Xilinx Zynq FPGA, TI DSP MCU 기반의

프로그래밍 및 회로 설계 전문가

강사 이상훈

(Innova Lee)

Gcccompil3r@gmail.com

학생 김민호

minking12@naver.com

3.리눅스 커널은 운영체제(OS)다.
OS 가 관리해야 하는 제일 중요한 5 가지에 대해 기술하시오.

1. Filesystem Manager(/usr/src/kernels → fs)
2. Device Manager(/usr/src/kernels → driver)
3. Memory Manager(/usr/src/kernels → mm)
4. Task Manager(/usr/src/kernels → kernel)
5. Network Manager(/usr/src/kernels → net)

4. Unix 계열의 모든 OS 는 모든 것을 무엇으로 관리하는가 ?

File

6. 32bit System 에서 User 와 Kernel 의 Space 구간을 적으시오.

32bit cpu 는 4gb 크기의 가상공간을 할당 . 0~3 을 user 에게 나머지를 커널 공간으로 사용

7.Page Fault 가 발생했을때 운영체제가 어떻게 동작하는지 기술하시오.
Page Fault Handler 가 동작
1. DMA Controller 에게 해당 Page 의 주소를 전달
2. DMA Controller 는 해당 Page 를 Physical Memory 로 로드
3. DMA Controller 의 작업이 끝나면 인터럽트 발생
4. 운영체제는 Page Fault 를 일으킨 프로세스의 수행을 재개

8. 리눅스 실행 파일 포맷

ELF(Executable and Linkable Format)

9. 프로세스와 스레드를 구별하는 방법에 대해 기술하시오.

프로세스안에 스레드를 생성시
pid 와 tgid 값이 생성되는데
스레드가 하나면 tgid 가 pid 와 같을 것이고 스레드를 여러개 생성했으면 첫번째 스레드 말고는 pid 값과 tgid 값이 다르다.

10. Kernel 입장에서 Process 혹은 Thread 를 만들면 무엇을 생성하는가 ?
task_struct

11. 리눅스 커널 소스에 보면 current 라는 것이 보인다.
이것이 무엇을 의미하는 것인지 적으시오. → 현재 실행중인 사용자 프로세스를 나타냄.
커널 소스 코드와 함께 기술하시오.

```
#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 percpu_read_stable(current_task);
}

#define current get_current()

#endif /* __ASSEMBLY__ */

#endif /* _ASM_X86_CURRENT_H */
```

12. Memory Management 입장에서 Process 와 Thread 의 핵심적인 차이점은 무엇인가 ?

메모리 사용이나 공유시 독립적이냐 종속적이냐의 차이
프로세스는 독립적 스레드는 종속적
전역변수를 바꾸면 스레드는 변경된다.

13. Task 가 관리해야하는 3 가지 Context 가 있다.
System Context, Memory Context, HW Context 가 있다.
이중 HW Context 는 무엇을 하기 위한 구조인가 ?

Context save

15. TASK_INTURRUPTIBLE 과 TASK_UNINTERRUPTIBLE 은 왜 필요한지 기술하시오.

어떤 조건이 발생하기를 기다리는 대기(sleep) 혹은 중단(block)상태이다. 기다리는 조건이 발생하면, 커널은 프로세스의 상태를 TASK_RUNNING(ready)으로 바꾸는 기능.

TASK_UNINTERRUPTIBLE 은 어떤 시그널을 받더라도 반응하지 않게 하려고 그냥 running 상태 안가게 하려고 ..

16. O(N)과 O(1) Algorithm 에 대해 기술하시오.
그리고 무엇이 어떤 경우에 더 좋은지 기술하시오.

시간의 복잡도를 알려주는 알고리즘.

o(n)은 수가 적으면 처리 시간이 길지 않지만 처리할 개수가 많아지면 그만큼 시간도 늘어난다.
o(1)은 처리할 개수와 시간이 같으니 개수가 많을때 사용한다.

19. UMA 와 NUMA 에 대해 기술하고
Kernel 에서 이들을 어떠한 방식으로 관리하는지 기술하시오.
커널 내부의 소스 코드와 함께 기술하도록 하시오.

UMA(Uniform Memory Access,균일 기억 장치 접근)

-모든 프로세서 들이 상호간에 연결되어서 하나의 메모리를 공유하는 기술
-프로세서 들은 메모리의 어느 영역 이던지 접근이 가능하며, 모든 프로세서가 걸리는 시간은 동일하다.
-이에 따라서 구조가 간단하고, 프로그래밍 하기가 쉽다.
-하지만 메모리에 한번에 하나씩 연결 만 이 가능하기에, 커지면 커질수록 효율성이 떨어진다.

NUMA(Non-Uniformed Memory Access,불균일 기억 장치 접근)

-UMA 가 가지고 있는 모델의 한계를 극복하고, 더 큰 시스템을 만들기 위해 만들어졌다.
-메모리에 접근하는 시간이, 프로세서와 메모리의 상대적인 위치에 따라서 달라진다.
-클러스터 시스템과 원리가 유사하기 때문에, 작은 클러스터 시스템이라 부르기도 한다.

21. ZONE_HIGHMEM 에 대해 아는대로 기술하시오.

하드웨어적인 한계 때문에 커널은 모든 페이지를 동등하게 다루지 못한다.
이런 제약 때문에 커널은 페이지를 서로 다른 영역으로 구분하여 관리한다.

