2. Operating System Structures

ECE30021/ITP30002 Operating Systems

2. Operating System Structures

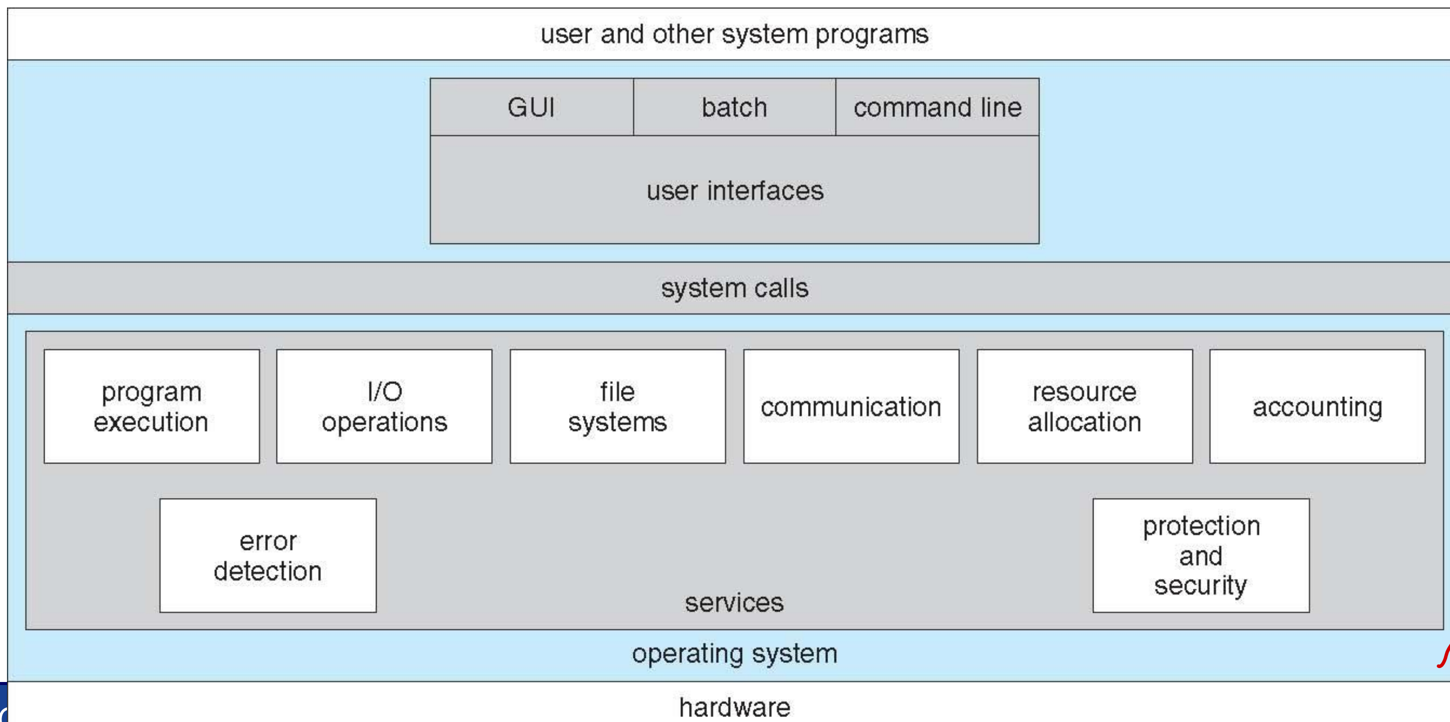
ECE30021/ITP30002 Operating Systems

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
  - Logging
  - Protection and security



OS, kernel

Operating-System User Interface

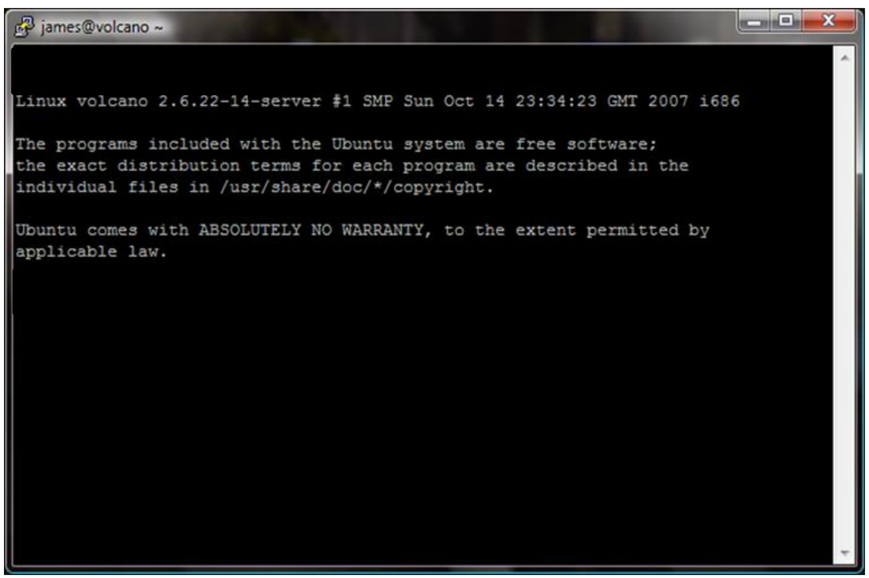
- Command-line interpreter (CLI)
  - Get and execute user-specified command

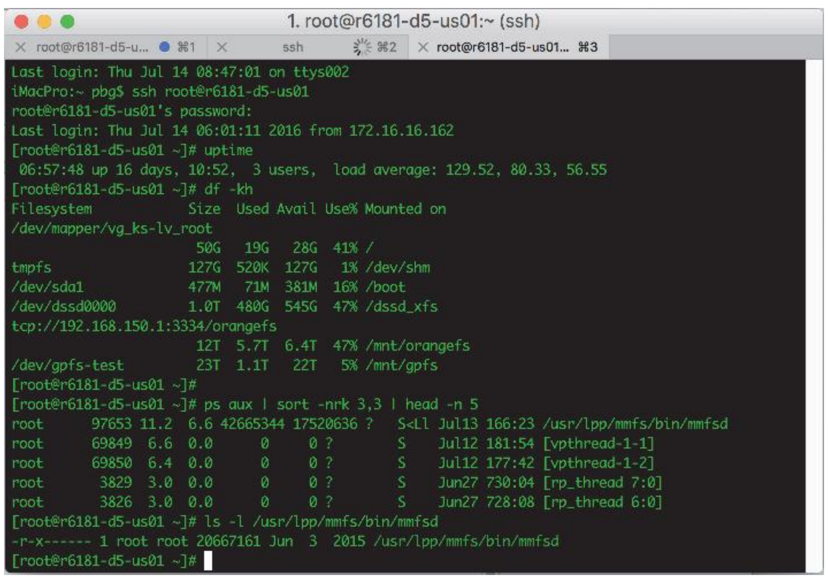
    Ex) UNIX shell, MS-DOS Prompt

- Graphical user interface (GUI)
  - 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)

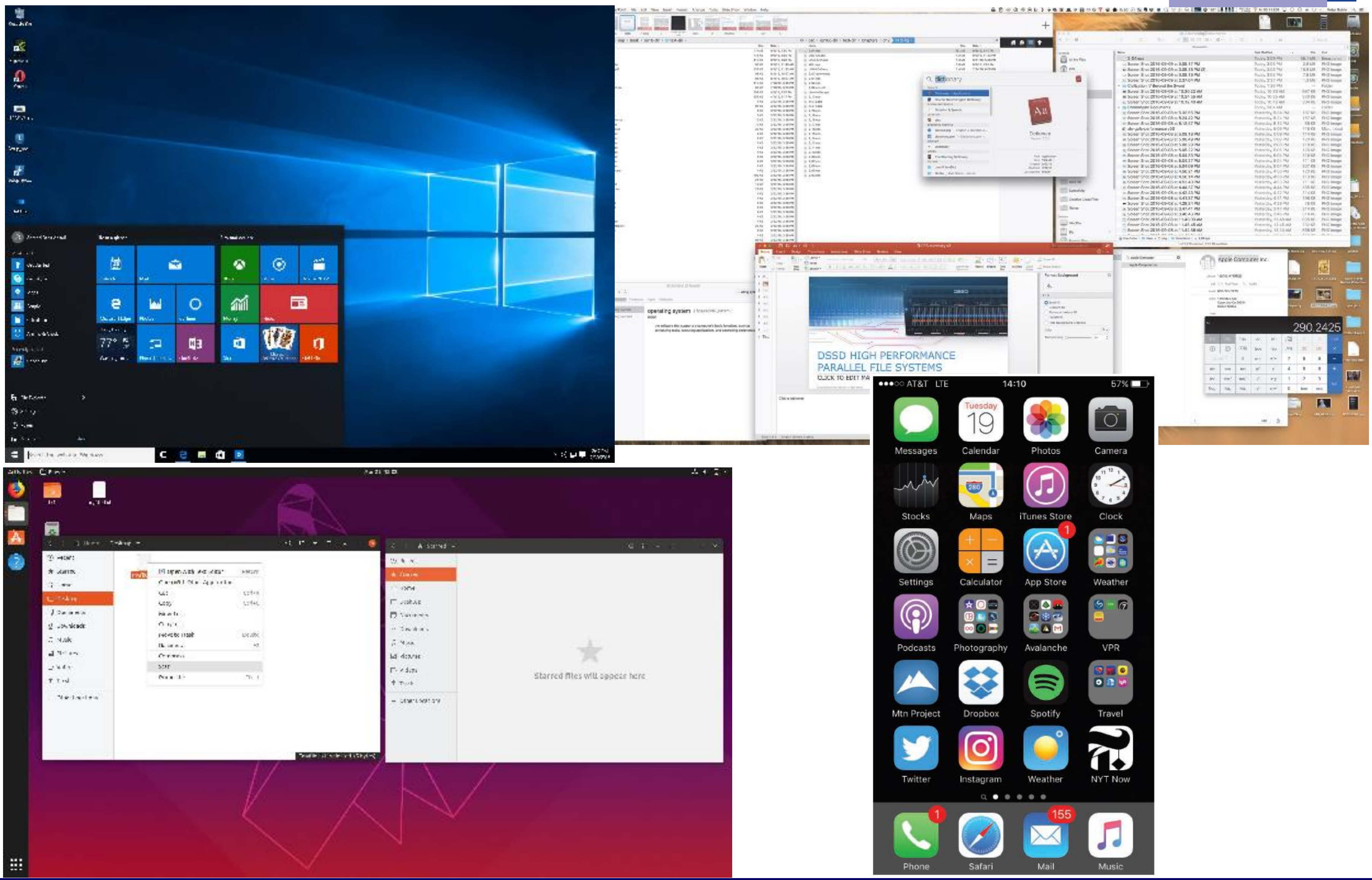
Command Line Interpreter

- Popular CLI terminals
  - putty (http://www.putty.org), xterm, MacOS terminal, …





