基于 Embed 系统的缓存利用策略①

石永生[1]，高照恒[2]

[1](江苏信息职业技术学院 计算机工程系，无锡 214101)
[2](上海理工大学 光电信息与计算机工程学院，上海 200090)

摘 要: 基于嵌入式在智能工控中缓存预分配的思想，提出一种预分配缓冲区管理机制 Index，该机制实现了缓存的静态分配和回收，并在试验中对该机制进行了改进，使其更好的适应高速小型数据报的分配和管理。同时对缓存动态分配回收策略做了研究提出一种连带释放缓存的思想。最大限度实现一次分配一次释放，很好的克服了缓冲区因多次分配回收造成的内存泄露问题。通过性能分析比较，Index 机制报文处理能力高效可靠。

关键词: 嵌入式缓存预分配策略；Index 机制；嵌入式缓存回收策略

Embedded System Cache Utilization Strategy

SHI Yong-Sheng[1], GAO Zhao-Heng[2]

[1](Department of Computer Engineering, Jiangsu College of Information Technology, Wuxi 230039, China)
[2](School of Optical-Electrical and Computer Engineering, University of Shanghai for Science and Technology, Shanghai 200090, China)

Abstract: This paper proposes a pre-allocated buffer management mechanisms Index. Based on the idea of memory pre-allocation, the mechanism supports static and dynamic allocation and backup allocation. The mechanism is proved, and make it better adapt to the small data report distribution and management. At the same time, a study of the dynamic recovery has been done and proposed a strategy for the associated-release of dynamic recovery propose of maximum allocation of just by once to achieve the release of a strategy. Problem of memory link is solved by avoiding repeatedly memory allocate and recovery. Through the analysis and comparison,performance index mechanism of message handing capacity, high efficiency and reliable.

Key words: pre-allocation for cache; strategy of Index; strategy of cache recovery

1 引言

传统协议栈缓冲区管理在实现静态分配缓存时小数据包经常占用大型缓存块造成系统缓存浪费，大型数据包又要跨越多个缓存块使管理缓冲区变得更加复杂。为此引入了静态预分配 Index 策略，很好的解决了缓存静态分配的大量浪费问题且使缓冲区的管理更加简单可靠，尤其在小型缓存的分配和处理上，Index 机制处理操作具有巨大的优势。为提高动态缓存区动态分配速度，充分利用了缓存块多指针。由于协议栈内部数据拷贝次数过多，协议处理层次复杂导致数据传输过程中操作系统和协议处理开销过大，从而影响了整个系统的性能。我们引入零拷贝是实现主机或路由器等设备高速网络接口的主要技术。零拷贝策略很好的解决了协议栈中报文拷贝次数过多的问题，我们通过减少或消除关键通信路径影响速度的操作，降低数据传输的操作系统和协议处理开销，从而有效提高通信性能，实现高速数据传输。引入了连带释放缓存策略，可有效避免缓存释放时分阶段释放，缓存回收机制需要多次回收而造成的缓存碎片，最大限度的实现了缓存申请后的空间连续性和独占性，缓存的集中释放策略可有效减少缓存碎片。

① 收稿时间:2011-05-17;收到修改稿时间:2011-06-19

2　嵌入式缓存利用策略

通过 Embed 系统来研究协议栈缓存的管理策略。Embed 系统不同于一些高端平台，拥有高性能的服务器和巨大的存储容量，并且很难做到多 AGENT 的协同处理。相比之下，Embed 系统更是一个独立处理，独立存储，独立管理的系统。

传统的 TCP/IP 协议栈对于 Embed 系统来说过于庞大不可能也不需要去完全实现，这在一定程度上使得 Embed 协议栈缓冲区管理需要更加针对性的策略。Embed 系统的存储资源有限且宝贵，需要更加高效的缓冲区管理策略实现统一的管理，这些管理包括：实时高效分配缓冲区，高效的缓冲池分配策略要求在系统允许的最差时间（WCET）内得到所需的缓冲区；缓冲区的回收，对使用完毕的缓冲区，即时释放并返还缓冲池；缓冲池的操作，即对池中相邻缓冲区合并，有效避免碎片并消除池中存在的碎片。

