

Unix File I/O

W4118 Operating Systems I

columbia-os.github.io

UNIX: Everything is a file

Unix, and its derivatives, handle input/output from a different resources with the same file-like interface:

- Files
- Peripheral devices
- Inter-process communication (IPC)
- Networking
- ...

Advantages? Disadvantages?

UNIX: Everything is a file

Unix, and its derivatives, handle input/output from a different resources with the same file-like interface:

- Files
- Peripheral devices
- Inter-process communication (IPC)
- Networking
- ...

- + Portability and code-reuse
- + Read, write, and execute permissions apply uniformly

Advantages? Disadvantages?

- Lack of specialization

open()

```
// Need to specify mode if file is being created...  
int open(const char *path, int oflag, mode_t mode);  
// ...otherwise, mode argument is omitted.  
int open(const char *path, int oflag);
```

open()

```
// Need to specify mode if file is being created...  
int open(const char *path, int oflag, mode_t mode);  
// ...otherwise, mode argument is omitted.  
int open(const char *path, int oflag);
```

Result:

- Creates an entry in the process's **File Descriptor Table**(not exactly):
 - Offset in the file
 - Open options
 - Metadata
- Returns the index of the entry, a.k.a. **file descriptor**

Example: `simple-file.c`

close()

```
int close(int fildes);
```

Deletes the file descriptor table entry at index `fildes`

What happens if the file is not closed before a process finishes?

lseek(): Where are we in the file?

```
off_t lseek(int fildes, off_t offset, int whence);
```

- If whence is `SEEK_SET`, the file offset shall be set to offset bytes
- If whence is `SEEK_CUR`, the file offset shall be set to its current location plus offset
- If whence is `SEEK_END`, the file offset shall be set to the size of the file plus offset – why??

Does `lseek` do any file I/O?

read()/write()

```
ssize_t read(int fildes, void *buf, size_t nbyte);
```

```
ssize_t write(int fildes, const void *buf, size_t nbyte);
```

For sockets, equivalent to:

```
ssize_t recv(int socket, void *buffer, size_t length, int flags)
```

```
ssize_t send(int socket, const void *buffer, size_t length, int flags)
```

What happens if multiple processes write to the same file?

C standard I/O library

```
FILE *fopen(const char *pathname, const char *mode); // open()
```

```
int fclose(FILE *stream); // close()
```

```
int fseek(FILE *stream, long offset, int whence); // lseek()
```

