File Systems

Today

- Files and directories
- File & directory implementation
- Sharing and protection

Next

- File system management & examples

Files and file systems

Most computer applications need to:

- Store large amounts of data (larger than their address space)
- that must survive process termination and
- can be access concurrently by multiple processes

→ Usual answer: Files – from user's perspective, the smallest allotment of logical secondary storage

File system – part of the OS dealing with files

- Supports the file abstraction of storage
- Naming – how do users select files?
- Protection – users are not all equal
- Reliability – information must be safe for long periods of time
- Storage mgmt. – efficient use of storage and fast access to files

File – attributes, types and operations

- Files are collection of data with some attributes
  - Names (low-level name or inode) and type, if supported, location, owner, last read/write times, …
- Different OSs support different file types
  - Regular, binary, directories, …
  - Character special (model terminals [/dev/tty], printers, etc) and block special files (model disks [/dev/hd1])
  - Extensions as hints and the use of magic numbers
    - Some typical file extensions
  - Pros and cons of strongly typed files
- Basic operations
  - Create, delete, write, read, file seek, truncate
  - Other operations can be built on this basic set (e.g. cp)

File structures & access methods

- Several file structures, three common ways
  - Byte sequence – Unix & Windows; user imposes meaning
  - Record sequence – think about 80-column punch cards
  - Tree – records have keys, tree is sorted by it

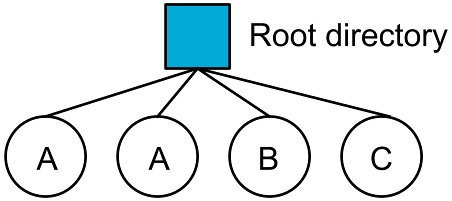
- Access methods
  - Sequential – tape model
    - Read/write next; simplest and most common
  - Random/direct access – disk model
    - Two approaches: read/write x, or position to x and read/write
    - Retain sequential access – read/write + update last position
  - Other access methods
    - On top of direct access, normally using indexing
    - Multi-level indexing for big files (e.g. IBM Indexed Sequential Access Method)

Directory structure

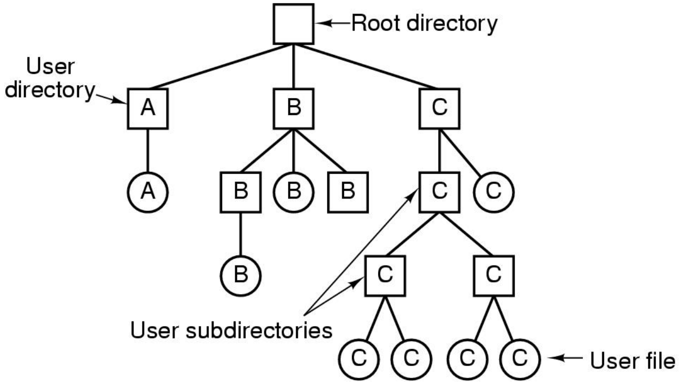
- To manage volume of info – partitions & directories
- Directory: set of nodes with information about all files
  - Name (inode as well), type, address, current & max. length, …
- Operations on directories
  - Open/close directories, create/delete/rename files from a directory, readdir, link/unlink, traverse the file system
- Directory organizations - goals
  - Efficiency – locating a file quickly.
  - Naming – convenient to users.
  - Grouping – logical grouping of files by properties (e.g. all Java progs., all games, …)

Single and hierarchical directory systems

- A single level directory system
  - Early PCs & supercomp. (CDC 6600), embedded systems?
  - Fast file searches, but name clashing



- Hierarchical
  - Avoid name clashing for users (MULTICS)
  - Powerful structuring tool for organization (decentralization)



- Path names
  - Now you need a way to specify file names; two approaches: absolute and relative
  - "." & ".."

The file system interface – files

- Creating files

  ```
  int fd = open("foo", O_CREAT | O_WRONLY | O_TRUNC);
  ```

  equivalent to

  ```
  int fd = creat("foo");
  ```

- Reading and writing
  - Check what's going on using strace

```
fabianb@eleuthera:~$ echo hola > foo
fabianb@eleuthera:~$ cat foo
```

```
fabianb@eleuthera:~$ strace cat foo
execve("/bin/cat", ["cat", "foo"], [/* 22 vars */]) = 0
…
open("foo", O_RDONLY)                    = 3
fstat(3, {st_mode=S_IFREG|0664, st_size=5, ...}) = 0
fadvise64(3, 0, 0, POSIX_FADV_SEQUENTIAL) = 0
read(3, "hola\n", 32768)                 = 5
write(1, "hola\n", 5hola)                = 5
read(3, "", 32768)                       = 0
close(3)                                 = 0
close(1)                                 = 0
close(2)                                 = 0
```

Just for reading

Returns a file descriptor; why 3?

