SOSCON

삼성 오픈소스 컨퍼런스

SAMSUNG OPEN SOURCE CONFERENCE

OPEN YOUR UNIVERSE WITH SOSCON

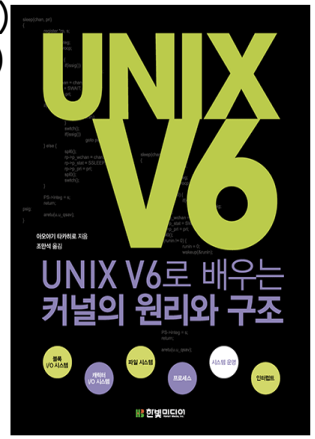
Linux Kernel Boot Process

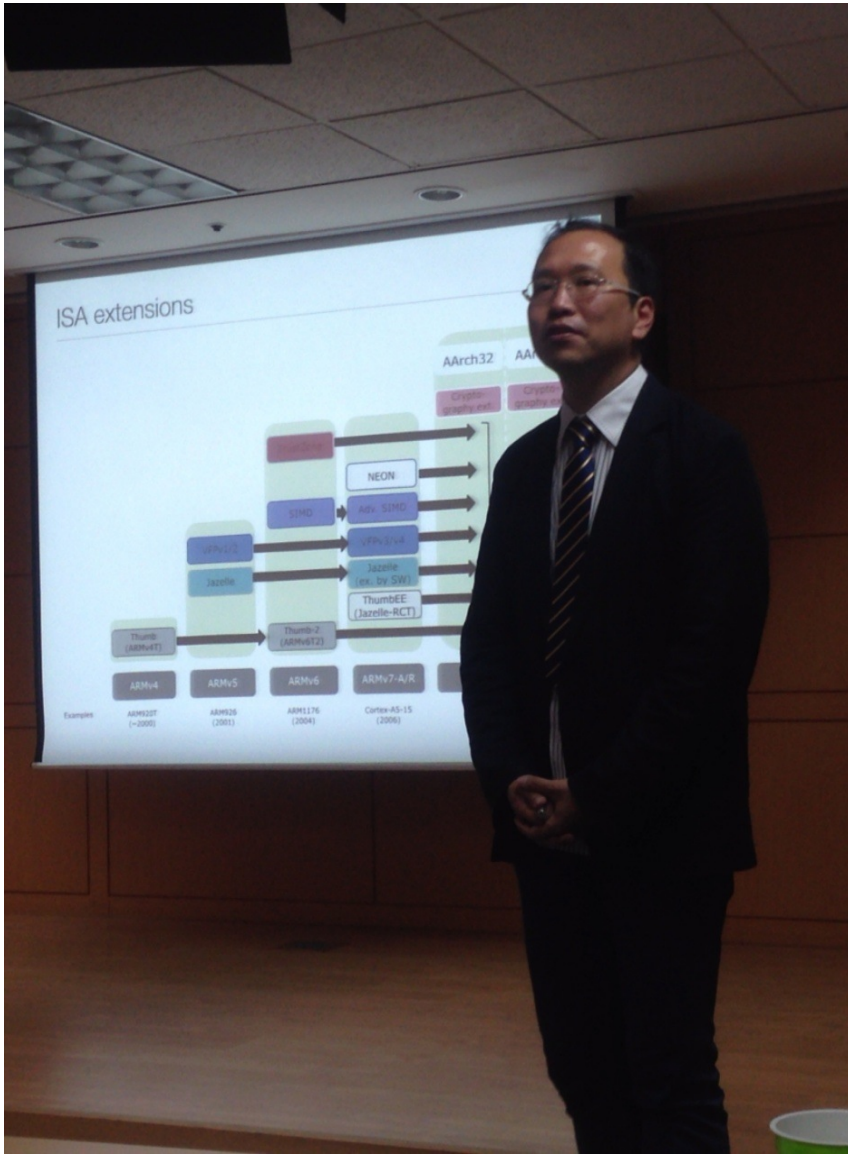
Open Frontier Lab.

Mario Cho (조만석, hephaex@gmail.com)

2015.Oct. 28(Wed.)

- Open Source Software Engineer
  - HPC (High Performance Computing) for Human Brain Mapping
  - 3D Medical Image Reconstruction (SART C T)
  - Enterprise Storage
  - NFV (Network Function Virtualization)
- Open Frontier Lab.
- Open Source
  - Linux Kernel (ARM, x86, ppc)
  - LLVM (x86, ARM, custom)
  - OpenStack : Orchestration (heat)
  - SDN (OpenDaylight, OVS, DPDK)
  - OPNFV: (Genesis,DPACC,Doctor)
- Technical Book





공개소프트웨어 개발자 센터 (http://devlab.oss.kr)

공개소프트웨어 개발자 센터 (Open Frontier Lab.)

공개 소프트웨어 개발자 지원
- http://devlab.oss.kr

1. 공개SW개발자센터 운영
2. 공개SW 커미터 후보 선발/육성
3. 우수공개SW 커뮤니티 선발, 지원
4. 공개SW 커뮤니티 행사, 장소 지원

공개소프트웨어 개발자센터
OPEN FRONTIER LAB

Open Frontier Lab. & 공개 SW 커뮤니티 지원

오픈프론티어

멘토링 프로그램

월 10시간/1인당 제공,
43명의 멘토 활동중

공개SW 프로젝트 운영지원

오픈프론티어 37명 선발, 36개 프로젝트 운영
(연구장려금 지원)

개발공간 및 장비 지원

개인별 전용좌석, 협업공간 제공, 장비구입비 지원

해외 컨퍼런스 참여 지원

해외 컨퍼런스 참여 지원






공개SW커뮤니티

우수 공개SW커뮤니티 선발,지원

연 5~6개 선발, 개발 및 운영 비지원

세미나, 컨퍼런스 개최 지원

행사 비용 지원

개발 및 협업 공간 제공

모임장소 상시지원

가상개발환경 지원

개발용 서버 사용 환경 지원





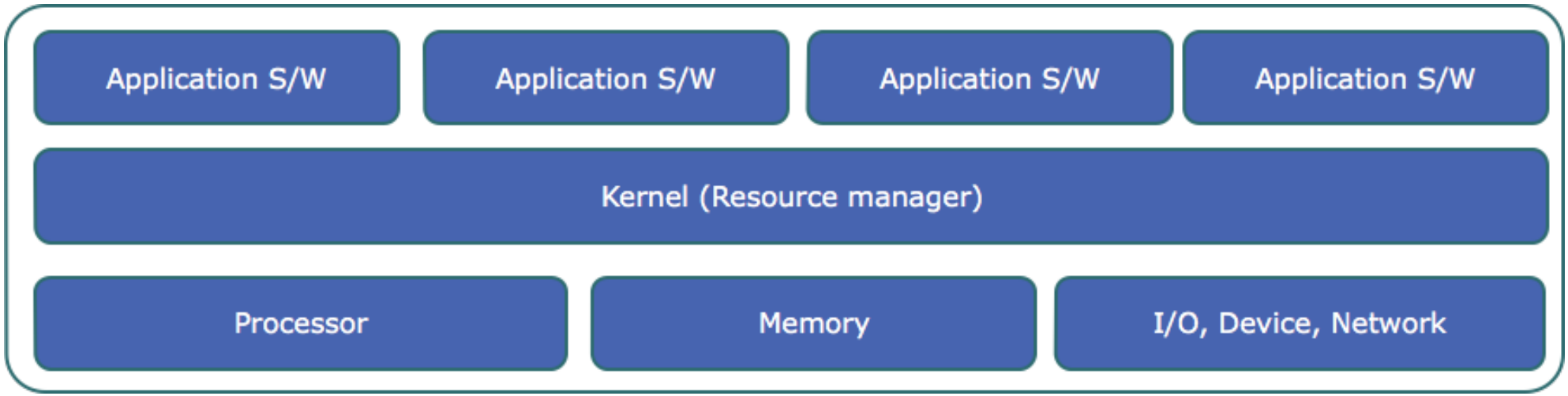
운영체제란?

운영체계

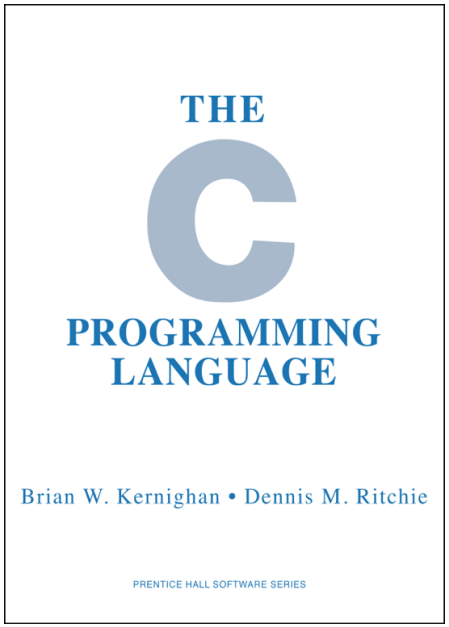
- 커널은 운영체제의 핵심으로 컴퓨터 시스템을 사용하는데 꼭 필요한 기능을 제공.

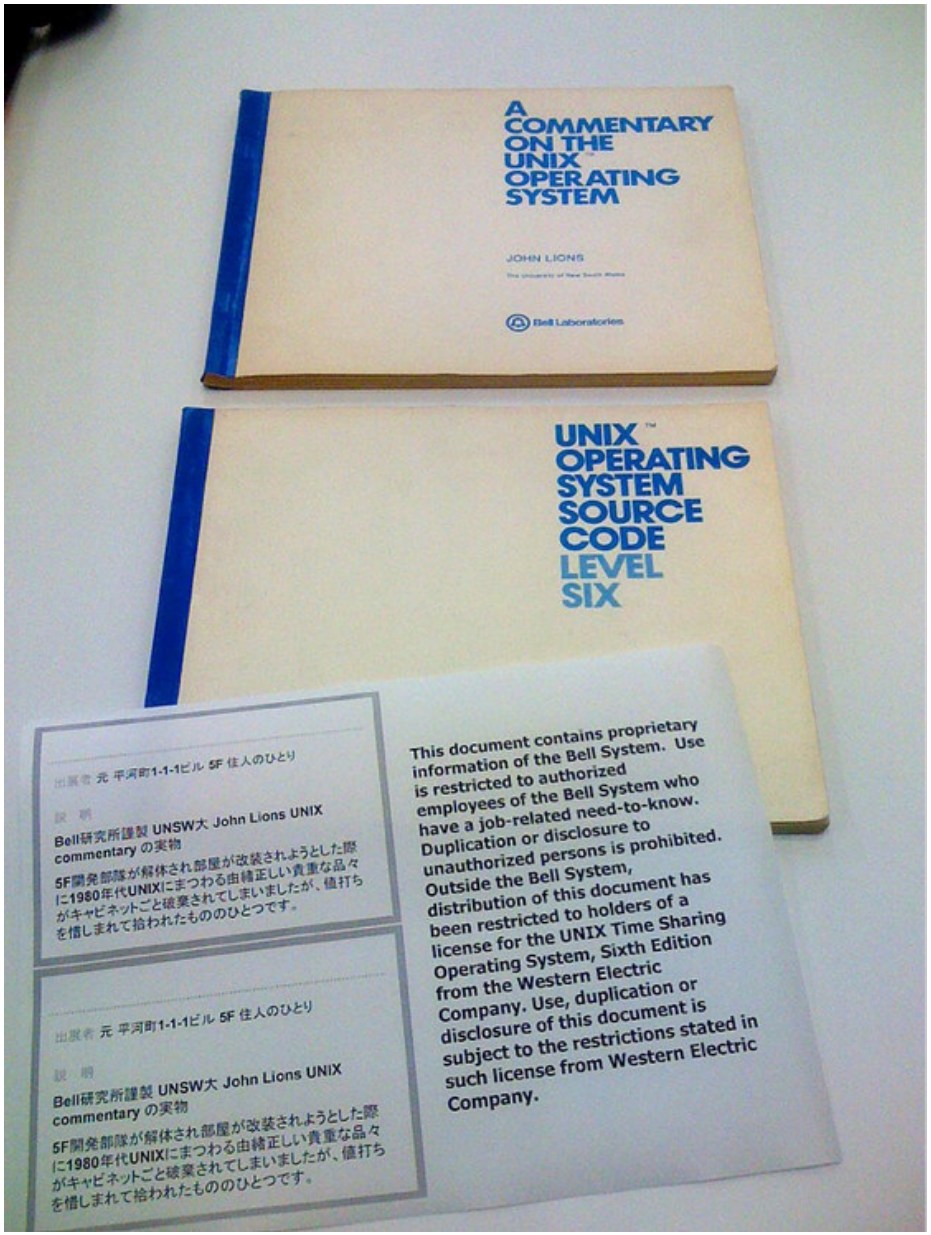
운영체계의 주 역활

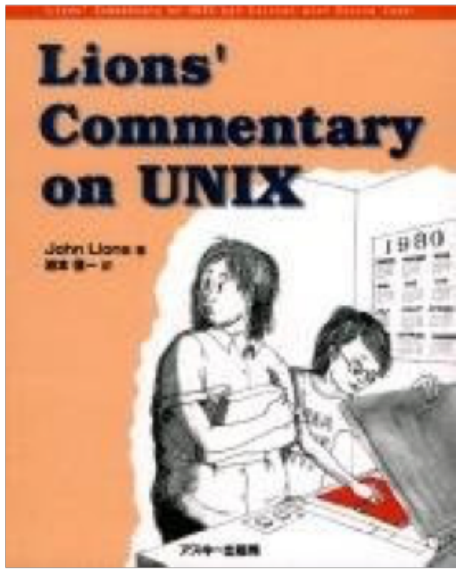
- 실행 프로그램(프로세스Process)관리
- 메모리 관리 (Memory Management)
- 파일 시스템 (File System)
- 파일과 주변 장치를 위한 I/O

