基于微核心的文件系统服务器虚拟文件子系统*

张 斌 孙玉方

(中国科学院软件研究所 北京 100080)

摘要 S5FS 与 UFS 是两个互不兼容的 UNIX 文件系统. 基于 Vnode 的文件系统体系结构使得一个文件系统可同时支持多种文件系统类型,如 S5FS 和 UFS. 本文介绍了 Vnode 界面和国家"八五"高科技攻关项目中一个虚拟文件子系统的实现.

关键词 Vnode,虚拟文件系统,微核心,UNIX,客户/服务器.

中图法分类号 TP316

操作系统最主要的功能之一是提供一个高效、可靠的文件系统以支持对文件的访问. 随着操作系统的不断发展,出现了很多文件系统类型,如 UFS,S5FS,HPFS 及 NTFS. 在兼容性已成为操作系统设计的一个主要目标的今天,开发一个新的操作系统,或是为已有的操作系统推出一个新版本,都必须考虑使其支持多种现有的文件系统类型. 然而,各种不同的文件系统类型之间,从体系结构到实现采用的算法及策略等各方面都存在很大的差异. 这种差异给构造一个可支持所有文件系统类型的公共文件系统造成了很大困难,使得在增加文件系统代码的同时,还要付出效率下降的代价. 基于 Vnode 的系统结构很好地解决了在一个文件系统中支持多个 UNIX 类文件系统(UFS,S5FS)的问题,同时它也可支持其它具有树型目录结构的文件系统(如 FAT),这样的系统结构具有良好的可扩充性和兼容性.[1]

作者在参加"八五"国家科技攻关项目"国产系统软件开发"中,与同事一起在微内核上实现了一个 UNIX 系统. 该系统的 UNIX 语义由多个互相独立的服务器及相应的透明模拟库实现. 这些服务器是运行于非特权态的系统任务. 文件系统的开发目标是实现一个 UNIX 文件系统服务器 FSS(file system server),要求支持 UFS 和 S5FS,符合 POSIX 语义,并便于今后的扩充,如增加对网络文件系统(NFS)的支持. 为满足设计目标的要求,文件服务器采用了基于 Vnode 的体系结构. 基于 Vnode 的文件系统可划分为两部分,即与文件系统实现无关的虚拟文件系统 VFS(virtual file system)和与文件系统实现相关的部分,后者以实现各种文件系统类型命名,如 UFS,S5FS. 本文先简要介绍 Vnode 界面,然后阐述 VFS 的

---

* 本文研究得到国家"八五"攻关项目基金资助.作者张斌,1967 年生,工程师,主要研究领域为软件工程,操作系统. 孙玉方,1947 年生,研究员,博士导师,主要研究领域为计算机软件,中文信息处理,信息处理标准.

本文通讯联系人:孙玉方,北京 100080,中国科学院软件研究所

本文 1997-04-23 收到修改稿

设计和功能模块的划分，并对路经名翻译过程和特别文件子系统作较详细说明.