2.1 静态缓存区预分配策略—Index

静态缓冲区预分配是提前分配一定数量，不同大小的缓冲块队列。因此具有极高的分配实时性，因为队列大小已预先限定，用此策略来处理高密度，且数据包较小的报文具有很强的针对性。

缓冲块队列逻辑上用散列实现。先对网络中的数据包进行多时间段批次抓取，统计如下:

表 1　网络数据包多批次抓取统计

Packet size(bytes)	*<16*	*16-31*	*32-63*	*64-127*	*128-255*	*256-511*	*>512*
Quantity	6,291	41,594	72,403	42,576	23,829	12,711	12,857
Percents(%)	2.96	19.60	34.12	20.05	11.22	5.98	6.05

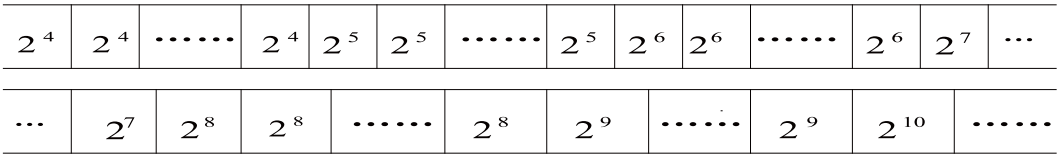

图 1　静态缓存的 Index 链

Index 策略首先建立一张 32 位字的索引表，每一位表示一种空闲的内存大小，在 32 位嵌入式系统中内存为 8B 对齐，我们设置最小的缓冲区大小为 16B，并以 2N 依次递增，最大的静态缓冲区大小为 512B，以数据包比率的 5%为一个缓冲区分配额，比率不足 5%的分配一个相应大小的缓冲区。逻辑上构成了一条静态散列，物理上是一条 Index 链。

其中大小为 2^n 的缓存区个数 S_n 为:

$$S_n = P_n N / r \quad (1)$$

P_n 为缓存区匹配容纳数据包数量占统计总量的比率 Percents，N 为单位比率可分配缓存个数的最大值，r 为分配缓存的额定的单位比率，M 为 Embed 系统缓冲池总量 Memory，则需满足：

$$\sum_{i=4}^{n} 2^n P_n N / r \le M \quad (2)$$

因 Embed 系统自身特点，这里我们设定最小缓存区大小为 2^4，最大缓存区大小 2^n 满足其大于最大数据包大小即可。有效解决了小数据报也要在占用大缓存的空间浪费，极大提高缓存利用率。

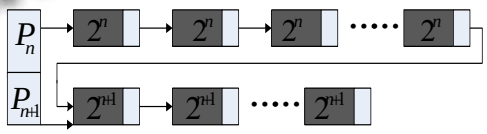

图 2　静态缓存的逻辑散列链

数据包存储时 Index 链存在这样一个弊端：假设在数据链中共有 5 个大小为 64B 的缓存区，当一段时间内网络中数据包大小恰好均是小于 64B 而又大于 32B，第六个这样的数据包到来而首先存入的第一个数据包又没有处理完毕时，第六个数据包存放在哪？对此，Index 链自身的特点可以使问题很好的解决，Index 链是一条优化后的散列，64B 缓冲区之后是 128B 的缓存区，可以临时存放此时过于密集的数据包，待前面

