μC/OS—II任务调度算法的改进

熊玉梅[1] 陈一民[2]
[1](上海电机学院计算机系 上海 200240)
[2](上海大学计算机工程与科学学院 上海 200072)

摘 要 提出了一种改进的μC/OS-II的调度算法，增加了时间片轮转算法，把μC/OS-II改造为一个以任务优先级调度为主，时间片轮转调度为辅的实时操作系统。并通过碰撞检测算法和路径规划算法在该系统中的应用研究，表明改进的μC/OS-II系统能很好地满足需要，具有良好的实用性。

关键词 μC/OS-II 优先级 时间片轮转 调度算法

AN IMPROVED SCHEDULING ALGORITHM OF μC/OS-II

Xiong Yumei[1] Chen Yiming[2]
[1](Department of Computer, Shanghai Dianji University, Shanghai 200240, China)
[2](School of Computer Engineering and Science, Shanghai University, Shanghai 200072, China)

Abstract An improved scheduling algorithm of μC/OS-II was proposed. By increasing time slice cycle algorithm, μC/OS-II was reformed to a real-time system, mainly with assignment priority scheduling and supported by time slice cycle scheduling. After studying the application of collision detecting and route planning algorithm in the system, it is shown that the improved μC/OS-II system can meet the demand quite well and has good usability.

Keywords μC/OS-II Priority Time slice cycling Scheduling algorithm

0 引 言

嵌入式技术是一种以应用为目标，以计算机技术为基础，软硬件可剪裁，满足系统功能，对可靠性、成本、体积、功耗都有严格要求的专用计算机系统设计技术[1]。通常嵌入式系统要求能充分发挥硬件的功能，且实时性较高，一般都采用可剪裁的嵌入式实时操作系统。μC/OS-II是其中较为成功的系统之一，它是 Jean J. Labrosse[2]编写的一个源代码公开的抢占式实时内核，专为嵌入式应用设计，可用于 8位，16位和 32位单片机或数字信号处理器（DSP）。具有可移植、可固化、可剪裁、多任务、可靠、安全等特点。

1 μC/OS-II的任务管理与调度算法

μC/OS-II通过任务控制块（TCB）和任务栈实现对任务的管理、调度和切换。基于任务优先级的抢占式调度方式的内核总是运行就绪表中优先级最高的任务，从而保证系统对实时性的要求。μC/OS-II中所有的任务的优先级都各不相同，即使当前就绪任务队列中有多个任务，也只有优先级最高的任务能得到执行。所以创建任务时必须考虑一个限制条件，当本任务的功能执行完毕后，必须把自己阻塞或删除，否则其它低优先级的任务永远得不到执行。这个限制在多数情况下问题不大，只要注意各任务之间的功能以及相互关系，合理安排他们的优先级及阻塞条件即可。但在某些特殊情况下，需要有多个任务同时运行，即根据时间片轮流执行，在这种情况下μC/OS-II本身的调度算法就有一定缺陷，我们可以用其他方法来弥补这种缺陷，如优先级反转[3]，利用时钟中断服务调整任务的优先级，使得几个任务可以轮流执行。但也有不足，首先它是具体应用相关的[4]，也就是说不同的应用需要不同的时钟中断服务程序；其次，频繁更改任务的优先级会增加系统开销，甚至导致不健壮的系统崩溃。在仔细研究了μC/OS-II的任务管理与调度算法之后，我们找到了更好的解决办法，改进μC/OS-II的调度算法，增加时间片轮转算法，把μC/OS-II改造为一个以任务优先级调度为主，时间片轮转调度为辅的实时操作系统，即μC/OS-II支持同等优先级的任务，不同优先级的任务用抢占式调度算法，而同优先级的任务用时间片轮转调度算法，时间片在创建任务时设定。下面我们先了解μC/OS-II的任务管理与调度算法，然后介绍我们的改进算法。

μC/OS-II的任务调度算法是以任务优先级为基础，最简单的算法就是遍历整个任务队列，找出优先级最高的就绪任务，但有两个问题，一是任务调度非常频繁，简单的算法会导致系统性能的下降；另一个问题是遍历算法的时间复杂度跟任务的数量相关，会增加系统的不确定性，降低实时性。因此μC/OS-II设计了另一种算法，它增加了空间复杂度，降低了算法的时间复杂度，使得调度时间是一个常量，而与任务的数量等其他因素无关。

