멀티프로세싱 예제 매뉴얼

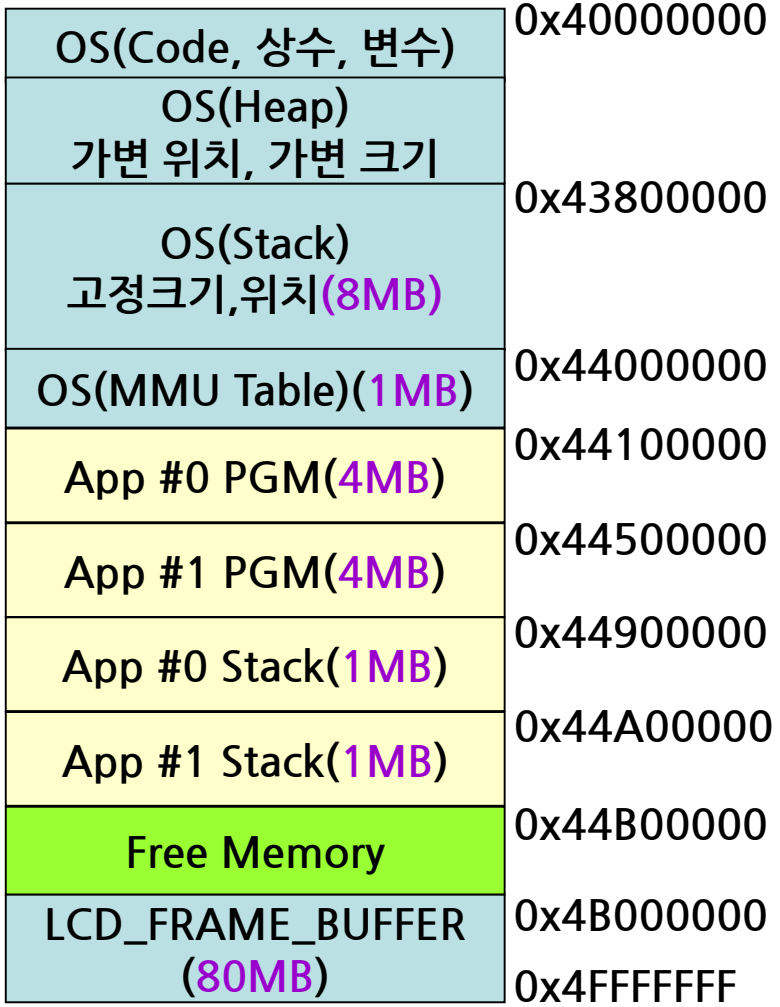
(WT4412, uBOOT 기반)

임베디드전문가 그룹 윌텍

멀티프로세싱 예제 메모리 레이아웃

CODE XPERT
임베디드 전문가 그룹 윌텍

- 멀티프로세싱을 위한 예제 프로그램의 메모리 레이아웃은 다음과 같다




```
#define     RAM_APP0            0x44100000
#define     RAM_APP1            (RAM_APP0+SIZE_APP0)
#define     SIZE_STACK0         (1*1024*1024)
#define     SIZE_STACK1         (1*1024*1024)
#define     STACK_LIMIT_APP0    (RAM_APP1+SIZE_APP1)
#define     STACK_LIMIT_APP1    (STACK_LIMIT_APP0+SIZE_STACK0)
#define     STACK_BASE_APP0     (STACK_LIMIT_APP0+SIZE_STACK0)
#define     STACK_BASE_APP1     (STACK_LIMIT_APP1+SIZE_STACK1)

#define     SIZE_APP0           (4*1024*1024)
#define     SIZE_APP1           (4*1024*1024)

#define     SECTOR_APP0         100
#define     SECTOR_APP1         5000
```

