ELEC 377 – Operating Systems

Week 8 – Class 1

Last Class

- Shell Scripting

Admin

- No class next Monday or Tuesday
- There IS A lab, still: Lab 4 – Part 1
- Quiz #3 moved to Thursday November 8th

Today

- File Systems
  - ◊ Concept
  - ◊ Attributes, Operations, Types
  - ◊ Structure
  - ◊ Access
  - ◊ Directory Structure

File System

- Abstract Layer above secondary storage
  - ◊ don’t worry about physical characteristics of actual media
  - ◊ IDE, SCSI, SNA, doesn’t matter
- Smallest allotment of secondary storage
  - ◊ all user data is stored in files
  - ◊ actual representation of storage is usually left to operating system

File Attributes

- Name
- Identifier
  - unique tag, identifies the file within the file system
  - real name of the file
- Type
- Location
- Size
- Protection
  - read/write/execute (UNIX)
  - access control lists
- Time, date, user identification
  - security

File Operations

- Create
- Write
- Read
- Reposition (seek)
- Delete
- Truncate
- Open

  - implicit/explicit

  - count (per process/ system wide)
- Close

File Types

- All files have some type associated with them
- OS enforced
  - ◊ TOPS, VMS, MVS
  - ◊ name interaction
- Program Enforced
  - ◊ UNIX, MS-DOS, Windows
  - ◊ File name extension is convention; only some are enforced (.EXE, .BAT)
- Mixed
  - ◊ Mac OS

File Structure

- None (Unix/MS-DOS)
  - ◊ Some structure recognized (executable file, directory)
  - ◊ sequence of bytes
- Record Structure
  - ◊ line
  - ◊ fixed/variable
- Mapping to Physical Devices
  - ◊ Usually OS
  - ◊ IBM Mainframes - can specify physical format of files
  - ◊ logical record size, block size
  - ◊ fragmentation

File Access

- Sequential (simple)
  - ◊ read (updates i/o pointer)
  - ◊ write (appends)
  - ◊ tape model
- Direct Access
  - ◊ read *n*
  - ◊ write *n*
  - ◊ goto *n*
  - ◊ read next
  - ◊ write next

  n is relative block number

Indexing

- Used for database style applications
- IBM BDAM/ISAM
- Implemented in libraries for OS that don’t have record based files (dbm, gdbm on Unix, etc.)
- one file contains keys and relative block numbers of records in the file that contains the actual data
  - ◊ may be more than one level
  - ◊ top level stays in memory
  - ◊ don’t have to search the entire file, only a few blocks

Directory Structure

- Organize Files
- Partitions
- Directory maps file names to physical files
  - stores attributes
- Operations
  - search for a file
  - create a file
  - delete a file
  - list
  - rename a file
  - traverse file system

Directory Structure

- Single Level Directory
  - ◊ Commodore 64, Early Dos, TRS-80, Apple II
  - ◊ Organization difficult (small disks)

- Two Level Directory
  - ◊ top level is directory
  - ◊ second level is user file directory
  - ◊ files are in user directories
  - ◊ path concept
  - ◊ search paths
  - ◊ little grouping capability

Directory Structure

- Tree Structured
  - ◊ what we are all familiar with
- MS-DOS, Windows, Unix, Macintosh
- Current Directory (working directory)
  - ◊ unique/per disk
- relative and absolute path names
  - ◊ c:foo.txt ../a/b/c
  - ◊ c:\foo.txt /usr/local/bin/acroread
- long path names
- search paths

Directory Structure

- Acyclic – shared subdirectories
- More than one path to a directory or file
- Unix
  - ◊ files can be shared, directories not shared
  - ◊ directories have a unique parent
  - ◊ symbolic links
    - –file with special attributes
    - –contains path (relative or absolute) to real file or directory
- Acyclic restriction allows sharing, but simplifies traversal
- General graph directories possible, but not really used