Size of buffer

4 of "hola" + eol

Writing it out

And close all

The file system interface – files

- Reading and writing files
  - … not sequentially

```
off_t lseek(int fildes, off_t offset, int whence);

If whence is SEEK_SET, offset is set to offset bytes
… is SEEK_CUR, … to its current location plus offset bytes
… is SEEK_END, … to the size of the file plus offset bytes
```

Always from a "current" offset; changed explicitly by `lseek`

  - … immediately, `fsync()`
- Renaming – what do you think `mv` does?

```
fabianb@eleuthera:~$ strace mv foo foo2
execve("/bin/mv", ["mv", "foo", "foo2"], [/* 22 vars */]) = 0
…
lstat("foo", {st_mode=S_IFREG|0664, st_size=5, ...}) = 0
lstat("foo2", 0x7fff0c0da550)          = -1 ENOENT (No such
file or directory)
rename("foo", "foo2")                  = 0
```

The file system interface – files

- Getting info about files

```
struct stat {
  dev_t     st_dev;     /* ID of device containing file */
  ino_t     st_ino;     /* inode number */
  mode_t    st_mode;    /* protection */
  nlink_t   st_nlink;   /* number of hard links */
  uid_t     st_uid;     /* user ID of owner */
  gid_t     st_gid;     /* group ID of owner */
  dev_t     st_rdev;    /* device ID (if special file) */
```

```
fabianb@eleuthera:~$ stat foo2
  File: `foo2'
  Size: 5               Blocks: 8          IO Block: 4096   regular file
Device: 801h/2049d      Inode: 4793831     Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/ fabianb)   Gid: ( 1000/ fabianb)
Access: 2013-11-26 05:34:56.000000000 -0600
Modify: 2013-11-26 05:34:53.000000000 -0600
Change: 2013-11-26 05:56:02.000000000 -0600
```

- Removing files

```
fabianb@eleuthera:~$ strace rm foo2
…
unlinkat(AT_FDCWD, "foo2", 0)          = 0
```

The file system interface – directories

- Making directories - `mkdir`
- Reading directories – what `ls` does

```
#include <stdio.h>
#include <assert.h>
#include <sys/types.h>
#include <dirent.h>

int main(int argc, char *argv[])
{
  DIR *dp = opendir(".");
  assert(dp != NULL);
  struct dirent *d;
  while ((d = readdir(dp)) != NULL) {
    printf("%d %s\n", (int) d->d_ino, d->d_name);
  }
  closedir(dp);
  return 0;
}
```

```
fabianb@eleuthera:~/scratch$ ./sillyls
5382152 sillyls.c~
5382148 .
5382351 ..
5382150 sillyls
5382149 foo
5382151 sillyls.c
```

The file system interface – directories

- Deleting directories – `rmdir()`
- Hard links – back to the unlink idea
  - Making links

```
fabianb@eleuthera:~/scratch$ ln foo foo2
fabianb@eleuthera:~/scratch$ cat foo2
Hola
```

```
fabianb@eleuthera:~/scratch$ ls -i foo foo2
5382149 foo  5382149 foo2
```

