2. Operating System Structures

ECE30021/ITP30002 Operating Systems

Agenda

- Operating-system services
- Interfaces for users and programmers
- Components and their interconnections
- Virtual Machines
- Design, implementation, generation
- System boot

Objectives

- To describe the services an operating system provides to users, processes, and other systems
- To discuss the various ways of structuring an operating system
- To explain how operating systems are installed and customized and how they boot

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)**, **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

- **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

- 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
      Some (such as CPU cycles, main memory, and file storage) may have special allocation code,
      others (such as I/O devices) may have general request and release code

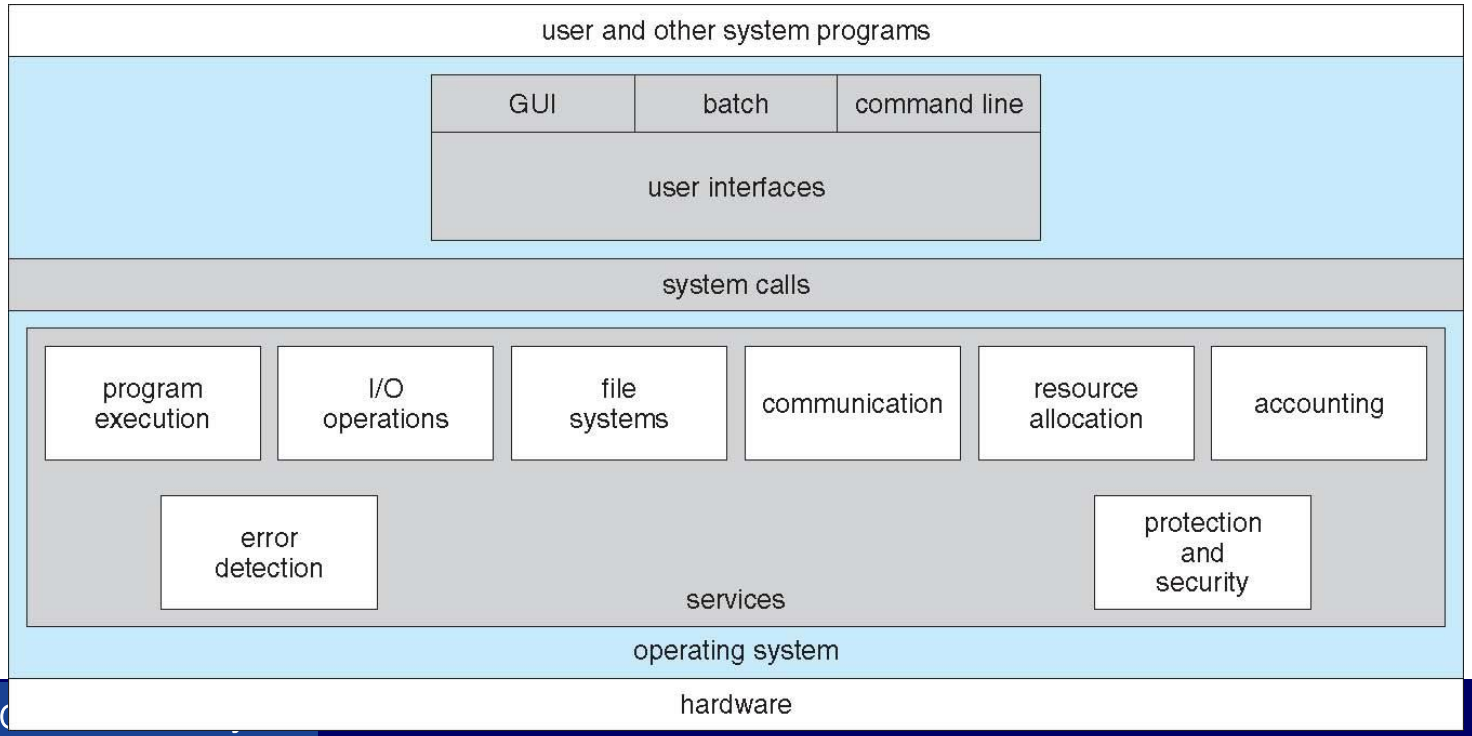
  - **Accounting -** To keep track of which users use how much and what kinds of computer resources

Operating System Services

- **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
  - If a system is to be protected and secure, precautions must be instituted throughout it.

