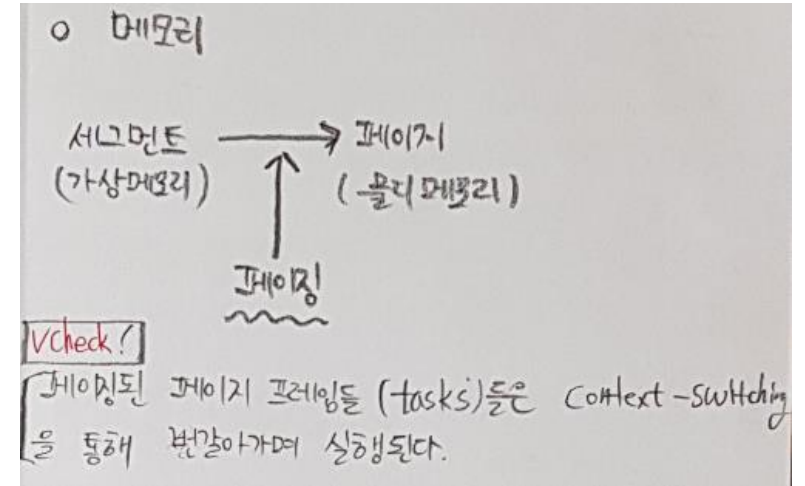
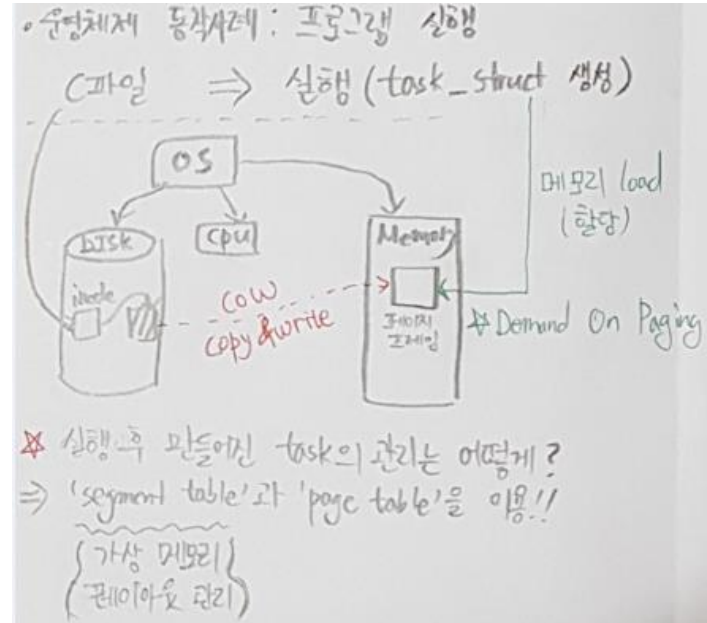
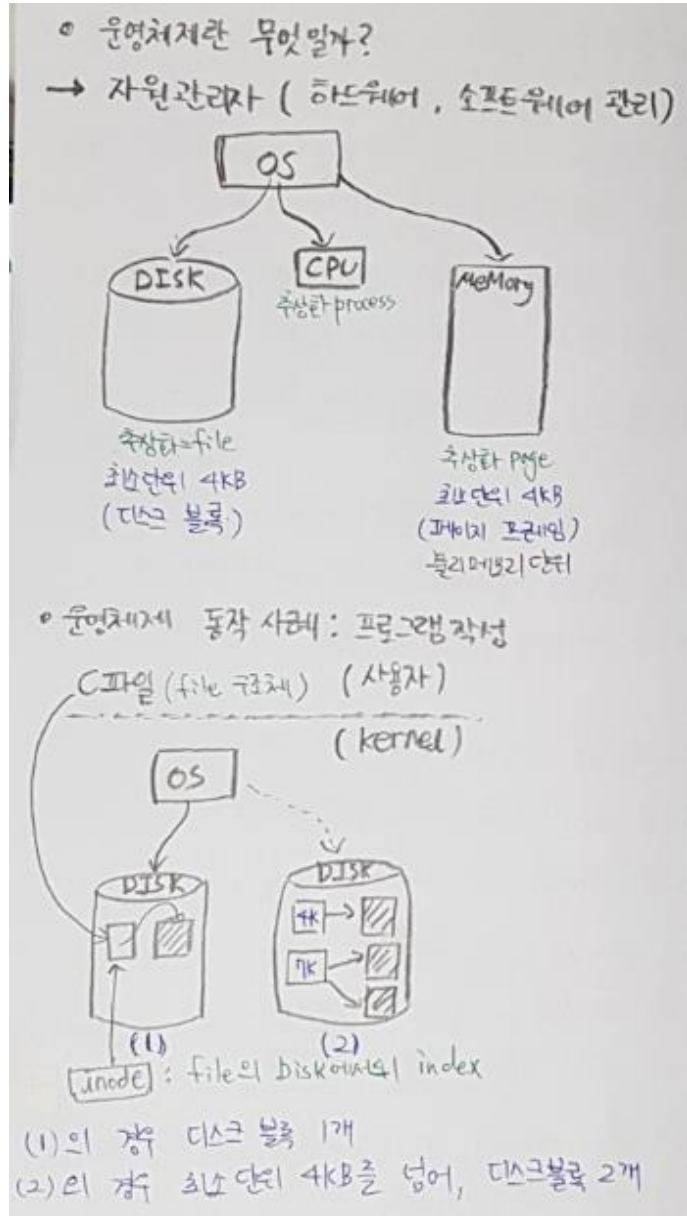


# TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 – Innova Lee(이상훈)  
[gcccompil3r@gmail.com](mailto:gcccompil3r@gmail.com)

학생 – GJ (박현우)  
[uc820@naver.com](mailto:uc820@naver.com)

# 1. 리눅스 커널 내부 구조 1 (운영체제)



# 1. 리눅스 커널 내부 구조 2 (동작 비유 1 ~ 2)

## 0 운영체제 동작비유 <1>

1) Firmware Level

⇒ OS가 없는 상태

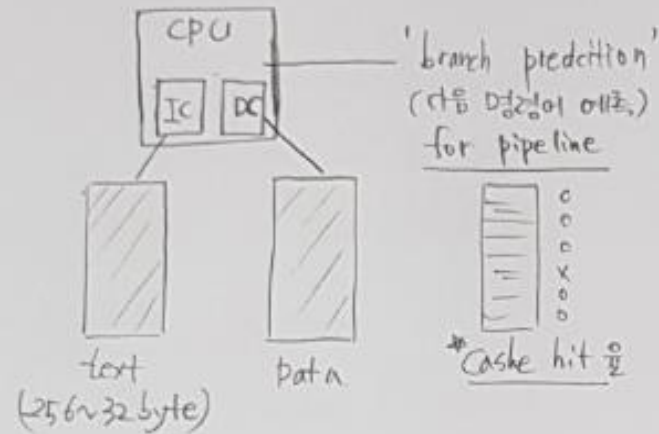
2) 다 직접 코딩하기에 비효율적

⇒ 'scheduling'도 안되고 'cache hit'를 알 수

없어서 값이 꼬인다.

★ scheduling (context switching) - timing 간

↳ CPU에 할당



3) ZImage (리눅스 커널 압축 이미지)

Root Filesystem (Linux만 있음)

⇒ (PwD, (/))  
bin usr ...

Bootloader (하드웨어 초기화)

⇒ 하드웨어 초기화 이전은 'floating 상태'

• 가상메모리 공간 확보

• linux 커널 올림

4) 시스템디자인에만 종속

⇒ 하드웨어 디자인

## ① 운영체제 관리 <5대요소>

1. Filesystem

2. Memory

3. Network

4. Device driver (장치구동 Software)

5. Process (task)

## 0 운영체제 동작비유 <2>

1) FIFO OS ⇒ RTOS는 아님

2) 정해진 time slice 만큼씩만 CPU 자원을 할당

⇒ 라운드루빈 (hybrid 방식) ↔ 정적우선순위

3) 단 하나의 태스크만 실제 running, 나머지는 대기 상태

⇒ 'single core'

4) 태스크의 속성을 고려한 time slice

우선순위

5) 높은 태스크가 선정 (preemption)을 요청

⇒ 완전히 끝까지 다 실행 후 다시 scheduling 한다.

↔ interrupt는 동작중 복귀 지점으로 한다.