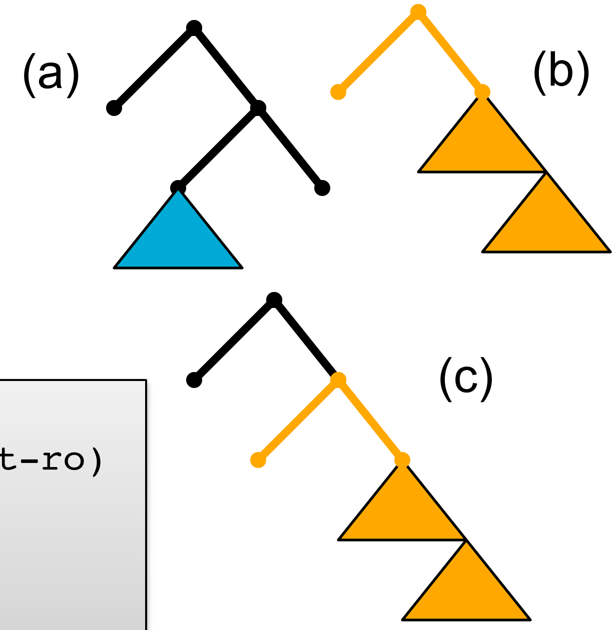
  - Creating a file – making an inode *and* linking it to a human-readable name and putting the name in a directory
  - Try creating a few links to a file, run stat, and delete them ....
- Symbolic or soft links
  - The problem with hard links and `ln -s`

```
fabianb@eleuthera:~/scratch$ ln -s foo foo4
fabianb@eleuthera:~/scratch$ ls -al foo*
-rw-rw-r-- 3 fabianb fabianb 5 Nov 16 06:51 foo
-rw-rw-r-- 3 fabianb fabianb 5 Nov 16 06:51 foo2
-rw-rw-r-- 3 fabianb fabianb 5 Nov 16 06:51 foo3
lrwxrwxrwx 1 fabianb fabianb 3 Nov 26 06:24 foo4 -> foo
```

File system mounting

- A FS must be mounted to be available
  - What if you have more than one disk? Put a self contained FS on each (*C:…)* or…
- Mounting – typically a mount point is an empty dir
  - Existing file system (a) & unmounted partition (b)
  - After it was mounted (c)

```
$ mount /dev/sda1 /users
```



- `fstab` file in Unix

```
fabianb@eleuthera:~$ mount
/dev/sda1 on / type ext3 (rw,relatime,errors=remount-ro)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw,noexec,nosuid,nodev)
none on /sys/fs/fuse/connections type fusectl (rw)
none on /sys/kernel/debug type debugfs (rw)
..
```

And now a short break …

xkcd

File systems implementation

- How can we build a simple file system? What data structures are needed? And how do you use them to keep track of things?
- Keeping track of
  - free blocks
  - what blocks go with which file
  - File names, attributes and directory hierarchy
  - And who has what access right to what objects

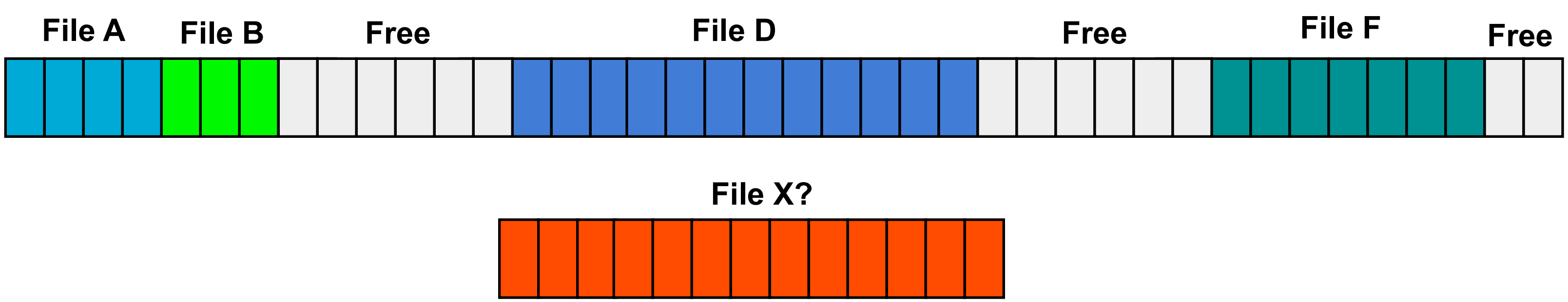
Disk space management

- Keeping track of free blocks
  - Storing the free list on a linked list
    - Use a free block for the linked list (holding as many free disk block numbers as possible)
  - A bit map (only one bit per block)
  - *When would the linked list require fewer blocks than the bitmap?*
    - Only if the disk is nearly full
- And if you tend to run out of free space, control usage
  - Quotas for user's disk use
  - Open file entry includes pointer to owner's quota rec.
  - Soft limit may be exceeded (warning)
  - Hard limit may not (log in blocked)