Operating System Services

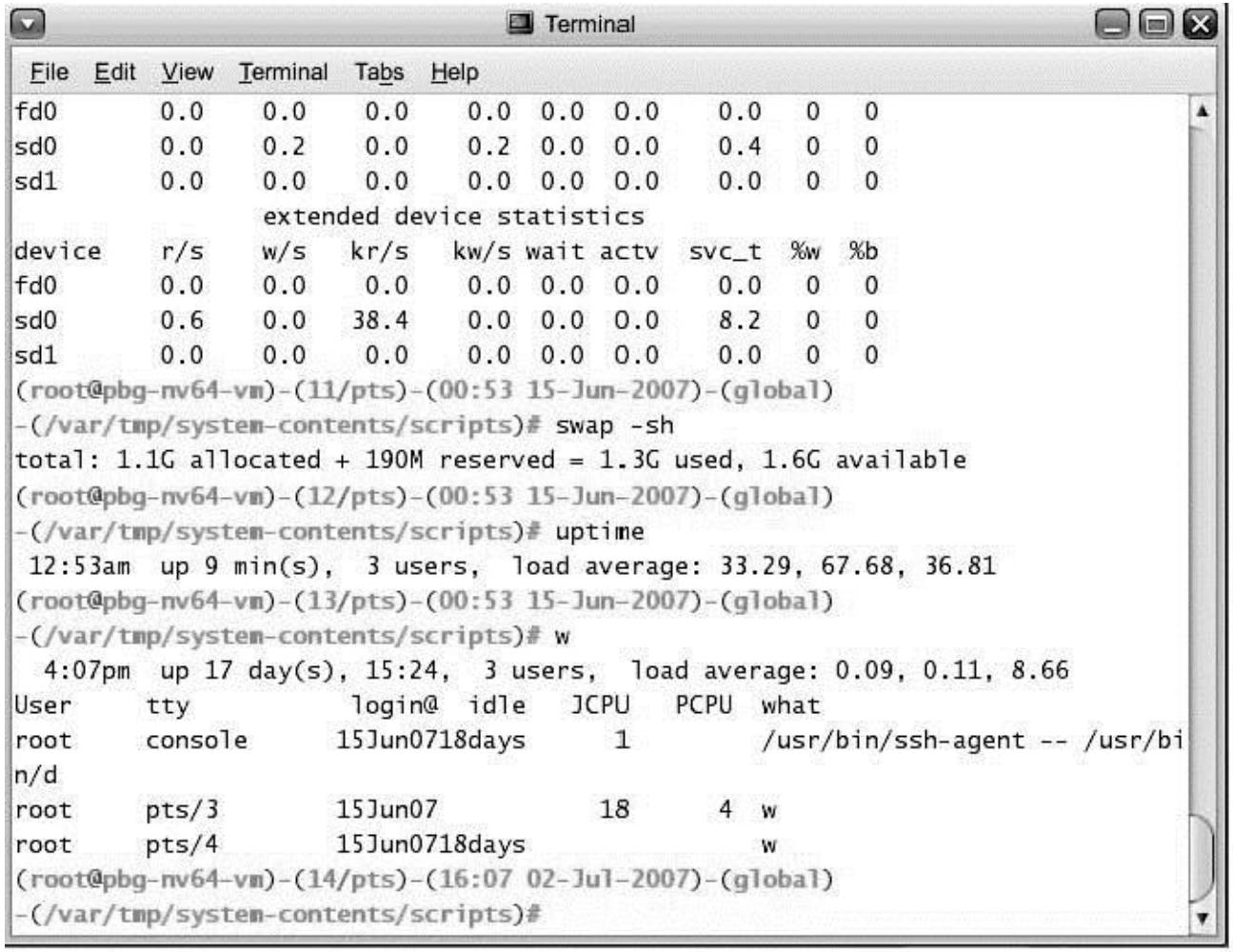
- Services for user
  - User interface
  - Program execution
  - I/O operation
  - File-system manipulation
  - Communications
  - Error detection
- Functions for efficient operation of system itself
  - Resource allocation
  - Accounting
  - Protection and security



OS User Interface – CLI

- CLI or **command interpreter** 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

Command Line Interpreter (CLI)



OS User 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)

The Mac OS X GUI

Touchscreen Interfaces

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

GUI (3D Desktop)

Programming Interfaces

- System calls
  - Primitive programming interface provided through interrupt
  - System-call interface
    - Connection between program language and OS