OS 역할을 수행하는 예제 프로젝트 main.c의 정의

Multi L&E 기반 APP 구조 #1

App0 예제 프로그램 : 900.APP0_0x44100000 프로젝트

- RO-BASE ➔ 0x44100000, DRAM 주소 ➔ 0x44100000
  - Compile하면 “app0_0x44100000.bin”이 C:₩OpenTFTPServer₩home에 복사된다

```
int Main(void)
{
    Uart_Printf(">>APP0\n");

    return 0;
}
```

Main.c

단순히 APP0 문자열만 인쇄하고 OS로 복귀

```
MEMORY
{
    RAM  (rwx) : ORIGIN = 0x44100000, LENGTH = 56M
}
```

dram_0x44100000.lds

RO-BASE를 0x44100000 으로 설정
실제로 다운로드될 주소임

```
# Project name & Link script
OUT_FILE_NAME   = app0_0x44100000
LDS_FILE_NAME   = dram_0x44100000.lds
```

Makefile

컴파일이 완료되면 app0_0x44100000.bin 생성
자동으로 C:₩OpenTFTPServer₩home 폴더로 이동

Multi L&E 기반 APP 구조 #2

CODEXPERT
임베디드 전문가 그룹 윌텍

App1 예제 프로그램 : 901.APP1_0x44500000 프로젝트

- RO-BASE ➔ 0x44500000, DRAM 주소 ➔ 0x44500000
  - Compile하면 “app1_0x44500000.bin”이 C:₩OpenTFTPServer₩home에 복사된다

```
int Main(void)
{
    Uart_Printf(">>APP1\n");

    return 0;
}
```

단순히 APP1 문자열만 인쇄하고 OS로 복귀

Main.c

```
MEMORY
{
    RAM  (rwx) : ORIGIN = 0x44500000, LENGTH = 56M
}
```

RO-BASE를 0x44500000 으로 설정
실제로 다운로드될 주소임

dram_0x44500000.lds

```
# Project name & Link script
OUT_FILE_NAME    = app1_0x44500000
LDS_FILE_NAME    = dram_0x44500000.lds
```

컴파일이 완료되면 app1_0x44500000.bin 생성
자동으로 C:₩OpenTFTPServer₩home 폴더로 이동

Makefile

001.Loader_Executor 코드 분석

CODE XPERT
임베디드 전문가 그룹 윌텍

Main 함수의 구성은 다음과 같다

```
int Main(void)
{
    Uart_Printf("APP0 Loading!\n");
    App_Read(SECTOR_APP0, SIZE_APP0, RAM_APP0);
    Uart_Printf("APP1 Loading!\n");
    App_Read(SECTOR_APP1, SIZE_APP0, RAM_APP1);

    for(;;)
    {
        Uart_Printf("\nAPP0 RUN\n");
        Run_App(RAM_APP0, STACK_BASE_APP0);
        Key_Wait_Key_Released();

        Uart_Printf("\nAPP1 RUN\n");
        Run_App(RAM_APP1, STACK_BASE_APP1);
        Key_Wait_Key_Released();
    }
}
```

Main.c

SD 100 섹터에서 두 APP을 읽어서
0x44100000 번지에 저장한다
다운로드 주소, SD 섹터번호, 크기 등은 조정가능
uBOOT에서 APP0와 1을 직접 DRAM으로 적재하는 경우 이 기능은 제외시키고 실험한다

보드의 Key를 누를때마다 App0, App1 교차 실행

Run_App 함수 분석

```
    .global Run_App
Run_App:
    push    {r4, lr}
    mrs     r4, cpsr
    cps     #0x1f
    mov     sp, r1
    blx     r0
    msr     cpsr_cxsf, r4
    pop     {r4, pc}
```

Asm_function.s

1. 모드를 SYS로 바꾸고 전달받은 Stack_Base를 SP에 저장
2. APP이 적재된 주소로 함수 호출
3. APP에서 복귀하면 백업한 CPSR 복원 ➔ 이전 모드(SVC)로 전환
4. APP으로 분기하는 것이지만 함수 호출과 동일한 개념
   ➔ Scratch 사용은 가능하지만 그 외 레지스터들은 보존해야 함

