11. Low Level File Input / Output
ENEE 140

Prof. Tudor Dumitraș

Associate Professor, ECE
University of Maryland, College Park

http://ter.ps/enee140

Today’s Lecture

- Where we’ve been
  - Scalar data types
  - Arrays and strings
  - Functions
  - Random number generation
  - Control flow
  - Structuring complex programs
  - Formatted and character file I/O
- Where we’re going today
  - File Input/Output (low level)
- Where we’re going next
  - File Input/Output (high level)

Review: Nested Loops

- You can nest loops

```
for (i=1; i<=3; i++) {
 for (j=1; j<=3; j++) {
    printf("%dx%d=%2d\t", i, j, i*j);
 }
 printf("\n");    // ready for next line
}
```

- Output

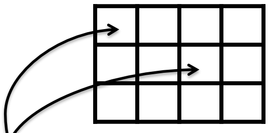
```
1x1= 1  1x2= 2    1x3= 3
2x1= 2  2x2= 4    2x3= 6
3x1= 3  3x2= 6    3x3= 9
```

Two-Dimensional Arrays

- Two-dimensional arrays

  `int a[3][4];` int array with 3 rows and 4 columns (12 elements)

  – Think of this as 3 arrays with 4 elements each



- Working with 2D arrays

  `a[0][0] = 0;` access element on first row and first column

  `a[1][2] = 0;` access element on row 1 and column 2

  ⚡ `a[0][4] = 0;` error: index out of bounds

  ⚡ `a[3][0] = 0;` error: index out of bounds

  – Use 2D arrays to represent matrices

- Arrays with 3, 4, 5, etc. dimensions

  `int a[2][3][4];` 3D array with 24 elements

Review: Binary Representation

- Numbers in base B $N = \sum n_i \cdot B^i$

- Examples
  - Binary (base 2): digits 0 and 1
    - $0011_2 = 1*2 + 1 = 3_{10}$
  - Octal (base 8): digits 0, 1, 2, 3, 4, 5, 6, 7
    - $14_8 = 1*8 + 4 = 12_{10}$

- Geeky joke:
  - *Why do programmers confuse Halloween and Christmas?*

6

6

Review: Bitwise Operations

- Operators for manipulating bits:

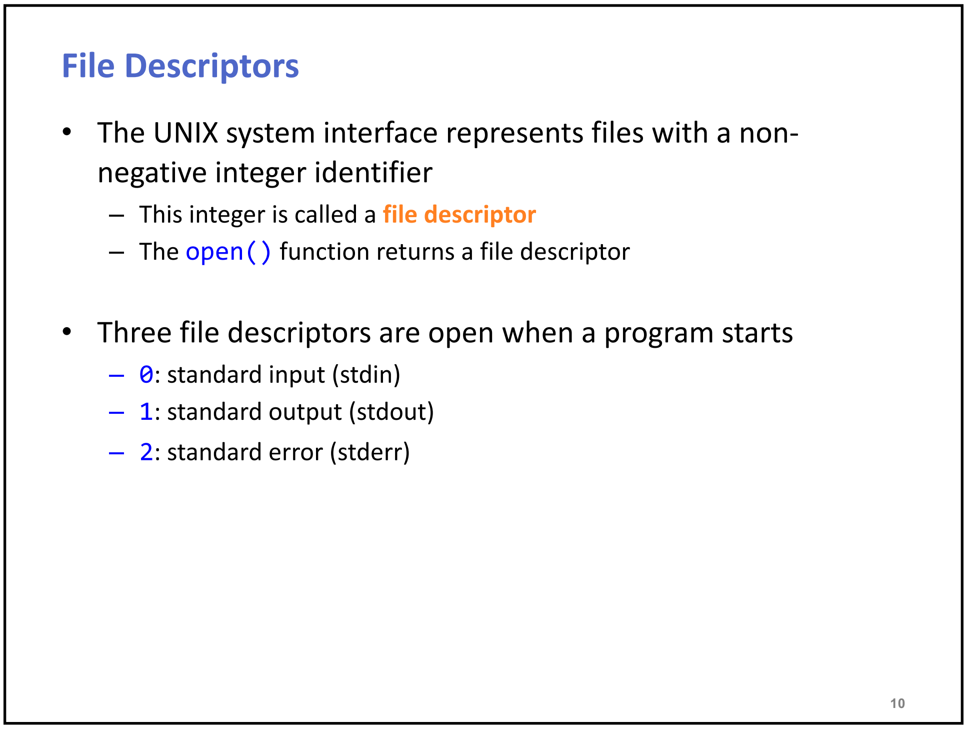
&	bitwise AND
|	bitwise OR
^	bitwise XOR (exclusive OR)
<<	left shift
>>	right shift
~	flip all bits (unary)

