2nd test.

노트북:	SW		
만든 날짜:	2018-04-21 오후 10:57	**수정한 날짜:**	2018-04-29 오후 8:19
작성자:	fstopdg@gmail.com		
URL:	http://cafe.naver.com/ArticleRead.nhn?clubid=26400661&articleid=3197&referrerAllArticles=true		

Theory

3.
Q,리눅스 커널은 운영체제(OS)다. OS가 관리해야 하는 제일 중요한 5가지에 대해 기술하시오.
A. Task, Memory, File system, Network, Devic driver

4.
Q. Unix 계열의 모든 OS는 모든 것을 무엇으로 관리하는가 ?
A. File

5.
Q. 리눅스에는 여러 장점이 있다.
아래의 장점들 각각에 대해 기술하라.

* 사용자 임의대로 재구성이 가능하다.
* 열악한 환경에서도 HW 자원을 적절히 활용하여 동작한다.
* 커널의 크기가 작다.
* 완벽한 멀티유저, 멀티태스킹 시스템
* 뛰어난 안정성
* 빠른 업그레이드
* 강력한 네트워크 지원
* 풍부한 소프트웨어

A.

- 리눅스 커널이 F/OSS(Free and open-source software)에 속하여 있으므로 소스코드를 원하는 대로 수정할 수 있다. 그러나 License 부분을 조심해야 한다.(License 관련 부분은 팀장급 이상과 상의후 결정)
- Monolithic kernel이라서 열악한 환경에서도 잘 동작한다. & 리눅스 커널이 가볍고 좋을뿐만 아니라 소스가 공개되어 있어 다양한 분야의 사람들이 지속적으로 개발하여 어떠한 열악한 환경에서도 잘 동작한다.
- 최적화가 잘 되었있다.
- 리눅스는 RT Scheduling 방식을 채택하여 Multi-Tasking을 효율적으로 잘 해낸다.
- 전 세계의 많은 개발자들이 지속적으로 유지보수하여 안정성이 뛰어남
- 위와 동일
- TCP/IP Stack이 원채 잘 되어 있다보니 Router 및 Switch 등의 장비에서 사용함
- GNU(Gnu Is Non Unix)정신에 입각하여 많은 Tool들이 개발되어 있다.

6.
Q. 32bit System에서 User와 Kernel의 Space 구간을 적으시오.
A. All : 4GB -> User : 0~3GB & Kernel : 3~4GB

7.
Q. Page Fault 가 발생했을때 운영체제가 어떻게 동작하는지 기술하시오.
A, Page Fault : 가상메모리를 물리메모리로 변환하는 도중 물리메모리에 접근시 할당된 페이지가 없을 때 발생하는 fault.
Page Fault가 발생시 현재 수행중이던 ip(pc) 레지스터를 저장하고 페이지에 대한 쓰기 권한을 가지고 있다면 Page fault handler를 구동시켜서 페이지를 할당하고 저장해놨던 ip(pc)를 복원하여 다시 Page Fault가 발생했던 루틴을 구동시킨다.
참고로 페이지에 대한 쓰기 권한이 없을 경우, Segmentation Fault를 발생시킨다.

8.ㅁ
Q. 리눅스 실행 파일 포맷이 무엇인지 적으시오.
A. ELF(Executible Linkable Format)

9.

Q. 프로세스와 스레드를 구별하는 방법에 대해 기술하시오.
A. Process(Thread leader) : pid = tgid
Thread : pid != tgid(Thread leader 제외), tgid는 전부 같다.

10.
Q. Kernel 입장에서 Process 혹은 Thread를 만들면 무엇을 생성하는가 ?
A. task_struct(task_struct 구조체가 메모리에 Load되어 객체화된다.)

11.ㅁ
Q. 리눅스 커널 소스에 보면 current라는 것이 보인다. 이것이 무엇을 의미하는 것인지 적으시오. 커널 소스 코드와 함께 기술하시오.
A. current: 현재 구동되는 task를 가르키는 매크로
vi -t current 를 통하여 확인
-> Intel(16으로 진입)

```
arch/x86/include/asm/current.h

#ifndef _ASM_X86_CURRENT_H
#define _ASM_X86_CURRENT_H

#include <linux/compiler.h>
#include <asm/percpu.h>

#ifndef __ASSEMBLY__
struct task_struct;

DECLARE_PER_CPU(struct task_struct *, current_task);

static __always_inline struct task_struct *get_current(void)
{
    return this_cpu_read_stable(current_task);
}

#define current get_current()

#endif /* __ASSEMBLY__ */

#endif /* _ASM_X86_CURRENT_H */
```

#define current get_current() 를 확인
get_current() 함수를 보게 되면, this_cpu_stable(current_task)를 반환한다.
this_cpu_stable(current_task)로 진입하면
#define this_cpu_read_stable(var) percpu_stable_op("mov", var) 를 볼 수 있다.
즉, percpu_stable_op("mov", var) 매크로를 통해 관리됨을 볼 수 있다.
Intel 방식의 특유의 세그먼트 레지스터를 사용하여 관리하는 것을 볼 수 있는 부분이다.

