OPERATING SYSTEMS CS3500 – CHAP - 2.

PROF. SUKHENDU DAS DEPTT. OF COMPUTER SCIENCE AND ENGG., IIT MADRAS, CHENNAI – 600036.

Email: **sdas@cse.iitm.ac.in**
URL**: //www.cse.iitm.ac.in/~vplab/os.html**

https://sites.google.com/smail.iitm.ac.in/3500-os/

Aug. – 2022.

OPERATING-SYSTEM SERVICES

OUTLINE

- Operating System Services
- User and Operating System-Interface
- System Calls
- Linkers and Loaders
- Why Applications are Operating System Specific
- Design and Implementation

OPERATING SYSTEM SERVICES

- Operating systems provide an environment for execution of programs and services to programs and users
- One set of operating-system services provides functions that are helpful to the user:
  - **User interface -** Almost all operating systems have a user interface (**UI**).
    - Varies between **Command-Line** (**CLI**), **Graphics User Interface** (**GUI**), **touch-screen, Batch**
  - **Program execution -** The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)
  - **I/O operations -** A running program may require I/O, which may involve a file or an I/O device
  - **File-system manipulation -** The file system is of particular interest. Programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.

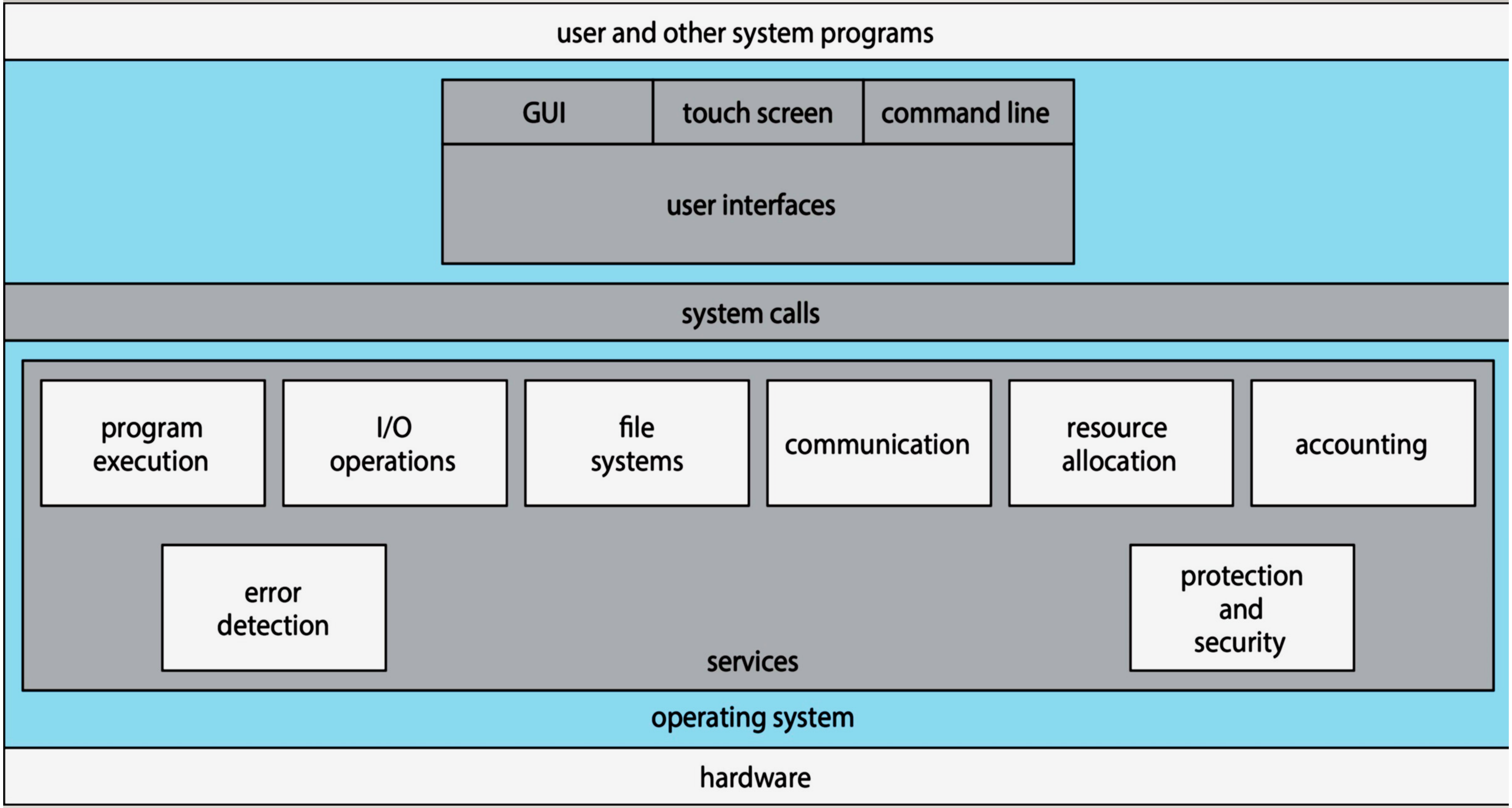
OPERATING SYSTEM SERVICES (CONT.)

- One set of operating-system services provides functions that are helpful to the user (Cont.):
  - **Communications** – Processes may exchange information, on the same computer or between computers over a network
    - Communications may be via shared memory or through message passing (packets moved by the OS)
  - **Error detection** – OS needs to be constantly aware of possible errors
    - May occur in the CPU and memory hardware, in I/O devices, in user program
    - 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 (CONT.)

- Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing
  - **Resource allocation -** When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
    - Many types of resources - CPU cycles, main memory, file storage, I/O devices.
  - **Logging -** To keep track of which users use how much and what kinds of computer resources
  - **Protection and security -** The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other
    - **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

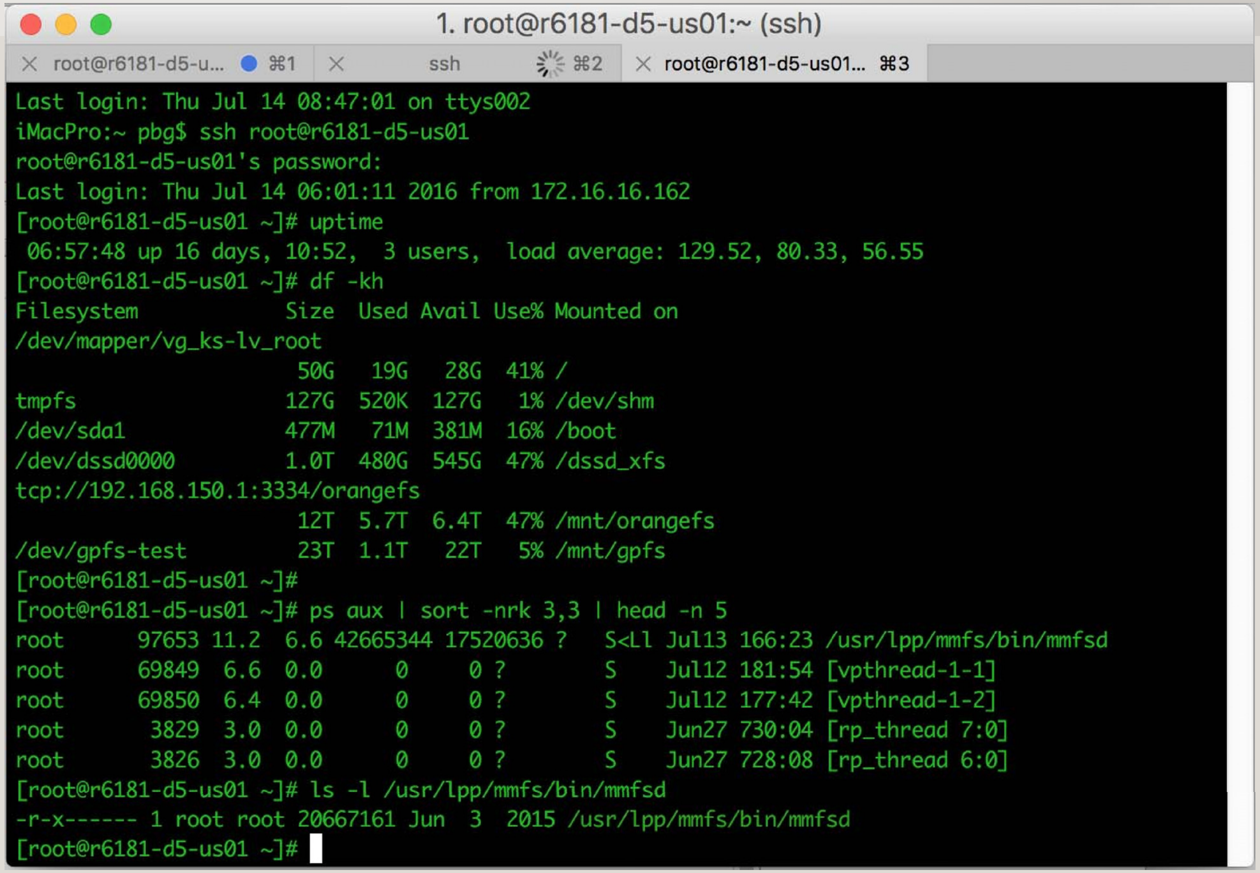
A VIEW OF OPERATING SYSTEM SERVICES



COMMAND LINE INTERPRETER

- CLI allows direct command entry
- Sometimes implemented in kernel, sometimes by systems program
- Sometimes multiple flavors implemented – **shells**
- Primarily fetches a command from user and executes it
- Sometimes commands built-in, sometimes just names of programs
  - If the latter, adding new features doesn't require shell modification

Bourne Shell Command Interpreter



```
osta@osta-VirtualBox:~$ uptime
 15:52:12 up 29 min,  2 users,  load average: 0.08, 0.30, 0.33
osta@osta-VirtualBox:~$ df -kh
Filesystem      Size  Used Avail Use% Mounted on
udev            3.9G     0  3.9G   0% /dev
tmpfs           796M  1.6M  794M   1% /run
/dev/sda5        50G  8.8G   39G  19% /
tmpfs           3.9G     0  3.9G   0% /dev/shm
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
tmpfs           3.9G     0  3.9G   0% /sys/fs/cgroup
/dev/loop1      128K  128K     0 100% /snap/bare/5
osta@osta-VirtualBox:~$ ps aux | sort -nrk 3,3 | head -n 5
osta        4457  3.7  3.7 3655680 303392 ?      Ssl  15:48   0:09 /usr/bin/gnome-shell
meena       2782  2.1  4.0 3688244 333404 ?      Ssl  15:29   0:30 /usr/bin/gnome-shell
osta        4287  1.2  0.8 254984 72040 tty3     Sl+  15:48   0:03 /usr/lib/xorg/Xorg v
t3 -displayfd 3 -auth /run/user/1000/gdm/Xauthority -background none -noreset -keeptty
 -verbose 3
osta        4753  0.8  0.6 814616 51284 ?        Ssl  15:48   0:01 /usr/libexec/gnome-t
erminal-server
osta        4481  0.6  0.4 278020 33576 ?        Sl   15:48   0:01 /usr/libexec/ibus-ex
tension-gtk3
osta@osta-VirtualBox:~$ ls -l /usr/bin/gnome-shell
-rwxr-xr-x 1 root root 23168 May 19  2021 /usr/bin/gnome-shell
osta@osta-VirtualBox:~$ 
```

df -kh

df : used to display information related to file systems about total space and available space

The '-h' option displays the disc space in human-readable format. It will display the size in powers of 1024, appending G for gigabytes, M for megabytes, and B for bytes.

