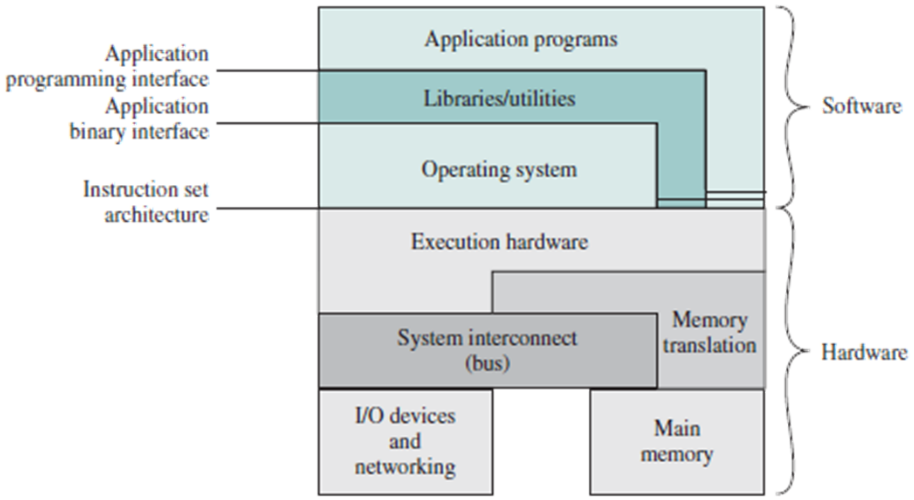
CSE320 System Fundamentals II
System APIs

YoungMin Kwon

Unix I/O

- Input/Output operations
  - Typically, applications do not access hardware directly
  - The request has to go through the system stack

- System stack
  - Application
  - Libraries
  - Operating System
  - Hardware


Application programming interface
Application binary interface
Instruction set architecture
Application programs
Libraries/utilities
Operating system
Execution hardware
Memory translation
System interconnect (bus)
I/O devices and networking
Main memory
Software
Hardware

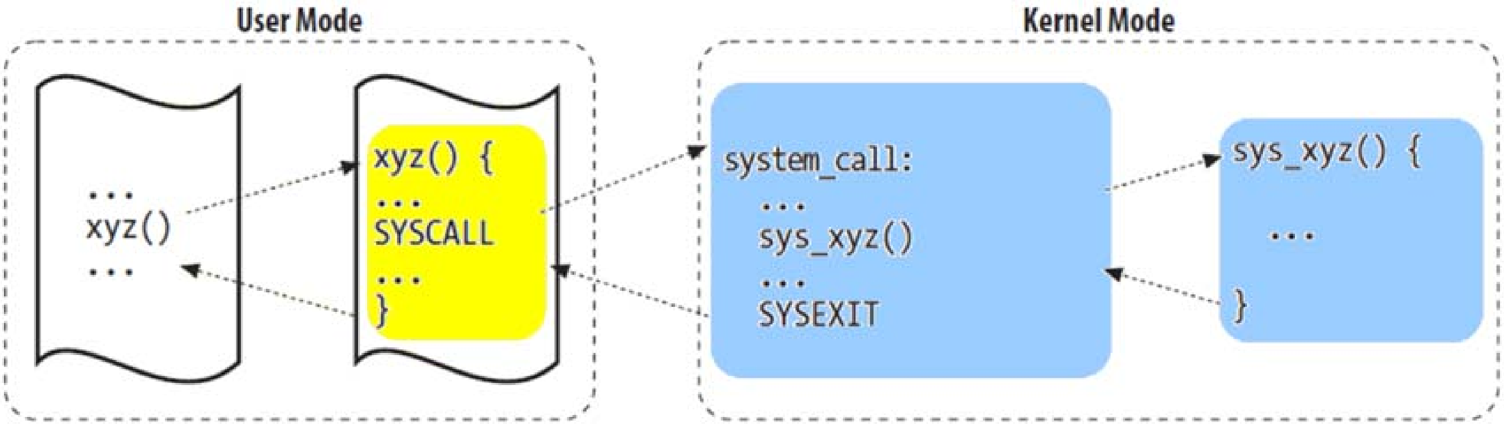
Unix I/O

- Input/Output
  - Process of copying data between main memory and external devices like disk drives, terminals, networks.

- Unix I/O
  - Unix I/O functions (read, write, …) are provided by the OS
  - Standard I/O library functions (printf, scanf, …) are implemented using Unix I/O functions

Unix I/O

- System call



```
...
xyz()
...
```

```
xyz() {
...
SYSCALL
...
}
```

```
system_call:
  ...
  sys_xyz()
  ...
  SYSEXIT
```

```
sys_xyz() {
  ...
}
```

Standard I/O
printf, scanf, …

Unix I/O
(wrapper routines)
read, write, …

System call
handler

System call
service routine

Why Unix I/O (when there is Standard I/O)

- Understanding Unix I/O will help understand other system concepts
  - e.g. Process creation and Opening a file
- Sometimes, there are no other choices but to use Unix I/O

Unix I/O: Files

- A Linux file is a sequence of bytes
  - B_0, B_1, …, B_k, …, B_m
- All I/O devices are modeled as *files*
  - E.g. networks, disks, terminals
  - Input and output are performed by reading from and writing to the appropriate files
  - This mapping enables simple low level APIs known as Unix I/O

Unix I/O: Files

- Opening a file
  - Announcing an app's intention to access an I/O device.
  - Kernel returns a descriptor

- Changing the current file position
  - A byte offset from the beginning of a file
  - The kernel maintains a file position for each open file
  - *seek* operation can change the file position

Unix I/O: Files

- Reading and writing files
  - *read* copies n bytes from the current position of a file to memory
  - *write* copies n bytes from memory to the current position of a file

- Closing files
  - Informs the kernel that the app has finished accessing the file

Some File Types

- A regular file
  - Contains arbitrary data
  - Text file, Binary file

- A directory
  - A file containing an array of links that map file names to files
  - `'.'` is a link to the directory itself, `'..'` is a link to the parent directory

