3강

System Structure

- 운영체제는 규모가 매우 크고 복잡한 소프트웨어
  - 설계 시 소프트웨어의 구조를 신중히 고려해야 함

    설계를 잘해야 하기에 스트럭쳐를 잘짜야 함

- 좋은 설계를 통해 쉬워지는 것들.
  - 개발(develop)
  - 수정 및 디버깅(modify and debug) - 오픈 소스 때문에 개발의 패러다임이 바뀌었다. 이제는 맨땅의 해딩이 아닌, 기존에 만들어진 것들을 잘 조합하고 자기의 contribution을 어떻게 할 것인가 의 문제가 되었다.
  - 유지 보수(Maintain)
  - 확장(Extend)

- 디자인 목표 중에 좋은 것이란?
  - 설계하고자 하는 시스템의 목적과 관계가 있음

OS design principle

Mechanism

- Determine *how* to do something
  - 무엇을 어떻게 할 것인가?
- Ex) The concrete algorithms, data structures

Policy

- Decide *what* to be done

    - 무엇이 되게 할 것인가? - high level 관점에서 슈퍼컴퓨터처럼 무조건 빨리 돌게 할 것인가? 아니면 PC처럼 interation을 중요시 할 것인가?
- Supposed to be higher level, and use mechanism

- Mechanism과 policy를 분리 함으로서 운영체제 설계를 보다 module화 할 수 있음

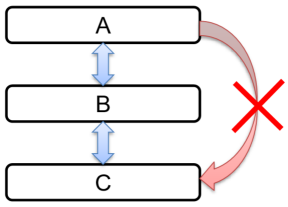
  → 분리해서 생각하면 편하다.

Layering

- OS의 복잡도를 낮추기 위한 방안
- Layer는 정의가 명확한(well-defined) 함수들로 이루어짐
- 하나의 layer는 인접한 layer와만 통신
    - 위, 아래에 인접한 layer만과 통신하며, 2단계 이상 건너뛴 layer와 직접적으로 통신하지 않음

      adjacent, 인접한 레이어만 서로를 호출한다.

      커널에서 버그가 나서 고치려면 그 커널을 내려야 한다. 그런데 그 레이어를 교체해도 다른 인접한 레이어는 그 사실을 모른다.



- 설계의 복잡도를 낮출 수는 있으나, 그로 인해서 overhead가 발생함
    - Ex) The 7-layers of the OSI model

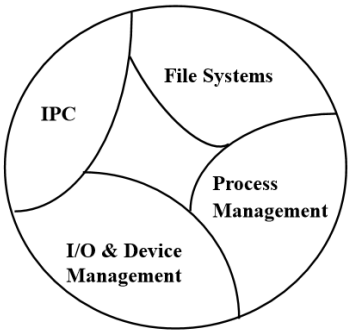
      그런데 어떤 경우는 A→C를 바로 호출하는게 좋을 수도 있다.

      그런데 반드시 B를 거쳐야 한다. 이게 문제이다. relax해야하나? relax를 어느정도 해야하나?

Layering vs Modularity

- Layering의 장점

- Layer의 수정이 다른 layer와 독립적임



< Modularity >

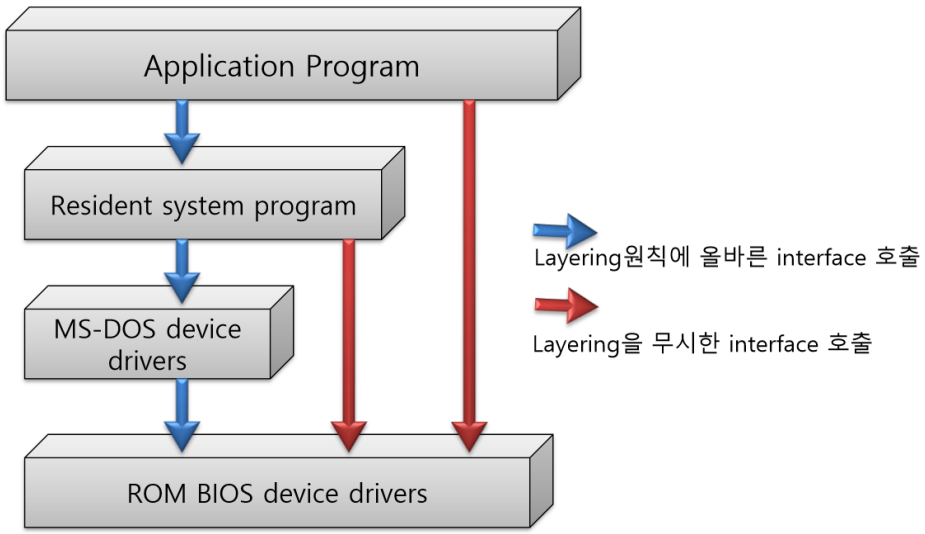
< Layering >

불완전한 Layering

MS-DOS가 이걸 잘 못했다.

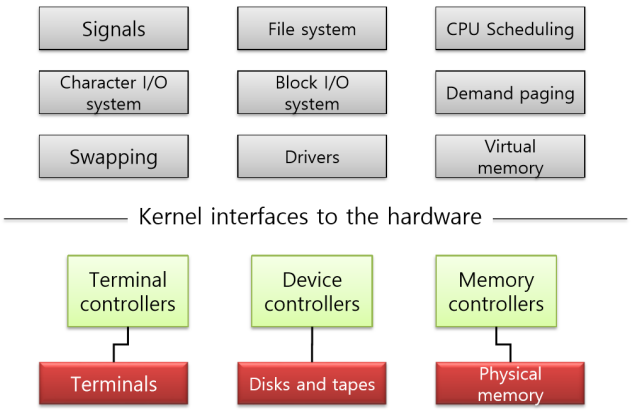
ROM BIOS device drivers .=. kernel

사용자가 직접 ROM BIOS device drivers를 호출 시킬 수 있는데 그 때문에 crush가 자주 일어났다.



< Example : MS-DOS. Interface is not well separated >

Kernel 내의 모듈들



< Example of modular structure - Linux >

kernel안의 모듈들로는 file system

커널내의 구조를 잘 보자

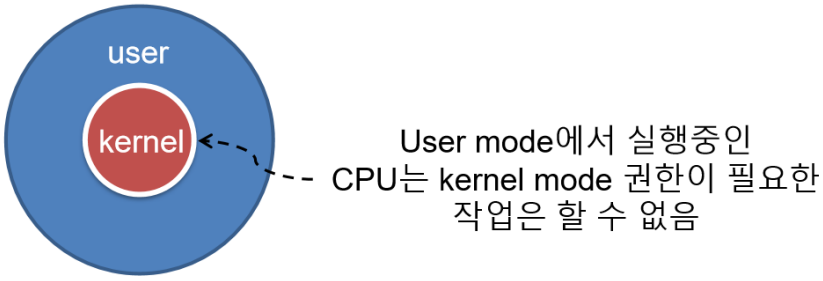
하단에는 메모리 장치들이 있다.