1 Vnode 界面

Vnode 界面的产生是为了抽象出文件系统操作的共性，以便在一个操作系统中同时支持多种文件系统类型，最初主要是为了支持现在已经成为 UNIX 环境下分布式文件系统事实标准的网络文件系统 NFS. 实现传统 UNIX 文件系统的最主要数据结构是索引结点(Inode)和安装表(Mount Table)，因此选择在 inode 和 mount(此处指特定文件系统的安装表，与下文 VFS 中的 mount 结构不同)结构这一层次上，根据与特定文件系统的相关性，把UNIX文件系统抽象为两部分：与文件系统类型实现无关的部分和与文件系统类型实现相关的部分. 用面向对象的方法描述这种抽象，就产生了 Vnode 界面. 在这个界面下，文件系统由 VFS 层和文件系统类型实现相关层构成. VFS 中的主要数据对象是 vnode 和 mount 结构，它们是文件和文件系统的抽象；在我们实现的系统中即是 ufs_inode/s5_inode 和 ufs_mount/s5_mount的抽象. Vnode 和 mount 结构的主要数据域如下

```
enum vtype{VNODE, VREG, VDIR, VBLK, VCHR, VLNK, VSOCK, VFIFO, VBAD}
struct vnode {
    long                v_flag;          /* 标志域 */
    long                v_refcount;      /* Vnode 引用计数 */
    struct mount        *v_mount;        /* 指向所属文件系统 mount 结构 */
    struct vnodeops     *v_op;           /* 指向操作私有数据的函数入口 */
    struct mount        *v_mounted;      /* 指向以该 vnode 作安装点的文件系统 */
    enum vtype          v_type;          /* 文件类型 */
    ...
    struct mutex        v_lock;          /* 互斥锁 */
    char                v_data[1];       /* 特定文件系统的私有数据 */
}
struct mount {
    struct mount        *m_next;         /* mount 链 */
    struct mount        *m_previ
    struct vfsops       *m_op;           /* 指向操作私有数据的函数入口 */
    struct vnode        *m_covered;      /* 指向该文件系统安装点头 */
    ...
    struct mutex        m_lock;          /* 互斥锁 */
    char                m_data[1];       /* 特定文件系统私有数据 */
}
```

vnode 和 mount 结构中最后一项均为可扩展的私有数据域，它们代表了文件系统实现相关层的数据对象，即 ufs_inode 和 ufs_mount 等结构. 这些数据结构是某种文件系统类型对文件和文件系统的特定描述，一般包含了数据块在盘上分布的物理特征，随文件系统类型的不同而存在差异. 除了私有数据域外，vnode 和 mount 结构均设置一个指针指向由函数入口地址构成的向量，这些函数完成对私有数据对象的操作，大多数函数与系统调用相对应，以某种特定的、文件系统相关的方式实现一种操作语义. 两组入口向量如下

```
struct    vfsops {                    struct    vnodeops {
    int   (* vfs_mount)( );               int   (* vn_lookup)( );
    int   (* vfs_unmount)( );             int   (* vn_open)( );
    int   (* vfs_root)( );                int   (* vn_read)( );
    int   (* vfs_sync)( );                int   (* vn_write)( );
    ...                                   ...
    }                                     }
```

以上这种对外隐藏私有数据和操作的面向对象的设计使文件系统的两个层次间获得了一个清晰界面,VFS 之下要增加新的文件系统类型时变得较为容易,可以方便地加入或摘除一种文件子系统.引入 VFS 后,整个系统中文件系统对外的接口由 VFS 提供.Vnode 界面中主要数据结构相互间关系如图 1 所示.

对于每个活动的文件或文件子系统,仅有唯一的 vnode 或 mount 结构与之对应.这样,在 VFS 中,对文件的操作转化为对 vnode 的操作,对文件系统的操作转化为对 mount 结构的操作.当系统中存在多个活动的文件子系统时,描述它们的 mount 结构组成一个双向链表,文件系统中的全局变量 rootfs 指向这个双向链的头.第 1 个 mount 结构总是代表缺省的文件系统,它在系统初启时自动安装,系统运行期间不可拆卸.每个 vnode 结构都含有一个指向其所属的文件子系统 mount 结构的指针,属于一个文件系统的所有 vnode 构成一个双向链表.

