Lecture 19: Files and Directories

Operating Systems

Content taken from: https://pages.cs.wisc.edu/~remzi/OSTEP/

https://www.cse.iitb.ac.in/~mythili/os/

File and Directory as Abstractions

- Two key abstractions for virtualizing persistent storage
- **File** is a linear array of bytes, stored persistently
  - File name (human readable)
  - OS-level identifier (**inode number**)
- **Directory** contains other subdirectories and files, along with their inode numbers
  - Stored like a file, whose contents are filename-to-inode mappings

Directory Tree

- Root directory
- **Separator** for naming subsequent sub-directories
- **Absolute pathname**
- **File Type**

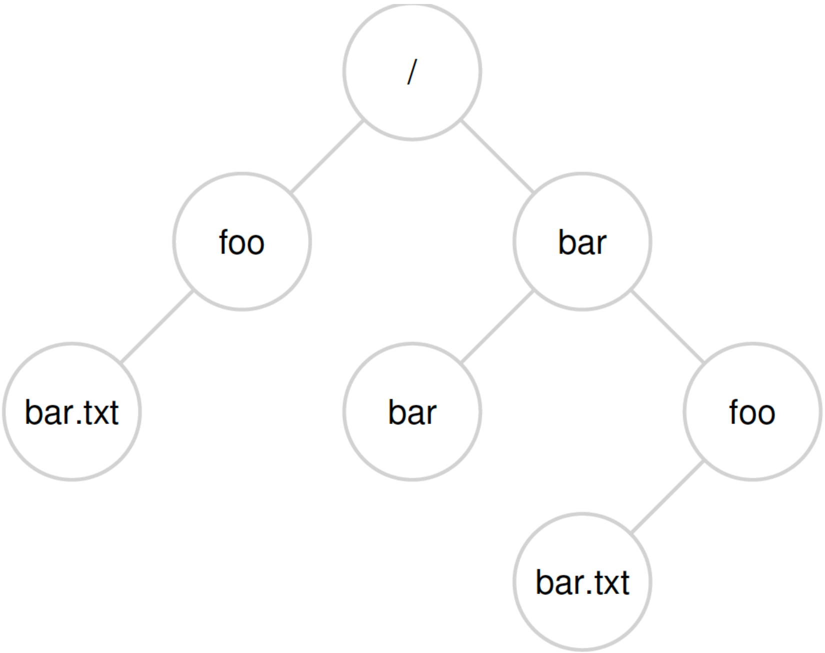


Figure 39.1: **An Example Directory Tree**

Creating a File

- **open()** system call

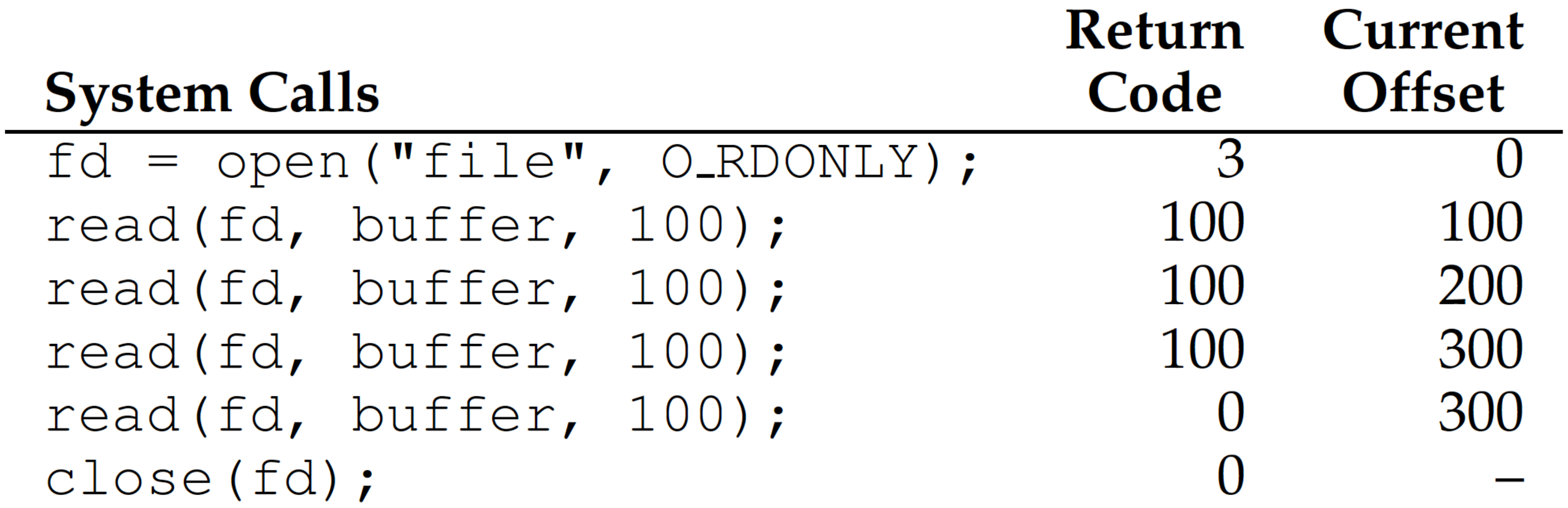
```
int fd = open("foo", O_CREAT|O_WRONLY|O_TRUNC,
              S_IRUSR|S_IWUSR);
```