μC/OS-II的任务总是处于等待（阻塞）、就绪或执行状态。调度算法会检查就绪任务队列中是否有高优先级的就绪任务，

收稿日期：2006－10－26。上海市教委基金资助项目（05VZ03）。熊玉梅，讲师，主研领域：计算机控制与多媒体技术。

?1994-2016 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

如有, 就执行任务切换, 否则继续执行当前任务。任务调度会什么时候执行呢? 任何可能引起任务状态变化的系统调用或中断(ISR)之后都要检查是否需要进行任务切换, 例如当前任务发送了信号量给其他任务(导致某个任务从阻塞变为就绪)或本身阻塞等待某个事件的发生, 或从中断处理例程中出来, 如图 1所示。

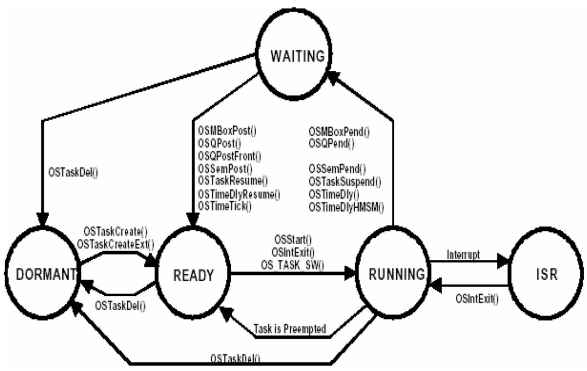

图 1　μC/OS-II状态转换图

调度算法的关键就是查找优先级最高的就绪任务。μC/OS-II采用了分组管理的办法来查询就绪任务, 它设置了 INT8U数组 OSRdyTb[]来记录任务状态(是否就绪)。它依据任务的优先级分组, 0~7为第一组, 对应 OSRdyTb[0]的每一位, 8~15为第二组, 对应 OSRdyTb[1]的每一位, 依此类推, 直到所有的任务都放到 OSRdyTb[]数组中, 为了加快查询速度, μC/OS-II采用了另一个 INT8U变量 OSRdyGrp来记录每组的状态, 若第 n组中有就绪任务, 即 OSRdyTb[n]!=0, 则 OSRdyGrp的第 n位置 1, 否则置 0。即 OSRdyGrp中的第 n位代表了第 n组任务的状态, 该组中有任何任务就绪, 那 OSRdyGrp的第 n位就被置 1, 而 OSRdyTb[n]中的每一位对应的是优先级为 n*8~n*8+7的任务。所以μC/OS-II最多有 8组任务, 每组最多 8个任务, 最多可支持 64个任务(若更改 OSRdyTb[]的类型为 INT16U, 支持的任务数可扩展到 128个, 调度算法类似)。

若优先级为 n的任务就绪, 那么高三位 Y=(n>>3)决定了 OSRdyGrp中的第 Y位应该置 1, 而低三位 X=(n&0x7)决定了在 OSRdyTb[Y]中的第 X位应该置 1。任务的优先级在创建时确定, Y和 X的值也同时创建并存储在任务的控制块中, 任务调度算法 1如下:

```
OSTCBY=priority>>3;
OSTCBBitY=1<<(priority>>3);
OSTCBX=priority&0x07;
OSTCBBitX=1<<(priority&0x07);
```

设置某个任务就绪的算法简单如下:

```
OSRdyGrp|=OSTCBBitY;
OSRdyTb[OSTCBY]|=OSTCBBitX;
```

将某个任务从就绪队列中移走的算法如下:

```
if((OSRdyTb[OSTCBY]&=~OSTCBBitX)==0)
OSRdyGrp&=~OSTCBBitY;
```

任务调度算法相对复杂一些, 但也很简单, 如下:

```
y=OSUnMapTb[OSRdyGrp];
x=OSUnMapTb[OSRdyTb[y]];
prio=(y<<3)+x;
if(prio<OSHiCur)
    OS_TASK_SW();  /*任务切换*/
```

