Outline

What's the problem?

- Suppose we want to write data at end of a file. . .
- What's wrong with this approach?

```
lseek(fd, 0, SEEK_END);
write(fd, buf, len);
```

- **Race condition**: Another program might append data to file between *lseek()* and *write()*
  - ⇒ we will overwrite that data
- Need to ensure two steps are executed **atomically**
  - i.e., no other process / thread can operate on file between two steps
- Solution: *open()* `O_APPEND` flag
  - ⇒ *write()* atomically seeks to EOF and writes

man7.org

Atomicity

Various parts of system call API provide an atomicity guarantee that is necessary to achieve correct results

- Another example: ensuring we are creator of a file
- **Wrong way**:
  - open once *without* `O_CREAT`;
  - if that fails (with `ENOENT`), open a second time with `O_CREAT`
- **Right way**: `open(..., O_CREAT | O_EXCL, ...)`
  - **Atomically** check that file does not exist and if so, create it

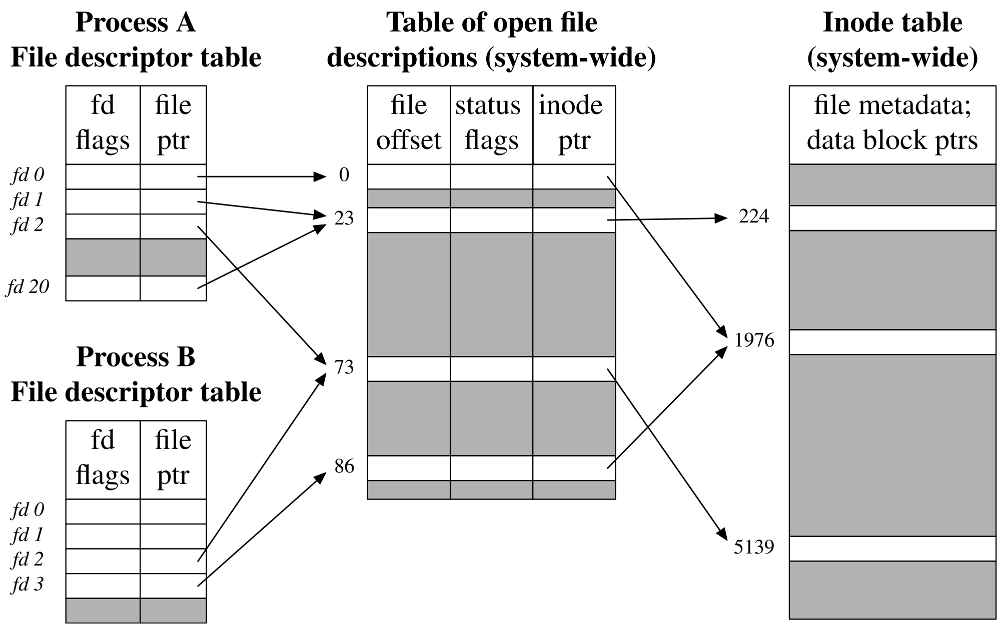
man7.org

[TLPI §5.1]

Outline

Relationship between file descriptors and open files

- Multiple file descriptors can refer to same open file
- 3 kernel data structures describe relationship:



File descriptor table

Per-process table with one entry for each FD opened by process:

- Flags controlling operation of FD (close-on-exec flag)
- Reference to open file description
- *struct fdtable* in `include/linux/fdtable.h`

Table of open file descriptions (open file table)

System-wide table, one entry for each open file on system:

- File offset
- File access mode (R / W / R-W, from *open()*)
- File status flags (from *open()*)
- Reference to inode object for file
- *struct file* in `include/linux/fs.h`

Following terms are commonly treated as synonyms:

- **open file description (OFD)** (POSIX)
- **open file table entry** or **open file handle**
    - ⚠ Ambiguous terms; POSIX terminology is preferable

man7.org

(In-memory) inode table

System-wide table drawn from file inode information in filesystem:

