Introduction to Operating Systems

CPSC/ECE 3220 Spring 2020

Lecture Notes
OSPP Chapter 11

(adapted from publisher's slides)

File Systems

- Abstraction on top of persistent storage
  - Magnetic disk
  - Flash memory (e.g., USB flash drive)
- Devices provide
  - Storage that (usually) survives across machine crashes
  - Block level (random) access
  - Large capacity at low cost
  - Relatively slow performance
    - Magnetic disk read takes 10-20M processor instructions

File System as Illusionist: Hide Limitations of Physical Storage

- Persistence of data stored in file system even if :
  - crash happens during an update, disk block becomes corrupted, flash memory wears out
- Controlled access to shared data
- Naming:
  - Named data instead of disk block numbers
  - Directories instead of flat storage
- Performance:
  - Data placement and data structure organization
  - Cached data

File System Abstraction

- File system
  - Persistent, named data
  - Operating system crashes (and disk errors) leave file system in a valid state
  - Journaling feature
- Access control on data
- File: named collection of data
  - Hierarchical organization (directories, subdirectories)
- Performance
  - Achieve close to the hardware limit in the average case

Files

- Metadata
  - owner, access permissions, timestamps (creation, last written), size, reference count, lock, etc.
- Data
  - May be unstructured or structured:
    - Stream of bytes (even if stored as blocks)
    - Records are collections of related fields, often with a key field used for searching and sorting

Identifying the File Type

- Metadata
- Extension identifier (.c, .o)
  - Can be used to specify usage or structure
  - OS can associate a specific application with an extension
  - Not all OSs require extensions
    - Does Linux require extensions?
- Magic number within file
  - Identify usage or structure

Three File Types in UNIX

- Regular file
  - Text (e.g., ASCII)
  - Binary (e.g., executable)
- Directory
  - Writes must be restricted to preserve structure
- Special file
  - Maps physical I/O device to the file system
  - Linux device files

File Type Philosophies

- UNIX philosophy as stated by Ritchie and Thompson (inventors):
  - “the structure of files is controlled by the programs that use them, not by the system”
  - Why do you think this is so?
-

Directories

- Directory
  - Group of named files or subdirectories
  - Mapping from file name to file metadata location
- Path
  - String that uniquely identifies file or directory
    - Ex: /web/home/joe/public_html/3220.html
- Links
  - Hard link: link from name to metadata location
  - Soft link: link from name to alternate name (which links to metadata location)
- Mount points
  - Mapping from name in one file system to root of another

Path Names

- Absolute path
  - Fully qualified
  - Starts at root (“/” in Unix systems, “C:” in Windows)
- Relative path
  - Partially qualified, ex: public_html/3220.html
  - Unqualified, ex: 3220.html
  - Starts at current working directory
  - . => this directory, .. => parent directory

UNIX File System API

- create, link, unlink, createdir, rmdir
  - Create file, link to file, remove link
  - Create directory, remove directory
- open, close, read, write, seek
  - Open/close a file for reading/writing
  - Seek resets current position
- fsync
  - File modifications can be cached
  - fsync forces modifications to disk (like a memory barrier)

File System Workload

- How are files used?
  - Most files are read/written sequentially
  - Some files are read/written randomly
    - Ex: database files, swap files
  - Some files have a pre-defined size at creation
  - Some files start small and grow over time
    - Ex: program stdout, system logs

Connection-Oriented Interface

- Explicit open and close operations for files
  - OS creates an internal data structure on open
- Read and write ops use descriptor (a.k.a. handle or stream) to identify the internal data structure
  - No need to reparse file name
- Per-open data structure contains:
  - Access permission under which file was opened
  - Location of file (e.g., inode number)
  - Pointer to current byte or record for sequential access