Terminal이라는 하드웨어가 있다. 그리고 이 하드웨어를 제어하는 프로그램이 있다.

그리고 그 드라이버와 커널이 소통한다.

user program은 또 커널과 다른 레이어에 존재한다. 커널 영역, 사용자 영역, 디바이스 드라이버 영역

CPU 실행모드



하드웨어 위에 바로 올라가는 것이 OS이다. 거기에서 돌아가는 것이 커널이다.

OS에서 바로 하드웨어에 접근하는 것을 막지는 못한다.

- CPU의 2가지 이상의 실행 모드

    - System protection을 위해서 필요
    - 실행 모드의 권한에 따라 접근할 수 있는 메모리, 실행 가능한 명령어가 제한됨
- 각각의 모드 별로 권한(privilege)이 설정됨
- Hardware 지원이 필요
    - Intel: ring 0~3, 4개의 모드 제공
    - 그 외 프로세서: 2개의 모드 제공

→ User mode에서 실행중인 CPU는 kernel mode 권한이 필요한 작업은 할 수 없음

User/kernel/hyperbize

User mode와 Kernel mode 비교

- Kernel mode
    - 운영체제가 실행되는 모드 (하드웨어를 제어)
    - Privilege 명령어 실행 및 레지스터 접근 가능
        - 예) I/O 장치 제어 명령어, memory management register – CR3
- User mode
    - 직접장치에 접근을 못함
    - Kernel 모드에 비해 낮은 권한의 실행 모드
    - Privilege 명령어 실행은 불가능
    - 모드를 변환하여 하드웨어에 접근한다. 그것이 시스템 콜이다.
- 실행 모드 전환 (execution mode switch)
    - CPU의 실행 모드는 설정은 시스템 보호가 목적
    - User mode에서 실행중인 어플리케이션이 kernel mode의 권한이 필요한 서비스를 받기 위한 방법이 필요!
        - 네트워크 서비스를 사용하는 모듈의 함수를 실행해야 한다는 것이다. 그것이 system call이다.

시스템 콜

- User mode에서 kernel mode로 진입하기 위한 통로
- 커널에서 제공하는 protected 서비스를 이용하기 위하여 필요

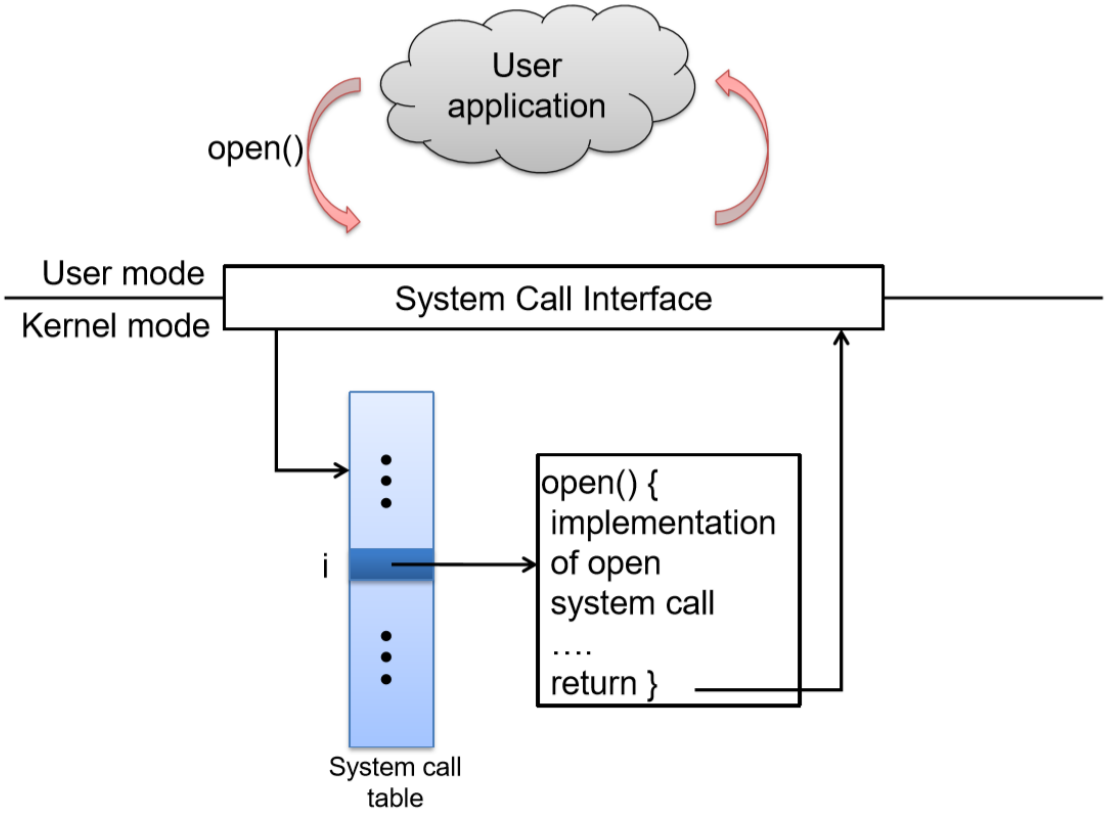
  user mode에서 유튜브 같은게 돌아가는데, 네트워크는 커널 모드에서 돌아간다. 그래서 시스템 콜이 필요한 것이다.

  - Open(2) : a file or device
  - Write(2) : to file or device
  - Msgsnd(2) : send a message
  - Shm(2) : attach shared memory
  - read(2)라는 시스템 콜을 통해서 이 NIC 패킷을 받아서 화면에 띄워준다.

notation

- (1) 이것은 명령임 ls(1)
- (2)이것은 시스템 콜이다.
- (3) 라이브러리 libc(3)

System Call



User application이 open이라는 함수를 호출한다.

그런데 이 함수는 사용자가 정의 하는 것이 아닌, system call이 정의하는 함수이다.

open()은 System Call Interface안에 들어간다.

System Call Interface은 open이라는 콜의 번호를 보고, 그 번호가 i, index가 되어서,

System call table을 lookup을 해서 open system call을 호출하는 것이다.

이 시스템 콜은 return할 때까지 기다린다.

System Call Interface의 레벨에서 라이브러리가 있다.