-> ARM(18로 진입)

```
include/asm-generic/current.h

#ifndef __ASM_GENERIC_CURRENT_H
#define __ASM_GENERIC_CURRENT_H

#include <linux/thread_info.h>

#define get_current() (current_thread_info()->task)
#define current get_current()

 #endif /* __ASM_GENERIC_CURRENT_H */
```

thread_info() -> task를 확인할 수 있다.
즉, 현재 구동되는 task임을 알 수 있다.

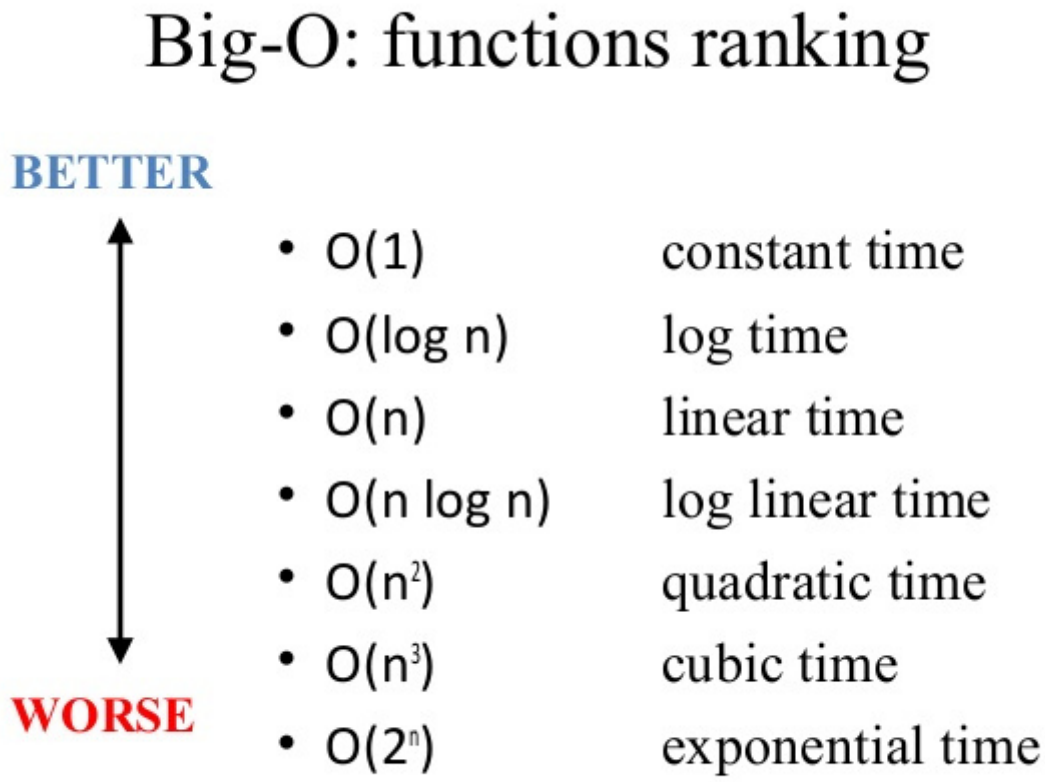
12.
Q. Memory Management 입장에서 Process와 Thread의 핵심적인 차이점은 무엇인가 ?
A. Process : Process 간에는 서로 메모리영역을 공유하지 않는다.
Thread : 자신의 그룹에 속한 모든 Task들과 메모리를 공유한다.

13.
Q.Task가 관리해야하는 3가지 Context가 있다. System Context, Memory Context, HW Context가 있다. 이중 HW Context는 무엇을 하기 위한 구조인가 ?
A. Task가 동작하다 Context Switching 등이 발생했을 경우, 다른 Task가 Register를 변경할 수 있으므로 자신이 현재 어디까지 진행했는지를 기록할 필요가 있다. 이러한 레지스터등의 정보를 저장하기 위한 용도이다.

