基于内存池的 KIWI 数据组织方法研究①

KIWI Data Organization Method Based on Memory Pool

赵　松　雷跃明　鹿　琪　(重庆大学 软件学院 重庆 400044)

摘　要： 基于对 KIWI 数据格式的索引数据和地图数据的统计分析，总结出其组织和存储的规律。针对嵌入式设备的应用特点，提出了一种基于内存池的数据组织方法，并给出了相关类的结构定义和核心方法的实现过程。最后对几种常用的换页算法进行了对比分析和实验验证，结果表明内存池方法显著地提高了数据加载与解析的效率。

关键词： KIWI 数据格式　内存池　页面置换算法

KIWI 格式是由日本 KIWI-W Consortium 制定的导航电子地图数据物理存储格式(PSF)标准，用以满足嵌入式系统应用快速精确和高效的要求[1]。KIWI 格式数据以分块的形式存储于物理介质中，并建立类四叉树的索引结构对数据块进行管理，其控制信息和标识信息以 bit 为单位存储来减少存储空间，KIWI 还具有良好的扩展性。目前，KIWI 格式是一些国内主要导航地图厂商的标准制图参考格式，有着很广泛的应用性。

1　引言

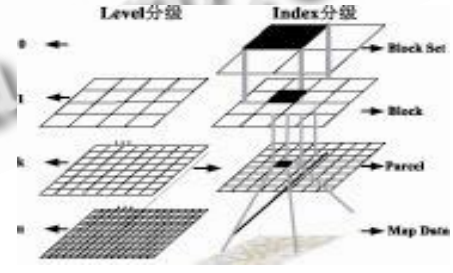
KIWI 数据包通常包含丰富的信息量，数据容量较大。而嵌入式导航设备又有 CPU 效率较低，内存容量受限等因素制约，因而如何实现对地图显示、引导信息、兴趣点查询等操作进行实时快速的响应，提高数据的解析效率就显得尤为重要。本文根据对 KIWI 数据的索引与数据包进行统计分析的结果，提出了使用内存池机制对数据进行组织和管理的方法，用以提高实际应用中 KIWI 数据的加载与解析速度。

2　KIWI的索引与数据包

KIWI 数据是以网格化方式进行存储。以 KIWI 主地图框架为例，数据按照不同比例尺表示的详略程度进行分层(Level)。每一层被划分成规则矩形的网格，每个网格对应着存储地图数据的基本单元，即数据包。基于这些网格来集中创建块集(Block Set)、块(Block)和包(Parcel)上的分级索引结构，如图 1(a)所示。

2.1　二者关系

KIWI 数据中的块集、块两级索引记录存储于包数据管理发布头(Parcel Data Management Distribution Header)中（如图 1(b)所示），对块索引提供的物理地址进一步解析可得到包索引。包索引中包含了实际地图数据包（Map Data）所存储的物理位置和存储空间，包索引与地图数据包之间是一一对应的。通过包索引对 KIWI 数据文件进行定位，最终可得到主地图数据包。



(a)数据层分级示意　(b)索引分级示意

图 1　KIWI 的 Level 分级和 Index 分级示意图

2.2　统计分析

对 KIWI1.22 版中国大陆数据进行统计，可得出其索引和数据包有效记录数分别为 1064 和 1647878 条。进一步统计每层的索引和数据包所占用的最大的存储空间，得出统计结果如表 1 所示。从

① 收稿时间:2008-08-30

中可以看出，随着层数的降低，地图的数据包划分的网格越来越密，因而存放索引数据所需的存储空间也越来越大。0 层的索引空间的最大值是整个地图数据索引空间的最大值，因而整个 KIWI 文件中索引大小不超过 32 KB。对各层数据包大小统计结果进行分析，可以判定 KIWI 文件中的数据包最大值不超过 128KB。因包含的地物信息特别丰富而大于 128KB 的数据包，会进一步被分割为更小的数据包存储于文件中。

表 1　索引和数据包记录统计表

每层最大值	索引（byte）	数据包（byte）
5 层	32	65216
4 层	352	125312
3 层	3296	128192
2 层	6432	120256
1 层	6624	116416
0 层	24608	84096

