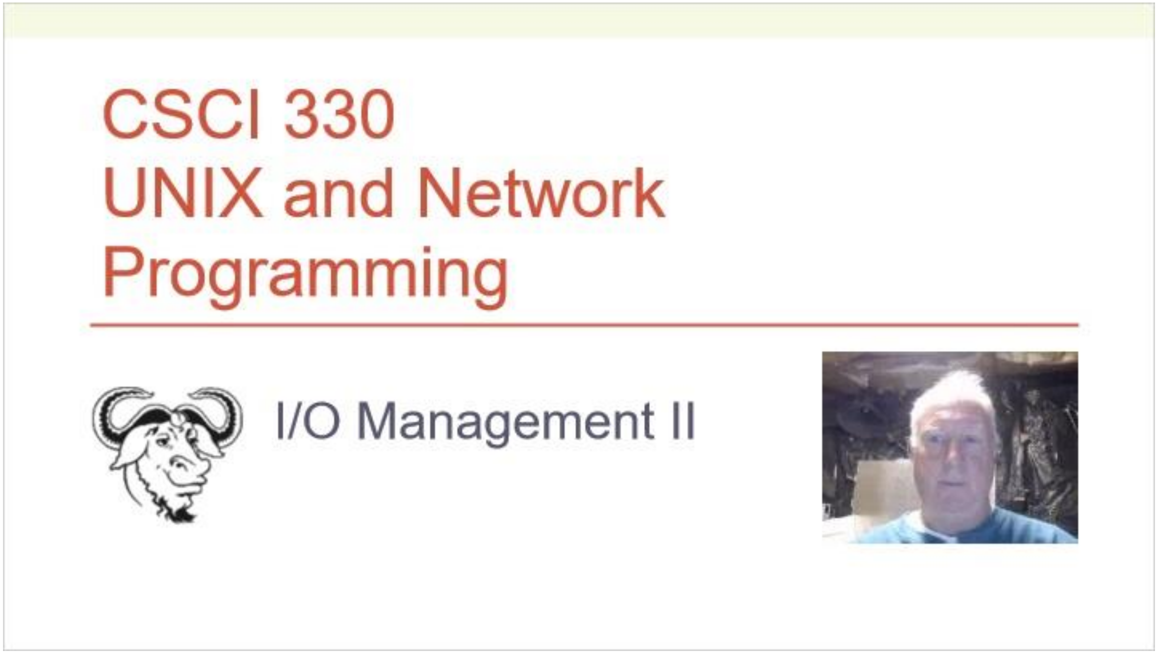
**Input & Output, Part 2**

## 1. IO Managementt II

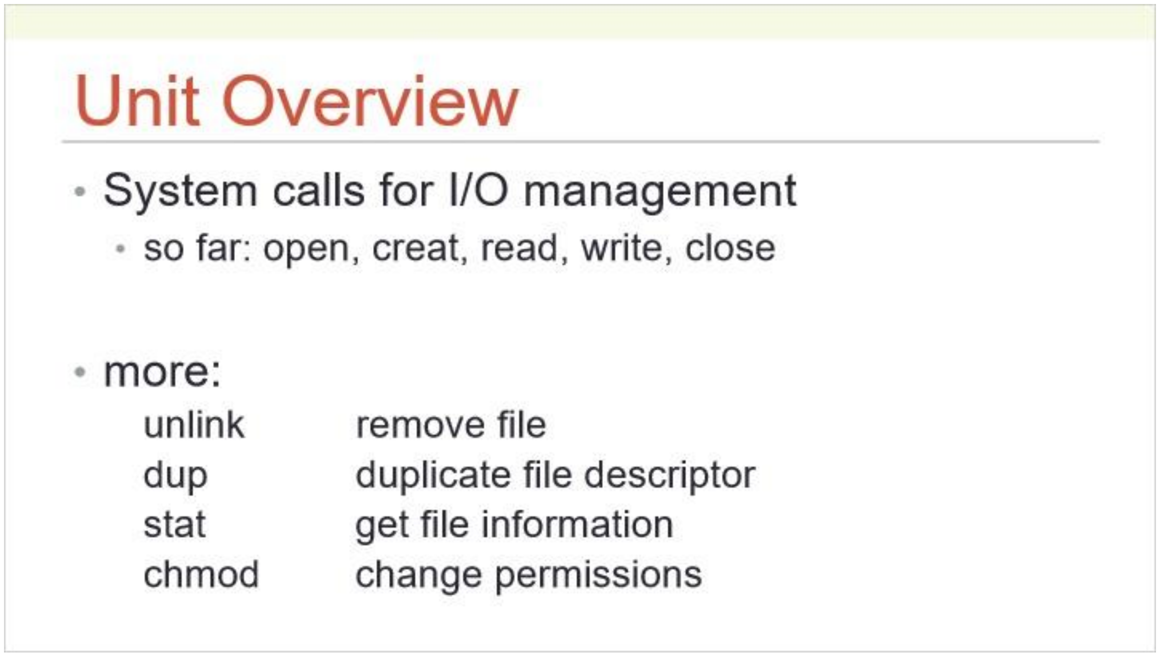
### *1.1 CSCI 330*



CSCI 330
UNIX and Network Programming

I/O Management II

*1.2 Unit Overview*

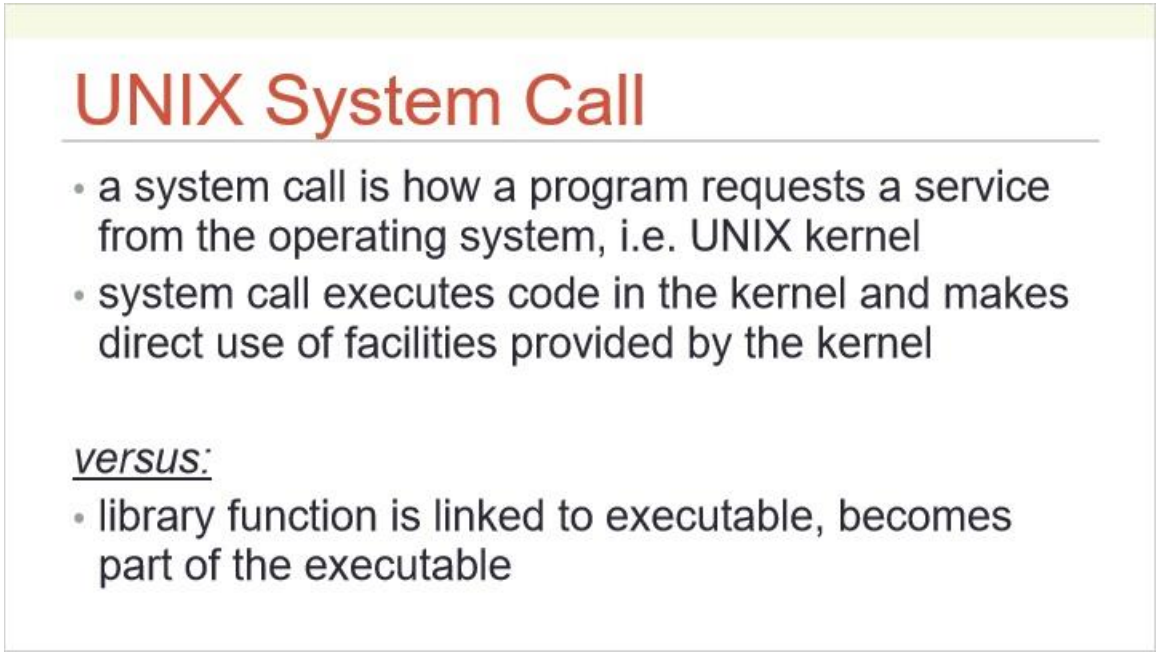


## Unit Overview

- System calls for I/O management
  - so far: open, creat, read, write, close
- more:

| | |
|---|---|
| unlink | remove file |
| dup | duplicate file descriptor |
| stat | get file information |
| chmod | change permissions |

