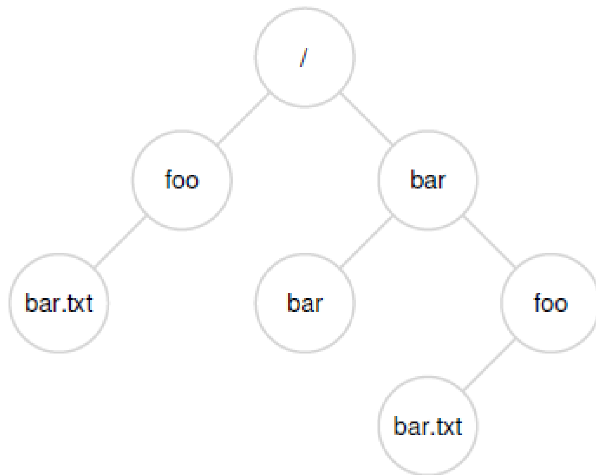
Files and Directories

Files and Directories

- File
  - simply a linear array of bytes, each of which you can read or write.
  - Each file has a low-level name referred to as its inode number.
- Directory
  - also has an inode number,
  - contains a list of (user-readable name, low-level name) pairs.
  - Can build an arbitrary directory tree (or directory hierarchy)



/

foo	10
bar	20

/bar

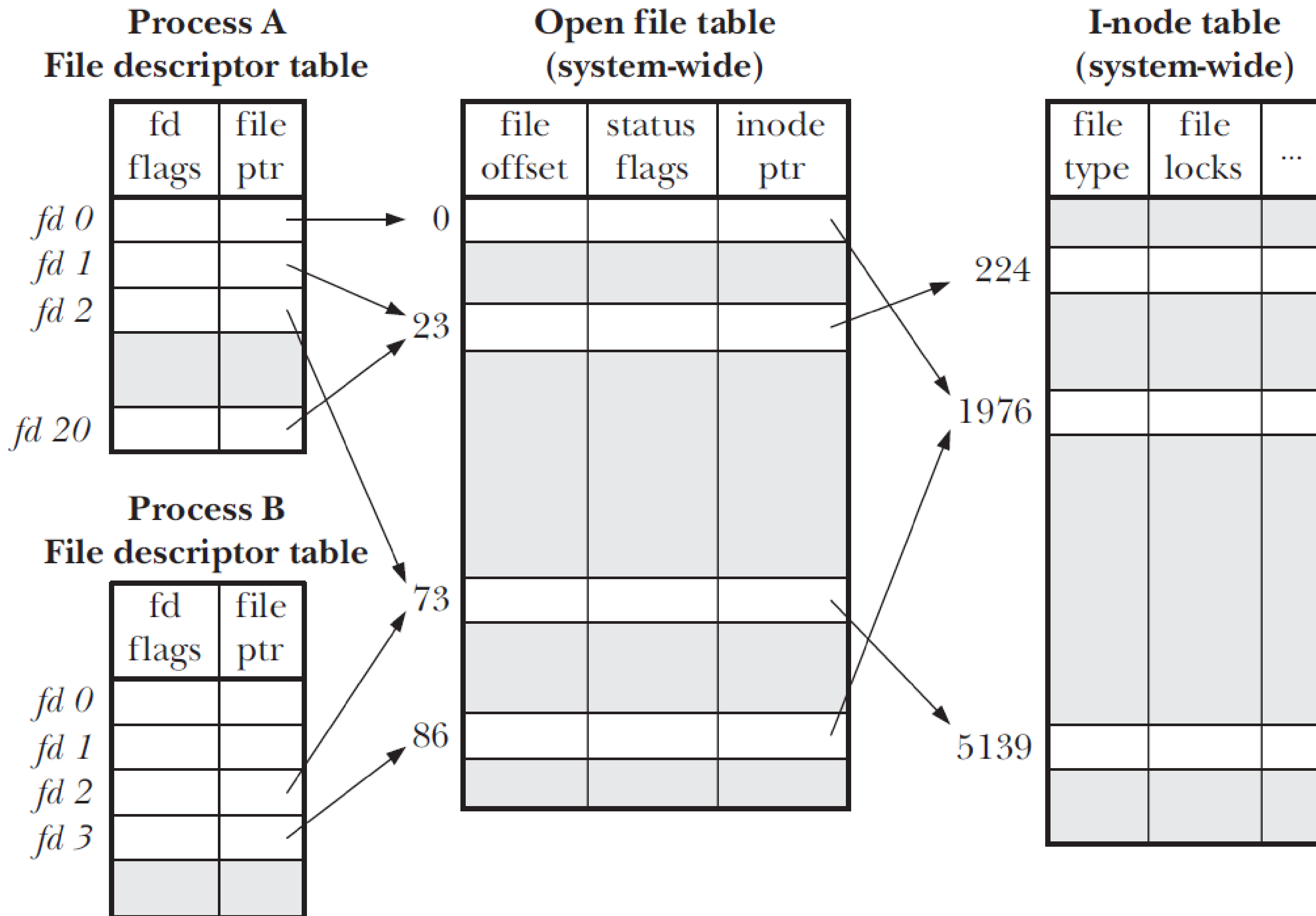
bar	201
foo	202

File System Interface - Creating Files

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

- The first parameter: file name
- The second parameter
  - O_CREAT: creates the file if it does not exist
  - O_WRONLY: ensures that the file can only be written to
  - O_TRUNC: if the file already exists, truncates it to a size of zero bytes thus removing any existing content.
  - More options can be found at the manual page.
- The third parameter specifies permissions
  - readable and writable by the owner.
- Returns a file descriptor
  - just an integer, private per process
  - Once a file is opened, you use the file descriptor to read or write the file
- cf. `int fd = creat("foo");`

File Descriptor & Inode



Reading and Writing Files

By running strace, you can trace which system calls a program makes and see the arguments and return codes.

```
prompt> strace cat foo
...
open("foo", O_RDONLY|O_LARGEFILE)= 3
read(3, "hello\n", 4096)        = 6
write(1, "hello\n", 6)          = 6
hello
read(3, "", 4096)               = 0
close(3)                        = 0
...
prompt>
```

opened for reading (not writing), the 64-bit offset be used (O LARGEFILE); open() succeeds and returns a FD (3). Why 3? Not 0 or 1?

4KB buffer, returning the number of bytes it read (6)

FD=1 stdout
Why not printf()?

Reading And Writing, But Not Sequentially

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

- Reading from some **random** offsets
- whence
  - determines exactly how the seek is performed.
  - SEEK_SET: offset = offset bytes.
  - SEEK_CUR: offset = current location + offset bytes.
  - SEEK_END: offset = file size + offset bytes.
- For each file a process opens, the OS tracks a “current” offset
- When a read or write of N bytes takes place, N is added to the current offset