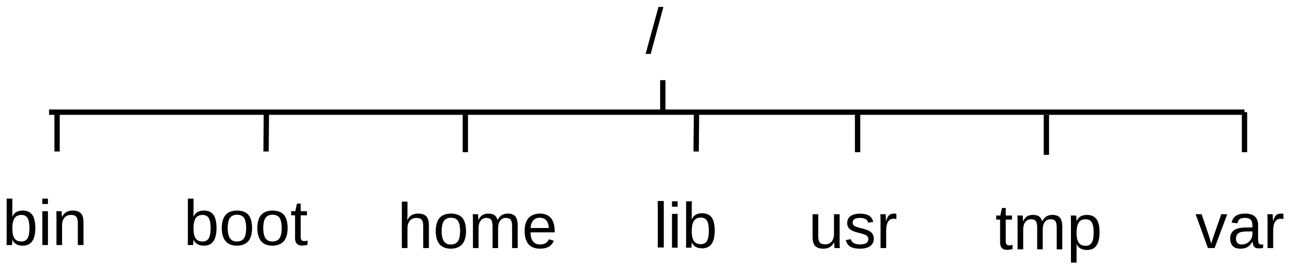
Mounting

- Before a directory is accessible it must be mounted
- Operating system checks the disk to make sure it has a valid file system on it (corrupted disk, uninitialized disk)
- Loads information about the file system into internal structures for future access
  - ◊ sets up buffers
- Automatic in some operating systems (Mac, MS-DOS)
  - ◊ when media is detected (Mac)
  - ◊ when file system access is attempted (Microsoft)

Mounting - explicit

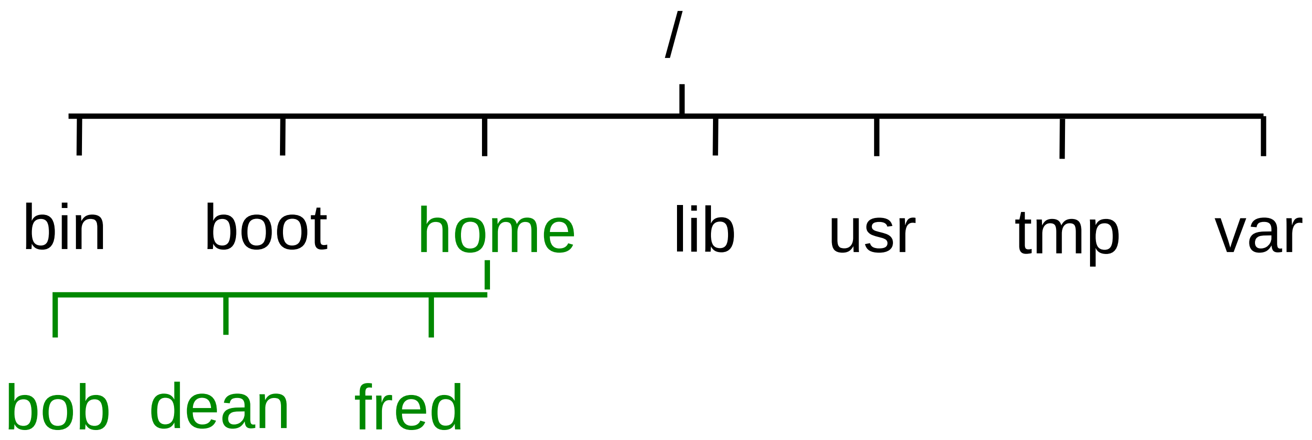
- UNIX - mounting is explicit
  - ◊ ftab file
- A file accessed through a *mount point*
  - ◊ Mount points in Windows are drive letters
  - ◊ Mount points on Mac and Amiga are file system volume names (both are two level)
  - ◊ Mount points in Unix are directories
  - ◊ First mounted system is mounted at location ‘/’ - known as the *root* of the file system
  - ◊ Any directory can be used as a mount point
  - ◊ traditional mount points are /, /usr, /usr/local, /home, /tmp, /var

