Operating Systems – EDA093/DIT401

Introduction to Operating Systems

Vincenzo Gulisano
vincenzo.gulisano@chalmers.se

UNIVERSITY OF
GOTHENBURG

What to read (Main textbook)

- Chapter 1
  - Especially: 1.1, 1.3, 1.6, 1.7

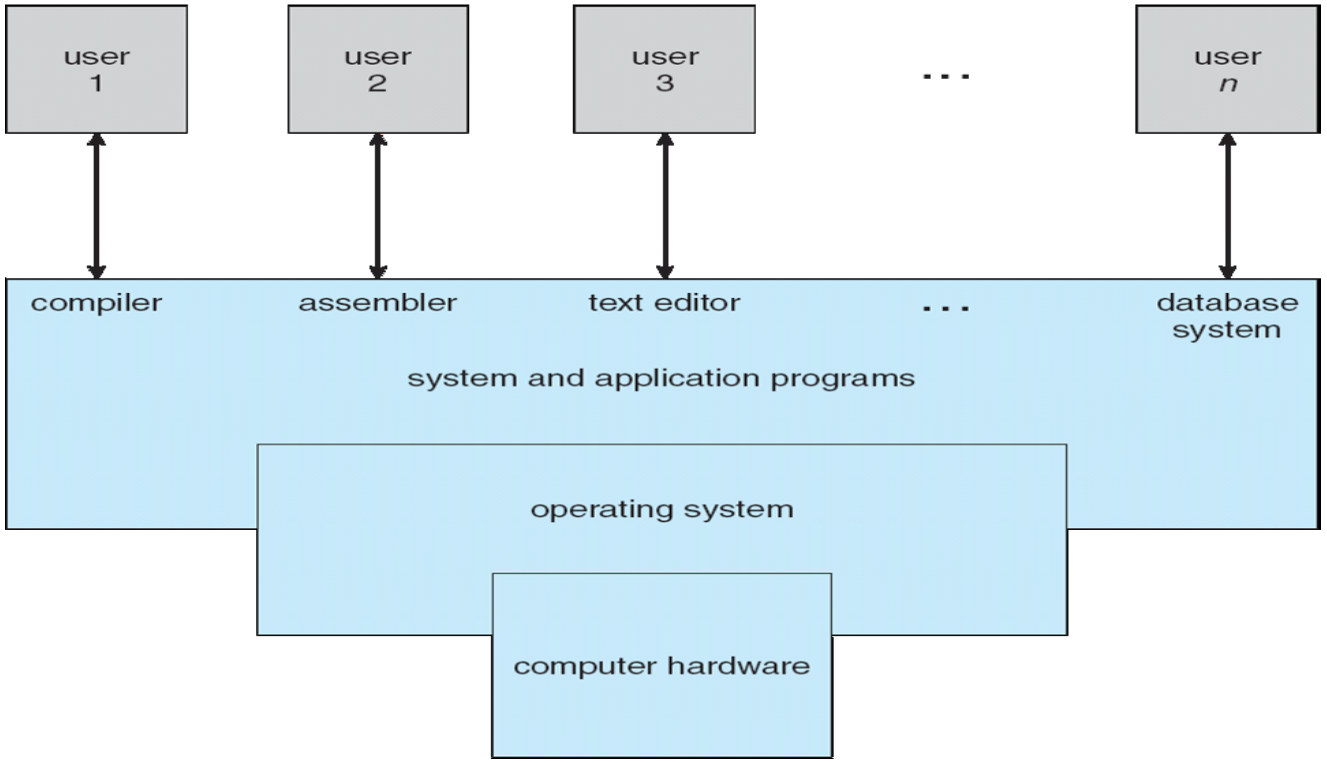
Agenda

- What is an Operating System?
- The event-driven (interrupt-drive) life of an OS
- System calls (overview)
- Dual-Mode and Multimode Operation
- Operating system services
- System calls (continued) and APIs [Self reading]
- System boot [Self reading]
- Operating systems structures

Agenda

- What is an Operating System?
- The event-driven (interrupt-drive) life of an OS
- System calls (overview)
- Dual-Mode and Multimode Operation
- Operating system services
- System calls (continued) and APIs [Self reading]
- System boot [Self reading]
- Operating systems structures

Components of a Computer System



- Hardware – basic computing resources: CPU, memory, I/O devices
- **Operating system: what is?**
- Application programs: Word processors, compilers, web browsers, database systems, video games
- Users: People, machines, other computers

How can we define what an Operating System is and does?

Source: http://www.amazon.com/Apple-MC916LL-Tablet-Black-Generation/dp/B0047DVWZS

Source: http://www.samsung.com/nz/consumer/home-appliances/washing-machines/front-loader/WW90H9600EW/SA

Source: http://linustechtips.com/main/topic/50821-dell-contemplates-shipping-chinese-pcs-with-ubuntu-kylin-os-by-default/

Operating System as Component of a Computer System

- Provides a set of services to system users (collection of service programs)
- Shields between the user and the hardware

- Resource manager:
  - CPU(s)
  - memory and I/O devices
- A control program
  - Controls execution of programs to prevent errors and improper use of the computer

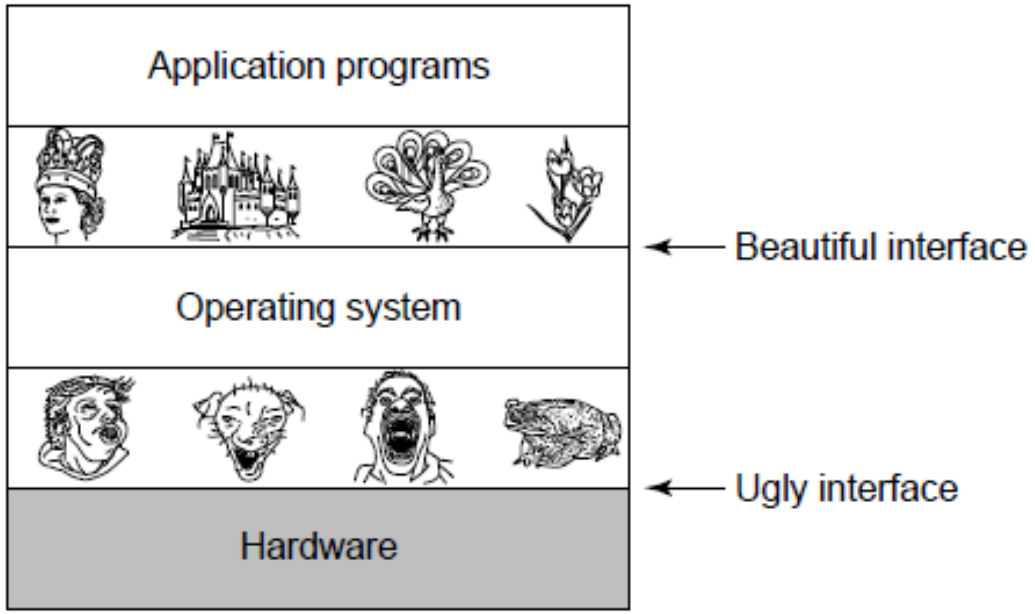


Fig: Modern OS, A. Tanenbaum

Agenda

- What is an Operating System?
- The event-driven (interrupt-drive) life of an OS
- System calls (overview)
- Dual-Mode and Multimode Operation
- Operating system services
- System calls (continued) and APIs [Self reading]
- System boot [Self reading]
- Operating systems structures

How does the OS manage devices and users?

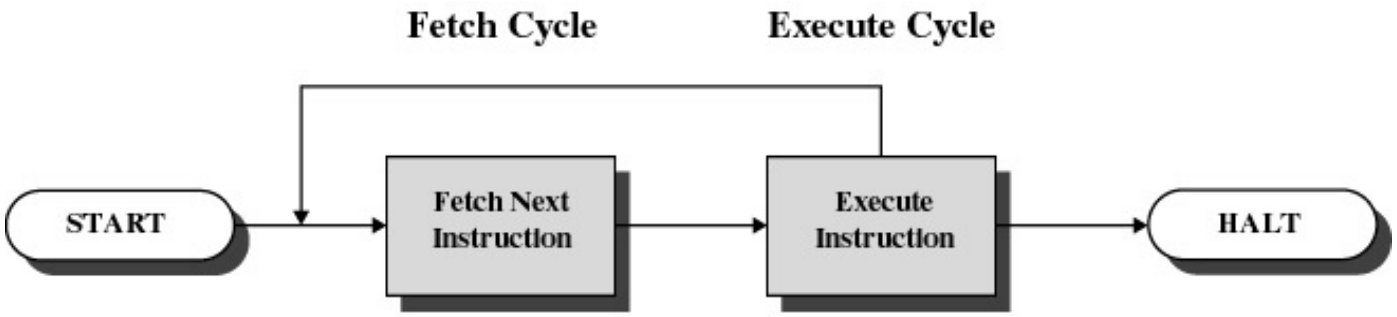
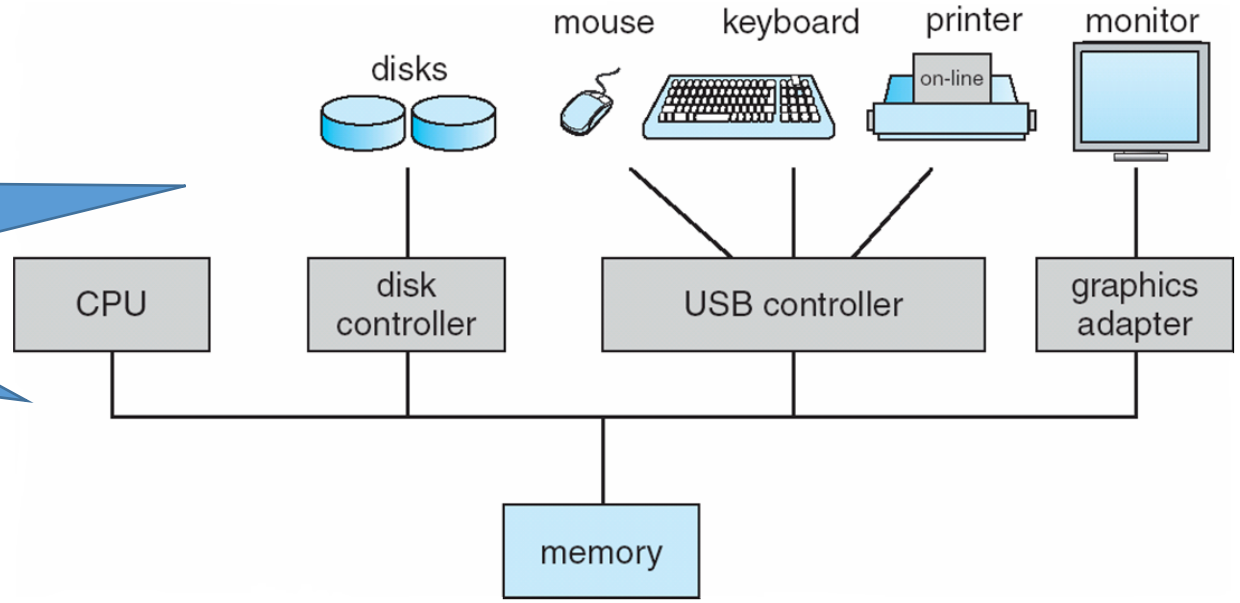


