2. Operating System Structures

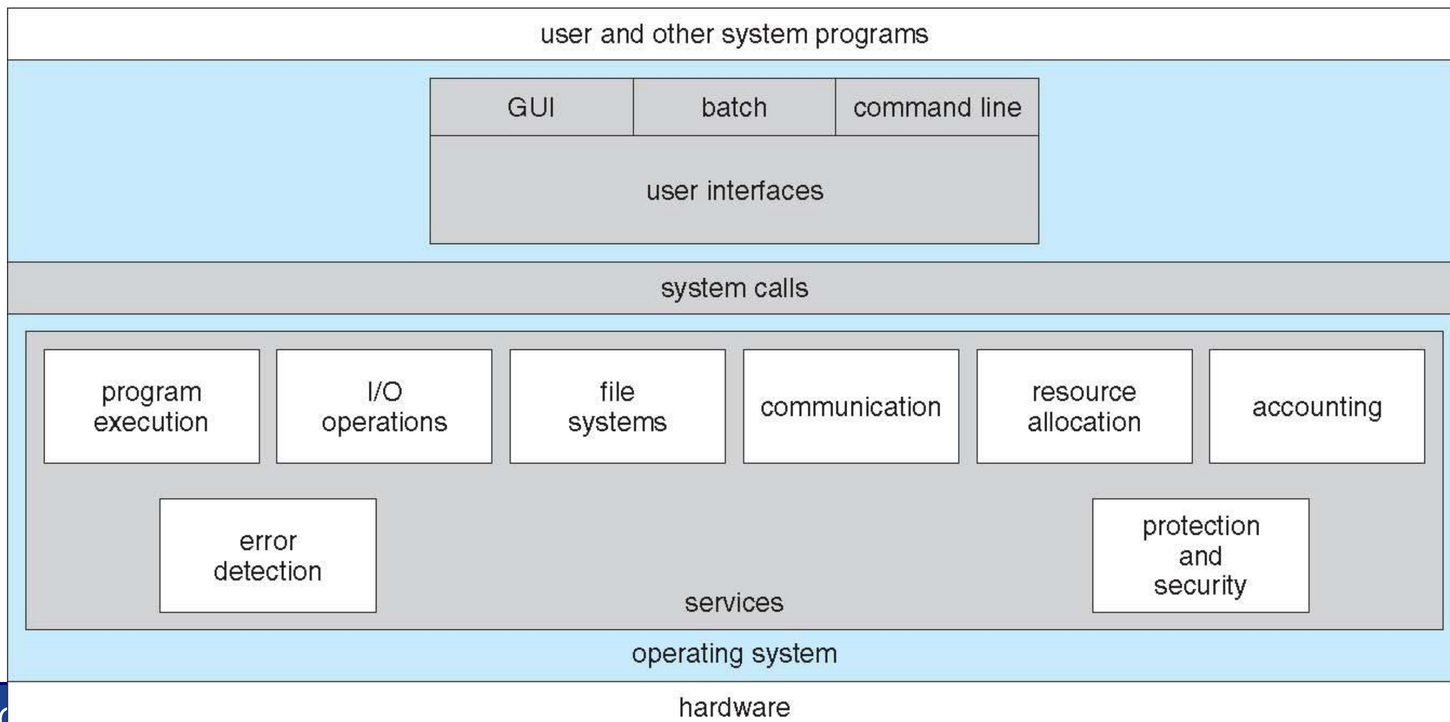
ECE30021/ITP30002 Operating Systems

Agenda

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

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



Operating-System User Interface

- Command interpreter
  - Get and execute user-specified command

    Ex) UNIX shell, MS-DOS Prompt

- GUI (Graphical User Interface)
  - Mouse-based windows-and-menu system
    - Desktop metaphor, icon, folder, …
  - History
    - Xerox Alto computer (1973)
    - Apple Macintosh (1980s)
    - MS-Windows
    - Desktops based on X-window (CDE, KDE, GNOME)
    - 3D desktop (XGL, SphereXP, …)

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

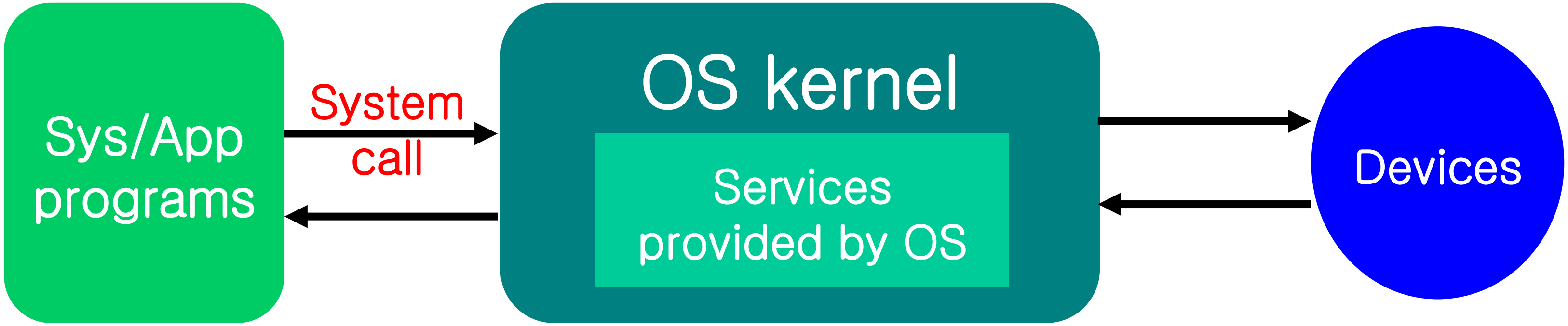
    Ex) MessageBox(…);

  Ex) Win32 API, POSIX API, Java API

System Calls

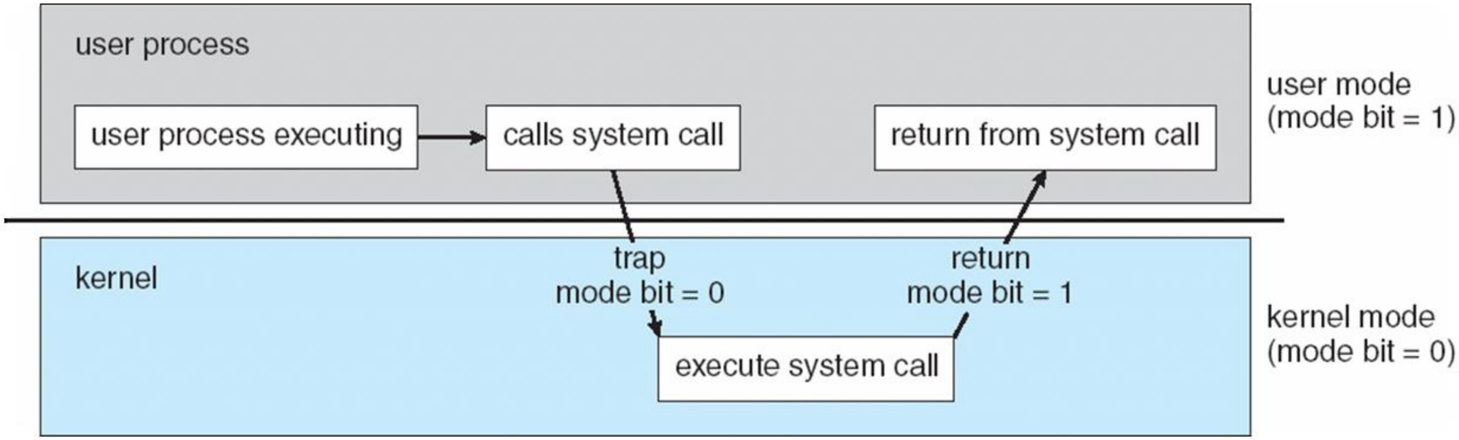
- **System call**: the mechanism used by an application program to request service from OS kernel
  - *"Function calls to OS kernel available through interrupt"*
  - Generally, provided as interrupt handlers written in C/C++ or assembly.
  - A mechanism to transfer control safely from lesser privileged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