      Ex) implementations of open(), close(), …

- API (Application Programming Interface)
  - High-level programming interface

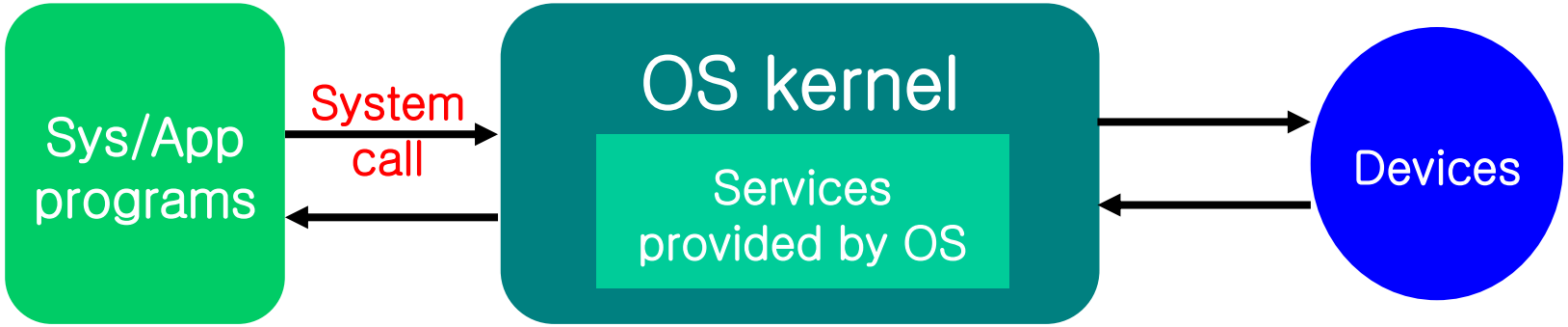
    Ex) MessageBox(…);

  Ex) Win32 API, POSIX API, Java API

System Calls

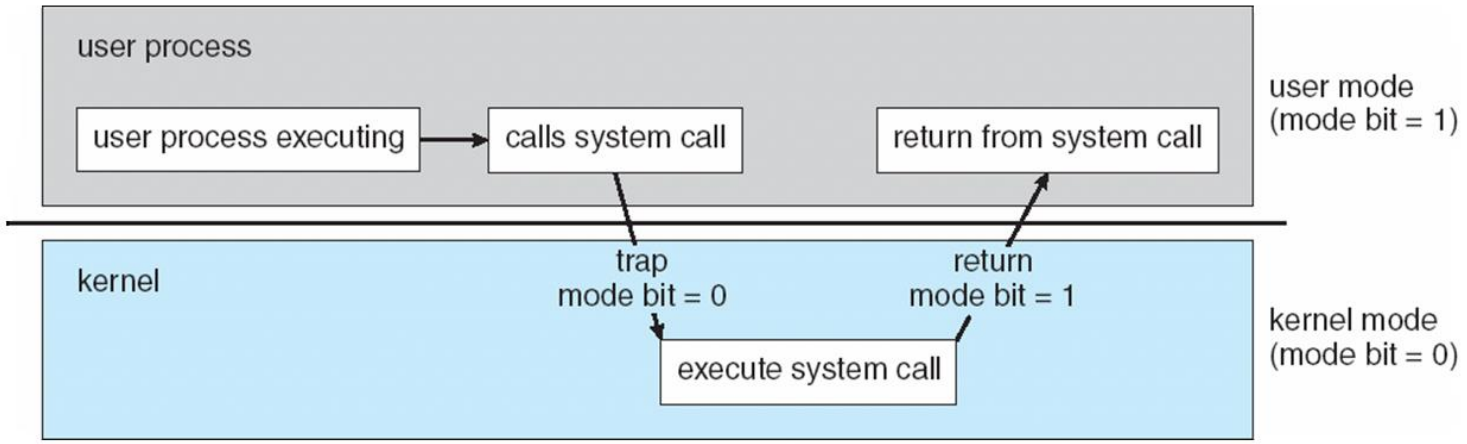
- **System call**: the mechanism used by an application program to request service from OS kernel
  - *"Function calls to OS kernel available through interrupt"*
  - Generally, provided as interrupt handlers written in C/C++ or assembly.
  - A mechanism to transfer control safely from lesser privileged modes to higher privileged modes.