-k: equivalent to --block-size=1K ; scale sizes by SIZE here 1K before printing them.

ps aux:

To monitor processes running on your Linux system.

sort -nrk 3,3

-n : to sort according to string numerical value

-r : reverses your results

-k : to sort according to particular columns

head -n 5

show the specified number of lines from the output

ls -l : The -l option signifies the long list format ; displays the file permissions, the number of links, owner name, owner group, file size, time of last modification, and the file or directory name

COMMON UBUNTU Commands

cat
lp
ls
df
sudo
firewall-cmd
dig/nslookup
chmod, chown
id, ip, du, lsof, netstat
top, env, ps, grep, tail, curl, dm

find, awk, traceroute, tar, history, sestatus

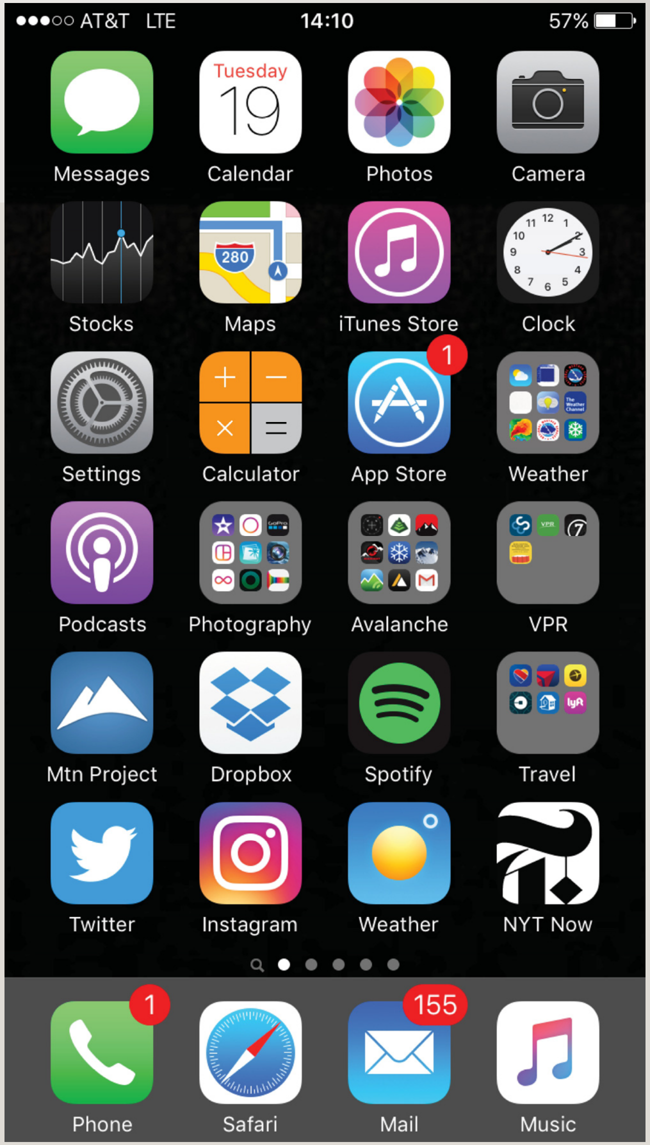
rsync, strace, tac, rev, sed, awk, cut, watch, diff