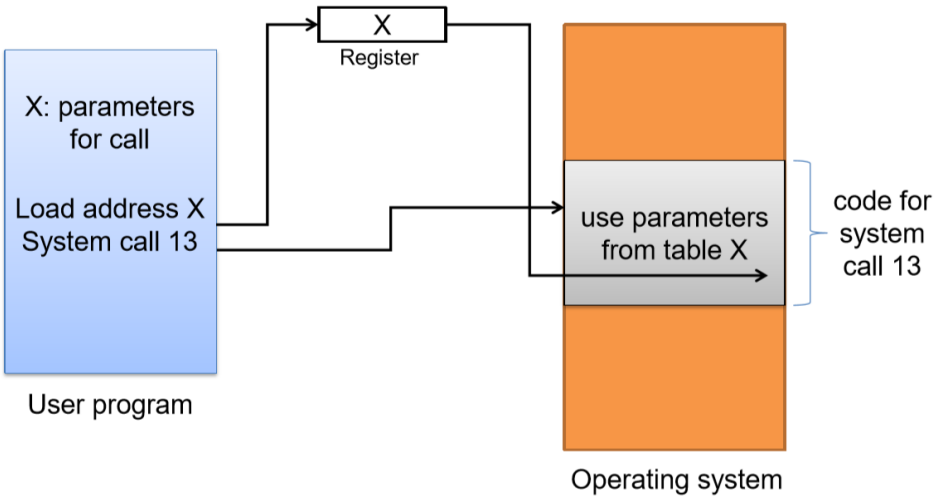
system call이라는 시스템을 이용해 user mode에서 kernel모드로 접근이 가능한 것이다!

System call: argument

System call로 들어갈 때, 어떻게 되는가?

예를 들어서 시스템 콜 13번을하면 open("13")이다. 여기서의 "13"은 argument이다.

이것이 register에 넘어간다. 어떤 것은 메모리로 넘어간다. 그런데, 해당하는 메모리의 값을 바꿔줌으로써 장난을 칠 수 있다. 이 값을 다른 index로 넘어가겠끔 만들 수가 있었다.



System call 예

EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

	Windows	Unix
Process Control	CreateProcess()	fork()
	ExitProcess()	exit()
	WaitForSingleObject()	wait()
File Management	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()
	InitializeSecurityDescriptor()	umask()
	SetSecurityDescriptorGroup()	chown()

API관점에서 동일하더라

리눅스가 Unix를 상속했다.

윈도우가 이름이 더 길다. 오른쪽은 짧다.

그리고 거의 1대1 맵핑이 된다.

커널은 내부는 비슷하다.

시스템 콜이 유사하다.

OS의 차이를 만드는 것은 OS의 System call이 뭐냐라는 것이다.

만약에 함수 이름이 같으면? 유닉스의 설계 철학은 시스템 콜의 숫자를 줄이자는 것이었다.

윈도우에서는 의미를 주어서 만들었다.

app들이 하드웨어에 접근하려고 할때 사용하는 것이다.

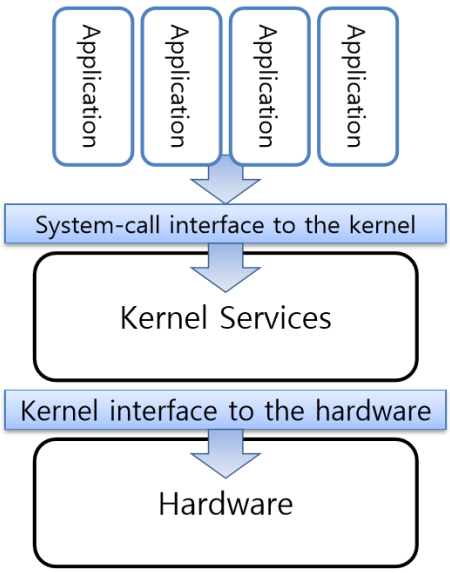
그것으로 파일을 오픈/쓰기도 하는 것이다.

framework을 사용하면 System call을 안쓴다.

framework은 extra layer를 만들어서 사람들이 단순히 이걸 호출하면 돼! 하는 것이다.

다만 System call을 직접 호출하면 더 효율적인 코딩이 가능하다.

일반적인 커널의 구조



APP이 system call interface를 사용하여 커널에 system call을 보내고,

커널은 네트워크를 읽어온다.

밑의 인터페이스는 driver interface이다.

윈도우에서는 HW Abstract Layer(HAL)이라고 부른다.

하드웨어 회사에서 HAL까지 만들어준다.

이 드라이버들로 임의의 하드웨어가 붙을 수 있게 해준다.

Kernel Architecture

커널은 크게 3가지로 나뉜다.

- Monolithic kernel
- Microkernel
- Hypervisor
  - Xen
  - KVM 과 Container

Monolithic kernel

하나는 Mono, 하나는 lithic이다. 무슨 뜻인가?

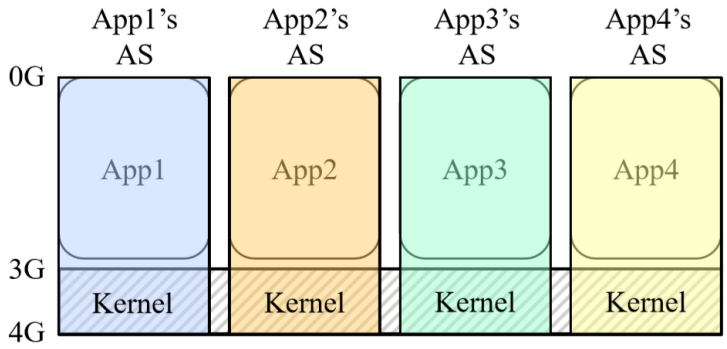
Mono → 하나, lithic → 덩어리

커널과 APP이 하나의 덩어리라는 것이다.

커널과 APP의 주소공간이 한 공간에 있다.

주소공간은 커널과 APP이 나누어 사용한다.

- 특징
  - 커널이 사용자(APP)와 같은 주소 공간에 위치
  - 주소 공간을 커널 코드와 사용자가 나누어서 사용
  - 커널 코드는 한 “덩어리 " 로 구성됨
    - 커널은 시스템 콜을 통하여 접근



Monolithic 커널에서의 주소 공간 매핑
(X86-32bit, ARM-32bit)

그림의 주소공간이 메모리를 의미

지금은 주소공간이 메모리라고 생각해라. (나중되면 달라진다.)

APP이 0~3G를 커널이 3~4G를 차지한다.

APP이 0~3G에 들어가는 것이 얼마나 작은지 큰지를 알아오라

2^32bit → 주소공간이 2^32bit(4G)이다. 라는 의미이다.

유튜브같은 것도 0~3G를 유튜브가 차지, 나머지 3~4G는 커널이 차지.

커널들은 각각이 같을 뿐더러, 물리적으로도 한 copy만을 사용한다.

APP1, APP2, APP3은 다른 주소 공간에 있다.