ZONE_HIGHMEM:이 영역은 "상위 메모리"를 포함한다. 상위 메모리는 커널 주소 공간으로 영구적으로 매핑되지 않는다
ZONE_HIGHMEM 에 포함되는 메모리를 상위 메모리, 시스템의 나머지 메모리를 하위 메모리라고 부른다.
ZONE_NORAML 은 보통 앞의 두 영역을 제외한 나머지 영역을 가리킨다.

22. 물리 메모리의 최소 단위를 무엇이라고 하며 크기가 얼마인가 ?
그리고 이러한 개념을 SW 적으로 구현한 구조체의 이름은 무엇인가 ?

페이지 프레임 , 4kb, ?

25. Kernel 은 Memory Management 를 수행하기 위해 VM(가상 메모리)를 관리한다.
가상 메모리의 구조인 Stack, Heap, Data, Text 는 어디에 기록되는가 ?

페이지 테이블

28. User Space 에도 Stack 이 있고 Kernel Space 에도 Stack 이 존재한다.
좀 더 정확히는 각각에 모두 Stack, Heap, Data, Text 의 메모리 기본 구성이 존재한다.
그 이유에 대해 기술하시오.

2 개의 CPU 실행 컨텍스트가 있기 때문에 2 개의 스택이 있습니다.
사용자 모드 스택은 함수, 지역 변수, 리턴 주소 등을위한 스택 프레임 생성과 관련하여 프로그램에 제공 될 것입니다. CPU 가 컨텍스트를 커널 모드로 전환 할 때 (예 : 시스템 호출 실행 중) 커널 메모리 및 데이터 구조에 액세스해야합니다 커널 스택을 사용하도록 전환합니다.

https://code.i-harness.com/ko/q/4b68a8

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

MMU 는 memory management unit 의 약자로, 메모리 관리의 핵심적인 역할을 담당합니다.
실제 메모리와 가상 메모리 사이에서 주소 변환 역할.

31. 하드디스크의 최소 단위를 무엇이라 부르고 그 크기는 얼마인가 ?

1 섹터(512 바이트)

34. task_struct 에서 super_block 이 하는 역할은 무엇인가 ?

슈퍼블럭은 파일시스템마다 불리는 이름이 다르며, MS-DOS 의 경우에는 FAT(File Allocation Table)이라고 부른다.
여기에는 파일을 찾기 위한 기본적인 정보들이 들어 있다. 물론, 유닉스 계열의 파일 시스템에서는 이에 대한 내용이 i-node 테이블에서 주로 관리된다.
일반적으로 슈퍼블럭은 해당 파일시스템에 대한 기본적인 정보들을 가지고 있다고 보면 된다.
https://m.blog.naver.com/PostView.nhn?blogId=credenda&logNo=80171512907&proxyReferer=https%3A%2F%2Fwww.google.co.kr%2F

36. Linux Kernel 에서 Interrupt 를 크게 2 가지로 분류한다.
그 2 가지에 대해 각각 기술하고 간략히 설명하시오.

Sw 인터럽트 → system call
Hw 인터럽트 → 키보드, 타이머 등 시스템 프로그램

44. 커널 내부에서 메모리 할당이 필요한 이유는 무엇인가 ?

사용자 모드에서 수행중인 프로세스가 추가적인 메모리를 요구하면 커널이 관리하는 자유 페이지 프레임의 리스트에서 페이지들이 할당된다. 페이지 교체 정책으로 인하여 물리 메모리 상에 흩어져 있는 자유 페이지들로 채워진다. 사용자 프로세스가 1 바이트만 필요하더라도 프로세스가 한 페이지 프레임을 할당 받았으므로 내부 단편화가 발생 한다.

커널 메모리는 보통 사용자 모드 프로세스에게 할당해 주기 위한 페이지 리스트와는 별도의 자유 메모리 풀에서 할당 받는다. 이는 다양한 크기의 자료 구조를 할당 할 수 있으며 단편화에 의한 낭비를 최소화 한다. 또한 사용자 모드 프로세스에 할당되는 페이지들은 물리 메모리에서 굳이 연속된 것일 필요가 없기 때문이다.

https://sqlangeles.com/2018/01/11/%EC%BB%A4%EB%84%90-%EB%A9%94%EB%AA%A8%EB%A6%AC-%ED%95%A0%EB%8B%B9/

45. 메모리를 불연속적으로 할당하는 기법은 무엇인가 ?
페이징

46. 메모리를 연속적으로 할당하는 기법은 무엇인가 ?
세그먼테이션

47. Mutex 와 Semaphore 의 차이점을 기술하시오.

세마포어(Semaphore):공유된 자원의 데이터를 여러 프로세스가 접근하는 것을 막는 것
뮤텍스(Mutex):공유된 자원의 데이터를 여러 쓰레드가 접근하는 것을 막는 것
- 세마포어는 뮤텍스가 될 수 있지만 뮤텍스는 세마포어가 될 수 없다.
- 세마포어는 소유할 수 없는 반면 뮤텍스는 소유가 가능하며 소유주가 이에 대한 책임을 진다.
- 뮤텍스의 경우 뮤텍스를 소유하고 있는 쓰레드가 이 뮤텍스를 해제할 수 있다. 하지만 세마포어의 경우 이러한 세마포어를 소유하지 않는 스레드가 세마포어를 해제할 수 있다.
- 세마포어는 시스템 범위에 걸쳐있고 파일시스템상의 파일 형태로 존재한다. 반면 뮤텍스는 프로세스 범위를 가지며 프로세스가 종료될 때 자동으로 clean up 된다.
★★★ 가장 큰 차이점은 관리하는 동기화 대상이 갯수이다.
뮤텍스는 동기화 대상이 오직 하나뿐일 때, 세마포어는 동기화 대상이 하나 이상일 때 사용한다.

