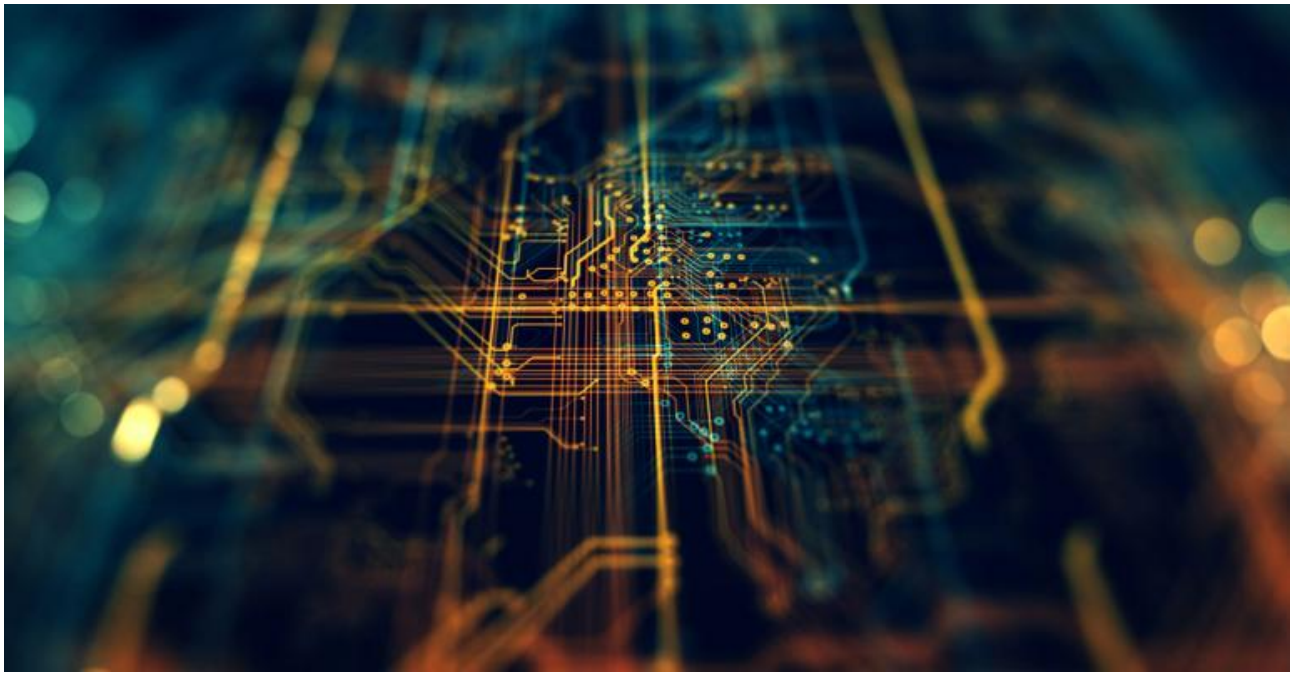
TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정



2018.04.17
2 개월차 시험

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com

학생 – 신민철
akrn33@naver.com

1.파이프 통신을 구현하고 c type.c 라고 입력할 경우
현재 위치의 디렉토리에 type.c 파일을 생성하도록 프로그래밍하시오.

```
#include<unistd.h>
#include<stdio.h>
#include<errno.h>
#include<stdlib.h>
#include<fcntl.h>

int main(void)
{
	int fd,typec, ret;
	char buf[1024];
	char check[] = "ctype.c";
	char pathname[] = "type.c";
	pid_t pid;
	fd = open("myfifo",O_RDWR);
	if((pid = fork()) > 0)
	{
		for(;;)
		{
			ret = read(0, buf, sizeof(buf));
			buf[ret] = 0;
			printf("Keyboard:%s\n",buf);
		}
	}
	else if(pid ==0)
	{
		for(;;)
		{
			ret = read(fd, buf, sizeof(buf));
			buf[ret] = 0;
			printf("myfifo : %s\n",buf);
			if(buf==check){
				typec = creat("type.c",0644);
```

```
                        close(typec);
                    }
                }
        }
        else
        {
                perror("fork()");
                exit(-1);
        }
        close(fd);
        return 0;
}
```

2.369 게임을 작성하시오.
2 초내에 값을 입력하게 하시오.
박수를 쳐야 하는 경우를 Ctrl + C 를 누르도록 한다.
2 초 내에 값을 입력하지 못할 경우 게임이 오버되게 한다.
Ctrl + C 를 누르면 "Clap!" 이라는 문자열이 출력되게 한다.

3.리눅스 커널은 운영체제(OS)다.
OS 가 관리해야 하는 제일 중요한 5 가지에 대해 기술하시오.
메모리,프로세스,파일
4.Unix 계열의 모든 OS 는 모든 것을 무엇으로 관리하는가 ?
task
5.리눅스에는 여러 장점이 있다.(배점 0.2 점)
아래의 장점들 각각에 대해 기술하라.
* 사용자 임의대로 재구성이 가능하다.
* 열약한 환경에서도 HW 자원을 적절히 활용하여 동작한다.
* 커널의 크기가 작다.
* 완벽한 멀티유저, 멀티태스킹 시스템
* 뛰어난 안정성
* 빠른 업그레이드

* 강력한 네트워크 지원
* 풍부한 소프트웨어

6.32 bit System 에서 User 와 Kernel 의 Space 구간을 적으시오.