Writing Immediately with fsync()

- write () system call
  - Just telling the file system: please write this data to persistent storage, at some point in the future
  - The file system, for performance reasons, will buffer such writes in memory for some time
  - Machine crashes after write() will make data to be lost.
- `fsync(int fd)`
  - forcing all dirty (i.e., not yet written) data to disk
  - returns once all of these writes are complete.

```
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);
```

cf. fdatasync()

Renaming Files

```
prompt> mv foo bar
```

```
int fd = open("foo.txt.tmp", O_WRONLY|O_CREAT|O_TRUNC,
S_IRUSR|S_IWUSR);
write(fd, buffer, size); // write out new version of file
fsync(fd);
close(fd);
rename("foo.txt.tmp", "foo.txt");
```

`rename()` is an atomic call with respect to system crashes

Getting Information About Files

- To see the metadata for a certain file, we can use the stat() or fstat()
- Take a pathname (or file descriptor) and fill in a stat structure

```
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 */
};
```

```
prompt> echo hello > file
prompt> stat file
File: ‘file’
Size: 6 Blocks: 8 IO Block: 4096 regular file
Device: 811h/2065d Inode: 67158084 Links: 1
Access: (0640/-rw-r-----) Uid: (30686/ remzi) Gid: (30686/ remzi)
Access: 2011-05-03 15:50:20.157594748 -0500
Modify: 2011-05-03 15:50:20.157594748 -0500
Change: 2011-05-03 15:50:20.157594748 -0500
```

Removing Files

- Why “unlink”? Why not just “remove” or “delete”?

```
prompt> strace rm foo
...
unlink("foo") = 0
...
```

Useful command “rm –rf”

https://www.theregister.co.uk/2017/02/01/gitlab_data_loss/

Making/Deleting Directories

- To create a directory, a single system call, mkdir(), is available

```
prompt> strace mkdir foo
...
mkdir("foo", 0777) = 0
...
prompt>
```

- When a directory is created, it is considered “empty”.
- An empty directory has two entries
  - one entry that refers to itself (“.”),
  - another entry that refers to its parent (“..”).

```
prompt> ls -a
./ ../
prompt> ls -al
total 8
drwxr-x--- 2 remzi remzi 6 Apr 30 16:17 ./
drwxr-x--- 26 remzi remzi 4096 Apr 30 16:17 ../
```

- Deleting a directory
  - rmdir()
  - directory must be empty (i.e., only has “.” and “..” entries) before it is deleted

