TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 : Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 : 황수정
sue100012@naver.com
2 개월 차 시험 오답노트

[임베디드 애플리케이션 구현 27] 프로그램을 실행한다고 하면 fork(), execve()의 콤보로 이어지게 된다. 이 때 실제 gcc *.c로 컴파일한 a.out을 ./a.out을 통해 실행한다고 가정한다. 실행을 한다고 하면 a.out File의 Text 영역에 해당하는 부분을 읽어야 한다. 실제 Kernel은 무엇을 읽고 이 영역들을 적절한 값으로 채워주는가 ?

ELF Header와 Program Headers를 읽고 값을 적절하게 채운다.

- 실행파일을 실행하면 task_struct가 생성되기 때문이다. 파일이 태스크로 수행되기 위해서는 파일을 구성하고 있는 내용들이 메모리에 적재되어야 한다. CPU에서 연산을 해주어야 하는데, CPU는 디스크를 직접 접근할 수 없어, 메모리에 적재되어야 하기 때문이다. 디스크에 저장되어있는 파일의 내용은 가상메모리에 복사되어 mm_struct 필드에 보관되고, 물리메모리에는 ELF header를 읽어 초기값이 저장된다.

[임베디드 애플리케이션 구현 28] User Space에도 Stack이 있고 Kernel Space에도 Stack이 존재한다. 좀 더 정확히는 각각에 모두 Stack, Heap, Data, Text의 메모리 기본 구성이 존재한다. 그 이유에 대해 기술하시오.

C언어를 사용하기 위해서는 반드시 Stack이 필요하다. Kernel 영역에서도 동작하는 코드가 올라가기 위한 Text 영역 전역 변수가 있는 Data 영역, 동적 할당하는 Heap, 지역 변수를 사용하는 Stack이 존재한다. 이는 역시 User 영역에서도 동일하므로 양쪽에 모두 메모리 공간이 구성된다.

- 커널 수준에서 수행되는 코드는 바로 리눅스 그 자체이기 때문이다. 리눅스도 C와 어셈블리로 작성된 소프트웨어이기 때문에 수행되기 위해서는 스택이 필요한 것(C언어를 사용하기 위해서는 반드시 스택이 필요)이다.

[임베디드 애플리케이션 구현 29] VM(가상 메모리)와 PM(물리 메모리)를 관리하는데 있어 VM을 PM으로 변환시키는 Paging Mechanism에 대해 Kernel에 기반하여 서술하시오.
mm_struct에 pgd라는 필드가 있다. Page Directory를 의미하는 것으로 pgd -> pte -> page로 3단계 Paging을 수행한다. 각각 10bit, 10bit, 12bit로 VM의 주소를 쪼개서 Indexing을 한다.

- 프로그램이 실행되어 프로세스가 되면서 태스크를 생성한다. 디스크에 저장되어있는 파일의 내용은 가상메모리에 복사되어 mm_struct 필드에 보관되고, 물리메모리에는 ELF header를 읽어 초기값이 저장된다. 이때 가상주소를 물리 주소로 변환할 때 이용되는 것이 페이지 테이블이다. 페이지 테이블은 가상주소를 물리 주소로 변환하는 주소 변환 정보를 기록한 테이블이다. demand on paging이 되는데 이는 쉽게 말하면 요청이 들어오면 그 때, 메모리 관리에 들어간다는 것이다. 즉, 태스크는 가상메모리를 사용하고 실제로 참조될 때 물리메모리에 페이지 폴트 처리과정을 통해 적재되는 것이다.

  가상메모리 주소를 10bit/10bit/12bit로 분할해 linear address에 저장한 뒤 페이지 디렉토리의 엔트리에 가상주소의 처음 10bit 를 저장(페이지 디렉토리는 cr3레지스터에 저장)한다. 두번째 10bit는 이전의 엔트리가 다음 페이지 테이블을 가리키고 있고, 몇 번째 엔트리에 저장되어 있는지는 linear address를 보고 찾게 된다. 마지막 12bit는 실제 물리 메모리에 적재되게 된다.

[임베디드 애플리케이션 구현 30] MMU(Memory Management Unit)의 주요 핵심 기능을 모두 적고 간략히 설명하시오.