Fig: Intro to OS, W. Stallings

Events (interrupts) mark the cycles of the OS!



Events / Interrupts are the basic “ticks” of the OS cycles

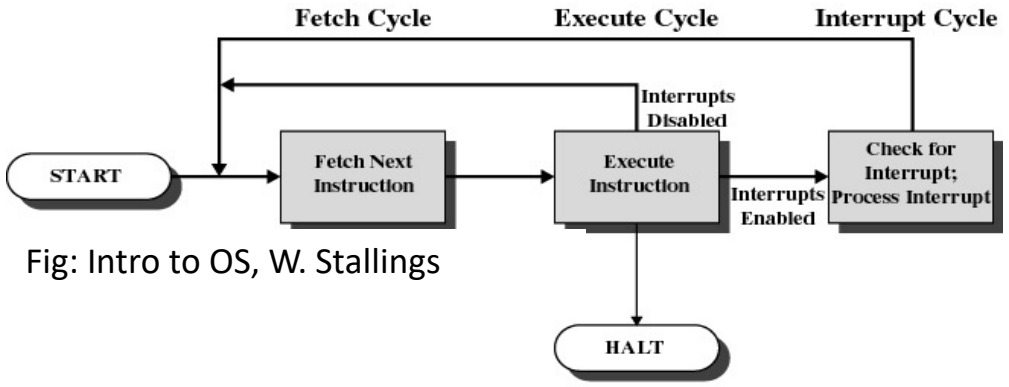


Fig: Intro to OS, W. Stallings

OS doing some work

Interrupt!

What’s happening?

Key stroke! Need to print character on screen…

Doing that…

Going back to my previous work…

Interrupt!

What’s happening?

Byte copied to disk!

OK then copying another byte…

Interrupt!

…

Agenda

- What is an Operating System?
- The event-driven (interrupt-drive) life of an OS
- System calls (overview)
- Dual-Mode and Multimode Operation
- Operating system services
- System calls (continued) and APIs [Self reading]
- System boot [Self reading]
- Operating systems structures

A deeper look at the OS…

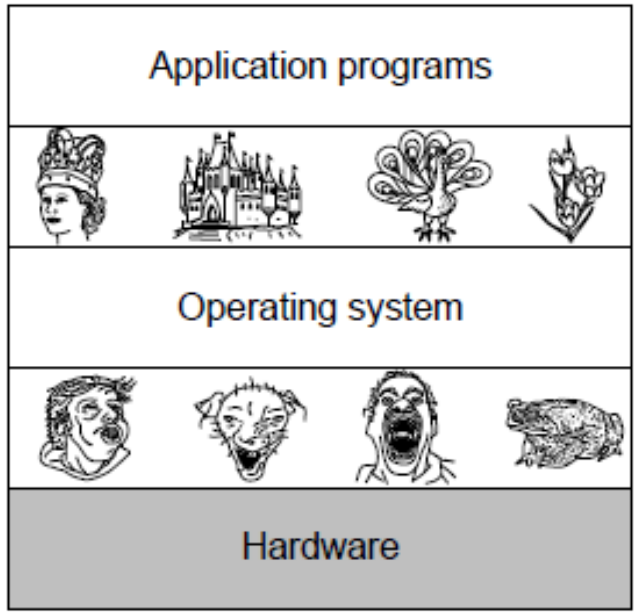
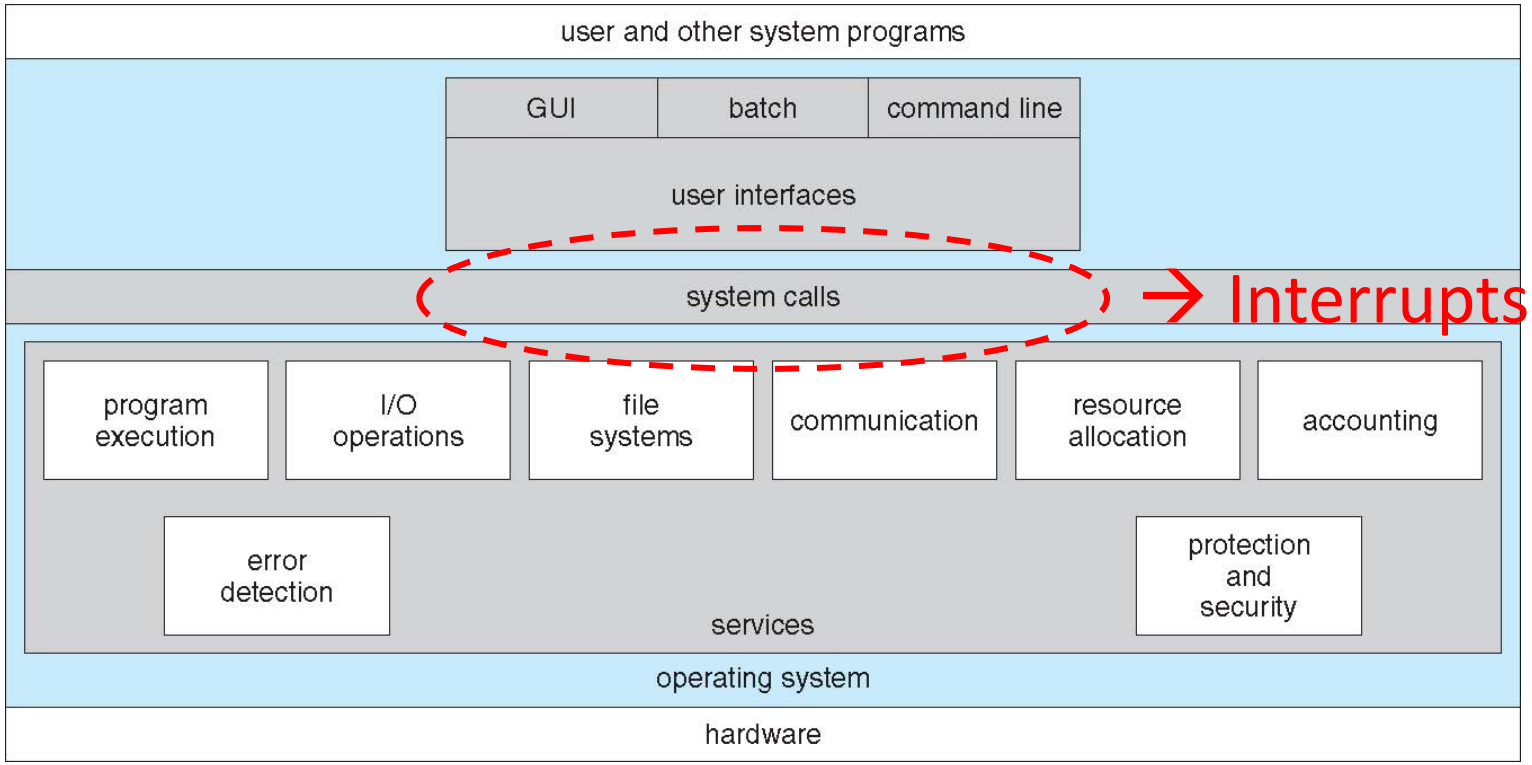