- Common usage: bit masks

`a = a & 1;`	set all but lowest order bit to 0
`a = a | 1;`	set lowest order bit to 1;
`b = (a>>2) & 1;`	find value of bit b2 from b31 … b3 b2 b1 b0

7

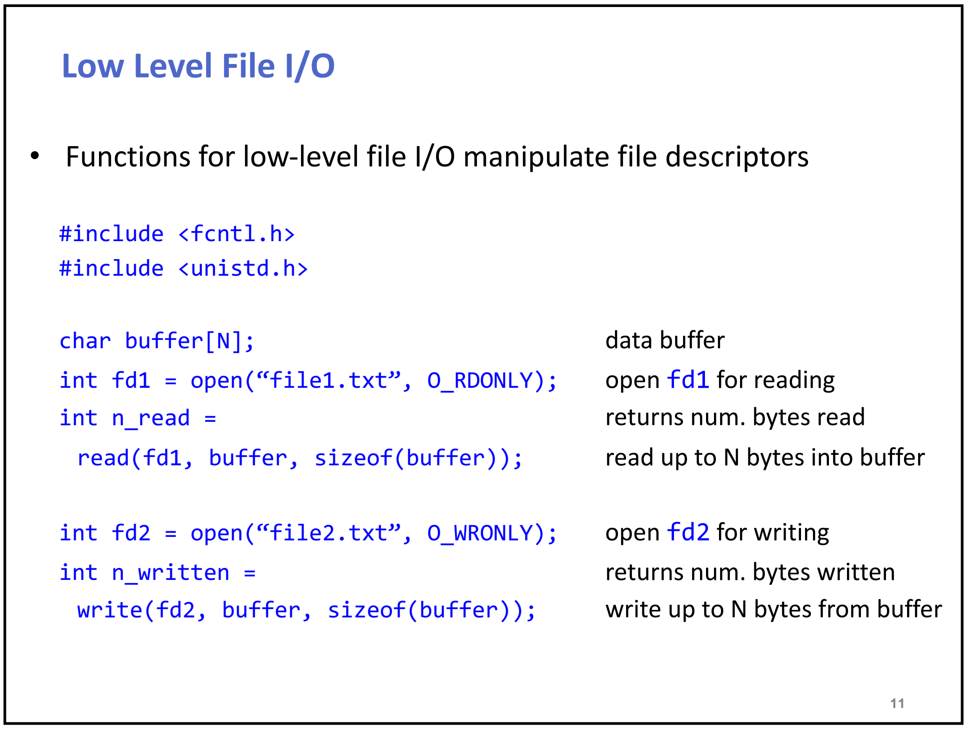
7



File Descriptors

- The UNIX system interface represents files with a non-negative integer identifier
  - This integer is called a **file descriptor**
  - The `open()` function returns a file descriptor
- Three file descriptors are open when a program starts
  - `0`: standard input (stdin)
  - `1`: standard output (stdout)
  - `2`: standard error (stderr)

