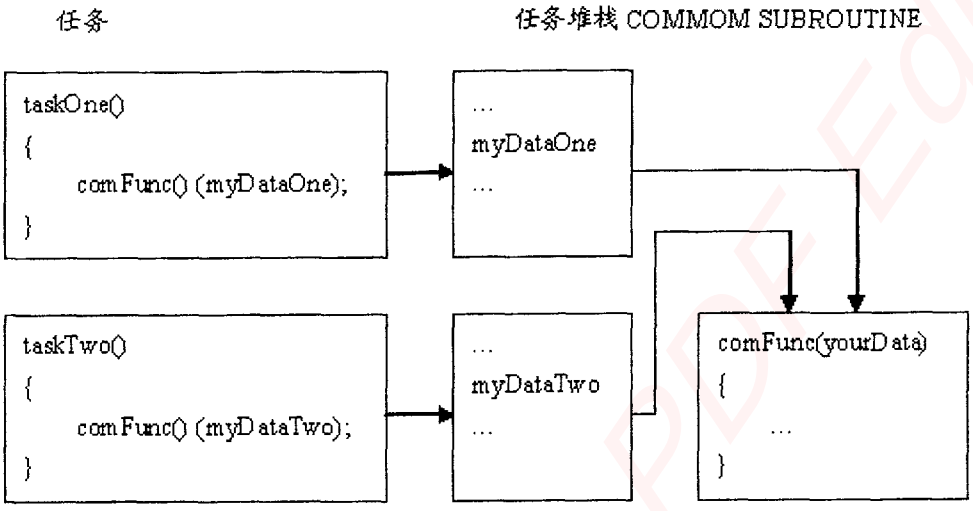

图 2 堆栈变量和共享代码

(2) 受保护的全局和静态变量

一些函数库包含公有数据，多个任务的同时调用很可能会导致对公有数据的破坏，使用起来要格外小心。系统采用信号量互斥机制来防止任务同时运行代码的临界区。

(3) 任务变量

一些公用函数要求对于每一调用程序都有明确的全局或静态变量值。为了满足这一点，VxWorks 提供的任务变量允许 4 字节变量加入到任务上下文中，当任务切换时变量的值也切换，如图 3。

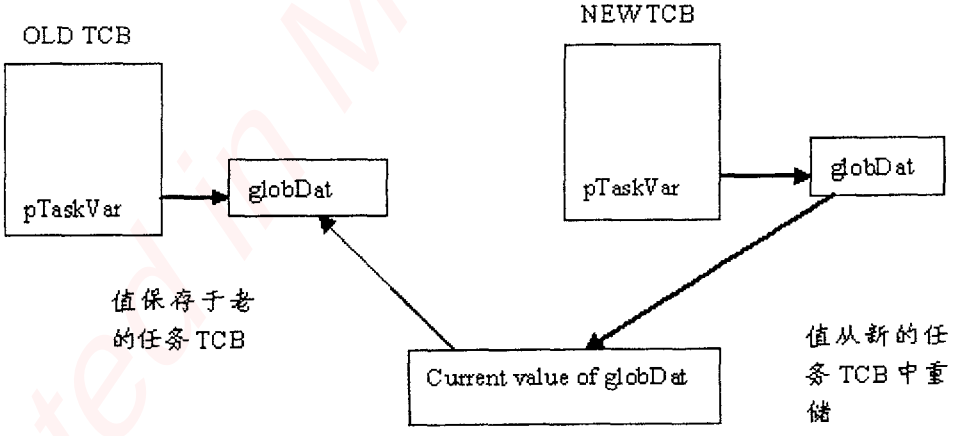

图 3 任务变量和上下文交换

编写可重入的函数，必须遵循这样的规则:一是将所有的局部变量申明为 auto(缺省态)或寄存器型。二是尽量不要使用 static 或 extern 变量。如有必要，要用互斥机制进行保护。