*1.3 UNIX System Call*



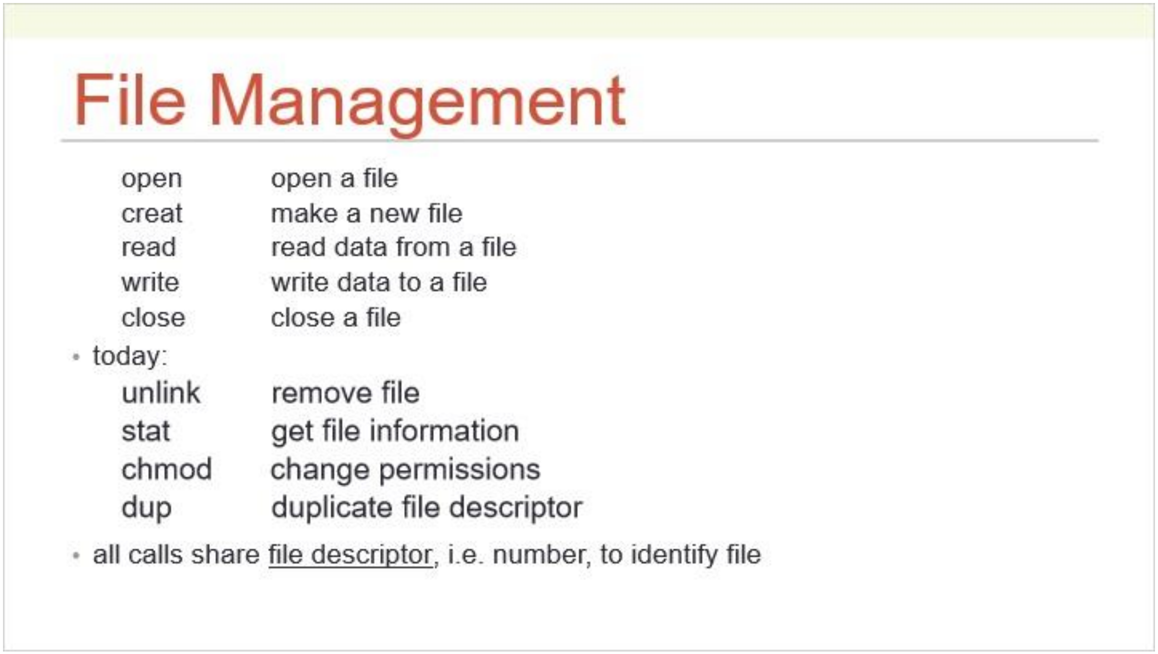
## UNIX System Call

- a system call is how a program requests a service from the operating system, i.e. UNIX kernel
- system call executes code in the kernel and makes direct use of facilities provided by the kernel

versus:

- library function is linked to executable, becomes part of the executable

## *1.4 File Management*



## File Management

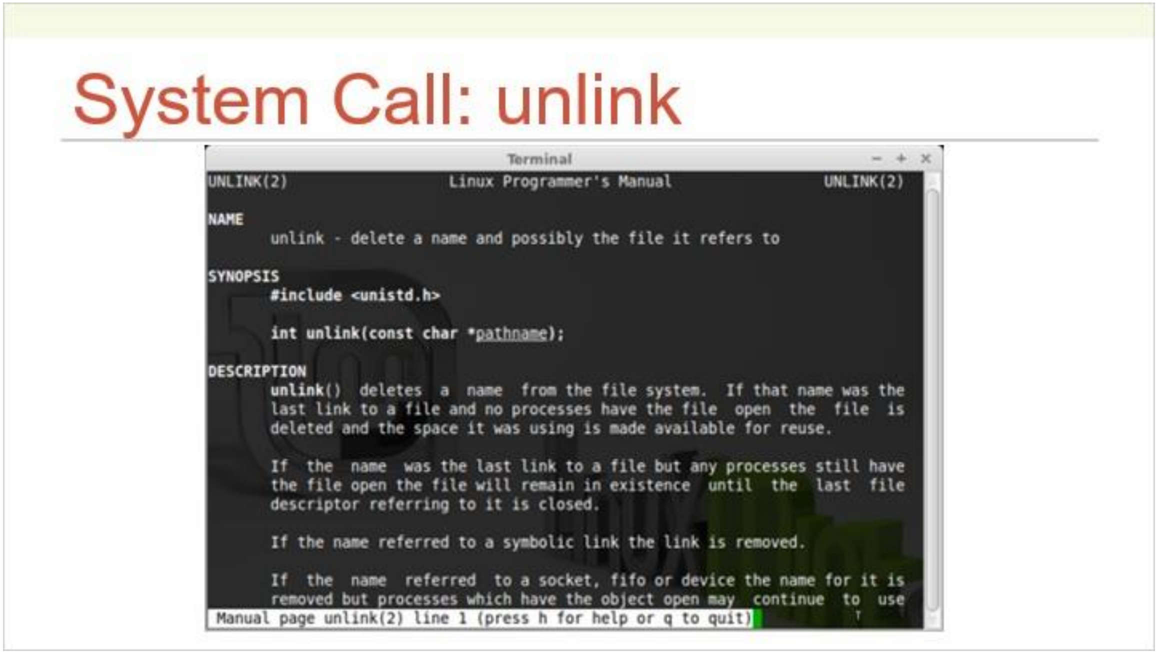
| | |
|---|---|
| open | open a file |
| creat | make a new file |
| read | read data from a file |
| write | write data to a file |
| close | close a file |

- today:

| | |
|---|---|
| unlink | remove file |
| stat | get file information |
| chmod | change permissions |
| dup | duplicate file descriptor |

- all calls share file descriptor, i.e. number, to identify file

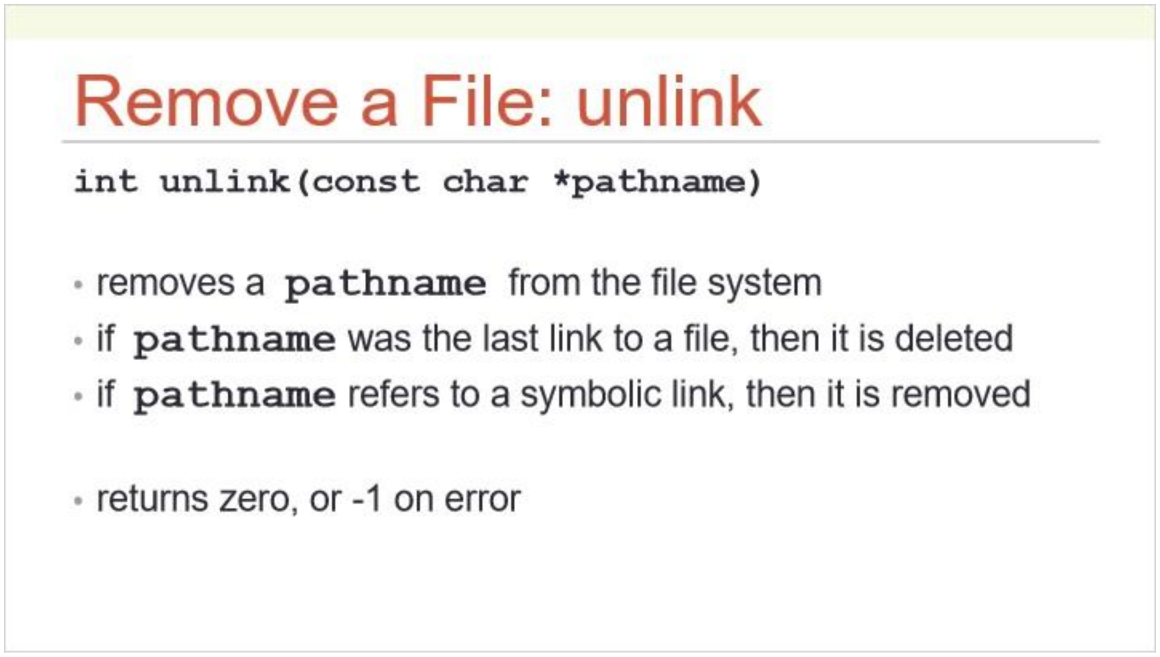
## *1.5 System Call: unlink*



## System Call: unlink

```
UNLINK(2)                  Linux Programmer's Manual                 UNLINK(2)

NAME
       unlink - delete a name and possibly the file it refers to

SYNOPSIS
       #include <unistd.h>

       int unlink(const char *pathname);

DESCRIPTION
       unlink()  deletes  a  name  from the file system.  If that name was the
       last link to a file and no processes have the file  open  the  file  is
       deleted and the space it was using is made available for reuse.

       If  the  name  was the last link to a file but any processes still have
       the file open the file will remain in existence  until  the  last  file
       descriptor referring to it is closed.

       If the name referred to a symbolic link the link is removed.

       If  the  name  referred  to a socket, fifo or device the name for it is
       removed but processes which have the object open may  continue  to  use
 Manual page unlink(2) line 1 (press h for help or q to quit)
```

## *1.6 Remove a File: unlink*



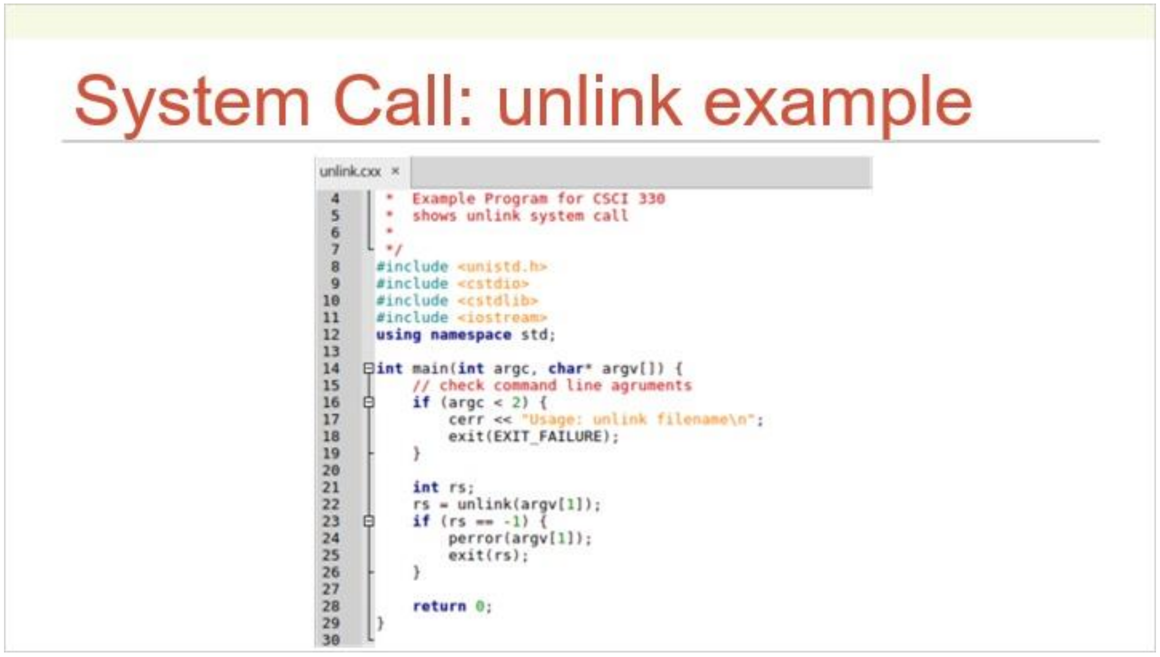
### Remove a File: unlink

```
int unlink(const char *pathname)
```