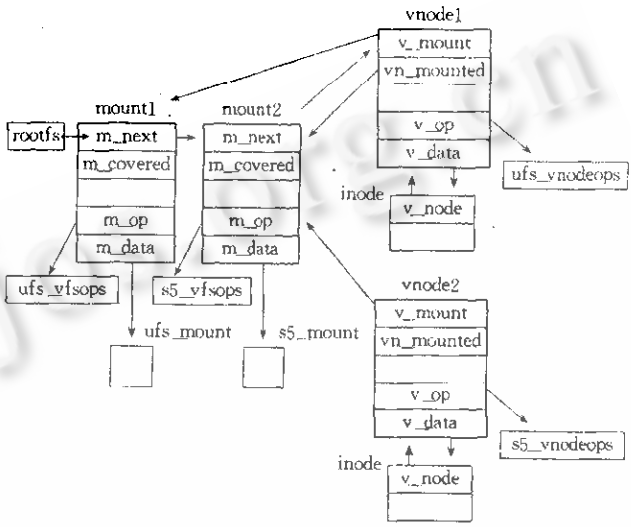

图1 Vnode界面的主要数据结构及相互关系

图 1 中 mount1,mount2 分别表示系统的根文件系统 UFS 和一个安装在其上的 S5FS,vnode1 是 UFS 上的一个目录节点,同时作为 S5FS 的安装点;vnode2 是 S5FS 上的一个文件.不同文件系统 mount 结构的 m_op 域所指向的函数入口向量不同;类似地,vnode 结构的 v_op 域也不同.安装一个文件系统时,VFS 为其分配并初始化 mount 结构;而在首次访问一个文件时,由文件系统类型相关层,如 ufs_iget()函数,分配并初始化 vnode 结构.之所以采用这种处理方式,是因为 VFS 可以从 POSIX 语义所规定的 mount()系统调用参数中获得足够的信息来初始化 mount 结构,但初始化 vnode 所需的一些信息必须从文件的inode 中获取.如打开一个已存在的文件,只需给出文件名和访问方式两个参数,但这个文件可以代表普通文件,也可以代表一个有名管道或设备,有关文件类型的信息保存在 inode 中,在读出文件的 inode 之前,VFS 无法知道文件的具体类型.

一个系统调用或文件系统内部的操作如路径名翻译,一般都要由 VFS 和文件系统类型相关层协作完成,即 VFS 需要调用文件系统类型相关的函数来完成对私有数据,如具有特定格式的目录的访问.为使 VFS 层的代码与支持的文件系统类型无关,在具体实现时,这些调用采用了带参数宏调用的形式,参数之一为 vnode 指针.

2　VFS 的模块设计和功能

在微内核上实现一个多服务器 UNIX 系统,软件采用客户/服务器结构.其中 FSS 既是透明模拟库的服务器,又是别的系统服务器,如设备服务器的客户.在我们实现的 FSS 中,

它与外部及内部模块之间的调用关系可简单地用图2表示.

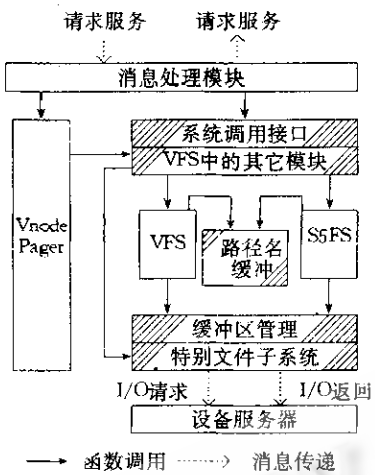

图2 VFS在文件服务器中的位置

为说明VFS在FSS中所起的作用,先简单描述一下FSS的工作方式:FSS接收到透明模拟库发送来的请求时,消息处理模块对消息进行解包,根据消息号判断请求类型,再调用VFS中相应的接口函数处理这次服务请求;函数返回后,消息处理模块把结果打包,送回给透明模拟库.如果FSS接收到核心发来的请求页或写页请求,则由Vnode_Pager(缺页管理任务)为这次请求进行服务.VFS同样为Vnode_Pager提供了接口函数以完成页的读写操作.

下面介绍FSS中的VFS各模块的功能与实现方法,重点介绍与传统UNIX系统有很大差别的路径名翻译过程和特别文件子系统.

公共子例程模块.实现VFS中最基本的公用操作:如增减vnode引用计数函数等.

