CSC 4304 - Systems Programming
Fall 2008

Lecture - IV

# File I/O

Tevfik Koşar

Louisiana State University
September 11th, 2008

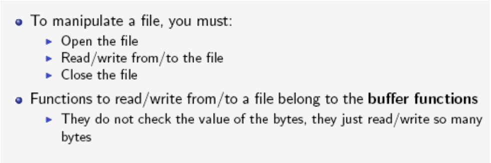
## Summary of Last Class

- Advanced Structures in C
  - Memory Manipulation in C
  - Pointers & Pointer Arithmetic
  - Parameter Passing
  - Structures
  - Local vs Global Variables
  - Dynamic Memory Management

## In Today's Class

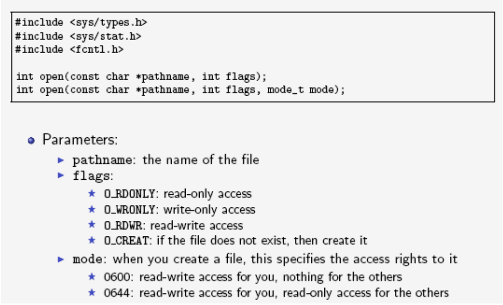
- File I/O
  - opening and closing files
  - reading from / writing to files
  - seeking files

## Managing Files



- To manipulate a file, you must:
  - Open the file
  - Read/write from/to the file
  - Close the file
- Functions to read/write from/to a file belong to the **buffer functions**
  - They do not check the value of the bytes, they just read/write so many bytes

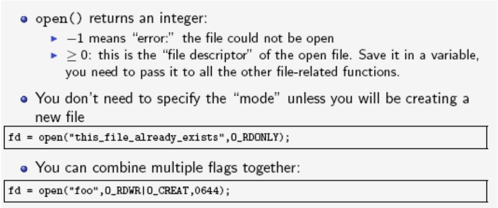
## Open a File



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

int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
```

- Parameters:
  - pathname: the name of the file
  - flags:
    - O_RDONLY: read-only access
    - O_WRONLY: write-only access
    - O_RDWR: read-write access
    - O_CREAT: if the file does not exist, then create it
  - mode: when you create a file, this specifies the access rights to it
    - 0600: read-write access for you, nothing for the others
    - 0644: read-write access for you, read-only access for the others

## Open a File

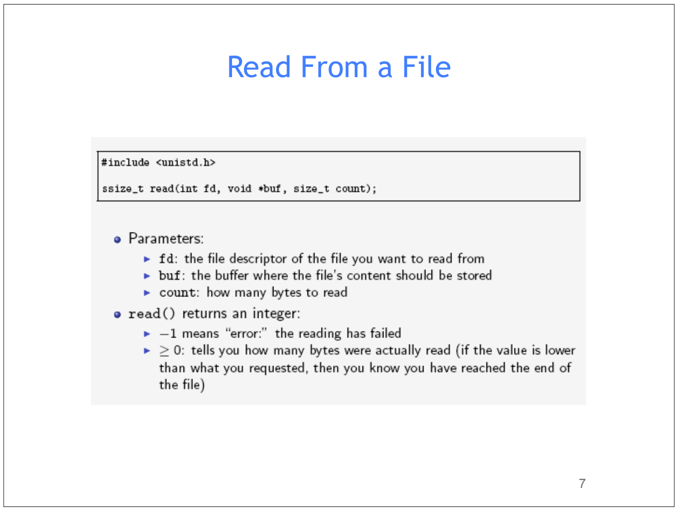


- open() returns an integer:
  - $-1$ means "error:" the file could not be open
  - $\geq 0$: this is the "file descriptor" of the open file. Save it in a variable, you need to pass it to all the other file-related functions.
- You don't need to specify the "mode" unless you will be creating a new file