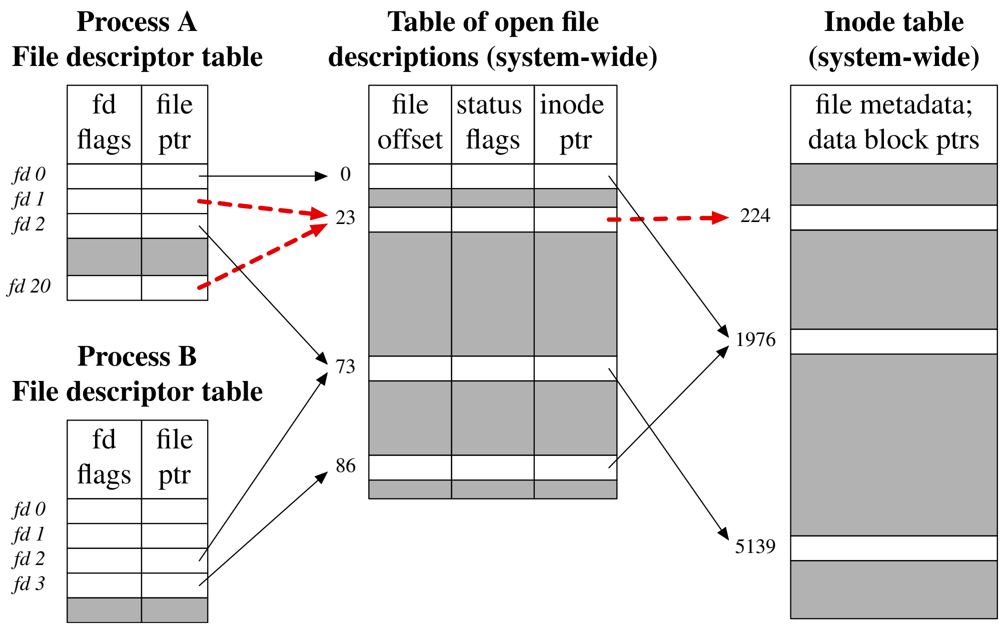
- File type (regular file, FIFO, socket, . . .)
- File permissions
- Other file properties (size, timestamps, . . .)
- *struct inode* in `include/linux/fs.h`

man7.org

Duplicated file descriptors (intraprocess)

A process may have multiple FDs referring to same OFD

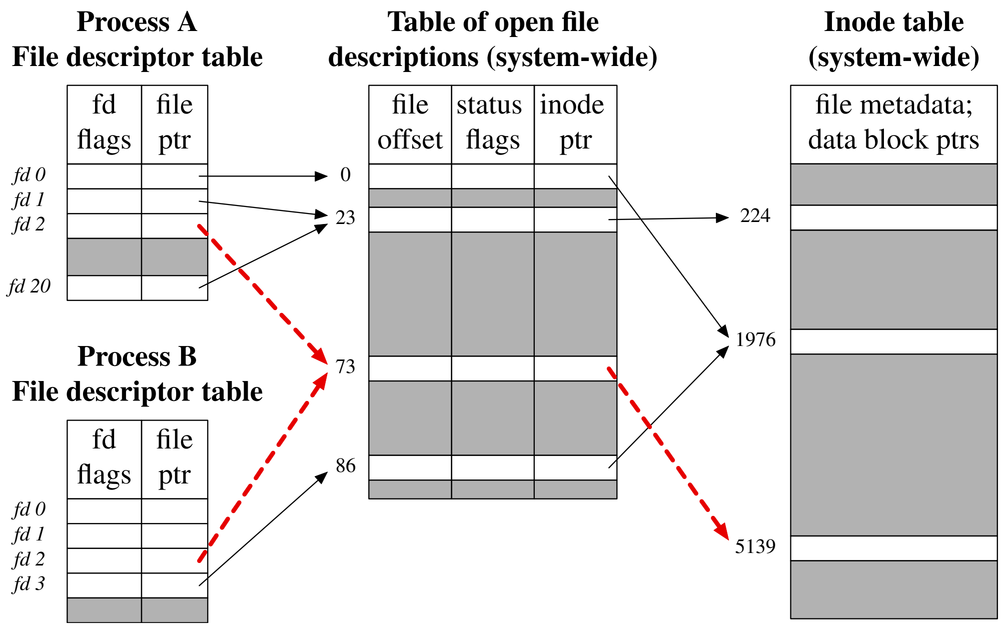
- Achieved using *dup()* or *dup2()*



Duplicated file descriptors (between processes)