- Returns a **file descriptor**
  - File descriptor is just an integer, private per process
  - Used to access a file and perform operations on it
  - Think of file descriptor as a pointer to an object of type file
  - File descriptor for
    - STD_IN is 0
    - STD_OUT is 1
    - STD_ERR is 2
- What happens when we create or open a file?
  - An entry is created in the system-wide **Open File Table (OFT).**
  - Each process keeps an array of file descriptors each of which refers to an entry in the OFT.

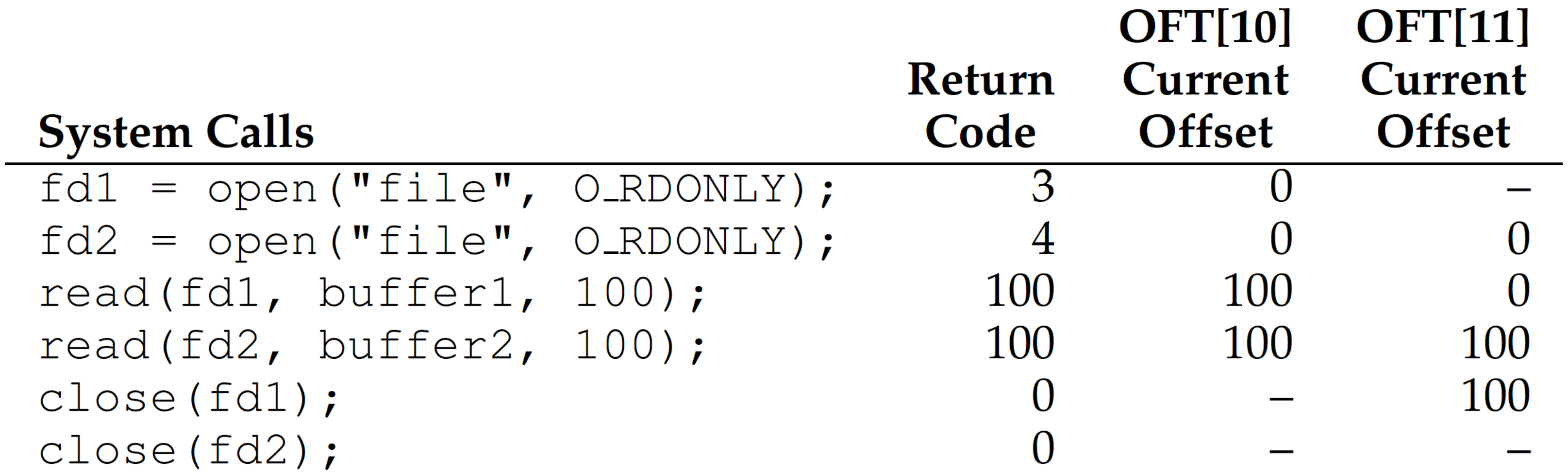
Reading and Writing Files

- Read(file_descriptor, buffer_to_read_into, buffer_size)
  - Returns the number of bytes successfully read
- Write(file_descriptor, buffer_to_write_from, bytes_to_write)
  - Returns the number of bytes successfully written
- Reading and writing happens **sequentially** by default
  - Successive read/write calls fetch from current offset

