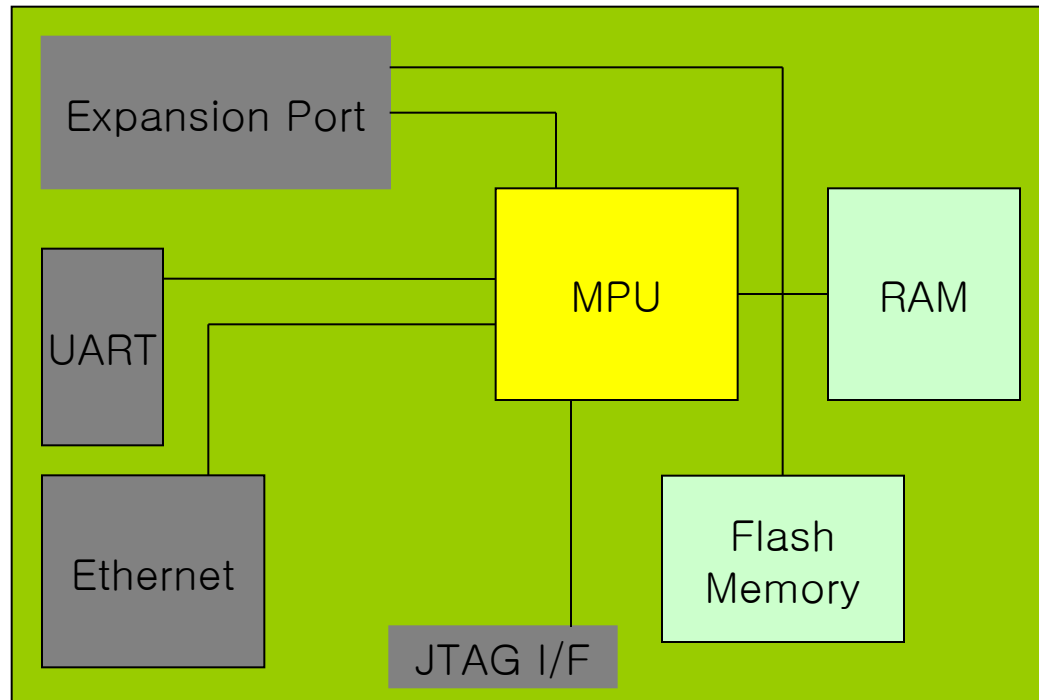

Introduction to Embedded Linux

임베디드 시스템

- 정의
 - 어떤 특정한 기능을 위해 Microprocessor/Microcontroller가 내장된 시스템
- 특징
 - 제한된 하드웨어 자원(최소한의 필요한 자원)
 - Processor, RAM, Flash memory, interfaces
 - 경량의 OS 및 Real-Time OS 사용
 - WinCE, Vxworks, Nucleus

임베디드 시스템의 H/W

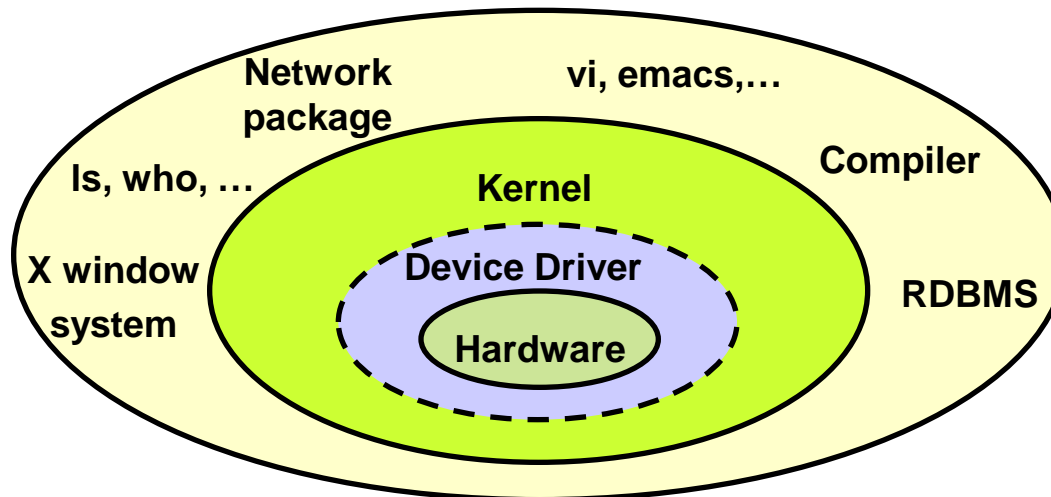


Target Board Block diagram

- JTAG(Joint Test Action Group: IEEE1149.1b)

