OPerating Systems Laboratory
OPS*Lab*

Course introduction
&
Unix System Overview

Prof. Marco Autili

University of L'Aquila

marco.autili@univaq.it

http://people.disim.univaq.it/marco.autili/

Course Organization (main topics)

- PART I - UNIX System Overview
- PART II - Command Line
  - Bash shell
- PART III - Bash Scripting
- PART IV - Programming in the UNIX Environment

Teaching Materials

Reference textbook

- [APUE3ed] Advanced Programming in the UNIX Environment
  3rd Edition, Richard Stevens and Steven Rago

Other materials

- My slides
- All the sources referenced within and at the end of my slides
- Source code of the practical examples

- Slides are not enough!

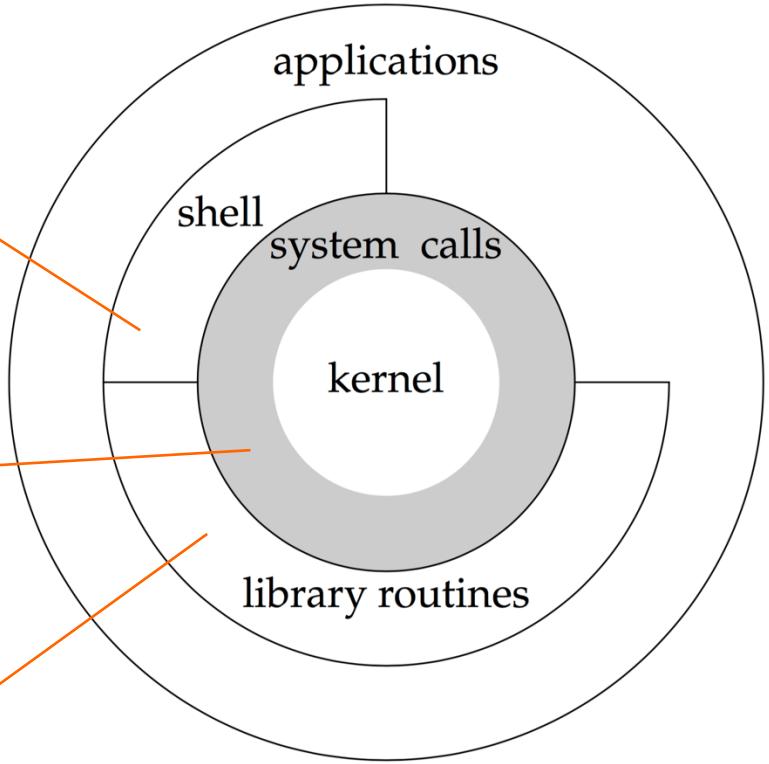
UNIX Architecture

- An operating system can be defined as the software that controls the hardware resources of the computer and provides an environment for running programs

Special application(s) that provides an interface for running other applications (commands, scripts, ...)

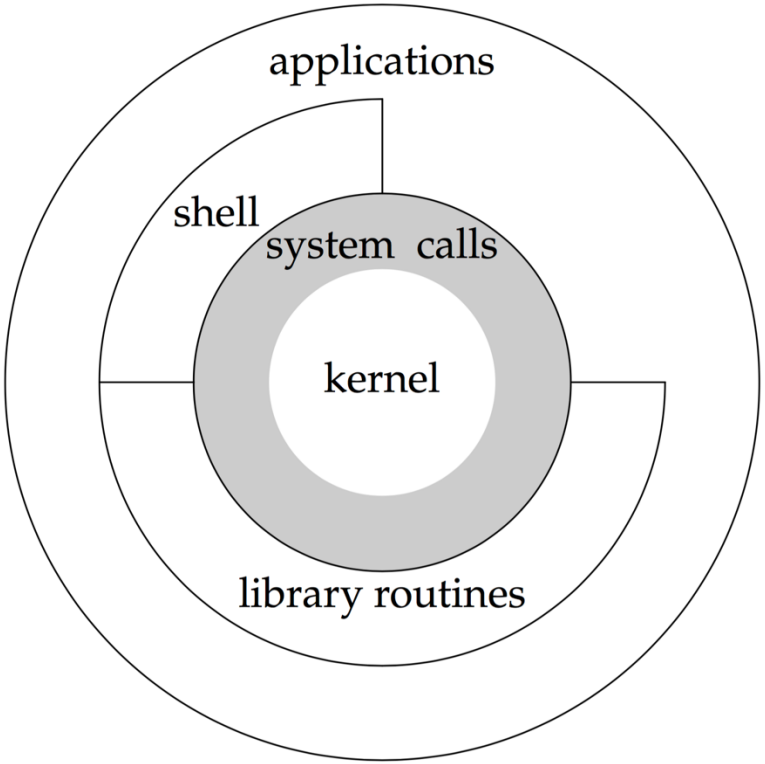
Interface to the kernel

Common functions built on top of the system call interface



In a broader sense...

- An operating system consists of the kernel and all the other software that makes a computer useful and gives the computer its personality. This other software includes system utilities, applications, shells, libraries of common functions, and so on



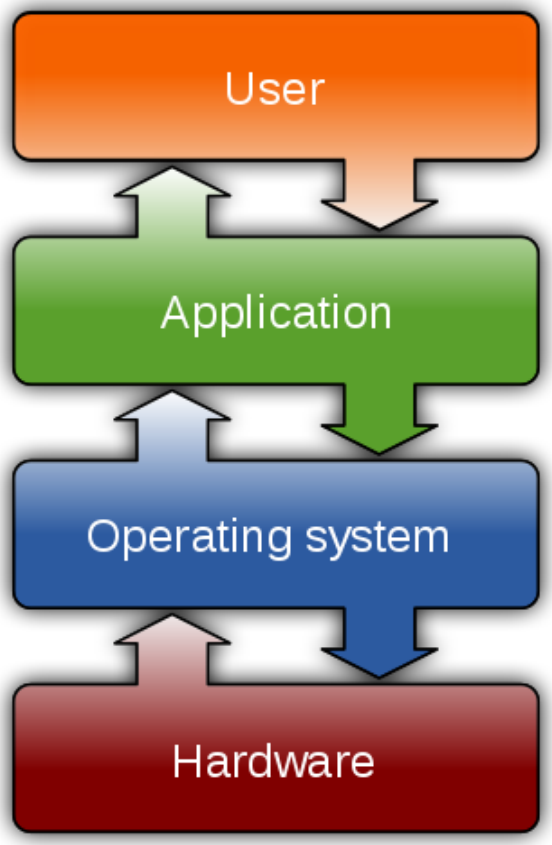
- There exist a number of variants of Unix (i.e., UNIX-like systems), notably,
  - The **Linux kernel** is a (free and open source) UNIX-like kernel that is used in many computer systems worldwide.
  - **Linux** is a generic name for a family of open-source Unix-like operating systems based on the Linux kernel, e.g., **GNU OS** adopts the Linux kernel
  - **Darwin** is the core Unix-like operating system of macOS, watchOS, tvOS, iPadOS, audios, visionOS, bridgeOS

A different view

Definition from Wikipedia

- An operating system (OS) is system software that manages computer hardware, software resources, and provides common services for computer programs

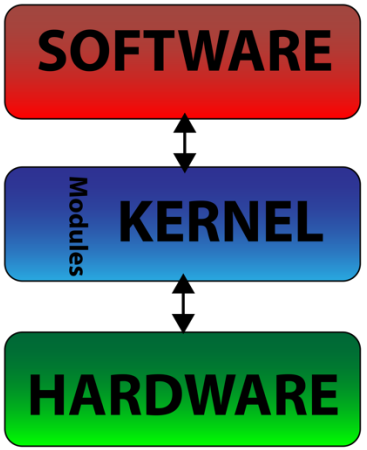
- Time-sharing operating systems schedule tasks for efficient use of the system and may also include accounting software for cost allocation of processor time, mass storage, printing, and other resources



https://en.wikipedia.org/wiki/Operating_system

Kernel

- The kernel is a computer program at the core of a computer's operating system and has complete control over everything in the system

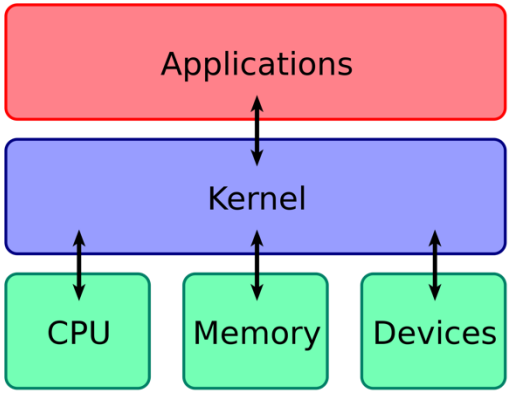


- It is the "portion of the operating system code that is always resident in memory" and facilitates interactions between hardware and software components

- A full kernel controls all hardware resources (e.g., I/O, memory, cryptography) via device drivers, arbitrates conflicts between processes concerning such resources, and optimizes the utilization of common resources, e.g., CPU and cache usage, file systems, and network sockets

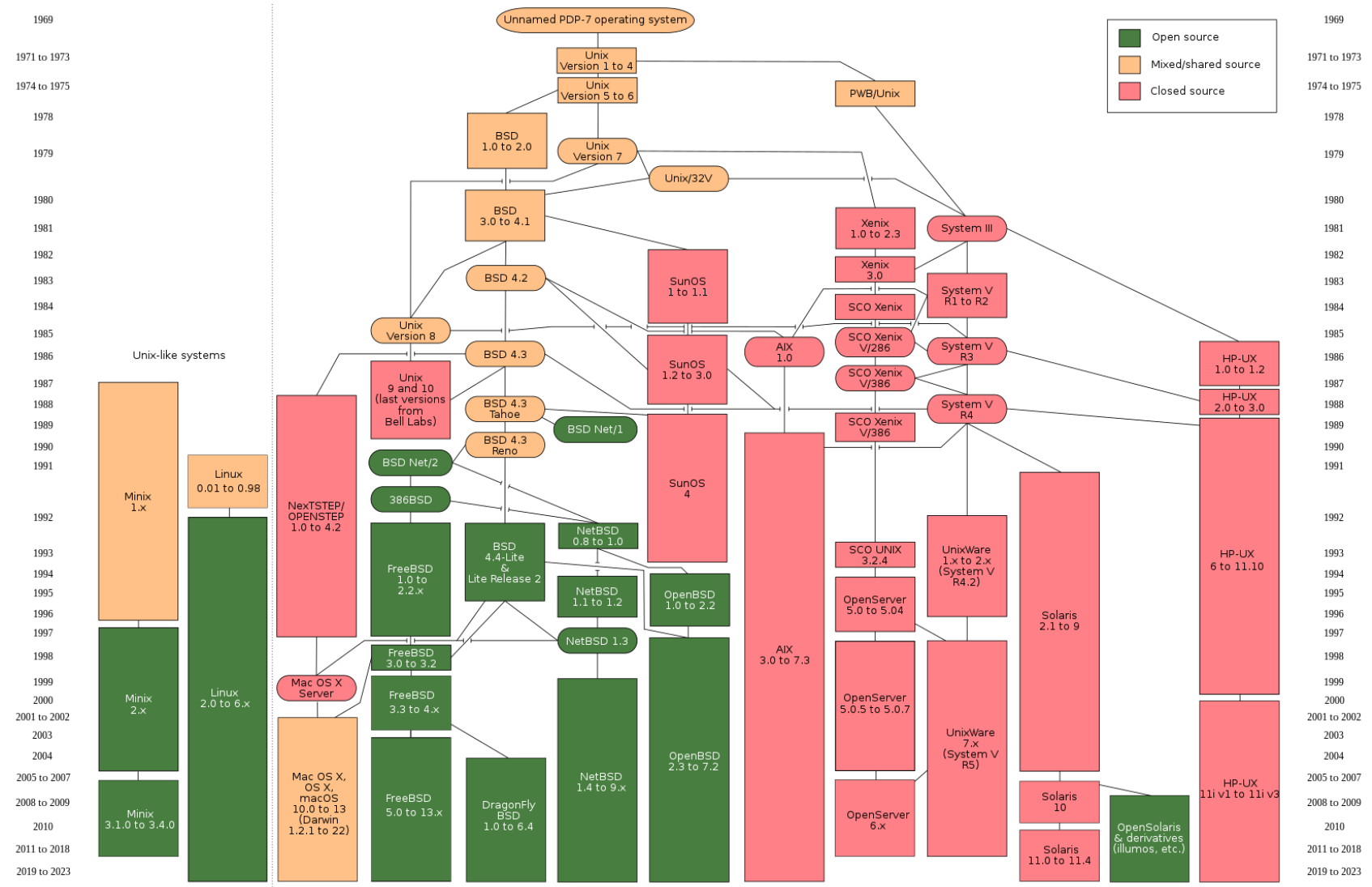
- On most systems, the kernel is one of the first programs loaded on startup (after the bootloader)

- It handles the rest of startup as well as memory, peripherals, and input/output (I/O) requests from software, translating them into data-processing instructions for the central processing unit

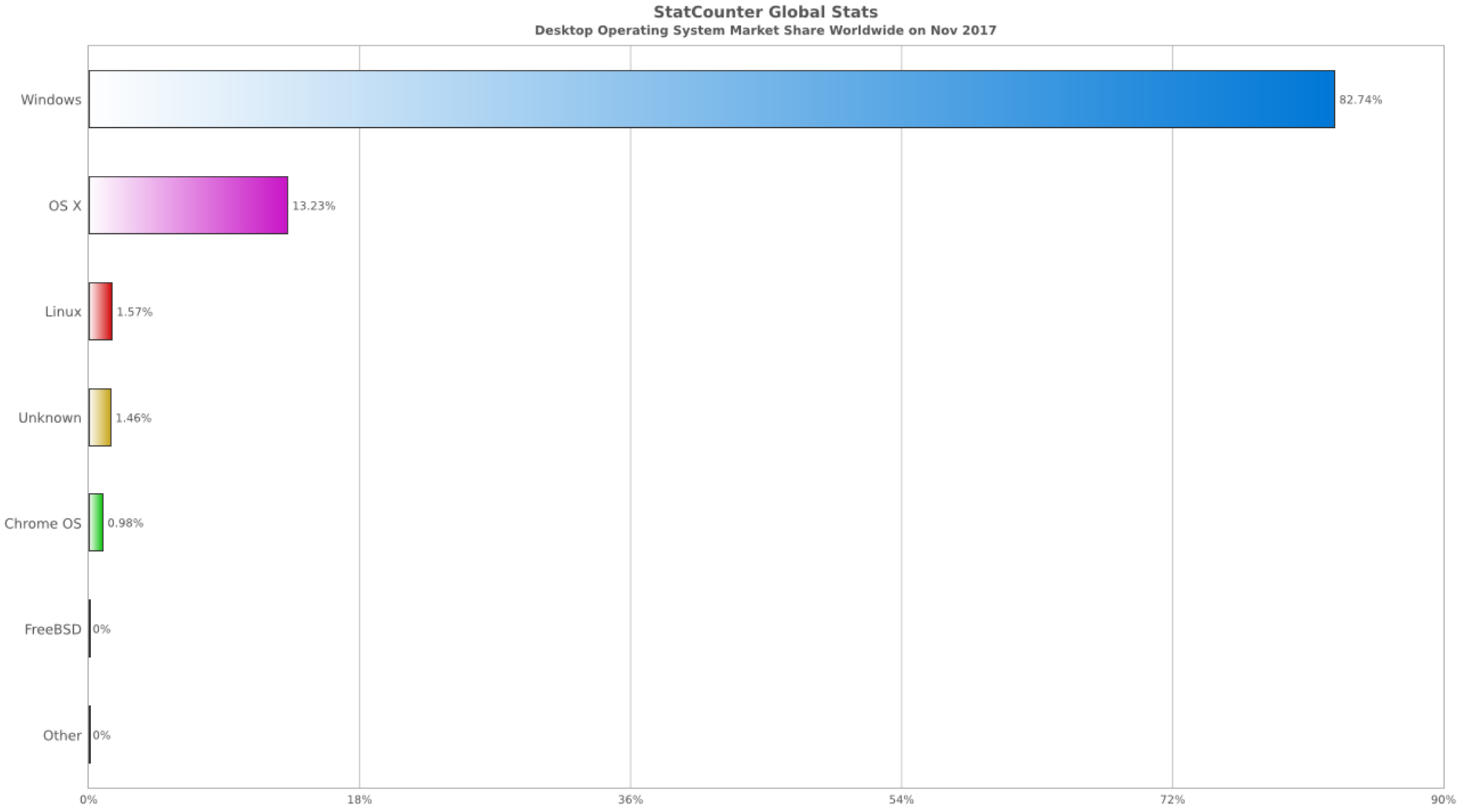


https://en.wikipedia.org/wiki/Kernel_(operating_system)

