

嵌入式实时操作系统的资源调度

赵志强, 谢康林

(上海交通大学计算机科学与工程系, 上海 200030)

摘 要: 主要讨论了在嵌入式RTOS中采用的资源调度策略, 给出了一种由于任务共享系统资源而引起的死锁解决方法。这种方法可以大大减少死锁的发生。

关键词: 嵌入式实时操作系统; 任务; 任务控制块; 资源; 调度; 先进先出

Resource Scheduling on Embedded RTOS

ZHAO Zhiqiang, XIE Kanglin

(Dept. of Computer Science and Engineering, Shanghai Jiaotong University, Shanghai 200030)

【Abstract】 The paper mainly discusses the policy of resource scheduling used on the embedded real time operation system(RTOS). And it gives a solution of the deadlock due to the resource share amost tasks. The solution can greatly reduce deadlock.

【Key words】 Embedded RTOS; Task; TCB; Resource; Scheduling; FIFO

随着电子技术的飞速发展, 以微处理器(MPU)、微控制器(MCU)为核心的嵌入式系统(Embedded system)已经渗透到社会生活的各个方面, 尤其是在工业自动化、过程控制、航空航天等要求响应速度快, 精度高的应用场合得到越来越广泛的应用。同时, 嵌入式实时操作系统的作用越来越重要。作为嵌入式实时系统的一个重要组成部分, 对外界事件响应速度和对精确事件的处理能力是对RTOS最重要的要求。这要求RTOS内核管理系统资源简洁合理, 采用高效的任务调度和资源管理策略。作者开发的RTOS系统内核, 称为Kitty RTOS, 下面描述其采用的资源管理算法。

1 资源管理

系统中的各种设备、堆栈内存、共享内存、临界区、系统链表等, 都是由不同的任务共享的, 对这些资源的合理管理和调度要优先满足高优先权任务的分配请求、防止资源分配引起的死锁、分配策略要简单高效等几个方面的要求。

在Kitty RTOS中任务使用资源时, 采用独占方式。考虑到任务对资源操作和访问的不可预测性, 比如共享内存块指针的修改、系统链表的修改、外设的初始化等。如果这些操作正在执行时被中断或抢占, 就可能造成系统崩溃, 所以要求资源的使用是互斥和不可剥夺的, 即当某个任务线程获得系统资源使用权后, 除非主动释放, 否则不能被其它的任务剥夺。由此在Kitty RTOS中采用如下的资源分配原则和策略:

- (1) 如果所请求的资源都空闲, 则任务分配成功返回;
- (2) 如果要求分配的资源正在被使用, 则对这个任务所申请的资源不予分配。任务或者分配失败返回(由请求的类型决定), 或者被阻塞, 等待资源被释放并向该资源登记一项分配请求;
- (3) 一个任务在释放所占有的资源时, 先检查资源的申请请求, 如果有正在等待该资源的任务, 则选择优先级最高的任务并将该资源分配之, 否则释放资源, 然后进行死锁检查和处理;
- (4) 正在使用的资源不能被抢夺分配;
- (5) 资源分配按序进行, 遵守优先权规则和FIFO规则。

要实现上述的资源分配策略, 可以采用屏蔽CPU中断和资源加锁机制。其中前一种方式实现中有一定的局限性。由于RTOS实时性的要求, 只适用于RTOS内核中简单高速的原子操作, 这些原子操作对执行时间有着严格的要求, 必须

在系统要求的时间内执行结束, 因此对大部分的资源访问控制要通过后一种方式, 即加锁机制来实现。在本文中, 把对某一资源加锁称为锁定资源。

由于对系统中各种资源的访问方式各不相同, 如果对每一种资源都采用不同的加锁方式, Kitty RTOS代码的长度将大大增加, 同时执行效率也会大为下降。考虑系统中所有的资源, 包括各种设备, 都在内存中有相应的映像, 因此在Kitty RTOS中, 资源加锁时采用资源的内存映像来标识一个资源, 在系统中使用如下的数据结构作为资源锁定用到的数据结构, 称为资源锁结构:

```
struct resource_lock_struct
{
    void * resource; //任意资源在内存的映像指针;
    TCB * next_rsc_lock; //等待某资源的TCB队列;
    long owner_id; //当前占有资源的任务线程的ID;
    long ldata; //扩展信息, 保留。
};
```

