第 37 卷 第 14 期
Vol.37 No.14

计 算 机 工 程
Computer Engineering

2011 年 7 月
July 2011

·开发研究与设计技术· 文章编号：1000—3428(2011)14—0271—03 文献标识码：A 中图分类号：TP316.84

μC/OS-II 中优先级位图算法的改进

陈艳辉，涂晓东，王 卫

(电子科技大学通信与信息工程学院，成都 611731)

摘 要：针对基于查表方式的优先级位图算法占用存储空间较大的问题，提出 2 种能够减少存储空间占用的改进算法。改进算法 1 通过去除原表中的冗余数据，缩小表格的大小；改进算法 2 完全脱离查表思想，采用一种新的方法获得当前具有最高优先级的就绪态任务。分析结果表明，优化后的算法可以有效节省系统的存储空间。

关键词：优先级位图；查表；存储空间；C 语言

Improvement of Priority Bit Map Algorithm in μC/OS-II

CHEN Yan-hui, TU Xiao-dong, WANG Wei

(College of Communication & Information Engineering, University of Electronic Science & Technology of China, Chengdu 611731, China)

【Abstract】Two enhanced algorithms are proposed to solve the problem that the lookup table based priority bit map algorithm occupies a large storage space. One of the optimized algorithms reduces the size of the table by remove the redundant data from table. The other optimized algorithm gets the highest priority task which is in the ready state by a new method. It completely rejects the way of Lookup-table. Analysis result shows the enhanced algorithm can save a large storage space.

【Key words】priority bit map; lookup table; storage space; C programming language

DOI: 10.3969/j.issn.1000-3428.2011.14.092

1 概述

随着计算机技术的发展，嵌入式实时系统广泛地应用于消费电子、汽车、国防、航空航天、工业控制、仪表和办公自动化等领域。嵌入式实时系统兼有嵌入式和实时性的特点，能以可预见的方式对外部的并发事件进行及时的处理。μC/OS-Ⅱ的任务调度是按优先级进行的，系统总是运行任务就绪表中优先级最高的任务[1]。要获得当前具有最高优先级的就绪任务，嵌入式实时系统为了提高实时内核的确定性，一般采用优先级位图的就绪任务处理算法。在优先级位图算法中，是通过查优先级判定表来实现获得最高优先级就绪态任务的，而优先级判定表在系统中占用了很大的存储空间。本文提出 2 种方法对优先级位图算法进行改进：(1)通过简化优先级位图来改进算法；(2)完全脱离优先级位图，不会用到查表。这 2 种方法都对优先级位图算法做了改进，能够更方便地获得具有最高优先级的就绪任务。

2 算法分析

本文以 64 个优先级为例来说明优先级位图算法[2]，优先级位图算法中使用到的数据结构有：

```
Char priorityReadyGroup;
Char priorityReadyTable [8];
Char priorityMapTable [8];
Char priorityDecisionTable [256];
```

其中，优先级映射表初始化为：

```
char priorityMapTable [8] = {0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80};
```

优先级判定表初始化为：

```
Char priorityDecisionTable [256] =
{
0, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,        /* 0x00 to 0x0F */
4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,        /* 0x10 to 0x1F */
5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,        /* 0x20 to 0x2F */
4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,        /* 0x30 to 0x3F */
6, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,        /* 0x40 to 0x4F */
4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,        /* 0x50 to 0x5F */
5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,        /* 0x60 to 0x6F */
4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,        /* 0x70 to 0x7F */
7, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,        /* 0x80 to 0x8F */
4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,        /* 0x90 to 0x9F */
5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,        /* 0xA0 to 0xAF */
4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,        /* 0xB0 to 0xBF */
6, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,        /* 0xC0 to 0xCF */
4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,        /* 0xD0 to 0xDF */
5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,        /* 0xE0 to 0xEF */
4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,        /* 0xF0 to 0xFF */
}
```

获取进入就绪态的最高优先级是通过优先级判定表来实现的。处理过程用 C 语言描述如下：

```
high3Bit = priorityDecisionTable[priorityReadyGroup];
low3Bit = priorityDecisionTable[priorityReadyTable [high3Bit]];
priority = (high3Bit<<3)+low3Bit;
```