Evolution of Unix and Unix-like systems

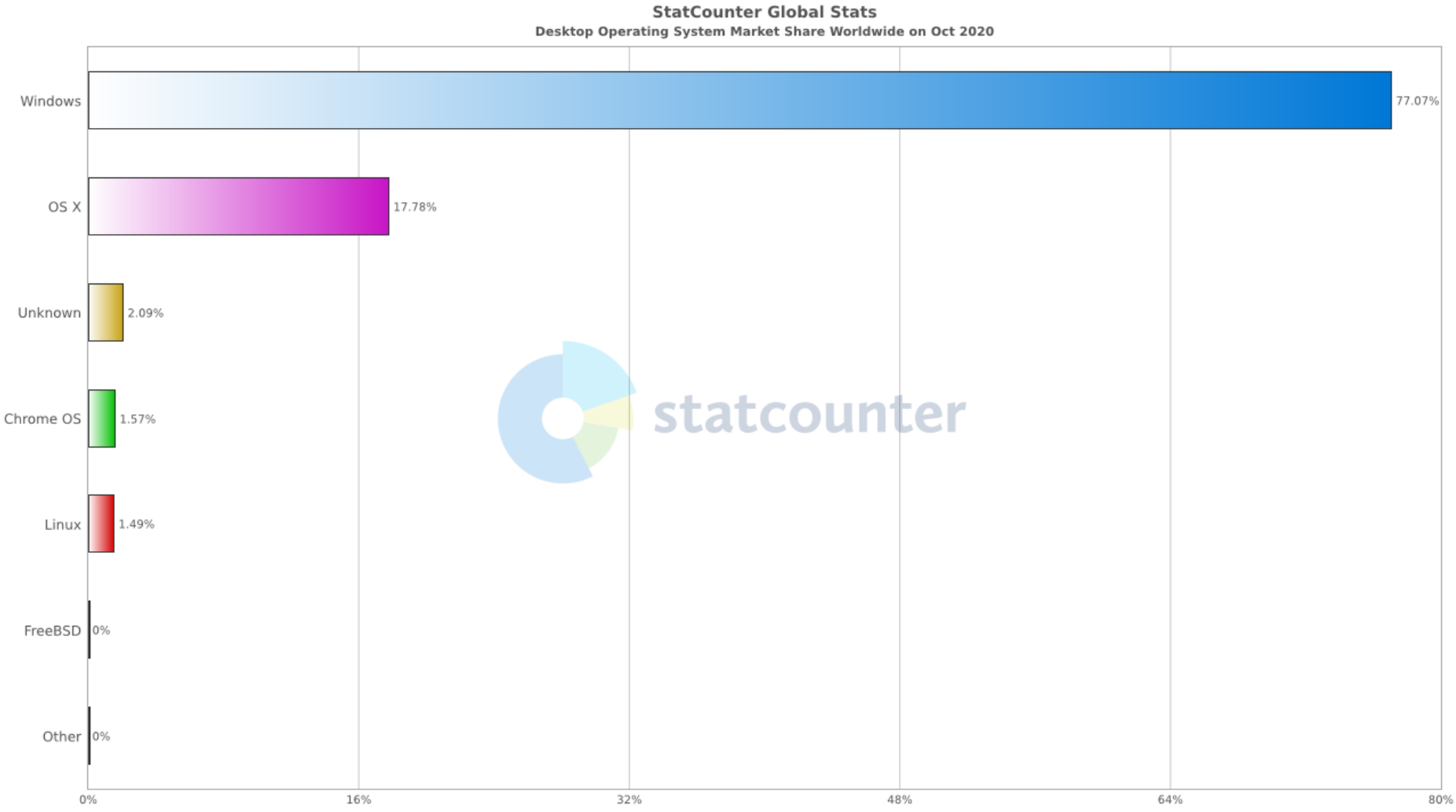


Desktop Operating System Market Share Worldwide (Nov 2017)



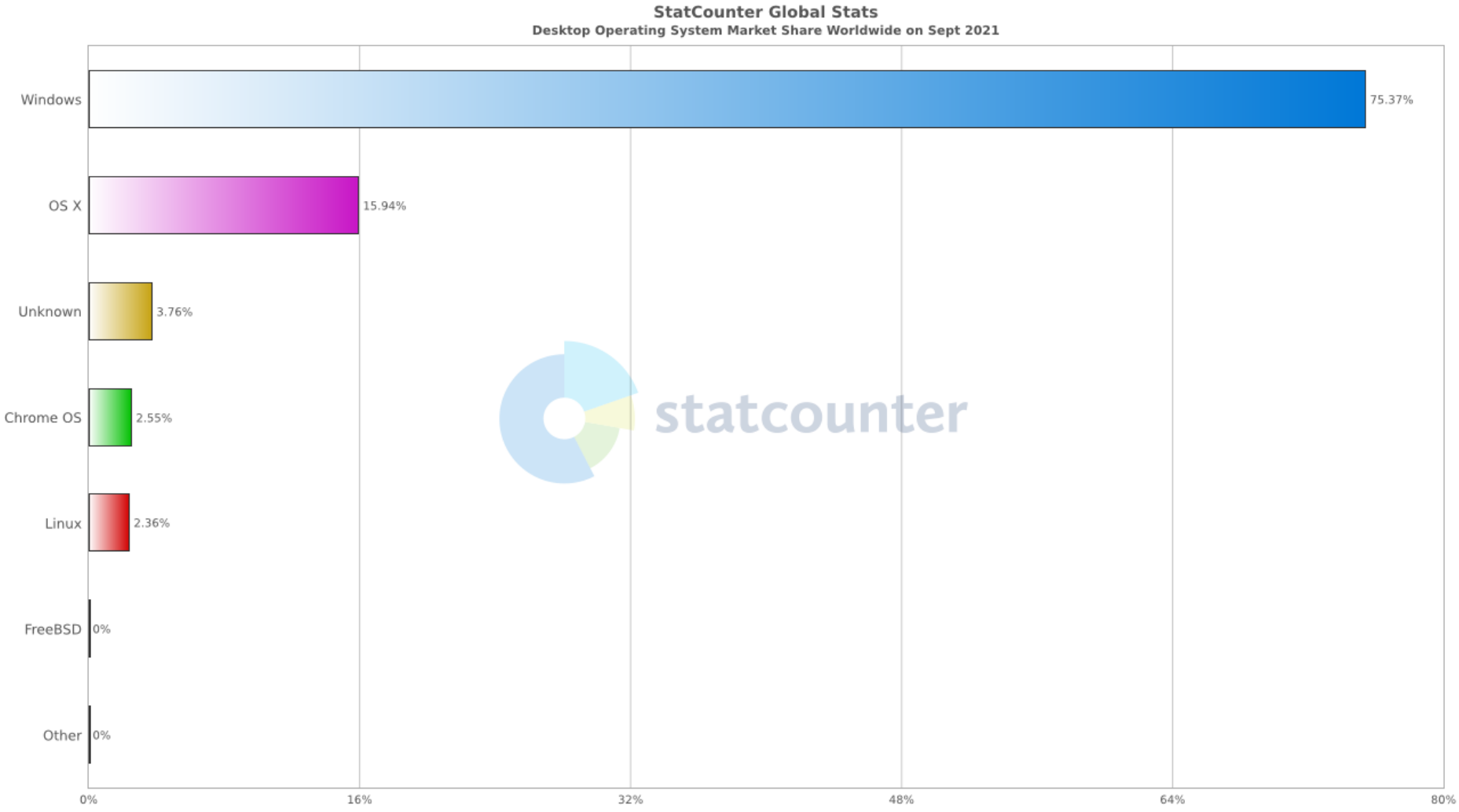
https://gs.statcounter.com/os-market-share/desktop/worldwide/#monthly-201711-201711-bar

Desktop Operating System Market Share Worldwide (Oct 2020)



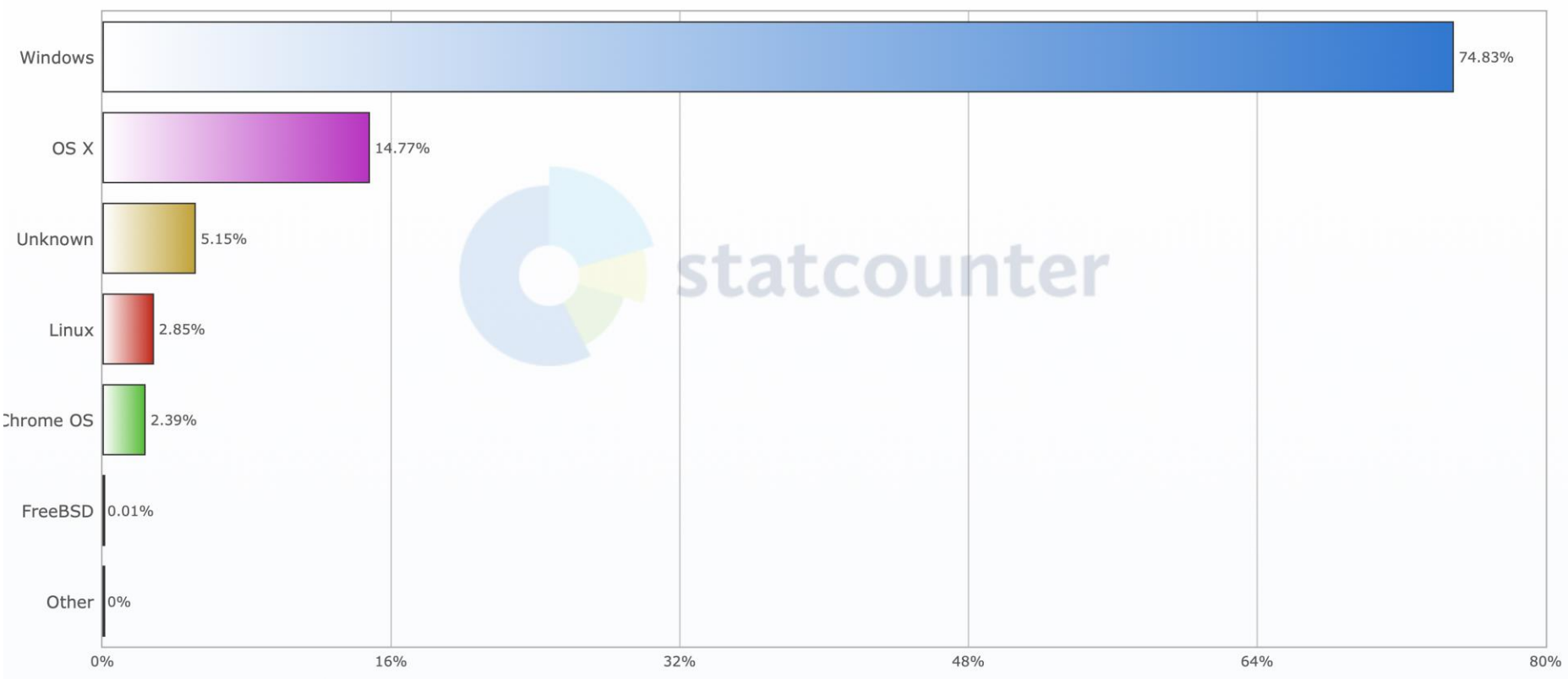
https://gs.statcounter.com/os-market-share/desktop/worldwide/#monthly-202010-202010-bar

Desktop Operating System Market Share Worldwide (Sept 2021)



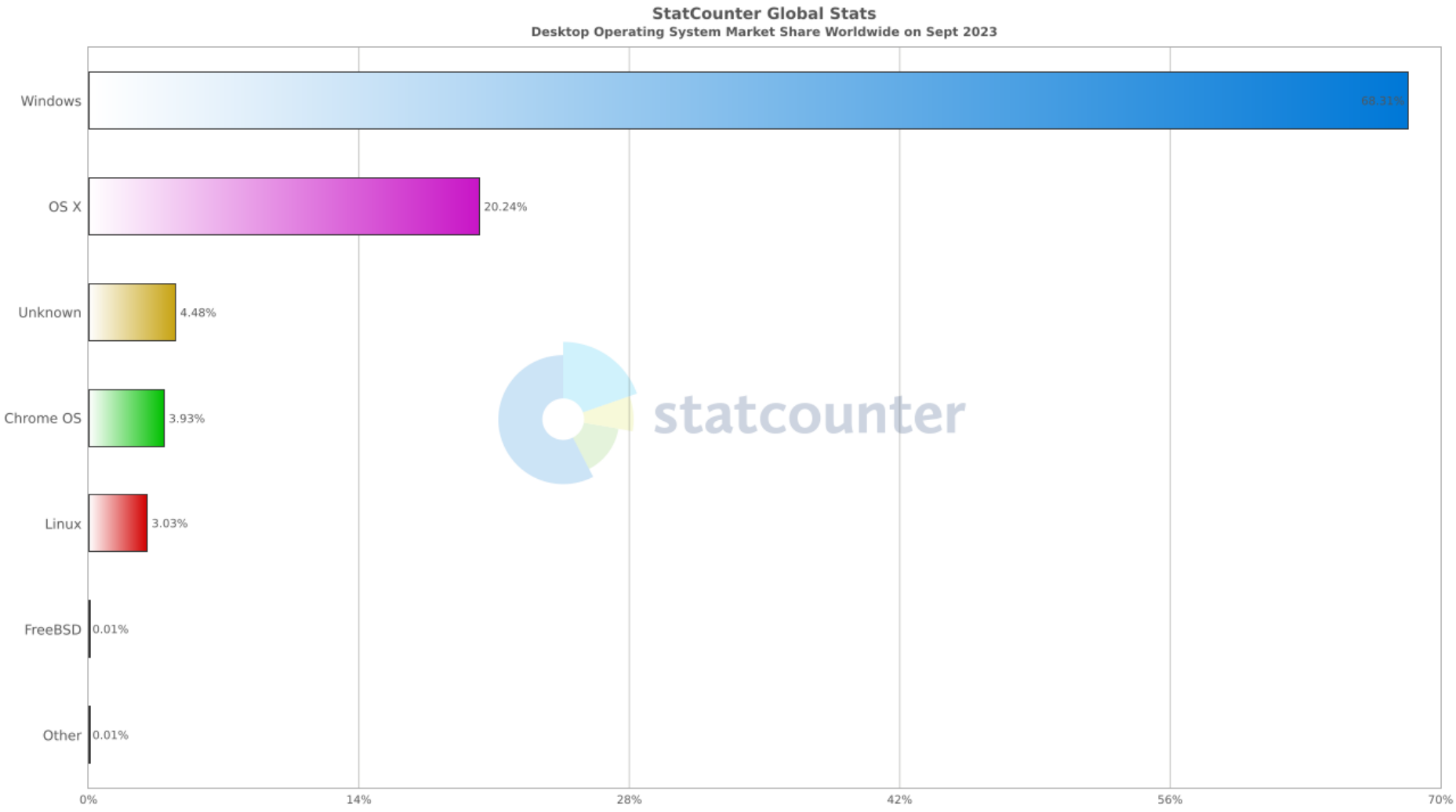
https://gs.statcounter.com/os-market-share/desktop/worldwide/#monthly-202109-202109-bar