출처: http://ninako21.tistory.com/500[infatuation]

48. module_init() 함수 호출은 언제 이루어지는가 ?
드라이버 초기화 시.
49. module_exit() 함수 호출은 언제 이루어지는가 ?
드라이버 해지시, 사용 자원 해지시.
50. thread_union 에 대해 기술하시오.
태스크당 할당되는 커널 스택이며
thread info 구조체를 포함하고 있다.

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

어떤 한 시점에 프로세서가 동시에 더 많은 명령어를 수행하고 싶다.
그것을 ILP 의 지표로 생각할 수 있다.
더 큰 ILP 를 얻기 위해 생각할 수 있는 두 가지 대표적인 방법들이 있다.
- (Super) Pipelining : 한 인스트럭션을 여러 단계로 쪼개서 처리.
- Multiple Issue : 한 번에 여러 인스트럭션을 이슈. 파이프라인이 여러개 존재하는 것으로 생각할 수 있다.

이 때, multiple issue 를 구현하는 두 가지 방법이 존재한다.
- Static multiple-issue(VLIW): 컴파일 타임에 정적으로 병렬화. 하나의 인스트럭션 대신 여러 인스트럭션이 모여있는 패킷을 issue. SW-based.
- Dynamic multiple-issue(Superscalar): 런타임에 동적으로 의존성을 확인한 뒤 병렬화. HW-based.

이 때, Superscalar machine 에서의 성능 극대화를 위해선 비순차적 명령어 처리(Out of Order Execution)가 필요하다. 근본적으로, 이는 명령어마다 처리하는데에 걸리는 시간이 다르며, 데이터/컨트롤 의존성이 존재하기 때문이다.

https://hjlog.me/post/87

57. CISC Architecture 와 RISC Architecture 에 대한 차이점을 기술하라.

분류	CISC	RISC
레지스터	8~16 개의 범용 레지스터 사용 부동소수점 연산 제공	16~32 개의 범용 레지스터 사용 부가적인 레지스터 사용 가능 기본적인 연산 제공
설계목적	최소의 프로그램 길이 1 개의 명령어로 최대의 동작	프로그램의 길이는 길어도 명령어 당 실행 시간의 최소화
명령어형식	다양한 길이와 형식 LOAD /STORE,레지스터와 메모리의 다양한 명령어 형식 제공	고정길이의 명령어 제공 두가지 형식 제공
명령어군	OS 와 RUN TIME 유틸리티를 지원하는 데이터 형과 명령어 제공	LOAD/STORE 범용 레지스터의 데이터 연산
인코딩	1 개의 명령어 = 1 개의 문장	1 개의 명령어 = 1 개의 오퍼랜드나 1 개 연산
구현측면성	마이크로 프로그램 제어 방식의 프로 웨어로 구성	하드웨어 제어 방식의 프로세서와 소프트 세서로 구성
기타특징	하드웨어가 강조됨	소프트웨어가 강조됨
코드크기와 사이클	작은 코드 크기, 단위시간동안 높은 사이클	단위시간동안 낮은 사이클수, 큰 코드 크기
트렌지스터	축약명령어를 저장을 위해 트랜지스터가 사용됨	메모리 레지스터에 보다 많은 트랜지스터가 사용됨

61 Intel 의 Hyper Threading 기술에 대해 상세히 기술하시오.

https://www.intel.co.kr/content/www/kr/ko/architecture-and-technology/hyper-threading/hyper-threading-technology.html

인텔® 하이퍼 스레딩 기술(인텔® HT 기술)은 프로세서 리소스를 보다 효율적으로 사용하여 각 코어에서 여러 개의 스레드를 실행할 수 있도록 합니다. 프로세서 처리량과 스레드가 많은 소프트웨어의 전반적인 성능을 높여주는 성능 기능입니다.

- 시스템 응답 속도를 유지하면서 부담이 큰 여러 개의 응용 프로그램을 동시에 실행생산성에 미치는 영향을 최소화하면서 시스템을 안전하게 보호하고 효율적이며 관리 가능한 상태로 유지미래 비즈니스 성장 및 새로운 솔루션 도입을 위한 역량 확보

- 기능 저하 없는 집중 그래픽 처리
인텔® HT 기술을 통해 멀티미디어 애호가는 백그라운드에서 바이러스 보호 소프트웨어와 같은 응용 프로그램을 실행하면서도 시스템 성능에 영향을 받지 않고 그래픽 집약적인 파일을 생성, 편집 및 인코딩할 수 있습니다.
- 더 많은 작업, 더 효율적인 비즈니스
인텔® HT 기술과인텔® 터보 부스트 기술(또는 인텔® 터보 부스트 기술 2.0, 인텔® 코어™ i5 프로세서 이상에서 사용 가능)이 모두 적용된 프로세서는 향상된 성능을 제공하고 작업을 더욱 신속하게 완료할 수 있습니다. 이러한 기술의 결합을 통해 여러 스레드의 동시 처리가 가능하고 작업량에 맞게 동적으로 조정되고 사용되지 않는 코어는 자동으로 비활성화됩니다. 사용 중인 코어에 프로세서 주파수는 증가하여 스레드가 있는 응용 프로그램에 더욱 향상된 성능을 제공합니다.

