第 31 卷第 10 期
2011 年 10 月

计算机应用
Journal of Computer Applications

Vol. 31 No. 10
Oct. 2011

文章编号: 1001 - 9081(2011)10 - 2606 - 03

doi: 10.3724/SP. J. 1087. 2011. 02606

μC/OS-Ⅱ内核任务调度模块的扩展

尹作为[1],郭　兵[1],沈　艳[2]

(1. 四川大学 计算机学院,成都 610064;　2. 成都信息工程学院 控制工程学院,成都 610225)

(aaasjm@126.com)

摘　要: 针对 μC/OS-Ⅱ不能支持同级任务调度的限制,提出一种基于时间片轮询调度的策略。该策略借用 μC/OS-Ⅱ内核中的两个优先级任务,充当时钟源和轮询引擎,让同级任务在最低优先级任务下轮流运行。在不失实时性的前提下,让内核支持多达 192 个同级任务。实验及对比表明,该方案简单实用。

关键词: μC/OS-Ⅱ; 同级任务; 时间片轮询调度; 实时; 模块扩展

中图分类号: TP301.6; TP316　　**文献标志码:** A

Expansion of task scheduling module in μC/OS-Ⅱ kernel

YIN Zuo-wei[1], GUO Bing[1], SHEN Yan[2]

(1. *College of Computer Science, Sichuan University, Chengdu Sichuan* 610064, *China*;
2. *School of Control Engineering, Chengdu University of Information Technology, Chengdu Sichuan* 610225, *China*)

Abstract: Since μC/OS-Ⅱ cannot support the restriction on task scheduling of the same priority, a strategy was proposed based on round-robin scheduling. It borrowed two priority tasks in the kernel as the clock source and the polling engine respectively, so that tasks with the same priority worked under the lowest priority in turns. The method allows the kernel to support up to 192 tasks in the same priority without losing the real-time quality. The experiment and comparison indicate that the proposed method is simple and practical.

Key words: μC/OS-Ⅱ; same priority task; round-robin scheduling; real-time; module expansion

0　引言

物联网时代来临,嵌入式系统用途愈加广泛。嵌入式系统一个极为核心的技术就是嵌入式操作系统。目前常用的嵌入式操作系统有 μClinux、μC/OS-Ⅱ、Windows CE、VxWorks 等。其中 μC/OS-Ⅱ以其实时性强、源码开放、高效而小巧等优点得到了比较广泛的应用。

μC/OS-Ⅱ是一个具有基本功能的操作系统内核,包括基于优先级的可剥夺式任务调度和简单的内存管理等。μC/OS-Ⅱ内核支持 64 种不同优先级的任务,不支持同级任务,不支持时间片轮询任务调度[1]。这样的系统完全为硬实时环境而设计,在某些实际应用中局限了系统的灵活性和应用范围,例如多点的温度或气压数据采集,这些任务往往需要同优先级调度。

给 μC/OS-Ⅱ添加同级任务调度,内核研究者提出了各种不同的方法。文献[2]提出一种将 8 个优先级任务转换为相同优先级任务,并基于时间片轮询调度的方法;文献[3]提出一种将任务调度在优先级抢占与时间片轮询之间轮流选用的方法;文献[4]提出一种将单一优先级任务扩展为优先级任务链表,从而达到添加同级任务的方法。上述方法从不同的技术角度为 μC/OS-Ⅱ添加了同级任务调度,尽管有的方法从理论上讲符合逻辑,但是实际操作较为复杂,方案实用性较差,实时性难以保证。在不失实时性,同时尽可能少改动原有内核架构的前提下,本文提出一种占用 2 个优先级任务,为系统扩展出最多 192 个同级任务基于时间片轮询调度的方案。