Desktop Operating System Market Share Worldwide (Sept 2022)



https://gs.statcounter.com/os-market-share/desktop/worldwide/#monthly-202209-202209-bar

Desktop Operating System Market Share Worldwide (Sept 2023)



https://gs.statcounter.com/os-market-share/desktop/worldwide/#monthly-202309-202309-bar

System Calls and Library Functions

- All operating systems provide "service points" through which programs request services from the kernel
- All implementations of the UNIX System provide a well-defined, limited number of entry points directly into the kernel called *system calls*

- The exact number of system calls varies depending on the operating system version
- Systems have seen incredible growth in the number of supported system calls
  - Version 7 of the Research UNIX System provided about 50 system calls
  - 4.4 BSD provided about 110
  - VR4 had around 120
  - Linux 3.2.0 has 380 system calls and FreeBSD 8.0 has over 450
  - … …

System Calls and Library Functions

- The system call interface has always been documented in Section 2 of the UNIX Programmer's Manual
    - Its definition is in the C language, no matter which implementation technique is actually used on any given system to invoke a system call
    - This differs from many older operating systems, which traditionally defined the kernel entry points in the assembly language of the machine
- The technique used on UNIX systems is for each system call to have a function of the same name in the standard C library (actually, a set if libraries, each with its own header files)

https://en.wikipedia.org/wiki/C_standard_library

System Calls and Library Functions

- Section 3 of the UNIX Programmer's Manual defines the general-purpose library functions available to programmers

- These functions are not entry points into the kernel, although they may invoke one or more of the kernel's system calls

  - For example, the printf(...) function may use the write(...) system call to output a string

  - but the strcpy(...) (copy a string) and atoi(...) (convert ASCII to integer) functions do not involve the kernel at all

System Calls and Library Functions

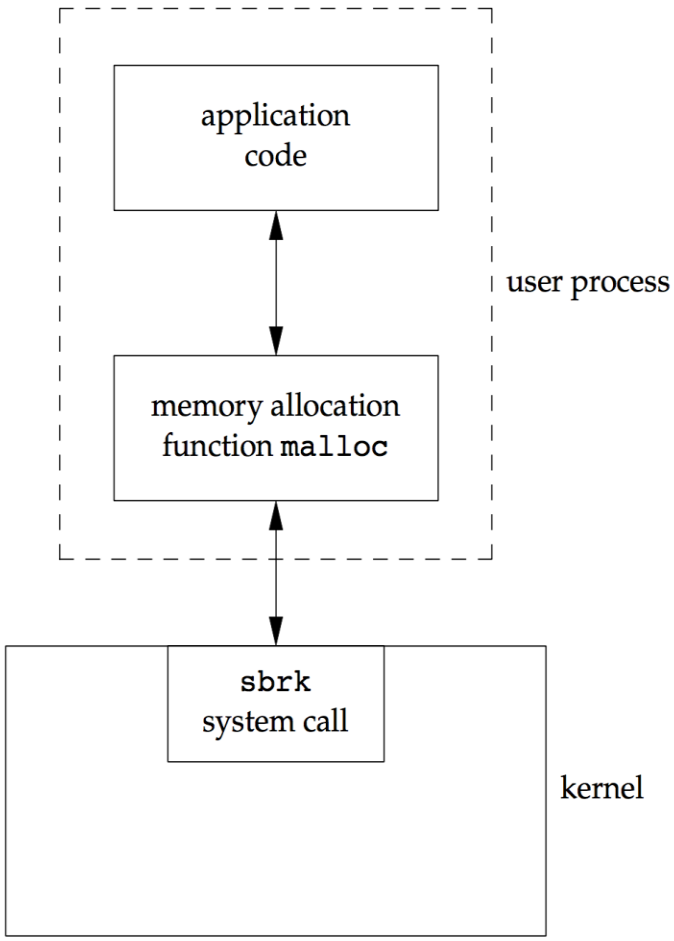


Figure 1.11 Separation of `malloc` function and `sbrk` system call

- The UNIX system call that handles memory allocation, sbrk(2), is not a general-purpose memory manager
  - It increases or decreases the address space of the process by a specified number of bytes
- The memory allocation function, malloc(3), implements one particular type of allocation
  - If we don't like its operation, we can define our own malloc function, which will probably use the sbrk system call
  - In fact, numerous software packages implement their own memory allocation algorithms with the sbrk system call

System Calls and Library Functions

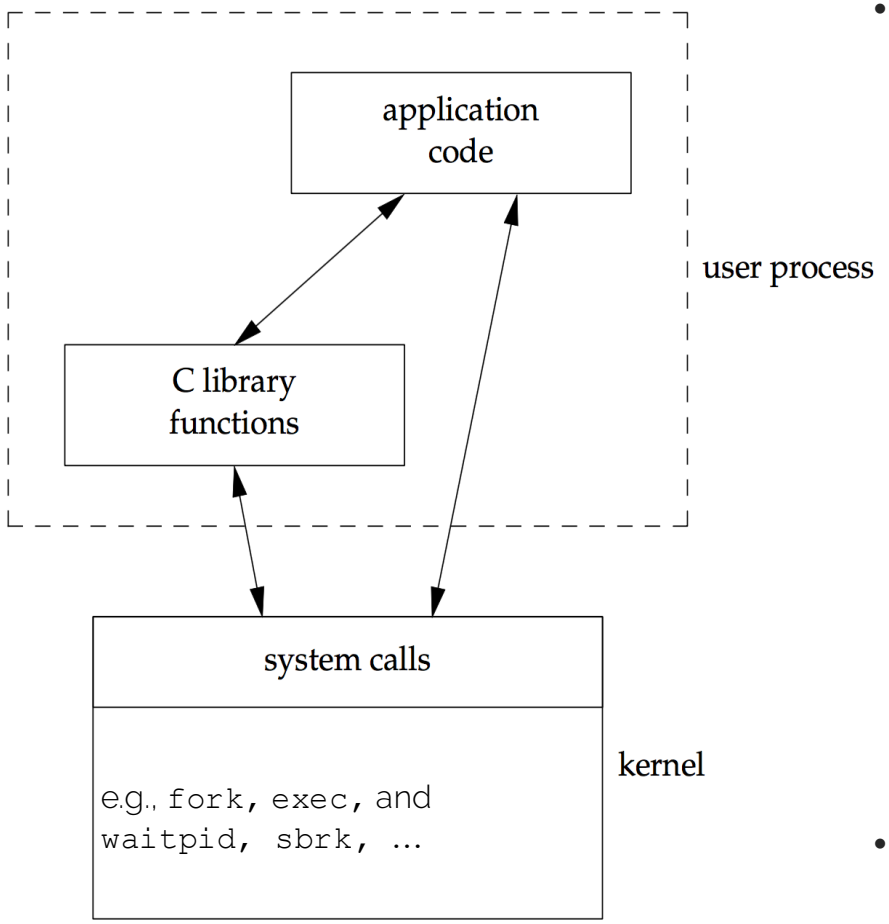


Figure 1.12 Difference between C library functions and system calls

- Another difference between system calls and library functions is that system calls usually provide a minimal interface, whereas library functions often provide more elaborate functionality
  - sbrk(2) VS malloc(3)
  - unbuffered I/O VS standard I/O functions
- Often, the term function refers to both system calls and library functions, except when the distinction is necessary

What about drivers?

- A driver provides a software interface to hardware devices, enabling operating systems and programs to access hardware functions without needing to know precise details about the hardware being used.

- A driver communicates with the device through the computer bus or communications subsystem to which the hardware connects. When a calling program invokes a routine in the driver, the driver issues commands to the device (drives it). Once the device sends data back to the driver, the driver may invoke routines in the original calling program.

- Drivers are hardware dependent and operating system specific. They usually provide the interrupt handling required for any necessary asynchronous time-dependent hardware interface.

https://en.wikipedia.org/wiki/Device_driver

What about drivers?

- A modern computer operating system usually segregates virtual memory into user space and kernel space. Primarily, this separation serves to provide memory protection and hardware protection from malicious or errant software behaviour.

- Kernel space is strictly reserved for running a privileged operating system kernel, kernel extensions (a.k.a. kernel modules or kexts → next slide), and most device drivers.

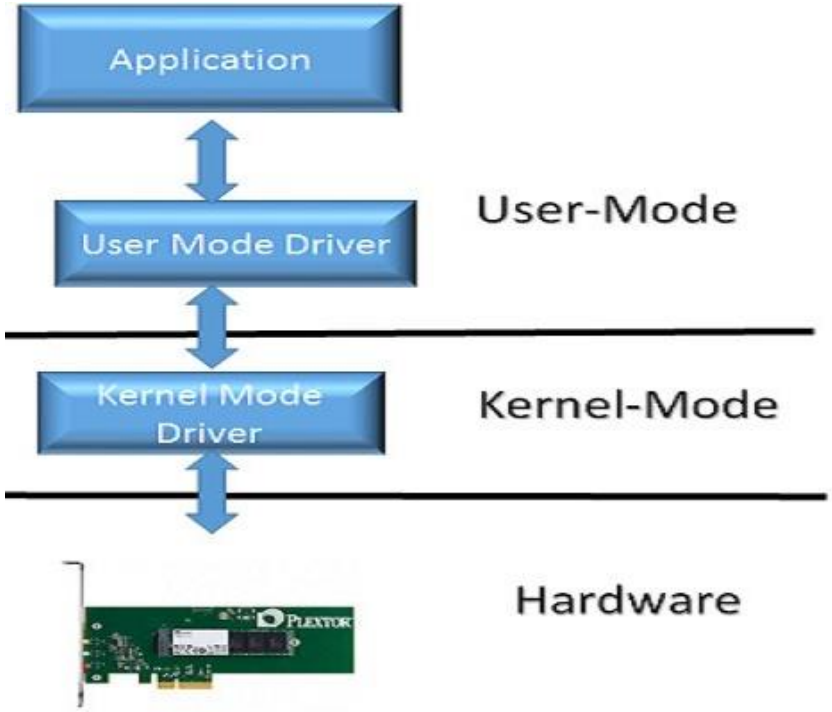
- In contrast, user space is the memory area where application software and some drivers execute.

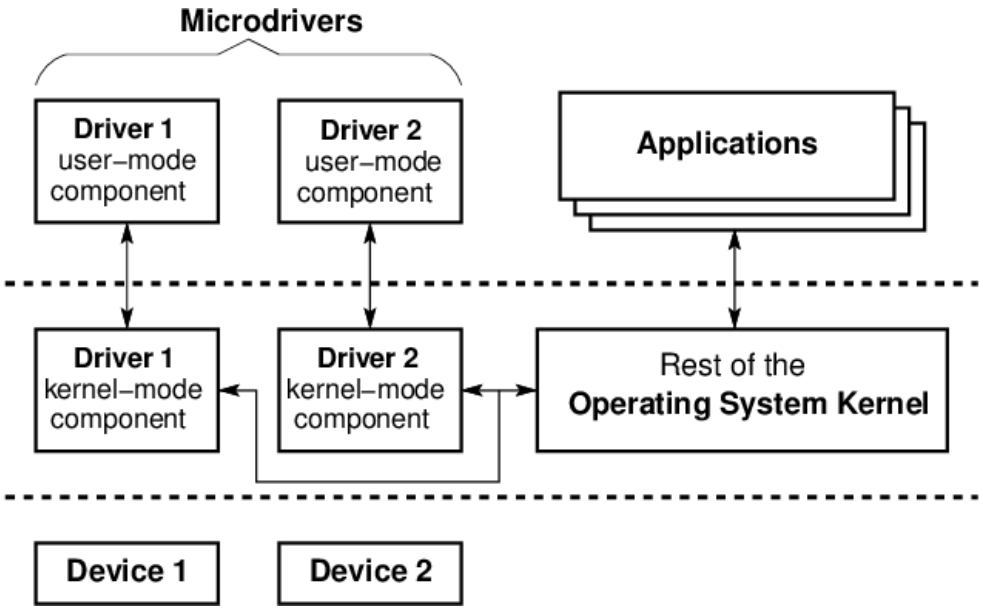
- The term user space (or userland) refers to all code that runs outside the operating system's kernel. User space usually refers to the various programs and libraries that the operating system uses to interact with the kernel: software that performs input/output, manipulates file system objects, application software, etc.

https://en.wikipedia.org/wiki/User_space_and_kernel_space

What about drivers?

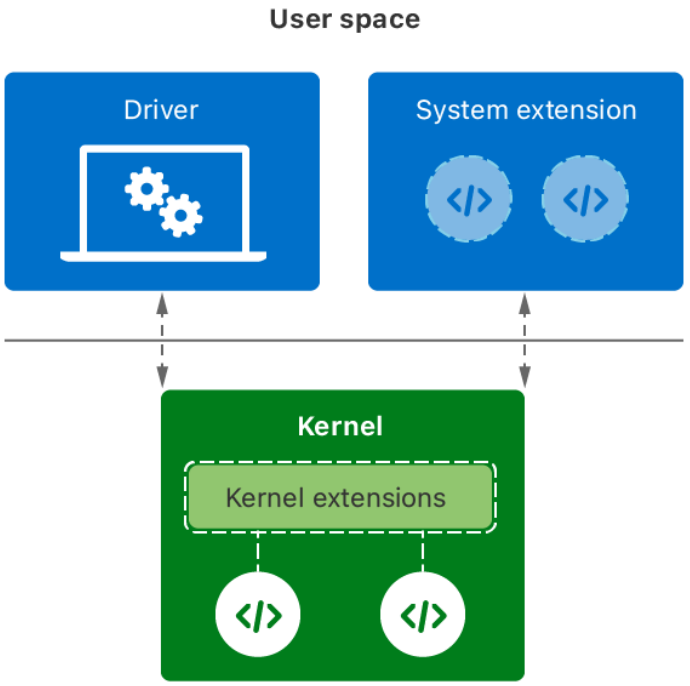
KERNEL MODE VS USER MODE





Kernel extensions VS system extensions on macOS

Kernel extensions (or kexts) let developers load code directly into the macOS kernel. By giving developers these kernel privileges, kexts can help those developers create some very powerful apps. Virtualization applications (such as Parallels Desktop or VMWare Fusion), virtual drive services (such as Box and Google Drive), and other software have long used kexts to integrate their code deeply into the Mac.



