ObjectCore 的设计与实现技术*

郭 江

(中国科学院软件研究所 北京 100080)

摘要 对象存储技术是对象管理系统 OMS(object management system)的核心,已经研究了许多年.文章主要对 ObjectCore——软件过程环境 ISPE(integrated software process environment)中,对象管理系统的存储部分进行了讨论,并对对象管理系统中,对象存储技术的几个重要方面进行了分析.

关键词 面向对象,对象存储,对象寻址.

中图法分类号 TP311

目前,已经有许多可以使用的面向对象数据库系统(OODBMS),如 ConceptBase,ENCORE,Orion,POSTGRES,Rose,O2,ObjectStore,Ontos,Iris,OpenODB,GemStone,Versant 等.但是这些 OODBMS 不适合作为软件过程工程环境 ISPE(integrated software process environment)的对象存储技术,它们过于庞大和复杂.为此,用我们设计和实现的软件过程工程环境(ISPE)[1]的一个核心部分 ObjectCore 进行对象的存储管理,用于支持 ISPE 中对象管理系统(OMS)的实现.

考虑到目前操作系统所支持的虚拟存储(虚存)体系结构对 ObjectCore 中的对象存储的限制,在 ObjectCore 的对象存储中引入了内存模型多级存储的概念[2,3],也就是将巨大的磁盘数据库集成到机器的虚拟地址中.这种方法将所有存储(不只是主存)都交由虚存管理系统来管理和控制,用户程序不需执行任何附加磁盘存储的输入/输出.实际上是将 ObjectCore 中的对象存储组织成持久存储的逻辑对象,一直存储到用户进行删除为止.

1 ObjectCore 中的对象寻址

ObjectCore 中的对象使用 16 位字节的指针标识符来寻址,指针由 8 位字节的对象地址及有关对象状态、授权特性和对象类型的附加信息组成.对象的存储通过加标记来完成,即用标记位来决定一个 16 位字节序列是否是一个指针.如果是指针,就施加限制,以防指针信息的破坏和覆盖,这种指针的分配如图 1 所示.

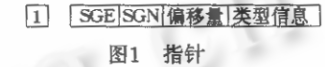

图1 指针

指针的地址部分由 3 个域组成:

偏移量:一个段组内 24 位字节的偏移.

SGN:24 位字节的段组号.

SGE:16 位字节的段组扩充部分.

对象不可避免地要引起段组的重用,在删除对象时(机器不使用垃圾收集),与之相关的段组就可以重用.但是,在重用时,新对象的地址可以赋给一个不同段组的扩充部分,同时该扩充部分也写入段组的头部.在去掉指针的指向时,就要将指针的扩充域和段组头部中的内容进行比较,如果不同,就表明指针指向以前占据虚地址空间部分的对象,发出一条错误信息.

如果使用 48 位的虚存地址来获得物理地址,那么 48 位的地址可以看作是一个 39 位的页地址和一个 9 位的偏移量,其中页大小为 512 字节.使用这么小的页是很有讲究的.这种小页系统在转换一个小对象进入内存时,所需的额外空间会比大页系统要少很多.

图2 虚地址

在这里要考虑的另一种思想是在用户级内存寻址机制中采用分布式网络的虚存地址,这种地址类似于系统的变体设计.虚地址的分配如图 2 所示.

这里的虚拟地址分成 4 个部分,即节点、卷、AS(地址空间)和偏移量.

---

* 作者郭江,1966 年生,博士,主要研究领域为软件工程,软件工程环境及数据库.

本文通讯联系人,郭江,北京 100080,中国科学院软件研究所

本文 1997-05-26 收到原稿,1997-09-10 收到修改稿

© 中国科学院软件研究所 http://www.jos.org.cn

• 虚存节点号表明了所需的物理内存的机器网络地址,有32位长.这个取值范围可能不够大,特别是考虑到某些网络寻址系统现在利用48位的地址域,就更显得小了.

• 磁盘可以由1个或多个卷号来确定,如果多个卷号指向1个磁盘,那么每个号就表示磁盘中的一个分区,该域是32位长.但是,带有节点号的这个域仅由6位来表示,这对实际的网络化系统可能是不够用的.