- removes a **pathname** from the file system
- if **pathname** was the last link to a file, then it is deleted
- if **pathname** refers to a symbolic link, then it is removed

- returns zero, or -1 on error

## *1.7 System Call: unlink example*

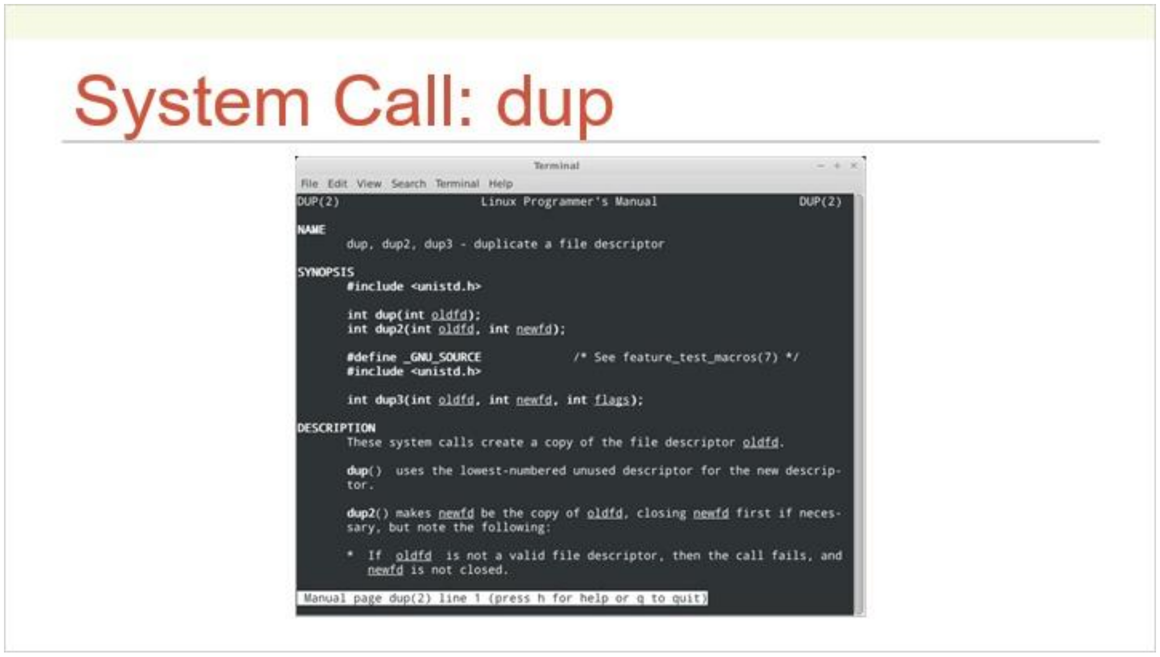


### System Call: unlink example

unlink.cxx ×

```
 *   Example Program for CSCI 330
 *   shows unlink system call
 *
 */
#include <unistd.h>
#include <cstdio>
#include <cstdlib>
#include <iostream>
using namespace std;

int main(int argc, char* argv[]) {
    // check command line agruments
    if (argc < 2) {
        cerr << "Usage: unlink filename\n";
        exit(EXIT_FAILURE);
    }

    int rs;
    rs = unlink(argv[1]);
    if (rs == -1) {
        perror(argv[1]);
        exit(rs);
    }

    return 0;
}
```

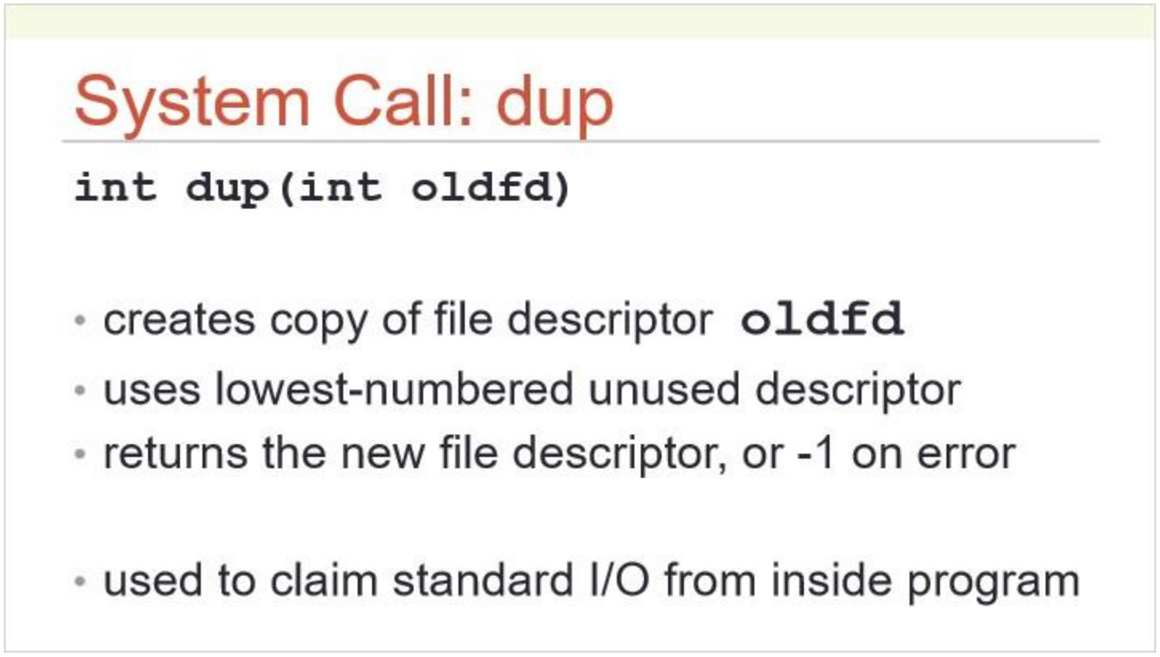
### *1.8 System Call: dup*



## System Call: dup

```
DUP(2)                    Linux Programmer's Manual                   DUP(2)

NAME
       dup, dup2, dup3 - duplicate a file descriptor

SYNOPSIS
       #include <unistd.h>

       int dup(int oldfd);
       int dup2(int oldfd, int newfd);

       #define _GNU_SOURCE             /* See feature_test_macros(7) */
       #include <unistd.h>

       int dup3(int oldfd, int newfd, int flags);

DESCRIPTION
       These system calls create a copy of the file descriptor oldfd.

       dup()  uses the lowest-numbered unused descriptor for the new descrip-
       tor.

       dup2() makes newfd be the copy of oldfd, closing newfd first if neces-
       sary, but note the following:

       *  If  oldfd  is not a valid file descriptor, then the call fails, and
          newfd is not closed.

 Manual page dup(2) line 1 (press h for help or q to quit)
```

### *1.9 System Call: dup*



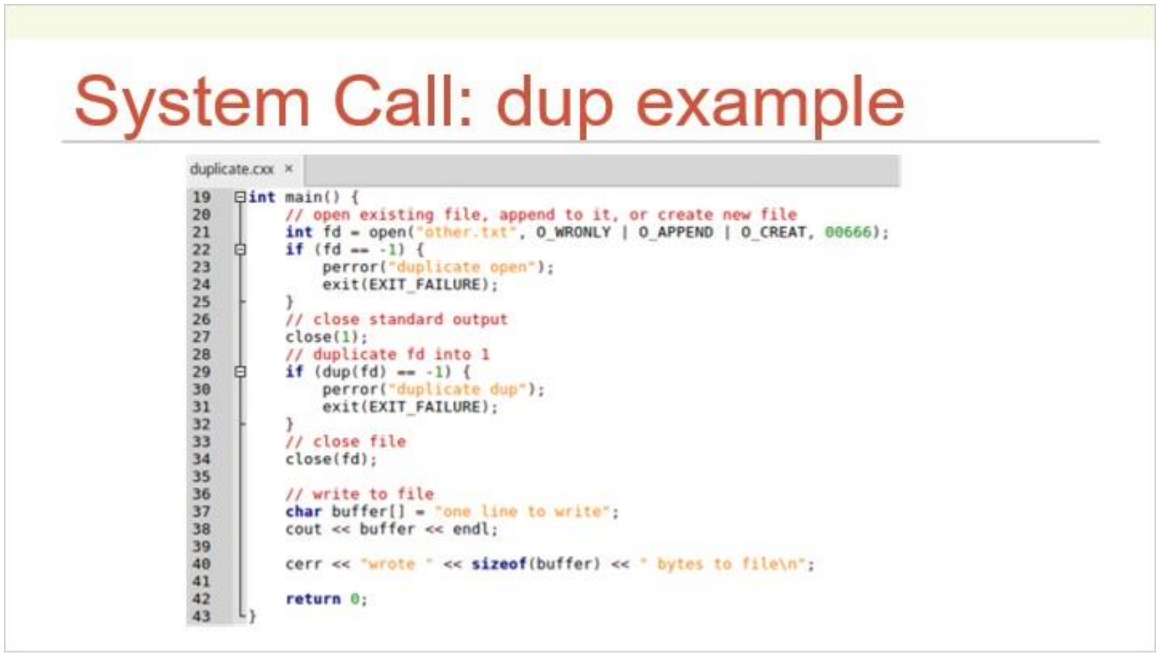
## System Call: dup

`int dup(int oldfd)`

- creates copy of file descriptor **`oldfd`**
- uses lowest-numbered unused descriptor
- returns the new file descriptor, or -1 on error