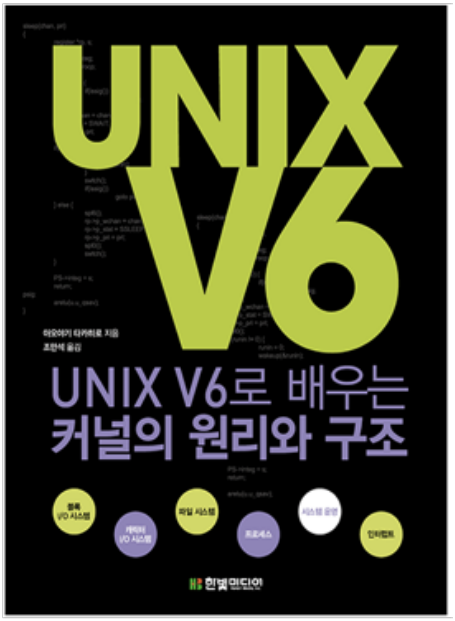


K&R (Ken Tompson & Dennis Richie)

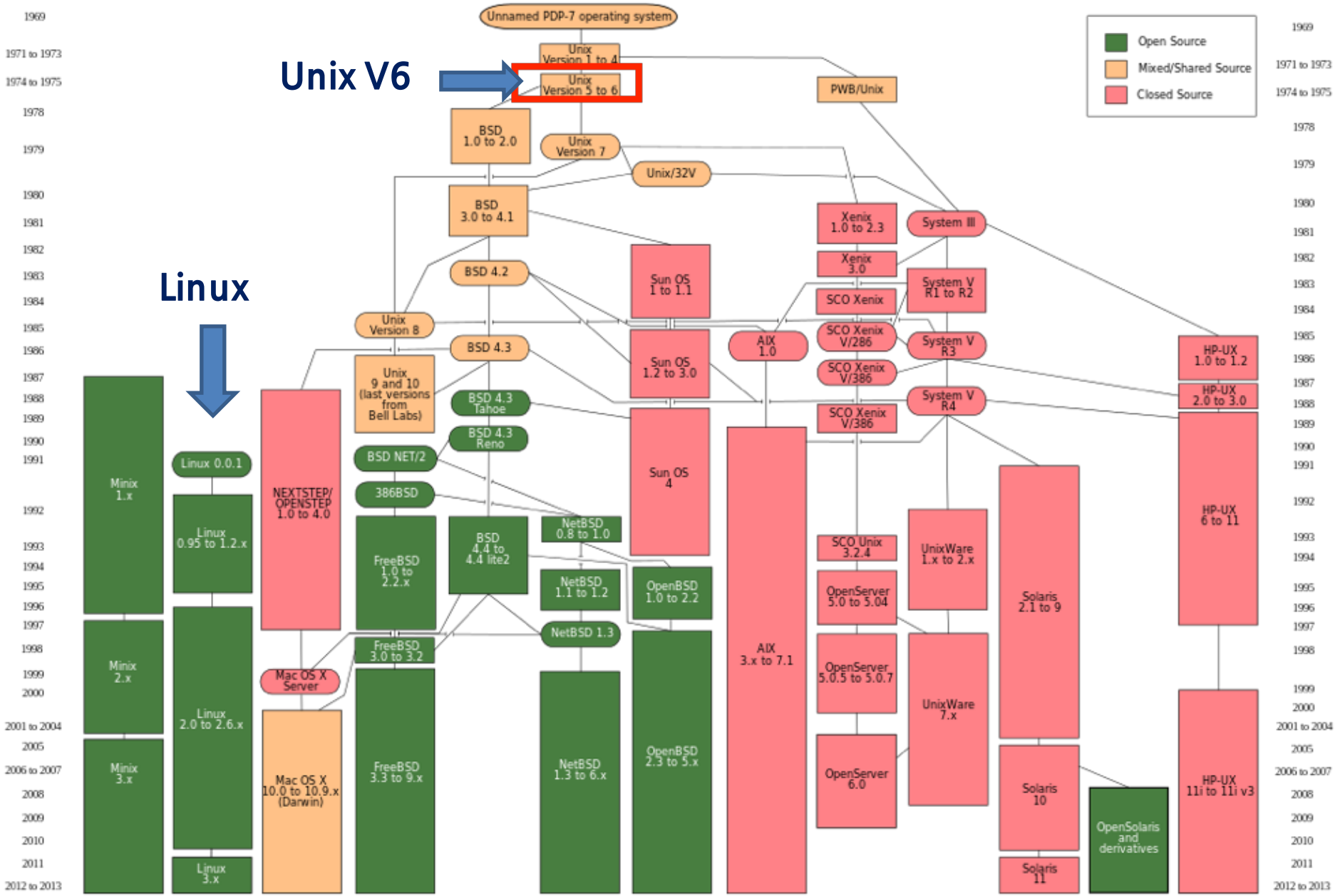



A COMMENTARY ON THE UNIX OPERATING SYSTEM
JOHN LIONS
Bell Laboratories
UNIX OPERATING SYSTEM SOURCE CODE LEVEL SIX
Bell研究所謹製 UNSW大 John Lions UNIX commentary の実物
5F開発部隊が解体され部屋が改装されようとした際に1980年代UNIXにまつわる由緒正しい貴重な品々がキャビネットごと破棄されてしまいましたが、値打ちを悟しまれて拾われたもののひとつです。
This document contains proprietary information of the Bell System. Use is restricted to authorized employees of the Bell System who have a job-related need-to-know. Duplication or disclosure to unauthorized persons is prohibited. Outside the Bell System, distribution of this document has been restricted to holders of a license for the UNIX Time Sharing Operating System, Sixth Edition from the Western Electric Company. Use, duplication or disclosure of this document is subject to the restrictions stated in such license from Western Electric Company.


Lions' Commentary on UNIX
John Lions


UNIX V6
UNIX V6로 배우는 커널의 원리와 구조
한빛미디어

Open Source OS From Unix V6 To Linux



운영체계

- Apple Mac OS system 1.x : 1984.x
- MS Windows 1.x : 1985.x
- Linux : 1991.8.26

comp.os.minix ›

What would you like to see most in minix?

199 posts by 183 authors G+1 Previous Page 1 Next

Linus Benedict Torvalds 8/26/91

Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).

I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)

Linus (torv...@kruuna.helsinki.fi)

PS. Yes - it's free of any minix code, and it has a multi-threaded fs. It is NOT protable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-(.

Click here to Reply

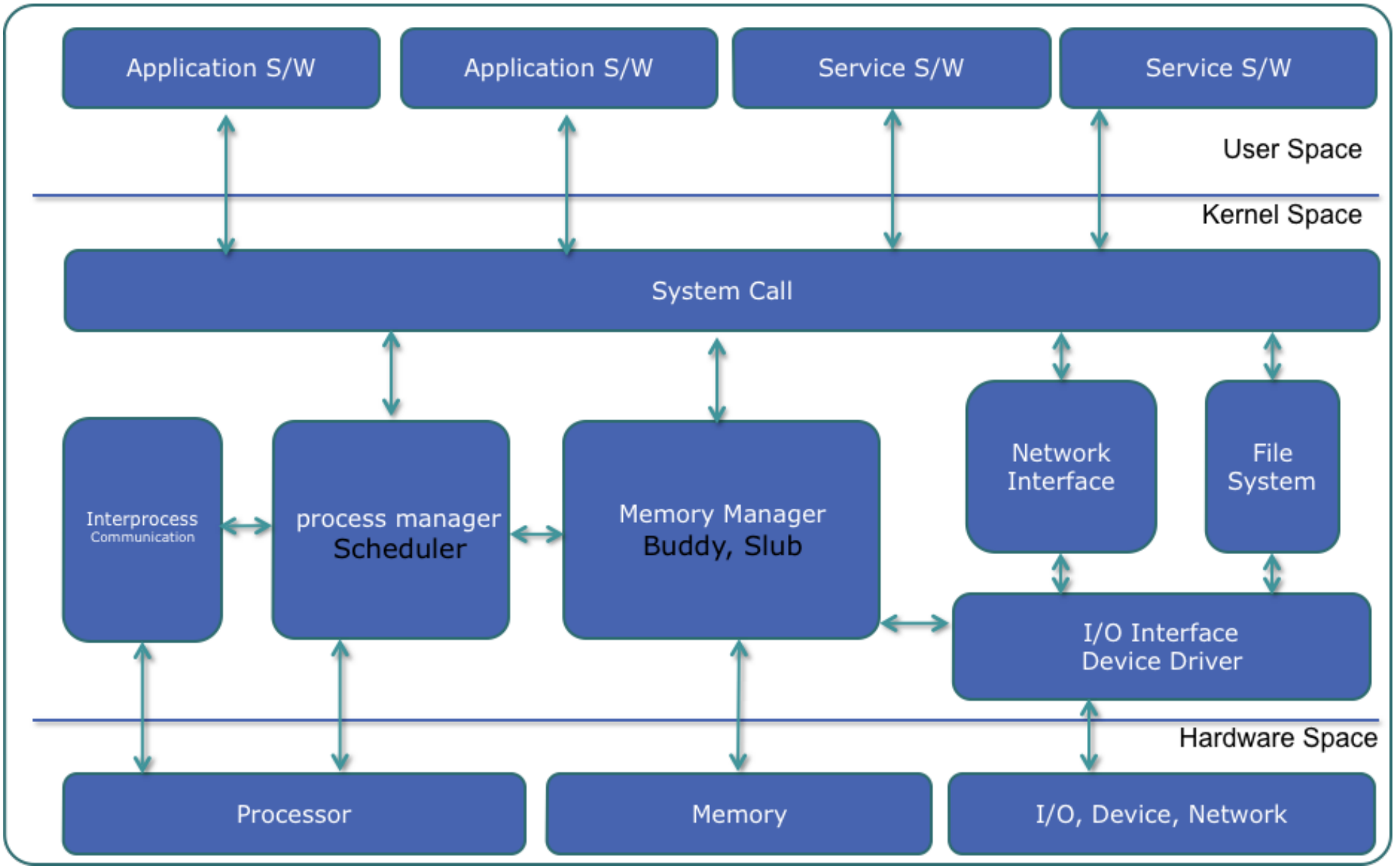
Jyrki Kuoppala 8/26/91

In article <1991Aug25....@klaava.Helsinki.FI>, torvalds@klaava (Linus Benedict Torvalds) writes:
>I've currently ported bash(1.08) and gcc(1.40), and things seem to work.
>This implies that I'll get something practical within a few months, and
>I'd like to know what features most people would want. Any suggestions
>are welcome, but I won't promise I'll implement them :-)

Tell us more! Does it need a MMU?