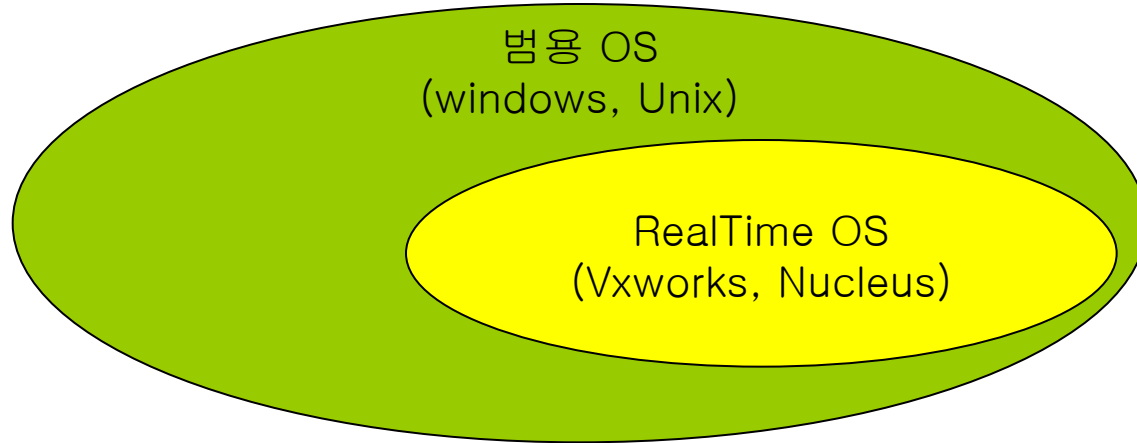
운영체제

- 운영체제란?
 - 자원 관리자 (Resource Manager)
 - 응용에게 자원에 대한 서비스 제공 (Computing Environment)
- 자원의 종류
 - 물리적인 자원 : 처리기, 메모리, 디스크, 터미널, 네트워크, ...
 - 추상적인 자원 : 태스크, 세그먼트/페이지, 파일, 드라이버, 통신 프로토콜, 패킷, 보안, ...



- RTOS(Real-Time OS)
 - VxWorks, pSOS, Nucleus
- UNIX
 - SVR4, Solaris
 - BSD, HP-UX
- Linux
- Windows System
 - Windows XP, 7, 10
 - Windows CE.NET

임베디드 시스템의 OS



- 범용 OS
 - 모든 태스크(Task)를 공평하게 처리
- Real-Time OS
 - 우선권이 높은 태스크부터 처리

임베디드 시스템의 OS

- Windows CE, Windows Embedded
 - MS사에서 임베디드 시스템을 위하여 제공하는 운영체제
 - 기존의 데스크 탑 PC와 동일한 윈도우 환경 제공
 - 데스크 탑 윈도우 및 응용 프로그램과의 호환성 우수
 - 프로그램 개발 환경이 아주 우수. (특히, GUI 개발 환경 우수)
 - 실행 환경에서 요구되는 H/W 사양이 높고, 가격이 비쌘
 - MS사에서 제공되는 라이브러리에 종속적

임베디드 시스템의 OS

- 임베디드 자바
 - SUN사에서 제공하는 JAVA를 이용한 운영체제
 - 임베디드 환경에 맞는 개발환경을 제공하고 있으며 웹 기반의 환경에 우수
 - 스크린폰, 셋톱박스 등을 중심으로 사용
 - 휴대폰 등 이동 서비스 : K자바 기반의 KVM(Kilobyte Virtual Machine) 기술을 적용

임베디드 시스템의 OS

- 리눅스
 - 일반 리눅스
 - 일반 데스크탑 환경인 고성능 프로세서와 대용량 메모리 환경에서 동작하는 범용 컴퓨터용 리눅스
 - 임베디드 리눅스
 - 저성능의 마이크로 프로세서와 제한된 메모리 환경에서 동작하는 임베디드 시스템용 리눅스

임베디드 시스템의 OS

■ Real-Time System

■ 정의

- 정해진 시간 내에 시스템이 결과를 출력하는 시스템
- 주어진 작업을 빨리 처리하는 것이 아니고 정해진 시간을 넘어서는 안 된다는 뜻임

■ 임베디드 시스템은 대부분 실시간 적인 요소 내포

■ 임베디드 시스템에 실시간 시스템이 포함됨

■ Hard real-time system/Soft real-time system

• Hard real-time system

- 정해진 시간 내에 작업의 결과가 절대적으로 출력되어야 하는 시스템 - 시간 내에 처리되지 않으면 치명적인 결과를 초래하는 경우
- 전투기의 비행제어 시스템, 핵발전소의 제어 시스템, 인공위성의 제어 시스템