# 1. 리눅스 커널 내부 구조 2 (동작 비유 3 ~ 5)

운영체제 동작비유 <3>

- 1) A-task는 B-task에게 전달해줄 사건이 생겼다.  
⇒ signal or IPC
- 2) signal 맞은 task
  - ① 기본적으로는 종료 0
  - ② 시그널 핸들러 등록시 종료x (예) signal(, 1)
- 3) 같은 시그널이 또 배달되지 않도록 해당 시그널을 블록  
⇒ sigemptyset() 함수 사용!
- 4) 시그널의 단점 ⇒ 시그널 번거로움 정보 전달 즉, 대량의 정보 전달이 안됨.
- 5) 다른 통신 기법  
⇒ Message Queue, 소켓 (원격 IPC), 파일 통신

운영체제 동작비유 <4>

- 1) 네트워크 담당하고 있는 TCP/IP  
⇒ TCP/IP 동기화  
(3-way-handshaking)
- 2) 'SYN' (synchronization)  
⇒ "있냐??" (두들김)
- 3) 'ACK' + 'SYN' ⇒ "있다" + "너도 있냐 (두들김)"  
(Acknowledge)
- 4) 'ACK' ⇒ 2~4과정으로 통신 성공
- 5) 'FIN' (Finalizing) ⇒ 통신 종료 요청
- 6) 'FIN-ACK' → 'ACK' ⇒ 최종 종료

운영체제 동작비유 <5>

- 1) 커널  $\xrightarrow{\text{명령}}$  filesystem  $\xrightarrow{\text{load}}$  a.txt
- 2) Ext2 (리눅스 파일)
- 3) 하드디스크의 제일 첫부분 슈퍼블록  
⇒ inode 관리 슈퍼블록

\* file\_operations (Read, open)

• 함수 선언되어 있음

⇒ 읽어와 할 정보가 다르기 때문이다.  
(파일 정보, 소켓 정보)

# 1. 리눅스 커널 내부 구조 3 (Linux kernel & 개념 구조)

◎ 리눅스 kernel

\* Mach는 Micro kernel (device driver)

⇒ /ego처럼 '탈부착'이 가능. (USB)

\* 리눅스는 monolithic kernel + Micro

⇒ 마이크로 커널의 장점을 활용 (Module 도입)

○ 리눅스 커널 구조 <5>

• 커널이란 운영체제의 핵심이다. <sup>프로세스</sup> CPU, <sup>메모리</sup> 메모리 그리고 <sup>장치</sup> 기타 디바이스 등의 시스템 리소스를 관리하고, 사용자 프로그램이 이를 사용할 수 있도록 한다.

↓  
'system call'

\* 디바이스 드라이버  
⇒ socket과 file의 인터페이스는 같다.

• 물리적 자원 (CPU, 메모리, 디스크, 터미널, 네트워크 및 주변 장치)

\* 추상적 자원 물리적 자원 ↔ 자원을 운영체제의 관리를 위한 개념

• 추상적 자원

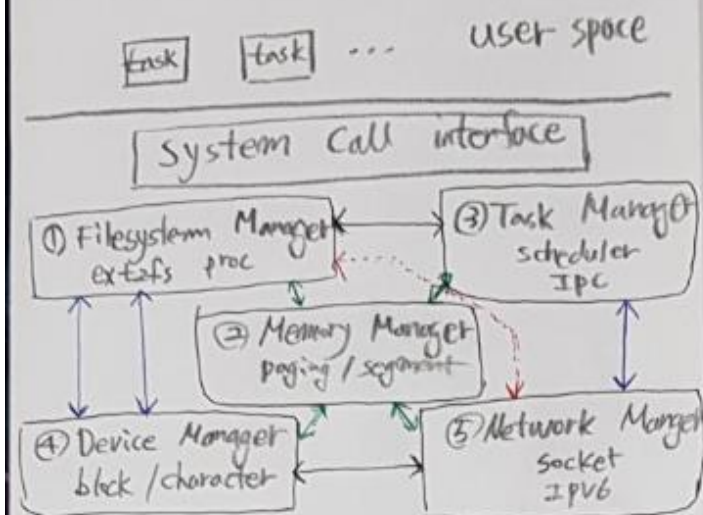
1) CPU = process (task)

2) 메모리 = segment + page (가상 + 물리)

3) 디스크 = 파일

4) 네트워크 = 프로토콜, 패킷

○ 리눅스 커널 개념 구조 <6>



• case: task 실행 (a.out)

1) ① (Disk 관리)에서 소스 코드 파일 검색 (실행전)

2) Demand on paging으로 ②에 복사  
(프로그램 동작시 메모리 필요)

3) ③에서 실행된 프로그램들을 관리. (실행 후)

④ Dram, Hard Disk 관리

⑤ lan card 등등..