  Ex) POSIX system calls: open, close, read, write, fork, kill, wait, …



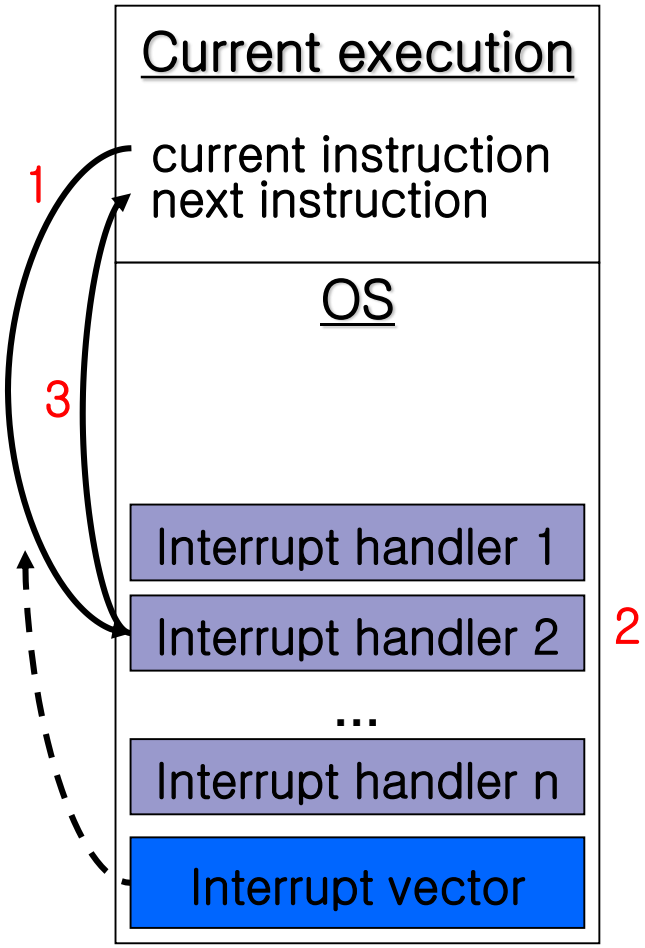
Dual Mode Operation

- User mode
  - User defined code (application)
  - **Privileged instructions, which can cause harm to other system, are prohibited**
    - Privileged instruction can be invoked only through OS system call
- Kernel mode (supervisor mode, system mode, privileged mode)
  - OS code
  - Privileged instructions are permitted



Interrupt Mechanism

- Interrupt handling
  1. CPU stops current work and transfers execution to interrupt handler
     - Interrupt vector: table of interrupt handlers for each types interrupt
  2. Interrupt is handled by corresponding handler
  3. Return to the interrupted program
  - Before interrupt handler is invoked, necessary information should be saved (return address, state)



Parameter Passing in System Call

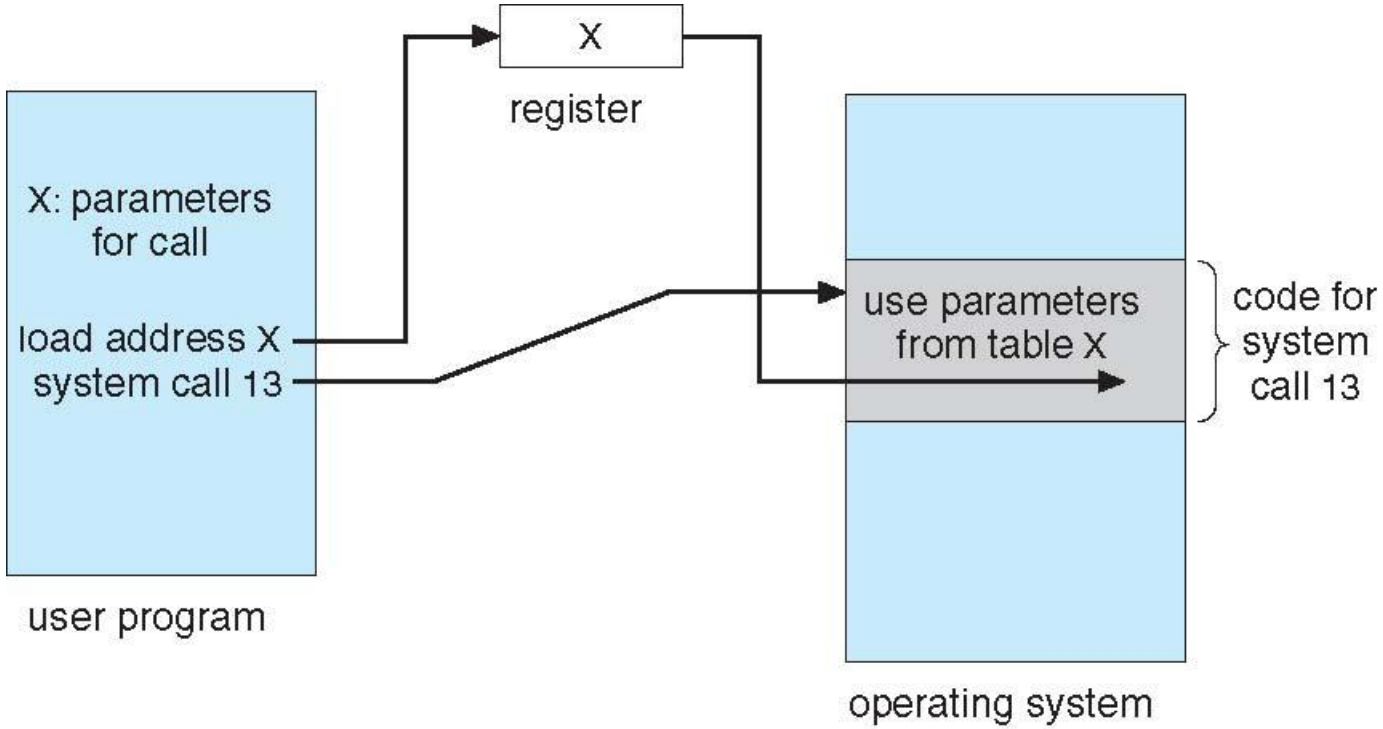
- Often, more information is required than simply identity of desired system call
  - Exact type and amount of information vary according to OS and call

