[Chapter 2]

경영학과/2018100233/김정민

I. Operating System Services

A. OS가 제공하는 것

1. 개요: OS는 하드웨어의 위, 유저 프로그램(applications)의 아래에 위치한다.

2. services

- file systems
- program execution
- I/O operations
- Communication
- Resource allocation
- Accounting
- Error detection
- Protection and security

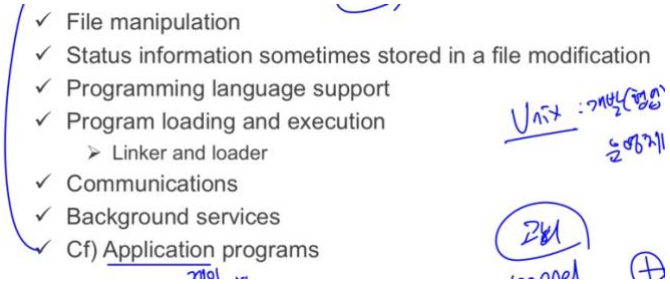
3. System calls

4. User interfaces

- Graphical User Interface (GUI)
  - 마우스, 키보드 등으로 제어
  - Ex. Mac OS X
  - iPhone 등의 Touch Screen Interface도 이와 연결되는 개념 (마우스, 키보드의 역할을 터치로 해결)
- Batch
- Command Line Interpreter (CLI)
  - 프롬프트에 명령어를 적어가면서 컴퓨터에게 일을 시킨다.
  - Bourne shell, bash(Bourne again shell) 등
  - 왜 이름이 shell(껍질)? 명령어 상에서는 필요한 기능들이 많이 보이지 않는다. (abstraction)

5. System programs

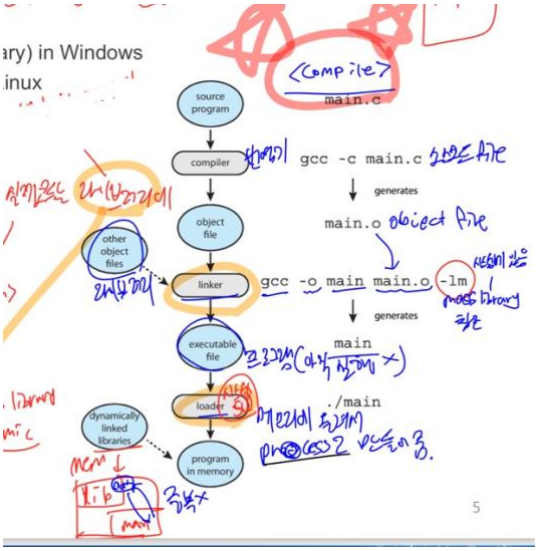
- 사용자(개발자)에게 프로그램을 개발하는 데 편리한 환경을 제공한다.
- Ex. 리눅스를 깔면 gcc가 따라서 깔린다. (맥, 유닉스도 마찬가지)
- OS마다 목적이 조금씩 다르다.
- Unix : 개발(협업) 목적
- Mac, windows : gpos (general purpose OS)



6. Linkers and Loaders

- Static linking
  - 컴파일러와 링커를 거쳐 실행 가능한 파일을 만들 때, 라이브러리의 필요한 함수 코드를 실행 가능한 파일에 직접 포함한다.
  - 프로그램 네 개에서 A() 함수를 호출할 때, 링커는 라이브러리에서 A() 함수의 바이너리 코드를 가져와서 4개의 프로그램에 각각 배정한다. 그래서 메모리에는 똑같은 코드가 4번 중복되게 로드된

다.
    - 따라서 실행파일의 크기가 증가할 수 있다.
    - 라이브러리가 업데이트되면, 프로그램들을 다시 컴파일해야된다.
- Dynamic linking
    - 메모리 한 켠에 라이브러리를 올려놓고 프로그램을 실행하다가 필요할 때마다 라이브러리에서 코드를 로드해와서 사용한다. (실행파일에 필요한 라이브러리 코드를 포함하지 않는다.)
    - 중복이 덜 발생한다.
    - 라이브러리의 업데이트가 편리하다.
    - Windows : .dll (dynamically linked library) 파일
    - Linux : .sa & .so (shared library) 파일



