# *Xilinx Zynq FPGA, TI DSP, MCU 기반의 프로그래밍 전문가 과정*

<리눅스 시스템 프로그래밍, 네트워크, 커널>

2018.04.17 - 2 개월차 시험 오답정리

(서술형)

**강사 - 이상훈**

gcccompil3r@gmail.com

**학생 - 안상재**

sangjae2015@naver.com

7. Page Fault 가 발생했을때 운영체제가 어떻게 동작하는지 기술하시오.

* 정답

Page Fault 는 가상 메모리를 물리 메모리로 변환하는 도중 물리 메모리에 접근했더니 할당된 페이지가 없을 경우 발생한다. 이것이 발생하면 현재 수행 중이던 ip(pc) 레지스터를 저장하고 페이지에 대한 쓰기 권한을 가지고 있다면 Page Fault Handler 를 구동시켜서 페이지를 할당하고 저장해놨던 ip(pc) 를 복원하여 다시 Page Fault 가 발생했던 루틴을 구동시킨다. 만약 쓰기 권한이 없다면 Segmentation Fault 를 발생시킨다.

* 나의 답

page fault가 나면 page fault 핸들러가 동작되고 유저영역으로 부터 동작한 것인지, 커널영역으로부터 동작한 것인지 판별해서 유저영역으로 부터 동작한 것이면 Segmentation fault가 나면서 빠져나간다. 만약 커널영역으로부터 동작한 것이면page fault 핸들러가 해당 메모리에 page를 할당하고 다시 해당 메모리에 접근한다.

→ fault 도 내부 인터럽트의 일종이므로 핸들러 수행 전과 후에 HW 레지스터를 context-switching 을 한다. Fault 가 발생해서 핸들러를 수행하고 나면 fault 를 발생시켰던 루틴을 다시 구동한다.

14. 리눅스 커널의 스케쥴링 정책중 Deadline 방식에 대해 기술하시오.

* 정답

리눅스 커널의 Deadline Scheduler 는 Dario Faggioli 에 의해 2013 년 3 월 Linux Kernel v3.14 에서 소개되었다. Deadline Scheduler 는 EDF + CBS 알고리즘 기반으로 동작한다. 임베디드 개발자들이 주로 원하는 기능이지만 우리의 펭귄 새끼 토발즈가 별로 탐탁치 않아 한다. (CPU Scheduler 만으로 Real-Time 을 해결할 수 없다고 생각해서 그러함) Task 마다 주어진 주기를 가지고 실행되는 것을 보장해야 한다. dl Task 의 우선 순위는 종종 변동되는데, 만료 시각이 먼저 다가 오는 Task 에 우선 처리할 기회가 주어진다. 한 마디로 지금 당장 급한것부터 처리하자는 의미다.

* 나의 답

deadline이 가장 가까운 태스크를 스케줄링 대상으로 선정한다. DEADLINE 정책을 사용하는 태스크의 경우에 우선순위는 의미가 없다.

→ Deaddline Scheduler 는 지금 당장 급한 태스크부터 처리하자는 취지이다.

15. TASK_INTURRUPTIBLE 과 TASK_UNINTERRUPTIBLE 은 왜 필요한지 기술하시오.

* 정답

우리가 사용하는 프로그램에서는 반드시 순차적으로 동작해야 하는 구간이 존재하고 또한 쓸대없이 순차적으로 동작하면 성능에 손해를 보는 구간도 존재한다. 이러한 것 때문에 User Level에서 thread를 사용할 경우 Critical Section이 존재한다면 Lock Mechanism을 사용한다. Lock Mechanism을 사용 중일 경우에는 Interrupt 를 맞으면 안되기 때문에 TASK_UNINTERRUPTIBLE이 필요하다. 반대로 그렇지 않은 경우라면 Interrupt에 무관하므로 TASK_INTERRUPTIBLE을 사용한다.

* 나의 답

- **TASK_INTURRUPTIBLE :** 태스크가 실행되는 도중에 비동기적인 사건이 발생했을때 인터럽트를 발생시켜서 처리하기 위해 **TASK_INTURRUPTIBLE** 상태가 필요하다**.**
- **TASK_UNINTERRUPTIBLE :** 태스크가 실시간적으로 수행해야할 연산(영상/음성, 레이다)이 너무 많아서 당장 인터럽트들을 받으면 안되는 상태의 경우에는 태스크를 **TASK_UNINTERRUPTIBLE**의 상태로 만들어서 인터럽트를 막는다**.**

→ Lock Mechanism 을 사용 중일 경우에는 UNINTERRUPTIBLE 를 사용함.