这里需要说明的是 OSUnMapTbl数组, 这也是调度算法的关键所在, 因为μC/OS-II中的优先级是唯一的, 所以它用优先级来表示任务控制块数组的下标。算法需要判断 OSRdyGrp和 OSRdyTb[]中的非零最低位 bit位, 即最高就绪优先级组和组中最高优先级就绪任务, 因为 OSRdyGrp和 OSRdyTb[]都为无符号 8位整型, 最多有 256种取值, 假如我们把每种取值的非零最低位 bit的位数存入数组 OSUnMapTbl, 那么马上就可以根据 OSRdyGrp和 OSRdyTb[]的值查到最低就绪位, 若 OSRdyGrp的值是 0x34, 那么 OSUnMapTbl[0x34]的值就应该是 3, 若 OSRdyTb[3]的值是 0x62, 则 OSUnMapTb[0x62]的值应该是 1, 所以马上就得到优先级最高的就绪任务是(3<<3)+1, 即 25。

2　改进算法

上面我们介绍了μC/OS-II的任务调度管理算法, 通过牺牲空间复杂度的方法, μC/OS-II的任务调度管理算法只需要十几条指令就实现了任务的调度及任务状态的改变。但μC/OS-II调度管理算法的缺点就是支持的任务数较少(最多 64个), 不支持多个同等优先级任务和时间片轮转调度算法, 下面就介绍我们的改进算法, 使得每个优先级可支持 4个任务, 最多可以支持到 256个任务。

为了支持我们的算法, 我们在任务控制块中增加了几个数据项:

```
INT8U OSTCBZ;  OSTCBBitZ;
INT16U OSTCBSliceCnt;
INT16U OSTCBSliceRemainCnt;
```

OSTCBZ用来记录任务在时间片轮转组中的位置, OSTCBBitZ则是用 OSMapTbl[ptcb->OSTCBZ]计算而得到的在 OSRdyBmpTbl[prio]和 OSRdyTsITb[prio]中的 Bit位, 这样我们就可以在调度算法中快速得到需要调度的任务。OSTCBSliceCnt用来记录任务时间片的值。而 OSTCBSliceRemainCnt是系统用来记录本任务还有多少时间片可执行, 它的值每次在时钟中断里减 1, 到零之后就轮换另一个任务执行。另外我们还定义了一组宏定义和几个全局变量:

```
#define  OS_TSL_SHIFT      2
#define  OS_TSL_NUMBER     4
#define  OS_TSL_MASK       0x03
INT8U OSRdyBmpTb[OS_BMP_SIZE];
INT8U OSRdyTsITb[OS_BMP_SIZE];
```

在改进算法中我们增加了同等优先级的任务, 我们称之为时间片轮转组, 它们之间的调度采用时间片轮转算法, 而查找最高就绪优先级还是采用原有算法, 但得到的 prio仅是当前就绪任务中的最高优先级的值, 此时在这个时间片轮转组中可能有 1个或多个任务都处于就绪状态, 此时我们不能再采用优先级的值作为任务在队列中的下标, 如何找到需要调度的任务是我们算法的一个关键, 我们根据μC/OS-II原来算法的特点作了一些扩展, 我们定义了 OS_TSL_SHIFT和 OS_TSL_NUMBER为时间片轮转组的容量, 从(prio<<OS_TSL_SHIFT)到((prio<<OS_TSL_SHIFT)+OS_TSL_NUMBER)就是优先级为 prio的时间片轮转组在任务队列中的位置, 若 OS_TSL_SHIFT为 0, 即为μC/OS-II原来的算法。我们设置 OS_TSL_SHIFT的值是 2, 即时间片轮转组的大小 OS_TSL_NUMBER为 4。而 OSRdyTsITb[prio]的值就是优先级为 prio的时间片轮转组的就绪映射表,

