The User Interface

- Files, directory, file descriptor, file systems
- File & Directories
  - File: logically a container for data
  - A hierarchical, tree-structured name space
  - Pathname: components in the path from the root to the node, by “/”
  - Special entries: “.” & “..”
  - Link: a directory entry for a file.

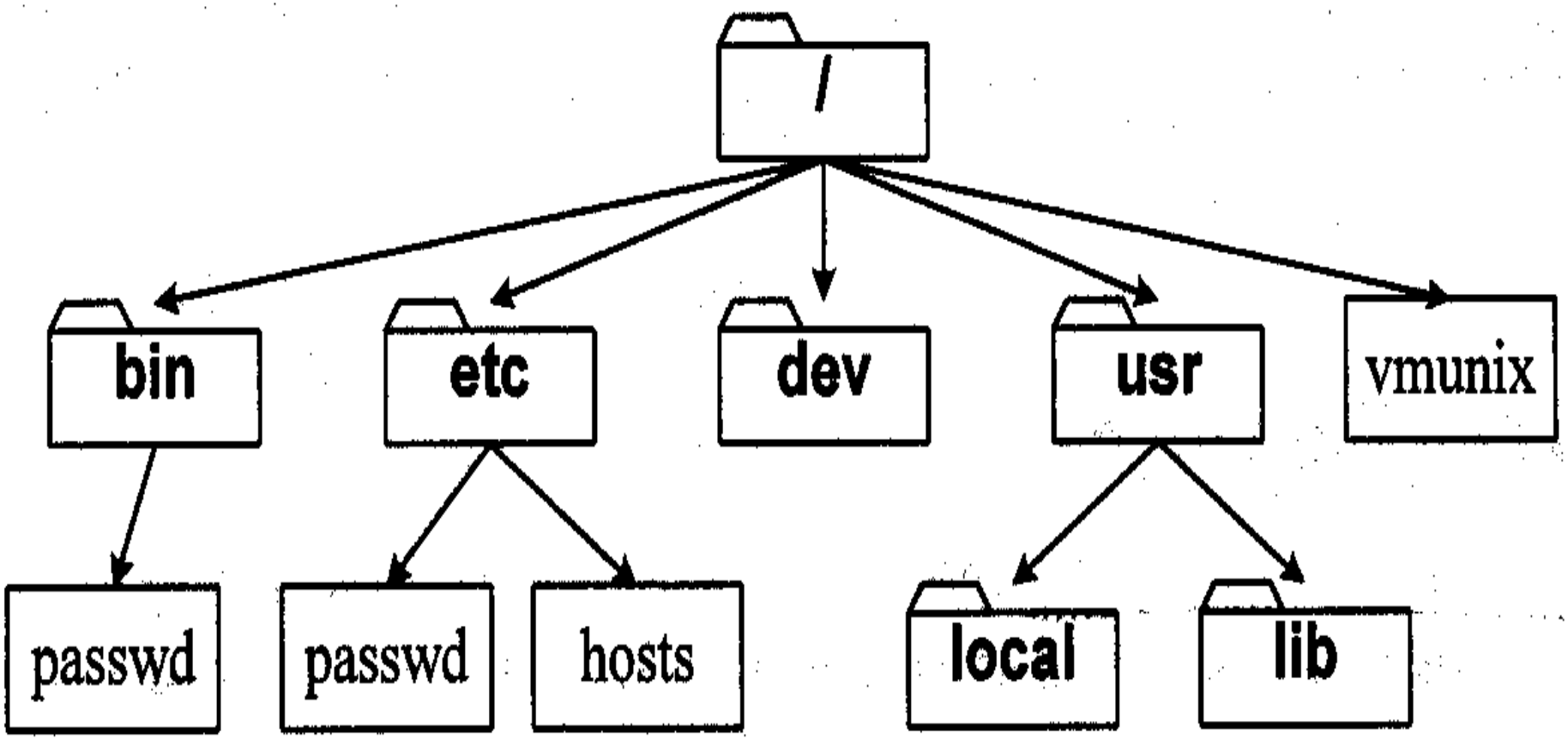


Figure 8-1. Files are organized in a directory tree.

