3장 임베디드 운영체제

한양대 최진식

출처: 숭실대학교 정 규식

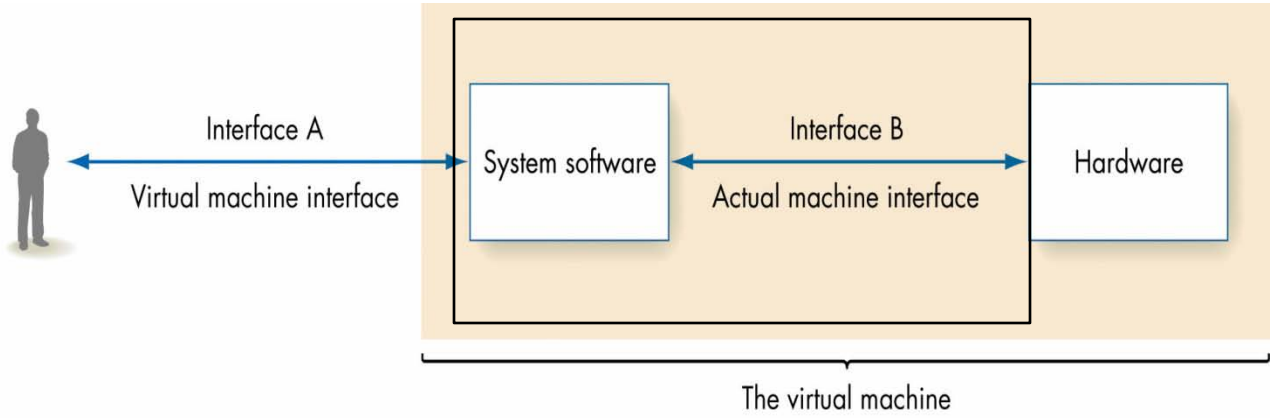
http://openwrt.ssu.ac.kr

ref: 유무선공유기를 이용한 임베디드 리눅스 시스템 구축 및 응용

목차

운영체제(Operating System) 개념(1/7)

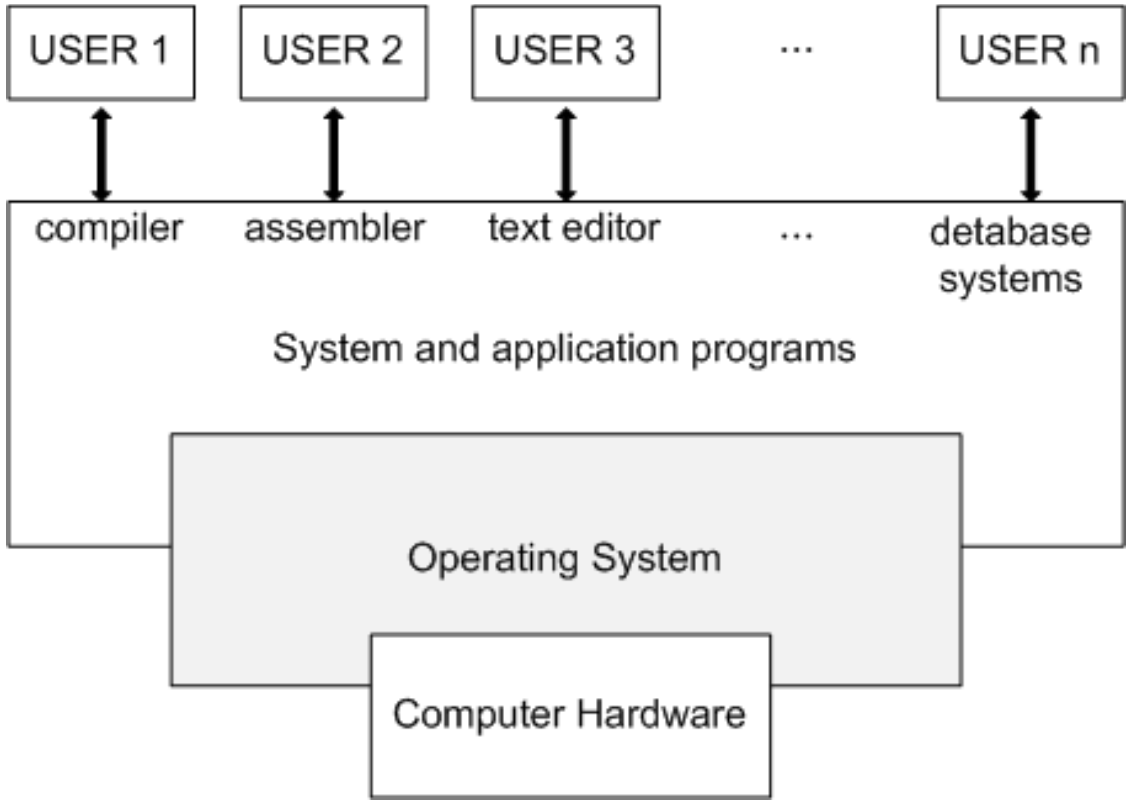
- 운영체제 ?
  - 사용자와 시스템과의 대화를 위한 인터페이스를 제공
    - 응용프로그램들이 시스템 하드웨어들로 접근하여 사용가능 하도록 인터페이스를 제공하며,
  - 컴퓨터의 하드웨어를 직접적으로 제어하고 관리
  - 시스템이 가지고 있는 자원을 최대로 활용



운영체제는 저수준의 프로그래밍 인터페이스를 제공해주는 물리적인 기계위에서 고수준의 프로그래밍 인터페이스 및 사용자 인터페이스를 제공해주는 가상 기계**(virtual machine)**
다양한 하드웨어 위에서 추상화된 인터페이스를 제공해줌으로써, 개발자가 복잡한 주변장치들을 제어하는 어려움을 없애줌

운영체제(Operating System) 개념 (2/7)

- 운영체제의 구성도
  - 컴퓨터의 여러 하드웨어들을 논리적으로 제어하며 응용프로그램과 컴퓨터 하드웨어를 연결



운영체제(Operating System) 개념 (3/7)

- 일반적인 운영체제의 목적
  - 응용 프로그램들이 실행될 수 있는 실행 환경을 제공하며, 파일시스템 등의 서비스를 제공한다.
  - 실행 응용 프로그램들끼리 서로 보호하여 한 프로세스가 다른 프로세스를 침범할 수 없게 한다.
  - 시스템 하드웨어 자원을 최대한 공정하게 각 프로세스에 할당하여 전체 시스템이 효율적으로 운용되도록 한다.
- 커널(kernel)
  - 운영체제의 여러 구성 요소중 가장 기본적인 역할과 핵심 기능을 하는 부분
  - 하드웨어와 운영체제의 다른 부분 사이에서 중재자
  - 기본 역할
    - 자원 관리 : 스케줄링을 통하여 한정된 시스템 자원을 효율적으로 관리하여 프로그램의 실행을 원활하게 한다.
    - 추상화 : 운영 체제의 복잡한 내부를 사용자가 신경 쓰지 않고 일관성 있는 인터페이스를 하드웨어에 제공하기 위해 하드웨어 추상화 계층(HAL)을 형성
    - 보안 : 컴퓨터 하드웨어와 프로세스의 보안

운영체제(Operating System) 개념 (4/7)

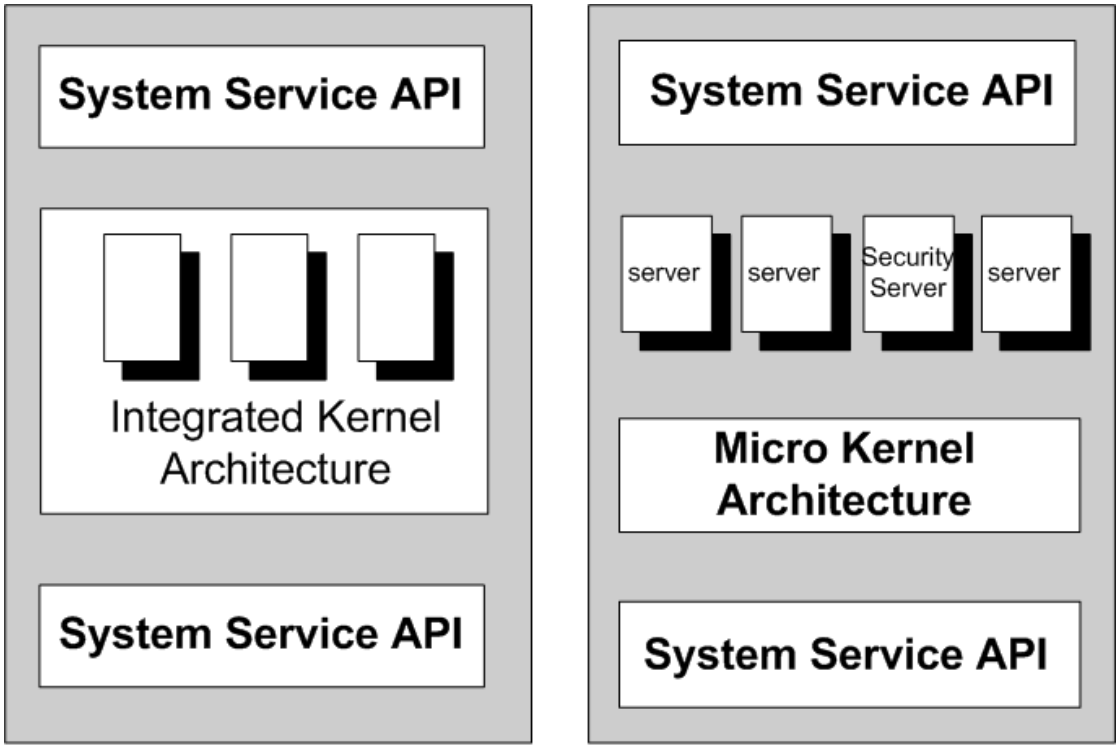
- 커널 아키텍쳐 구현 방식
  - 단일형 커널(Monolithic Kernel)
    - 운영체제가 제공하는 모든 서비스들을 하나의 커널이 통합하여 제공하는 구조
    - 사용자 응용 프로그램과 함께 동작
    - OS 개발단계에서 코드의 집적도가 매우 높은 반면 수정하기 어려움
    - 한 모듈의 문제점은 전체 시스템의 성능에 영향을 주게 되며 유지보수가 힘듦.
    - 운영체제가 완벽하게 구현이 되었다면 구성 요소의 내부 집적이 효율적으로 내부 시스템을 이용 가능하게 해주기 때문에 우수한 성능을 냄.
    - 유닉스, BSD, 리눅스, 솔라리스, 윈도우즈 9X, AIX 등이 있다.