Examples



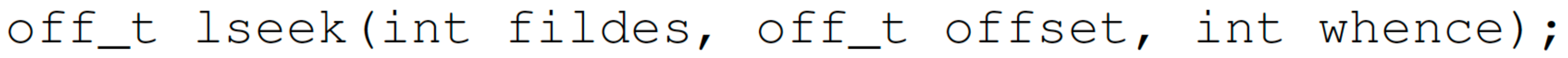
System Calls	Return Code	Current Offset
fd = open("file", O_RDONLY);	3	0
read(fd, buffer, 100);	100	100
read(fd, buffer, 100);	100	200
read(fd, buffer, 100);	100	300
read(fd, buffer, 100);	0	300
close(fd);	0	–



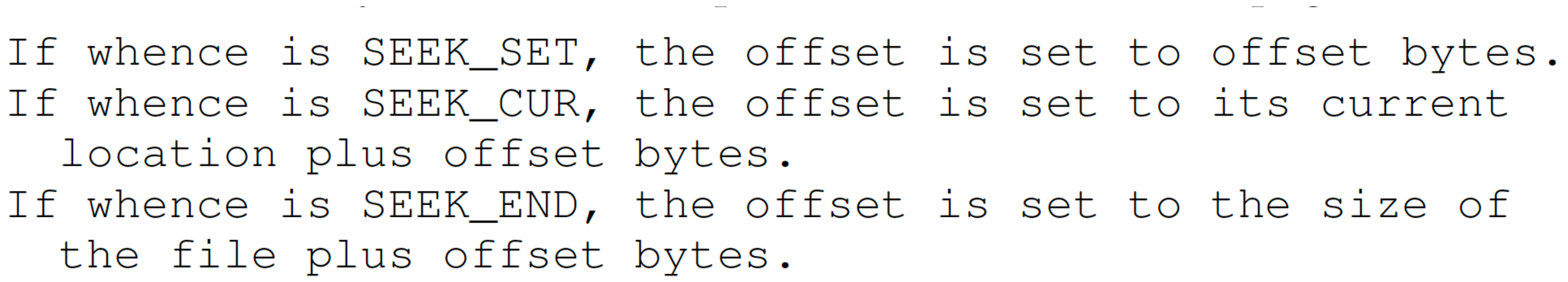
System Calls	Return Code	OFT[10] Current Offset	OFT[11] Current Offset
fd1 = open("file", O_RDONLY);	3	0	–
fd2 = open("file", O_RDONLY);	4	0	0
read(fd1, buffer1, 100);	100	100	0
read(fd2, buffer2, 100);	100	100	100
close(fd1);	0	–	100
close(fd2);	0	–	–

Reading And Writing, But Not Sequentially

- Sometimes, however, it is useful to be able to read or write to a specific offset within a file
- lseek() system call be used to perform read or write operations at **random** offsets within the document

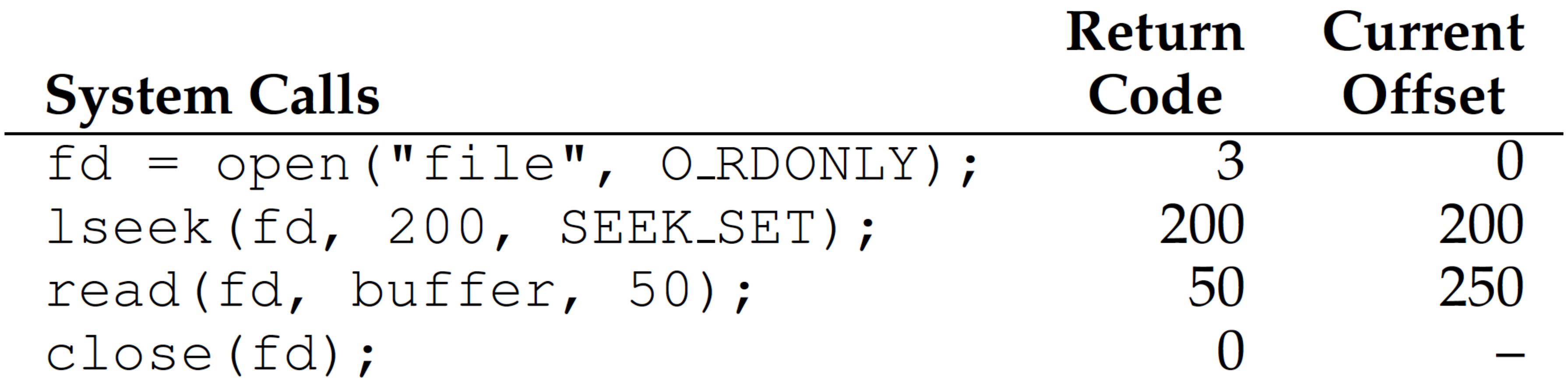


```
off_t lseek(int fildes, off_t offset, int whence);
```



```
If whence is SEEK_SET, the offset is set to offset bytes.
If whence is SEEK_CUR, the offset is set to its current
  location plus offset bytes.
If whence is SEEK_END, the offset is set to the size of
  the file plus offset bytes.
```