1　思路与模型

在添加基于时间片轮询的同级任务调度时,模仿了 μC/OS-Ⅱ内核原有数据结构。

1.1　总体思路

设计的总体思路是:模仿优先级任务链表,创建一个同级任务链表。创建两个优先级任务 A 和 B,且 B = A + 1。A 以时间片为单位周期性地进入等待或运行,从而周期性地中断 B。B 每次运行总是从同级任务链表取一个处于就绪态的同级任务,然后跳入同级任务执行。当没有比 A 更高优先级的任务时,同级任务以时间片为单位轮流在 B 下执行。B 每次中断,总是将断点保存到当前运行的同级任务堆栈。B 下次被内核调度运行,不会返回同级任务,而是重新进入 B 的任务,开始新一轮的轮询。

1.2　时钟源

时间片轮询调度算法,首先面临的问题是时钟来源。可以利用原有系统的功能,模拟时钟源。创建一个优先级任务 A(下面用 A 标识此任务),利用系统延时函数 OSTimeDly(),周期性地让 A 进入等待状态。创建一个优先级任务 B(下面用 B 标识此任务,B = A + 1),B 的作用是轮询调度同级任务。每当 A 进入等待状态,如果没有比 A 更高优先级的任务,内核会调度 B 运行。一旦 A 等待结束进入就绪态,内核会中断 B,由此 B 周期性地运行或中断,则自发为 B 提供了时钟源。由于 A 和 B 是内核中的两个优先级任务,它们服从内核调度,所以不会破坏内核的实时性。

收稿日期: 2011 - 04 - 02; 修回日期: 2011 - 06 - 01。　基金项目: 国家 863 计划项目(2008AA01Z105)。

作者简介: 尹作为(1986 -),男,湖北仙桃人,硕士研究生,主要研究方向: 嵌入式实时操作系统、SoC;　郭兵(1970 -),男,山东泰安人,教授,博士生导师,博士,CCF 高级会员,主要研究方向: 嵌入式实时系统、SoC、中间件;　沈艳(1973 -),女,湖南永州人,副教授,博士,主要研究方向: 分布式测量系统、嵌入式系统、无线传感器网络、机器人。

1.3　构造同级任务

μC/OS-Ⅱ 以一个字节存储任务优先级，使用 64 种不同的优先级状态。一个字节可以表达 256 种状态，剩余的 192 种状态可用来标识同级任务。同级任务的状态标识从 64 开始，255 为最大。192 个同级任务仍然用优先级数字标识，但是这里的优先级数字不再代表同级任务的优先级别，仅仅作为区分同级任务的编号。

同级任务的数据结构模仿优先级任务。仿照优先级任务链表创建同级任务链表，同级任务的优先级编号从 64 开始。在调用 OSTaskCreate( ) 等函数创建任务时，如果任务优先级是介于 64 和 255 之间，说明任务是同级任务，需要添加到同级任务链表中。UserTbl ‖ 数组表示同级任务控制块数组。每当创建新的同级任务时，总是将空的任务链表头 UserFreeList 指向的任务控制块分配给该任务。在给任务控制块中的各成员赋值后，就按任务控制块链表的头指针 UserList 将其加入到任务控制块链表中[5]。UserCur 指向链表中即将调度的同级任务。具体数据结构如图 1 所示。

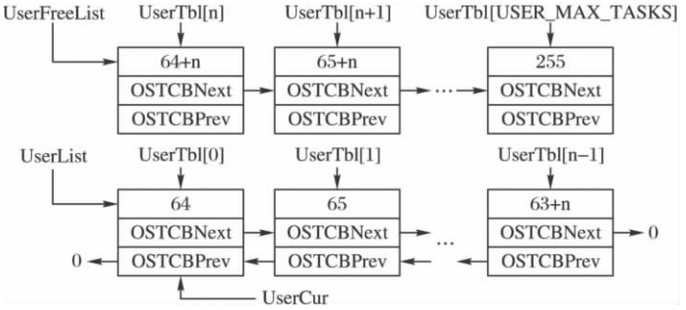

图 1　同级任务控制块数据结构

1.4　轮询引擎

1.2 节中已经说明，“同级任务轮询引擎”是系统的一个优先级任务，用 B 标识。当 B 第一次进入任务代码时，会将当前运行位置保存到一个全局变量（在 C 语言中可通过 setjmp( ) 来完成），便于在其他地方可以跳回此处。此后，B 取得同级任务链表中 UserCur 指向的任务控制块，检查 UserCur 指向的任务是否处于就绪状态，是则跳入此同级任务去运行，否则调整 UserCur 指向下一个控制块。