Directory syscalls

```
dirp = opendir(char *filename);
direntp = readdir (dirp);
rewinddir(dirp);
status = closedir(firp);
struct dirent {
    int_t d_ino;
    char d_name[NAME_MAX +1];
}
```

File Attributes

- Kept in the *inode* (index node)
- File attributes:
  - File type
  - Number of hard links
  - File size
  - Device ID
  - Inode number
  - User and Group Ids of the owner of the file.
  - Timestamps
  - Permissions and mode flags (suid, sgid, sticky)

File Descriptors

- fd is a per-process object.

```
fd = open (path, oflag, mode);
```

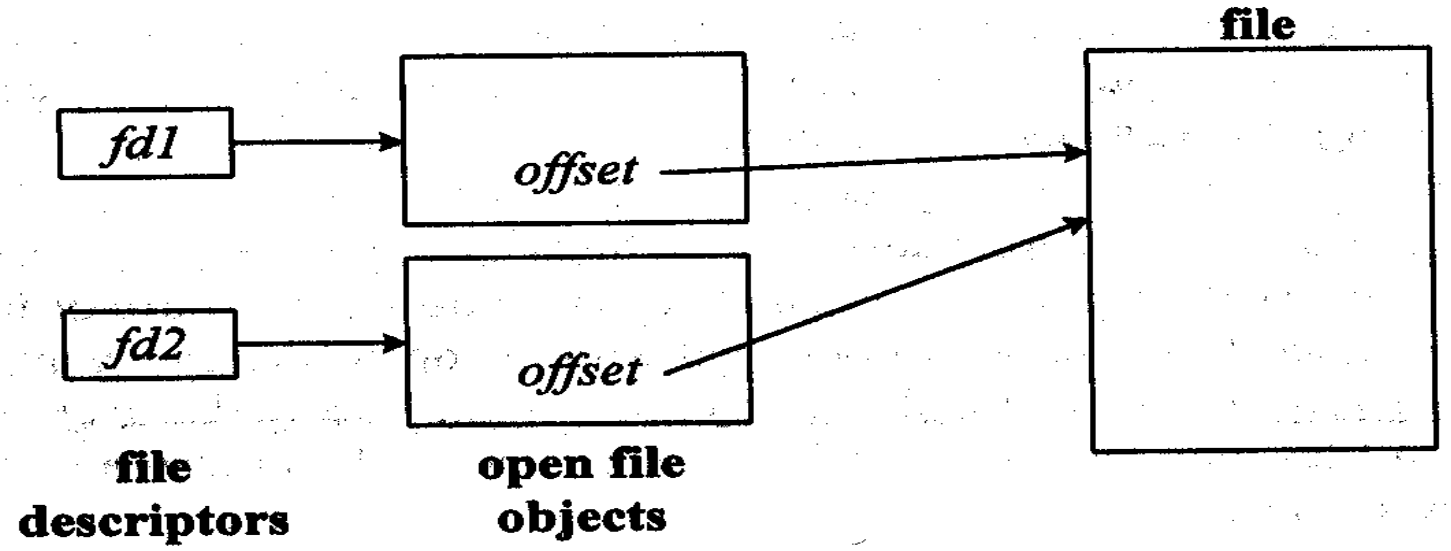


Figure 8-2. A file is opened twice.

