ELEC 377 – Operating Systems

Week 8 – Class 2

Last Class

- File Systems
  - ◊ Directory Structure
  - ◊ Mounting
- File System Implementation
  - ◊ Introduction

Admin

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

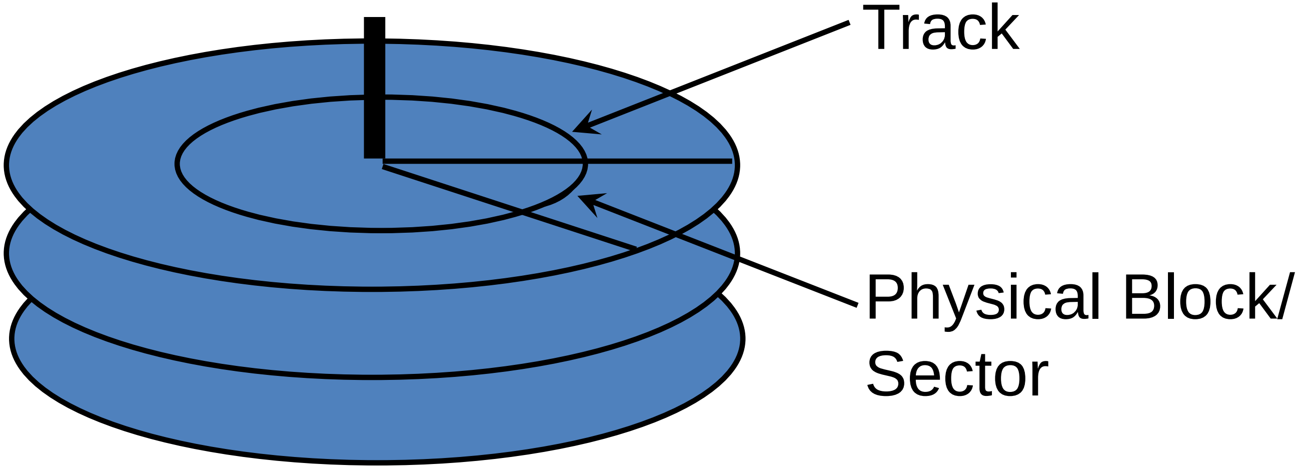
Today

- File System Implementation
- I/O Hardware

File System Implementation

- The file system is composed of several levels
  - ◊ logical file system
    - directory structure, protection, permission
    - metadata
  - ◊ file organization level
    - logical block management
  - ◊ basic file system
    - reads and writes blocks
  - ◊ I/O control
  - ◊ devices

Disk Devices



- Cylinder - the same track on all platters

Disk Drives

- logical blocks (Macintosh), or clusters (MS-DOS), is operating system concept
  - ◊ disks are divided into partitions (usually on cylinder boundaries)
  - ◊ blocks within partition are numbered by most operating systems
    - logical block number
  - ◊ HFS and FAT limited to 16 bit numbers - 65536 logical blocks
  - ◊ 2 G partition has 4294967296 physical blocks
  - ◊ 16K physical blocks per logical block
  - ◊ logical block for 2G partition is 8K in size