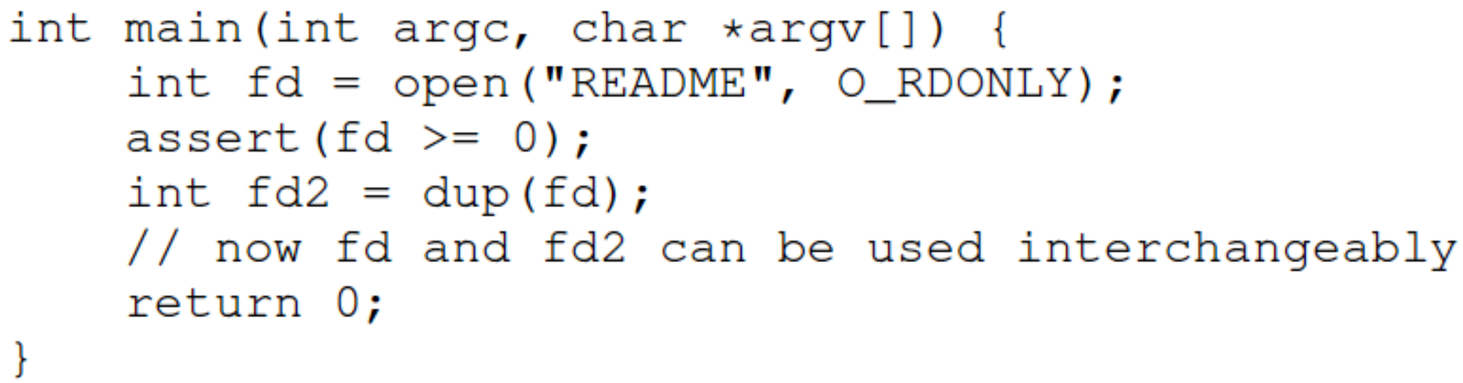
Example



System Calls	Return Code	Current Offset
`fd = open("file", O_RDONLY);`	3	0
`lseek(fd, 200, SEEK_SET);`	200	200
`read(fd, buffer, 50);`	50	250
`close(fd);`	0	–

