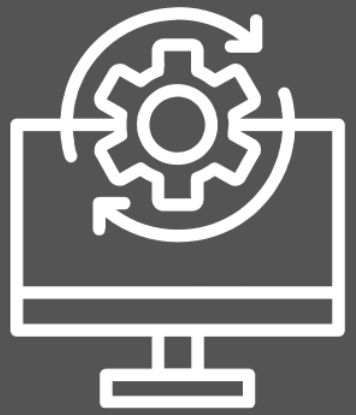


Operating Systems

Operating System Structures (Part I)

ETighe

Outline

- Operating System Services
- User and Operating System-Interface
- System Calls
- System Services/Programs

Overview of Operating System Services

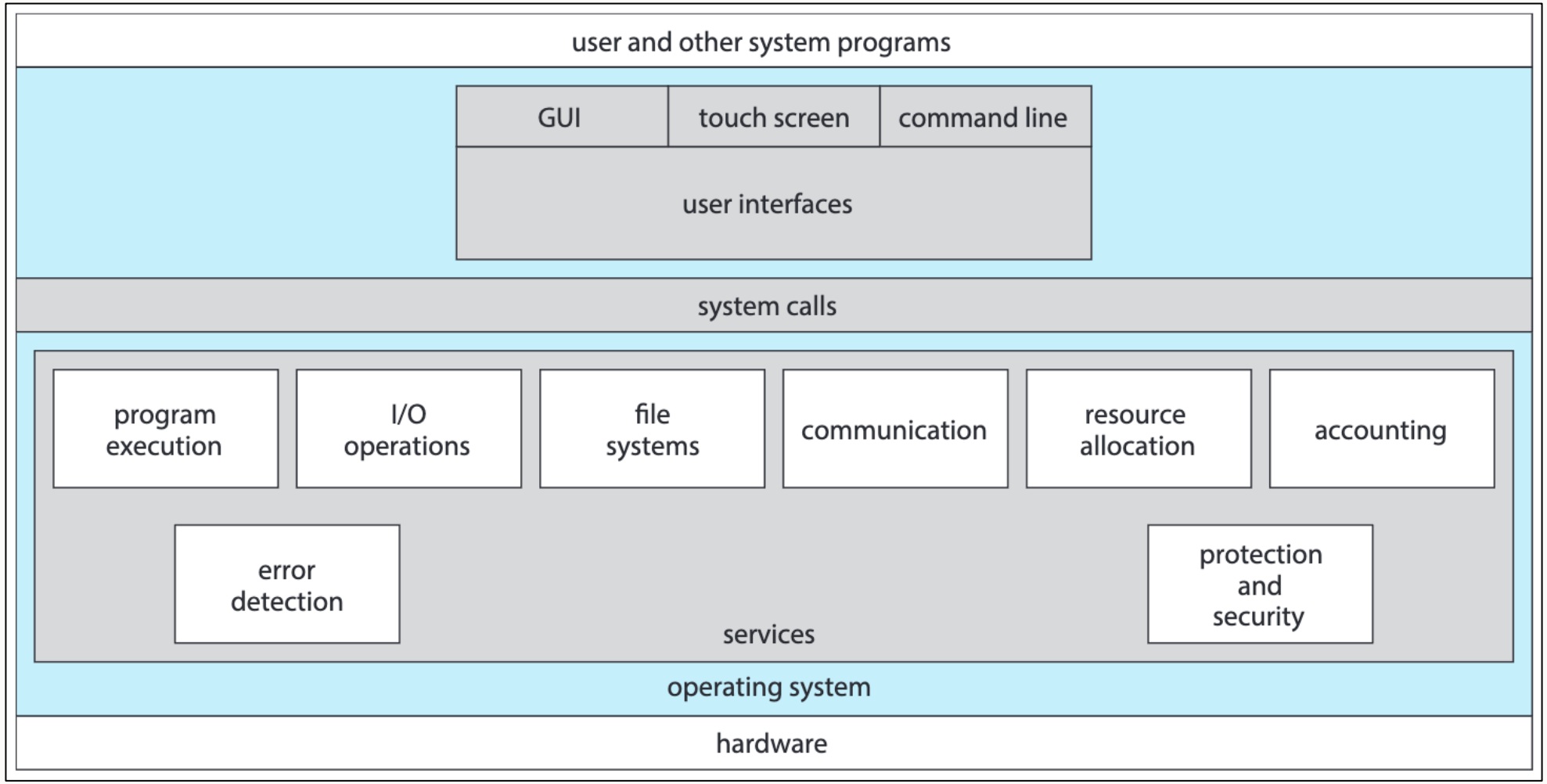


Image from Silberschatz et al. (2018)

Operating System Services

- Helpful services provided to the user include:
  - User interface – most OS have UI
    - Command line (CLI), Graphical based (GUI), touch screen, batch
  - Program execution – the ability to load a program into memory; run a program; end execution either normally or abnormally (indicating error)
  - I/O Operations – Provide processes the ability to access a file or I/O device

Operating System Services

- Helpful services provided to the user include:
  - File-system manipulation - Programs need to read and write files and directories, create and delete them, search them, list file Information, permission management
  - Communications – Allow for the exchange of info between processes on the same computer or between computers over a network
    - Via shared memory or message passing (packets moved by the OS)

Operating System Services

- Helpful services provided to the user include:
  - Error detection – OS needs to be constantly aware of possible errors
    - May occur in the CPU and memory hardware, I/O devices, and/or user programs
    - For each type of error, OS should take the appropriate action to ensure correct and consistent computing
    - Debugging facilities can greatly enhance the user's and programmer's abilities to efficiently use the system

Operating System Services

- Other OS services ensure efficient operation of the system via resource sharing:
  - Resource allocation – Ability to allocate resources to multiple users or multiple jobs running concurrently
    - E.g. CPU cycles, main memory, file storage, I/O devices
  - Logging – The tracking of which users use how much and what kinds of resources