17. 현재 4개의 CPU(0, 1, 2, 3)가 있고 각각의 RQ에는 1, 2개의 프로세스가 위치한다. 이 경우 2번 CPU에 있는 부모가 fork()를 수행하여 Task를 만들어냈다. 이 Task는 어디에 위치하는 것이 좋을까 ? 그리고 그 이유를 적으시오.

* 정답

| |
|---|
| 2번에 위치하는 것이 좋다. fork()는 같은 Task를 복사하게 된다. 그리고 다시 그 코드를 사용할 것이라면 CPU의 Instruction Cache, Data Cache를 재활용하는 것이 최고다. 결론적으로 Cache Affinity를 적극 활용하자는 것이다. |

* 나의 답

| |
|---|
| **2번 CPU의 자식 프로세스는 RQ에 위치한다. 그 이유는 RQ에 있는 프로세스에 의해 생성이 된 프로세스이기 때문이다.** |

→ 문제 의미를 잘 못 파악함.

→ cpu 캐시의 지역성을 적극 활용해서 성능을 높이기 위해 fork()를 수행한 cpu 에 태스크를 위치시킨다.

18. 15번 문제에서 이번에는 0, 1, 3에 매우 많은 프로세스가 존재한다. 이 경우 3번에서 fork()를 수행하여 Task를 만들었다. 이 Task는 어디에 위치하는 것이 좋을까 ? 역시 이유를 적으시오.

* 정답

| |
|---|
| 2번에 위치하는 것이 좋다. 이번에는 주어진 시간내에 0, 1, 3 은 Scheduling 을 수행할 수 없기 때문에 오히려 상대적으로 Task가 적은 2번에 배치하는 것이 좋다. Cache Affinity를 포기할지라도 아예 Task 를 동작시킬 기회가 없는것보다 좋기 때문이다. |

→ 자신을 fork()한 cpu 에 태스크가 많이 있다면 캐시의 지역성을 포기하고 수행될 기회가 있는 다른 cpu 에 위치해야 한다.

25. Kernel 은 Memory Management 를 수행하기 위해 VM(가상 메모리)를 관리한다. 가상 메모리의 구조인 Stack, Heap, Data, Text 는 어디에 기록되는가? (Task 구조체의 어떠한 구조체가 이를 표현하는지 기술하시오)

* 정답

| |
|---|
| task_struct 구조체에 있는 mm_struct 내에 start_code, end_code등으로 기록됨 |

26. 23번에서 Stack, Heap, Data, Text등 서로 같은 속성을 가진 Page를 모아서 관리하기 위한 구조체 무엇인가 ?(역시 Task 구조체의 어떠한 구조체에서 어떠한 식으로 연결되는지 기술하시오)

* 정답

| |
|---|
| task_struct 내에 mm_struct 포인터를 따라가 보면 vm_area_struct 구조체가 있다. 이 녀석이 서로 같은 Segment 들을 모아서 관리한다. |

27. 프로그램을 실행한다고 하면 fork(), execve()의 콤보로 이어지게 된다. 이때 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 ?

* 정답

| |
|---|
| ELF Header 와 Program Headers 를 읽고 값을 적절하게 채운다. |

29. VM(가상 메모리)와 PM(물리 메모리)를 관리하는데 있어 VM 을 PM 으로 변환시키는 Paging Mechanism 에 대해 Kernel 에 기반하여 서술하시오.

* 정답

| |
|---|
| mm_struct에 pgd라는 필드가 있다.<br>Page Directory를 의미하는 것으로 pgd -> pte -> page로 3단계 Paging을 수행한다.<br>각각 10bit, 10bit, 12bit로 VM의 주소를 쪼개서 Indexing을 한다. |

→ 3 단계 페이징이란 pgd → pte → page 3 개의 구조체를 거치며 indexing 하는 것이다.