Mounting - Unix



/bin/mkdir

Mounting - Unix



/

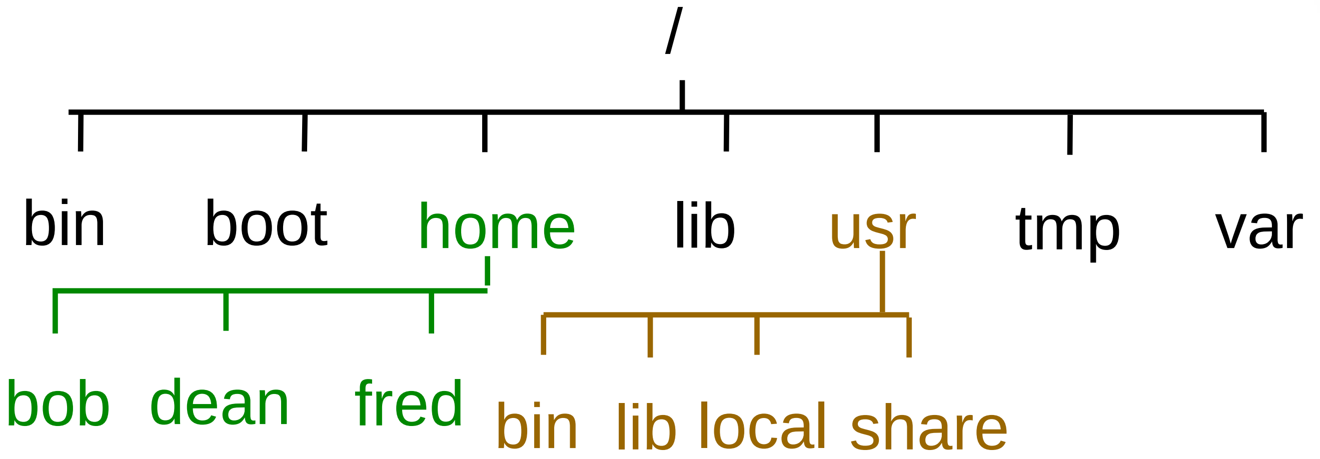
bin boot home lib usr tmp var

bob dean fred

/bin/mkdir

/home/dean/assignment1/ass1.java

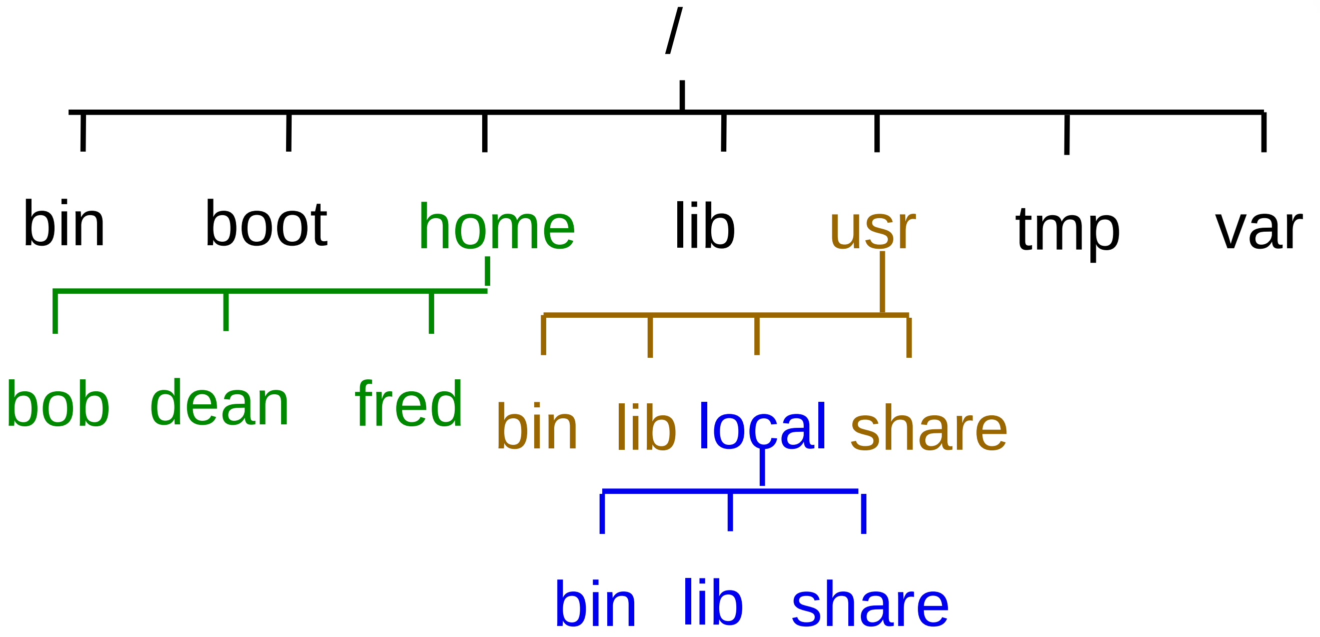
Mounting - Unix



/bin/mkdir

/home/dean/assignment1/ass1.java

Mounting - Unix

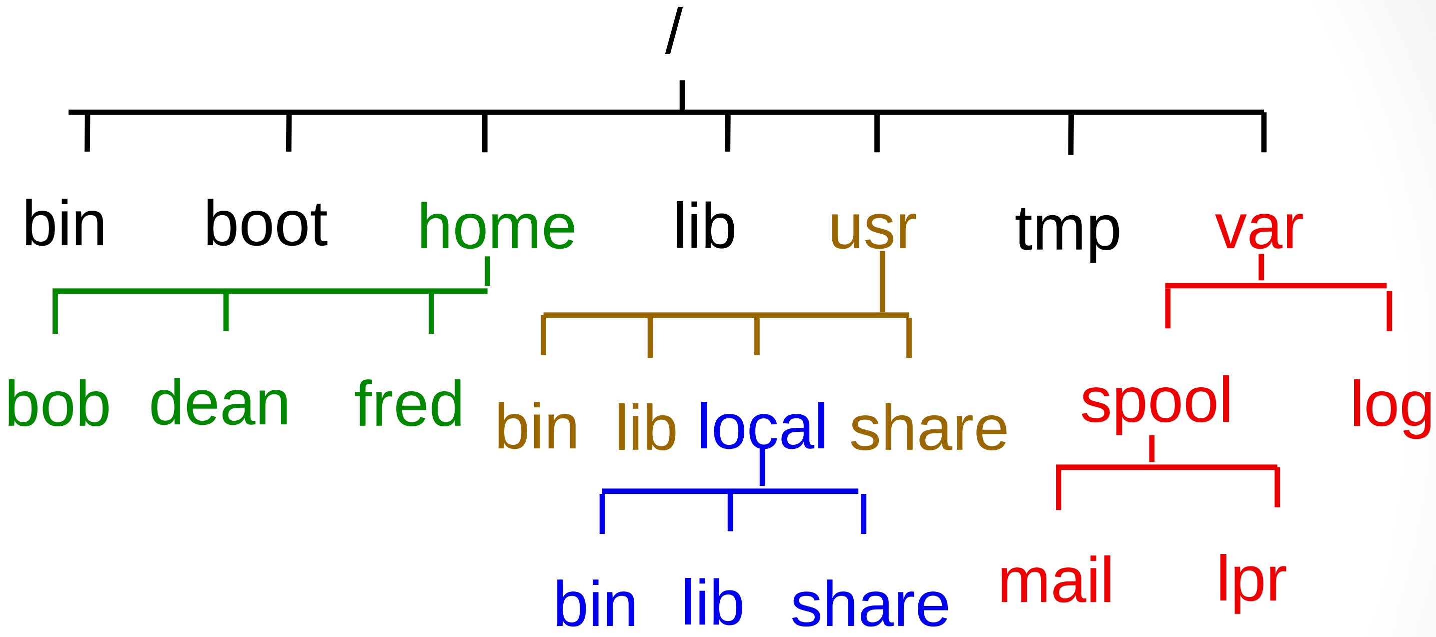


/bin/mkdir

/home/dean/assignment1/ass1.java

/usr/local/bin/

Mounting - Unix



/bin/mkdir

/home/dean/assignment1/ass1.java

/usr/local/bin/

/var/log/httpd/access.log

File System Implementation

- Most operating systems support more than one file system
  - ◊ Windows 2000 – FAT, NTFS, FAT32, ISO9660
  - ◊ MacOS - HFS, HFS+, FAT, FAT32, ISO9660
  - ◊ Linux - FAT, NTFS, HFS, FAT, FAT32, ISO9660, HPFS, others

File System Implementation

- The file system is composed of several levels
  - ◊ logical file system

    - directory structure, protection, permission
  - ◊ file organization level

    - logical block management
  - ◊ basic file system

    - reads and writes blocks
  - ◊ I/O control
  - ◊ devices

On Disk Structures

- boot control block
  - ◊ block containing code to start the operating system
  - ◊ small-not big enough to hold operating system
  - ◊ On PCs also contains the partition table for the disk
- partition control block
  - ◊ controls information inside of the partition
  - ◊ number free block list and counters
  - ◊ number and size of blocks
  - ◊ superblock (Unix), MasterFileTable (ntfs)
- Directory Structure
- File Control Block (1 for each file)
  - ◊ contains information about the file
  - ◊ NTFS stores in Master File Table