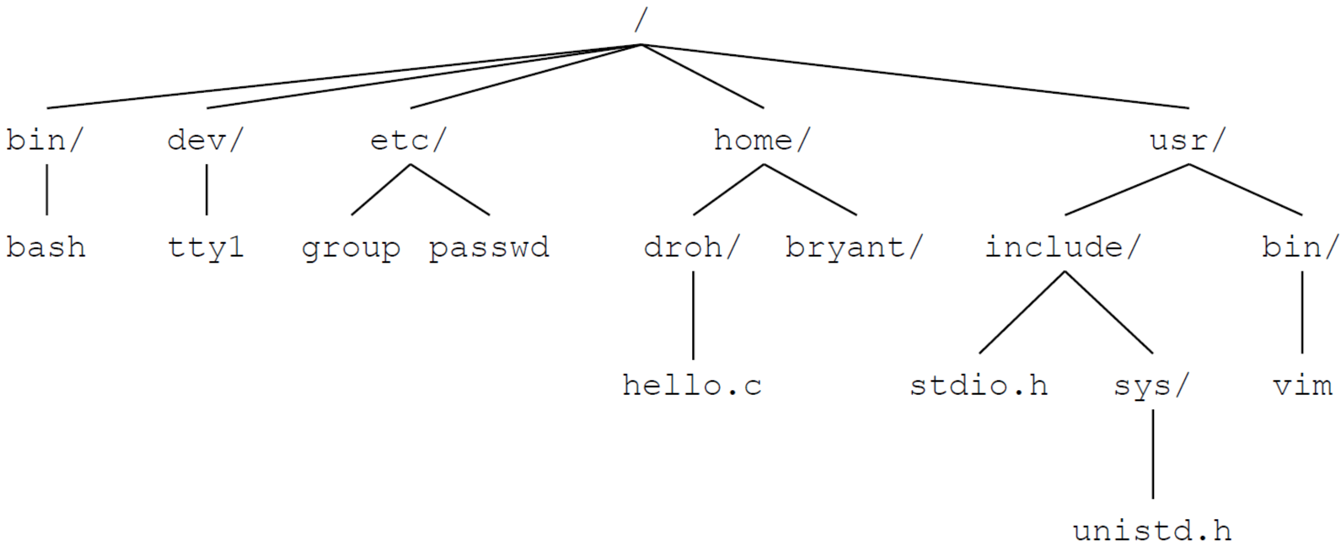
- A socket
  - A file used to communicate with another process

- Other files
  - Named pipes, symbolic links, character and block devices

Files

- Linux kernel organizes all files in a single directory hierarchy anchored by the *root directory "/"*

- Each process has a *current working directory*



Opening and Closing Files

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open(char *filename, int flags, mode_t mode);

// flags
O_RDONLY // Reading only
O_WRONLY // Writing only
O_RDWR   // Reading and Writing

O_CREAT  // If file doesn't exist, create a truncated one
O_TRUNC  // If file already exists, truncate it
O_APPEND // Set the file position to the end of the file

// modes
S_IRUSR, S_IRGRP, S_IROTH  // User, group, other can read this file
S_IWUSR, S_IWGRP, S_IWOTH  // User, group, other can write this file
S_IXUSR, S_IXGRP, S_IXOTH  // User, group, other can execute this file
```

Opening and Closing Files

```c
#include <unistd.h>
int close(int fd);
```

```c
#define DEF_MODE  S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP|S_IROTH|S_IWOTH
#define DEF_UMASK S_IWGRP|S_IWOTH

umask(DEF_UMASK);     //umask bits will be unset
int  fd = open("foo.txt", O_CREAT|O_TRUNC|O_WRONLY, DEF_MODE);
close(fd);
```

Reading and Writing Files

```
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t n);
ssize_t write(int fd, const void *buf, size_t n);
```

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
int main() {
    int fd1 = open("foo.txt", O_CREAT|O_WRONLY, S_IRUSR|S_IWUSR);
    int fd2 = open("foo.txt", O_RDONLY, 0);
    close(fd1);
    close(fd2);
    printf("fd1: %d, fd2: %d\n", fd1, fd2);
}
```

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
int main() {
    char buf[100];
    int fd1, fd2, n, i;
    // read from the standard input
    n = read(STDIN_FILENO/*0*/, buf, sizeof(buf));

    // write to a file
    fd1 = open("foo.txt", O_CREAT|O_WRONLY, S_IRUSR|S_IWUSR);
    for(i = 0; i < n; )
        i += write(fd1, buf + i, n - i);
    close(fd1);

    // read from a file
    fd2 = open("foo.txt", O_RDONLY, 0);
    for(i = 0; i < n; )
        i += read(fd2, buf + i, n - i);
    close(fd2);

    // write to the standard output
    for(i = 0; i < n; )
        i += write(STDOUT_FILENO/*1*/, buf + i, n - i);
}
```

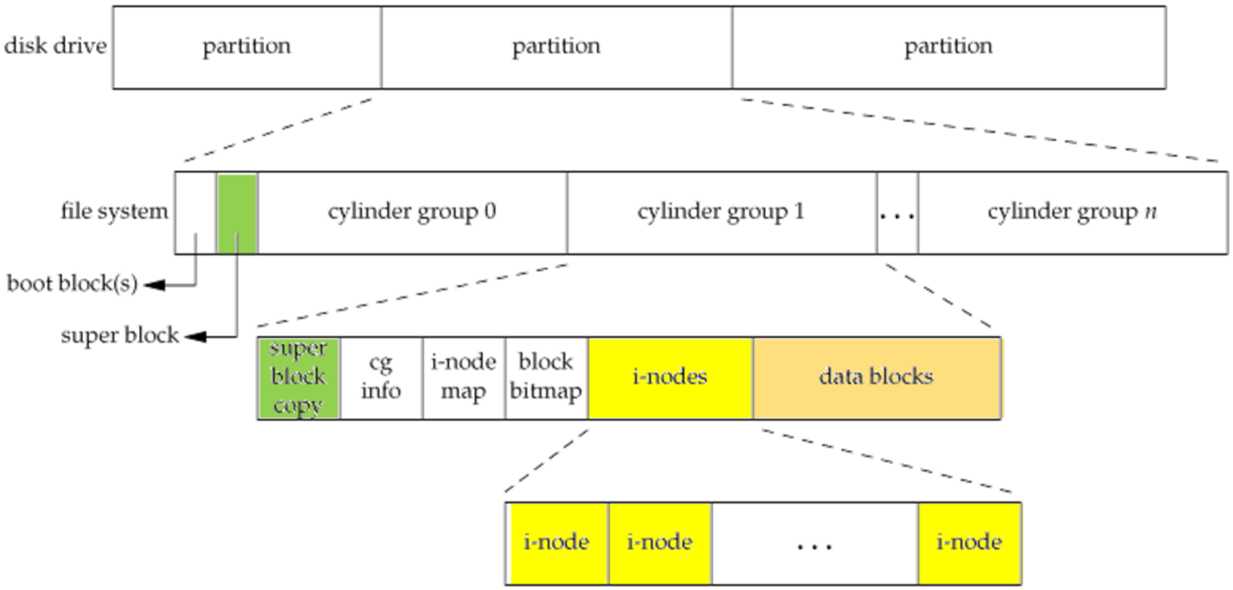
Reading File Metadata