USER OPERATING SYSTEM INTERFACE - GUI

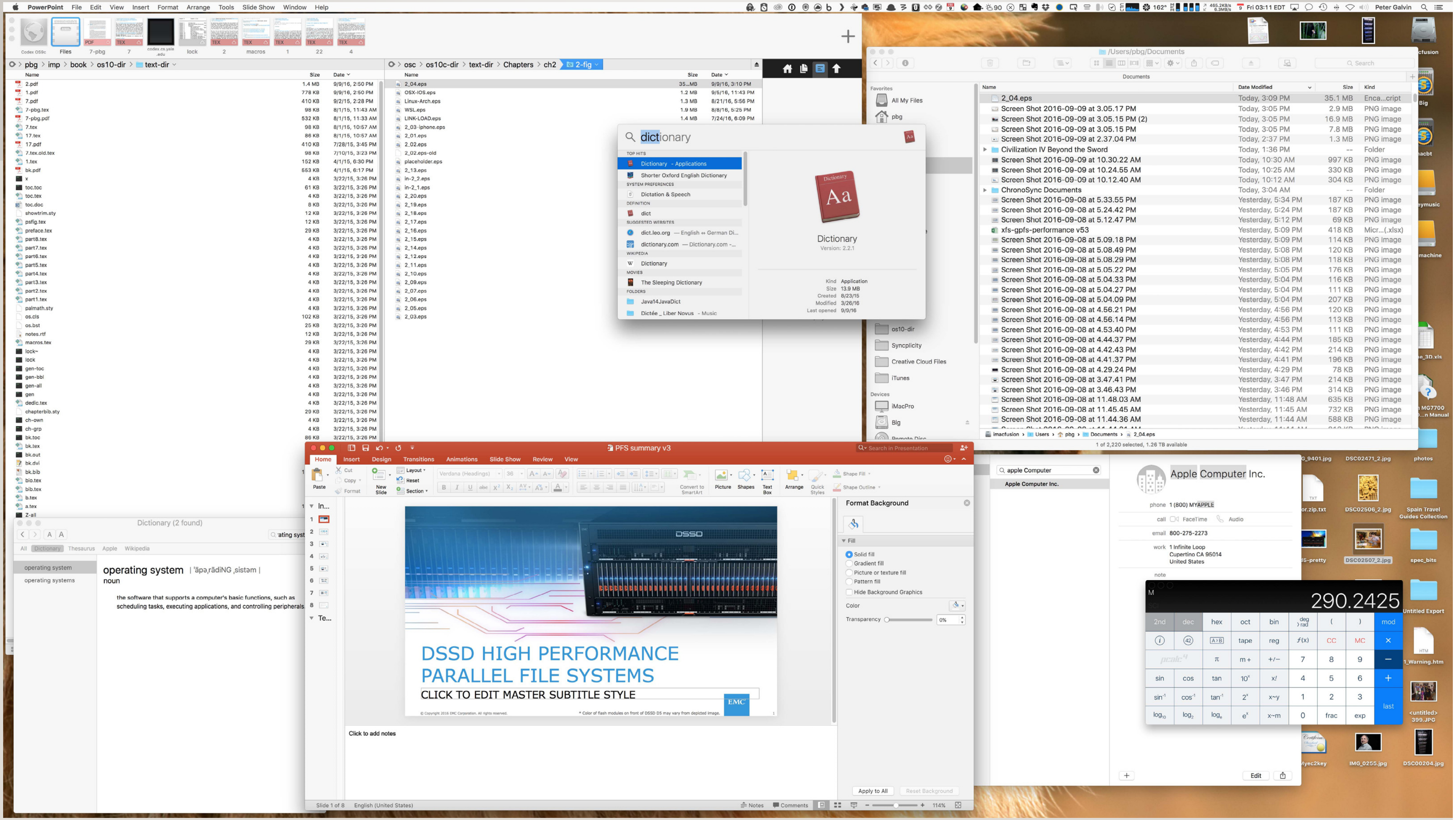
- User-friendly **desktop** metaphor interface
  - Usually mouse, keyboard, and monitor
  - **Icons** represent files, programs, actions, etc
  - Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a **folder**)
  - Invented at Xerox PARC
- Many systems now include both CLI and GUI interfaces
  - Microsoft Windows is GUI with CLI "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)