14.
Q. 리눅스 커널의 스케쥴링 정책중 Deadline 방식에 대해 기술하시오.
A. 지금 당장 급한 Task 먼저 처리하는 Scheduling 방식이다.
참고.
리눅스 커널의 Deadline Scheduler는 Dario Faggioli에 의하여 2013년 3월 Linux Kernel v3.14 에서 소개되었다.
Deadline Scheduler는 EDF + CBS 알고리즘 기반으로 동작한다.
임베디드 개발자들이 주로 원하는 기능이지만, Linus benedict torvalds는 별로 탐탁치 않아 한다.(CPU Scheduler 만으로 Real-Time을 해결할 수 없다고 생각해서 그러하다.)
Task마다 주어진 주기를 가지고 실행되는 것을 보장해야 한다.
dl Task의 우선 순위는 종종 변동되는데, 만료시각이 먼저 다가오는 Task에 우선 처리할 기회가 주어진다.

15.
Q. TASK_INTURRUPTIBLE과 TASK_UNINTERRUPTIBLE은 왜 필요한지 기술하시오.
A. 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 사용중일 경우에는 Interrupt를 맞으면 안되기 때문에 TASK_UNINTERRUTIBLE이 필요하다.
반대로 그렇지 않은 경우라면 Interrupt에 무관하므로 TASK_INTERRUPTIBLE을 사용한다.

16.
Q. O(N)과 O(1) Algorithm에 대해 기술하시오. 그리고 무엇이 어떤 경우에 더 좋은지 기술하시오.
A. O(N)은 데이터의 양이 많아지면 많아질수록 알고리즘의 성능이 떨어지는데 바하여
O(1)은 데이터의 양이 많아지더라도 알고리즘의 성능이 언제나 동일하다.



참고. 데이터의 양이 증가한 경우의 성능

17.

Q. 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.

A. 3번째 CPU(2)

이유 : 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.

Cache Affinity : Multiproccesor환경에서 각 Proccess에 대하여 Proccess가 사용할(사용하지 않을) CPU를 명시하는 기능

18.

Q. 17번 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.

A. 3번째 CPU(2)

이유 : 이번에는 주어진 시간내에 0, 1, 3은 Scheduling을 수행할 수 없기 때문에 오히려 상대적으로 Task가 적은 2번에 배치하는 것이 좋다.

Cache Affinity를 포기할지라도 아예 Task를 동작시킬 기회가 없는 것보다 좋기 때문이다.

19.

Q. UMA와 NUMA에 대해 기술하고 Kernel에서 이들을 어떠한 방식으로 관리하는지 기술하시오.커널 내부의 소스 코드와 함께 기술하도록 하시오.

A. 메모리 접근 속도가 같은 것을 Bank라 한다. 그리고 Kernel에서 이름은 Node라 한다.

UMA는 모든 CPU가 메모리 접근 속도가 같은 SMP와 같은 것을 의미한다.

NUMA는 CPU마다 메모리 접근 속도가 다른 Intel의 i계열의 CPU군을 의미한다.

Linux Kernel에선 이를 contig_page_data를 통해 전역변수로 관리한다.

그리고 UMA의 경우엔 Node가 1개인데 pglist_data 구조체로 표현된다.

NUMA의 경우엔 pglist_data가 여러개 연결되어 있다.

20.

Q. Kernel의 Scheduling Mechanism에서 Static Priority와 Dynamic Priority 번호가 어떻게 되는지 적으시오.
A. Static Priority : 0 ~ 99 & Dynamic Priority : 100 ~ 139

21.
Q. ZONE_HIGHMEM 에 대해 아는대로 기술하시오.
A. 커널 공간 1GB의 가상메모리가 모든 물리메모리를 커버해야 한다.
32bit 시스템의 경우에는 1 : 3 이라 커널 공간이 1GB밖에 없다.
물리메모리는 8GB, 64GB 가 달려있으니 이것을 커버하기 위한 기법 ZONE_HIGHMEM이 필요하다.
64bit 시스템의 경우에는 1 : 1 이라 커널 공간이 8EB이다.
이 용량은 물리메모리를 충분히 커버가능하기 때문에 ZONE_HIGHMEM이 필요없다.

기본적으로 ZONE_NORMAL 이 특정 구간까지는 1 : 1로 mapping한다.
이후 ZONE_NORMAL이 처리하지 못하는 모든 메모리 영역을 ZONE_HIGHMEM이 처리하게 된다.

User에서는 어떠한 공간이든 각 Task별로 mapping을 하여 사용하지만, Kernel에서는 최대한 빠른 속도를 얻기 위하여 ZONE_DMA(DMA32) 및 물리메모리와 가상메모리를 미리 1 : 1로 mapping하기 위하여 ZONE_NORMAL을 사용한다.
그러나 물리메모리가 커널로의 1 : 1 mapping을 허용하는 영역 크기를 초과하는 경우 이 영역을 CONFIG_ZONE_HIGHMEM영역으로 구성하여 커널에서 사용할 때에는 필요할 때마다 mapping하여 사용한다.