3.4 符号表的使用

如果符号表中的符号出现了重名，譬如两次下载的目标文件中有函数重名，则要作散列处理，之后对该函数的调用是最后加入符号表的函数，而之前已经装载的模块则不会受到影响。

3.5 任务优先级与CPU占用

VxWorks系统中优先级分为256级，从0到255，其中0为最高级优先级，255为最低优先级。任务的优先级在任务创建时被分配，但在任务运行时可通过系统调用taskPrioritySet()动态改变其优先级。当操作系统在目标板上启动成功后，系统级任务已在运行，对主机与目标机之间的通信进行管理，因此用户任务优先级要低于系统级任务，一般最高为100。同时，对于用户各任务优先级的确定，如何让各任务间良好地协调工作，要根据任务的紧急程度以及实际情况进行给定。在一个任务中用taskSpawn()函数创建一个同优先级或低优先级的任务，如果原任务一直占用CPU，新任务就不会开始运行。调用函数 taskDelay()可以使任务放弃CPU一定的时间，从而实现任务间时间上的同步；也可以放弃CPU零时间，将任务移至同优先级就绪队列的末尾，这样就可以实现多个同优先级的任务并发运行。有时为了执行效率等原因，任务的运行需要禁止基于优先级的抢占，这可以通过调用taskLock( )实现。如果任务1调用taskLock( )禁止了高优先级任务对它的抢占，当任务1被阻塞或被暂停，核心将调度下一个具有最高优先级的就绪任务运行。如果这时任务1又就绪且被调度运行，抢占又被禁止。但是，禁止基于优先级的抢占可以阻止任务切换，却并不会屏蔽中断。调用taskUnLock( )可以解除优先级抢占的禁止，通过调用taskLock( )和taskUnLock( )可以实现对临界资源的互斥访问。

3.6 任务删除安全

VxWorks提供的两个函数taskSafe( )和taskUnsafe( )将通知意外删除任务而引起的问题。当任务调用 taskSafe( )时，从调用的那一刻起，该任务就被保护起来而不会被其它任务删除。如果任务1试图删除已经调用taskSafe( )的任务2，则任务1将被阻塞，直到任务2调用taskUnsafe( )。保护只能由任务自己实现，一个任务不能safe或unsafe另外一个任务。taskSafe( )和taskUnsafe( )支持嵌套模式。如果有嵌套发生，一个计数器将开始工作，每有一个taskSafe( )被调用，则计数器加1；调用1个taskUnsafe( )，则计数器减1。只有当计数器为0时，才能删除该任务。

3.7 越界问题

造成越界的原因主要是在函数中定义了比较大的数组，以致进栈时越界。这样在编写程序时，就要求在堆栈许可的范围内定义数组。如果确实需要比较大的内存空间，可以使用操作系统的内存分配函数来获得内存。由于堆栈越界后有可能使任务的控制信息被破坏，使得对堆栈越界的检测比较困难，例如可以在栈底写入一串特殊字符，用另外一个任务或者中断服务程序经常来检查是否被改写来判断越界。VxWorks提供一个异常处理句柄（handler）和一个名为tExcTask的任务来处理异常。异常出现后任务成为挂起状态（suspend），并且不能转变为其它状态。在VxWorks中，有一个异常向量表来对应

各种异常，外部中断也作为一种特殊的异常。VxWorks 的做法是把多种异常的处理映射到同一个异常处理函数进行处理，并且 VxWorks 提供了向这个异常处理函数中钩挂用户的异常处理函数的接口 excHookAdd()，也可以将某一个异常向量映射到指定的处理函数。越界问题实际上是在程序编写过程中最频繁发生的,需要在编程时格外小心。

4 结论

多任务调度与运行是 VxWorks 操作系统的核心，它的好坏直接影响到系统的好坏，特别是在嵌入式实时系统中，本文中分析了 VxWorks 中多任务运行时可能的异常情况，研究了如何防止和处理这些异常，为更好的处理异常给出了一些参考。

参考文献

[1] 罗国庆. VxWorks 与嵌入式软件开发[M]. 北京:机械工业出版社. 2003. 127-140
[2] 孔祥营，柏桂枝. 嵌入式实时操作系统 VxWorks 及其开发环境 Tornado[M]. 北京:中国电力出版社. 2002. 91-95
[3] 王学龙. 嵌入式 VxWorks 系统开发与应用[M]. 北京:人民邮电出版社. 2002. 16-39
[4] 李方敏. VxWorks 高级程序设计[M]. 北京:清华大学出版社. 2004. 8-35

（上接第 33 页）

为了建立工程图的明细栏，从资源条上拖出明细栏落入到工程图标题栏的右下角，就可生成相应序号的零部件列表，表中的各项自动从部件内各零部件的属性中读取。明细栏的层次从菜单的格式中点出，一般选择“主模型+仅顶层”。明细栏的排序通过右击生成的明细栏，选取“名称、代号”等就可完成排序。零部件序号的生成方法为:“自动零件标号”图标→点生成的明细栏→选取视图后确定。

6 结束语

通过以上模板定制，我们建立了 UG NX4.0 从建模到工程图的基本工作环境，可以绘制出完整的工程图。但各种图、表的属性填写方式也为手工填写，深层次的应用还需作相应的二次开发才能实现。

参考文献

[1] 洪如瑾 编著. 《UG NX CAD 快速入门指导》. 北京：清华大学出版社. 2003 年 3 月
[2] 赵翠莲 翻译. 《UG 制图基础培训教程》. 北京：清华大学出版社. 2002 年 1 月