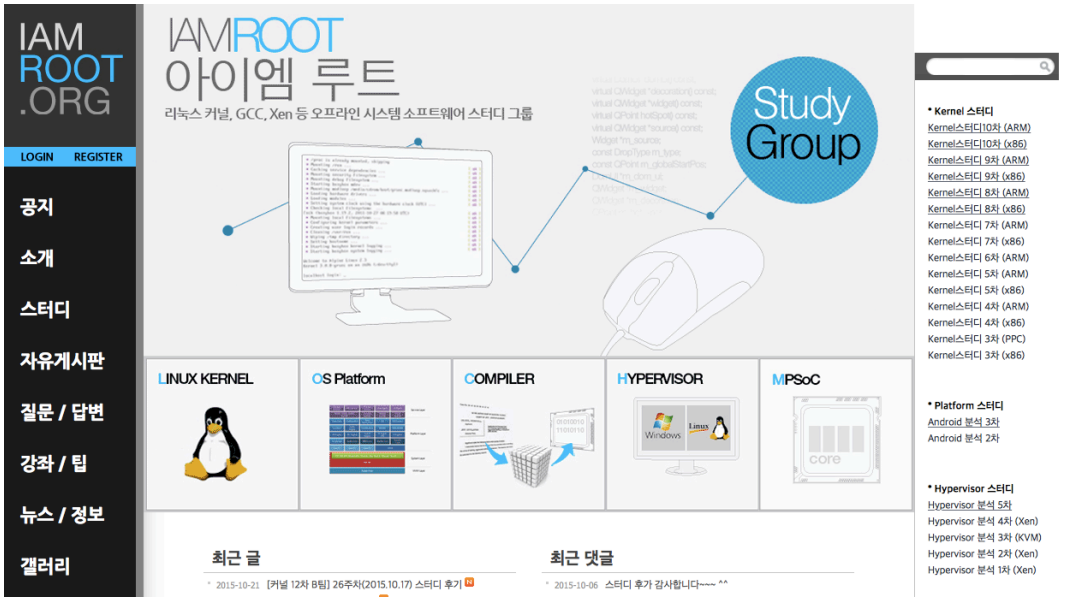
>PS. Yes - it's free of any minix code, and it has a multi-threaded fs.
>It is NOT protable (uses 386 task switching etc)

Linux Kernel Architecture

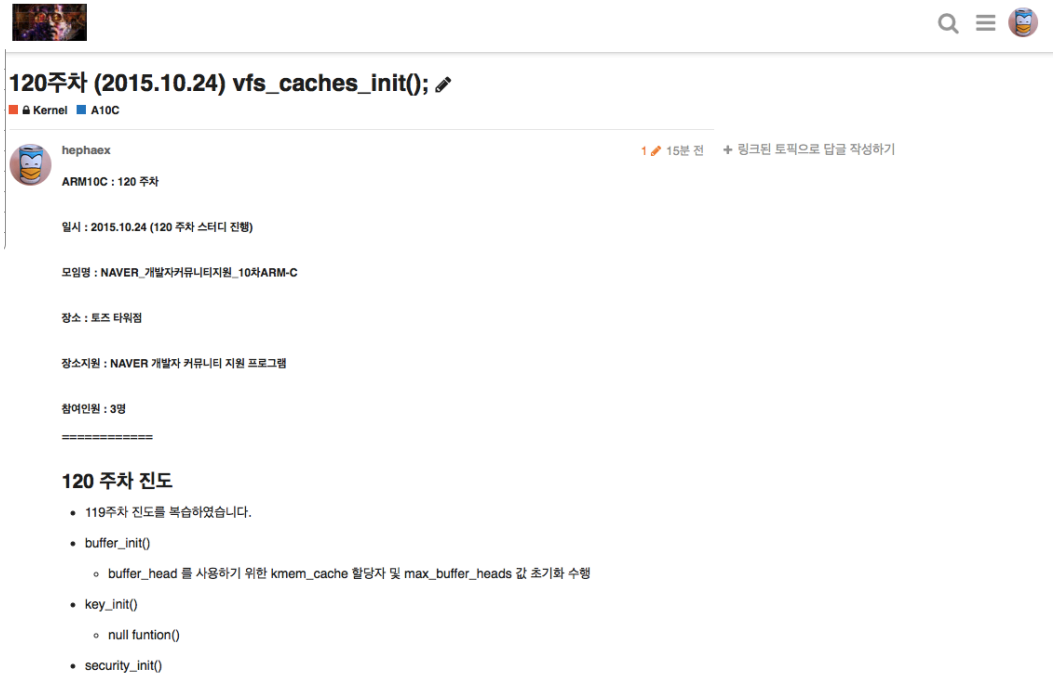


리눅스 커널 커뮤니티

http://iamroot.org
- Linux 커널 스터디



http://kernelstudy.net
- ARM, x86, 리눅스 커널 분석
- LLVM 컴파일러.
- SDN/NFV



120주차 (2015.10.24) vfs_caches_init();

Kernel A10C

hephaex 1 15분 전 + 링크된 토픽으로 답글 작성하기

ARM10C : 120 주차

일시 : 2015.10.24 (120 주차 스터디 진행)

모임명 : NAVER_개발자커뮤니티지원_10차ARM-C

장소 : 토즈 타워점

장소지원 : NAVER 개발자 커뮤니티 지원 프로그램

참여인원 : 3명

==========

120 주차 진도

- 119주차 진도를 복습하였습니다.
- buffer_init()
  - buffer_head 를 사용하기 위한 kmem_cache 할당자 및 max_buffer_heads 값 초기화 수행
- key_init()
  - null funtion()
- security_init()

http://kernel.bz
- 커널 연구회

Not logged in
Log in now
Create an account
Subscribe to LWN

Weekly edition
Current [$]
Inactive openSUSE members • Visible-light networks • Recent kernel security holes • Simple wait queues • Rich ACLs • Upcoming GStreamer work • ...
Previous
Simpler playback for

Weekly edition | Kernel | Security | Distributions | Contact Us | Search
Archives | Calendar | Subscribe | Write for LWN | LWN.net FAQ | Sponsors

Welcome to LWN.net

LWN featured content

[$] Rich access control lists

[Kernel] Posted Oct 20, 2015 20:19 UTC (Tue) by corbet

Access control lists (ACLs) can implement finer-grained access permissions for files than the traditional Unix mode bits. Linux has ACL support, but the POSIX ACLs supported by Linux now have been showing their age for a while. POSIX ACLs may soon be superseded by a more capable mechanism known as RichACLs. Click below (subscribers only) for a look at RichACLs and what they bring to Linux.

Full Story (comments: 74)

Permissive licenses, community, and copyleft

[Front] Posted Oct 14, 2015 18:46 UTC (Wed) by n8willis

On the final day of LinuxCon Europe 2015, HP's Chief Technology Officer Martin Fink delivered a bold keynote about software licensing. Fink recapped the negative effects of license proliferation and addressed projects that use their choice of license as hostile act against the competition. He then ended the session with an extended appeal to move the open-source software industry away from permissive licenses like Apache 2.0 and toward copyleft licenses like the GPL. Not doing so, he said, puts the FOSS community

What is LWN.net?

LWN.net is a reader-supported news site dedicated to producing the best coverage from within the Linux and free software development communities. See the LWN FAQ for more information, and please consider subscribing to gain full access and support our activities.

Current news

Kernel prepatch 4.3-rc7

[Kernel] Posted Oct 25, 2015 6:09 UTC (Sun) by corbet

The 4.3-rc7 kernel prepatch is out. "So it may still be Saturday at home, but with the Kernel Summit in Korea coming up, I'm ahead of the curve in a +0900 timezone, and it's Sunday here. So it's release day." This looks to be the final prepatch, with 4.3 likely to come out on November 1.

Comments (none posted)

Coghlan: 27 languages to improve your Python

[Development] Posted Oct 25, 2015 1:00 UTC (Sun) by corbet

Python language developer Nick Coghlan has posted a survey of 27 languages that, he thinks, have lessons for Python. "One of the things we do as part of the Python core development process is to look at features we appreciate having available

커널을 공부하기 위해서 필요한 도구

- Note PC or Desktop or Cloud connection

- Editor : Vim, Emacs, ???
- Tools: ctags, gtags, cscope, …

- 참고 서적
  - AP Manual
    - ARM Cortex A Series Programmer Guide
    - ARM Cortex A15 Technical Reference Manual
    - kernel 분석에 필요한 AP user manual (가장 중요하지만, , ,)
  - DTB (Device Tree Blob)
    - Power ePAPR_APPROVED_v1.1
  - Kernel 책들
    - UNIX V6 로 배우는 커널의 원리와 구조 (운영체제에 대한 근본원리)
    - 코드로 알아보는 ARM 리눅스 커널 (2.6.x 커널 기준)
    - 리눅스 커널 해설 (2.6.x 커널)
- 어셈블러
    - GNU : GNU Assembler Directive
    - GNU : GNU Assembler Manual
    - GNU : GNU Linker Manual

커널을 공부를 시작할 때 팁

커널은 코드의 양이 크기 때문에 (약 1500만 라인) 많은 지식을 필요로 하고, 많은 시간을 걸쳐서 하는 개발자 마라톤과 같은 부분이 있으므로 꾸준히 공부할 동료가 필요합니다.

1. 함께 공부할 친구를 찾는다. (개발자 커뮤니티)
2. 스터디에서는 존칭을 사용
3. 스터디에 부담 주는 행동 자제. (한사람이 무언가 해오게 하는 행동들.)
4. 커널에 맞는 구하기 쉬운 디바이스 선정 (라즈베리 파이 2)
5. 꾸준히 공부 할 수 있는 공간
6. 스터디 내용을 공유할 수있는 인터넷 공간 (커뮤니티 게시판, 공유 문서)

커널 분석 목표 설정

리눅스 커널 코드 분석에 들어가기 전에 어떤 것을 분석할지 목표를 분명히 합니다.
아울러 target device도 함께 정합니다. (추천 라즈베리 파이2)

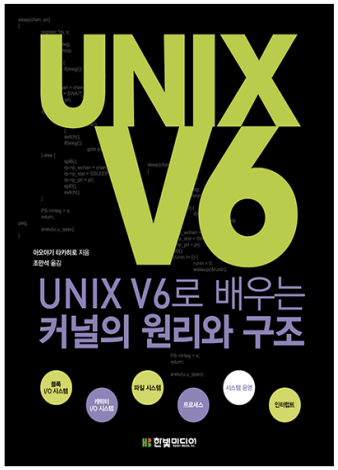
- kernel boot process
  - boot loader에서 부터 시작해서 하드웨어 초기화, 메모리 초기화, 인터럽트 설정,
  - init process 에 이르는 일련의 과정을 분석.
  - 내용이 많지만, 하드웨어, 커널에서 사용하는 주요 메크로, 알고리즘을 배울 수 있음.

- mm_init() 분석
  - start_kernel()에 보면 메모리를 초기화 하는 mm_init()이 있습니다.
  - 리눅스 커널은 buddy와 slub을 이용해서 메모리를 관리합니다.
  - buddy와 slub이 어떻게 초기화 되고 커널에 메모리 초기호/할당/제거 를 분석

- Device Driver 분석
  - GPIO 디바이스를 분석하고 라즈베리 파이2를 이용해서 실험
  - network 디바이스를 분석해서 network stack을 심화