路径名缓冲模块.路径名翻译是FSS进行的最频繁的操作,其效率的高低直接影响整个系统的性能.为提供高效的路径名翻译操作,在VFS中提供路径名缓冲功能.

vnode操作模块.大多数系统调用分两步完成:首先将路径名或文件描述符转换为vnode,然后对vnode进行操作.因此VFS提供一组以vnode为对象的内部调用接口函数,设置这些接口函数使VFS具有更好的模块化结构,同时为文件系统的一些内部操作提供方便.

文件记录锁模块.实现文件记录锁功能,以支持fcntl( )系统调用中对文件的加锁命令选项.与锁的语义相关的数据结构全部放在VFS层.

文件打开表管理模块.维护用户文件打开表和系统文件打开表,主要是文件表的初始化、文件表项的分配与释放.

缓冲区管理模块.我们目前实现的是一个本地文件系统(Local File System),缓冲区作为文件系统物理磁盘块的cache.

系统调用接口模块.VFS的对外接口函数模块,该模块提供符合POSIX语义的系统调用界面,攻关项目中我们实现了近50条针对文件及文件系统操作的系统调用.

路径名翻译模块.路径名翻译模块完成由给定的路径名到vnode的翻译.这个模块涉及的主要数据结构除mount和vnode外,还包括nameidata.nameidata结构封装了路径名翻译操作所需的全部参数,包括路径名字符串、目录指针、credential结构指针、本次翻译目的标志和一个指向路径名代表的vnode指针等域.路径名翻译操作在VFS层完成的主要工作是处理特殊情况(如退化的路径名)及实现安装点的自动跨越.由于路径名翻译的流程很好地体现VFS与文件系统类型相关层之间的关系,在此给出LOOKUP()函数的简要流程.

算法中使用的宏定义为

```
#define    VOP_LOOKUP(v, n, r)
           (r)=(*((v)->v_op->vn_lookup))((v)(n))
lookup (struct nameidata *ndp)
```

```
        根据翻译目的标志域初始化 nameidata 结构;
        获取起始目录 vnode;
dirloop:
        取下一个路径名分量;
        处理路径名为空的退化情况;
        if(分量为"..",且工作目录为某一被安装文件系统的根){
                获取工作目录所属文件系统的 mount 结构;
                获取 mount 结构对应的安装点;
                工作目录置为安装点;
        }
        VOP_LOOKUP(dp,ndp,error); /* 文件系统类型相关的例程 */
        if(是符号链接){
            重置 ndp 中路径名 buffer;
            goto dirloop;
        }
        if(是安装点){
            获取被安装的文件系统的 mount 结构;
            由 mount 结构获取该文件系统的根结点;
            结果 vnode 设置为被安装文件系统的根;
            }
        if(还有路径名分量)
            goto dirloop;
        返回 vnode 指针; /* 在 ndp->ni_vp 中 */
```

特别文件系统模块. 特别文件系统 SPECFS(special file system)并不是一个真正意义上的文件系统. 它不可安装(mount( )),不能通过系统调用(如 statfs( ))获取其状态信息,也没有与之对应的设备. 引入特别文件系统是出于对系统进行模块化设计的考虑.

由于历史的原因,UNIX 系统把设备当作文件管理[2],因此设备与文件系统密切相关. 在构造支持多种文件系统类型的 FSS 时,可以把传统 UNIX 文件系统中与设备有关的功能提取出来形成一个新的子系统 specfs. specfs 由一组函数 spec_vnodeops 构成,它们提供对设备文件的基本操作功能,并负责与设备驱动程序(通过与设备服务器的 rpc)接口. 不同类型的文件系统有自己的 specfs,如 ufs_specfs,但其中的函数基本都直接取自 specfs,少数函数在调用 specfs 中对应函数的基础上增加为数不多的语句,以体现特定的实现方式. 引入 specfs 后,具体的文件系统一般不必再解释对设备特别文件的操作,这样就进一步增强了系统的模块化结构,文件系统逻辑功能的划分也更加清晰.