커널은 다른 주소공간이지만, 하나의 덩어리로 작동한다.

커널이 다른 주소공간에 있는데, 덩어리로 작동한다는 것에 대해서 다시한번 설명해주실 수 있으실까요 ㅠㅠ

→ 커널이 다른 주소공간에 있는데 하나로 동작한다.

→ 창이 여러개있다.

→ 버추얼 메모릴 사용해서 메모리를 같은 주소로 맵핑시킴 물리메모리를 (맵핑을 사용한다. 커널이 다른 주소에 있는데, 하나로 동작하는 것은 맵핑을 사용하기 때문이다.)

Monolithic kernel 장단점

- 장점
  - 어플리케이션과 모든 커널 서비스가 같은 주소 공간에 위치하기 때문에, 시스템 콜 및 커널 서비스 간의 데이터 전달 시에 오버헤드가 적음(다시 말해 빠르다.)
- 단점
  - 모든 서비스 모듈이 하나의 바이너리로 이루어져 있기 때문에 일부분의 수정이 전체에 영향을 미침
  - 각 모듈이 유기적으로 연결되어 있기 때문에 커널 크기가 커질수록 유지 보수가 어려움
  - 한 모듈의 버그가 시스템 전체에 영향을 끼침

    뭐하나 고치면 다 영향을 받기 때문에 디버깅이 어렵다.

    윈도우를 만드는데 있어서 커널개발자의 수가 x, 커널 테스터가 y라고하면 뭐가 더 클까?

    테스트하는 사람이 훨씬 많다.

    뭐하나를 건들이면 모든 것을 다 테스트 해야한다.

    커널 자체만 2천만 라인 된다.

    그리고 해킹까지 고려해야 한다.

    그렇기에 테스팅의 난이도가 더 올라간다.

Microkernel

Monolithic의 단점을 고치기 위해서 연구를 한 결과물이다.

사실 안쓰지만 아이디어가 재미있고, 클라우딩 시스템을 보더라도 참고할 것이 많다.

- 특징
  - 커널 서비스를 기능에 따라 모듈화 하여 각각 독립된 주소 공간에서 실행

    커널이 크니까 모듈화를 하다.

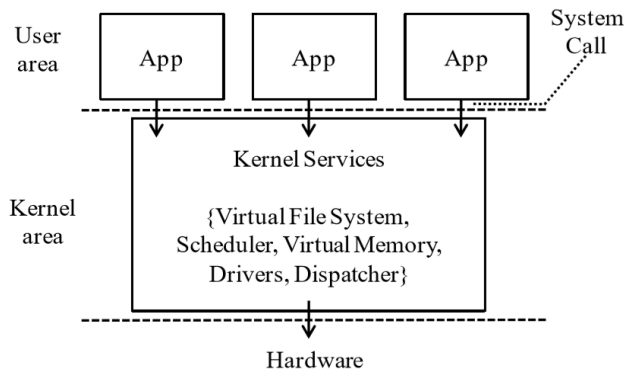
- 이러한 모듈을 서버라 하며, 서버들은 독립된 프로세스로 구현

  그리고 각각 독립된 주소공간에서 돌리자, 이것을 그리고 모듈은 커널 서버라고 한다.

- 마이크로 커널은 서버들 간의 통신(IPC), 어플리케이션의 서비스 콜 전달과 같은 단순한 기능만을 제공

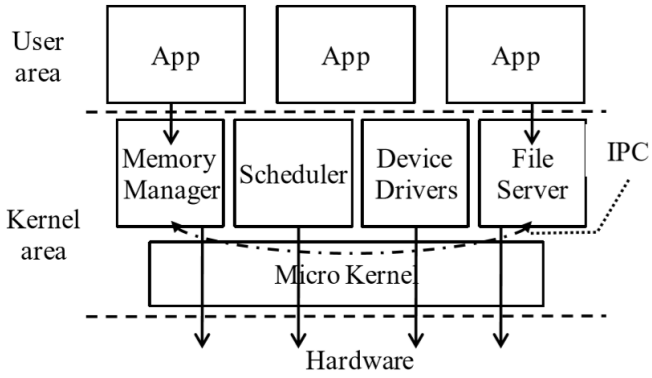
  서버들 간의 통신, 어플리케이션의 서비스 콜 전달이 두가지를 한다.

Monolithic kernel과 Microkernel



< Monolithic Kernel >

하나의 커널위에서 돌아간다.



< Micro Kernel >

모든 것들이 독립적으로 돌아간다.

그리고 이것들이 microkernel로 통신을 한다.

monolithic 커널에서는 device driver가 커널에 있다.

Microkernel 장단점

- 장점
  - 각 커널 서비스 서버가 따로 구현되어 있기 때문에 서로 간의 의존성이 낮음
    - monolithic 커널 보다 독립적인 개발이 가능
    - 커널의 개발 및 유지 보수가 상대적으로 용이

      심지어 각각 따로따로 부팅이 가능하다. → 특정 device driver만을 내릴 수 있었다.

  - 커널 서비스 서버의 간단한 시작/종료 가능
    - 불필요한 서비스의 서버는 종료
      - 많은 메모리 및 CPU utilization확보 가능
  - 이론적으로 micro 커널이 monolithic보다 안정적
    - 문제 있는 서비스는 서버를 재시작하여 해결

      이론상으로는 더 안정적임

      memory에서 fault가 나도 os전체를 죽이지 않아도 된다.
  - 서버 코드가 protected memory에서 실행되므로 검증된 S/W 분야에 적합함
    - 임베디드 로봇 산업, 의료 컴퓨터 분야

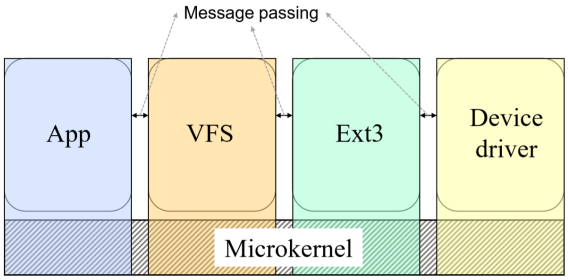
      컴퓨터가 오동작하지 않아야 하는 곳들에 더 적합
- 단점
  - Monolithic 커널보다 낮은 성능을 보임

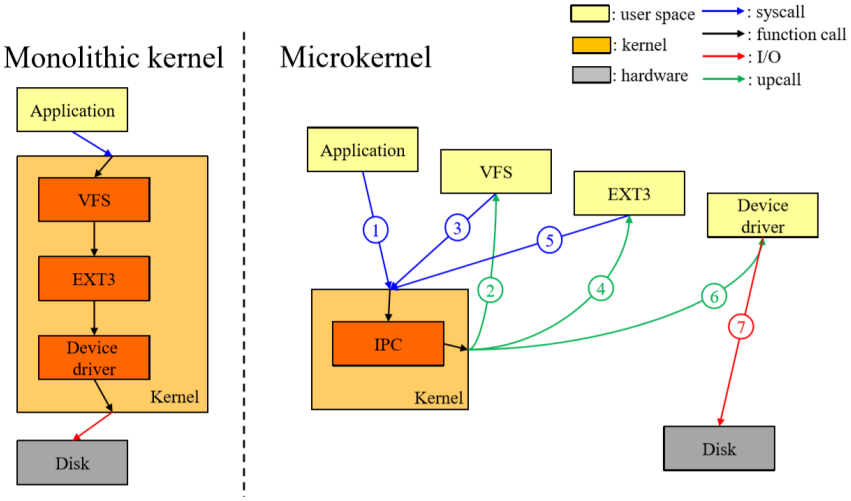
    성능이 많이 낮다.