Graphical User Interface (GUI)



Programming Interfaces

S.W interrupt
1. Exception
2. systemcall

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

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

Example) POSIX I/O system calls (declared in unistd.h)

open을 이용해서 fopen 구현

호환성을 생각하면 (C의 표준 이용

고급기능을 사용하려면 Unix 명령사용

```
int open(const char *pathname, int flags, mode_t mode);
int close(int fd);
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);

// size_t: unsigned int, ssize_t: signed int
```

System Calls

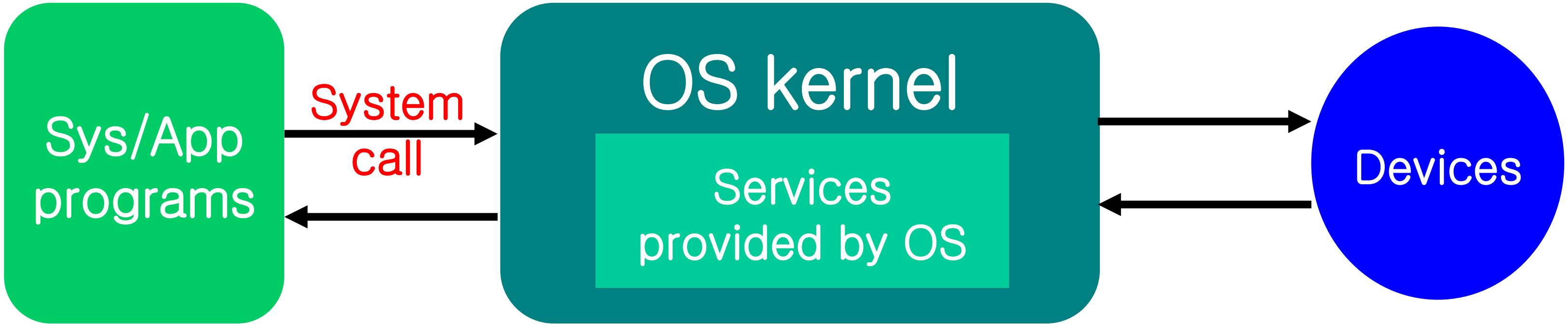
- **System calls**: the mechanism used by an application program to request service from OS kernel
  - *"Function calls to OS kernel available through interrupt"*
  - Generally, provided as interrupt handlers written in C/C++ or assembly.
  - A mechanism to transfer control safely from lesser privileged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