7.Page Fault 가 발생했을때
운영체제가 어떻게 동작하는지 기술하시오.
페이지폴트 발생시 운영체제는 권한을 확인하고 맞으면 페이지를 재할당해 준다.

8.리눅스 실행 파일 포맷이 무엇인지 적으시오.
ELF

9.프로세스와 스레드를 구별하는 방법에 대해 기술하시오.
프로세스는 메모리를 공유하지 않고 스레드는 메모리를 공유한다.
프로세스는 프로세스리더자격을 가지고 그룹아이디와 프로세스아이디가 동일하다.
스레드는 프로세스아이디와 그룹아이디가 동일하지않다.
10.Kernel 입장에서 Process 혹은 Thread 를 만들면 무엇을 생성하는가 ?
태스크구조체를 만들어서 관리하게된다.
11.리눅스 커널 소스에 보면 current 라는 것이 보인다.
이것이 무엇을 의미하는 것인지 적으시오.
커널 소스 코드와 함께 기술하시오.

12.Memory Management 입장에서 Process 와 Thread 의 핵심적인 차이점은 무엇인가 ?
프로세스는 메모리를 공유하지 않고 스레드는 메모리를 공유한다.
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 ?

14.리눅스 커널의 스케쥴링 정책중 Deadline 방식에 대해 기술하시오.

Deadline 방식은 기존의 FIFO,RR 방식의 단점인 먼저시작한 프로세스가 끝날때까지 다음 프로세스에게
CPU 점유를 양보하지 않고 계속 점유하고있거나, 아니면 100 개가 넘어갈 때 까지 우선순위를 주지 않아서
비효율적인 스케쥴링을 벗어나 HashTable 을 이용한 Key 와 Value 의 값의 쌍으로 이루어진 자료구조를
사용하여서 시간복잡도가 O(N) 에서 O(1)의 시간복잡도를 가질 수 있게 하여 속도가 향상 될 수 있도록
하였다.
15.TASK_INTURRUPTIBLE 과 TASK_UNINTERRUPTIBLE 은 왜 필요한지 기술하시오.

매우 우선순위가 높은 프로그램은 예를들어서 예상치못한 상황이 발생 했을 때 중요한 프로그램이기 때문에
그냥 종료된다면 불이익을 볼 수 있기 때문에 인터럽트를 허용하게 해야한다 하지만 중요도가 높지 않은 프로그램은
인터럽트가 필요없이 꺼져도 불이익을 보지 않는다면 인터럽트를 허용하지 않아도 된다.

16.O(N)과 O(1) Algorithm 에 대해 기술하시오.
그리고 무엇이 어떤 경우에 더 좋은지 기술하시오.
O(N)은 시간이 지나면 지날수록 처리해야할 양이 많아지는데,
O(1)은 시간이 지나도 key 와 value 의 값의 쌍으로 이루어지므로 시간이 지나도 처리해야 할 양은 증가하지 않는다
연산처리가 한번만 실행되지 않는이상 O(1)이 훨씬 안정적이고 좋은 알고리즘이다.

17.현재 4 개의 CPU(0, 1, 2, 3)가 있고 각각의 RQ 에는 1, 2 개의 프로세스가 위치한다.
이 경우 2 번 CPU 에 있는 부모가 fork()를 수행하여 Task 를 만들어냈다.
이 Task 는 어디에 위치하는 것이 좋을까 ?

그리고 그 이유를 적으시오.

Task 는 실행해야할 태스크가 많지 않은 cpu 에 할당된다.
한곳에서 모두 처리하면 처리속도가 다른씨피유와 같이 처리했을 때 보다 느리다.

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

0, 1, 3 번 CPU 중 다음에 할당해야 할 Task 가 적은 곳으로 할당한다.

19.UMA 와 NUMA 에 대해 기술하고 Kernel 에서 이들을 어떠한 방식으로 관리하는지 기술하시오.
커널 내부의 소스 코드와 함께 기술하도록 하시오.
Uniform Memory Access == UMA(단일메모리접근)
Non Uniform Memory Access == NUMA(비단일메모리접근)
단일메모리접근은 버스인터페이스가 하나여서 코어들이 지나가는 통로가 하나뿐이라서 병목현상이 심하다.
이에비해 비단일메모리접근은 버스인터페이스가 두개이상이어서 지나가는 통로가 여러개라 병목현상이 덜하다.
예를 들면 1 차선 도로보다는 8 차선 도로가 병목현상이 없고 차들이 시원시원하게 달리는 것과 비슷한 예다.

20.Kernel 의 Scheduling Mechanism 에 서 Static Priority 와 Dynamic Priority 번호가 어떻게 되는지 적으시오.