- 독립된 서버들 간의 통신(IPC) 및 Context switching 때문에 성능이 낮아진다.
- L4 마이크로커널은 하드웨어를 활용하여 성능 개선

  L4가 성능 개선을 보여줬다. Monolithic 커널에 근접하는 수준이었다.

Message passing of Microkernel





monolithic은 한줄로 그냥 내려간다. (함수호출을 계속하는 것이다.)

Microkernel은 독립된 주소공간으로 돌아가는 것이다.

Message passing은 밀리세컨드 단위이고, 이것이 7번이나 일어나는 반면에

upcall : 커널이 주소공간을 호출하는 것이다. (초록색)

monolithic은 마이크로 세컨 단위로 몇번안쓰인다. upcall도 잘 안쓰인다.

Q. ppt 23페이지에 커널 서비스들이 user space로 묶여있는데 그럼 얘들은 더 이상 커널 시스템에 포함되지 않는건가요?

A. 이들은 커널 시스템에 소속된다, 이들은 범 커널이다.

다만 이것들이 마치 분리되어 APP처럼 동작한다.

이들이 유저스페이스에 있는데, 그리고 이들이 커널 서버에 있는데, 이들이 APP에서 작동하지만, APP과 같은 privilege를 가지면 안된다.

이런 커널 서버들을 두 링의 사이에 들어간다. 그러면 이들이 해킹을 못한다.

옆에 있는 APP이 이 커널서버를 죽이지 못한다. 다른 privilege를 하드웨어적으로 제공한다.

즉, 이들은 APP이 위치하는 fing이 아닌 다른 링 위에 위치한다.

Q. 그러면 microkernel을 사용했을때 사용자의 security가 더 보장되는것인가요?

A. 우선 security가 뭔지를 알아야한다.

답은 yes라고 하기 어렵다.

Attack surface → 공격을 받을 만한 표면(?)

말하기가 너무 어렵다!

일반적인 Attack surface은 공격받을 표면이 얼마나 되느냐를 체크하는 것이다.

monolithic 커널의 Attack surface의 면은 어디인가?

→ kernel과 application과 연결된 부분! 바로 system call 부분이다.

system call interface만큼의 attack surface가 된다.

microkernel 커널의 Attack surface의 면은 어디인가?

그 Attack surface가 늘어난다. system call 부분도 늘어나고, microkernel 커널에서는 호출의 횟수도 더 늘어난다. 그 각각이 전부 Attack surface가 됩니다.

따라서 microkerne은 Attack surface의 관점에서 보면 결코 더 secure 하지도 않다.

I/O도 attack surface에 포함된다. 왜냐하면 범 커널을 crash시킬 수 있기에 그러하다.

Q. 혹시 시스템콜도 암호화(?)가 되어 진행되나요??

A. 가능하기는 하다. 무거워질 뿐이다.

Gvisor? → 마이크로 커널의 일종인 라이브러리 커널을 사용하고, 그리고 system call을 줄인다.

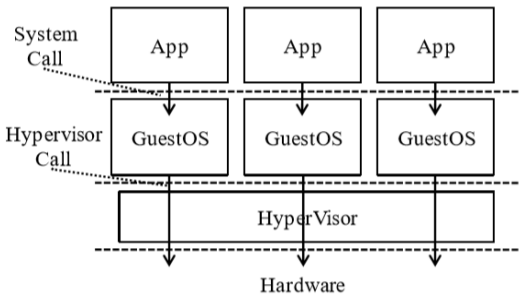
Q. microkernel에서 서비스 서버들은 monolithic에서 app들이 하듯이 각자 일정량의 메모리를 할당받나요

A. yes. fixed된 메모리는 아니다.

Q. 마이크로커널은 느려서 더이상 연구하지 않은건가요? 기술이 발전되어 communication overhead가 줄면 사용될 여지가 있나요?

A. 연구하는 사람들은 있겠지만, 이미 트렌드가 지나가 버렸다. 더 이상 이것이 줄 수 있는 benefit이 없다.

가상화



< Hypervisor>

virtualization

os는 HW를 abstract한다 - cpu/memory/A/N

하드웨어와 OS가 1:1 로 존재

그래서 OS가 죽으면 모든 것이 끝났다.

하드웨어와 OS가 1:1 로 존재 하는 구조를 바꿈!!

Hypervisor라는 계층을 넣어 통해 여러 os를 돌아가게 만들 수 있게 되었다.

Guest OS를 여러개 올리는 것이 가능

기본적으로 버추얼 머신을 사용한다.

- GuestOS는 Linux, Windows 모두 가능
- 하이퍼바이저 계층 도입
- 1960년대부터 존재했었음

  VM을 도입하는 것은 다른 ISA(Instruction Set Atcli - os와 hw사이에 통신하는 명령어 집합)

  ISA가 다르면 돌리기가 굉장히 어려웠다. 그래서 VM를 생각해낸 것이다.

  하드웨어 위에 다른 하드웨어 os를 올리니 에뮬레이션을 해버렸다.

  그래서 굉장히 성능이 떨어졌었더.

- 그러나 성능 저하로 활용되지 못하였음

  유용하나 쓰기는 어렵다는 결론이 나왔음

- Xen 의 등장(2003)으로 computing 의 기반 기술로 자리 매김

  오버해드 가상화의 가장 큰 문제는 오버해드였는데, 이 논문을 기준으로 오버해드를 5%이하로 줄일 수 있게 되었다.

- 클라우드의 enabling technology 5G 는 네트워크 가상화를 도입함

  : computing 을 넘어서 네트워크까지 가상화 적용

  XEN 덕분에 5G의 특징은 네트워크의 망을 virtualization하는 것이다.