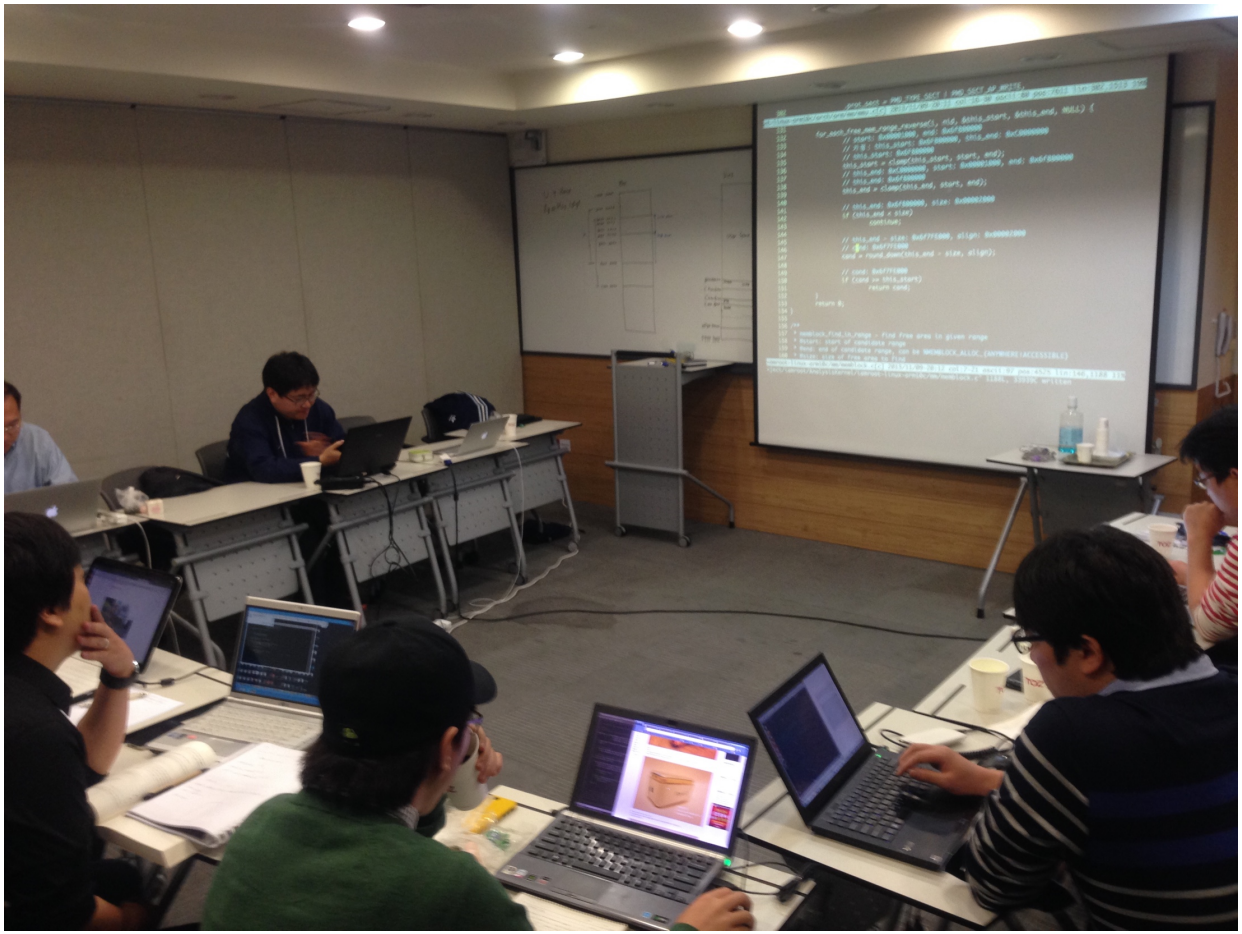
- 그외 스케쥴러 분석 등등.

1. 각자 다른 사람들이 모여서 스터디를 할 경우 서로간의 눈 높이를 맞추기 위해서 이론서를 함께 읽습니다.

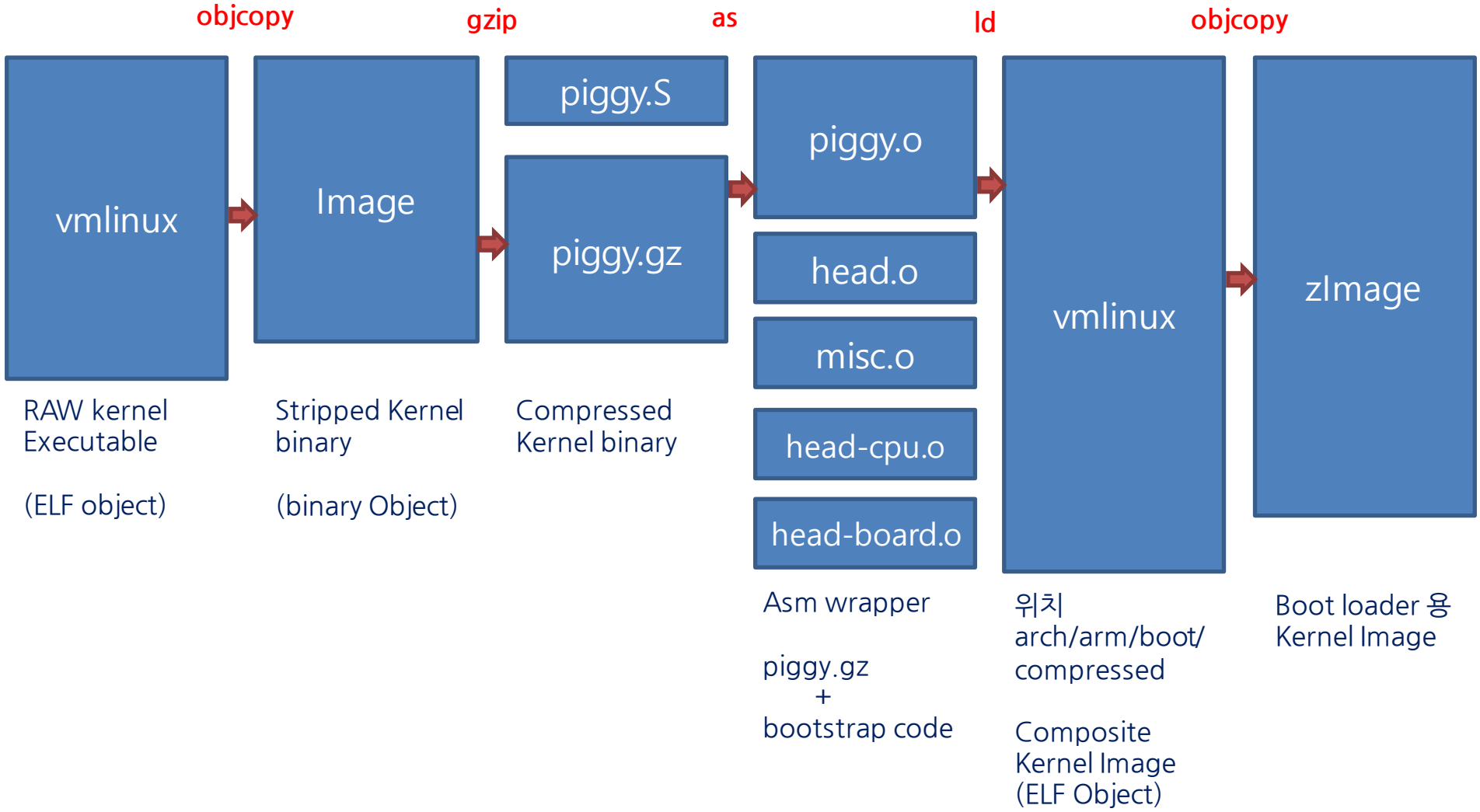


2. 이론서는 얇은 책으로 해서 라운드 로빈 방식으로 서로 조금씩 돌아가면서 읽습니다.

장소: 대학교 강의실, 회의실, 토즈(?)
방법:
- 소스 코드 드라이빙
- 코드를 보면서 토론.
- 구글 Doc등으로 자료 공유
- Wiki등을 활용해서 내용정리
- Q&A로 오프라인 자료 보강



- Make를 통해서 커널 소스 코드를 컴파일후 kernel image가 만들어지는 과정



예시1) 실제로 vmlinux를 확인해 보자.

arch/arm/boot/compressed/vmlinux 는 ELF binary입니다.
따라서

[ELF Header]
[Program Header Table]
[Section]
  ...
[Section]
[Section Header Table]

식으로 Section이 구성됩니다.

vmlinux section 정보확인

arm-none-eabi-readelf -S ./vmlinux

```
There are 22 section headers, starting at offset 0x2beb1c:


Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .text             PROGBITS        00000000 001000 005a94 00  AX  0   0 32
  [ 2] .rodata           PROGBITS        00005a94 006a94 000d10 00   A  0   0  4
  [ 3] .piggydata        PROGBITS        000067a4 0077a4 29be70 00   A  0   0  1
  [ 4] .got              PROGBITS        002a2614 2a3614 00003c 00  WA  0   0  4
  [ 5] .pad              PROGBITS        002a2650 2a3650 000008 00   A  0   0  0
  [ 6] .bss              NOBITS          002a2658 2a3658 000024 00  WA  0   0  4
  [ 7] .stack            NOBITS          002a2680 2a3680 001000 00  WA  0   0  1
  [ 8] .debug_line       PROGBITS        00000000 2a4680 00263f 00      0   0  1
  [ 9] .debug_info       PROGBITS        00000000 2a6cbf 008b02 00      0   0  1
  [10] .debug_abbrev     PROGBITS        00000000 2af7c1 001697 00      0   0  1
  [11] .debug_aranges    PROGBITS        00000000 2b0e58 000188 00      0   0  8
  [12] .debug_ranges     PROGBITS        00000000 2b0fe0 0010e8 00      0   0  8
  [13] .debug_frame      PROGBITS        00000000 2b20c8 0009b8 00      0   0  4
  [14] .debug_loc        PROGBITS        00000000 2b2a80 008d36 00      0   0  1
  [15] .debug_str        PROGBITS        00000000 2bb7b6 0013f7 01  MS  0   0  1
  [16] .comment          PROGBITS        00000000 2bcbad 000010 01  MS  0   0  1
  [17] .note.gnu.gold-ve NOTE            00000000 2bcbc0 00001c 00      0   0  4
  [18] .ARM.attributes   ARM_ATTRIBUTES  00000000 2bcbdc 00002d 00      0   0  1
  [19] .symtab           SYMTAB          00000000 2bcc0c 0012d0 10     20 194  4
  [20] .strtab           STRTAB          00000000 2bdedc 000b5f 00      0   0  1
  [21] .shstrtab         STRTAB          00000000 2bea3b 0000e0 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
```

vmlinux section 정보 재확인

arch/arm/boot/zImage의 시작와 끝을 확인합니다.

hexdump arch/arm/boot/zImage | head -n4

```
0000000 00 00 a0 e1 00 00 a0 e1 00 00 a0 e1 00 00 a0 e1
*
0000020 02 00 00 ea 18 28 6f 01 00 00 00 00 58 26 2a 00
0000030 00 90 0f e1 19 0e 00 eb 01 70 a0 e1 02 80 a0 e1
```

```
[ 1] .text              00000000 001000 005a94 00
[ 2] .rodata            00005a94 006a94 000d10 00
[ 3] .piggydata         000067a4 0077a4 29be70 00
[ 4] .got               002a2614 2a3614 00003c 00
[ 5] .pad               002a2650 2a2650 000008 00
```

hexdump arch/arm/boot/zImage | tail -n4

```
02a2630 a4 67 00 00 14 26 2a 00 60 09 00 00 78 26 2a 00
02a2640 74 26 2a 00 00 00 00 00 00 00 00 00 00 00 00 00
02a2650 00 00 00 00 00 00 00 00
02a2658
```

그리고 .text 섹션의 시작 위치(0x001000) 와 .pad 섹션의 끝 오프셋 (0x2a3630)을 확인합니다.

hexdump -s 0x1000 arch/arm/boot/compressed/vmlinx | head -n4

```
0001000 00 00 a0 e1 00 00 a0 e1 00 00 a0 e1 00 00 a0 e1
*
0001020 02 00 00 ea 18 28 6f 01 00 00 00 00 58 26 2a 00
0001030 00 90 0f e1 19 0e 00 eb 01 70 a0 e1 02 80 a0 e1
```