UNIX Data Structures for Open Files

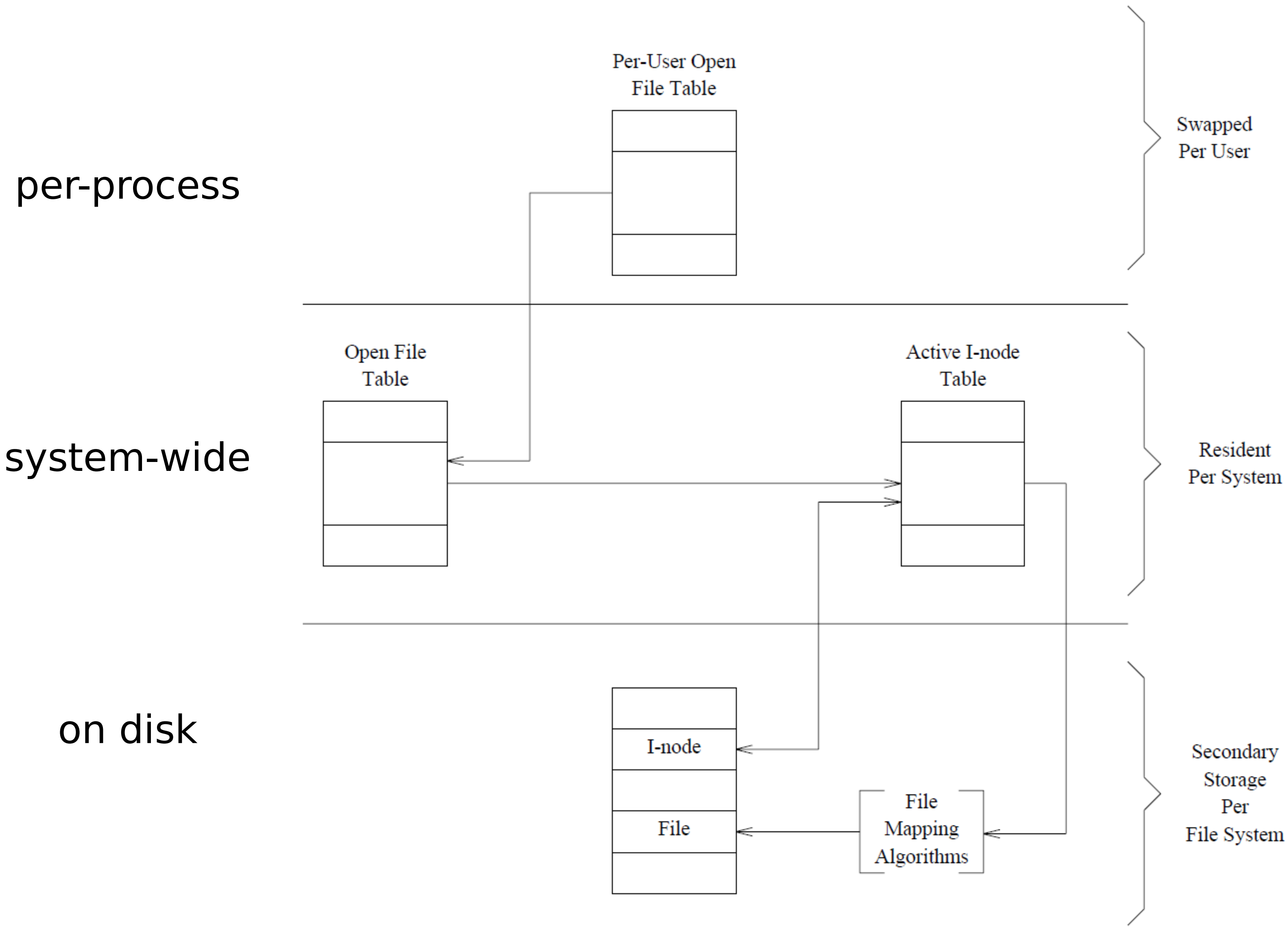

Diagram from K. Thompson, “UNIX Implementation,” Bell System Tech. Jrnl., 1978

UNIX Data Structures for Open Files (2)

