opensolaris

13 FILE SYSTEM FRAMEWORK

—— *virtual file system framework*

zhanglifen
mszhanglifen@163.com

Outline

- Solaris File System Framework
- The vnode
- The vfs Object

13.1 Solaris FILE SYSTEM FRAMEWORK

- Solaris virtual file system framework
  - the virtual file system framework implementes multiple file system types.
  - It allows Sun's distributed computing file system (NFS) to coexist with the UFS file system in SunOS 2.0
- Solaris file systems can be categorized into the following types:
  - Storage based — Regular file systems . The Solaris UFS and PC/DOS file systems are examples.
  - Network file systems — for example, NFS
  - Pseudo file systems —The /proc pseudo file system is example.

13.1.1 Unified File System Interface

- **The framework provides a single set of well-defined interfaces .**
- **Two key objects represent these interfaces:**
  - the virtual file, or *vnode:* The vnode interfaces implement file-related functions.
  - the virtual file system, or *vfs* objects: the vfs interfaces implement file system management functions.

the file system layers is shown below

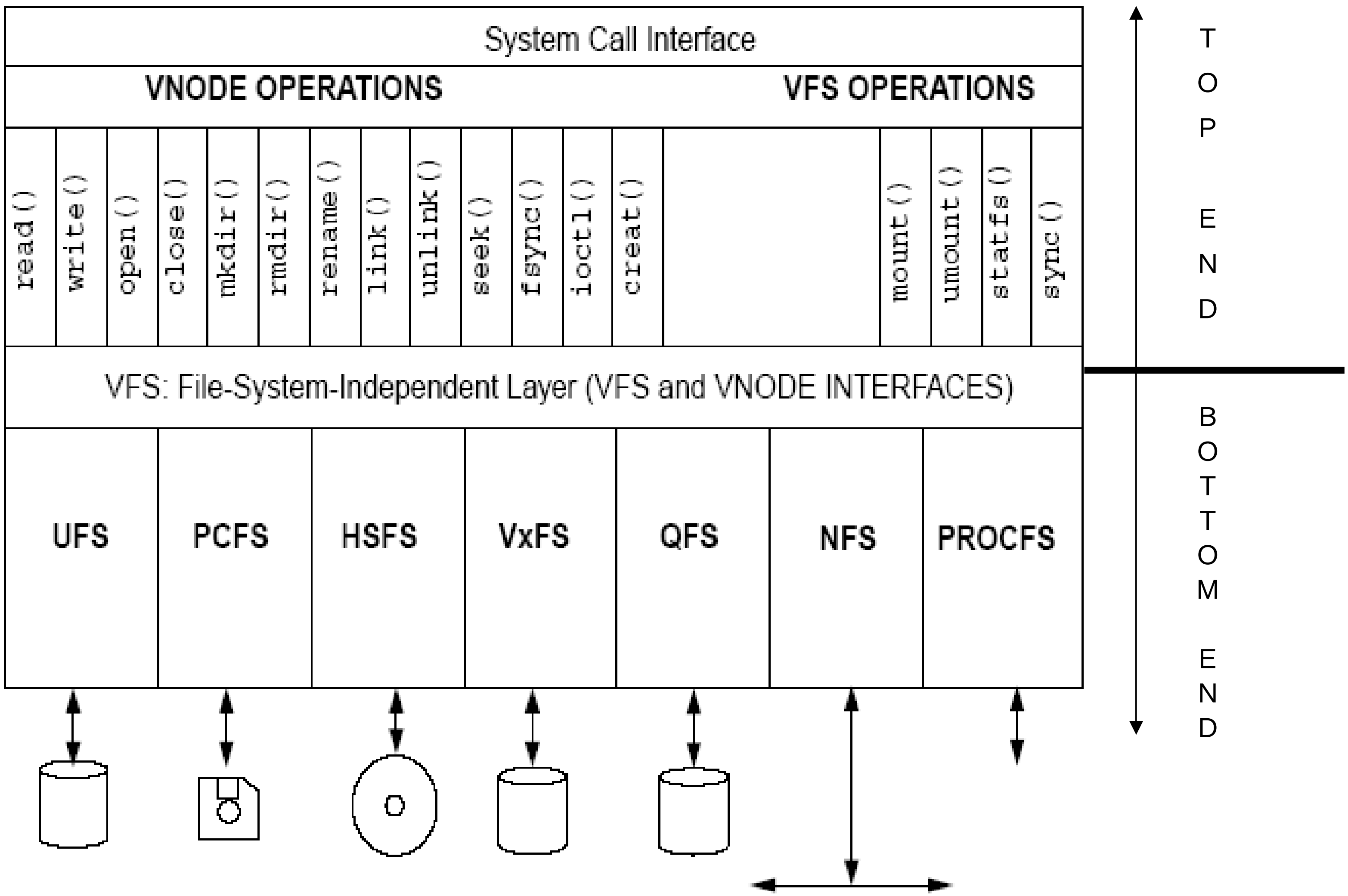


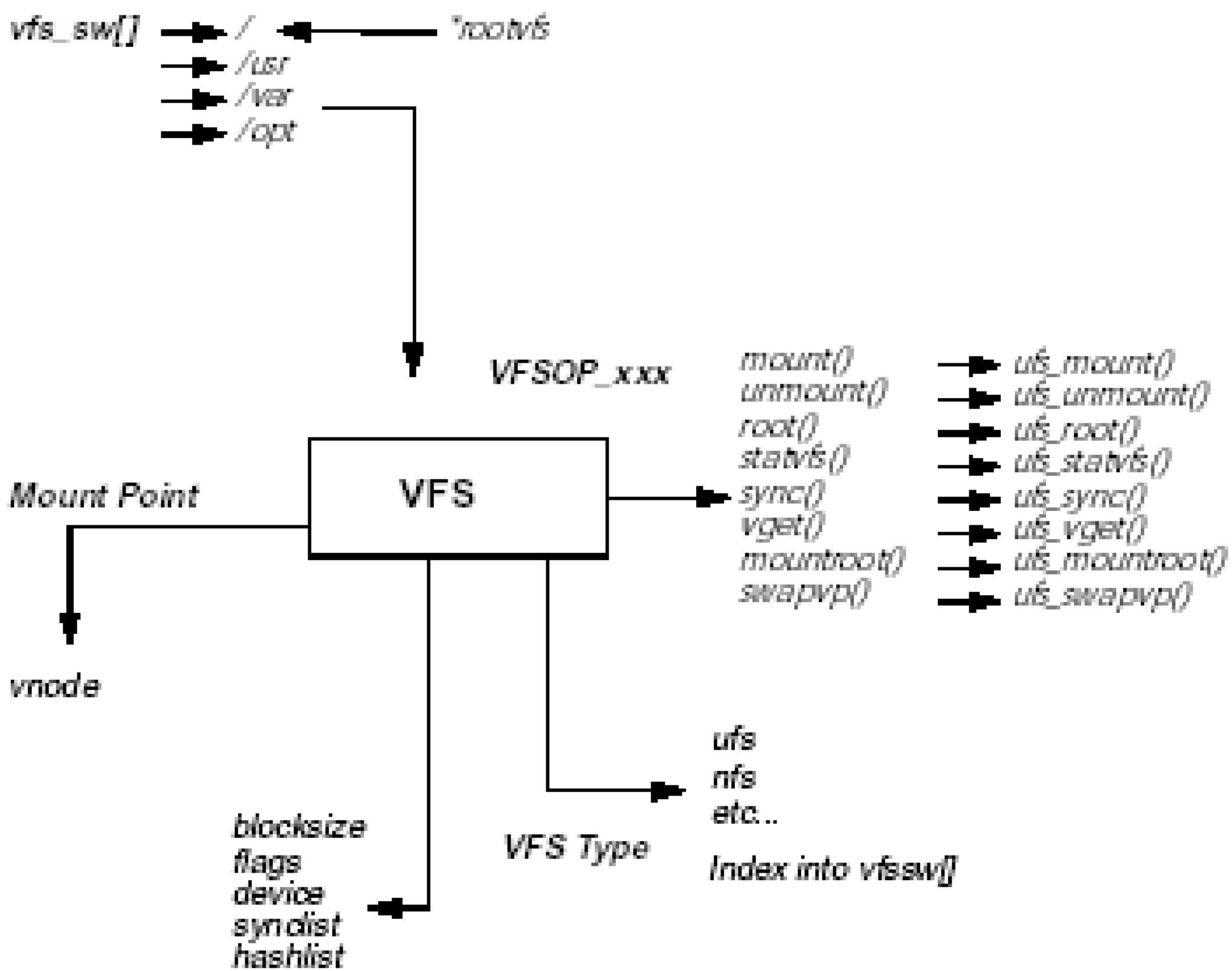
Figure 13.1 shows the file system layers.