hexdump -s 0x2a3630 arch/arm/boot/compressed/vmlinx | head -n4

```
02a3630 a4 67 00 00 14 26 2a 00 60 09 00 00 78 26 2a 00
02a3640 74 26 2a 00 00 00 00 00 00 00 00 00 00 00 00 00
02a3650 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Kernel image가 메모리에 로드되기 까지

전원 on -> AP 자체 초기화 -> boot address 로 jmp

전원 on
Hardware internal initialize

BIOS (x86)
- POST device 초기화
- 부팅 매체 탐색
- 부트 로더 로딩

ARM 등
- Boot device select (HW 설정에 따름)
- Flash, ROM, eMMC, SD card
- Stage 1: 부트로더 로딩 (machine code, 내부 롬 실행)

Boot loader (x86)
- LILO (2.6.x 이전에 주로 사용)
- GRUB
  - Stage 1: MBR
  - Stage 2: 압축된 커널 로딩

ARM boot loader
- U-boot
- Angel (StrongARM)
- boot loader
- Stage 2: 압축된 커널 로딩

arch/x86/boot/header.S
- bootsect_start:

arch/x86/boot/compressed/head_32.S
- startup_32:

arch/x86/boot/compressed/misc.c
- decompress_kernel()

arch/x86/kernel/head_32.S
- startup_32:

init/main.c
- start_kernel()

arch/arm/boot/compressed/head.S
- start:

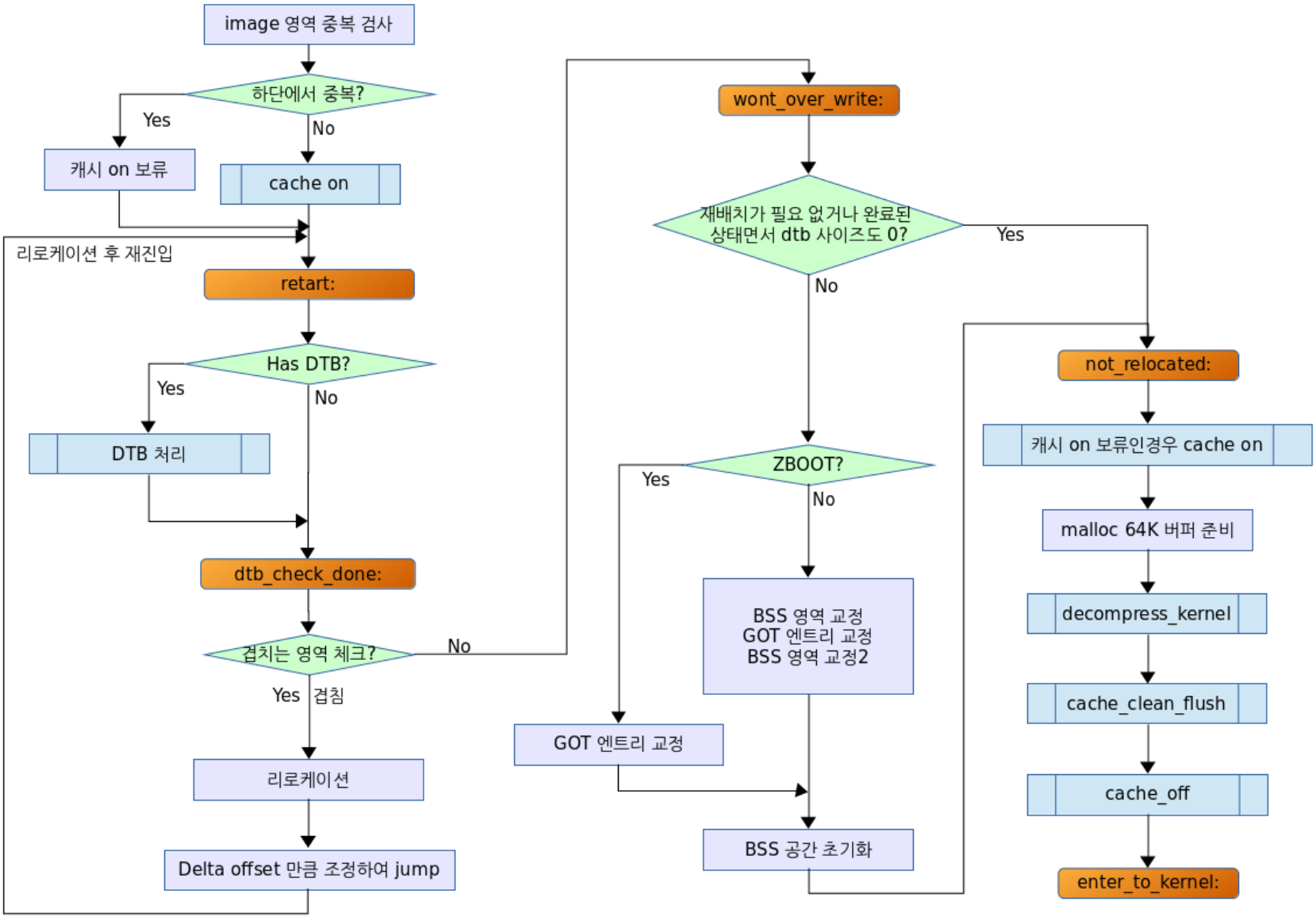
arch/arm/boot/compressed/misc.c
- decompress_kernel()

arch/arm/kernel/head.S
- hw 초기화, mmu 초기화

init/main.c
- start_kernel()

decompress/head.s

* 이미지 출처: http://iamroot.org/wiki/doku.php?id=kernel_original_12_a:부트_관련



Boot Loader -> start_kernel()

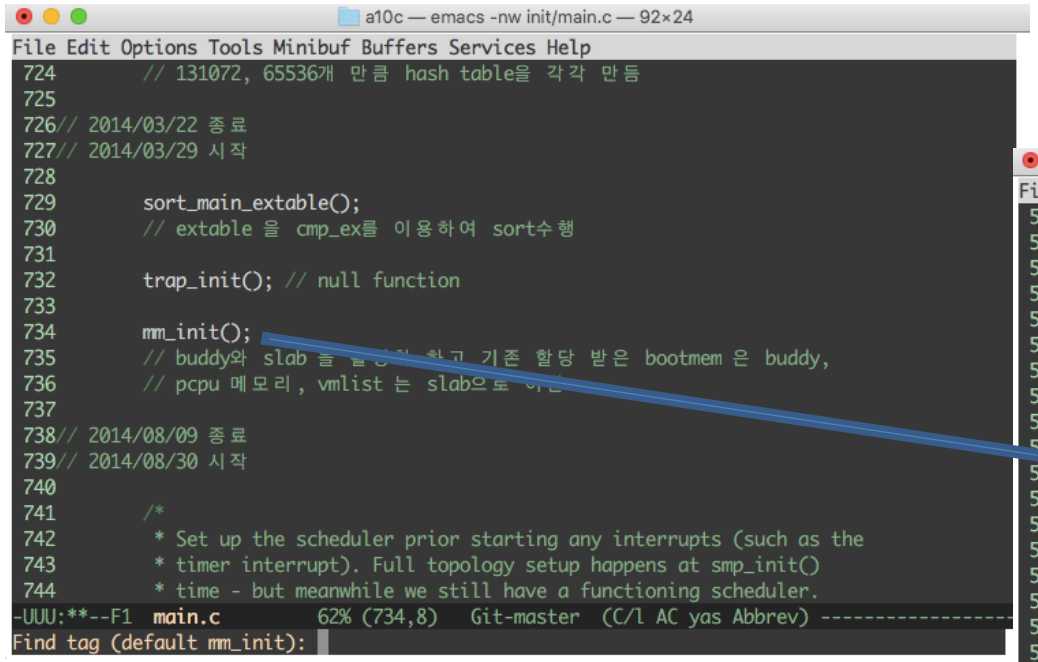
- Start_kernel()은 init 프로세스가 동작할 때까지 커널을 초기화합니다.
- 코드는 init/main.c 에 정의합니다.

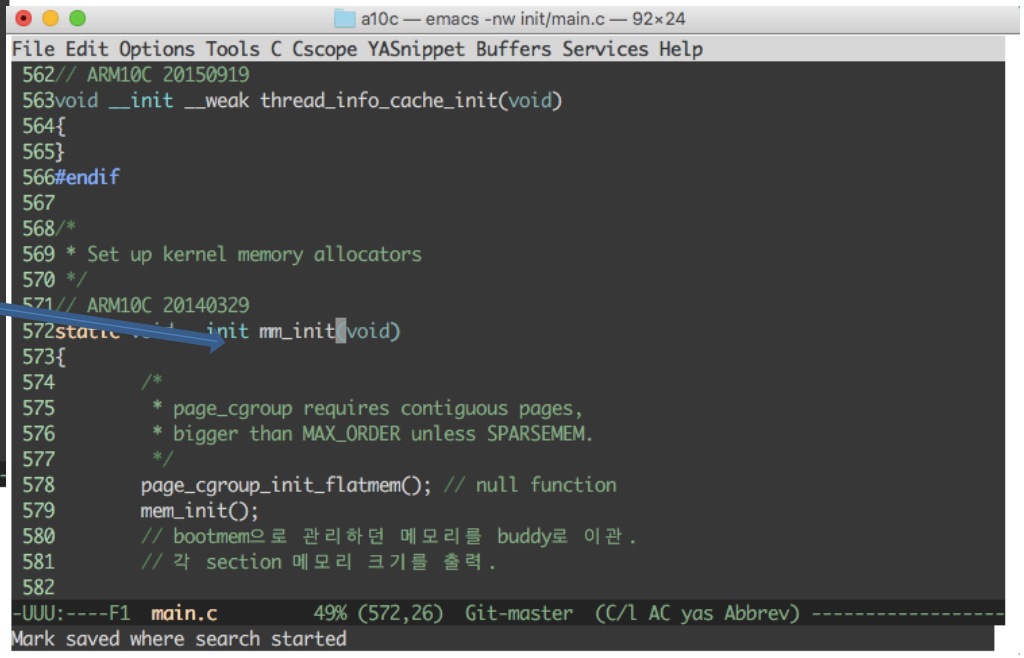
```
614// ARM10C 20130824
615asmlinkage void __init start_kernel(void)
616{
617         char * command_line;
618         extern const struct kernel_param __start___param[], __stop___param[$
619         // ATAG,DTB 정보로 사용
620
621         /*
622          * Need to run as early as possible, to initialize the
623          * lockdep hash:
624          */
625         lockdep_init();
626         smp_setup_processor_id();
627         debug_objects_early_init();
628
629         /*
630          * Set up the the initial canary ASAP:
631          */
632         boot_init_stack_canary();
633
634         cgroup_init_early();
635         // cgroup 를 사용하기 위한 cgroup_dummy_root, cgroup_subsys 의 구조$
636
637         local_irq_disable();
638         // IRQ를 disable 함
639
-UUU:**--F1  main.c         52% (613,0)   Git-master  (C/l AC yas Abbrev) ------
```

예시2) tag를 이용해서 함수를 추적하자.

리눅스 커널 개발을 위한 여러가지 도구제공
- ctags. gtags. cscope, …

예) start_kernel()->mm_init()->mem_init()





예시2) tag를 이용해서 함수를 추적하자.

리눅스 커널 개발을 위한 여러가지 도구제공
- ctags. gtags. cscope, …

예) start_kernel()->mm_init()

a10c — emacs -nw init/main.c — 92×24

```
File Edit Options Tools C Cscope YASnippet Buffers Services Help
562// ARM10C 20150919
563void __init __weak thread_info_cache_init(void)
564{
565}
566#endif
567
568/*
569 * Set up kernel memory allocators
570 */
571// ARM10C 20140329
572static void __init mm_init(void)
573{
574        /*
575         * page_cgroup requires contiguous pages,
576         * bigger than MAX_ORDER unless SPARSEMEM.
577         */
578        page_cgroup_init_flatmem(); // null function
579        mem_init();
580        // bootmem으로 관리하던 메모리를 buddy로 이관.
581        // 각 section 메모리 크기를 출력.
582
-UUU:----F1  main.c         49% (572,26)  Git-master  (C/l AC yas Abbrev) ----------
Mark saved where search started
```

a10c — emacs -nw init/main.c — 92×24