```
#include <unistd.h>
#include <sys/stat.h>

int stat(const char *filename, struct stat *buf);
int lstat(const char *filename, struct stat *buf);
int fstat(int fd, struct stat *buf);

struct stat {
    dev_t     st_dev;     /* ID of device containing file */
    ino_t     st_ino;     /* inode number */
    mode_t    st_mode;    /* file type, 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 file system I/O */
    blkcnt_t  st_blocks;  /* number of 512B 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 */
};
```

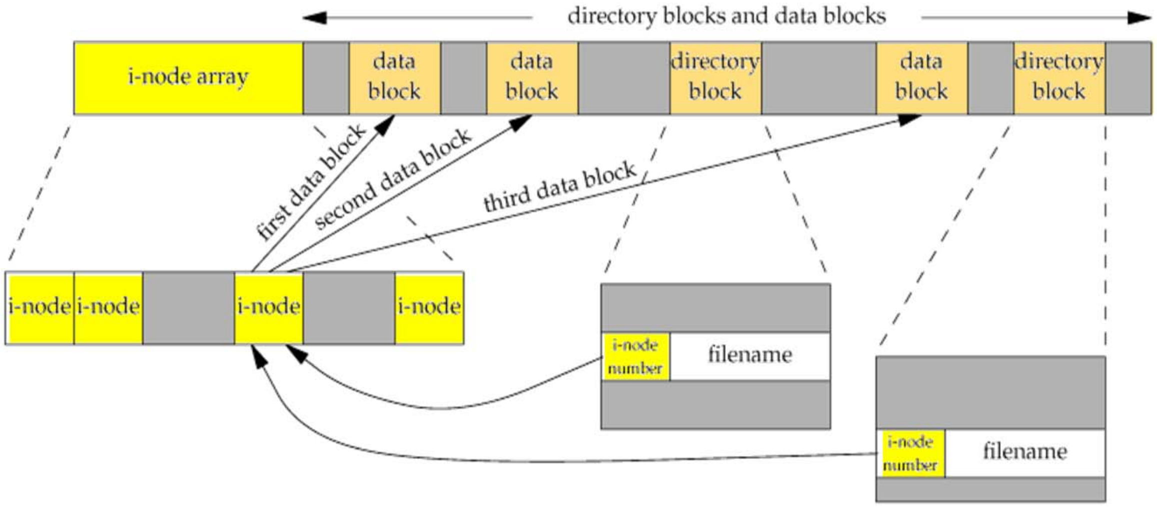
Reading File Metadata

```c
#include <unistd.h>
#include <sys/stat.h>
#include <time.h>
#include <stdio.h>

void printstat(char *fname);  //forward declaration

int main()
{
    printstat(".");
    printstat("./foo.c");
    printstat("/dev/tty0");
}
```

```
void printstat(char *fname)
{
    struct stat sb;
    stat(fname, &sb);

    printf("\n----------------------------------------\n");
    printf("File name:          %s\n", fname);
    printf("File type:          ");
    switch (sb.st_mode & S_IFMT) {
    case S_IFBLK:  printf("block device\n");     break;
    case S_IFCHR:  printf("character device\n"); break;
    case S_IFDIR:  printf("directory\n");        break;
    case S_IFIFO:  printf("FIFO/pipe\n");        break;
    case S_IFLNK:  printf("link\n");             break;
    case S_IFREG:  printf("regular file\n");     break;
    case S_IFSOCK: printf("socket\n");           break;
    default:       printf("unknown?\n");         break;
    }
    printf("i-node number:      %ld\n", (long)sb.st_ino);
    printf("File size:          %lld bytes\n", (long long)sb.st_size);
    printf("Last status change: %s", ctime(&sb.st_ctime));
    printf("Last access:        %s", ctime(&sb.st_atime));
    printf("Last modification:  %s", ctime(&sb.st_mtime));
}
```



disk drive
partition
partition
partition
file system
cylinder group 0
cylinder group 1
. . .
cylinder group *n*
boot block(s)
super block
super block copy
cg info
i-node map
block bitmap
i-nodes
data blocks
i-node
i-node
. . .
i-node



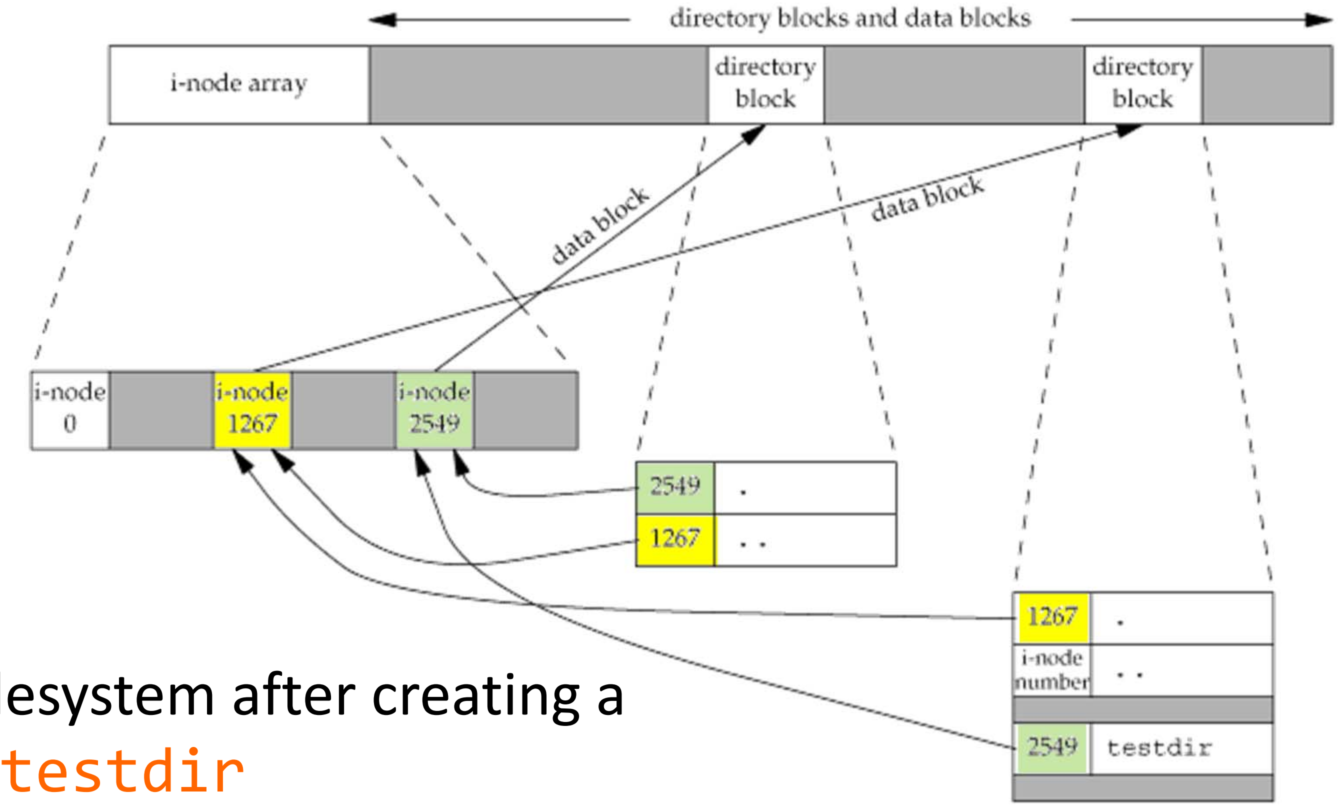
directory blocks and data blocks
i-node array
data block
data block
directory block
data block
directory block
first data block
second data block
third data block
i-node
i-node
i-node
i-node
i-node number
filename
i-node number
filename

Some Fields of super_block