• Soft real-time system

- 정해진 범위를 넘는 시간 지연이 발생하더라도 그것이 시스템의 에러가 되지 않는 시스템

임베디드 시스템의 OS

- 실시간 시스템에서의 S/W
 - 간단하고 단순한 순차적인 작업에 관련
 - 순차적인 프로그램으로 충분하였음
 - 8bit, 16bit 마이크로프로세서 및 마이크로컨트롤러 사용
- 임베디드 시스템에서의 운영체제
 - 시스템의 규모가 커짐에 따른 **Multi Tasking** 기능 요구
 - **Network**이나 **multimedia**가 시스템의 기본으로 자리 잡음
 - Networking, GUI, Audio, Video
 - 임베디드 시스템의 특성상 실시간이라는 요소를 만족해야 함
 - 지능성이 부가되고 기능이 많아지고 복잡해짐
 - 순차적인 프로그램 작성이 불가능하여 운영체제가 도입됨

임베디드 시스템의 OS

- 상용 RTOS : Hard Real-Time/Multi-thread/ Preemptive
 - pSOS, VxWorks, VRTX 등 다수
 - 일반 운영체제와 거의 같은 기능을 수행
 - 시간 제약성, 신뢰성 등을 일반 운영체제 보다 중요시 함
 - 일반적으로 한가지 목적에 최적화 되어있음
- 임베디드 OS : Soft Real-Time/Multi-process/ non preemptive
 - Windows CE
 - 임베디드 리눅스
 - 임베디드 자바
- 최근 동향
 - 임베디드 OS 세계시장 :
 - 최근 WinCE, 임베디드 리눅스가 기존의 RTOS 보다 시장 점유율이 높아지는 추세
- OS 선정
 - 시스템의 특성 파악 그 시스템에 적합한 OS 선정이 매우 중요

임베디드 OS 솔루션 고려사항-1

- 제품 개발 내용 및 범위의 정의
- 비용, 시간, 안정성
 - Open source vs. Proprietary Product
 - 직접 개발 vs. 외주(Outsourcing)

임베디드 OS 솔루션 고려사항-2

- Open source
 - 모든 필요한 것을 찾을 수 있다
 - Setup Time?: 솔루션 확보 및 환경구축
 - 문제 해결 방법?
 - 필요한 기술력 확보를 위한 기간?
- Proprietary Product
 - 더 편리하고 효율적인 툴
 - 빠른 Setup Time
 - 믿을 수 있는 기술 지원
 - 종속성?
- 직접개발
 - 제품 유지보수의 안정성
 - 문제해결 방법 및 기술 지원
 - 충분한 개발인력 확보
- Outsourcing
 - 빠른 제품 개발
 - 제품 유지 보수 문제

임베디드 OS의 주요 개념

- 태스크
 - 수행중인 프로그램 (an instance of a running program)
 - 프로그램의 수행 환경 (an execution environment of a program)
 - 스케줄링 단위 (scheduling entity)
 - 제어 흐름과 주소 공간의 집합 (a control flow and address space)
- 멀티 태스킹
 - 여러 개의 태스크를 동시에 실행시키는 것
 - 일반 OS 에서의 태스크
 - 각 태스크들은 대부분 무관한 프로그램임
 - 임베디드 시스템에서의 태스크
 - 하나의 큰 응용 프로그램을 논리적으로 나눈 것
 - 기능상 매우 밀접한 관계
 - 태스크 사이에 이루어지는 작업들이 많다.
 - 응용 프로그램을 실행을 위해 여러 기능들이 동시 실행 요구
 - 순차적이 아닌 동시 실행의 필요성이 있다.

임베디드 OS의 주요 개념

- 스케줄러(Scheduler)
 - OS의 핵심기능
 - 다음 번에 어떤 태스크를 실행해야 하는 지를 결정하는 코드 부분
 - 태스크 선택 정책 : 우선순위 기반의 스케줄링
 - FIFO(First In First Out), Round-robin 등
 - 선점(Preemptive)
 - 어떤 태스크가 수행되고 있을 때 커널이 중간에 그 태스크의 수행을 중지 시키고 다른 태스크의 기능을 수행시키는 기능
 - 선점형 커널 / 비선점형 커널
 - 다른 태스크로 실행이 넘어갈 때 문맥전환(context switching) 발생
- Context switching
 - 일단 현재 수행 중인 태스크 상황 하에서의 시스템 상태(문맥)를 TCB(Task Control Block)이라는 특정한 자료구조에 저장하고, 다음에 새로운 태스크의 문맥을 가져와 시스템 상태를 복원한 후에 실행하는 것
 - Context switching은 overhead이기 때문에 짧을 수록 효율적 임
 - thread의 개념을 통해 이를 보완하는 방법

임베디드 OS의 주요 개념

■ Mutual exclusion

- 두개의 태스크가 동시에 하나의 공유자원에 접근하려고 할 때 한 태스크에게 자원 사용에 대한 배타적 권리를 보장하는 것
- **Critical section** : 공유자원을 **access**하는 일련의 코드부분
 - 다른 태스크에 의해서 중단되어서는 안 되는 일련의 명령 혹은 코드 블록
- 상호배제 기법
 - 인터럽트 발생을 방지
 - **Critical section**에 들어가기 전에 인터럽트를 **disable**시키고 (**CLI**) 빠져 나오면서 인터럽트를 다시 **enable** 시키는 방법 (**STI**)
 - 단일 **CPU**의 경우 단순하게 사용 가능
 - semaphore 이용
 - semaphore를 얻지 못하면 공유자원을 얻을 수 없으며 일단 semaphore를 얻으면 공유자원을 마음 놓고 쓸 수 있다.
 - 다른 태스크를 위해서 공유자원을 다 쓰면 semaphore를 풀어야 한다.
 - Semaphore가 0 이면 **waiting**한다.