Implementing files – Contiguous allocation

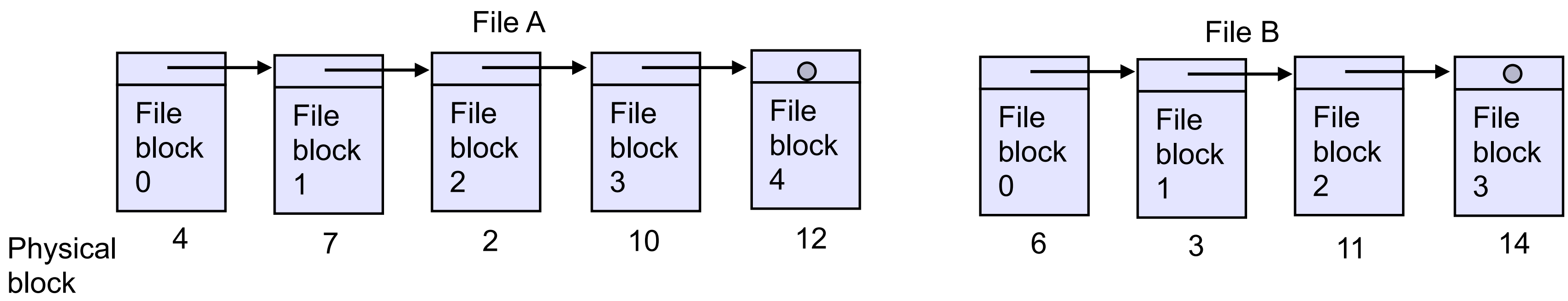
- Contiguous allocation
  - Each file is a contiguous run of disk blocks
  - e.g. IBM VM/CMS
  - Pros:
    - Simple to implement
    - Excellent read performance
  - Cons:
    - Fragmentation

Where would it make sense?



Implementing files – Linked lists

- Linked list
  - Files as a linked list of blocks
  - Pros:
    - Every block gets used
    - Simple directory entry per file (address of first block)
  - Cons:
    - Random access is a pain
    - List info is in block → block data size not a power of 2
    - Reliability (file kept together by pointers scattered throughout the disk)



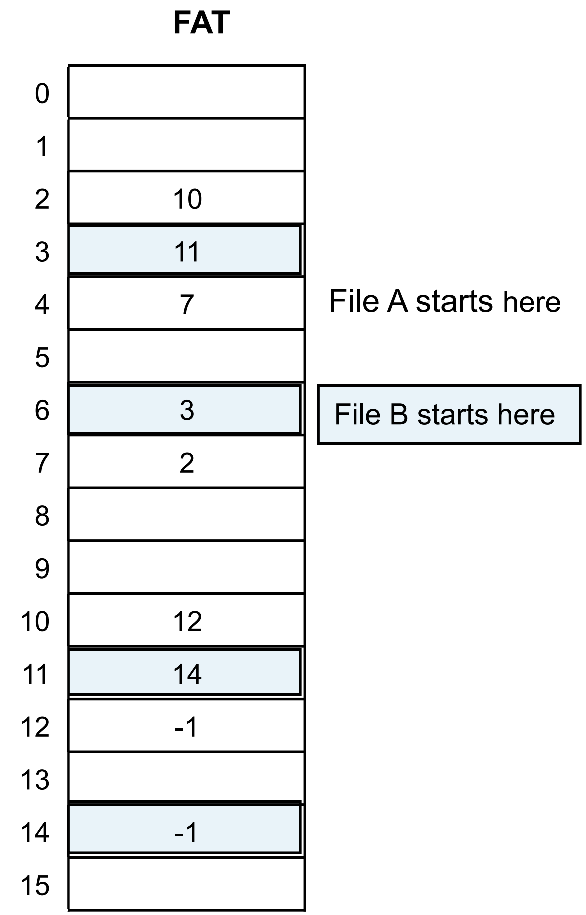
Implementing files – FAT

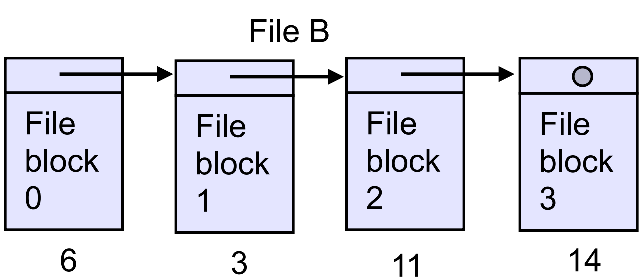
- Linked list with a table in memory
  - Files as a linked list of blocks
  - Pointers kept in FAT (File Allocation Table)
  - Pros:
    - Whole block free for data
    - Random access is easy
  - Cons:
    - Overhead on seeks or
    - Keep the entire table in memory

