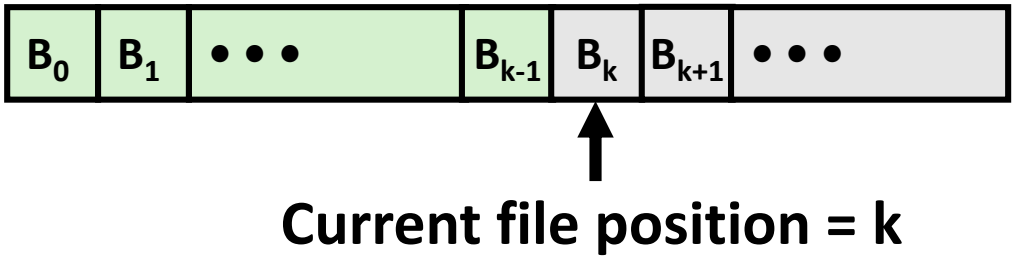
# Unix I/O Overview

- **A Linux *file* is a sequence of *m* bytes:**
  - $B_0, B_1, \ldots, B_k, \ldots, B_{m-1}$
- **Cool fact: All I/O devices are represented as files:**
  - **`/dev/sda2`** (**`/usr`** disk partition)
  - **`/dev/tty2`** (terminal)
- **Even the kernel is represented as a file:**
  - **`/boot/vmlinuz-3.13.0-55-generic`** (kernel image)
  - **`/proc`** (kernel data structures)

# Unix I/O Overview

- **Elegant mapping of files to devices allows kernel to export simple interface called *Unix I/O:***
  - Opening and closing files
    - **`open()`** and **`close()`**
  - Reading and writing a file
    - **`read()`** and **`write()`**
  - Changing the ***current file position*** (seek)
    - indicates next offset into file to read or write
    - **`lseek()`**



# File Types

- **Each file has a *type* indicating its role in the system**
  - *Regular file:* Contains arbitrary data
  - *Directory:* Index for a related group of files
  - *Socket:* For communicating with a process on another machine

- **Other file types beyond our scope**
  - *Named pipes (FIFOs)*
  - *Symbolic links*
  - *Character and block devices*