운영체제(Operating System) 개념 (5/7)

- 마이크로 커널(Micro Kernel)
  - 필수 커널 루틴으로만 코어 커널 구성
    - 자원 관리
    - 스케줄링
  - 더 많은 기능은 서버(커널의 컴포넌트 서버)라고 불리는 응용 소프트웨어(사용자 모드)를 통해 제공
    - 모듈 형태로 응용 적응적으로 다양한 구성이 가능
    - 여러 운영체제 서비스들이 서로 통신을 할 경우 성능 저하
  - AmigaOS, Phoenix-RTOS, QNX, Symbian OS 등

운영체제(Operating System) 개념 (6/7)

- 단일형 커널 **vs** 마이크로 커널 비교



단일형 커널 마이크로 커널

운영체제(Operating System) 개념 (7/7)

- 운영체제 종류
  - 메인프레임 운영체제 : **OS/360, OS/390**
  - 서버 운영체제 : **Unix, Window 2000, Linux**
  - 멀티프로세서 운영체제
  - **PC** 운영체제 : **Window 98, Window 2000, Linux**
  - 실시간 운영체제 : **Vxworks, QNX**
  - 임베디드 운영체제 : **PalmOS, Windows CE, LINUX**
  - 스마트카드 운영체제 : **Interpreter for Java Virtual Machine**
  - 웹 운영체제 : 구글 크롬**(Google Chrome)**

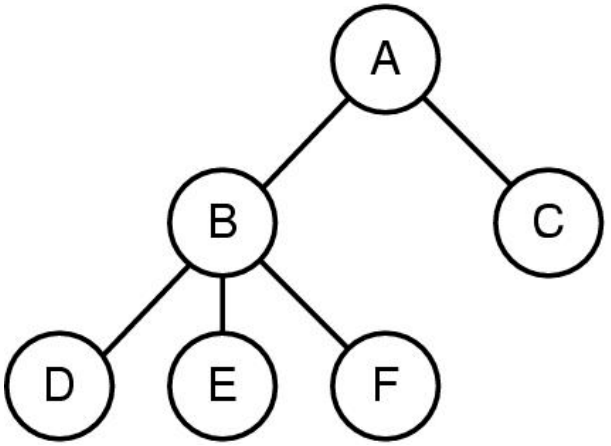
OS overview[1)](1/14) - OS Major Components

- **Process and thread management**
- **Resource management**
  - **CPU**
  - **Memory**
  - **Device**
- **File system**
- **communications**
- **Bootstrapping**

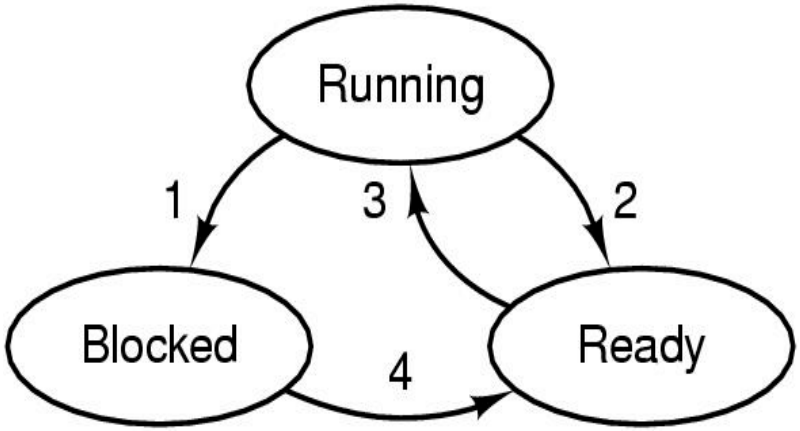
1) Adapted from lecture notes of Prof. A.S. Tanenbaum and Prof. Y. Zhou

OS overview(2/14) - Process Review

- What is a process?
  - A running program
- What does a process include
  - Address space
  - Process table entries (state, registers)
- How to create/kill a process
  - Fork()
  - Execve()
  - Kill()
  - Exit()



OS overview(3/14) - Process States



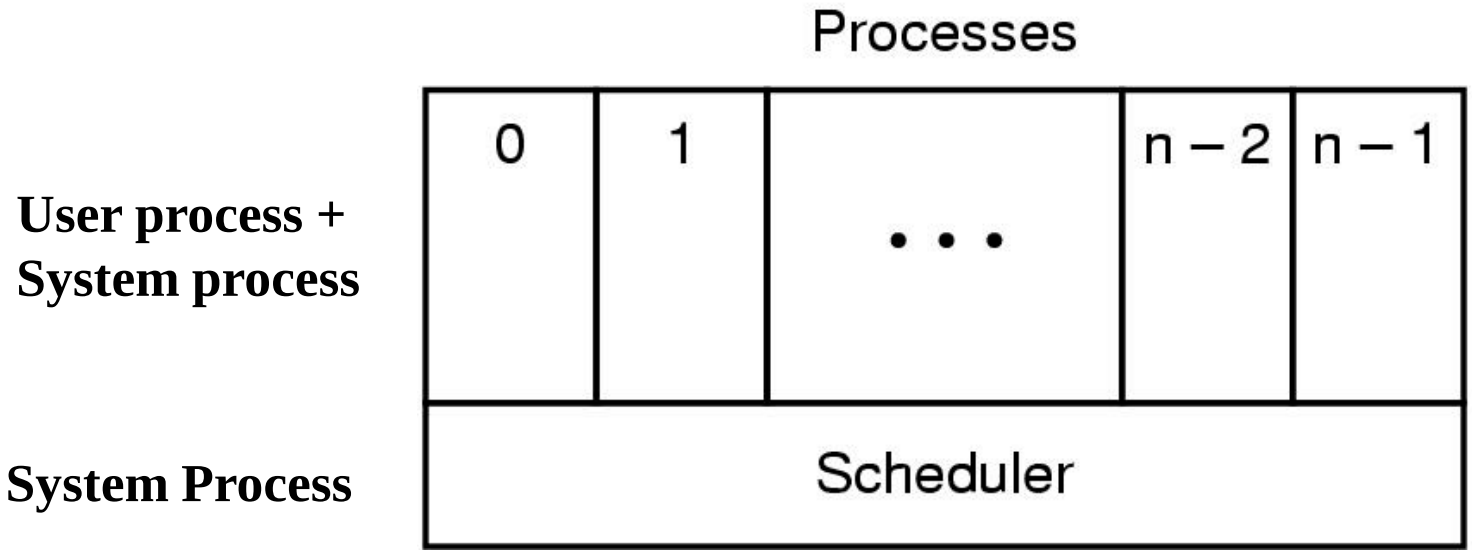
1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

process run)

- Other states**: suspended, terminated**

- **Transitions between states shown**

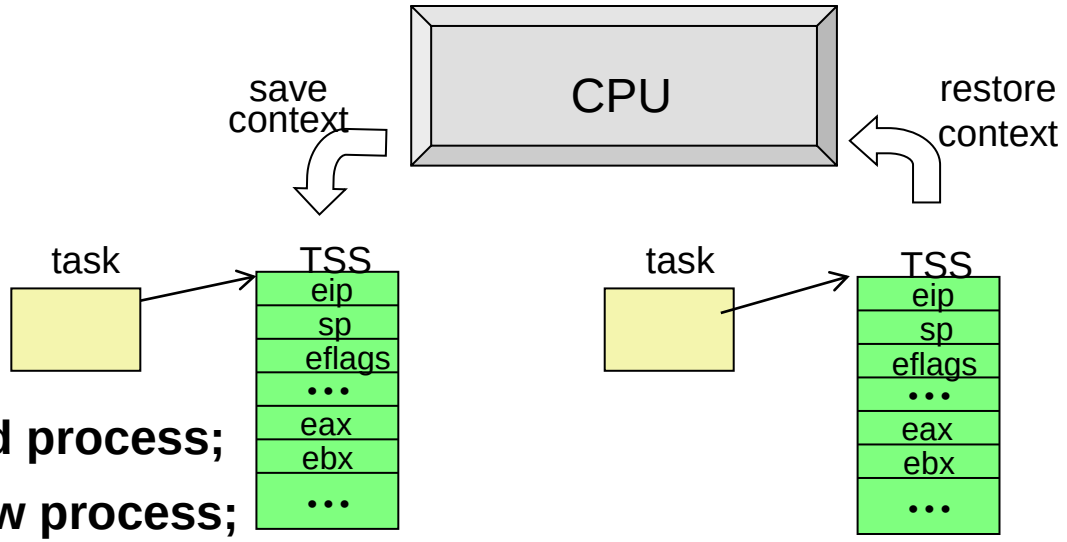
OS overview(4/14) - Process States



- Lowest layer of process-structured OS
  - handles interrupts, scheduling
- Above that layer are sequential processes

OS overview(5/14) - Context Switch

- Switch CPU from one process to another
- Performed by scheduler



- It includes:
  - save PCB state of the old process;
  - load PCB state of the new process;
  - Flush memory cache;
  - Change memory mapping (TLB);
- Context switch is expensive(1-1000 microseconds)
  - No useful work is done (pure overhead)
  - Can become a bottleneck
- Need hardware support.

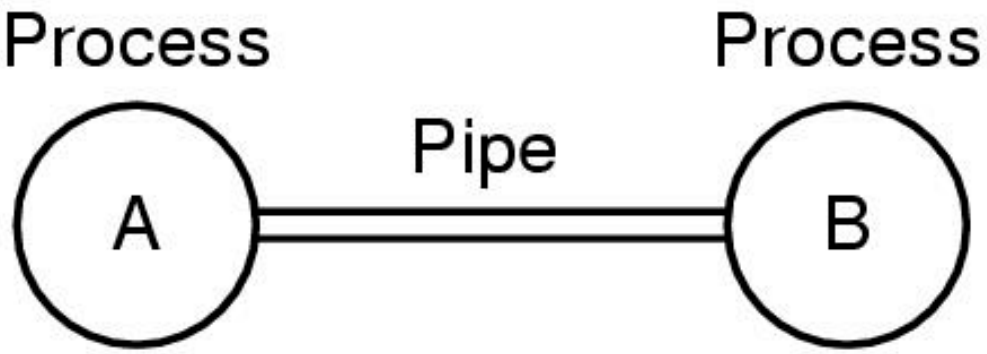
OS overview(6/14) - Process Synchronization

- **Why do we need to synchronize?**
- **How to synchronize?**

OS overview(7/14) - Process communication

IPC = 프로세스간 자원을 공유 (resource sharing)

- -공유 메모리**(Shared memory)**
- -세마포어**(Semaphore)**
- -시그널**(Signal)**
- -파이프**(Pipe)**
- -메시지 큐**(Message queue)**



Two processes connected by a pipe

-

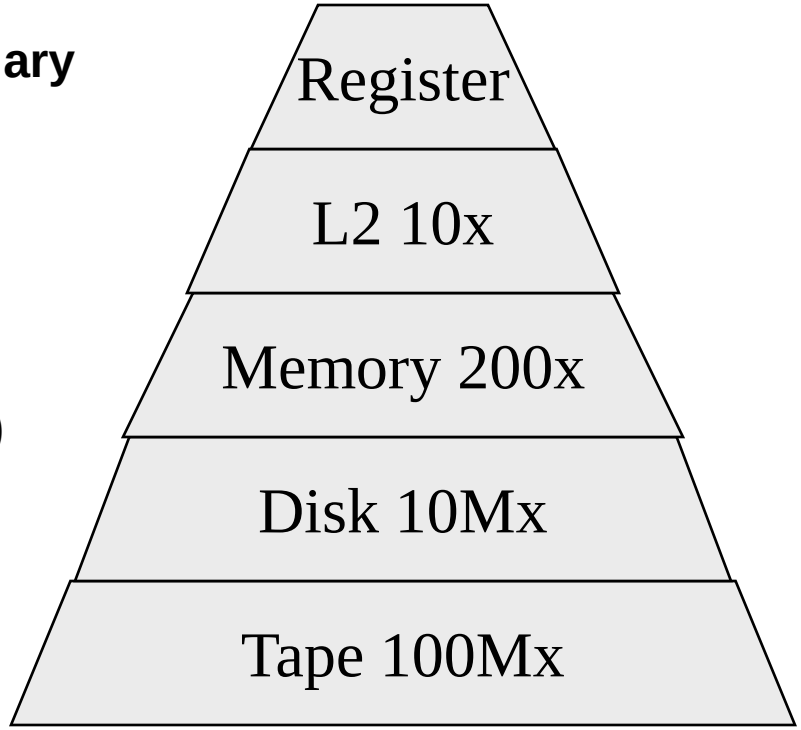
OS overview(8/14) - Processor (CPU) Management

- **Goals**
  - **Time sharing**
  - **Multiple CPU allocations**
- **Issues**
  - **Do not waste CPU resources**
  - **Synchronization and mutual exclusion**
  - **Fairness**
  - **deadlock free**