3　内存池机制的设计与实现

3.1 传统组织方式的局限性

导航系统在运行过程中发生地图平移时，需要跨越网格并重新加载 KIWI 数据。首先，根据定位计算得出包的索引值，去查找该数据包的存储信息。传统设计方式是每次需要加载新数据包时，都向系统申请相应的内存空间，再将地图数据读入到内存中，然后交付给导航系统其他模块使用。当前数据包使用完成之后，其占用的内存空间就会被释放掉。若在嵌入式设备上采用这种方式，由于数据包大小不固定，系统需要耗费大量时间来查找大小合适的内存块；此外频繁地向系统申请与释放内存容易造成大量内存碎片；而将数据读入内存而进行频繁的 I/O 访问还会导致运行速度受到很大影响。在长时间运行后，整个系统的实时性和稳定性将难以得到保证，因此传统的设计方式在嵌入式中的应用有待进一步改进。

3.2 内存池机制的设计思路

内存池(Memory Pool)机制可以有效地解决传统设计中存在的性能缺陷。内存池会事先分配好一个通用内存缓冲区用于存放数据包，可避免频繁的内存申请和释放。在导航系统运行过程中，将某些经常会被使用的地图数据包预留在内存池中，从而使得地图数据不需要直接去 KIWI 文件中读取，也就解决了 I/O 频繁访问的耗时问题。对比传统设计方式，使用内存池机制组织 KIWI 数据，减少地图文件的读取次数，因而可以更好的应用于嵌入式系统。

内存池采用平均划分页面大小的方式，在创建时整个内存池和池内页面的大小都可以被设定。从数据结构来看，内存池包含一个内存池头(Memory Pool Header)，该部分主要通过维护一张索引和数据的全局映射表来管理池中的数据。在 KIWI 数据中，每一个数据包唯一对应一个 6byte 的索引，用于表示该数据包在物理媒体的存储信息，因而这个索引可做为数据包的标识符(pid)。在映射表中还包含了每个页面的控制信息(page control)，而控制信息中又包含页面在池内的偏移量(offset)、载入时间(loadtime)、位计数器(bitcounter)和实际数据大小等信息，如图 2。

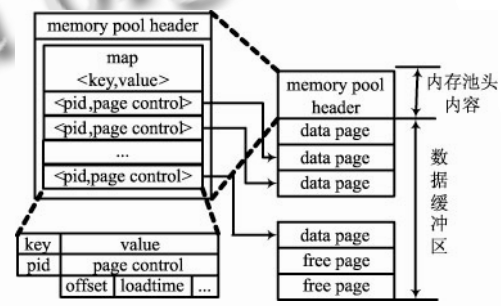


图 2　内存池结构示意简图

3.3 页面置换算法

当发生页面失效时，内存池必须在缓冲区中选择一个页面将其移出内存池，以便为即将调入的页面让出空间，完成内存与磁盘间的交互。因而导航系统的内存池需要选择合适的页面置换算法，进行内存页面的更新管理。操作系统中常见的几种页面置换算法比较如下[2]：

①最优页面置换算法(Optimal)：淘汰未来一段较长的时间内不会被访问的页面，是理想的置换算法，易于描述但难以实现，可以作为比较基准。

②二次机会算法(Second Chance Algorithm,以下简称 SCA)：它是先进先出置换算法（FIFO）的改进，其原理是扫描所有页面，将很少使用并且在内存中的驻留时间较长的页面进行标记，在发生页面失效时将其淘汰，移出内存池。

③老化算法(Aging，以下简称 AGA)：它是最近最久未使用置换算法（LRU）原理的延伸，其原理是创建一个 n 位的页面有效计数器，在每次页面扫描时，更新该页面的相应位的有效计数器，当发生页面失效时，淘汰所有页面中有效计数器最小的页面；

SCA 比较 FIFO 有较大的改进，AGA 是 LRU 的有

效近似算法，这两种算法在操作系统的换页机制中都有很好的应用，因而可以作为内存池管理中的页面置换算法。本文也将对此两种算法进行实现，进而进行效率上的比较分析。

3.4 具体实现

3.4.1 类成员定义

根据图 2 所示的内存池结构图，首先创建三个主要类，内存池类 MemoryPool，内存池管理头类 MemoryPoolHeader 和页面控制信息类 PageControl，如图 3 所示。

MemoryPool 依赖 MemoryPoolHeader，其成员包括一个内存头指针和一个数据指针。主要成员函数是用于加载数据包的 LoadParcel()和设定初始内存池大小和池内页面大小的 SetSize()。