1. HAT(HW Address Translation) 는 가상 메모리 공간을 실제 물리 메모리 공간으로 변환한다.

2. TLB(Translation Lookaside Buffer) 는 가상 메모리 주소와 대응되는 물리 메모리 주소를 Caching한다.

- 단순하게 MMU에 기능만 적어서 오답노트를 하였습니다. 정확하게 잘 모르고 있었습니다.
  CPU는 가상 주소로부터 물리 주소로 변환을 담당하는 별도 하드웨어가 있다. 주소 변환을 위해서는 페이지 디렉터리와 페이지 테이블을 탐색해야 하는데, 이것을 하드웨어적으로 처리한다는 것이다. 이것이 MMU이다. MMU에 페이지 테이블의 시작점(pgd)과 변환하고자 하는 가상 주소를 입력으로 넣어주면 된다. MMU의 주요 핵심 기능은 가상 메모리 주소를 실제 메모리 주소로 변환하며, 메모리 보호, 캐시 관리, 버스 중재이다.

[임베디드 애플리케이션 구현 31] 하드디스크의 최소 단위를 무엇이라 부르고 그 크기는 얼마인가 ?

Sector, 512 byte

- 디스크 블록하고 헷갈려서 작성하였습니다.

[임베디드 애플리케이션 구현 32] Character 디바이스 드라이버를 작성할 때 반드시 Wrapping 해야 하는 부분이 어디인가 ? (Task 구조체에서 부터 연결된 부분까지를 쭉 이어서 작성하라)

task_struct->files_struct->file->file_operations 에 해당함.

[임베디드 애플리케이션 구현 33] 예로 유저 영역에서 open 시스템 콜을 호출 했다고 가정할 때 커널에서는 어떤 함수가 동작하게 되는가? 실제 소스 코드 차원에서 이를 찾아서 기술하도록 한다.

Kernel Source 에서 아래 파일에 위치한다. fs/open.c 에 위치하며 SYSCALL_DEFINE3(open, ~~~) 형태로 구동된다. 이 부분의 매크로를 분석하면 결국 sys_open() 이 됨을 알 수 있다.

[임베디드 애플리케이션 구현 35] VFS(Virtual File System)이 동작하는 Mechanism 에 대해 서술하시오.

VFS는 Super Block인 ext2_sb_info와 같은 것들을 읽어 super_block에 채워 넣는다. 그리고 읽고자 하는 파일에 대한 메타 정보를 ext2_inode와 같은 것들을 읽어 inode에 채운다.
이후에 디렉토리 엔트리인 ext2_dir_entry를 읽어 dentry 구조체에 채우고 현재 위치 정보에 대한 정보를 위해 path 구조체를 채운 이후 실제 File에 대한 상세한 정보를 기록하기 위해 file 구조체를 채운다.
각각 적절한 값을 채워넣어 실제 필요한 파일을 Task와 연결시킨다.

[임베디드 애플리케이션 구현 39] 예로 모니터 3 개를 쓰는 경우 양쪽에 모두 인터럽트를 공유해야 한다. Linux Kernel에서는 어떠한 방법을 통해 이들을 공유하는가?

외부 인터럽트의 경우 32 ~ 255까지의 번호를 사용한다. 여기서 128(0x80)에 해당하는 System Call만은 제외한다. idt_table에서 128을 제외한 32 ~ 255 사이의 번호가 들어오면 실제 HW Device다.
여기서 같은 종류의 HW Device가 들어올 수 있는데 그들에 대해서 Interrupt를 공유하게 하기 위해 irq_desc라는 Table을 추가로 두고 active라는 것으로 Interrupt를 공유하게끔한다.

[임베디드 애플리케이션 구현 40] System Call 호출시 Kernel에서 실제 System Call을 처리하기 위해 Indexing을 수행하여 적절한 함수가 호출되도록 주소값을 저장해놓고 있다. 이 구조체의 이름을 적으시오.

Intel 의 경우에 sys_call_table
ARM 의 경우에는 __vectors_start + 0x1000 에 해당함

[임베디드 애플리케이션 구현 41] 38에서 User Space에서 System Call 번호를 전달한다. Intel Machine에서는 이를 어디에 저장하는가 ? 또한 ARM Machine에서는 이를 어디에 저장하는가 ?

Intel 의 경우에는 ax 레지스터에 ARM 의 경우에는 r7 레지스터에 해당한다.

[임베디드 애플리케이션 구현 42] Paging Mechanism에서 핵심이 되는 Page Directory 는 mm_struct의 어떤 변수가 가지고 있는가 ?

pgd

[임베디드 애플리케이션 구현 43] 또한 Page Directory를 가르키는 Intel 전용 Register가 존재한다. 이 Register의 이름을 적고 ARM 에서 이 역할을 하는 레지스터의 이름을 적으시오.