임베디드 OS의 주요 개념

■ Semaphore

- 공유변수 사용
- mutual exclusion을 만들어 공유자원을 제어
- 태스크 사이의 동기화에 사용 가능
- 사용 방법
 - 공유자원에 해당하는 semaphore를 만든다.
 - 공유자원을 사용하기 직전에 해당 semaphore 얻는다.
 - 변수 값이 0이 아닌 경우 그 변수 값을 1 감소 시킨다.
 - 0이면 양수가 될 때까지 기다린다. (SLEEP상태)
 - 공유자원을 다 쓰면 그 값을 1 증가 시킨다.
- 종류
 - Binary semaphore : 공유자원의 변수가 0,1인 경우
 - Counting semaphore : 1이상의 값

임베디드 OS의 주요 개념

- Task communication
 - 태스크간에 통신하는 방법
 - Global variable을 쓰는 방법 / message passing 방법
 - Global variable
 - exclusive access를 해야 하며, ISR이 포함된 인터럽트를 disable해야 한다.
 - Task 사이에서는 인터럽트를 disable하는 방법 외에 semaphore를 사용할 수 있다.
 - message passing
 - Mailbox, queue, pipe – message의 크기에 따라 결정됨
- Task synchronization
 - 태스크 간의 동기화 기능
 - Semaphore/event flag/signal 등을 사용

임베디드 OS의 주요 개념

- **Interrupt service**
 - Asynchronous event를 CPU에 알리는 방법
 - Interrupt는 외부에서 들어오는 중요한 신호로서 시간에 민감한 경우가 있기 때문에 interrupt latency가 짧은 것이 좋음.
 - ISR 자체도 짧은 것이 좋는데 그 이유는 ISR 자체가 길어지면 interrupt nesting이 되기 쉽기 때문.
 - ISR에서는 보통 그에 상응하는 태스크 수준의 service routine을 부르고(HISR) 끝나도록 구성.
 - HISR에서는 마치 태스크처럼 존재해서 수행이 된다.

임베디드 OS의 주요 개념

■ Reentrancy code

- 인터럽트, 선점의 개념과 연관
- 하나의 함수를 여러 태스크가 동시에 수행 가능
- ‘Reentrant하다’
 - 태스크 1에서 함수 1을 수행하다가 도중에 태스크 2로 제어권이 넘어가고 이 함수를 다시 호출해도 함수 1이 제대로 동작하는 것
 - 전역변수를 사용하지 말 것
 - 전역 변수는 **shared resource**이기 때문에 **mutual exclusive** 하도록 만들어 주지 못하는 문제가 생긴다.
 - Reentrant하지 못한 **code**는 공유하지 않거나, 공유해야 하는 경우에는 **semaphore**를 쓴다거나 동일 우선 순위를 갖는 태스크 사이에서 **round-robin**을 하지 않아야 한다.

임베디드 리눅스

- 임베디드 리눅스
 - 임베디드 시스템을 위해 경량화된 리눅스
 - 임베디드 시스템에 포팅(Porting)된 리눅스
- 포팅(Porting)
 - 어떤 프로그램을 특정 플랫폼에서 동작될 수 있도록 프로그래밍하는 기술
 - 예
 - 윈도우 기반의 MSN Messenger → Linux 기반의 MSN Messenger
 - Intel processor PC 기반의 Linux → ARM processor 보드 기반의 Linux

임베디드 리눅스 특징

■ 장점

- Open Source
 - 학습 용이
- PowerPC, ARM, MIPS 등 다양한 CPU Platform 지원함
- 로열티가 없으므로 가격 경쟁력이 우수
- 사용자 층이 넓어 오류 수정이 빠르고 안정성이 우수

■ 단점

- 커널 크기
 - 상용 RTOS < Linux < 범용 OS
- 완전하지 못한 Real Time
 - 제한적 preemption
- 개발 환경의 부족
 - 일반적으로 Text 기반
- 불투명한 안정성
- GUI 환경을 개발하기 어려움
- 제품화하기 위한 솔루션 구성이 어려움
- 많은 업체들과 개발자들이 독자적으로 개발하고 있어 표준화가 어려움