Operating System Services

- Other OS services ensure efficient operation of the system via resource sharing:
  - Protection and security – Concurrent processes should not interfere with each other in a multiuser or networked computer system
    - Protection involves ensuring that all access to system resources is controlled
    - Security of the system from outsiders requires user authentication; extends to defending external I/O devices from invalid access attempts

User and Operating System Interface

Command Line Interpreter (CLI)

- Allows direct command entry
- Sometimes implemented in kernel, sometimes by systems program
- Sometimes multiple flavors implemented – shells
  - E.g. C shell, Bourne-Again shell, Korn shell

Command Line Interpreter

- Primarily fetches a command from user and executes it
- Sometimes commands built-in, sometimes just names of programs
  - For the latter, adding new features doesn't require shell modification because modification is done in the program

Bourne Shell Command Interpreter

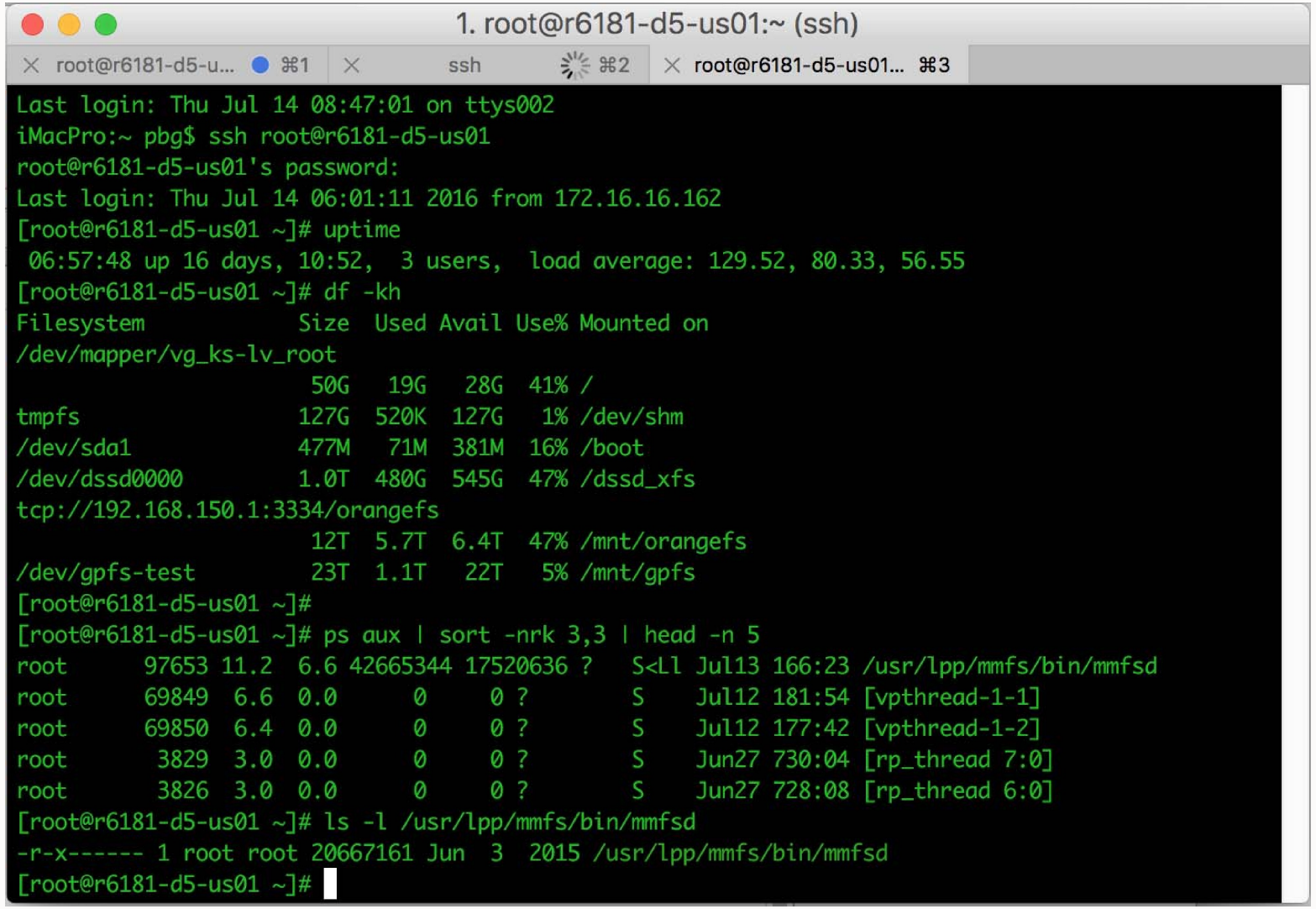


Image from Silberschatz et al. (2018)

User Operating System Interface - GUI

- User-friendly desktop metaphor interface
  - Usually mouse, keyboard, and monitor
  - Icons represent files, programs, actions, etc
  - Various mouse buttons over objects in the interface cause various actions
    - E.g. provide information, options, execute function, open directory/folders
  - Invented at Xerox PARC in the early 1970s, although became more widespread due to Apple Macintosh

User Operating System Interface - GUI

- Many systems now include both CLI and GUI interfaces
  - Microsoft Windows is GUI with Command shell
  - Apple Mac OS X is Aqua GUI interface with UNIX kernel underneath and shells available
  - Unix and Linux have CLI with optional GUI interfaces (CDE, KDE, GNOME)