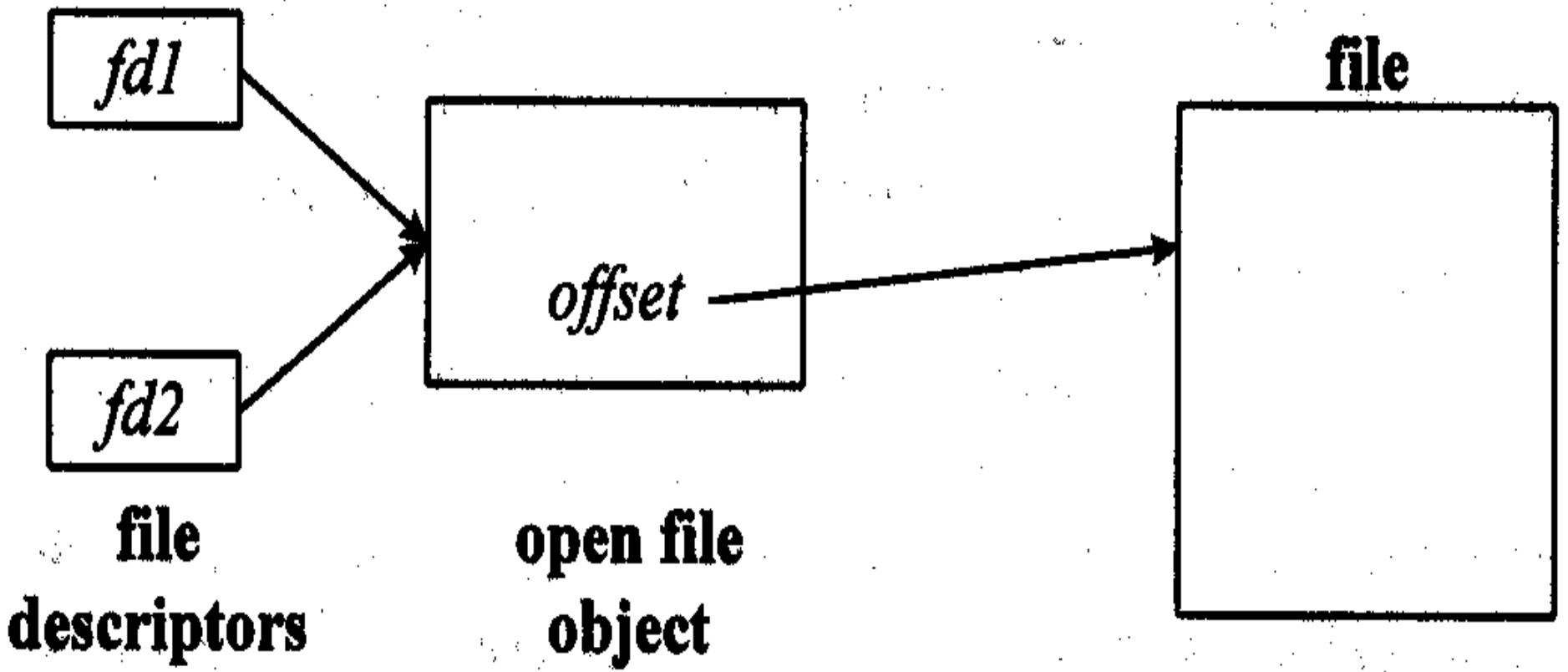


Figure 8-3. Descriptor cloned through *dup*, *dup2*, or *fork*.

File I/O

- Random and sequential access
  - lseek ()
  - nread = read(fd, buf, count);
- Scatter-Gather
  - nbytes = writev(fd, iov, iovcnt);

File Locking

- Read and write are atomic.
- Advisory locks: protect from cooperative processes, flock() in 4BSD; chmod in svr3
- Svr4: r/w locks.
- Mandatory locks:kernel

File system

- Mount-on
  - A directory is covered by the mounted file system.
  - mount table & vfs list
- Logical disks
  - A linear sequence of fixed sized, randomly accessible, blocks.
  - Partition