Two processes may have FDs referring to same OFD

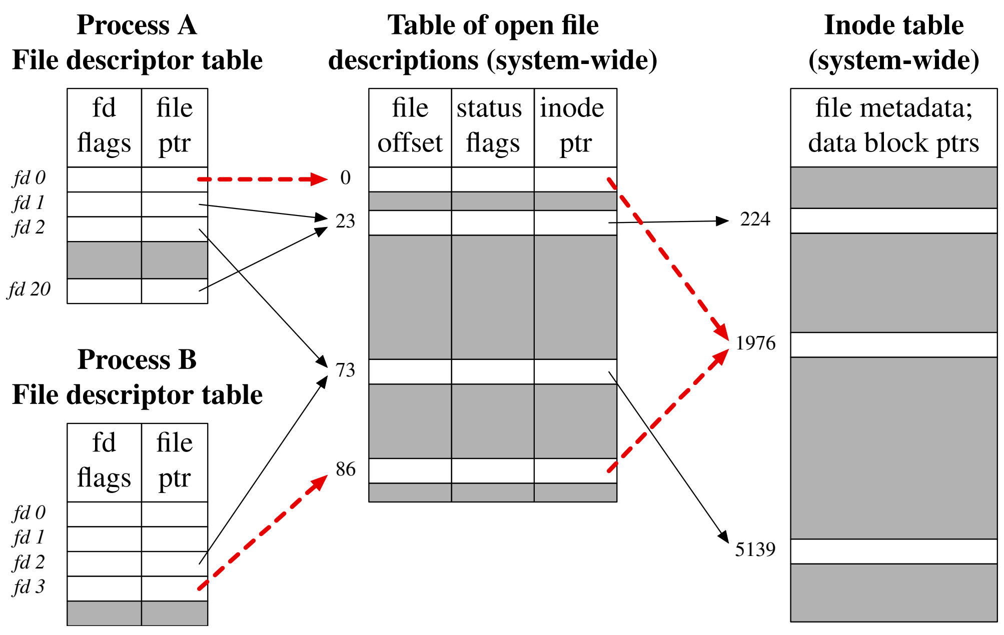
- Can occur as a result of *fork()*



Distinct open file table entries referring to same file

Two processes may have FDs referring to distinct OFDs that refer to same inode

- Two processes independently *open()*ed same file



Why does this matter?

- Two different FDs referring to same OFD share file offset
  - (File offset == location for next *read()*/*write()*)
  - Changes (*read()*, *write()*, *lseek()*) via one FD visible via other FD
  - Applies to both intraprocess & interprocess sharing of OFD
- Similar scope rules for status flags (`O_APPEND`, `O_SYNC`, ...)
  - Changes via one FD are visible via other FD
    - (`fcntl(F_SETFL)` and `fcntl(F_GETFL)`)
- Conversely, changes to FD flags (held in FD table) are private to each process and FD
- *kcmp(2)* `KCMP_FILE` operation can be used to test if two FDs refer to same OFD
  - Linux-specific

[TLPI §5.4]

Outline

A problem

```
./myprog > output.log 2>&1
```

- What does the shell syntax, `2>&1`, do?
- How does the shell do it?
- Open file twice, once on FD 1, and once on FD 2?
  - FDs would have separate OFDs with distinct file offsets ⇒ standard output and error would overwrite
  - File may not even be *open()*-able:
    - e.g., `./myprog 2>&1 | less`
- Need a way to create duplicate FD that refers to same OFD

man7.org

[TLPI §5.5]

Duplicating file descriptors

```
#include <unistd.h>
int dup(int origfd);
```

- Arguments:
  - *origfd*: an existing file descriptor
- Returns new file descriptor that refers to same OFD
- **New file descriptor is guaranteed to be lowest available**