process control 명령어

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

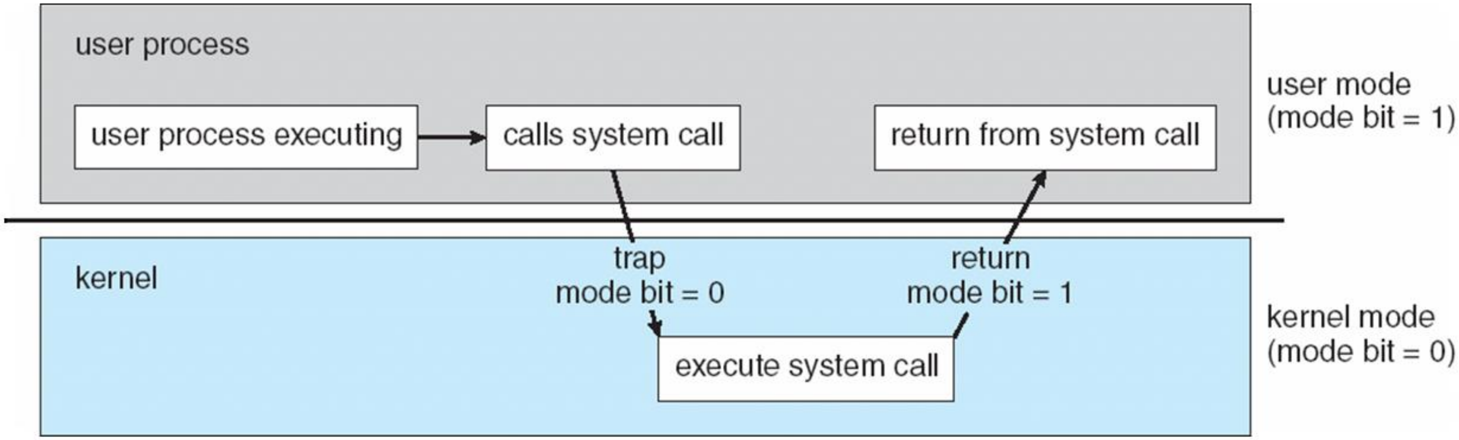
새로운 process 생성

명령 보내기



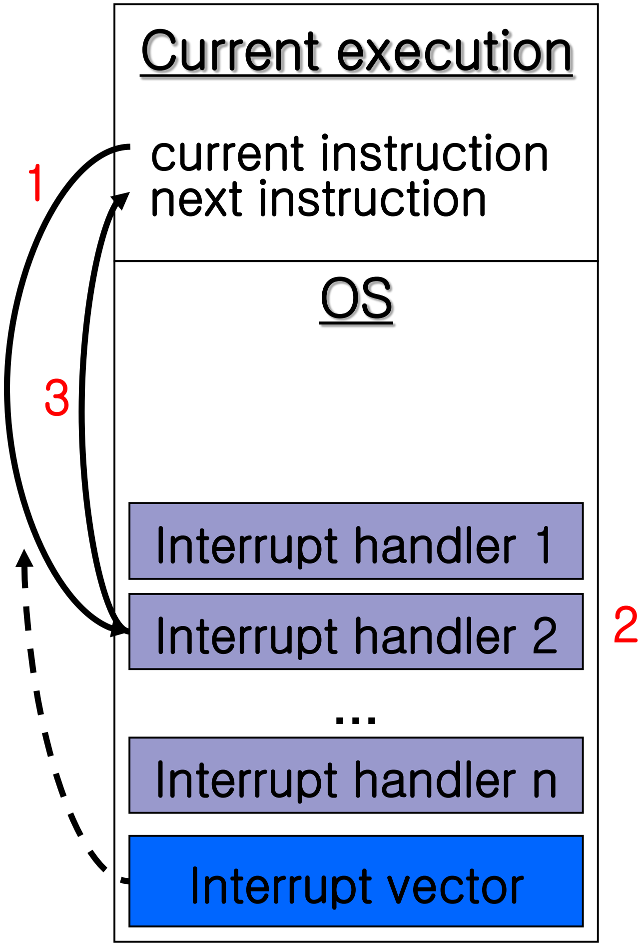
Dual Mode Operation

- 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



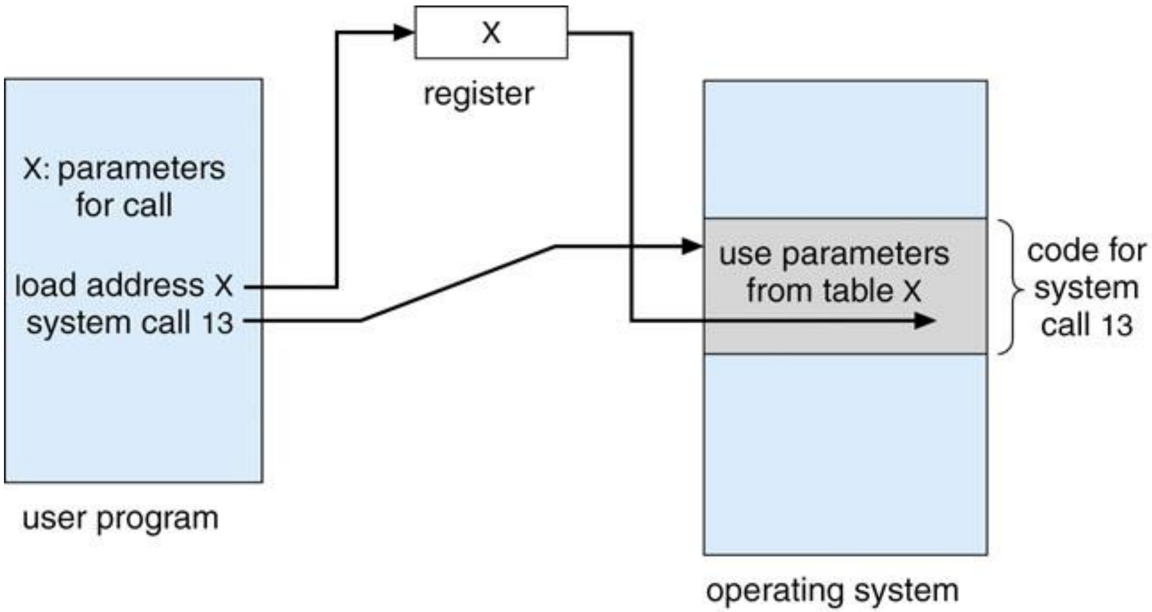
Interrupt Mechanism

- 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

- 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

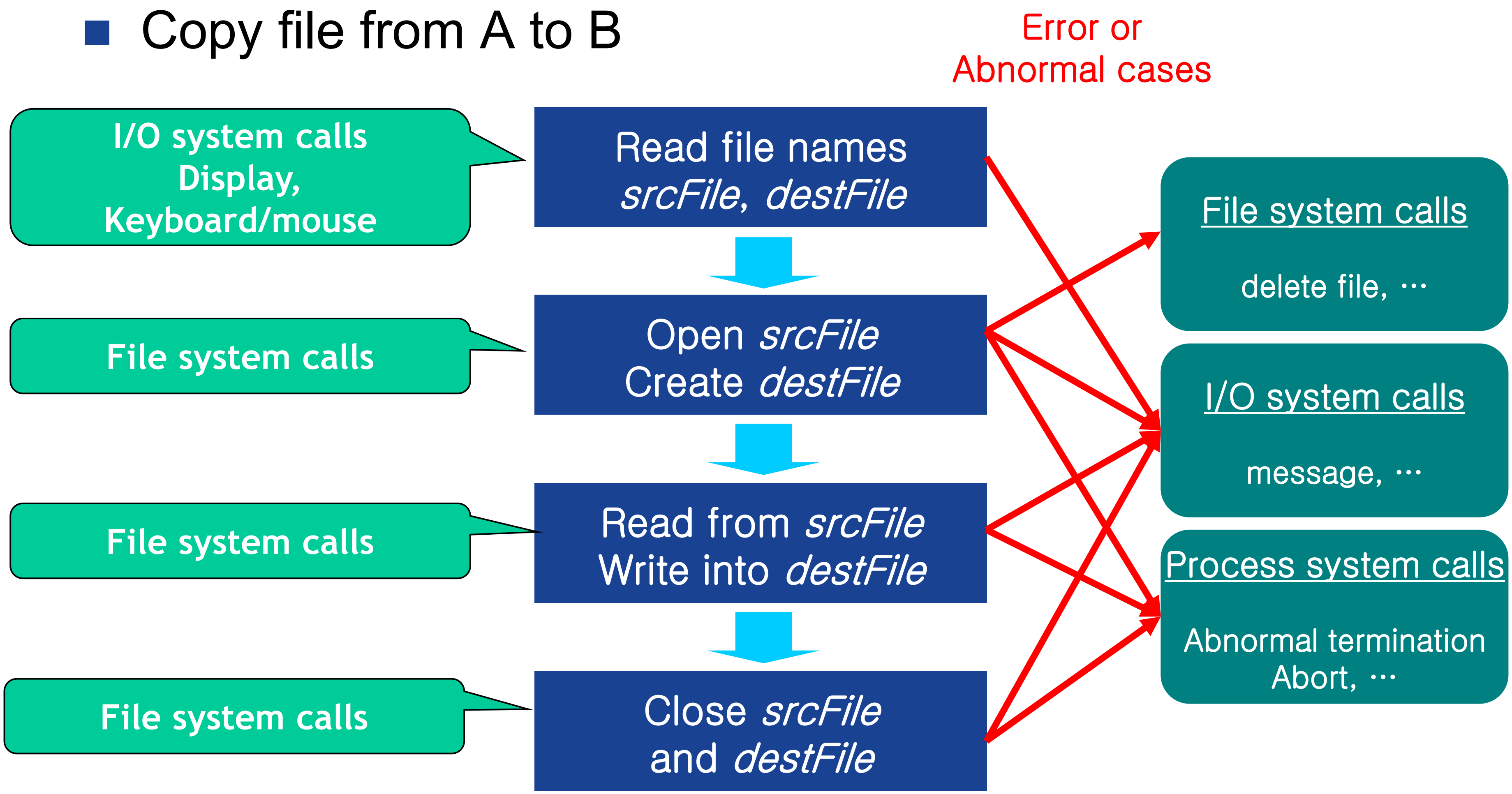


Types of System Calls

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

Example

- Copy file from A to B

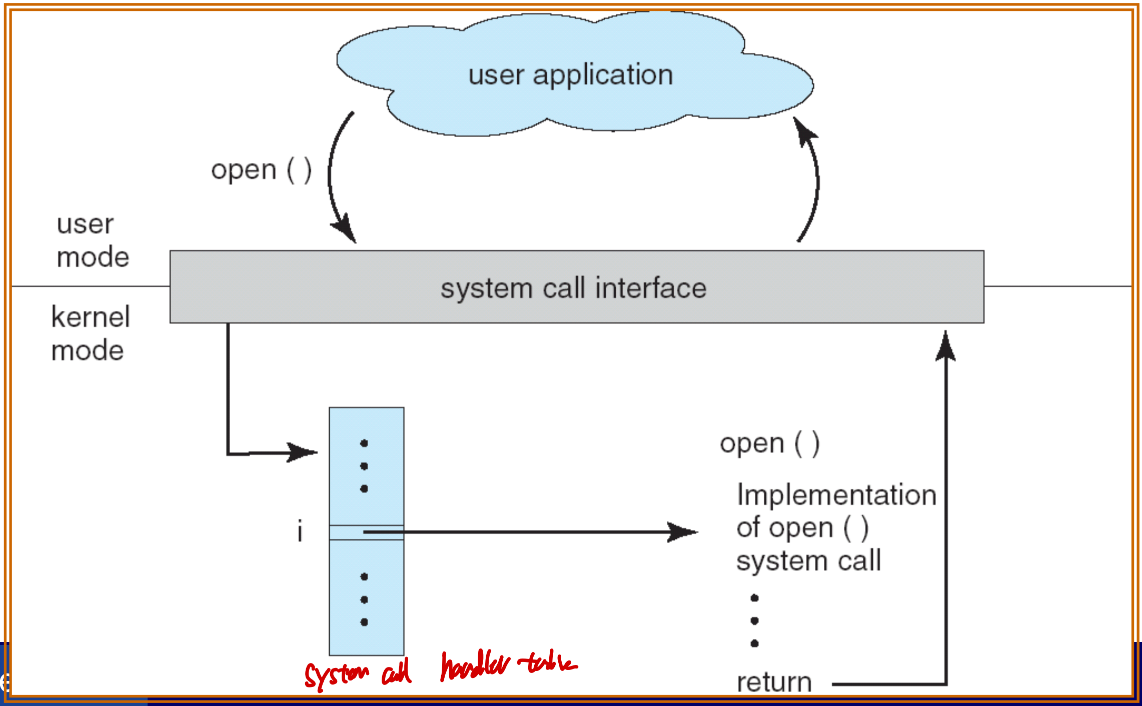


System–Call Interface

- 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

- Example of system-call interface in Linux

User program

```
int main()
{
   …
   open();
   …
}
```

System–call Interface (libc)

```
open()
{
   …
   movl 5, %eax   ‘system call number
   int $0x80      ‘generate interrupt
   …
}
```

5를 %eax에 넣어라

interrupt

80

Interrupt Handling Mechanism

kernel mode

OS kernel

kernel function

```
sys_open()
{
   …
}
```

System–Call Interface

- Typically, a number is associated with each system call.

  c.f. IRQ of system call: 0x80 on Linux, 0x21 on DOS/Windows
  - 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

- What does system-call interface do?
  - Passing information to the kernel
  - Switch to kernel mode int 명령
  - 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 OS의 고급기능을 이용하고 싶을 때
- fread(): standard function defined in C language
  - fread() is implemented using read()

Application Programming Interface

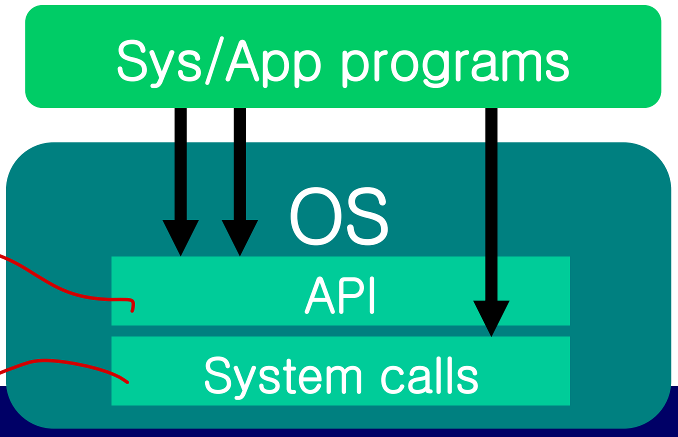
- **API**: interface that a computer system (OS), library or application provides to allow requests for service
  - A set of functions, parameters, return values available to application programmers.