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



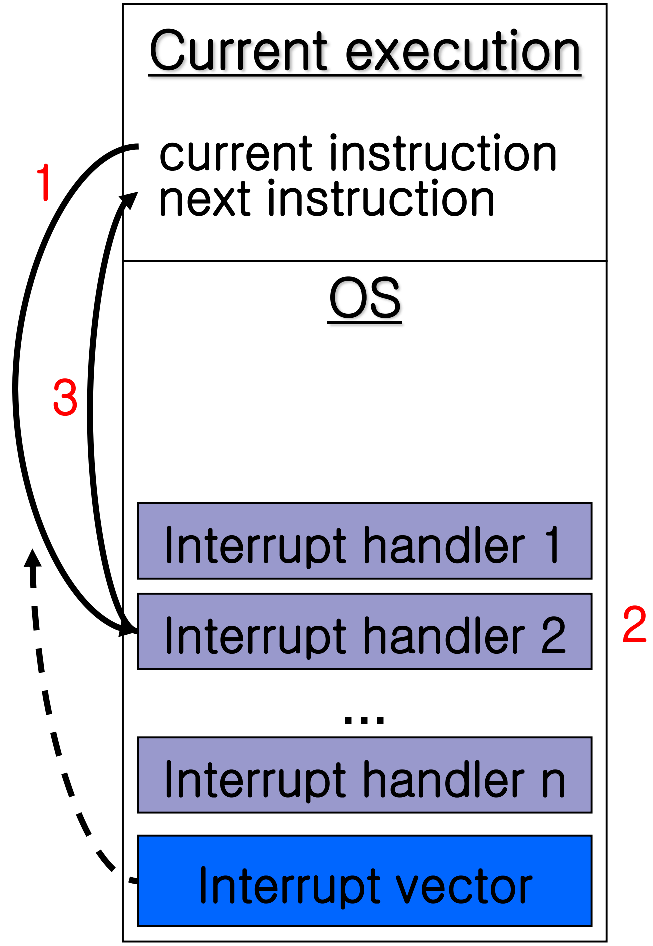
Dual Mode Operation

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



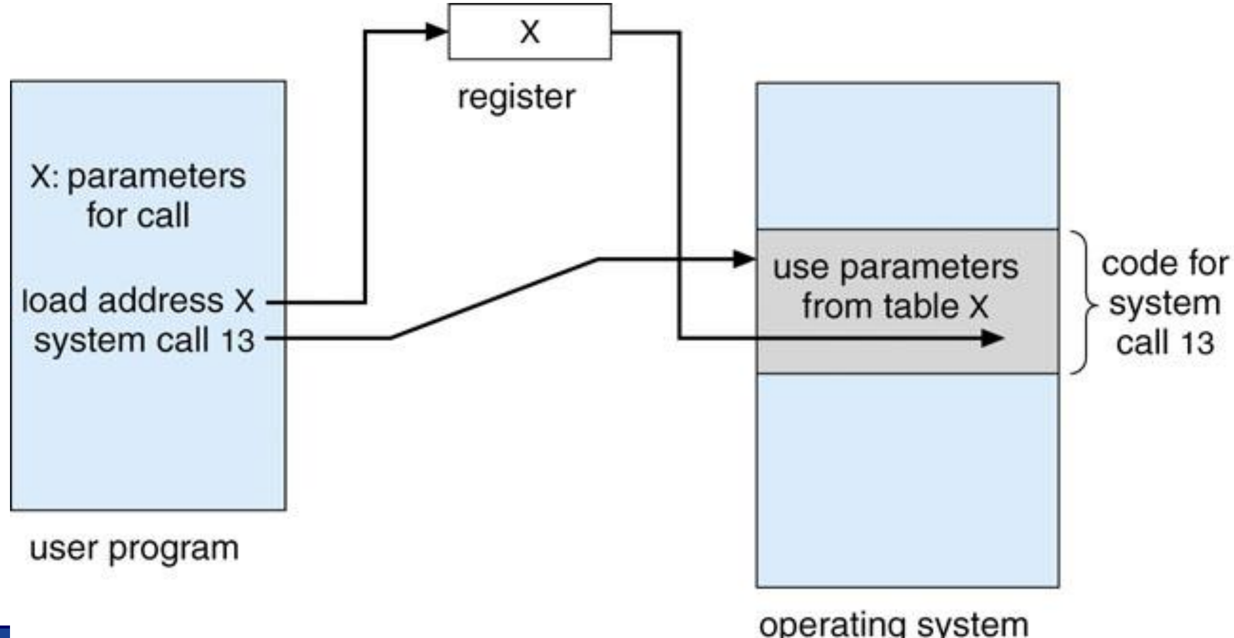
Interrupt Mechanism

- 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

- 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

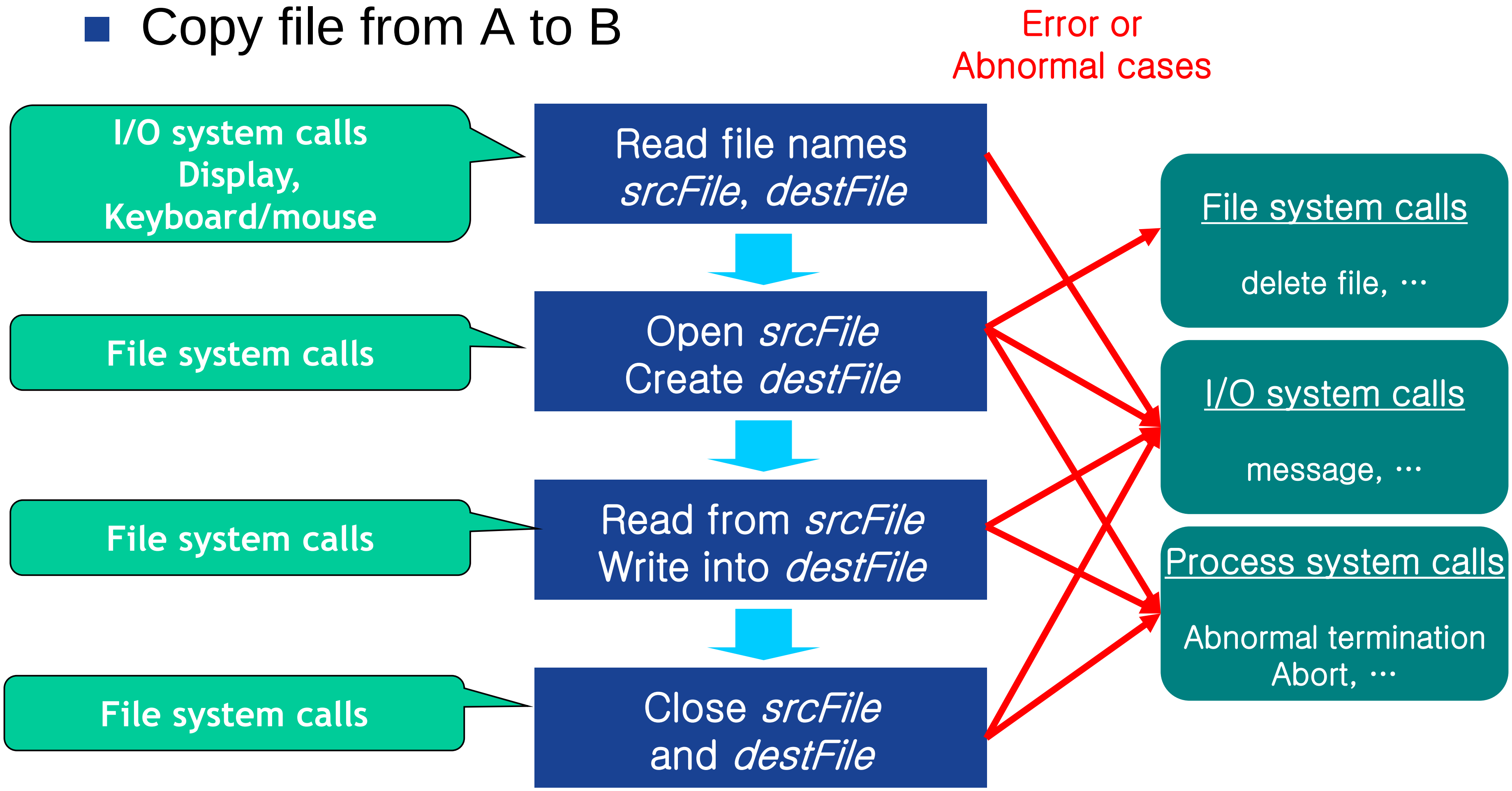


Types of System Calls

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

Example

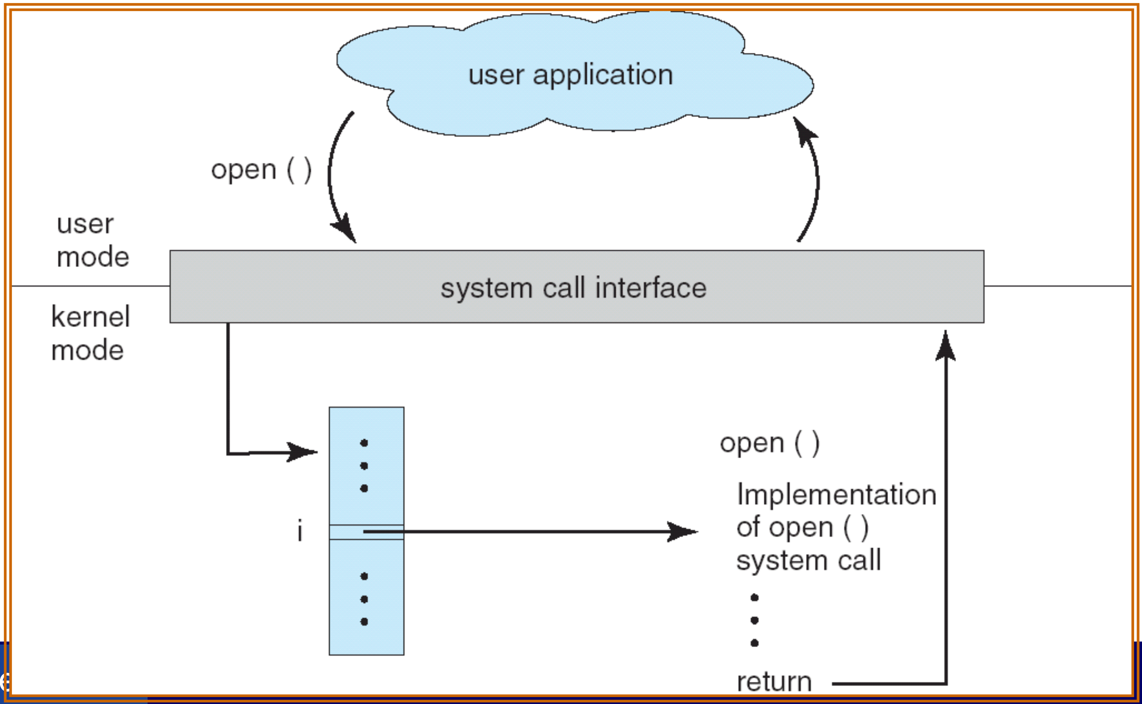
- Copy file from A to B



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, …)