```
// struct super_block: information about a file system
//
struct super_block {
    struct list_head s_list;         /* list of all superblocks */
    dev_t s_dev;                     /* identifier */
    unsigned long s_blocksize;       /* block size in bytes */
    unsigned char s_dirt;            /* dirty flag */
    struct file_system_type *s_type; /* filesystem type */
    struct super_operations *s_op;   /* superblock methods */
    unsigned long s_flags;           /* mount flags */
    unsigned long s_magic;           /* filesystem’s magic number */
    struct dentry *s_root;           /* directory mount point */
    int s_count;                     /* superblock ref count */
    int s_need_sync;                 /* not-yet-synced flag */
    struct list_head s_inodes;       /* list of inodes */
    struct list_head s_dirty;        /* list of dirty inodes */
    fmode_t s_mode;                  /* mount permissions */
...
};
```

Some Fields of inode

```
// struct inode: metadata about a file
//
struct inode {
    struct list_head i_sb_list;    /* inodes in the superblock */
    struct list_head i_dentry;     /* dentries referencing this inode*/
    unsigned long i_ino;           /* inode number */
    unsigned int i_nlink;          /* number of hard links */
    uid_t i_uid;                   /* user id of owner */
    gid_t i_gid;                   /* group id of owner */
    loff_t i_size;                 /* file size in bytes */
    struct timespec i_atime;       /* last access time */
    struct timespec i_mtime;       /* last modify time */
    struct timespec i_ctime;       /* last change time */
    umode_t i_mode;                /* access permissions */
    struct inode_operations *i_op; /* inode ops table */
    struct file_operations *i_fop; /* default inode ops */
    struct super_block *i_sb;      /* associated superblock */
    void *i_private;               /* fs private pointer */
...
};
```



Sample filesystem after creating a directory `testdir`

- Try `ls -ia`

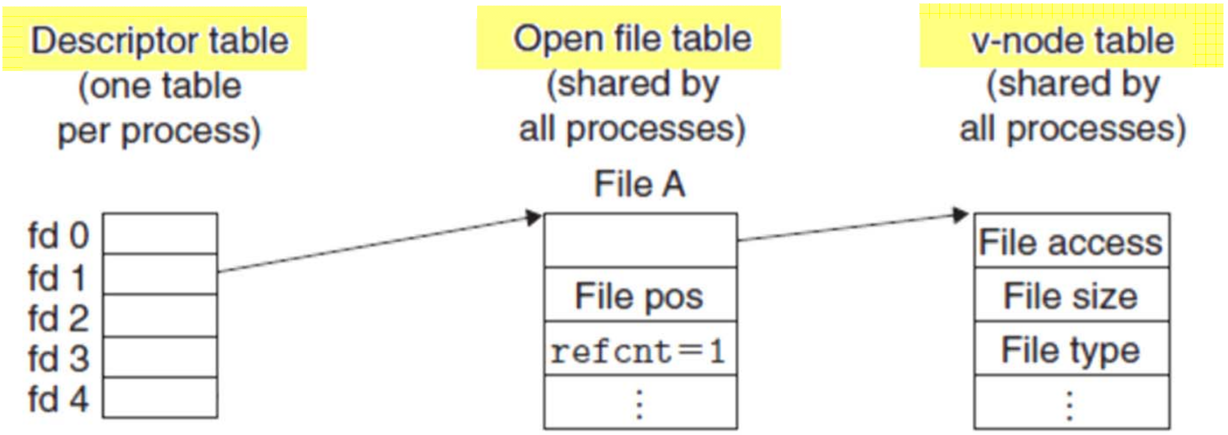
```
ykwon4@youngbox2:~/home/cse320$ ls -ia .
 5770154 .          10094940 func_env   5770074 loader       10095067 process   10095145 tmp
 5767413 ..         10094959 hw        10095026 mem_alloc    10095076 signal    10095161 var_flow
10094914 assembler  10094979 intro     10095044 network      10095094 sys_apis
…
ykwon4@youngbox2:~/home/cse320$ ls -ia sys_apis/
10095094 .       10095119 bar.txt      10095264 fork_wait.c      10095096 redirect.c
 5770154 ..      10095118 directory.c  10095106 monitor.c        10095109 redirect_pipe.c
10095095 a.txt   10095108 forkopen.c   10095103 readwrite.c      10095115 shell_base.c
```

Reading Directory Contents

```
#include <dirent.h>
DIR *opendir(const char *name);
int closedir(DIR *dirp);
struct dirent *readdir(DIR *dirp);