Mac OS x GUI

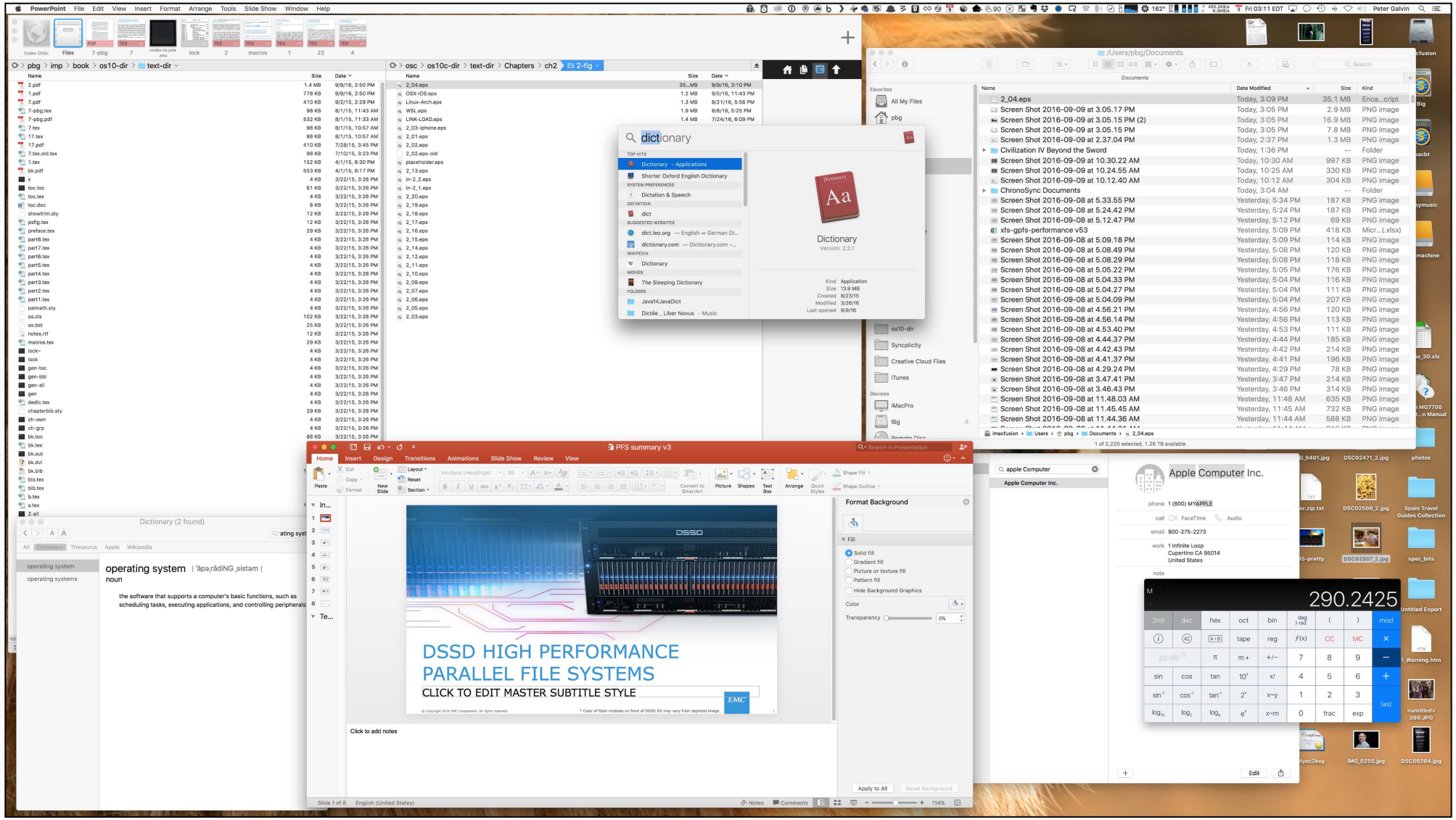


Image from Silberschatz et al. (2018)

Touchscreen Interfaces

- Touchscreen devices require new interfaces
  - Mouse not possible or not desired
  - Actions and selection based on gestures
  - Virtual keyboard for text entry
- May also utilize voice commands

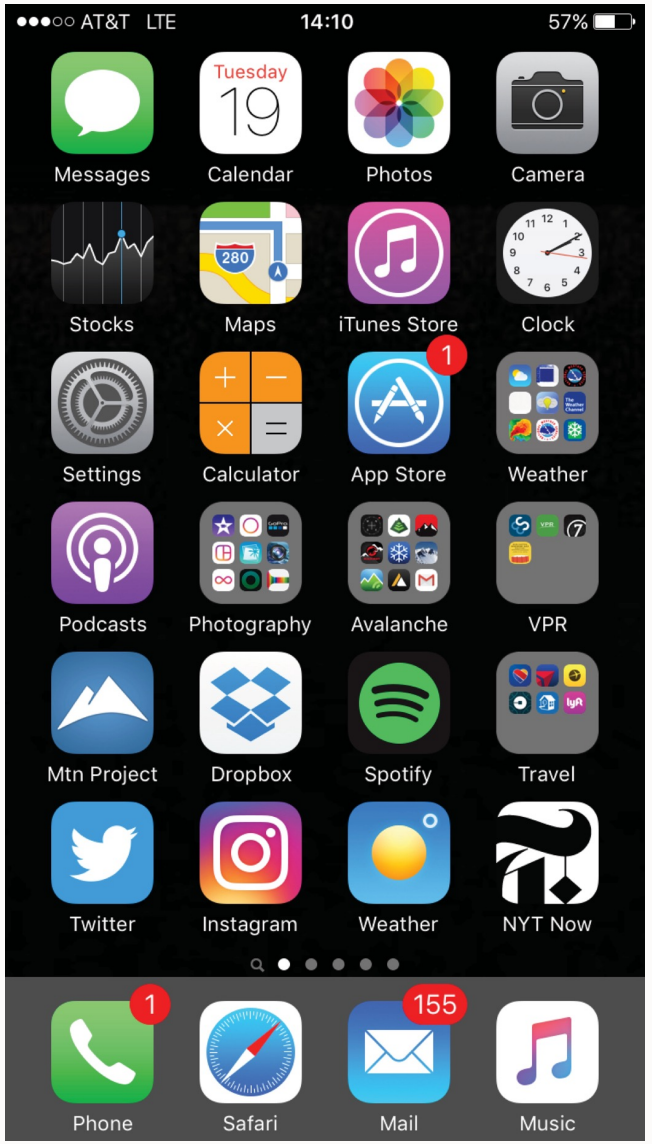


Image from Silberschatz et al. (2018)