- 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
  - Some OS prefer the block or stack method because they do not limit the number or length of parameters being passed

Parameter Passing in System Call

- Internally, system call is serviced through interrupt
  - Additional information can be necessary
- Parameter passing methods
  - Register (simple information)
  - Address of block (large information)
  - System stack

Passing of parameters as a table

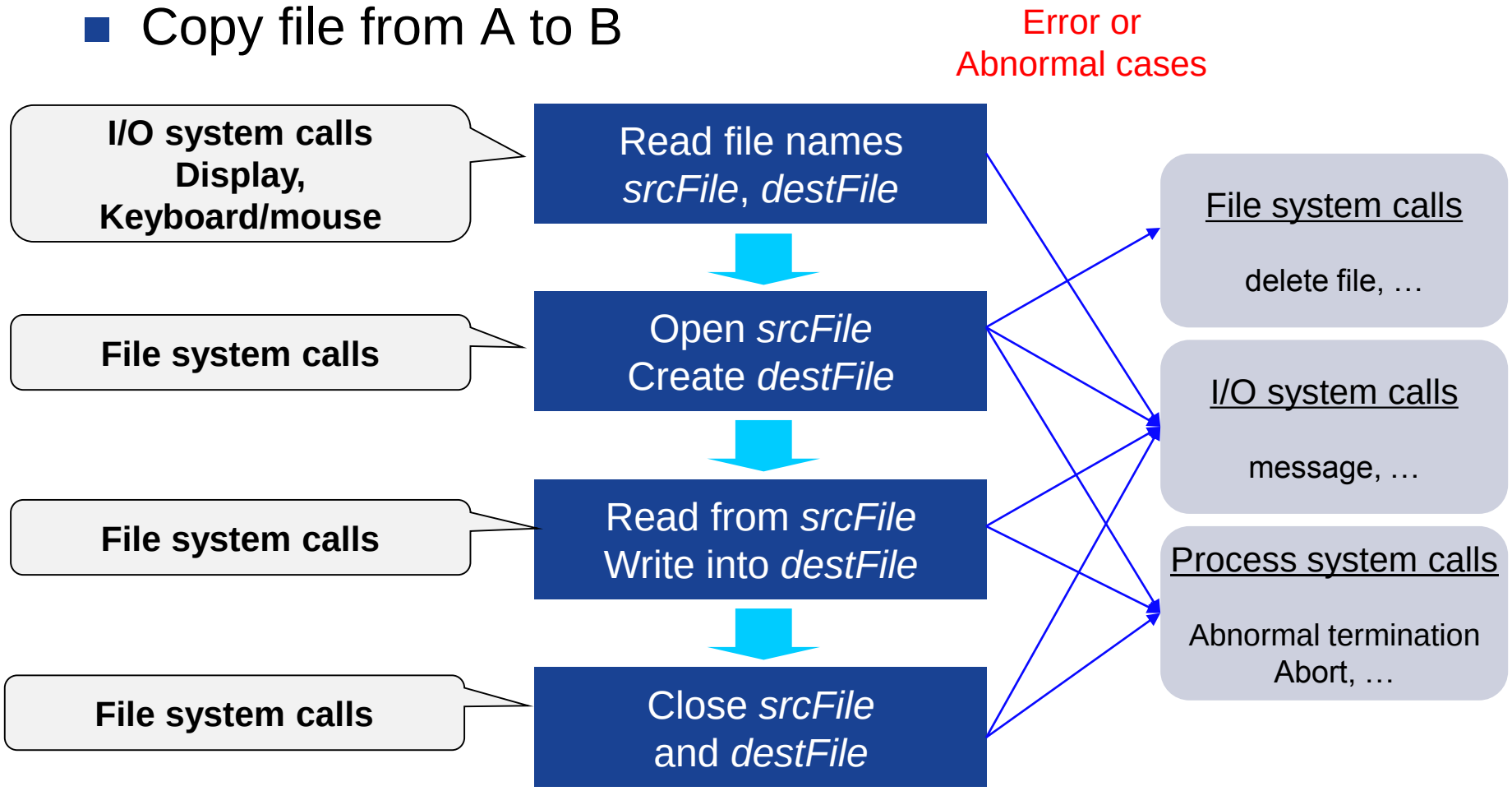


Types of System Calls

- Process control
- File management
- Device management
- Information maintenance
- Communication

Example

- Copy file from A to B



System-Call Interface

- How to invoke system calls in high-level language?

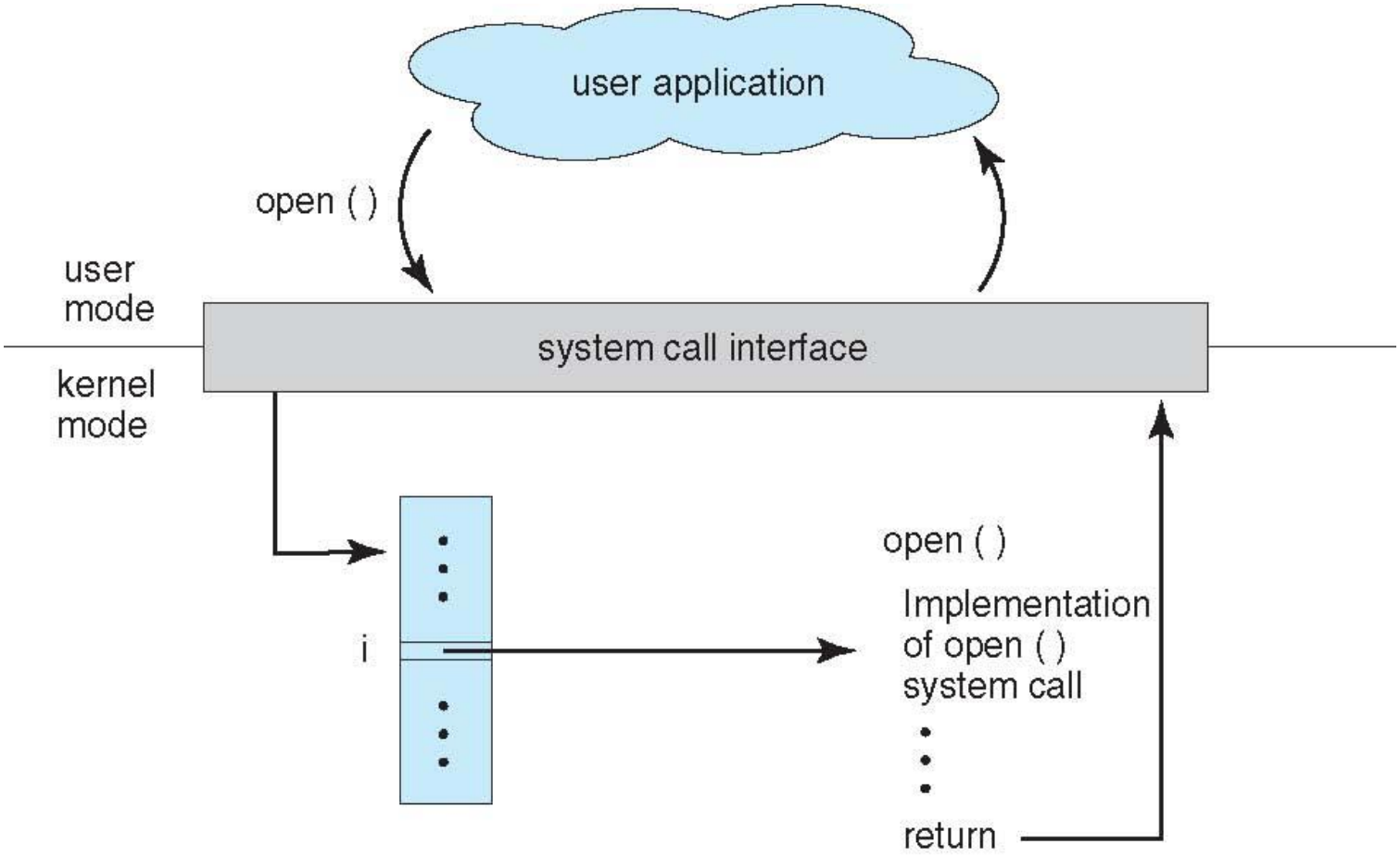
  Ex) int open(const char *path, int oflag);