Duplicating file descriptors

- FDs 0, 1, and 2 are normally always open, so shell can achieve `2>&1` redirection by:

```
close(STDERR_FILENO);         /* Frees FD 2 */
newfd = dup(STDOUT_FILENO);   /* Reuses FD 2 */
```

- But what if FD 0 had been closed beforehand?
  - *dup()* would reuse FD 0...
    - We need a better API

Duplicating file descriptors

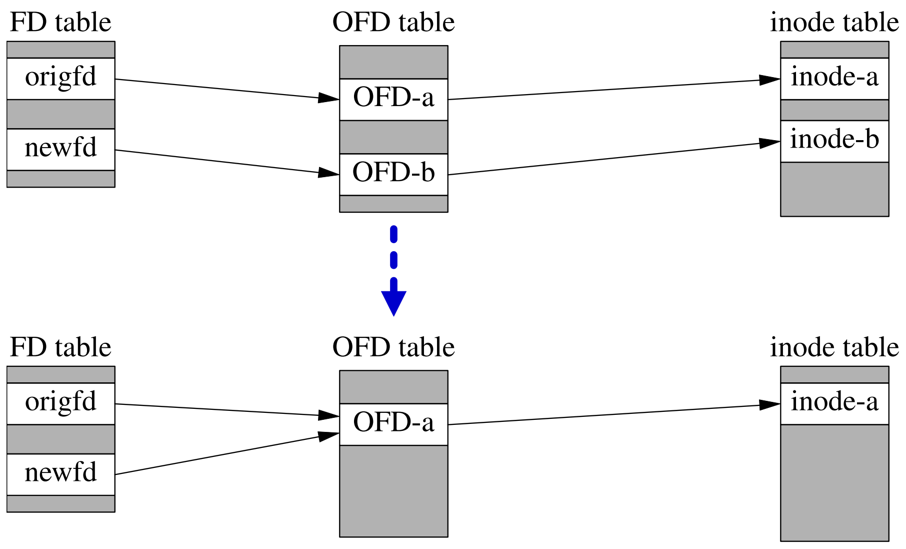
```
#include <unistd.h>
int dup2(int origfd, int newfd);
```

- Like *dup()*, but uses *newfd* for the duplicate FD
  - **Silently** closes *newfd* if it was open
  - Close + reuse of *newfd* is done as an atomic step
    - Important: otherwise, *newfd* might be re-used in between
  - Does nothing if *newfd* == *origfd*
  - Returns new file descriptor (i.e., *newfd*) on success
- `dup2(STDOUT_FILENO, STDERR_FILENO);`
- See *dup2(2)* manual page for more details

man7.org

[TLPI §5.5]

Understanding *dup2(origfd, newfd)*



- If *newfd* was an open FD, OFD-b will be released if *newfd* was the last FD that referred to it
- After *dup2()*, *origfd* and *newfd* share file offset and file status flags

man7.org

Outline

File status flags

- Control semantics of I/O on a file
  - (`O_APPEND`, `O_NONBLOCK`, `O_SYNC`, . . .)
- Associated with open file description
- Set when file is opened
- Can be retrieved and modified using *fcntl()*

man7.org

[TLPI §5.3]

fcntl(): file control operations

```
#include <fcntl.h>
int fcntl(int fd, int cmd /* , arg */ );
```

Performs control operations on an open file

- Arguments:
  - *fd*: file descriptor
  - *cmd*: the desired operation
  - *arg*: optional, type depends on *cmd*
- Return on success depends on *cmd*; –1 returned on error
- Many types of operation
  - file locking, signal-driven I/O, file descriptor flags . . .

man7.org

Retrieving file status flags and access mode

- Retrieving flags (both access mode and status flags)

```
int flags = fcntl(fd, F_GETFL);
```

- Check access mode

```
int amode = flags & O_ACCMODE;
if (amode == O_RDONLY || amode == O_RDWR)
    printf("File is readable\n");
```

- ⚠ 'read' and 'write' are not separate bits!

```
if (flags & O_RDONLY)       /* Wrong!! */
    printf("File is readable\n");
```