10



Low Level File I/O

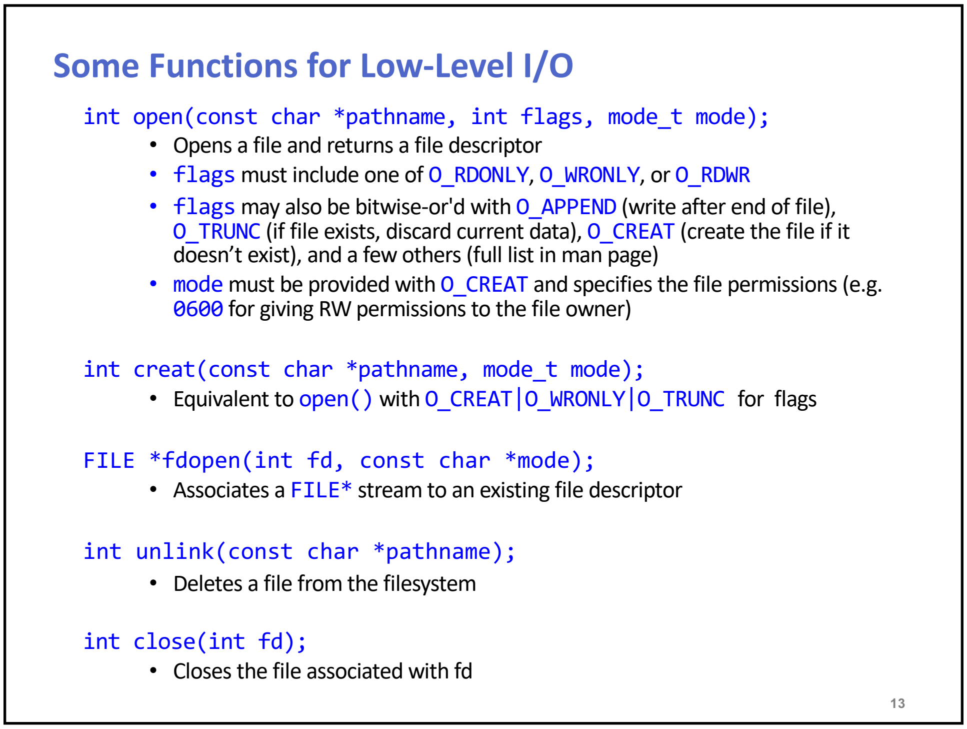
- Functions for low-level file I/O manipulate file descriptors

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

char buffer[N];                           data buffer
int fd1 = open(“file1.txt”, O_RDONLY);    open fd1 for reading
int n_read =                              returns num. bytes read
  read(fd1, buffer, sizeof(buffer));      read up to N bytes into buffer

int fd2 = open(“file2.txt”, O_WRONLY);    open fd2 for writing
int n_written =                           returns num. bytes written
  write(fd2, buffer, sizeof(buffer));     write up to N bytes from buffer
```

11



Some Functions for Low-Level I/O

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

- Opens a file and returns a file descriptor
- flags must include one of O_RDONLY, O_WRONLY, or O_RDWR
- flags may also be bitwise-or'd with O_APPEND (write after end of file), O_TRUNC (if file exists, discard current data), O_CREAT (create the file if it doesn't exist), and a few others (full list in man page)
- mode must be provided with O_CREAT and specifies the file permissions (e.g. 0600 for giving RW permissions to the file owner)

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

- Equivalent to open() with O_CREAT|O_WRONLY|O_TRUNC for flags

```
FILE *fdopen(int fd, const char *mode);
```

- Associates a FILE* stream to an existing file descriptor

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

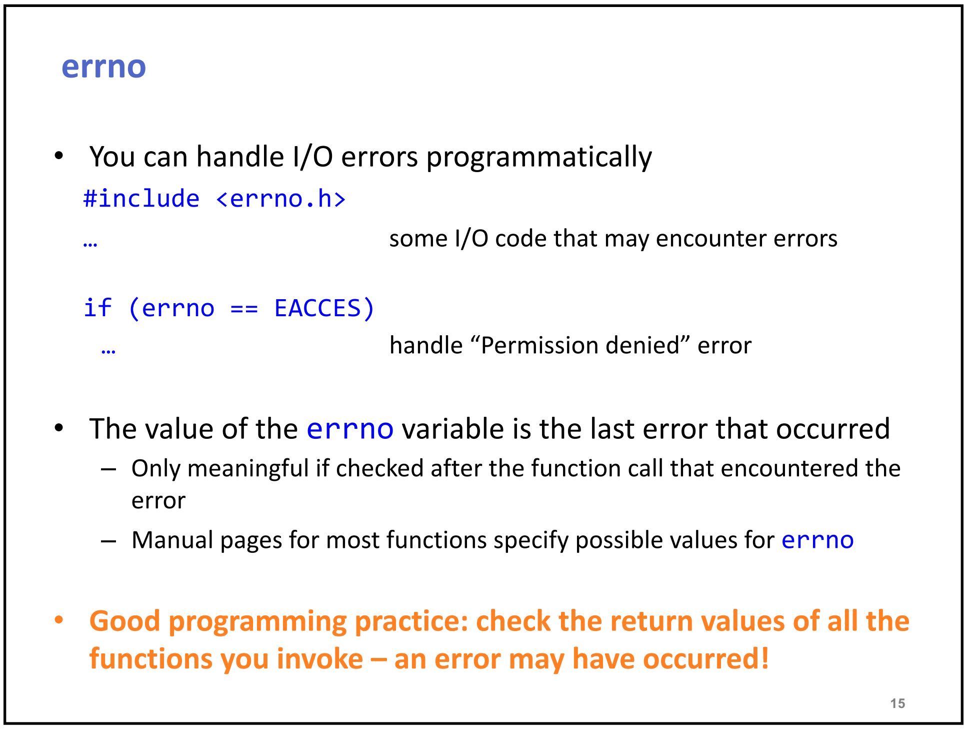
- Deletes a file from the filesystem