Static Priority 는 정적우선순위이고, Dynamic Priority 는 동적우선순위인데,

정적우선순위는 우선순위가 정적이기 때문에 바뀌지 않는다, 하지만 동적우선순위는 우선순위가 높은 순서에 따라서
우선순위가 변하게 된다.

21.ZONE_HIGHMEM 에 대해 아는대로 기술하시오.
물리메모리는 순차적으로 0~16MB 영역(ZONE_DMA), 16~896MB 영역(ZONE_NORMAL), 이후(ZONE_HIGHMEM)
으로 나뉘어지고, 가상주소 0xC00000000 부터 0xFFFFFFFF 까지가 물리주소 0 번지부터 매핑되기 때문에
커널영역의 가상주소로 물리메모리의 1g 만 address 할 수있다.

22.물리 메모리의 최소 단위를 무엇이라고 하며 크기가 얼마인가 ?
그리고 이러한 개념을 SW 적으로 구현한 구조체의 이름은 무엇인가 ?
물리메모리의 최소단위는 페이지 이고 크기는 4kb 이다. 이를 관리하는 구조체의 이름은 페이지프레임이다.

23. Linux Kernel 은 외부 단편화와 메모리 부하를 최소화하기 위해 Buddy 할당자를 사용한다.
Buddy 할당자의 Algorithm 을 자세히 기술하시오.
버디할당자는 zone 구조체에 존재하는 free_area 배열을 통해 구축된다 따라서 버디는 zone 당 하나식 존재하게 된다.
free_area 라는 배열의 각 엔트리는 free_area 라는 이름의 구조체이며 이 구조체는 free_list 와 nr_free 라는 필드를 갖는다.
free_area 배열은 10 개의 엔트리를 갖는다. 0~9 까지 각각의 숫자는 해당 free_area 가 관리하는 할다의 크기를 나타낸다. 예를들어 0 인 경우 2 의 0 승 즉, 1 개의 페이지 프레임이 할당의 단위임을 뜻하고, 1 인 경우 2 의 1 승 즉, 2 개의 페이지 프레임이 할당된다. 버디는 2 의 정수 승 개수의 페이지 프레임들을 할당해준다.

24. 21 번에 이어 내부 단편화를 최소화 하기 위해 Buddy 에서 Page 를 받아 Slab 할당자를 사용한다.

Slab 할당자는 어떤식으로 관리되는지 기술하시오.
버디할당자는 4kb 인데 이것보다 작은 용량이 할당 된다면 공간이 낭비될 것이다. 이 때문에
이 공간을 64byte 크기로 분할한다.(64bx64b==4kb) 그리고 사용자가 64byte 의 공간을 요청하면 버디할당자로부터 받아오는 것이 아니라 미리 할당받아 관리하고있던 이 공간에서 떼어주는 것이다.

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)
task_struct 안의 mm_struct 가 rb 트리의 형태로 이들을 관리한다.

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

27. 프로그램을 실행한다고 하면 fork(), execve()의 콤보로 이어지게 된다.
이때 실제 gcc *.c 로 컴파일한 a.out 을 ./a.out 을 통해 실행한다고 가정한다.
실행을 한다고 하면 a.out File 의 Text 영역에 해당하는 부분을 읽어야 한다.
실제 Kernel 은 무엇을 읽고 이 영역들을 적절한 값으로 채워주는가 ?
컴파일한 실행파일은 바이너리코드로 변경된다.
kernel 은 바이너리코드를 읽고 적절한 값으로 채워준다.

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.

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

30. MMU(Memory Management Unit)의 주요 핵심 기능을 모두 적고 간략히 설명하시오.
가상메모리를 할당해주어 물리메모리공간의 부족함을 해소하게 해준다.
31. 하드디스크의 최소 단위를 무엇이라 부르고 그 크기는 얼마인가 ?
페이지라고 부르고 4kb 이다.
32. Character 디바이스 드라이버를 작성할 때 반드시 Wrapping 해야 하는 부분이 어디인가 ?
(Task 구조체에서 부터 연결된 부분까지를 쭉 이어서 작성하라)

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

34. task_struct 에서 super_block 이 하는 역할은 무엇인가 ?
super_block 객체는 현재 사용중인(마운트 된) 파일시스템 당 하나씩 주어진다. 실제로 각 파일시스템은 자신이
관리하고있는 파티션에 파일시스템 마다 고유한 정보를 수퍼블록에 저장해 둔다. VFS 는 이를 읽어서 관리하기 위해
범용적인 구조체인 수퍼블록 객체를 정의한다.

35. VFS(Virtual File System)이 동작하는 Mechanism 에 대해 서술하시오.
파일은 디스크의 추상화인데 디스크에 파일이 있어서 파일은 메모리에 올라가야한다. 페이지프레임 단위로 관리하다보면 단편화가 발생하게되는데 그것을 효율적으로 관리하게 하기위해서 메커니즘이 필요한데 그 작업을 가상

파일시스템이 커널과 함께 서포트하는것이다. 가상파일시스템은 파일포맷을 여러개 처리해야할 때 도와준다.