Analogy: Video Games

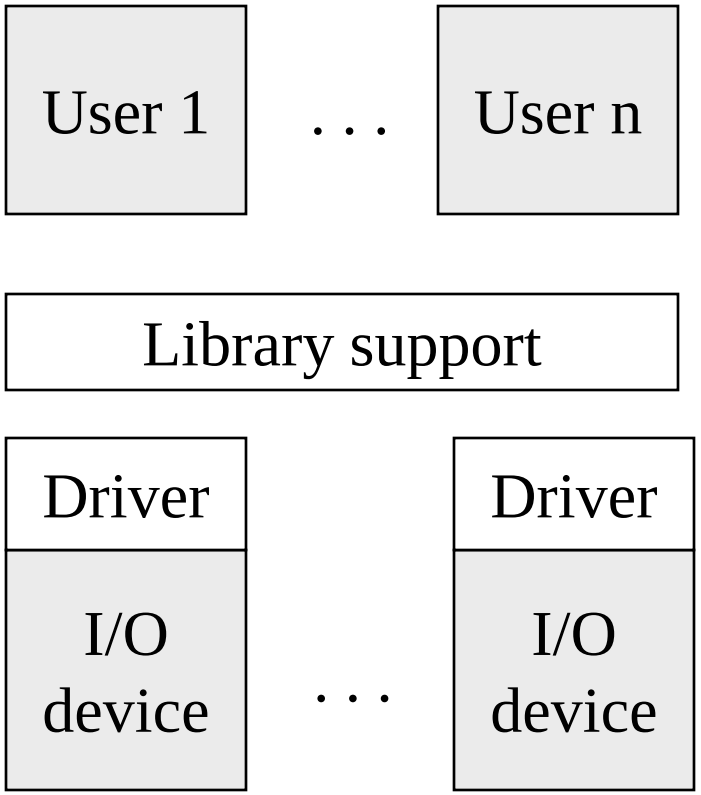
OS overview(9/14) - Memory Management

- **Goals**
  - **Support programs to run**
  - **Allocation and management**
  - **Transfers from and to secondary storage**
- **Functions**
  - 가상 메모리 시스템
  - 메모리 보호**(Protection)**
  - 메모리 매핑**(Memory mapping)**
- **Issues**
  - **Efficiency & convenience**
  - **Fairness**
  - **Protection**

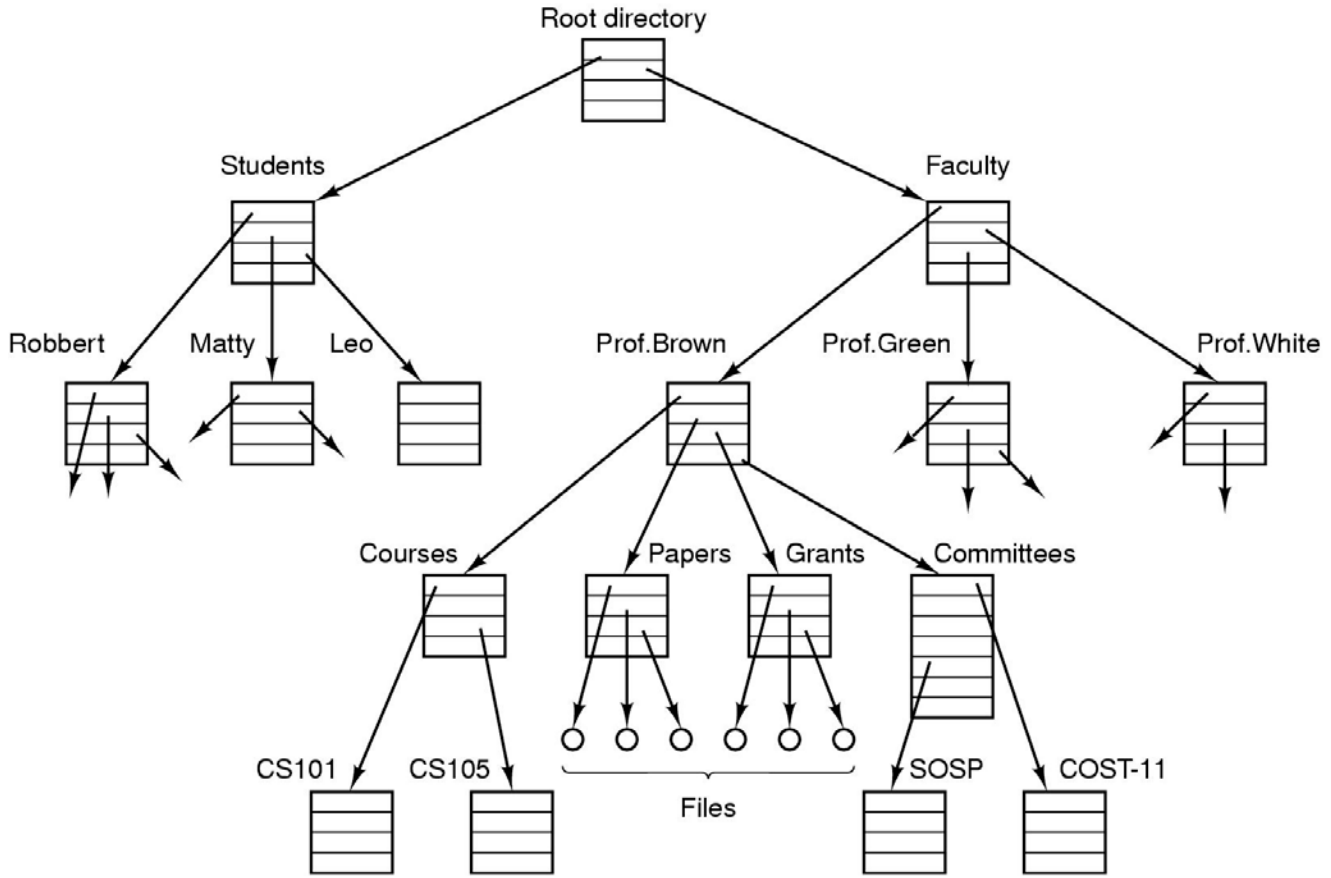


OS overview(10/14) - I/O Device Management

- **Goals**
  - **Interactions between devices and applications**
  - **Ability to plug in new devices**
- **Issues**
  - **Efficiency**
  - **Fairness**
  - **Protection and sharing**

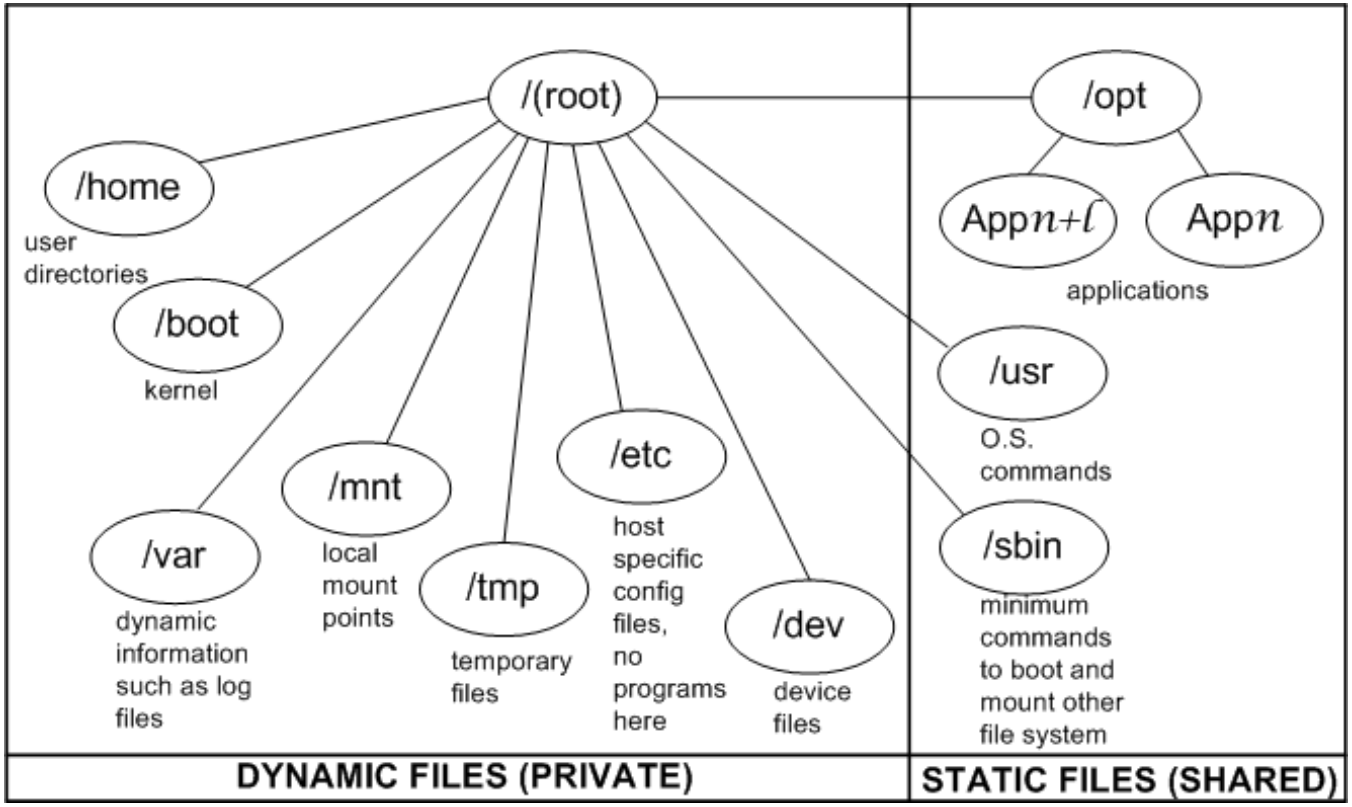


OS overview(11/14) - File System Example



root file system(11/14)

- **Root File System**
  - **root** 디렉토리(**/**)로 마운트 되는 파일 시스템으로 리눅스가 동작하기 위해 기본적으로 필요한 프로그램 및 설정 파일을 가지고 있다**.**
  - **Kernel**에서 초기화 작업 후 임시로 **RAM disk**를 **Root File System**으로 마운트하여 필요한 초기 작업을 수행하고 그 다음에 **disk**에 있는 실제 **root file system**을 마운트 함



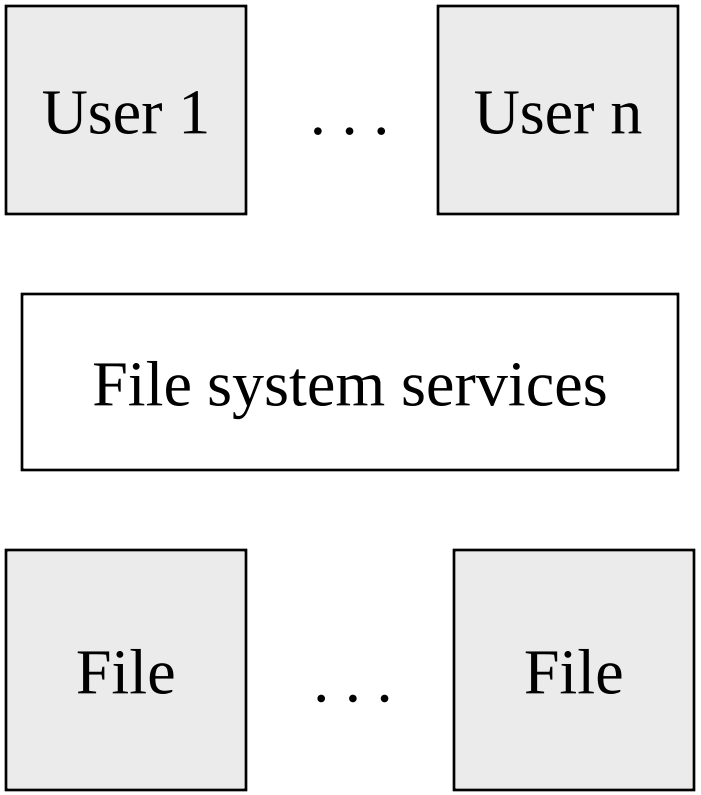
리눅스 파일 시스템 구조(12/14)

- 최상위 디렉토리 구조 설명

디렉토리 명	설 명
/	루트 디렉토리
/bin	가장 필수적인 실행명령
/boot	커널, LILO 등 부팅에 관련된 파일
/dev	장치파일 모음
/etc	시스템 전체 설정파일
/home	사용자의 홈디렉토리
/lib	C 라이브러리 등 가장 필수적인 공유 라이브러리
/mnt	임시 마운트용 디렉토리
/proc	시스템 정보를 위한 가상적인 디렉토리
/root	루트 사용자의 홈 디렉토리
/sbin	시스템 관리용 실행파일
/tmp	임시 파일 생성용 디렉토리
/usr	대부분 어플리케이션이 설치되는 디렉토리
/var	시스템 운영 과정에서 생성되는 각종 임시 파일