RTOS内核管理一个上述结构的数组resource_table, 数组中的每一个元素称为资源锁。系统初始化后, 系统中没有锁定的资源, 所有的资源锁都空闲, 数组内每一个元素的所有成员都被初始化为NULL(如果为指针类型)或0(数值类型)。同时在Kitty RTOS内核还实现两个操作resource_lock和resource_unlock:

(1) resource_lock的原型为 void *resource_lock(void *rsc, int wait); 功能是锁定一个资源, 参数rsc是一个指针, 标识要加锁的资源。wait是等待标志, 如果等于0, 则资源忙时操作返回NULL, 指示锁定资源失败; 如果非0, 则分配资源的任务线程被阻塞直到请求锁定的资源被释放后再重新被唤醒;

(2) resource_unlock的原型为 void resource_unlock(void *rsc); 功能是释放一个被锁定的资源, 参数rsc的含义同上。

资源的加锁: 加锁时, resource_lock首先将当前任务插入阻塞队列。如果等待失败, 则返回NULL指示资源分配失败; 否则, 遍历resource_table, 在resource_table内的非空闲

作者简介: 赵志强(1973~), 男, 硕士生, 主研方向为实时嵌入式操作系统、模糊逻辑控制等; 谢康林, 教授

收稿日期: 2002-01-04 修回日期: 2002-03-18

资源锁的resource成员内查找src, 结果有两种可能的情况:

(1)如果找到, 假若为锁r_lock, 则说明rsc标识的资源已经被锁定分配占用, 如果参数wait=0, 则返回NULL表示资源分配失败; 如果参数wait=1, 判断r_lock的owner_id是否与当前任务线程的ID相等。如果相等, 则rsc已经被当前任务锁定, 清除CPU互斥标志, 开中断, 最后结束操作成功返回; 否则再查找r_lock中next_rsc_lock队列中TCB的ID是否有和owner_id相等的节点, 如果没有, 则根据优先权规则和FIFO规则将当前任务TCB插入r_lock的next_rsc_lock的队列中; 反之, 则空操作, 将当前任务线程改为阻塞状态, 执行suspend操作, 直到rsc被释放后当前任务再被唤醒;