Fig: Modern OS, A. Tanenbaum

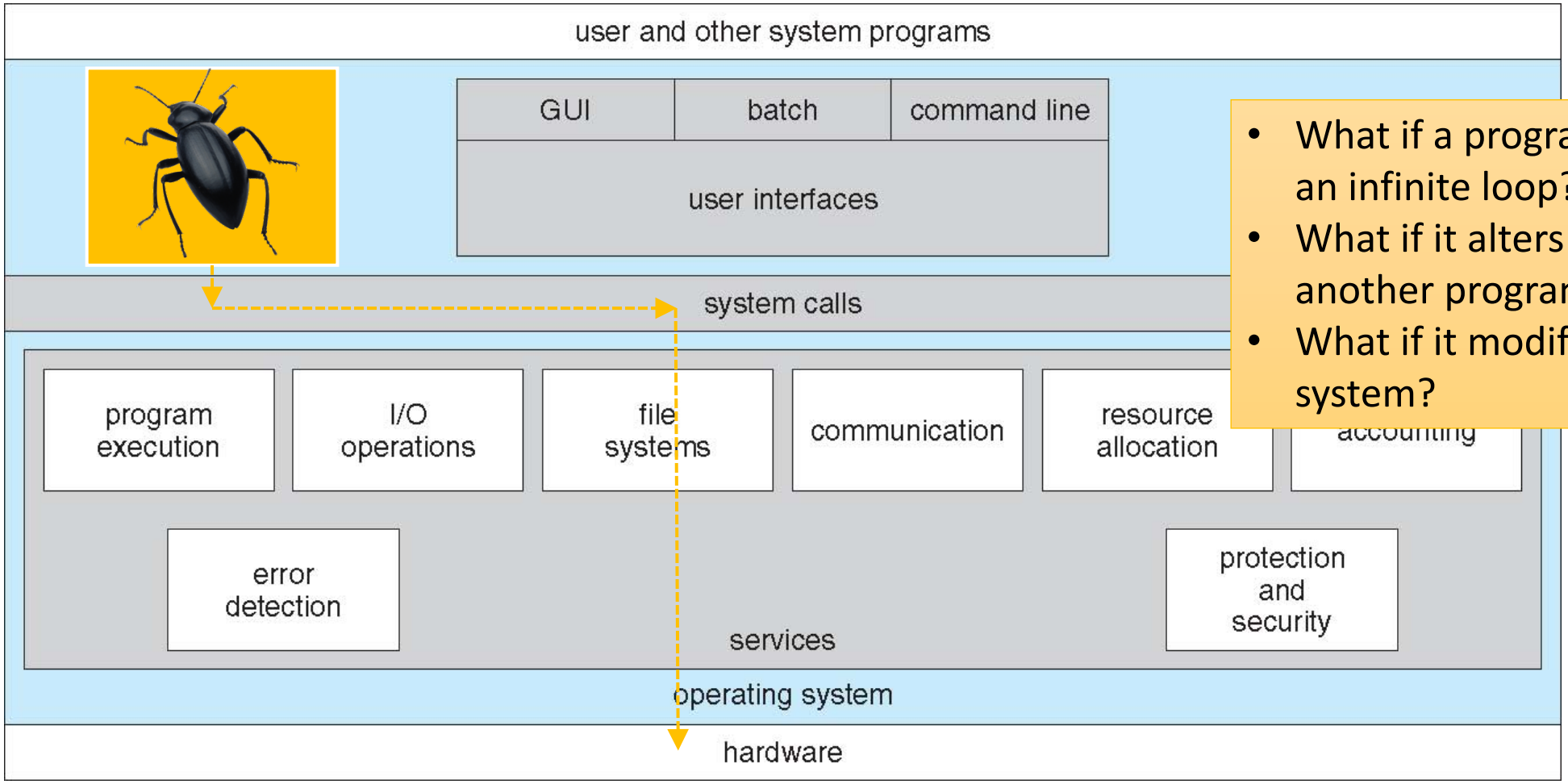


How does the OS allow users and programs to use and interface with the hardware and the resources of a computer?

Agenda

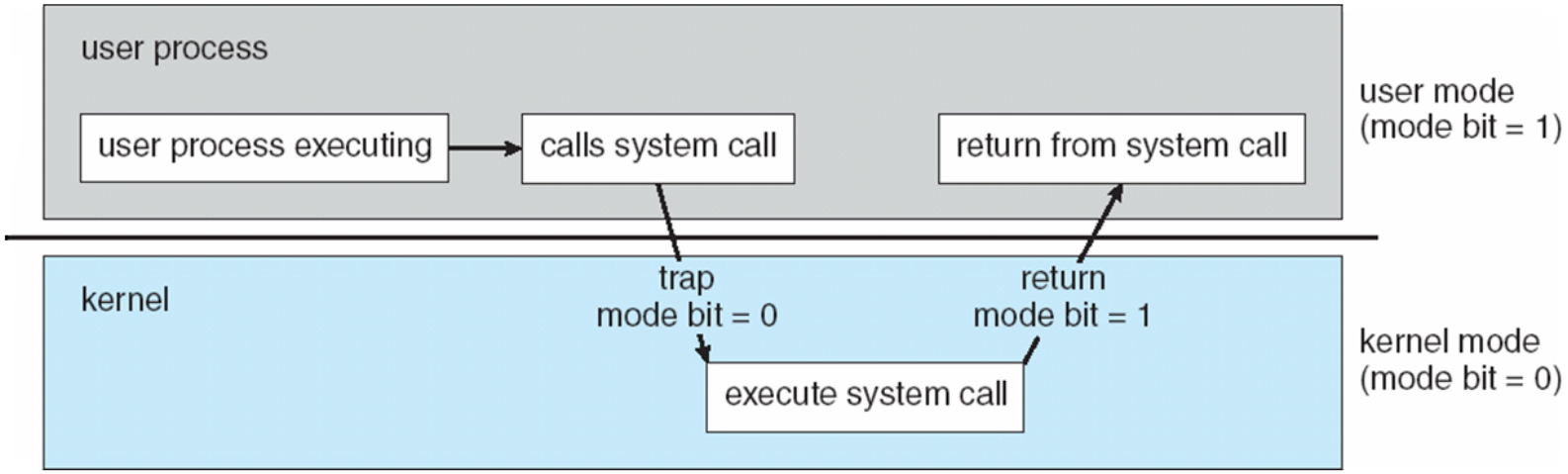
- What is an Operating System?
- The event-driven (interrupt-drive) life of an OS
- System calls (overview)
- Dual-Mode and Multimode Operation
- Operating system services
- System calls (continued) and APIs [Self reading]
- System boot [Self reading]
- Operating systems structures

Wait a moment…



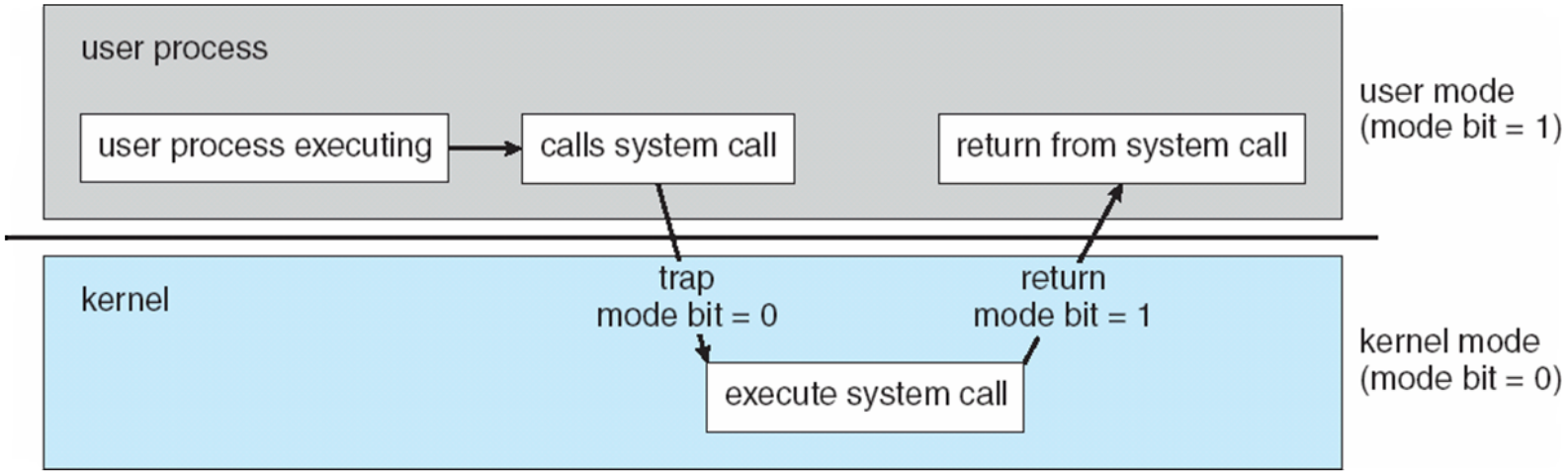
- What if a program gets stuck in an infinite loop?
- What if it alters the data of another program?
- What if it modifies the operating system?