On Disk Structures

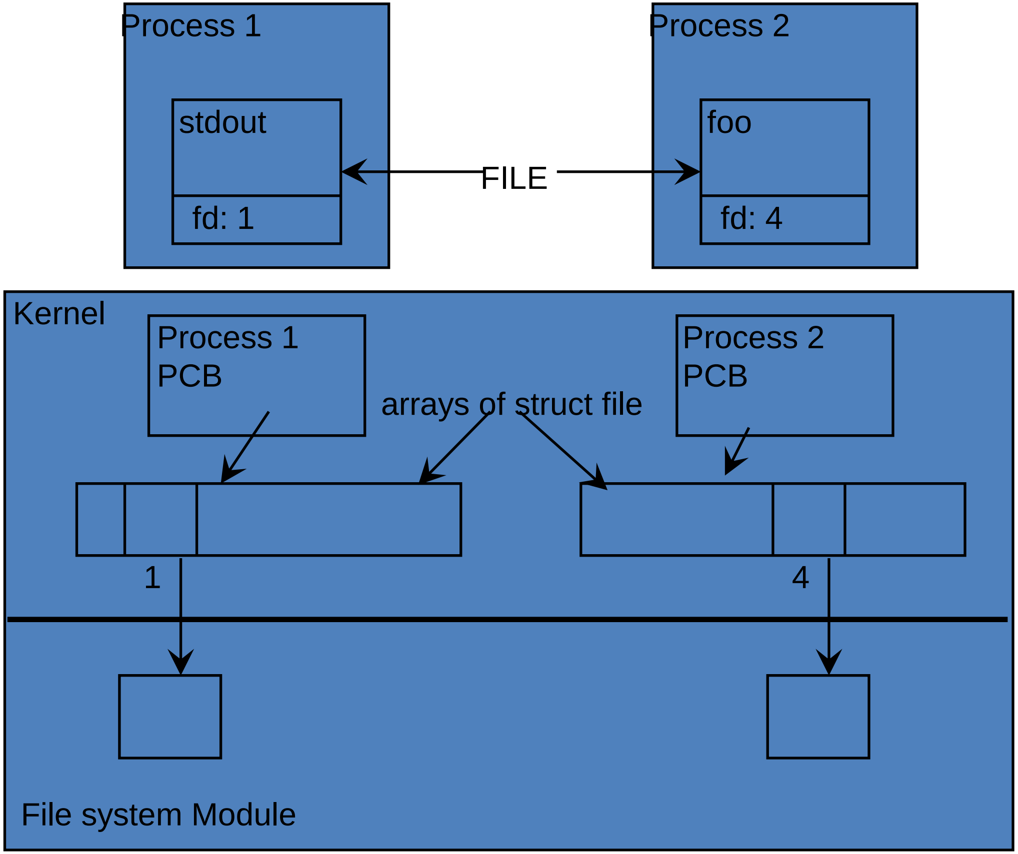
- boot control block
  - ◊ block containing code to start the operating system
  - ◊ small- not big enough to hold operating system
- partition control block
  - ◊ controls information inside of the partition
  - ◊ free block list and counters
  - ◊ number and size of blocks
  - ◊ superblock (Unix),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

Opening a File

- Search Directory Structure
  - ◊ may be cached in memory
  - ◊ find file
- 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



Sequential Reading From a File

- Use pointer or index to access process FCB table
- Is current position pointer at end of the current block buffered in memory?
  - ◊ copy data to user buffer, if not
  - ◊ if more data, or the position pointer is at end of the currently buffered block, ask file system for next block
  - ◊ use pointer from process FCB to get System FCB
    - for continuous files, read next block on disk
    - for linked files, find the link and access
    - for indexed table go to index node (in memory) and get next block
  - ◊ return pointer-buffered block to process level
  - ◊ copy rest of data to user memory.

Closing a File

- use pointer or index to access process FCB
- use pointer from process FCB to find system FCB
  - ◊ decrement counter
  - ◊ if counter is 0 (last process with file open) then mark FCB for deletion
    - will be flushed when out of system FCBs and we have to reuse system FCBs. Otherwise still in memory if another file opens it before we reclaim the block
    - lazy tends to do well in OS.
- free process FCB

Virtual File Systems

- As mentioned earlier, 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
- B-Tree
  - ◊ stores indexed records on disk (Index files)
  - ◊ records contain a parent id (used to build the directory hierarchy)

