Linux Systems Programming

Setting the Scene

Chris Brown

In This Module …

What is systems programming?

Kernel space , user space

System calls, library routines

The mechanics:

Header files

Compilation

Error handling

Language choice:

C

Python

Demonstration:

Install the tools

Hello World

Assumed Knowledge

C language

Syntax, data types, structures, pointers

Python

Not essential (but see "Python Fundamentals" for a good intro)

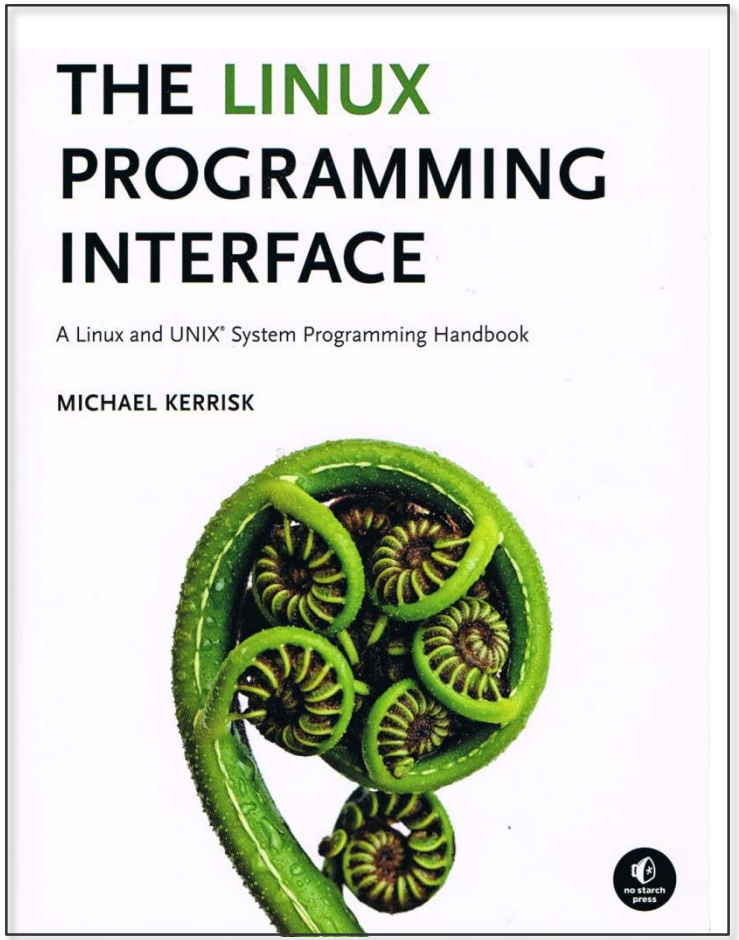
Linux

Command-line comfort

Concepts:

- **Files, links, directories, permissions**
- **Processes and pipes**

Book Recommendation



1500 pages!

