LG 부트캠프 1기 3차 2반 5팀

프로젝트 결과보고서

ARM Cortex-A MP Core 기반 OS 설계

Embedded System Path 2반 5팀
김소흥 연구원
이해건 연구원
조나경 연구원

목차구성 CONTENTS COMPOSITION

1. 주제 및 결과 요약

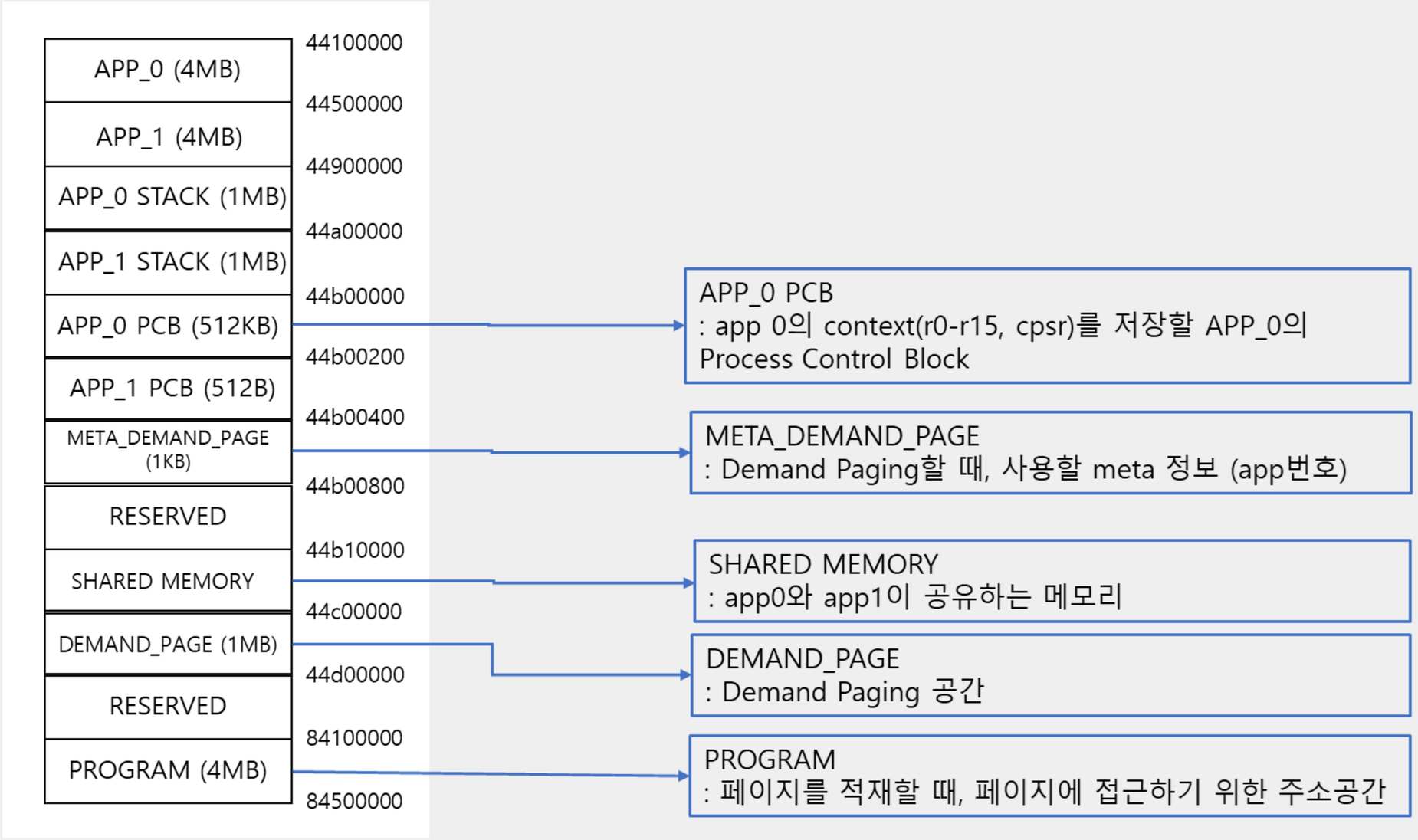
○ Project 주제 및 목표

- ARM Cortex-A MP Core환경에서 동작하는 OS를 설계한다.
- 설계한 OS는 멀티 프로세싱, 시스템 콜 서비스, Demand Paging을 제공한다.

