

부트로더 (Bootloader)

연세대학교 컴퓨터정보통신공학부
윤 상 군

부트로더

- 부트로더(bootloader)란
 - 전원이 켜졌을 때에 처음으로 수행되는 프로그램으로
 - 하드웨어를 초기화하고 동작을 검사한 후에
 - 커널을 메모리에 로드하여 실행을 시작하게 한다.
- CPU reset 주소
 - 전원이 공급되기 시작하거나 하드웨어 reset을 시키면 CPU의 reset 주소(0번지 또는 마지막 부분 번지)에서 코드가 실행되기 시작함
 - reset 주소가 있는 위치에 비휘발성 메모리가 배치됨
 - 플래시 메모리, EPROM 등

Bootloader의 주요 기능 및 종류

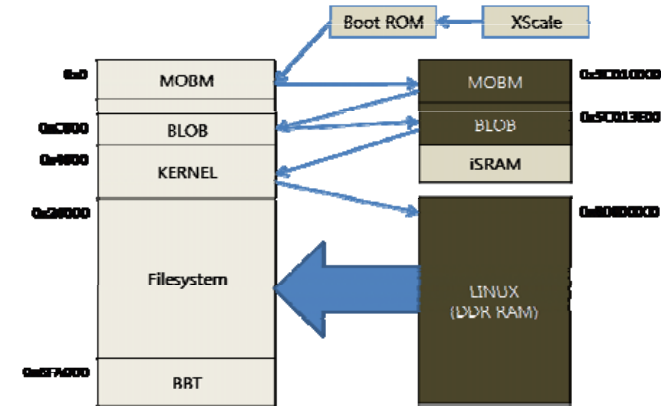
- 부트로더의 주요 기능
 - interrupt vector 설정, interrupt handler 포함
 - 메모리 설정 및 초기화
 - I/O interface 초기화
 - stack pointer 초기화
 - kernel booting
- 부트로더의 종류
 - lilo (linux loader)
 - grub (GRand Unified Bootloader)
 - file system과 kernel image format을 인식하는 기능 포함
 - **blob** (Boot Loader Object) .. **실습기기에서 사용**
 - u-boot, redboot, ppcboot, armboot, ...

부트로더의 주요 동작

- hardware 초기화
 - Memory setting CPU Clock setting
 - GPIO setting Serial setting
 - MAC address 획득 및 Ethernet port setting
- Flash 에서 RAM으로 copy
 - bootloader, kernel, ramdisk image copy
- Kernel booting
- Command mode 제공
 - target board에 개발한 program을 전송하기 위한 기능들.
 - download image to SDRAM
 - Serial – xmodem, uuencode
 - Ethernet – tftp*, bootp
 - fusing Flash memory
 - Write, Read, Erase, Lock/Unlock

PXA320의 booting 절차

- 전원이 공급되면 PXA320의 Boot ROM에 있는 Boot 코드가 실행함
 - BootROM은 0번지와 0x5c00_0000 (0x5e00_0000 ?)번지에 이중으로 맵핑되어 있음
 - 0번지의 코드는 곧바로 또 다른 맵핑 주소로 jump하여 실행됨
- BootROM
 - NAND Flash의 0번지에 있는 MOBM (mini OEM boot module) 코드를 SRAM으로 복사하여 실행시킴
- MOBM
 - Flash 메모리의 한 번에 접근할 수 있는 크기는 64KB 이하임
 - Blob는 이보다 큰 크기이므로 간단한 boot 기능을 할 수 있는 초기화 프로그램인 mobm을 먼저 실행시킴
 - 내부 SRAM 초기화, NAND Flash controller 초기화 등을 하고 blob 이미지를 내부 SRAM으로 복사한 후에 blob의 startup 코드로 branch 함



Bootloader 생성

- blob 소스 설치하기
 - 부트로더를 설치하기 위해서는 커널 소스가 필요하므로 먼저 linux 커널 소스 압축파일의 압축을 풀어서 설치함. (BSP의 Kernel 디렉토리에 위치함)

```
# tar xvfz linux.tar.gz -C ~/src/Kernel
```