的缓存处理完毕。此时又没有造成缓存区过多的浪费，相反，Index 机制充分利用了未分配的缓存。

2.2 动态缓冲区设计

动态缓冲区可根据用户指定大小分配缓存，用户申请的空间大小就要在数据处理过程才能得出，甚至随着进程的改变而改变，缓存的分配，释放和回收也要支持这种动态变化的需要。

我们所涉及的 Index 静态缓存在分配小数据缓存是速度快，存储利用率高，但在分配大数据缓存 2n 时，即使每次均将数据包放入相应大小的数据缓存格中，仍旧有 25%的 2nB 缓存浪费掉了。

从数据包的抓取统计表中可以看到，数据包大小在 256B 和 512B 之间和大于 512B 的百分比分别为 5.98%和 6.05%，缓存利用率为 E，其缓存浪费因子分别为 d9 和 d10,

$$d_j = \frac{2^j p_j * 0.25}{\sum_{i=4}^{n} P_j * 2^n * 0.25} * 100\%(n = 10) \quad (3)$$

其中在静态 Index 策略中 n=10 便足够容纳下最大的数据包，即便不足以存放，我们的迭代指数 n 可以继续递增，直到满足相应的存储。

同样，我们可以计算其他 Index 缓存影响因子，并对其加以统计分析。

我们看出，这种情况下缓存分配次数仅仅占用 5.98% 和 6.05%的大缓存分配，相应的内存浪费因子高达 d9=0.18, d10=0.36。即 12.03%的使用率确浪费了系统资源的 54%，对大缓存分配，我们避免使用静态缓存 Index 分配策略，而使用动态缓存分配策略。

表 2 多级次缓存资源影响因子统计

Memory Size	<16	16-31	32-63	64-127	128-255	256-511	>512
E-rate	<0.01	0.03	0.12	0.15	0.16	0.18	0.36

在静态缓冲池 S_Buffer 外，我们定义动态缓冲区 D_Buffer 用于存储大数据报文，且将每个缓冲区分为 64K 的数据块，对于大型数据报文将其存入动态缓存中，这里定义的大型数据报文为大于 256B，在静态缓存中必须存入大小在 256B 以上缓存块的数据报。这样采用动态缓冲区分配时每个数据报文至少占用 4 个数据块，平均每个报文浪费半个数据块的缓存。动态缓存由连续的动态缓存块构成。

常用动态缓存分配算法有最佳适应算法，最差适应算法和最佳适应算法，首先分析算法的时间性能。我们先对动态缓存块结构体进行定义。

D_Buffer 块结构体定义如下

```
typedef struct D_Buffer Buffer
    {
        D_Buffer * pPreBlock;
        D_Buffer * pNextBlock;
        Char packet[BufferSize];
        Short int reference;
    }
```

动态分配缓冲区时，最先适应算法的时间效率最高，最佳适应和最坏适应算法都需要对所有的空闲区进行一次搜索，为进一步优化最先适应算法时间效率，在 D_Buffer 缓冲区中引入 rear 指针，开始指向 D_Buffer 缓冲区的初始块，每次分配完毕后，指向 D_Buffer 分配块的下一个缓冲块，这样既可保证在每次进行下一次缓存块分配时，无需从开始缓存块重新搜索可分配的缓存区。待 rear 指针指向了 D_Buffer 缓存最后或虽未指向最后位置但已无法完成下一次分配时，rear 指针重新指向 D_Buffer 缓存块初始位置，重新搜索。

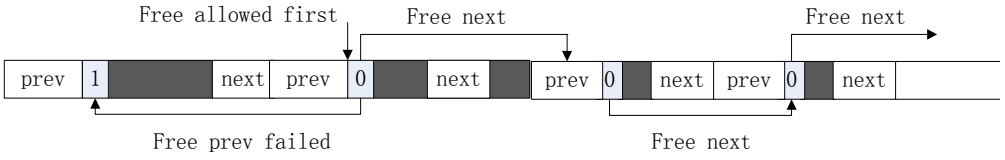

图 3 缓存释放的连带策略

2.3 协议栈处理层次的报文零拷贝策略

缓冲区管理要能方便的操作可变长缓存，可在缓存头部和尾部添加数据，可以从缓存中移除数据，并尽量减少网络层由于最大传输单元 MTU 的限制对传输层传来的数据包进行分片及对分片的重新组合引起的拷贝。也只有当要发送的数据真正发生移动时才操作整个数据报文。这样提高了协议栈处理性能，节省了缓存空间。