○ Project 진행 결과

- 멀티 프로세싱 구현 완료
- 시스템 콜 서비스 구현 완료
- Cache를 사용한 Demand Paging 구현 완료

2. 개발 상세_메모리 Layout



2. 개발 상세_멀티 프로세싱

○ 목표

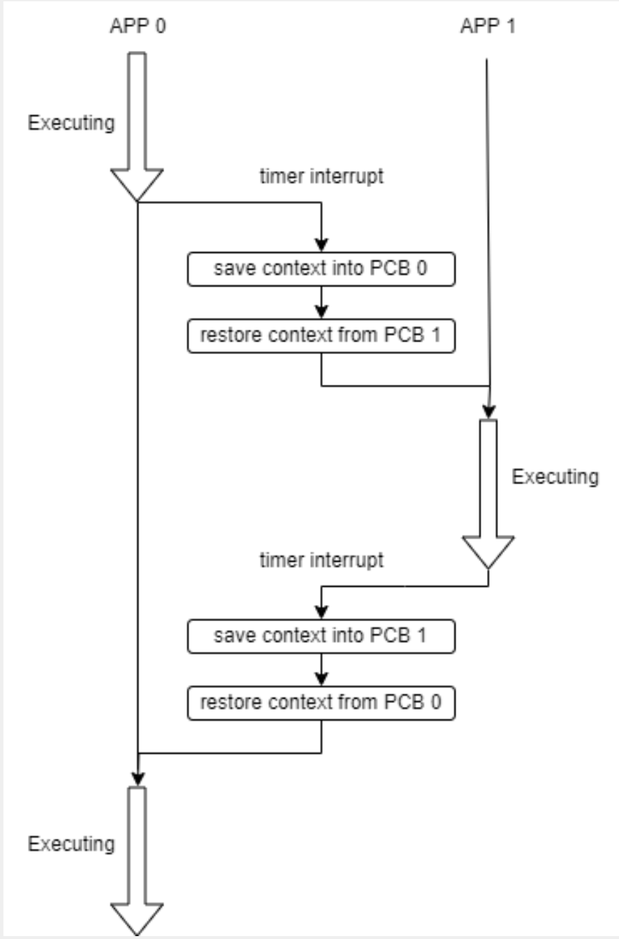
- Context Switch를 구현하여 멀티 프로세싱을 지원한다

○ 개발 상세

(1) PCB(Process Control Block) 설계
- app0 와 app1의 context를 담을 PCB 설계
- 프로그램 실행 초기에 context(r0-r15, cpsr) 저장

(2) Context Switch 구현
- Timer에 의해 Interrupt 발생
- IRQ Handler에 의해 Context Switch 수행
- 실행하고 있는 app의 context는 Store/실행할 app의 context는 Load

*context switch process



2. 개발 상세_시스템 콜 서비스

○ 목표

- OS 커널이 제공하는 서비스에 대해, 응용 프로그램의 요청에 따라 커널에 접근하도록 한다.

○ 개발 상세

(1) 시스템 콜 서비스 설계

- 해당 서비스를 호출할 경우, SVC Handler호출
- SVC Handler에 의해 서비스 실행

*SVC Handler

```
HandlerSVC:
    push    {r4-r6, lr}
    @@@@@ get SVC NUM
    ldr     r4, [lr, #-4]
    bic     r4, r4, #0xff000000
    @@@@@ sysmode
    cps     #0x1f
    @@@@@ call SVC HANDLER
    ldr     r12, =SVC_Vector
    ldr     r5, [r12, r4, lsl #2]
    @@@@@ backup
    mov     r6, lr
    @@@@@ SVC vector
    blx     r5
    @@@@@ restore
    mov     lr, r6

    cps     #0x13

    ldmfd   sp!, {r4-r6, pc}^
```

2. 개발 상세_Demand Paging

○ 목표

- 프로세스에서 사용할 메모리 페이지를 생성 시에 모두 할당하지 않고,
필요할 때마다 동적으로 할당하도록 구현한다

○ 개발 상세

(1) MMU 2nd Translation Table 설계

- MMU 2nd Translation Table 설계 수행
- 이 때 1차 테이블의 도메인 설정은 client로 설정한다.
- 2차 테이블의 AP속성은 permission fault가 발생할 수 있도록 000(no access)로 설정한다.

```
void SetAppTransTablePage(unsigned int pTT_1st, unsigned int uVaStart, unsigned int uVaEnd, unsigned int acf_2nd)
{
    unsigned int i;
    unsigned int* ptt_2nd = 0;

    unsigned int sizeOfPage = 1 << 12; // 4kB

    for (i = uVaStart; i < uVaEnd; i += sizeOfPage)
    {
        unsigned int secondIndex = (i & 0xff000) >> 10;
        ptt_2nd = (unsigned int*) ((pTT_1st & ~0x3ff)|secondIndex);
        *ptt_2nd = acf_2nd|(1 << 1);
    }
}
```