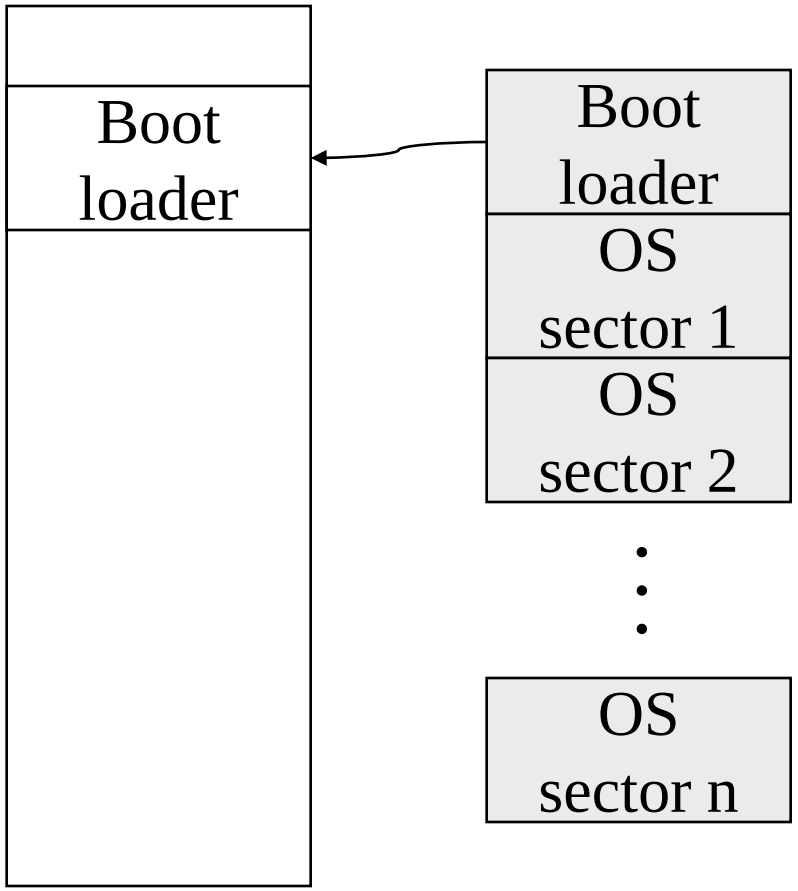
OS overview(12/14) - File System

- **A typical file system**
  - **Open a file with authentication**
  - **Read/write data in files**
  - **Close a file**



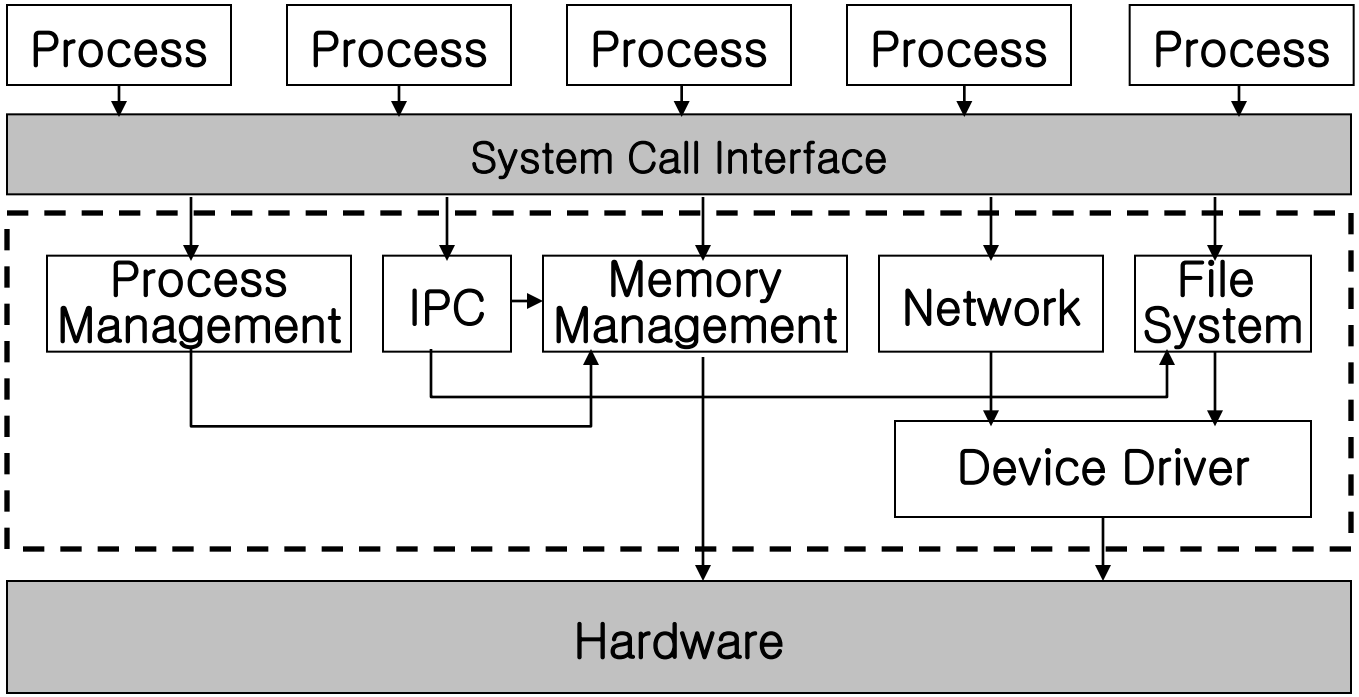
OS overview(13/14) - Bootstrapping

- **Power up a computer**
- **Processor reset**
  - **Set to known state**
  - **Jump to ROM code**
- **Load in the boot loader from stable storage**
- **Jump to the boot loader**
- **Load the rest of the operating system**
- **Initialize and run**



OS overview(14/14) - System Call

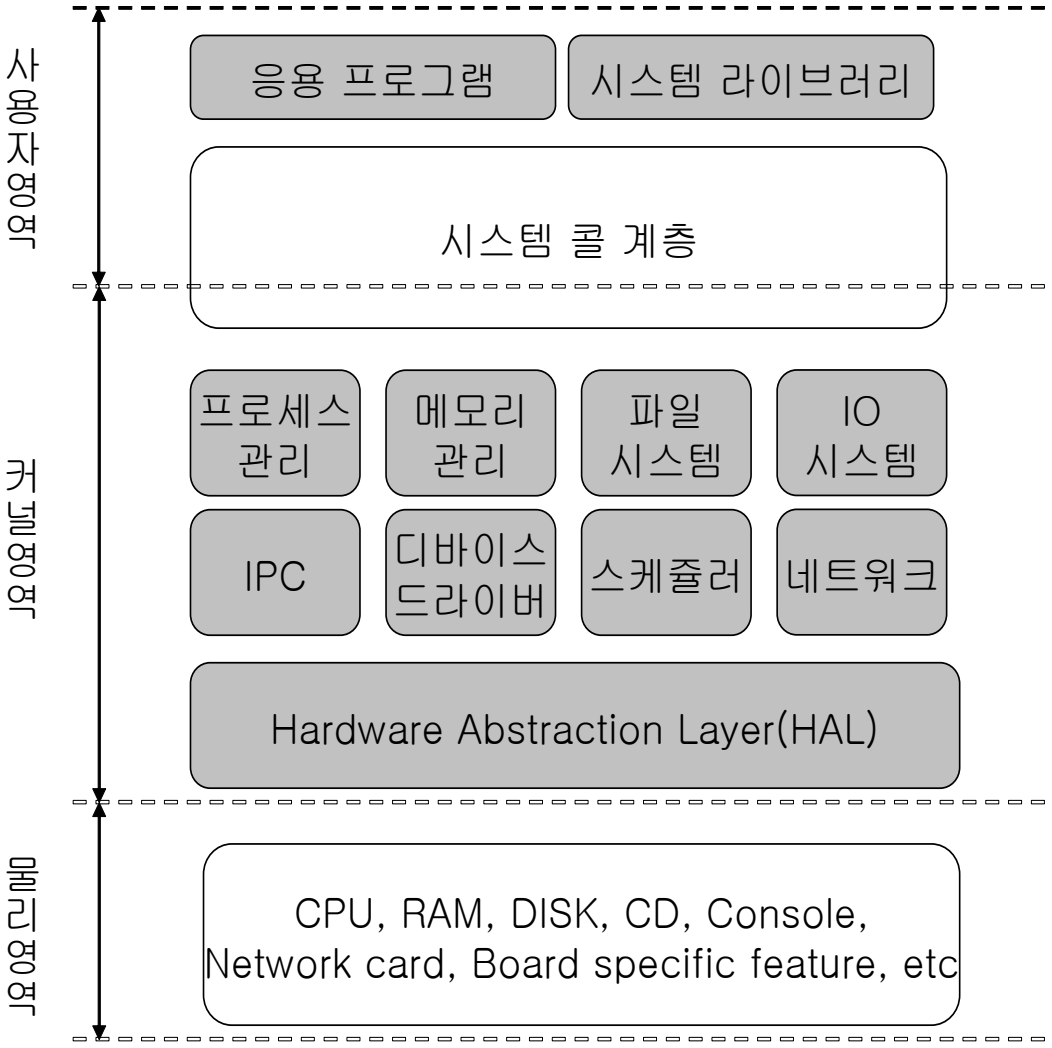
- **Interface between OS and user programs is defined by the set of system calls that OS provides**
- **(ex) Linux has a read system call with 3 parameters: one to specify the file, one to tell where the data are to be put, and one to tell how many bytes to read**
  - **Count = read(fd, buffer, nbytes)**



목차

- **OS(운영체제)**
  - 개념
  - 개론**(overview)**
- 임베디드 운영체제
  - 실시간 운영체제 **(Realtime OS)**
  - 범용 임베디드 운영체제
    - **Windows CE**
    - 임베디드 리눅스
- 임베디드 리눅스 구조

임베디드 운영체제 구조


사용자영역
응용 프로그램
시스템 라이브러리
시스템 콜 계층
커널영역
프로세스 관리
메모리 관리
파일 시스템
IO 시스템
IPC
디바이스 드라이버
스케쥴러
네트워크
Hardware Abstraction Layer(HAL)
물리영역
CPU, RAM, DISK, CD, Console,
Network card, Board specific feature, etc

임베디드 운영체제

- 특수한 목적을 위해 설계되는 하드웨어에서 구동
- 특징
  - 최대한 간단 해야 함
  - 시간의 제약을 받음(실시간)
  - 시스템의 하드웨어 자원, 특히 메모리 사용에 많은 제약
  - 저전력 동작 필요
  - 설치 시 사용자의 많은 설정 작업을 요구
- 임베디드 운영체제 분류
  - 실시간 운영체제
  - 범용 임베디드 운영체제

실시간 운영체제(Realtime OS)

- 실시간 운영체제(**RTOS: Real-Time Operating System**)
  - 주어진 특정 시간 내에 인터럽트를 처리하도록 보장하는 운영체제
  - 특정 태스크를 중단시키고 다른 태스크를 수행할 수 있는 선점형(**preempted**) 멀티 태스킹 지원
  - **POSIX API** 지원
  - (예) **VxWorks, pSOS, VRTX, QNX, OSE, Nucleus** 등
- 특징
  - 다중 스레드 지원, 선점 가능
  - 스레드간 우선 순위 보장
  - 스레드간 동기화 지원
  - 운영체제의 행동이 명확
    - 인터럽트 지연시간, 시스템 콜 처리 시간,
    - 운영체제와 드라이버가 인터럽트를 **masking**하는 시간

실시간 운영체제(Realtime OS) 분류

- 실시간 시스템 (Real Time System) ?
  - 어떤 시스템에 응답을 요구하는 상황 또는 명령의 발생을 이벤트라고 하고, 어떤 이벤트가 발생하게 되면 그것에 대해서 일정 시간 내에 해당 반응을 수행하도록 요구되는 시스템
- 실시간 시스템 종류
  - 경성 실시간 시스템 (Hard Real Time System)
    - 주어진 마감 시간을 만족시키지 못한 경우에 막대한 재산적 손실이나 인명 피해를 야기할 수 있는 응용에 적용되는 시스템
    - 항공기 제어, 군사장비, 무기 방어 시스템, 원자력 발전소 제어 및 의료기기 같은 응용
  - 연성 실시간 시스템 (Soft Real Time System)
    - 시간 제약 조건을 만족시키지 못하더라도 치명적이지 않고, 마감시간을 넘겨서 수행 마쳐도 계산결과가 의미 있는 경우에 적용되는 시스템
    - 텔레매틱스 단말기, 실시간 멀티미디어 서비스, 네트워크 라우터, 디지털 홈 네트워크등의 응용

Realtime OS와 범용 OS 비교

- Realtime OS (RTOS) 와 non-Realtime OS로 구분
- RTOS에서는 deadline 기반의 scheduling 정책사용 <-> 범용 OS에서는 공평한 분배정책사용 (ex, round robin 정책)
- RTOS는 대부분 범용 OS보다 가볍다
- 범용 OS 와는 달리 RTOS에서는 표준 하드웨어 환경이 정해져 있지 않다. 응용프로그램뿐만아니라 OS 자체에서도 사용자가 환경 설정 또는 porting 노력이 요구됨
- Linux는 원래 non-Realtime OS임. Realtime 관련 일부 기능이 표준버젼에 추가되었고 Realtime 지원하는 linux 관련 연구 진행중임.