- Bootloader 소스 압축파일의 압축을 풀어서 설치함 (BSP의 Bootloader 디렉토리에 위치)

```
# tar xvfz blob.tar.gz -C ~/src/Bootloader
```

Bootloader 생성(2)

- blob 소스 파일이 설치된 디렉토리로 이동


```
# cd ~/src/Bootloader/blob
```
- Makefile 생성 – configure 명령어 사용


```
# ./configure --host=arm-linux --with-board=XHYPER320TKU --with-linux-prefix=$LINUX --with-network=eth --enable-xllp --enable-xlli --enable-fvscaler --enable-nand --with-cpu-product-id=$MONAHANS
```
- blob 실행파일 생성


```
# make
```

... blob가 생성됨
- 다음 shell script를 사용하여 위의 두 명령어를 간편하게 수행할 수 있음


```
# ./config.sh
```

Default IP주소 설정하기

- IP 설정
 - Boot loader와 host PC 와의 통신을 위한 IP 설정을 한다.
 - Bootloader 소스의 다음 파일의 내용을 수정하여 IP 주소를 설정
"blob/src/blob/xhyper320tku.c"

```
unsigned char serverip[4]={192,168,10,2};  
unsigned char clientip[4]={192,168,10,3};
```

MOBM과 blob/mobm링크이미지 생성

- MOBM 이미지 생성
 - # cd mobm
 - # **make** ... mobm이 생성됨
- 링크 이미지 생성
 - link_image 명령어를 사용하여 blob와 mobm 이미지를 합쳐서 하나의 파일을 생성함
 - # link_image가 있는 디렉토리(util)로 두 이미지 복사
 - # ./link_image -d descr_v2_lb.txt >/dev/null 2>&1
 - descr_v2_lb.txt에 링크에 필요한 정보가 제공됨
 - 간편하게 하기 위해서 다음 shell script가 image 디렉토리에 제공
 - # cd image
 - # **./mkboot.sh** ... boot.bin이 생성

Bootloader 설치

- Bootloader를 Target에 설치하기
 - jtag 케이블을 사용한 설치
 - 기존의 bootloader를 이용한 설치
- blob의 tftp 명령어를 이용한 bootloader 설치
 - boot.bin을 /tftpboot 로 복사
 - # cp boot.bin /tftpboot
 - target 시스템 접속
 - # minicom

```
blob> tftp boot.bin ; tftp사용하여 RAM으로 download  
blob> nandwrite -z 0x80800000 0x0 0x40000  
memory시작 flash시작 길이  
; RAM에 있는 boot.bin을 flash에 쓰기
```

- tftp를 사용하기 위해서 필요하면 IP 주소를 설정해야 함

MOBM 분석

- startup 코드: "mobm/startup/StartUp.S" 에서 시작
 - reset handler
 - initialize Interrupt Controller
 - assign SP location for all modes of operation
 - enter FIQ, IRQ, Abort, Undef, SVC mode and set up their stack pointers, respectively
 - enable Coprocessor registers
 - initialize the DDR SDRAM controller
 - initialize memory required by C code
 - branch to OEM boot main program (OBMBootMain)
- main 코드: "mobm/main/obm.c"
 - OBMBootMain()
 - Flash의 blob 이미지를 DRAM으로 복사하고
 - TransferControlToOSLoader()를 호출하여 blob로 branch

MOBM에 대한 ld-script – linker 사용 규칙

■ startup/startup.lds

```

OUTPUT_FORMAT("elf32-littlearm", "elf32-littlearm", "elf32-littlearm")
OUTPUT_ARCH(arm)
ENTRY(_start)
SECTIONS
{
    . = 0x5c014000;
    _start = .;

    . = ALIGN(4);
    .text : {
        *(.text)
        . = ALIGN(4);
    }
    . = ALIGN(4);
    .rodata : { *(.rodata) }
    . = ALIGN(4);
    .data : { *(.data) }
    . = ALIGN(4);
    .got : { *(.got) }
    . = ALIGN(4);
    .bss : { *(.bss) }
}