32bit 시스템에서는 1 : 1 mapping이 일부만 가능하기에 ZONE_NORMAL을 초과하는 메모리가 이 영영을 사용한다.
64bit 시스템에서는 모든 물리메모리가 1 : 1 mapping이 가능하므로 ZONE_HIGHMEM을 사용하지 않는다.

22.
Q. 물리 메모리의 최소 단위를 무엇이라고 하며 크기가 얼마인가 ? 그리고 이러한 개념을 SW 적으로 구현한 구조체의 이름은 무엇인가 ?
A. 물리메모리의 최소단위 : Page Frame & Page Frame을 SW적 개념으로 구현한 구조체의 이름은 Page이다.

23.
Q. Linux Kernel은 외부 단편화와 메모리 부하를 최소화하기 위해 Buddy 할당자를 사용한다. Buddy 할당자의 Algorithm을 자세히 기술하시오.
A. Buddy Allocator가 메모리를 관리하는 방식에서 할당할 경우, Bitmap을 같이 고려하는 것은 확실히 Overhead다.
그러나 메모리를 해제하고 어떤 녀석이 어디에 얼마만큼 비어있는지 파악하는 것은 바로 이 Buddy Mechanism을 이용하면 굉장히 유용하다.
Order(0)일때는 총 8K를 비교하는데, 4K, 4K 공간에서 2개의 상태값이 같은지를 본다. 찾고자하는 것은 지금 이공간의 Maximum이 얼마만큼 되는지를 찾아서 4K를 찾는다고 하면 4K만 정확히 떼주고 16K라면 16K를 떼주기 위한 의도다.
Order(1)일 경우에는 16K를 비교하게 된다. 8K와 8K의 상태값이 서로 같으면 역시 Bitmap을 0으로 표기하고 둘의 상태값이 서로 다르면 1이 표기되면서 8K공간을 활용할 수 있다는 의미가 된다.
둘다 사용하지 않을 경우에도 0인데 제공할 수 있는 Maximum에 해당하는 공간을 가지고 있기 위함이다.
Lazy buddy의 경우 Bitmap을 없애고 nr_free를 사용한다.

24.
Q. 21번에 이어 내부 단편화를 최소화 하기 위해 Buddy에서 Page를 받아 Slab 할당자를 사용한다. Slab 할당자는 어떤식으로 관리되는지 기술하시오.
A. Slab Allocator는 Buddy로부터 미리 Page를 할당받아서 32byte부터 128K까지 적정한 수준의 크기로 분할을 하여 가지고 있다.
그리고 우리가 짜잘한 크기의 메모리할당을 요청하면 미리 받아둔 이 조각들을 우리에게 넘겨준다.
내부적으로도 사용할 수 있는 공간이 존재하므로 Free, Full, Partial이라는 상태 정보를 기록하여 새로 Free한 부분을 사용해야할지 Partial한 부분을 사용할지 결정할 수도 있다.

25.
Q. Kernel은 Memory Management를 수행하기 위해 VM(가상 메모리)를 관리한다. 가상 메모리의 구조인 Stack, Heap, Data, Text는 어디에 기록되는가 ?(Task 구조체의 어떠한 구조체가 이를 표현하는지 기술하시오)
A. task0_struct 구조체에 있는 mm_struct 내에 start_code, end_code등으로 기록됨

26.
Q. 25번에서 Stack, Heap, Data, Text등 서로 같은 속성을 가진 Page를 모아서 관리하기 위한 구조체 무엇인가 ? (역시 Task 구조체의 어떠한 구조체에서 어떠한 식으로 연결되는지 기술하시오)
A. task_struct 내에 mm_struct 포인터를 따라가 보면 vm_area_struct 구조체가 있다. 이 vm_area_struct가 서로 같은 Segmen들을 모아서 관리한다.

27.
Q. 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 ?
A. ELF Header 와 Program Headers 를 읽고 값을 적절하게 채운다.

28.
Q. 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.
A. C언어를 사용하기 위해서는 반드시 Stack이 필요하다.
Kernel 영역에서도 동작하는 코드가 올라가기 위한 Text영역, 전역변수가 있는 Data영역, 동적할당하는 Heap, 지역변수를 사용하는 Stack이 존재한다.
이는 역시 User 영역에서도 동일하므로 양쪽에 모두 메모리 공간이 구성된다.

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

30.
Q. MMU(Memory Management Unit)의 주요 핵심 기능을 모두 적고 간략히 설명하시오.
A. HAT(HW Address Translation) & TLB(Translation Lockaside Buffer)
HAT : 가상메모리 공간을 실제 물리메모리 공간으로 변환한다.
TLB : 가상메모리 주소와 대응되는 물리메모리 주소를 Caching한다.