13.1.2 File System Framework Facilities

- The vnode/vfs interfaces
  - The “top end” of the file system module implement vnode and vfs objects.
  - The “bottom end” of the file system uses other kernel interfaces to access, store, and cache the data they represent.
  - Disk-based file systems interface to device drivers to provide persistent storage of their data.
  - they interface to network file systems access the networking subsystem to transmit and receive data to remote systems.
  - Pseudo file systems typically access local kernel functions and structures to gather the information they represent.

13.1.2 File System Framework Facilities

The VFS Interface



13.1.2 File System Framework Facilities

- **Loadable file system modules** are dynamically loaded at the time each file system type is first mounted.
- **The vnode/vfs framework** implementes file functions and file system management functions.
- **File system caching** implements caching interface with the HAT layer of the virtual memory system to map, unmap, and manage the memory used for caching.
- **Path-name management** converts paths into vnode pointers.
- Directory name caching provides a mechanism to cache pathname-to-vnode mappings.

13.2 The vnode

- A vnode is a representation of a file in the Solaris kernel.
- The vnode is said to be objectlike .
- it is an encapsulation of a file’s state and the methods that can be used to perform operations on that file.
- The vnode hides the implementation of the file system and exposes file system-independent data and methods for that file to the rest of the kernel.

A vnode object

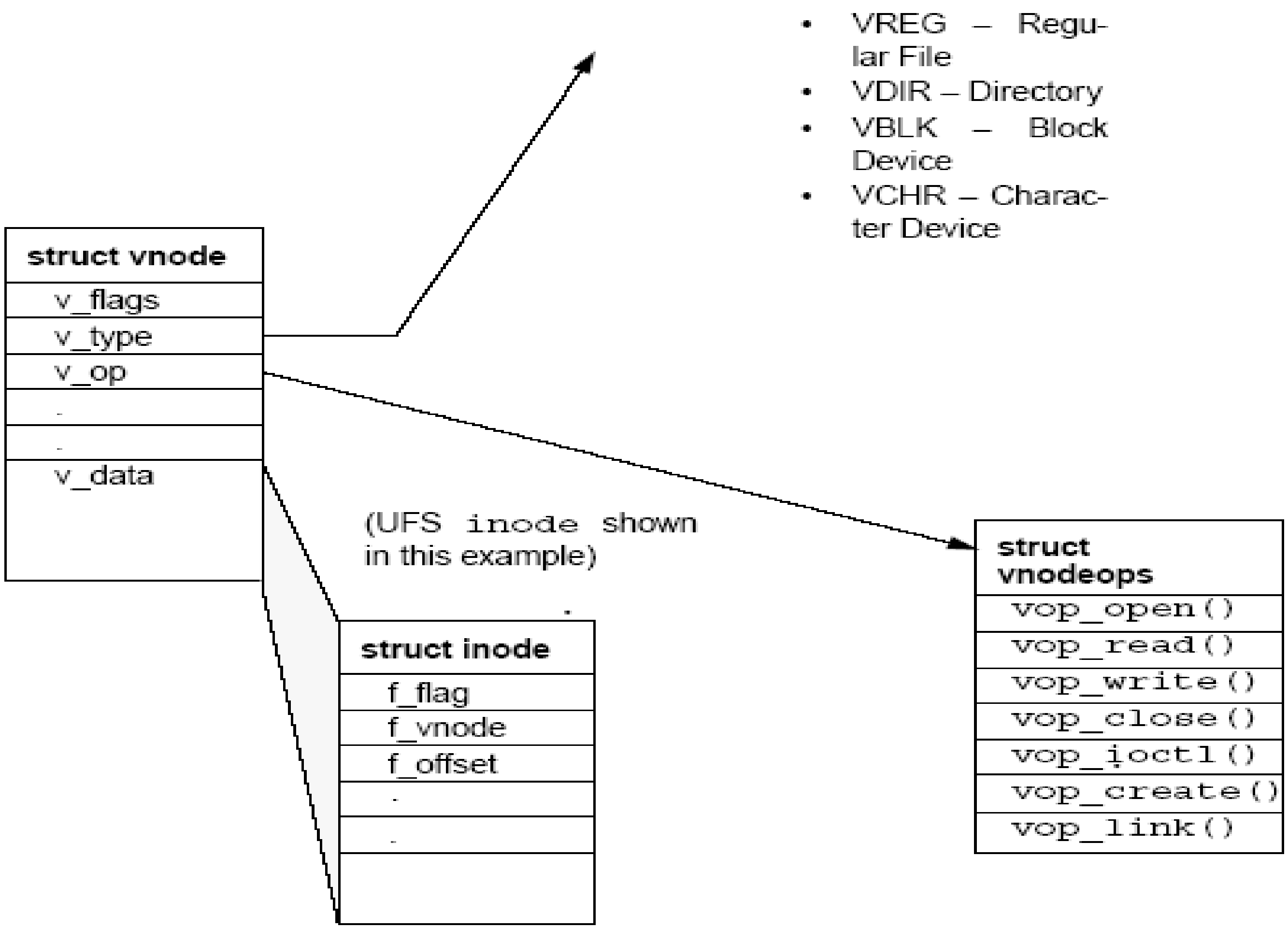


Figure 13.2 A vnode object

A vnode object contains three important items

File-system-independent data

- the type of vnode :file, directory, character device, Block device, Hard link, Named pipe, etc.
- flags of vnode : state, pointers to the file system that contains the vnode, a reference count to the vnode.

Functions to implement file methods

- A structure of pointers to file-system-dependent functions, to implement file's open(),close(), read(), and write().

File-system-specific data

- Data that is used internally by each file system implementation; typically the in-memory inode . UFS uses an inode, NFS uses an rnode,and tmpfs uses a tmpnode.

A vnode object

- For example, to read from a file without knowing that it resides on a UFS file system, the kernel would simply call the file-system-independent macro for read(), VOP_READ(), which would call the vop_read() method of the vnode, which in turn calls the UFS function, ufs_read().