# Regular Files

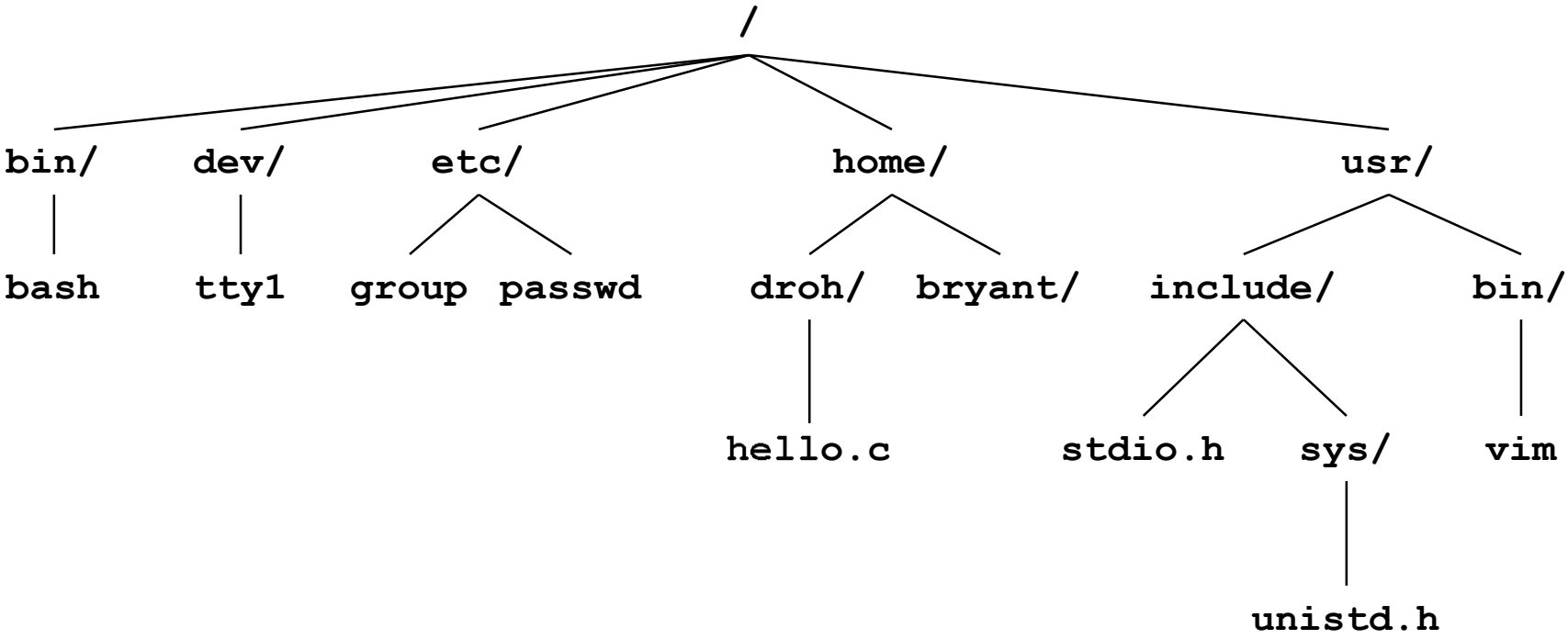
- **A regular file contains arbitrary data**
- **Applications often distinguish between *text files* and *binary files***
  - Text files are regular files with only ASCII or Unicode characters
  - Binary files are everything else
    - e.g., object files, JPEG images
  - Kernel doesn't know the difference!
- **Text file is sequence of *text lines***
  - Text line is sequence of chars terminated by *newline char* ('`\n`')
    - Newline is `0xa`, same as ASCII line feed character (LF)
- **End of line (EOL) indicators in other systems**
  - Linux and Mac OS: '`\n`' (`0xa`)
    - line feed (LF)
  - Windows and Internet protocols: '`\r\n`' (`0xd 0xa`)
    - Carriage return (CR) followed by line feed (LF)

# Directories

- **Directory consists of an array of *links***
  - Each link maps a *filename* to a file
- **Each directory contains at least two entries**
  - `.` (dot) is a link to itself
  - `..` (dot dot) is a link to *the parent directory* in the *directory hierarchy* (next slide)
- **Commands for manipulating directories**
  - `mkdir`: create empty directory
  - `ls`: view directory contents
  - `rmdir`: delete empty directory