31.
Q. 하드디스크의 최소 단위를 무엇이라 부르고 그 크기는 얼마인가 ?
A. Sector, 512byte

32.
Q. Character 디바이스 드라이버를 작성할 때 반드시 Wrapping 해야 하는 부분이 어디인가 ? (Task 구조체에서 부터 연결된 부분까지를 쭉 이어서 작성하라)
A. task_struct -> files_struct -> file -> file_operation

33.
Q. 예로 유저 영역에서 open 시스템 콜을 호출 했다고 가정할 때 커널에서는 어떤 함수가 동작하게 되는가 ? 실제 소스 코드 차원에서 이를 찾아서 기술하도록 한다.
A.
do_sys_open()함수가 동작

```
fs/open.c

long do_sys_open(int dfd, const char __user *filename, int flags, umode_t mode)
{
    struct open_flags op;
    int fd = build_open_flags(flags, mode, &op);
    struct filename *tmp;

    if (fd)
        return fd;

    tmp = getname(filename);
    if (IS_ERR(tmp))
        return PTR_ERR(tmp);

    fd = get_unused_fd_flags(flags);
    if (fd >= 0) {
        struct file *f = do_filp_open(dfd, tmp, &op);
        if (IS_ERR(f)) {
```

```
            put_unused_fd(fd);
            fd = PTR_ERR(f);
        } else {
            fsnotify_open(f);
            fd_install(fd, f);
        }
    }
    putname(tmp);
    return fd;
}

SYSCALL_DEFINE3(open, const char __user *, filename, int, flags, umode_t, mode)
{
    if (force_o_largefile())
        flags |= O_LARGEFILE;

    return do_sys_open(AT_FDCWD, filename, flags, mode);
}
```

34.
Q. task_struct 에서 super_block 이 하는 역할은 무엇인가 ?
A. super_block은 루트 파일 시스템(' / ')의 위치 정보를 가지고 있다.
또한 super_block은 파일시스템의 메타 데이터를 가지고 있다.

35.
Q. VFS(Virtual File System)이 동작하는 Mechanism 에 대해 서술하시오.
A. VFS는 Super Block인 ext2_sb_info 와 같은 것들을 읽어 super_block에 채워넣는다.
그리고 읽고자 하는 파일에 대한 meta data를 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 파일은 Task와 연결시킨다.

36,
Q. Linux Kernel 에서 Interrupt를 크게 2가지로 분류한다. 그 2 가지에 대해 각각 기술하고 간략히 설명하시오.
A. 내부 인터럽트와 외부 인터럽트로 나뉜다.
내부 인터럽트는 CPU내에서 일어나는 인터럽트에 해당하며
외부 인터럽트는 CPU외부의 실제 Device가 발생시키는 인터럽트이다.

37.
Q. 내부 인터럽트는 다시 크게 3분류로 나눌 수 있다. 3가지를 분류하시오.
A. Fault, Trap, Abort

38.
Q. 37번에서 분류한 녀석들의 특징에 대해 기술하시오.
A. Fault의 경우 Page Fault가 대표적이므로 발생시 현재 진행중인 주소를 복귀주소로 저장하고 Fault에 대한 처리를 진행하고 다시 돌아와서 Fault가 났던 부분을 다시 한 번 더 수행한다.
Trap의 경우 System Call이 대표적이므로 발생시 현재 진행중인 바로 아래 주소를 복귀주소로 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.

39.
Q, 예로 모니터 3 개를 쓰는 경우 양쪽에 모두 인터럽트를 공유해야 한다. Linux Kernel에서는 어떠한 방법을 통해 이들을 공유하는가 ?
A, 외부인터럽트의 경우 32 ~ 255까지의 번호를 사용한다.
여기서 128(0x80)에 해당하는 System Call만은 제외한다.
idt_table에서 128을 제외한 32~255 사이의 번호가 들어오면 실제 HW Device이다.

여기서 같은 종류의 HW Device가 들어올 수 있는데 그들에 대해서 Interrupt를 공유하게 하기 위하여 irq_desc라는 Table을 추가로 두고 active라는 것으로 Interrupt를 공유하게끔한다.