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

37. 내부 인터럽트는 다시 크게 3 분류로 나눌 수 있다.
3 가지를 분류하시오.

Segmentation fault, System Call, Page fault

38. 35 번에서 분류한 녀석들의 특징에 대해 기술하시오.

39. 예로 모니터 3 개를 쓰는 경우 양쪽에 모두 인터럽트를 공유해야 한다.
Linux Kernel 에서는 어떠한 방법을 통해 이들을 공유하는가 ?

signal 을 통해서 전부다 처리해준다.

40. System Call 호출시 Kernel 에서 실제 System Call 을 처리하기 위해
Indexing 을 수행하여 적절한 함수가 호출되도록 주소값을 저장해놓고 있다.
이 구조체의 이름을 적으시오.

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

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

path

43. 또한 Page Directory 를 가르키는 Intel 전용 Register 가 존재한다.
이 Register 의 이름을 적고 ARM 에서 이 역할을 하는 레지스터의 이름을 적으시오.

44. 커널 내부에서 메모리 할당이 필요한 이유는 무엇인가 ?

커널도 컴퓨터 프로그램이기 때문에 메모리를 할당 받아서 내부인터럽트를 처리해줘야한다.
45. 메모리를 불연속적으로 할당하는 기법은 무엇인가 ?
예를들어 여러 디스크블록 번호가 있는데 1,2,3,4 와 같이 연속적으로 할당된 방법이 연속할당방법이다.
파일에게 불연속된 디스크 블록을 할당하는 방법으로 읽는 속도가 연속적으로 할당하는 방법보다 느리다.
46. 메모리를 연속적으로 할당하는 기법은 무엇인가 ?
예를들어 여러 디스크블록 번호 1.2.3.4.5.6.7 이 있는데 1, 3, 5, 7 와 같이 불연속적으로 할당하는 기법이다.
이 기법은 다음 디스크블록 번호를 가리키고 있어 불연속적이고 여기저기 왔다갔다 해서 느리지만 메모리 낭비를 줄일 수 있다.
47. Mutex 와 Semaphore 의 차이점을 기술하시오.
mutex 는 예를들면 화장실에서 안에 사람이 있으면 그 안에 사람이 나올때까지 계속 두들긴다.
Semaphore 는 화장실에서 사람이 나오지 않으면 다른화장실을 찾아서 헤맨다.
48. module_init() 함수 호출은 언제 이루어지는가 ?
디바이스 드라이버가 장착될 때 이루어진다.(초기화)
49. module_exit() 함수 호출은 언제 이루어지는가 ?
디바이스 드라이버가 탈착될 때 이루어진다. (해제)
50. thread_union 에 대해 기술하시오.
쓰레드를 관리하게 되는 커널 공용체
51. Device Driver 는 Major Number 와 Minor Number 를 통해 Device 를 관리한다.
실제 Device 의 Major Number 와 Minor Number 를 저장하는 변수는 어떤 구조체의 어떤 변수인가 ?
(역시 Task 구조체에서부터 쭉 찾아오길 바람)

52. 예로 간단한 Character Device Driver 를 작성했다고 가정해본다.
그리고 insmod 를 통해 Driver 를 Kernel 내에 삽입했으며

mknod 를 이용하여 /dev/장치파일을 생성하였다.
그리고 이에 적절한 User 프로그램을 동작시켰다.
이 Driver 가 실제 Kernel 에서 어떻게 관리되고 사용되는지 내부 Mechanism 을 기술하시오.

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 Mechanism 이 필요한 이유가 무엇인지 자세히 기술하라.

54. Character Device Driver 를 아래와 같이 동작하게 만드시오.
read(fd, buf, 10)을 동작시킬 경우 1 ~ 10 까지의 덧셈을 반환하도록 한다.
write(fd, buf, 5)를 동작시킬 경우 1 ~ 5 곱셈을 반환하도록 한다.
close(fd)를 수행하면 Kernel 내에서 "Finalize Device Driver"가 출력되게 하라!

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

56. Compiler 의 Instruction Scheduling 에 대해 기술하라.

57. CISC Architecture 와 RISC Architecture 에 대한 차이점을 기술하라.
RISC 는 "Reduced Instruction Set Computer"의 약자로 CISC 구조 CPU 가 지니는 명령어 중에서 주로사용하는 명령어가 10%정도밖에 되지 않는데서 착안한것. 명령어의 수를 대폭 줄이고 명령어의 길이를 일정하게 디자인해서 만든것이 RISC 구조 근래의 ARM 이 RISC 구조이다.

CISC 는 "Complex Instruction Sec Computer"의 약자로 복잡한 명령어 체계를 가지는 컴퓨터이다. 우리가 사용하는 인텔의 16 비트 CPU 까지가 CISC 구조였다.
이러한 구조는 명령어수가 많고 그 크기가 일정치 않기 때문에 복잡하고 성능향상에 제한이 된다.

64 비트와 32 비트
58. Compiler 의 Instruction Scheduling 은 Run-Time 이 아닌 Compile-Time 에 결정된다.
고로 이를 Static Instruction Scheduling 이라 할 수 있다.
Intel 계열의 Machine 에서는 Compiler 의 힘을 빌리지 않고도
어느저도의 Instruction Scheduling 을 HW 의 힘만으로 수행할 수 있다.
이러한 것을 무엇이라 부르는가 ?

