

The plan ...

-
- A bit more about file descriptors
- Project 2 questions
- Exam preparation
- Threads

File Descriptors

CS 3113

File Descriptors

- Integer that references a table inside of the kernel memory space
 - File Descriptor Table
- There is one table for each process
- Program provides the FD any time we make a system call that is about accessing an open stream (could be a file, pipe, etc)

Representing Files Inside the Kernel

- Open File Table:
 - Store information about how the files are being accessed
 - File offset, status information
- Inode Table (for files):
 - Store disk-level information about the file
 - Location of the file on the disk, permissions, type of file, ...

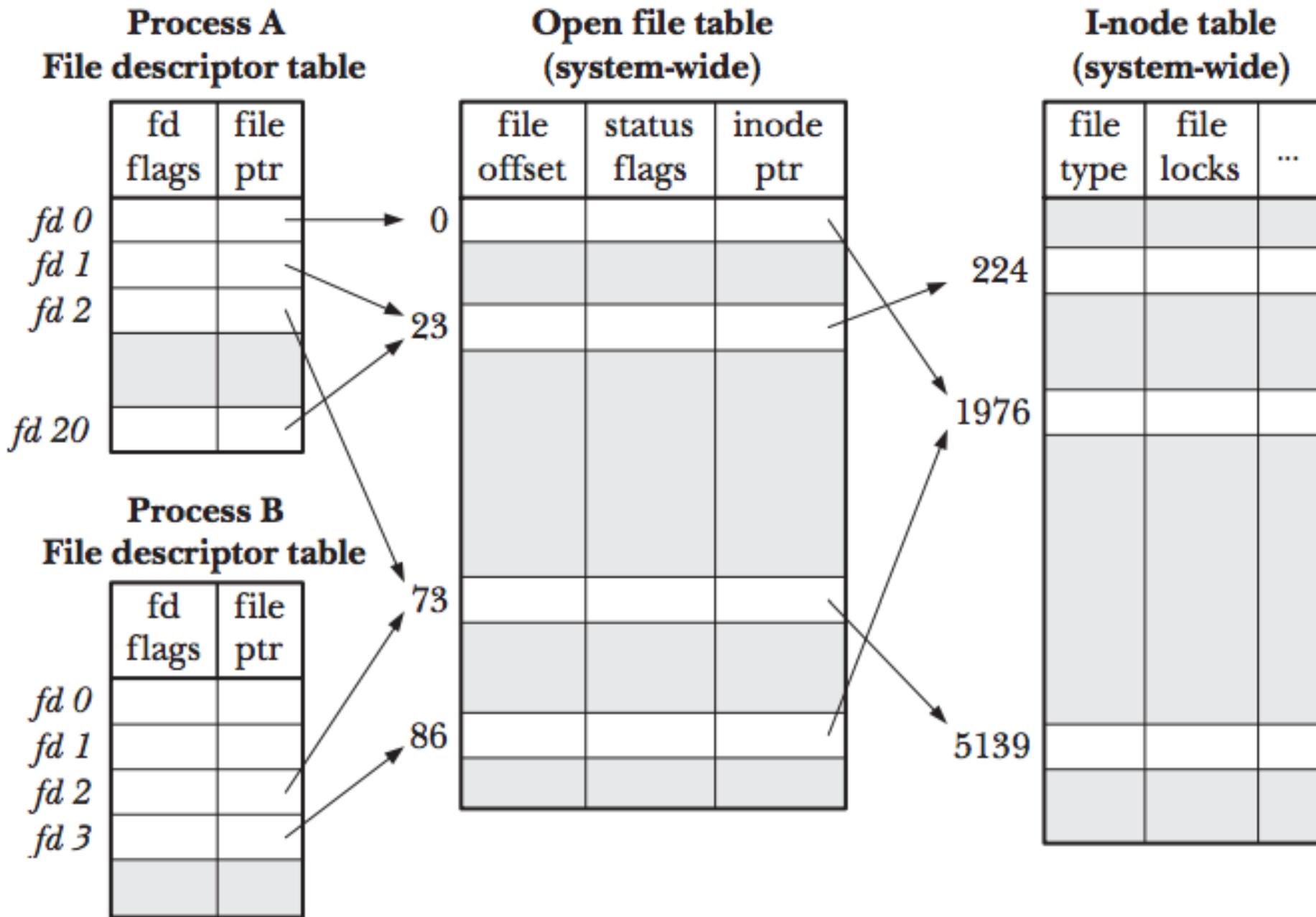


Figure 5-2: Relationship between file descriptors, open file descriptions, and i-nodes

Representing Files Inside the Kernel

Key ideas:

- Multiple processes can share an entry in the open file table
 - Sharing happens through a fork()
 - Shared file offset
- Multiple processes can open the same file
 - Each has its own entry in the open file table
 - Independent file offsets

Duplicating File Descriptors

- In various cases, we have a need to copy the contents of one File Descriptor Table entry to another entry
- This is particularly useful when we want to replace an existing file descriptor with another

Duplicating File Descriptors

```
// Open a named pipe
fd = open("my_pipe", O_WRONLY);
// Close the file descriptor that correspond to stdout
close(1);
// Copy the contents from the recently opened pipe to fd 1
dup2(fd, 1);

// Now, anything written to file descriptor 1 will be
// written to the named pipe!
```

STDOUT

- The globally declared symbol `stdout` is a pointer to a `FILE` structure
- What is in the `FILE` structure?

STDOUT

What is in the FILE structure?

- A file descriptor associated with the FILE
- A buffer that stores (for the case of STDOUT) outgoing data until it is needed or flushed. Flushes happen when
 - Certain characters are written (e.g., '\n')
 - `fflush(stdout)` is called
 - The buffer fills up

FILE vs file descriptor (STDOUT)

- stdout maps to file descriptor 1
- So, anything written to FILE stdout, is then written to file descriptor 1
- This includes: printf!
 - `printf("Hello, World!\n");`
 - `fprintf(stdout, "Hello, World!\n")`
- By default (in our set up), STDOUT is routed to the starting terminal

Duplicating File Descriptors

```
// Open a named pipe
fd = open("my_pipe", O_WRONLY);
// Close the file descriptor that correspond to stdout
close(1);
// Copy the contents from the recently opened pipe to fd 1
dup2(fd, 1);

// Now, magic can happen
printf("Hello, World!\n")           // Routed to the named pipe
```

Duplicating File Descriptors

- Once the `dup2()` happens, your program does not even realize that it is now talking to another entity (a named pipe, in this case)
- Particularly useful when you want to generate output to the terminal in some cases, but output to files or other processes in other cases

dup2 demo