But that access has some downsides. While kexts have given developers the freedom to build powerful, complex functions, it also presents them with challenges in terms of development, security, and stability.

To remedy the problems with kexts, Apple introduced system extensions, which provide some similar capabilities to kexts, but in a more controlled environment.

System extensions enable developers to create apps that extend the functionality of macOS without requiring kernel-level access.

https://blog.kandji.io/guide-for-apple-it-macos-system-and-kernel-extensions

OneDrive/Dropbox VS kexts

Apple Confirms macOS 12.3 Deprecates Kernel Extensions Used by Dropbox and OneDrive

Thursday January 27, 2022 10:29 am PST by Joe Rossignol

Apple today seeded the first beta of macOS 12.3 to developers for testing. In the release notes for the update, Apple confirms that it has deprecated kernel extensions used by Dropbox and Microsoft OneDrive and notes that both cloud storage services have replacements for the functionality currently in beta.

Earlier this week, Dropbox announced that users who update to macOS 12.3 may temporarily encounter issues with opening online-only files in some third-party apps on their Mac. Dropbox did not provide a reason for this issue, but it is now clear that it relates to the kernel extensions that enabled this functionality being deprecated by Apple.

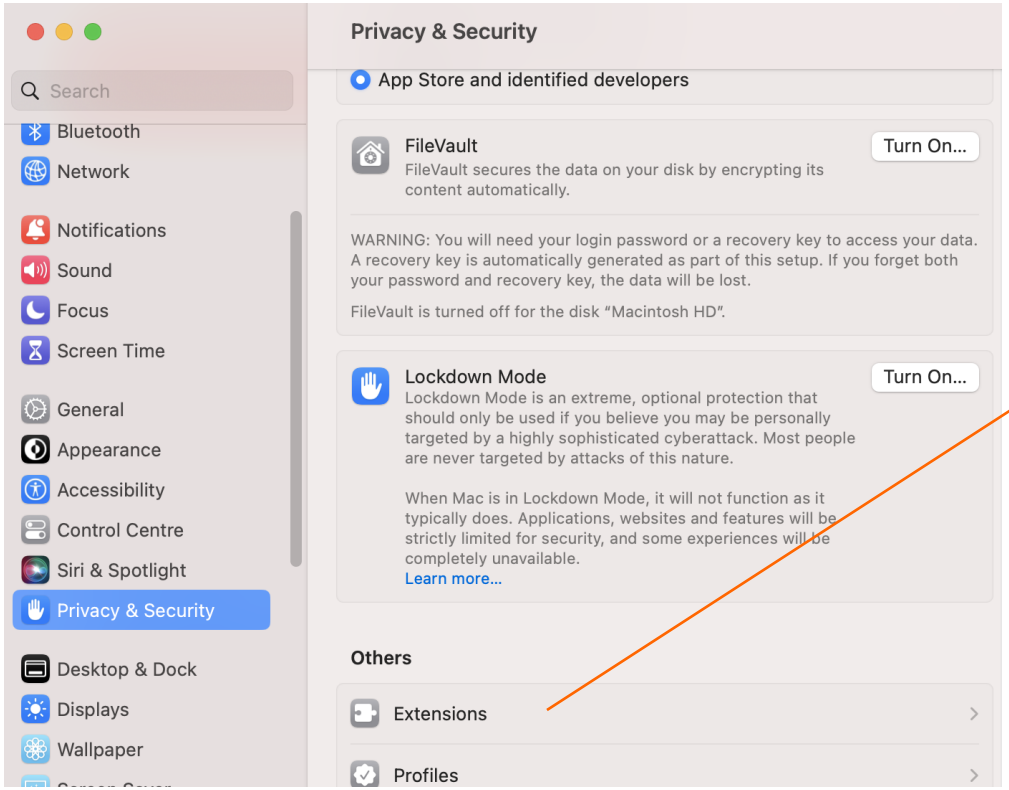
In a support document and an email to customers, Dropbox said it is actively working on full support for online-only files on macOS 12.3 and will begin rolling out an updated version of its Mac app to beta testers in March. In the meantime, Dropbox users who update to macOS 12.3 will still be able to open online-only files in Finder.

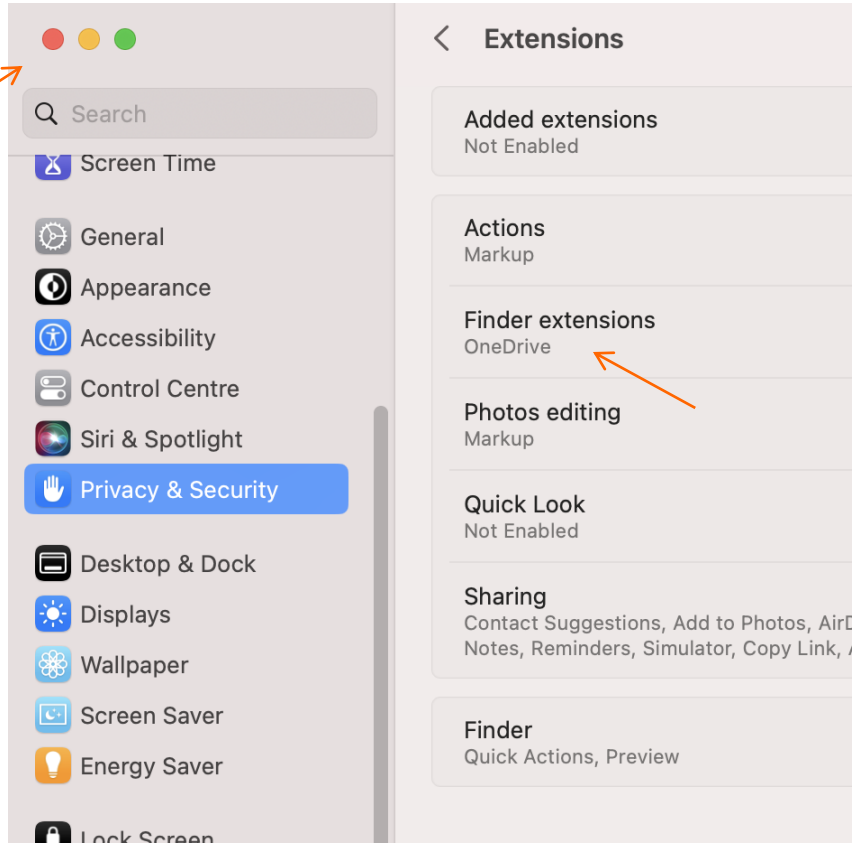
Microsoft is also working on a new online-only files experience for OneDrive that is "more integrated with macOS" and "will have long-term support from Apple."

Tags: Dropbox, OneDrive
Related Forum: macOS Monterey

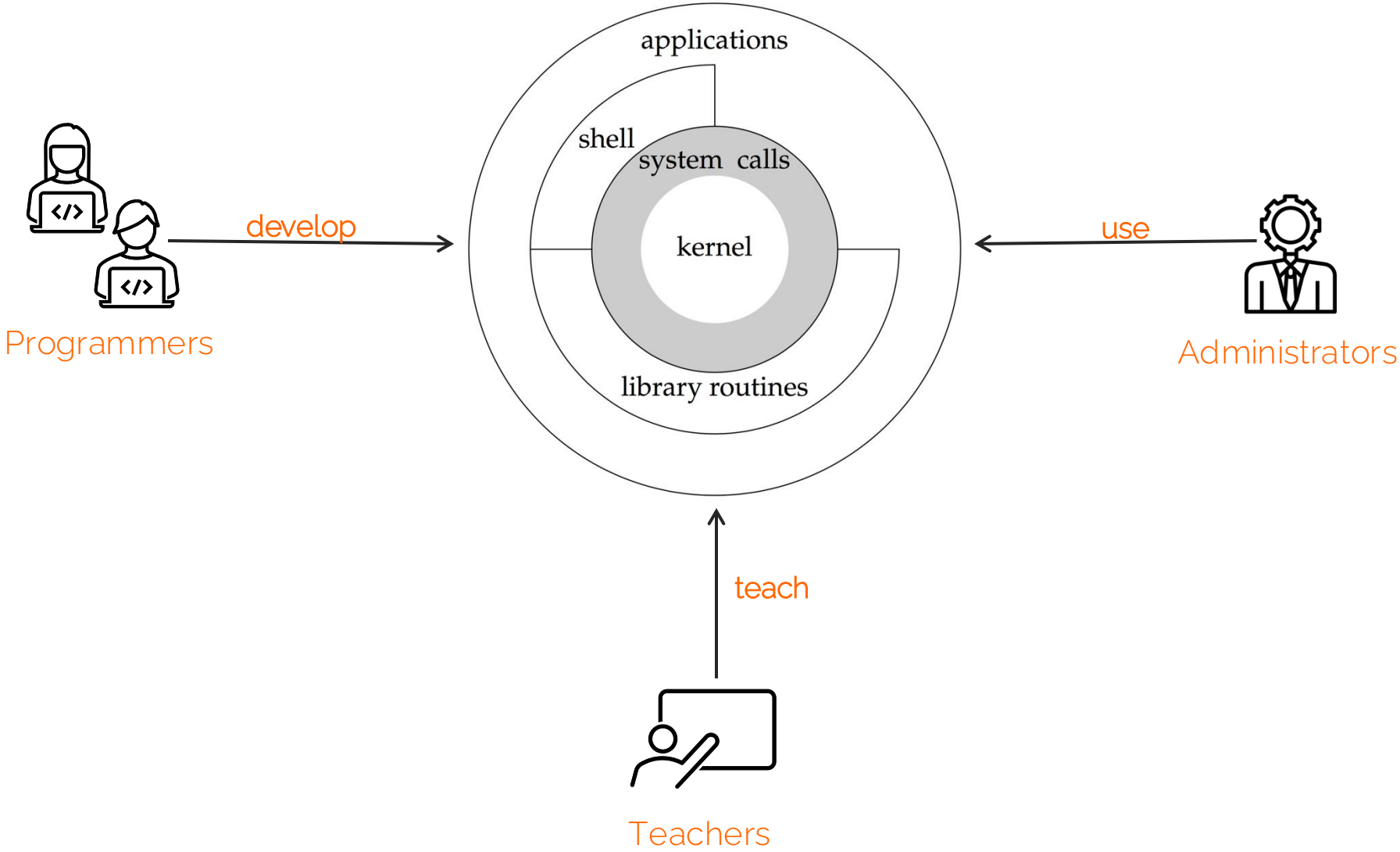
https://www.macrumors.com/2022/01/27/macos-12-3-deprecates-dropbox-onedrive-kexts/

More on…

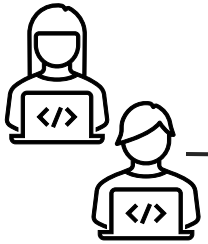

Privacy & Security
Search
Bluetooth
Network
Notifications
Sound
Focus
Screen Time
General
Appearance
Accessibility
Control Centre
Siri & Spotlight
Privacy & Security
Desktop & Dock
Displays
Wallpaper
App Store and identified developers
FileVault
FileVault secures the data on your disk by encrypting its content automatically.
Turn On...
WARNING: You will need your login password or a recovery key to access your data. A recovery key is automatically generated as part of this setup. If you forget both your password and recovery key, the data will be lost.
FileVault is turned off for the disk "Macintosh HD".
Lockdown Mode
Lockdown Mode is an extreme, optional protection that should only be used if you believe you may be personally targeted by a highly sophisticated cyberattack. Most people are never targeted by attacks of this nature.
When Mac is in Lockdown Mode, it will not function as it typically does. Applications, websites and features will be strictly limited for security, and some experiences will be completely unavailable.
Learn more...
Turn On...
Others
Extensions
Profiles


Search
Screen Time
General
Appearance
Accessibility
Control Centre
Siri & Spotlight
Privacy & Security
Desktop & Dock
Displays
Wallpaper
Screen Saver
Energy Saver
Extensions
Added extensions
Not Enabled
Actions
Markup
Finder extensions
OneDrive
Photos editing
Markup
Quick Look
Not Enabled
Sharing
Contact Suggestions, Add to Photos, AirD
Notes, Reminders, Simulator, Copy Link,
Finder
Quick Actions, Preview