Reading Directories

- prints the contents of a directory

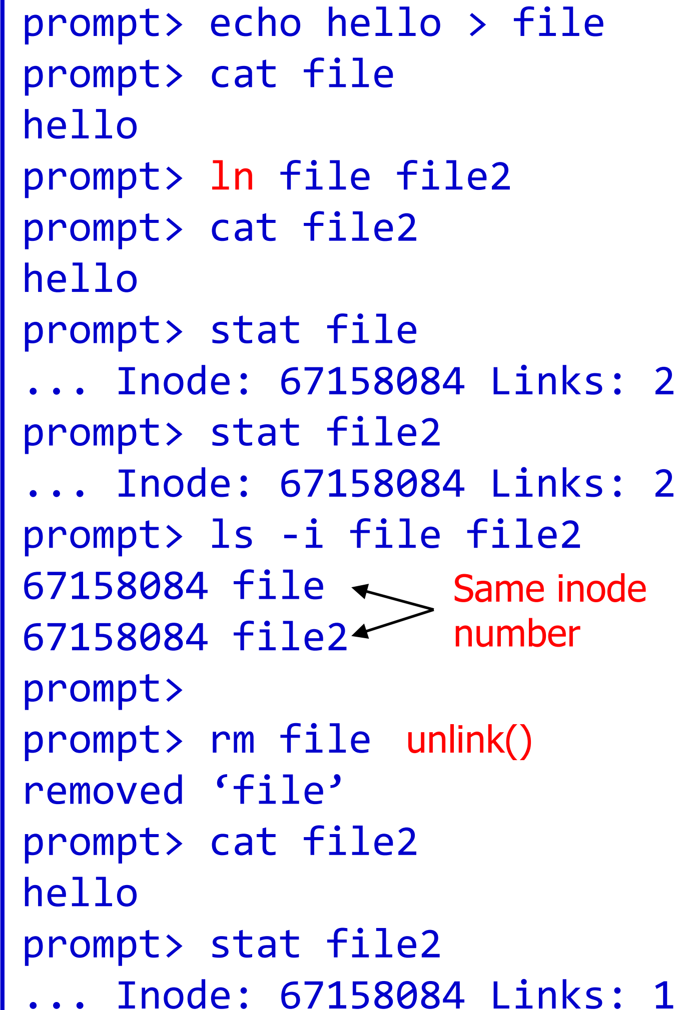
```
int main(int argc, char *argv[]) {
     DIR *dp = opendir(".");
     assert(dp != NULL);
     struct dirent *d;
     while ((d = readdir(dp)) != NULL) {
          printf("%lu %s\n", (unsigned long) d->d_ino, d->d_name);
     }
     closedir(dp);
     return 0;
}
```

```
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 */
};
```

Hard Links

- link() system call "link" a new file name to an old one
  - create another name to refer to the same file.
  - refers it to the *same* inode number
  - The file is not copied

- When unlink() is called, it removes the "link" between file name and inode.
  - checks a reference count within the inode number.
  - Only when the reference count = 0, FS frees the inode and related data blocks, and thus truly "delete" the file.



Symbolic Links (Soft Links)

- **Hard link limitations**
  - A hard link to a directory will create a cycle in the directory tree
  - can't hard link to files in other disk partitions
    - inode numbers are only unique within a particular file system

```
prompt> echo hello > file
prompt> ln -s file file2
prompt> cat file2
hello
prompt> stat file
... regular file ...
prompt> stat file2
... symbolic link ...
prompt> ls -al
drwxr-x--- 2 usr usr 29 May 3 19:10 ./
drwxr-x--- 27 usr usr 4096 May 3 15:14 ../
-rw-r----- 1 usr usr 6 May 3 19:10 file
lrwxrwxrwx 1 usr usr 4 May 3 19:10 file2 -> file
prompt> rm file
prompt> cat file2
cat: file2: No such file or directory
```

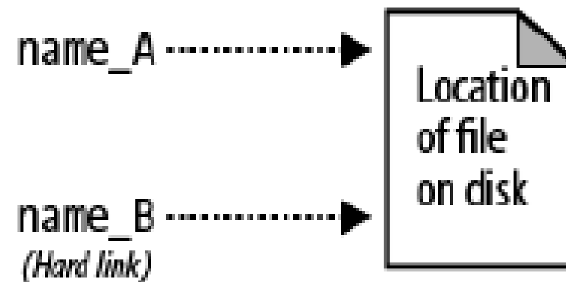
- **Symbolic link**
  - actually a file itself, of a different type
  - possibility for a **dangling reference**