```
File Edit Options Tools C Cscope YASnippet Buffers Services Help
770 * mem_init() marks the free areas in the mem_map and tells us how much
771 * memory is free.  This is done after various parts of the system have
772 * claimed their memory after the kernel image.
773 */
                                    20140329
775void __init mem_init(void)
776{
777#ifdef CONFIG_HAVE_TCM // CONFIG_HAVE_TCM=n
778        /* These pointers are filled in on TCM detection */
779        extern u32 dtcm_end;
780        extern u32 itcm_end;
781#endif
782
783        // max_pfn : 0x80000, PHYS_PFN_OFFSET: 0x20000, *mem_map: NULL
784        // pfn_to_page(0xA0000): page 10번째 section 주소 + 0xA0000
785        max_mapnr   = pfn_to_page(max_pfn + PHYS_PFN_OFFSET) - mem_map;
786        // max_mapnr: page 10번째 section 주소 + 0xA0000
787
788        /* this will put all unused low memory onto the freelists */
789        free_unused_memmap(&meminfo);
790        // bank 0, 1에 대해 bank 0, 1 사이에 사용하지 않는 공간이 있거나
-UUU:----F1  init.c         81% (775,12)  Git-master  (C/l AC yas Abbrev) ----------
```

start_kernel()의 주요 함수와 그 처리

start()_kernel()
- smp_setup_processor_id();// physical CPU 0 (가칭) 으로 boot 진행
- cgroup_init_early(); // cgroup을 사용하기 위한 구조체 초기화
- boot_cpu_init(); // 현재 cpu(core id)를 읽어서 cpu_XXX_bits를 설정함

- setup_arch(); // dtb를 읽어서 메모리 블록과 영역 초기화

- page_alloc_init(); // cpu_chain에 page_alloc_cpu_notify를 연결

- mm_init(); // 메모리 블록에서 buddy초기화 후 stub 초기화

- sched_init(); // scheduller가 사용하는 구조체 초기화, init_task설정

- rcu_init(); // rcu자료 구조, bh, sched, preemt 초기화

- early_irq_init(); // irq_desc 0~15까지 object를 할당 받고 초기화
- init_IRQ(); // GIC, COMBINNER가 사용할 메모리 할당과 자료 구조 설정

- call_function_init(); // 각 cpu core에서 사용할 call_single_queue 초기화

- pidmap_init(); // pidmap 초기화

- rest_init() // init_process (PID 1번) 실행을 위한 초기화 후실행

setup_arch()의 주요 함수와 그 처리

start()_kernel()->setup_arch()
- setup_processor(); // 각 프로세스에 의존적인 초기화 함수, 구조체를 할당하고 CPU의 스택을 설정함.

- sanity_check_meminfo(); // memory bank에서 valloc limit 만큼 메모리를 자름.

- arm_memblock_init(); // initrd로 전달된 meminfor를 참조하여 메모리 블록 구조체를 초기화
  // kernel 사용 영역을 reserve, hardware(chip)에 관련된 영역 reserve
  // memblock.reserved 에 있는 page table , dtb 추가

- pageing_init() // MMU를 위한 page table(PGD, PTE) 생성 (1M,4k단위로 매핑하면서 생성)
  - bootmem_init() // memory bank를 bootmem으로 변환함, free, reserved에 맞게 bitmap생성
  // 256MB단위로 8개의 section_mem_map을 구성

- unflatten_device_tree() // device tree를 flat tree 에서 tree로 구성
  // of_allnodes, of_chosen, of_aliases, aliases_lookup 을 만듬.

- smp_init_cpus() // init task를 실행하는 cpu를 초기화

mm_init() 의 주요 함수와 그 처리

start()_kernel()->mm_init()
setup_arch()에서 초기화된 bootmem에서 메모리 관리를 buddy로 바꾸고, 이후 slub을 활성화 합니다.
vmlist에 등록한 vm struct 들을 slab으로 이관하고 RB tree를 구성합니다.

- mem_init();
  // bootmem으로 관리하던 메모리를 buddy로 이관.

- kmem_cache_init();
  // slub을 활성화 시킴

- percpu_init_late();
  // dchunk로 할당 받은 pcpu 메모리 값들을 slab으로 카피하여 이관

- vmalloc_init();
  // vmlist에 등록된 vm struct 들을 slab으로 이관하고 RB Tree로 구성

mm_init()->mem_init()

start()_kernel()->mm_init()->mem_init()
bootmem으로 관리하던 메모리를 buddy로 바꾸는 과정입니다. bootmem에 사용하지 않는 공간이나 align되지 않으면 free_unused_memmap() 으로 정렬하고, free_all_bootmem()에서 bootmem에서 buddy로 바뀜니다.

- free_unused_memmap(&meminfo);
  // bank 0, 1에 대해 bank 0, 1 사이에 사용하지 않는 공간이 있거나
  // align이 되어 있지 않으면 free_memmap을 수행

- free_all_bootmem();
  // bootmem으로 관리하던 메모리를 buddy로 이관.

- free_highpages();
  // highmem의 reserved 영역을 제외하고 buddy order 0 에 추가

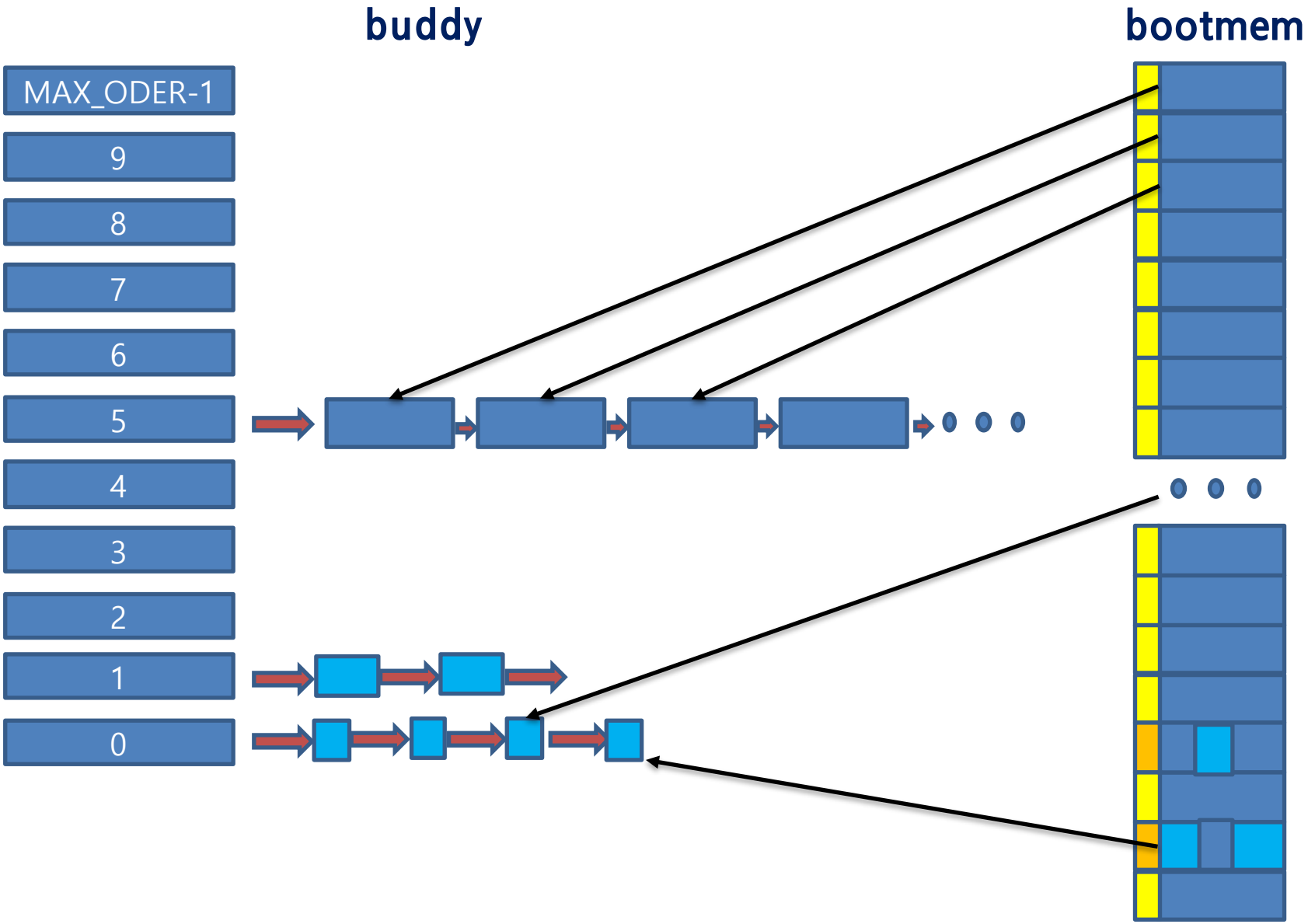
- mem_init_print_info(NULL);
  // 각 메모리 섹션의 정보를 구하여 출력.

mm_init()->mem_init()->free_all_bootmem()

start()_kernel()->mm_init()->mem_init()->free_all_bootmem()->free_all_bootmem_core(bdata)->__freepages_bootmem(pfn_to_page(start),order)
bootmem으로 관리하던 메모리를 buddy로 바꾸는 과정입니다.

```
- while (start < end) {
  - shift = idx & (BITS_PER_LONG - 1);
  - if (IS_ALIGNED(start, BITS_PER_LONG) && vec == ~0UL) {
        __free_pages_bootmem(pfn_to_page(start), order);
          // CPU0의 vm_event_states.event[PGFREE] 를 32로 설정함
          // page에 해당하는 pageblock의 migrate flag를 반환함
          // struct page의 index 멤버에 migratetype을 저장함
          // struct page의 _count 멤버의 값을 0 으로 초기화함
          // order 5 buddy를 contig_page_data에 추가함
          // (&contig_page_data)->node_zones[ZONE_NORMAL].vm_stat[NR_FREE_PAGES]: 32 로 설정
          // vmstat.c의 vm_stat[NR_FREE_PAGES] 전역 변수에도 32로 설정
          count += BITS_PER_LONG;
          start += BITS_PER_LONG;
    } else { // node_bootmem_map[0]의 값이 0아닐 경우
          while (vec && cur != start) {
              if (vec & 1) {
                  __free_pages_bootmem(page, 0);
                  // CPU0의 vm_event_states.event[PGFREE] 를 1로 설정함
                  // page에 해당하는 pageblock의 migrate flag를 반환함
                  // struct page의 index 멤버에 migratetype을 저장함
                  // struct page의 _count 멤버의 값을 0 으로 초기화함
                  // order 0 buddy를 contig_page_data에 추가함
                  // (&contig_page_data)->node_zones[ZONE_NORMAL].vm_stat[NR_FREE_PAGES]: 1 로 설정
                  // vmstat.c의 vm_stat[NR_FREE_PAGES] 전역 변수에도 1로 설정
                  count++;
              }
    {
```

free_all_bootmem()



mm_init()->vmalloc_init()

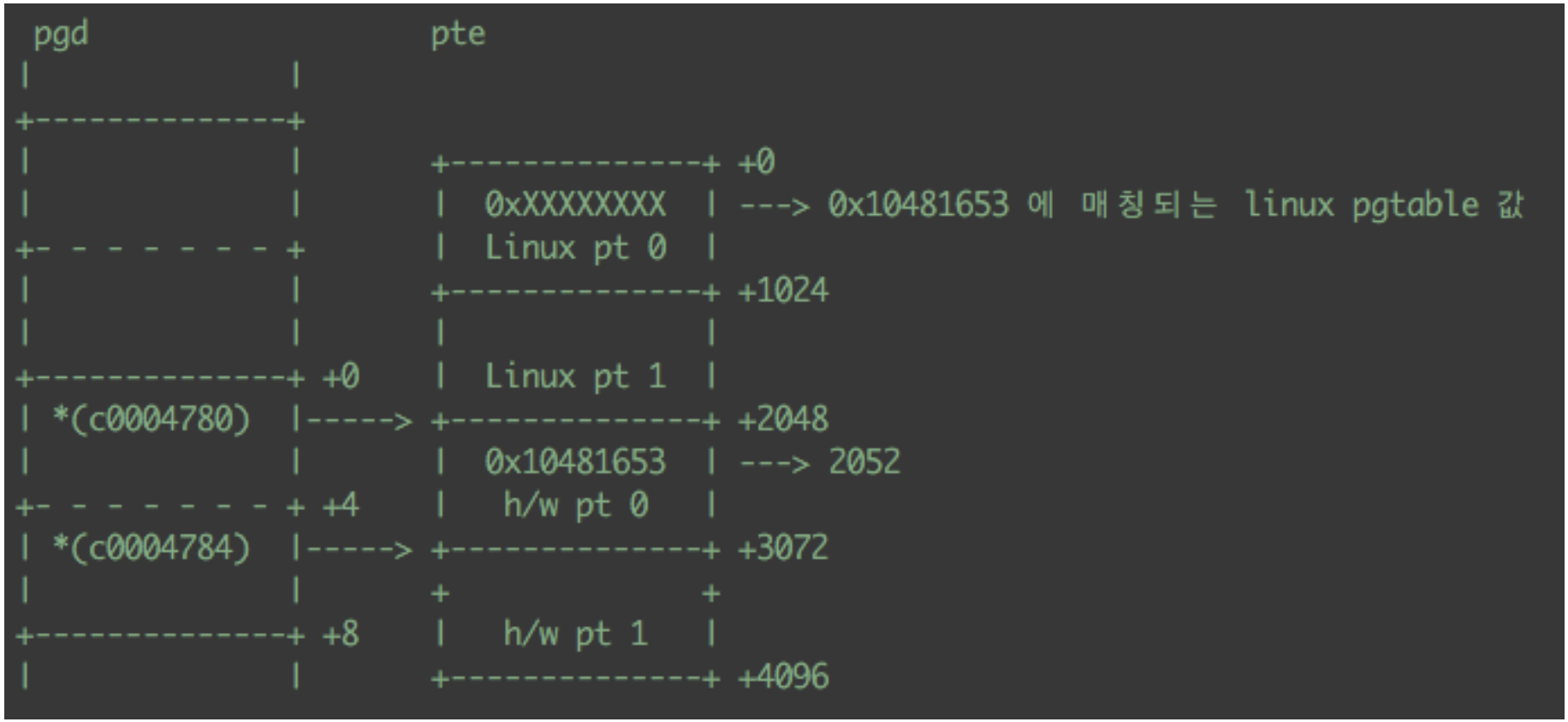
start()_kernel()->mm_init()->vmalloc_init()
vmlist에 등록한 vm struct 들을 slab으로 이관하고 RB tree를 구성합니다.