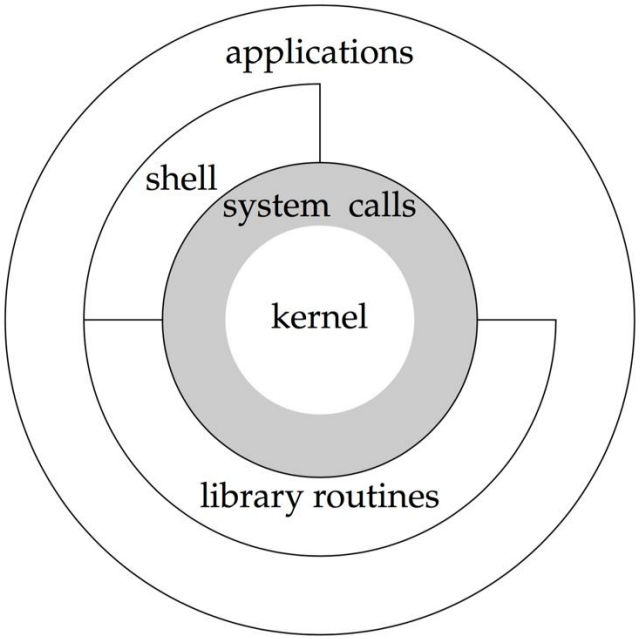
OPS stakeholders



OPS stakeholders



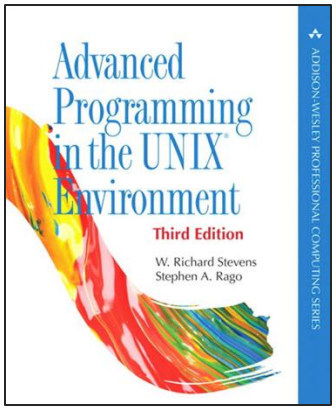
develop



use

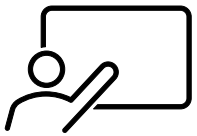
Programmers

Administrators



our reference textbook

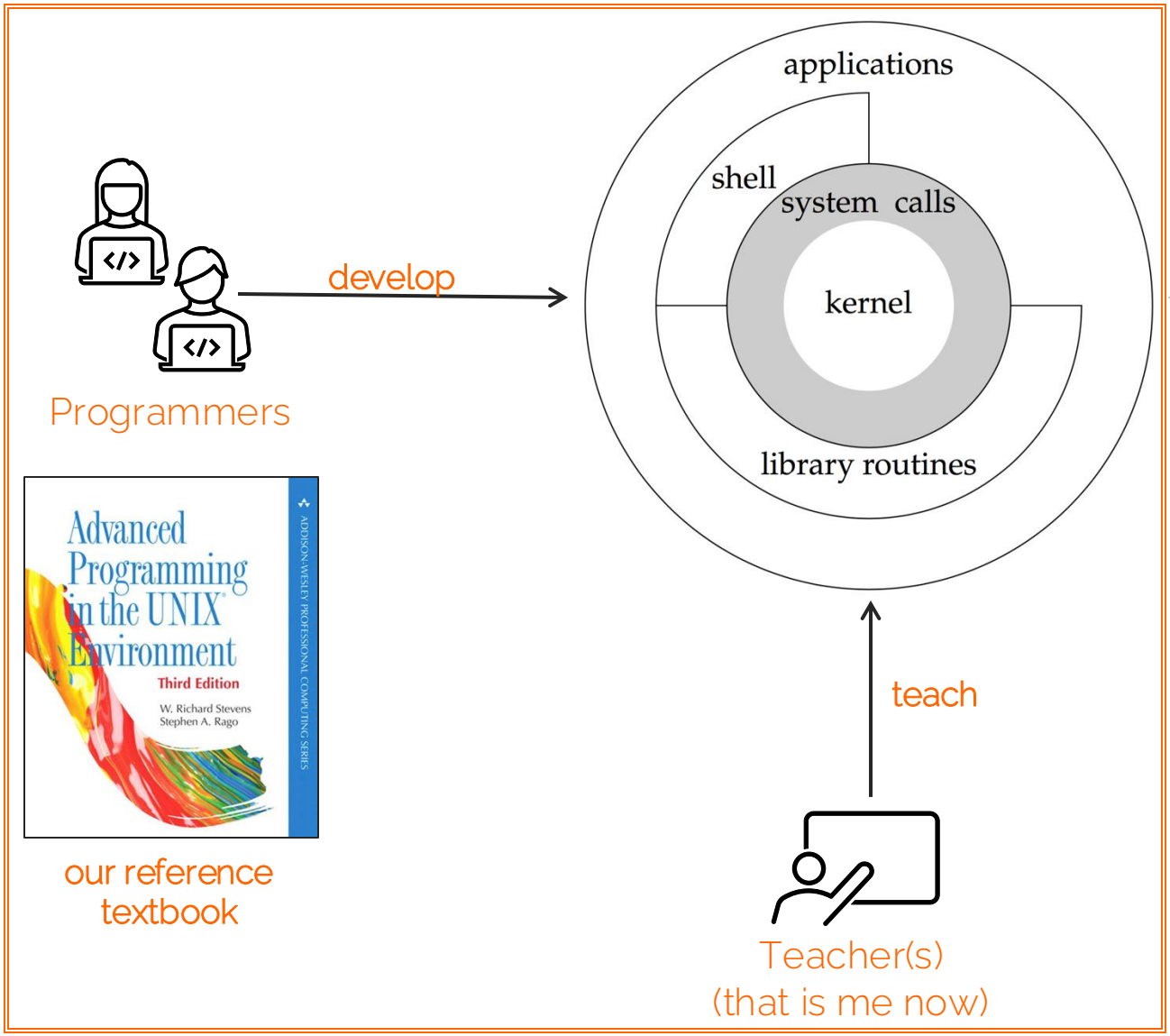
teach



Teacher(s)
(that is me now)

e.g., a user manual for administrators

Teaching OPS*Lab* (this course)



POSIX

- Portable Operating System Interface (POSIX) is a family of standards specified by the IEEE Computer Society for maintaining compatibility between operating systems

- POSIX defines the Application Programming Interface (API), along with command line shells and utility interfaces, for software compatibility with variants of Unix and other operating systems

- Richard Stallman suggested the name POSIX (📣 pronounced as pahz-icks, not as poh-six) to the IEEE instead of former IEEE-IX. The committee found it more easily pronounceable and memorable, and thus adopted it

https://en.wikipedia.org/wiki/POSIX

User IDentification (intro)

User ID

- The user ID from our entry in the password file is a numeric value that identifies us to the system (see next slide)
- It is assigned by the system administrator when our login name is assigned, and we cannot change it
- The user ID is normally assigned to be unique for every user
- The kernel uses the user ID to check whether we have the appropriate permissions to perform certain operations

Group ID

- It is assigned by the system administrator when our login name is assigned
- Typically, the password file contains multiple entries that specify the same group ID
- Groups are normally used to collect users together into, e.g., projects or departments. This allows the sharing of resources, such as files, among members of the same group (e.g., members of a department)

Supplementary Group IDs

- In addition to the group ID specified in the password file for a login name, most versions of the UNIX System allow a user to belong to more than one groups

Logging in

- Password file: /etc/passwd (usually)

sar:✷:205:105:Stephen Rago:/home/sar:/bin/ksh

- seven colon-separated fields
  - the login name (sar), encrypted password (✷), numeric user ID (205), numeric group ID (105), a comment field (Stephen Rago), home directory (/home/sar), and shell program (/bin/ksh)

- All contemporary systems have moved the encrypted password to a different file (see Chapter 6 of the reference book for more details)

Information	FreeBSD 8.0	Linux 3.2.0	Mac OS X 10.6.8	Solaris 10
account information	`/etc/passwd`	`/etc/passwd`	Directory Services	`/etc/passwd`
encrypted passwords	`/etc/master.passwd`	`/etc/shadow`	Directory Services	`/etc/shadow`
hashed password files?	yes	no	no	no
group information	`/etc/group`	`/etc/group`	Directory Services	`/etc/group`

Passwd file on my macOS Catalina

Single user mode (also referred to as maintenance mode and runlevel 1) is a mode of operation of a computer running Linux or another Unix-like operating system that provides as few services as possible and only minimal functionality

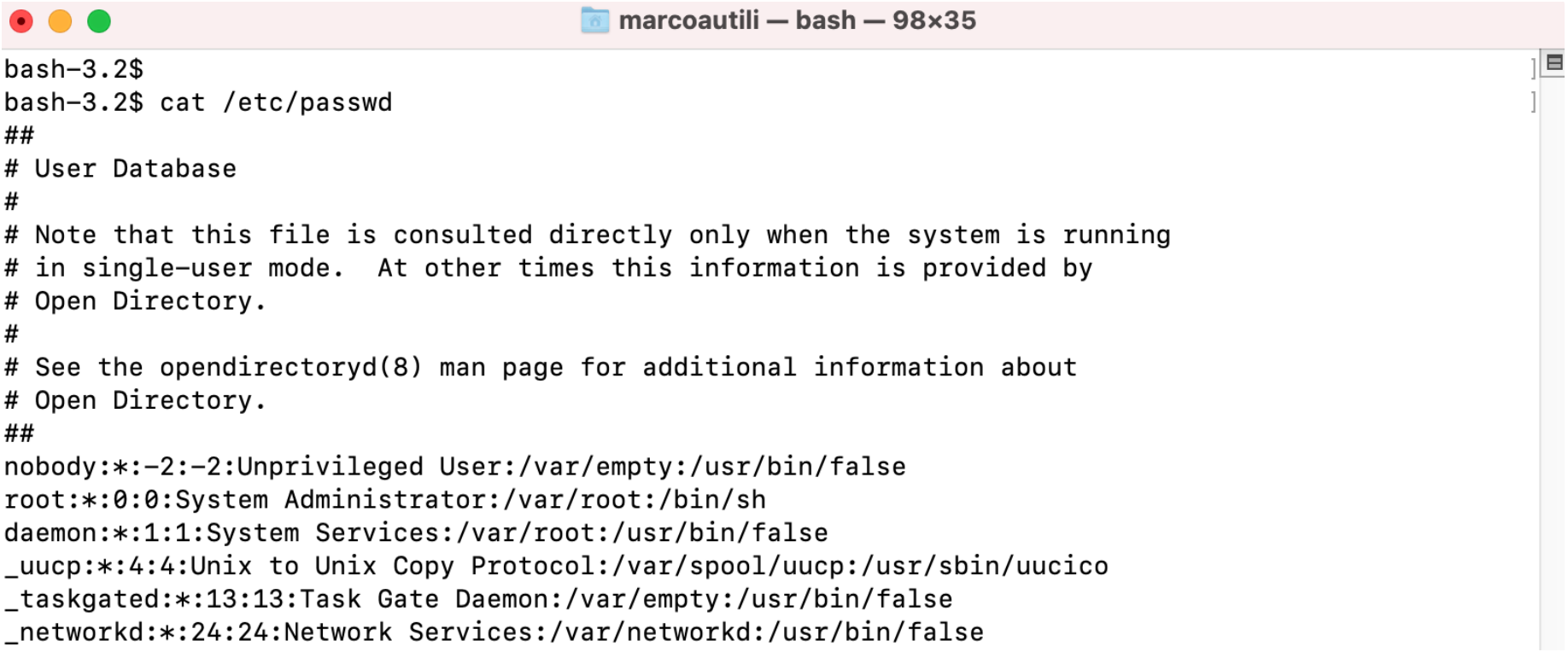
Untitled (passwd copy) — Edited

```
##
# User Database
#
# Note that this file is consulted directly only when the system is running
# in single-user mode.  At other times this information is provided by
# Open Directory.
#
# See the opendirectoryd(8) man page for additional information about
# Open Directory.
##
nobody:*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false
root:*:0:0:System Administrator:/var/root:/bin/sh
daemon:*:1:1:System Services:/var/root:/usr/bin/false
_uucp:*:4:4:Unix to Unix Copy Protocol:/var/spool/uucp:/usr/sbin/uucico
_taskgated:*:13:13:Task Gate Daemon:/var/empty:/usr/bin/false
```

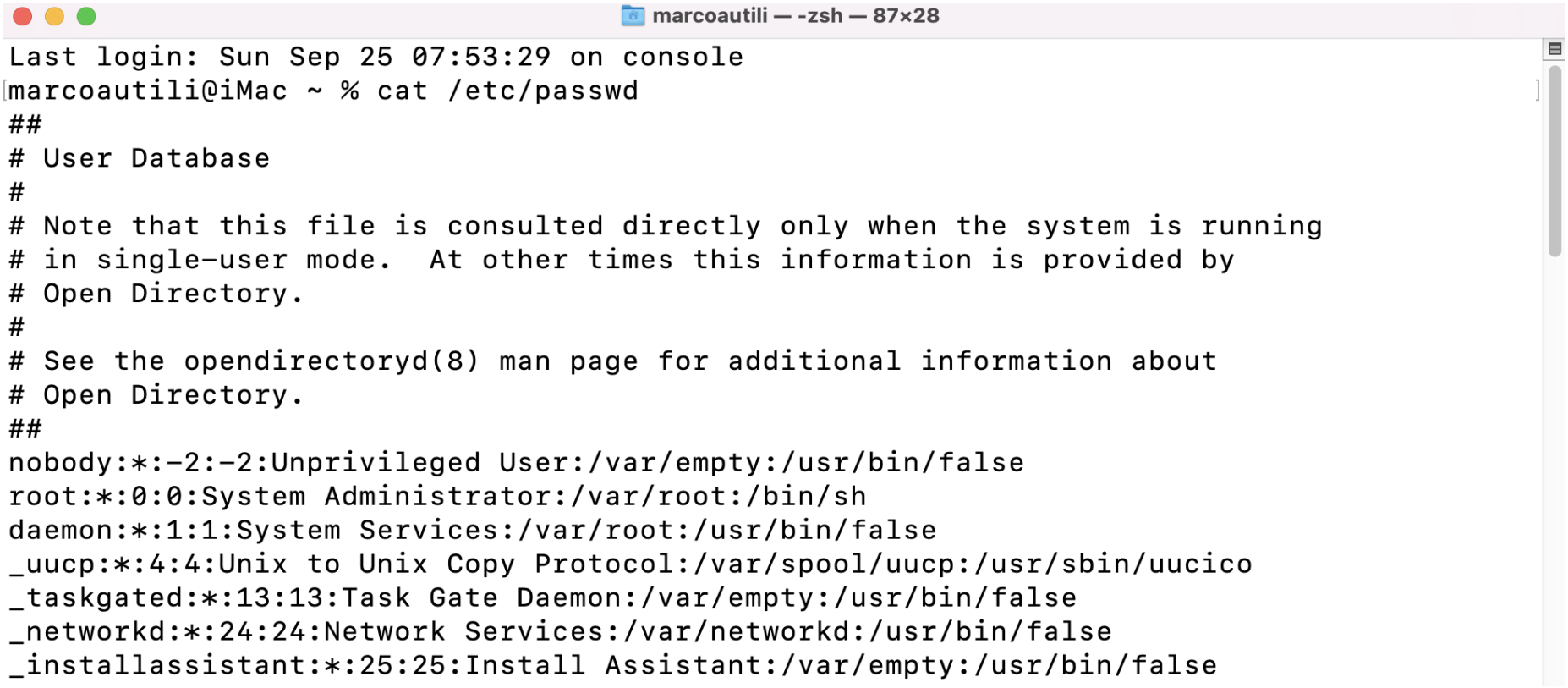
See man opendirectoryd

It replaces "DirectoryService" ...