    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



더 고급화된 함수

window가 버전에 따라 많이 바뀜 → low 레벨!
호환성 ↓

Example of API

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

return value

```
BOOL   ReadFile c  (HANDLE        file,
                    LPVOID        buffer,
                    DWORD         bytes To Read,
                    LPDWORD       bytes Read,
                    LPOVERLAPPED  ovl);
```

function name

parameters

- 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

Examples of System Calls

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

Process Control: Load/Execution

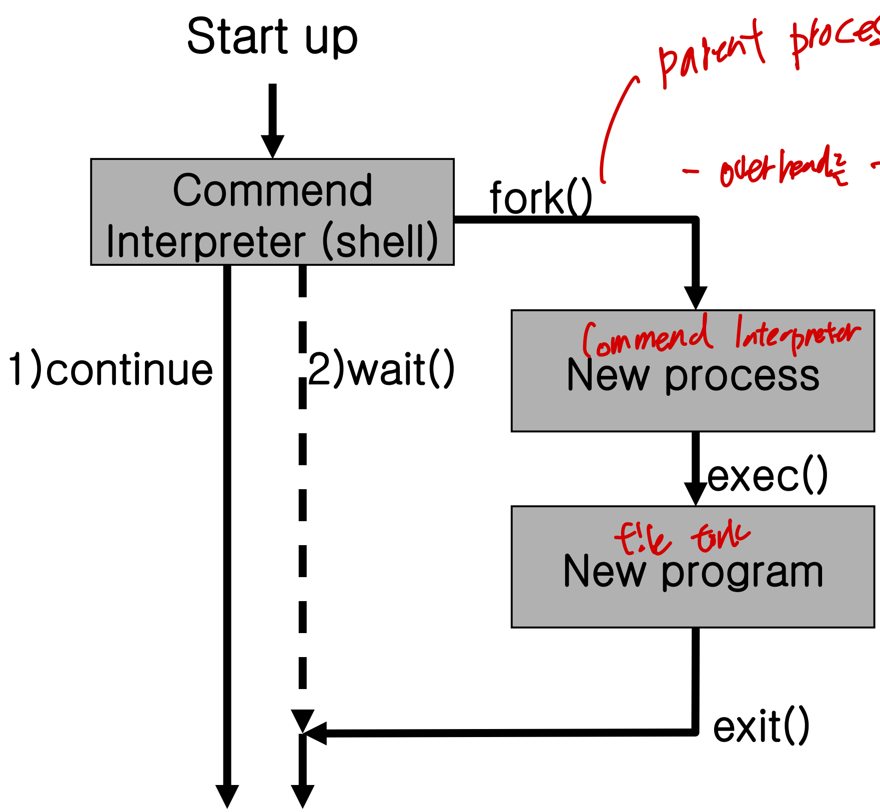
- A program can load/execute another program.
  Ex) CLI, Windows Explorer, MacOS Finder

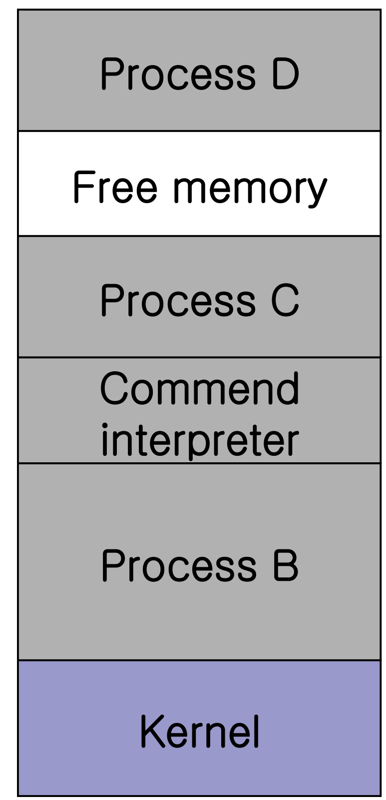
- While, the parent program can
  - Be lost (replaced by the child program)
  - Be paused
  - Continue execution: multi-programming/multitasking
    - Create process/submit job

ls , cat : 이것이 다 실행파일이다.

Example: FreeBSD UNIX

- Multitasking system





< FreeBSD running multiple program >

Example: FreeBSD UNIX

- Command interpreter may continue to execute

- Two cases of parent's execution
  - Case 1, continue to execution
    - New program is executed in background
      - Console input is impossible

  - Case 2, wait the child
    - New process takes I/O access
    - When the process terminates (exit()), the control is returned to parent (e.g. shell) with a status code (0 or error code)

Reading Assignment : 다음과제에서 사용

- Search Internet for documents on the following functions. Read them to understand how to make your program run another program.
  - fork() parent를 복사하기 때문에 아랫건 코드 필요없음
  - exec() family functions 파라미터를 이런 형식으로 넣는데에 따라 바뀜
    - execlp()
    - execvp()
  - wait()

int main (int argc char * argv [])
2

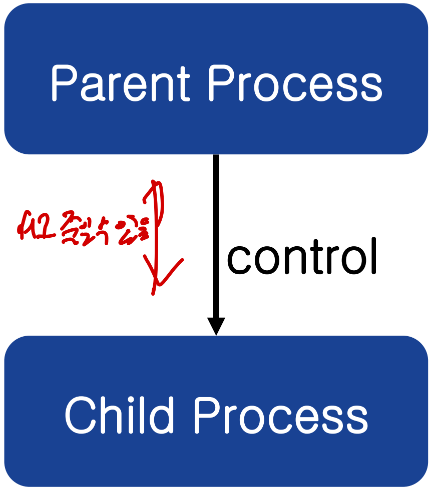
ls -al
[0] [1]

Process Control: Load/Execution

- Controlling new process
  - Get/set process attributes
    - Priority, maximum execution time, …
  - Terminate process

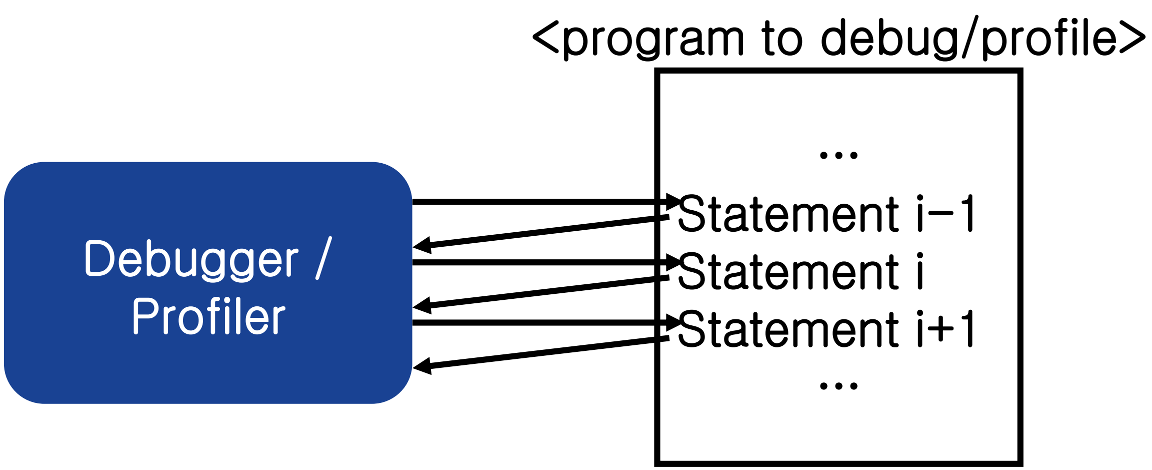
남의 process는 못죽임

- 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



Process Control: Termination

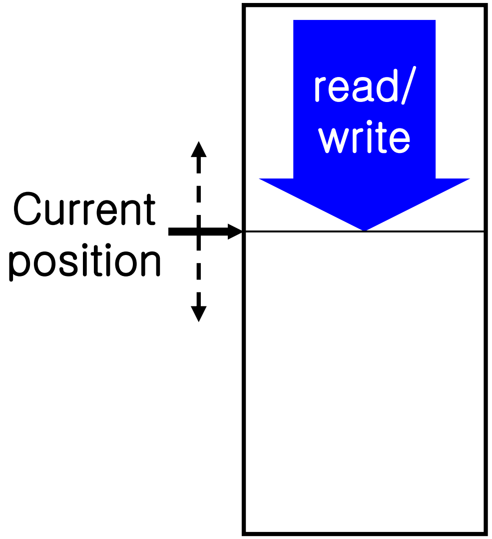
- Normal termination (end) exit()
  - Deallocate resources, information about current process

비정상 종료

- Abnormal termination (abort)
  - Dump memory into a file for debugging and analysis
  - Ask user how to handle

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 독점적인 사용 권한을 주는거

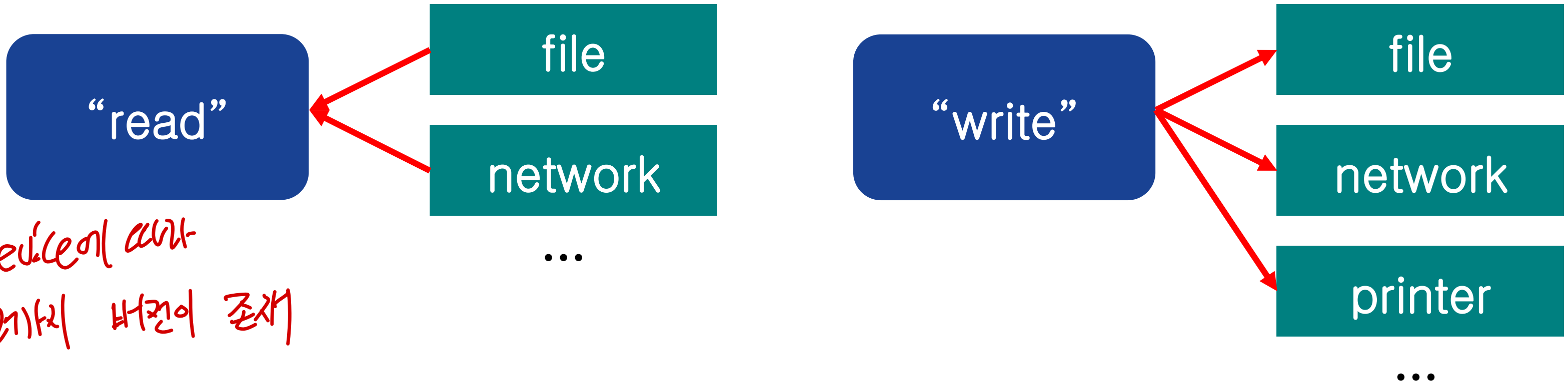
- Resources
  - Physical device (disk, tape, …)
  - Abstract/virtual device (file, …)