UserCur 指向的同级任务运行一个时间片后，A（提供时钟源的任务，见 1.2 节）从等待态进入就绪态，进而中断 B。B 立即将当前的运行断点保存到 UserCur 指向的同级任务堆栈中。当下一次 B 调度运行时，B 会跳回第一次运行时设置的全局变量断点处（在 C 语言中可通过 longjmp( ) 来完成），开始新一轮的轮询。

A 不仅提供了 B 的时钟源，还负责更新 UserCur。每次 A 从等待态进入运行态时，将 UserCur 重新指向下一个节点，由此使得 B 每次轮询时都是从新的同级任务开始。这里有必要设置一个全局变量 UserCount，表示同级任务链表中处于就绪态任务的个数，如果其值为 0，A 不需要更新 UserCur，B 也会进入等待状态，从而不浪费系统资源。轮询引擎运行过程如图 2 所示。

2　同级任务调度的实现

所用源代码是 μC/OS-Ⅱ 2.52，以下所有函数及变量均基于此。

2.1　时钟节拍

内核中每发生一次时钟中断，均会调用 OSTimeTick( )。OSTimeTick( ) 负责更新任务的延时节拍，将延时结束的任务更新到就绪表，添加对同级任务的处理伪代码如下:

```
ptcb = OSTCBList;
while (ptcb->OSTCBPrio != OS_IDLE_PRIO) {
//原系统代码不做任何修改 }
//添加对同级任务的处理
ptcb = UserList;
while(ptcb != 0){
  OS_ENTER_CRITICAL();
  if (ptcb->OSTCBDly != 0) {
    if ( --ptcb->OSTCBDly == 0) {
      if ((ptcb->OSTCBStat & OS_STAT_SUSPEND) == OS_
        STAT_RDY) {
        更新 B 到就绪表;
        //有同级任务就绪,将 B 添加到就绪表
        UserCount++;                    //同级任务就绪个数更新
        }else{ ptcb->OSTCBDly = 1; }
  }}
  ptcb = ptcb->OSTCBNext;
  OS_EXIT_CRITICAL();
}
```