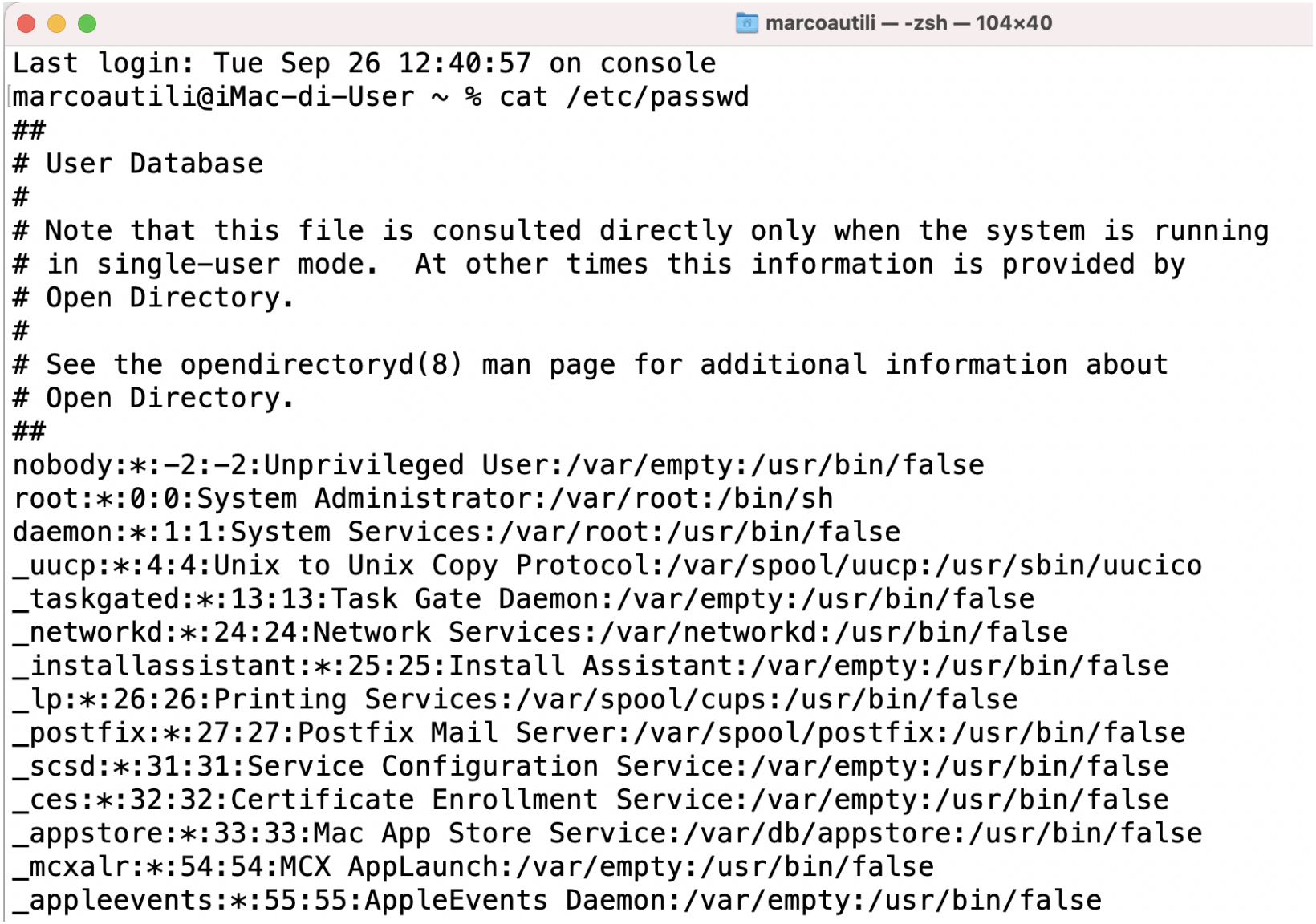
Passwd file on my macOS Big Sur



Passwd file on my macOS Monterey



Passwd file on my macOS Ventura



Shells

- A shell is a command-line interpreter that reads user input and executes commands

- The user input to a shell is normally from the terminal (an *interactive shell*) or sometimes from a file (called a *shell script*)

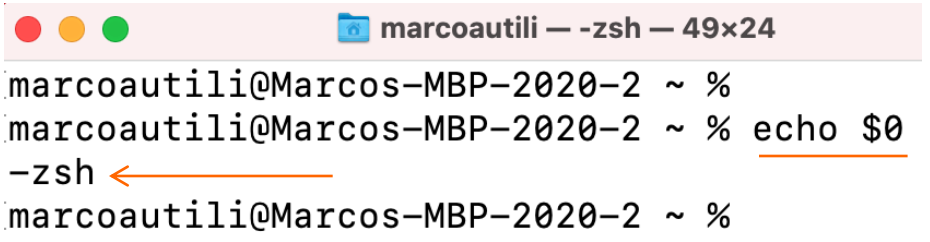
sar:✶:205:105:Stephen Rago:/home/sar:/bin/ksh

Name	Path	FreeBSD 8.0	Linux 3.2.0	Mac OS X 10.6.8	Solaris 10
Bourne shell	`/bin/sh`	•	•	copy of `bash`	•
Bourne-again shell	`/bin/bash`	optional	•	•	•
C shell	`/bin/csh`	link to `tcsh`	optional	link to `tcsh`	•
Korn shell	`/bin/ksh`	optional	optional	•	•
TENEX C shell	`/bin/tcsh`	•	optional	•	•

Online shell:
http://learnshell.org

See next slide

- Default shell in previous versions of macOS
- In Catalina, Big Sure, Monterey and Ventura it is **zsh**



http://learnshell.org

Code

Run | Reset | Solution

```
1  #!/bin/bash
2  # Welcome to the Interactive Shell Tutorial.
3  # Start by choosing a chapter and
4  # write your code in this window.
5
6  echo "Hello, World!";
7  echo
8  echo $0;
9  echo
10
11 cat /etc/passwd
12
```

Output

Expected Output

```
Hello, World!

./prog.sh

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
```

Shell types (some of)

Just like people know different languages and dialects, your UNIX-like systems usually offer a variety of shell types

- sh or Bourne Shell: the original shell still used on UNIX systems and in UNIX-related environments. This is the basic shell, a small program with few features. While this is not the standard shell, it is still available on every Linux system for compatibility with UNIX programs
- bash or Bourne Again shell: the standard GNU shell, intuitive and flexible. Probably most advisable for beginning users while being at the same time a powerful tool for the advanced and professional user. On Linux, bash is the standard shell for common users. This shell is a so-called *superset* of the Bourne shell, a set of add-ons and plug-ins. This means that the Bourne Again shell is compatible with the Bourne shell: commands that work in sh, also work in bash. However, the reverse is not always the case. All examples and exercises in this book use bash
- csh or C shell: the syntax of this shell resembles that of the C programming language. Sometimes asked for by programmers
- tcsh or TENEX C shell: a superset of the common C shell, enhancing user-friendliness and speed. That is why some also call it the Turbo C shell
- ksh or the Korn shell: sometimes appreciated by people with a UNIX background. A superset of the Bourne shell; with standard configuration a nightmare for beginning users
- zsh the Z shell (Zsh) is a Unix shell that can be used as an interactive login shell and as a command interpreter for shell scripting. Zsh is an extended Bourne shell with many improvements, including some features of Bash, ksh, and tcsh

The file /etc/shells gives an overview of known shells on a Unix-like system (see next slide)

Switching among shells

```
marcoautili — bash — 80×28
Last login: Fri Sep  9 12:39:22 on ttys001
iMac-2:~ marcoautili$ echo $0
-bash
iMac-2:~ marcoautili$ sh
sh-3.2$ echo $0
sh
sh-3.2$ ksh
$ echo $0
ksh
$ bash
bash-3.2$ echo $0
bash
bash-3.2$ $0
bash-3.2$ cat /etc/shells
# List of acceptable shells for chpass(1).
# Ftpd will not allow users to connect who are not using
# one of these shells.

/bin/bash
/bin/csh
/bin/ksh
/bin/sh
/bin/tcsh
/bin/zsh
bash-3.2$
```

The shell I am running (try also **echo $SHELL**)

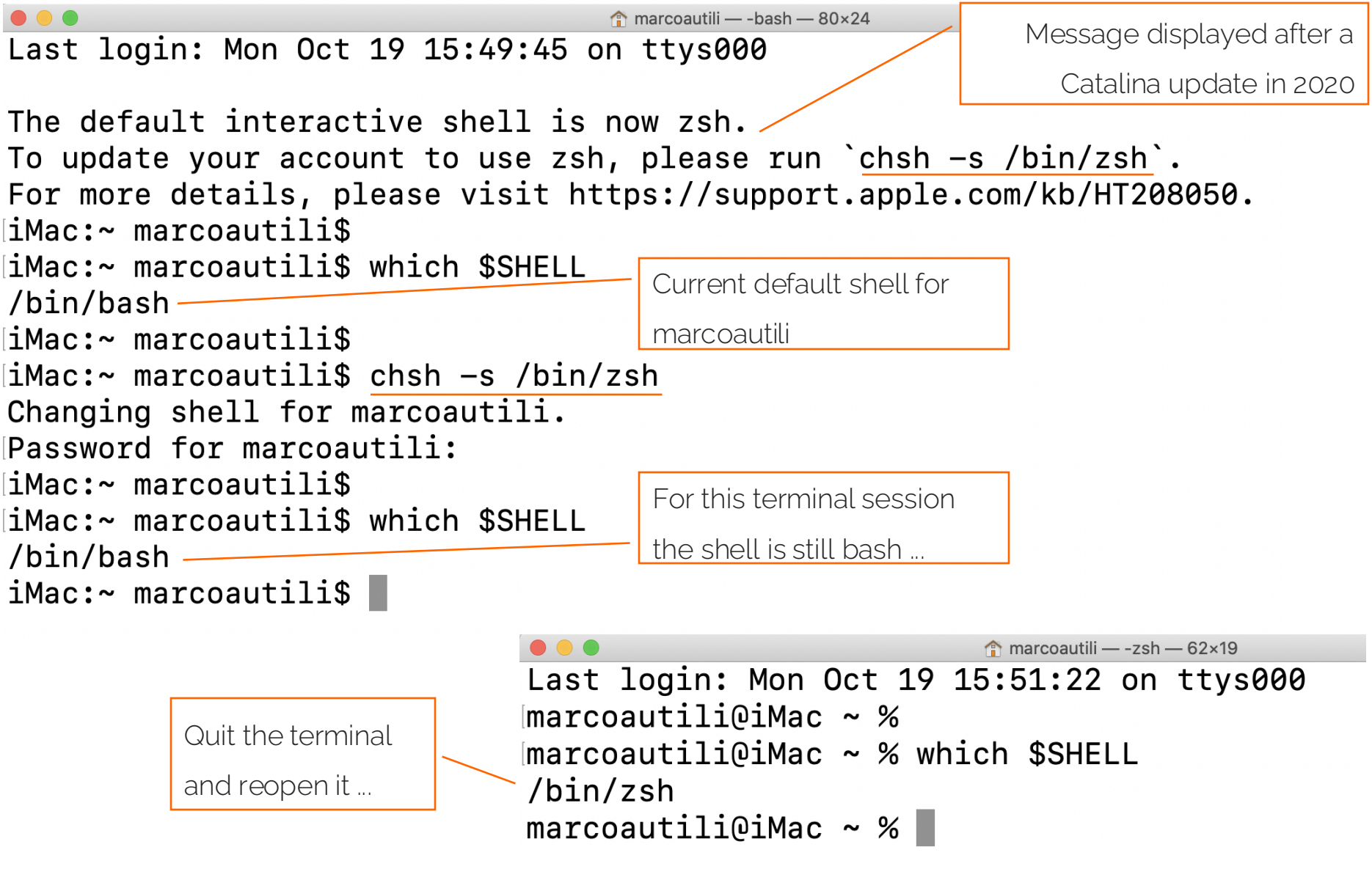
Switch to the **sh** shell

- The chpass utility allows editing of the user database information associated with user or, by default, the current user
- Curiosity:
  - on FreeBSD the chfn, chsh, ypchpass, ypchfn and ypchsh utilities behave identically to chpass
  - on MacOS the chfn, chsh utilities behave identically to chpass (on both systems there is only one program)

https://www.freebsd.org/cgi/man.cgi?query=chpass&sektion=1

- The notation chpass(1) is the normal way to reference a particular entry in the UNIX system manuals
- It refers to the entry for chpass in Section 1
- Sections are normally numbered 1 through 8, and all the entries within each section are arranged alphabetically
- On the terminal type
  - man 1 chpass
  - man s1 chpass

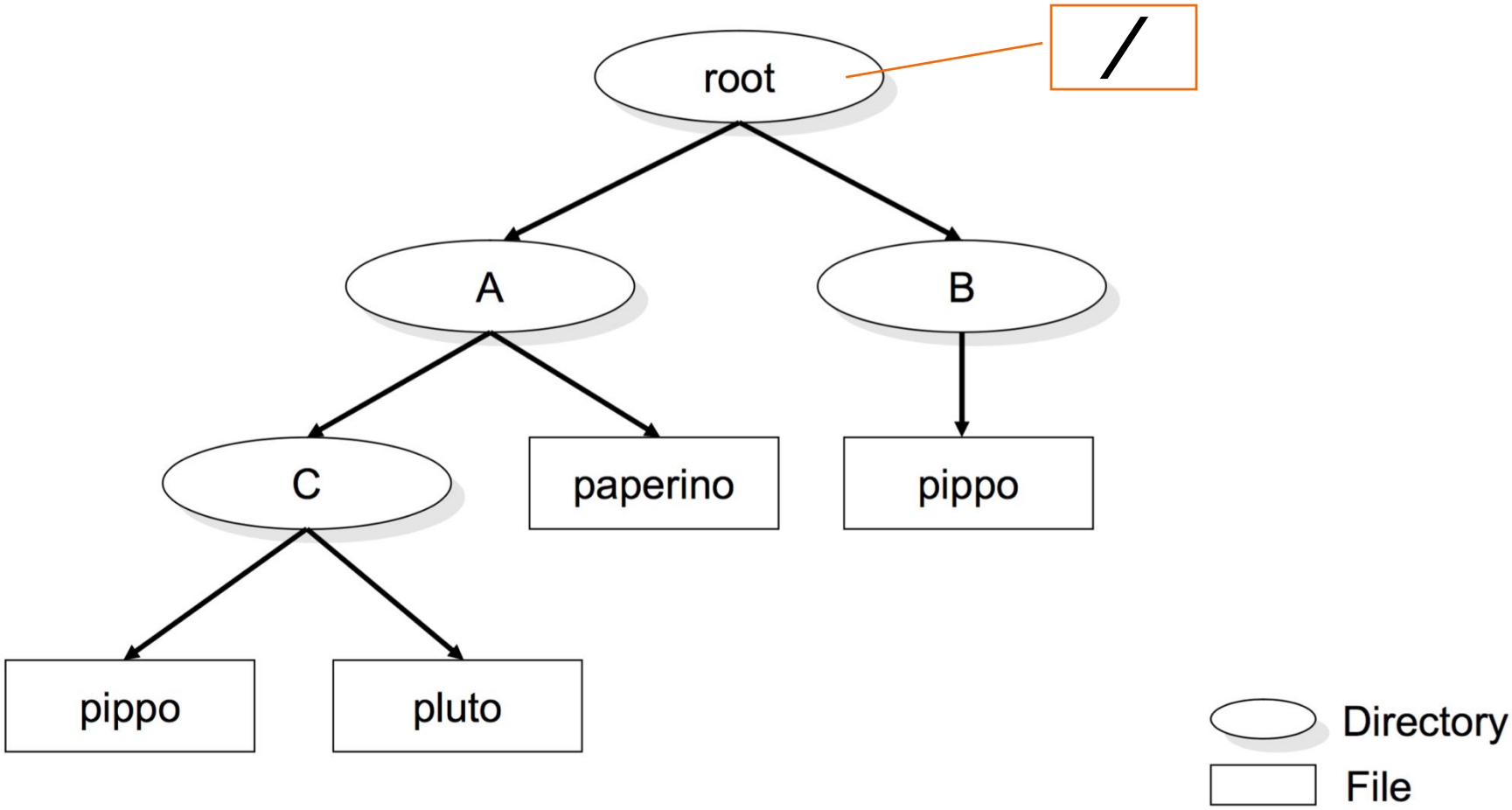
Changing my default shell



UNIX file system

- Hierarchical arrangement of directories and files
- The root directory is named "/"
- A directory is a file that contains directory entries (Chapter 4)
- Logically, each entry contains a filename along with the attributes of the file, such as
  - type of file (regular file, directory), the size of the file, the owner of the file, permissions for the file (whether other users may access this file), and when the file was last modified
  - stat and fstat return a structure of information containing all the attributes of a file
- Two filenames are automatically created whenever a new directory is created
  - "." dot (current directory)
  - ".." dot-dot (parent directory)