#include <dirent.h>
#include <stdio.h>
int main(int argc, char **argv) {
    DIR *dp;
    struct dirent *dep;
    char *name = (argc != 2 ? "/": argv[1]);

    dp = opendir(name);
    while((dep = readdir(dp)) != NULL)
        printf("%s\n", dep->d_name);

    closedir(dp);
    return 0;
}
```

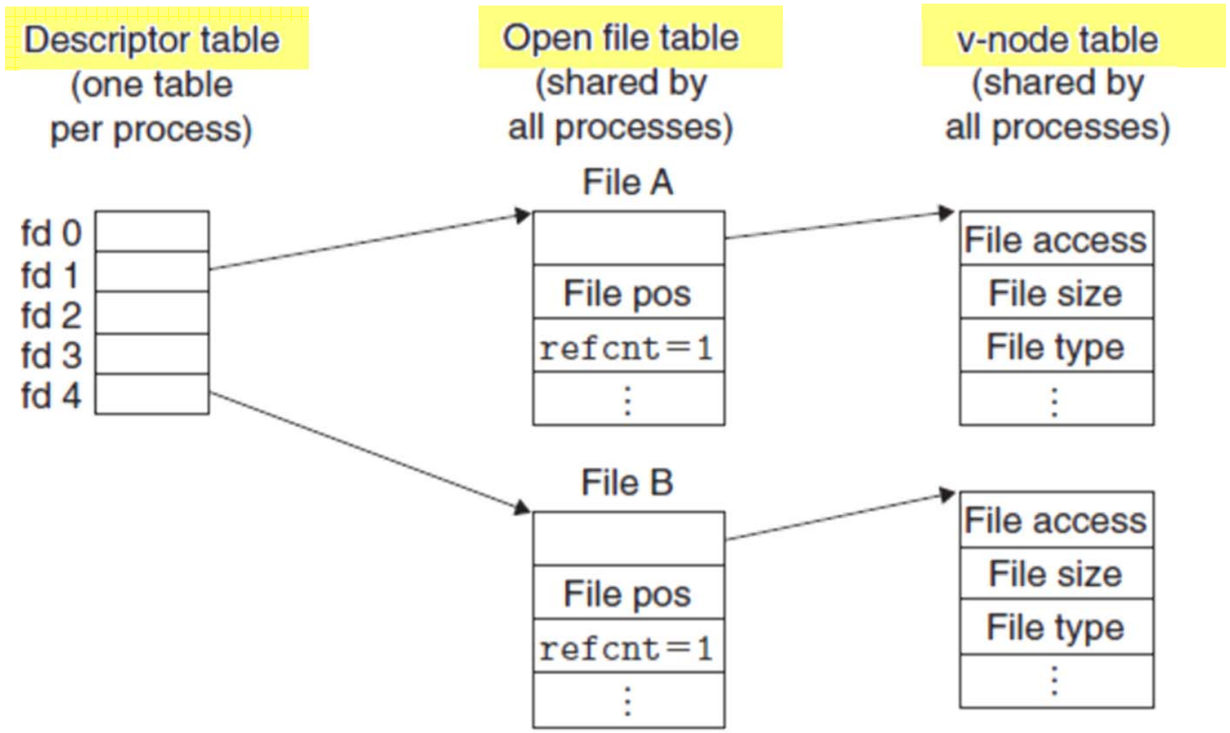
Sharing Files

- Descriptor Table
  - Each process has its own descriptor table
  - Each open descriptor entry points to a file table entry
- Open File Table
  - The set of open files is represented by a file table shared by all processes
  - File position, Reference count, Pointer to an entry in the v-node table
- V-node table
  - V-node table is shared by all processes
  - Each entry contains most of the stat structure including s_mode, st_size



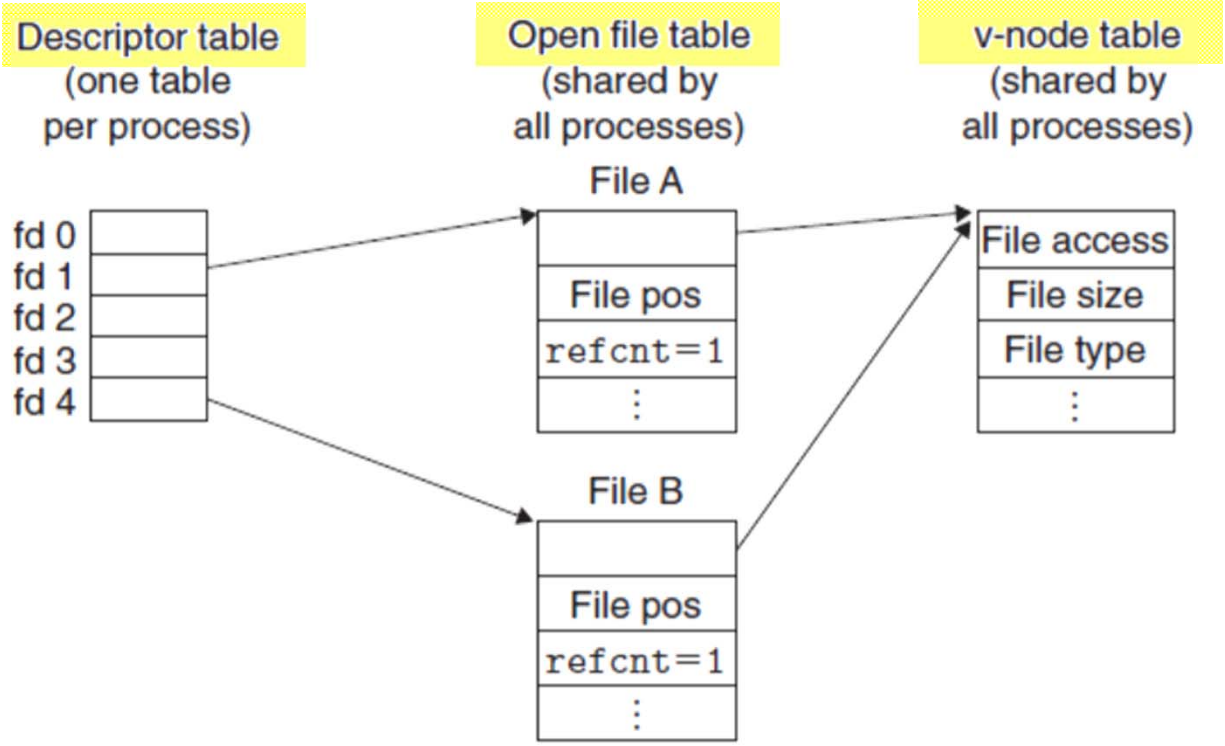
Sharing Files

- Typically, two descriptors reference distinct files



Sharing Files

- A process can open the same file twice
  - Each descriptor has its own file position



Sharing Files

- Two related functions
  - Fork
    - Creates a child process by duplicating the calling process

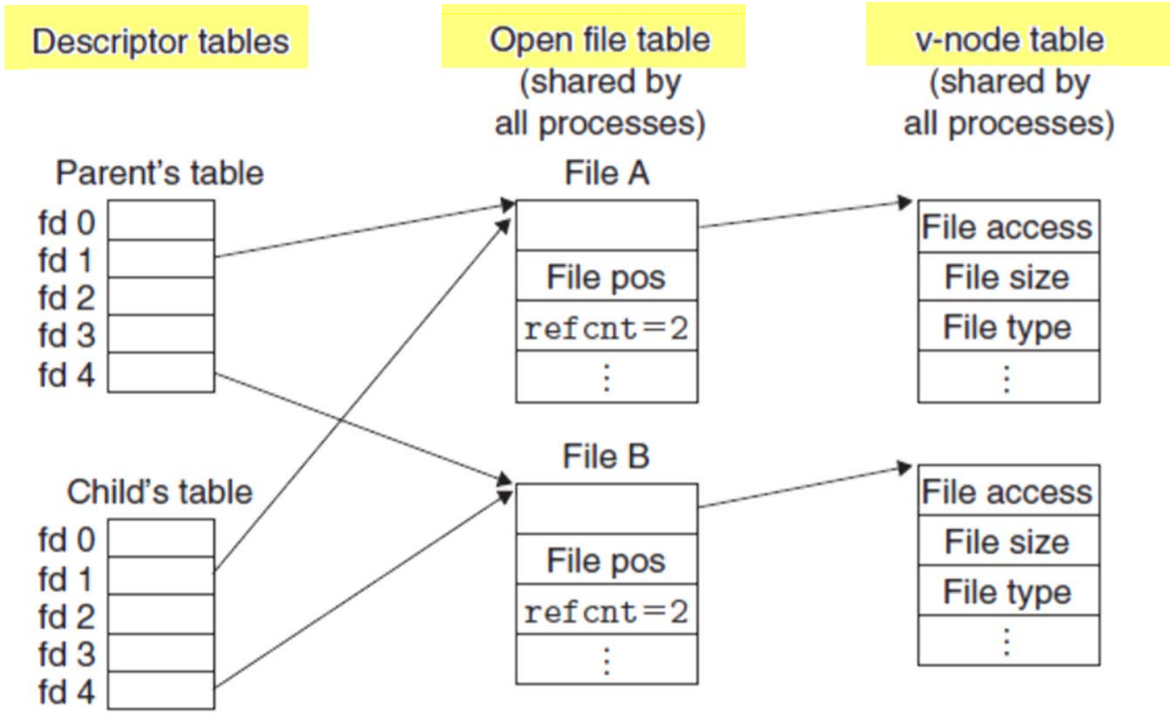
```
pid_t fork(void);
```

  - Execve
    - Executes a program by replacing the calling process with the new program