Intel 의 경우엔 CR3
ARM 의 경우엔 CP15

[임베디드 애플리케이션 구현 44] 커널 내부에서 메모리 할당이 필요한 이유는 무엇인가?

커널 스택으로 주어진 메모리 공간은 고작 8K 에 해당한다. (물론 이 값은 ARM 이라고 가정하였을 때고 Intel 은 16 K 에 해당한다) 문제는 스택 공간이라 특정 작업을 수행하고 이후에 태스크가 종료되면 정보가 사라질 수도 있다. 이 정보가 없어지지 않고 유지될 필요가 있을 수도 있다.
뿐만 아니라 커널 자체도 프로그램이기 때문에 메모리 공간이 필요하다. 운영체제 또한 C 로 만든 프로그램에 불과하다는 것이다. 그러니 프로그램이 동작하기 위해 메모리를 요구하듯 커널도 필요하다.

- 커널 자체도 프로그램이라서 동작하기 위해서 스택 할당이 필요하고 이는 메모리 할당이 필요하다는 것을 쓰려 했는데 정작 다른 답안을 작성하였습니다. 또한 메모리 크기와 정보가 사라지지 않고 유지되어야 할 때 등의 이유는 기재하지 않았습니다.

[임베디드 애플리케이션 구현 45] 메모리를 불연속적으로 할당하는 기법은 무엇인가 ?
vmalloc()
[임베디드 애플리케이션 구현 46] 메모리를 연속적으로 할당하는 기법은 무엇인가 ?
kmalloc()

- 파일시스템이 저장 공간의 기본 접근단위인 디스크 블록을 관리하는 기법을 작성하였습니다. 불연속적, 연속적 할당이라 하면 이 것이 기본적으로 떠올라서..

[임베디드 애플리케이션 구현 47] Mutex 와 Semaphore 의 차이점을 기술하시오.

Mutex 와 Semaphore 모두 Context Switching 을 유발하게 된다. 차이점이라면 Mutex 는 공유된 자원의 데

이터를 **여러 쓰레드가 접근하는 것을 막는다.** Semaphore 는 공유된 자원의 데이터를 **여러 프로세스가 접근하는 것을 막는 것**이다.

- Semaphore가 무엇인지, 무엇을 방지하기 위해서 사용되는지만 작성하였습니다. 정작 중요한 차이점을 기술하지 않았습니다.

[임베디드 애플리케이션 구현 51] Device Driver는 Major Number와 Minor Number를 통해 Device를 관리한다. 실제 Device의 Major Number와 Minor Number를 저장하는 변수는 어떤 구조체의 어떤 변수인가 ? (역시 Task 구조체에서부터 쭉 찾아오길 바람)

task_struct 내에 files_struct 내에 file 내에 path 내에 dentry 내에 inode 구조체에 존재하는 i_rdev 변수에 저장한다.

[임베디드 애플리케이션 구현 52] 예로 간단한 Character Device Driver를 작성했다고 가정해본다. 그리고 insmod를 통해 Driver를 Kernel내에 삽입했으며 mknod를 이용하여 /dev/장치파일을 생성하였다. 그리고 이에 적절한 User 프로그램을 동작시켰다. 이 Driver가 실제 Kernel에서 어떻게 관리되고 사용되는지 내부 Mechanism을 기술하시오.

Device Driver 에서 class_create, device_create 를 이용해서 Character Device 인 /dev/장치파일을 만든다.
그리고 Character Device Driver를 등록하기 위해서는 register_chrdev()가 동작해야 한다.
동적으로 할당할 경우에는 alloc_chrdev_region() 이 동작한다. 이때 file_operations를 Wrapping할 구조체를 전달하고 Major Number와 장치명을 전달한다.
chrdevs 배열에 Major Number에 해당하는 Index에 file_operations를 Wrapping한 구조체를 저장해둔다.

그리고 이후에 실제 User 쪽에서 생성된 장치가 open 되면 그때는 Major Number에 해당하는 장치의 File Descriptor가 생성되었으므로 이 fd_array 즉, file 구조체에서 i_mode를 보면 Character Device임을 알 수 있고 i_rdev를 보고 Major Number와 Minor Number를 파악할 수 있다.
그리고 chrdevs에 등록한 Major Number와 같음을 확인하고 이 fd_array에 한해서는 open, read, write, lseek, close등의 file_operations 구조체 내에 있는 함수 포인터들을 앞서 Wrapping한 구조체로 대체한다.
그러면 User에서 read()등을 호출할 경우 우리가 알고 있는 read가 아닌 Device Driver 작성시 새로 만든 우리가 Wrapping한 함수가 동작하게 된다.