실습 1. 두 응용 프로그램의 교차 실행

CODEXPERT
임베디드 전문가 그룹 윌텍

001.Loader_Executor 프로젝트 컴파일 및 실행

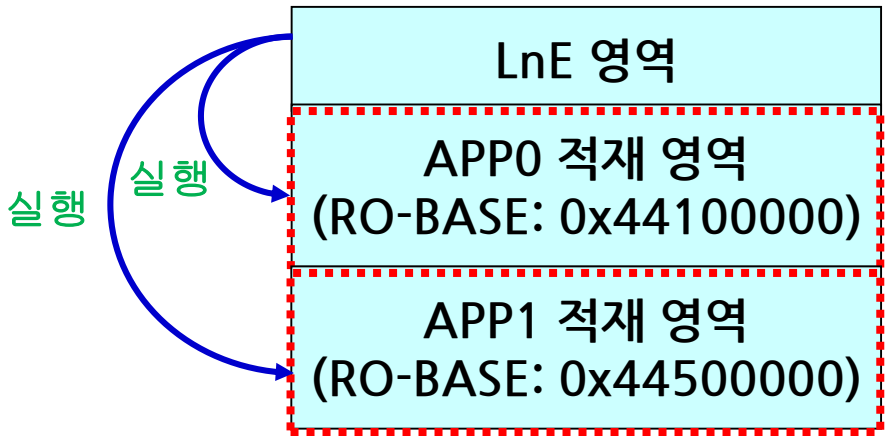
- 다음과 같이 APP0, APP1과 OS를 실행시킨다

```
WT4412 # tftpboot 44100000 app0_0x44100000.bin
WT4412 # tftpboot 44500000 app1_0x44500000.bin
WT4412 # tftpboot 40000000 dram_0x40000000.bin
WT4412 # go 40000000
```

- SETENV 및 SAVEENV로 자동 실행되도록 할 수도 있다

```
WT4412 # setenv bootcmd 'tftpboot 44100000 app0_0x44100000.bin; tftpboot 44500000 app1_0x44500000.bin;
                         tftpboot 40000000 dram_0x40000000.bin; go 40000000'
WT4412 # saveenv
```

- 001.Loader_Executor의 동작 ➔ 보드의 Key를 누를때 마다 두 APP을 교차 실행



0x40000000 : LnE 적재 위치

0x44100000 : APP0 적재 위치

0x44500000 : APP1 적재 위치

App0와 App1의 구조

CODEXPERT
임베디드 전문가 그룹 윌텍

- App의 Crt0.s 파일의 코드는 다음과 같은 형식을 갖는다
  - 인터럽트나 Exception, malloc 등 사용 불가 ➔ 모든 Exception은 OS에서 관리
    - SP을 설정하지 않음 ➔ OS 역할을 수행하는 L&E에서 App 호출시 SP를 초기화 함
    - Heap은 OS가 관리하는 영역이며 System Call에 의하여 할당 및 반환이 가능하다
    - 예제는 System Call 자원관리를 지원하지 않으므로 App에서 힙 관련 함수는 사용 불가함

```
    .global Run_App
Run_App:
    push    {r4, lr}
    mrs     r4, cpsr
    cps     #0x1f
    mov     sp, r1
    blx     r0
    msr     cpsr_cxsf, r4
    pop     {r4, pc}
```

OS의 Asm_function.s

```
    .global __start
__start:
    push {r0-r3, r12, lr}

    .extern __ZI_BASE__
    .extern __ZI_LIMIT__
    .extern Main

    ldr     r0, =__ZI_BASE__
    ldr     r1, =__ZI_LIMIT__
    mov     r2, #0x0
1:
    cmp     r0, r1
    strlo   r2, [r0], #4
    blo     1b

    bl      Main

    pop {r0-r3, r12, pc}
```

APP의 Crt0.s

독립적인 프로그램이지만 OS에서 호출되는 함수처럼 설계해야 한다
App은 Exception을 지원하지 않으므로 벡터테이블이 없다
또한, OS에서 App 실행시 SP를 설정하므로 SP 설정도 불필요하다