```

ELF32 형식코드
little endian 사용

_start를 가장 처음 실행할 루틴으로 지정

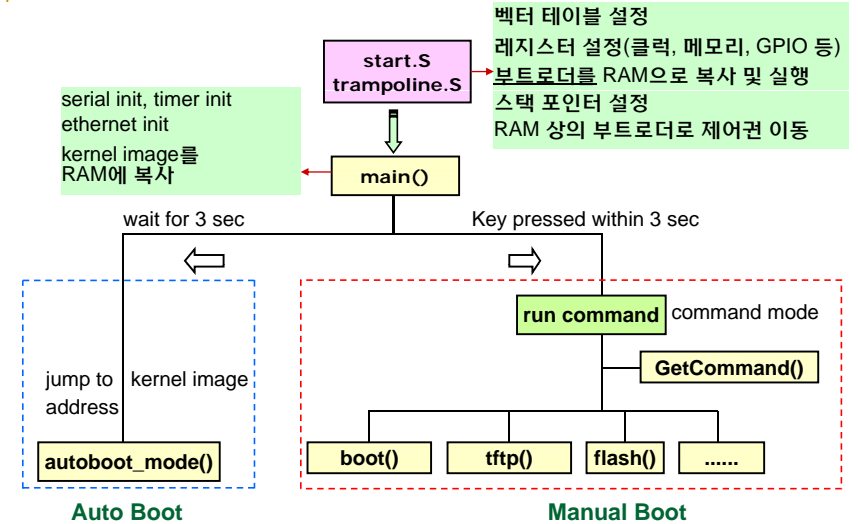
_start = object의 시작 주소
= startup.o의 시작코드 주소

OBJECTS =
startup/startup.o
main/obm.o
dfc/dfc.o
flash/flash.o
download/down.o
misc/misc.o