59. Pipeline 이 깨지는 경우에 대해 자세히 기술하시오.
점프연산을 하게되면 파이프라인이 깨제기된다.
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 ?

61. Intel 의 Hyper Threading 기술에 대해 상세히 기술하시오.
core 가 4 개인 cpu 를 fork 를 활용해서 core 를 8 개의 성능으로 바꾸어 주는 엄청난 기술,
칩의 크기가 작고 성능은 우수해지면서 core 가 8 개인 것보다 저렴하다.
62. 그동안 많은 것을 배웠을 것이다.
최종적으로 Kernel Map 을 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.

63. 파일의 종류를 확인하는 프로그램을 작성하시도록 하시오.

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<unistd.h>
#include<stdio.h>

int main(int argc,char* argv[]){
        struct stat buf;
        char ch;
        stat(argv[1], &buf);

        if(S_ISDIR(buf.st_mode))
                perm[0] = 'd';
        if(S_ISREG(buf.st_mode))
                perm[0] = '-';
        if(S_ISFIFO(buf.st_mode))
                perm[0] = 'p';
        if(S_ISLNK(buf.st_mode))
                perm[0] = 'l';
        if(S_ISSOCK(buf.st_mode))
                perm[0] = 's';
        if(S_ISCHR(buf.st_mode))
                perm[0] = 'c';
        if(S_ISBLK(buf.st_mode))
                perm[0] = 'b';
        printf("%c\n",ch);
        return 0;
}
```

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

65. Shared Memory 를 통해 임의의 파일을 읽고 그 내용을 공유하도록 프로그래밍하시오.

```
//메모리 받기
#include "shm.h"

int main(void)
{
	int mid;
	SHM_t *p;

	mid = CreateSHM(0x888);

	p = GetPtrSHM(mid);

	getchar();
	printf("이름 : [%s], 점수 : [%d]\n", p->name,p->score);

	FreePtrSHM(p);기
	return 0;

}
//메모리 보내기
#include "shm.h"

int main(void)
{
	int mid;//메모리아이디 Memory I D mid
	SHM_t *p;//SHM_t 의 포인터형.

	mid = OpenSHM(0x888);//페이지프레임에 접근할 아이디값을
```

```
	p = GetPtrSHM(mid);//p 가 가리키는 곳의 주소에

	getchar();
	strcpy(p->name,"아무개");
	p->score = 93;

	FreePtrSHM(p);

	return 0;
}
//메모리공유 헤더파일
#include<sys/ipc.h>
#include<sys/shm.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<errno.h>
//메모리공유 c 라이브러리파일
#include "shm.h"

int CreateSHM(long key)
{
	return shmget(key, sizeof(SHM_t), IPC_CREAT|0777);
}
int OpenSHM(long key)
{
	return shmget(key, sizeof(SHM_t), 0);
}
SHM_t* GetPtrSHM(int shmid)
{
	return (SHM_t*)shmat(shmid,(char*)0,0);
}
int FreePtrSHM(SHM_t* shmptr)
```

```
{
	return shmdt((char*)shmptr);
}

typedef struct
{
	char name[20];
	int score;
}SHM_t;

int CreateSHM(long key);
int OpenSHM(long key);
SHM_t* GetPtrSHM(int shmid);
int FreePtrSHM(SHM_t* shmptr);
```

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.

uname -r 을 쉘프롬프트창에 치고 자신의 커널버전을 확인한 다음에
sudo apt-get https://mirrors.edge.kernel.org/pub/linux/kernel/v4.x/4.13.0-38-generic.tar.gz
을 쳐서 다운받고
다운받은 파일을 tar 를 이용해 압축풀고 vi -t task_struct 를 치고 144 번을 들어가면 task_struct 를
들어갈 수 있다.

67. Multi-Tasking 의 원리에 대해 서술하시오.
(Run Queue, Wait Queue, CPU 에 기초하여 서술하시오)
컴퓨터 프로그램이 실행되면 씨피유는 이것을 받아서 처리하게 되는데

씨피유는 하나의 프로세스 말고는 처리하지 못한다. 그래서 하나씩 처리하게되는데 처리하는 동안
기다리는 프로세스가 Wait Queue 이고 처리하는 프로세스가 Run Queue 인데 실제로 CPU 는 하나의 프로세스밖에 처리하지 못하지만 속도가 매우 빨라서 한번에 여러개의 프로세스를 처리하는것과 같은 효과를 보게되는것이다.

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

69. System Call Mechanism 에 대해 기술하시오.
System Call Mechanism 은 유일한 소프트웨어 인터럽트이다.
유저영역에서 커널에게 요청하면 커널이 받아서 처리해주는 인터럽트이다.
70. Process 와 VM 과의 관계에 대해 기술하시오.
프로세스는 mm_struct 라는 구조체로 관리되는데 이 구조체는
가상메모리를 할당받아서 생성된다. 만약 가상메모리가 없다면 프로세스도 관리되지 못할것이다.

71. 인자로 파일을 입력 받아 해당 파일의 앞 부분 5 줄을 출력하고
추가적으로 뒷 부분의 5 줄을 출력하는 프로그램을 작성하시오.

72. 디렉토리 내에 들어 있는 모든 File 들을 출력하는 Program 을 작성하시오.