内核中并没有同级任务的就绪表，当有同级任务就绪时，则使轮询引擎 B 进入就绪态。B 一旦获得执行时间，同级任务则按时间片轮询规则得到运行。

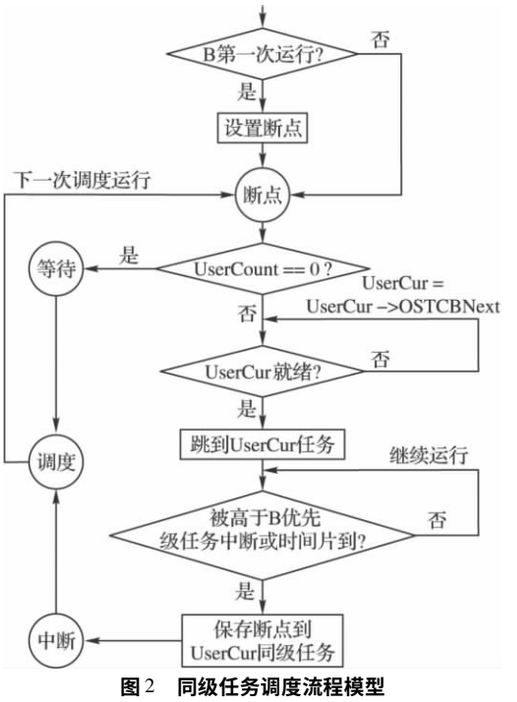

图 2　同级任务调度流程模型

2.2　任务延时

内核中的任务空闲时，需调用延时函数进入等待状态，以释放系统资源。常用的延时函数是 OSTimeDly( )，其他延时函数也是借用此函数完成[7]。同级任务是借用轮询引擎 B 得以执行，当前执行的同级任务进入等待态后，不能简单地让 B 也进入等待，而是进行一次调度，让其他处于就绪态的同级任务有机会继续在 B 下运行。OSTimeDly( ) 添加对同级任务的处理伪代码如下:

```
if (ticks > 0) {
  OS_ENTER_CRITICAL();
  if (OSTCBCur->OSTCBPrio == B){        //如果是轮询引擎 B
    UserCur->OSTCBDly = ticks;
                                  // UserCur 指向的同级任务延时
    if ( --UserCount > 0){                //还有其他同级任务
      OS_EXIT_CRITICAL();
      OS_Sched();                                 //主动调度一次
```

```
            return;
        }}
        //后面是原系统代码,不做任何修改
    }
```

当没有同级任务处于就绪态时,B 会调用原来代码,进入等待状态,不会浪费系统资源。

2.3 调度规则

为了完成同级任务基于时间片轮询调度,需对系统中有关中断保存和调度规则进行修改。μC/OS-Ⅱ 2.52 源码中,OS_Sched( ) 与 OSCtxSw( ) 与任务调度相关,OSTickISR( )、OSIntCtxSw( )、OSIntExit( ) 与中断相关。完成任务切换或中断跳转的函数是 OSCtxSw( ) 和 OSIntCtxSw( )。

OS_Sched( ) 与 OSIntExit( ) 作用类似,不同的是后者调用之前断点已经自动保存过。轮询引擎 B 每次中断需要将断点保存到 UserCur 指向的同级任务堆栈,下次调度运行不是返回上一个同级任务,而是返回第一次设置的全局变量断点处,开始新一轮的轮询。由于每时每刻在 B 下运行的是一个同级任务,故当任务切换与 B 有关时,需主动将断点保存到当前在 B 下运行的同级任务堆栈,修改 OS_Sched( ) 伪代码为:

```
if ( OSPrioHighRdy ! = OSPrioCur) {
    //原系统代码不加修改}
    if ( OSPrioHighRdy == B) {                    //最高就绪任务是 B
        OS_TASK_SW( );
        //主动切换任务, OS_TASK_SW( ) 内部会保存同级任务断点
} //原系统代码不加修改
```

对于 OSIntExit( ) 则不需修改,因为此函数调用之前断点已经自动保存过。

OSTickISR( )、OSCtxSw( )、OSIntCtxSw( ) 3 个函数位于 Os_cpu_a. asm 中,是用汇编语言编写,需要添加相应的判断和跳转代码。对于判断两个任务的优先级是否一样,需要比较任务控制块中的 OSTCBPrio 变量。由于此变量在 OS_TCB 结构的尾部,不便于寻址,可将它改到 OS_TCB 结构中的第二个位置,即位于 OSTCBStkPtr 变量之后。(OSTCBStkPtr 也要方便寻址,故其在第一[1])。这 3 个函数,在原有中断保存和任务切换代码上,需要服从以下逻辑规则:

当被中断的任务是轮询引擎 B 时,需将断点保存到 UserCur 指向的同级任务堆栈中,而不是任务 B 自身的堆栈,添加汇编伪代码如下:

```
LES     BX,DWORD PTR DS: _OSTCBCur;
        取当前运行的优先级任务
CMP     BYTE PTR ES: [BX +4], #B;
        判断当前运行的优先级任务是否为 B
JNE     _OLD;  不是 B,跳到原有代码
MOV     AX, SEG( _UserCur);  取 UserCur 指向的同级任务
MOV     DS,AX;
LES     BX,DWORD PTR DS: _ UserCur;
MOV     ES: [BX +2], SS;  保存断点到 UserCur 指向的同级任务
MOV     ES: [BX +0], SP;
JMP     _NEW;  保存断点后,跳过原有保存断点语句
```