부트로더

13

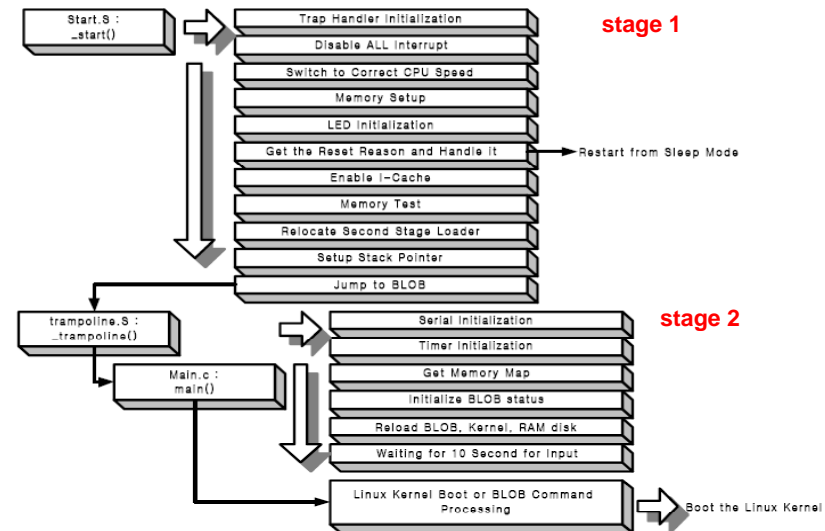
Blob의 전체적인 흐름



부트로더

14

부트로더의 실행순서

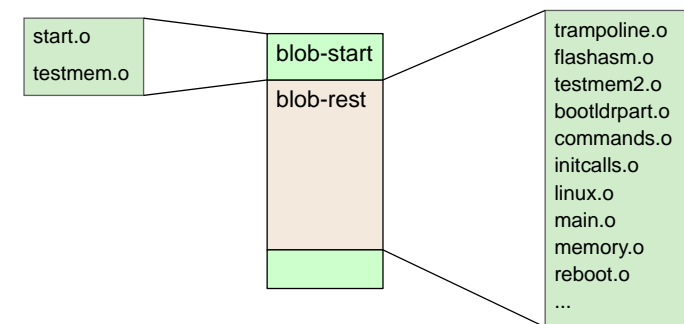


부트로더

15

blob 이미지 생성 과정

- stage 2 image (blob-rest) 생성
- stage2 image를 포함하여 stage1 image(blob-start)를 생성



부트로더

16

start-ld-script : stage 1을 위한 ld-script 파일

```

OUTPUT_FORMAT("elf32-littlearm", "elf32-littlearm", "elf32-littlearm")
OUTPUT_ARCH(arm)
ENTRY(_start)
SECTIONS
{
    . = 0x00000000;
    . = ALIGN(8);
    .text : { *(.text)
        __piggy_start = .;
        blob-rest-piggy.o
        . = ALIGN(8);
        __piggy_end = .;
    }
    . = ALIGN(8);
    .rodata : { *(.rodata) }
    . = ALIGN(8);
    .data : { *(.data) }
    . = ALIGN(8);
    .got : { *(.got) }
    . = ALIGN(8);
    .bss : { *(.bss) }
}

```

Annotations:

- `_start(start.S)`를 가장 처음 실행할 루틴으로 지정
- ELF32 형식코드 little endian 사용
- `_start`의 주소
- 8의 배수 위치로 정렬
- stage2 이미지 포함
- 여러 section 들
 - text section
 - read only data section
 - data section
 - global offset table section
 - bss section (block started by symbol)

부트로더

17

초기화 코드 (start.S)

■ Exception Vector 설정

- exception요인에 따라서 분기하는 주소가 정해져 있음
- 0번지부터 8 words(32bits)에 branch 명령어가 위치

```

.text
.globl _start
_start:
    b reset
    b undefined_instruction
    b software_interrupt
    b prefetch_abort
    b data_abort
    b not_used
    b irq
    b fiq

```

Annotations:

- start-ld-script에서 0번지로 지정됨
- 각 exception handler로 branch
- "start-mhn.S"


```

.globl reset
reset:
    ....
            
```

부트로더

18

초기화 코드 ("start-mhn.S")

```

.globl reset
reset:
    /* set CPU to SUC32 mode */
    /* Is this necessary to new BLOB? - sc */
    mrs r0, cpsr // move CPSR to r0
    bic r0, r0, #0x1f // clear bit4-0
    orr r0, r0, #0x13 // set SUC mode (10011)
    msr cpsr, r0 // move to CPSR

    /* First, mask **ALL** interrupts */
    /* Step 1 - Enable CP6 permission */
    mrc p15, 0, r1, c15, c1, 0 @ read CPAR
    orr r1, r1, #0x40 // 0100 0000 (cp6 access)
    mcr p15, 0, r1, c15, c1, 0 // move to CPAR
    cpwait r1

    /* Step 2 - Mask ICMR & ICMR2 */
    mov r1, #0
    mcr p6, 0, r1, c1, c0, 0 @ ICMR
    mcr p6, 0, r1, c7, c0, 0 @ ICMR2

    real_reset:
        bl gpiosetup // call GPIO setup proc
        bl memsetup // call memory setup proc
        .....
        bl normal_boot

```

Annotations:

- CP15: XScale uP system control
- CP6: interrupt controller

(계속)

부트로더

19

Coprocessor 15 Processor CPAR Register Processor CPAR
Register 15
opcode_2 = 0
CRm=1

User Settings	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Bit																				CP13	CP12	CP11	CP10	CP9	CP8	CP7	CP6	CP5	CP4	CP3	CP2	CP1	CP0
Reserved																																	
Reset	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Table 124: Interrupt Controller Register Mapping

Coprocessor 6 Register Number	Name	Access Mode	Description
CR0	ICIP	R	Interrupt Controller IRQ Pending register
CR1	ICMR	R/W	Interrupt Controller Mask register
CR2	ICLR	R/W	Interrupt Controller Level register
CR3	ICFP	R	Interrupt Controller FIQ Pending register
CR4	ICPR	R	Interrupt Controller Pending register
CR5	ICHP	R	Interrupt Controller Highest Priority register
CR6	ICIP2	R	Interrupt Controller IRQ Pending register 2
CR7	ICMR2	R/W	Interrupt Controller Mask register 2