Dual-mode and Multimode Operations (I)



- Two different modes: User mode / Kernel mode
- A mode bit (hardware) differentiates between the two
- Is it a privileged instruction?
  - Can only execute in kernel mode (mode bit = 0)
- Running an user program?
  - First switch mode bit to 1… then run the code

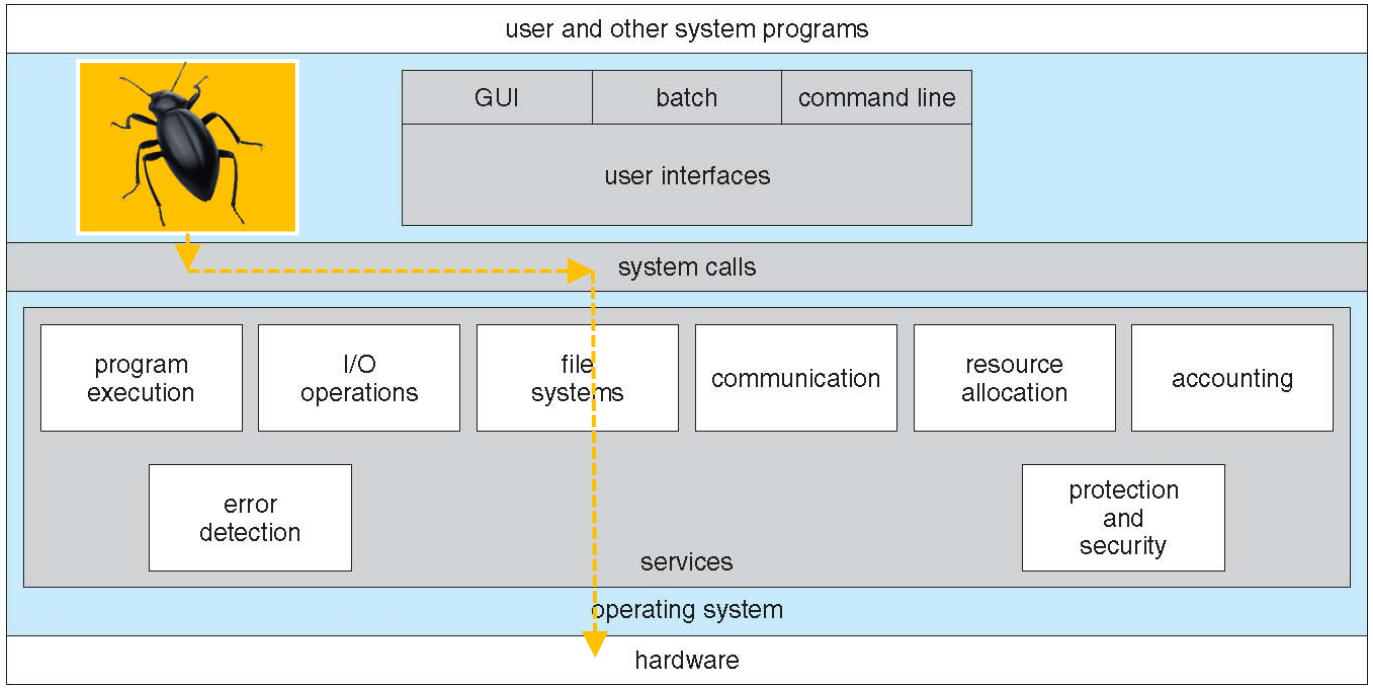
Dual-mode and Multimode Operations (II)



- More than 2 modes can exist
  - Example: to allow a Virtual Machine Manager (VMM) to execute more instructions than an user program
  - Example: Intel 64 CPUs allow for 4 privilege levels
- If a user program tries to execute a privileged instruction
  - The hardware sends an interrupt to the OS
  - The OS terminates the program (or does something else depending on the case)

A special interrupt mechanism: the timer

- When the OS decides to run a user program, how can it be sure the latter will generate an interrupt (in a reasonable amount of time)?

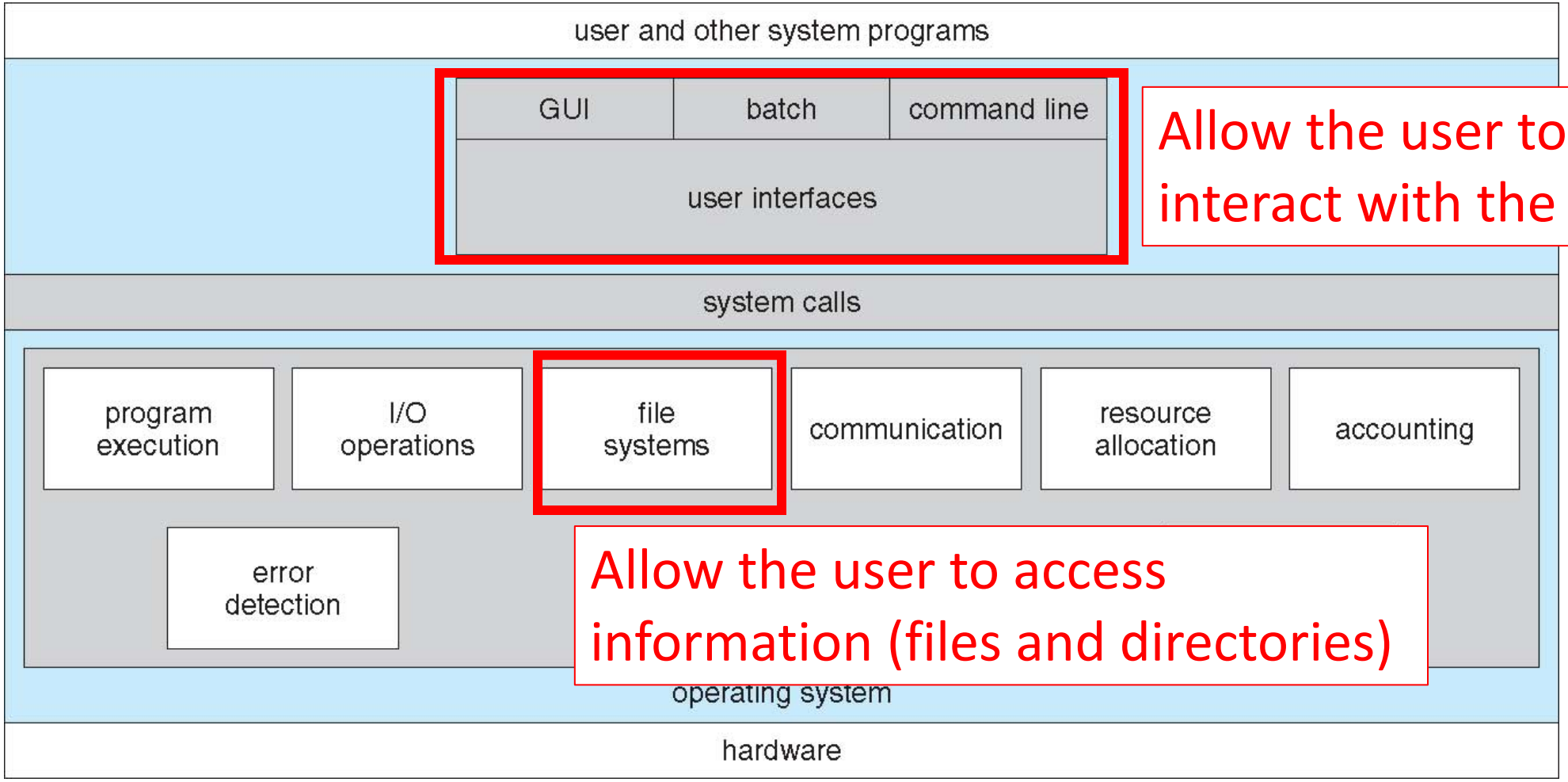


- A timer can be used to periodically raise an interrupt to the OS
- The OS can then take control