当即将调度运行的优先级任务是 B 时,需要返回到第一次 B 运行时设置的全局变量断点处,而不是上一个同级任务,添加汇编伪代码如下:

```
LES     BX,DWORD PTR DS: _OSTCBHighRdy;
        取就绪的最高优先级任务
CMP     BYTE PTR ES: [BX +4],#B;
        判断就绪的最高优先级任务是否为 B
JNE     _OLD;  不是 B,跳到原有代码
CALL FAR PTR    _OSLONGJMP;
                是 B,返回第一次设置的全局变量断点处
```

3 实验

为了检验改造后的 μC/OS-Ⅱ,需要把 μC/OS-Ⅱ 移植到合适的处理器上运行。这里选择 Intel 的 x86 系列 CPU。硬件平台包括 Intel Pentium Dual-Core CPU 2.30 GHz,2 GB 内存。操作系统平台采用 Windows XP,编译器采用 Borland C++ V4.5,源码为 μC/OS-Ⅱ 2.52。移植代码针对 x86 的实模式,使用 BC4.5 在大模式下编译和链接。

让 A 的优先级为 58,B 的优先级为 59,如此一来,除去系统保留优先级外,同级任务轮流运行在最低优先级下,其他优先级任务都可以中断 B 的运行,从而不干扰任何优先级任务的实时性。实验中除 A、B 外,另创建了 5 个优先级任务,同时创建了优先级编号从 64 到 73 共 10 个同级任务,任务堆栈 TASK_STK_SIZE 均设为 512,时间片单位为 100 ms。任务的工作是调用 PC_DispChar( ) 在屏幕不同行上打印出不同的字符,优先级任务分别打印 0 ~ 4,同级任务分别打印 A ~ G。经测试,系统工作状态良好,结果如图 3 所示。

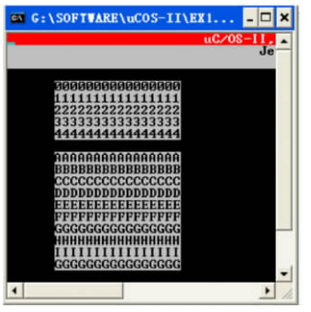

图 3 实验运行结果

4 方案对比

给 μC/OS-Ⅱ 添加同级任务调度,内核研究者提出过各种不同的方法。下面将其他 3 种典型思路与本文思路进行对比,分别分析方案的优缺点。

1) 文献 [2] 方案: 将 64 个任务分成两组,其中一组保持不变,仍为具备不同优先级的任务; 另外一组则将任务的优先级视为同一优先级,其中的任务为同级任务,按相应的进程调度算法进行调度。

优点 实现简单,代码改动量少,不失实时性,支持少量同级任务。

缺点 系统支持的任务总数没有得到扩展,虽然支持了少量同级任务,但同时减少了相应不同优先级任务数量,未能明显扩展系统功能,方案适用的范围有限,实用性差。

2) 文献 [3] 方案: 创建一个同级任务链表,当优先级任务链表空闲时,调度切换到同级任务链表,同级任务运行完各自的时间片后,调度再切换到优先级链表。

优点 可支持大量的同级任务,较大地扩展了内核功能。

缺点 首先是失去了实时性,一旦进入同级任务链表,必须等到链表中所有同级任务运行完各自的时间片后,才能再次进入优先级任务链表,优先级任务的运行得不到实时响应; 其次是系统改动较大,实现较为复杂,实时性的缺失限制了适用的范围。