TOUCHSCREEN INTERFACES

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



THE MAC OS X GUI



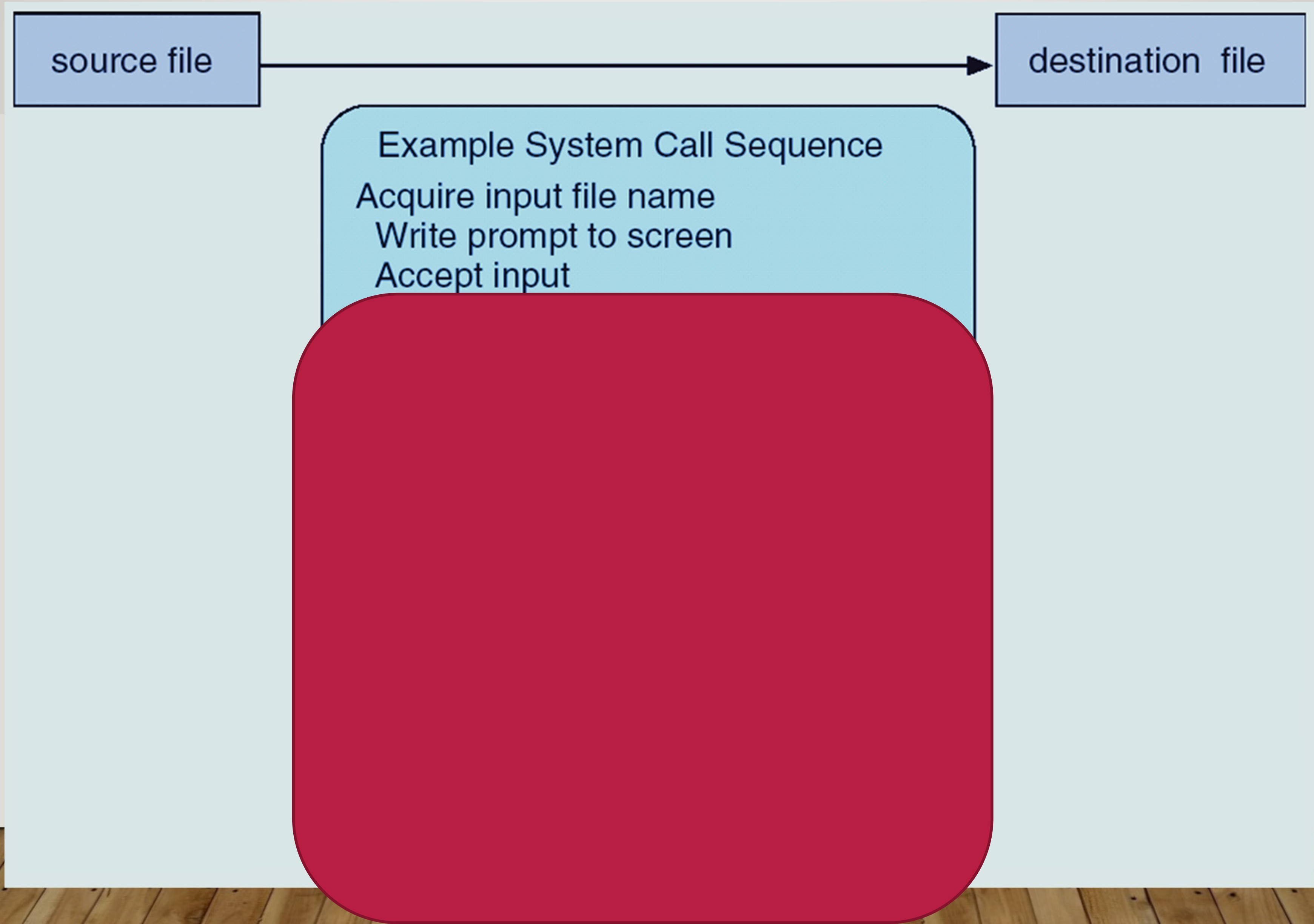
SYSTEM CALLS

- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level **Application Programming 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), and Java API for the Java virtual machine (JVM)