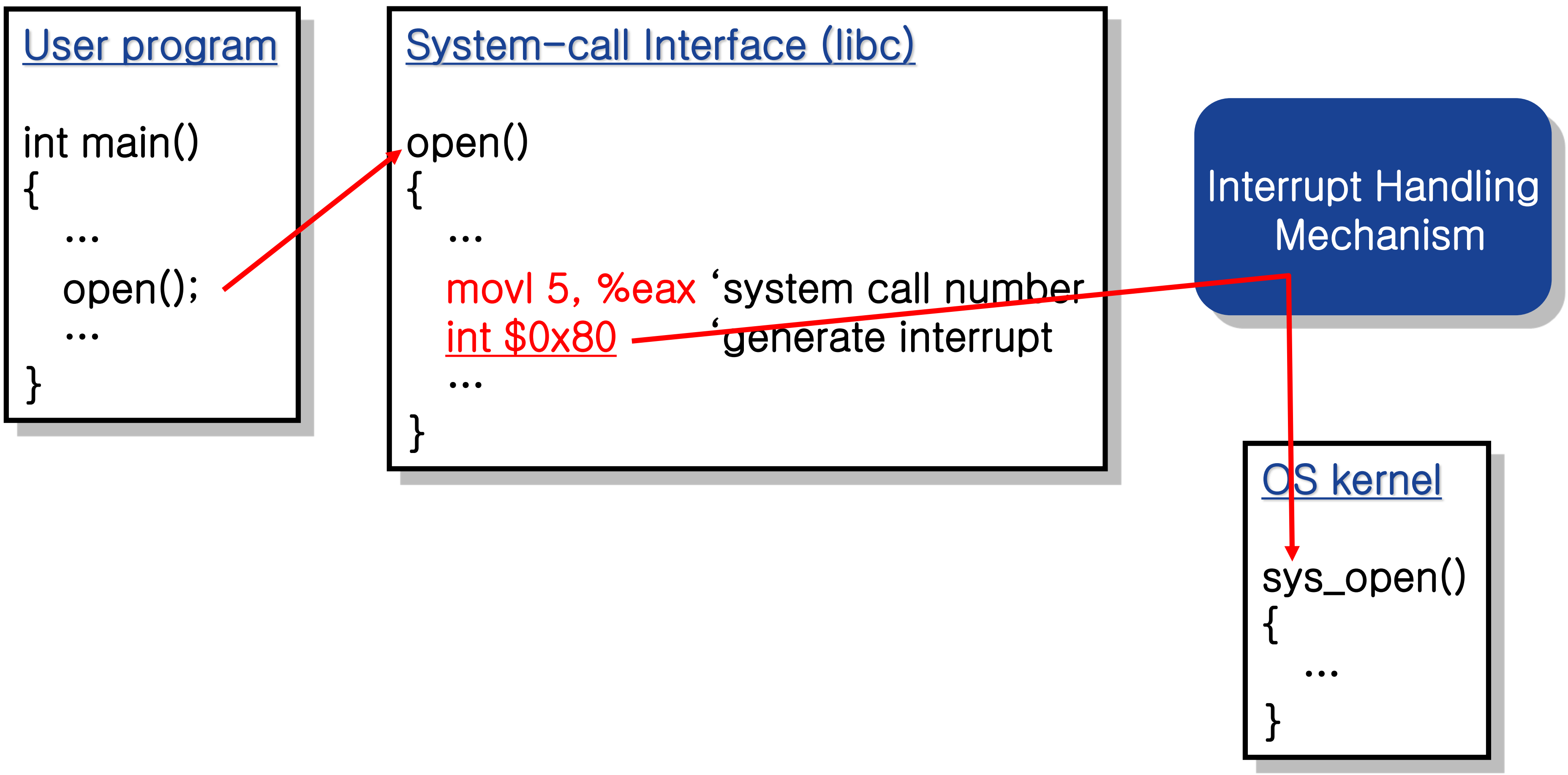


System-Call Interface

- 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

- 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

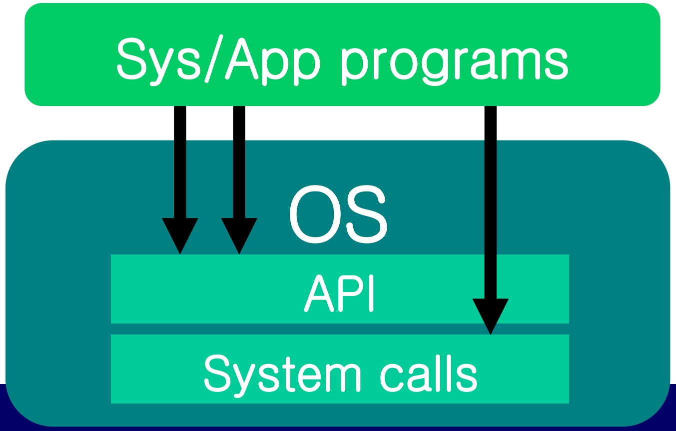
- **API**: interface that a computer system (OS), library or application provides to allow requests for service
  - A set of functions, parameters, return values available to application programmers.

    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



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()

Process Control: Load/Execution

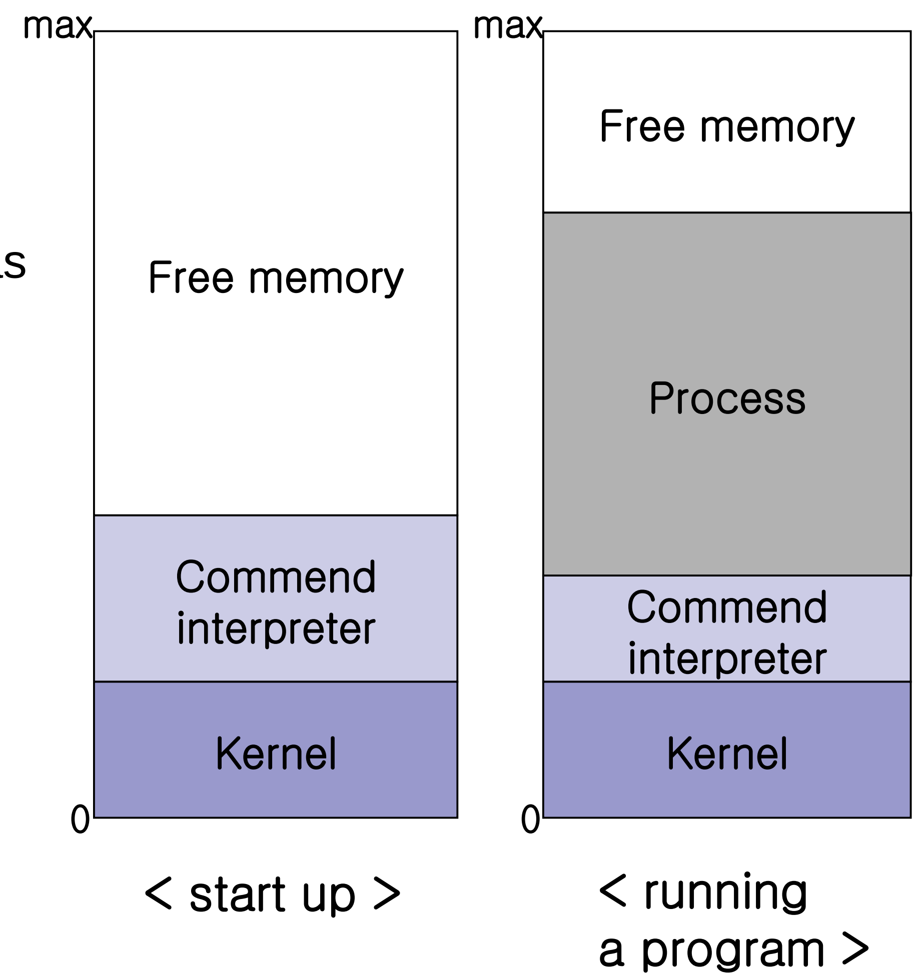
- A program can load/execute another program.

  Ex) Command interpreter

- Then, the parent program can
  - Be lost (replaced by the child program)
  - Be saved (paused)
  - Continue execution: multi-programming
    - Create process/submit job

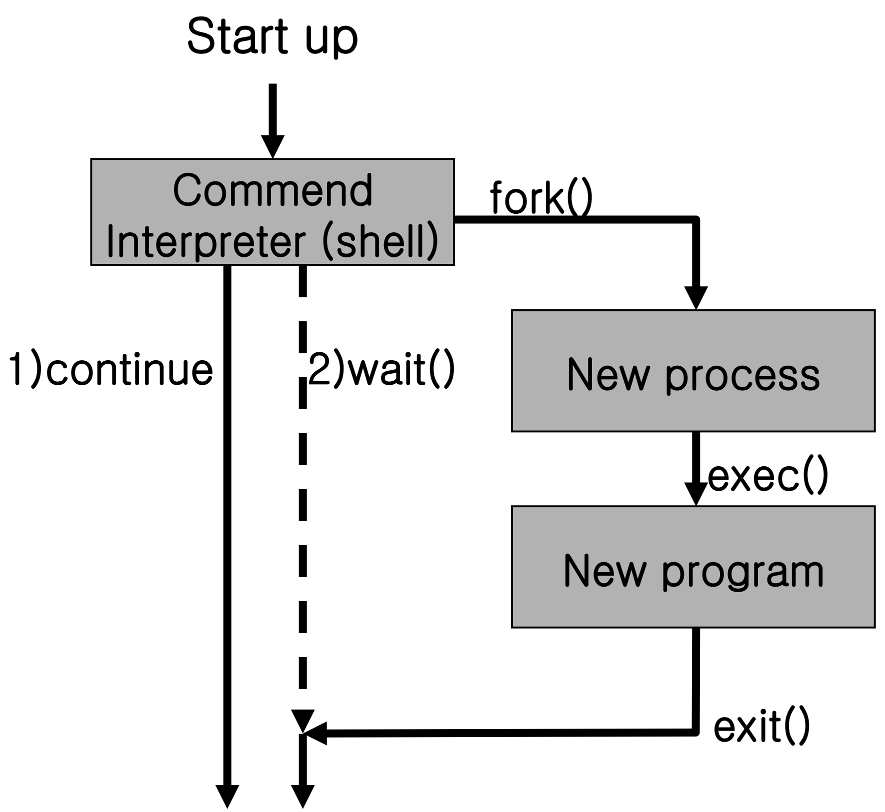
Example: MS-DOS

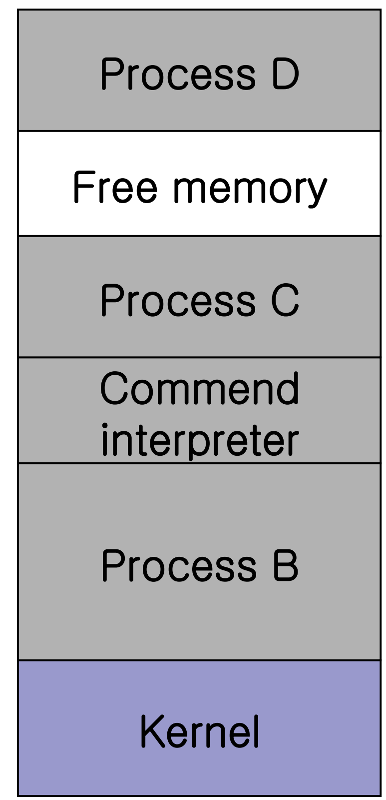
- Single-tasking system
  1. Command interpreter is invoked at system start
  2. Load a program to memory
     - Write over itself to provide as much memory as possible
  3. Run the program
  4. Terminates
     - When error occurs, error code is saved in memory
  5. Overwritten part of command interpreter is reloaded and resume execution
  6. Report error code and continues



Example: FreeBSD UNIX

- Multitasking system





< FreeBSD running multiple program >

Example: FreeBSD UNIX

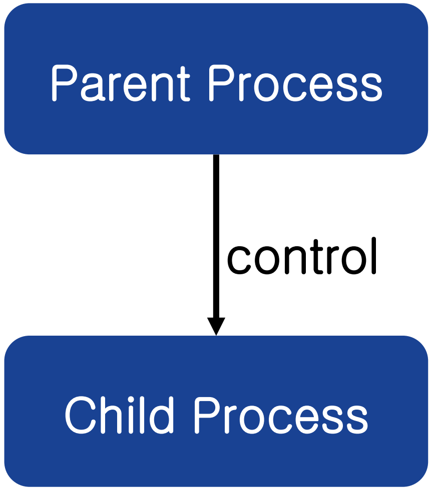
- Command interpreter may continue to execute
- Two cases of execution
  - Case 1, shell continues to execution
    - New program is executed in background
      - Console input is impossible
  - Case 2, shell waits new program
    - New program takes I/O access
    - When the program terminates (exit()), the control is returned to shell with a status code (0 or error code)

Reading Assignment

- Search Internet for documents on the following functions. Read them to understand how to make your program run another program.
  - fork()
  - exec() family functions
    - execlp()
    - execvp()
  - wait()