Q. 초기 VM이 목표로 한 것이 각 프로그램마다 다른 instruction set을 사용하도록 하는 것인가요??

A. yes.

Q. 현재에도 ISA가 표준화되지 않았기 때문에 가상화를 이용하는 것인가요?

A. no. 요즘은 거의 표준화되었다고 보아야 한다. 현재 남은 것은 intel, arm 두개가 남았다고 본다.

→ 공식은 아니지만, 실질적인 ISA가 남아버렸다.

옛날에는 보드로 머신 코드를 다 썼다.

e2 a1 ac ... 이런식으로 넣고, 퇴근할 때 코드 봅았음.

그러면 초기화됨.

Q. 서로 다른 ISA에서 컴파일된 프로그램을 하나의 하드웨에서 돌아가게하면 어떤 장점이 있나요??

A. 기기를 하나만 사도 된다.

요즘 OS는 가상화를 중심으로 발전하고 있다.

Q. 교수님 그러면 기존 노트북환경(윈도우)에서 버츄얼 머신을 실행시키면 CPU는 어떻게 할당되는지 궁금합니다.

A. 1. VM(guest os)가 cpu 를 할당

2. 그 받은 cpu를 VM이 APP dp gkfekd

Virtualization advantages

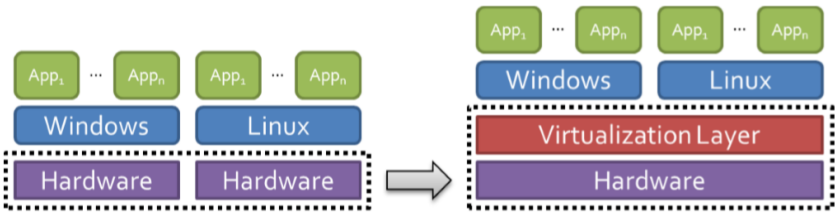
옛날에는 하나의 OS와 HW가 1:1이었다.

data center

→ 어떤 data center에 사람이 많이 몰리면 그곳에 하드웨어를 계속 붙여야 하는 문제가 생겼다.

→ 그런데 가상화를 사용하면 OS와 HW가 분리되어 있기에 seem less하게(APP이 모르게) 하드웨어를 넘기거나 붙이는 것이 가능하다.

- Consolidation
    - allow easier provisioning ← 하드웨어제공, 준비가 쉽다. (데이터 트래픽이 몰릴 것을 대비하는 것이다.) 이것은 가상화 layer 덕분이다.
    - reduce HW cost



Q. 하드웨어가 교환가능하다는 말씀은 하드웨어 "리소스" 가 교환가능하다는 말씀이신가요?

A. yes.

Q. AWS가 자동으로 처리용량을 조절해주는 것도 위와 같은 원리인가요?

A. yes, AWS는 xen을 사용하고 있다

More advantages

차 속의 cpu가 몇 개나 동작할까?

제너시스는 100개정도 들어간다. 고사양일수록 많이들어간다.

일반적인 차도 60~70개정도 들어간다.

이 CPU에 들어가는 코드는 몇 줄 정도 될 것같나?

지금은 천만~2천만 라인을 넘어간다.

windows의 코드는? 얼마전까지 2천만 라인이었다.

→ 차 한대당 윈도우하나정도 돌아가는 것이다.

그런데 CPU가 다 종류가 다르고, ISA도 다 다르다.

그래서 코드를 바꾸지 않고 하드웨어를 바꿔야하는데,

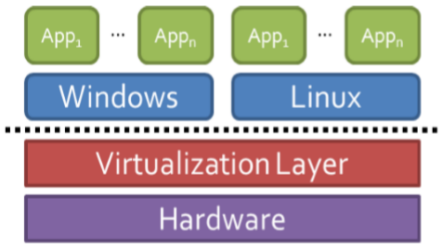
이것이 Decoupling이다.

- Decoupling
  - Eliminate dependency between software and hardware

    → Increase software portability

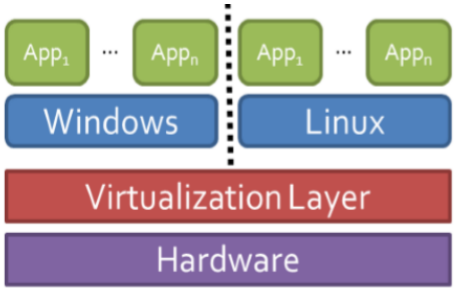
e.g.) Applications can run on new hardware without modification

자동차에서 많이 쓰인다.



- Isolation
  - Guest OS are isolated from each other → Increase system reliability

    e.g.) Device driver failure does not affect other VMs



바이러스가 자동차를 공격했다고 하자. 그래서 윈도우가 멈춘 것이다. 하지만 차는 가야한다. 그래서 하나의 os가 멈춰도 영향을 받지 않고 시스템이 움직이게 하는 것이다.

Hypervisor

- 특징
  - 가상화된 컴퓨터 H/W 자원을 제공하기 위한 관리 계층

    H/W 자원을 abstract한다.
    - 게스트 OS와 H/W 사이에 위치함
      - 게스트 OS – hypervisor가 제공하는 가상화된 H/W 자원을 이용하는 운영체제

- 각 게스트 OS들은 각각 서로 다른 가상 머신(Virtual Machine) 에서 수행되며 서로의 존재를 알지 못함
    - H/W에 대한 접근은 hypervisor에게 할당 받은 자원에 대해서만 수행

- Hypervisor 는 각 게스트 OS간의 CPU, 메모리 등 시스템 자원을 분배하는 등 최소한의 역할을 수행.

→ 이것은 micro kenel은 아니다. 다만 관점이 완전히 다르다. 여러개의 OS를 돌린다는 것이다.

Hypervisor 장단점

- 장점
    - 하나의 물리 컴퓨터에서 여러 종류의 게스트 OS 운용이 가능
        - 한 서버에서 다양한 서비스를 동시에 제공
    - 실제의 컴퓨터가 제공하는 것과 다른 형태의 명령어 집합 구조 (Instruction Set Architecture)를 제공
        - 다른 H/W환경으로 컴파일 된 게스트 OS및 응용프로그램도 실행 가능

- 단점

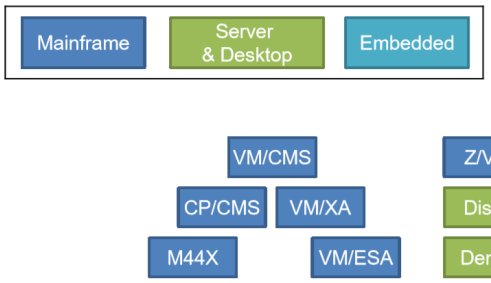
  1) 일단 메모리가 2배소모됨(자원 소모가 2배)
    - H/W를 직접적으로 사용하는 다른 운영체제에 비해 성능이 떨어짐
        - 반가상화 (Para-virtualization)로 성능저하 문제를 해결하려 함

          반가상화? OS와 Hyper-V를 직접 붙이면 애뮬레이션을 해야해서 성능이 떨어지는데, 그래서 코드를 수정을 해서 glue 해야하는 것이다. 그런데 이것을 할 수 있는 사람이 얼마 없다.