실시간 임베디드 운영체제 적용기술

- 실시간 임베디드 운영체제의 기능
  - 실시간 스케쥴링 요구
  - 인터럽트 지연 시간 **(**인터럽트 감지에서 해당 인터럽트 서비스 루틴으로의 분기까지 시간**)** 최소화
  - 임계영역**(critical region)** 처리
    - 어떤 태스크가 공유 불가능한 시스템 자원을 사용하는 동안 배타적 접근을 보장하기 위해 지원하는 방법
  - 빠른 부팅시간의 지원
  - 전력관리 지원 **-** 모바일의 증가와 느린 배터리 발전 속도로 인해서 중요한 문제

실시간 운영체제 적용 기술(1)

- 실시간 스케줄링
  - 태스크(**task**) 시간적 특징에 따라
    - 주기(**periodic**) 태스크**:** 일정한 주기에 따라 반복적으로 수행
    - 비주기(**aperiodic**) 태스크**:** 반복성이 전혀 없음
    - 산발(**sporadic**) 태스크**:** 특정 주기 특성을 가지나 언제 수행될지는 모름
  - 마감시간을 어겼을 때의 심각성에 따라
    - 임계(**critical**) 태스크
    - 비임계(**noncritical**) 태스크
  - 수행순위 높은 태스크에 **CPU** 양보 여부에 따라
    - 선점(**preemptive**) 태스크**:** 높은 우선순위를 갖는 태스크에 양보
    - 비선점(**nonpreemptive**) 태스크**:** 양보하지 않음
  - 태스크에 우선순위 부여 하는 방법
    - 정적 우선순위 기법**:** 우선순위가 한번 부여되면 변하지 않음
    - 동적 우선순위 기법**:** 상황에 따라 우선순위가 변함

실시간 운영체제 적용 기술(2)

- 고정우선순위 선점 스케줄링 기법
  - 태스크마다 고정된 우선순위를 부여하고 높은 우선순위를 갖는 태스크가 도착하면 수행중인 태스크를 선점
  - 대부분의 상용 운영체제에서 채택

- 최단마감시간 우선(**EDF: earliest deadline first**) 알고리즘
  - 준비상태에 있는 태스크들 중에서 마감시간이 가장 짧은 태스크를 선택하여 수행

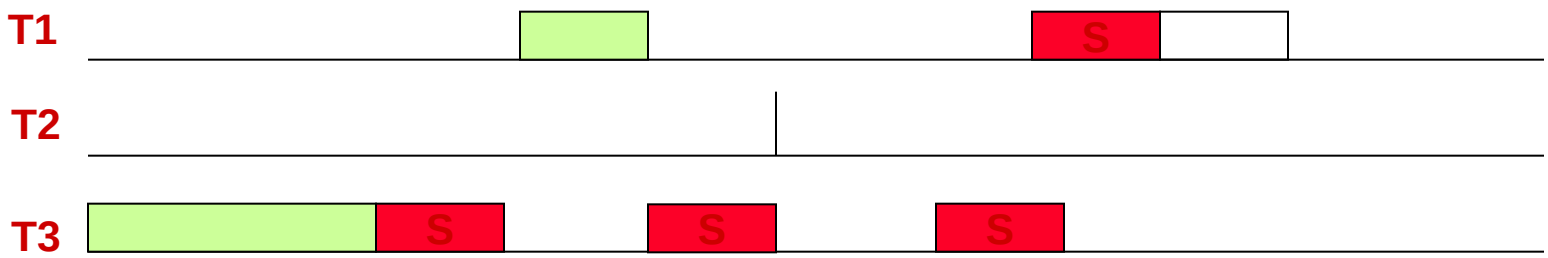
실시간 운영체제 적용 기술(3)

- 인터럽트 지연시간(latency)
  - 인터럽트 처리 단계
    1. **CPU**의 인터럽트 감지
    2. 공통 인터럽트 서비스 루틴 분기
    3. 공통 인터럽트 서비스 실행
    4. 해당 인터럽트 서비스 루틴 분기
    5. 해당 인터럽트 서비스 실행

    인터럽트 지연시간
  - 실시간 시스템에서는 빠른 응답을 해야 하므로 최소화해야 함

실시간 운영체제 적용 기술(4)

- 임계영역(critical region)의 처리
  - 임계영역: 공유될 수 없는 시스템 자원을 접근하는 영역을 말하며, 배타적 접근을 해야 함
  - 선점형인 경우에도 임계영역에 있는 태스크는 중단될 수 없음
  - 우선순위 역전 현상
    - 낮은 우선순위을 갖는 태스크가 동일한 임계영역을 필요로 하는 높은 우선순위를 갖는 태스크를 블록
    - **T1**보다 **T2**가 낮은 우선순위임에도 먼저 종료



실시간 운영체제 적용 기술(5)

- 우선순위상속
  - 임계 영역의 사용으로 인해서 높은 우선순위을 가지는 태스크 T_H 가 낮은 우선순위을 가지는 태스크 T_L 에 의해서 블록될 때, 낮은 우선순위을 가지는 태스크가 일시적으로 높은 우선순위을 가지는 태스크의 우선순위 자체를 상속
  - 블로킹이 끝나면 원래의 우선순위을 되찾음
    - 위 그림에서, **T2**가 **T3**을 선점하려고 할 때 **T3**은 이미 **T1**의 우선순위을 물려받았으므로 선점당하지 않음
  - 여러 임계 영역에 대해서 **deadlock** 발생 가능

- 우선권 상한
  - 우선권 상속에서 발생하는 **deadlock** 문제를 해결

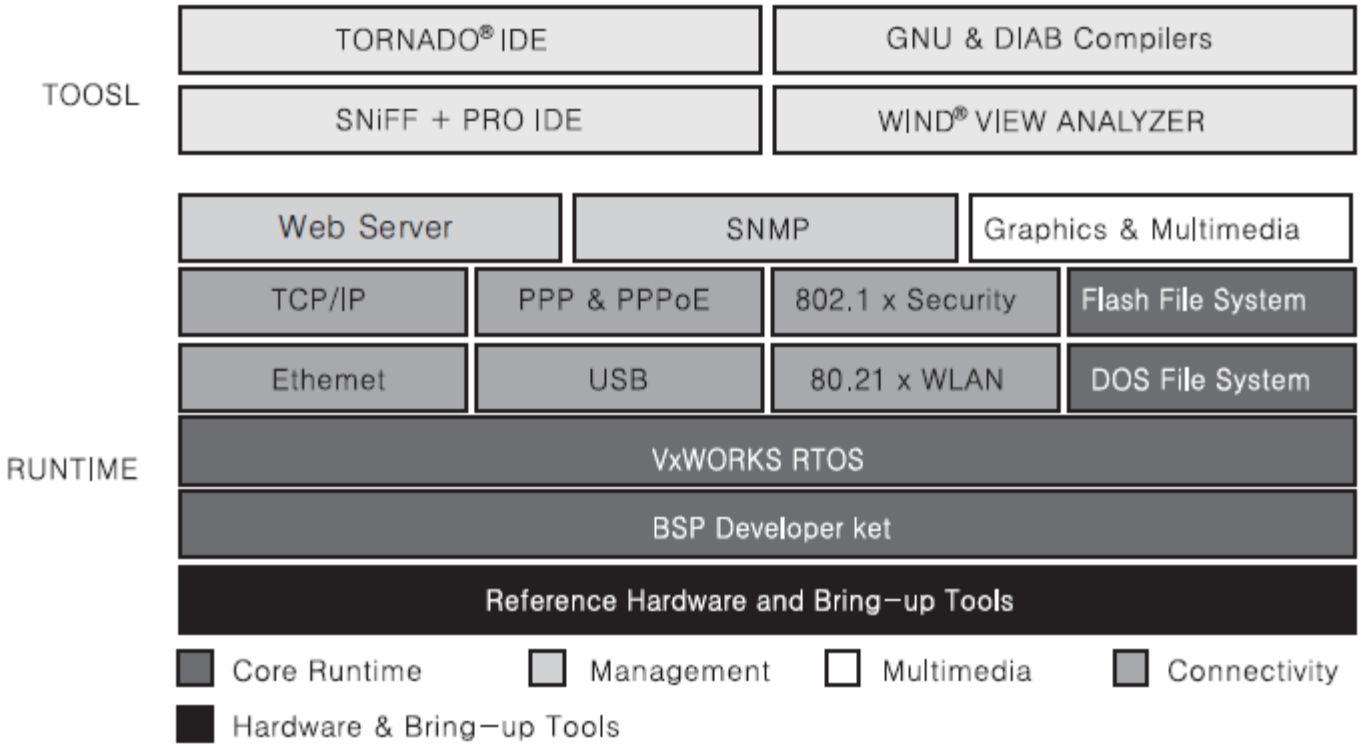
실시간 임베디드 운영체제 종류(1/2)

- **VxWorks**
  - **Wind River System**사에서 개발
  - 안정적이면서 완성도 높은 개발환경으로 가장 높은 시장 점유율
  - 코어 기능을 하는 **Wind** 마이크로커널, 네트워크 라이브러리, 파일시스템, 입출력관리 모듈, **C** 표준 라이브러리 등으로 구성
- **pSoS**
  - **Integrated Systems** 사에서 개발하여 **WindRiver System** 사에 통합
  - 실시간 멀티태스킹 커널인 **pSOS+**와 여러 개의 소프트웨어 모듈로 구성
- **QNX**
  - **QNX Software Systems** 사에서 개발한 **UNIX-like RTOS**
  - **QNX Newtrino**라는 커널을 코어로 내장하였으며, 자원관리자를 통하여 대형 시스템에서부터 작은 내장형 시스템까지 광범위하게 사용