```
int execve(const char *filename,
           const char *argv[],
           const char *envp[]);
```

Sharing Files

- Open files and then call fork
  - Child has its own duplicate *copy of parent's descriptor table*
  - *Open file tables are shared* and so does the *file position*



Sharing Files

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>

void fork_then_open();
void open_then_fork();

int main(int argc, char **argv)
{
    fork_then_open();
    open_then_fork();
}
```

```c
void fork_then_open() {
    int pid = fork(); then open
    int fd = open("foo.txt", O_CREAT|O_RDWR, S_IRUSR|S_IWUSR);

    if(pid == 0/*child*/) {
        sleep(1); //wait for the parent to write first
        char buf[100] = {0};
        read(fd, buf, sizeof(buf));
        close(fd);
        printf("[%s]\n", buf);
        exit(0);
    }
    else {
        int status;
        write(fd, "1234567890", 11);
        close(fd);
        waitpid(pid, &status, 0);
    }
}
```

```c
void open_then_fork() {
    int fd = open("bar.txt", O_CREAT|O_RDWR, S_IRUSR|S_IWUSR);
    int pid = fork(); // open then fork

    if(pid == 0/*child*/) {
        sleep(1); //wait for the parent to write first
        char buf[100] = {0};
        // lseek(fd, 0, SEEK_SET);
        read(fd, buf, sizeof(buf));
        close(fd);
        printf("[%s]\n", buf);
        exit(0);
    }
    else {
        int status;
        write(fd, "1234567890", 11);
        close(fd);
        waitpid(pid, &status, 0);
    }
}
```