UNIX file system



Filename

- The names in a directory are called filenames

- The only two characters that cannot appear in a filename are the slash character (/) and the null character (see next slide)
  - The slash separates the filenames that form a pathname (described next)
  - the null character terminates a pathname

- It's good practice to restrict the characters in a filename to a subset of the normal printing characters
  - If we use some of the shell's special characters in the filename, we must use the shell's quoting mechanism to reference the filename, and this can get complicated
  - For portability, POSIX.1 recommends restricting filenames to consist of the following characters: letters (a-z, A-Z), numbers (0-9), period (.), dash (-), and underscore (_)

Good to know…

- In C library and Unix conventions, the null character is used to terminate text strings
- Such null-terminated strings can be known in abbreviation as ASCIZ or ASCIIZ (📣 ass-kee), where here Z stands for "zero"

Caret notation is often used to represent control characters on a terminal (e.g., keyboard) → See next slide

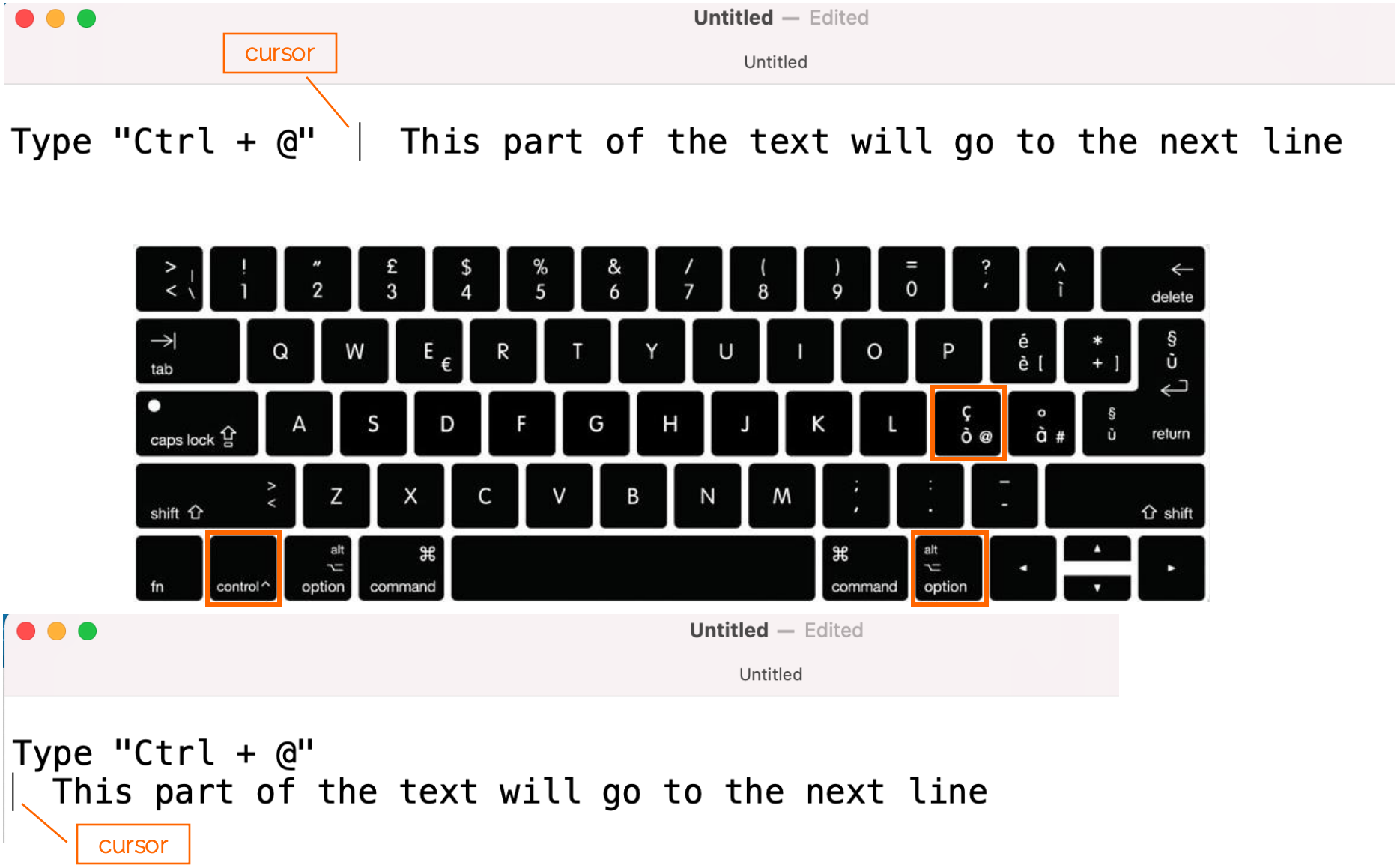
Character escape sequences used in C programming language and many other languages influenced by it, such as Java and Perl

Binary	Oct	Dec	Hex	Abbreviation			[b]	[c]	[d]	Name ('67)
				'63	'65	'67				
000 0000	000	0	00	NULL	NUL		NUL	^@	\0	Null
000 0001	001	1	01	SOM	SOH		SOH	^A		Start of Heading
000 0010	002	2	02	EOA	STX		STX	^B		Start of Text
000 0011	003	3	03	EOM	ETX		ETX	^C		End of Text
000 0100	004	4	04	EOT			EOT	^D		End of Transmission

- The inherent ambiguity of many control characters, combined with their historical usage, created problems when transferring "plain text" files between systems
- The best example of this is the newline problem on various operating systems

https://en.wikipedia.org/wiki/ASCII

Try with a plain text editor (e.g., TextEdit)



Pathname

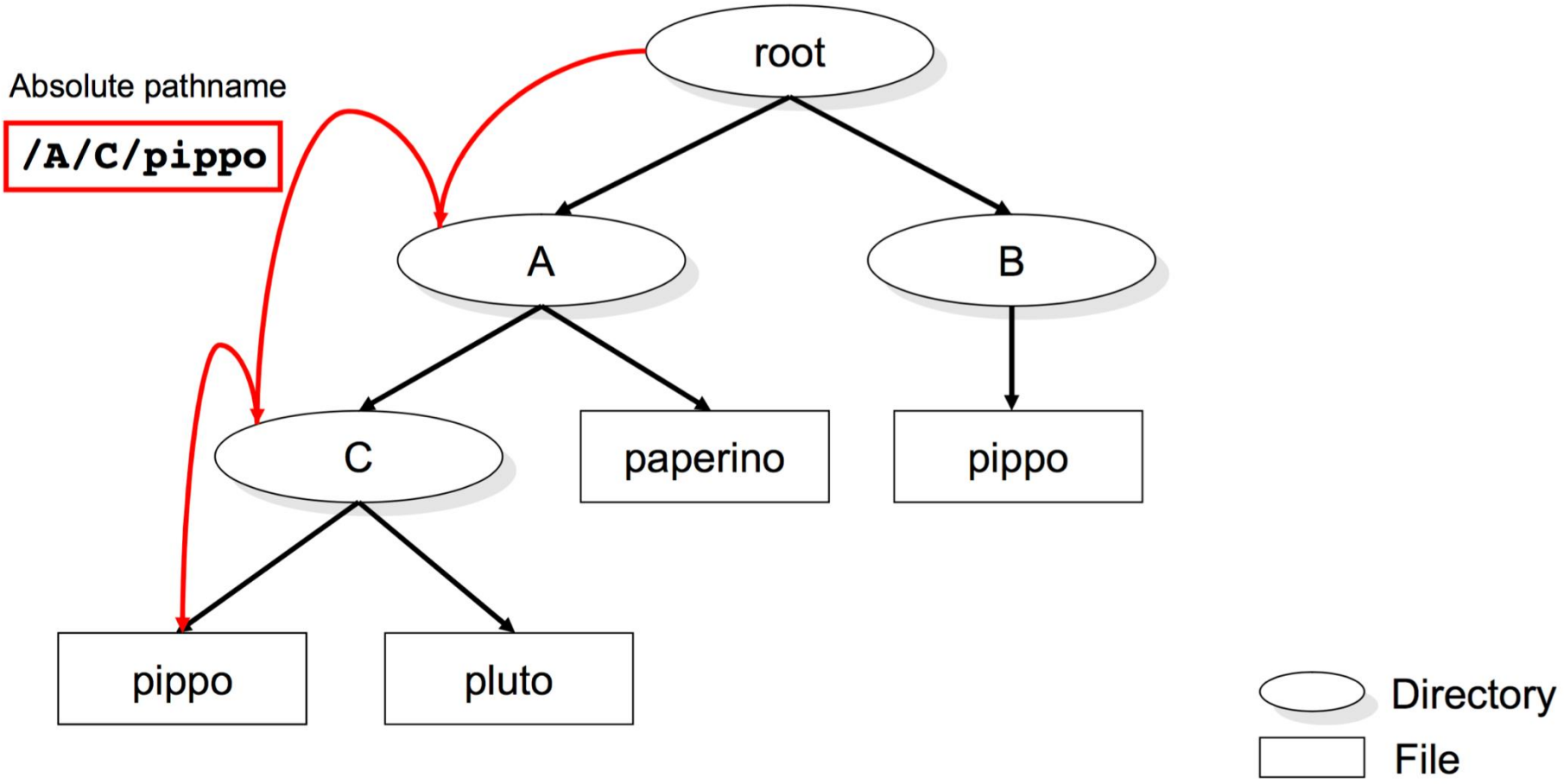
A pathname is a sequence of one or more filenames, separated by slashes and optionally starting with a slash

- A pathname that begins with a slash is called an absolute pathname, otherwise, it is called a relative pathname

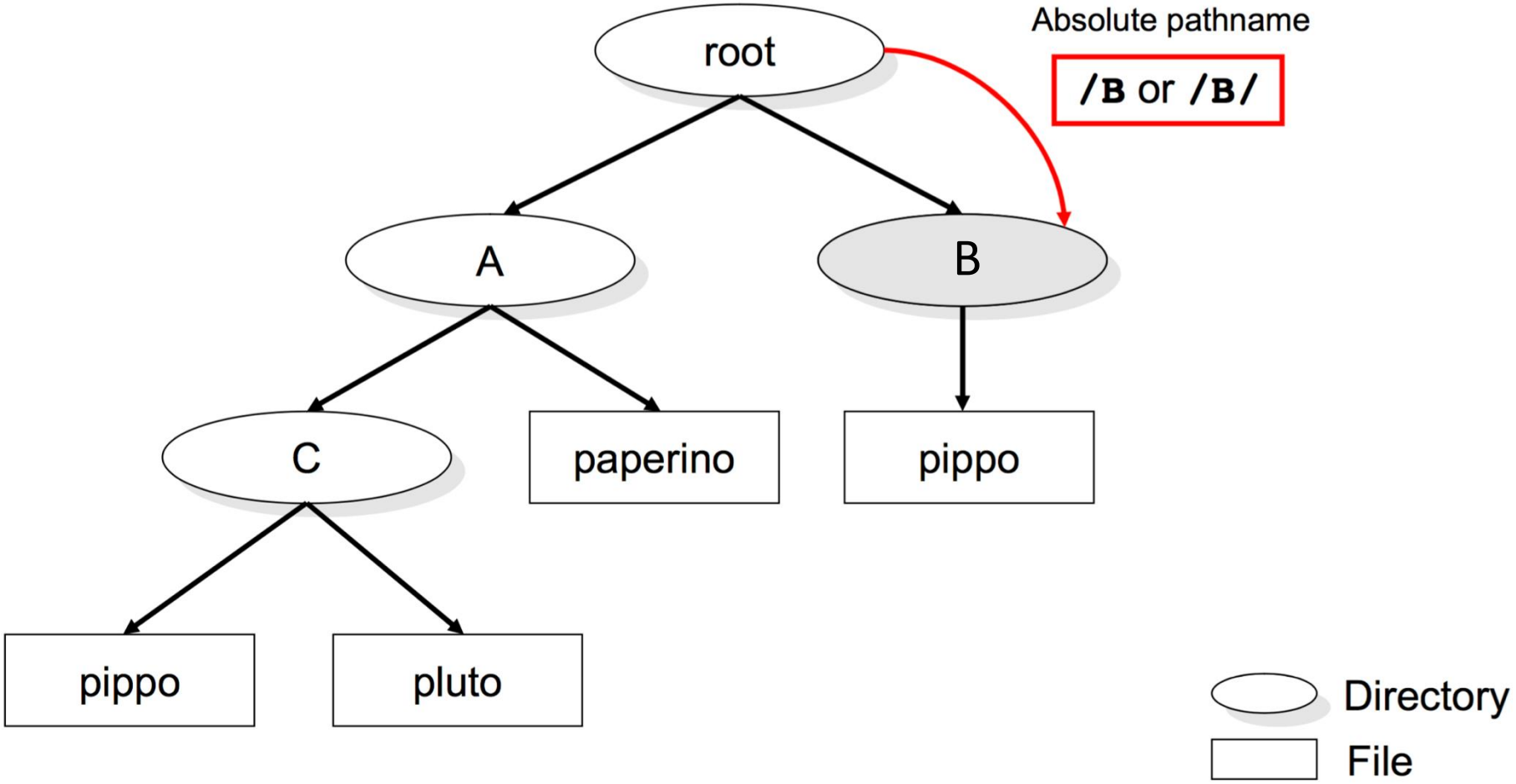
- Relative pathnames refer to files relative to the current directory

- The name for the root of the file system (/) is a special-case absolute pathname that has no filename component