30. MMU(Memory Management Unit)의 주요 핵심 기능을 모두 적고 간략히 설명하시오.

* 정답

| |
|---|
| 1. HAT(HW Address Translation) 는 가상 메모리 공간을 실제 물리 메모리 공간으로 변환한다.<br>2. TLB(Translation Lookaside Buffer) 는 가상 메모리 주소와 대응되는 물리 메모리 주소를 Caching한다. |

→ MMU 의 핵심 기능에는 HAT, TLB 2 가지가 있다.

31. 하드디스크의 최소 단위를 무엇이라 부르고 그 크기는 얼마인가 ?

* 정답

| |
|---|
| Sector, 512 byte |

32. Character 디바이스 드라이버를 작성할 때 반드시 Wrapping 해야 하는 부분이 어디인가 ? (Task 구조체에서 부터 연결된 부분까지를 쭉 이어서 작성하라)

* 정답

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

* 나의 답

| |
|---|
| 사용자 태스크가 호출할 함수들과 특정 하드웨어를 위한 디바이스 드라이버 코어의 함수를 연결해주는 부분 |

→ write, open, read 등의 파일 관련 연산이 file_operations 구조체에 있음. 그 부분을 새롭게 만든 함수로 Wrapping 해주어야 함.

38. 내부 인터럽트에서 분류된 녀석들의 특징에 대해 기술하시오.

* 정답

| |
|---|
| Fault의 경우 Page Fault가 대표적이므로 발생 시 현재 진행 중인 주소를 복귀 주소로 저장하고 Fault에 대한 처리를 진행하고 다시 돌아와서 Fault가 났던 부분을 다시 한 번 더 수행한다. Trap의 경우 System Call이 대표적이므로 발생 시 현재 진행 중인 바로 아래 주소를 복귀 주소로 저장하고 System Call에 대한 수행을 처리한 이후 System Call 바로 아래 주소부터 실행을 시작한다 (함수 호출의 복귀 주소와 비슷한 형태) Abort의 경우 심각한 오류에 해당하므로 그냥 종료한다. |

→ Fault 는 핸들러 함수를 수행하고 Fault 가 났던 부분을 다시 한 번 더 수행해서 제대로 수정하지만, Trap 의 경우는 오류이므로 핸들러 함수를 수행하고 나서 Trap 이 났던 부분을 다시 수행하지 않는다.

41. 38 에서 User Space 에서 System Call 번호를 전달한다. Intel Machine 에서는 이를 어디에 저장하는가?또한 ARM Machine 에서는 이를 어디에 저장하는가?

* 정답

| Intel 의 경우에는 ax 레지스터에, ARM 의 경우에는 r7 레지스터에 해당한다. |
|---|

42. Paging Mechanism 에서 핵심이 되는 Page Directory 는 mm_struct 의 어떤 변수가 가지고 있는가 ?

* 정답

| pgd |
|---|

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

* 정답

| Intel 의 경우엔 CR3, ARM 의 경우엔 CP15 |
|---|

47. Mutex 와 Semaphore 의 차이점을 기술하시오.

* 정답

| Mutex 와 Semaphore 모두 Context Switching 을 유발하게 된다. 차이점이라면 Mutex 는 공유된 자원의 데이터를 여러 쓰레드가 접근하는 것을 막는다. Semaphore 는 공유된 자원의 데이터를 여러 프로세스가 접근하는 것을 막는 것이다. |
|---|

* 나의 답

| - 뮤텍스는 스레드들간에 공유가 배제되는 객체이다. 공유 자원이 수행 중에 오직 한 태스크 에게만 소유되어야 할 필요가 있을 때 그 자원에 대한 뮤텍스 객체를 생성시킨다.<br>- 세마포어는 여러 개의 공유자원과 여러 개의 카운트 값을 가지는 뮤텍스라고 볼 수 있다. 공유자원에 대한 접속을 제어하기 위한 신호이다. 공유 자원에 접근할 수 있는 최대 허용치 만큼 동시에 사용자 접근을 할 수 있게 한다. |
|---|