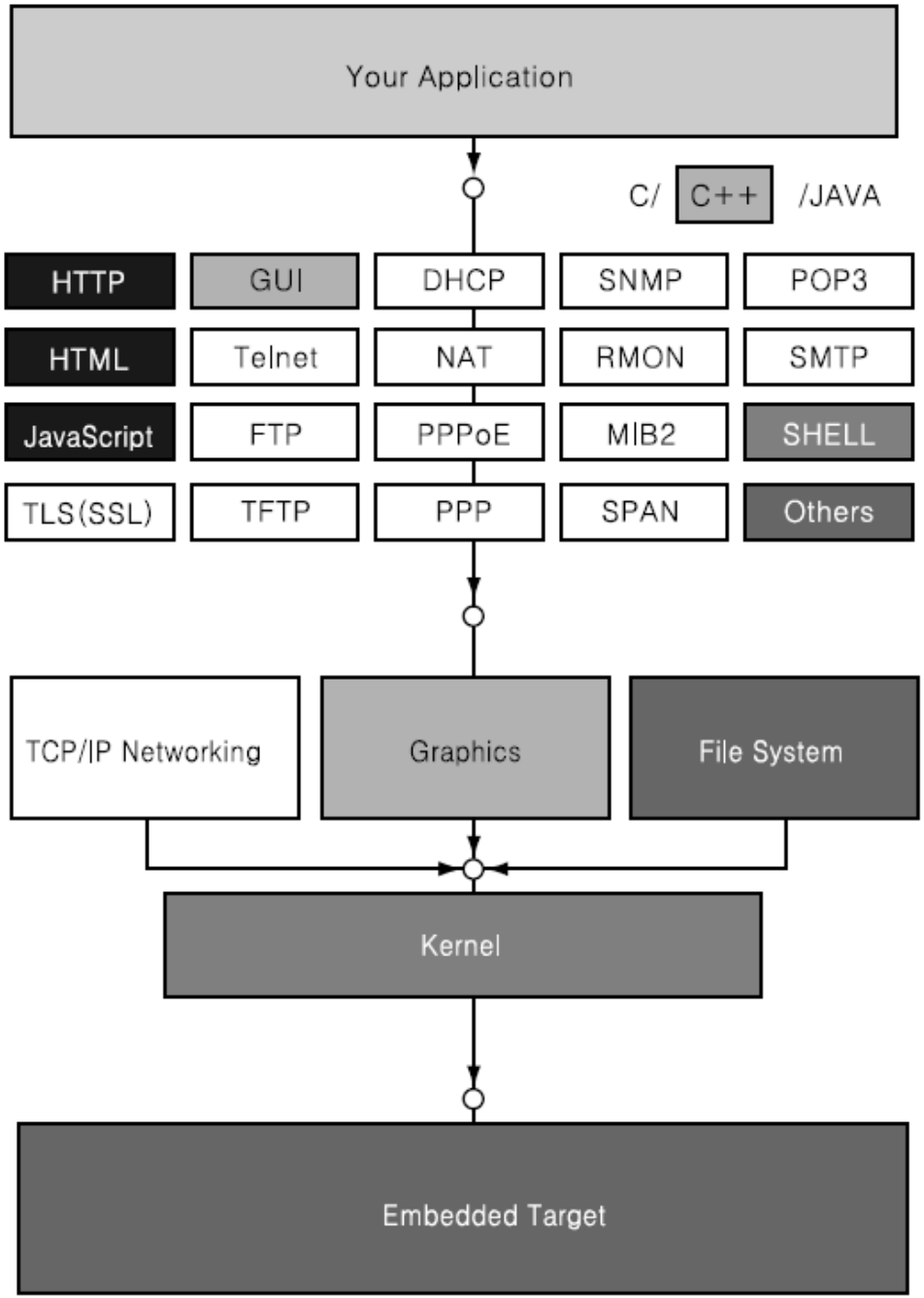
실시간 임베디드 운영체제 종류(2/2)

- **VRTX**
  - **VRTX(Versatile Real-Time Executive)**는 다양한 디버깅 도구를 보유한 **RTOS**로서 현재는 **Mentor Graphics**사에서 공급**.**
  - **VRTX nanokernel**을 기반으로 모듈화 하여 사용자의 요구에 따라 시스템의 재설정이 용이하고 메모리도 최적화 시킬 수 있음
- **Nucleus**
  - **Mentor Graphics**사의 임베디드 시스템 사업부에서 개발
  - 다양한 **CPU** 플랫폼에서 동작 가능
  - 휴대 전화기나 **PMP** 같은 휴대용 기계 및 셋톱박스 등 가정용 전자 제품에 주로 사용
  - 코드와 데이터를 합쳐서 **13 KB** 정도의 작은 메모리 사용

VxWorks


TOOSL
TORNADO® IDE
GNU & DIAB Compilers
SNiFF + PRO IDE
WIND® VIEW ANALYZER
RUNTIME
Web Server
SNMP
Graphics & Multimedia
TCP/IP
PPP & PPPoE
802.1 x Security
Flash File System
Ethernet
USB
80.21 x WLAN
DOS File System
VxWORKS RTOS
BSP Developer ket
Reference Hardware and Bring-up Tools
Core Runtime
Management
Multimedia
Connectivity
Hardware & Bring-up Tools

Nucleus


Your Application
C/
C++
/JAVA
HTTP
GUI
DHCP
SNMP
POP3
HTML
Telnet
NAT
RMON
SMTP
JavaScript
FTP
PPPoE
MIB2
SHELL
TLS(SSL)
TFTP
PPP
SPAN
Others
TCP/IP Networking
Graphics
File System
Kernel
Embedded Target

범용 임베디드 운영체제

- 범용 운영체제 기반으로 **RTOS** 보다 많은 기능 지원
- 마이크로소프트사의 **Windows CE**
  - 기존 윈도우 인터페이스에 모바일 네트워크 기능 강화하여 가전제품, **PDA** 및 셋탑박스등 응용에 적용
  - **2002**년 **Window CE .net**
    - **non-x86** 프로세서 계열 또는 실시간 운영체제로 권장
  - **Window XP** 임베디드
    - 네트워킹과 서버 시스템 적용으로 권장
- 임베디드 리눅스**(Embedded Linux)**
  - 안정성과 신뢰성이 확보된 공개 기술
  - 응용 개발의 저비용
  - **2000**년대에 들어 레드햇, 몬타비스타, 리니오 등이 임베디드 리눅스 개발 및 기술 지원 사업에 주력

범용 임베디드 운영체제 **Linux (1/3)**

- 임베디드 리눅스
  - 리눅스는 초기 데스크탑 또는 서버급 시스템용으로 사용
  - 임베디드 시스템의 응용에 맞도록 리눅스 커널을 최적화 하여 현재는 임베디드 시스템의 운영체제로도 많이 사용됨
  - 상용 임베디드 운영체제에 비해서 실시간 성능이 좋지 못하고, **Windows CE**에 비해서는 개발환경이 좋지 못하다**.** 그러나 임베디드 리눅스는 오픈소스를 제공하고 라이센스 비용이 없기 때문에 장점

범용 임베디드 운영체제 Linux (2/3)

- 임베디드 리눅스 장점
  - 오픈 소스 : 오랜 기간 많은 개발자에 의한 검증을 통하여 기술적으로 안정성과 보안 기술의 확보
  - 개발 환경 : **GNU** 도구 사용 및 **Unix**의 표준인 **POSIX**와 호환성을 제공함으로 리눅스 커널도 메모리 관리. 프로세스 및 쓰레드 생성, 프로세서 간 통신, 파일 시스템, **TCP/IP**에 관한 공통적인 인터페이스를 따르는 **API** 제공.
  - 다양한 하드웨어 지원 : 기본적으로 많은 아키텍처**(x86, Alpha, PPC, ARM, SPARC** 및 메인프레임에 이르는 다양한 **CPU**를 지원**)**와 고성능 **I/O** 장치 지원
  - 저 비용 : 컴파일러, 링커, 라이브러리, 셸**(shell)**과 같은 툴과 유틸리티들을 무료로 사용 가능
  - 독립성 : 벤더 종속성이 높은 고가의 **IDE(**통합개발환경**)** 사용할 필요가 없음

범용 임베디드 운영체제 **Linux (3/3)**

- 임베디드 리눅스 단점
  - 다른 임베디드 시스템 전용 운영체제에 비해서 실시간성이 취약함**.** 리눅스에서 이를 보완하기 위해 여러 일들이 진행되고 있슴 **(Preemptive kernel** 지원**, dual kernel** 구조형태로 기존 **linux** 이외에 **Real-time** 처리 모듈 지원**)**
  - 상용 실시간 운영체제에 비해서 메모리 사용량이 많고 전력 사용량이 높음**.**
  - 응용프로그램이 부족
  - 장비 개발할 때 개발 환경 접근이 어렵음**.** 커널을 비롯하여 마이크로프로세서**,** 네트워크**,** 및 각종 툴 등 다루어야 함

임베디드 리눅스와 데스크탑 리눅스 차이점

- 사용되는 파일 시스템과 디바이스 드라이버 서로 다르다. 임베디드 시스템은 플래시 드라이버 또는 플래시 용 파일시스템(**CRAMFS** 또는 **JFFS2**)을 필요로 하지만, 일반 데스크탑용 리눅스는 그렇지 않다.
- 임베디드 리눅스는 컴파일, 디버깅, 프로파일링 등의 개발 툴에 중점을 두는데 일반 데스크탑 용 리눅스는 워드프로세서, 인터넷 검색 등의 응용프로그램에 중점을 둔다.
- 배포판에 포함되는 유틸리티가 서로 다르다. 임베디드 리눅스에서는 적은 디스크 용량 및 적은 메모리 사용량을 요구하는 유틸리티가 사용된다. 임베디드 리눅스에서는 **Ash, Tinylogin, BusyBox** 등이 포함되며 **C** 라이브러리는 **uClibc**가 사용된다. 데이크탑에서는 **C** 라이브러리는 **Glib**가 사용된다.
- 윈도우 시스템과 **GUI** 환경 다르다. 일반적으로 데스크탑에서는 **X** 윈도우 시스템가 사용디는데 임베디드 리눅스에서는 **Microwindows(Nano-X)**가 제공.
- 임베디드 리눅스에서는 시스템 관리 기능을 거의 사용하지 않지만, 데스크탑용 리눅스에서는 사용한다.

리눅스가 임베디드용으로 너무 큰 **OS**인가?

- 아니다 **-** 네트워킹과 파일 시스템을 가진 최소형 임베디드 리눅스는 **4MB of SDRAM** 과 **2MB of flash**로 충분하다.
- 4MB of flash 와 16 or 32 MB of SDRAM이면 여러가지 응용을 올릴 수 있다.
  - Uclinux - a linux port for no-MMU platforms such as Motorola 68K, ARM7
  - ELKS(Embedded Linux Kernel Subset) - embedded linux를 palm pilot에 포팅
  - Thinlinux - digital cameras, MP3 players 및 비슷한 embedded applications 적용을 목표로 함

상용 임베디드 리눅스 배포판 사용 ?

- 임베디드 리눅스 경험이 없다면 상용 임베디드 리눅스 배포판 사용하는 게 좋다.
- 상용 임베디드 리눅스 사용시 이점
  - 기술지원이 잘된다.
  - 개발 툴과 유틸리티 지원이 잘된다.
  - 커널 업그레이드가 잘 된다.

임베디드 리눅스 배포판(1/4)

- **BlueCat Linux**
  - 리눅스웍스(**www.lynuxworks.com**)에서 판매하는 리눅스 커널 **2.6**기반의 상용 임베디드 리눅스 운영체제
  - 모바일 기기에서부터 다중 프로세스 시스템에 이르기 까지의 광범위한 시스템을 타깃
  - **BlueCat**은 선점형 커널이며, **NPTL(New POSIX Thread Library)** 기반의 새로운 **POSIX Thread** 구현을 통해서 향상된 기능을 제공
  - **XScale, PowerPC, IA32, ARM, MIPS, x86** 호환의 타겟을 제공
- **ELDK**
  - **Denx** 소프트웨어 엔지니어링사(**www.denx.de**)는 **ELDK(Embedded Linux Development Kit)**의 형태로 오픈 소스 리눅스 배포판을 제공
  - 완전한 고성능 소프트웨어 개발 환경을 지원
  - **PowerPC, ARM, MIPS** 및 **XScale** 계열 프로세서를 지원하며, **PowerPC** 버전의 **ELDK**는 **x86/Linux, x86/FreeBSD, SPARC/Solaris** 호스트에서 동작하고 다양한 종류의 **PowerPC** 타깃 프로세서를 지원
  - 엄격한 실시간 응답을 보장하기 위해서 **RTAI(Real-Time Application Interface)** 확장을 제공
  - 손쉬운 환경설정을 위한 **SELF(Simple Embedded Linux Framework)**를 제공.
  - **GUI** 기반의 응용 프로그램을 실행할 수 있는 **Microwindows** 시스템 지원

임베디드 리눅스 배포판 (2/4)