      20GB disk & 1KB block size →

      20 million entries in table →

      4 bytes per entry ~ 80MB of memory

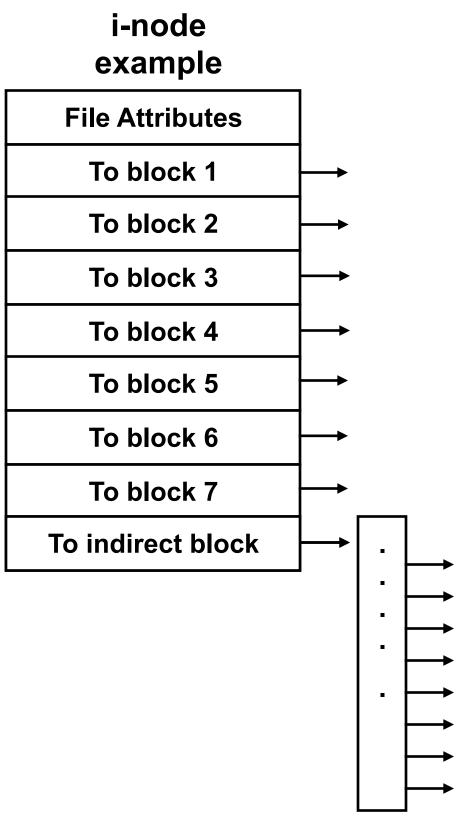




Implementing files – i-nodes

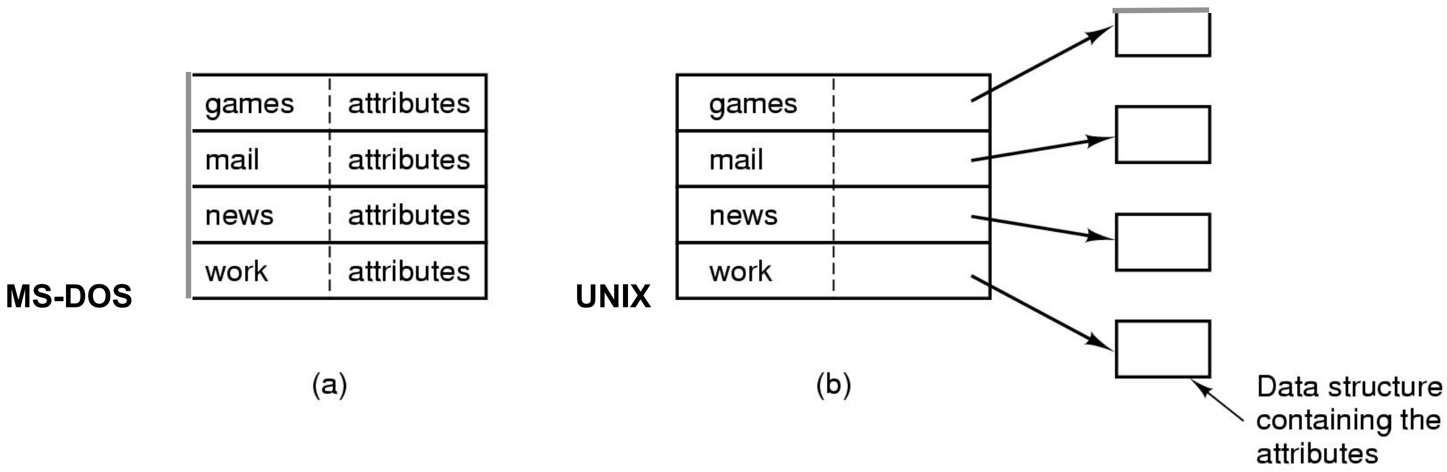
- I-nodes - index-nodes
  - Files as linked lists of blocks, all pointers in one location: i-node
  - Each file has its own i-node
  - Pros:
    - Support direct access
    - No external fragmentation
    - Only a file i-node needed in memory (proportional to # of open files instead of to disk size)
  - Cons:
    - Wasted space (how many entries?)
  - More entries – what if you need more than 7 blocks?