*Note that the system-call names used throughout this text are generic

EXAMPLE OF SYSTEM CALLS

- System call sequence to copy the contents of one file to another file



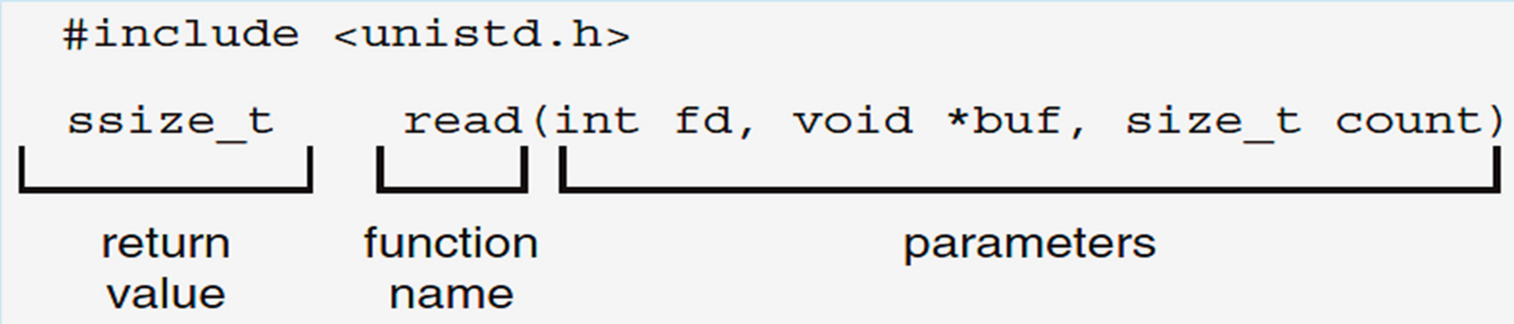
EXAMPLE OF STANDARD API

EXAMPLE OF STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
man read
```

on the command line. A description of this API appears below:



A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

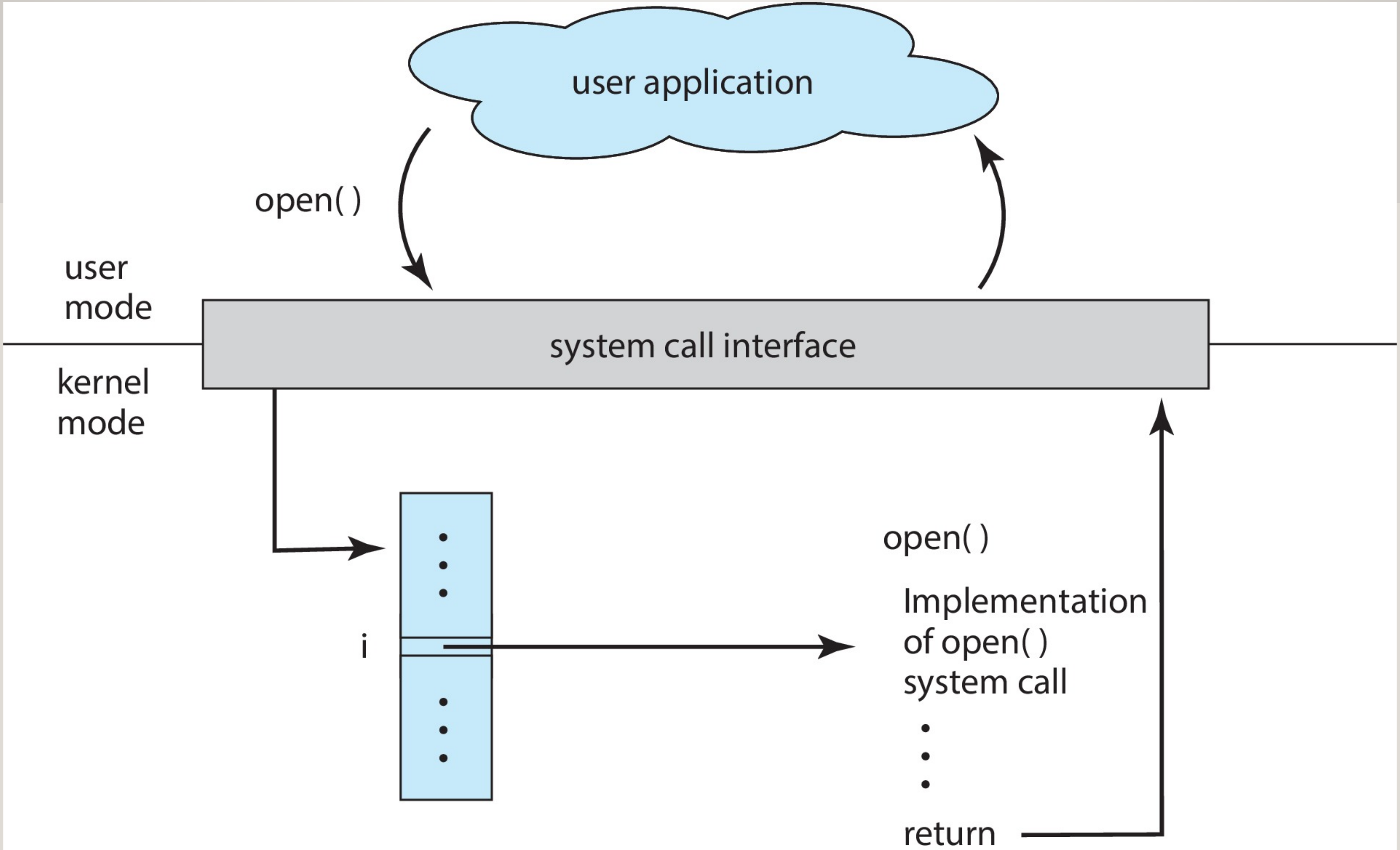
- `int fd`—the file descriptor to be read
- `void *buf`—a buffer into which the data will be read
- `size_t count`—the maximum number of bytes to be read into the buffer

On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns −1.

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
- The caller need know nothing 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 OS interface hidden (eg encapsulation in object class) from programmer by API
    - Managed by run-time support library (set of functions built into libraries included with compiler)

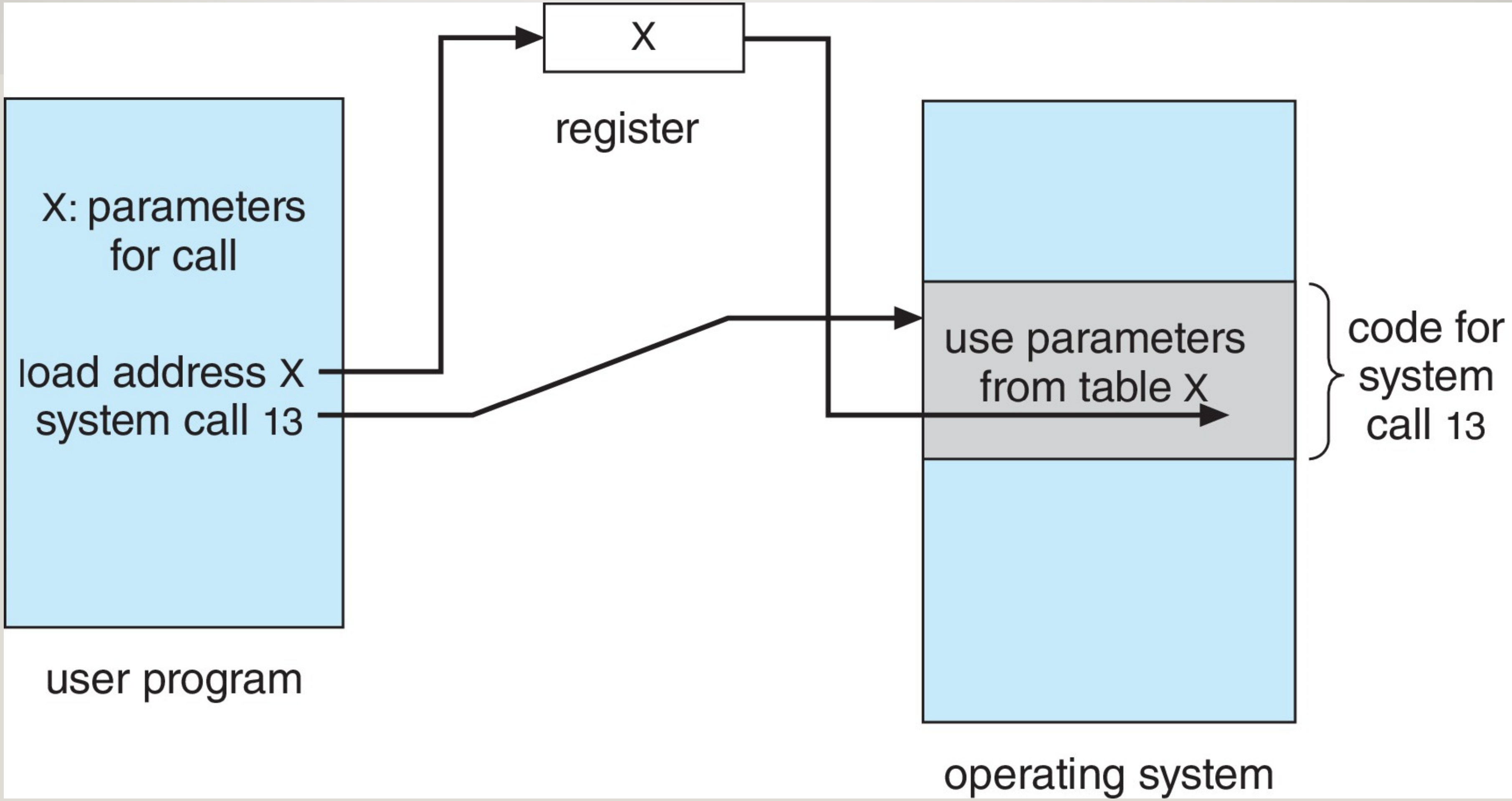
API – SYSTEM CALL – OS RELATIONSHIP



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
- Three general methods used to pass parameters to the OS
  - Simplest: pass the parameters in registers
    - 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, or **pushed**, onto the **stack** by the program and **popped** off the stack by the operating system
  - Block and stack methods do not limit the number or length of parameters being passed

PARAMETER PASSING VIA TABLE



TYPES OF SYSTEM CALLS

- **Process control**
  - create process, terminate process
  - end, abort
  - load, execute
  - get process attributes, set process attributes
  - wait for time
  - wait event, signal event
  - allocate and free memory
  - Dump memory if error
  - **Debugger** for determining **bugs, single step** execution
  - **Locks** for managing access to shared data between processes
- **File management**
  - create file, delete file
  - open, close file
  - read, write, reposition
  - get and set file attributes
- **Device management**
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices

TYPES OF SYSTEM CALLS (CONT.)

- **Information maintenance**
  - get time or date, set time or date