- used to claim standard I/O from inside program

## *1.10 System Call: dup example*

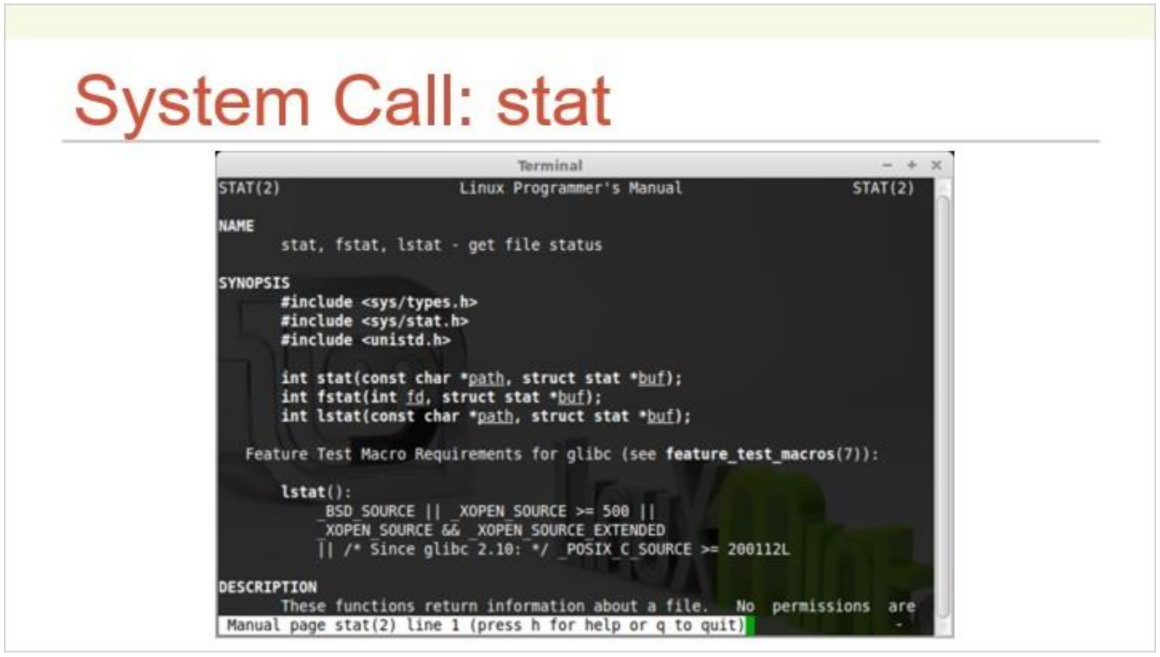


### System Call: dup example

duplicate.cxx ×

```
int main() {
    // open existing file, append to it, or create new file
    int fd = open("other.txt", O_WRONLY | O_APPEND | O_CREAT, 00666);
    if (fd == -1) {
        perror("duplicate open");
        exit(EXIT_FAILURE);
    }
    // close standard output
    close(1);
    // duplicate fd into 1
    if (dup(fd) == -1) {
        perror("duplicate dup");
        exit(EXIT_FAILURE);
    }
    // close file
    close(fd);

    // write to file
    char buffer[] = "one line to write";
    cout << buffer << endl;

    cerr << "wrote " << sizeof(buffer) << " bytes to file\n";

    return 0;
}
```

## *1.11 System Call: stat*

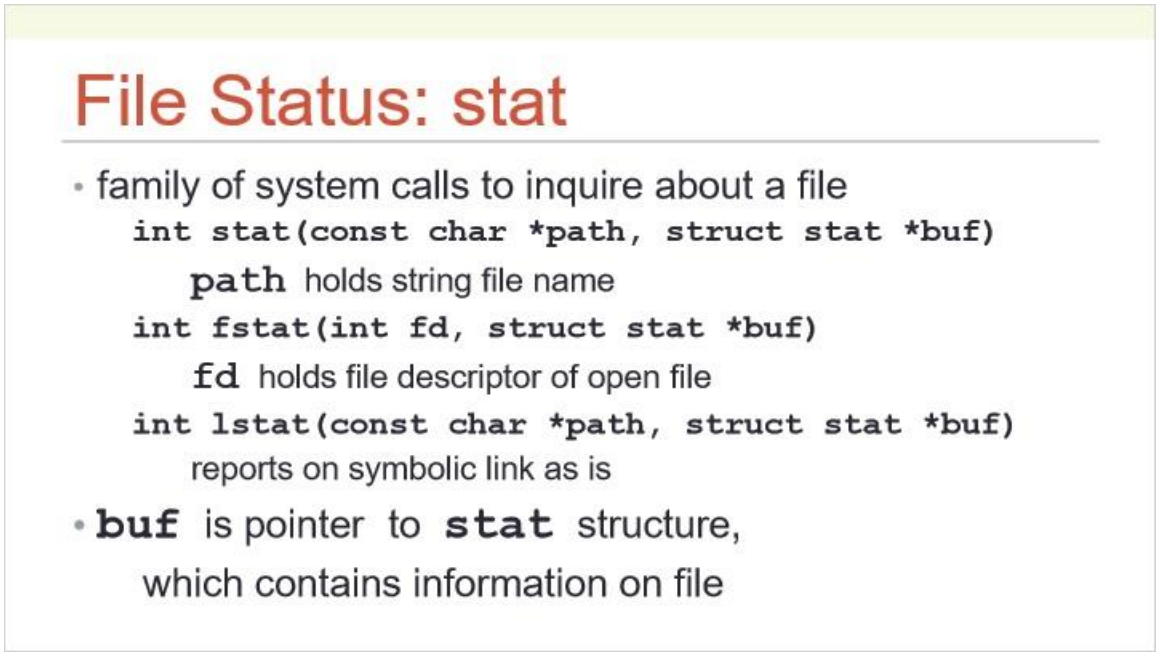


### System Call: stat

Terminal

```
STAT(2)                    Linux Programmer's Manual                    STAT(2)

NAME
       stat, fstat, lstat - get file status

SYNOPSIS
       #include <sys/types.h>
       #include <sys/stat.h>
       #include <unistd.h>

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

   Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

       lstat():
           _BSD_SOURCE || _XOPEN_SOURCE >= 500 ||
           _XOPEN_SOURCE && _XOPEN_SOURCE_EXTENDED
           || /* Since glibc 2.10: */ _POSIX_C_SOURCE >= 200112L

DESCRIPTION
       These functions return information about a file.   No  permissions  are
 Manual page stat(2) line 1 (press h for help or q to quit)
```

## *1.12 File Status: stat*



### File Status: stat

- family of system calls to inquire about a file

  `int stat(const char *path, struct stat *buf)`

  **`path`** holds string file name

  `int fstat(int fd, struct stat *buf)`

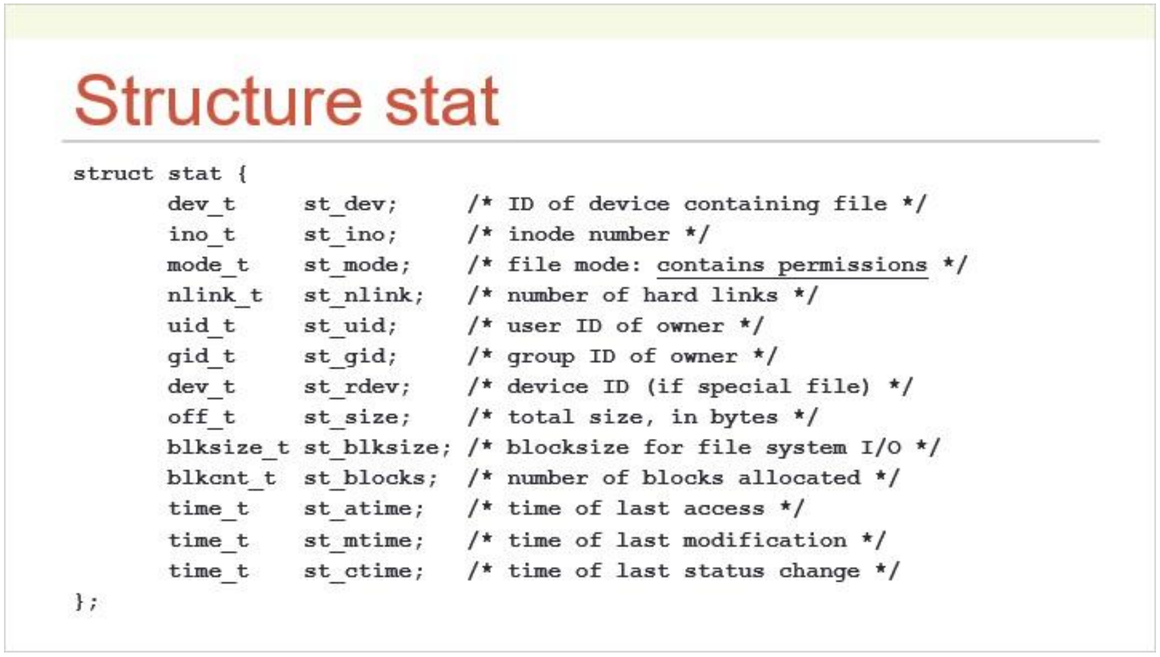
  **`fd`** holds file descriptor of open file