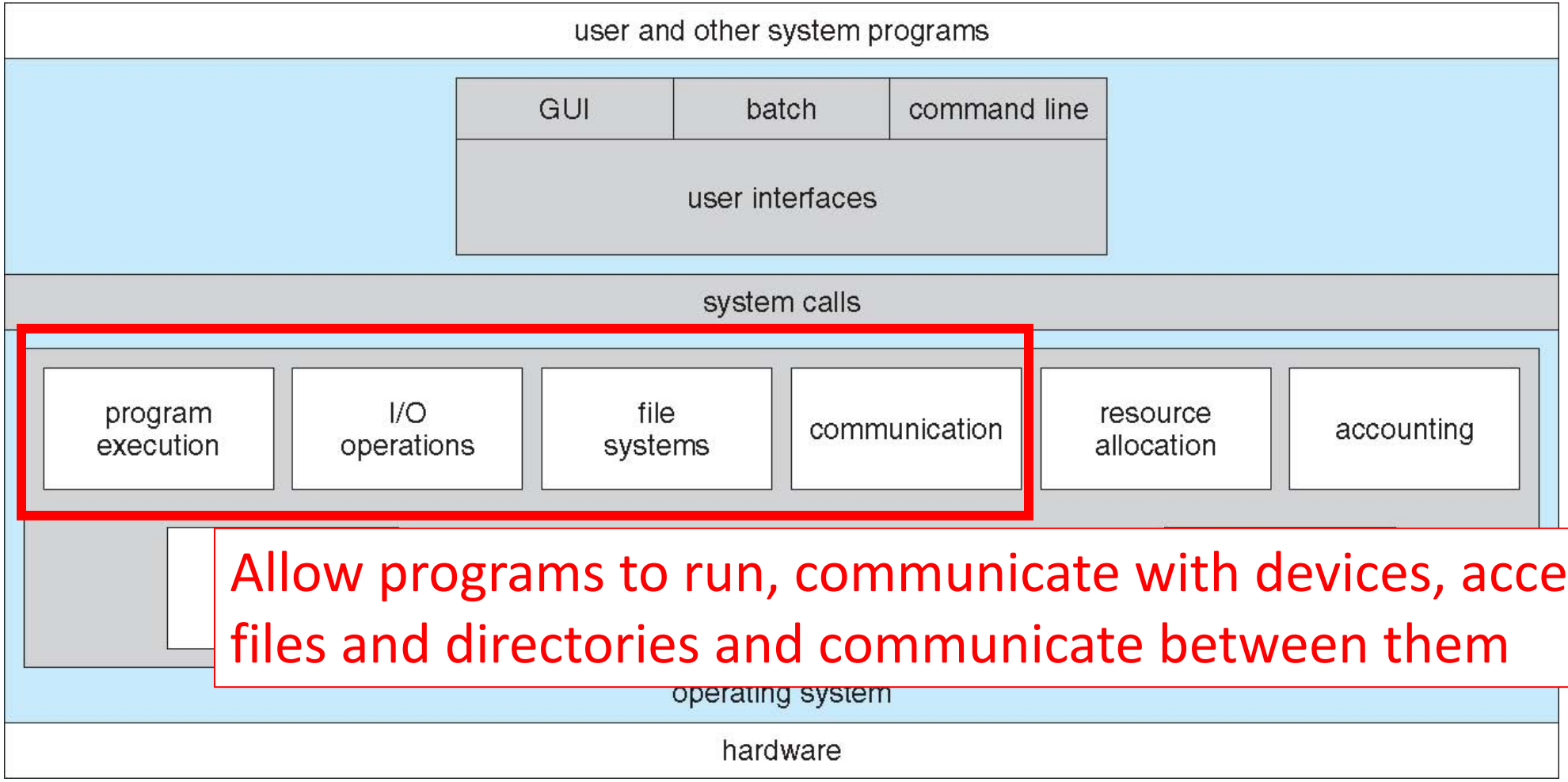
Agenda

- What is an Operating System?
- The event-driven (interrupt-drive) life of an OS
- System calls (overview)
- Dual-Mode and Multimode Operation
- Operating system services
- System calls (continued) and APIs [Self reading]
- System boot [Self reading]
- Operating systems structures

What does the operating system provides?

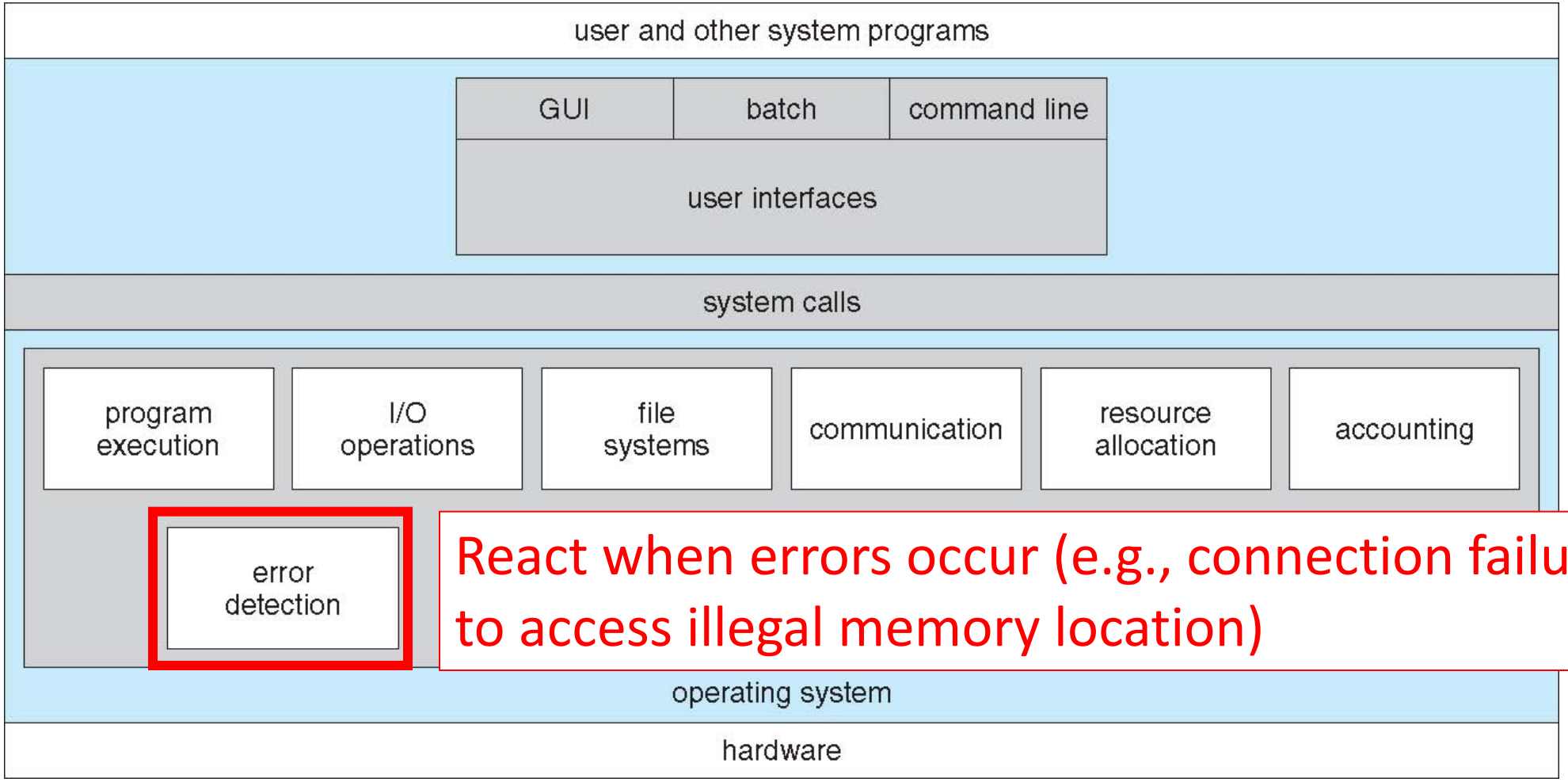


What does the operating system provides?



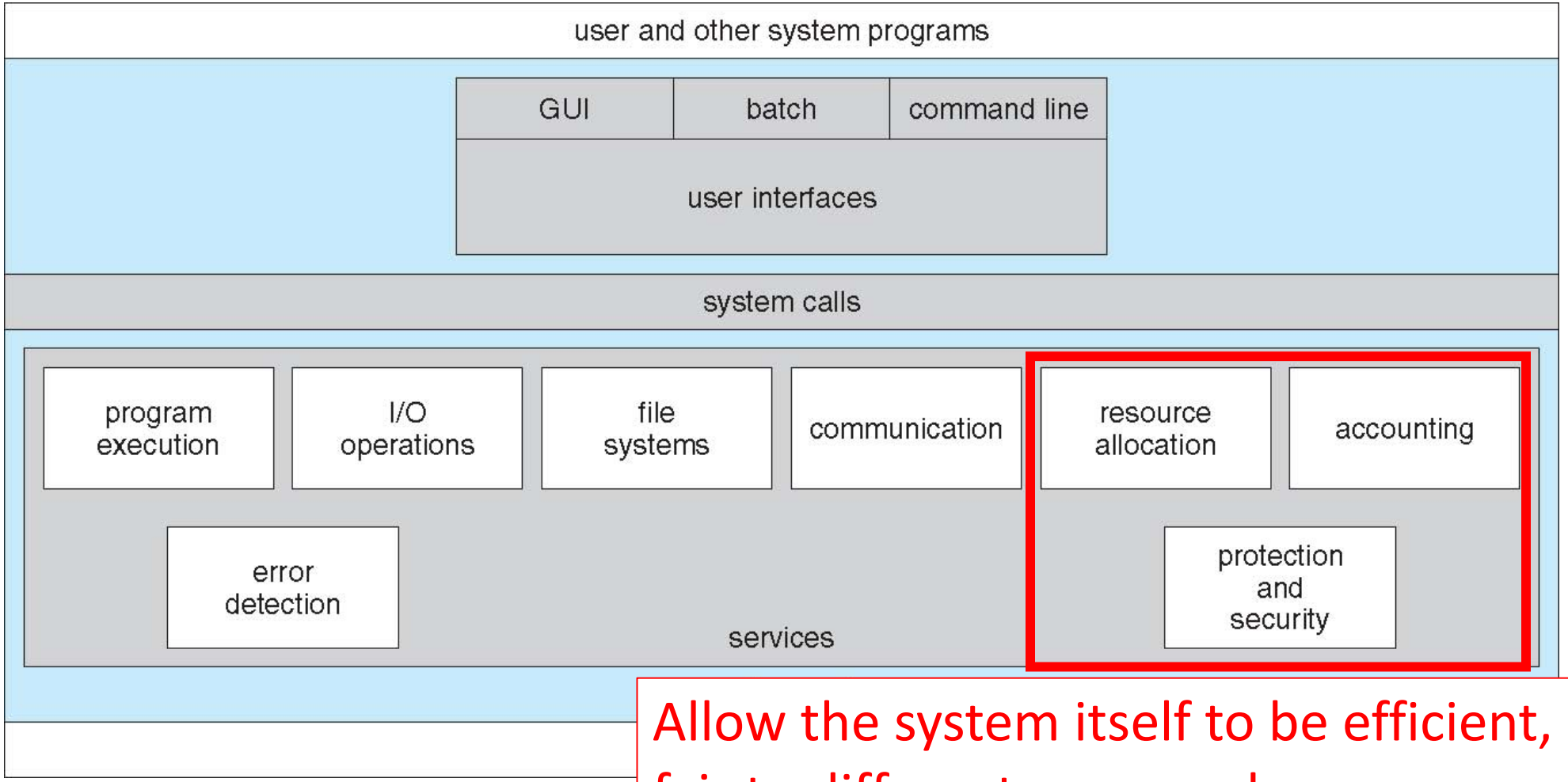
Allow programs to run, communicate with devices, access files and directories and communicate between them

What does the operating system provides?



React when errors occur (e.g., connection failure, attem
to access illegal memory location)

What does the operating system provides?