            - 단 게스트 OS의 H/W 의존적인 코드에 대한 수정이 요구됨
                - 높은 기술적인 능력이 필요함

- OS의 소스가 공개되지 않았다면 게스트 OS로 수정이 불가능

History of hypervisor



Generation	1st	2nd	3rd	4th
Year	1960~1990	1990~2000	2000~2010	2010~2020
Main technology	• Early virtualization technology	• Supporting trap of sensitive instruction	• Paravirtualizing Intel/ARM processor • Binary conversion • Intel VT HW virtualization	• ARM A15 HW virtualization • Virtualizing I/O device (such as SR-IOV) • Virtualizing GPGPU

2000년대 동안 활용도가 거의 없는 상태로 머물러있다가(AI처럼) 2003년에 para-V가 가능해졌고 컴퓨팅의 핵심기술이 되었다.

Q. 하드웨어 가격도 많이 낮아졌는데 여전히 가상화로 하나의 하드웨어에 여러 개의 os을 동작시키려는 이유가 무엇인가요?

A. 잠실 운동장을 가득채울 정도의 하드웨어들이 있는데, 만일 새로운 윈도우 버젼이 나오면, 거기를 사람들이 다 움직여서 바꿔야하는데, 그럼 그 사람들의 가격때문에 힘들다.

Q. 만약에 이용자들이 입력을 보내면 반드시 윈도우에서 프로그램 A를 돌려서 그 결과만 돌려보내 주는 서버가 있다면, 가상화를 한다면 윈도우만 수십 개를 서버에 띄워야 할 것인데, 이 경우 가상화의 이점이 있나요?

A. 가상화 이전에 이는 웹서버들이 이런 것을 하는데, 이것을 수만개를 돌리는 thread라는 것을 이용한다. (사실 몰라도 된다. 이건 가상화와 관련된 것도 아니다.)

경량화 된 hypervisor

요즘 Xen을 제대로쓸 줄아는 사람은 없다.

그래서 이것을 리눅스에서 돌리자고 함.

→ 이것이 KVM이다.

Q. 교수님, 저희가 사용하는 window에서 linux를 VM으로 작동시키다가 windows를 종료하면 linux도 자동으로 종료되는 걸로 알고있는데 이것은 isolation 특징을 갖지 않는건가요??

A. 이것은 KVM이다.

- KVM
  - 리눅스에 포함된 hypervisor
  - Xen의 paravirtualization을 없앰
  - VT-x (Intel) 하드웨어를 이용한 성능 가속 (느려지니까 가속이 필요함)

  재부팅을 하더라도 host 에는 전혀 영향이 없음

  KVM은 Xen의 복잡성을 줄이기 위해 나옴

  Guest os의 위에 APP이 올라감.

  → 이 게스트 OS끼리는 서로 모르고, 그위의 프로그램은 더욱이 모른다.

  → security가 좋다.

  그러나 무겁다. Memory도 많이 든다.

  Data center는 PC에서는 가볍게, FireWall에는 보안을 담당하기에 더 효율적으로한다.

- 컨테이너
  - Share root filesystem and libraries
  - So lighter than KVM
  - Package all the dependencies into “image”

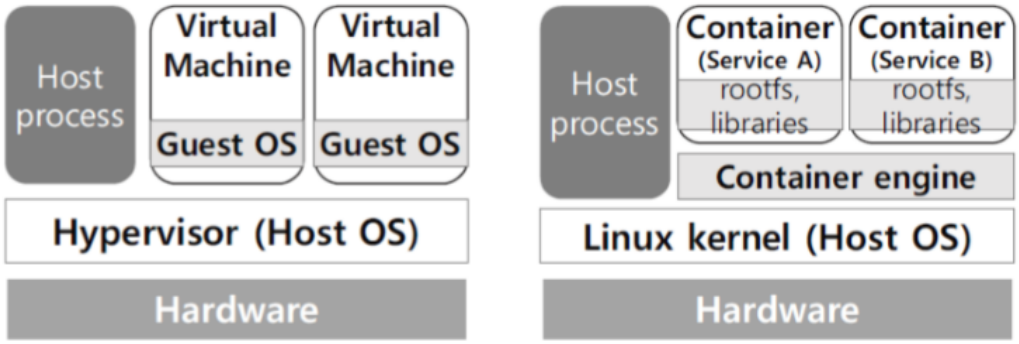
이걸 좀 강조를 하고 싶다.

요즘의 트렌드이다.

서로 볼 수는 있지만 서로의 File system은 따로 사용한다.

컨테이너가 돌아가는데 필요한 모든 것을 image라고 하는데 즉, 베포가 필요한 것이다.

migration이 굉장히 용이해진 것이다.



Q. VT-x 기술이 Intel이 Hypervisor 엔지니어를 위해 새로운 ISA를 지원하는 CPU를 만든 것이라고 생각하는데 혹시 맞을까요?

A. Guest os에 들어왔다 나갔다 하는 것을 빨리해주는 것이다.

state를 save해야 하는데, 이것을 가속함으로써 VT-x의 instruction을 만들었다.

Q. 지금 설명해주시는 컨테이너의 대표적인 것이 docker 가 맞을까요

A. yes

Q. Docker와 같은 컨테이너 기반 프로그램의 경우에는 Docker가 Host OS(Windows, Linux)위에서 실행되는 하나의 container engine이 되는것인가요?

A. yes, container engine = docker

Name Space? 두개의 Name Space가 달라야 서로 모른다. (이것이 isolation의 의미이기도 하다.)

어디에 뭐가 있는지 완전히 모르기에 커뮤니케이션이 안 된다.

KVM두개의 VM이 완전히 별도의 Name Space가 다른 것이다.

컨테이너는 file과 lib을 별도의 name spae로 가지는 것이다.

실제로 저장된 파일은 같다. 그런데 이것을 보는 컨테이너들의 Name space는 다르다.

KVM이 가지고 있는 NS의 isolation이 더 단단하다

컨테이너는 파일 시스템만 isolation시킨다.

KVM은 guest OS가 존재하지 않는다.

PC에서 윈도우가 돌아간다. 그위에서 리눅스를 띄운다.

컨테이너에서는 윈도우 위에 윈도우 프로그램을 띄우는 것이다. 리눅스를 돌리지 않는다.