  - get system data, set system data
  - get and set process, file, or device attributes

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

- **Communications**
  - create, delete communication connection
  - send, receive messages if **message passing model** to **host name** or **process name**
    - From **client** to **server**
  - **Shared-memory model** create and gain access to memory regions
  - transfer status information
  - attach and detach remote devices

EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

The following illustrates various equivalent system calls for Windows and UNIX operating systems.

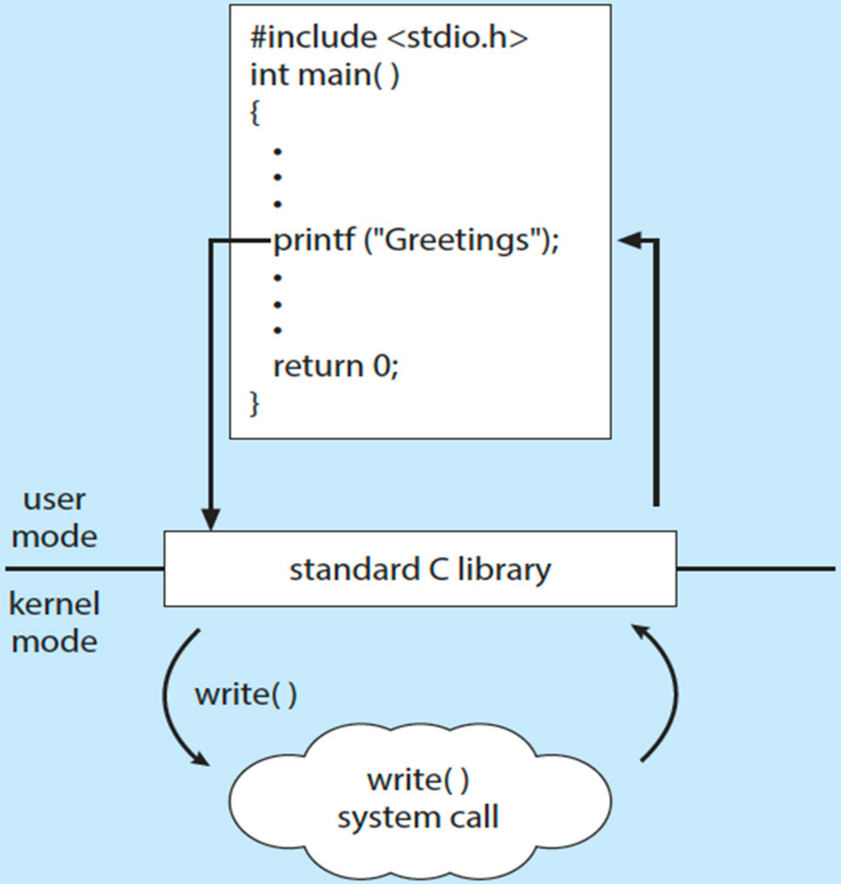
	Windows	Unix
Process control	`CreateProcess()`	`fork()`
	`ExitProcess()`	`exit()`
	`WaitForSingleObject()`	`wait()`
File management	`CreateFile()`	`open()`
	`ReadFile()`	`read()`
	`WriteFile()`	`write()`
	`CloseHandle()`	`close()`
Device management	`SetConsoleMode()`	`ioctl()`
	`ReadConsole()`	`read()`
	`WriteConsole()`	`write()`
Information maintenance	`GetCurrentProcessID()`	`getpid()`
	`SetTimer()`	`alarm()`
	`Sleep()`	`sleep()`
Communications	`CreatePipe()`	`pipe()`
	`CreateFileMapping()`	`shm_open()`
	`MapViewOfFile()`	`mmap()`
Protection	`SetFileSecurity()`	`chmod()`
	`InitlializeSecurityDescriptor()`	`umask()`
	`SetSecurityDescriptorGroup()`	`chown()`

STANDARD C LIBRARY EXAMPLE

- C program invoking printf() library call, which calls write() system call

THE STANDARD C LIBRARY

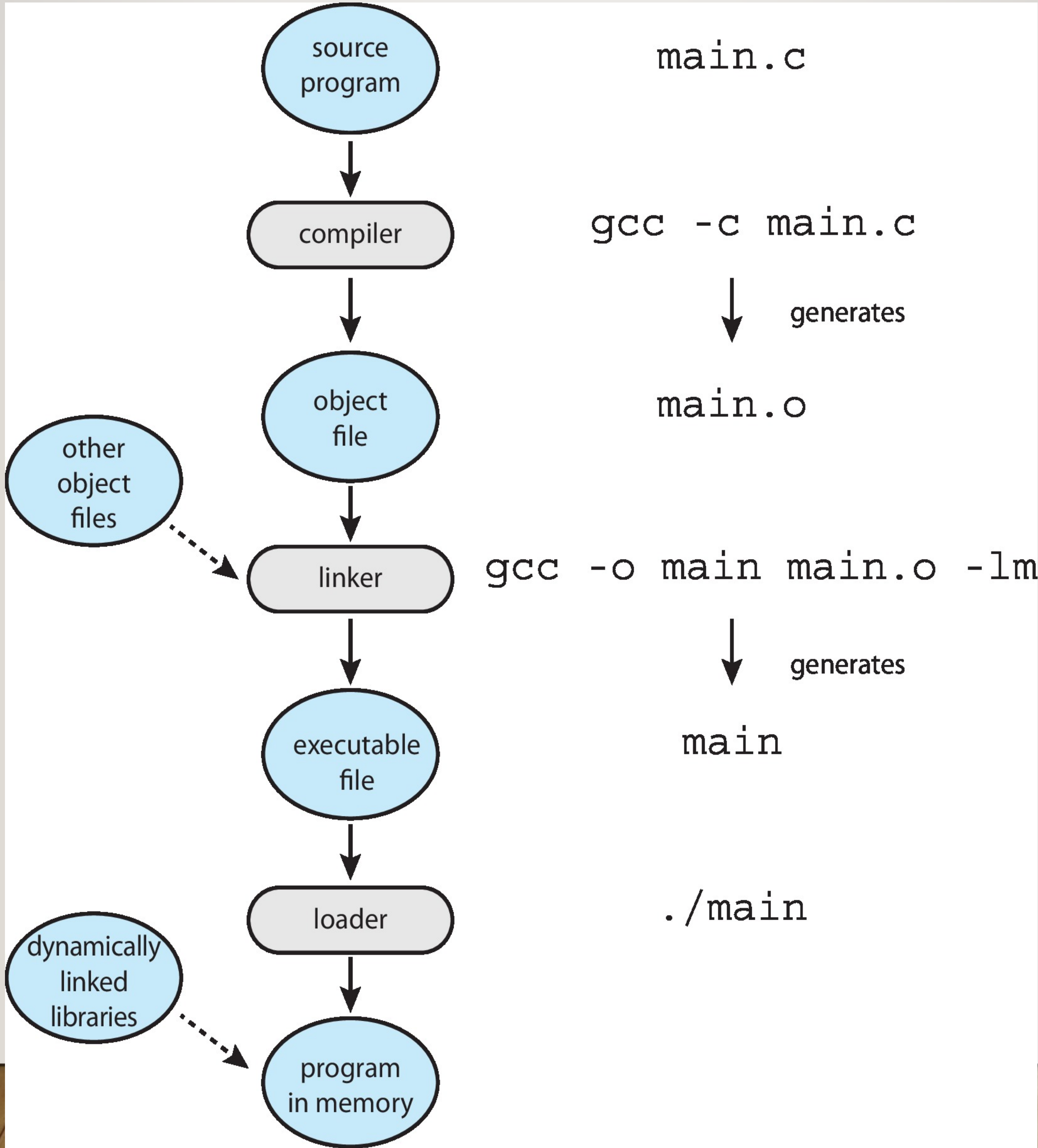
The standard C library provides a portion of the system-call interface for many versions of UNIX and Linux. As an example, let's assume a C program invokes the `printf()` statement. The C library intercepts this call and invokes the necessary system call (or calls) in the operating system—in this instance, the `write()` system call. The C library takes the value returned by `write()` and passes it back to the user program:



LINKERS AND LOADERS

- Source code compiled into object files designed to be loaded into any physical memory location – **relocatable object file**
- **Linker** combines these into single binary **executable** file
  - Also brings in libraries
- Program resides on secondary storage as binary executable
- Must be brought into memory by **loader** to be executed