40.
Q. System Call 호출시 Kernel에서 실제 System Call을 처리하기 위해 Indexing을 수행하여 적절한 함수가 호출되도록 주소값을 저장해놓고 있다. 이 구조체의 이름을 적으시오
A. File

41.
Q. 38에서 User Space에서 System Call 번호를 전달한다. Intel Machine에서는 이를 어디에 저장하는가 ? 또한 ARM Machine에서는 이를 어디에 저장하는가 ?
A. Intel : sys_call_table
ARM : __vectors_start + 0x100

42.
Q. Paging Mechanism에서 핵심이 되는 Page Directory 는 mm_struct의 어떤 변수가 가지고 있는가 ?
A. pgd

43.
Q. 또한 Page Directory를 가르키는 Intel 전용 Register가 존재한다. 이 Register의 이름을 적고 ARM 에서 이 역할을 하는 레지스터의 이름을 적으시오.
A. Intel : CR3
ARM : CP15

44.
Q. 커널 내부에서 메모리 할당이 필요한 이유는 무엇인가 ?
A. 커널스택으로 주어진 메모리 공간은 고작 8K에 해당한다.(물론 이 값은 ARM이라고 가정하였을 때고 Intel은 16K에 해당한다.)
문제는 스택공간이라는 특정 작업을 수행하고 이후에 태스크가 종료되면 정보가 사라질 수 도 있다. 이 정보가 없어지지 않고 유지될 필요가 있을 수도 있다.
뿐만 아니라 커널 자체도 프로그램이기 때문에 메모리 공간이 필요하다. 운영체제 또한 C로 만든 프로그램에 불과하다는 것이다.
그러니 프로그램이 동작하기 위하여 메모리를 요구하듯 커널도 필요하다.

45.
Q. 메모리를 불연속적으로 할당하는 기법은 무엇인가 ?
A. vmalloc()

46.
Q. 메모리를 연속적으로 할당하는 기법은 무엇인가 ?
A. kmalloc()

47.
Q. Mutex 와 Semaphore 의 차이점을 기술하시오.
A. Mutex와 Semaphore 모두 Content Switching을 유발하게 한다.
차이점이라면 Mutex는 공유된 자원의 데이터를 여러 쓰레드가 접근하는 것을 막는다.
Semaphore는 공유된 자원의 데이터를 여러 프로세스가 접근하는 것을 막는 것이다.

48.
Q. module_init() 함수 호출은 언제 이루어지는가 ?
A. module_init() 함수는 insmod 명령어로 Device Driver Module을 부착시킬때 동작한다.

49.
Q. module_exit() 함수 호출은 언제 이루어지는가 ?
A. module_exit() 함수는 rmmod 명령어로 Driver Module을 탈착시킬때 동작한다.

50.
Q. thread_union 에 대해 기술하시오.
A. include/linux/sched.h/ 에 있는 공용체로 내부에는 커널 스택 정보와 thread_info를 가지고 있다.

이 안에는 현재 구동중인 task의 정보가 들어있고 Context Switching등에 활용하기 위하여 cpu_context_save 구조체가 존재한다.

```
include/linux/sched.h/

union thread_union {
    struct thread_info thread_info;
    unsigned long stack[THREAD_SIZE/sizeof(long)];
};
```

51.
Q. 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)
A.
-> task_struct
  -> files_struct
    -> file
      -> path
        -> indoe
          -> i_rdev 가 관리

52.
Q. 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.
A.

53.
Q. 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
A. Device Driver나 기타 여러 Kernel 내부의 Mechanism을 수행하는데 있어서 자료를 저장할 공간이 Kernel 역시 필요할 것이다.
그 공간을 확보하기 위해 Memory Alloctaion Mechanism이 Kernel에도 존재한다.

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

56.
Q. Compiler의 Instruction Scheduling에 대해 기술하라.
A.

57.
Q. CISC Architecture와 RISC Architecture에 대한 차이점을 기술하라.
A.

58.
Q. Compiler의 Instruction Scheduling은 Run-Time이 아닌 Compile-Time에 결정된다. 고로 이를 Static Instruction Scheduling이라 할 수 있다. Intel 계열의 Machine에서는 Compiler의 힘을 빌리지 않고도 어느저도의 Instruction Scheduling을 HW의 힘만으로 수행할 수 있다. 이러한 것을 무엇이라 부르는가 ?
A.

59.
Q. Pipeline이 깨지는 경우에 대해 자세히 기술하시오.
A. 분기(jmp or branch)가 발생했을 경우이다.
분기가 발생하게 되면 바로 아래에서 Fetch 및 Decode해오던 값들이 무의미해지기 때문이다.

60.
Q. 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 ?

A.

61.
Q. Intel의 Hyper Threading 기술에 대해 상세히 기술하시오.
A. Hyper Threading은 Pentium 4에서 최초로 구현된 SMT(Simultaneous Multi-Threading) 기술명이다.
Kernel 내에서 살펴봤던 Context를 HW 차원에서 지원하기 때문에 실제 1개 있는 CPU가 논리적으로 2개가 있는 것처럼 보이게 된다.
HW상에서 회로로 이를 구현하는 것인데 Kernel에서 Context Switching에선 Register들을 저장했다면 이것을 HW에서는 이러한 HW Context들을 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 TLP를 극대화하기에 좋은 기술이다.

