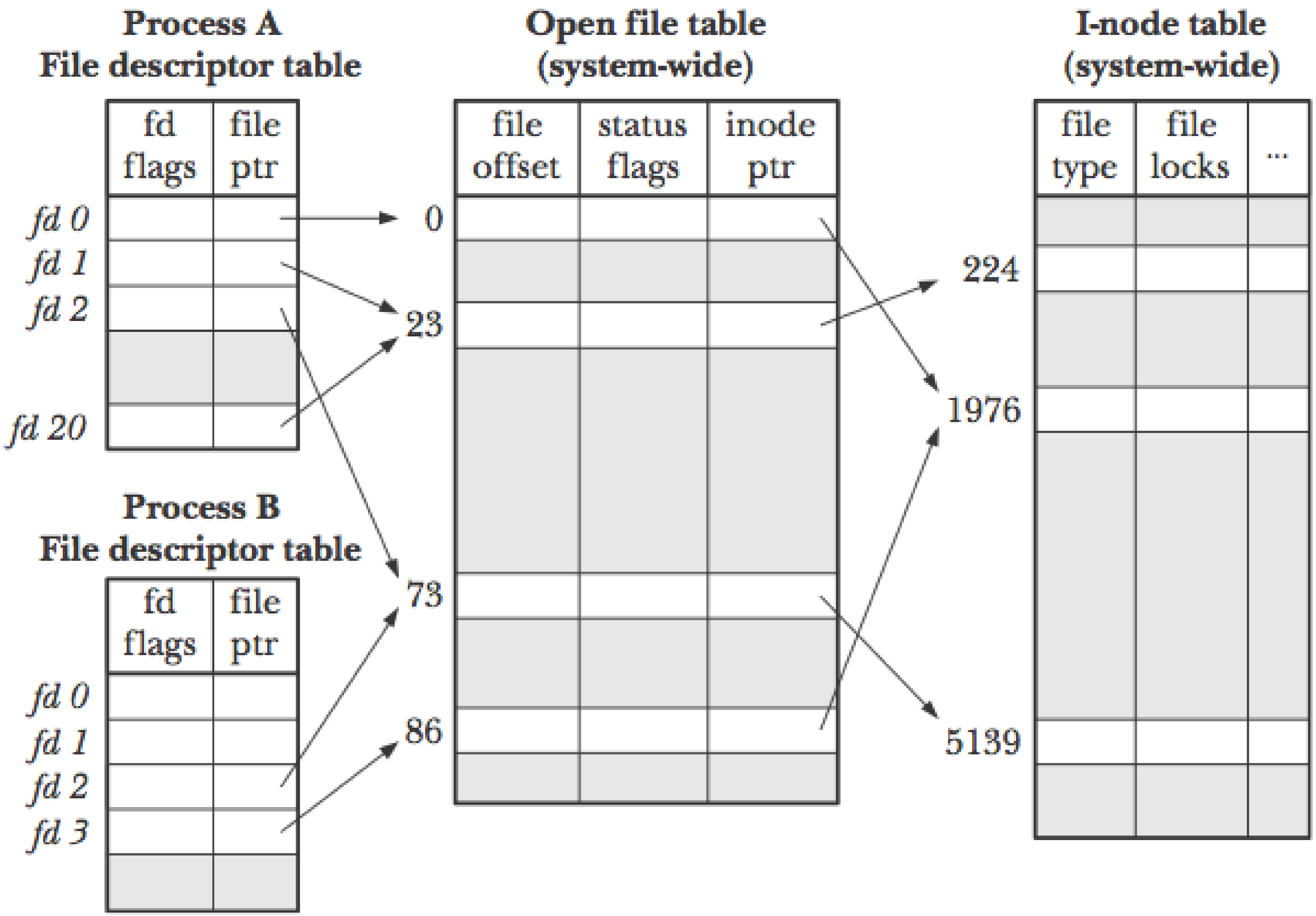
I/O Redirection

Per Process Data

- Global data
- Stack
- Code
- Environmental Variables
- ...
- **File descriptor table**

File Descriptor Table



File Descriptor (fd)

- Integer index into the file descriptor table
- Calls like open() return the next one available
- This is different from fopen which returns FILE*
- open(), close(), read(), write() work on fd's
  - System calls
  - STDIN_FILENO, STDOUT_FILENO
- fopen(), fclose(), fscanf(), fprintf() work on FILE*
  - C library calls
  - stdin, stdout
  - Can get fd by calling fileno(FILE*)

Standard File Descriptors

- fd=0 – stdin
  - A standard place to read input from
- fd=1 – stdout
  - A standard place to write output to
- fd=2 – stderr
  - A standard place to write errors to

close(stdin)???

- If we close stdin, the process won't be able to read data from a centralized place
  - Though we can still access files, sockets, etc
- However, by calling open on a different file
  - We can replace this stdin slot with that file
- This redirection of input from stdin to a file is the basics of I/O redirection
  - *cmd* < *file*

Basic I/O Redirection Types

- Input redirection
  - Read from file
  - *cmd* < *file*
  - cmd reads from file instead of stdin
- Output redirection
  - Overwrite file
  - *cmd* > *file*
  - cmd writes to file instead of stdout

Input Redirection Example

```
int fd = open(path);

if (fork() == 0) {
    //Child
    close(STDIN_FILENO);
    dup(fd);
    close(fd);
    //Execute process
}
else {
    //Parent
    close(fd);
}
```

dup

- #include <unistd.h>
  - dup(int oldfd)
  - Returns the new fd; -1 on error
- Duplicates a file descriptor
  - dup simply assigns to lowest slot

Reasons to dup

- Keep from forking if open doesn't work
  - Fork is an expensive operation
- Redirect to an existing fd
  - *ls > file 2>&1*
- To open the same file but with different attributes
  - Read and write to the same socket

Why Bother with I/O Redirection?

- Program can program to standard interface while caller can change where the I/O goes
  - Testing (read from test file)
  - Debugging (splitting output and errors)
  - Logging (save output)
  - Network communication (read/write to/from socket files)
- It's part of your project...
- It's the foundation of how pipes work