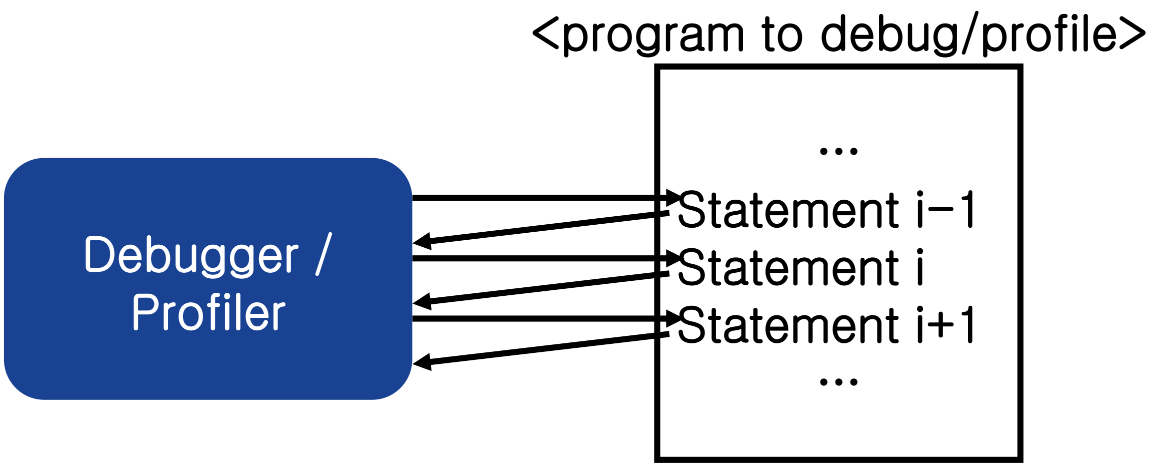
Process Control: Load/Execution

- Controlling new process
  - Get/set process attributes
    - Priority, maximum execution time, …
  - Terminate process
- Waiting for new job/process
  - Wait for a fixed period of time
  - Wait for event / signal event

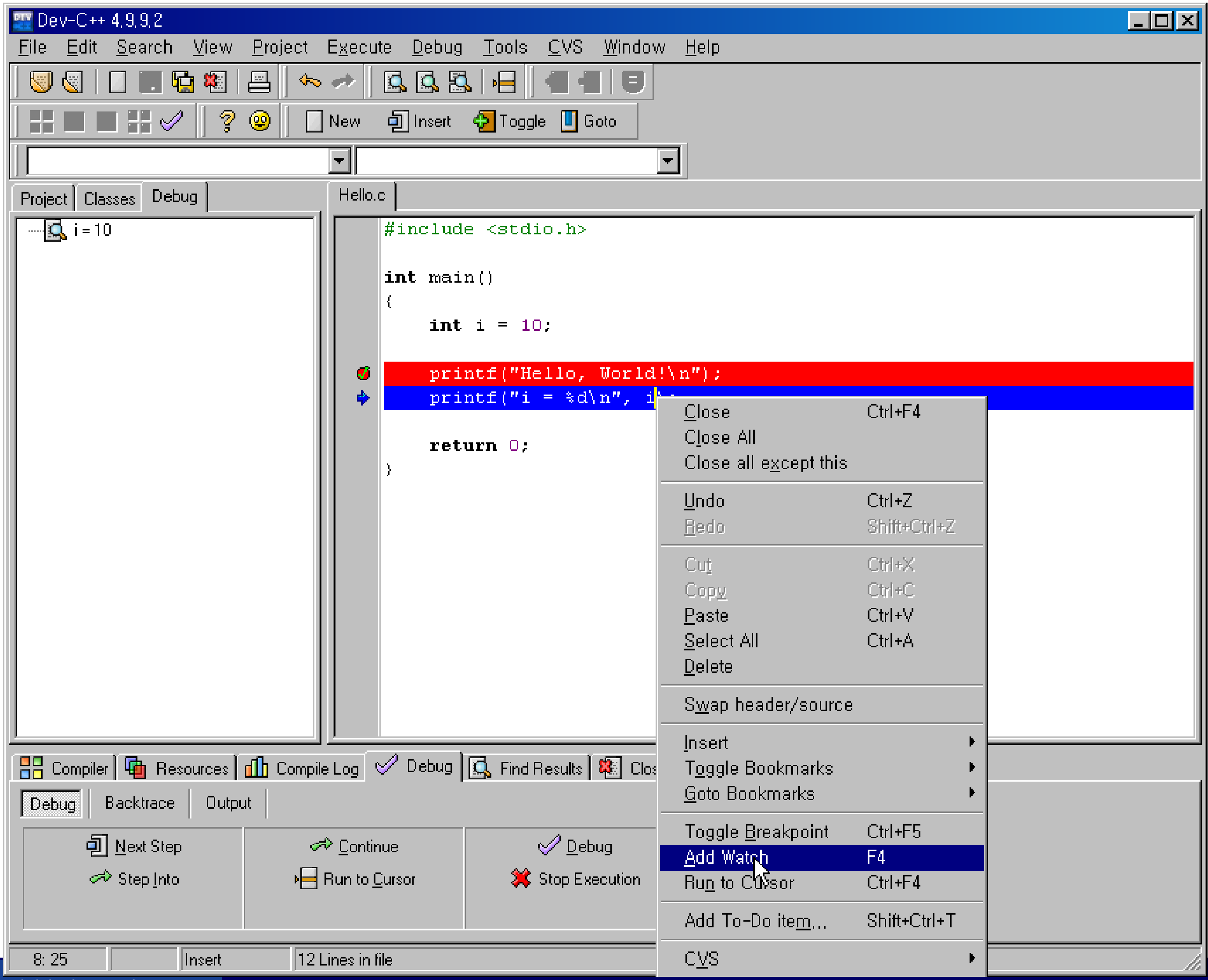


Process Control: Load/Execution

- Debugging
  - Dump
  - Trace: trap after every instruction



Debugger



Dev-C++ 4.9.9.2
File Edit Search View Project Execute Debug Tools CVS Window Help
New Insert Toggle Goto
Project Classes Debug
i = 10
Hello.c
#include <stdio.h>
int main()
{
int i = 10;
printf("Hello, World!\n");
printf("i = %d\n", i);
return 0;
}
Close Ctrl+F4
Close All
Close all except this
Undo Ctrl+Z
Redo Shift+Ctrl+Z
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Select All Ctrl+A
Delete
Swap header/source
Insert
Toggle Bookmarks
Goto Bookmarks
Toggle Breakpoint Ctrl+F5
Add Watch F4
Run to Cursor Ctrl+F4
Add To-Do item... Shift+Ctrl+T
CVS
Compiler Resources Compile Log Debug Find Results
Debug Backtrace Output
Next Step Continue Debug
Step Into Run to Cursor Stop Execution
8: 25 Insert 12 Lines in file

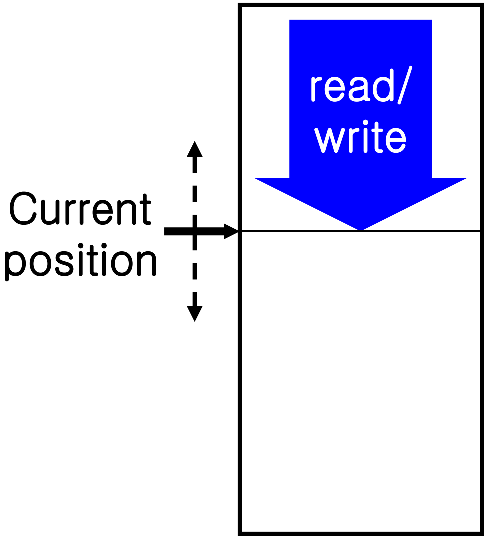
Process Control: Termination

- Normal termination (end)
  - Deallocate resources, information about current process

- Abnormal termination (abort)
  - Dump memory into a file for debugging and analysis
  - Ask user how to handle
    - Interactive system: command interpreter
    - GUI system: pop-up window
    - Batch system: terminates entire job and continue with next job
      - Control card: command to manage execution of process

File Management

- Create/delete files
- Read/write/reposition
- Get/set file attribute
- Directory operation
- More service
  - move, copy, …



➔ Functions can be provided by either system calls, APIs, or system programs

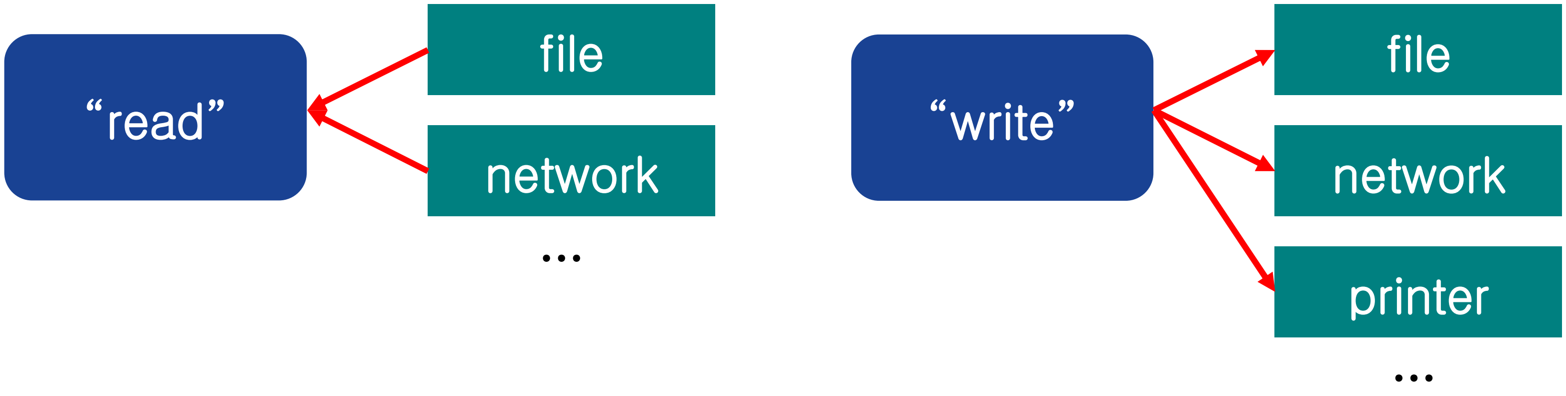
Device Management

- Resources
  - Physical device (disk, tape, …)
  - Abstract/virtual device (file, …)

- Operations
  - Request for exclusive use ≈ open()
  - Read, write, reposition ≈ read(), write(), ⋯
  - Release ≈ close()

Device Management

- Combined file-device structure
  - Mapping I/O into a special file
  - The same set of system calls on both files and devices