부트로더

20

초기화 코드 ("start.S")

```
/* some defines to make life easier */
BLOB_START:    .word    BLOB_ABS_BASE_ADDR    0x80008000
piggy_start:   .word    __piggy_start
piggy_end:     .word    __piggy_end    } stage2 이미지의 start, end

.globl normal_boot
normal_boot:
    /* check the first 1MB of BLOB_START in increments of 4k */
    mov    r7, #0x1000    // 2^12 = 4KB
    mov    r6, r7, lsl #6    // 4KB << 6 = 256KB
    ldr    r5, BLOB_START

mem_test_loop:
    mov    r0, r5
    bl     testram
    teq    r0, #1
    beq    endless_loop    /* oops, something went wrong :( */
    add    r5, r5, r7    // increment address
    subs   r6, r6, r7    // decrement count
    bne    mem_test_loop
```

stage 2 image를 BLOB_START 위치(DRAM)로 재배치(복사)후 실행

```
relocate:    /* relocate the second stage loader */
    adr     r1, _start
    ldr     r0, piggy_start
    add     r0, r0, r1
    ldr     r2, piggy_end
    add     r2, r2, r1
    ldr     r1, BLOB_START

    /* r0 = source address
    * r1 = target address
    * r2 = source end address
    */

copy_loop:
    ldmbia  r0!, {r3-r10}
    stmbia  r1!, {r3-r10}
    cmp     r0, r2
    ble     copy_loop
    /* blob is copied to ram, so jump to it */
    ldr     r0, BLOB_START
    mov     pc, r0

endless_loop:
    b       endless_loop
```

r0: source addr = _start+piggy_start
r2: source end addr = _start+piggy_end
r1: target addr = BLOB_START

load/store multiple increment after
여기서는 한 번에 8 word씩 복사

PC ← BLOB_START

rest-ld-script : stage 2를 위한 ld script 파일

```
OUTPUT_FORMAT("elf32-littlearm", "elf32-littlearm", "elf32-littlearm")
OUTPUT_ARCH(arm)
ENTRY(_trampoline)
SECTIONS
{
    . = (0x80008000);
    . = ALIGN(8);
    .text : {
        __text_start = .;
        *(.text)
        __text_end = .;
    }

    . = ALIGN(8);
    .bss : {
        __bss_start = .;
        *(.bss)
        *(COMMON)
        . = ALIGN(8);
        __stack_start = .;
        . = . + 8 * 1024;
        __stack_end = .;
        __bss_end = .;
    }
}
```

시작주소 = _trampoline의 주소 (SDRAM주소)
이 부분은 SDRAM으로 복사되어 실행됨

프로그램에서 사용하는 label에 현재 주소 지정
같은이름의 section은 이웃하게 배치됨

stack이 bss 영역의
뒤부분에 배치됨

rest-ld-script (2)

- sections
 - text, rodata, data, got
 - commandlist: 명령어들의 모음이 포함됨, 매크로 사용하여 삽입
 - initlist: 초기화 동작 영역
 - exitlist: 종료시 동작 영역
 - ptaglist: 커널로 분기 시에 전해야할 정보(tag) 포함 영역
 - bss (block started by symbol):
 - C언어의 uninitialized 변수용 영역
 - 뒷부분을 stack 영역으로 사용

Stage 2 시작 코드 (trampoline.S)

- trampoline.S : bss 영역, stack pointer 초기화 후에 main으로 점프

```
.globl _trampoline
_trampoline:
    /* clear the BSS section */
    ldr    r1, bss_start
    ldr    r0, bss_end
    sub    r0, r0, r1    // r0 = # of bytes in bss section

    /* r1 = start address */
    /* r0 = #number of bytes */
    mov    r2, #0
clear_bss:
    stmia  r1!, {r2}    // 한번에 1워드씩 clear
    subs  r0, r0, #4
    bne   clear_bss
```

```
/* setup the stack pointer */
ldr    r0, stack_end
sub    sp, r0, #4

/* jump to C code */
bl     main
/* if main ever returns we just call it again */
b      _trampoline

bss_start:    .word    __bss_start
bss_end:      .word    __bss_end
stack_end:    .word    __stack_end
```