C 코드를 위한 Crt_Startup

멀티프로세싱을 위한 수정된 APP1 분석

App1의 수정된 예제 프로그램 : 902.APP1_0x44100000 프로젝트

- RO-BASE ➔ 0x44100000, DRAM 주소 ➔ 0x44500000
  - 즉, 0x44500000 번지에 적재되는 APP1이지만 RO-BASE를 APP0와 동일하게 한다

```
int Main(void)
{
    Uart_Printf(">>APP1, RO-BASE = 0x44100000\n");

    return 0;
}
```

Main.c

기존 APP1과 인쇄하는 문자열을 다르게 한다

```
MEMORY
{
    RAM  (rwx) : ORIGIN = 0x44100000, LENGTH = 56M
}
```

dram_0x44100000.lds

RO-BASE를 0x44100000 으로 설정
OS는 APP0, APP1 모두 동일 VA로 인식
다만, APP1의 PA는 0x44500000 번지임

```
# Project name & Link script
OUT_FILE_NAME    = app1_0x44100000
LDS_FILE_NAME    = dram_0x44100000.lds
```

Makefile

컴파일이 완료되면 app1_0x44100000.bin 생성
자동으로 C:₩OpenTFTPServer₩home 폴더로 이동

실습 2. 멀티프로세싱을 위한 eSOS 동작

CODEXPERT
임베디드 전문가 그룹 윌텍

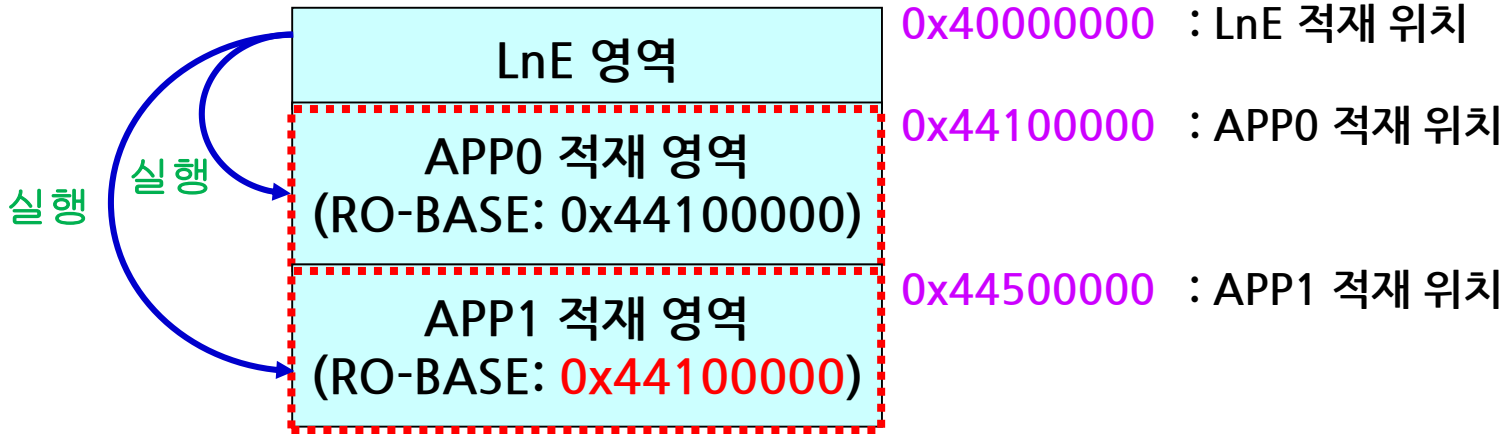
- eSOS(embedded Simple OS) ➔ 002.OS_Template 프로젝트 컴파일
  - 다음과 같이 APP0, APP1과 OS를 실행시킨다
    - APP1의 RO-BASE는 0x44100000
    - 그러나 APP1은 다운로드 주소는 0x44500000

```
WT4412 # tftpboot 44100000 app0_0x44100000.bin
WT4412 # tftpboot 44500000 app1_0x44100000.bin
WT4412 # tftpboot 40000000 dram_0x40000000.bin
WT4412 # go 40000000
```