7. System call services
- Dual mode operation (user / kernel)
- Trap을 활용한 function call
    - 웹서버에서 read() 명령어가 들어오면, user mode는 kernel mode에 trap을 날린다. 그리고 현재의 상태를 저장해놓고 기다린다.
    - 디스크에서 파일을 읽는 I/O는 kernel한테 권한이 있다.
    - Kernel이 수행한 후에 리턴값을 user에게 전달한다.
    - 이때 user는 리턴값을 받아서 바로 CPU를 사용하지 못할 수도 있다. (scheduler)
- 벡터값 활용
    - 자주 쓰이는 인터럽트 및 트랩(SW 인터럽트)들을 벡터값에 매핑해서 사용한다.
    - Kernel mode는 매핑 테이블을 가지고 있고, 인터럽트를 전달받게 되면, 필요한 명령을 바로 찾아서 수행할 수 있다.
    - SW 인터럽트는 다양하다.
    - 0 : open(), 1 : read(), 2 : write(), 3 : close() <- SW 인터럽트 (트랩)
    - 0 : Timer, 1 : DMA, 2 : 키보드, 3 : 마우스 <- HW 인터럽트
- 파라미터는 레지스터에 저장해서 OS로 전달한다.
    - 펑션콜의 경우에는 CPU 레지스터 혹은 스택에 쌓아놓고 indirection
- 시스템 콜은 직접적으로 사용자가 건드릴 일이 없는 편이다.
    - 잘못된 수정 등의 위험 요소가 있다.
- UNIX (POSIX 표준) vs Windows (무근본) 시스템콜 비교

■ System call service

Process Management	fork	CreateProcess	Create a new process
	waitpid	WaitForSingleObject	Wait for a process to exit
	execve	(none)	CreateProcess = fork + execve
	exit	ExitProcess	Terminate execution
	kill	(none)	Send a signal
File Management	open	CreateFile	Create a file or open an existing file
	close	CloseHandle	Close a file
	read	ReadFile	Read data from a file
	write	WriteFile	Write data to a file
	lseek	SetFilePointer	Move the file pointer
	stat	GetFileAttributesEx	Get various file attributes
	chmod	(none)	Change the file access permission
File System Management	mkdir	CreateDirectory	Create a new directory
	rmdir	RemoveDirectory	Remove an empty directory
	link	(none)	Make a link to a file
	unlink	DeleteFile	Destroy an existing file
	mount	(none)	Mount a file system
	umount	(none)	Unmount a file system
	chdir	SetCurrentDirectory	Change the curent working directory

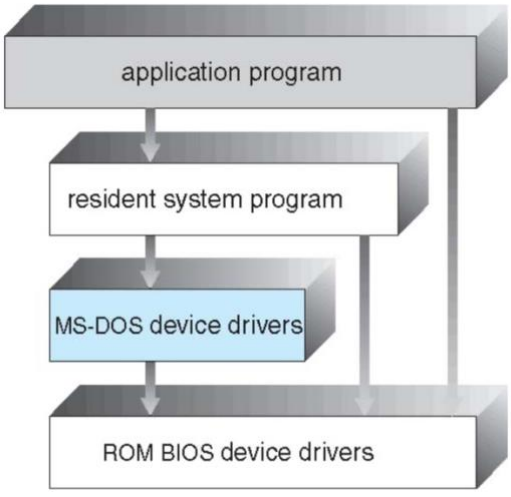
II. Operation System Structure

A. 커널의 종류 (큰 분류)

1. Monolithic kernel
   - OS가 가져야하는 모든 기능을 한 덩어리(integrated kernel)에 넣는다.
   - Ex. Unixware, Solaris, AIX, HP-UX, Linux, 등
2. Micro kernel
   - 정말 필수적인 것들만 커널에 담는다.
   - Multiple servers
   - Message passing
   - 플러스 알파의 성격을 지닌 기능들은 application의 형태로 user mode로 올린다. (user mode의 권한이 강화된다.)
   - Ex. Mach, Chorus, Linux mk, 등