- **Embedian**
  - 기존의 데비안**(Debian) GNU/Linux** 시스템을 임베디드 리눅스 환경에서 사용하기 위한 것이다**.** 기존 크기를 줄인 최적화된 운영체제
  - 데비안의 코어는 동일하며**,** 패키지 빌드 방식에서 차이가 있다**.**
  - 현재 인텔의 **IA32,** 모토롤라 **M68K, SPARC, Alpha, PowerPC, MIPS, HP PA-RISC, IA64** 및 **S390** 등의 아키텍처에 포팅되어 있으며**,** 주 타깃은 **PowerPC** 아키텍처이다**.**
  - 개발 환경으로는 **Stag**와 **Emdebsys**를 제공한다**. Stag**는 데비안 **GNU/Linux** 패키지 관리 시스템에서 사용하는 프레임워크이다**.**
- **ELinOS**
  - **SYSGO(www.sysgo.com)**에서 판매되는 상용 임베디드 리눅스 운영체제이다**.**
  - 지능형 기기에 임베디드 리눅스를 구성하기 위한 리눅스 기반의 개발 환경이다**.**
  - **RTAI, LTT(Linux Trace Toolkit)**등을 지원
  - **PowerPC, x86, ARM/XSCALE, MIPS**와 **SH**등의 아키텍처를 지원
  - 개발환경으로 **CODEO**와 **COGNITO**로 구성된 **IDE**를 제공

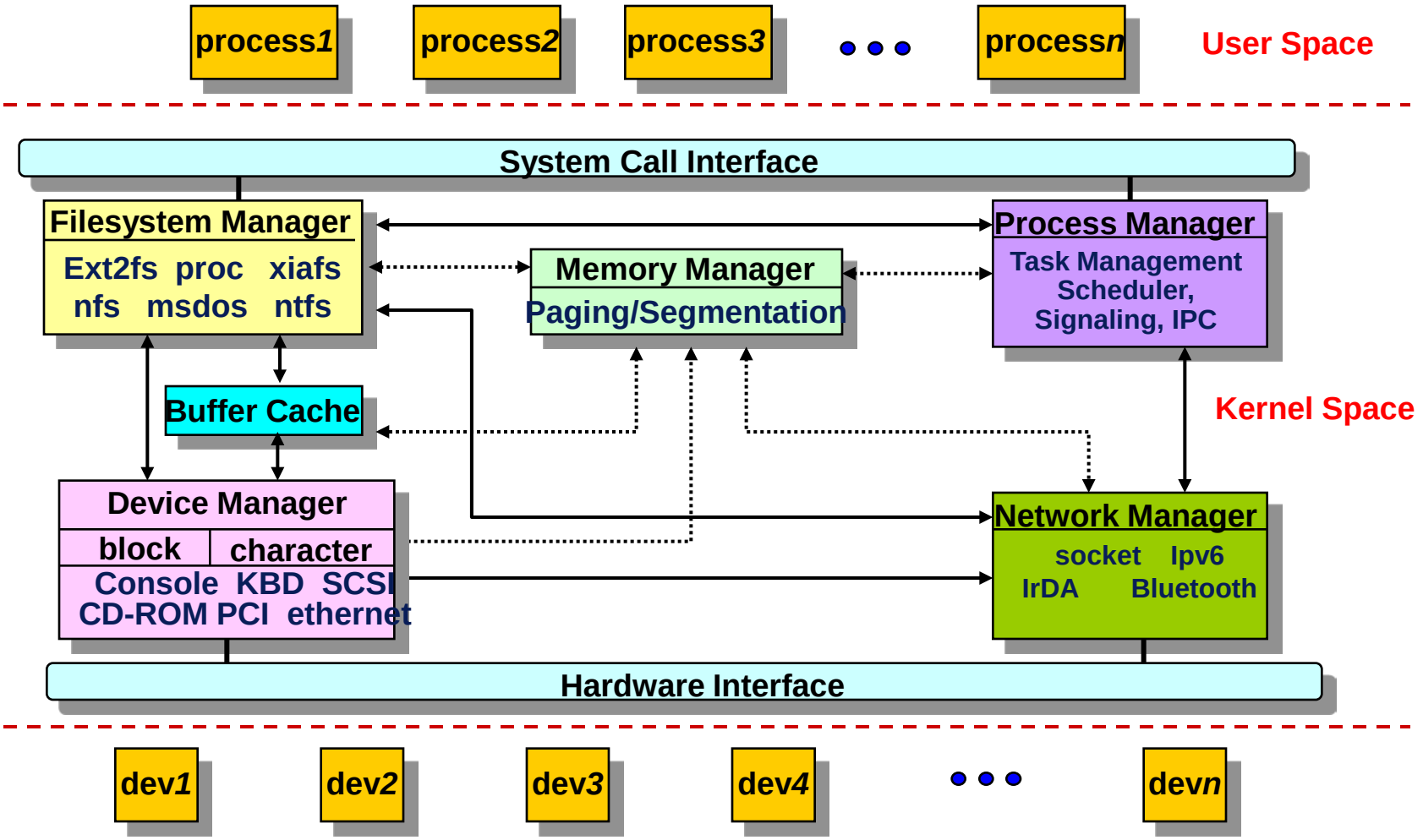
임베디드 리눅스 배포판 (3/4)

- **Metroworks (www.metroworks.com)**
  - 툴, 운영체제, 미들웨어, 소프트웨어 스택을 포함하는 완전한 상용 임베디드 리눅스 운영체제 제공
  - **x86, ARM, PowerPC, ColdFire**등의 아키텍처를 지원
  - 개발환경으로는 **CodeWarrior Development Studio**와 **PCS(Platform Creation Suite)**를 제공
- **MontaVista Linux (www.mvista.com)**
  - 차별화된 응용 분야에 따라서 세 가지의 배포판을 제공
  - 개발 환경으로는 **GUI** 기반의 **IDE**로서 **MontaVista DevRocket**를 제공하는데, 이것은 윈도우즈, **Solaris,** 리눅스 호스트 상에서 동작
  - **100**개 이상의 플랫폼과 **7**개 **CPU** 아키텍처에 걸쳐 약 **30**개의 프로세서 유형을 지원**.**

임베디드 리눅스 배포판 (4/4)

- **RTLinuxPro**
  - 듀얼 커널 구조에 기반한 엄격한 실시간성과 **POSIX** 표준을 만족하는 **FSMLabs**사**(www.fsmlabs.com)** 사의 상용 임베디드 운영체제
  - **1**초 이내로 부팅 시간을 단축하였으며**,** 특정 플랫폼에서는 **200msec** 이하의 빠른 부팅시간을 제공
  - **x86, ARM, StrongARM, MIPS, PowerPC, Alpha, Fujitsu FRV** 등 아키텍처 지원
  - **RTLinuxPro**의 개발환경으로 임베디드 리눅스를 위한 개발환경과 **RTCore** 상에서 동작할 실시간 응용 프로그램을 위한 개발환경을 제공
- **TimeSys Linux**
  - 리눅스 **2.6** 커널을 기반으로한 상용 임베디드 리눅스 운영체제
  - 선점형 커널**, POSIX**를 포함하며**,** 고정밀 타이머**,** 우선순위 상속**,** 인터럽트 처리개선**,** 소프트 **IRQ** 처리등을 지원
  - 시스템 프로세서 자원 중 일부를 중요한 응용 프로그램을 위해서 할당해 둠으로써 프로세서에 과부하가 걸린 상황에서도 실시간 성능을 보장하도록 하였다**.** 이것을 위해서 **CPU** 예약을 위한 간단한 **API**를 제공
  - **ARM, XScale, PowerPC, MIPS, SuperH, IA-32**등의 다양한 프로세서를 지원하며**,** 윈도우즈와 리눅스 호스트에서의 교차 개발을 지원

리눅스 운영체제 구조



(Source : Linux Kernel Internals)

목차

- **OS(운영체제)**
  - 개념
  - 개론**(overview)**
- 임베디드 운영체제
  - 실시간 운영체제 **(Realtime OS)**
  - 범용 임베디드 운영체제
    - **Windows CE**
    - 임베디드 리눅스
- 임베디드 리눅스 구조

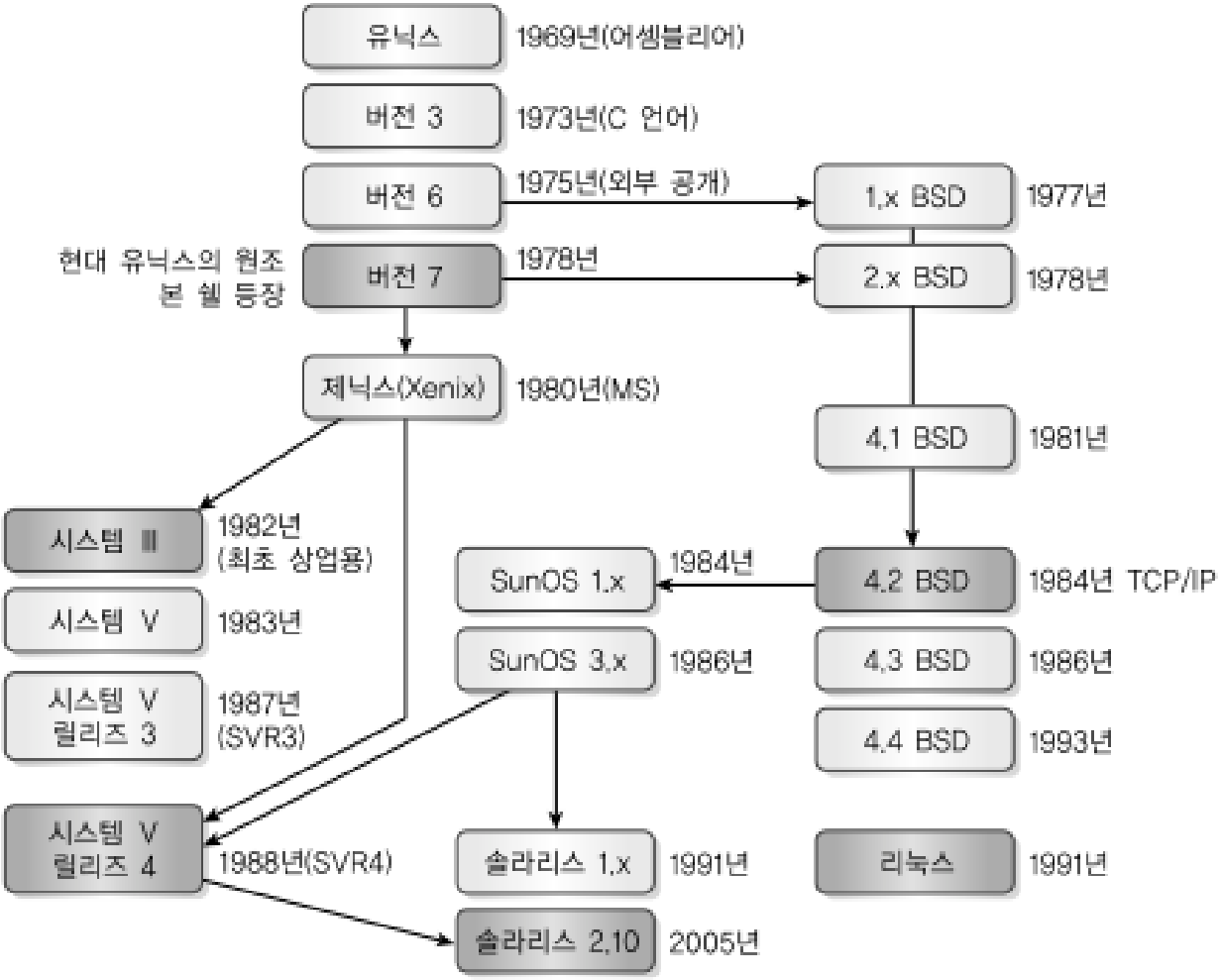
리눅스 시스템의 역사 (1)

- **UNIX** 시스템의 역사
  - **1969, AT&T**산하의 벨연구소에서 켄 톰슨과 데니스 리치가 유닉스 개발
  - **1973, C**언어를 이용하여 재개발