• ObjectCore中的对象存储分成了多个独立的"块",其中每个有一个唯一的标识符.这些块就是人们所熟知的"地址空间",由地址空间号来确定,其中每个空间分成固定大小的页.注意,这些地址空间号不能重用,甚至在内存块被删除之后也不能重用.

• 为了访问每个地址空间中的数据信息,要提供一个偏移量,以便地址加偏移量就能确定一个唯一的内存位置.

以这样一种严格的方式对虚地址进行分区,虚地址就可以分成段(特别是虚地址不能重用).但是,由于虚地址范围比物理内存地址范围要大得多(甚至在使用率最大的情况下,虚地址在系统的使用周期中也不会用完),因而就不会构成严重的问题.要注意的是,虚地址并不是由执行任务来直接使用的,而是通过对象标识来访问对象(或段).这些标识符指向对象的段方法,并形成对象保护的基础.

2 地址译码

在地址空间中的页数非常多时,如2^{39},传统的层次结构页表就不实用了,这就要考虑映射地址空间层次结构可能产生的问题.一个512字节的页可以容纳每个为64位的页描述符64个,因而要映射全部地址空间就需要2^{38}个页.当然,不需要全部映射,只需映射当前驻留的页,但是一个对象地址空间可能零散使用,每个对象占据地址空间为16兆字节的段组.这就需要三级映射表,以便最小的驻留对象在映射信息时可以连接到3页的框架上.除了空间限制以外,还有速度方面的考虑,将逻辑地址转换成物理地址可能涉及到6、7级的间址,为了避免这些问题,就要开发独立于系统的转换页表的表示.[4,5]转换页表中包含了每个内存物理页的记录,这些内存物理页在所包含的虚页上建立索引.

3 段方法

使用段方法是为了让所有段都经过段方法来调用,每个段分成若干部分.段的第1部分是"段控制部分",这部分包含了段的实际大小,并描述了有关其内容的细节.实际的数据部分称为"信息部分",以段控制数据为基础,进行信息存储和检索.段的最后一部分是"段方法"部分,在这个区域中,可以放置对其他段的调用,这样,在系统中就可以表示任意复杂的图结构.

这种固定的段分配方式有一些限制,同时增加了面向对象语言实现的复杂性.在考虑类型继承时,这个问题更加明显.在典型的面向对象语言中,对继承的支持是通过当前的段类型和新段(包含了父类型信息)的组合而来的,组合的过程是通过在旧段末尾标记新段来实现的.但这种方案有时行不通,因为不能任意组合数据和段方法.继承可以使用来自"高层"段的某种间址来实现,但这明显增加了执行初始化间址时的代价.在调用段中包含的信息时,所需要的只是段地址(而所有有关段的其他信息都是自包含的).考虑到效率的问题,就提出了"段方法变量",在调用段时使用.用这样一个变量调用元素的过程如图3所描述.

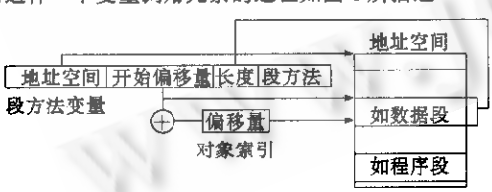

图3 对象映射

通过访问一个段,并从该段装入一个段方法,就可以遍历所有可访问的数据结构.段方法的装入必须受到用户的有意保护,因而要为这种功能提供一个特殊的函数,并用另一个函数来执行段方法的修改.所有寻址的根都存在一个特殊的段中(每个进程一个),称为"基表",可以使用一个附加的函数来访问该表的内容.注意,这些函数会明显影响函数集的正交性,因而增加了系统的复杂性.

这样,构造段方法就能使所有段地址驻留在当前的地址空间中.[6]限制数据结构的范围要有一定代价,会影响到节省的内存空间以及在垃圾收集期间所需的工作量,因而最好将相关的对象(如段)组在一个单个的地址空间中.但是,当前地址空间外的段可以使用间址段方法来调用,而它们本身则指向全部的虚地址.

4 分 页

每个分配的地址空间必须整个存放在一个卷中(尽管一个卷可能容纳多个地址空间).为了有效地进行分区,每个地址空间需要一个磁盘页表,而且要放在它所描述的地址空间的保护区中.每个卷有一个根页,指向该卷目录,而卷目

© 中国科学院软件研究所 http://www.jos.org.cn

录又指向地址空间的分页表.