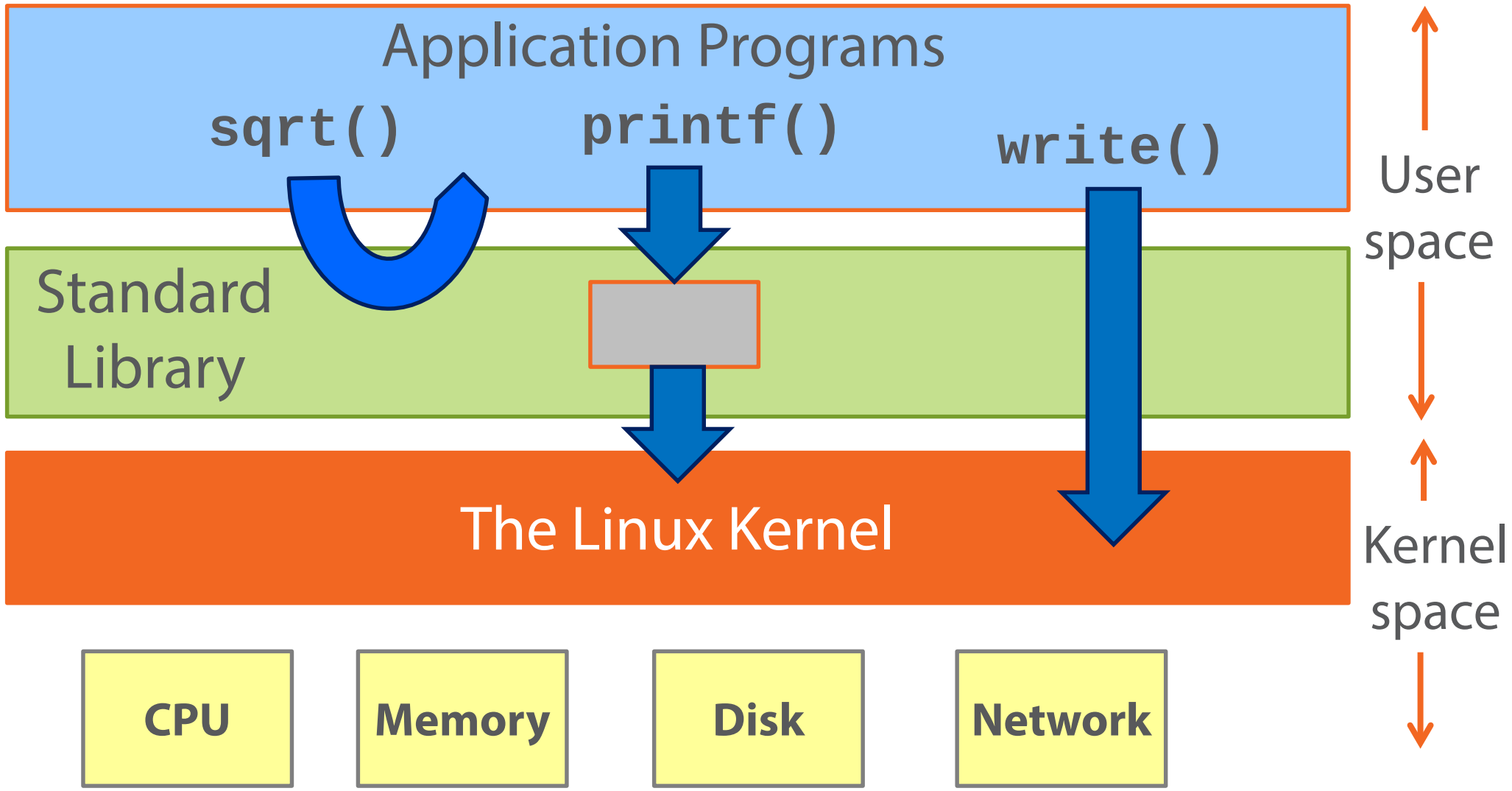
Authoritative

Well written

Lots of examples

Excellent!

Kernel Space and User Space



Hello World

```
#include <fcntl.h>
void main()
{
  int fd;
  fd = open("foo", O_WRONLY | O_CREAT, 0644);
  write(fd, "hello world", 11);
  close(fd);
}
```

Error Handling

```
#include <fcntl.h>
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
...
  fd = open("foo", O_WRONLY | O_CREAT, 0644);
  if (fd < 0) {
    printf("error number %d\n", errno);
    perror("foo");
    exit(1);
  }
```

Error Handling

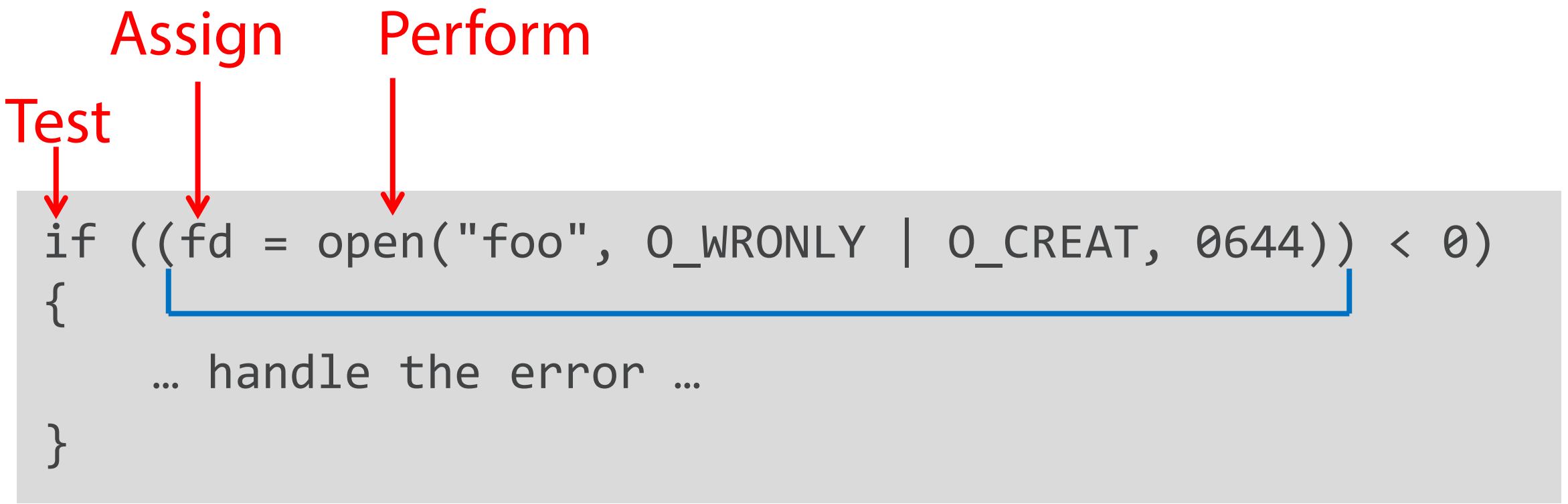
```
#include <fcntl.h>
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
...
  fd = open("foo", O_WRONLY | O_CREAT, 0644);
  if (fd < 0) {
    printf("error number %d\n", errno);
    perror("foo");
    exit(1);
  }
```