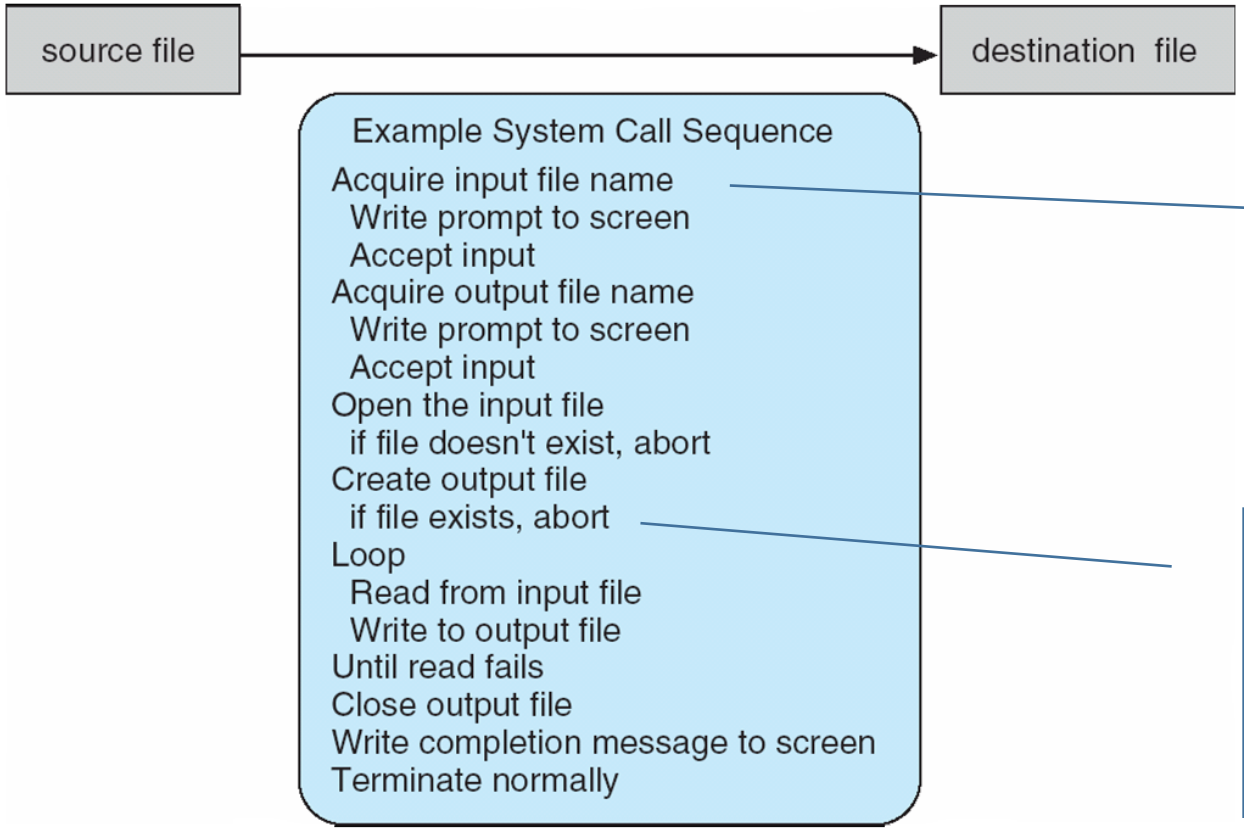


Allow the system itself to be efficient, fair to different users and secure

Agenda

- What is an Operating System?
- The event-driven (interrupt-drive) life of an OS
- System calls (overview)
- Dual-Mode and Multimode Operation
- Operating system services
- System calls (continued) and APIs [self reading]
- System boot [Self reading]
- Operating systems structures

Let’s say we want to copy a file…



If a graphical menu is used to let the user pick a source file, many more system calls are needed (and I/O)

We could also decide to replace it (asking first the user)… More system calls!!!

Application Programming Interfaces (APIs)

- APIs provide high-level combination of low-level system calls
  - Reducing by some order of magnitude the number of low-level system calls required to perform some operations (usually thousands of system calls per second are performed by the OS)

- What are the advantages?
  - Portability
  - Simplicity

Sample 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:

```
#include <unistd.h>

ssize_t    read(int fd, void *buf, size_t count)
```

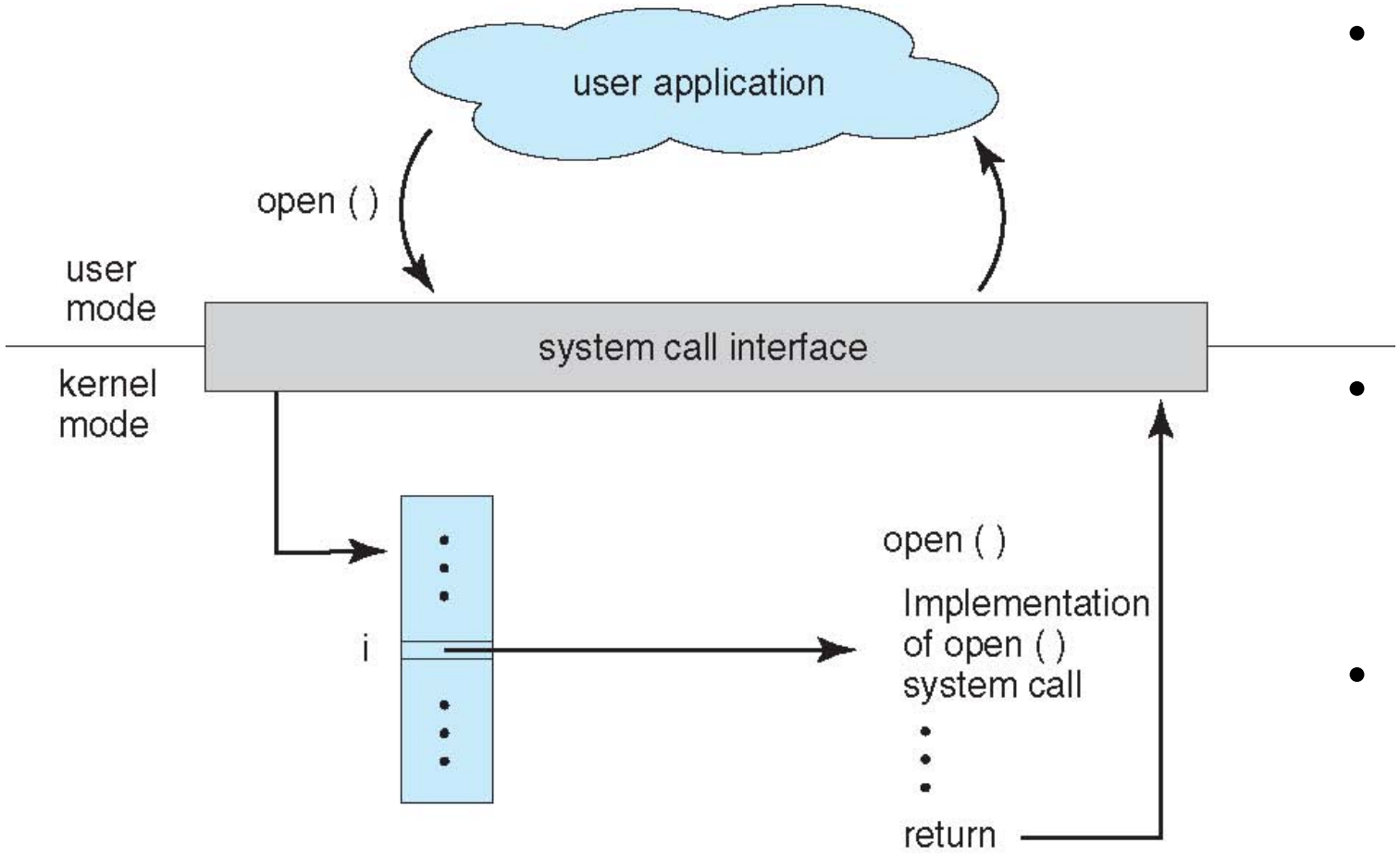
return value — function name — parameters

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 where the data will be read into
- `size_t count`—the maximum number of bytes to be read into the buffer

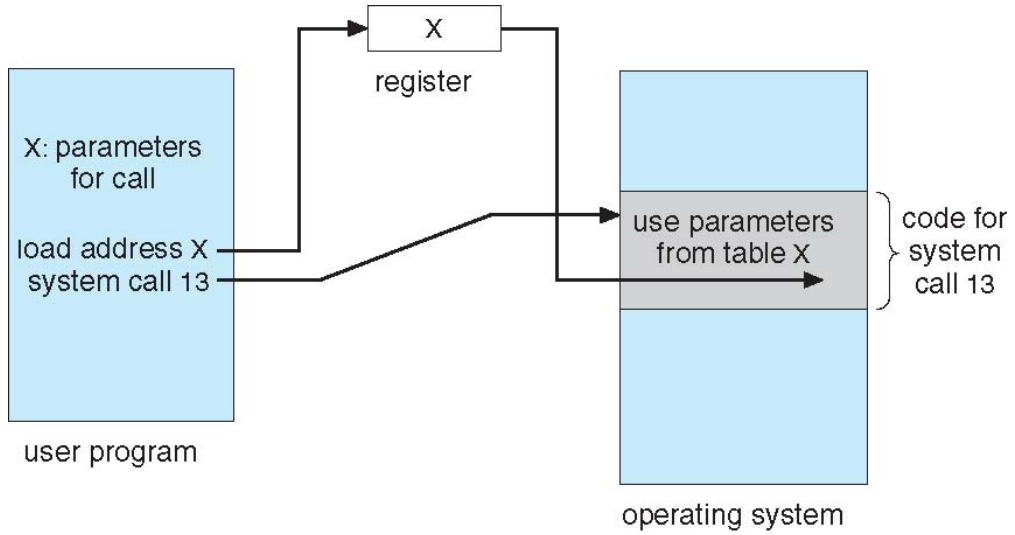
On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns −1.