Shared File Table Entries

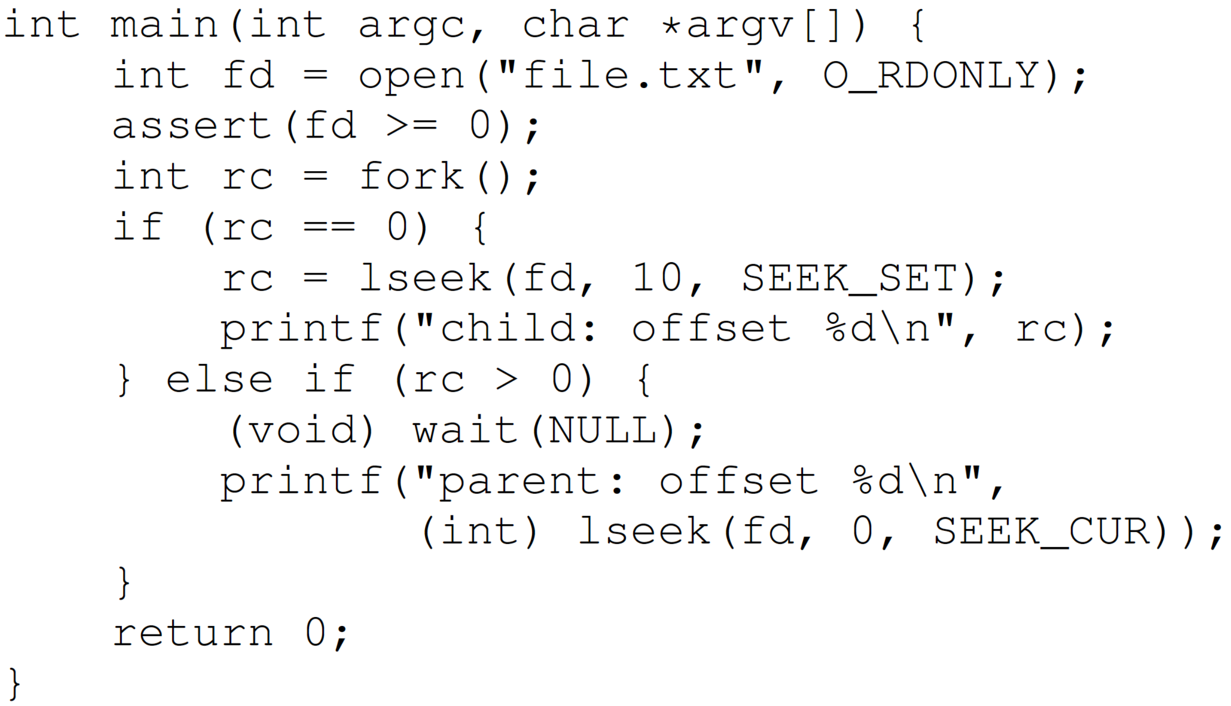
- Two processes accessing the same file at the same time
  - Each will have its own entry in the OFT
  - Each process can independently perform read/write on the file.
- In some cases, an entry in OFT is **shared**.
- **Dup()** system call



```
int main(int argc, char *argv[]) {
    int fd = open("README", O_RDONLY);
    assert(fd >= 0);
    int fd2 = dup(fd);
    // now fd and fd2 can be used interchangeably
    return 0;
}
```

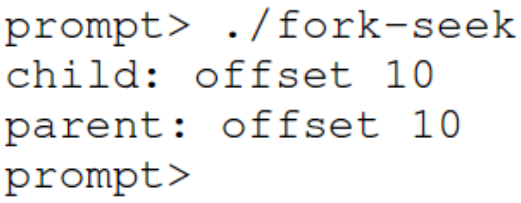
Figure 39.4: **Shared File Table Entry With dup() (dup.c)**

Parent and child process sharing file table entry



```
int main(int argc, char *argv[]) {
    int fd = open("file.txt", O_RDONLY);
    assert(fd >= 0);
    int rc = fork();
    if (rc == 0) {
        rc = lseek(fd, 10, SEEK_SET);
        printf("child: offset %d\n", rc);
    } else if (rc > 0) {
        (void) wait(NULL);
        printf("parent: offset %d\n",
               (int) lseek(fd, 0, SEEK_CUR));
    }
    return 0;
}
```

Figure 39.2: **Shared Parent/Child File Table Entries (`fork-seek.c`)**



```
prompt> ./fork-seek
child: offset 10
parent: offset 10
prompt>
```