?1994-2016 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

跟 OSRdyGrp 类似，我们调用 OSUnMapTbl[OSRdyTsITbl[prio]]就可以得到时间片轮转组中的那个任务应该先执行（从0到 OS_TSL_NUMBER－1），为了简化调度程序，当一个任务的时间片执行完毕，我们将会把它从 OSRdyTsITb[prio]就绪映射表中移走，这样虽然该任务是就绪的，但不会被调度到，当 OSRdyTsITb[prio]就绪映射表中所有的任务都被移走，即都执行过了，我们再把 OSRdyTsITb[prio]从备份就绪映射表 OSRdyBmpTb[prio]中恢复回来。

而与改进算法相关的函数有：任务管理函数（创建、删除、阻塞、恢复）、任务调度函数（有两个地方，OS_Sched和 OSIntExit）、时钟节拍服务程序（OSTimeTick）、任务就绪（OS_EventTaskRdy）和任务阻塞（OS_EventTaskWait）等。所有相关函数都需要修改，下面是几个关键的算法的说明。

任务创建算法的修改，增加了从时间片轮转组中查找空闲位置，若没有找到，表明同等优先级的任务数已达上限，创建任务失败，否则初始化任务控制块，设置相应的值，包括 OSRdyBmpTb[]和 OSRdyTsITb[]。

```
/*在任务队列中查找空闲位置*/
for(taskid=0; taskid<OS_TSL_NUMBER; taskid++){
  if(OSTCBIdTb[(prio<<OS_TSL_SHIFT)+taskid]==(OS_TCB*)0)
    break;
}
if(taskid<OS_TSL_NUMBER){ /*找到空闲位置，得到新的任务ID*/
  taskid=((INT16U)prio<<OS_TSL_SHIFT)+taskid;
    ……
ptcb->OSTCBZ=taskid&OS_TSL_MASK;
ptcb->OSTCBBitZ=OSMapTbl[ptcb->OSTCBZ];
/* 把任务放入时间片就绪映射表和备份表 */
OSRdyBmpTb[prio]|=ptcb->OSTCBBitZ;
OSRdyTsITb[prio]|=ptcb->OSTCBBitZ;
    ……
}
```

任务调动算法的修改相当简单，用μC/OS-II老的调度算法得到就绪任务的最高优先级之后，增加一行代码就可以得到需要执行的任务ID：

```
OSIdHighRdy=((INT16U)prio<<OS_TSL_SHIFT)+OSUnMapTbl[OSRdyTsITb[prio]];
```

这行代码的功能就是利用映射关系 OSUnMapTbl[OSRdyTsITb[prio]]得到该任务在时间片轮转组中的位置，再根据该任务的优先级偏移后相加就得到应该执行任务的下标，然后算法的其他部分跟以前一样即可。

改进算法的关键在时钟节拍服务程序，时钟中断服务程序定时呼叫节拍服务程序，时钟节拍服务程序原来的基本功能是实现任务的延迟功能，这里我们扩展了时间片轮转功能，添加了如下代码：

```
if(OSTCBCur->OSTCBSliceRemainCnt)
/*当前任务的时间片减1 OSTCBCur->OSTCBSliceRemainCnt--;
/* 如果当前任务剩余时间片为0，切换新任务*/
if(OSTCBCur->OSTCBSliceRemainCnt==0){
/* 恢复任务的时间片值，OSTCBSliceCnt为备份的时间片 */
OSTCBCur->OSTCBSliceRemainCnt=OSTCBCur->OSTCBSliceCnt;
/* 把任务从时间片轮转组的就绪映射表中移走 */
OSRdyTsITb[OSTCBCur->OSTCBPrio]&=~OSTCBCur->OSTCBBitZ;
/* 如果所有就绪任务都执行过了，即时间片就绪映射表为0，则恢复后重新再来*/
if(OSRdyTsITb[OSTCBCur->OSTCBPrio]==0){
OSRdyTsITb[OSTCBCur->OSTCBPrio]=OSRdyBmpTb[OSTCBCur->OSTCBPrio];
}……}
```