- 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

device-driver : operation 제공



device에 따라 여러가지 버전이 존재

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

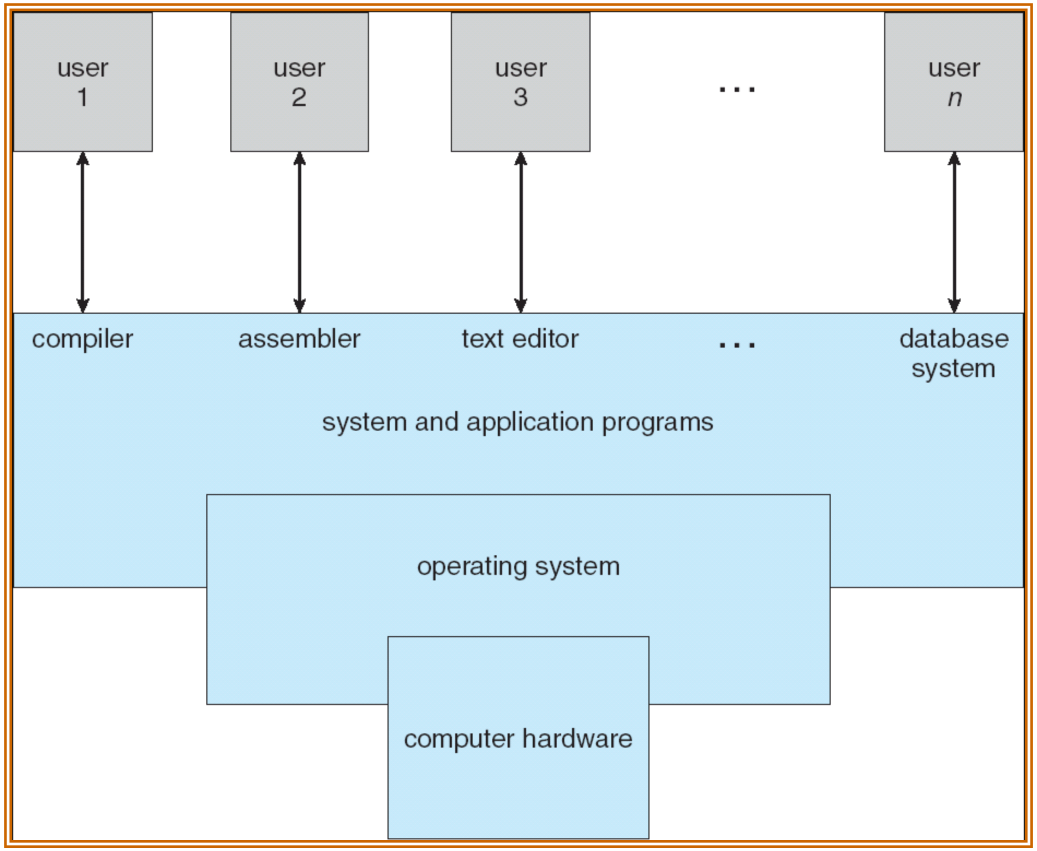
System Programs

kernel
system program
application

kernel
} - application

user-mode에서 돌아감

- 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
  - Monolithic structure
    - MS-DOS, original UNIX, Linux
  - Layered – an abstraction
  - Microkernel – Mach

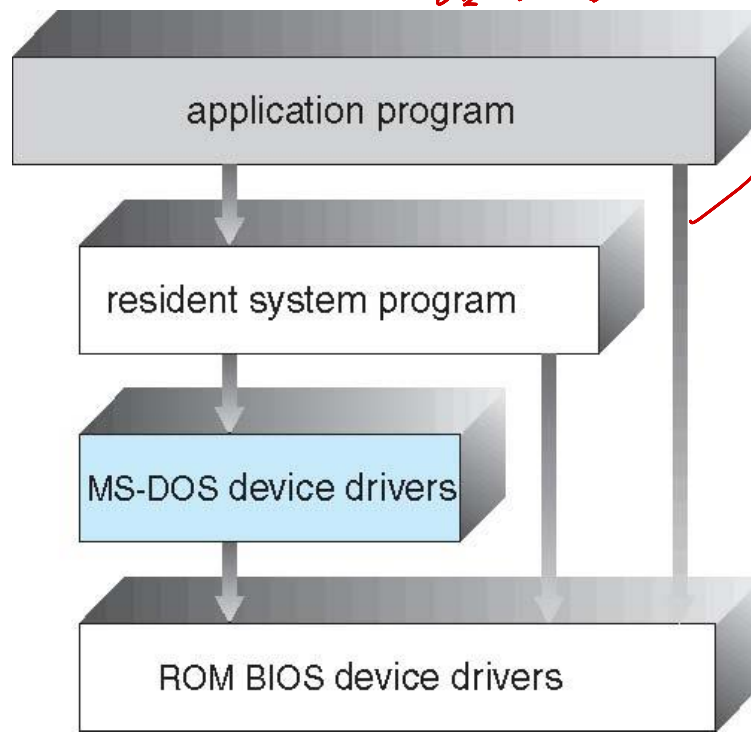
Simple Structure

single user, single task

- 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

장점 : 뛰어갈 수 있다

단점 : 위험하다



< Structure of MS-DOS >

단일체의

Monolithic Structure

- 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.
  - Fast!

요즘 Unix 구획이 잘 나뉘었음

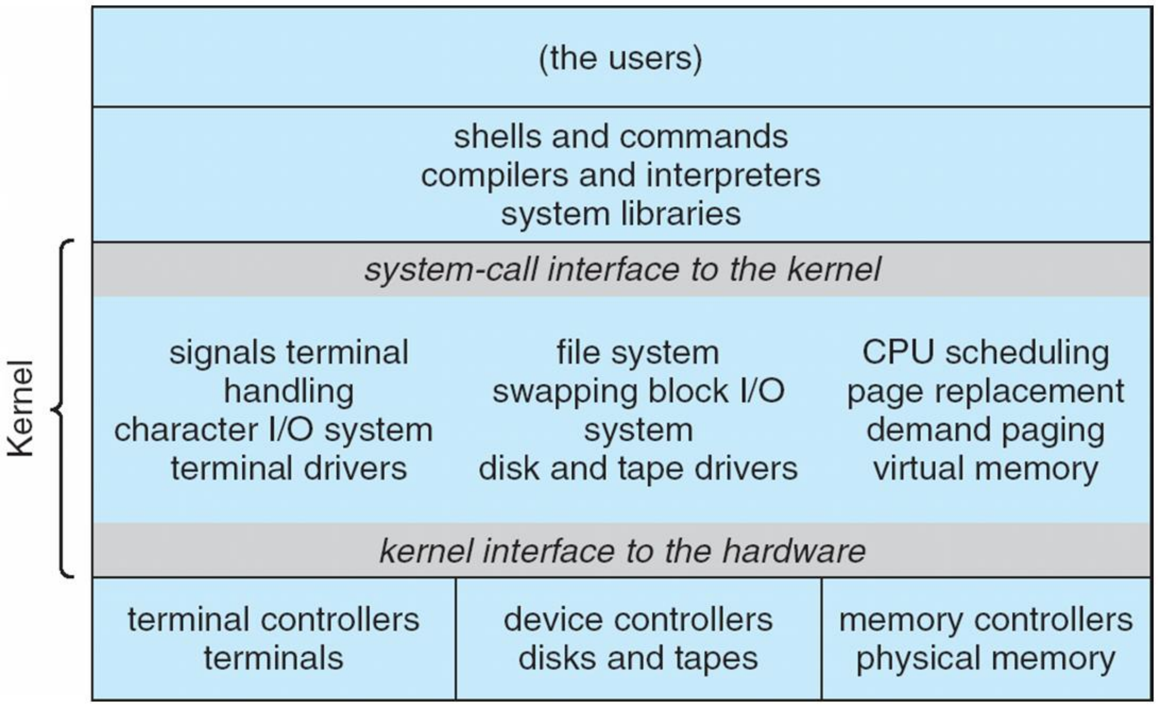
Ex) Original UNIX(1973)

- Also limited by H/W functionality
- Systems programs
  - Shell, commands compiler, interpreter, system library, …

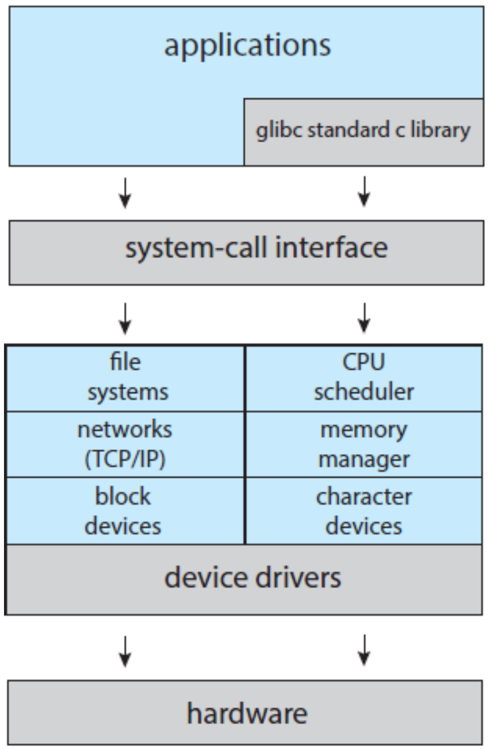
Ex) Linux (1991)