  `int lstat(const char *path, struct stat *buf)`

  reports on symbolic link as is

- **`buf`** is pointer to **`stat`** structure, which contains information on file

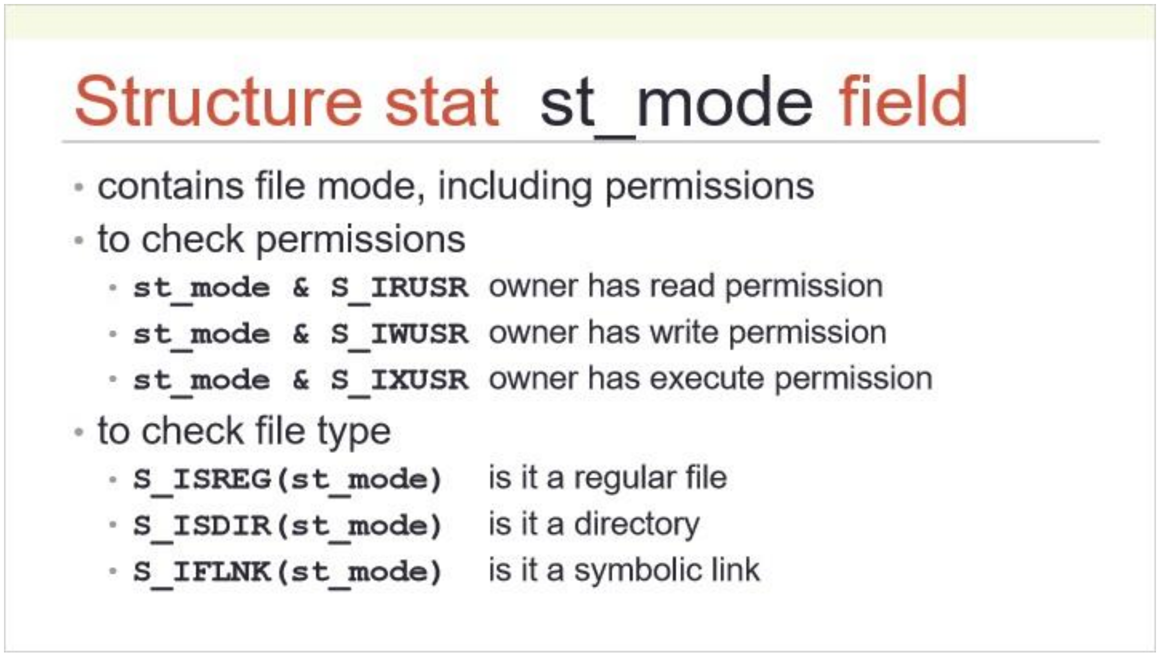
## *1.13 Structure stat*



### Structure stat

```
struct stat {
       dev_t     st_dev;     /* ID of device containing file */
       ino_t     st_ino;     /* inode number */
       mode_t    st_mode;    /* file mode: contains permissions */
       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 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 */
};
```

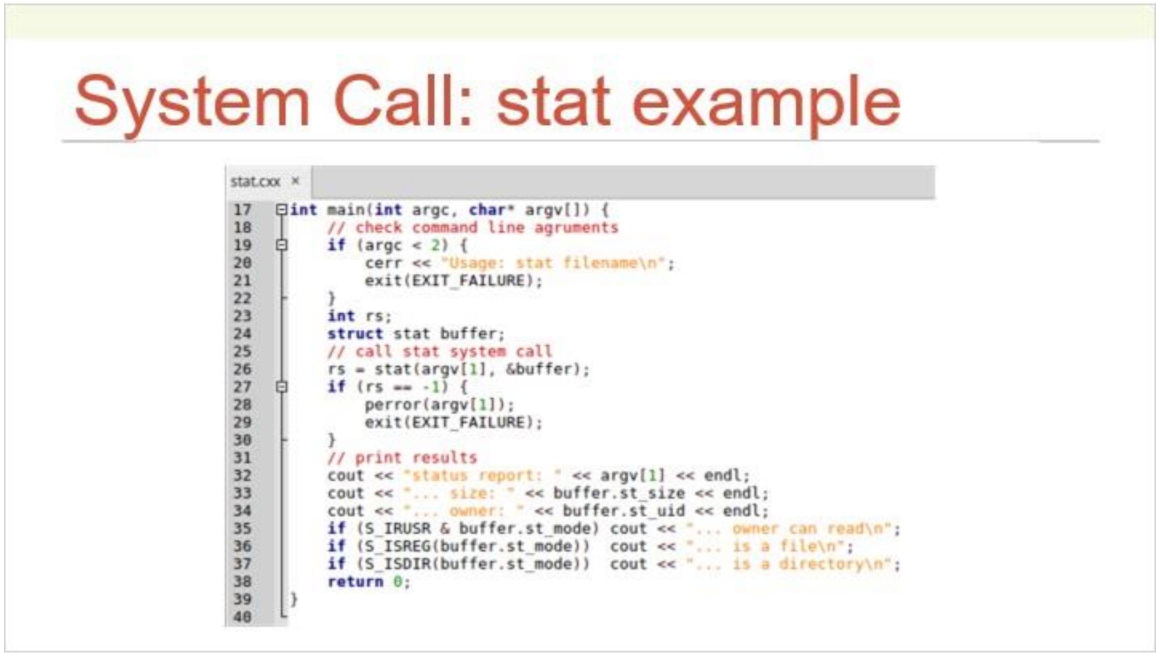
## *1.14 Structure stat  st_mode  field*



### Structure stat  st_mode field

- contains file mode, including permissions
- to check permissions
  - `st_mode & S_IRUSR` owner has read permission
  - `st_mode & S_IWUSR` owner has write permission
  - `st_mode & S_IXUSR` owner has execute permission
- to check file type
  - `S_ISREG(st_mode)` is it a regular file
  - `S_ISDIR(st_mode)` is it a directory
  - `S_IFLNK(st_mode)` is it a symbolic link

## *1.15 System Call: stat example*

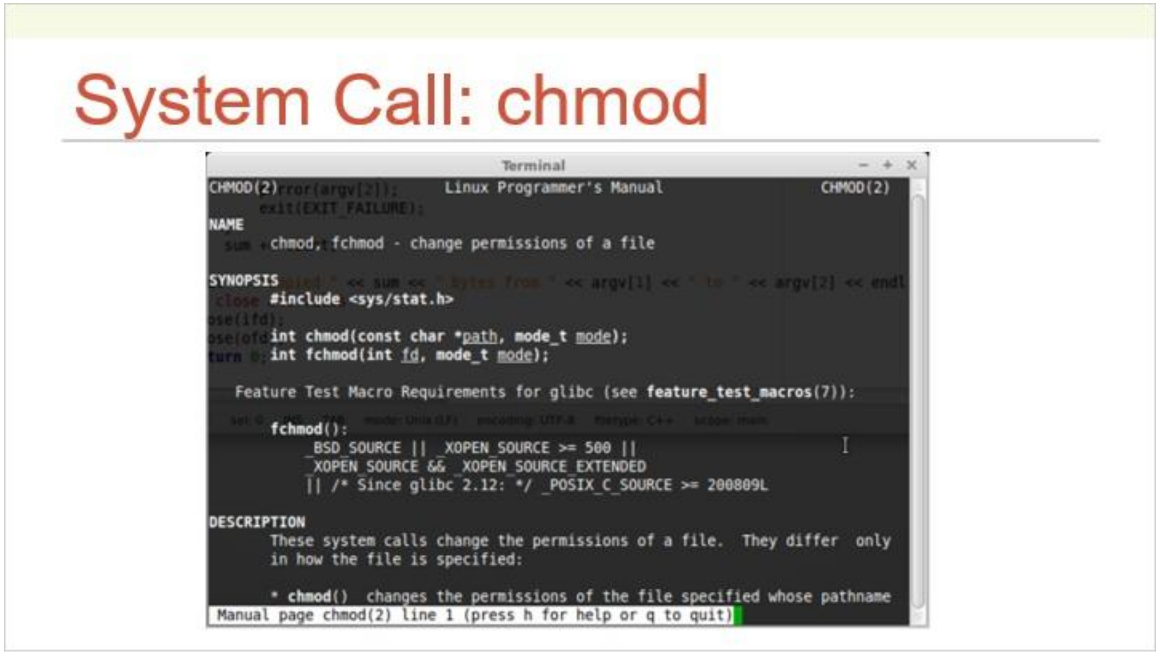


### System Call: stat example

stat.cxx

```
int main(int argc, char* argv[]) {
    // check command line agruments
    if (argc < 2) {
        cerr << "Usage: stat filename\n";
        exit(EXIT_FAILURE);
    }
    int rs;
    struct stat buffer;
    // call stat system call
    rs = stat(argv[1], &buffer);
    if (rs == -1) {
        perror(argv[1]);
        exit(EXIT_FAILURE);
    }
    // print results
    cout << "status report: " << argv[1] << endl;
    cout << "... size: " << buffer.st_size << endl;
    cout << "... owner: " << buffer.st_uid << endl;
    if (S_IRUSR & buffer.st_mode) cout << "... owner can read\n";
    if (S_ISREG(buffer.st_mode))  cout << "... is a file\n";
    if (S_ISDIR(buffer.st_mode))  cout << "... is a directory\n";
    return 0;
}
```