*2차 T/T 설계

2. 개발 상세_Demand Paging

(2)-1 .페이지 요청 시, Demand Paging공간에 페이지 적재

- 페이지 요청 시, Permission Fault로 인해 DABT, PABT발생
- Handler에서 페이지 적재 수행
- 페이지 적재 완료 후에는 2차 T/T테이블을 통해 PA에 접근할 수 있도록 2nd Descriptor 내용 업데이트
- AP bit(011), PA(해당 Demand Paging공간 주소) update

```
int i = 0;
int curAppNum = getCurAppNum();
unsigned int ttbr[] = {MMU_PAGE_TABLE_BASE, MMU_PAGE_TABLE_BASE + (1 << 14)};
unsigned int asid[] = {(1 << 4), (1 << 4)|1};
addr = addr & ~0xfff;

unsigned int* source_va = (unsigned int*) (addr+(1 << 30));
unsigned int* nextMetaDemandPage = (unsigned int*) (metaDemandPageBase + curDemandPage); //초기화 필요
unsigned int* nextDemandPage = (unsigned int*) (demandPageBase + (curDemandPage << 12));

for(i = 0; i < 1024; i++)
{
    *(nextDemandPage + i) = *(source_va + i);
}
```

* 페이지 적재

```
void set2ndTTAdrress(unsigned int uVa, unsigned int sourceAdrress, int appNum, unsigned int acf)
{
    unsigned int* targetAdrress = get2ndTTAdrress(uVa, appNum);
    *targetAdrress = (sourceAdrress & ~0xfff)|acf|(1<<1);
}
```

* 2nd Descriptor 내용 업데이트

2. 개발 상세_Demand Paging

(2)-2. Demand Paging공간이 다 차 있을 때 페이지 요청 시, Swap-out 수행

- 기존에 Demand Paging공간에 적재되어 있던 페이지 Swap-out 수행
- 2차 T/T테이블을 통해 접근할 수 없도록 2nd Descriptor 내용 업데이트 (AP bit를 000으로 update)
- 새로운 페이지는 (2)-1과 동일하게 수행하여 적재한다.

```
// 이미 값이 들어있을 때 처리
if (*nextMetaDemandPage != 1 << 31)
{
    int appNum = *nextMetaDemandPage & 0xfff;
    unsigned int address = *nextMetaDemandPage & ~0xfff;
    address = address - (1 << 30);

    CoSetASID(asid[appNum]);
    CoSetTTBase(ttbr[appNum]|(1<<6)|(1<<3)|(0<<1)|(0<<0));

    restoreDemandPage(nextDemandPage, *nextMetaDemandPage);
    set2ndTTAdrress(address, 0, appNum, PAGE_2ST_RW_NCNB_LOCAL_NO_ACCESS);

    CoSetASID(asid[curAppNum]);
    CoSetTTBase(ttbr[curAppNum]|(1<<6)|(1<<3)|(0<<1)|(0<<0));

    L2C_CleanAndInvalidate_All();
}
```

* Swap-out

3. 결과 분석 및 기대 효과

개발 목표	목표 달성률	기대 효과	미완성 항목	개선책
멀티 프로세싱	100%	프로세스가 동시 실행될 수 있다.	-	-
시스템 콜 서비스	100%	사용자 모드가 커널영역의 기능 까지 사용할 수 있게 된다.	-	-
Demand Paging	100%	물리적 메모리보다 큰 용량의 메 모리를 지원할 수 있다.	-	캐시 성능 최적화

4. 향후 연구 과제

○ Demanding page 에서의 cache 최적화

- 현재 간단하게 구현된 cache 성능을 최적화 한다.

○ IRQ Handler 설계

- IRQ 인터럽트가 발생할 경우 기존에 구현된 핸들러를 사용하는 것이 아니라, 직접 처리하도록 구현할 수 있다.

5. 프로젝트 수행 후기

○ 김소흥 연구원

임베디드 시스템에 전원이 들어오고 나서부터 OS가 실행되고 APP이 실행되기 까지의 모든 과정을 직접 구현해봄으로써 시스템 전체에 관한 이해력을 높일 수 있었습니다. C언어의 함수와 어셈블리의 함수를 혼합해 사용하는 과정에서 레지스터 정보가 망가지고 이를 처리하는 과정이 가장 어려웠는데 분기가 일어날 때 마다 모든 레지스터에 저장된 값을 손으로 적어가며 코드를 작성하는 방식으로 해결했습니다.

○ 이해건 연구원

현업에서는 이미 세팅되어 있는 임베디드 장치에서 어플리케이션을 다뤘는데, 어셈블리 언어까지 사용하며 low level 의 환경을 경험해봄으로써 자사 제품에 대한 이해도를 높일 수 있었습니다. 마지막 프로젝트인 demanding page 구현에서 cache 를 적용하는 것에 어려움이 있었는데, 구현에 앞서 각 테이블 적용시켜야 하는 cache 정책을 단계별로 정리하고 적용시킴으로써 해결했습니다.

○ 조나경 연구원

하루나 이틀 단위로 정해진 기한내에 각 주제의 프로젝트를 완성하는 과정을 통해 현업에서도 적용할 수 있는 한정된 시간 내에서의 문제 해결력을 키울 수 있었습니다. Page table 을 구현하는 과정에서 비트 단위의 연산과 16진수를 혼합하여 사용하는 것에 어려움이 있었는데, 디버깅 과정에서 모든 레지스터나 변수를 포함하여 로그를 출력함으로써 해결할 수 있었습니다.