Monolithic Structure

개발, 유지보수 힘듦



Original UNIX



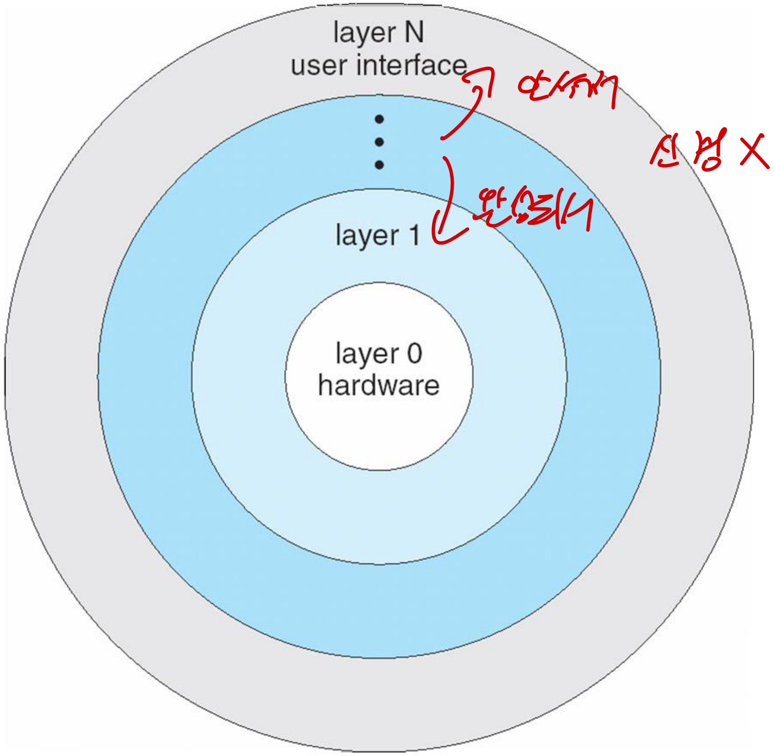
Linux

조기순 방법 3가지

1. Layered Approach : 양파같은 구조

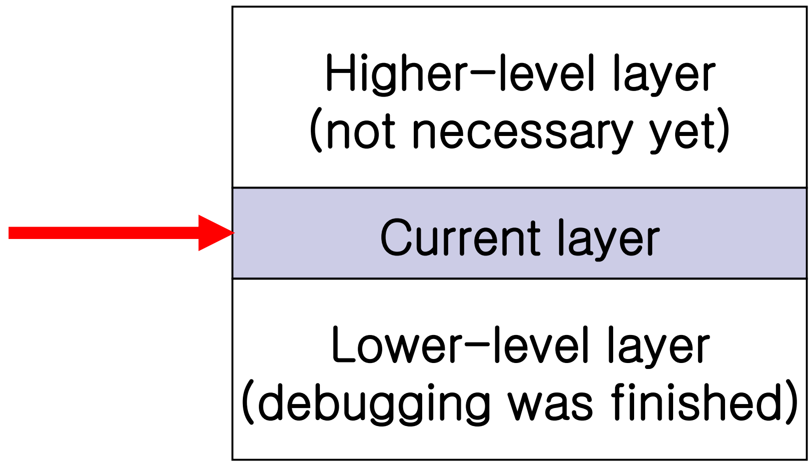
↳ 장점 디버깅 쉬움

- OS 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

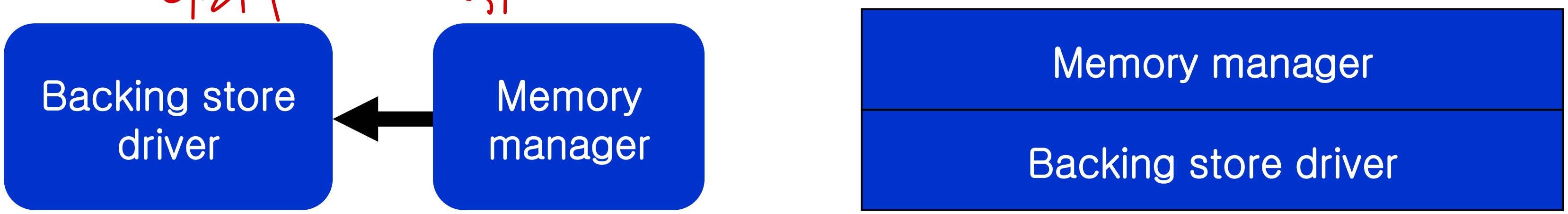
- Advantages of layered approach: simple to construct and debug
  - If we develop from lower-level layer to higher-level layer, we can concentrate on current layer at each stage
  - A layer doesn't need to know detail of lower-level layer



Layered Approach

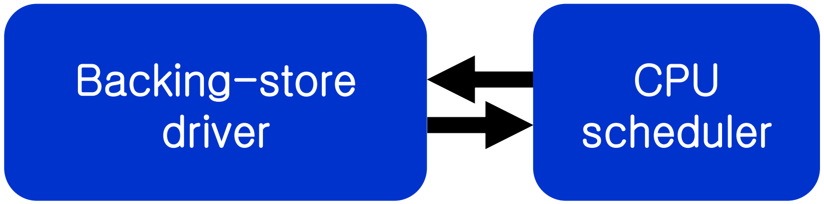
- Difficulties of layered approach
  - Defining various layers needs careful planning

아래 위



  - How to define hierarchy between the modules requires each other

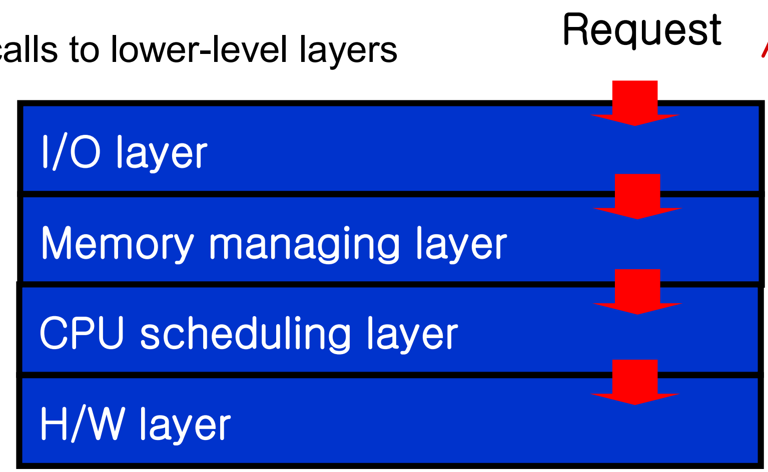
[illegible]



Layered Approach

- Difficulties of layered approach
  - Inefficiency
    - Repeating calls to lower-level layers

시간이 오래 걸릴수 있다.



- Remedy
  - Apply fewer layers - Take advantage and avoid difficulties

Microkernels 커널은 최대한 작게

장점: kernel이 문제가 생길 확률이 작아짐 → kernel에 버그가 없으면 system이 굳건함

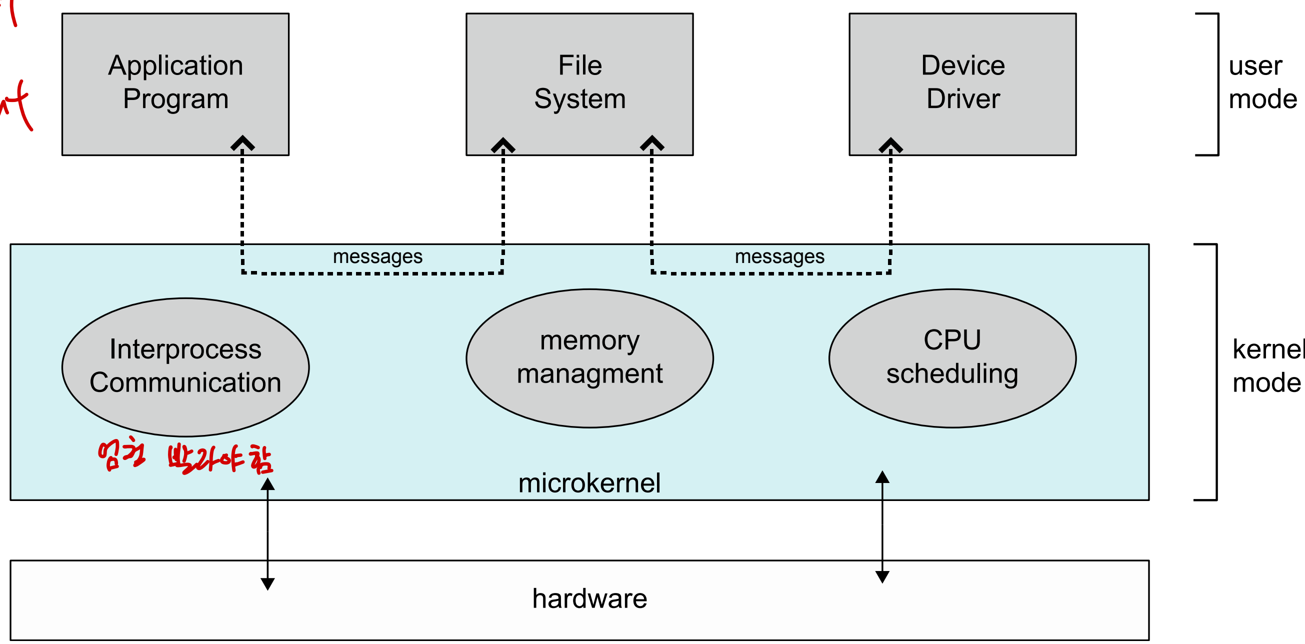
단점: 느림

- Smaller kernel 꼭 커널에서 해야 하는 것만 남기고 다 user-mode로 보냄
  - 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