임베디드 리눅스 특징

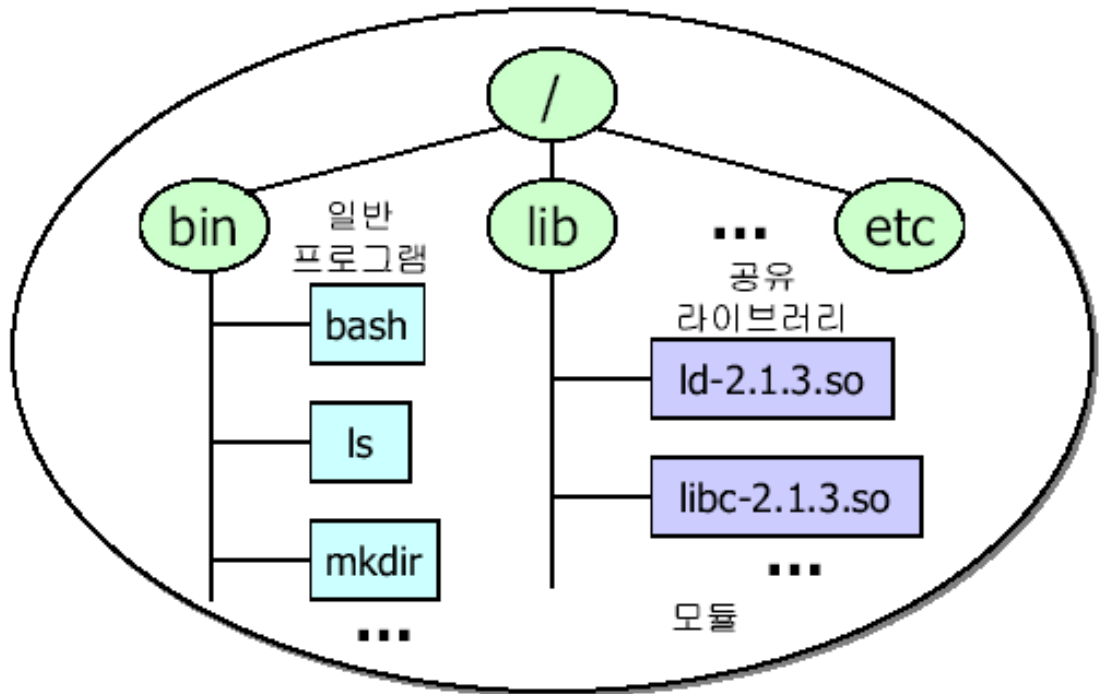
- 커널 버전의 선택
 - 임베디드 시스템의 크기를 고려
 - 필요한 기능을 고려
 - 확장성을 고려
 - 커널 버전
 - 버전 숫자 : X.Y.ZZ
 - X : 커널의 버전
 - Y : 릴리즈 번호, 홀수->개발 중, 짝수->안정된 버전
 - ZZ : Modifications, 사소한 변화를 의미
 - 최신 버전
 - 새로운 다양한 기능이 이미 추가되어 있음
 - 크기가 매우 크다는 단점이 있음