```
#include<stdio.h>
#include<fcntl.h>
#include<sys/types.h>
#include<dirent.h>
#include<unistd.h>
#include<sys/stat.h>
#include<string.h>

void recursive_dir(char* dname);
```

```
int main(int argc, char* argv[])
{
	recursive_dir(".");
	return 0;
}

void recursive_dir(char* dname)
{
	struct dirent* p;
	struct stat buf;
	DIR* dp;
	chdir(dname);
	dp = opendir(".");
	printf("t%s:\n",dname);
	while(p = readdir(dp))
		printf("%s\n",p->d_name);
	rewinddir(dp);
	while(p = readdir(dp))
	{
		stat(p->d_name,&buf);
		if(S_ISDIR(buf.st_mode))
			if(strcmp(p->d_name,".")&&		strcmp(p->d_name,".."))
				recursive_dir(p->d_name);
	}
	chdir("..");
	closedir(dp);
}
```

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

74. C.O.W Architecture 에 대해 기술하시오.

75. Blocking 연산과 Non-Blocking 연산의 차이점에 대해 기술하시오.
Blocking vs Non Blocking 차이?
cpu 자원을 잡고있느냐 마느냐에 대한 차이다
블로킹을 하면 특정 조건을 만족할때까지 리소스 활용불가
논블로킹은 상관없이 준비되면 실행 아니면 대기
블로킹은 뭔가 할려고 하면 막는다. cpu 를 잡고 놔주지않는다.
내 작업이 완료되지않고는 놔주지않음
전화통화 , 친구가 안받으면 상대방은 못듣는다(블로킹)
카톡은 보내놓으면 상대방이 보면 보고 안보면 안보는거(논블로킹)
효율성은 논블로킹이 좋다.
블로킹은 반드시 완료가되야 다음작업이 이루어진다

순서가 중요한 프로그램에서는 블로킹을 쓴다

76. 자식이 정상 종료되었는지 비정상 종료되었는지 파악하는 프로그램을 작성하시오.

```
#include<stdio.h>
int main(){
	int pid,i;
	if(pid = fork() > 0){
		if(wait())
			printf("자식정상종료\n");
		else
			printf("자식비정상종료\n");

	}
	else if(pid ==0){
		while(i < 5){
			sleep(1);
			system("date");
			i++;
```

```
            }
        }
        printf("after\n");
        return 0;
}
```

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

```
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<signal.h>
#include<unistd.h>
#include<stdlib.h>

int daemon_init(void)
{
        int i;
        if(fork() > 0)
                exit(0);
        setsid();
        chdir("/");
        umask(0);
        for(i = 0; i < 64; i++)
                close(i);
        signal(SIGCHLD, SIG_IGN);
        return 0;
}

int main(void)
{
        daemon_init();
        while(1){
        printf("ns\n");
```

```
        }
        return 0;
}
```

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

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.

goto 는 함수간 이동이 불가능하다 왜냐하면 스택을 해제할 수 없기 때문이다.
함수간 이동이 불가능한 점을 setjump 와 longjump 로 이동이 가능해진다.

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <setjmp.h>

jmp_buf env1;
jmp_buf env2;

void test1(void)
{
        longjmp(env1, 1);
}

void test2(void)
{
        longjmp(env1, 2);
}

void test3(void)
{
```

```
        longjmp(env2, 1);
}

int main(void)
{
        int ret;
        if((ret = setjmp(env1)) == 0)
        {
                printf("this\n");
                test1();
        }
        else if(ret == 1)
        {
                printf("1\n");
                test2();
        }
        else if(ret == 2)
                printf("2\n");
        else
        {
                printf("goto letsgo label\n");
                goto letsgo;
        }

        if((ret = setjmp(env2)) == 0)
        {
                printf("second label\n");
                test3();
        }
        else
                longjmp(env1, 3);
letsgo:
                goto err;

        return 0;
```

```
err:
        printf("Error!!!\n");
        exit(-1);
}
```

80. 리눅스에서 말하는 File Descriptor(fd)란 무엇인가 ?
파일디스크립터란, 파일의 약속이다 예를들면 0 번은 표준입력인 콘솔에 키보드를 입력하는 약속이고,
1 번은 모니터에 출력하게되는 디스크립터 약속이다. 2 번은 표준에러에 대한 약속이다. 그리고 이 후 부터는 디스크립터번호를 매겨주면 그것이 해당 디스크립터 번호가 된다 예를 들면 다른 컴퓨터와의 통신을 할 때 요청을 해온 소켓파일디스크립터 번호에 write 를 하게되면 다른컴퓨터의 있는 파일에 작성이 가능해진다.