[임베디드 애플리케이션 구현 53] Kernel 자체에 kmalloc(), vmalloc(), __get_free_pages()를 통해 메모리를 할당할 수 있다. 또한 kfree(), vfree(), free_pages()를 통해 할당한 메모리를 해제할 수 있다. 이러한 Mechanism이 필요한 이유가 무엇인지 자세히 기술하라.

Device Driver나 기타 여러 Kernel 내부의 Mechanism을 수행하는데 있어서 자료를 저장할 공간이 Kernel 역시 필요할 것이다. 그 공간을 확보하기 위해 Memory Allocation Mechanism이 Kernel에도 존재한다.

[임베디드 애플리케이션 구현 55] OoO(Out-of-Order)인 비순차 실행에 대해 기술하라.

데이터 의존성이 존재하게 되면 어쩔 수 없이 Stall이 발생하게 된다. 이러한 Stall을 최소화하기 위해 앞서 실행했던 코드와 의존성이 없는 코드를 찾아서 아래의 의존성이 있는 코드 위로 끌어올려서 실행하는 방식

이다.

[임베디드 애플리케이션 구현 56] Compiler의 Instruction Scheduling에 대해 기술하라.

위의 OoO 개념으로 Compile-Time에 Compiler Level에서 직접 수행한다. Compiler가 직접 Compile-Time에 분석하므로 최적화의 수준이 높으면 높을수록 Compile Timing이 느려질 것이다.
그러나 성능만큼은 확실하게 보장할 수 있을 것이다.

[임베디드 애플리케이션 구현 57] CISC Architecture와 RISC Architecture에 대한 차이점을 기술하라.

CISC와 RISC의 가장 큰 차이점은 다이 사이즈다. CISC는 다이 사이즈가 크다보니 여러가지 기능 유닛들을 포함할 수 있다. 그러다보니 상대적으로 전력을 많이 소비하게 되는 면모를 보인다. RISC는 다이 사이즈가 작다보니 CISC처럼 여러 기능 유닛들을 집어넣을 수 없다. 그러한 이유로 전력 소모량도 상대적으로 적다. 또한 Dynamic하게 서포팅을 해주는 것이 없으므로 Compiler의 최적화 알고리즘은 RISC 쪽이 더 신경을 많이 써야 한다. (CISC 쪽은 HW가 어느정도 커버 해주므로)

[임베디드 애플리케이션 구현 58] Compiler의 Instruction Scheduling은 Run-Time이 아닌 Compile-Time에 결정된다. 고로 이를 Static Instruction Scheduling이라 할 수 있다. Intel 계열의 Machine에서는 Compiler의 힘을 빌리지 않고도 어느 정도의 Instruction Scheduling을 HW의 힘만으로 수행할 수 있다. 이러한 것을 무엇이라 부르는가 ?

Dynamic Instruction Scheduling

[임베디드 애플리케이션 구현 60] CPU 들은 각각 저마다 이것을 가지고 있다. Compiler 개발자들은 이것을 고려해서 Compiler를 만들어야 한다. 또한 HW 입장에서도 이것을 고려해서 설계를 해야 한다. 여기서 말하는 이것이란 무엇인가 ?

ISA(Instruction Set Architecture) : 명령어 집합

[임베디드 애플리케이션 구현 61] Intel의 Hyper Threading 기술에 대해 상세히 기술하시오.

Hyper Threading 은 Pentium 4 에서 최초로 구현된 SMT(Simultaneous Multi-Threading) 기술명이다. Kernel 내에서 살펴봤던 Context를 HW 차원에서 지원하기 때문에 실제 1개 있는 CPU가 논리적으로 2개가 있는것처럼 보이게 된다. HW상에서 회로로 이를 구현하는 것인데 Kernel에서 Context Switching에선 Register들을 저장했다면 이것을 HW 에서는 이러한 HW Context 들을 복사하여
Context Switching 의 비용을 최소화하는데 목적을 두고 있다. TLP(Thread Level Parallelization) 입장에서보면 Mutex 등의 Lock Mechanism 이 사용되지 않는한 여러 Thread 는 완벽하게 독립적이므로 병렬 실행될 수 있다. 한마디로 Hyper Threading 은 Multi-Core 에서 TLP 를 극대화하기에 좋은 기술이다.