每个节点包含一个地址转换单元(ATU),它将虚存地址映射到物理内存位置.每个 ATU 作为一个转换页表来实现,其大小正比于物理内存映射的大小.如果需要的页在物理内存中不存在,就产生一个页出错信息.在这一点上有两种可能的情况:一种是出错的页在当前节点中,在这种情况下,页可以从磁盘上卸下来;另一种是页存在于其他节点中,如局部节点请求网络节点页所产生的结果.

对于局部就可解决的地址错误,所需页的磁盘地址空间可以从其基本页表中读入.该表可以用 12～27 位的虚地址偏移量的值来索引,因而包含 16 位磁盘地址的 2^{16}个入口.这个基本页表也可能不存在,那么就会产生一个地址错.这里产生的结果是访问地址空间的二级页表,包含有 32 个磁盘地址入口.因此,每个入口相应于基本页表($(2^{16}*2)/4KBytes=32$)的一个页,基本页表所需的部分就可以从磁盘装入.如果地址空间的二级页表不在,那么这个页的磁盘地址空间就可以通过访问根页表来找到,而且它总可以在每卷地址空间的起始处找到.

这种磁盘地址树的分配对于小于 256 个内存页(每页包含 4K 字节)的地址空间而言,整个树在地址空间的第 1 页就可以找到.实际上,对于小于 3.6K 字节的地址空间来说,整个地址空间可以在一个页中找到(因此解决了单个磁盘访问中的出错问题).[7]

5　内存分配

为了在一个永久性存储中实现对象持久性,就涉及到所谓的"被动空间"和"主动空间"."主动空间"由虚地址空间构成,这样,计算就可以在内部执行.如果调用一个不存在于主动空间的持久对象,那么就可以由一些内部管理机制从被动空间传送过来.因而主动存储的功能就以看作是被动存储的缓冲区,由于传送机制是自动和透明的,因而保持了单级对象存储的策略.

另外,在一个细粒度系统中,很多临时对象在创建后很快就会释放掉(如临时局部变量).如果每个对象都需要一个持久性标识符(PID)(如唯一标识符(UID)或很长的唯一永久性标识符(ID)),那么这将成为系统的一个很严重的瓶颈.为了减少这种额外的开销,每个分配在主动空间中的 PID 是由对象短名(OSN)来确定的.这些 OSN 为 24 位长(而 PID 为 48 位),因而允许有 $16*10^9$ 个对象同时驻留在主动空间中.由于 OSN 比 32 位小,因而可以在一个存储周期中存取.

在这里并不直接解释对象 UID,如果一个对象目前不在主动空间中,那么就通知"对象过滤器"管理系统,执行被动 UID 到主动 OSN 的转换是管理系统的责任.作为一个二级管理系统,全局对象管理系统用于解决对象请求,它要求在节点之间传送对象.

在内存分配的设计方案中,要考虑赋予每个进程一个 16 位的进程 ID 号,该号码用于为每个系统环境选择一个环境说明(因而给每个进程提供了对象存储的不同视图).每个环境说明由 4 个入口组成:

- 当前环境.这包含了当前环境的段方法,包括临时变量.
- 当前根对象.它指向了当前环境操作(称为一个类型特定操作(TSO))的用户定义对象.
- 当前对象类型,它允许当前根对象访问所有其他 TSO.
- 参数对象,它可以访问调用参数.这将允许传递数据和段方法,因而可以由值或名来模拟调用.一个代码调用由下面的形式组成.

INVOKE (target root object,TSO-index,parameter).

其中目标根对象由段方法索引(即对象的局部进程名)在当前环境、当前根对象、当前类型对象或当前参数对象中确定.这就需要为进程创建一个新的环境说明(包括环境自己的段方法表)以及在创建环境时涉及到的 7 阶段机制.

在通过段方法索引来访问一个对象时,环境说明的入口起始位置是由索引的高位来确定的,它选择使用一个特定的段方法表,而该表则由索引剩余的低位来确定,由它为问题中的对象选择段方法.因此,每个进程的段方法有自己的名字,这就是人们所熟知的"局部进程对象名".[8]这种映射机制如图 4 所示,在图中产生的段方法就可以用于访问有关的对象.