```
fd = open("this_file_already_exists",O_RDONLY);
```

- You can combine multiple flags together:

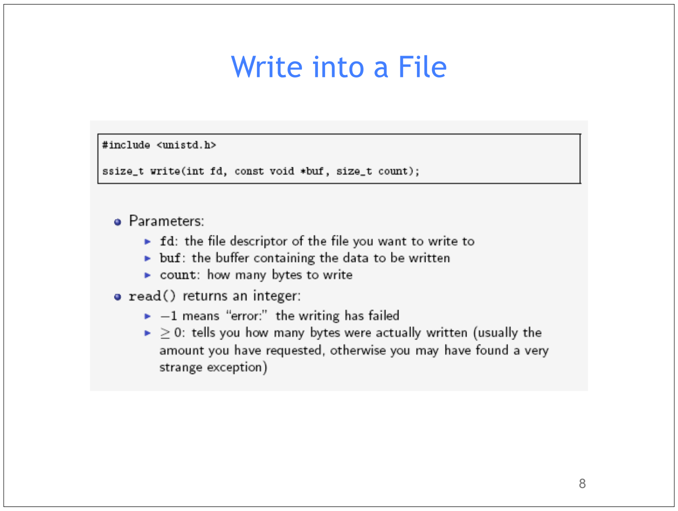
```
fd = open("foo",O_RDWR|O_CREAT,0644);
```



## Read From a File

```
#include <unistd.h>

ssize_t read(int fd, void *buf, size_t count);
```

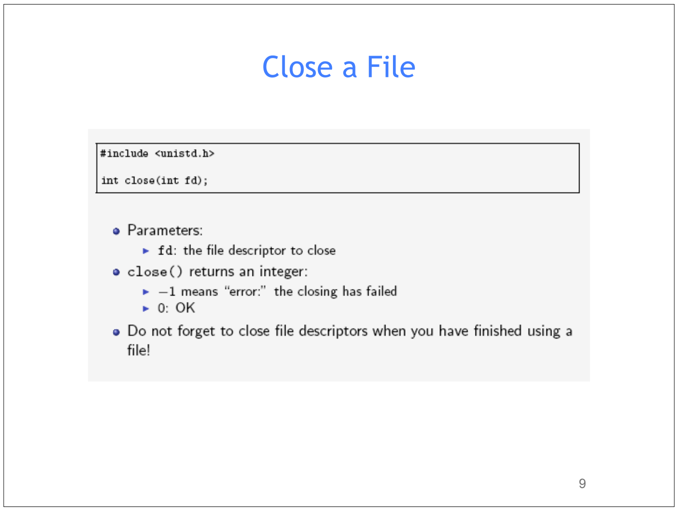
- Parameters:
  - `fd`: the file descriptor of the file you want to read from
  - `buf`: the buffer where the file's content should be stored
  - `count`: how many bytes to read
- `read()` returns an integer:
  - $-1$ means "error:" the reading has failed
  - $\geq 0$: tells you how many bytes were actually read (if the value is lower than what you requested, then you know you have reached the end of the file)



## Write into a File

```
#include <unistd.h>

ssize_t write(int fd, const void *buf, size_t count);
```

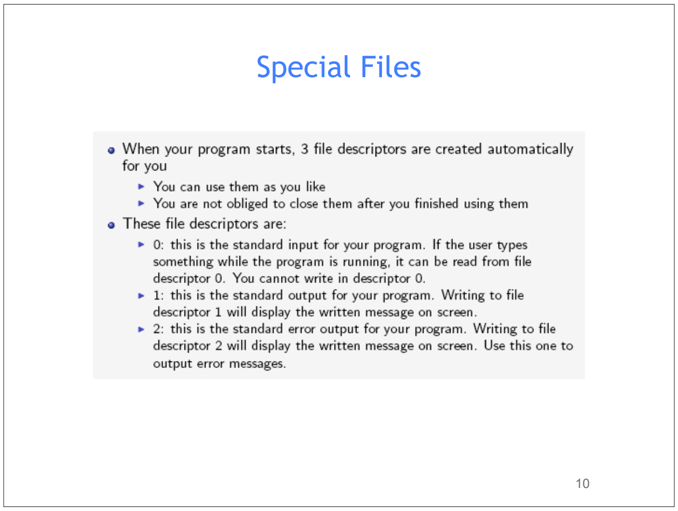
- Parameters:
  - `fd`: the file descriptor of the file you want to write to
  - `buf`: the buffer containing the data to be written
  - `count`: how many bytes to write
- `read()` returns an integer:
  - $-1$ means "error:" the writing has failed
  - $\geq 0$: tells you how many bytes were actually written (usually the amount you have requested, otherwise you may have found a very strange exception)



## Close a File

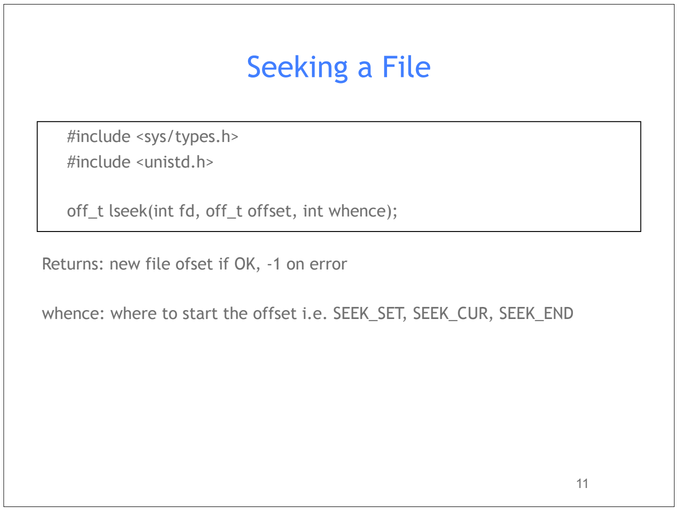
```
#include <unistd.h>