per-process

system-wide

copy inode info
into kernel memory

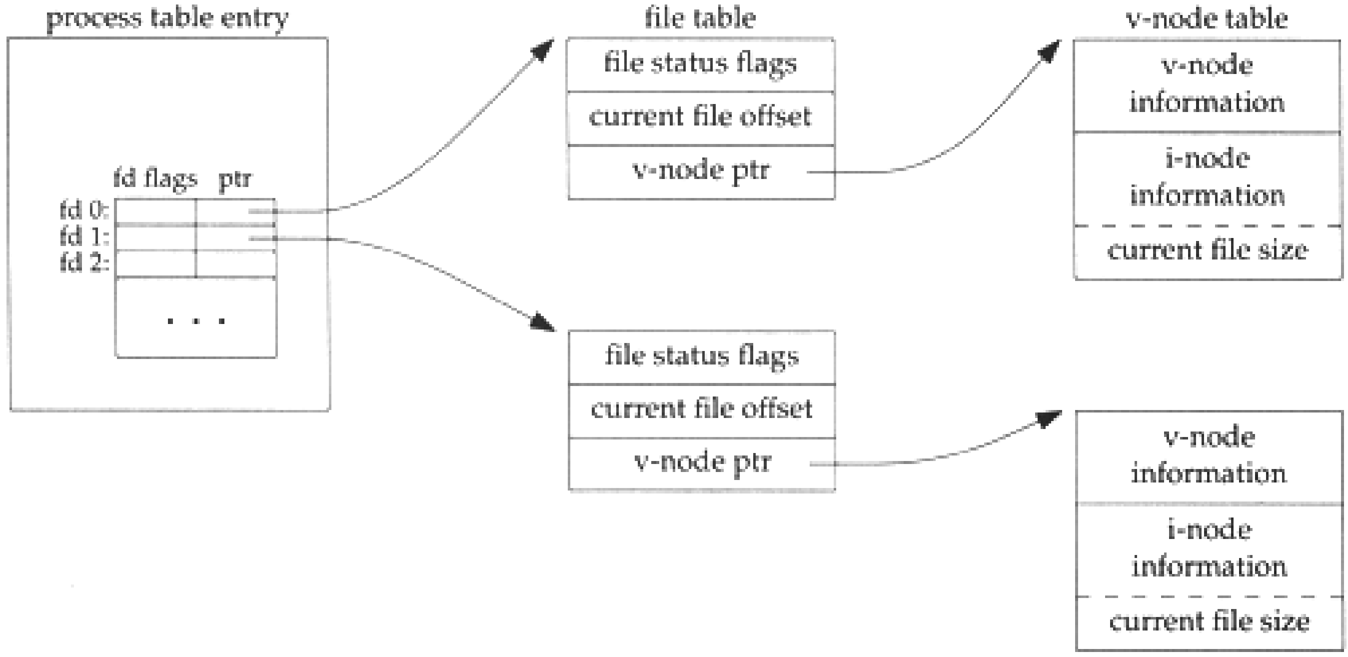

Diagram from W. Richard Stevens, Chapter 3, “UNIX File I/O,” Advanced Programming in the UNIX® Environment, 2003

UNIX Data Structures for Open Files (3)

Two processes open the same file – each has its own current file pointer ("offset")

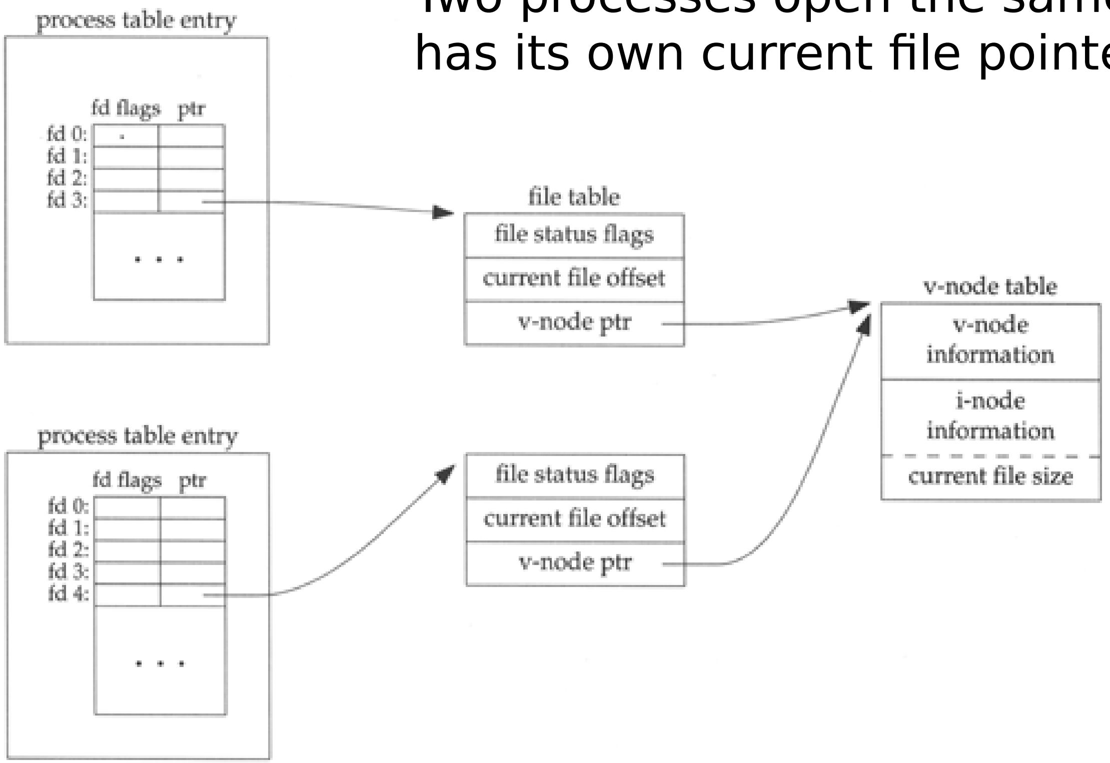

Diagram from W. Richard Stevens, Chapter 3, "UNIX File I/O," Advanced Programming in the UNIX® Environment, 2003

File System Workload

- File sizes
  - Are most files small or large?
    - SMALL
  - Which accounts for more total storage: small or large files?
    - LARGE

File System Workload

- File access
  - Are most accesses to small or large files?
    - SMALL
  - Which accounts for more total I/O bytes: small or large files?
    - LARGE