# 1. 리눅스 커널 내부 구조 4 (Linux kernel 역할 & 태스크 관리)

③ 역할 설명 ⇒ 운영체제는 시스템 호출을 통해 (서비스) 태스크가 자원을 사용할 수 있게 해주는 자원관리자

③ 1) 태스크 생성 ⇒ fork, pthread\_create  
2) 실행 ⇒ exec  
3) 상태 전이 (state-transition)  
4) 스케줄링 (time slice)  
5) 시그널 처리  
6) 프로세스간 통신 (IPC)

② 1) 물리 메모리 관리  
2) 가상 메모리 관리 (stack, heap, data, text)  
3) 세그멘테이션, 페이징, <sup>DRAM</sup> 페이지 부재 결함 처리

Diagram: User space (segmentation fault) or kernel space (page fault) → page fault handling → 페이지 할당 ⇒ 재접근 ⇒ '할당 성공'

○ 태스크 관리 < 7 >

- 리눅스는 태스크를 통해 다양한 생명과 변환을 제공한다. life cycle exec
- 프로세스와 스레드 ⇒ task\_struct
  - pid = tgid (프로세스)
  - tgid는 같으나 pid는 다름. (스레드)
- 32bit CPU
  - 0 ~ 3GB - 사용자 3
  - 3 ~ 4GB - 커널 1
- 64bit CPU
  - $2^{64} = 16EB$  128TB - 사용자 나머지 - 커널
- region (segment) 가상메모리
  - stack / heap / data / text
  - 지역변수, 동적할당, 전역변수, 명령어, 문자, 함수표현어

# 1. 리눅스 커널 내부 구조 5

## Kernel/ linux

- arch
  - > cpu 모델에 따라서 바뀌어야 할 코드 (context switching code)
- bootloader
  - > 하드웨어적 제어
- mm
  - > 페이징 수행하는 연산 레지스터
- blackfin
  - > 오디오 관련 제어
- openrisc
  - > 나만의 cpu설계가 가능해진다.
- x86
  - > intel cpu
- c6x
  - > TI, DSP
- ia64
  - > graphic card 설계를 위한 공부
- kernel
  - > 커널과 관련된 software
- drivers
  - > device driver와 관련된 정보  
( pci , armba, fmc(초고속 무선 통신 interface), gpio, gpu, power block, bluetooth, tty(터미널), mtd (nand flash), regulator (정전압) , iio(오실로스코프), dma(메모리에 직접 접근), w1( 무선), phy(유선) , net )
- net
  - > hamradio (무선 통신), can (통신), irda (적외선)
- wireless
  - > ti
- Avr32
  - > cortex A
- Nios2
  - > 알테라 fpga(인텔 인수), x86에서 밀고 있는 architecture
- arm (하위 호환을 하지 않음 – 종류가 많은 이유)
  - Exynos (삼성), omap (TI), zynq (자이링스)
  - s3c24xx (삼성)
  - keystone (TI DSP), 2 (레이더용)
  - ipc18xx, ipc 32xx (NXP) (차량용)
  - bcm (라즈베리파이)
  - davinci (TI) (블랙박스, 스마트 tv, CCTV)
  - stm32 (차량용)
  - tegra (NVIDIA)

# 1. 리눅스 커널 내부 구조 5

## Kernel/ linux

- include, lib  
-> library
- block  
-> block driver device에 관한 것만 있음 (block은 규모가 크다)  
-> ssd같은 것을 제어하는 데에 필요한 알고리즘이 있다.
- init  
-> 커널의 시작 코드 (vi -t start\_kernel -> 9 -> kernel 초기화)
- usr  
-> user와 관계된 정보
- Documentation  
-> kernel과 관계된 문서들
- ipc  
-> 세마포어, 스핀락, mutex
- scripts  
-> 커널 구동용 스크립트
- virt  
-> cloud service를 위한 기술
- microblaze  
-> fpga에서 구동되는 가상의 cpu
- Sparc  
-> 슈퍼 컴퓨터용 cpu
- Powerpc  
-> 자동차
- mm (메모리 매니지먼트)  
-> 가상 메모리, 물리 메모리 관련된 것들이 있다.
- firmware  
-> 통신을 하면서 주고 받기 위함.
- net  
-> network
- sound  
-> VOLT, ALSA driver
- video  
-> 영상 스트리밍 (V4L2)
- crypto(암호화, 복호화 알고리즘) , security, certs
- Fs (파일 시스템)  
-> open, read, write 와 관련된 것들



# 1. 리눅스 커널 내부 구조 5

Kernel/ linux

- 실행파일 정보
- > readelf -h a.out
- > elf = 리눅스 실행 파일 포맷
- > dwarf = 디버깅 파일 포맷
- > pe = 윈도우 실행 파일 포맷