  - SETENV 및 SAVEENV로 자동 실행되도록 할 수도 있다

```
WT4412 # setenv bootcmd 'tftpboot 44100000 app0_0x44100000.bin; tftpboot 44500000 app1_0x44100000.bin;
                         tftpboot 40000000 dram_0x40000000.bin; go 40000000'
WT4412 # saveenv
```

  - 002.OS_Template 의 동작 ➔ RO-BASE가 동일한 두 APP을 교차로 실행



eSOS의 APP별 MMU 설정

CODE XPERT
임베디드 전문가 그룹 윌텍

- eSOS(Edunix Simple O.S)는 각 APP 실행전에 MMU-T/T를 설정한다

```
void Main(void)
{
    Uart_Printf("APP0 Loading!\n");
    App_Read(SECTOR_APP0, SIZE_APP0, RAM_APP0);
    Uart_Printf("APP1 Loading!\n");
    App_Read(SECTOR_APP1, SIZE_APP0, RAM_APP1);

    for(;;)
    {
        Uart_Printf("\n실행할 APP을 선택하시오 [1]APP0, [2]APP1 >> ");
        x = Uart1_Get_Char();

        if(x == '1')
        {
            Uart_Printf("\nAPP0 RUN\n", x);
            SetTransTable(RAM_APP0, (RAM_APP0+SIZE_APP0-1), RAM_APP0, RW_WBWA);
            SetTransTable(STACK_LIMIT_APP0, STACK_BASE_APP1-1, STACK_LIMIT_APP0, RW_WBWA);
            CoInvalidateMainTlb();
            Run_App(RAM_APP0, STACK_BASE_APP0);
        }

        if(x == '2')
        {
            Uart_Printf("\nAPP1 RUN\n", x);
            SetTransTable(RAM_APP0, (RAM_APP0+SIZE_APP1-1), RAM_APP1, RW_WBWA);
            SetTransTable(STACK_LIMIT_APP1, STACK_BASE_APP1-1, STACK_LIMIT_APP1, RW_WBWA);
            CoInvalidateMainTlb();
            Run_App(RAM_APP0, STACK_BASE_APP1);
        }
    }
}
```

SD 100, 5000 섹터에서 두 APP을 읽어서 0x44100000, 0x44500000 번지에 저장한다

APP0: VA 0x441섹션 ➔ PA 0x441섹션

APP1: VA 0x441섹션 ➔ PA 0x445섹션

Main.c

프로젝트를 위한 비복귀용 대용량 APP

- 멀티프로세싱 실험용 APP 코드가 너무 짧으면?
  - 프로세스 스위칭을 위한 Tick Time 발생 전에 코드가 실행이 종료될 수 있다
    - 따라서 멀티프로세싱 실험을 위하여 크기도 크고 비복귀하는 APP을 설계한다
  - 기존 APP0 교체 ➔ 903.MULTI_APP0_0x44100000 프로젝트
    - multi_app0_0x44100000.bin이 생성된다
    - LCD에 2장의 큰 그림을 무한히 교대로 디스플레이 한다 ➔ 파일 크기 2,087,664 바이트
  - 기존 APP1 교체 ➔ 904.MULTI_APP1_0x44100000 프로젝트
    - multi_app1_0x44100000.bin이 생성된다
    - UART에 숫자를 무한히 인쇄한다 ➔ 파일 크기 38,144 바이트
- APP0, APP1을 기존 APP과 교체한 후 002.OS_Template 프로젝트 실행
  - APP 번호를 선택하여 원하는 APP을 실행
    - 단, APP이 무한루프 구조이므로 다른 APP을 실행하고자 하면 보드를 리셋시켜야 한다
    - 이 APP을 기반으로 멀티프로세싱 OS 스케쥴러를 설계한다