其中，high3Bit 为就绪任务最高优先级的高三位；low3Bit 为就绪任务最高优先级的低三位；priority 为就绪任务的最高优先级。如当前 priorityReadyGroup 的二进制值为 000100 11(十进制为 19)，以 19 为下标，从优先级判定表 priorityDecision Table 中获得就绪任务最高优先级的高三位为 0(二进制为 000)，假定当前 priorityReadyTable[0]的二进制值为 00001110 (十进制为 14)，则从优先级判定表 priorityDecisionTable 中获得就绪任务最高优先级的低三位为 1(二进制为 001)，因此，

作者简介：陈艳辉(1984－)，女，硕士研究生，主研方向：嵌入式操作系统，光纤存储网络；涂晓东，副教授；王 卫，硕士研究生

收稿日期：2011-02-22 **E-mail：**crying_leaf@163.com

得出当前就绪任务的最高优先级为 1。

3 算法的改进

3.1 改进算法 1

嵌入式实时操作系统与通用操作系统的区别在于：通用操作系统通常都期望获得最大的平均吞吐量，并防止系统出现饥饿和死锁状态，但是对嵌入式实时操作系统来说，这是远远不够的，嵌入式实时操作系统要求有极高的确定性，并具有内存资源占用少和低功耗的要求。本文对优先级位图算法的改进就是基于这一点，对优先级位图算法中的优先级判定表进行修改，将优先级判定表缩小或者是完全摆脱优先级判定表。

观察优先级判定表 priorityDecisionTable[256]可以发现，该表中的数据冗余严重，导致表格浪费大量存储空间。表格 priorityDecisionTable[256]除第 1 列以外，其他的各列元素的值均相同，因此只需保留原表的第 1 行和第 1 列即可获取进入就绪态的最高优先级。原表占用的空间为 256 Byte，优化后只占用 32 Byte。大大减小了表格占用的存储空间。将优先级判定表优化后，获取进入就绪态的最高优先级的算法也要相应改变，如下：

```
Char priorityDecisionTable_1[] = {0,4,5,4,6,4,5,4,7,4,5, 4,6,4,5,4};
/*原表的第 1 列*/
Char priorityDecisionTable_2[] = {0,0,1,0,2,0,1,0,3,0,1, 0,2,0,1,0};
/*原表的第 1 行*/
/*priority 高三位的确定*/
if (0 == priorityReadyGroup & 0x0F)
/*priorityReadyGroup 的低四位为 0，则表明 high3bit 在原表的第一列*/
high3bit = priorityDecisionTable_1[priorityReadyGroup >> 4 ];
else
high3bit = priorityDecisionTable_2[priorityReadyGroup & 0x0F ];
/*priority 低三位的确定，方法同高三位*/
if(0==priorityReadyTable[high3bit]&0x0F)low3bit=priorityDecisionTable_1[priorityReadyTable [high3bit] >> 4 ];
else
low3bit=priorityDecisionTable_2[priorityReadyTable [high3bit] & 0x0F ];
priority = (high3bit << 3) + low3bit;
```

由于 μC/OS-Ⅱ中的任务是按优先级进行调度的，因此只要找到优先级后，以优先级为下标就可以得到相应任务的任务控制块，进而可以对其进行调度。这里同样以上节中的实例用到的 priorityReadyGroup 的二进制值 00010011 为例，得到的就绪任务的最高优先级同样为 1。

3.2 改进算法 2

本文提出另一种获取就绪态的最高优先级任务的方法，这种方法完全摆脱了查表的思想，直接通过找到 priorityReadyGroup 和与之对应的 priorityReadyTable[]中出现 1 的最低位，来获取处于就绪态的最高优先级任务。大概的算法如下：

```
char i;
char high3bit = 0, low3bit = 0;
/*找 priorityReadyGroup 中出现 1 的最低位*/
for(i=0;i<8;i++){
if(priorityReadyGroup & priorityMapTable[i] == 1)
/*用 priorityMapTable[ ]作为测试值*/
break;
else
high3bit++;
}
/* 找 priorityReadyTable[high3bit]中出现 1 的最低位*/
for(i=0;i<8;i++){
if(priorityReadyTable[high3bit] & priorityMapTable[i] = = 1)
break;
else
low3bit++;
}
priority = (high3bit<<3)+low3bit;
```