ld-script에서 정의됨

메모리/플래쉬 Layout과 관련한 상수 정의

“include/blob/arch/xhyper320tku.h”

```
#define BLOB_ABS_BASE_ADDR (0x80008000)
/* where do various parts live in RAM */
#define BLOB_RAM_BASE (0x80200000)
#define KERNEL_RAM_BASE (0x80800000)
#define PARAM_RAM_BASE (0x80210000)
#define RAMDISK_RAM_BASE (0x80600000)
#define KERNEL_XIP_BASE (0x10040000) /* define kernel_xip_base */
/* where do they live in flash */
#define BLOB_FLASH_BASE (0x10000000)
#define BLOB_FLASH_LEN (256 * 1024)
#define PARAM_FLASH_BASE (BLOB_FLASH_BASE + BLOB_FLASH_LEN)
#define PARAM_FLASH_LEN (0) /* no parameters */
#define KERNEL_FLASH_BASE (PARAM_FLASH_BASE + PARAM_FLASH_LEN)
#define KERNEL_FLASH_LEN (2 * 1024 * 1024)
#define LOAD_RAMDISK 0 /* load ramdisk into ram */
#define RAMDISK_FLASH_BASE (KERNEL_FLASH_BASE + KERNEL_FLASH_LEN)
#define RAMDISK_FLASH_LEN (4 * 1024 * 1024)
/* the position of the kernel boot parameters */
#define BOOT_PARAMS (0x80000100)
/* the size (in kbytes) to which the compressed ramdisk expands */
#define RAMDISK_SIZE (8 * 1024)
/* TFTP download RAM base */
#define TFTP_RAM_START (0x80800000)
```

Makefile

OCFLAGS = -O binary -R .note -R .comment -S

object file의 symbol 및 relocation 정보 제거하고 내용의 memory dump 생성

```
blob-rest$(EXEEXT): blob-rest-elf32
$(OBJCOPY) $(OCFLAGS) $< $@
```

create an object file from the binary so we can link it into the first stage loaders

```
blob-rest-piggy.o: blob-rest
```

```
$(LD) -r -o $@ -b binary $<
```

```
blob$(EXEEXT): blob-elf32
$(OBJCOPY) $(OCFLAGS) $< $@
```

\$< : 필요항목 이름
\$@ : 타겟 이름

```
blob-elf32$(EXEEXT): $(blob_elf32_OBJECTS) $(blob_elf32_DEPENDENCIES)
```

```
@rm -f blob-elf32$(EXEEXT)
```

```
$(LINK) $(blob_elf32_LDFLAGS) $(blob_elf32_OBJECTS) \
$(blob_elf32_LDADD) $(LIBS)
```

start-ld-script 사용

```
blob-rest-elf32$(EXEEXT): $(blob_rest_elf32_OBJECTS) $(blob_rest_elf32_DEPENDENCIES)
```

```
@rm -f blob-rest-elf32$(EXEEXT)
```

```
$(LINK) $(blob_rest_elf32_LDFLAGS) $(blob_rest_elf32_OBJECTS) \
$(blob_rest_elf32_LDADD) $(LIBS)
```

rest-ld-script 사용

main 코드 – main.c

- main 함수 작업 과정
 - blob_status 구조체 초기화 – serial port 설정, parameter 등의 설정
 - serial port 설정 : `serial_init()`
 - init 영역 초기화 함수 실행 : `init_subsystems()`
 - GPL 문자 serial 출력
 - 메모리 양 확인
 - 플래쉬로부터 RAM으로 커널과 ramdisk를 load: `do_reload()`
 - 지정된 시간 동안 키 입력 기다림
 - 입력이 있으면: 해당 루틴 실행
 - 입력이 없으면: 자동 부팅 실행

main.c

```
int main(void)
{
    int numRead = 0;
    char commandline[MAX_COMMANDLINE_LENGTH];
    u32 conf = 0;
    void *params;

    /* initialise status */
    blob_status.paramType = fromFlash;
    blob_status.kernelType = fromFlash;
    blob_status.ramdiskType = fromFlash;
    blob_status.downloadSpeed = baud_38400;
    blob_status.terminalSpeed = TERMINAL_SPEED;
    blob_status.load_ramdisk = LOAD_RAMDISK;
    blob_status.cmdline[0] = '\0';
    blob_status.boot_delay = 3;

    serial_driver = &mhn_serial_driver;
    serial_init(TERMINAL_SPEED);

    /* call subsystems (blob_status.* may change) */
    init_subsystems();           // init section 초기화 함수 실행
}
```

main.c (계속)