62.
Q. 그동안 많은 것을 배웠을 것이다. 최종적으로 Kernel Map을 그려보도록 한다. (Networking 부분은 생략해도 좋다) 예로는 다음을 생각해보도록 한다. 여러분이 좋아하는 게임을 더블 클릭하여 실행한다고 할 때 그 과정 자체를 Linux Kernel 에 입각하여 기술하도록 하시오. (그림과 설명을 같이 넣어서 해석하도록 한다) 소스 코드도 함께 추가하여 설명해야 한다.
A.

66.
Q. 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.
A. 첨부(A guide to install the kernel)

67.
Q. Multi-Tasking의 원리에 대해 서술하시오. (Run Queue, Wait Queue, CPU에 기초하여 서술하시오)
A. CPU마다 RQ, WQ가 1개씩 존재한다.
active 배열과 expired 배열이 존재해서 어떤 우선순위의 Process가 현재 올라가 있는지 bitmap을 체크하게 되며 queue에는 만들어진 task_struct가 들어가 있게 된다. 즉, bitmap을 보고 우선순위에 해당하는 것이 존재하면 빠르게 queue[번호]로 접근해서 해당 task_struct를 RQ에 넣거나 주어진 Time slice가 다했는데 수행할 작업이 남아 있다면 RQ에서 WQ로 집어넣는 등의 작업을 수행한다. 결국 Scheduling이란 작업이 Multi-Tasking을 지원하는데 있어 핵심인 기술이다. 여러 Task들을 동시다발적으로 동작하는 것처럼 보이게하는 트릭이라 할 수 있겠다.

68.
Q. 현재 삽입된 디바이스 드라이버의 리스트를 보는 명령어는 무엇인가 ?
A. lsmod

69.
Q. System Call Mechanism에 대해 기술하시오.
A. System Call은 User가 Kernel에 요청하여 작업을 처리할 수 있도록 지원한다. 내부적으로 굉장히 복잡한 과정을 거치지만 User가 read()등만 호출하면 실질적인 복잡한 모든 과정은 Kernel이 수행하게 되어 편의성을 제공해준다.

70.
Q. Process와 VM과의 관계에 대해 기술하시오.
A. Process는 자신의 고유한 공간으로 가상의 4GB를 가지고 있다. 실제 이 공간을 모두 사용하지 않으며 Demand On Paging에 따라 필요한 경우에만 실제 물리메모리 공간을 Paging Mechanism을 통해 할당받아 사용한다.

73.
Q. **Linux에서 fork()를 수행하면 Process를 생성한다. 이때 부모 프로세스를 gdb에서 디버깅하고자하면 어떤 명령어를 입력해야 하는가 ?**
A.

74.
Q. **C.O.W Architecture에 대해 기술하시오.**
A. Copy On Write의 약자로 실질적인 쓰기가 발생하는 시점에 복사를 하겠다는 것이다. 프로세스가 사용하는 모든 메모리 공간을 일일히 모두 복사하면 그 복사비용이 상당히 크다는 것으로 실제쓰기가 발생하기 전까지는 실질적인 복사를 수행하지 않는다.
그렇게 함으로써 시스템 전체의 부하를 상대적으로 많이 줄일 수 있다.

75.
Q. **Blocking 연산과 Non-Blocking 연산의 차이점에 대해 기술하시오.**
A. Blocking 연산의 경우 CPU를 계속 잡고 있으므로 다른 일을 수행할 수 없다.
Non-Blocking 연산의 경우 CPU를 잡고 있지 않고 이런 일을 할꺼니까 '준비되면 알아서 해줘' 라는 식으로 처리를 하기 때문에 Blocking 연산에 비해 성능면에서 뛰어나다. 그러나 반드시 Blocking 연산이 필요한 경우 또한 존재한다.(e.g Network 통신)

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

81.
Q. stat(argv[2], &buf)일때 stat System Call을 통해 채운 buf.st_mode의 값에 대해 기술하시오.
A. buf.st_mode에는 리눅스 커널 inode의 i_mode와 같은 값이 들어가 있다.
파일의 종류 4bit와 setuid, setgid, sticky bit, 그리고 rwx가 3개씩 존재한다.

82.
Q. 프로세스들은 최소 메모리를 관리하기 위한 mm, 파일 디스크립터인 fd_array, 그리고 signal을 포함하고 있는데 그 이유에 대해 기술하시오.
A. 자신이 실제 메모리 어디에 위치하는지에 대한 정보가 필요하고 또 자신이 하드 디스크의 어떤 파일이 메모리에 로드되어 프로세스가 되었는지의 정보가 필요하다.
마지막으로 프로세스들 간에 signal을 주고 받을 필요가 있기 때문에 signal이 필요하다.