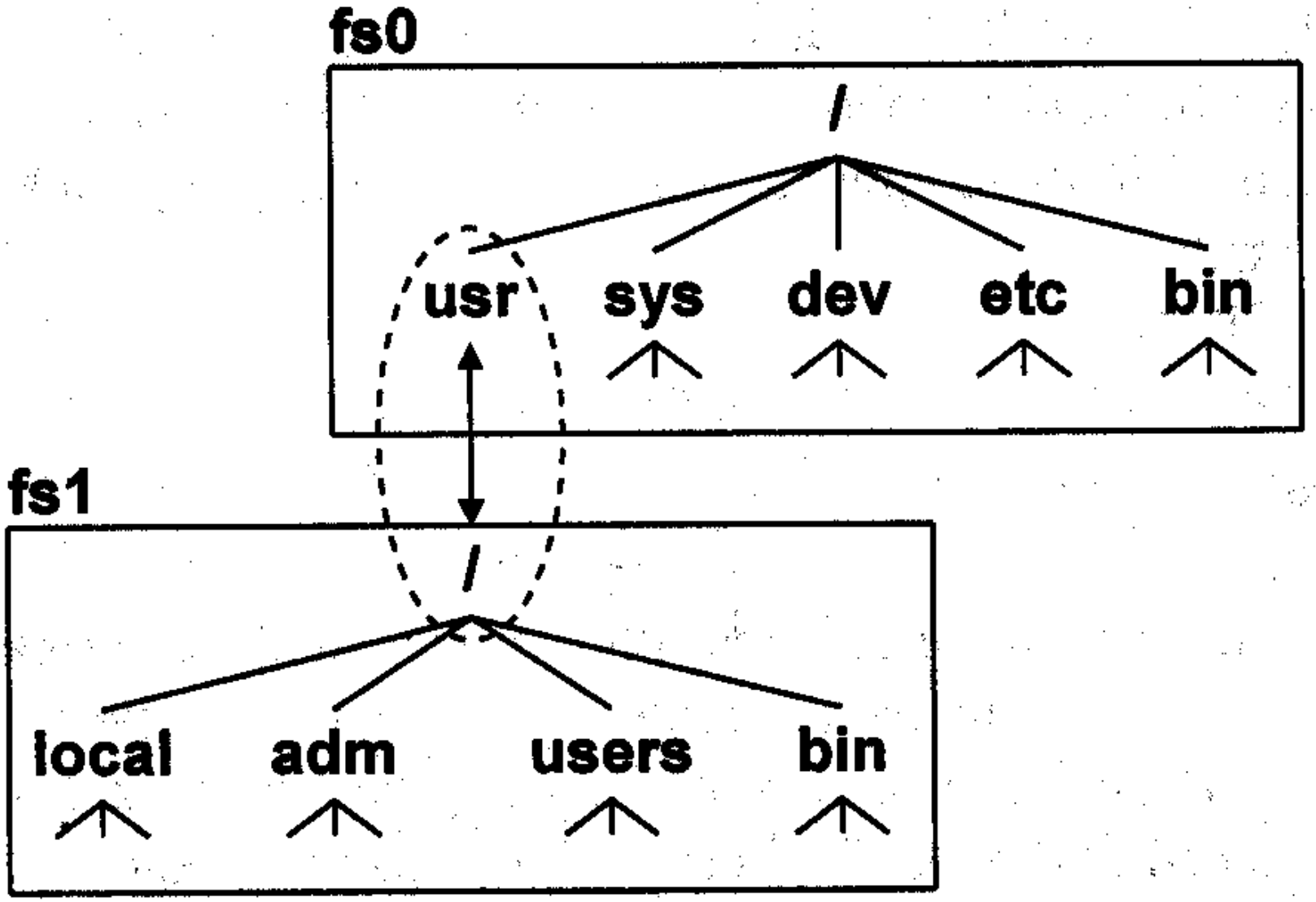


Figure 8-5. Mounting one file system onto another.

Logical Disks

- A logical disk is a storage abstraction that the kernel sees as a linear sequence of fixed sized, randomly accessible blocks.
- newfs, mkfs, partition
- Advanced Topics:
  - Volume
  - Disk mirror
  - Stripe sets
  - RAID

Device I/O

- Block & character devices
- Character:

```
struct {
  int (*d_open)();
  int (*d_close)();
  int (*d_read)();
  int (*d_write)();
} cdevsw[];
```

- Major & minor device number:
  - indexes in the device table

Opening a file

- int fd = open(char *pathname, int flags,*mode_t mode*);
  - pathname: filename (directory, ...)
  - flags: read, write, ...
  - mode: file access permissions (optional)
  - returns
    - fd: file descriptor
    - -1: in case of errors
  - Example:
    - fd=open(name,O_RDWR|O_CREAT,S_IRWXU)

Opening a file

- fd = open(pathname, mode)

1. Allocate a descriptor
2. Allocate an open file object
3. Lookup path name
4. Check permissions
5. Check operation
6. Not exist, O_Creat, VOP_CREAT; ENOENT
7. VOP_OPEN
8. If O_TRUNC, VOP_SETATTR
9. Initialize
10. Return the index of the descriptor