通过这些代码，我们就实现了时间片轮转算法，另外再增加一些辅助代码就完成了整个算法的改进，所谓的辅助代码，就是所有跟任务就绪、阻塞有关的代码，我们在设置任务就绪或阻塞时不要忘了同时设置时间片就绪表，代码类似：

```
/* 任务就绪 */
OSRdyGrp|=ptcb->OSTCBBitY;
OSRdyTb[ptcb->OSTCBY]|=ptcb->OSTCBBitX;
OSRdyBmpTb[ptcb->OSTCBPrio]|=ptcb->OSTCBBitZ;
OSRdyTsITb[ptcb->OSTCBPrio]|=ptcb->OSTCBBitZ;
/* 任务阻塞 */
OSRdyTsITb[ptcb->OSTCBPrio]&=~ptcb->OSTCBBitZ;
if((OSRdyBmpTb[ptcb->OSTCBPrio]&=~ptcb->OSTCBBitZ)==0x00){
if((OSRdyTb[ptcb->OSTCBY]&=~ptcb->OSTCBBitX)==0x00){
OSRdyGrp&=~ptcb->OSTCBBitY;
  }
}
```

搜索μC/OS-II中所有相关的代码，修改所有类似代码。我们就得到了完整的抢占式结合时间片轮转的调度算法，经过测试，它在基本没有增加系统开销的情况下很好地完成了调度任务，为我们系统顺利地开展发挥了重要作用。

3 应 用

随着计算机技术和机械自动化技术的飞速发展，嵌入式操作系统也因为其实时性好、系统开销小、扩展性强等特点而在仿真[5]、机器人[6]等各方面得到广泛应用。我们的机器人路径规划算法就是运行在改进的μC/OS-II系统中，它通过传感器收集机器人周围的障碍信息，根据碰撞检测算法和路径规划算法修改行进方向、规划路径。我们将信息收集设为高优先级任务，而将信息整理、碰撞检测、路径规划等任务设为相同级别，使用时间片轮转调度算法，在保证数据不丢失的前提下，使得每个任务都有一定的处理时间，简化了整个系统设计，减少了多任务之间的依赖关系。实验证明，改进的μC/OS-II系统能很好地满足我们的需要。

4 总 结

在仔细研究分析了μC/OS-II的任务管理与调度算法之后，改进了它的调度算法，增加时间片轮转算法，把它改造为一个以任务优先级调度为主、时间片轮转调度为辅的实时操作系统。通过碰撞检测算法和路径规划算法在该系统中的应用研究，表明改进的μC/OS-II系统能很好地满足我们的需要，具有很好的实用性。

参 考 文 献

[1] 吕京建，肖海桥. 面向二十一世纪的嵌入式系统[J]. 半导体技术，2001.

（下转第89页）

?1994-2016 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

定位追踪功能，提取信号强度与特征资料库数据进行比对，粗略估计设备的位置，然后移动该测试系统，根据其信号随距离的变化判断非法设备及攻击源的位置。

4 基于 PDA的无线局域网测试系统的实现

本系统运行于手持式设备夏普 SL—5600型 PDA上，操作系统为 linux2.4.18，运行环境为 QT2.3.2，具有 WLAN物理层测试、性能测试、协议分析、安全测试、连接故障诊断功能，可通过 PDA上的 USB数据线与 PC机通信，将数据和测试报告导出，并可利用无线网卡接入无线局域网，连接 internet实现网络浏览及邮件发送。

4.1 系统应用环境

如图 4在有线网络中接入无线接入点（AP），多个无线设备与该 AP进行连接，均连入 internet这就构建了一个小型的 WLAN网络。将基于 PDA的无线局域网测试系统放于该网络中，移动该手持设备即可进行网络性能和安全的测试。

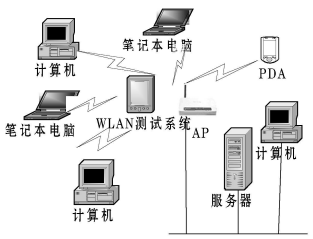

图 4　系统应用环境

4.2 系统运行流程

（1）启动系统，初始化 socket监听网卡接口，捕获数据包。图 5是捕获数据的一部分。

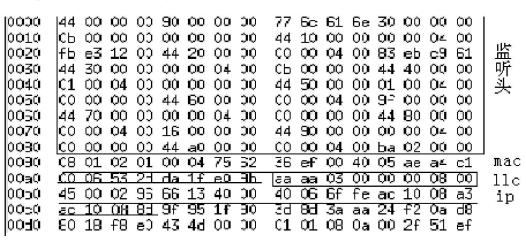

图 5　捕获的数据包

（2）预处理模块首先对采集的数据包进行帧类型的分类合并，去除不需要的数据，如数据帧只需要采集到 IP层的数据，然后将所需字段的信息，转换成上层分析模块识别的数据结构。