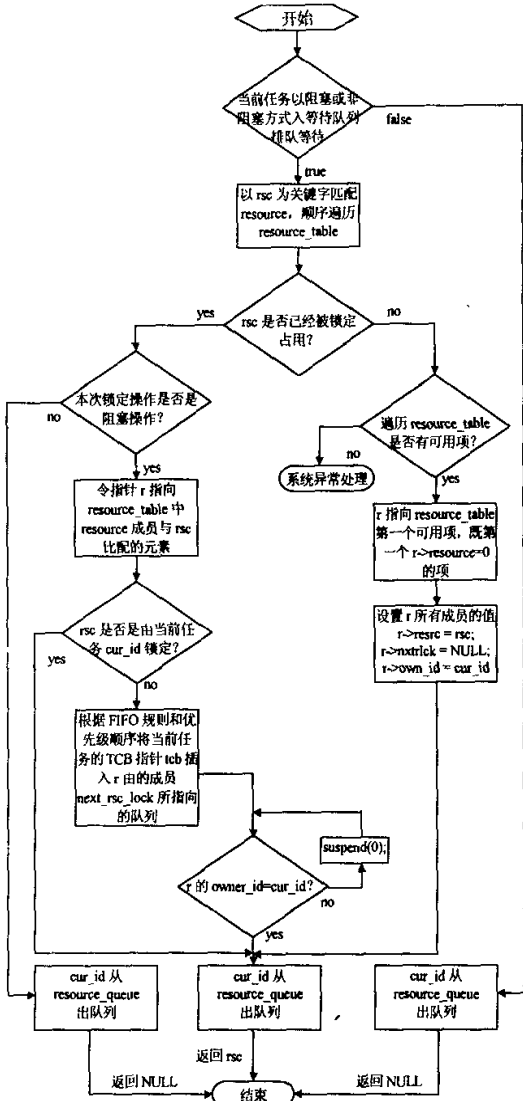


图1 resource_lock流程图

(2)如果没有找到, 则rsc标识的资源空闲, 在resource_table中选择一个空闲的锁r_lock, 并赋值使resource等于rsc, next_rsc_lock等于NULL, owner_id等于当前任务线程的ID, ldata则根据具体情况赋值, 清除CPU互斥标志, 开中断, 最后结束操作返回。

图1是resource_lock的程序流程图, 变量r是resource_lock_struct类型的指针, 变量cur_id当前运行任务线程ID, 操作

suspend(0)的功能是阻塞当前正在运行的任务。

资源的释放: 释放已经锁定的资源通过resource_unlock, 假设由resource_unlock传入的参数rsc标识一个已经由resource_lock锁定的资源指针, resource_unlock先使当前任务一阻塞方式排队等待, 获得处理权后以rsc和当前任务的ID为关键字遍历resource_table, 查找匹配的锁元素。如果匹配成功, 检测匹配成功的锁元素r的next_rsc_lock成员是否为NULL: 如果非空, 则next_rsc_lock标识的是下一个要求使用rsc的任务的tcb, 更新next_rsc_lock的值, 使next_rsc_lock指向链表的下一个节点并唤醒next_rsc_lock原来所标识的任务; 反之, 如果next_rsc_lock为空, 即其值等于0, 则将r所有的成员都清零释放r, 最后将当前任务出队列后从操作返回, 否则, 则转入系统异常处理。resource_unlock操作的流程图如图2所示。

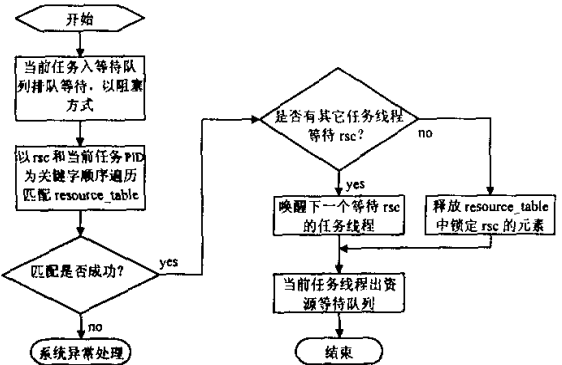
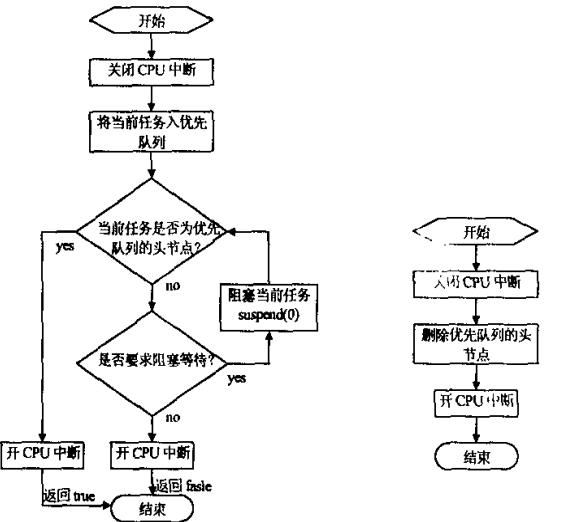


图2 resource_unlock流程图



(a) Resource_wait 流程图

(b) Out_resource_queue 流程图

图3 Resource_wait和Out_resource_queue子操作流程图

资源相关的死锁处理: 因为在Kitty内核中, 资源的分配是动态进行的, 可能发生死锁。为了降低发生死锁的概率, 不同任务的资源的锁定和释放操作要求同步, 按照FIFO规则和优先级高者优先的规则顺序严格进行。在