首次访问一个设备特别文件时,先进行路径名翻译以获得这个文件的 vnode. 当文件系统类型相关的例程从盘上读出文件的 inode 后,对其类型进行判断. 若是字符设备特别文件或块设备特别文件,首先将文件对应 vnode 的 v_op 域由默认值设置为 &ufsspec_vnodeops(假定这是一个 UFS 中的设备特别文件),即对设备特别文件进行操作的函数入口组成的向量地址. 这种替换使以后对设备特别文件的操作改向到了设备特别文件系统中.

3　结束语

"COSIX V2.0"文件系统开发已经过国家主管部门的技术鉴定和验收,表明 FSS 的设计包括 VFS 的设计基本上是正确可行的. 但要使之成为一个实用系统,还需进行许多完善、优化工作. 就 VFS 而言,我们认为还可以进行如下一些考虑.

(1) 将尽可能多的功能,如读写系统调用及 VFS 中用户文件打开表的管理等放到透明模拟库中实现,减少对服务器的访问以提高系统效率. 要做到这一点,首先需要解决系统文

件打开表中文件读写偏移量在透明模拟库和FSS之间共享访问的同步问题.但由于目前进程服务器对用户进程不提供多线程支持,一时还没有找到解决这个问题的较好方法.

读写操作放到透明模拟库中实现,还有其它技术困难.如强制锁功能的实现.有关文件加锁状态的信息只能依附于VFS的vnode结构,透明模拟库无法访问到这些数据.一个办法是透明模拟库仍然向FSS请求服务,在FSS中由VFS完成强制锁的检查,如果当前不允许进行本次读写操作,则服务线程睡眠直至被唤醒;最终透明模拟库收到FSS发来的允许操作的应答消息之后,把映射到透明模拟库空间的文件数据传送给用户.但是这个方案似乎与尽可能减少透明模拟库向FSS发消息以提高系统性能的初衷不一致.

(2) VFS中有一个缓冲区管理模块,这是继承了传统UNIX系统的特征.引入缓冲区的目的在于改善系统读写磁盘文件的性能.在把读写操作放到透明模拟库中采用虚存映射方式实现后,这个缓冲区管理模块可以消除.因为微内核是把内存当作外存的高速缓冲来管理,已经实现了数据缓冲池功能.

FSS中的缓冲区导致Vnode Pager使用效率较低的接口memory_object_data_provided()向内核返回数据,这个接口要求内核进行一次数据拷贝.当消除FSS中的缓冲区管理模块后,Vnode Pager使用新的接口memory_object_data_supply()向内核返回数据,该接口采用了偷页优化以消除内核的数据拷贝.在偷页优化过程中,虚存管理系统(VM)把物理页从文件服务器中移走,并将其直接链入存储对象(Memory_object)[3]的物理页表.

消除缓冲区需要考虑的另一个问题是:把目录文件也统一当作存储对象管理,采用虚存映射的方式访问目录.

参考文献

1 Berny Goodheart, James Cox. The magic garden explained. Prentice-Hall, Inc., 1994.

2 Maurice J Bach. The design of the UNIX operating system. Prentice-Hall, Inc., 1986.

3 陈华瑛. MACH核心分析. 计算机研究与发展,1994,31(9):1~6.

A VIRTUAL FILE SYSTEM OF FILE SYSTEM SERVER ON MICROKERNEL

ZHANG Bin SUN Yufang

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

Abstract Both S5FS and UFS are UNIX file system implementations, and not compatible with each other. A vnode-based file system can accommodate multiple file systems such as S5FS and UFS. This paper introduces the vnode interface and an implementation of VFS(virtual file system).

Key words Vnode, virtual file system, microkernel, UNIX, client/server.

Class number TP316