인텔® HT 기술로 기업에서는 다음 작업들을 수행할 수 있습니다:

- 속도 저하 없이 더 많은 동시 작업을 수행함으로써 생산성 향상
- 인터넷 및 전자 상거래 응용 프로그램에 대하여 빠른 응답 시간을 제공하여 고객 사용 환경 개선
- 동시 처리 가능한 거래 수 증가
- 미래의 64 비트에 대비하면서 기존의 32 비트 응용 프로그램 기술 활용

97. 리눅스 커널 arch 디렉토리에서 c6x 가 무엇인지 기술하시오.
TI 사의 DSP

95. 리눅스 커널의 arch 디렉토리에 대해서 설명하시오.
회사별로 cpu 운영체계가 들어가 있다.

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

93. Critical Section 이 무엇인지 기술하시오.
https://m.blog.naver.com/PostView.nhn?blogId=cmw1728&logNo=220479810666&proxyReferer=https%3A%2F%2Fwww.google.co.kr%2F

1. 임계 영역(Critical Section)

-원래 임계 영역이란 둘 이상의 쓰레드가 동시에 실행될 경우 생길 수 있는 동시 접근 문제를 발생시킬 수 있는 코드 블록을 임계 영역이라고 한다.
2. 크리티컬 섹션
-크리티컬 섹션이란 위에서 설명한 임계 영역에 대한 문제를 해결하기 위한 동기화 기법을 의미한다.
-마이크로소프트에서 Critical Section(원래의 임계 영역)에 대한 동기화 기법을 똑같이 Critical Section 이라고 이름을 지어서 헷갈릴 수 있다.
-특정 임계 영역에 대한 키(크리티컬 섹션 오브젝트)를 가져야만 임계영역에 접근하도록 하는 것이다.

73. Linux 에서 fork()를 수행하면 Process 를 생성한다.
이때 부모 프로세스를 gdb 에서 디버깅하고자하면 어떤 명령어를 입력해야 하는가 ?

https://kldp.org/node/24364

a.out 이 fork 를 해서 수행이 된다고 했을때...
$ ps aux
...
1234 ... a.out
1236 ... a.out
$
였을때, 원래 프로세스가 1234 이고 fork 가 된 프로세스가 1236 이면...
$ gdb a.out 1236

인지를 하면, 수행중인 프로세스에서 debugging 이 됩니다.

74. C.O.W Architecture 에 대해 기술하시오.
http://www.iamroot.org/xe/index.php?mid=Kernel&document_srl=23297

1. 현재 리눅스에서 fork 는 COW(copy on write) 기법을 사용하여 프로세스 생성시 모든 자원을 복사하는 것이 아니고,변경사항이 생길 경우에만 복사하도록 구현되어 있다고 합니다.
따라서 현재 fork 가 갖는 단점은 부모 프로세스의 페이지 테이블을 복사하는 것과 자식 프로세스를 기술하기 위한 프로세스 구조체를 할당받는 시간과 메모리 뿐이라고 합니다.

2. vfork()처럼 부모 프로세스의 데이터 부분에 대한 참조만 소유하고 있다가 실제 변경이 발생하는 시점에 복사를 하여 사용하는 방식.

79. goto 는 굉장히 유용한 C 언어 문법이다.
그러나 어떤 경우에는 goto 를 쓰기가 힘든 경우가 존재한다.
이 경우가 언제인지 기술하고 해당하는 경우
문제를 어떤식으로 해결 해야 하는지 프로그래밍 해보시오.

http://soen.kr/lecture/ccpp/cpp1/4-5-1.htm

goto 문이 많으면 이리 저리 뛰어 다니느라 정신이 없다.
얼키고 설킨 코드를 스파게티 소스라고 하는데 마치 스파게티 면발이 꼬여 있는 것처럼 복잡해서 웬만한 체력으로는 이런 코드를 관리하기 어렵다.
또한 goto 문은 프로그램의 구조를 해치기 때문에 goto 문을 사용한 소스는 이식성과 재사용에 무척 불리하다.
특정 동작을 하는 코드를 다른 프로그램에서 재사용하려면 goto 문에 의해 엉켜 있는 실을 다 풀어야 하고 옮긴 후에 다시 그 프로그램에 맞게 연결해야 하기 때문이다.
아무 규칙이나 형식없이 제어를 마음대로 옮길 수 있다보니 부작용이 많다.
이런 여러 가지 이유로 아주 특별한 경우가 하닌 한은 goto 문을 사용하지 말 것을 권장하고 있다.
goto 문이 아니면 도저히 해결할 수 없는 그런 문제는 없다.
goto 문이 없어도 for, while, switch 같은 제어문으로 필요한 모든 구조를 다 만들 수 있다는 것이 이미 수학적으로 증명되어 있다.