(下转第281页)

```

Do Until rs.EOF '循环查找
    If LCase(Trim(rs("dbname") & "")) = LCase(Trim
(dbname)) Then '强制转化为小写来比较
        intPid = CInt(rs("spid"))
        oSQLServer.KillProcess intPid '杀死连接进程
    End If
    rs.MoveNext
Loop
rs.Close
End If
oSQLServer.Disconnect '断开连接

```

负责数据库备份与恢复的模块在Bus_Backup组件中的名字为Db_backup, 它的结构图如图4。其中Bk_backup()是对数据库进行新的备份, Bk_Restore()对指定的数据库备份文件进行恢复, Bk_Del()删除指定的数据库备份文件, Bk_GetAll()可以获得当前所有的数据库备份文件列表, 此外还有一些可以设置和获取系统等参数的函数未列出。

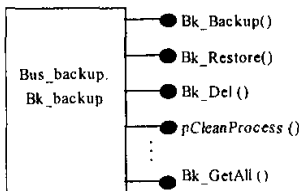


图4 Bk_backup结构

同样在表现层的ASP文件也提供了供管理员管理的接口, 具有一定权限的管理员可以进行数据库备份、恢复、删除, 也可以下载指定的数据库备份文件到本地。

2.4 安全性的考虑

☆☆

(上接第274页)

resource_lock和resource_unlock的流程图中, 开始的子操作“入等待队列”和“当前任务出队列”就是处理任务间资源锁定和释放操作的同步问题的, 这里将其命名为Resource_wait和Out_resource_queue子操作, 对这两个子操作采用优先队列对不同申请资源的任务进行管理, 图3是在resource_lock和resource_unlock中使用的Resource_wait和Out_resource_queue子操作的程序流程图。

2 应用

上述的算法经实践证明是可行有效的, 下面是在Kitty RTOS中的几个具体加锁例子示意。Ansi的标准内存分配函数为void *malloc(size_t size)在执行中必须对堆栈加锁:

```

void *malloc(size_t size){
    resource_lock(heap_point,wait);
    pmem =kernel_malloc(size); //allocate memory
    resource_unlock(heap_point,wait);
    return p_mem;
}

```

设备打开和关闭例程:

```

int open(DEV dev, MODE mode){
    if(mode==READ)
        resource_lock(&dev->write,TRUE);
    if(mode==WRITE)
        resource_lock(&dev->read,TRUE);
    return dev->open();
}

void close(DEV dev, MODE mode){

```

由于网站备份与恢复功能的重要性, 需要对这部分的安全性多加考虑。首先实现备份与恢复管理的ASP文件物理目录不应该放在原来的站点主目录下, 有条件的可以放在另外的可靠服务器上; 另外由于提供了备份文件的下载功能, 一定要注意备份文件不会被没有权限的用户下载, 可以在ASP中对用户存取进行控制, 更好的办法是在IIS中设置对存放备份的目录进行访问控制, 通过这些措施来保证备份文件的安全下载。

3 总结

Windows DNA架构由于拥有许多良好的特性而得到了很多网站广泛的应用, 同时由于面临着许多威胁, 网站的备份与恢复也得到了很高的重视, 本文就介绍了这样一种基于Windows DNA架构的远程网站备份与恢复技术, 它的优点是可以供管理员方便地, 通过浏览器就可以远程进行备份与恢复, 而且系统结构清晰、实现简单, 并且具有良好的可扩展性和安全性。该技术在我们的实践中多次运用, 取得了很好的效果。

参考文献

- 1 微软公司.Windows DNA技术白皮书.http://www.Microsoft/china/dna
- 2 Sundblad S.前导工作室译.Windows DNA 可扩展设计.北京:机械工业出版社, 2001
- 3 Pattison T.王新昌译.COM+与Visual Basic 6分布式应用程序设计(第2版).北京:机械工业出版社, 2001
- 4 陈文博, 夏长虹.DNA、组件对象模型与商务逻辑计算.计算机世界, 1999-09-20
- 5 Anderson R.刘福太译.ASP 3 高级编程.北京:机械工业出版社, 2000

```

dev->close();
if(mode==READ)
    resource_unlock(&dev->write,TRUE);
if(mode==WRITE)
    resource_unlock(&dev->read,TRUE);
}