System Calls

System Calls

- Provides an interface to the different OS services
- Typically available as functions written in C and C++
  - Sometimes in assembly

```
cp in.txt out.txt
```

Copy command — Input file — Output file

System Calls

- A visualization of the system call sequence for copying the contents of one file to another file

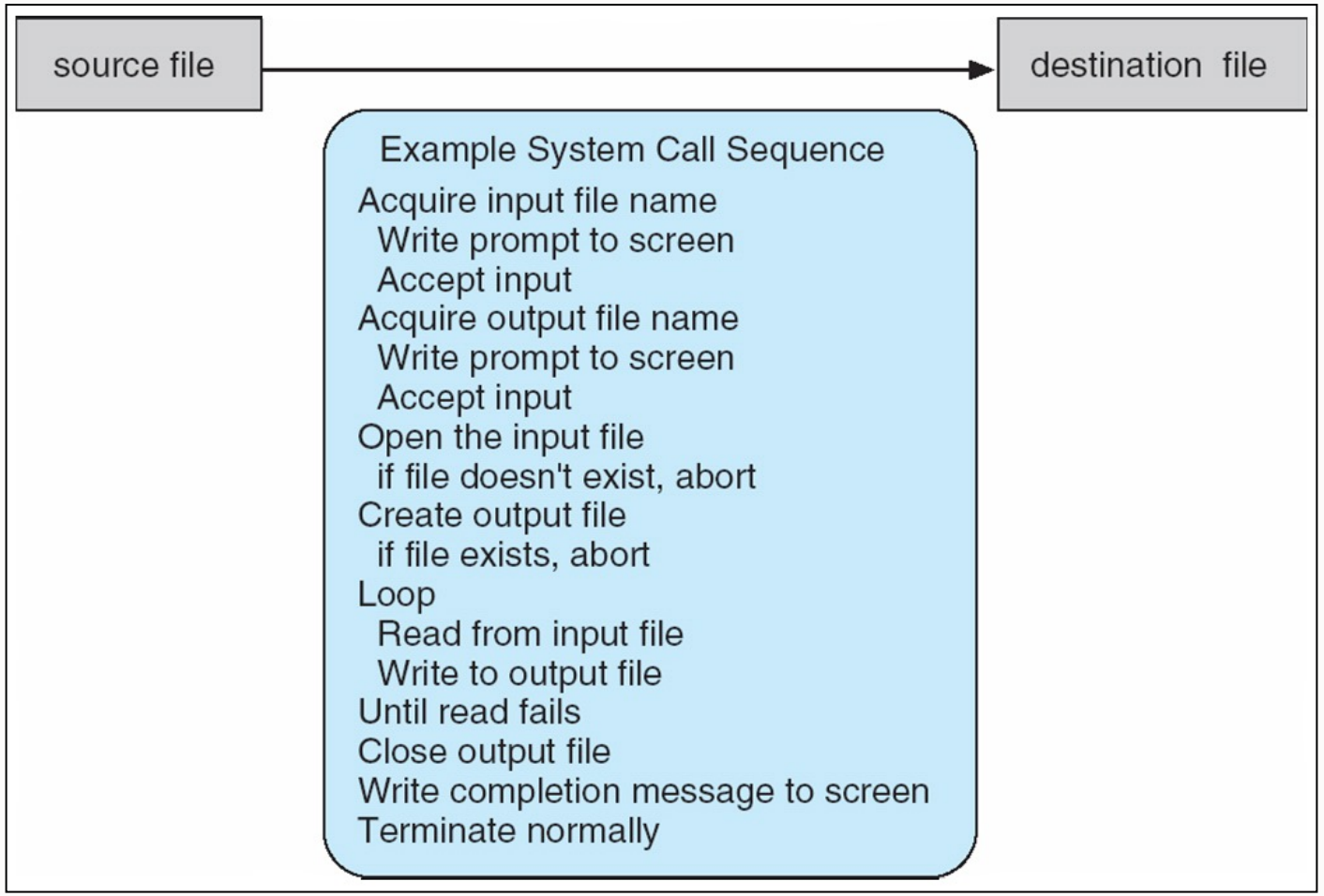


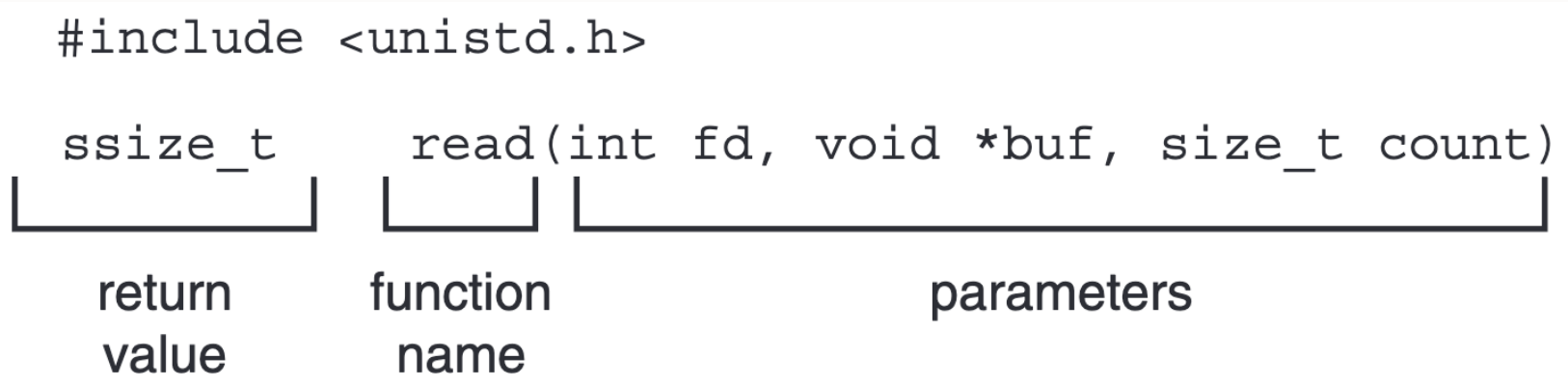
Image from Silberschatz et al. (2018)