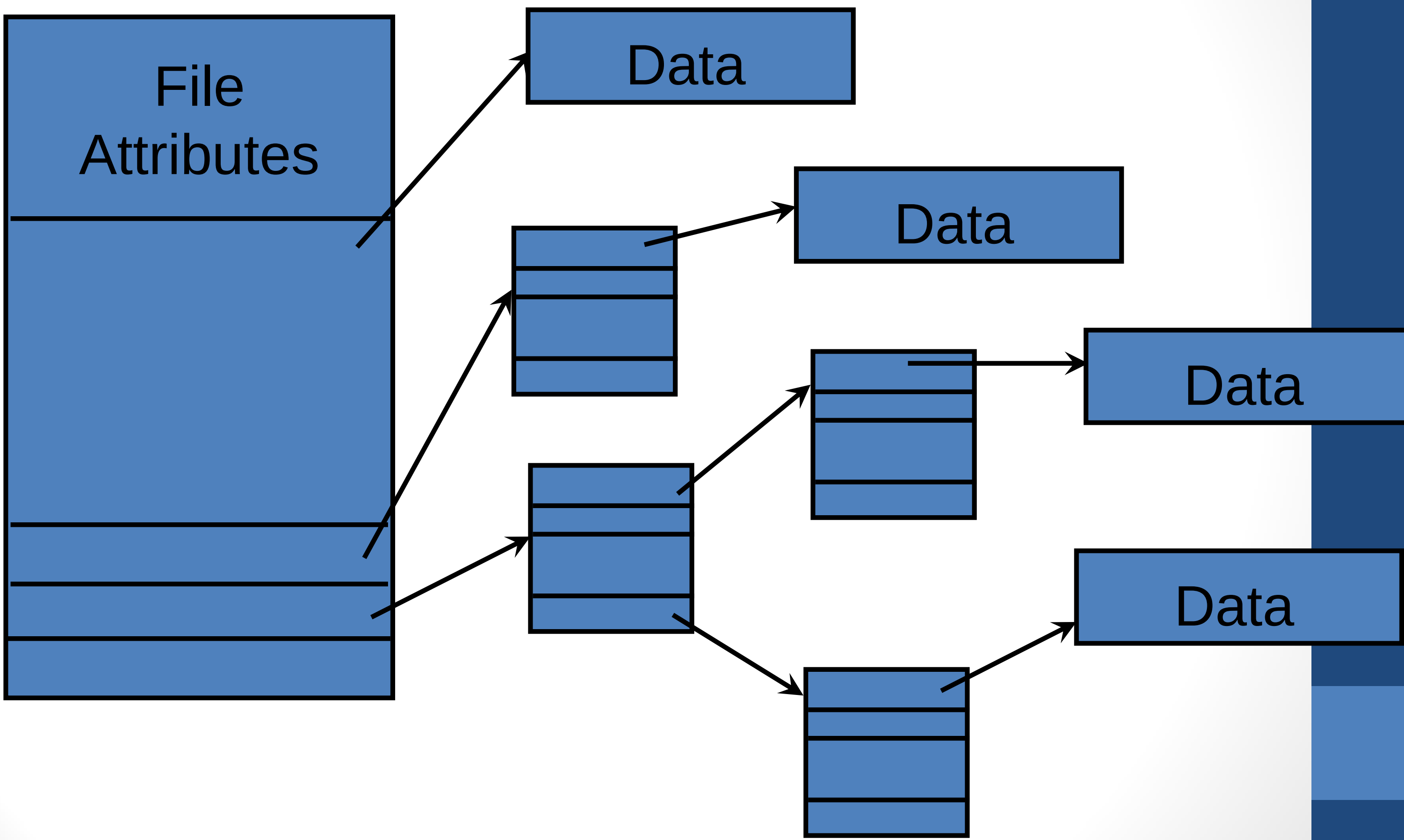
Allocating Disk Blocks to Files

- 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

- 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

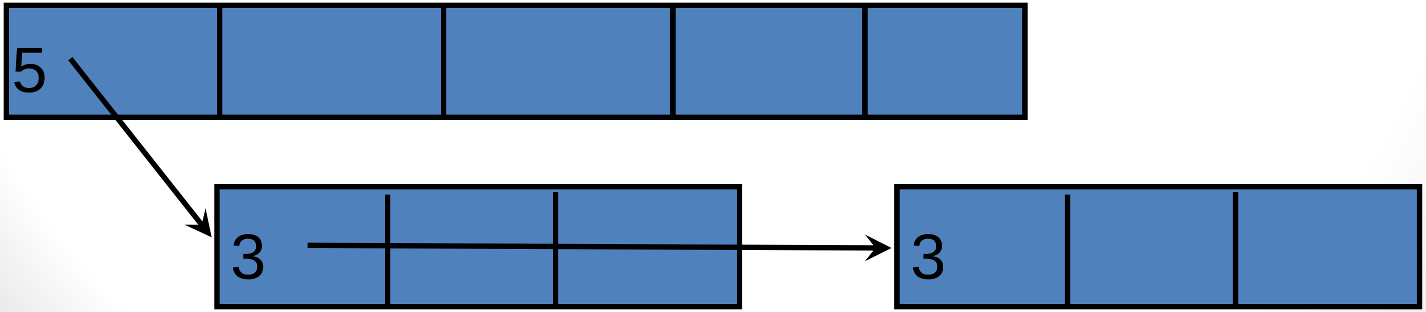


Free Space Management

- Reclaim lost space
  - ◊ Mainframes take easy way out (Track allocation, regeneration)
  - ◊ keep track of free blocks (file alloc, file delete)
- Bit Vector (like FAT link map)
  - ◊ one or more blocks (overhead)
  - ◊ each bit represents a data block on the system
  - ◊ easy to get contiguous space
  - ◊ OK for smaller disks
- Linked List
  - ◊ link all free blocks into a list
  - ◊ no overhead

Free Space Management

- Grouping
  - ◊ lists of pointers (multi level)
  - ◊ link free blocks into a tree, group by disk location
- Counting
  - ◊ blocks often allocated and released in groups
  - ◊ usually several blocks together
  - ◊ First free block contains count of the number of contiguous blocks and a pointer to next group