Information Maintenance

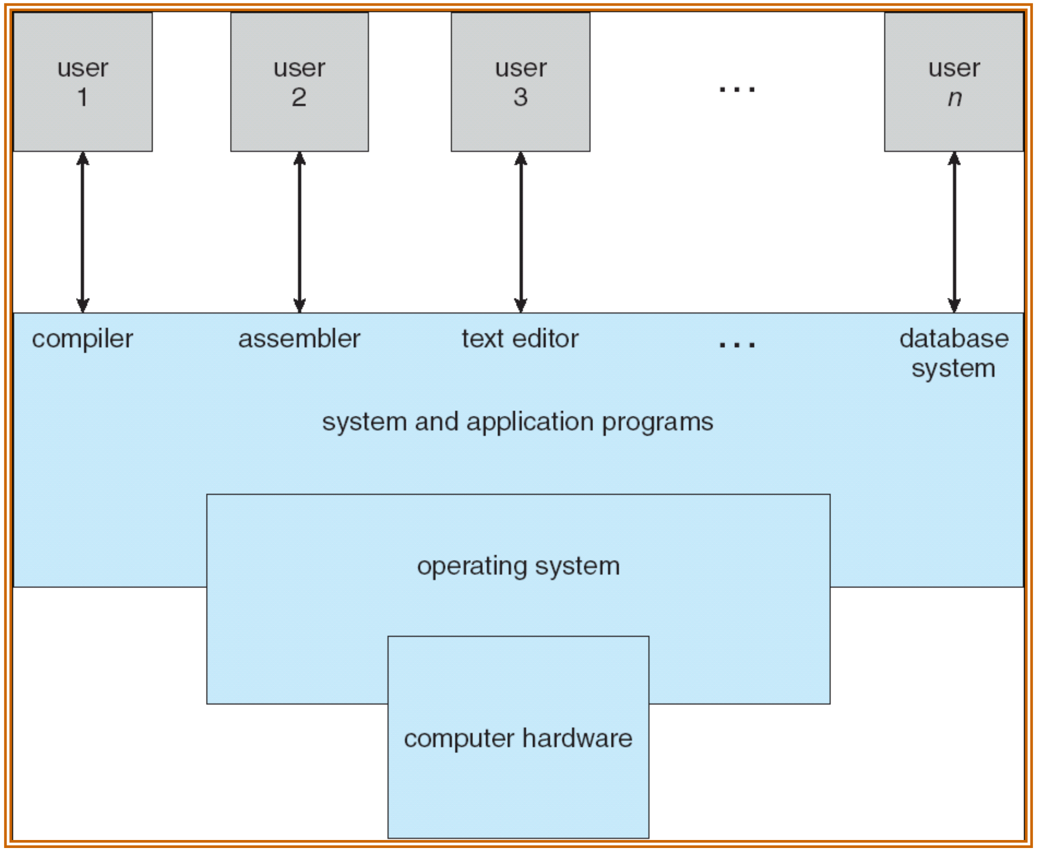
- Transfer information between OS and user program
  - Current time, date
  - Information about system
    - # of current user, OS version, amount of free memory/disk space

- OS keeps information about all it processes

  Ex) /proc of Linux

System Programs

- System program: a program to provide a convenient environment for program development and execution.



System Programs

- System programs can be divided into:
  - File manipulation
  - Status information sometimes stored in a File modification
  - Programming language support
  - Program loading and execution
  - Communications
  - Background services

Agenda

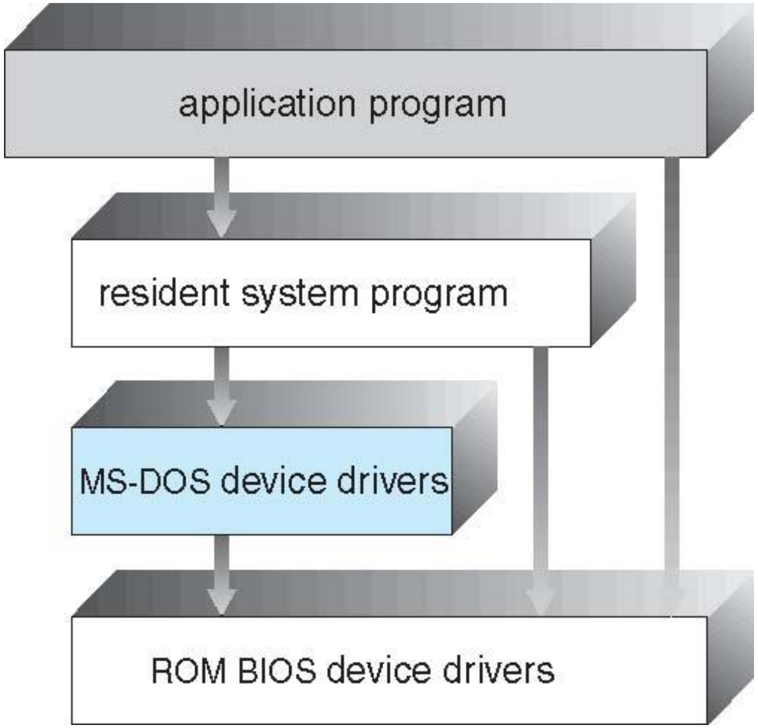
- Operating-system services
- Interfaces for users and programmers
- **Components and their interconnections**
- Virtual Machines
- Design, implementation, generation
- System boot

Operating-System Structure

- General-purpose OS is very large program
- Various ways to structure ones
  - Simple structure – MS-DOS
  - More complex – UNIX
  - Layered – an abstrcation
  - Microkernel – Mach

Simple Structure

- MS-DOS (1981)
  - Started as small, simple limited system
    - Provide most functionality in least space
  - Interface / level of functionality are not well separated
    - No dual mode or H/W protection
    - Application program can access I/O directly
    - Vulnerable to errant program
      - An error in a program can crash all system
    - Limited on specific H/W



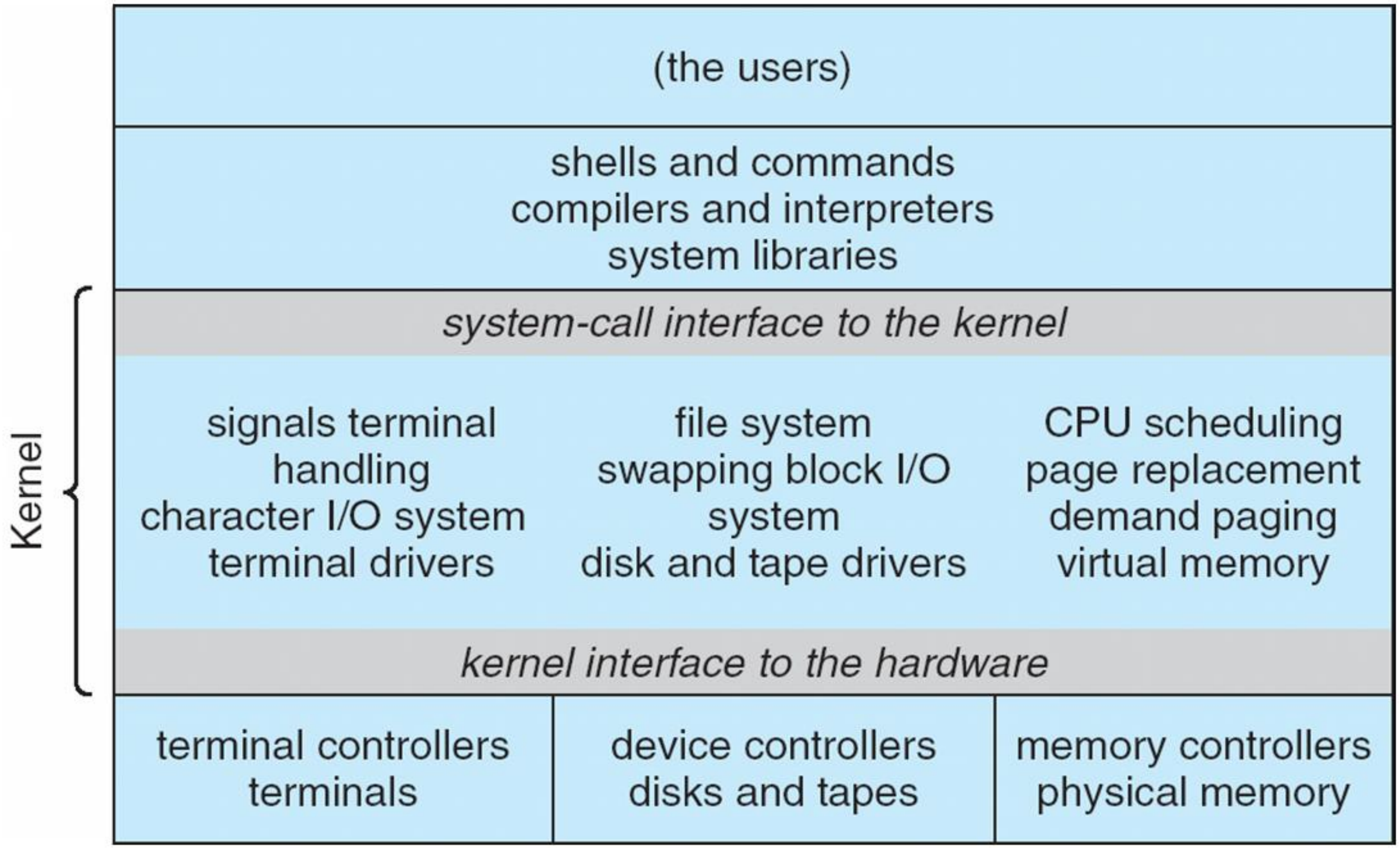
< Structure of MS-DOS >

Non-Simple Structure

- Original UNIX(1973)
  - Also limited by H/W functionality
  - Systems programs
    - Shell, commands compiler, interpreter, system library, …
  - Monolithic kernel
    - Consists of everything below the system-call interface and above the physical hardware
    - File system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level.