```

注: suspend(0)的功能是阻塞当前任务

3 讨论

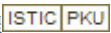
上述的处理方法在一些通信设备和RTU中的实际应用中取得了很好的效果, 当然, 这种方法不能完全避免资源等待引起的死锁问题, 但是可以大大降低这种可能性。之所以不采用防止死锁的发生的算法, 是因为死锁检查的时间复杂度太高, 且除了资源引起外, 死锁还和任务同步、阻塞调用等引起的等待关系有关。因此, 在Kitty RTOS中, 有专门的系统检测和调度任务处理, 这里不予讨论。

其次, 和提供给用户的共享锁和互斥锁的应用程序API接口不同, 这种加锁方式只是RTOS内核用来在内核内部对资源管理的手段, 不直接作为应用程序API和系统服务例程提供给应用程序调用。而系统的内核设计对所有调用资源分配和释放操作的地方, 都根据资源类型的不同做了仔细的局部处理, 考虑了可能引起的死锁问题。

参考文献

- 1 RT-Linux 文献和源代码.http://ftp.rtlinux.com
- 2 Linux 内核文献和源代码.http://www.kernel.org
- 3 POSIX.4 标准文献.http://www.ieee.org
- 4 Embedded System Documents.M-system Co.

嵌入式实时操作系统的资源调度

作者: 赵志强, 谢康林
作者单位: 上海交通大学计算机科学与工程系, 上海, 200030
刊名: 计算机工程 
英文刊名: COMPUTER ENGINEER
年, 卷(期): 2003, 29 (2)
被引用次数: 13次

参考文献(4条)

1. [Embedded System Documents.M-system Co](#)
2. [POSIX.4标准文献](#)
3. [Linux内核文献和源代码](#)
4. [RT-Linux文献和源代码](#)

引证文献(13条)

1. 李天全 [IPTV机顶盒嵌入式系统研究](#)[期刊论文]-[中国高新技术企业](#) 2009 (11)
2. 赖奕霖 [嵌入式系统测试中关于双机通讯的研究](#)[期刊论文]-[福建电脑](#) 2007 (12)
3. 赖俊宇, 李双庆 [μC/OS-II与RTLinux的实时原理及其性能改进分析](#)[期刊论文]-[电脑知识与技术\(学术交流\)](#) 2007 (3)
4. 江少明 [脉冲MIG焊机送丝驱动控制的数字化研究与实现](#)[学位论文]硕士 2006
5. 梁雨婷 [基于MCF5235的eCos的移植、改进以及在VoIP终端中的应用研究](#)[学位论文]硕士 2006
6. 黄鹏 [基于μC/OS-II的车载实时操作系统研究](#)[学位论文]硕士 2006
7. 蒋文杰 [嵌入式实时操作系统TLinux的实现与改进](#)[学位论文]硕士 2006
8. 周晓中 [基于单片机的实时内核设计及其应用研究](#)[学位论文]硕士 2005
9. 朱自民 [船舶网络信息监视系统的设计与实现](#)[学位论文]硕士 2005
10. 吕华洋 [嵌入式实时操作系统在磁力轴承控制中的应用研究](#)[学位论文]硕士 2005
11. 杨静 [基于DSP的嵌入式实时操作系统平台及其应用](#)[学位论文]硕士 2005
12. 洪启峰 [基于DSP的实时操作系统研究](#)[学位论文]硕士 2005
13. 王红红 [实时嵌入式操作系统在电力系统自动监控及远程抄表中的应用](#)[学位论文]硕士 2004

本文链接: http://d.g.wanfangdata.com.cn/Periodical_jsjgc200302108.aspx