MemoryPoolHeader 依赖 PageControl，其成员有池大小 m_nPoolSize，页面大小 m_nPageSize，当前空闲页面个数 m_nFreePageCount（用于表示当前池内剩余空闲页面数量），空闲数组 m_pbFreePage（用于存储当前内存池中所有页面的空闲或者占用的状态），及控制信息映射表 m_MapPageCtrl（C++标准库 map 容器[3]用于存放 pid 和 PageControl 的关联组合）。主要成员函数为页面置换算法的两种实现方式 SwapParcelSCA()和 SwapParcelAGA()，两者在页面失效时被调用，返回值是被交换出内存池的页面在池中的偏移值。

控制信息类包含成员页面偏移量 m_nPageOffset，载入时间 m_tLoadTime，位计数器 m_bBitCounter 和实际数据大小 m_nDataSize。根据 3.2 节可知，m_tLoadTime 和 m_bBitCounter 分别只在 SCA 算法和 LRU 算法中起作用。

每个内存池只包括一个池管理头，而内存池管理头与控制信息的关系，控制信息的个数就是内存池划分的页面个数。

3.4.2 核心方法实现

本节将对内存池的核心方法 LoadParcel 的实现过程进行描述。在创建内存池时，根据内存池的总大小 m 和每个页面大小 s，计算出页面的个数 n= ⌐m /s ¬（向下取整）。将内存池头的空闲数组（设为 A）的大小初始化为 n，每个元素初始值为 0，用以表示所有数据页面尚处于空闲状态。当内存池 M 申请加载标示符为 pid 的数据包 P 时，当前内存池含有的空闲页面数量 N，POS 是用来保存 P 在 M 中偏移量 offset，S 是内存池管理所使用的页面置换算法。

对于申请加载的数据包 P，首先需要在当前内存池内进行查找是否存在。若存在，则直接返回。若不存在则从文件读取到内存池中。若当前池内无空闲位置，就利用算法 S 移出选定将被置换的数据包，随后

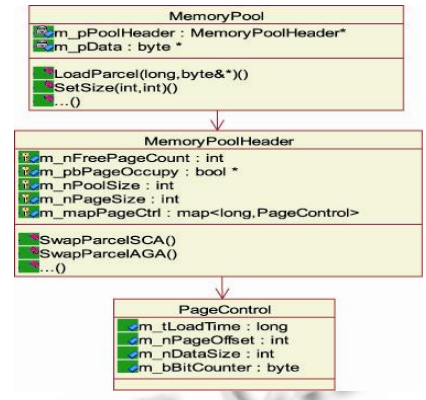


图 3　内存池相关类图

将 P 加载到空闲位置中。LoadParcel 具体工作流程如图 4 所示：

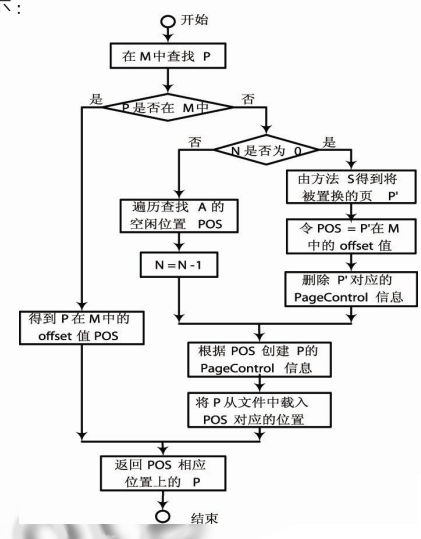


图 4　核心方法 LoadParcel 工作流程图

3.4.3 KIWI 中的实现

基于 KIWI 的内存池机制分别创建索引内存池(Index Pool)和数据内存池(Parcel Pool)，来集中存放相应类型的数据。索引池页面负责存放 KIWI 的块一级索引数据，根据表 1 中的统计结果，将其大小定义为 32K。数据内存池负责存放实际地图数据包，页面大小定义为 128K。内存池在运行过程中始终保证有适量的空闲页面。

内存池加载数据包的工作对外隐藏了查找数据包、添加包、删除包等实现细节，在加载工作完成后，此时导航系统传入的数据指针指向了内存池中 POS 偏移位置处的数据，即实际地图数据包的第一个字节，导航系统的其他模块从而可以根据 KIWI 格式规范对数据进行进一步解析。

4　实验验证

为了验证基于内存池技术的数据组织方式的正确性，进行了验证实验，本实验以二次机会算法和老化算法为基础，依据中国大陆数据进行统计分析，所使用的操作系统平台是 PocketPC 2003，内存大小为 64Mb，CPU 的主频是 300MHZ，扩展卡为 4G，KIWI 数据量大小为 1.44Gb。