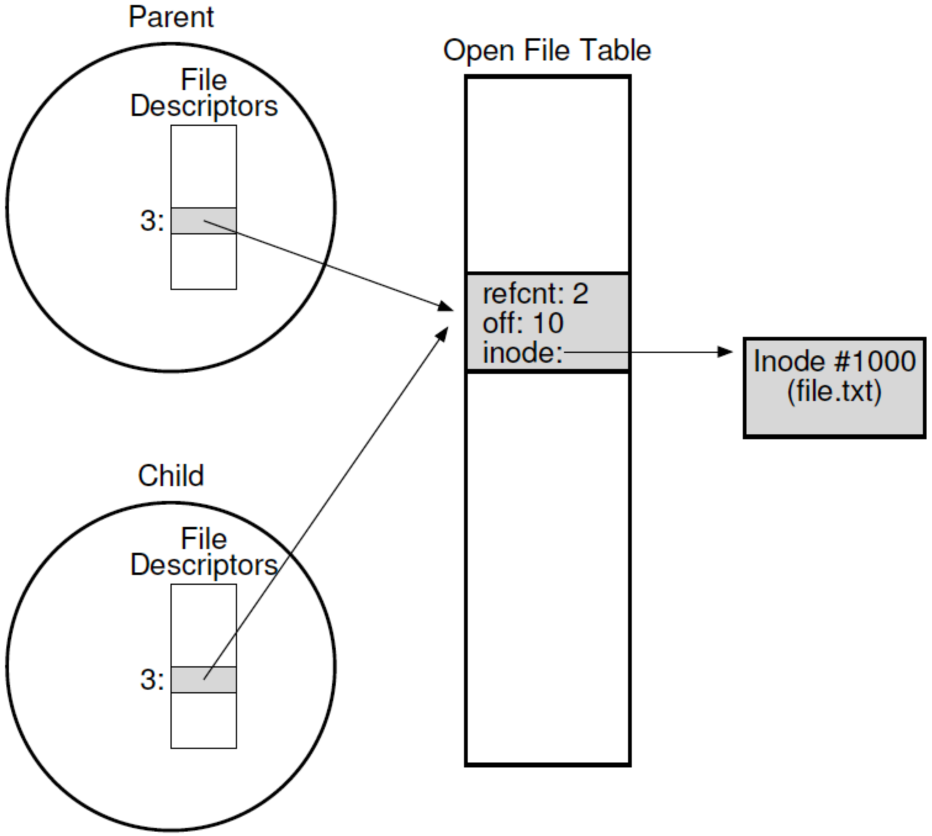
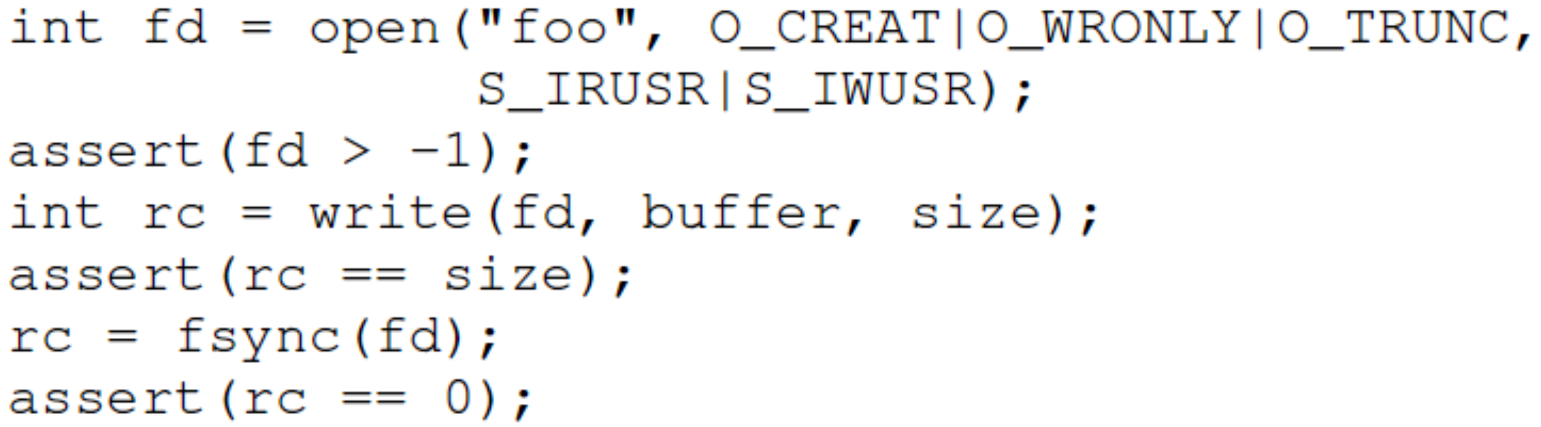


Figure 39.3: **Processes Sharing An Open File Table Entry**

Writing Immediately with fsync()