81. stat(argv[2], &buf) 일 때 stat System Call 을 통 해 채 운 buf.st_mode 의 값에 대해 기술하시오.

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

83. 디렉토리를 만드는 명령어는 mkdir 명령어다.
man -s2 mkdir 을 활용하여 mkdir System Call 을 볼 수 있다.
이를 참고하여 디렉토리를 만드는 프로그램을 작성해보자!

84. 이번에는 랜덤한 이름(길이도 랜덤)을 가지도록 디렉토리를 3 개 만들어보자!
(너무 길면 힘드니까 적당한 크기로 잡도록함)

85. 랜덤한 이름을 가지도록 디렉토리 3 개를 만들고

각각의 디렉토리에 5 ~ 10 개 사이의 랜덤한 이름(길이도 랜덤)을 가지도록 파일을 만들어보자!
(너무 길면 힘드니까 적당한 크기로 잡도록함)

86. 85 번까지 진행된 상태에서 모든 디렉토리를 순회하며
3 개의 디렉토리와 그 안의 모든 파일들의 이름 중 a, b, c 가 1 개라도 들어있다면 이들을 출력하라!
출력할 때 디렉토리인지 파일인지 여부를 판별하도록 하시오.

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

88. 반 인원이 모두 참여할 수 있는 채팅 프로그램을 구현하시오.

```
//채팅 클라이언트
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>
#include<pthread.h>
#include<arpa/inet.h>
#include<sys/socket.h>
#include<sys/epoll.h>

#define BUF_SIZE        200000
#define NAME_SIZE    32

typedef struct sockaddr_in si;
typedef struct sockaddr * sap;

char name[NAME_SIZE] = "[DEFAULT]";
char msg[BUF_SIZE];

void err_handler(char* msg)
```

```
{
	fputs(msg, stderr);
	fputc('\n',stderr);
	exit(1);
}

void * send_msg(void* arg)
{
	int sock = *((int*)arg);
	char name_msg[NAME_SIZE + BUF_SIZE];
	int i;
	for(;;)
	{
		//fgets(msg, BUF_SIZE, stdin);

		if(!strcmp(msg, "q\n") || !strcmp(msg, "Q\n"));
		{
			close(sock);
			exit(0);
		}
		for(i = 0; i < 1024; i++){
			msg[i] = '!';
		}
		write(sock, msg, strlen(msg));
		sprintf(name_msg, "%s %s", name, msg);
		write(sock, name_msg, strlen(name_msg));
	}

	return NULL;
}
void* recv_msg(void* arg){
	int sock = *((int*)arg);
	char name_msg[NAME_SIZE + BUF_SIZE];
	int str_len;
```

```
	for(;;){
		str_len = read(sock, name_msg, NAME_SIZE + BUF_SIZE - 1);

		if(str_len==-1)
			return (void*)-1;

		name_msg[str_len] = 0;
		fputs(name_msg, stdout);
	}
	return NULL;
}

int main(int argc, char** argv)
{
	int sock;
	si serv_addr;
	pthread_t snd_thread, rcv_thread;
	void* thread_ret;

	if(argc != 4){
		printf("Usage: %s <IP> <port> <name>\n",argv[0]);
		exit(1);
	}
	sprintf(name,"[%s]", argv[3]);
	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, (sap)&serv_addr, sizeof(serv_addr)) == -1)
		err_handler("connect() error");

	pthread_create(&snd_thread, NULL, send_msg, (void*)&sock);
	pthread_create(&snd_thread, NULL, recv_msg, (void*)&sock);
	pthread_join(snd_thread, &thread_ret);
	pthread_join(rcv_thread, &thread_ret);

	close(sock);
	return 0;
}

//채팅서버
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<fcntl.h>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<sys/epoll.h>
#include<pthread.h>
#include<unistd.h>

#define BUF_SIZE 128
#define MAX_CLNT  256

typedef struct sockaddr_in  si;
typedef struct sockaddr *  sp;

int clnt_cnt = 0;
int clnt_socks[MAX_CLNT];
pthread_mutex_t mtx;
```

```
void err_handler(char* msg)
{
	fputs(msg, stderr);
	fputc('\n',stderr);
	exit(1);
}

void send_msg(char* msg, int len)
{
	int i;
	pthread_mutex_lock(&mtx);

	for(i = 0; i < clnt_cnt; i++)
		write(clnt_socks[i], msg, len);

	pthread_mutex_unlock(&mtx);
}

void *clnt_handler(void* arg)
{
	int clnt_sock = *((int*)arg);
	int str_len =0, i;
	char msg[BUF_SIZE];

	while((str_len = read(clnt_sock, msg, sizeof(msg))) != 0)
		send_msg(msg, str_len);

	pthread_mutex_lock(&mtx);

	for(i = 0; i < clnt_cnt; i++)
	{
		if(clnt_sock == clnt_socks[i])
		{
			while(i++ < clnt_cnt -1)
				clnt_socks[i] = clnt_socks[i + 1];
```

```
                break;
            }
        }

        clnt_cnt--;
        pthread_mutex_unlock(&mtx);
        close(clnt_sock);

        return NULL;

}

int main(int argc, char** argv)
{
        int serv_sock, clnt_sock;
        si serv_addr, clnt_addr;
        socklen_t addr_size;
        pthread_t t_id;

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

        pthread_mutex_init(&mtx, NULL);

        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,10) == -1)
		err_handler("listen() error");

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

		pthread_mutex_lock(&mtx);
		clnt_socks[clnt_cnt++] = clnt_sock;
		pthread_mutex_unlock(&mtx);

		pthread_create(&t_id, NULL, clnt_handler,
(void*)&clnt_sock);
		pthread_detach(t_id);
		printf("Connected Client IP: %s\n",
inet_ntoa(clnt_addr.sin_addr));
	}
	close(serv_sock);
	return 0;
}
```