3) 文献 [4] 方案: 优先级任务扩展为优先级任务链表,每个优先级链表里面包含了多个相同优先级的任务。

优点 可支持大量的同级任务,极大地扩展了内核功能。

缺点 实现复杂,内核改动量大。从逻辑上讲可行,但实际操作中可行性和正确性难以保证。 (下转第 2616 页)

水平。

3.5 仿真结论

通过以上实验，说明了基于文件分割的存储概率模型是有效的。通过仿真实验，找出实现 BLOB 存储的最佳分块为 3 MB ~ 5 MB。用户上载行为和下载行为的仿真实验显示，随着被传文件逐渐增大，上载文件的速率缓缓下降，且 BLOB 数据上传的效率低于下载的效率，这一问题可采用将媒体文件分块存储在若干台服务器上的方法解决[16]。总之，媒体文件的分块存储方式在一定程度上改善了媒体数据的存储质量，降低了存储失效的概率。

4 结语

媒体数据呈爆炸式增长趋势，数据存储和管理面临严峻考验，应用数据库存储 BLOB 数据为媒体数据管理提供了新思路。本文研究了 RDBMS 中 BLOB 数据存储原理，建立了基于文件分割的 BLOB 存储可靠性概率模型，并针对该模型设计了新的存储结构和算法，在对算法的仿真实验中，找出了较优分块大小，并对各种用户行为进行了仿真。该算法的优势在于极大改善了存储稳定性、可靠性和数据一致性，把媒体存储失效变成了小概率事件。

参考文献:

[1] 王文丰，赵跃龙，曾文英，等. 一种网络存储技术新方案: 智能网络磁盘集群存储系统[J]. 小型微型计算机系统，2008，29(7)：1211 - 1214.

[2] 万继光，詹玲. 集群多媒体存储系统的两级元数据管理[J]. 小型微型计算机系统，2009，30(4)：752 - 756.

[3] 陈仁章，孟小华. 大型网络教学平台架构设计及实现[J]. 计算机工程与设计，2010，31(11)：2455 - 2469.

[4] NICOLAE B, ANTONIU G, BOUGE L. BlobSeer: How to enable efficient versioning for large object storage under heavy access concurrency [C]// Proceedings of 2009 EDBT/ICDT Workshops. New York: ACM Press, 2009: 18 - 25.

[5] CHEN TAO, KHAN A, SCHNEIDER M, *et al*. iBLOB: Complex object management in databases through intelligent binary large objects [C]// Proceedings of the 3rd International Conference on Objects and Databases. Berlin: Springer, 2010: 85 - 99.

[6] 苏峰，黄正军. GIS 空间数据管理模式探讨[J]. 计算机仿真，2003，20(8)：140 - 143.

[7] 文永革，彭声泽. 基于 Web 的非结构化数据管理方法的研究与实践[J]. 计算机系统应用，2008，17(5)：101 - 103.

[8] 纪兆辉，胡孔法. 基于 ADO. Net 2.0 实现数据库系统中 BLOB 数据的处理[J]. 淮海工学院学报: 自然科学版，2009，18(2)： 28 - 31.

[9] 杨宁，申强，谢静. SQL Server 数据库中图像存取技术研究[J]. 南京晓庄学院学报，2010(3)：82 - 84.

[10] 王姝. 数据库图像字段数据存储与读取技术应用研究[J]. 科学技术与工程，2010，10(3)：682 - 685.

[11] 赵金东，于沛. Web 应用中大文件传输问题的研究[J]. 烟台大学学报: 自然科学工程版，2007，20(2)：132 - 134.

[12] 陈卫卫，吴海佳，胥光辉. 分布式存储中文件分割的最优化模型[J]. 解放军理工大学学报: 自然科学版，2010，11(4)：413 - 416.

[13] 黄华毅，黄穗. 基于 ADO . NET的 SQL Server 数据库 image 对象的存取及其效率分析[J]. 现代电子技术，2003，26(6)：58 - 60.