这种方法主要是用 for 循环语句来实现算法，这里提出了另一种方法，不要用到 for 循环语句，就可以得到 priorityReadyGroup 和与之对应的 priorityReadyTable[]中出现 1 的最低位，这个算法主要用到了“按位与”和“按位异或”。char x=84; /*二进制 0101,0100*/，则 x&(x-1)的二进制值为：0101，0000。最右边的 1 变成了 0，这个值再与 x 按位异或：x ^ (x & (x-1))这个表达式的值为：0000,0100(十进制为 4)。这样就把 x 中最右边的 1 保留下来，而其他位全为 0，并且通过这种处理之后得到的十进制值为 2^n，如图 1 所示。

```
X&(X-1):
        01010100
      & 01010000
        01010000

X^(X&(X-1)):
        01010100
      ^ 01010000
        00000100
```

图 1 “按位与”和“按位异或”实例

把这个方法用到优先级位图算法中，得到以下算法：

```
unsigned int priority = 0;          /*优先级*/
unsigned int low3bit,high3bit;      /*低三位,高三位*/
/*获取就绪态的最高优先级*/
high3bit=get_priority(priorityReadyGroup^(priorityReadyGroup&(priorityReadyGroup-1))); /*得到高三位*/
low3bit=get_priority(priorityReadyTable[high3bit]^(priorityReadyTable[high3bit] & (priorityReadyTable [high3bit] -1)));
 /*得到低三位*/
```

其中 get_priority()定义如下：

```
unsigned int get_priority(int temp){
  switch(temp) {
  case 1: return 0;
  case 2: return 1;
  case 4: return 2;
  case 8: return 3;
  case 16: return 4;
case 32: return 5;
        case 64: return 6;
        case 128: return 7;
        }
}
```

这个算法完全没有用到优先级判定表，不需用查表的方法就可以得到具有最高优先级的就绪态任务，从而让系统可以节省一定的存储空间。

4 算法效率分析

算法效率是指算法执行的时间，对于同一个问题如果有多个算法实现，执行时间越短，算法效率就越高。为了比较同一问题的不同算法的效率，一般是从算法中选取一种对于

所研究的算法类型来说是基本操作的原操作，以该基本操作重复执行的次数作为算法的时间量度[3]。算法的时间量度记为：$T(n)=O(f(n))$[4]。由改进算法 1 和原算法相比较可知，2 个算法的时间复杂度均为 $O(1)$，但是改进算法 1 减少了表格对系统存储空间的占用。改进算法 2 中的第 1 种算法用到了 for 循环语句，因此该算法的时间复杂度为 $O(n)(n=8)$。改进算法 2 中的第 2 种算法时间复杂度为 $O(1)$。

5　结束语

本文提出的 2 种算法均对用优先级位图查找最高优先级就绪任务算法做了改进，第 1 种算法是通过缩小优先级判定表来减小其占用的空间，占用空间从原表的 256 Byte 优化到 32 Byte，减少了算法的空间开销，并且和原有的算法相比具有相同的算法复杂度。另一种算法完全摆脱查表的思想，不用定义 priorityDecisionTable[256]，而是直接通过查找 priority ReadyGroup 和与之对应的 priorityReadyTable[]中出现 1 的最低位，更加节省存储空间[5]。但是相对第 1 种算法和原有的算法，第 2 种算法存在一个缺点，就是当就绪任务的优先级低时，查找起来时间开销较大，不同优先级所花费的时间开销也不同。

参考文献

[1] 迟瑞娟，付　兵，刘吉孟. 基于嵌入式实时操作系统的寻线机器人设计[J]. 计算机工程, 2009, 35(18): 240-242.

[2] 罗　蕾. 嵌入式实时操作系统及应用开发[M]. 北京: 北京航空航天大学出版社, 2007.