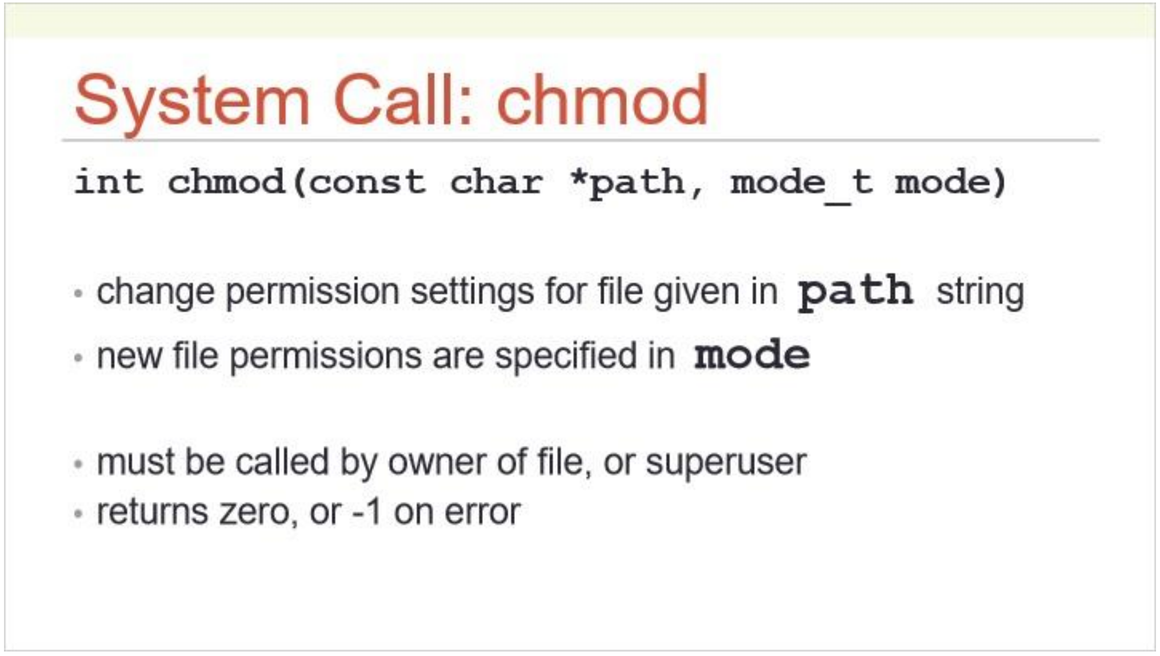
## *1.16 System Call: chmod*



### System Call: chmod

```
CHMOD(2)                 Linux Programmer's Manual                 CHMOD(2)

NAME
       chmod, fchmod - change permissions of a file

SYNOPSIS
       #include <sys/stat.h>

       int chmod(const char *path, mode_t mode);
       int fchmod(int fd, mode_t mode);

   Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

       fchmod():
           _BSD_SOURCE || _XOPEN_SOURCE >= 500 ||
           _XOPEN_SOURCE && _XOPEN_SOURCE_EXTENDED
           || /* Since glibc 2.12: */ _POSIX_C_SOURCE >= 200809L

DESCRIPTION
       These system calls change the permissions of a file.  They differ  only
       in how the file is specified:

       * chmod()  changes the permissions of the file specified whose pathname
 Manual page chmod(2) line 1 (press h for help or q to quit)
```

## *1.17 System Call: chmod*



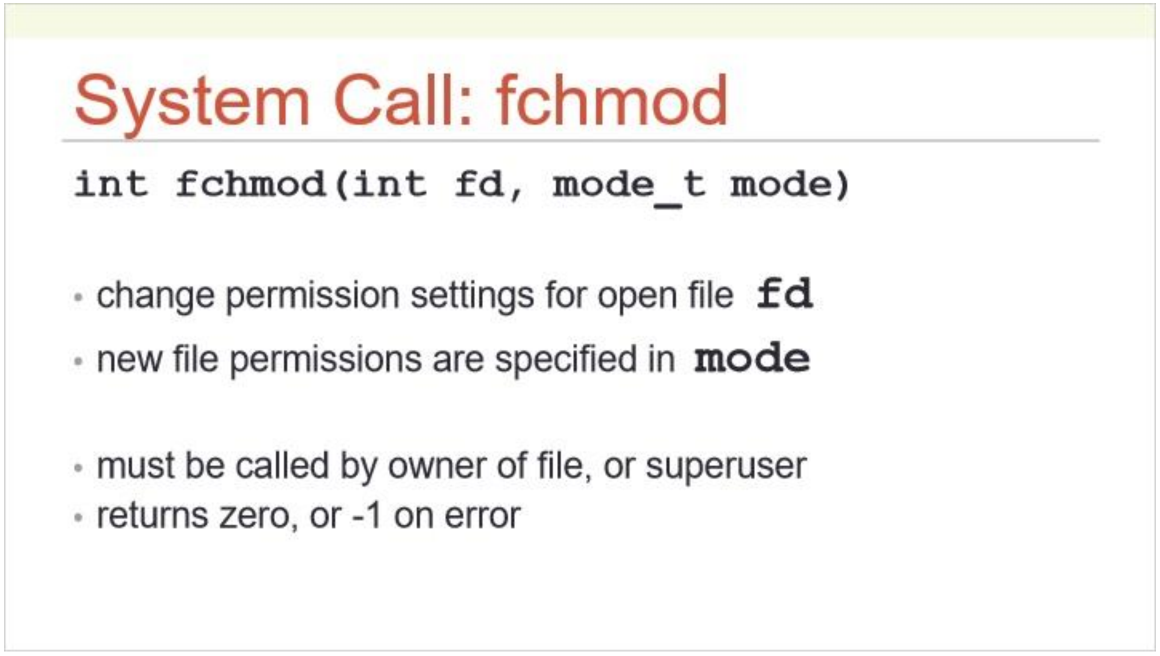
### System Call: chmod

`int chmod(const char *path, mode_t mode)`

- change permission settings for file given in `path` string
- new file permissions are specified in `mode`

- must be called by owner of file, or superuser
- returns zero, or -1 on error

### *1.18 System Call: fchmod*



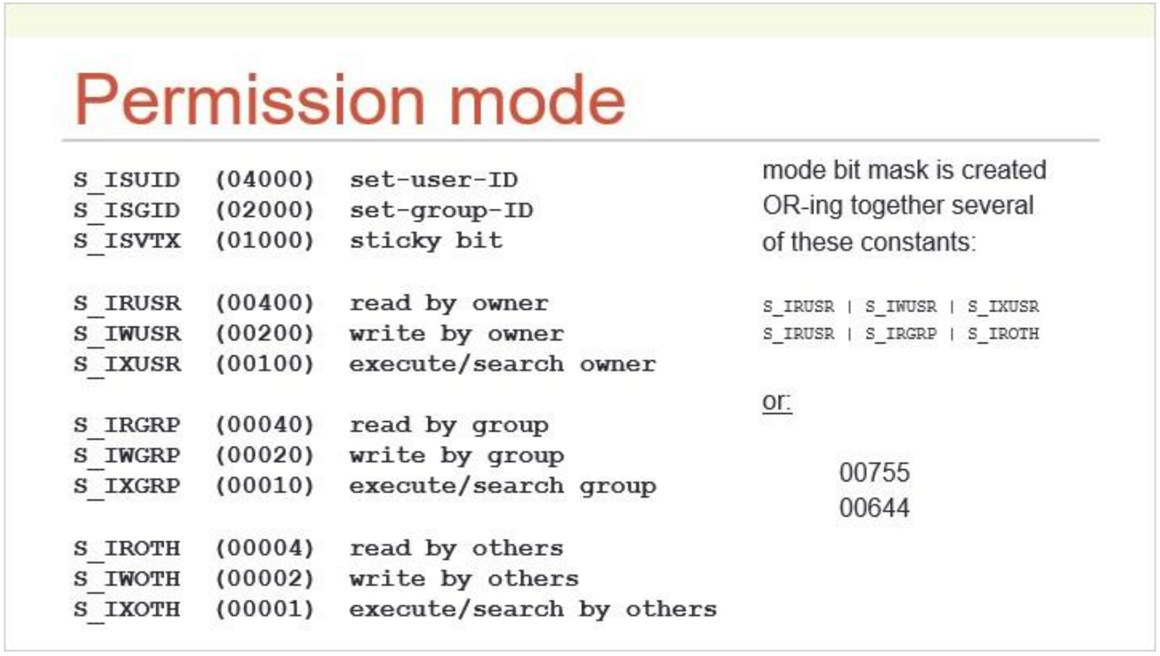
## System Call: fchmod

```
int fchmod(int fd, mode_t mode)
```