（3）分析模块分析捕获的数据包，首先根据监听头取得信号强度、噪声强度，如图 6所示，然后分析出该数据包为一个 IP数据帧，源 IP地址：172.16.8.163，目的 IP地址：172.16.8.141，数据未加密，表明该网络存在安全隐患，将该信息提交给上层响应模块，同时对分析得到的信息进行存储，以便进行数据回放。

图 6　信号、噪声强度实时曲线图

（4）响应模块根据分析模块提供的网络分析，采取不同的措施，一是实时进行报警，二是若有非法设备及攻击源存在，会启动定位跟踪，及时发现不安全因素，三是提供详细的测试报告，记录 WLAN网络的整体状态及工作性能、报警情况等。

4.3 与基于 PC机的无线局域网测试系统比较

本系统与基于 PC机的无线局域网测试系统比较，具有以下特点：

（1）本系统采用全面综合的测试分析方法，从物理层和网络应用角度完成对无线网络的全面测试。

（2）本系统可以在有网络的任何地点轻松地与无线局域网连接，对网络的动态运行状况进行测试分析，摆脱了线缆的束缚。

（3）PDA体积小、携带方便、耗电少，更好地满足网络人员外出作业使用，并方便移动于网络多点进行测试，使测试更全面、精确。

（4）本系统全部测试分析功能都基于掌上电脑，具有直观、智能、先进的操作界面，操作简单，只需轻轻一点，便可帮助你获得所需的信息。

5 结束语

无线网络测试是计算机网络发展的一个重要研究方向，本文提出了一种全面综合的测试分析方法和系统结构，为无线网络测试提供了一个理论框架；并在此基础上开发了一款基于 PDA的手持式无线局域网测试系统，成功用于无线局域网的测试、管理、维护等阶段，给出了一个很好的示列。

参 考 文 献

[1] ANSI/IEEE std 802.11, 1999 Edition. Standard for Information Technology-telecommunications and Information Exchange Between Systems-local and Metropolitan Network-Specific Requirement Part 11. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications [S]. copyright 1999 IEEE.

[2] 迪安公司. 无线局域网测试观点 [EB/OL]. http://www.airmagnet.com.cn/technology/article/WLANTest.pdf, 2003.5.

[3] 迪安公司. WLAN测试的实质 [EB/OL]. http://airmagnet.com.cn/articles/index.php?modules=show&id=154, 2003.7.

[4] 王国梁，陶洋，张静. 网络测试及性能评价模型研究 [J]. 中国数据通信，2003.11: 26-29.

[5] 李海林，王美琴，高振明. 基于 linux的 802.11b无线局域网数据包捕获方法 [J]. 计算机应用研究，2004.21(12): 270-272.

[6] Guo Fanglu, Chiueh Tzi-cker. Sequence Number-Based MAC Address Spoof Detection [EB/OL]. http://www.ecsl.cs.sunysb.edu/tr/TR182.pdf.

（上接第 86页）

[2] Jean J. Labrosse. 嵌入式实时操作系统 μC/OS-II第二版. 北京航空航天大学出版社，2003.

[3] 刘智臣，孟益民. 嵌入式操作系统 μC/OS-II中优先级反转问题及其解决方案. 科学技术与与工程，2005.5(1): 23-27.

[4] 冉明，邢汉承. 基于 μC/OS-II的嵌入式系统的设计. 微机发展，2005.15(1): 91-93.

[5] 田霖，李正熙，童朝南. 实时仿真与嵌入式系统. 微计算机信息，2005.21(8): 55-57.

[6] 张帆，高富强，顾兴海. 嵌入式系统在机器人视觉中的应用. 自动化与仪器仪表，2005(1): 38-39.

?1994-2016 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net