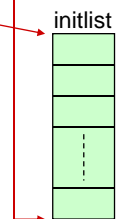
```
/* print the required GPL string */
...
get_memory_map();           // get the amount of memory
...
if (params)
    parse_ptags(params, &conf); // parse all the tags
...
do_reload("kernel");        // KERNEL_RAM_BASE에 kernel를 load
do_reload("ramdisk");       // RAMDISK_RAM_BASE에 ramdisk를 load
...
if (retval == 0) {           // key input
    commandline[0] = '\0'; parse_command("nkernel");
    commandline[0] = '\0'; parse_command("boot");
}
printf("\nAutoboot aborted\n");
ether_init();

/* infinite command loop */
```

main 호출 subroutine – init_subsystem

```
■ init_subsystem()      (lib/init.c)
void init_subsystems(void)
{
    int i;
    /* call all subsystem init functions (initial section에 위치) */
    for (i = INIT_LEVEL_MIN; i <= INIT_LEVEL_MAX; i++)
        call_funcs((initlist_t *)&__initlist_start, (initlist_t *)&__initlist_end,
                    INIT_MAGIC, i);
}

/* init.h */
typedef void (*initfunc_t)(void);
typedef struct {
    u32 magic;      initfunc_t callback;    int level;
} initlist_t;
```



main 호출 subroutine – init_subsystem(2)

■ call_funcs() (lib/init.c)

```
static void call_funcs(initlist_t *start, initlist_t *end, u32 magic, int level)
{
    initlist_t *item;

    for(item = start; item != end; item++) {
        if(item->magic != magic) { /* magic값 검사 */
            printerror(EMAGIC, NULL);
            ...
            return;
        }
        if(item->level == level) { /* level값 검사, 같으면 함수 호출 */
            item->callback(); /* call callback function */
        }
    }
}
```

main 호출 subroutine – init_subsystem(3)

■ initlist와 관련된 매크로

```
/* init.h */
#define __init __attribute__((unused, __section__(".initlist")))

#define __initlist(fn, lvl) \
static initlist_t __init_##fn __init = { \
    magic: INIT_MAGIC, \
    callback: fn, \
    level: lvl }

/* rest-ld-script */
...
. = ALIGN(4);
.initlist : {
    __initlist_start = .;
    *(.initlist)
    __initlist_end = .;
}
```

// .initlist section에 위치하는 변수 선언용

__init_fn 변수는 초기화 선언되며 .initlist 섹션에 위치함.

##: 두 문자열을 합하는 #define에서 사용하는 연산자

INIT_MAGIC: initlist 엔트리임을 나타내기 위한 값

main 호출 subroutine – init_subsystem(4)

■ __initlist 매크로 사용 예

```
/* xhyper320tku.c: Hybus X-hyper320TKU specific stuff */
...
void xhyper320tku_init_hardware(void)
{
    .....
}

__initlist(xhyper320tku_init_hardware, INIT_LEVEL_DRIVER_SELECTION);
```

.initlist 섹션에 구조체 변수 __init_xhyper320tku_init_hardware가 선언되며 이 구조체 변수의 callback 함수로 xhyper320tkuo_init_hardware가 지정됨

main 호출 subroutine – parse_command

■ parse_command() (lib/command.c)

```
int parse_command(char *cmdline)
{
    commandlist_t *cmd;
    int argc, num_commands, len;
    char *argv[MAX_ARGS];

    parse_args(cmdline, &argc, argv);
    /* only whitespace */
    if (argc == 0) return 0;

    num_commands = get_num_command_matches(argv[0]);
    if (num_commands < 0) return num_commands; /* error */
}
```

```
/* command.h */
typedef int (*commandfunc_t)(int, char *[]);
typedef struct commandlist {
    u32 magic;
    char *name;
    char *help;
    commandfunc_t callback;
    struct commandlist *next;
} commandlist_t;
```

commandlist에서 명령어 검색하여 match된 개수 반환

main 호출 subroutine – parse_command(2)