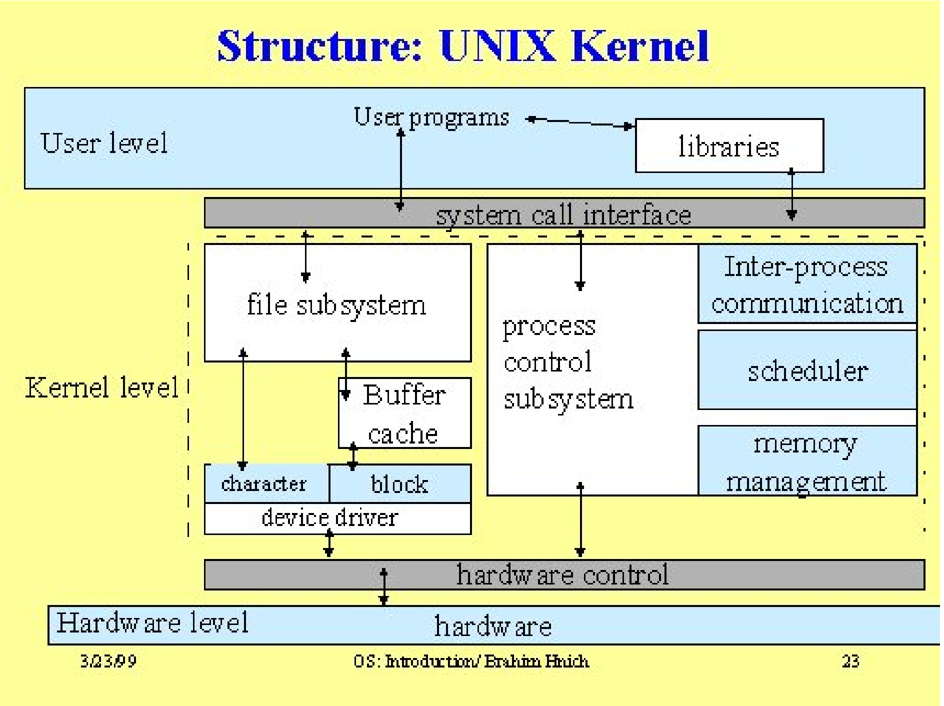
Simple Structure

- Original UNIX



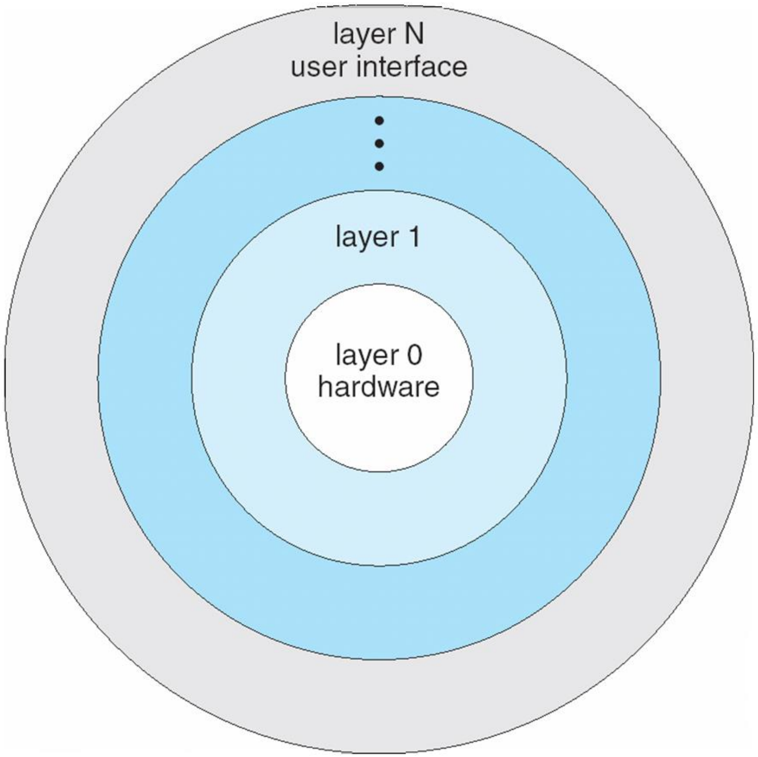
Modern Operating Systems

- Modern OS's can be broken into pieces appropriately
  - Easy to implement
  - Flexible
  - Information hiding



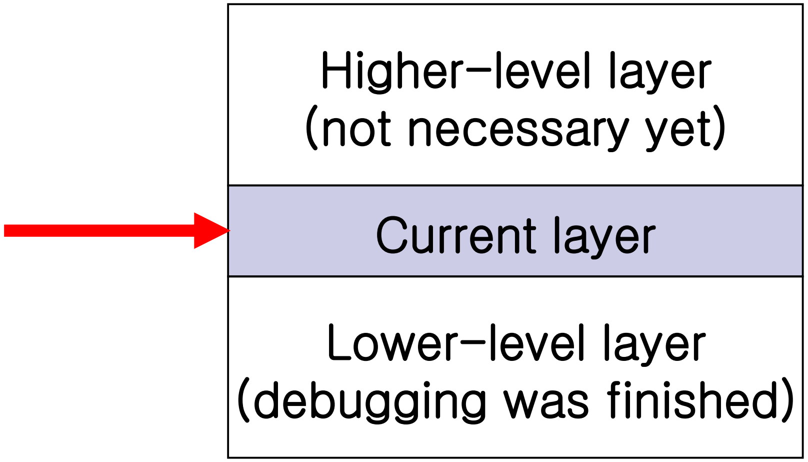
Layered Approach

- OS is composed of layers
- Layer
  - Implementation of abstract objects and operation
  - Each layer M can invoke lower-level layers
  - Each layer M can be invoked by higher-level layers
- Each layer uses functions/services of only lower-level layers



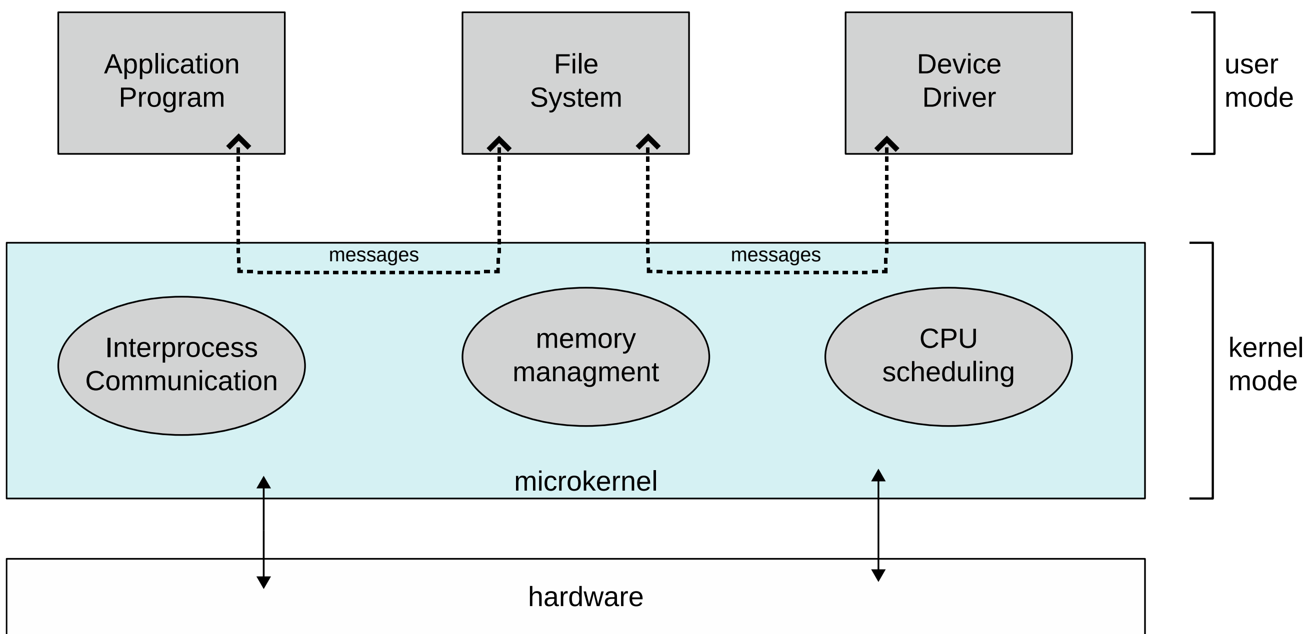
Layered Approach

- Advantages of layered approach: simple to construct and debug
  - If we develop from lower-level layer to higher-level layer, we can concentrate on current layer at each stage
  - A layer doesn't need to know detail of lower-level layer



Microkernels

- Smaller kernel
  - All unessential components are not implemented in kernel but as system/user-level programs.
    - Only essential components are included in kernel
    - Other components are provided by system/user programs



Microkernels

- Generally, process/memory management, communication facility are in the kernel.
- System calls are provided through message passing.
  - Clients and services are running in user space
  - Kernel provides only a message passing facility between client and server

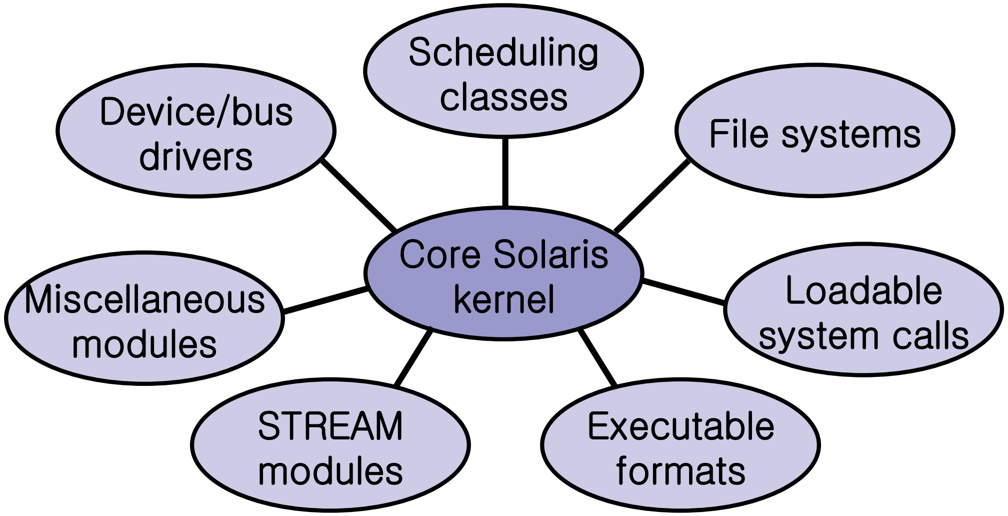
Microkernels

- Advantages of microkernel
  - Ease of extending
  - Ease to port
  - Security and reliability
    - Most services are on user space

- Disadvantages
  - Performance decrease due to increased system function overhead.

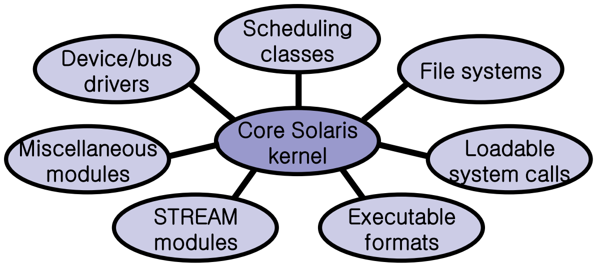
Modules

- Kernels with loadable modules (Linux, Solaris, etc)
  - Uses object-oriented approach
  - Each core component is separate
  - Each talks to the others over known interfaces
  - Each is loadable as needed within the kernel

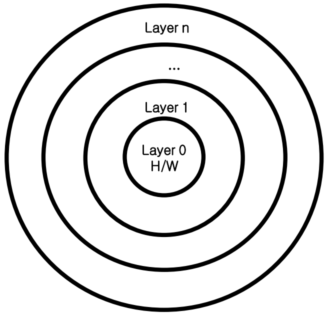


Modules

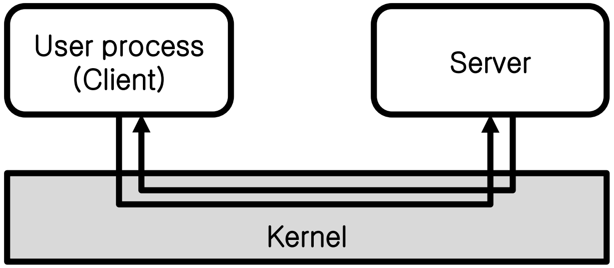
- Advantage
  - Provides core services
  - Allows certain features to be implemented dynamically