- **System-call interface**: link between runtime support system of **programming language** and OS system calls
  - Implementation of I/O functions available in programming language (ex: glibc, MS libc, …)

System-Call Interface

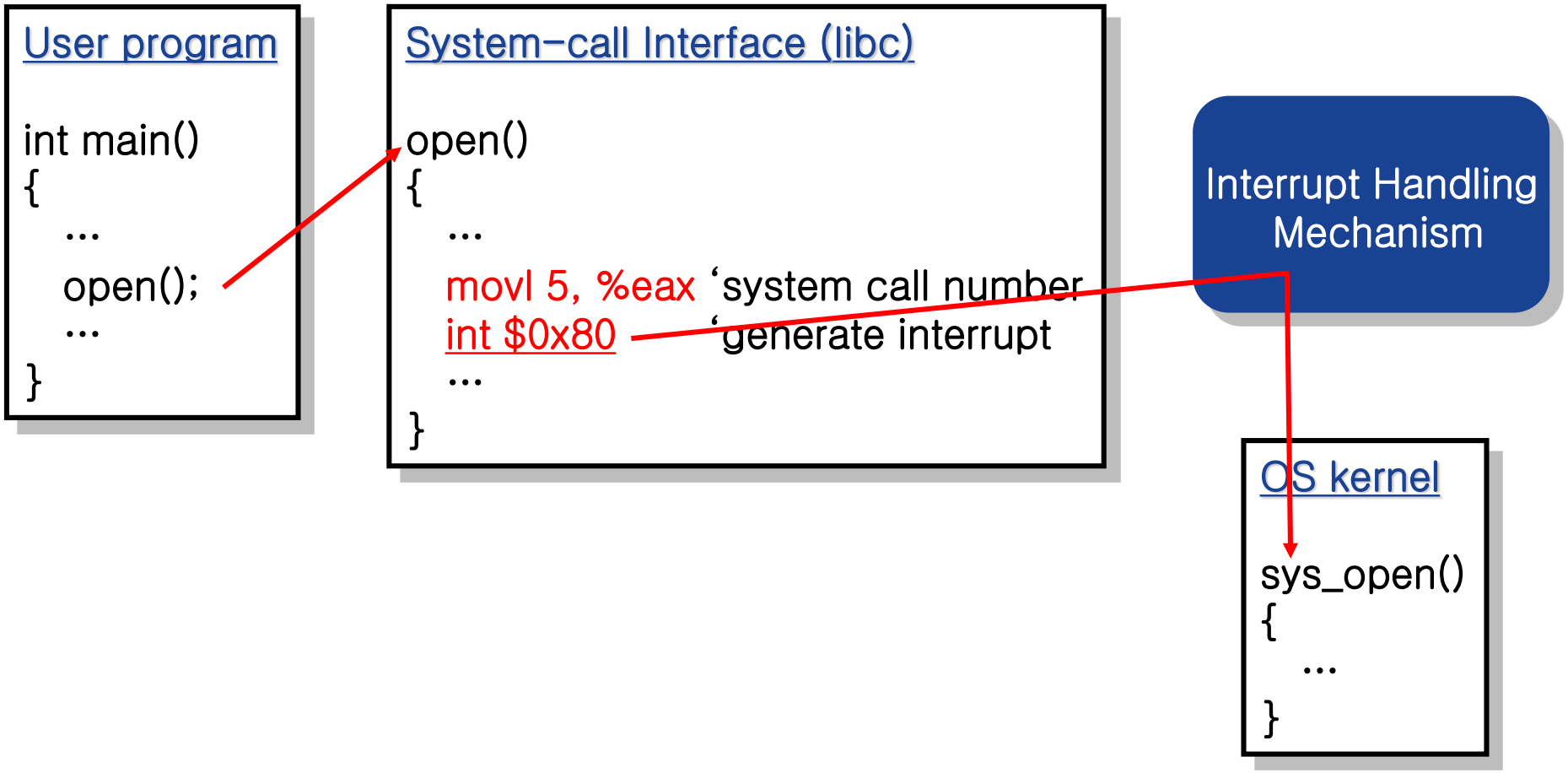
- Typically, a number is associated with each system call.
  - System-call interface maintains a table indexed according to these numbers.
- The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values.
- The caller needs to know nothing 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 from programmer by API
    - Managed by run-time support library (set of functions built into libraries included with compiler)

System-Call Interface



System-Call Interface

- Example of system-call interface in Linux



System-Call Interface

- What does system-call interface do?
  - Passing information to the kernel
  - Switch to kernel mode
  - Any data processing and preparation for execution in kernel mode
  - ETC.