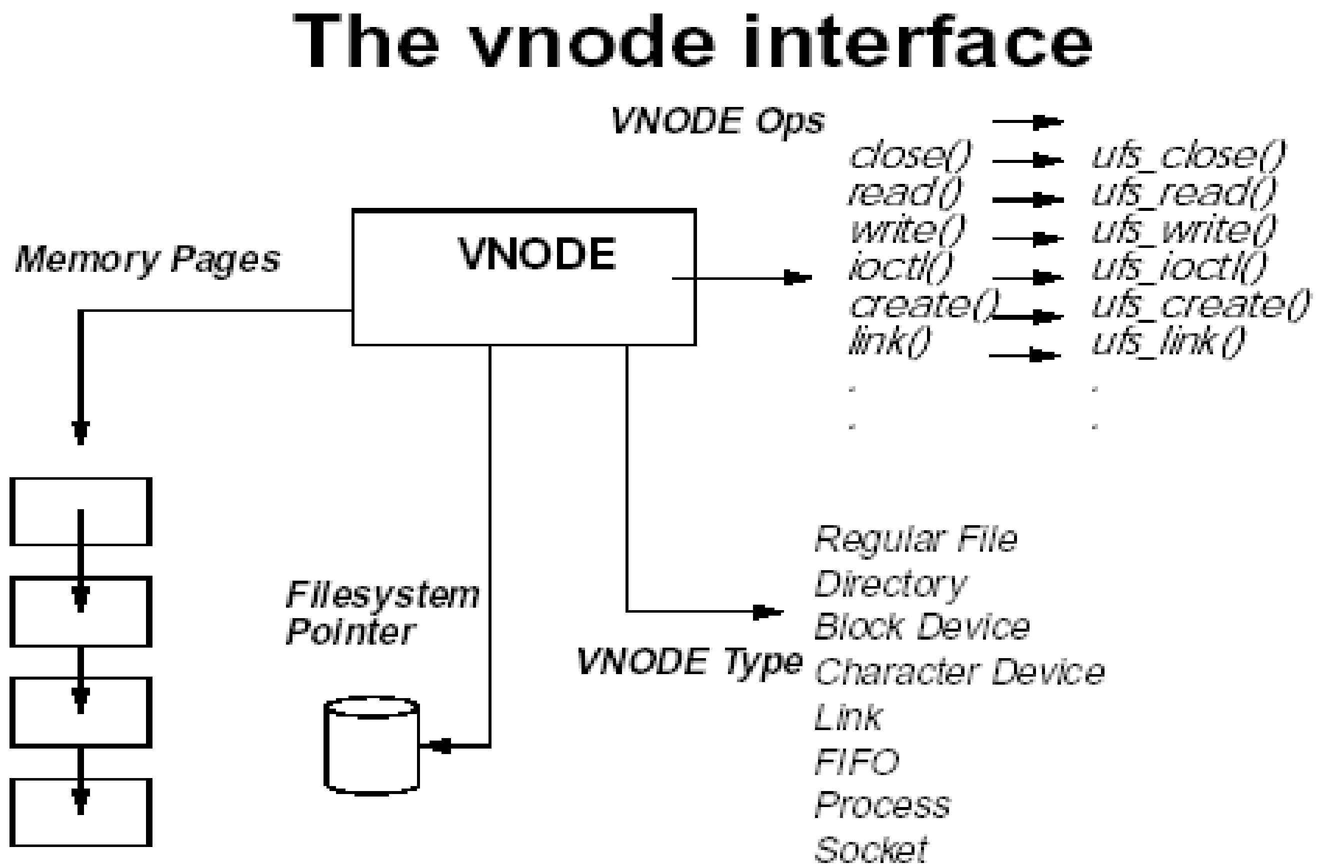
The data structure of a vnode

- typedef struct vnode {
- kmutex_t v_lock; /* protects vnode fields */
  ushort_t v_flag; /* vnode flags (see below) */
  uint_t v_count; /* reference count */
  struct vfs *v_vfsmountedhere; /* ptr to vfs mounted here */
  struct vnodeops *v_op; /* vnode operations */
  struct vfs *v_vfsp; /* ptr to containing VFS */
  struct stdata *v_stream; /* associated stream */
  struct page *v_pages; /* vnode pages list */
  enum vtype v_type; /* vnode type */
  dev_t v_rdev; /* device (VCHR, VBLK) */
  caddr_t v_data; /* private data for fs */
  struct filock *v_filocks; /* ptr to filock list */
  struct shrlocklist *v_shrlocks; /* ptr to shrlock list */
  kcondvar_t v_cv; /* synchronize locking */
  } vnode_t;

13.2.2 Vnode Methods

- The vnode interface provides the set of file system object methods
- The **Vnode Methods** perform all file-system-specific file operations.
- The figure is shown below.

13.2.2 Vnode Methods



13.2.3 vnode Reference Count

- A vnode is created by the file system at the time a file is first opened or created and stays active until the file system decides the vnode is no longer needed.
- The vnode framework provides an infrastructure that keeps track of the number of references to a vnode.
- It is important to distinquish a vnode reference from a lock:
  - A lock ensures exclusive access to the data,
  - the reference count ensures persistence of the object.

13.2.4 Interfaces for Paging vnode Cache

- Solaris unifies file and memory management by using a vnode to represent the backing store for virtual memory.
- A page of memory represents a particular vnode and offset.
- The file system uses the memory relationship to implement caching for vnodes within a file system.
- The virtual memory system provides a set of functions for cache management and I/O for vnodes.

13.2.5 Block I/O on vnode Pages

- The block I/O subsystem provides Three functions for initiating I/O to and from vnode pages.
- The table shows to initiate I/O between a physical page and a device:

Function	Description
`bdev_strategy()`	Initiates an I/O, using the block I/O device.
`pageio_done()`	Waits for the block device I/O to complete.
`pageio_setup()`	Sets up a block buffer for I/O on a page of memoryso that it bypasses the block buffer cache by setting the `B_PAGEIO` flag and putting the page list on the `b_pages` field.

13.3 The vfs Object

- The vfs layer provides an administrative interface int the file system to support commands like mount and umount in a file-system-independent manner.
- The interface achieves independence by means o a virtual file system (vfs) object.
- The vfs object represents an encapsulation of a file system's state and a set of methods for each of the file system administrative interfaces.