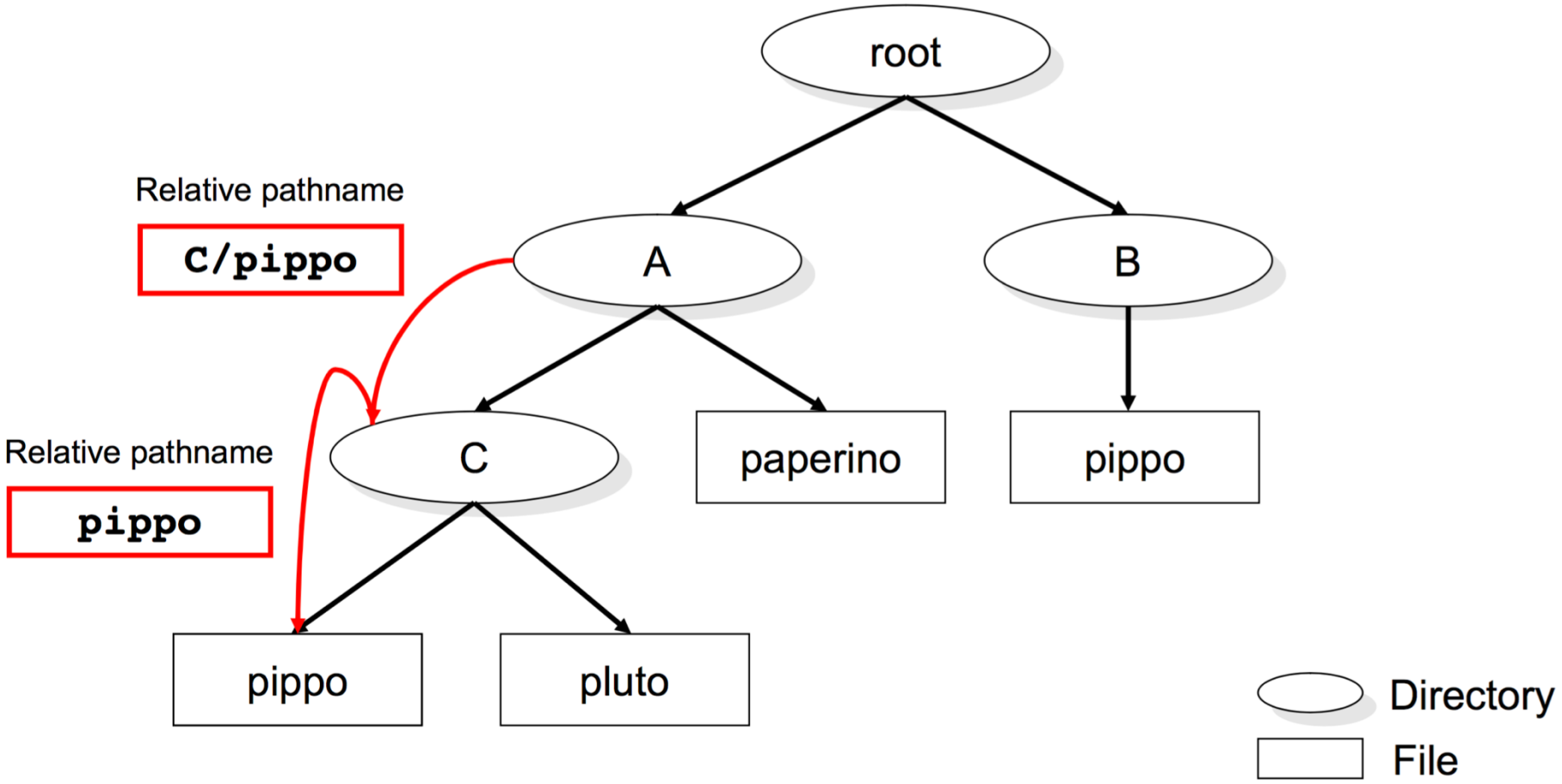
Pathname


Absolute pathname
/A/C/pippo
root
A
B
C
paperino
pippo
pippo
pluto
Directory
File

Pathname

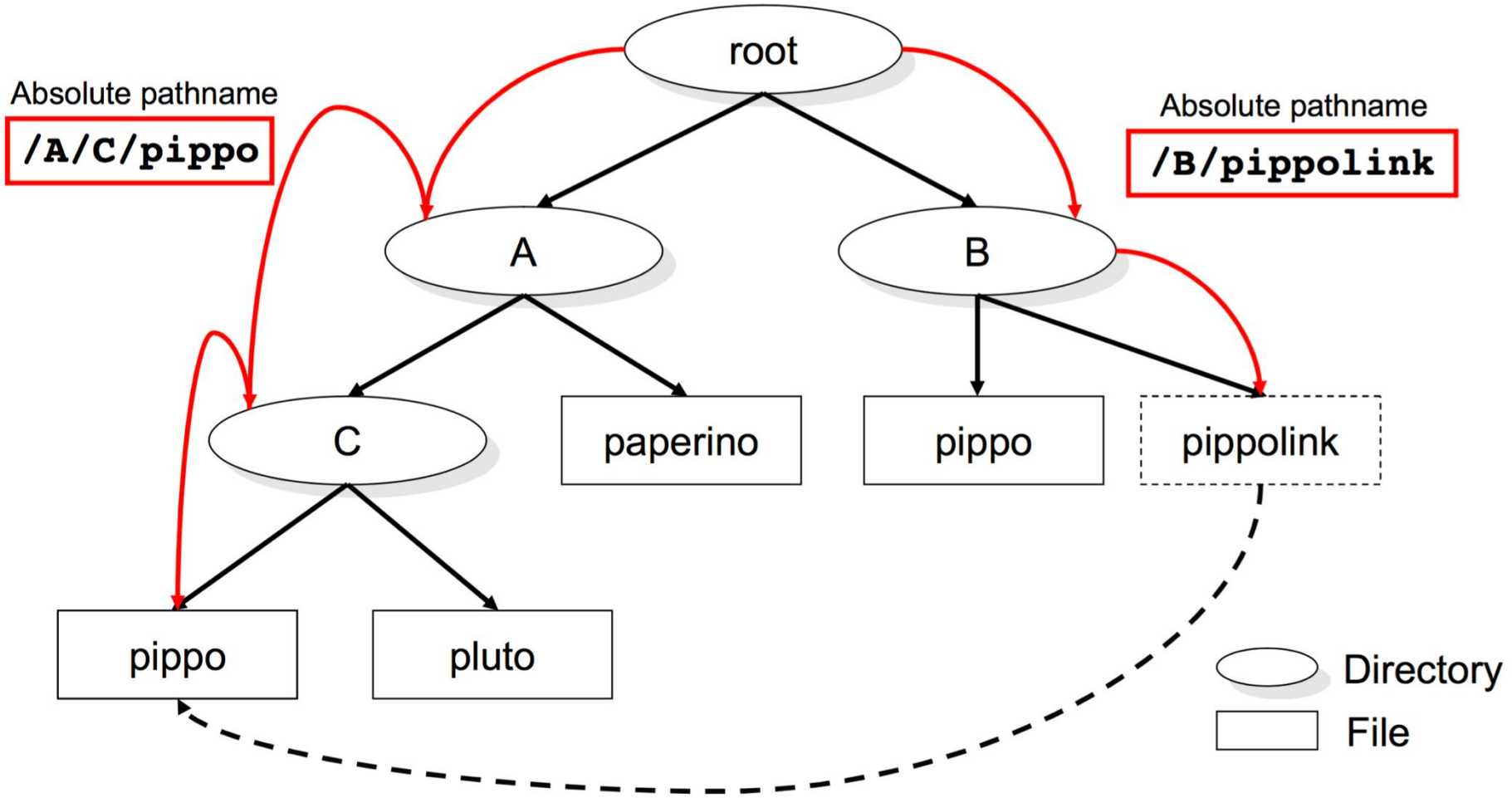

root
Absolute pathname
/B or /B/
A
B
C
paperino
pippo
pippo
pluto
Directory
File

Pathname



Note that absolute pathnames are actually relative pathnames with respect to the root directory /

Pathnames and Links

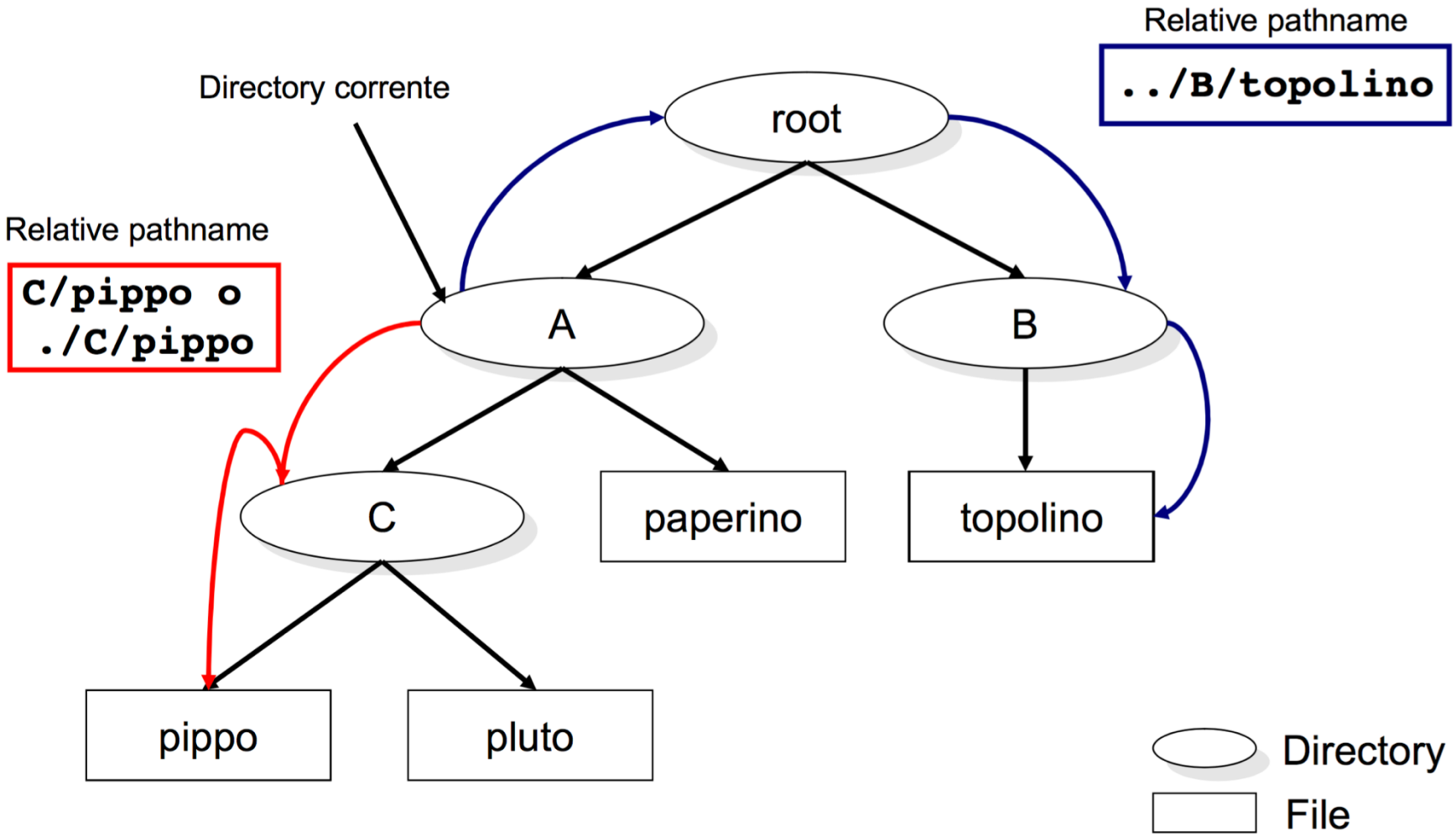


Working Directory

It is the directory from which all relative pathnames are interpreted (aka, current working directory)

- Upon log in, the working directory is set to the home directory
  - the home directory is obtained from our entry in the password file
- Every process has a working directory
- A process can change its working directory with the chdir function (it is a Linux/Unix system call)
  - doc/memo/joe is a relative pathname
    - doc must be a directory within the working directory
    - we can't tell whether joe is a file or a directory
  - /usr/lib/lint is an absolute pathname
    - refers to the file or directory lint in the directory lib, in the directory usr, which is in the root directory

Working Directory



The “ls” command (a simple implementation of)

The “ls” command lists all the files in a directory

```
#include "apue.h"
#include <dirent.h>

int
main(int argc, char *argv[])
{
    DIR             *dp;
    struct dirent   *dirp;

    if (argc != 2)
        err_quit("usage: ls directory_name");

    if ((dp = opendir(argv[1])) == NULL)
        err_sys("can't open %s", argv[1]);
    while ((dirp = readdir(dp)) != NULL)
        printf("%s\n", dirp->d_name);

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

Figure 1.3 List all the files in a directory

Input and Output

File Descriptors

- File descriptors are normally small non-negative integers that the kernel uses to identify the files accessed by a process. Whenever it opens an existing file or creates a new file, the kernel returns a file descriptor that we use when we want to read or write the file.

Standard Input, Standard Output, and Standard Error

- By convention, all shells open three descriptors whenever a new program is run: standard input, standard output, and standard error
- If nothing special is done, as in the simple command ls then all three are connected to the terminal
- Most shells provide a way to redirect some or all of these three descriptors to a file. For example, ls > file.txt executes the ls command with its standard output redirected to the file named file.txt

```
ls > file.txt
```

Unbuffered vs buffered I/O

Unbuffered I/O (Chapter 3)

- Unbuffered I/O is provided by the functions open, read, write, lseek, and close
- These functions all work with file descriptors

```
#include "apue.h"

#define BUFFSIZE    4096

int
main(void)
{
    int     n;
    char    buf[BUFFSIZE];

    while ((n = read(STDIN_FILENO, buf, BUFFSIZE)) > 0)
        if (write(STDOUT_FILENO, buf, n) != n)
            err_sys("write error");

    if (n < 0)
        err_sys("read error");

    exit(0);
}
```

Figure 1.4 Copy standard input to standard output

Unbuffered vs buffered I/O

Standard I/O (Chapter 5)

- standard I/O functions (or routines) provide a buffered interface to the unbuffered I/O functions so to
  - relieve us from having to choose optimal buffer sizes
  - simplify dealing with lines of input (a common occurrence in UNIX applications)

```
#include "apue.h"

int
main(void)
{
    int     c;

    while ((c = getc(stdin)) != EOF)
        if (putc(c, stdout) == EOF)
            err_sys("output error");

    if (ferror(stdin))
        err_sys("input error");

    exit(0);
}
```

Figure 1.5 Copy standard input to standard output, using standard I/O

Programs and Processes

Program

- an executable file residing on disk in a directory
  - A program is read into memory and is executed by the kernel as a result of one of the seven exec functions

Processes and Process ID

- An executing instance of a program is called a process,
  - Some operating systems use the term task to refer to a program that is being executed
- The UNIX System guarantees that every process has a unique numeric identifier called the process ID
  - the process ID is always a non-negative integer

Process Control

- There are three primary functions for process control: fork, exec, and waitpid
- The exec function has seven variants, but we often refer to them collectively as simply the exec function

Signals

- Signals are a technique used to notify a process that some condition has occurred

- The process has three choices for dealing with the signal
  - Ignore the signal - This option isn't recommended for signals that denote a hardware exception, such as dividing by zero or referencing memory outside the address space of the process, as the results are undefined

  - Let the default action occur - For a divide-by-zero condition, the default is to terminate the process

  - Provide a function that is called when the signal occurs (this is called "catching" the signal). By providing a function of our own, we'll know when the signal occurs, and we can handle it as we wish

Signals

Many conditions generate signals, e.g.,

- Two terminal keys are used to interrupt the currently running process
  - the interrupt key: often the DELETE key or Control-C
  - the quit key: often Control-backslash
- If a process divides by zero, the signal whose name is SIGFPE (floating-point exception) is sent to the process
- Another way to generate a signal is by calling the kill function
  - We can call this function from a process to send a signal to another process.
- Naturally, there are limitations
  - we must be the owner of the other process (or the superuser) to be able to send it a signal

Interprocess Communication

Interprocess Communication (IPC) and Process Synchronization

- Pipes
- FIFOs
- Message queues
- Shared memory
- Semaphores