Save entry to point to address of block of addresses



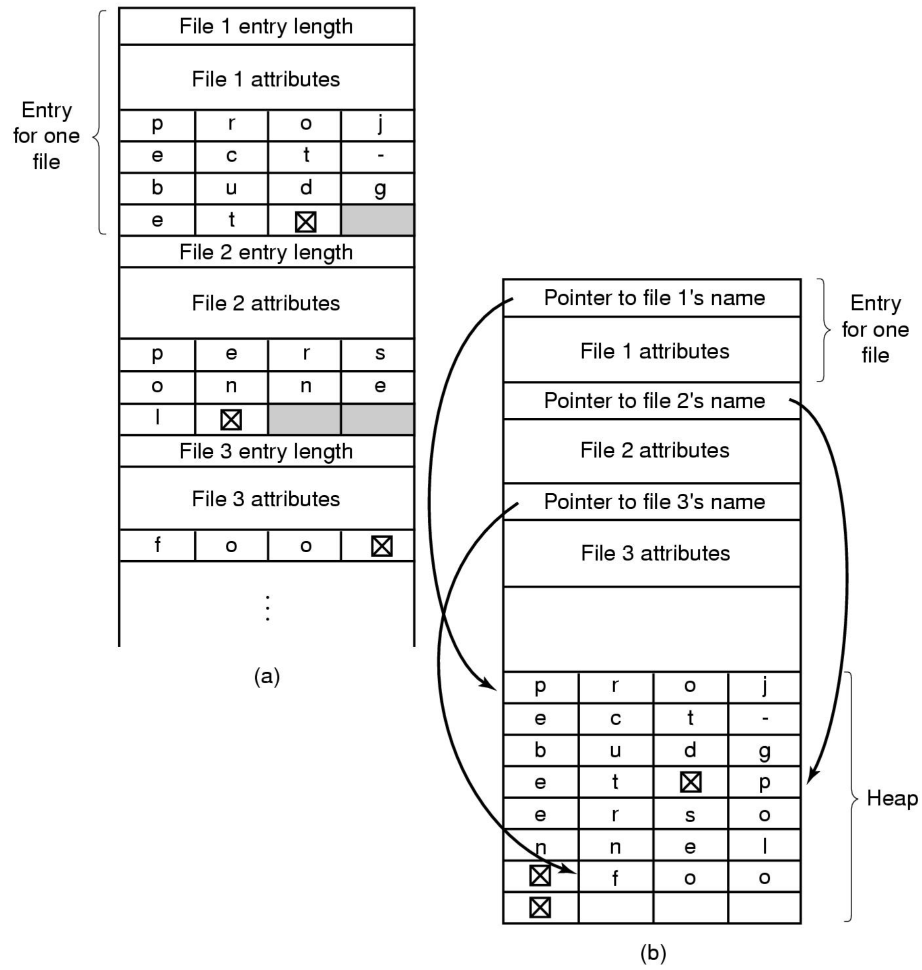
Implementing directories

- Directory system function: map ASCII name onto what's needed to locate the data
- Related: where do we store files' attributes?
  - A simple directory: fixed size entries, attributes in entry (a)
  - With i-nodes, use the i-node for attributes as well (b)
- As a side note, you find a file based on the path name; this mixes what your data is with where it is – *what's wrong with this picture?*



Implementing directories

- So far we've assumed short file names (8 or 14 char)
- Handling long file names in directory
  – In-line (a)
    - Fragmentation
    - Entry can span multiple pages (page fault reading a file name)
  – In a heap (b)
    - Easy to +/- files
- Searching large directories
  – Hash
  – Cash



Protection …

- FS must implement some kind of protection system
  - To control who can access a file (user)
  - To control how they can access it (e.g., read, write, or exec)
- More generally
  - Generalize files to objects (the “what”)
  - … users to principals (the “who”, user or program)
  - … read/write to actions (the “how”, or operations)
- A protection system dictates whether