77. 데몬 프로세스를 작성하시오.
잠시 동안 데몬이 아니고 영구히 데몬이 되게 하시오.

https://stackoverflow.com/questions/17954432/creating-a-daemon-in-linux

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <syslog.h>

static void skeleton_daemon()
{
    pid_t pid;

    /* Fork off the parent process */
    pid = fork();

    /* An error occurred */
    if (pid < 0)
        exit(EXIT_FAILURE);

    /* Success: Let the parent terminate */
    if (pid > 0)
        exit(EXIT_SUCCESS);

    /* On success: The child process becomes session leader */
    if (setsid() < 0)
        exit(EXIT_FAILURE);

    /* Catch, ignore and handle signals */
    //TODO: Implement a working signal handler */
    signal(SIGCHLD, SIG_IGN);
    signal(SIGHUP, SIG_IGN);

    /* Fork off for the second time*/
    pid = fork();

    /* An error occurred */
    if (pid < 0)
        exit(EXIT_FAILURE);

    /* Success: Let the parent terminate */
    if (pid > 0)
        exit(EXIT_SUCCESS);

    /* Set new file permissions */
    umask(0);

    /* Change the working directory to the root directory */
    /* or another appropriated directory */
    chdir("/");

    /* Close all open file descriptors */
    int x;
    for (x = sysconf(_SC_OPEN_MAX); x>=0; x--)
    {
        close (x);
    }

    /* Open the log file */
    openlog ("firstdaemon", LOG_PID, LOG_DAEMON);
}

int main()
{
    skeleton_daemon();

    while (1)
    {
        //TODO: Insert daemon code here.
        syslog (LOG_NOTICE, "First daemon started.");
        sleep (20);
        break;
    }

    syslog (LOG_NOTICE, "First daemon terminated.");
    closelog();

    return EXIT_SUCCESS;
}
-- INSERT --
```

```
minking@minking-Z20NH-AS51B1U:~/2test2$ ps -xj | grep firstdaemon
 9849 10167 10166  9849 pts/0    10166 S+    1000   0:00 grep --color=auto firstdaemon
minking@minking-Z20NH-AS51B1U:~/2test2$
```

75. Blocking 연산과 Non-Blocking 연산의 차이점에 대해 기술하시오.

블럭/논블럭는 함수호출에서의 이야기이다.
- A 라는 함수를 호출했을때, A 라는 함수를 호출 했을 때 기대하는 행위를 모두 끝마칠때까지 기다렸다가 리턴되면, 이것은 블로킹 되었다고 한다.
- A 라는 함수를 호출 했는데, A 라는 함수를 호출 했을 때 기대하는 어떤 행위를 요청하고바로 리턴되면 이것은 논블럭킹 되었다고 한다.

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

정의 : 컴퓨터 프로그래밍에서 파일디스크립터란 파일에 접근하기 위해 추상화 시켜놓은 장치를 이야기 합니다.
리눅스는 모든 장치를 파일로 관리 하기 때문에 파일디스크립터를 이용하여 장치에 접근 할 수 있게 (핸들링할수있게) 됩니다.

유닉스(리눅스) 계열의 시스템에서 열린파일(디바이스 장치)을 구분하는 단위이기도 합니다.

커널은 프로세스 단위로 열린파일 목록 (open()을 통해 파일 즉, 일반 파일및 모든 장치 를 열게 되면 디스크립터 번호가 부여되고 이것이 테이블로 관리됨) 을 담아 둘 수 있는 테이블을 관리하는데, 파일 디스크립터가 이 테이블에 등록되어 관리되고 있습니다.

테이블에 새로운 디스크립터를 등록하게 되면 파일디스크립터는 0 부터 순차적으로 1 만큼 자동으로 등록 되게 됩니다.
윈도우의 핸들과 개념은 거의 같다고 봐도 무방하지만 순차 등록되지 않는다는 차이점을 가지고 있습니다.

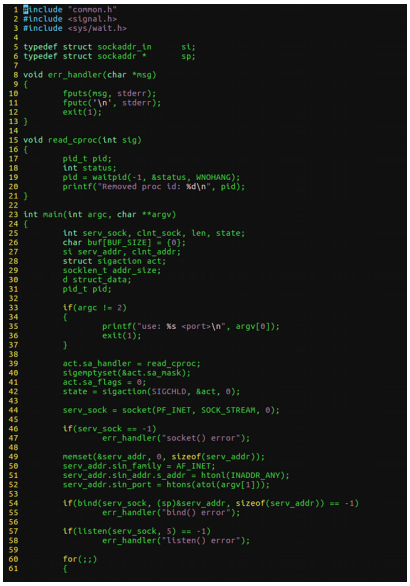
그럼 프로세스를 실행할 대 open 함수를 통해 파일을 하나 열었다고 생각해 봅시다. 그 파일의 파일디스크립터는 0 번일까요?
답은 아니오 입니다. '0', '1', '2' 의 번호는 이미 예약되어 있기 때문입니다.

'0' -> 표준입력 : 키보드
'1' -> 표준출력 : 모니터
'2' -> 표준에러 : 모니터

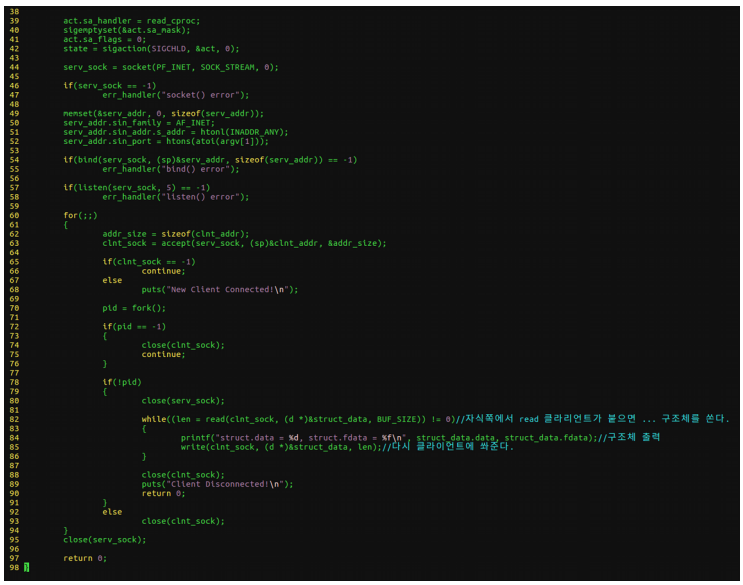
3 번부터 번호를 부여 받는다.

87. 클라우드 기술의 핵심인 OS 가상화 기술에 대한 질문이다.
OS 가상화에서 핵심에 해당하는 3 가지를 기술하시오.
Isolation, Partitioning, Hardware Independence

91. 네트워크 상에서 구조체를 전달할 수 있게 프로그래밍 하시오.
선생님 소스 … 입니다… 시간이 너무 많이 남아서…



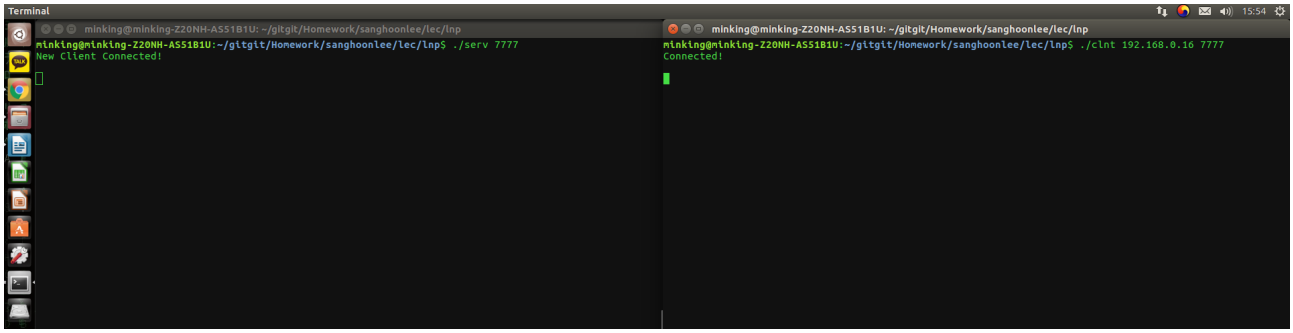
```c
#include "common.h"
#include <signal.h>
#include <sys/wait.h>