- change permission settings for open file **`fd`**
- new file permissions are specified in **`mode`**

- must be called by owner of file, or superuser
- returns zero, or -1 on error

### *1.19 Permission mode*



## Permission mode

```
S_ISUID   (04000)   set-user-ID
S_ISGID   (02000)   set-group-ID
S_ISVTX   (01000)   sticky bit

S_IRUSR   (00400)   read by owner
S_IWUSR   (00200)   write by owner
S_IXUSR   (00100)   execute/search owner

S_IRGRP   (00040)   read by group
S_IWGRP   (00020)   write by group
S_IXGRP   (00010)   execute/search group

S_IROTH   (00004)   read by others
S_IWOTH   (00002)   write by others
S_IXOTH   (00001)   execute/search by others
```

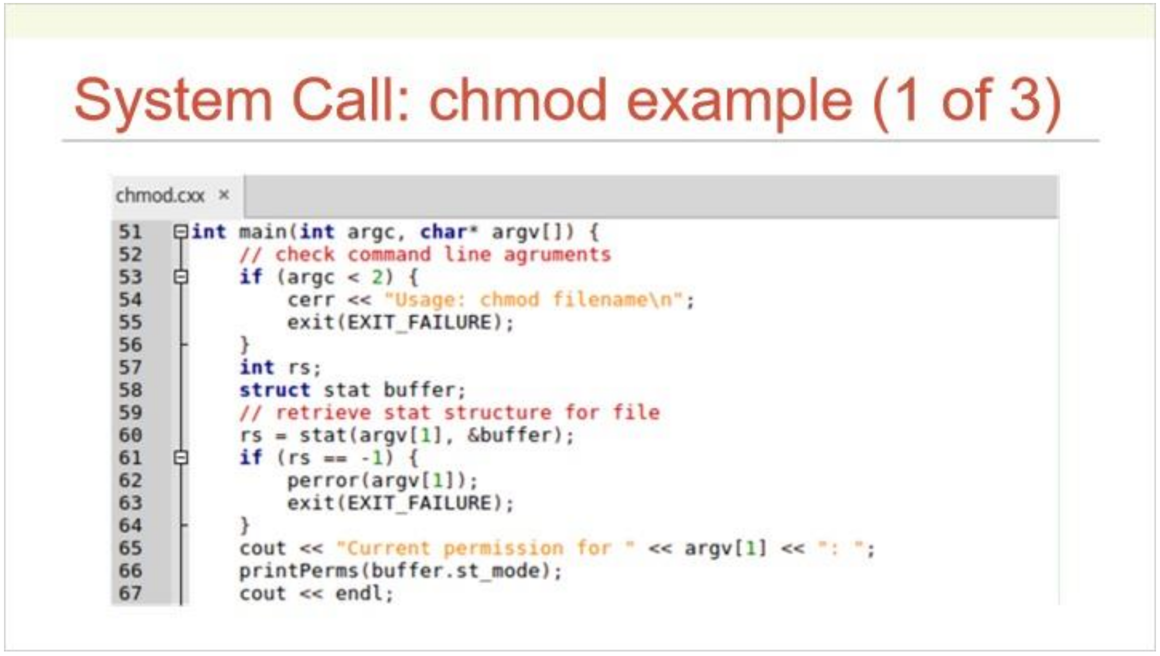
mode bit mask is created OR-ing together several of these constants:

```
S_IRUSR | S_IWUSR | S_IXUSR
S_IRUSR | S_IRGRP | S_IROTH
```

or:

00755
00644

## *1.20 chmod example (1 of 3)*

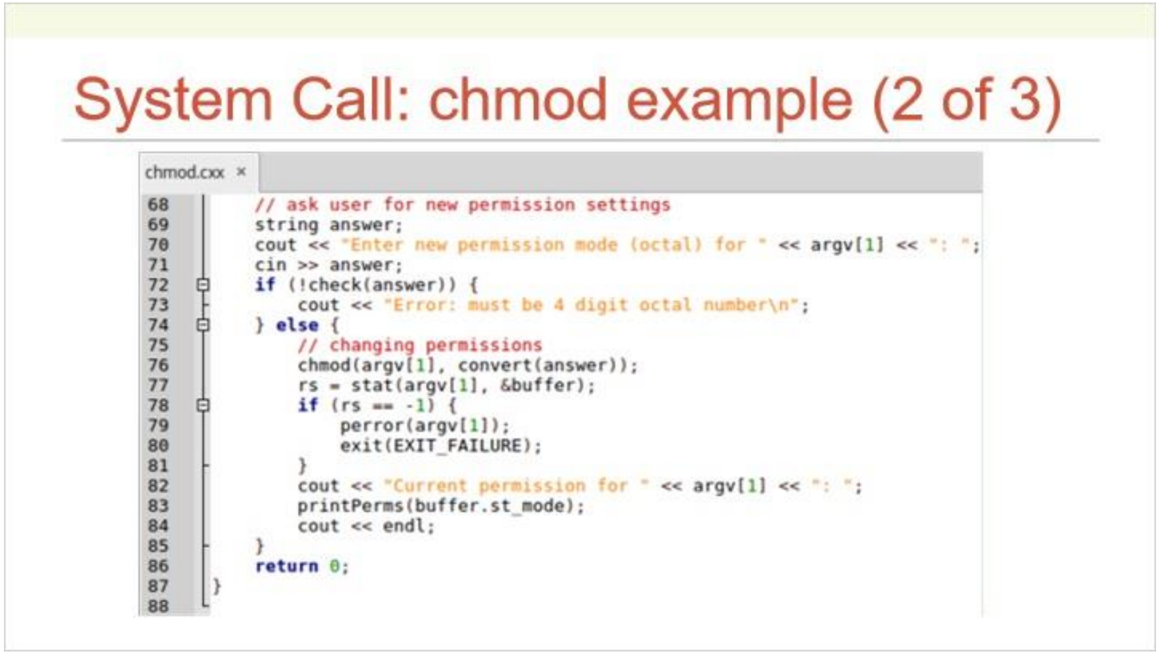


System Call: chmod example (1 of 3)

chmod.cxx ×

```
int main(int argc, char* argv[]) {
    // check command line agruments
    if (argc < 2) {
        cerr << "Usage: chmod filename\n";
        exit(EXIT_FAILURE);
    }
    int rs;
    struct stat buffer;
    // retrieve stat structure for file
    rs = stat(argv[1], &buffer);
    if (rs == -1) {
        perror(argv[1]);
        exit(EXIT_FAILURE);
    }
    cout << "Current permission for " << argv[1] << ": ";
    printPerms(buffer.st_mode);
    cout << endl;
```

## *1.21 chmod example (2 of 3)*

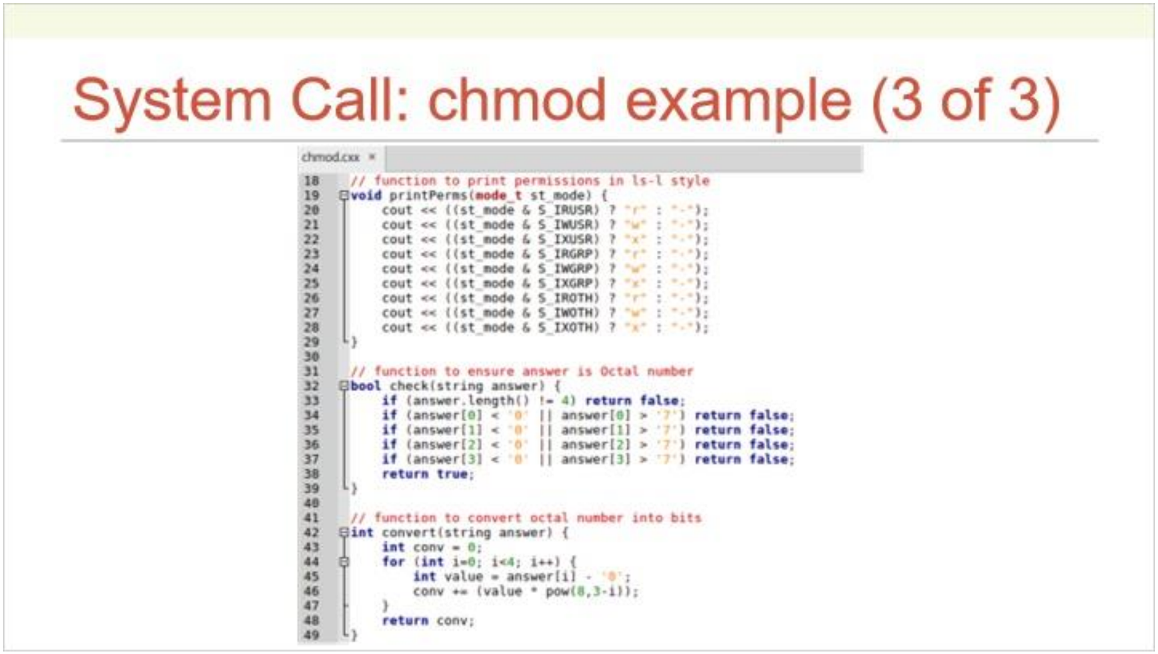


System Call: chmod example (2 of 3)