Closing a File

- int close(fd)
  - fd: file descriptor
  - returns:
    - 0 if successful
    - -1 in case of errors
  - Example
    - sts=close(fd);

Reading from File

- ssize_t read(int fd, void *buf, size_t count);
  - fd: file descriptor
  - buf: pointer to buffer space
  - count: I/O size (in bytes)
  - returns:
    - number of bytes read
    - -1 in case of error
  - Example:
    - nb=read(fd,buffer,sizeof(buffer));

Writing to File

- ssize_t write(int fd, const void *buf, size_t count);
  - fd: file descriptor
  - buf: pointer to bufferspace
  - count: I/O size
  - returns:
    - number of bytes written
    - -1 in case of error
  - Example:
    - nb=write(fd,string,strlen(string)+1);

File Seek

- off_t lseek(int fildes, off_t offs, int whence);
  - Positions File Pointer for subsequent I/O
  - fd: file descriptor
  - offs: offset (in bytes)
  - whence: specifies whetheroffs is relative to start of file, current position or end of file
  - returns:
    - offset: resulting offset location
    - -1: in case of error
  - Example:
    - bytes=lseek(fd,offset,SEEK_SET);

File Information

- int fstat(int fd, struct stat *buf);
  - Returns information about an (open) file
  - fd: file descriptor
  - buf: pointer to a struct stat
  - returns
    - 0: if successful
    - -1: in case of errors
  - Example:
    - sts=fstat(fd,&statbuf);
    -
    -

File I/O (1)

- Read(to a user buffer address)
  - Fd-> the open file object, verify mode-> vnode-> get the rw-lock->call s5read()
  - Offset -> block number & the offset -> uiomove()-> call copyout()
  - The page not in memory? page fault->the handler->s5getpage()->call bmap()
  - logical to physical, search vnode's page list, not in? allocates a free page and call the disk driver to read the file
  - Sleep until the I/O completes. Before copy to user data space, verify the user has access
  - s5read() returns, unlock, advance the offset, return the number of bytes read

File I/O (2)

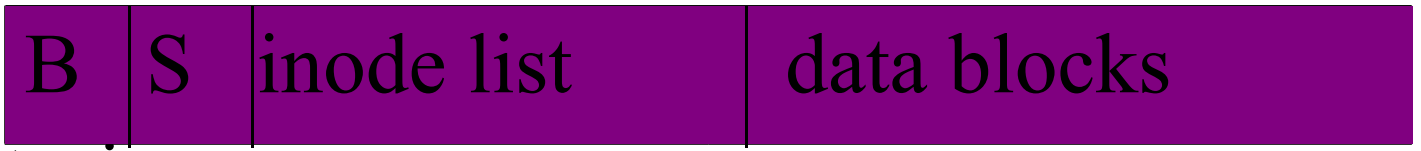
- Write:
  - Not immediately to disk
  - May increase the file size
  - May require the allocation of data blocks
  - Read the entire block, write relevant data, write back all the block

Link, Unlink, Rename

- int link(const char *oldpath, const char *newpath);
  - creates a new (hard) link to file
- int unlink(const char *pathname);
  - unlinks a file (and possibly deletes it)
- int rename(const char *oldpath, const char *newpath);
  - renames a file, moving it between directories (if required)
- All return
  - 0: if successful
  - -1: in case of errors

The System V File System(s5fs)

- The layout of s5fs partition:



- Directories:
  - s5fs directory is a special file containing a list of files and subdirectories.

73	.
38	..
9	file1
0	deletedfile
110	subdirectory 1
65	archana

Figure 9-2. *s5fs* directory structure.