→ Mutex 와 Semaphore 는 스레드와 프로세스의 차이

55. OoO(Out-of-Order)인 비순차 실행에 대해 기술하라.

* 정답

| 데이터 의존성이 존재하게되면 어쩔 수 없이 Stall이 발생하게 된다. 이러한 Stall을 최소화하기 위해 앞서 실행했던 코드와 의존성이 없는 코드를 찾아서 아래의 의존성이 있는 코드 위로 끌어올려서 실행하는 방식이다. |
|---|

* 나의 답

| **CPU**는 원래 명령어들이 들어오는 순서대로 처리해주었는데(순차 실행) **CPU**의 실행 속도를 빠르게 하기 위해 명령어의 실행 순서를 임의로 바꾸는 것을 말한다. |
|---|

→ 데이터의 의존성이 없는 코드를 먼저 실행시킴으로 써 성능을 극대화함.

56. Compiler 의 Instruction Scheduling 에 대해 기술하라.

* 정답

비순차 실행 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.

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 어느정도의 Instruction Scheduling을 HW의 힘만으로 수행할 수 있다. 이러한 것을 무엇이라 부르는가 ?

* 정답

Dynamic Instruction Scheduling

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 ?

* 정답

ISA(Instruction Set Architecture) : 명령어 집합

* 나의 답

**A :** 기계어**(**어셈블리어**)**

→ 해당 CPU 만의 어셈블리어도 ISA 에 속한다고 할 수 있을 것 같다.

80. 리눅스에서 말하는 File Descriptor(fd)란 무엇인가 ?

* 정답

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.

* 나의 답

시스템으로 부터 할당받은 파일을 대표하는 정수값**,** 리눅스는 모든 것을 파일로 보기 때문에 프로세스에서 파일들에 접근할때 파일 디스크립터라는 개념을 씀**.**

→ fd 는 file 구조체에 대한 포인터의 index

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

* 정답

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

* 나의 답

**A : mm_struct**는 프로세스의 가상메모리 영역의 크기와 영역의 종류등을 관리하기 위해 사용되는 구조체이고**, fd_array**는 **fd(**파일 디스크립터**)**를 인덱스로 사용해서 파일 객체를 가리키는 배열이다**. signal**은 시그널 핸들러를 관

리하기 위한 구조체 이다. 각각의 구조체 및 배열은 프로세스의 자원들을 효율적으로 관리하기 위해 커널영역에 정의되어 있는 변수이다.

→ mm_struct : 메모리상의 프로세스의 위치에 대한 정보
fd_array : 하드 디스크의 어떤 파일이 프로세스가 되었는지에 대한 정보
signal : 프로세스간의 주고받는 signal 에 대한 정보

94. 유저에서 fork() 를 수행할때 벌어지는 일들 전부를 실제 소스 코드 차원에서 해석하도록 하시오.

* 정답

우리가 만든 프로그램에서 fork() 가 호출되면 C Library 에 해당하는 glibc 의 __libc_fork() 가 호출됨. 이 안에서 ax 레지스터에 시스템 콜 번호가 기록된다. 즉, sys_fork() 에 해당하는 시스템 콜 테이블의 번호가 들어가고 이후에 int 0x80 을 통해서 128 번 시스템 콜을 후출하게 된다. 그러면 제어권이 커널로 넘어가서 idt_table(Interrupt Descriptor Table)로 가고 여기서 시스템 콜은 128 번으로 sys_call_table 로 가서 ax 레지스터에 들어간 sys_call_table[번호] 의 위치에 있는 함수 포인터를 동작시키면 sys_fork() 가 구동이 된다. sys_fork() 는 SYSCALL_DEFINE0(fork) 와 같이 구성되어 있다.

* 나의 답

유저에서 **fork()**를 수행하면 프로세스가 하나 더 만들어지고 가상메모리 레이아웃을 부모 프로세스로부터 그대로 복사해온다. 자식 프로세스는 **fork()**를 한 명령어부터 수행이 되고 부모 프로세스와 멀티 태스킹으로 수행하게 된다.

98. Intel 아키텍처에서 실제 HW 인터럽트를 어떤 함수를 가지고 처리하게 되는지 코드와 함께 설명하시오.

* 정답

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

* 나의 답

**irq_desc** 구조체에 정의되어 있는 함수 포인터 다발을 이용해 인터럽트를 처리한다.

99. ARM 에서 System Call 을 사용할 때 사용하는 레지스터를 적으시오.

* 정답

r7