(계속)

```
if (num_commands < 0) return num_commands; /* error */
if (num_commands == 0) return -ECOMMAND; /* no command matches */
if (num_commands > 1) return -EAMBIGCMD; /* ambiguous command */
len = strlen(argv[0]);
/* single command, go for it */
for(cmd = commands; cmd != NULL; cmd = cmd->next) {
    if(cmd->magic != COMMAND_MAGIC)
        return -EMAGIC;
}
if(strncmp(cmd->name, argv[0], len) == 0) {
    /* call function */
    return cmd->callback(argc, argv);
}
return -ECOMMAND; /* 일치하는 것이 없음 */
}
```

부트로더

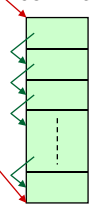
37

main 호출 subroutine – parse_command(3)

■ init_commands() (lib/command.c)

```
commandlist_t *commands;
static void init_commands(void)
{
    commandlist_t *lastcommand;
    commandlist_t *cmd, *next_cmd;
    commands = (commandlist_t *) &__commandlist_start;
    lastcommand = (commandlist_t *) &__commandlist_end;
    cmd = next_cmd = commands;
    next_cmd++;
    while(next_cmd < lastcommand) {
        cmd->next = next_cmd;
        cmd++;
        next_cmd++;
    }
    __initlist(init_commands, INIT_LEVEL_OTHER_STUFF);
}
```

command 정보의 연결리스트 생성



initlist에 init_commands 등록

부트로더

38

main 호출 subroutine – parse_command(4)

■ commandlist 관련 매크로

```
/* command.h */
#define __command__attribute__((unused, __section__(".commandlist")))
#define __commandlist(fn, nm, hlp) \
static commandlist_t __command_##fn __command = { \
    magic:  COMMAND_MAGIC, \
    name:   nm, \
    help:   hlp, \
    callback: fn \
}
```

.commandlist 섹션에 위치하는 구조체 변수 생성

부트로더

39

main 호출 subroutine – parse_command(5)

■ __commandlist 사용 예

```
/* main.c */
...
static int Reload(int argc, char *argv[])
{
    if(argc < 2) return -ENOPARAMS;
    return do_reload(argv[1]);
}

static char reloadhelp[ ] = "reload {blob|param|kernel|ramdisk}\n"
    "Reload <argument> from flash to RAM\n";

__commandlist(Reload, "reload", reloadhelp);
```

부트로더

40

main 호출 subroutine – parse_command(6)

■ command 추가 방법

```
static int example_command(int argc, char *argv[])
{
    int i;
    for( i = 0; i < argc; i++ ) {
        SerialOutputString("argv[");
        SerialOutputDec(i);
        SerialOutputString("] = \"");
        SerialOutputString(argv[i]);
        SerialOutputString("\n");
    }
    return 0;
}

static char examplehelp[] = "example command\n";
__commandlist(example_command, "example", examplehelp);
```

boot kernel

```
static void (*ramKernel)(int zero, int arch, u32 params) =
    (void (*)(int, int, u32)) KERNEL_RAM_BASE;
static int boot_linux(int argc, char *argv[])
{
    ..... setup tags .....

    printf("\nStarting kernel at 0x%08x...\n\n", ramKernel);
    serial_flush_output();

    /* disable subsystems that want to be disabled before kernel boot */
    exit_subsystems();           // blob 종료시 실행 함수 수행

    /* start kernel */
    ramKernel(0, ARCH_NUMBER, BOOT_PARAMS);
    // r0, r1, r2 = arguments lr=pc, pc=KERNEL_RAM_BASE
    printf("Hey, the kernel returned! This should not happen.\n");
    return 0;
}
```