Inodes

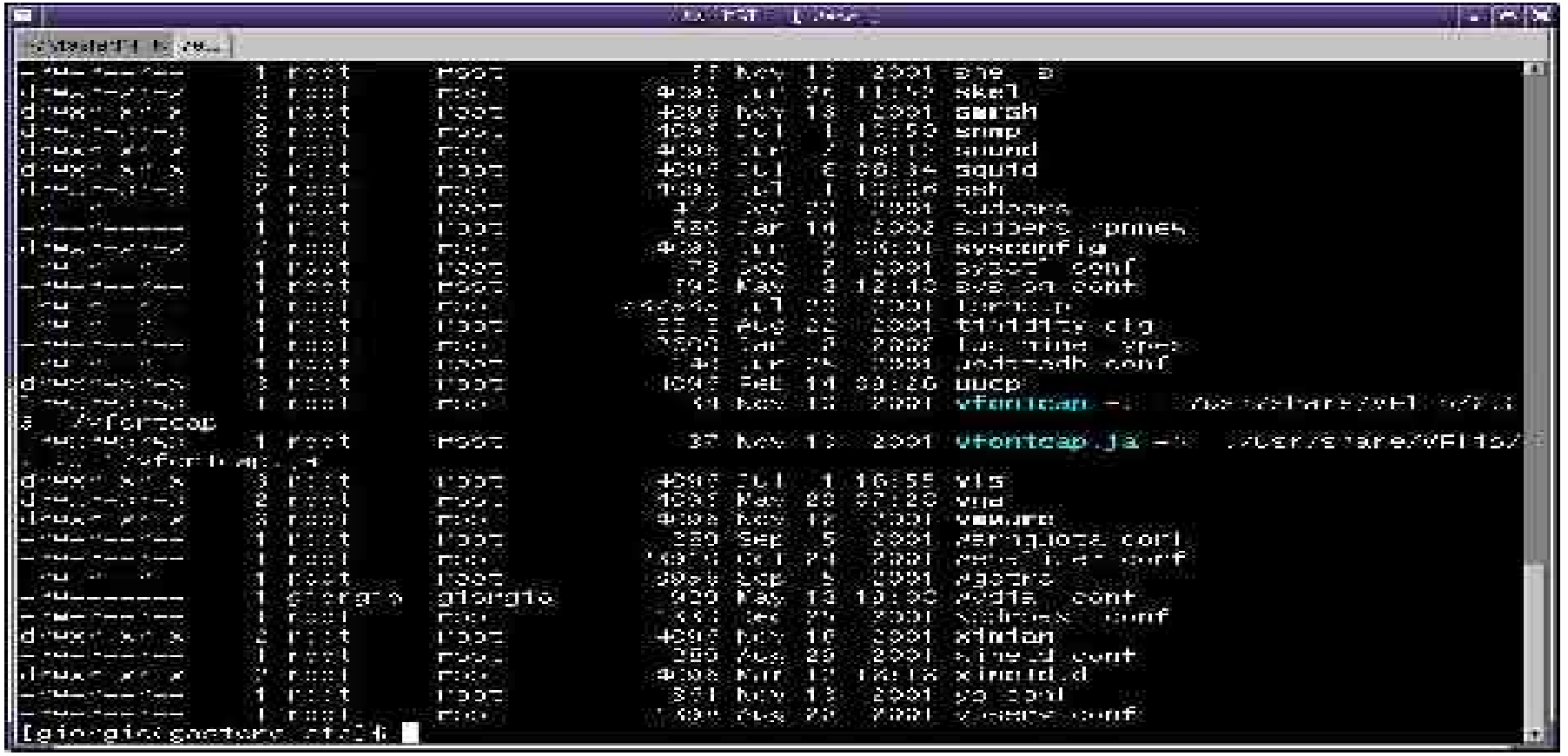
- The inode contains administrative information,or meta data.
  - The node list contains all the inodes.
  - On-disk inode
  - In-core inode

Inode Fields

Table 9-1. Fields of `struct dinode`

Field	Size (bytes)	Description
`di_mode`	2	file type, permissions, etc.
`di_nlinks`	2	number of hard links to file
`di_uid`	2	owner UID
`di_gid`	2	owner GID
`di_size`	4	size in bytes
`di_addr`	39	array of block addresses
`di_gen`	1	generation number (incremented each time inode is reused for a new file)
`di_atime`	4	time of last access
`di_mtime`	4	time file was last modified
`di_ctime`	4	time inode was last changed (except changes to `di_atime` or `di_mtime`)