typedef struct sockaddr_in      si;
typedef struct sockaddr *       sp;

void err_handler(char *msg)
{
        fputs(msg, stderr);
        fputc('\n', stderr);
        exit(1);
}
void read_cproc(int sig)
{
        pid_t pid;
        int status;
        pid = waitpid(-1, &status, WNOHANG);
        printf("Removed proc id: %d\n", pid);
}

int main(int argc, char **argv)
{
        int serv_sock, clnt_sock, len, state;
        char buf[BUF_SIZE] = {0};
        si serv_addr, clnt_addr;
        struct sigaction act;
        socklen_t addr_size;
        d struct_data;
        pid_t pid;

        if(argc != 2)
        {
                printf("use: %s <port>\n", argv[0]);
                exit(1);
        }

        act.sa_handler = read_cproc;
        sigemptyset(&act.sa_mask);
        act.sa_flags = 0;
        state = sigaction(SIGCHLD, &act, 0);

        serv_sock = socket(PF_INET, SOCK_STREAM, 0);

        if(serv_sock == -1)
                err_handler("socket() error");

        memset(&serv_addr, 0, sizeof(serv_addr));
        serv_addr.sin_family = AF_INET;
        serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
        serv_addr.sin_port = htons(atoi(argv[1]));

        if(bind(serv_sock, (sp)&serv_addr, sizeof(serv_addr)) == -1)
                err_handler("bind() error");

        if(listen(serv_sock, 5) == -1)
                err_handler("listen() error");

        for(;;)
        {
```



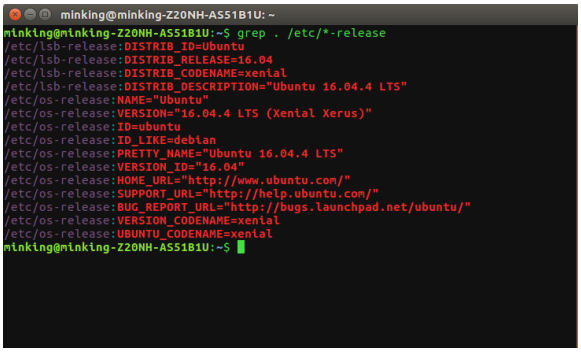
```c

        act.sa_handler = read_cproc;
        sigemptyset(&act.sa_mask);
        act.sa_flags = 0;
        state = sigaction(SIGCHLD, &act, 0);

        serv_sock = socket(PF_INET, SOCK_STREAM, 0);