Q. VT-x 가 새로운 ISA를 제공해주는 것이고 GuestOS의 App이 이를 이용할 수 있으려면 Complier 단에서 지원이 이뤄져야할 것 같은데, 언어 개발자도 이에 맞춰 반영해줘야 하는게 맞나요?

A. yes

previliedge기능을 수행하려면 하드웨어적인 부분을 수행하는 것이 필요하다

Q. 컨테이너의 security는 kvm에 비해 취약한건가요?

A. yes

숙제

자기 노트북의 프로그램의 크기를 찾아와라

자기가 쓴 프록스램 5개, 윈도우 파일의 크기를 알아와라 (DISK에 저장되어 있는 것)

→ 이름과 크기

→ 이것도 물어본다.

숙제 : 리눅스 시스템 콜을 검색해라

→ 설명이 나오고, syscall.h가 나온다. 여기서 보면 몇 번 코드인지가 나온다.

Open, Write, Msgsnd(메시지 send), Shm(shmat)의 인덱스

→ 2, 1, 28

→ shmat은 30번

Shm

%rax	Name	Entry point
29	shmget	sys_shmget

%rax	Name	Entry point
30	shmat	sys_shmat
31	shmctl	sys_shmctl
67	shmdt	

버스가 무엇인가? CPU와 Memory를 연결해주는 것을 bus라고 한다.

Bus는 컴퓨터 내부의 통신을 위한 통로이며

LAN은 한정된 지역 내의 독립된 컴퓨터 간의 소통을 위한 시스템이다.

LAN은 호스트와 호스트 간의 연결을 위한 것이다.

버스는 대역폭을 가지기는 하지만, 대역폭을 의미하지는 않는다.

숙제: 버스와 랜의 속도의 차이가 어떻게 되는가?

1)버스와 랜의 속도를 알아와라

2)대역폭, clock속도는 어떻게 되는가?

3)naming은 어떻게 되는가?, 거기에 붙을 때 어떻게 장치들을 identify한다는 말이냐? 주소가 있다는 것인데, 그것이 identifier이다.

이것은 그냥 제출하자

bus의 네이밍? 메모리와 cpu가 여러개 붙을 수 있고, GPU도 붙을 수 있다. 이것들의주소, 어떻게 identifier하는가? 를 제출해봐라 담주 화욜 수욜전까지 제출해봐라

질문

Q. user area와 user space는 다른건가요?

A. 두개가 같은 것이다.

Q. 마이크로커널이 커널자체의 크기는 줄어들고 유저스페이스의 크기가 늘어나게 만든건가요??

A. 커널이 줄어든 것은 맞고, 유저스페이스가 늘어가는 것은 아니다.

Q. ppt 23페이지에 커널 서비스들이 user space로 묶여있는데 그럼 얘들은 더 이상 커널 시스템에 포함되지 않는건가요?

A.

Q. 그렇다면 마이크로 커널은 역할마다 각기 다른 제조사의 제품을 사용 할 수 있도록 표준이 있나요?

A. NO 표준이 없다. 표준을 만들정도로 발전하지 못했다. 이 커널이 한번도 제품화된적이 없었다.

Q. 만약 app의 크기가 3gb를 넘으면 커널도 3:1의 비율로 커지나요??

A. 아니다. APP은 3GB를 넘어갈 수 없다. 3GB자체도 큰 것이다. 넘어가면 커널이 멈춰버린다.

Q. 커널 1G안에 어떤 것이 있나요? 저번 시간에 system call table에 대해 배웠는데 이 table이 들어있는 건가요...?

A. 모든 data structure가 다 들어있다.

Q. 그럼 각각 다른 application이라고 할 지라도 application마다 커널의 크기는 똑같나요?

A. 같다.

Q. system-call interface와 kernel interface도 저 한덩어리에 포함되어 있나요?

A. interface는 그저 instruction에 불과하다.

Q. 하드웨어 만드는 회사에서 운영체제를 만드는데도 유리했을 거 같은데 그렇게 하지 않았던 이유가 있나요?

A. 애플은 그렇게 한다. 생각이 나뉘었다. 전부 한덩어리로 만들자는 것이 apple, 하드웨어와 os를 분리하여 가자고 생각한다. IBM은 특수했다. 윈도우를 사서 그저 IBM PC에 올렸으면 자기들이 커졌을 것이다.

Q. 메모리의 주소가 고정적이라면, 하나의 app이 실제로 아무리 작은 공간을 차지하더라도, 3G가 할당되어있나요

A. 모든 커널이 monolithic하다. 그 이유는 overhead가 적은 것때문에 그렇다. 어플리케이션과 모든 커널 서비스가 같은 주소 공간에 위치하기 때문에,

시스템 콜에서의 오버해드가 적다. (같은)

물리메모리는 no, 가상메모리는 yes

Q. multiprogramming과 multitasking의 차이점은 무엇인가요?

A.

Q. 그럼 multitasking이 multiprogramming을 포함하는 개념이라고 봐도 되나요??

A. 뭉뜽그려서 문맥에 따라서 생각해라 multitasking을 그냥 일반사용자들에게 편하게 쓰는 용어이다.

Q. multiprogramming은 일단 실행중이면 user가 관여할 수 없는 것 아닌가요? multitasking은 그런 개념은 아닌 걸로 이해했었습니다

A. 관여할 수 없다. 관여할 수 있다면 timesharing이다.

독립적이라는 말은 layer의 교체가 용이하다는 말이다.

Q. 운영체제의 표준이 있는가?

A. 표준으로 나오는 것들이 있다. 프로토콜처럼 딱뿌러지게 나오지는 않는다. 그치만 있기는 하다.

Q. system call 에서 이상이 생겨 멈춰 return을 하지 못하면 user application들은 계속 멈춰있는건가요?

A. yes그렇다. 그렇지만 이렇게 죽는다면 그것은 OS로써의 자격이 없다.

Q. LAN이 Multi processing Bus가 parallel processing과 유사한 개념인가요?

A. 최소한 최근까지는 LAN으로 안하고 BUS로 했다.

→ 그런데 요즘은 LAN 속도가 BUS에 근접하고 있다.

parallel processing는 LAN을 사용하려고 한다. 그 이유는 LAN이 빨라지고 있기 때문이다.

Q. 정상적인 경로로 악의적인 시스템 콜을 함으로써 해킹이 가능한 경우도 있나요? 그렇다면 그것을 어떻게 판단하고 막을 수 있나요?

A.

Q. system call이 os간의 차이를 낸다고 하셨던 부분을 조금 더 자세히 설명해 주실 수 있을까요? 단순히 함수명이나 인자의 차이정도인지, 아니면 로직이 전반적으로 달라지는지 궁금합니다.

A. 둘다 yes. 단순히 함수명이나 인자가 달라지는 것도 차이를 만들고, 로직이 전반적으로 달라지는 것도 차이를 만든다.

리눅스에서 돌아가는 시스템 콜은 윈도우에서는 아예 컴파일이 안된다.