  - **Relocation** assigns final addresses to program parts and adjusts code and data in program to match those addresses
- Modern general purpose systems don't link libraries into executables
  - Rather, **dynamically linked libraries** (in Windows, **DLLs**) are loaded as needed, shared by all that use the same version of that same library (loaded once)
- Object, executable files have standard formats, so operating system knows how to load and start them

THE ROLE OF THE LINKER AND LOADER



WHY APPLICATIONS ARE OPERATING SYSTEM SPECIFIC

- Apps compiled on one system usually not executable on other operating systems
- Each operating system provides its own unique system calls
  - Own file formats, etc.
- Apps can be multi-operating system
  - Written in interpreted language like Python, Ruby, and interpreter available on multiple operating systems
  - App written in language that includes a VM containing the running app (like Java)
  - Use standard language (like C), compile separately on each operating system to run on each
- **Application Binary Interface** (**ABI**) is architecture equivalent of API, defines how different components of binary code can interface for a given operating system on a given architecture, CPU, etc.

DESIGN AND IMPLEMENTATION

- Design and Implementation of OS is not "solvable", but some approaches have proven successful
- Internal structure of different Operating Systems can vary widely
- Start the design by defining goals and specifications
- Affected by choice of hardware, type of system
- **User** goals and **System** goals
  - User goals – operating system should be convenient to use, easy to learn, reliable, safe, and fast
  - System goals – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient
- Specifying and designing an OS is highly creative task of **software engineering**

IMPLEMENTATION

- Much variation
  - Early OSes in assembly language
  - Then system programming languages like Algol, PL/1
  - Now C, C++
- Actually usually a mix of languages
  - Lowest levels in assembly
  - Main body in C
  - Systems programs in C, C++, scripting languages like PERL, Python, shell scripts
- More high-level language easier to **port** to other hardware
  - But slower
- **Emulation** can allow an OS to run on non-native hardware