87.
Q.클라우드 기술의 핵심인 OS 가상화 기술에 대한 질문이다. OS 가상화에서 핵심에 해당하는 3 가지를 기술하시오.
A.

93.
Q. Critical Section 이 무엇인지 기술하시오.
A. 프로세스나 스레드가 동시 다발적으로 접근하여 정보를 공유할 수 있는 영역을 의미함
연산이 아직 끝나지 않은 상태로 연산을 수행하게 될 경우 해당 연산이 의미가 없어질 수 있다는 문제점이 존재함

94.
Q. 유저에서 fork() 를 수행할때 벌어지는 일들 전부를 실제 소스 코드 차원에서 해석하도록 하시오.
A. 우리가 만든 프로그램에서 fork()가 호출되면 C library에 해당하는 glibc의 __libc_fork()가 호출됨
이 안에서 ax레지스터에 시스템 콜 번호가 기록된다. 즉, sys_fork()에 해당하는 시스템 콜 테이블의 번호가 들어가고 이후에 int 0x80을 통해서 128 번 시스템 콜을 호출하게 된다.
그러면 제어권이 커널로 넘어가서 int_table(Interrupt Descriptor Table)로 가고 여기서 시스템 콜은 128번 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_DEFINE(fork)와 같이 구성되어 있다.

95.
Q. 리눅스 커널의 arch 디렉토리에 대해서 설명하시오.
A. CPU(HW) 의존적인 코드가 위치한 영역이다.

96.
Q. 95 번 문제에서 arm 디렉토리 내부에 대해 설명하도록 하시오.
A. ARM은 하위 호환이 안되고 다양한 반도체 벤더들이 개발을 하고 있기 때문에 해당 디렉토리에 들어가면 회사별 주요 제품들의 이름이 보이는 것을 확일할 수 있다.

97.
Q. 리눅스 커널 arch 디렉토리에서 c6x 가 무엇인지 기술하시오.
A. TI DSP에 해당하는 architecture임

98.
Q. Intel 아키텍처에서 실제 HW 인터럽트를 어떤 함수를 가지고 처리하게 되는지 코드와 함께 설명하시오.
A. 일반적인 HW 인터럽트는 어셈블리 루틴 common_interrupt 레이블에서 처리한다.

이 안에서 do_IRQ() 라는 함수가 같이 함께 일반적인 HW 인터럽트를 처리하기 위해 분할한다.

99.
Q. ARM 에서 System Call 을 사용할 때 사용하는 레지스터를 적으시오.
A. r7

Programme

54.
Q.
A.

63.
Q. 파일의 종류를 확인하는 프로그램을 작성하시도록 하시오.
A.

64.
Q. 서버와 클라이언트가 1 초 마다 Hi, Hello 를 주고 받게 만드시오.
A.

65.
Q. Shared Memory를 통해 임의의 파일을 읽고 그 내용을 공유하도록 프로그래밍하시오.
A.

71.
Q.
A.

72.
Q.
A.

76.
Q.
A.

77.
Q.
A.

78.
Q.
A.

79.
Q.
A.

83.
Q.
A.

84.
Q.
A.

85.
Q.

A.

86.
Q.
A.

88.
Q.
A.

89.
Q.
A.

90.
Q.
A.

91.
Q.
A.

92.
Q.
A.

Q.
1. 레지스터 ip(pc)에서 (pc)의미??(7번 문제)
2. Therad leader = Process??(9번 문제)
3. Cache Affinity = Processor Affinity, CPU pinning : 전부 동일
4. 27번문제 이해x(Text영역에 해당하는 부분은 Machine language 아닌가요??)
5. Paging을 SW적으로도 구동(Paging), HW적으로도 구동(MMU) 둘이 어떤 방식으로 구현 : 가상메모리주소를 물리메모리주소로 변환하는 방법에는 2가지가 존재한다. pgd를 이용한 Paging 방법과 CPU의 MMU를 통한 방법이 있다. 상황에 따라 둘 중에 한가지 방법을 선택해서 사용한다.
6. 하드디스크의 최소단위(diskblock?? -> 이것은 논리적 하드디스크의 최소단위??)(31번 문제)
7. 커널스택으로 주어진 메모리 공간?? Task마다 kernel이 존재하여 task_struct생성시마다 kernel 새로 생성?? 커널 스택공간이 없어지지 않고 유지??(44번 문제)

참고.
Chapter 6, 7, 8, 9, 부록 숙지 후 풀어야 할 문제
52번, 62번,

참고
시험범위외 문제
55번, 56번, 57번, 58번, 60번, 73번

참고.
32번 공부
40번 공부
41번 공부
43번 공부
47번 공부
52번 공부
59번 공부

61번 공부
80~99 복습