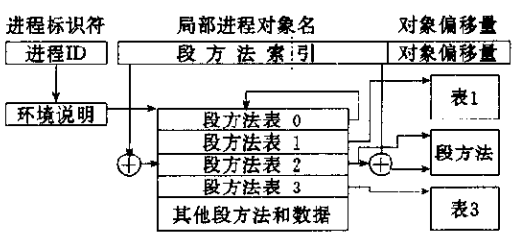

图4　段方法路径的环境

对象 OSN 可以从所选择的段方法中抽取出来:OSN 的高 20 位通过两级表示来确定对象驻留的页,如果对象

© 中国科学院软件研究所 http://www.jos.org.cn

平均大小为300字节,那么设计者就要将映射进程限制到每页为4K,包含16个对象.因此,OSN的最后4位用于选择每页中所需的对象头.另外,这个头包含了在访问时进行动态类型和访问权限检查的信息,因此,允许对象上锁和共享访问这样的机制.当然,这种对象调用的方法减轻了对集中式映射表的需求(会造成大量数据的管理任务).

所选择的对象实际上被分成了两部分:数据部分和非数据(段方法)部分.在段方法部分中,根据对象的UID,为该对象链接到TSO.这包括一些系统在表达复杂数据结构所需的相关段方法.

分成两部分是想保护对象不受无意或恶意的访问.这里,对象的段方法部分只能由负的地址偏移量访问,需要使用特殊的指针.很明显,这种对象分配方案在考虑类型继承时会有一定的限制.

维护和遍历多级表在所有类似的对象管理实现中都是必要的,由于这个原因,引入了ATC,ATC是一个地址译码缓冲器(ATC).这种ATC包含了当前访问对象的物理地址以及其他有关信息,这些信息可以用于由附加逻辑来验证访问权限和大小约束.它包含了4 096个最常用的地址译码.ATC又分为16个部分,256个入口,并将最近执行的16个进程分配到一个部分中.这样,估计的命中率将高于95%.

6 结 论

在目前的实现技术和理想情况之间还有很大差别,这是由OODBMS的不同观点以及何种特征可以有效实现等问题所引起的.在讨论的问题中,有3点非常重要:

- 段技术的使用可以在内存中创建可变大小的内存块.
- 保护段不受无意或恶意的修改和访问.
- 给系统提供足够大的地址空间,这样,在系统的生命周期中不需要重用地址.

参考文献

1 郭江,黄涛,廖越虹.软件过程环境的设计与实现.软件学报,1997,8(12):928~936
(Guo Jiang, Huang Tao, Liao Yue-hong. The design and implementation of integrated software process environment. Journal of Software, 1997,8(12):928~936)

2 Soltis G. Design of a small business data processing system. IEEE Computer, 1977,10(9):9~14

3 Houdek M. IBM System/38 support for capability-based addressing. In: Proceedings of the 8th Symposium on Computer Architecture. Los Alamitos, California: IEEE Computer Society Press, May 1981

4 Abramson D. Hardware management of a large virtual memory. In: Proceedings of the 4th Australian Computer Science Conference. Queensland, Australia: University of Queensland Press, May 1981

5 Edwards D. The MU6-G: a new design to achieve mainframe performance from a mini sized computer. In: Proceedings of the 7th Annual Symposium on Computer Architecture. Los Alamitos, California: IEEE Computer Society Press, May 1980

6 Keedy J. An implementation of capabilities without a central mapping table. In: Proceedings of the 17th Annual Hawaii International Conference on System Sciences. Los Alamitos, California: IEEE Computer Society Press, January 1984

7 Wulf W. HYDRA/C.mmp: an experimental system. New York: McGraw-Hill Press, 1981

8 Kaiser J. An object-oriented architecture to support system reliability and security. In: Proceedings of the International Workshop on Computer Architecture to Support Security and Persistence of Information. Orlando: Morgan Kaufmann Publishers Inc., May 1990

Design and Implementation of ObjectCore

GUO Jiang

(*Institute of Software The Chinese Academy of Sciences Beijing* 100080)

Abstract The technologies of object store are the cores of OMS (object management system). Many persons have done a lot of efforts. The ObjectCore—— the object store system of the ISPE (integrated software process environment) object management system is discussed in this paper. And some important aspects of technologies of object store of OMS are discussed as well.

Key words Object oriented, object store, object addressing.

© 中国科学院软件研究所 http://www.jos.org.cn