int close(int fd);
```

- Parameters:
  - `fd`: the file descriptor to close
- `close()` returns an integer:
  - $-1$ means "error:" the closing has failed
  - 0: OK
- Do not forget to close file descriptors when you have finished using a file!



## Special Files

- When your program starts, 3 file descriptors are created automatically for you
  - You can use them as you like
  - You are not obliged to close them after you finished using them
- These file descriptors are:
  - 0: this is the standard input for your program. If the user types something while the program is running, it can be read from file descriptor 0. You cannot write in descriptor 0.
  - 1: this is the standard output for your program. Writing to file descriptor 1 will display the written message on screen.
  - 2: this is the standard error output for your program. Writing to file descriptor 2 will display the written message on screen. Use this one to output error messages.

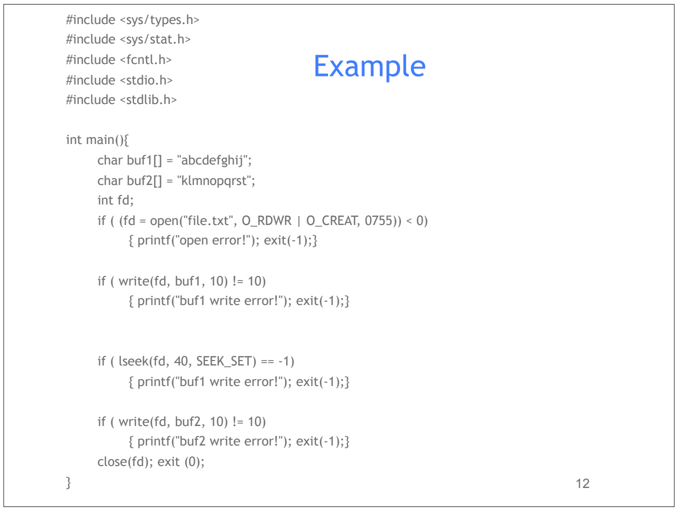


## Seeking a File

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

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

Returns: new file ofset if OK, -1 on error

whence: where to start the offset i.e. SEEK_SET, SEEK_CUR, SEEK_END

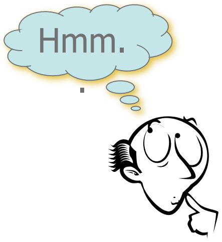


## Example

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

int main(){
        char buf1[] = "abcdefghij";
        char buf2[] = "klmnopqrst";
        int fd;
        if ( (fd = open("file.txt", O_RDWR | O_CREAT, 0755)) < 0)
                { printf("open error!"); exit(-1);}

        if ( write(fd, buf1, 10) != 10)
                { printf("buf1 write error!"); exit(-1);}


        if ( lseek(fd, 40, SEEK_SET) == -1)
                { printf("buf1 write error!"); exit(-1);}

        if ( write(fd, buf2, 10) != 10)
                { printf("buf2 write error!"); exit(-1);}
        close(fd); exit (0);
}
```

## Summary

- File I/O
  - opening and closing files
  - reading from / writing to files
  - seeking files
- Next Week: Continue File I/O



- Read Ch.3 from Stevens
- HW-1 due Sep 18th
- Project-1 out next week

## Acknowledgments

- Advanced Programming in the Unix Environment by R. Stevens
- The C Programming Language by B. Kernighan and D. Ritchie
- Understanding Unix/Linux Programming by B. Molay
- Lecture notes from B. Molay (Harvard), T. Kuo (UT-Austin), G. Pierre (Vrije), M. Matthews (SC), and B. Knicki (WPI).