Efficiency and Performance

- Efficiency
  - ◊ dependent on use of system
  - ◊ 80 – 20 (% indexing)
  - ◊ fragmentation (three meanings)
- Performance
  - ◊ disk cache/page cache
    – unified virtual memory
  - ◊ unified buffer cache
  - ◊ sequential read - FIFO buffer replacement
    -free behind - free the page as soon as next page is accessed
    - read ahead - read next page before it is accessed
  - ◊ ram disk

Recovery

- Consistency checking
  - ◊ fsck on unix
  - ◊ walk file system and make sure no errors
    - blocks both in file and on free list
    - inodes with incorrect dates
- Journaled File Systems
  - ◊ transaction based
  - ◊ reliable log is used
    – usually several data blocks in the file system
    – actions are written to log and put on disk before action is taken, then removed from log
    – don't have to check entire disk, just look at files who have entries in the log

I/O Hardware

- Large Varieties of I/O devices
- Common Concepts
  - ◊ ports
  - ◊ Bus
    - daisy chain
    - shared direct access
  - ◊ Controller
- I/O instructions control the devices
- Devices have addresses
  - ◊ direct I/O instructions
  - ◊ memory mapped I/O
  - ◊ mixture

I/O Hardware

- Many buses that interact with each other
  - ◊ PCI bus
  - ◊ SCSI bus
  - ◊ IDE bus
- Communicate with controllers
  - ◊ Registers
    - status register
    - control/command register
    - input register(s)
    - output register(s)

Polling - busy waiting

- Status Register
  - ◊ busy bit - controller is busy
- Command Register
  - ◊ command ready bit

Device driver loops checking busy bit

- ◊ origin of term busy wait
- ◊ only useful if device is fast. If device is slow, then a lot of CPU cycles are wasted

Interrupts

- Interrupt current process on CPU
  - ◊ transfer to interrupt- handler
- Usually multiple interrupt vectors
  - ◊ limited number of interrupt vectors so some devices have to share the vector (handler chaining)
  - ◊ don’t have to check status register of every device
- Multiple levels of interrupts
  - ◊ priority of interrupts
  - ◊ some interrupts can be masked (disabled)
  - ◊ multiple level interrupts (used to divide handlers)
- Interrupts also used for exceptions (divide by zero) and traps are system calls.

DMA

- Direct Memory Access
  - ◊ some devices (Disk Drives, Network Controllers) transfer data in blocks
- Transferring the data in and out of the controllers one byte at a time is waste of CPU cycles
  - ◊ give the controller access to the memory bus
  - ◊ DMA controller for the bus mediates the transfer
  - ◊ Controller tells DMA controller ready to transfer data, DMA holds memory bus for controller
  - ◊ CPU is interrupted when memory transfer is done
- DMA steals cycle from CPU access
  - ◊ slows down current process
  - ◊ gain from hardware (especially for VM)

DMA

- DMA controller might use physical address
  - ◊ OS does translation of addresses when loading DMA registers
  - ◊ buffers must be contiguous in physical memory

- DMA might use logical address
  - ◊ MMU between DMA controller and memory – buffers might not be contiguous in memory

Device Driver Interface

- View to the application program
  - ◊ view to OS
- Devices are abstracted into several classes of devices
  - ◊ abstract differences in devices
    - – IDE disk vs SCSI disc
    - – many “equivalent” devices
  - ◊ common I/O commands for other parts of OS (e.g. file system) and application programs
  - ◊ extend with mechanism for issuing special commands (ioctl in UNIX)
  - ◊ different approaches between different operating systems

Types of Devices

- Character Devices
  - ◊ byte at a time
  - ◊ USB, modem, keyboard, mouse

- Block devices
  - ◊ minimal unit of transfer is a block
  - ◊ ideal for DMA use
  - ◊ disk drives, tape drives, network interfaces

Other Parameters

- Sharable/Dedicated
  - ◊ more than one process can access at a time
- Device Speed
  - ◊ very wide range of speed
- I/O direction
  - ◊ CD-ROMs are read only.
  - ◊ Other devices are write only (some printers)

Network Interface

- Low level device driver is block driven (packets)
- Application level is character stream driven (TCP/IP)
  - ◊ sockets
  - ◊ look like files
- Application level is also block driven (UDP/IP)
- select()
  - ◊ In UNIX terminals look like files
  - ◊ programs had to read from more than one terminal at a time
  - ◊ extended to networks
- All other sorts of interfaces (FIFO's, streams, queues, mailboxes, etc.)

Clocks and Timers

- Real Time
  - ◊ current time
  - ◊ usually only read at startup
  - ◊ elapsed time
  - ◊ INTERRUPTS
- OS Schedules timer interrupts
  - ◊ limited number of timers
  - ◊ Round Robin Scheduler need the timer as well
  - ◊ Used to run the OS Clock