# Directory Hierarchy

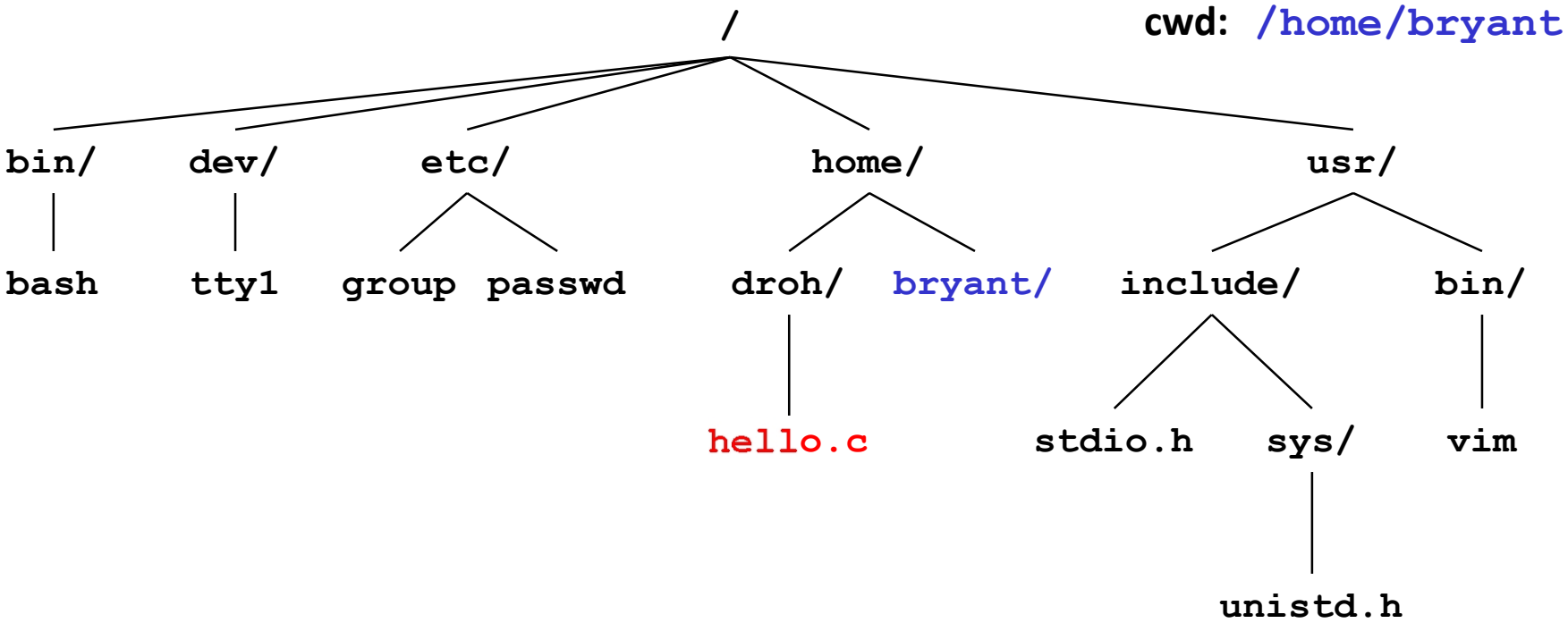
- **All files are organized as a hierarchy anchored by root directory named / (slash)**




- **Kernel maintains *current working directory (cwd)* for each process**
  - Modified using the `cd` command

# Pathnames

- **Locations of files in the hierarchy denoted by *pathnames***
  - *Absolute pathname* starts with '/' and denotes path from root
    - `/home/droh/hello.c`
  - *Relative pathname* denotes path from current working directory
    - `../droh/hello.c`



# Opening Files

- **Opening a file informs the kernel that you are getting ready to access that file**

```
int fd;    /* file descriptor */

if ((fd = open("/etc/hosts", O_RDONLY)) < 0) {
   perror("open");
   exit(1);
}
```

- **Returns a small identifying integer *file descriptor***
  - `fd == -1` indicates that an error occurred
- **Each process created by a Linux shell begins life with three open files associated with a terminal:**
  - 0: standard input (stdin)
  - 1: standard output (stdout)
  - 2: standard error (stderr)

# Closing Files

- Closing a file informs the kernel that you are finished accessing that file

```
int fd;     /* file descriptor */
int retval; /* return value */

if ((retval = close(fd)) < 0) {
   perror("close");
   exit(1);
}
```

- Closing an already closed file is a recipe for disaster in threaded programs (more on this later)
- Moral: Always check return codes, even for seemingly benign functions such as `close()`

# Reading Files

- **Reading a file copies bytes from the current file position to memory, and then updates file position**

```
char buf[512];
int fd;        /* file descriptor */
int nbytes;    /* number of bytes read */

/* Open file fd ...  */
/* Then read up to 512 bytes from file fd */
if ((nbytes = read(fd, buf, sizeof(buf))) < 0) {
   perror("read");
   exit(1);
}
```

- **Returns number of bytes read from file `fd` into `buf`**
  - Return type **`ssize_t`** is signed integer
  - **`nbytes < 0`** indicates that an error occurred
  - ***Short counts*** (**`nbytes < sizeof(buf)`** ) are possible and are not errors!