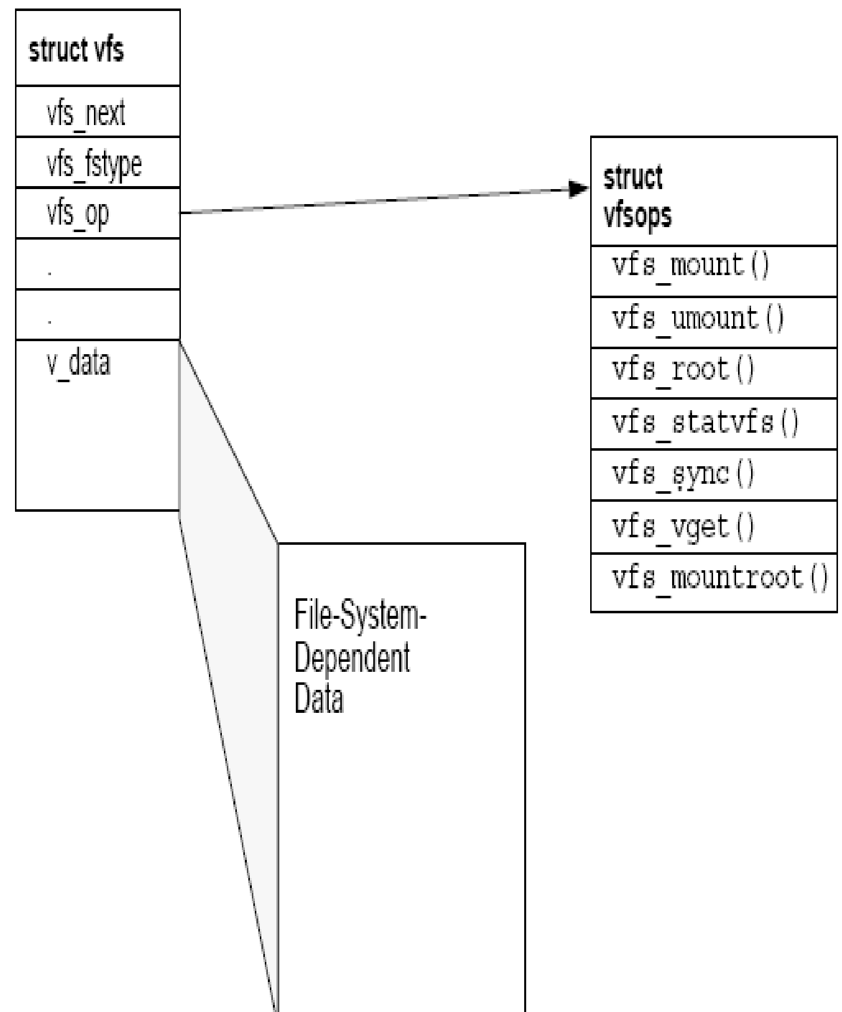


Figure 13.3 illustrates the vfs object.

Structure per mounted file system

```
typedef struct vfs {
struct vfs *vfs_next; /* next VFS in VFS list */
struct vfsops *vfs_op; /* operations on VFS */
struct vnode *vfs_vnodecovered; /* vnode mounted on */
uint_t vfs_flag; /* flags */
uint_t vfs_bsize; /* native block size */
int vfs_fstype; /* file system type index */
fsid_t vfs_fsid; /* file system id */
caddr_t vfs_data; /* private data */
dev_t vfs_dev; /* device of mounted VFS */
ulong_t vfs_bcount; /* I/O count (accounting) */
ushort_t vfs_nsubmounts; /* immediate sub-mount count */
struct vfs *vfs_list; /* sync list pointer */
struct vfs *vfs_hash; /* hash list pointer */
ksema_t vfs_reflock; } /* mount/unmount/sync lock */
```

Operations supported on virtual file system

- typedef struct vfsops {
- int (*vfs_mount)(struct vfs *, struct vnode *, struct mounta *,
- struct cred *);
- int (*vfs_unmount)(struct vfs *, struct cred *);
- int (*vfs_root)(struct vfs *, struct vnode **);
- int (*vfs_statvfs)(struct vfs *, struct statvfs64 *);
- int (*vfs_sync)(struct vfs *, short, struct cred *);
- int (*vfs_vget)(struct vfs *, struct vnode **, struct fid *);
- int (*vfs_mountroot)(struct vfs *, enum whymountroot);
- int (*vfs_swapvp)(struct vfs *, struct vnode **, char *);
- }

13.3.1 The File System Switch Table

- The file system switch table is a systemwide table of file system types.
- Each file system type that is loaded on the system can be found in the virtual file system switch table.
- The file system switch table provides an ASCII list of file system names (e.g., ufs, nfs), the initialization routines, and vfs object methods for that file system.
- The vfs_fstype field of the vfs object is an index into the file system switch table.