System call interface



- Typically, a number 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

How to pass parameters to a system call?



- 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

Types of system calls (I)

- 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

Types of system calls (II)

- 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 (III)

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

Types of system calls (IV)

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

Some examples

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

Agenda

- What is an Operating System?
- The event-driven (interrupt-drive) life of an OS
- System calls (overview)
- Dual-Mode and Multimode Operation
- Operating system services
- System calls (continued) and APIs [Self reading]
- System boot [self reading]
- Operating systems structures

How does the computer know where / how to lead the kernel at startup?

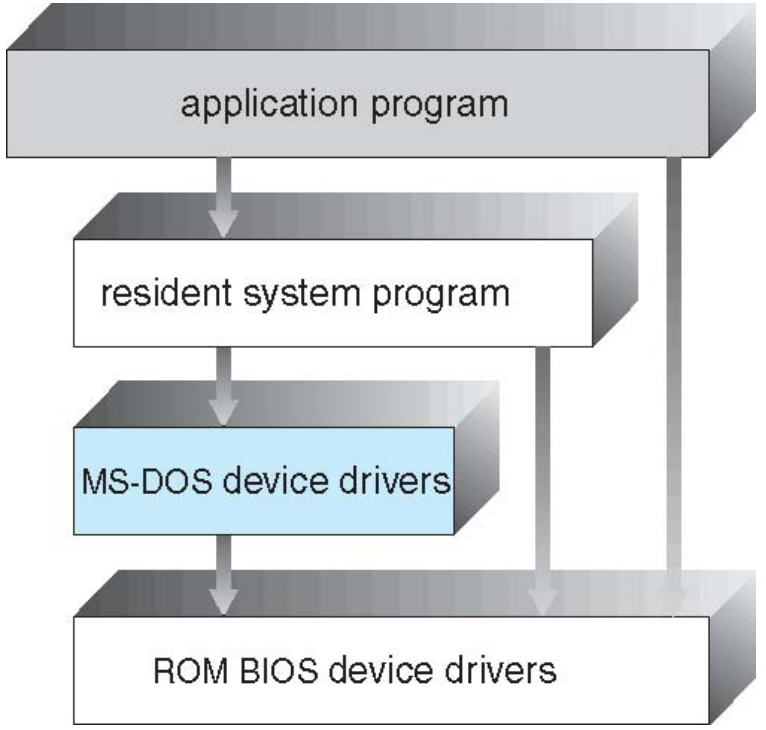
- When power initialized on system, execution starts at a fixed memory location
  - Firmware ROM used to hold initial boot code
- Operating system must be made available to hardware so hardware can start it
  - Small piece of code – **bootstrap loader**, stored in **ROM** or **EEPROM** locates the kernel, loads it into memory, and starts it
  - Sometimes two-step process where **boot block** at fixed location loaded by ROM code, which loads bootstrap loader from disk
- Common bootstrap loader, **GRUB**, allows selection of kernel from multiple disks, versions, kernel options
- Kernel loads and system is then **running**

Agenda

- What is an Operating System?
- The event-driven (interrupt-drive) life of an OS
- System calls (overview)
- Dual-Mode and Multimode Operation
- Operating system services
- System calls (continued) and APIs [Self reading]
- System boot [Self reading]
- Operating systems structures

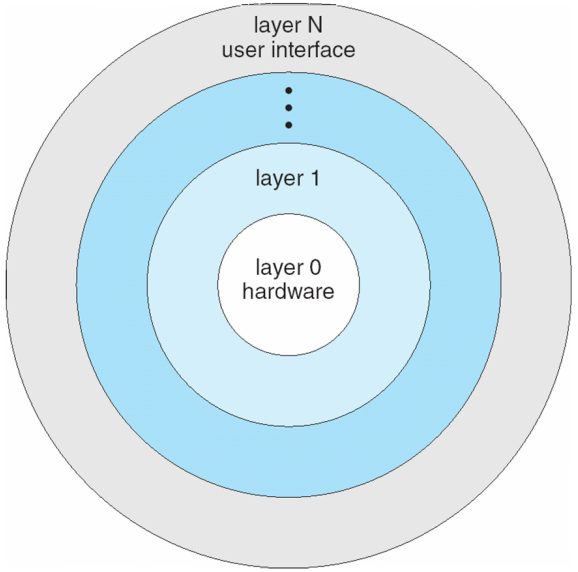
Operating Systems structure

- Simple Structure
  - Monolithic structure
  - Designed for hardware without dual mode
  - Vulnerable to wrong/malicious code

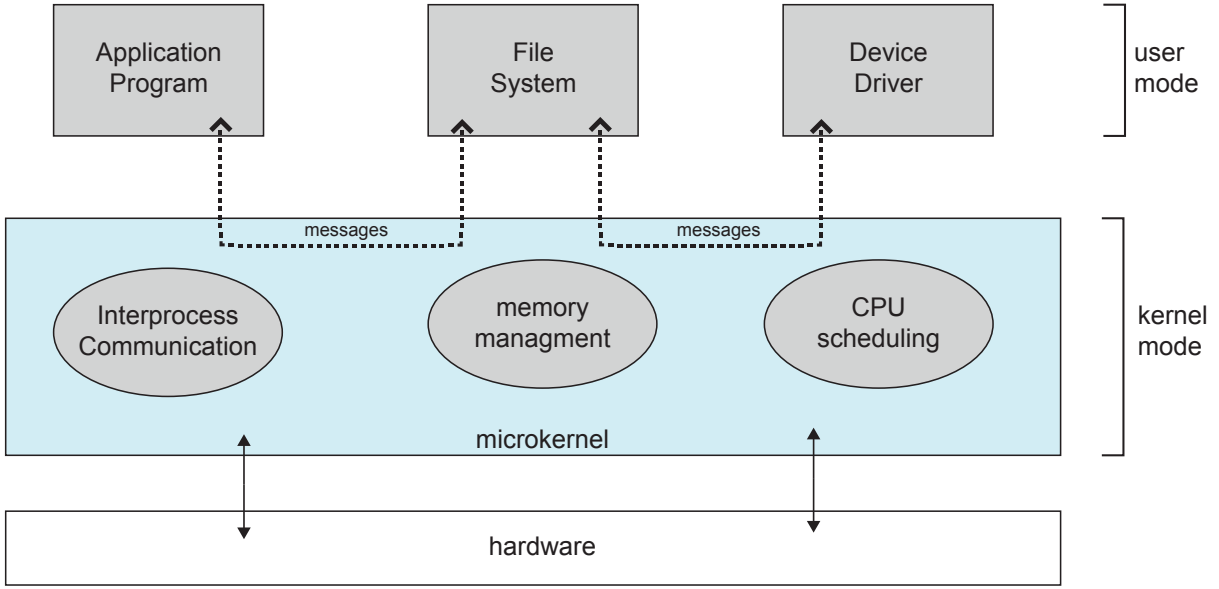


Operating Systems structure

- Simple Structure
- Layered Approach
  - Modular approach
  - Each layer maintains some data structures and a set of routines
    - Easier to debug
    - Problematic for “circular references”



Operating Systems structure



- Simple Structure
- Layered Approach
- Microkernels
  - Mach, one of the firsts modularized (micro)kernel
  - Everything that is not essential → system or user level program
    - Essential parts remain in the microkernel
  - Smaller
  - Makes sense if system/user level programs can be updated frequently
  - More secure (most programs are know executed in user mode)
  - Might suffer in performance

Operating Systems structure

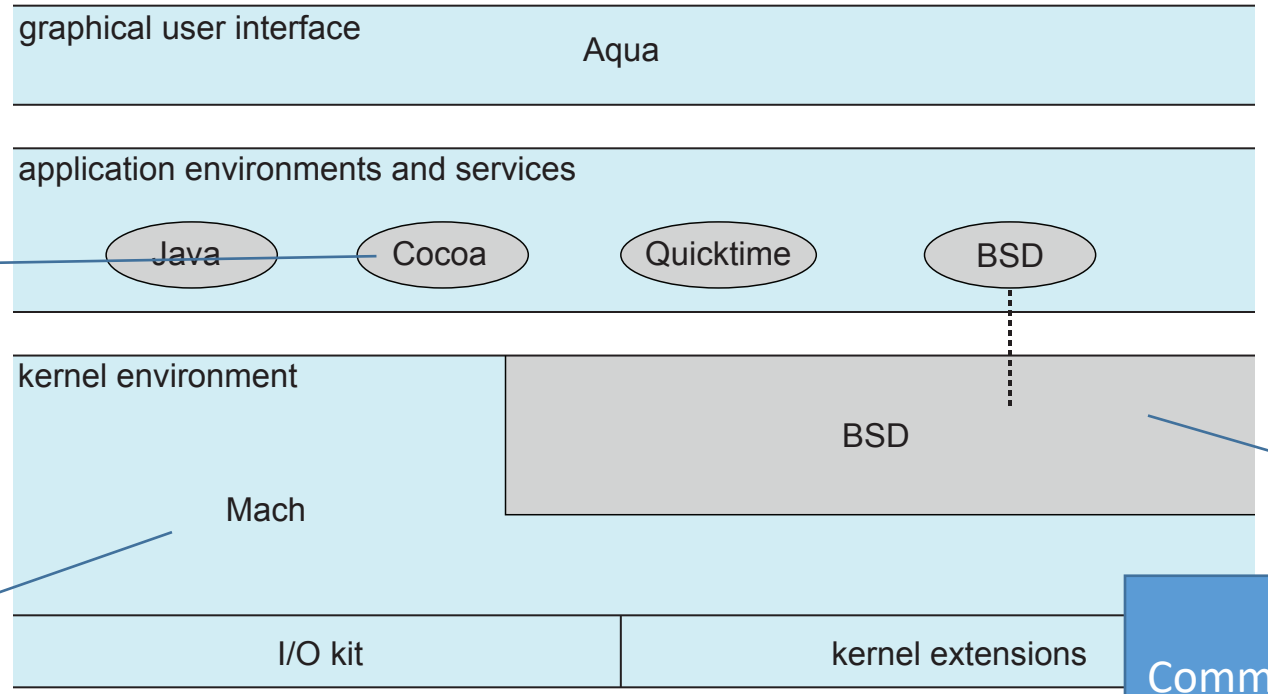
- Simple Structure
- Layered Approach
- Microkernels
- Modules
  - Can be loaded in the kernel (at boot or on demand)
  - UNIX (Solaris, Linux and Mac OS X) and Windows
  - Modules can be loaded without recompiling the kernel