- Comparison with layered structure
  - More flexible (any module can any other modules)



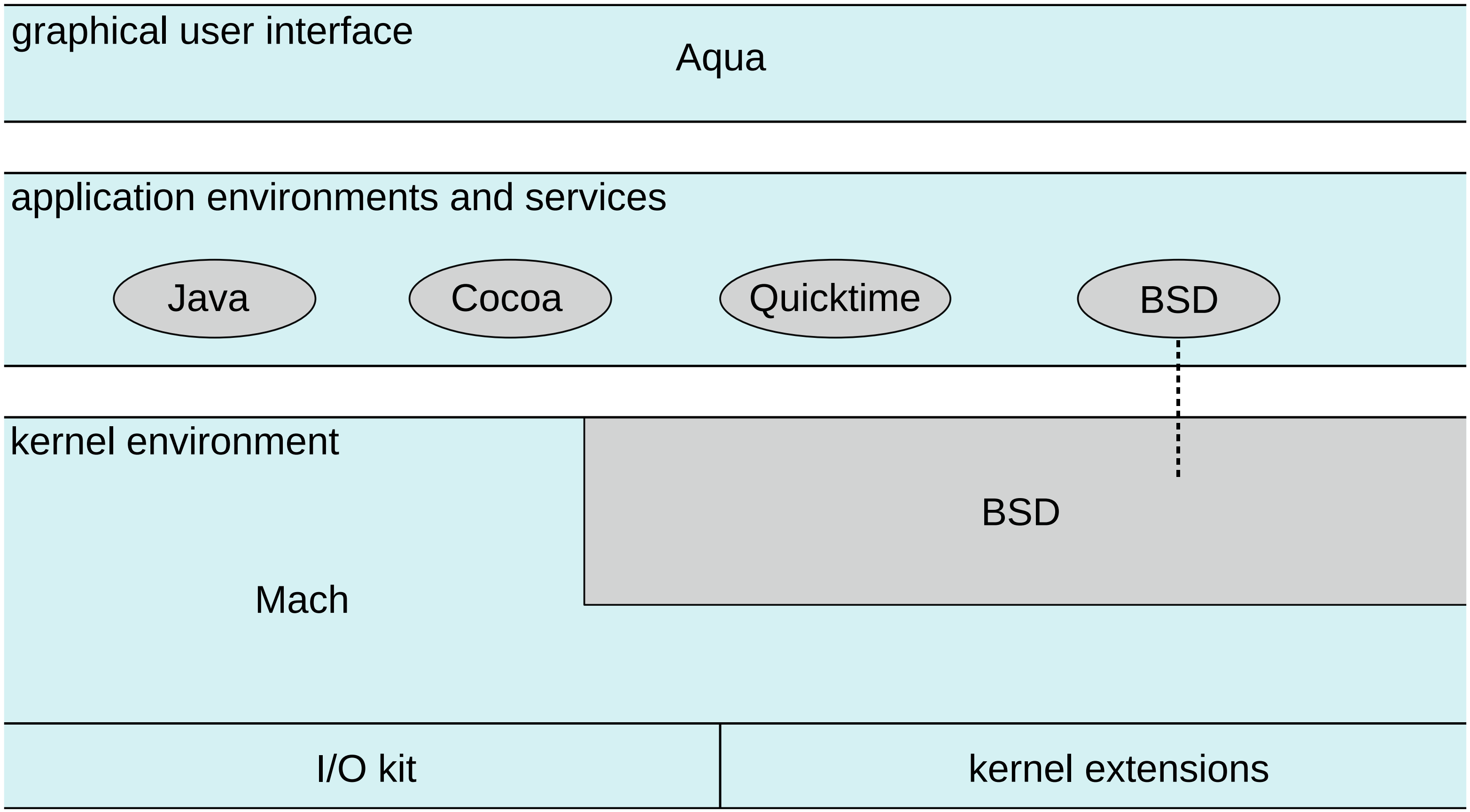
- Comparison with microkernel
  - Each module can run in kernel mode
  - Modules don't need to invoke message passing



Hybrid Systems

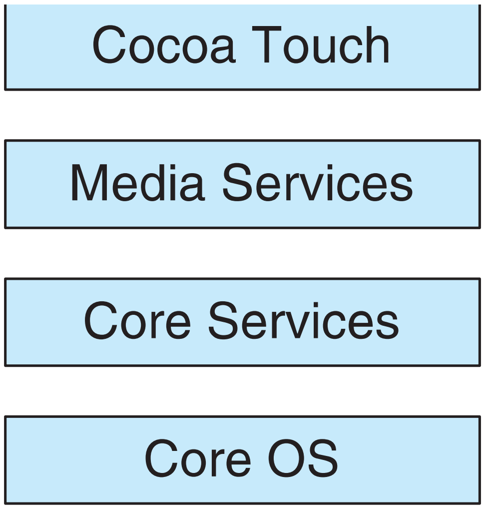
- Most modern operating systems are actually not one pure model
  - Hybrid combines multiple approaches to address performance, security, usability needs
  - Linux and Solaris kernels in kernel address space, so monolithic, plus modular for dynamic loading of functionality
  - Windows mostly monolithic, plus microkernel for different subsystem personalities
- Apple Mac OS X hybrid, layered, Aqua UI plus Cocoa programming environment
  - Below is kernel consisting of Mach microkernel and BSD Unix parts, plus I/O kit and dynamically loadable modules (called kernel extensions)

Mac OS X Structure



iOS

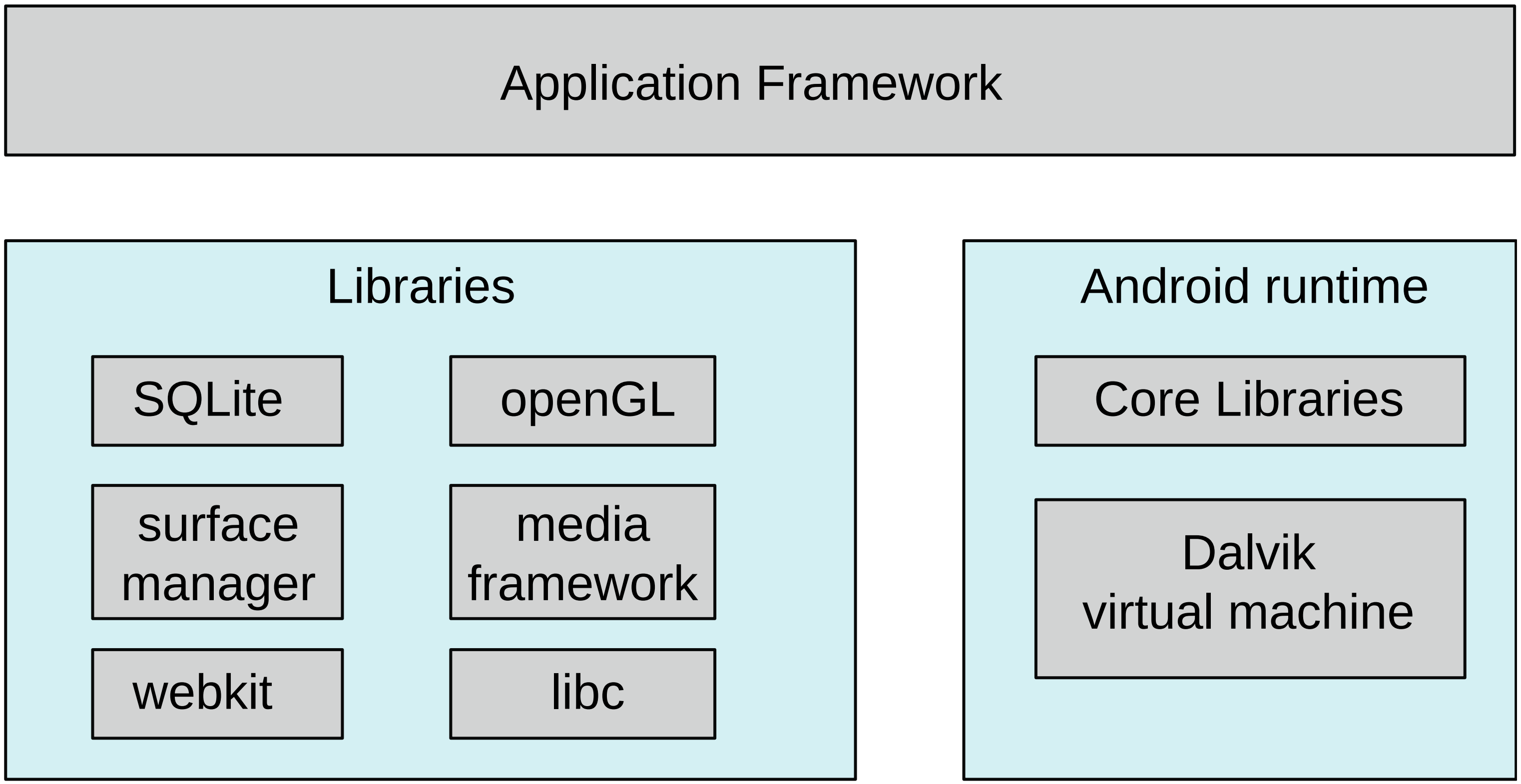
- Apple mobile OS for ***iPhone***, ***iPad***
  - Structured on Mac OS X, added functionality
  - Does not run OS X applications natively
    - Also runs on different CPU architecture (ARM vs. Intel)
  - **Cocoa Touch** Objective-C API for developing apps
  - **Media services** layer for graphics, audio, video
  - **Core services** provides cloud computing, databases
  - Core operating system, based on Mac OS X kernel



Android

- Developed by Open Handset Alliance (mostly Google)
  - Open Source
- Similar stack to IOS
- Based on Linux kernel but modified
  - Provides process, memory, device-driver management
  - Adds power management
- Runtime environment includes core set of libraries and Dalvik virtual machine (now, replaced by ART)
  - Apps developed in Java plus Android API
    - Java class files compiled to Java bytecode then translated to executable than runs in Dalvik VM
- Libraries include frameworks for web browser (webkit), database (SQLite), multimedia, smaller libc