[14] 宋国兵，陈奇. 文件数据的数据库 Blob 存储及效率分析[J]. 计算机工程与设计，2010，31(21)： 4625 - 4727.

[15] 李洁琼，冯丹. 一种面向下一代互联网的广域网智能存储系统[J]. 计算机科学，2010，37(10)：279 - 282.

[16] 胡涛，许胤龙. 分布存储 VOD 系统的负载均衡设计及其仿真[J]. 计算机仿真，2009，26(4)：186 - 189.

(上接第 2608 页)

4) 本文方案: 创建一个同级任务链表，当优先级任务链表空闲时，同级任务以时间片轮流运行在最低优先级任务之下，一旦有优先级任务就绪，可立即中断同级任务而得到响应。

优点　不失实时性，支持 192 个同级任务，较大地扩展了内核功能，简单实用。

缺点　占用了 2 个优先级任务。

从上面对比看出: 对比文献 [2] 方案，本文方案支持更为灵活的同级任务数量，占用的优先级任务较少; 在文献 [3] 方案的基础上，本文方案保证了内核的实时性，且系统改动量较文献 [3] 方案要少; 相对于文献 [4] 方案而言，本文方案简单实用，可行性更高，且系统改动量较文献 [4] 方案大大减少。从实用性角度而言，本文方案支持 62 个不同优先级任务，192 个同级任务，且不失实时性，扩展了系统的适用范围。当然本文方案也不是绝对完美，其中一个较为突出的限制在于: 所有的同级任务均在预先设定的优先级任务下进行调度，即它们均具备相同的优先级权限，没有进一步扩展到任意优先级。

5 结语

本文的出发点在于: 从简单实用的角度为 μC/OS-Ⅱ 扩展出基于时间片轮询调度的同级任务。创新点在于: 在尽量少改动内核架构及不失实时性的前提下，借用系统的两个优先级任务，充当时钟源和同级任务轮询引擎，将 μC/OS-Ⅱ 扩展出 192 个同级任务。实验及对比表明，本文方案简单实用。本文的工作仅对内核中最核心的任务调度模块进行了扩展，要做一个完善可靠、支持同级任务调度的 μC/OS-Ⅱ 系统，还需要对内核其他模块(如信号量、邮箱、调度算法优化等)进行修改，这将是下一步要进行的工作。

参考文献:

[1] LABROSSE J. 嵌入式实时操作系统 μC/OS-Ⅱ[M]. 2 版. 邵贝贝，译. 北京: 北京航空航天大学出版社，2003: 72 - 366.

[2] 成后发，杨春金. μC/OS-Ⅱ 操作系统内核的改进[J]. 通讯和计算机，2006，19(6)：52 - 55.

[3] 邹航，李小文. μC/OS-Ⅱ 内核任务调度算法的改进[J]. 重庆邮电大学学报: 自然科学版，2010，22(3)：360 - 363.

[4] 何海涛. μC/OS-Ⅱ 中优先级抢占的时间片调度算法的实现[J]. 计算机系统应用，2009，18(11)：73 - 75.

[5] 任哲. 嵌入式实时操作系统 μC/OS-Ⅱ 原理及应用[M]. 北京: 北京航空航天大学出版社，2005: 15 - 97.

[6] 姚燕南，薛钧义，姚向华，等. 微型计算机原理与接口技术[M]. 北京: 高等教育出版社，2004: 115 - 122.

[7] LABROSSE J. μC/OS-Ⅱ 2.52 内核源码[CP/OL]. [2010 - 12 - 18]. http: //www. micrium. com.

[8] 郭兵，沈艳，林永宏，等. SoC 技术原理与应用[M]. 北京: 清华大学出版社，2006.

[9] 肖建明. 一种改进的时间片轮询调度算法[J]. 计算机应用，2005，25(12)：447 - 448.