B. 커널 구조의 진화 과정

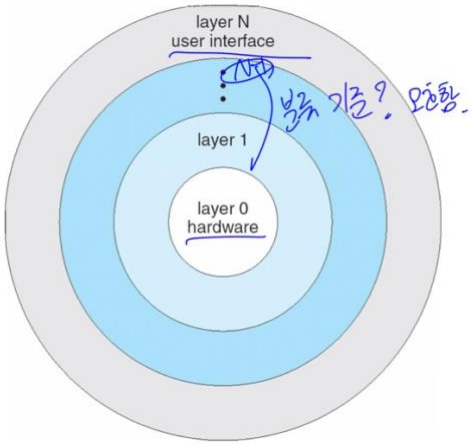
1. MS-DOS (Simple structure)
   - OS보다는 "monitor"라고 볼 수 있다.
   - 싱글 프로세스 시스템



   -
2. Monolithic structure
   - 전통적인 UNIX
     - Multiprocessing, time-sharing, GUI 지원
   - OS에게 필요한 모든 기능을 한 덩어리로 만든다. -> 커널
   - 커널 모드가 수행될 때 돌아가는 코드의 양이 많다. (무겁다.)

- SW engineering issue (단점)
  - 커널에 모든 게 다 들어있기 때문에 의존성이 높다. -> 하나 고치면 연쇄 수정 일어날 가능성 있다.
  - 유지보수 비용이 높다.
  - 하드웨어가 좋아야 한다. (과거에는 나름 중요한 제약조건)

3. Layered approach
- Micro kernel 이전에 등장한 "개념"
- 말 그대로 개념적이고 이상적이다.
- 레이어를 나누는 기준이 모호해서 실제로 구현하기는 어렵다.

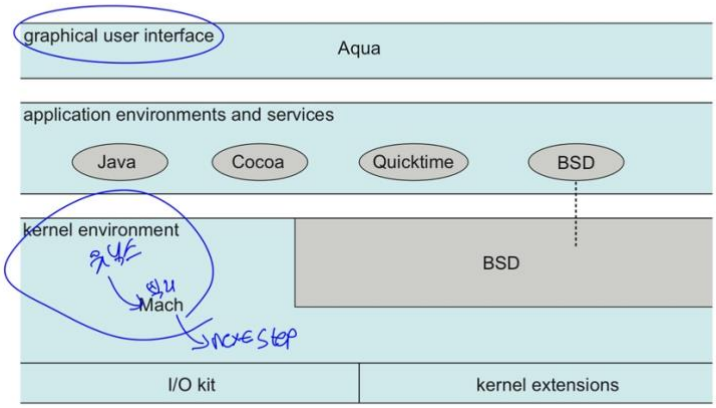


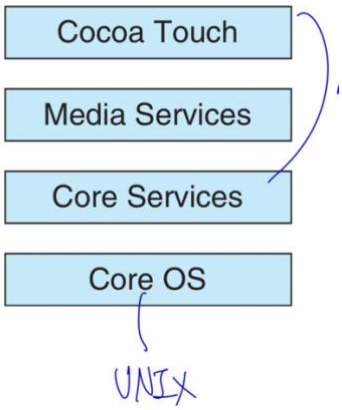
4. Microkernel structure
- 파일시스템, 디바이스 드라이버와 같은 부수적인 기능들은 유저모드의 application으로 올린다.
- IPC, Memory management, CPU scheduling과 같은 필수적인 기능들은 커널에 포함된다.
- 즉, 커널의 기능이 최소화된다.
- 마이크로 커널의 경우 커널이 가벼워지는 대신, 단점도 존재한다.
  - 시스템 콜 한 번으로 모든 걸 해결할 수 없다.
  - Trap을 통해 application들과 데이터를 주고받으면서 작업을 수행한다.
  - 여러 단계를 거쳐서 작업을 수행하기 때문에 오버헤드가 존재한다.
  - 모드 스위치가 자주 발생할수록 성능은 떨어진다.

5. Modular approach
- Loadable Kernel Module(LKM)
- Ex. Linux(과거에는 Monolithic), Solaris
- 커널을 코어로 두고, 필요한 모든 기능들을 커널에 연결된 모듈화한다.
- 기능들은 서로 직접적으로 연결되어 있지 않기 때문에 의존성이 낮다.
- 컴파일을 통해 필요한 기능들만 커널로 이미지를 만들어서 사용한다.
- 커널의 크기가 매번 달라진다.