4 bytes for holding the pathname

Hard link vs. Symbolic link

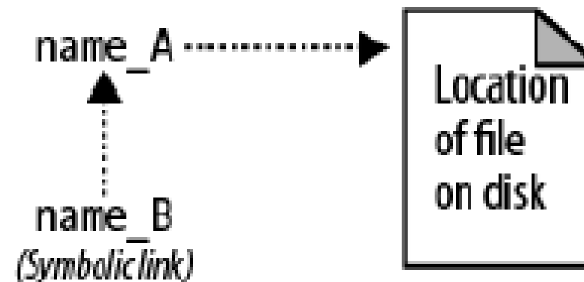
Hard link:

```
$ ln name_A name_B
```



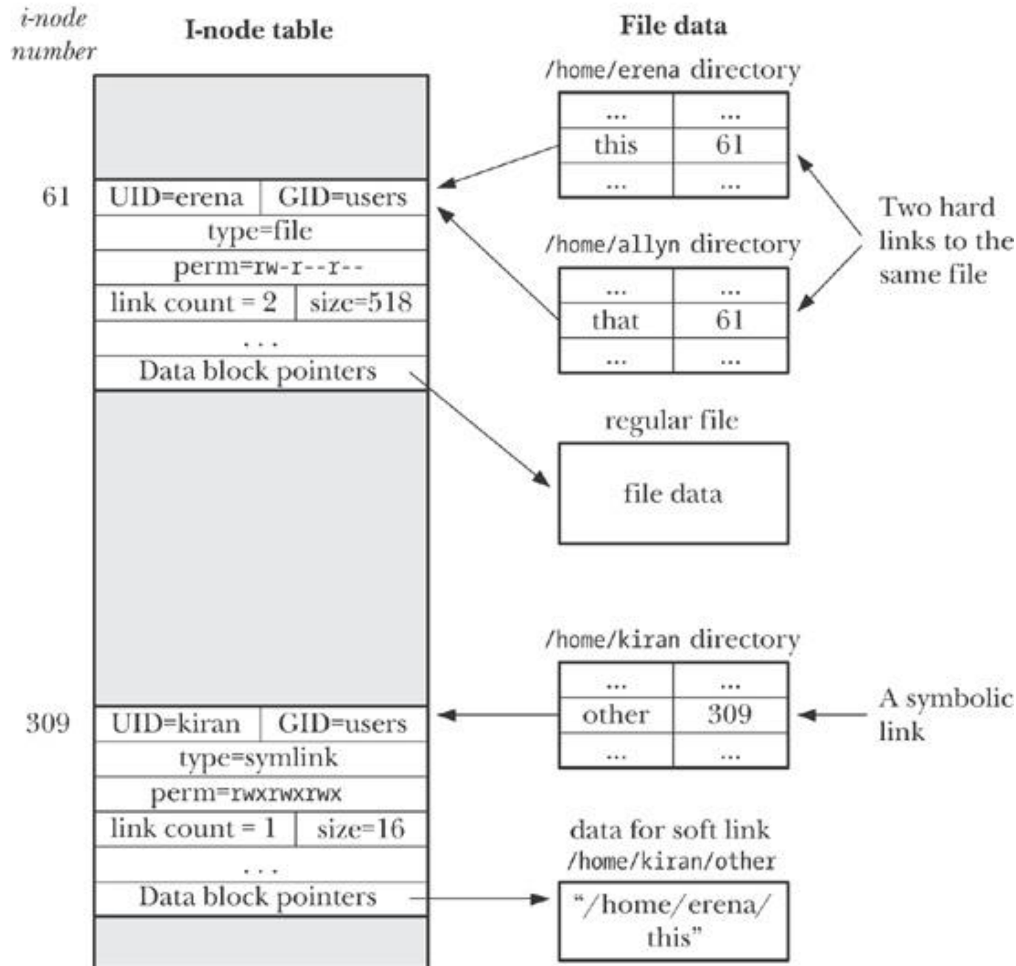
Symbolic link:

```
$ ln -s name_A name_B
```



Hard link vs. Symbolic link

$ ln /home/erena/this /home/allyn/that
$ ln –s /home/erena/this /home/kiran/other



Making and Mounting a File System

- To make a file system, mkfs
  - Input: device (e.g., /dev/sda1), file system type (e.g., ext3)
- To make a file system to be accessible within the uniform file-system tree, mount

```
prompt> mount -t ext3 /dev/sda1 /home/users
prompt> ls /home/users/
a b
prompt> mount
/dev/sda1 on / type ext3 (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
/dev/sda5 on /tmp type ext3 (rw)
/dev/sda7 on /var/vice/cache type ext3 (rw)
tmpfs on /dev/shm type tmpfs (rw)
AFS on /afs type afs (rw)
```

Homework

- Homework in Chap 39 (FS APIs)