    ➔ 고급 언어로 작성한 최초의 운영체제
  - 이후 상용유닉스**(**시스템**V)** 계열과 **BSD** 계열로 분리하여 각각 발전
    - 다양한 계열의 유닉스 운영체제 등장 ➔ 유닉스 운영체제간의 프로그램 호환성 대두
  - **1989, AT&T**와 썬마이크로시스템즈가 두 계열의 장점을 결합하여 **SVR4**를 공동개발
    - 현재 사용하는 대부분의 유닉스의 기반임
  - **1991**년 유닉스를 기반으로 리눅스 개발

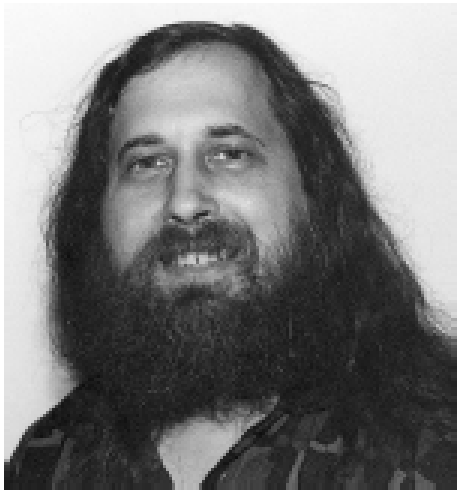
리눅스 시스템의 역사 (2)

- **UNIX** 시스템의 역사



A Short History of GNU(1)

- 1984: Richard Stallman starts GNU project
  - GNU's Not Unix
  - http://www.gnu.org
- Purpose: Free UNIX
  - "Free as in Free Speech, not Free Beer"
- First step: re-implementation of UNIX Utilities
  - C compiler, C library
  - emacs
  - bash
- To fund the GNU project, the Free Software Foundation is founded
  - http://www.fsf.org

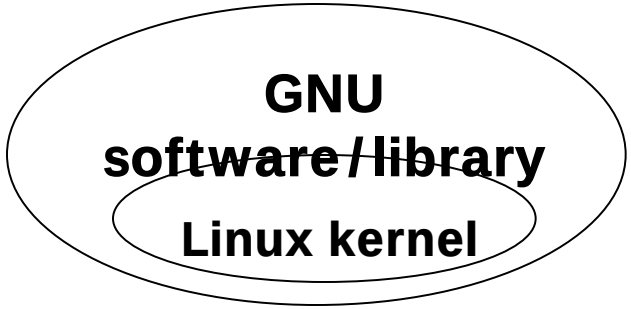


A Short History of Linux(2)

- 1991: Linus Torvalds writes 1st version of Linux kernel
  - Initially a research project about the 386 protected mode
  - Linus' UNIX -> Linux
  - Combined with the GNU and other tools forms a complete UNIX system
- 1992: First distributions emerge
  - Linux kernel
  - GNU and other tools
  - Installation procedure
- The rest is history...

GNU & Linux

- GNU/Linux System
  - Linux kernel
  - GNU software/library



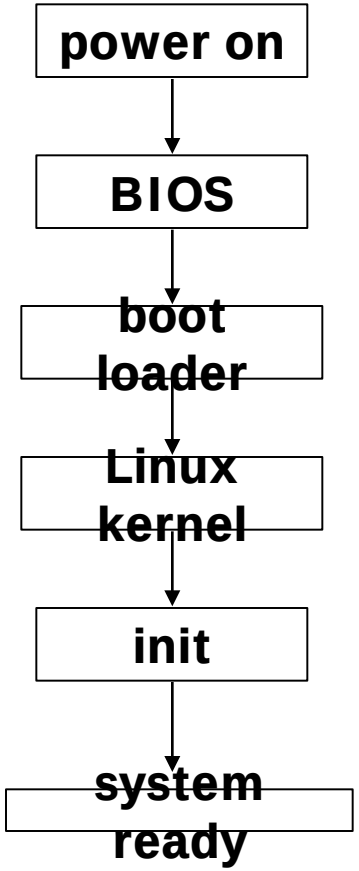
Distributions：

- Red Hat, Debain, Ubuntu SuSe, Mandrake, Redflag…

Mobile：

Android

Linux Startup Flow



- BIOS
  - Checks memory, loads options from non-volatile memory, checks for boot devices, loads MBR of boot device and executes it
- MBR
  - Contains a “boot loader” and the partition table
  - Traditionally set up by LILO/GRUB
- Boot loader
  - Loads the compressed kernel image into memory
  - The kernel uncompress itself and starts…
- Init process
  - Configuration file /etc/inittab
  - run levels

임베디드 리눅스 구조 - HAL

- **HAL (hardware abstraction layer)**
  - 하드웨어 보드내 장치를 초기화하거나 사용하는 장치 종류에 따라 달라지는 장치 사용 소프트웨어 루틴들을 말함.
  - 커널이나 디바이스 드라이버는 **HAL**을 사용하여 하드웨어를 제어하도록 **HAL**은 하드웨어 플랫폼을 가상화해줌.
  - **HAL**은 드라이버를 다른 하드웨어에 쉽게 포팅할 수 있도록 지원해줌.
  - 리눅스가 처음에는 **x86** 데스크탑용으로만 개발되어 다른 하드웨어로의 포팅을 고려하지 않았지만 최신 커널 버전에서는 다른 아키텍처에서도 쉽게 사용할 수 있도록 **API**를 제공
  - **2.6**에서 지원하는 프로세서 아키텍처로는 **x86, MIPS, PowerPC, ARM, M68K, CRIS, V850, SuperH**등이 있슴
  - **HAL**에는 프로세서, 캐시 메모리, **MMU(memory management unit)**등의 하드웨어를 관리하고, 메모리 맵 설정, 예외 인터럽트의 처리, **DMA** 관리, 버스관리, 전원관리, 및 타이머와 시스템 콘솔의 제어가 포함된다.

임베디드 리눅스 구조 – 메모리관리(1/2)

- 드라이버, 파일시스템, 네트워크스텍과 같은 커널 서브 시스템을 위한 동적 메모리를 제공
- 사용자 응용 프로그램을 위해 가상 메모리 지원
- 리눅스에서는 OS 와 응용프로그램은 각각 별도로 컴파일한다. 반면에 RTOS에서는 OS와 응용프로그램을 묶어서 컴파일한다
- 리눅스에서는 메모리를 페이지 단위로 나누어서 관리한다. 페이지의 대표적인 크기는 4KB 임

임베디드 리눅스 구조 – 메모리관리(2/2)

- 메모리 관리자 작업들
  - 가상 메모리 제공 : 실제 물리적 메모리 보다 많은 용량의 가상 메모리를 제공
  - 메모리 매핑 : 이미지와 데이터 파일을 프로세스의 주소공간에 매핑하기 위해서 사용된다. 메모리 매핑에서 파일의 내용은 프로세스의 가상 주소 공간에 직접 연결
  - 공유 메모리 : 가상 메모리를 공유하는 기능을 제공
  - 스왑 메모리 (swap memory) : 페이지 단위로 현재 실행되지 않는 프로세스 영역을 하드 디스크에 저장하는 작업을 의미.

임베디드 리눅스 구조- 스케쥴러

- 스케쥴러
  - 멀티 테스킹을 지원하며 실시간 스케쥴링 정책을 지원하기 위해서 계속 진화
  - 리눅스 스케쥴러의 실행 단위 - 커널 쓰레드/ 사용자 프로세스 / 사용자 쓰레드
  - 실시간 스케쥴링 정책
    - **2.2** 커널이후로는 **POSIX** 실시간 커널을 지원
    - **2.4** 커널이후에는 몬타비스타의 선점형 커널 패치와 앤드류 모튼의 저지연 패치가 적용
    - **2.6** 커널에서는 **O(1)** 스케쥴러라는 새로운 스케쥴러가 도입, **POSIX** 타이머와 같은 실시간 기능들이 커널에 포함

임베디드 리눅스 구조- 파일 시스템

- **File System 개념**
  - 개념: 운영체제가 저장 매체에 파일을 저장하는 방식
    - 자장매체 파티션: 저장 매체의 공간을 구획지어 독립된 가상 공간화
      - Fdisk /dev/hda //어떤 디스크의 파티션을 나눌지 생각 후 실행
    - 파일 시스템 구성
      - 파일 정보를 기록한 자료 구조의 생성 (inode 구조)
        - » Ex2, ex3, minix, CDROM, isofs, hpfs, NFS, sysv, …
      - Mount: 파일 시스템 구조 내의 파일을 사용할 수 있도록 논리적인 파티션을 시스템에 연결함
- 임베디드 **File System 개념**
  - **Flash File System:** 하드디스크와 유사
    - 가비지 컬랙션: 삭제 후 사용 가능
    - 닳기 균등화: 블록의 삭제회수에 의한 수명
    - 플래쉬 파일시스템:파일시스템을 플래쉬메모리에 올리기 위한 중계
    - **MTD(Memory Technology Device):** 플래쉬메모리에 무관하게 동작하도록 지원

임베디드 리눅스 구조- 파일시스템(1/3)

- 리눅스에서는 여러 파일 시스템이 **VFS (Virtual File system)** 라는 **layer**에서 관리된다
- 리눅스 기기에서는 최소한 파일 시스템이 필요하다**. RTOS**에서는 그렇지 않다**.**
- 리눅스에서 파일 시스템이 필요한 이유
  - 응용프로그램이 **OS**와 별도로 컴파일되므로 별도 저장공간이 필요하다
  - 모든 저수준 기기들도 파일처럼 접근하여 사용가능
- 리눅스에서는 디스크 기반 파일 시스템 이외에 플래시 또는 **ROM** 기반 파일 시스템을 지원한다**.** 네트워크 파일 시스템 **(NFS)**도 지원한다
- 파일시스템은 **OS**가 파일을 시스템 디스크상에 구성하는 방식이다**.** 디스크 파티션상에 파일이 연속적이고 일정한 규칙을 가지고 저장된다**.**

임베디드 리눅스 구조- 파일시스템(2/3)

- 리눅스에서 파일시스템 저장 규칙
  - 슈퍼블록 : 파일시스템의 전체적인 정보(크기, 마운트 회수, 블록 그룹 넘버, 블록 크기, 첫 번째 아이노드 등)를 포함하고 있다.
  - 아이노드(inode) : 파일 이름을 제외한 해당 파일의 모든 정보(데이터블록의 위치정보, 파일형태, 파일소유자, 아이노드)를 포함하며 각 파일 이름에 부여되는 고유 번호이다. 아이노드 테이블에서 아이노드 번호를 가지고 검색하여 알려준다.
  - 데이터 블록 : 파일에서 데이터를 저장하기 위해서 사용되는 공간이다. 이러한 데이터 블록은 아이노드에 저장되며, 하나의 아이노드는 보통 다수의 데이터 블록을 가지고 있다.
  - 디렉토리 블록 : 파일 이름과 아이노드 번호를 저장한다.
  - 간접 블록 : 아이노드에 데이터 블록의 위치 정보를 저장할 공간이 부족한 경우, 위치 정보를 저장하기 위한 공간을 동적 할당하게 된다. 이 때 할당된 공간을 간접 블록이라고 한다.

임베디드 리눅스 구조- 파일시스템(3/3)

- **EXT2 :** 리눅스의 기본 파일 시스템
- **CRAMFS :** 압축된 읽기 전용의 파일 시스템
- **ROMFS :** 읽기 전용 파일 시스템
- **RAMFS :** 읽기와 쓰기가 가능한 **RAM**을 위한 파일 시스템
- **JFFS2 :** 플래시 메모리에서 사용되는 저널링 파일 시스템
- **PROCFS :** 시스템 정보를 얻기위한 가짜**(Pseudo)** 파일 시스템
- **DEVFS :** 디바이스 파일을 관리하기 위한 가짜 파일 시스템

- 각 파일시스템은 성능, 공간활용, 또는 데이터 복구에 최적화 됨
- 타겟 시스템은 요구 조건에 맞게 파일 시스템을 선택하여 사용함

MTD (Memory Technology Device) 서브시스템

- **MTD** 서브시스템
  - 플래시 메모리와 같은 다양한 메모리 형태의 장치를 지원하기 위해서 만들어진 기술
  - 메모리 장치가 사용하는 상위 레벨의 데이터 구조와 저장형식을 하위 레벨 장치의 복잡한 구조와 분리시킬 수 있도록 함.