13.3.1 The File System Switch Table

- File system type switch table is shown below:

```
typedef struct vfssw {
char *vsw_name; /* type name string */
int (*vsw_init)(struct vfssw *, int);
/* init routine */
struct vfsops *vsw_vfsops; /* file system operations vector */
int vsw_flag; /* flags */
} vfssw_t;
```

13.3.2 The Mounted vfs List

- You can obtain a list of mounted file systems by starting at rootvfs and following the vfs -> vfs_next chain, as shown in Figure 13.4.

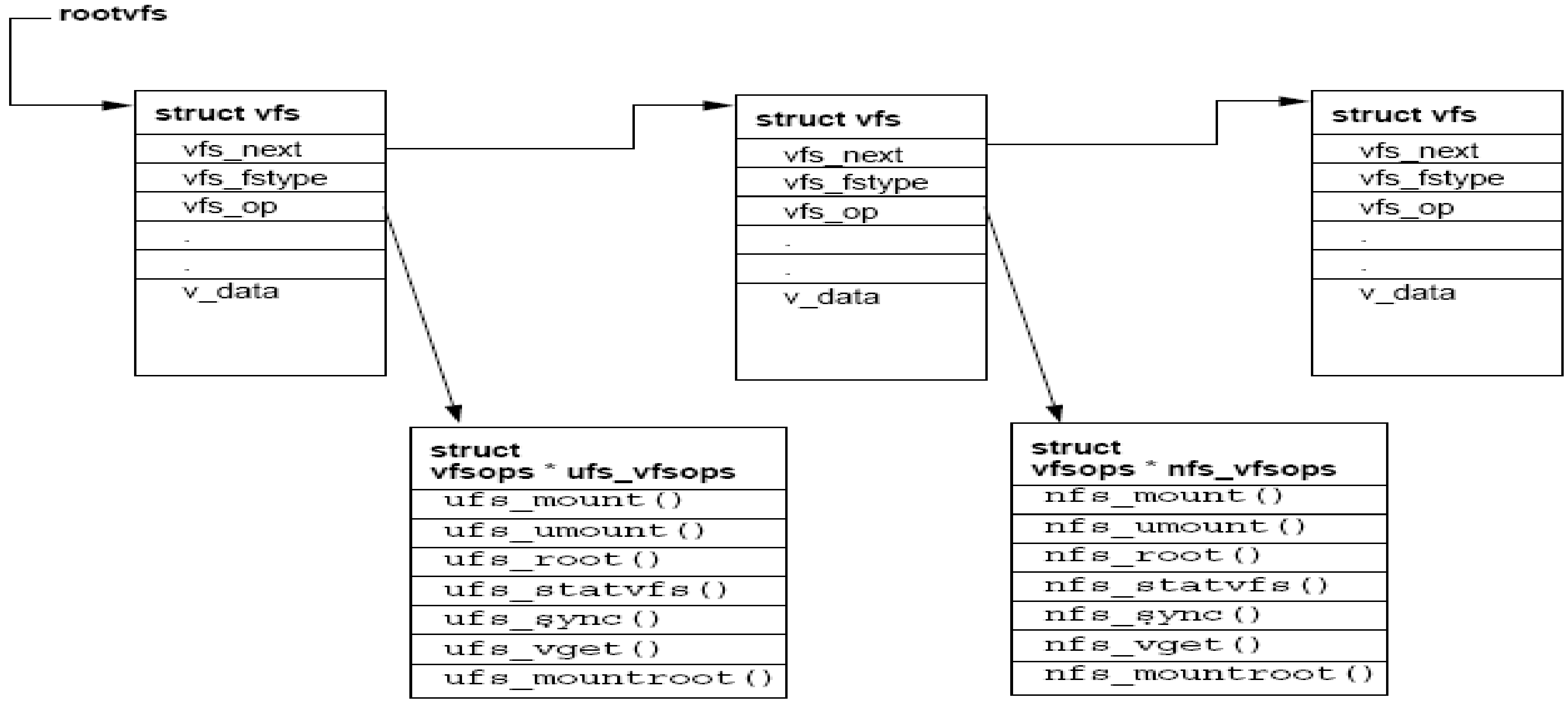


Figure 13.4 *The Mounted* vfs *List*

Reference

- Jim Mauro, Richard McDougall, Solaris Internals-Core Kernel Components, Sun Microsystems Press, 2000
- Solaris internals and performance management, Richard McDougall, 2002