chmod.cxx ×

```
    // ask user for new permission settings
    string answer;
    cout << "Enter new permission mode (octal) for " << argv[1] << ": ";
    cin >> answer;
    if (!check(answer)) {
        cout << "Error: must be 4 digit octal number\n";
    } else {
        // changing permissions
        chmod(argv[1], convert(answer));
        rs = stat(argv[1], &buffer);
        if (rs == -1) {
            perror(argv[1]);
            exit(EXIT_FAILURE);
        }
        cout << "Current permission for " << argv[1] << ": ";
        printPerms(buffer.st_mode);
        cout << endl;
    }
    return 0;
}
```

### *1.22 chmod example (3 of 3)*

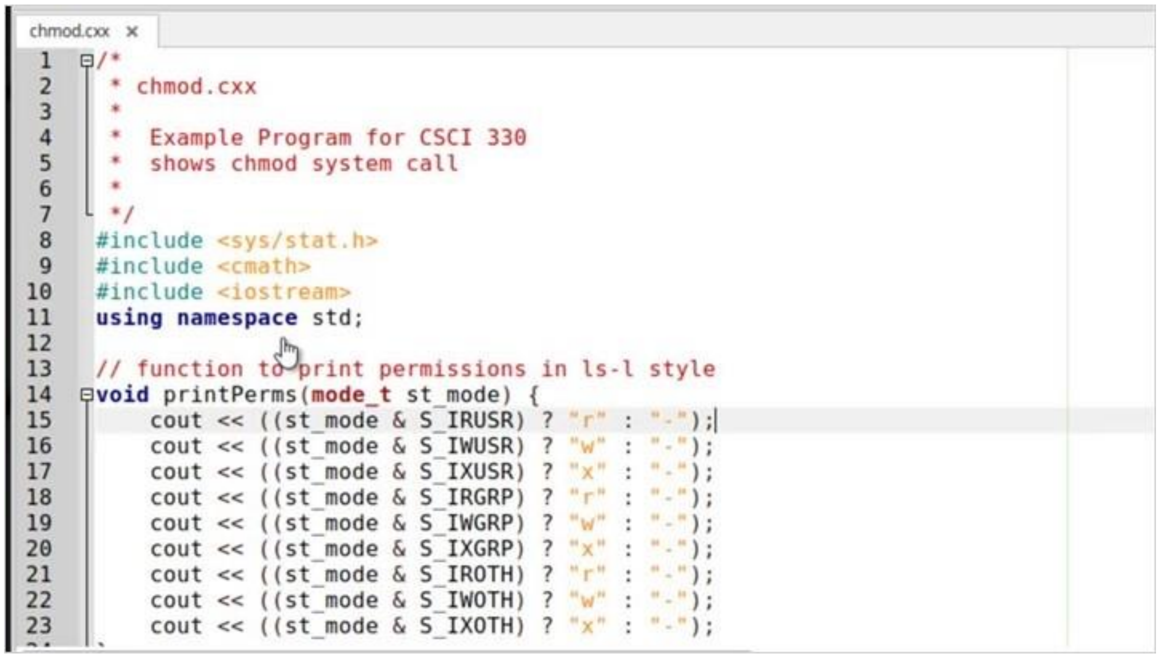


## System Call: chmod example (3 of 3)

chmod.cxx

```
// function to print permissions in ls-l style
void printPerms(mode_t st_mode) {
    cout << ((st_mode & S_IRUSR) ? "r" : "-");
    cout << ((st_mode & S_IWUSR) ? "w" : "-");
    cout << ((st_mode & S_IXUSR) ? "x" : "-");
    cout << ((st_mode & S_IRGRP) ? "r" : "-");
    cout << ((st_mode & S_IWGRP) ? "w" : "-");
    cout << ((st_mode & S_IXGRP) ? "x" : "-");
    cout << ((st_mode & S_IROTH) ? "r" : "-");
    cout << ((st_mode & S_IWOTH) ? "w" : "-");
    cout << ((st_mode & S_IXOTH) ? "x" : "-");
}

// function to ensure answer is Octal number
bool check(string answer) {
    if (answer.length() != 4) return false;
    if (answer[0] < '0' || answer[0] > '7') return false;
    if (answer[1] < '0' || answer[1] > '7') return false;
    if (answer[2] < '0' || answer[2] > '7') return false;
    if (answer[3] < '0' || answer[3] > '7') return false;
    return true;
}

// function to convert octal number into bits
int convert(string answer) {
    int conv = 0;
    for (int i=0; i<4; i++) {
        int value = answer[i] - '0';
        conv += (value * pow(8,3-i));
    }
    return conv;
}
```

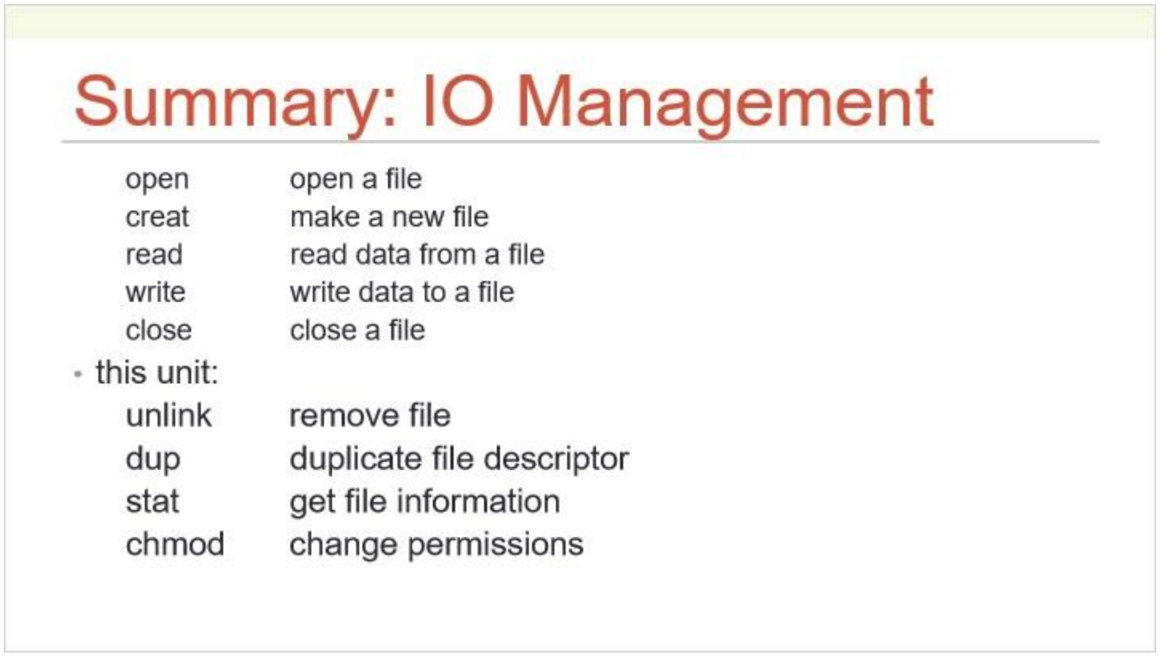
### *1.23 chmod.cxx*



chmod.cxx

```
/*
 * chmod.cxx
 *
 *  Example Program for CSCI 330
 *  shows chmod system call
 *
 */
#include <sys/stat.h>
#include <cmath>
#include <iostream>
using namespace std;

// function to print permissions in ls-l style
void printPerms(mode_t st_mode) {
    cout << ((st_mode & S_IRUSR) ? "r" : "-");
    cout << ((st_mode & S_IWUSR) ? "w" : "-");
    cout << ((st_mode & S_IXUSR) ? "x" : "-");
    cout << ((st_mode & S_IRGRP) ? "r" : "-");
    cout << ((st_mode & S_IWGRP) ? "w" : "-");
    cout << ((st_mode & S_IXGRP) ? "x" : "-");
    cout << ((st_mode & S_IROTH) ? "r" : "-");
    cout << ((st_mode & S_IWOTH) ? "w" : "-");
    cout << ((st_mode & S_IXOTH) ? "x" : "-");
```

## *1.24 Summary: IO Management*



## Summary: IO Management

| | |
|---|---|
| open | open a file |
| creat | make a new file |
| read | read data from a file |
| write | write data to a file |
| close | close a file |

- this unit:

| | |
|---|---|
| unlink | remove file |
| dup | duplicate file descriptor |
| stat | get file information |
| chmod | change permissions |