[임베디드 애플리케이션 구현 66] 자신이 사용하는 리눅스 커널의 버전을 확인해보고
https://mirrors.edge.kernel.org/pub/linux/kernel/v4.x/$(자신의 버전).tar.gz 를 다운받아보고
이 압축된 파일을 압축 해제하고 task_struct 를 찾아보도록 한다. 일련의 과정을 기술하면 된다.

cd ~
mkdir kernel
cd kernel
wget https://mirrors.edge.kernel.org/pub/linux/kernel/v4.x/linux-4.4.tar.gz
tar zxvf linux-4.4.tar.gz
cd linux-4.4
ctags -R; mkcscsope.sh
vi -t task_struct

[임베디드 애플리케이션 구현 68] 현재 삽입된 디바이스 드라이버의 리스트를 보는 명령어는 무엇인가 ?
lsmod

[임베디드 애플리케이션 구현 73] Linux에서 fork()를 수행하면 Process를 생성한다. 이때 부모 프로세스를 gdb에서 디버깅하고자하면 어떤 명령어를 입력해야 하는가 ?

set follow-fork-mode child
자식을 디버깅 하고자 하는 경우에는 set follow-fork-mode child지만 부모의 경우는 디폴트 설정 값이므로 특별한 명령어를 줄 필요가 없다.

- 부모의 경우 child를 parent로 바꾸어서 답안을 작성했습니다. 특별한 명령어를 줄 필요가 없다는 것을 몰랐습니다.

[임베디드 애플리케이션 구현 80] 리눅스에서 말하는 File Descriptor(fd)란 무엇인가 ?

리눅스 커널에 존재하는 Task를 나타내는 구조체인 task_struct내에 files_struct내에 file 구조체에 대한 포인터가 fd_array다. 거기서 우리가 System Programming에서 얻는 fd는 바로 이 fd_array에 대한 index다.

- 파일 디스크립터에 파일 정보를 가진 것이라는 이론은 기입하였지만, 리눅스 커널 안에 어디에 위치해서 그 것이 무얼 뜻하는지는 기입하지 않았습니다.

[임베디드 애플리케이션 구현 82] 프로세스들은 최소 메모리를 관리하기 위한 mm, 파일 디스크립터인 fd_array, 그리고 signal을 포함하고 있는데 그 이유에 대해 기술하시오.

자신이 **실제 메모리 어디에 위치하는지**에 대한 정보가 필요하고 또 자신이 **하드 디스크의 어떤 파일이 메모리에 로드되어 프로세스가 되었는지**의 정보가 필요하며 마지막으로 프로세스들 간에 signal을 주고 받을 필요가 있기 때문에 signal이 필요하다.

- 시그널을 공유하는 메커니즘이 필요하다는 말만 기재하고 그 이유에 대해 작성하지 않아 오답노트를 진행하였습니다.

[임베디드 애플리케이션 구현 87] 클라우드 기술의 핵심인 OS 가상화 기술에 대한 질문이다. OS 가상화에서 핵심에 해당하는 3 가지를 기술하시오.

CPU 가상화, 메모리 가상화, I/O 가상화

[임베디드 애플리케이션 구현 95] 리눅스 커널의 arch 디렉토리에 대해서 설명하시오.

CPU(HW) 의존적인 코드가 위치한 영역이다.

[임베디드 애플리케이션 구현 96] 95 번 문제에서 arm 디렉토리 내부에 대해 설명하도록 하시오.

ARM 은 하위 호환이 안되고 다양한 반도체 벤더들이 개발을 하고 있기 때문에 해당 디렉토리에 들어가면 회사별 주요 제품들의 이름이 보이는 것을 확인할 수 있다.

[임베디드 애플리케이션 구현 97] 리눅스 커널 arch 디렉토리에서 c6x 가 무엇인지 기술하시오.
TI DSP 에 해당하는 Architecture 임

[임베디드 애플리케이션 구현 98] Intel 아키텍처에서 실제 HW 인터럽트를 어떤 함수를 가지고 처리하게 되는지 코드와 함께 설명하시오.

일반적인 HW 인터럽트는 어셈블리 루틴 common_interrupt 레이블에서 처리한다.
이 안에서는 do_IRQ() 라는 함수가 같이 함께 일반적인 HW 인터럽트를 처리하기 위해 분발한다.

[임베디드 애플리케이션 구현 99] ARM 에서 System Call 을 사용할 때 사용하는 레지스터를 적으시오.

r7