```
int close(int fd);
```

- Closes the file associated with fd

13

13



errno

- You can handle I/O errors programmatically

```
#include <errno.h>
…
```

some I/O code that may encounter errors

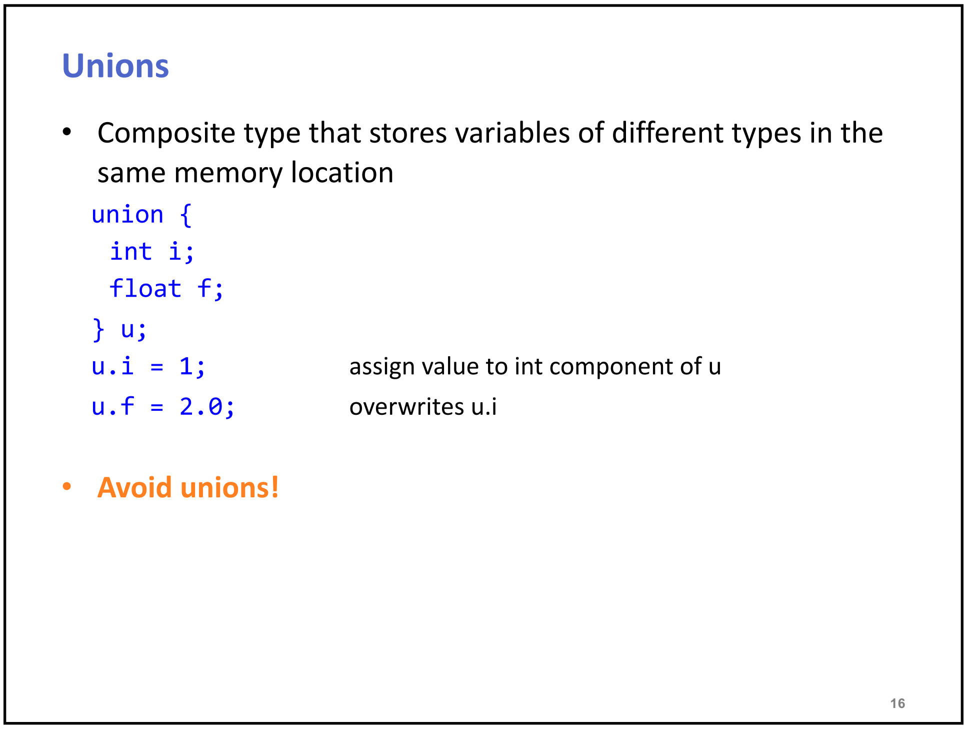
```
if (errno == EACCES)
  …
```

handle "Permission denied" error

- The value of the errno variable is the last error that occurred
  - Only meaningful if checked after the function call that encountered the error
  - Manual pages for most functions specify possible values for errno
- **Good programming practice: check the return values of all the functions you invoke – an error may have occurred!**

15

15



Unions

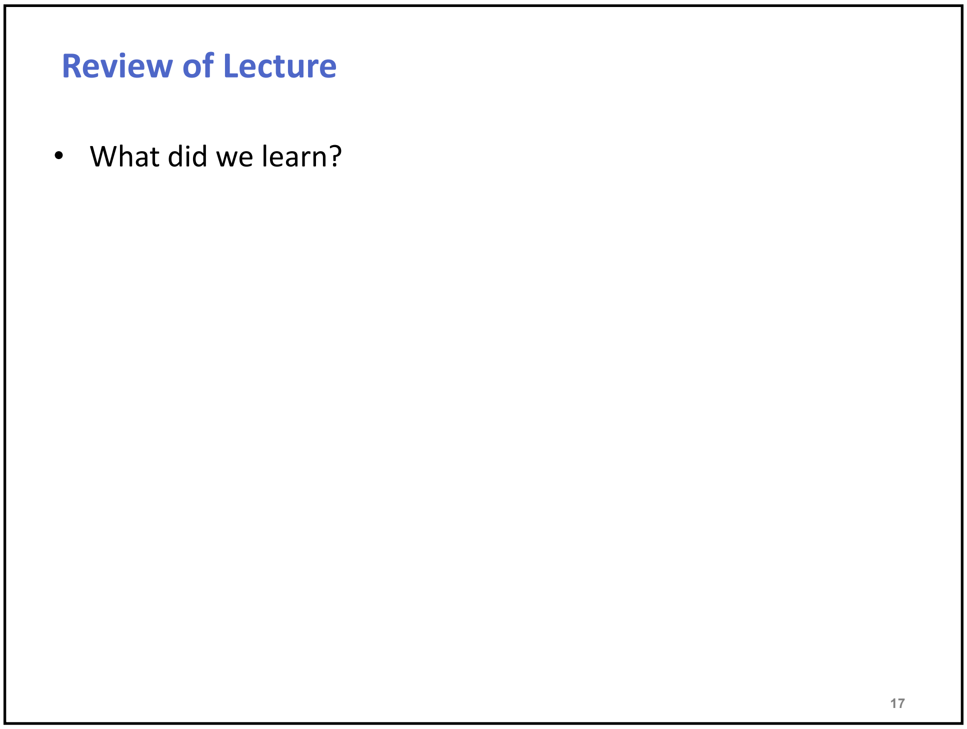
- Composite type that stores variables of different types in the same memory location