```
- for_each_possible_cpu(i) {
    INIT_LIST_HEAD(&vbq->free);
     // &vbq->free: &(&vmap_block_queue + __per_cpu_offset[0])->free 리스트 초기화
    init_llist_head(&p->list);
     // llist의 first를 NULL로 초기화
    }
- for (tmp = vmlist; tmp; tmp = tmp->next) {
    va = kzalloc(sizeof(struct vmap_area), GFP_NOWAIT);
      // sizeof(struct vmap_area): 52 bytes, GFP_NOWAIT: 0
      // kzalloc(52, 0): kmem_cache#30-o9
    va->va_start = (unsigned long)tmp->addr;
      // va->va_start: (kmem_cache#30-o9)->va_start: 0xf6100000
    va->va_end = va->va_start + tmp->size;
      // va->va_end: (kmem_cache#30-o9)->va_end: 0xf6110000
    va->vm = tmp;
     // va->vm: (kmem_cache#30-o9)->vm: SYSC
    __insert_vmap_area(va);
     // vm SYSC 정보를 RB Tree 구조로 삽입
  }
```

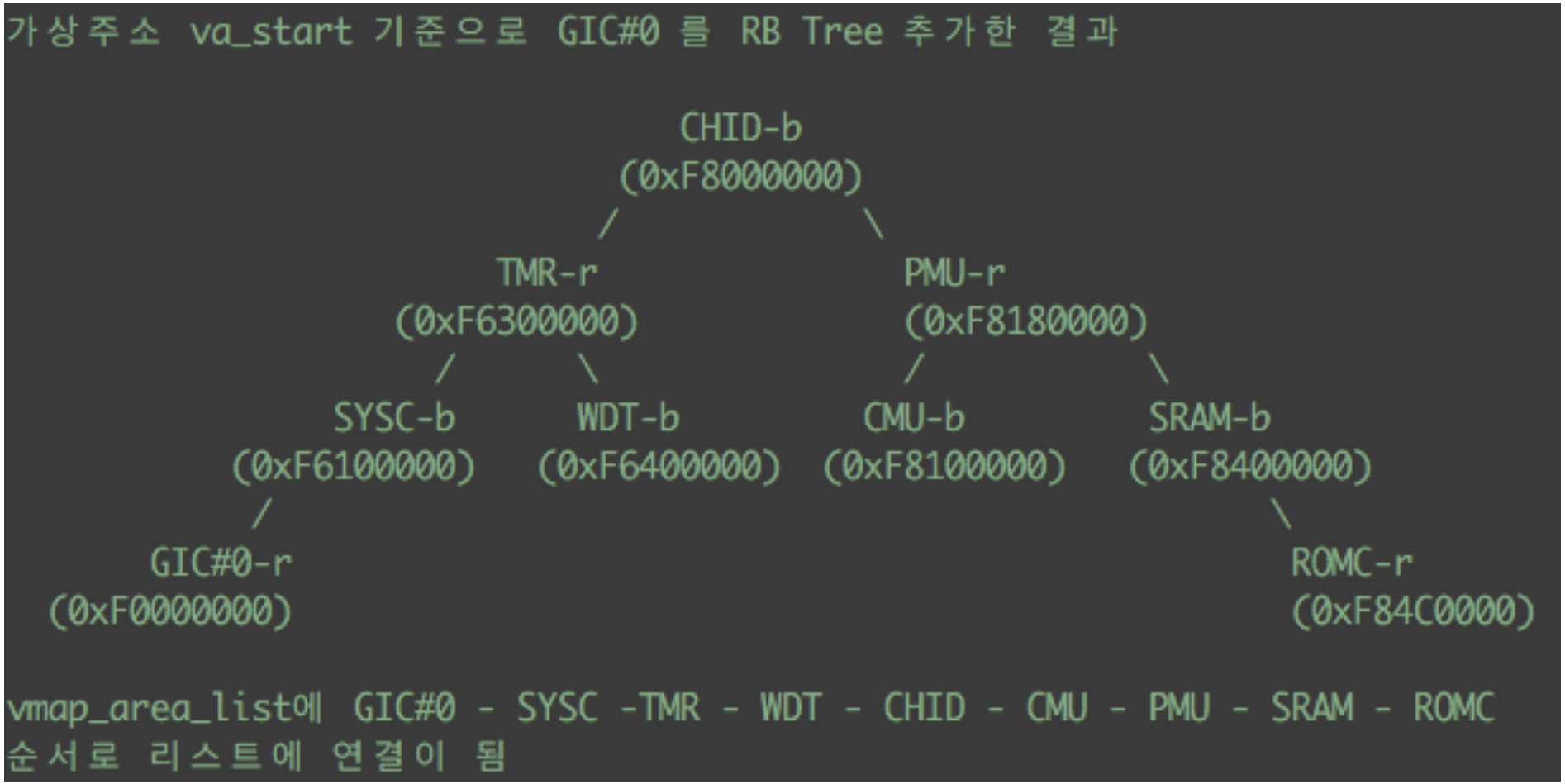
Irq_init() 의 주요 함수와 그 처리

start()_kernel()->irq_init()

- of_irq_init(); // device tree에 있는 gic node에서 node의 resource 값으로
// alloc area (GIC#0)를 만들고 rb tree에 alloc area를 추가
// vmap_area_list에 GIC#0 - SYSC - TMR - WDT - CHID - CMU - PMU - SRAM -ROMC
// 순서로 리스트에 연결
// gic node에 resource를 pgtable에 매핑함.



device tree에서 device tree 있는 gic node에서 node의 resource 값을 가져옴
vmap_area_list에 GIC#0 - GIC#1 - SYSC -TMR - WDT - CHID - CMU - PMU - SRAM - ROMC순서로 리스트에 연결



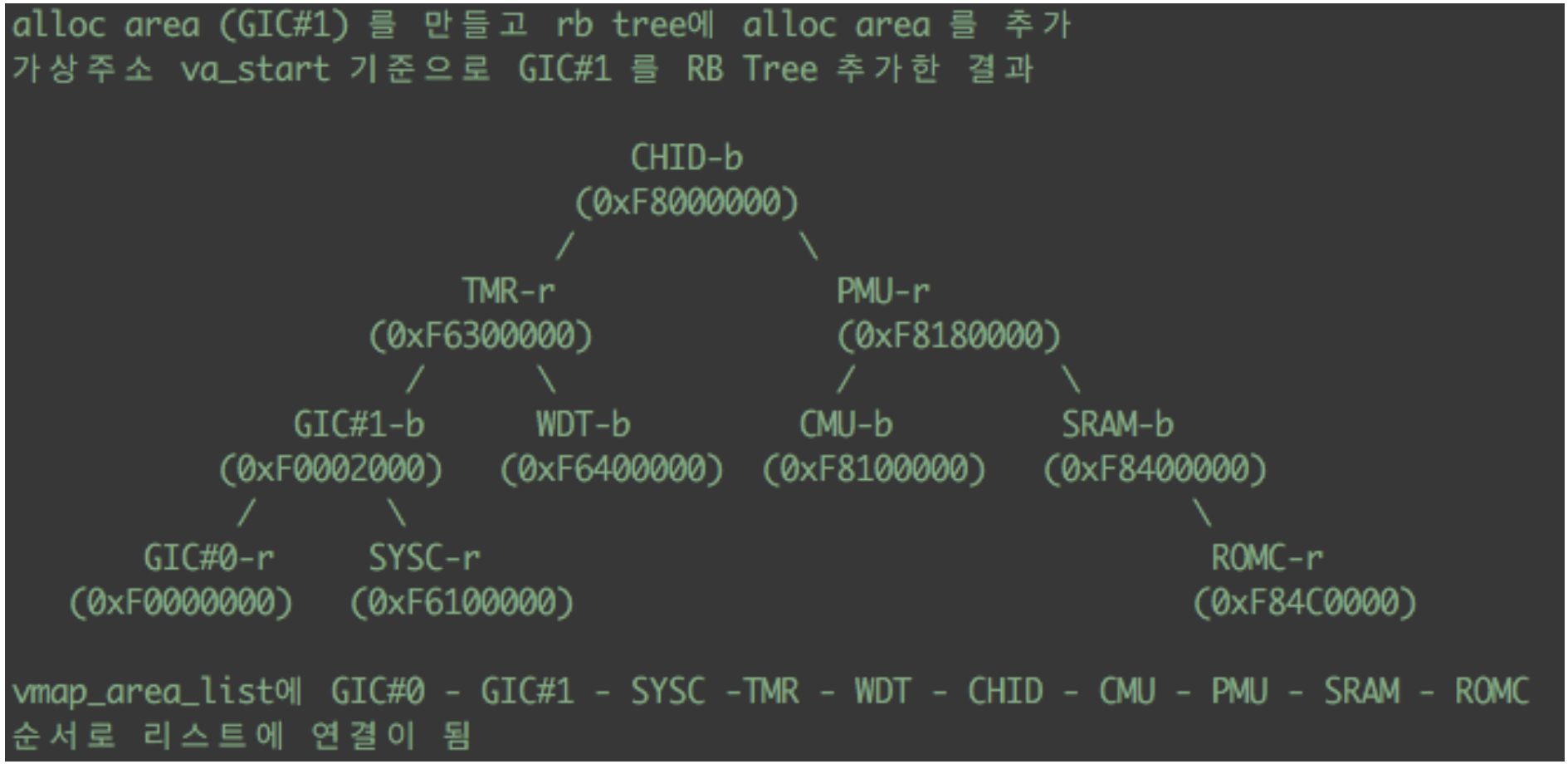
Irq_init() 의 주요 함수와 그 처리

start()_kernel()->irq_init()

- of_irq_init(); // device tree에 있는 gic node에서 node의 resource 값으로
// alloc area (GIC#1)를 만들고 rb tree에 alloc area를 추가
// vmap_area_list에
// GIC#0 - GIC#1 - SYSC - TMR - WDT - CHID - CMU - PMU - SRAM -ROMC

irq_init()->of_vmalloc_init()->__insert_vmap_area()

device tree에서 device tree 있는 gic node에서 node의 resource 값을 가져옴