di_mode



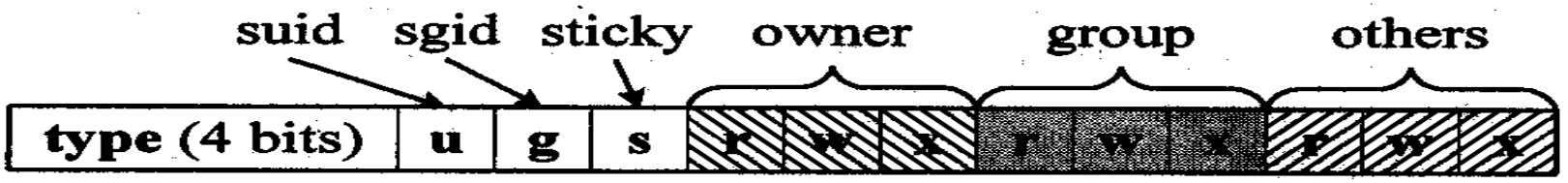


Figure 9-3. Bit-fields of `di_mode`.

Block array of inode, di_addr

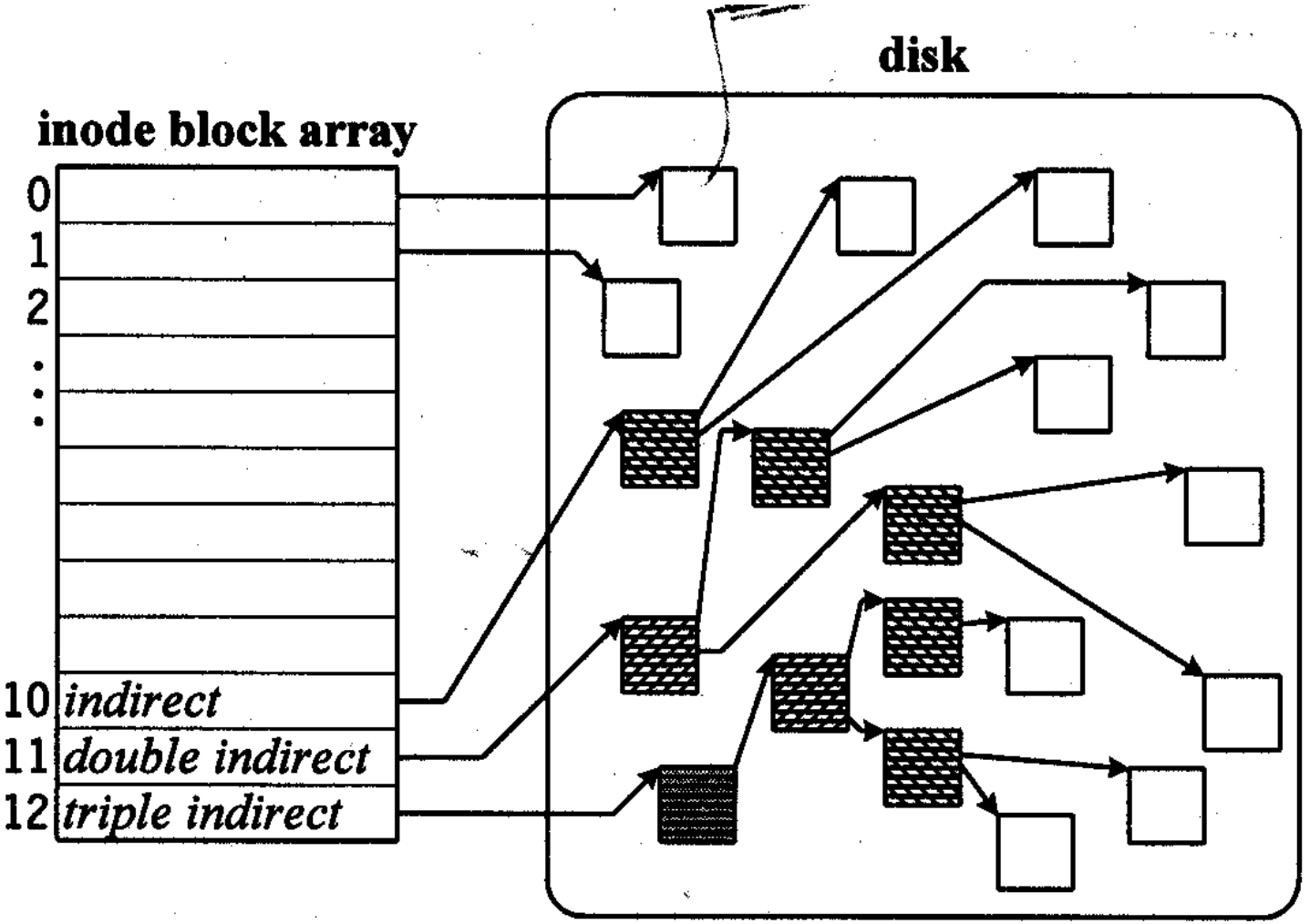


Figure 9-4. disk block array in *s5fs* inode.