In Memory Structures

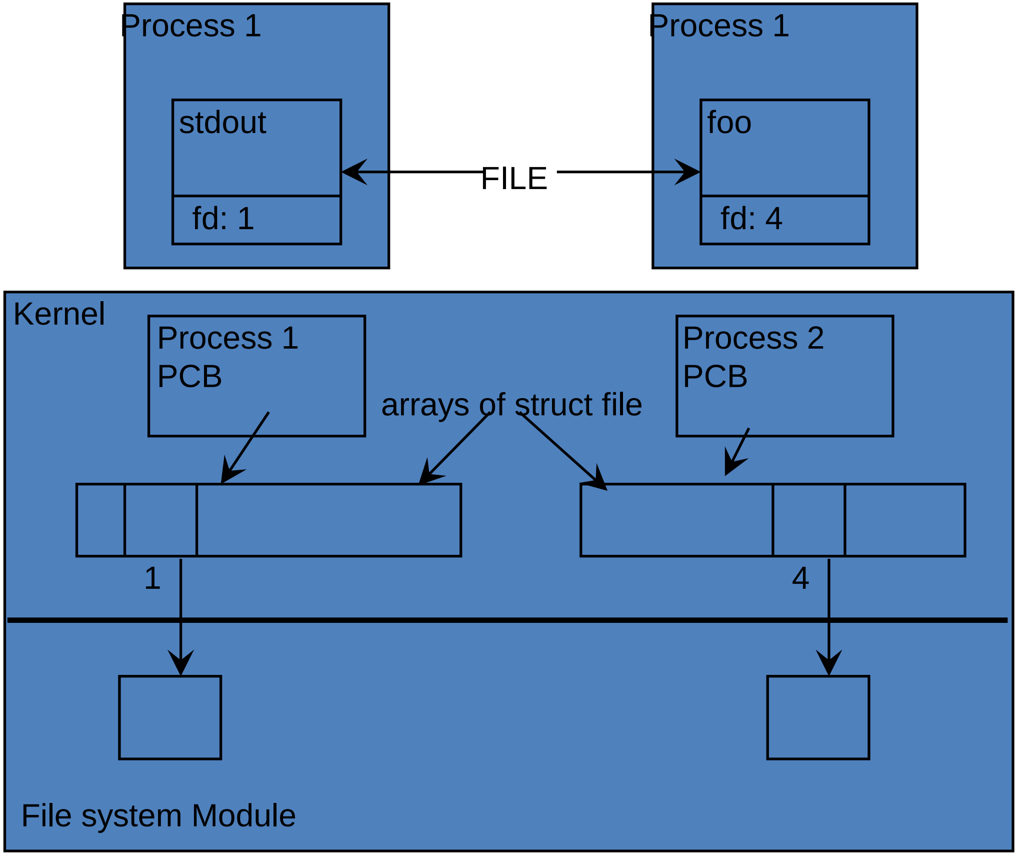
- Partition Control Block
  - ◊ for mounted file systems
  - ◊ may have more information than on disk version
- Directory information
  - ◊ current directory
  - ◊ recent directories (locality of reference)
- System wide file table
  - ◊ every open file
  - ◊ other accounting information
- Per process file table
  - ◊ points to system wide file table
  - ◊ buffers and current file position for this process

Opening a File

- Search Directory Structure
  - ◊ may be cached in memory (load if not)
  - ◊ find file identifier
- See if FCB is already in memory (System FCB table)
  - ◊ copy into memory if not
  - ◊ do any special inits (such as truncate file on open for write)
  - ◊ increment counter
- Allocate entry in process FCB table
  - ◊ point to System FCB table
  - ◊ allocate buffers
  - ◊ initialize current file position

!return pointer or index to process FCB table.!