vmap_area_list에 GIC#0 - GIC#1 - SYSC -TMR - WDT - CHID - CMU - PMU - SRAM - ROMC순서로 리스트에 연결

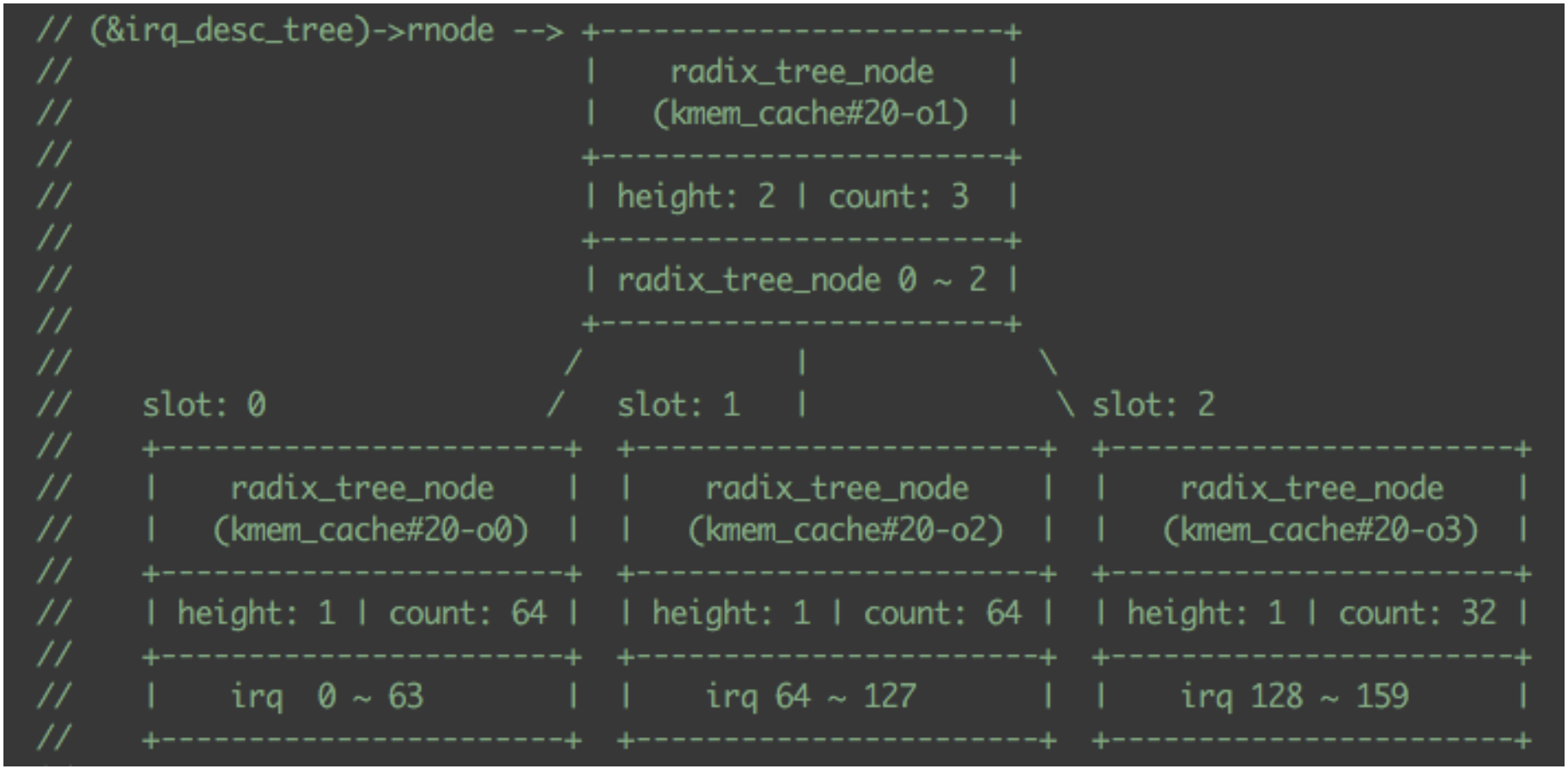


IRQ_init()

start()_kernel()->IRQ_init()->irq_chip_init()->of_irq_init()

device tree에서 device tree 있는 gic node에서 node의 resource 값을 가져옴
vmap_area_list에 GIC#0 - SYSC -TMR - WDT - CHID - CMU - PMU - SRAM - ROMC순서로 리스트에 연결

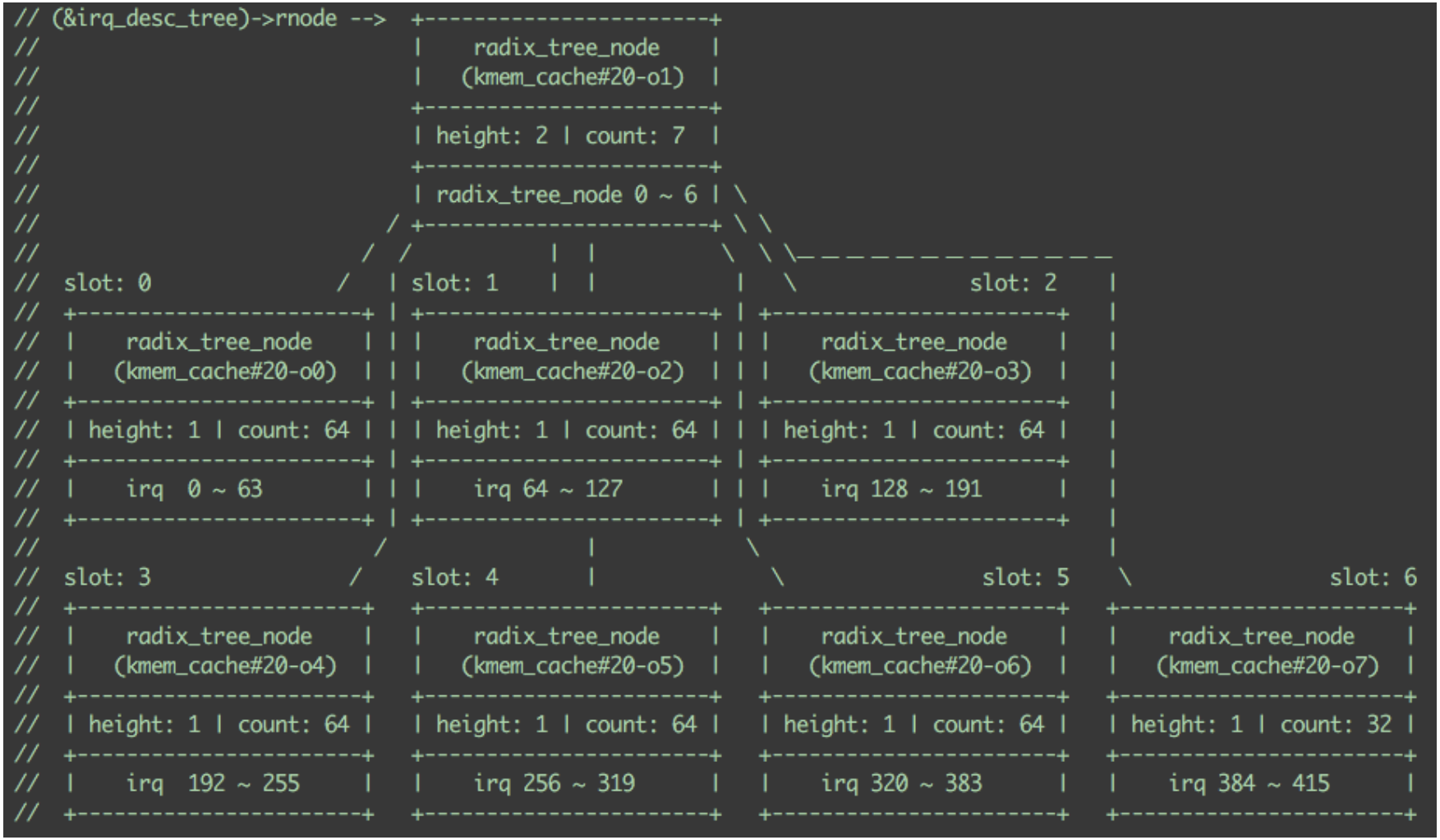
struct irq_desc의 자료 구조크기 만큼 160개의 메모리를 할당 받아 radix tree 구조로 구성



IRQ_init()

start()_kernel()->IRQ_init()->irq_chip_init()->of_irq_init()

struct irq_desc의 자료 구조크기 만큼 160개의 메모리를 할당 받아 radix tree 구조로 구성



start()_kernel()->rest_init()
init 프로세스(프로세스 No. 1)을 동작시키기 위한 커널 쓰레드와 커널 쓰레드 데몬(프로세스No.2)를 생성하고
CPU를 대기상태와 함께 스케쥴러를 호출합니다.

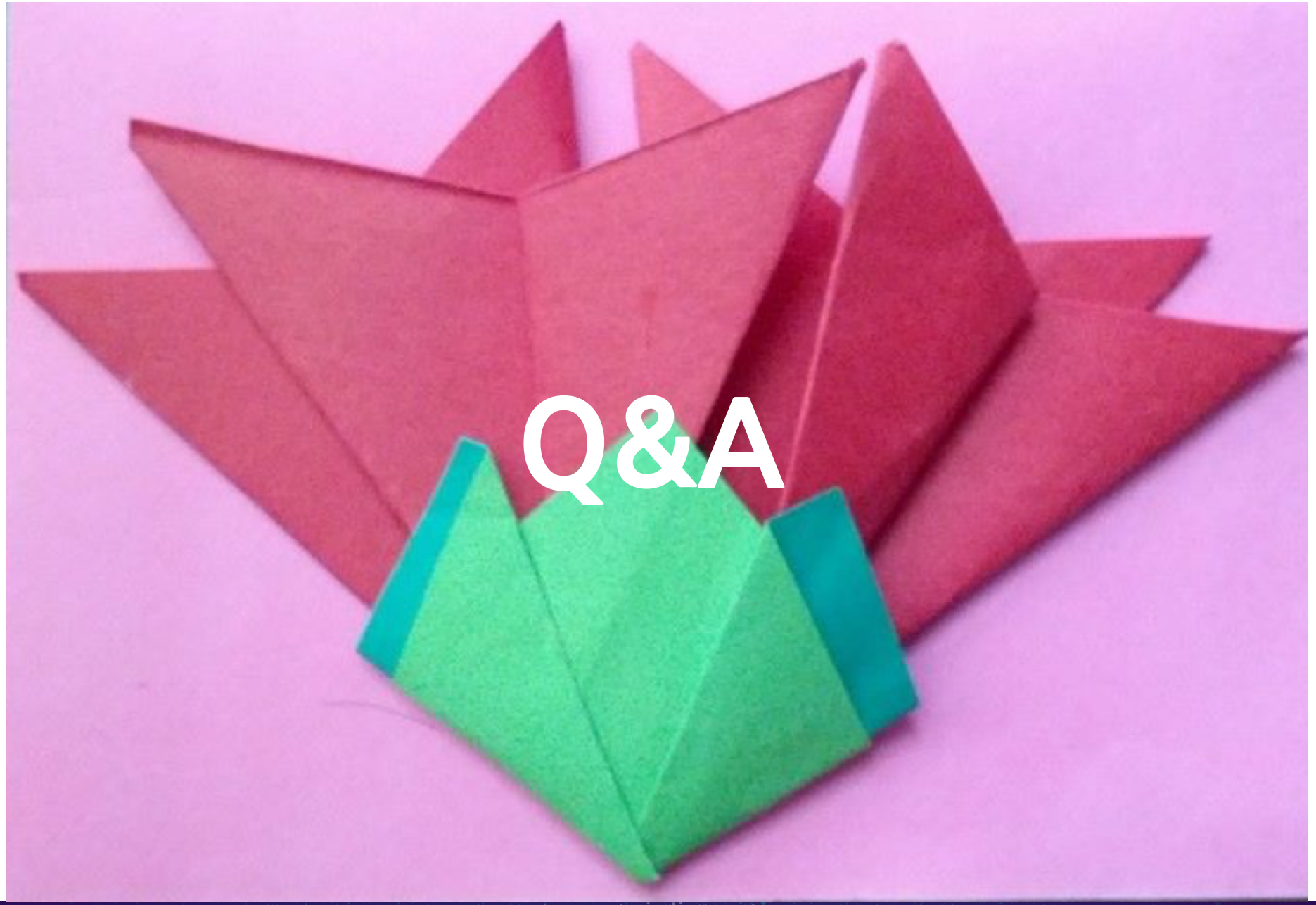
- kernel_thread(kernel_init, NULL, CLONE_FS | CLONE_SIGHAND);
  // kernel_thread()는 kernel_init (커널 초기화 쓰레드)를 만들고,
  // 스케쥴러 초기화와 함께 루트파일 시스템을 마운트 합니다.
  // 그후 init program을 동작시킨 후 init_process로 바뀝니다.

- pid = kernel_thread(kthreadd, NULL, CLONE_FS | CLONE_FILES);
  // kthreadd (커널 쓰레드 데몬) 쓰레드를 만듭니다.
  // init 과 kthread 이외의 커널 쓰레드는 kthreadd에 의해서 만들어 집니다.
  // kthreadd의 자식 쓰레드가 되는 것입니다.

- complete(&kthreadd_done);
  // kthreadd_done은 kernel_init 쓰레드 동기를 위해서 사용하는 변수 입니다.
  // kernel_init 쓰레드는 여기까지는 대기상태로 있습니다.

- schedule_preempt_disabled();
  // schedule(); 를 실행합니다.
  // 프로세스 스케쥴러라고도 말하는 런큐 (run queue)에게 프로세스 제어를 넘깁니다.
  // 이후 kernel_init과 kthreadd 쓰레드가 활성화되어 자식 프로세스를 생성할 수 있습니다.

- cpu_startup_entry(CPUHP_ONLINE);
  // cpu_idle_loop()로 init 프로세스와 kthreadd 프로세스를 대기상태로 해서 런큐에 따라 할당합니다.


Q&A


SOSCON
삼성 오픈소스 컨퍼런스
SAMSUNG OPEN SOURCE CONFERENCE
OPEN YOUR UNIVERSE WITH SOSCON
THANK YOU!