当从应用层获得数据报文，需要先通过网络接口卡通过 DMA 方式将数据报文送至缓冲区，在缓冲区中报文要经过 TCP/IP 协议栈的层层处理，由软中断负责从缓冲区中读取数据报文驱动报文处理器，再拷贝到用户态缓存中。真正实现数据移动的是从网络接口卡到数据缓冲区，再从缓冲区到用户态的缓冲区，在中间协议栈处理过程中，不同层次的协议栈可以从数据缓冲区中存取报文，不同协议层之间传递时传递的是数据报指针，即实现的是必须的数据拷贝之外，协议栈处理层次的报文零拷贝。

3 测试及性能分析

对 Index 策略执行的时间效率和有效性可靠性进行测试，对此首先需要注重提高小数据报文的流量，其次在动态缓存释放策略进行可靠性和稳定性测试，延迟释放可能会影响协议栈处理速度，对于降低碎片率和提高速度两方面我们需进行权衡。

3.1 测试设计

在 X86CPU.256M 内存，测试环境为 Fedoral Core4 系统，GCC 编译器，利用 Libcap 工具和虚拟网卡读取网络数据，通过网络环境测试，结果证明该缓冲区机制的正确性和健壮性，同时也说明了该缓冲区机制具有良好的可移植性，可以移植到不同的软件和硬件平台。不断进行缓存的申请和释放。

3.2 测试分析及测试结果

静态缓冲区的设计一般采用的是定长策略方法，对定长的静态缓冲区，选用缓冲块额定定长为 128B 和 256B 的方案进行比较，对于跨越多个缓存块的大小，平均浪费半个缓存块大小。定义对于报文处理速度相同，且同时收到大量小型数据报文时，采用比较额定定长缓存块静态存储和 Index 策略的缓存利用率 E 和缓存占用总量 M，定长缓存块长度 K 置为较常用的 128B 和 256B，存储 T 个随机小数据报文计算相应的缓存利用率和存储使用量，置 T=1000，则，M1=128，000B　M2=256，000B

$$M' = \sum_{i=4}^{8} 2^i P * 1000 \quad (4)$$

$$E = \sum_{i=4}^{8} (2^i / K) * p \quad (5)$$

表 3　定长及 Index 策略对小数据报处理分析

Static Mem	128(B)	256(B)	Index
Total Mem(M)	128,000	256,000	82,848
Utilization(E)	44%	21%	75%

其次我们对于大数据报在分配过程中的缓存碎片情况，定义一个有意义的缓存碎片率为不可分配缓存块数与缓冲池中已存储的大数据报比值，即已存储数据报个数越多且缓存中留下的容量小于大数据报的缓存区数越少，则缓存率越低。我们基于 MemView 工具对缓存分配监测，从缓存链扫描情况分析缓存连带释放策略的缓存分配情况。

表 4　缓存释放策略比较

缓存释放策略	1000S	5000s	10000s	40000s
多次释放	3/214	9/273	12/281	9/320
连带释放	2/231	3/252	3/294	3/304

虽然连带释放策略在缓存内存储数据单元增加的情况下并没有出现碎片数量的增加，但我们同时注意到随着存储数据的增加，连带释放策略的缓存使用量较多次释放策略缓存用量增加的更明显。在数据处理速度较慢的环境下缓存使用量增量将更为突出。