System Calls

- Mostly accessed by programs via a high-level Application Program Interface (API) rather than direct system call use
- Three most common APIs are:
  - Win32 API for Windows
  - POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X)
  - Java API for the Java virtual machine (JVM)

Example of Standard API

- Consider the **`read()`** function available in UNIX and Linux systems



Parameters

- `int fd`— the file descriptor to be read
- `void *buf`—a buffer into which the data will be read
- `size t count`—the maximum number of bytes to be read into the buffer

System Call Implementation

- Typically, a number is associated with each system call
  - System-call interface maintains a table indexed according to these numbers
- The system call interface invokes the intended system call in OS kernel and returns status of the system call and any return values

System Call Implementation

- The caller does not need know about how the system call is implemented
  - Just needs to obey API and understand what OS will do as a result call
- Most details of the OS interface are hidden from programmer by API
  - Managed by run-time support library (set of functions built into libraries included with compiler)

API x System Call x OS Relationship

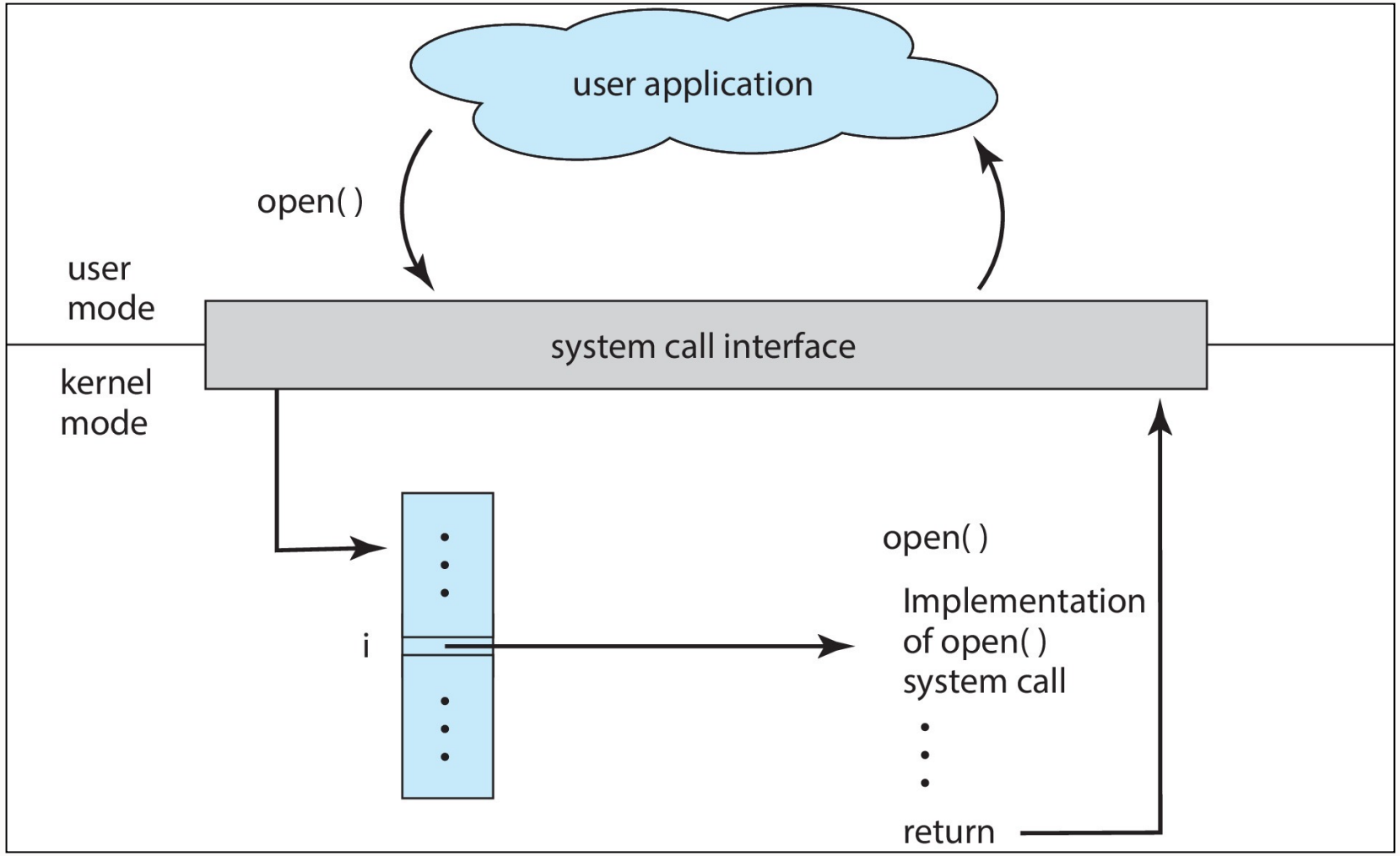


Image from Silberschatz et al. (2018)