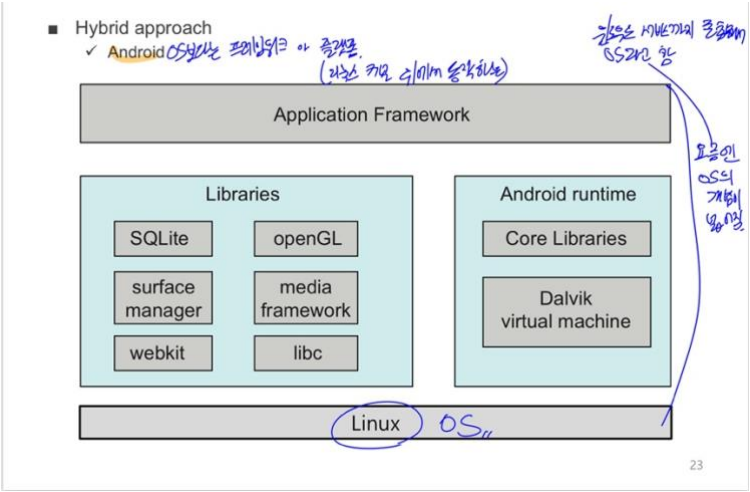
6. Hybrid approach
- Mac OS X, IOS
  - 커널은 유닉스 기반의 Mach 마이크로 커널과 BSD의 요소를 결합한 형태다.
  - Aqua라는 GUI를 통해 컴퓨터를 조작한다.
  - Application은 자바, 코코아, 퀵타임 등을 이용한다.

■ Hybrid approach
✓ Mac OS X



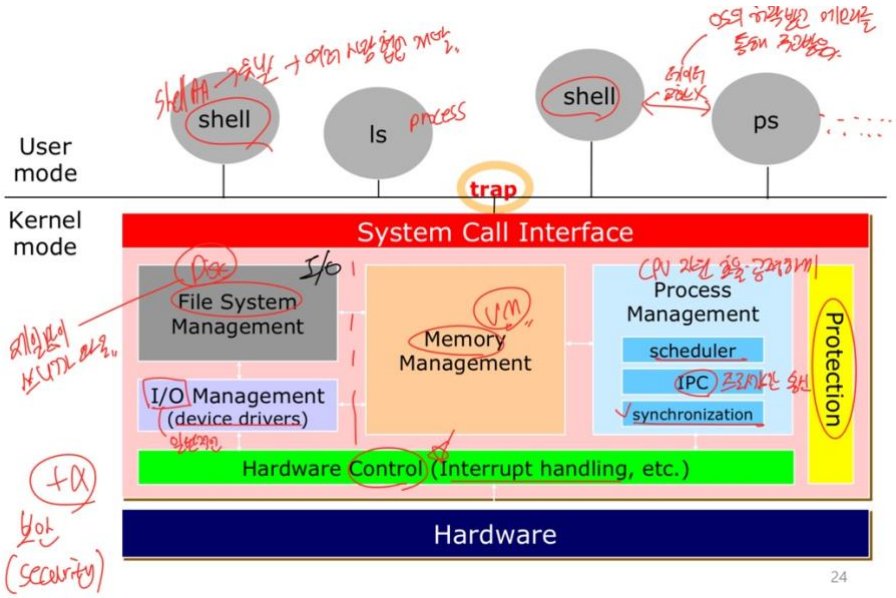


- Android
  - OS보다는 리눅스 커널 위에서 동작하는 프레임워크나 플랫폼에 가깝다.
  - 최근에는 OS의 개념이 넓어지면서 서비스 애플리케이션까지 포함해서 OS라고 하기도 한다.



III. Operating System (2장 요약)

A. OS 구조



1. Dual mode (User)
   - 유저 모드에는 다양한 application이 존재한다.
   - Shell은 여러 사람이 협업하면서 개발하기 위한 것이다.
   - 프로세스 간에는 데이터를 직접적으로 교환할 수 없다.
     - OS에게 trap을 보내서 메모리를 통해서 주고받아야 한다.
2. Dual mode (Kernel)
   - Hardware Control (Interrupt Handling)
   - Protection (메모리 프로텍션, CPU 프로텍션 등)
   - Process Management
     - Scheduler
     - IPC
     - Synchronization
   - Memory management
     - Virtual memory
   - I/O management
     - File System (Disk)
       - 제일 많이 쓰이는 I/O이기 때문에 따로 빼놓는다.
     - Device drivers