Opening a File



Virtual File Systems

- As mentioned on the first slide, most operating systems support more than one file system
  - ◊ multiple local file systems
  - ◊ multiple network file systems
- Virtual File System
  - ◊ layer above file system
  - ◊ maps file system specific view to operating system view
- Unix inode concept
  - ◊ does not exist in SMB file sharing (Windows)
  - ◊ SMB to VFS interface requires generation and caching of inode information

Directory Implementation

- Linear list of names with reference to data block
  - ◊ simple
  - ◊ time-consuming for large directories

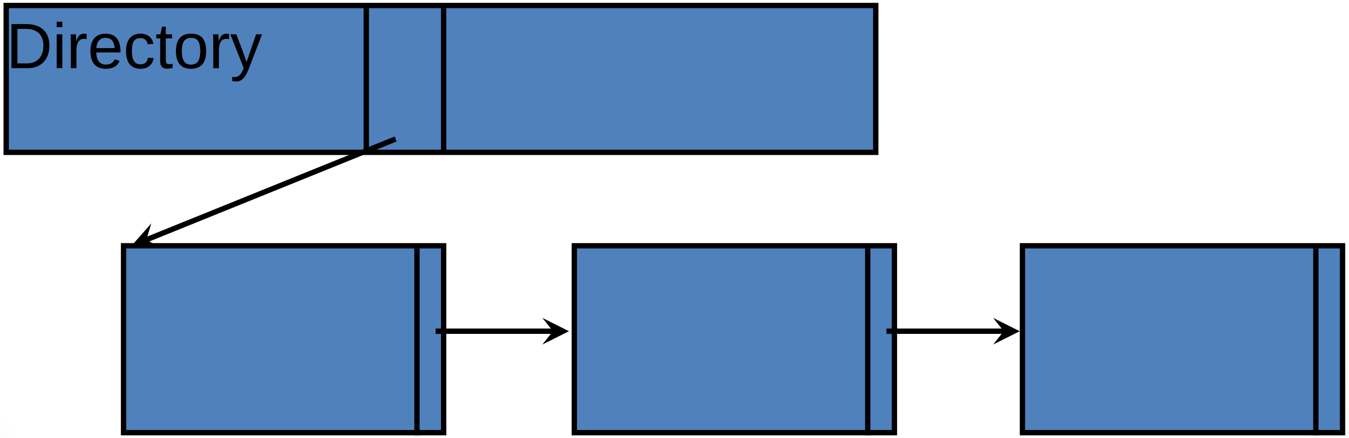
- Hash Table
  - ◊ decrease directory search time
  - ◊ fixed size
  - ◊ collisions

Allocating Disk Blocks to Files

- Contiguous
  - ◊ IBM VM/CMS - Data Set, Partitioned Data Set
  - ◊ blocks for a file are contiguous
  - ◊ directory contains starting block, length
  - ◊ fast for read/write
  - ◊ direct access is easy
  - ◊ problems with size, fragmentation (same as memory)

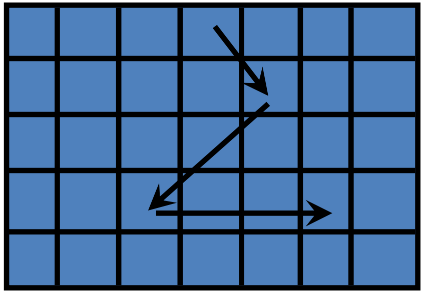
Allocating Disk Blocks to Files

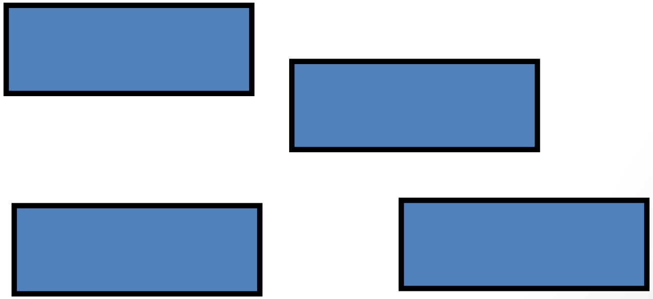
- Linked Allocation
  - ◊ directory gives first and maybe last block in file
  - ◊ each block has a pointer to next block
  - ◊ less data stored in each block (alt. FAT)
  - ◊ direct access is not nearly as easy (follow chain)



Allocating Disk Blocks to Files

- Linked Allocation
  - ◊ Fat Allocation
  - ◊ blocks grouped together into clusters (MS-DOS)
  - ◊ 16 bit - 65536 clusters
  - ◊ 128K reserved for link table (small enough to keep in ram)
  - ◊ table is indexed by block number, and gives the next block in the chain
  - ◊ data block now contains only data