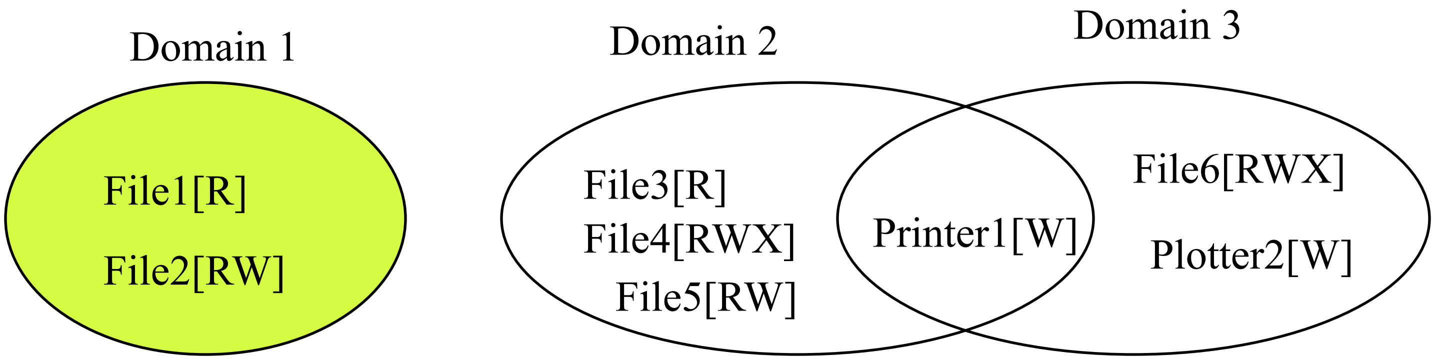
  a given *action* performed by

  a given *principal* on

  a given *object* should be allowed
  - e.g., you can read or write your files, but others cannot

Protection …

- Useful to discuss protection mechanisms: domains
  - A domain – a set of *(object, rights)* pairs
  - At every instant in time, process runs in some domain
    - In Unix, this is defined by (UID, GID); exec a process with SETUID or SETGID bit on is effectively switching domains



Protection domains

- Keeping track of domains; conceptually, a large protection matrix

Domain	File1	File2	File3	File4	File5	File6	Printer1	Plotter1
1	**Read**	**Read Write**						
2		Read	Read	Read Write Execute	Read Write		Write	
3						Read Write Execute	Write	Write

- A protection matrix with domains as objects
  - Now you can control domain switching

Domain	File1	File2	File3	File4	File5	File6	Printer1	Plotter1	Domain1	Domain2	Domain3
1	**Read**	**Read Write**								**Enter**	
2		Read	Read	Read Write Execute	Read Write		Write				
3						Read Write Execute	Write	Write			

Protection – Access Control Lists

- Implementing access matrix – too large & sparse …
- Access control list
  – Associating with each object a list of domain that may access it (and how)
  – Users, groups and roles

Domain	File1	File2	File3	File4	File5	File6	Printer1	Plotter1	Domain1	Domain2	Domain3
1	**Read**	**Read Write**								**Enter**	
2		Read	Read	Read Write Execute	Read Write		Write				
3						Read Write Execute	Write	Write			

```
File1 → D1: R
File2 → D1: RW; D2: R
File3 →
```

Protection – Capabilities

- Capabilities
  - Slice the matrix by rows – a list of objects & rights per domain

Domain	File1	File2	File3	File4	File5	File6	Printer1	Plotter1	Domain1	Domain2	Domain3
1	**Read**	**Read Write**								**Enter**	
2		Read	Read	Read Write Execute	Read Write		Write				
3						Read Write Execute	Write	Write			

`Domain1` → `F1: R; F2: RW; D2:Enter`

  - Need to protect the C-list
    - Tagged architectures (IBM AS/400)
    - Keep it in the kernel (Hydra)
    - Manage them cryptographically (Amoeba)

- *Faster to use but do no support selective revocation*

Protection in Unix

- Unix: short version access lists & groups
  - Objects – individual files; Principals – owners/group/world
  - Actions: read, write, execute (3 bits per access mode)
  - Mask provides a default (creation with 777, mask 022 → 755)
- More general access lists - setfacl & getfacl

```
fabianb@eleuthera:~/scratch$ ls -l foo
-rw-rw-r-- 3 fabianb fabianb 5 Nov 16 06:51 foo

fabianb@eleuthera:~/scratch$ getfacl foo
# file: foo
# owner: fabianb
# group: fabianb
user::rw-
group::rw-
other::r--
```

- Granting an additional user read access
  - But check acl is set in fstab!

```
fabianb@eleuthera:~/scratch$ setfacl -m u:jeanine:r foo
```

Next Time

- Details on file system implementations and some examples …