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);  
// read()
```

```
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE  
*stream); // write()
```

FILE *stream **replaces** **int** fd

C I/O library buffering

Goal: reduce number of `read()/write()` syscalls while performing stream operations

C I/O library buffering

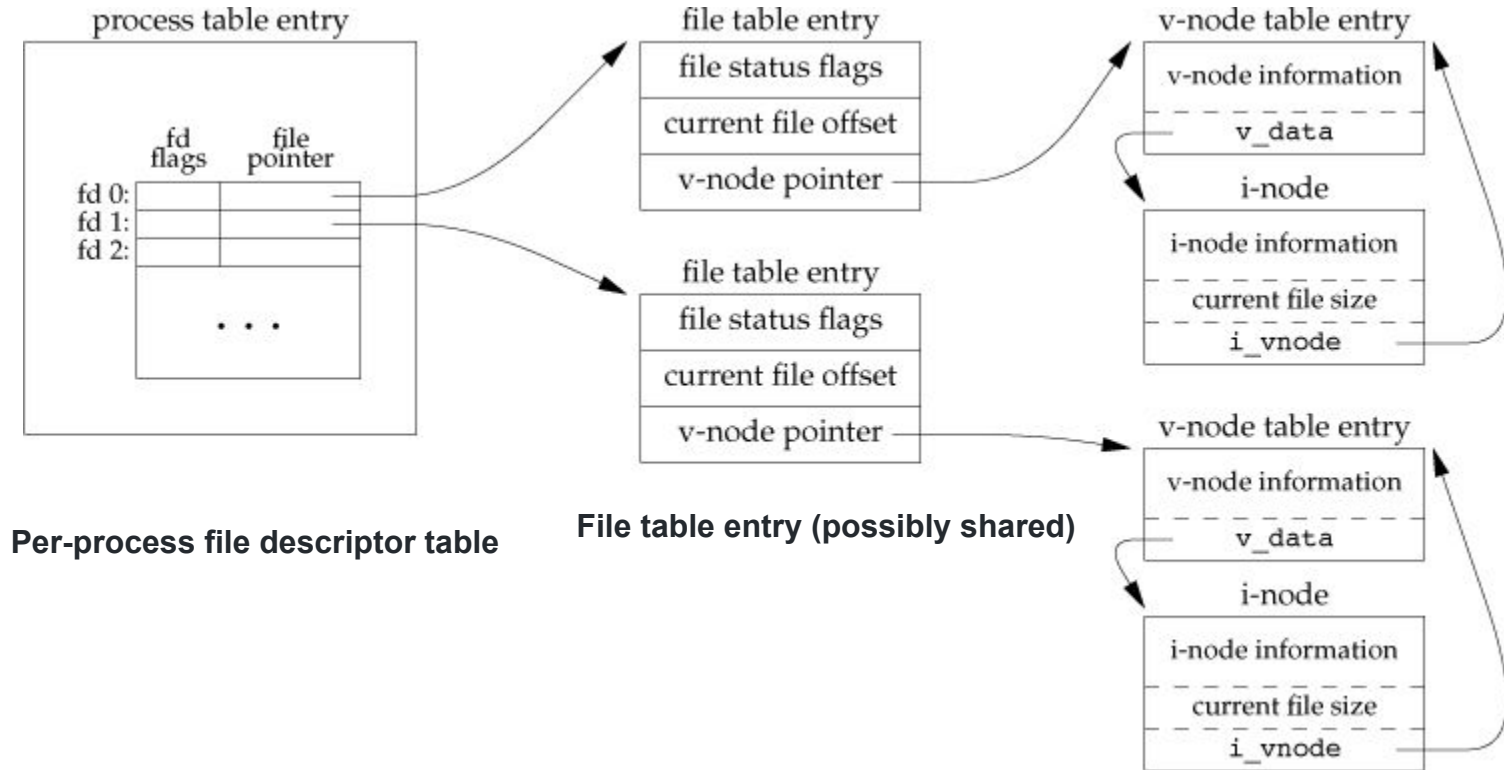
Goal: reduce number of `read()/write()` syscalls while performing stream operations

- **Solution:** `fread()/fwrite()` call `read()/write()` once in a while, then use underlying buffer.

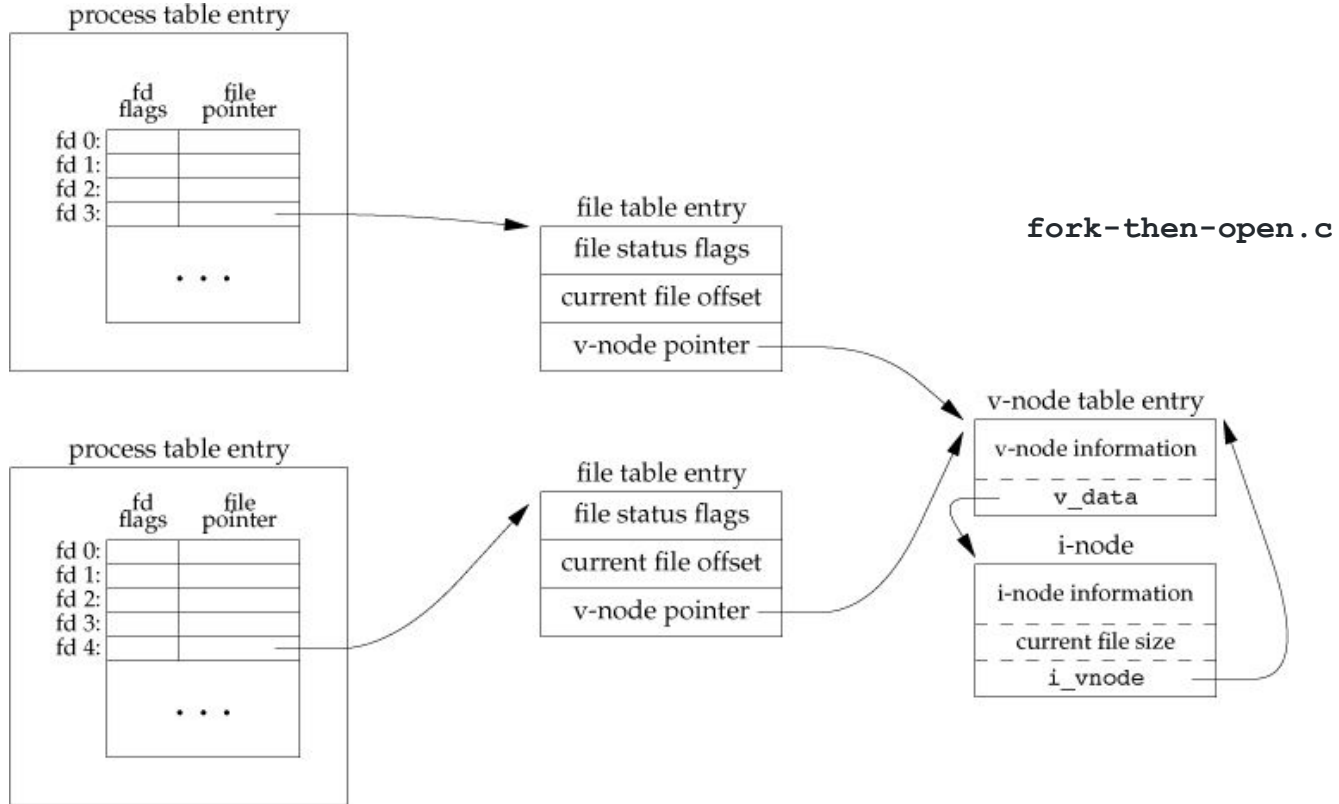
Trace syscall invocations via `strace`:

- `io.c` VS `std-io.c` note how many times `read()` is called in each program