API x System Call x OS Relationship

In the context of the C programming language

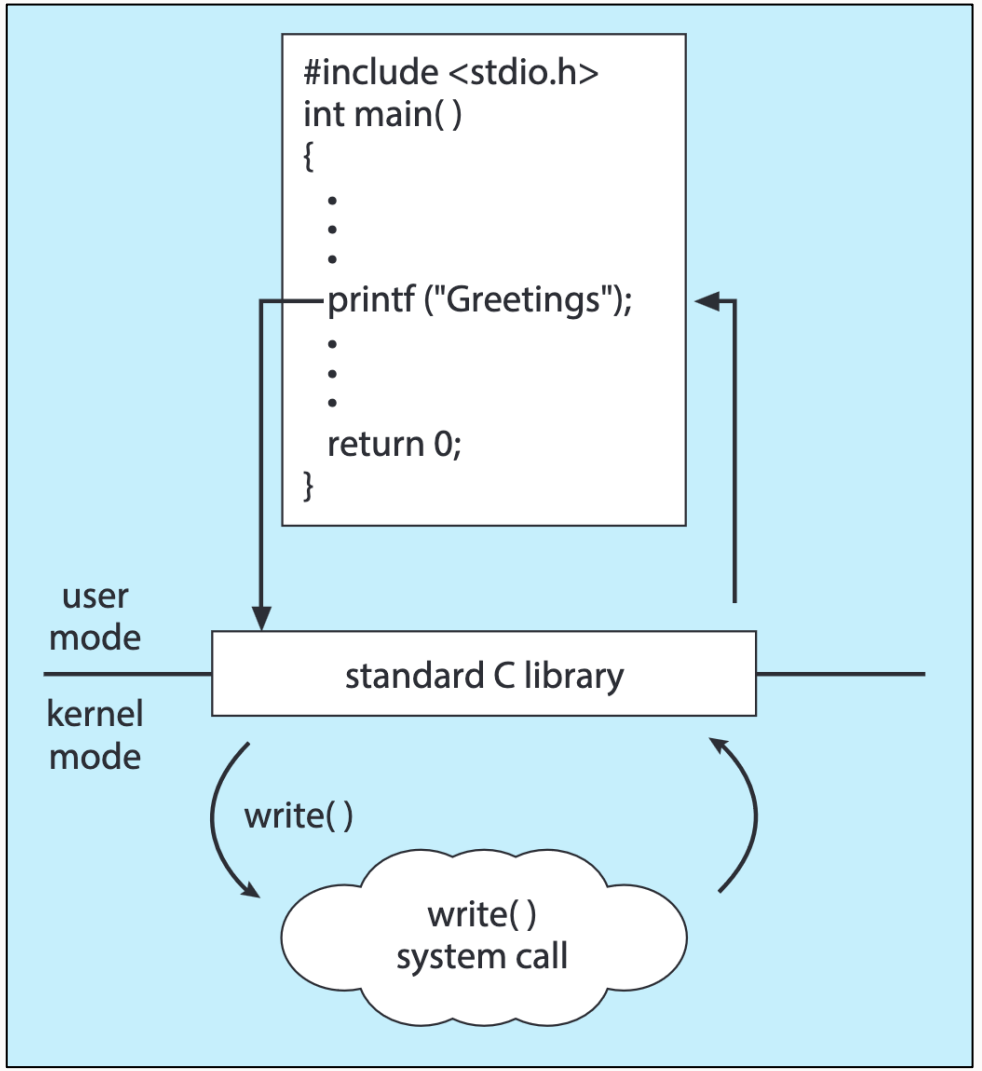


Image from Silberschatz et al. (2018)

System Call Parameter Passing

- Often, more information is required than simply identity of desired system call
  - Exact type and amount of information vary according to OS and call

System Call Parameter Passing

- 3 general methods used to pass parameters to the OS
  - Pass the parameters in registers (simplest)
    - In some cases, may be more parameters than registers
  - Parameters stored in a block, or table, in memory, and address of block passed as a parameter in a register
    - This approach taken by Linux and Solaris
  - Parameters placed (pushed) onto the stack by the program and popped off the stack by the operating system

Note: Block and stack methods do not limit the number or length of parameters being passed