[3] 严蔚敏，吴伟民. 数据结构(C 语言版)[M]. 北京: 清华大学出版社, 2006.

[4] Aho A V, Hopcroft J E, Ullman J D. Data Structures and Algorithms[M]. [S. l.]: Addsion-Wesley Publishing Company, Inc., 1983.

[5] 汤子瀛，哲凤屏，汤小丹. 计算机操作系统[M]. 西安: 西安电子科技大学出版社, 2001.

编辑　陈　文

(上接第 267 页)

6　结束语

本文研究基于 GPU 的位并行多模式串匹配。实验结果表明，利用 GPU 可以使移植以后的位并行多模式串匹配算法与等同条件下的 CPU 程序相比获得约 13 倍的加速比。相对于已有一些算法[3-4]所获得的加速比更大，但 GPU 由于硬件的局限，对实时性和吞吐量有很高要求的情况仍不适用，今后将对这些方面进行改进。

参考文献

[1] Gonzalo N, Mathieu R. 柔性字符串匹配[M]. 中科院计算所网络信息安全研究组, 译. 北京: 电子工业出版社, 2007.

[2] 武永超，华　蓓. 基于网络处理器的多模式串匹配研究[J]. 计算机工程, 2009, 35(8): 166-168.

[3] Huang Nen-Fu, Hung Hsien-Wei. A GPU-based Multiple Pattern Matching Algorithm for Network Intrusion Detection Systems[C]// Proc. of the 22nd International Conference on Advanced Information Networking and Applications. Okinawa, Japan: IEEE Computer Society, 2008.

[4] Giorgos V, Spiros A, Michalis P, et al. Gnort: High Performance Network Intrusion Detection Using Graphics Processors[EB/OL]. (2008-10-26). http://gpgpu.org/2008/10/26/gnort-high-performance-network-intrusion-detection-using-graphics-processors.

[5] Nvidia. Nvidia CUDA Programming Guide Version2.3.1[EB/OL]. (2009-08-10). http://developer.download.nvidia.com/compute/cuda/2_3/toolkit/docs/NVIDIA_CUDA_Programming_Guide_pdf.

编辑　陈　文

(上接第 270 页)

如果难度值在平均难度上下 0.3 范围内，为正常难度，否则需要重新抽取试题生成试卷。这样对于上表中抽取的第 2 份试卷需要重新生成。但可能会有教师在编辑试题设置难度属性时，对所有试题难度随意设置为几个简单值，如 4 种难度属性分别为 0.15、0.4、0.7、0.9 代表容易、较易、较难、难。那么所生成的试卷难度均为 0.503，都不符合此方法的要求，均需反复重新抽取试题，最终会导致试卷生成失败。因此，可以将此方法做适当修改，计算所生成试卷的难度平均值。这样做即公平又可行。

(2)在生成的多份试卷中，也可能会出现知识点覆盖比较少的情况，在生成的试卷中也需要进行检测，如所覆盖的知识点个数没有达到设定知识点个数的 75%，则重新抽取试题生成试卷。

7　结束语

本文研究一种应用于考试系统中的递归随机分割算法。实验结果表明：为了提高试卷生成的质量，建立高质量的试题库是非常重要的，在定型、筛选试题时，需要考虑试题分布是否均匀(越重要的章节，包括的知识点越多，试题库中包含相应的试题就会越多)，从而使试题库中的试题能更好地满足各类生成试卷的要求。

参考文献

[1] 朱振元，朱　承. 网络无纸化考试系统的开发研究[J]. 微型机与应用, 2002, 21(6): 43-46.

[2] 郁建政. 网络题库系统的设计与实现[J]. 电脑知识与技术, 2008, 26(3): 1729-1731.

[3] 闭应洲，苏德富，陈宁江. 基于矩阵编码的遗传算法及其在自动组卷中的应用[J]. 计算机工程, 2003, 29(6): 73-75.

[4] 王尚平，王　琪，张亚玲. 基于身份认证的网络考试安全机制[J]. 计算机工程, 2009, 35(18): 136-138.

编辑　陈　文