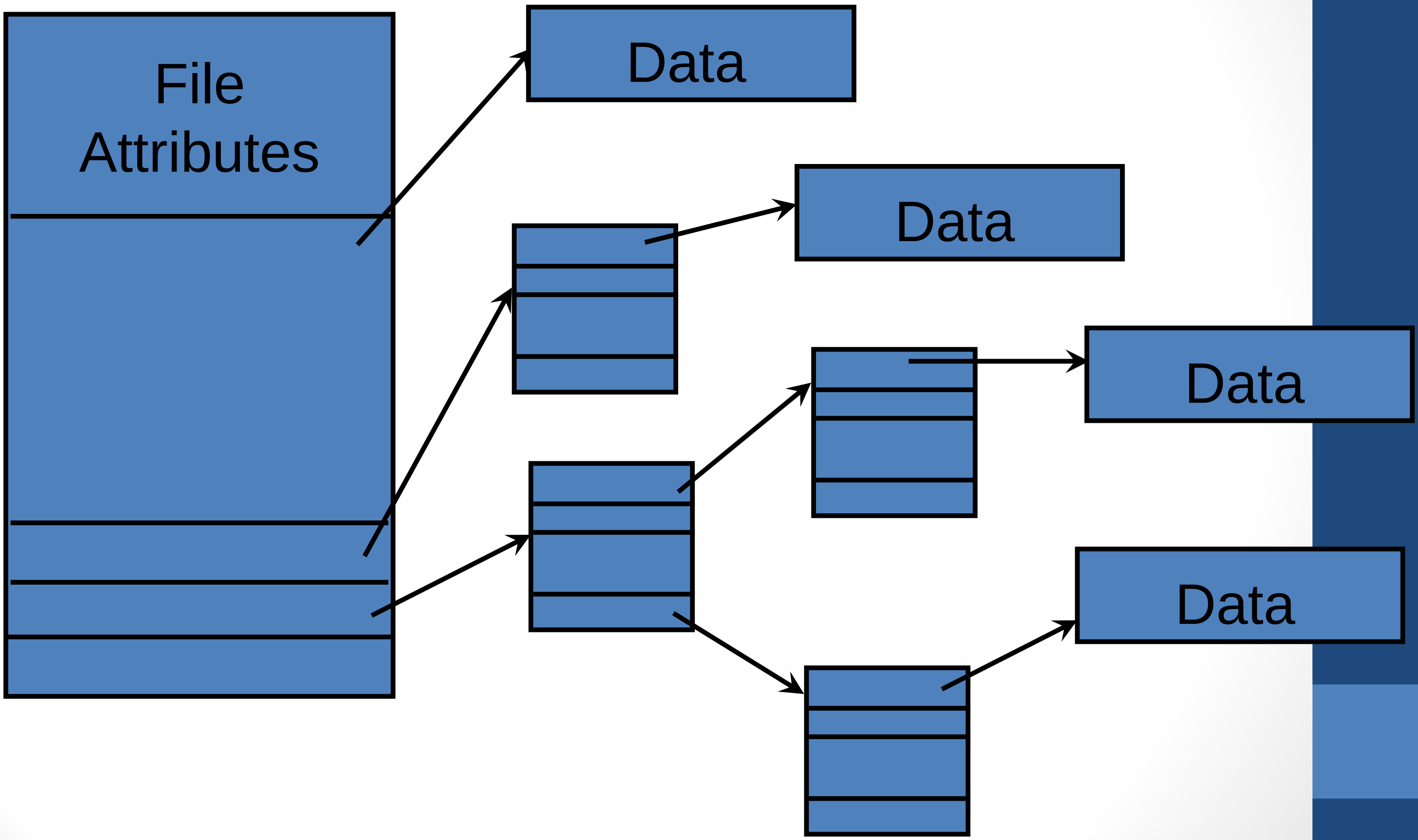




Allocating Disk Blocks to Files

- Indexed Allocation
  - ◊ Use one or more disk block for the file that contains the pointers to the data bocks
  - ◊ more overhead
  - ◊ direct access into file is easy
  - ◊ May need more than one index block
    - linked list
    - tree (internal nodes are index, leafs are data)
    - combined (Unix)
    - 80 - 20 distribution

Allocating Disk Blocks to Files


File Attributes
Data
Data
Data
Data