Parameter Passing via Table

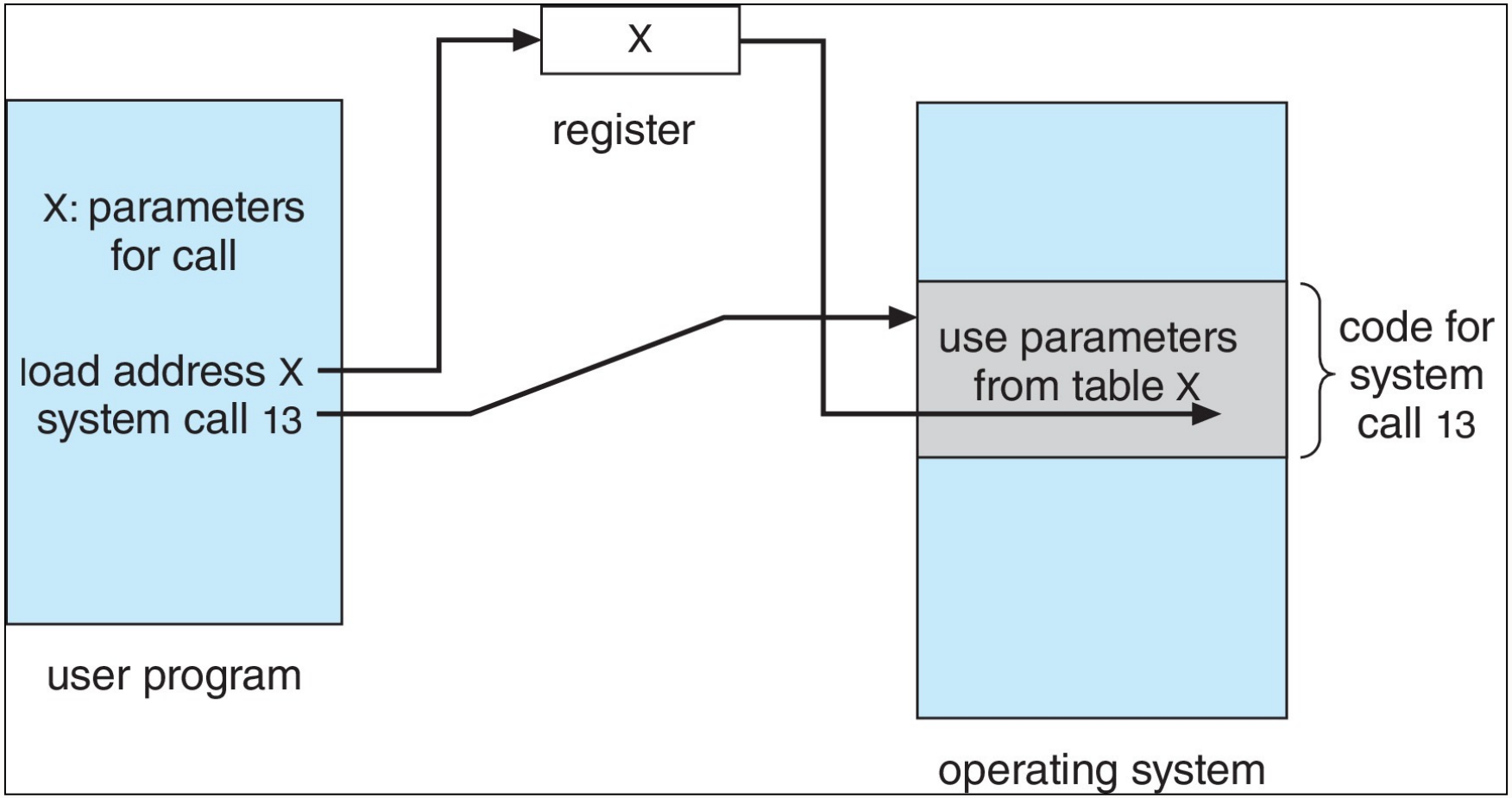


Image from Silberschatz et al. (2018)

Types of System Calls

- Process control
  - create process, terminate process
  - load, execute
  - get process attributes, set process attributes
  - wait for time/event, signal event
  - allocate and free memory

Types of System Calls

- Process control
  - Dump memory if an error occurs
    - Memory dump may be written to a log file and examined by a debugger
  - Locks for managing access to shared data between processes

Types of System Calls

- Device management
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices

Types of System Calls

- File management
  - create file, delete file
  - open, close file
  - read, write, reposition
  - get and set file attributes

Types of System Calls

- Protection
  - Control access to resources
  - Get and set permissions
  - Allow and deny user access

Example of Windows and Unix System Calls

	Windows	Unix

Process control	`CreateProcess()`
	`ExitProcess()`
	`WaitForSingleObject()`
		`fork()`
		`exit()`
		`wait()`

File management	`CreateFile()`
	`ReadFile()`
	`WriteFile()`
	`CloseHandle()`
		`open()`
		`read()`
		`write()`
		`close()`

Device management	`SetConsoleMode()`
	`ReadConsole()`
	`WriteConsole()`
		`ioctl()`
		`read()`
		`write()`

Information maintenance	`GetCurrentProcessID()`
	`SetTimer()`
	`Sleep()`
		`getpid()`
		`alarm()`
		`sleep()`

Communications	`CreatePipe()`
	`CreateFileMapping()`
	`MapViewOfFile()`
		`pipe()`
		`shm_open()`
		`mmap()`

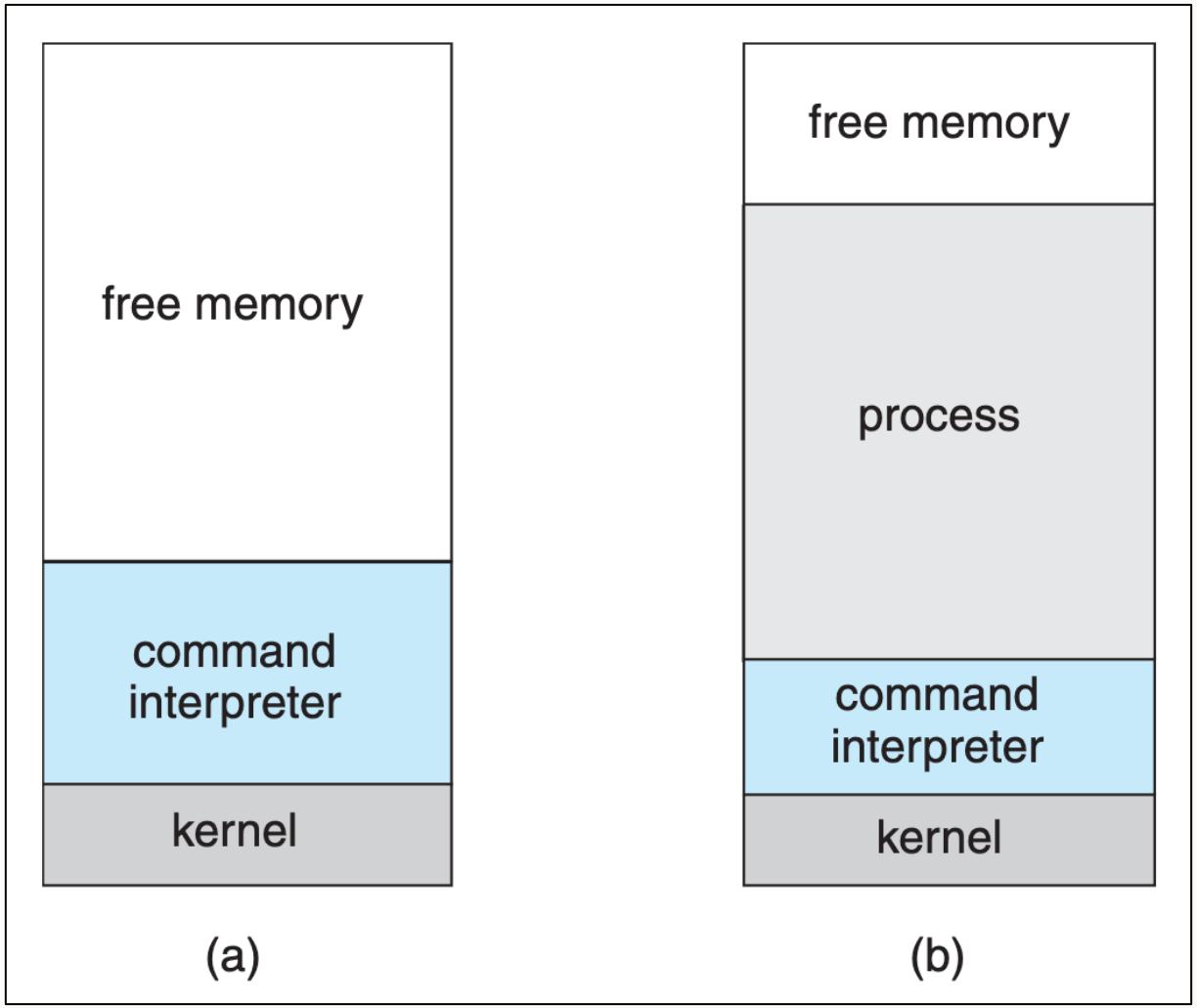
Protection	`SetFileSecurity()`
	`InitlializeSecurityDescriptor()`
	`SetSecurityDescriptorGroup()`
		`chmod()`
		`umask()`
		`chown()`