Cf. System call vs. I/O functions in programming language

Ex) read(), vs. fread()

- read(): provided by OS
- fread(): standard function defined in C language
  - fread() is implemented using read()

Application Programming Interface

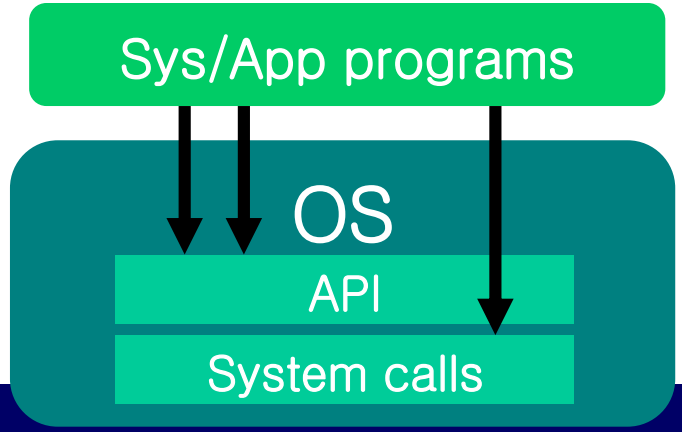
- **API**: interface that a computer system (OS), library or application provides to allow requests for service
  - A set of functions, parameters, return values available to application programmers.

  Ex) Win32 API, POSIX API, etc.
    - MessageBox(..), CreateWindow(…), …
  - Can be strongly correlated to system calls

    Ex) POSIX API ≈ UNIX system calls
  - Can provide high-level features implemented with system calls

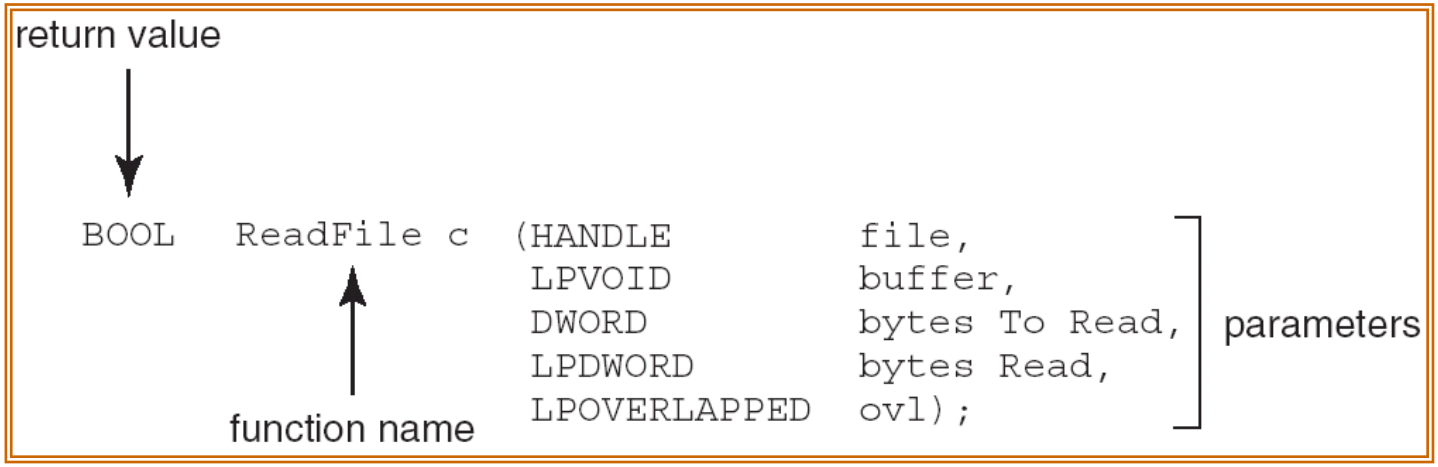
    Ex) Win32 API is based on system calls

    Ex) POSIX thread library API



Example of API

- Win32 API function ReadFile() —a function for reading from a file



- A description of the parameters passed to ReadFile()
  - HANDLE file—the file to be read
  - LPVOID buffer—a buffer where the data will be read into and written from
  - DWORD bytesToRead—the number of bytes to be read into the buffer
  - LPDWORD bytesRead—the number of bytes read during the last read
  - LPOVERLAPPED ovl—indicates if overlapped I/O is being used

Application Programming Interface

- Why API rather than system call?
  - Ease to use
    - API provides higher-level interface than system call
  - System portability

Examples of System Calls

	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()