임베디드 리눅스 구성

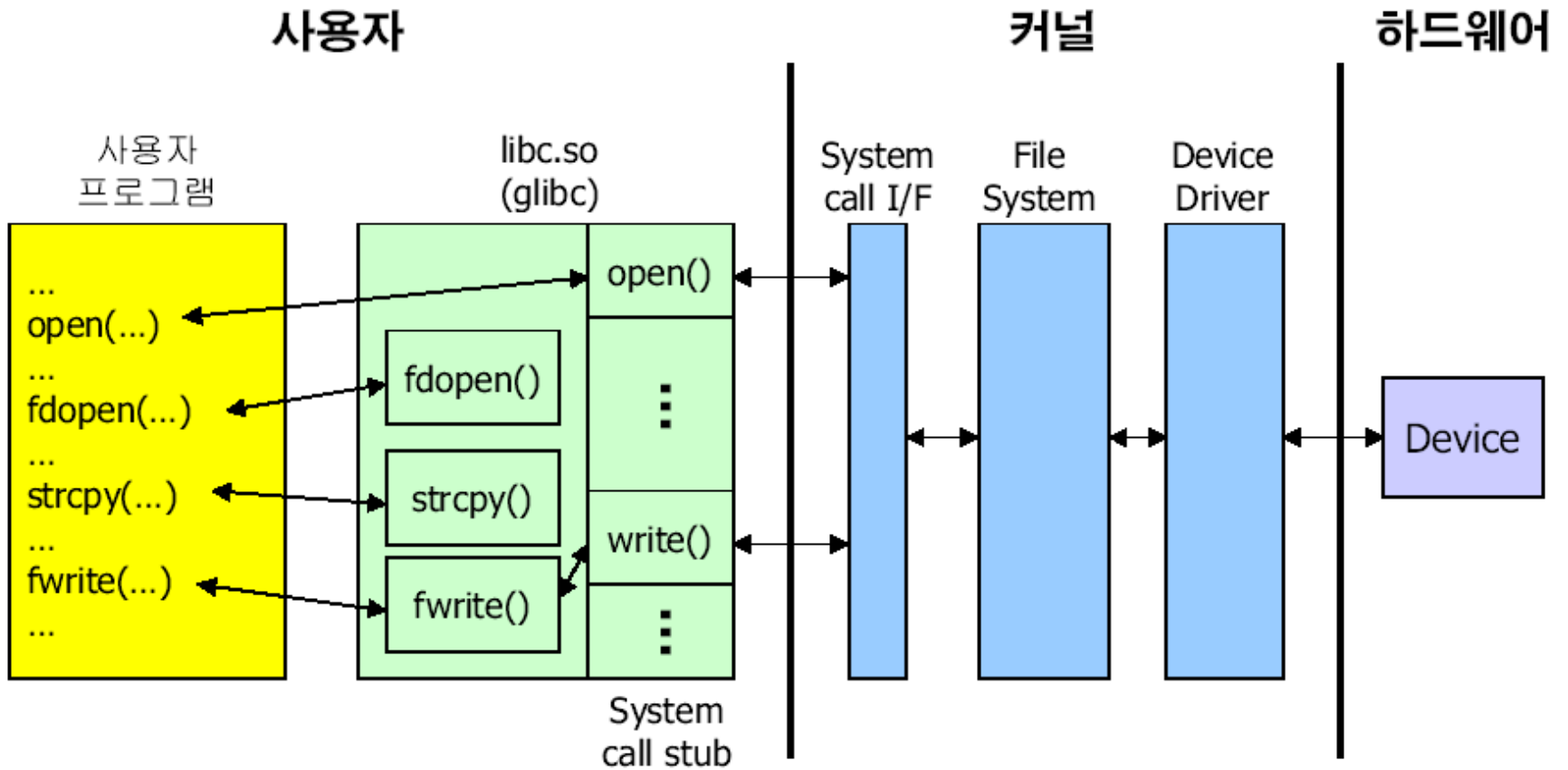
커널이미지
bziImage

+

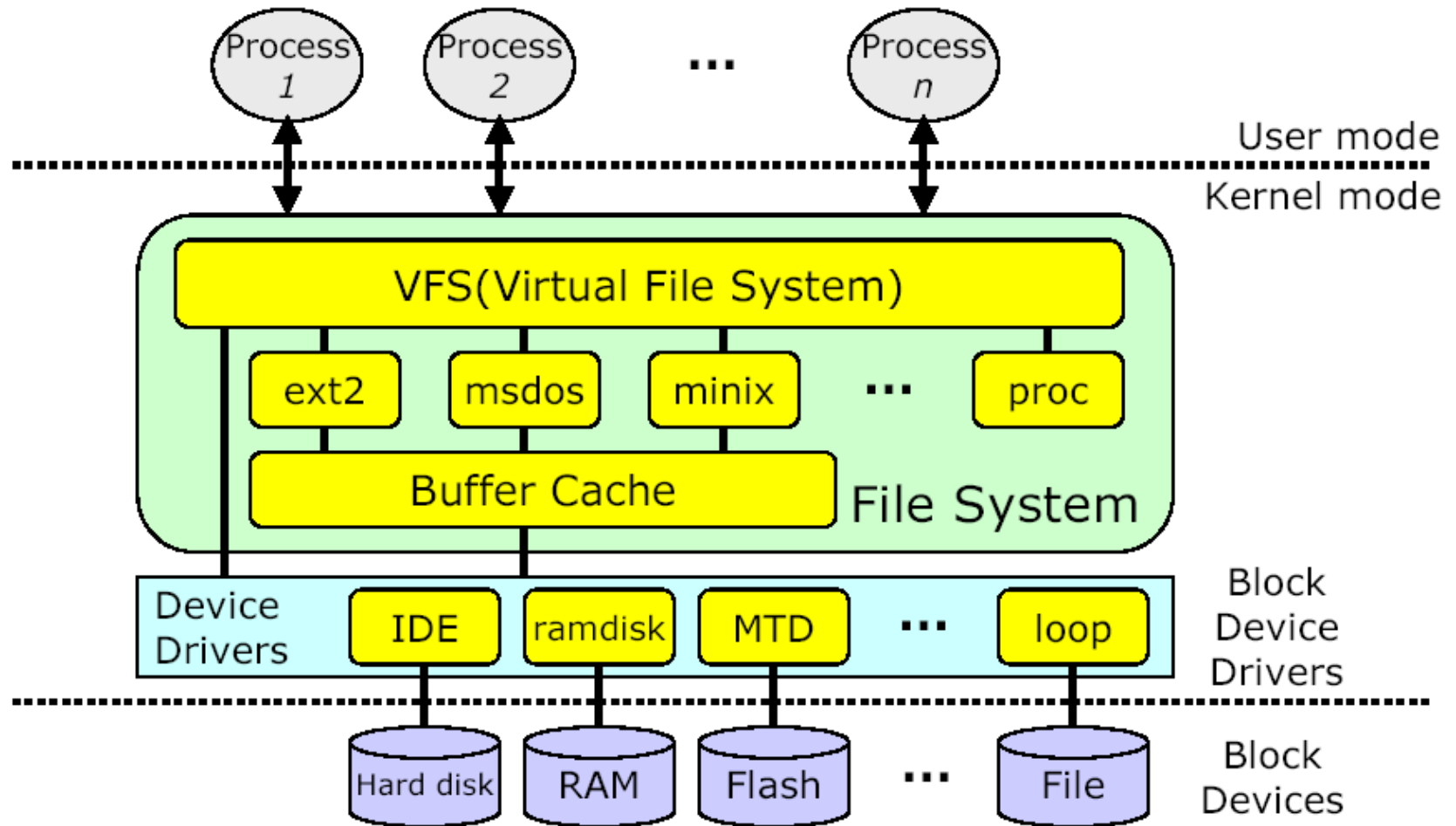
(루트) 파일 시스템



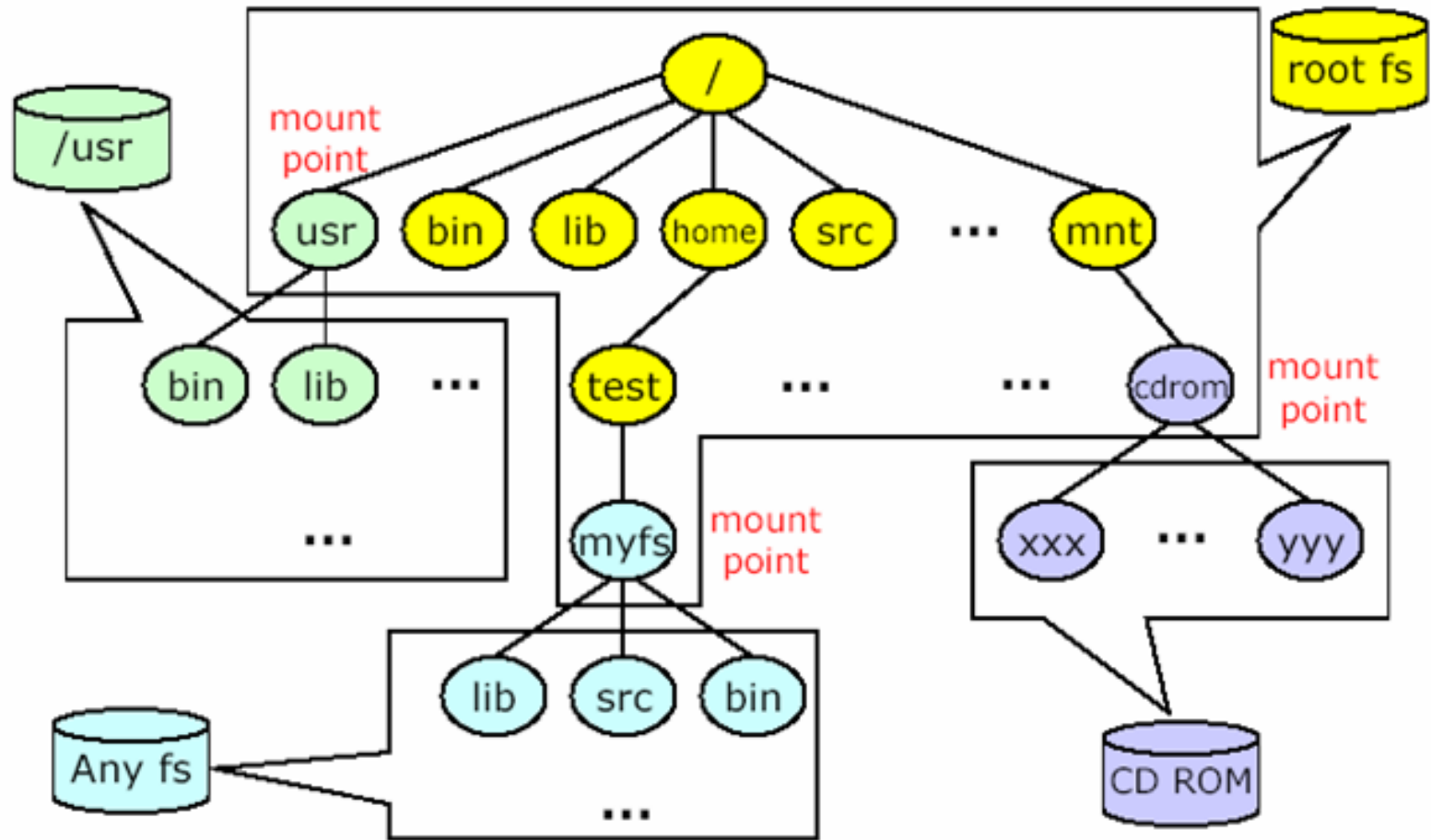
시스템 호출, 라이브러리 예



파일 시스템