以重庆市地图数据为实例，将重庆市重庆大学设为实验起点，实验终点为重庆市朝天门码头。利用控制程序，在起点与终点之间进行多次（共 100 次）地图平移，每次移动的步幅为地图数据包区域的 1/4。实验过程中分别实时记录下传统方式(不使用内存池)、SCA、AGA 下的数据包加载时间和查找命中率的平均值。将所得的原始数据进行整理和对比，可得如下所示的结果图。

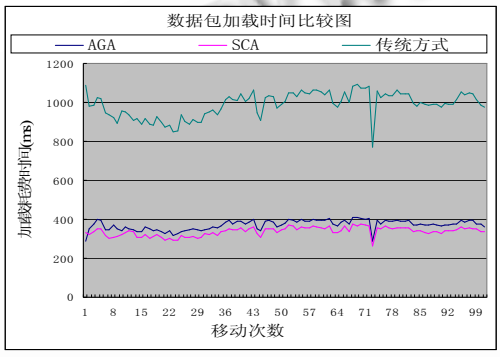


图 5　数据包加载时间比较

从图 5 中计算得出，传统方式的平均耗费时间为 988ms，而 AGA 和 SCA 的平均耗费时间取整后分别为 371ms 和 337ms，两种算法大约可以减少数据包加载时间分别为 62%和 65%。从图 6 中看出，而对于两种页面置换算法来说，AGA 具有更高的命中率，它高出 SCA 的数据页的命中率是 6%~8%，但是 SCA 中数据包加载耗时比 AGA 平均少 34ms，能够更好的满足了导航系统的实时性需求。

5　结语

在统计分析 KIWI 数据存储特性之后，根据这一特点，制定有针对性的内存管理机制。实验表明，在嵌入式环境下应用内存池技术对 KIWI 数据索引、地图数据进行组织是非常有效的。该方法使得导航系统在 KIWI 数据的解析过程中能够更少的占用内存空间，并能减少 I/O 的读写次数，有效提高了数据包的读取效率，从而更好的满足了嵌入式导航设备的地图数据组织和管理的需要。可以认为，内存池技术不仅适用于 KIWI 数据的组织管理，也适合于灵活的、基于页面的、带缓冲的文件共享访问。

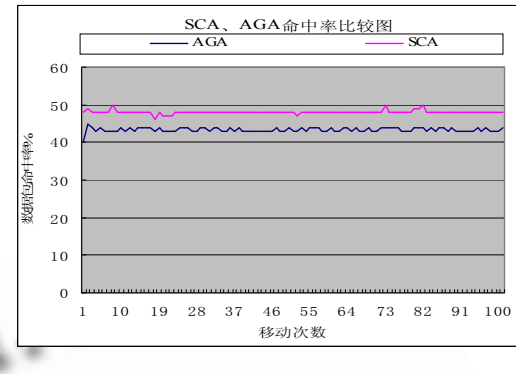


图 6　SCA、AGA 命中率比较

参考文献

1 徐敬海,李清泉,宋莺.基于 KIWI 数据格式的地图显示研究.武汉大学学报, 2005,30(10):865－869.

2 ANDREW S.T. Modern Operating System:2nd ed.北京:机械工业出版社, 2002:189－246.

3 侯捷.STL 源码剖析.武汉:华中科技大学出版社, 2002:237.

4 KARL W, STEPHAN H, JOSE P. Improving Performance by Cache Driven Memory Management: IEEE High-Performance Computer Architecture Symposium Proceedings, 1995: 234－242.

5 冯宏华,徐莹,程远,等.C++应用程序性能优化:北京:电子工业出版社,2007:183－203.

6 刘有源,黄新明,陈定方,等.KIWI 数据格式在导航系统中的应用研究.计算机辅助设计与图形学学报,2003,15(4):503－508.

7 武雪玲,李清泉,李必军等.导航电子地图数据格式 KIWI 中主地图数据的提取.测绘信息与工程,2003,31(4):13－15.

8 王斌,李清泉,李汉武等.由 KIWI 数据格式到 MapInfo 数据格式的转换研究.测绘信息与工程,2005,30(10):45－46.

9 郭丙轩,张京莉,张志超.基于内存池的空间数据调度算法.计算机工程, 2008,34(3):63－64.

10 王明路,王希敏,王哲.嵌入式系统中池式内存分配方法的分析.计算机与数字工程, 2008, 36:57－61.