The superblock

- Size in blocks of the file system
- Size in blocks of the inode list
- Number of free blocks and inodes
- Free block list
- Free inode list

Free block list

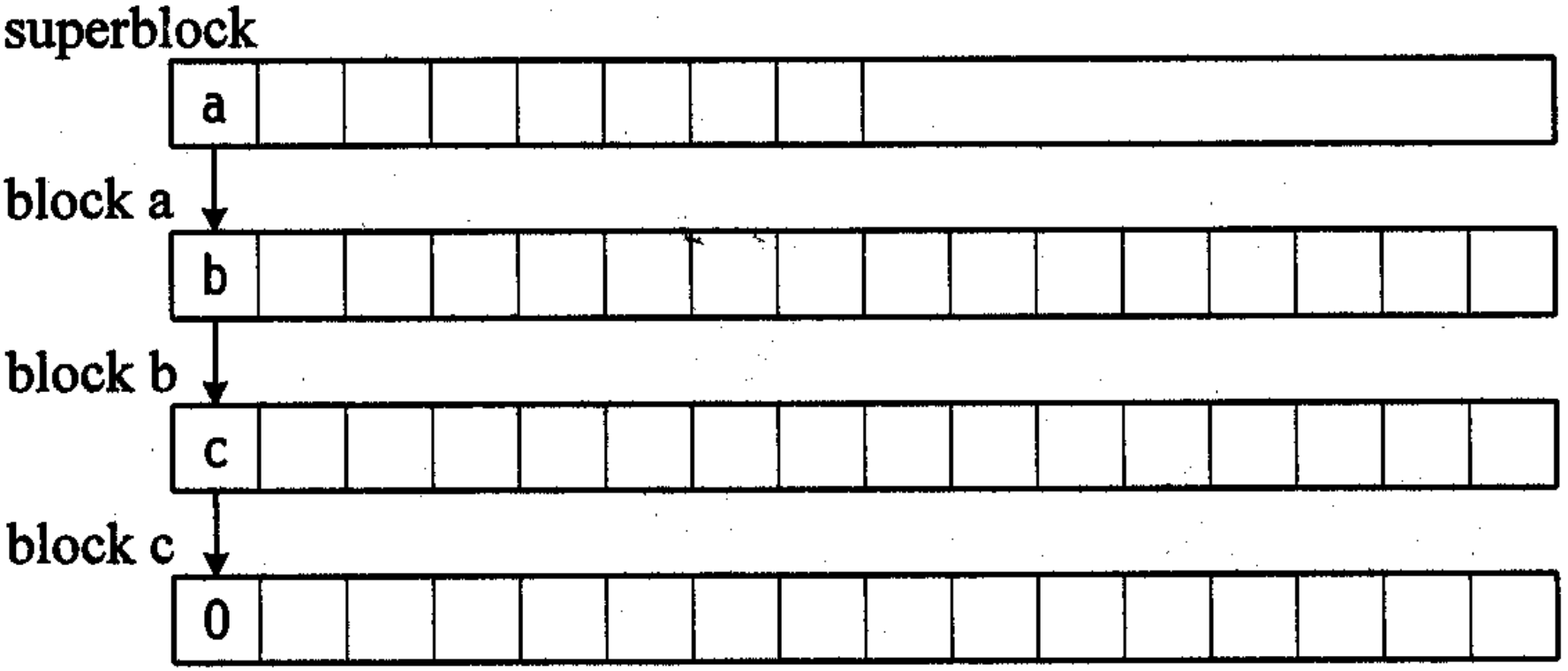


Figure 9-5. free block list in *s5fs*.