Android Architecture



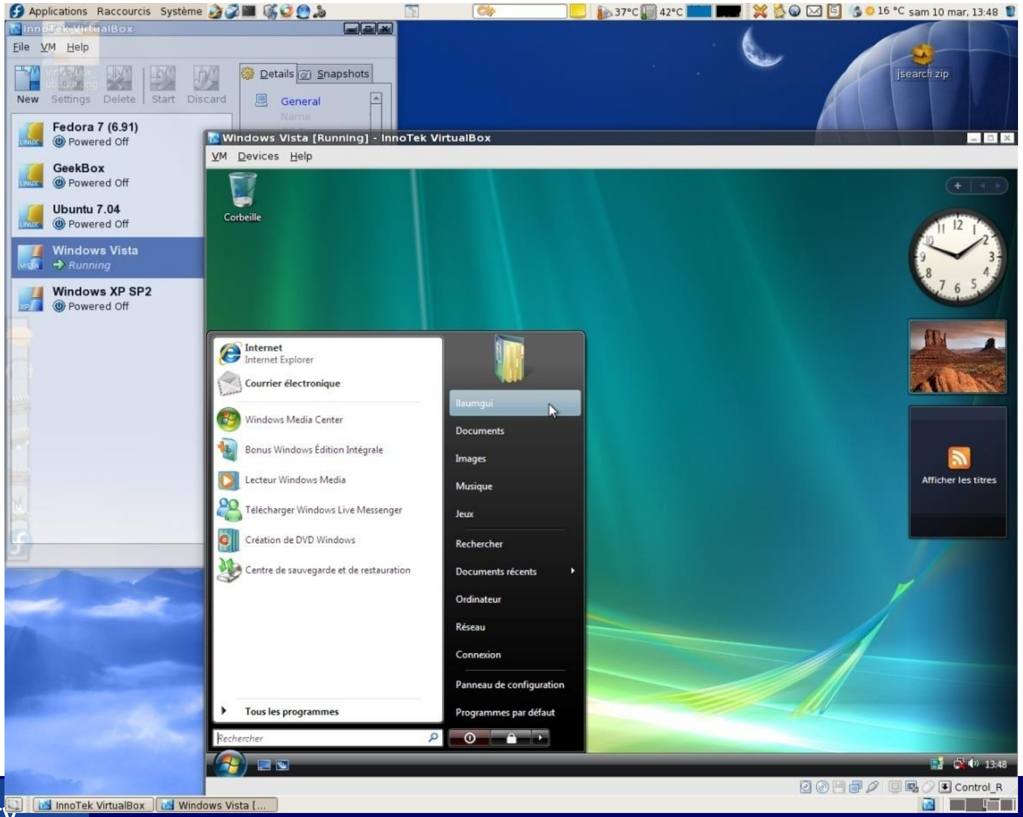
Agenda

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

Virtual Machines

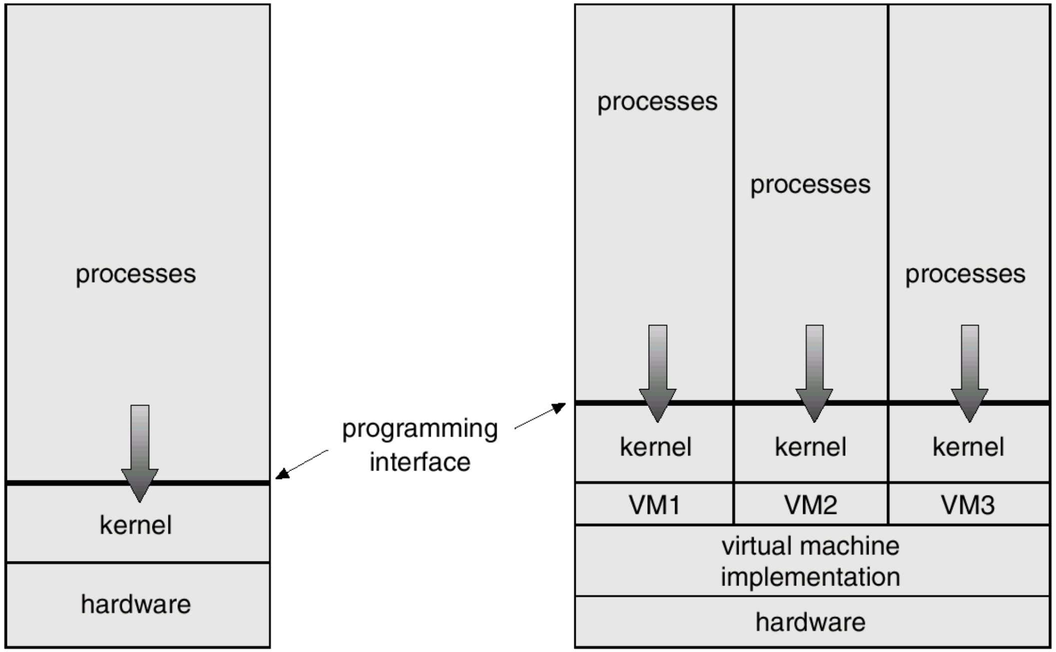
- **Virtual machine**: software that creates a virtualized environment (machine) between the computer platform and its operating system, so that the end user can operate software on an abstract machine.

  Ex) VMWare, VirtualPC, VirtualBox(www.virtualbox.org)



Virtual Machines

- Abstract H/W of single computer into several different execution environment
  - A number of different identical execution environments on a single computer, each of which exactly emulates the host computer.



Virtual Machines

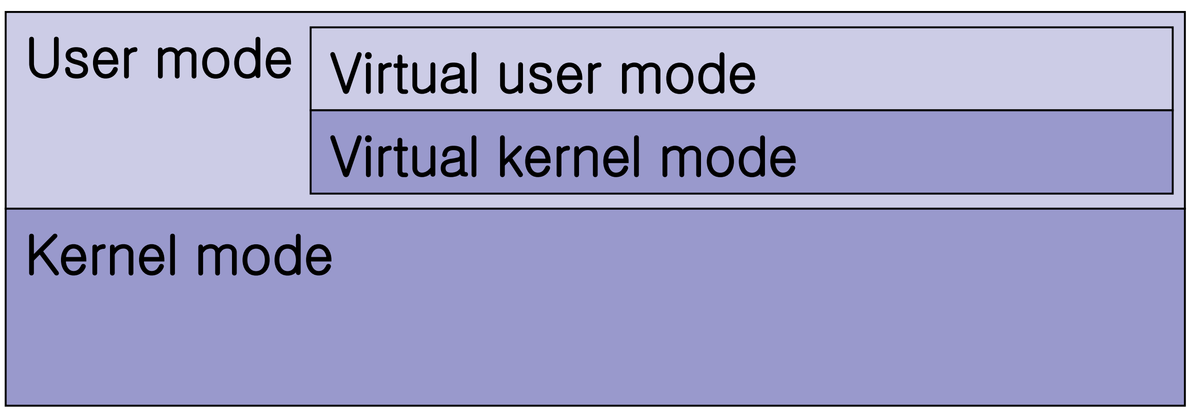
- Each process seems to have its own CPU and memory
  - CPU scheduling + virtual memory technologies
    - Virtual memory allows software to run in a memory address space whose size and addressing are not necessarily tied to the computer's physical memory.
- Major difficulty: disk space
  - It is impossible to allocate same disk drive to each virtual machine
  - Solution: virtual disks (minidisks)
    - Identical in all respects except size

Virtual Machines

- Implementation problems
  - Exact duplication of underlying machine requires much work
  - Support for dual mode operation: virtual dual mode

    Cf. VM S/W can run in kernel mode, but VM itself is executed in user mode

    - Virtual user mode / virtual kernel mode
      - System call from virtual user mode is simulated by VM monitor
    - Many CPUs support more than two privilege levels.



Virtual Machines

- Benefits of VM
  - Complete protection of various system resources

  cf. Sharing between VM's
    - Shared minidisk
    - Virtual network connection

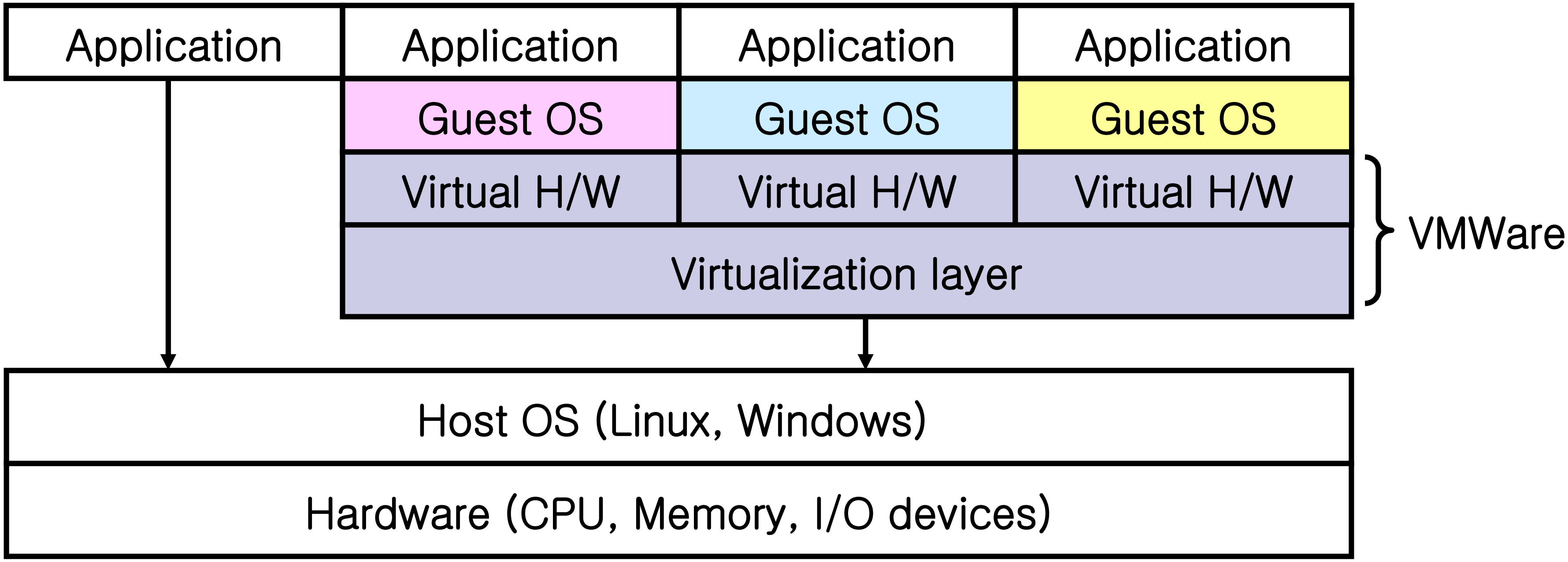
- Perfect vehicle for operating-systems research, development, and education
  - Changing OS is dangerous -> test is very important
  - Working on VM, system programmer don't have to stop physical machine

Virtual Machines

- Inevitable differences from host system
  - Disk size
  - Execution time
    - Multiprogramming among many VM's can slow down VM's in unpredictable ways
    - Privileged instructions on VM are slow because they are simulated
    - Virtual I/O can be faster (spooling) or slower (interpreted)

Examples of VM: VMware

- A commercial VM of Intel 80x86 H/W
  - Runs on Windows or Linux
  - Allows the host to run guest operating systems as VM's
  - Major use
    - Testing an application on several different OS's



Examples of VM: JVM

- Java
  - OOP language developed by SUN, 1995
  - Components
    - Language specification + Large API library
    - Specification for JVM (Java Virtual Machine)
  - Java objects are specified with class structure in bytecode
    - Bytecode: architecture-neutral code executed on JVM
    - *"Compile Once! Run Everywhere!"*

Examples of VM: JVM