4 结语

本文在传统静态缓存分配策略的基础上提出了一种预分配的 Index 静态分配策略，该策略在处理小数据报文方面分配时间性能与传统策略相当，对特定缓存要求严格控制平台如专用 Embed 系统方面较有优势，大大节省了系统缓存占有量并为大数据包缓存处理节省了宝贵空间。这样做的代价是大量大型数据包到来时可能会引起缓存分配不足，也将是我们下一步要解决的问题。此外连带缓存释放策略在一定程度上缓解了缓存一次申请多次释放而引起的缓存碎片，降低了缓存碎片率。

（下转第 172 页）

义、较为独立的信息单元；呈现出一定民族特征的基本图案基元，并构建多级图案基元库。然后由设计者设计或选择构图模式，进行图案纹样的填充与调整，生成民间艺术图案。根据设计效果表明，技术方案可以在保持艺术家创意的基础上大大提高其工作效率，并可对原始创意快速提供多种基元表达方案进行选择。为此，本文所提出的创意与制作过程有效分工的方式对民间艺术图案设计和非物质文化遗产保护均有很好的促进作用，有助于加深 CAD 技术开发人员与艺术家之间的相互理解与交流，凭借现代科技技术提升民间艺术图案的价值。

参考文献

1 潘云鹤.智能 CAD 方法与模式.北京:科学出版社,1997.31–42.

2 Kaplan CS. Computer Generated Islamic Star patterns. Proc. of Bridges 2000, Mathematical Connections in Art. Music and Science, 2000:105–112.

3 Raghavachary S. Tile-based kolam patterns. ACM SIGGRAPH'04 Skotehes, New York, NY, USA, ACM Press. 2004:58–69.

4 Wong MT, Zongker DE, Salesin DH. Computer generated floral ornament. SIGGRAPH'98: Proc. of the 25th Annual Conference on Computer Graphics and Interactive Techniques, NY, USA, ACM Press, 1998. 423–434.

5 俞杰,耿卫东,潘云鹤.面向产品创新设计的 CAD 方法综述.计算机辅助设计及图形学学报,1999,21(2):96–99.

6 吴奕立,颜钢锋.织物计算机模拟显示方法的探索纺织学报,1999,20(5):36–38.

7 Xu J, Kaplan C, Mi XF. Computer generated paper cutting. Proc. of the 15th Computer Graphics and Applications, Washington. IEEE Computer Society. HI USA, 2007: 343–350.

8 彭冬梅,刘肖健,孙守迁.信息视角:非物质文化遗产保护的数字化理论.计算机辅助设计及图形学学报,2008,20(1):117–123.

9 张显全,于金辉,蒋凌琳.计算机辅助生成剪纸形象.计算机辅助设计与图形学学报,2006,32(11):248–250.

10 包恩伟,孙守迁.基于组件特征模型的产品布局设计.工程图学学报,1998,(3):19–24.

11 孙守迁,包恩伟,潘云鹤.面向产品布局设计的组件特征模型.计算机辅助设计与图形学学报,1999,11(1) 21–25.

12 Aragios N, Deriche R. Geodesic active contours and level sets for the detection and tracking of moving objects. IEEE Trans. on Pattern Analysis and Machine Intelligence, 2000, 22(3):266–280.

（上接第 126 页）

参考文献

1 Teng QM, Wang H, Chen XQ. A HAL for component-based embedded operating systems COMPSAC, 2005,(2):23–24.

2 谭永宏.基于 TMS320C6701 的嵌入式智能视觉监控系统设计与实现.计算机系统应用,2008,17(9):73–76.

3 Llikalr, Iyerr, Newelld. Microarchitectural anatomy of a commercal TCP/IPstack. Communication Technolygy Laboratory, 2004,5(10):38–43.

4 Anthony J. Massa Embedded Software Development with eCos. Prentice Hall PTR,2002:338–344.

5 Zhou XY, Huang WW, Shi L. Detecing of memory error in C programs based on source code instrumentation Ship Electronic Engineering, 2004,24(9):70–73.

6 Giudici P. Applied Data Ming: Static Methods for Bussiness and Industry. Beijing, China: Electronics Industry Press, 2004.