        if(serv_sock == -1)
                err_handler("socket() error");

        memset(&serv_addr, 0, sizeof(serv_addr));
        serv_addr.sin_family = AF_INET;
        serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
        serv_addr.sin_port = htons(atoi(argv[1]));

        if(bind(serv_sock, (sp)&serv_addr, sizeof(serv_addr)) == -1)
                err_handler("bind() error");

        if(listen(serv_sock, 5) == -1)
                err_handler("listen() error");

        for(;;)
        {
                addr_size = sizeof(clnt_addr);
                clnt_sock = accept(serv_sock, (sp)&clnt_addr, &addr_size);

                if(clnt_sock == -1)
                        continue;
                else
                        puts("New Client Connected!\n");

                pid = fork();

                if(pid == -1)
                {
                        close(clnt_sock);
                        continue;
                }

                if(!pid)
                {
                        close(serv_sock);

                        while((len = read(clnt_sock, (d *)&struct_data, BUF_SIZE)) != 0)//자식쪽에서 read 클라리언트가 붙으면 ... 구조체를 쓴다.
                        {
                                printf("struct.data = %d, struct.fdata = %f\n", struct_data.data, struct_data.fdata);//구조체 출력
                                write(clnt_sock, (d *)&struct_data, len);//다시 클라이언트에 써준다.
                        }

                        close(clnt_sock);
                        puts("Client Disconnected!\n");
                        return 0;
                }
                else
                        close(clnt_sock);
        }
        close(serv_sock);

        return 0;
}
```

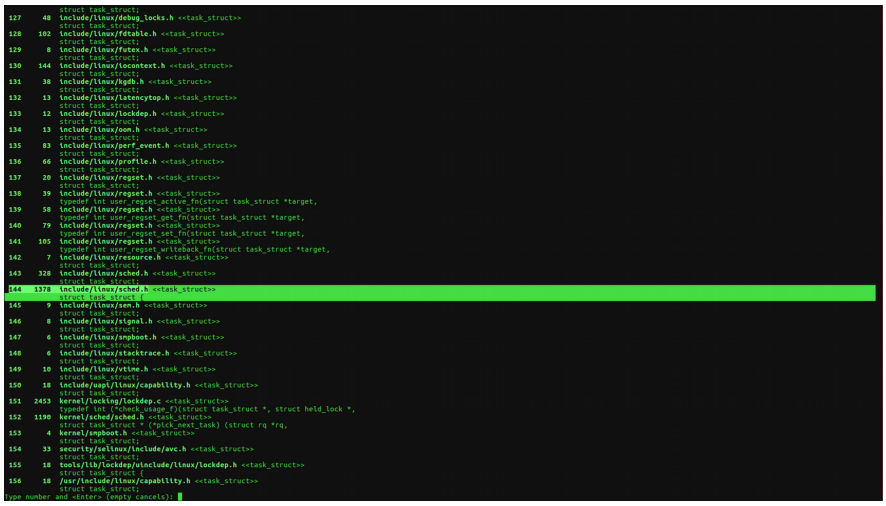
```c
#include "common.h"

void err_handler(char *msg)
{
        fputs(msg, stderr);
        fputc('\n', stderr);
        exit(1);
}
void read_proc(int sock, d *buf)
{
        for(;;)
        {
                int len = read(sock, buf, BUF_SIZE);

                if(!len)
                        return;

                printf("msg from serv: %d, %f\n", buf->data, buf->fdata);
        }
}

void write_proc(int sock, d *buf)
{
        char msg[32] = {0};

        for(;;)
        {
                fgets(msg, BUF_SIZE, stdin);

                if(!strcmp(msg, "q\n") || !strcmp(msg, "Q\n"))
                {
                        shutdown(sock, SHUT_WR);
                        return;
                }

                buf->data = 3;//q안들어가면 값을 전달.
                buf->fdata = 7.7;

                write(sock, buf, sizeof(d));
        }
}

int main(int argc, char **argv)
{
        pid_t pid;
        int i, sock;
        sl serv_addr;
        d struct_data;
        char buf[BUF_SIZE] = {0};

        if(argc != 3)
        {
                printf("use: %s <IP> <port>\n", argv[0]);
                exit(1);
        }

        sock = socket(PF_INET, SOCK_STREAM, 0);

        if(sock == -1)
                err_handler("socket() error");
```

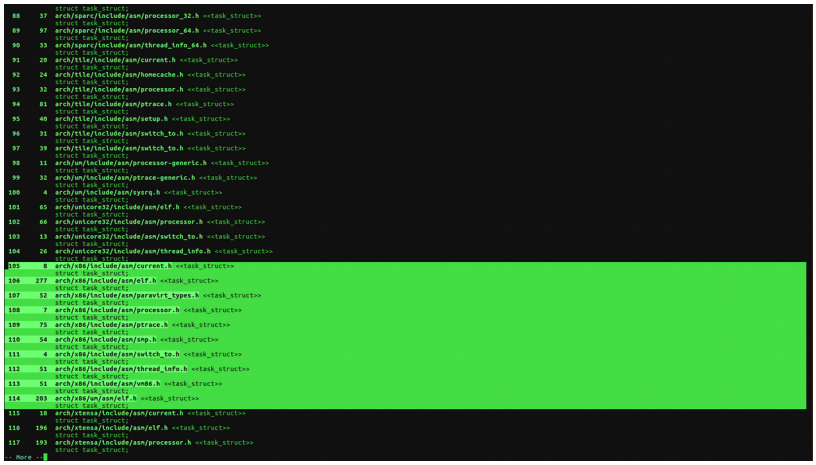
```c
void write_proc(int sock, d *buf)
{
        char msg[32] = {0};