```
union {
  int i;
  float f;
} u;
u.i = 1;            assign value to int component of u
u.f = 2.0;          overwrites u.i
```

- **Avoid unions!**

16

16

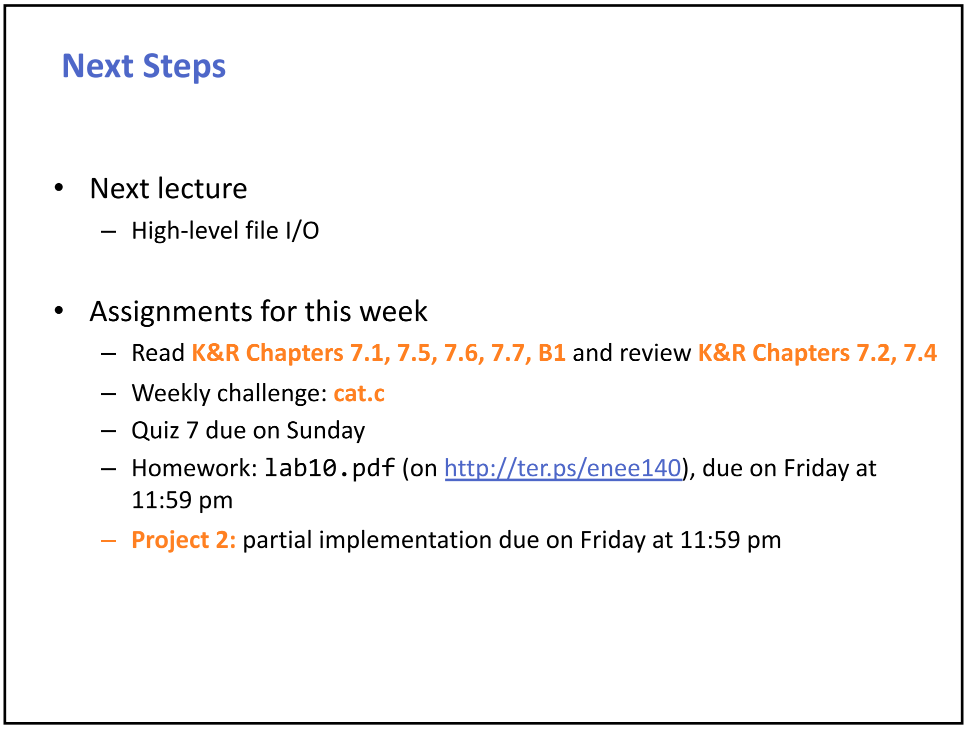


Review of Lecture

- What did we learn?

17

17



Next Steps

- Next lecture
  - High-level file I/O

- Assignments for this week
  - Read **K&R Chapters 7.1, 7.5, 7.6, 7.7, B1** and review **K&R Chapters 7.2, 7.4**
  - Weekly challenge: **cat.c**
  - Quiz 7 due on Sunday
  - Homework: `lab10.pdf` (on http://ter.ps/enee140), due on Friday at 11:59 pm
  - **Project 2:** partial implementation due on Friday at 11:59 pm