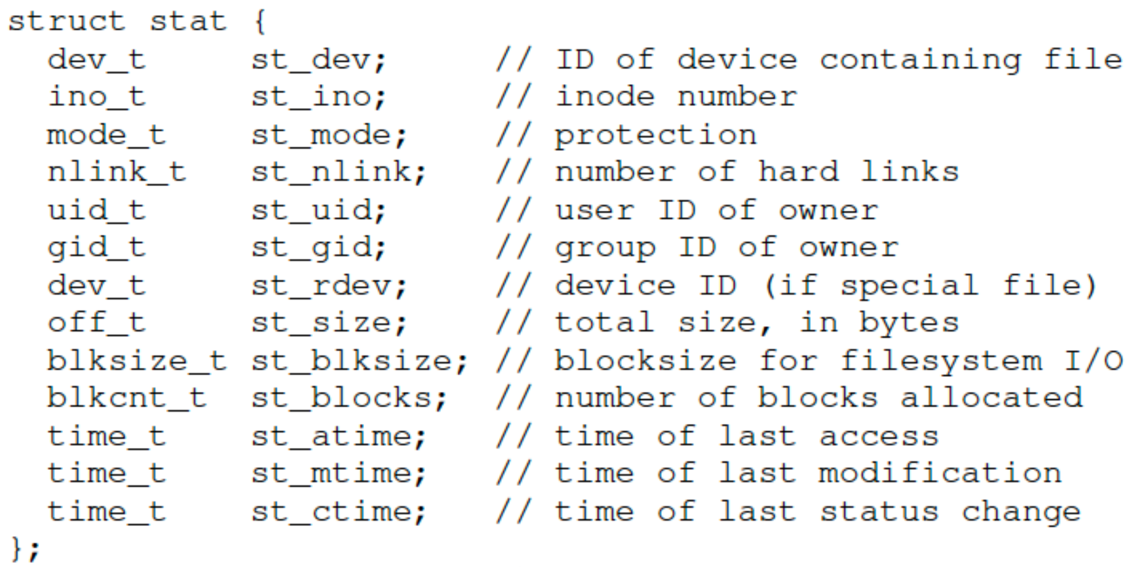
- Writes performed using **write()** are buffered in memory for some time for performance reasons.
- The writes are issued to the storage device at a later point in time.
- We can force immediate write to storage device using **fsync()**



```
int fd = open("foo", O_CREAT|O_WRONLY|O_TRUNC,
              S_IRUSR|S_IWUSR);
assert(fd > -1);
int rc = write(fd, buffer, size);
assert(rc == size);
rc = fsync(fd);
assert(rc == 0);
```

Other system calls

- **rename(“new_file_name”, “old_file_name”)** for renaming a file
- **fstat(file_descriptor)** for fetching meta data associated with a file
  - Metadata is stored in a structure called as **inode.**
  - **Inode** is a persistent data structure stored on the disk.



```
struct stat {
  dev_t     st_dev;     // ID of device containing file
  ino_t     st_ino;     // inode number
  mode_t    st_mode;    // protection
  nlink_t   st_nlink;   // number of hard links
  uid_t     st_uid;     // user ID of owner
  gid_t     st_gid;     // group ID of owner
  dev_t     st_rdev;    // device ID (if special file)
  off_t     st_size;    // total size, in bytes
  blksize_t st_blksize; // blocksize for filesystem I/O
  blkcnt_t  st_blocks;  // number of blocks allocated
  time_t    st_atime;   // time of last access
  time_t    st_mtime;   // time of last modification
  time_t    st_ctime;   // time of last status change
};
```

Figure 39.5: **The `stat` structure.**

Directory-related System calls

- **mkdir(“file_name”, permissions)** for creating a directory
- **rmdir()** for deleting a empty directory
- **Reading Directory**

```
int main(int argc, char *argv[]) {
    DIR *dp = opendir(".");
    assert(dp != NULL);
    struct dirent *d;
    while ((d = readdir(dp)) != NULL) {
        printf("%lu %s\n", (unsigned long) d->d_ino,
               d->d_name);
    }
    closedir(dp);
    return 0;
}
```

```
struct dirent {
  char           d_name[256]; // filename
  ino_t          d_ino;       // inode number
  off_t          d_off;       // offset to the next dirent
  unsigned short d_reclen;    // length of this record
  unsigned char  d_type;      // type of file
};
```