Operating Systems structure

- Simple Structure
- Layered Approach
- Microkernels
- Modules
- Hybrid Systems
  - Mixing other structures
  - Usual case for modern operating systems

Hybrid systems – Mac OS X

Layered structure

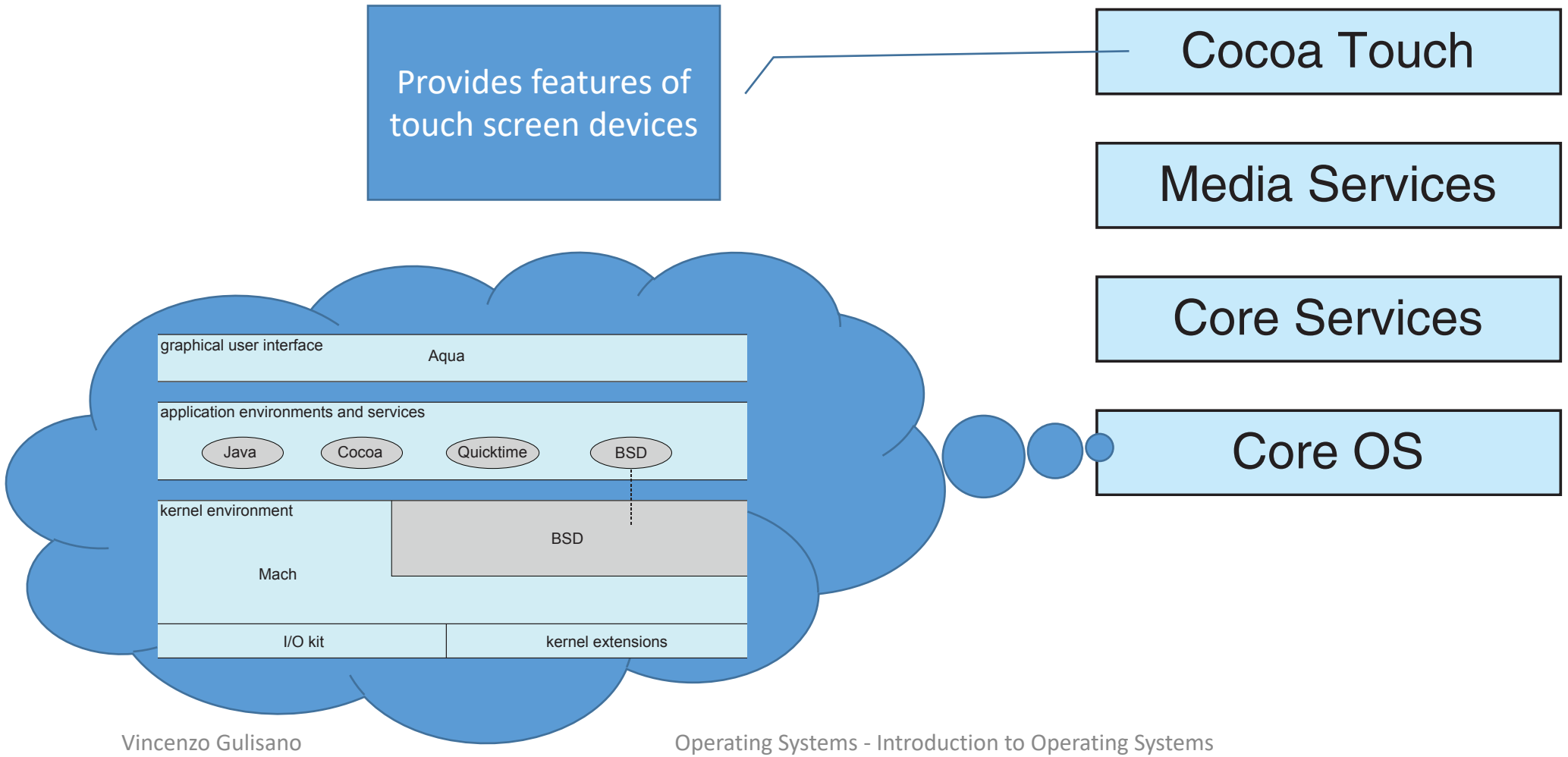


This is for Objective-C programming (Mac OS X applications)

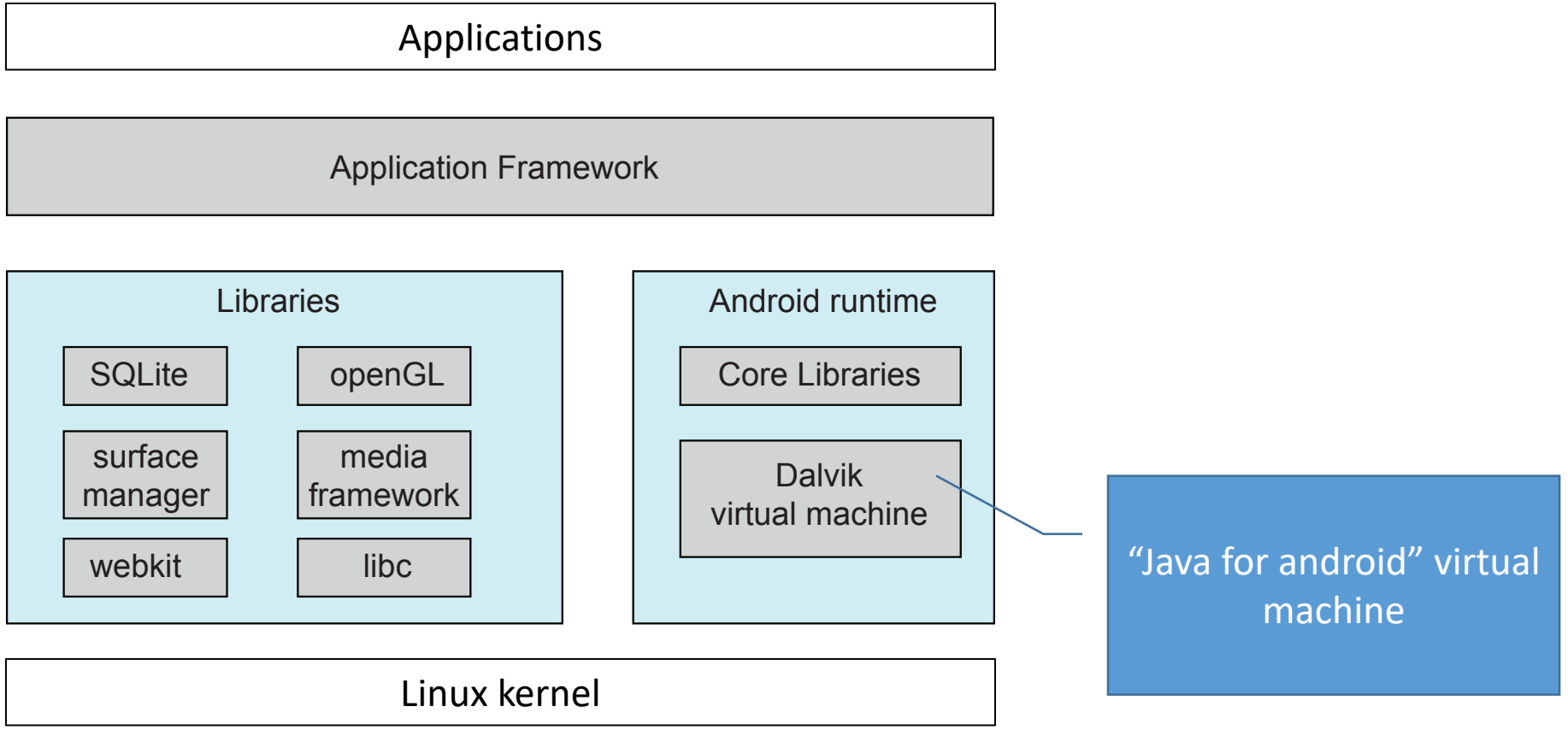
Memory management, RPCs and Inter-Process communication, thread scheduling

Command-line interface, networking, file system

Hybrid systems – iOS



Hybrid systems – Android



Thank you for your attention!

Questions?

For questions and feedback:
https://forms.gle/UCYYnUPudSXy3Yvv9