        for(;;)
        {
                fgets(msg, BUF_SIZE, stdin);

                if(!strcmp(msg, "q\n") || !strcmp(msg, "Q\n"))
                {
                        shutdown(sock, SHUT_WR);
                        return;
                }

                buf->data = 3;//q안들어가면 값을 전달.
                buf->fdata = 7.7;

                write(sock, buf, sizeof(d));
        }
}

int main(int argc, char **argv)
{
        pid_t pid;
        int i, sock;
        sl serv_addr;
        d struct_data;
        char buf[BUF_SIZE] = {0};

        if(argc != 3)
        {
                printf("use: %s <IP> <port>\n", argv[0]);
                exit(1);
        }

        sock = socket(PF_INET, SOCK_STREAM, 0);

        if(sock == -1)
                err_handler("socket() error");

        memset(&serv_addr, 0, sizeof(serv_addr));
        serv_addr.sin_family = AF_INET;
        serv_addr.sin_addr.s_addr = inet_addr(argv[1]);
        serv_addr.sin_port = htons(atoi(argv[2]));

        if(connect(sock, (sp)&serv_addr, sizeof(serv_addr)) == -1)
                err_handler("connect() error");
        else
                puts("Connected!\n");

        pid = fork();

        if(!pid)
                write_proc(sock, (d *)&struct_data);
        else
                read_proc(sock, (d *)&struct_data);

        close(sock);

        return 0;
```



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.







78. SIGINT 는 무시하고 SIGQUIT 을 맞으면 죽는 프로그램을 작성하시오.

signal(SIGINT, SIG_IGN)

signal(SIGQUIT ...)

76. 자식이 정상 종료되었는지 비정상 종료되었는지 파악하는 프로그램을 작성하시오.

Wait 기능 이용 .

https://www.joinc.co.kr/w/man/2/wait

```
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <string.h>

int main()
{
  int pid;
  int status;

  pid = fork();
  // 자식 프로세스
  if (pid < 0)
  {
    perror("FORK ERROR :");
    exit(0);
  }

  if (pid == 0)
  {
    int i;
    for (i = 0; i < 5; i++)
    {
      printf("Child : %d\n", i);
      sleep(2);
    }
    exit(3);
  }
  else
  {
    // 부모프로세스는 자식프로세스가
    // 종료할때까지 기다린다.
    printf("I wait Child(%d)\n", pid);
    wait(&status);
    printf("Child is exit (%d)\n", status);
  }
}
```

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

syscall register

60. CPU 들은 각각 저마다 이것을 가지고 있다.
Compiler 개발자들은 이것을 고려해서 Compiler 를 만들어야 한다.
또한 HW 입장에서도 이것을 고려해서 설계를 해야 한다.
여기서 말하는 이것이란 무엇인가 ?

라이브러리

100. 벌써 2 개월째에 접어들었다.

그동안 굉장히 많은 것들을 배웠을 것이다.

상당수가 새벽 3 ~ 5 시에 자서 2 ~ 4 시간 자면서 다녔다.

또한 수업 이후 저녁 시간에 남아서 9 시 ~ 10 시까지 공부를 한 사람들도 있다.

하루 하루에 대한 자기 자신의 반성과 그 날 해야할 일을 미루지는 않았는지 성찰할 필요가 있다.

그날 해야할 일들이 쌓이고 쌓여서 결국에는 수습하지 못할정도로 많은 양이 쌓였을 수도 있다.

사람이란 것이 서 있으면 앉고 싶고 앉으면 눕고 싶고 누우면 자고 싶고 자면 일어나기 싫은 법이다.

내가 정말 죽을둥 살둥 이것을 이해하기 위해 열심히 했는지 고찰해보자!

2 개월간 자기 자신에 대한 반성과 성찰을 수행해보도록 한다.

또한 앞으로는 어떠한 자세로 임할 것인지 작성하시오.

일단 잠을 설쳐가면서 열공한 부류에 끼는지 못끼는지 불분명하다 .

분명 남긴 남아서 했는데 뭘모르고 갔던적이 더 많다.

돈주고 과외라도 받고 싶다.

특히 시스템 프로그래밍을 시작하고서부터는 거의 뇌를 놓고 수업을 들은 것 같다.

커널때는 정신도 몸도 그냥 배배꼬이고 답이 없었다.

커널도 하나도 모르다가 정상용 학우가 안타까워 보였는지 과외를 해주는 바람에 **3** 장까지는 그나마 알게 되었다.

남은기간 버릴건 버리고 챙길건 챙겨야 겠다는 생각이 든다.

하루하루 쌓여가는 진도, 숙제가 부담스럽다.

어떻게 해야하나

쉬는날 나와야겠다

주와 부를 나눠서 생활을 이어가야 하는데,

주말에는 학원을 나갔다는 보상심리에 빠져 나태해진 나를 본게 한두번이 아니다.

무튼 이젠

중단한 느낌이 아니라 잠시 일시정지를 했다는 생각으로,

다시 초기에 열정적이였던 학습 모드로 복귀 해야겠다.