I/O Redirection

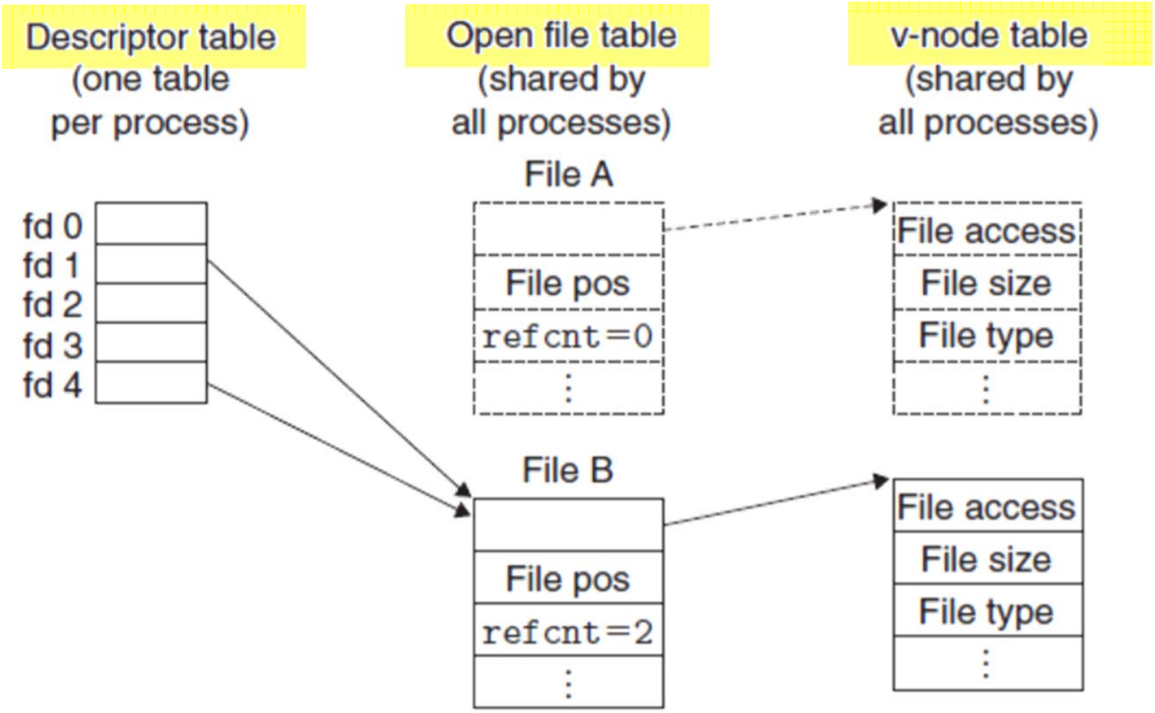
- Redirect output to a file, read input from a file
  - `$ ls > foo.txt`
  - `$ wc < foo.txt`
  - `$ ls | wc`

- dup2:
  - `int dup2(int oldfd, int newfd);`
  - Copy the descriptor entry in oldfd to the entry in newfd.
  - If newfd was already open, close newfd before copying oldfd

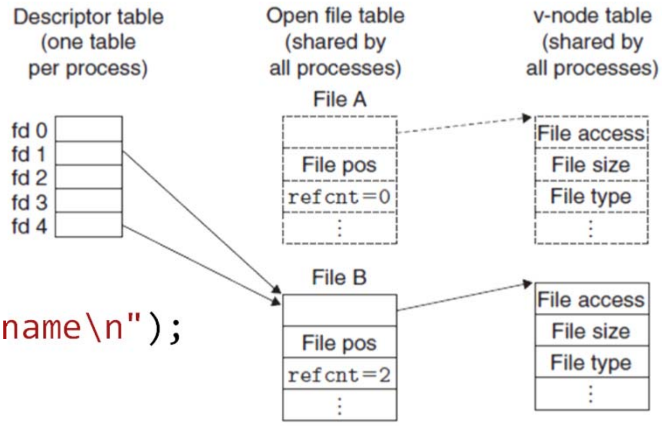
- I/O redirection:
  - Before run execve,
  - Open the file and
  - Dupe its descriptor entry to entry 1 (output) or to entry 0 (input)

I/O Redirection

- After dup2(4,1) when entry 1 was pointing to File A



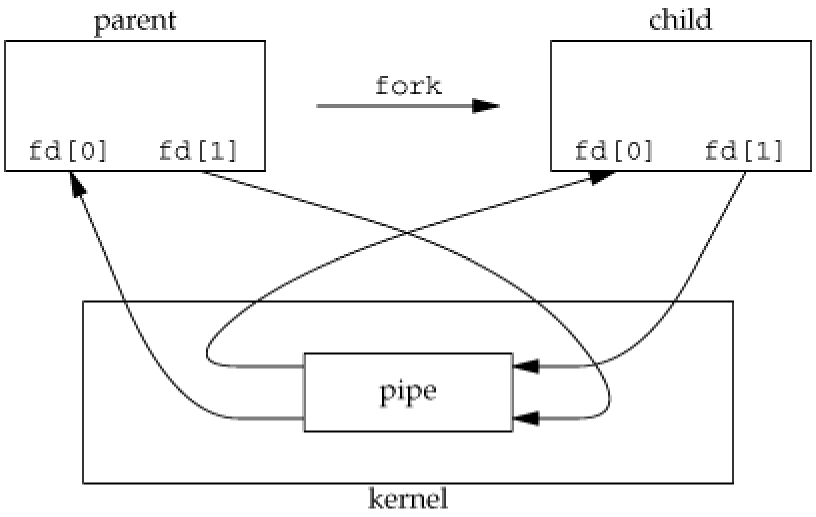
```
//redirect.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/wait.h>
int main(int argc, char **argv) {
    if (argc != 3) {
        printf("usage: a.out command output_file_name\n");
        exit(0);
    }

    int pid = fork();
    if (pid == 0/*child*/) {
        int fd = open(argv[2], O_CREAT|O_TRUNC|O_WRONLY, S_IRUSR|S_IWUSR);
        dup2(fd/*src*/, 1/*dst*/);
        char *param[2] = { argv[1], NULL };
        execvp(param[0], param);
    }
    else {
        int status;
        waitpid(pid, &status, 0);
        printf("child exit status: %d\n\n", status);

        int fd = open(argv[2], O_CREAT|O_RDONLY, 0);
        dup2(fd/*src*/, 0/*dst*/);
        char *param[2] = { "/bin/cat", NULL };
        execvp(param[0], param);
    }
}
```



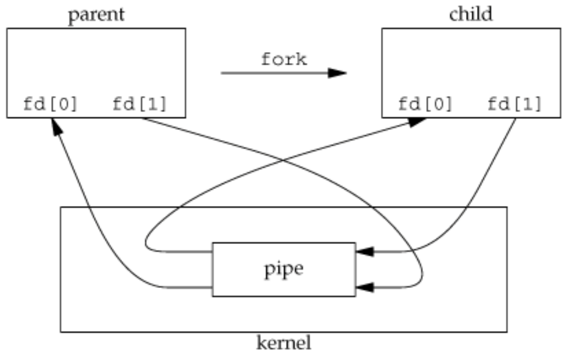
Pipe

- Unix Inter Process Communication mechanism

```
#include <unistd.h>
int pipe(int fd[2]);
```



```c
#include <unistd.h>

int main() {
    int fd[2];
    pipe(fd);

    pid_t pid = fork();
    if(pid == 0/*child*/) {
        close(fd[0]);
        dup2(fd[1], 1);
        char *param[] = {"/bin/ls", NULL};
        execve(param[0], param, NULL);
    }
    else {
        close(fd[1]);
        dup2(fd[0], 0);
        waitpid(pid, NULL, 0);
        char *param[] = {"/usr/bin/wc", NULL};
        execve(param[0], param, NULL);
    }
    return 0;
}
```