process management

memory management

CPU에 대해서 알지 않는 기능을 빼내냄



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

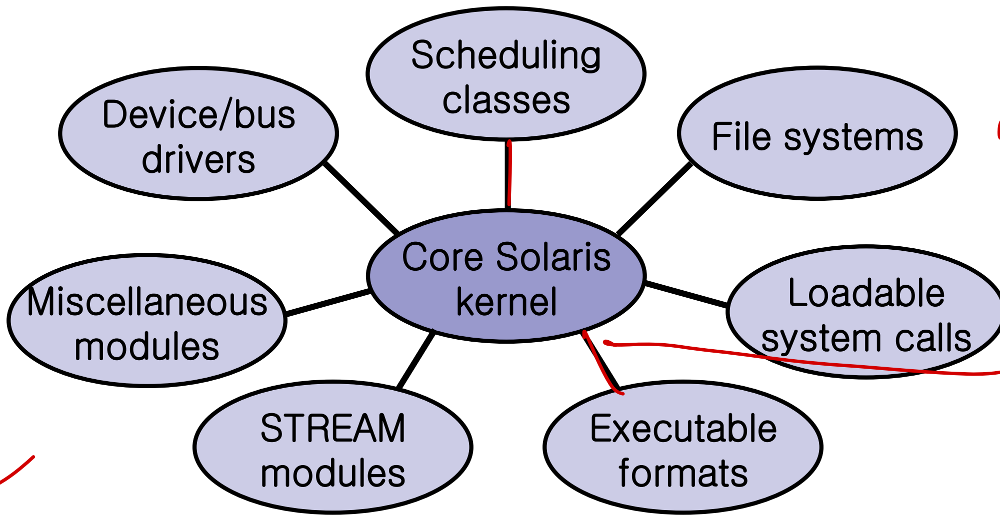
- Loadable kernel modules (LKM) flexable : 실행중인 모듈에 실시간으로 모듈 추가 가능
  - Uses object-oriented approach
  - Each core component is separated
  - Each talks to the others over known interfaces
  - Each is loadable as needed within the kernel

Ex) Linux, Solaris, etc.

내가 쓰는 모듈만 꺼내 사용가능

device driver 를 동시에 켜고 있을 필요 X

kernel mode



빠르다

function call과 맞먹는 속도

Modules

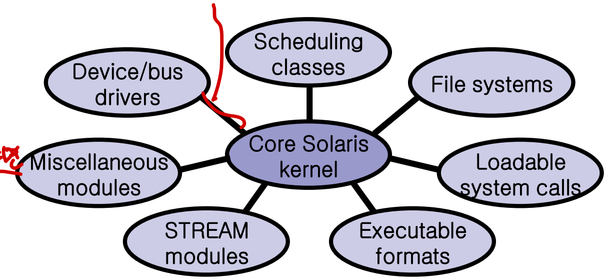
kernel 에서
printk 사용해야

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

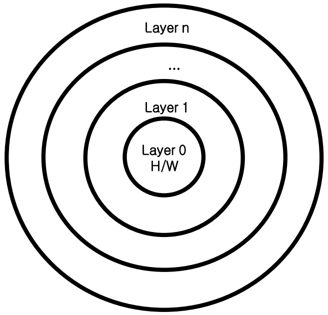
자유롭게 넣고 뺄 수 있음

약간 위험함

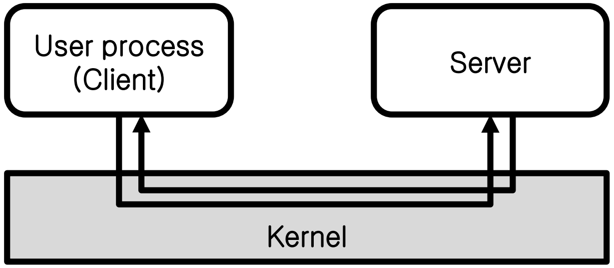
모듈 하나에 버그가 있으면 다 죽음



- 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

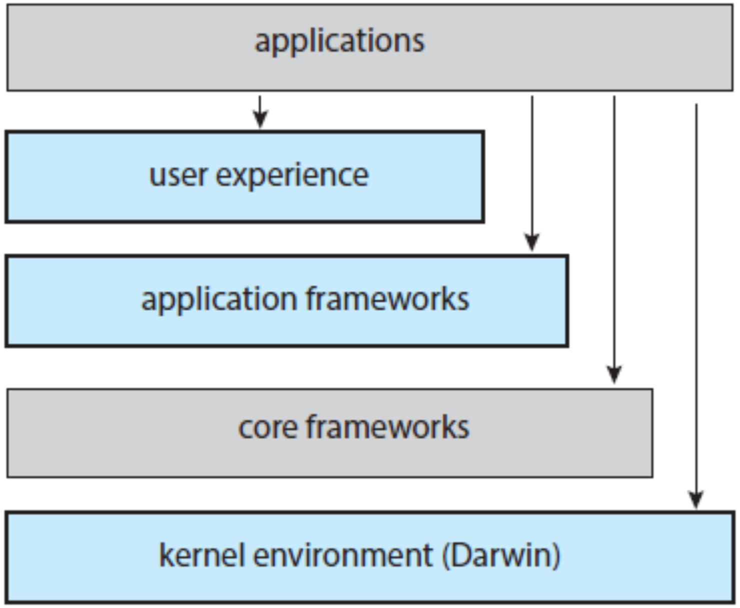


Hybrid Systems

- 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)

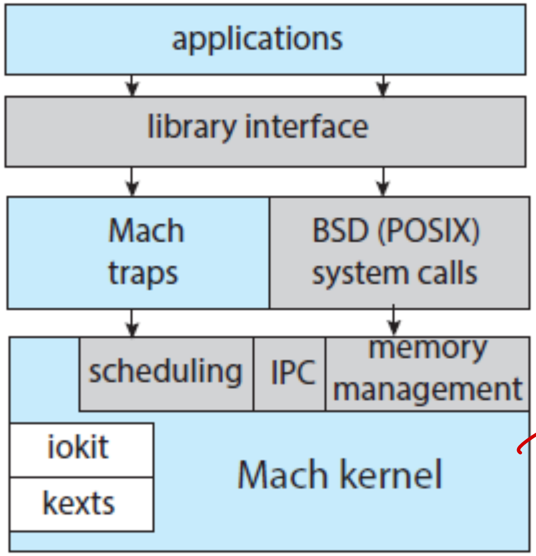
MacOS, iOS

- User experience layer
  - Defines the software interface that allows users to interact with the computing devices.
  - Aqua UI (MacOS), Springboard UI (iOS)
- Application frameworks layer
  - Provide an API for the Objective-C and Swift programming languages.
  - Cocoa (MacOS), Cocoa Touch (iOS) frameworks
- Core frameworks
  - Defines frameworks that support graphics and media including Quicktime and OpenGL.



MacOS, iOS

- Kernel environment (Darwin): hybrid structure
  - A layered system that consists primarily of the Mach microkernel and the BSD UNIX kernel.

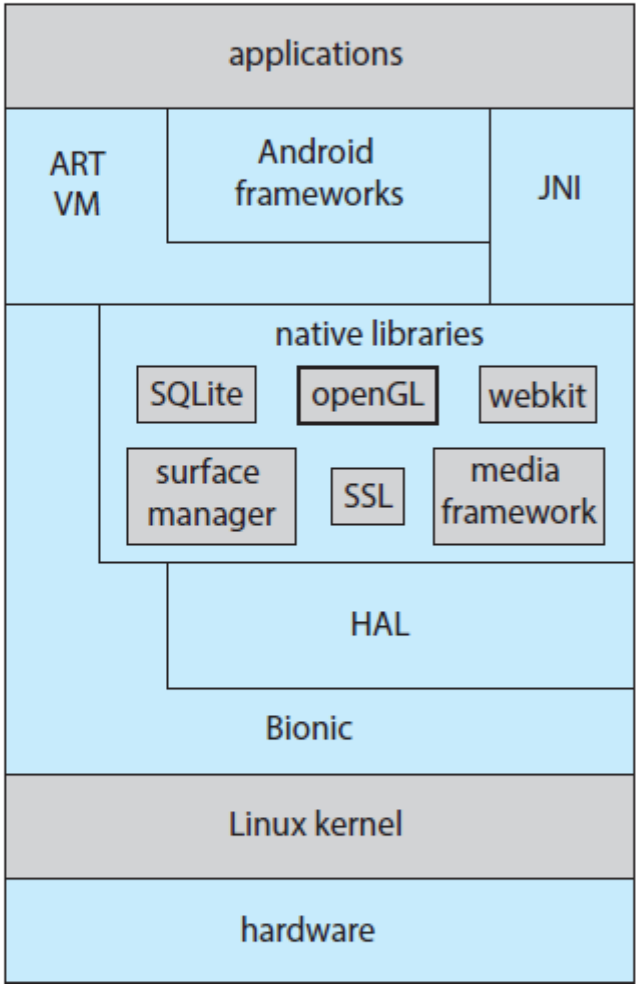


이방형 커방형, 다층

micro kernel

Android

- Android Run-Time (ART)
  - Ahead-of-time (AOT) compilation
- Java native interface (JNI)
- Native libraries
- H/W abstraction layer (HAL)
  - Consistent view independent of specific H/W
- Bionic: standard C library for Android
  - Android version of glibc.