Files in the kernel

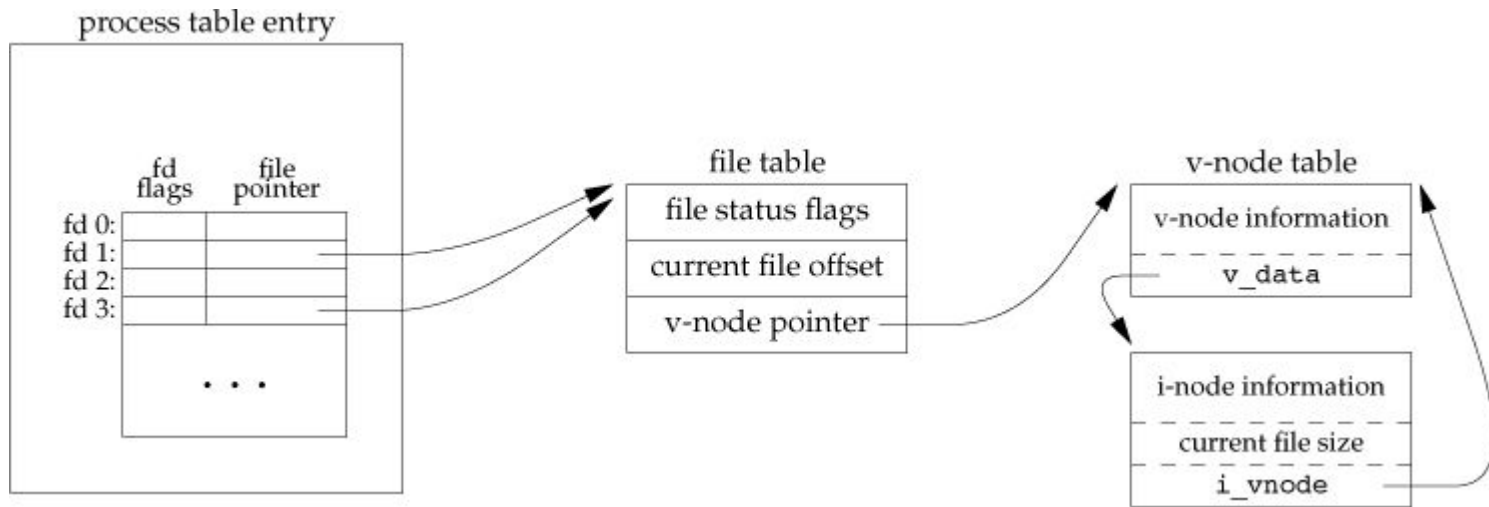


Files in the kernel – Independent Processes



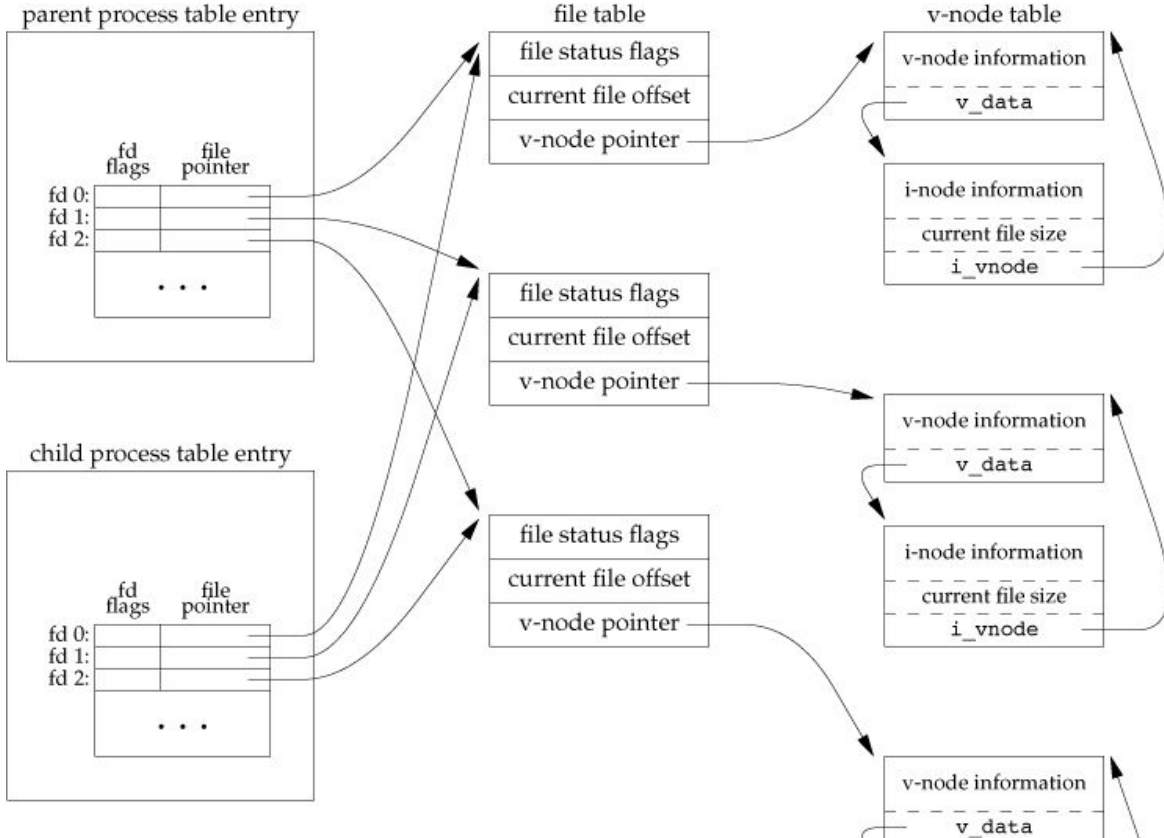
dup()

```
int dup(int oldfd);
```



New file descriptor table slot points to the same file table entry

Parent and child after fork



`open-then-fork.c`