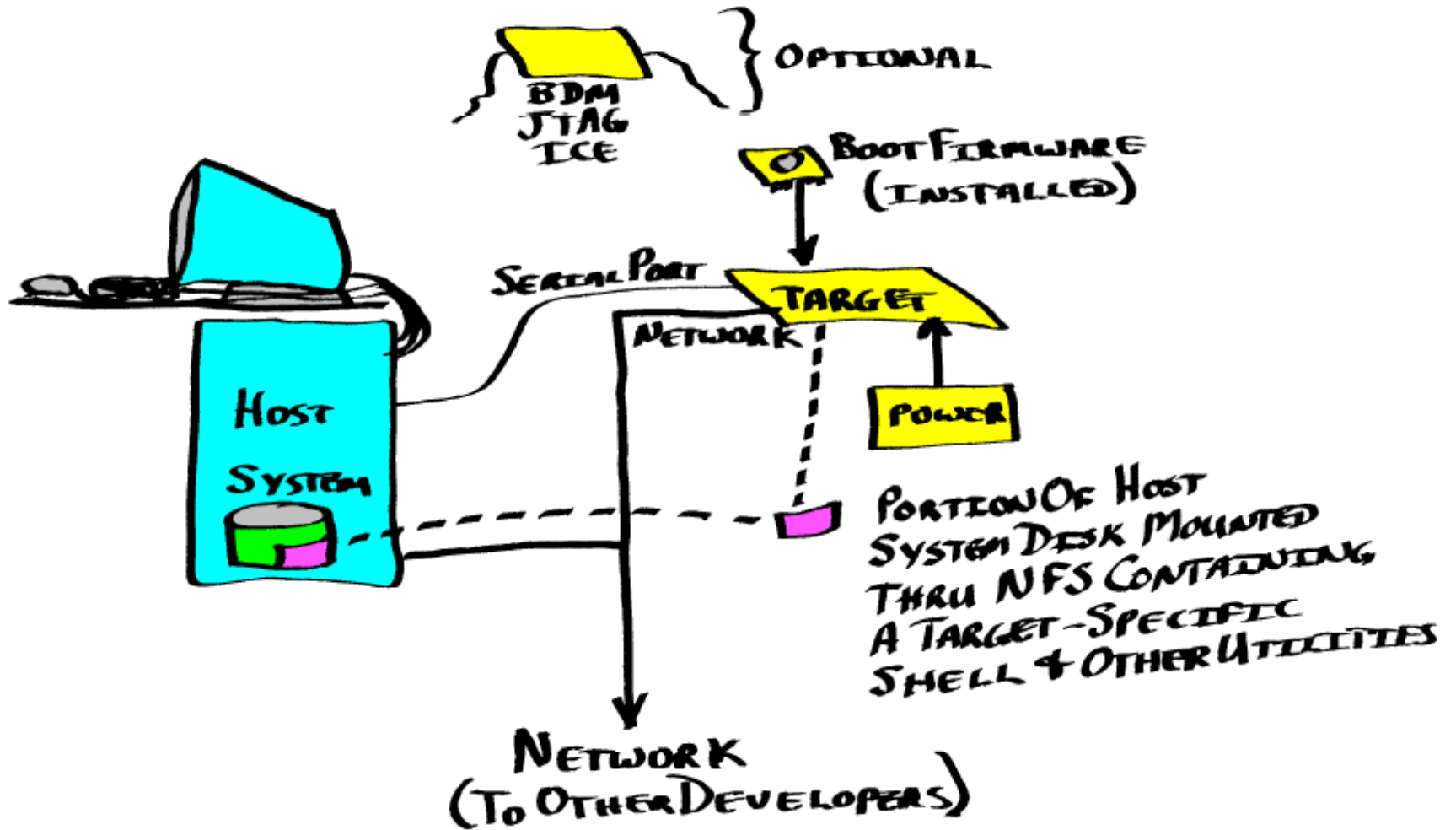
파일 시스템 (Mount)



개발 과정

- H/W 개발
- 부트로더 개발
- 커널 포팅
- 디바이스 드라이버 개발
- 응용 프로그램 개발
- 시스템 통합 (패키징)

개발 환경-1



개발환경-2

- 호스트 시스템
 - 일반 PC 또는 워크스테이션
 - 크로스 컴파일(Cross Compile) 환경이 설정됨
 - 타겟의 모든 S/W 개발
 - 시리얼 라인을 통하여 타겟 시스템의 콘솔 역할
 - TFTP, NFS 서비스 제공
- 크로스 컴파일
 - S/W가 타겟 시스템(processor)에서 돌아갈 수 있도록 하는 컴파일
 - 호스트와 타겟 시스템은 일반적으로 프로세서가 다름
- 타겟 시스템
 - 리눅스를 올릴 장비
 - 호스트 시스템에서 작성된 S/W들이 다운로드되어 실행됨