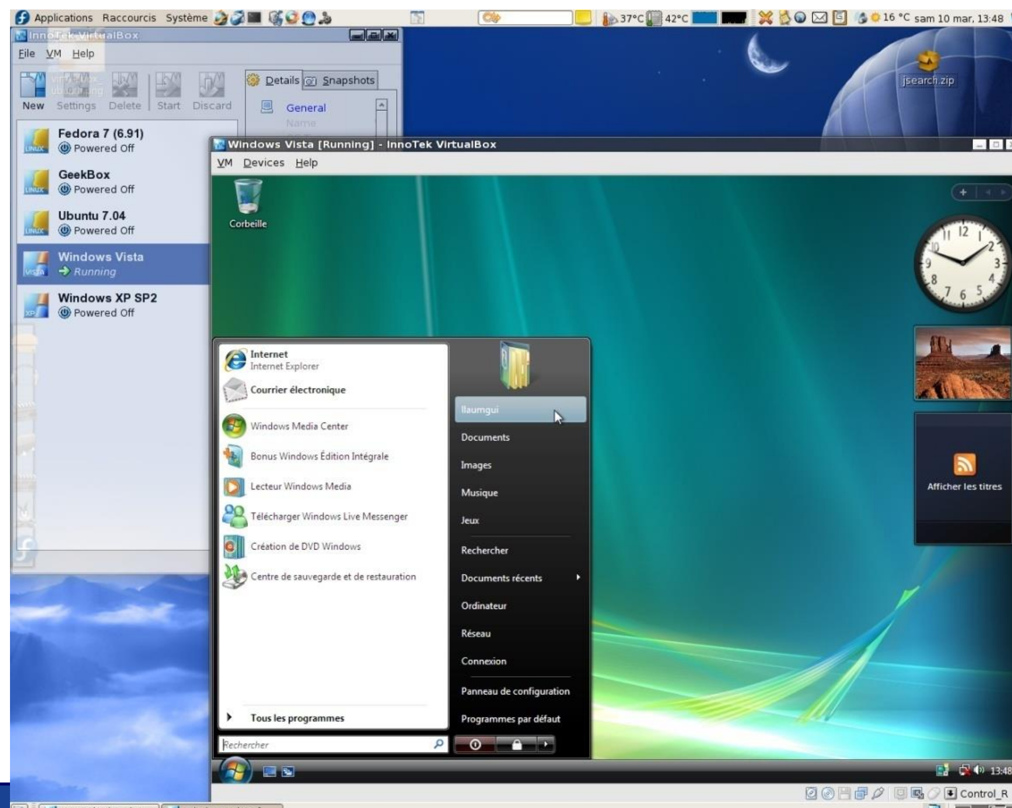
Agenda

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

Virtual Machines

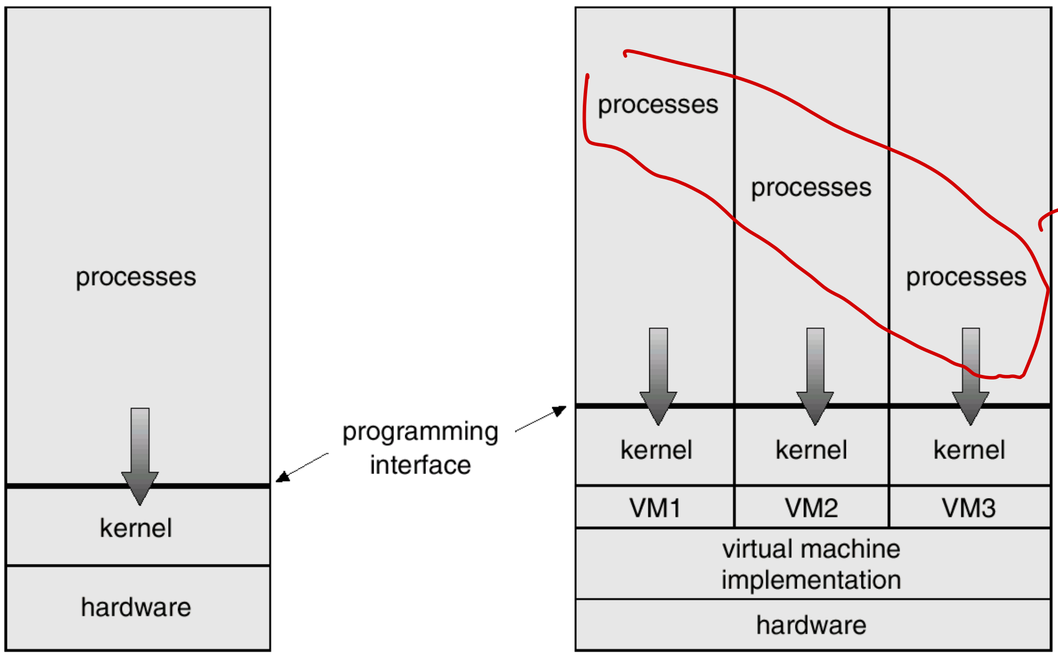
- **Virtual machine**: software that creates a virtualized environment (machine) between the computer platform and its operating system, so that the end user can operate software on an abstract machine.

  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.



process 수 많 > core 수 (o)

multi-tasking 으로 더 많이 사용

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

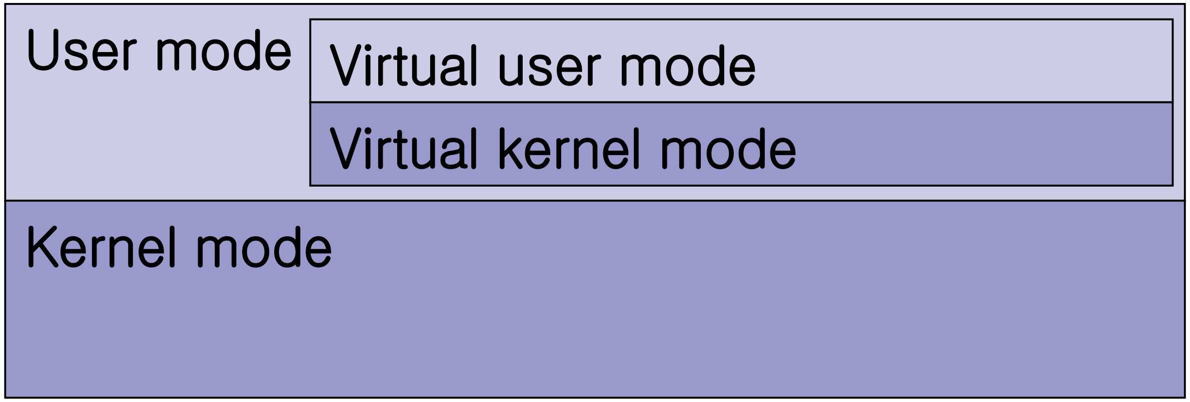
    큰 파일 하나를 만들어서 disk처럼 사용
  - Solution: virtual disks (minidisks)
    - Identical in all respects except size

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.



실제로 돌리지 않고
시뮬레이션 사용
↳ Kernel 모드 느낌

Virtual Machines

가상 머신은 백업 파일 사용 가능

- Benefits of VM host machine에서 실습하다가 죽으면 OS 다시 깔아야함
  - 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

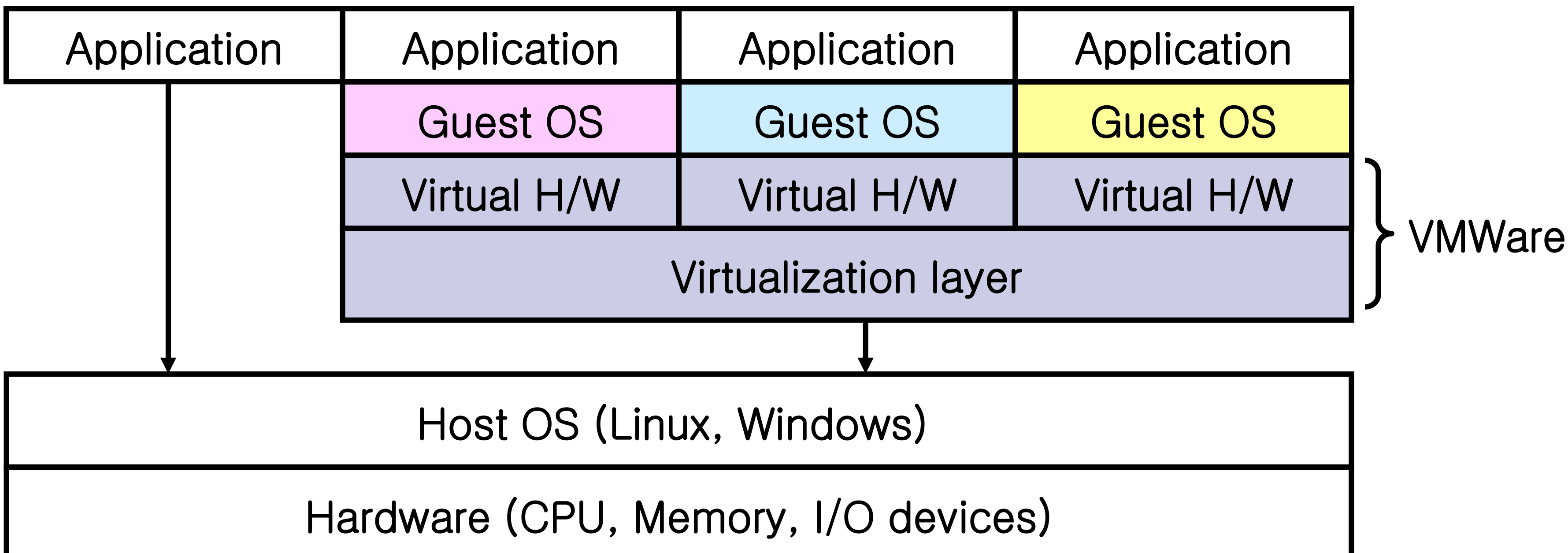
Virtual Machines

불가피한

- 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 유료, virtual box보다 좋음

- 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 가상머신

- 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