Image from Silberschatz et al. (2018)

Example: MS-DOS

- Single-tasking OS (i.e. one user, one task at a time)
- Shell invoked when the system is booted
- Simple method to run program
  - No process created
    - Due to single memory space
  - Loads program into memory, overwriting all but the kernel
- Program exit → shell reloaded

Example: MS-DOS



At system startup

Running a program

Image from Silberschatz et al. (2012)

Example: FreeBSD

- Unix variant
- Multitasking
- User login → invoke user's choice of shell
  - Shell executes `fork()` system call to create process
  - Executes `exec()` to load program into process
  - Shell waits for process to terminate or continues with user commands
- Process exits with code of 0 – no error or > 0 – error code

Example: FreeBSD

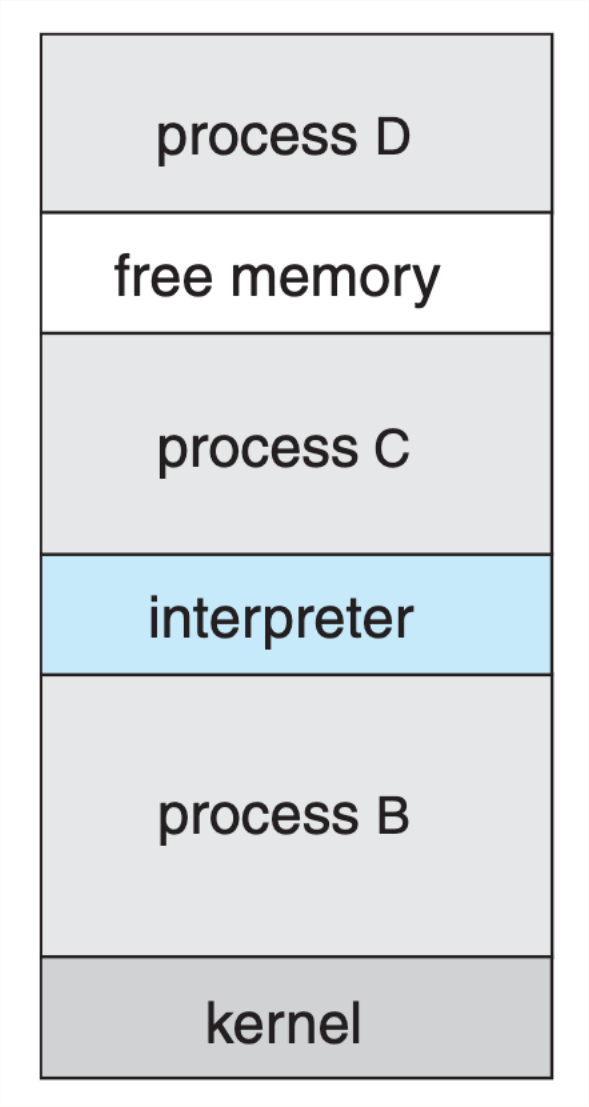


Image from Silberschatz et al. (2012)

System Services/Programs

System Programs

- System programs provide a convenient environment for program development and execution
  - Most users' view of the operating system is defined by system programs, not the actual system calls
  - Some of them are simply user interfaces to system calls; others are considerably more complex

System Programs

- They can be divided into:
  - File manipulation
  - File modification
  - Status information
  - Programming language support
  - Program loading and execution
  - Communication
  - Application programs

System Programs

- File management
  - Programs that create, delete, copy, rename, print, dump, list, and generally manipulate files and directories
- File modification
  - Text editors to create and modify files
  - Special commands to search contents of files or perform transformations of the text

System Programs

- Status information
  - Some programs ask the system for info - date, time, amount of available memory, disk space, number of users
  - Others provide detailed performance, logging, and debugging information
  - Typically, these programs format and print the output to the terminal or other output devices
  - Some systems implement a registry - used to store and retrieve configuration information

System Programs

- Programming-language support
  - Compilers, assemblers, debuggers and interpreters sometimes provided
- Program loading and execution
  - Once a program is ready for execution, it must be loaded into memory
  - Include absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language

System Programs

- Communications
  - Provide the mechanism for creating virtual connections among processes, users, and computer systems
  - Allow users to send messages to one another's screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another

System Programs

- Background Services
  - Launch at boot time
    - Some at system startup, then terminate
    - Some from system boot to shutdown
  - Provide facilities like disk checking, process scheduling, error logging, printing
  - Run in user context not kernel context
  - Known as services, subsystems, daemons

System Programs

- Application programs
  - Don’t pertain to system
  - Run by users
  - Not typically considered part of OS
  - Launched by command line, mouse click, finger poke

Questions?

Thank you ☺