  - Access mode is a 2-bit field that is an enumeration:
    - 00 == `O_RDONLY`; 01 == `O_WRONLY`; 10 == `O_RDWR`
    - (`O_ACCMODE` == 11_2)
- Access mode can't be changed after file is opened

man7.org

Retrieving and modifying file status flags

- Retrieving file status flags

```
int flags = fcntl(fd, F_GETFL);
if (flags & O_NONBLOCK)
    printf("Nonblocking I/O is in effect\n");
```

- Setting a file status flag

```
int flags = fcntl(fd, F_GETFL);      /* Retrieve flags */
flags |= O_APPEND;                   /* Set "append" bit */
fcntl(fd, F_SETFL, flags);           /* Modify flags */
```

  - ⚠ Not thread-safe...
    - (But in many cases, flags can be set when FD is created, e.g., by *open()*)

- Clearing a file status flag

```
int flags = fcntl(fd, F_GETFL);      /* Retrieve flags */
flags &= ~O_APPEND;                  /* Clear "append" bit */
fcntl(fd, F_SETFL, flags);           /* Modify flags */
```

man7.org

Outline

API summary

```
// Adjust file offset:
off_t new_offset = lseek(int fd, off_t offset, int whence);
            // whence is SEEK_SET / SEEK_CUR / SEEK_END

// Fetch / modify file status flags:
int flags = fcntl(fd, F_GETFL);     // Fetch status flags and
                                    // access mode
int fcntl(fd, F_SETFL, flags);      // Update status flags

// File descriptor duplication:
int newfd dup(int origfd);          // Make lowest unused FD a
                                    // duplicate of 'origfd'
int dup2(int origfd, int newfd);    // Make 'newfd' point to same
                                    // object as 'origfd'
```

man7.org

Outline

Exercise

1. Show that duplicate file descriptors share file offset and file status flags by writing a program ([template: `fileio/ex.fd_sharing.c`]) that:
   - Implements the function *printFileDescriptionInfo()*, which, given a file descriptor as an argument, prints the file descriptor number as well as the file offset and the state of the `O_APPEND` file status flag associated with that file descriptor. For readability, *all three values should be printed on one line*.
   - Opens an existing file (supplied as *argv[1]*) and duplicates (*dup()*) the resulting file descriptor, to create a second file descriptor.
   - Uses the *printFileDescriptionInfo()* function to display the file offset and the state of the `O_APPEND` file status flag via the first file descriptor.
     - Initially the file offset will be zero, and the `O_APPEND` flag will not be set
   - Changes the file offset (*lseek()*, slide 5-5) and enables (turns on) the `O_APPEND` file status flag (*fcntl()*, slide 5-39) via the second file descriptor.
   - Uses the *printFileDescriptionInfo()* function to display the file offset and the state of the `O_APPEND` file status flag via the first file descriptor.

   Hints:
   - Remember to update the `Makefile`!
   - `while inotifywait -q . ; do echo -e '\n\n'; make; done`

man7.org

Exercise

2. ⏲ ⏲ ⏲ The program `fileio/fd_overwrite_test.c` can be used to demonstrate that if a program opens the same file twice, the two file descriptors do not share a file offset, and thus writes via one file descriptor will overwrite writes via the other file descriptor. By contrast, if a program opens the file and duplicates the resulting file descriptor, then the two file descriptors do share a file offset, and writes via one file descriptor will not overwrite writes via the other file descriptor. The program is used with a command-line as follows:

```
$ ./fd_overwrite [-d] <file> <string>...
```

By default, the program *open()*s the specified file twice, but if the *–d* option is specified, then it *open()*s the file once and duplicates the resulting file descriptor. The remaining arguments are strings that are alternately written to the two file descriptors (thus, the first string is written to FD 1, the second to FD 2, the third to FD 1, and so on).

Run the program with the following two command lines, and explain the output that appears in the two files:

```
$ ./fd_overwrite_test    x a A b B c C
$ ./fd_overwrite_test -d y a A b B c C
```

man7.org