errno defined here

Typically, system calls return -1 on failure

They set the global variable errno

A Common Idiom



C and Python

- Low-level systems programming
- Dennis Ritchie, 1972
- Statically typed
- Procedural
- Fully compiled

- High-level, multi-purpose
- Guido van Rossum, 1991
- Dynamically typed
- Multi-paradigm including O-O
- Interpreted

The "man" Pages

- Traditional source of documentation in UNIX and Linux
  - Usually installed locally
  - Also available online (e.g. `linux.die.net/man`)
- Accessed via the `man` command:
  - `man chmod`
    - Gets the user command
  - `man 2 chmod`
    - Gets the system call

Section	Contents
1	User commands
2	System calls
3	Library routines
7	Miscellaneous

Understanding the "man" Pages

```
SYNOPSIS
       #include <unistd.h>

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

DESCRIPTION
       write() writes up to count bytes from the buffer pointed buf
       to the file referred to by the file descriptor fd.

       The number of bytes written may be less than count  if,  for
       example, there is insufficient space on the underlying phys-
       ical medium, or the RLIMIT_FSIZE resource limit  is  encoun-
       tered  (see  setrlimit(2)), or the call was interrupted by a
       signal handler after having written less than  count  bytes.
       (See also pipe(7).)
```

You will probably need this header file

Function prototype (*not* an example of an actual call)

Cross-reference

Data Types

Typedef name	Actual Type	Description
pid_t	int	A process ID or process group ID
gid_t	unsigned int	A numeric group identifier
uid_t	unsigned int	A numeric user identifier
time_t	long int	Time (in seconds) since "the epoch"
size_t	unsigned long	The size of an object in bytes
ssize_t	long int	The size of an object, or a negative error indication
mode_t	unsigned int	File permissions
off_t	long int	A file offset or size
socklen_t	unsigned int	The size of a socket address structure

Typedefs improve portability

Most resolve to some sort of integer

Python Documentation

- Excellent documentation online: https://docs.python.org
  - Searchable
- Can be downloaded as PDF, HTML, HTML, plain text or EPUB

Understanding the Python Documentation

open( ) is a built-in function
No external module to import

Parameters with a default value are optional

open(*file*, *mode='r'*, *buffering=-1*, *encoding=None*, *errors=None*, *newline=None*, *closefd=True*, *opener=None*)

Open *file* and return a corresponding *file object*. If the file cannot be opened, an `OSError` is raised.

file is either a string or bytes object giving the pathname (absolute or relative to the current working directory) of the file to be opened or an integer file descriptor of the file to be wrapped. (If a file descriptor is given, it is closed when the returned I/O object is closed, unless *closefd* is set to `False`.)

Standards

This course is largely compliant with POSIX .1

IEEE standard

Specifies an API for a set of system services

Does not specify the underlying operating system (but modeled on UNIX)

SUA (Subsystem for UNIX-based applications) provides POSIX compliance for Windows

SUSv4 (Single Unix System) is a closely related standard.

Demonstration

Install development environments

Write / compile / run "hello world"

Error handling

Python equivalents

Moving Forward …

In this module:

- Kernel space / user space
- System calls, header files,
- Error handling
- C and Python

Coming up in the next module:

- File input/output
- Four ways to copy a file