Programming Assignment 6

- Download myls.zip from the course webpage and implement the TODOs
  - This program should produce the same result as a unix command ls -al
  - Compare the results for different directories like
    - ls -al .
    - ls -al /dev
    - ls -al /etc
- Due date: 10/25/2022

```
static void print_dir(char *path) {
    //TODO: open directory

    //count entries to allocate names
    int nr_names = 0;
    struct dirent *dep;
    while((dep = readdir(dp)) != NULL)
        nr_names++;
    char **names = malloc(sizeof(char*) * nr_names);

    //update width and dupe names
    field_info_t fi;
    memset(&fi, 0, sizeof(struct field_info));
    seekdir(dp, 0);

    //TODO: for each entries the directory
    //         - update the entry in names array
    //         - update the field info of fi
    //hint: use readdir, strdup, and update_field_info

    //TODO: close directory

    //sort names
    sort_names(names, nr_names);

    //TODO: print total and file information
    //hint: use print_file with path, name, and fi
}
```

```
typedef struct field_info {
    //width
    int w_nlink;
    int w_user;
    int w_group;
    int w_size;
    int w_date;
    //block size
    long nr_blocks;
} field_info_t;

static void update_field_info(char *path, char *name,
                              field_info_t *pfi) {
    char buf[MAXPATH];
    char fullpath[MAXPATH];
    sprintf(fullpath, "%s/%s", path, name);

    struct stat sb;
    //TODO: get file information using lstat

    //TODO: update the width fields of pfi
    //   the width should be the max of the current width and
    //   the number of printed chars to buf
    //   use print_* with w param set to 0 to compute the width
    //   print_*(char *buf, ...) function prints to buf and returns
    //   the number of chars printed excluding '\0'.

    //TODO: if file type is directory or regular file, add half of
    //      the allocated block counts to nr_blocks of pfi
}
```

```c
static void print_file(char *path, char *name, field_info_t *pfi) {
    char buf[MAXPATH];
    char fullpath[MAXPATH];
    sprintf(fullpath, "%s/%s", path, name);

    struct stat sb;
    //TODO: get file information using lstat

    //TODO: print the following fields to buf using print_* functions
    //   print_*(char *buf, ...) function prints to buf and returns
    //   the number of chars printed excluding '\0'.

    //file type

    //permissions

    //link count

    //user

    //group

    //size

    //date

    //file name
}
```

```c
static int max_width(int a, int b) {
    return a > b ? a : b;
}

static void sort_names(char **names, int nr_names) {
    //TODO: sort names in the increasing order
    //   names: array of names
    //   nr_names: number of names

}

static int print_type(char *buf, mode_t mode) {
    //TODO: print the file type (b, c, d, p, l, -, s) to buf
    //         b: block device      c: char device
    //         d: director          p: fifo
    //         l: link              -: regular file
    //         s: socket
    //hint: use sprintf

}

static int print_permission(char *buf, mode_t mode) {
    //TODO: print the r, w, x permissions of the user, group, and other
    //hint: for each character, use sprintf at buf++ location

    return 9;
}
```

```
static int print_nlink(char *buf, int w, nlink_t nlink) {
    //TODO: print link count to buf
    //hint: use sprintf with "%*ld" and w for the width

}

static int print_user(char *buf, int w, uid_t uid) {
    //TODO: print user name to buf
    //hint: use getpwuid to get passwd struct
    //hint: use sprintf with "%*s" and -w for the width

}

static int print_group(char *buf, int w, gid_t gid) {
    //TODO: print user name to buf
    //hint: use getgrgid to get group struct
    //hint: use sprintf with "%*s" and -w for the width

}

static int print_size(char *buf, int w, off_t size) {
    //TODO: print the file size to buf
    //hint: use sprintf with "%*ld" and w for the width
    return sprintf(buf, "%*ld", w, size);
}
```

```c
static int print_date(char *buf, int w, time_t sec) {
    static char *mon[] = { "Jan", "Feb", "Mar", "Apr",
                           "May", "Jun", "Jul", "Aug",
                           "Sep", "Oct", "Nov", "Dec" };
    //print the date of the file

    //TODO: get this_year using time and localtime functions

    //TODO: get tm struct using localtime and sec

    //TODO: print date and return the number of chars printed
    //   if the year of the file is not this_year, print month, day,
    //      and year of the file
    //      hint: width of the year is w <= 11? 4: 5
    //   otherwise, print month, day, hour, and min of the file

}

static int print_name(char *buf, mode_t mode, char *fullpath, char *name) {
    //TODO: print the name to buf and return the number of chars printed
    //   if the file type is link, print name -> link
    //      hint: use readlink and null terminate the buffer
    //   otherwise, print the name

}
```

```
$ ./a.out /dev
total 4
drwxr-xr-x 16 root root    3240 Sep 21 15:10 .
drwxr-xr-x 19 root root    4096 Sep  6 08:40 ..
crw-r--r--  1 root root       0 Sep  6 08:40 autofs
drwxr-xr-x  2 root root     320 Sep 21 15:10 block
crw-rw----  1 root disk       0 Sep  6 08:44 btrfs-control
drwxr-xr-x  2 root root    2580 Sep  6 08:45 char
crw--w----  1 root tty        0 Sep  6 08:41 console
lrwxrwxrwx  1 root root      11 Sep  6 08:40 core -> /proc/kcore
drwxr-xr-x  3 root root      60 Sep  6 08:45 cpu
crw-------  1 root root       0 Sep  6 08:40 cpu_dma_latency
crw-------  1 root root       0 Sep  6 08:40 cuse
drwxr-xr-x  5 root root     100 Sep  6 08:40 disk
crw-------  1 root root       0 Sep  6 08:40 ecryptfs
lrwxrwxrwx  1 root root      13 Sep  6 08:40 fd -> /proc/self/fd
crw-rw-rw-  1 root root       0 Sep  6 08:40 full
crw-rw-rw-  1 root root       0 Sep  6 08:47 fuse
crw-------  1 root root       0 Sep  6 08:40 hpet
drwxr-xr-x  2 root root       0 Sep  6 08:40 hugepages
crw-------  1 root root       0 Sep  6 08:40 hwrng
lrwxrwxrwx  1 root root      12 Sep  6 08:40 initctl -> /run/initctl
…
```