

File System Implementation

- Disk space allocation
- Free-space management
- Directory implementation



Disk Space Allocation

- Initially empty disk
- File creation allocates space
- Write operations extend file
- Deleted files release space
- Problem: managing used/free

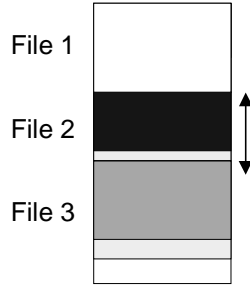
Contiguous Allocation

- All file blocks contiguous
 - For 100-block file, blocks N to N+99
- Allocate all blocks on create?
 - Must anticipate maximum size
 - May end up with wasted space
- External fragmentation

Contiguous Allocation Example

Initially empty disk
 Create file 1 .. 3
 Write file 2, 1, 3
 Delete file 1
 Write file 2 ??

Can't move file 3;
 file 2 extension is
 limited

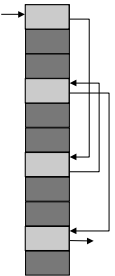


Contiguous Allocation Issues

- **Initial size guess important**
 - Too low -- can't write all data
 - Too high -- wasted space
 - Internal fragmentation
- **Basis for estimate**
 - Advance knowledge of size?
 - Half of biggest free segment?

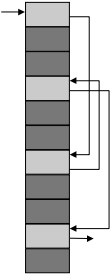
Linked Allocation (1)

- **File blocks not contiguous**
- **Linked list implementation**
- **Disk block content**
 - Data bytes
 - Pointer to "next" block
 - Pointer overhead vs data



Linked Allocation (2)

- OK for sequential access
- Random access a problem
 - Must follow links from beginning
 - Blocks may be scattered across the disk

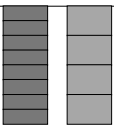


FAT Linked List

- File allocation table
- Linked list has pointers
 - Instead of actual data
- Multiple list nodes per block
 - In contiguous space?
- Less list traversal overhead

Cluster Size

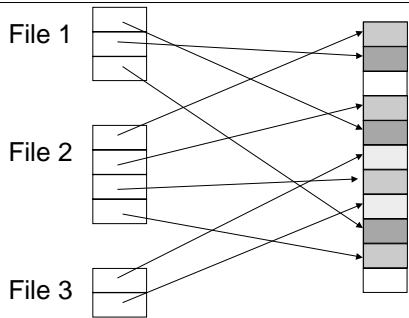
- Fixed disk block size
- Two problems
 - Pointer overhead versus data
 - Absolute pointer size limits disk size
- Solution: multi-block clusters
 - Problem: internal fragmentation



Indexed Allocation

- **Keep array of block pointers**
- **Efficient direct access**
 - Look up block in array
 - Use pointer to access block (cluster?)
- **Need not be contiguous**
 - But array optimization possible

Indexed Allocation



Free Space Management

- **Some blocks free (unless full)**
- **How choose next block?**
 - “Free” file? (tracked like real ones)
 - Bit vector?
 - Search for free byte or bits
- **Proximity algorithm?**
 - Contiguous where possible?

UNIX Disk Structure

- **Several UNIX layouts**
 - **Space management**
 - Blocks (4K-8K bytes?)
 - Fragments (512 bytes?)
 - **Space efficiency**
 - Small and large files
- Fast file system (FFS), UNIX file system (ufs)

UNIX File Status

```
int stat (const char* pathname,
          struct stat* buf);

int fstat (int fildes,
           struct stat* buf);

int lstat (const char* pathname,
           struct stat* buf);
```

“stat” versus “lstat”?

File Status Structure

```
struct stat
{
  dev_t    st_dev;
  ino_t    st_ino;
  mode_t   st_mode;
  nlink_t  st_nlink;
  uid_t    st_uid;
  gid_t    st_gid;
  dev_t    st_rdev;
  off_t    st_size;
  ...
};
```

May differ across implementations

Device ID (FS)

File serial # (inode)

File type & perms.

links to file

User/group ID's

Device # for special files

File size (bytes)

File Status Structure

```

struct stat
{
  ...
  time_t  st_atime;
  time_t  st_mtime;
  time_t  st_ctime;
  uint_t  st_blksize;
  int     st_blocks;
};

```

Times: access, modification, status change

Best I/O block size

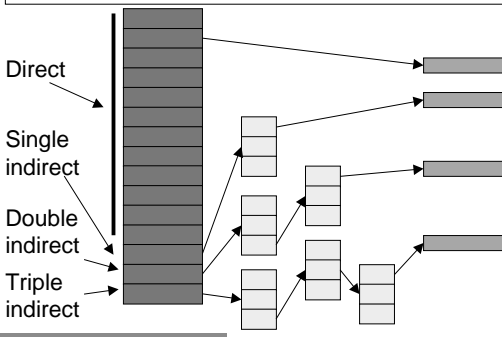
disk blocks allocated

Index Node (inode)

Pronounced "eye-node"

- One for each file in filesystem
- Maintains list of data blocks
- Referenced by file serial number
 - Also called "inode"
- Direct/indirect block pointers

Inode Block Pointers



Block Pointer Addressing (1)

Assume 4K blocks, 4-byte block pointers

Pointer type	# blocks addressed
Direct	1
Single	1,024 (1K)
Double	1,048,576 (1M)
Triple	1,073,741,824 (1G)

1G blocks = 4Tbyte = 4.4E+12 bytes

4-byte pointer addresses 4Gbyte

Block Pointer Addressing (2)

Assume 32K blocks, 8-byte block pointers

Pointer type	# blocks addressed
Direct	1
Single	4,096 (4K)
Double	16,777,216 (16M)
Triple	68.7E+9 (64G)
Quadruple	281E+12 (256T)

281E+12 blocks = 9.2E+18 bytes

8-byte pointer addresses 1.8E+19 bytes

UNIX File Types

- Regular file
- Directory file
- Character special file
- Block special file
- FIFO
- Socket
- Symbolic link

UNIX File Permissions

```
int chmod (const char* pathname,
           mode_t mode);
int fchmod (int fildes,
           mode_t mode);
```

S_IRUSR S_IRGRP S_IROTH
S_IWUSR S_IWGRP S_IWOTH
S_IXUSR S_IXGRP S_IXOTH

Horizontal lines for notes under the permissions section.

UNIX File Ownership

```
int chown (const char* pathname,
           uid_t owner,
           gid_t group);
int fchown (int fildes,
           uid_t owner,
           gid_t group);
int lchown (const char* pathname,
           uid_t owner,
           gid_t group);
```

Horizontal lines for notes under the ownership section.