89. 88 번 답에 도배를 방지 기능을 추가하시오.

90. 89 번 조차도 공격할 수 있는 프로그램을 작성하시오.

91. 네트워크 상에서 구조체를 전달할 수 있게 프로그래밍 하시오.

```
//구조체전달 클라이언트
```

```
#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;
            buf->fdata = 7.7;

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

int main(int argc, char **argv)
{
        pid_t pid;
        int i, sock;
        si 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;
}


//구조체전달 서버
#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(;;)
	{
		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)
			{
				printf("struct.data = %d, struct.fdata = %f\n",
struct_data.data, struct_data.fdata);
				write(clnt_sock, (d *)&struct_data, len);
```

```
                }

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

        return 0;
}
//구조체전달서버클라이언트 헤더파일
#ifndef __COMMON_H__
#define __COMMON_H__

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <arpa/inet.h>
#include <sys/socket.h>

typedef struct sockaddr_in    si;
typedef struct sockaddr *     sp;

typedef struct __d{
        int data;
        float fdata;
} d;

#define BUF_SIZE            32

#endif
```

92. 91 번을 응용하여 Queue 와 Network 프로그래밍을 연동하시오.

93. Critical Section 이 무엇인지 기술하시오.
여러 Task 들이 동시에 접근해서 겹치게 되는 공간인데 이 때문에 나온 메커니즘이 semaphore 메카니즘과 spinrock 메카니즘이다.
이것을 해결하지 않게되면 데이터가 꼬이게 된다.
94. 유저에서 fork() 를 수행할때 벌어지는 일들 전부를
실제 소스 코드 차원에서 해석하도록 하시오.

95. 리눅스 커널의 arch 디렉토리에 대해서 설명하시오.
리눅스 커널 기능 중 하드웨어 종속적인 부분들이 구현된 디렉터리이다. 그래서 이름도
architecture 를 의미하는 arch 이다. 이 디렉터리는 CPU 의 타입에 따라 하위 디렉터리로 다시 구분된다.
96. 95 번 문제에서 arm 디렉토리 내부에 대해 설명하도록 하시오.
arch 디렉터리는 CPU 의 타입에 따라 하위 디렉터리로 다시 구분된다.
대표적으로 i386,ARM(Advanced RISC Machine), 모토로라의 68 시리 즈 ,SUN 의 Sparc, IBM 의 PPC(Power PC) 등 이 있 다 . arch/arm/boot 디렉토리에는 시스템의 부팅 시 사용하는 부트스트랩 코드가 구현되어 있다. arch/arm/kernel 에는 태스크 관리자 중에서 문맥 교환이나 쓰레드 관리 같은 기능, arch/arm/mm 에는 메모리 관리자 중에서 페이지 부재 결함 처리 같은 기능 등의 하드웨어 종속적인 부분이 구현되어 있다. arch/arm/lib 에는 커널이 사용하는 라이브러리 함수가 구현되어 있으며, arch/arm/math-emu 에는 FPU(Floating Point Unit)에 대한 에뮬레이터가 구현되어 있다.
97. 리눅스 커널 arch 디렉토리에서 c6x 가 무엇인지 기술하시오.
ti 사의 DSP 관련 프로그램이다.
98. Intel 아키텍처에서 실제 HW 인터럽트를
어떤 함수를 가지고 처리하게 되는지 코드와 함께 설명하시오.

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

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.

그 날 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.

과정진행중에 열심히 안하지도, 그렇다고 죽도록 열심히 한 것은 아니지만 현재의 제 상태를 봐서는
죽도록 해야 가능할까 말까 한 것 같습니다. 처음에 학원에 들어오기 전 죽도록 열심히 하자는 처음의
각오는 어디에 갔는지 안 보이고 나태해진 것 같은 느낌도 듭니다. 그래도 2 개월차시험이 없었으면
이러한 저의 나태한 마음을 더욱 더 자신에게 조금 더 쉬엄쉬엄 해도 돼 라고 말을하고 더 나태해
졌을 것이라고 생각합니다. 2 개월차 시험을 계기로 내가 이거밖에 안되는건가 하는 생각도 들긴 하지만
그렇다고 더 안해버리면 여기서 계속 저는 이거밖에 안되는거라
생각되니 더 이악물고 열심히 해야될 것 같습니다.
그렇지 않아도 전공자도 아니어서 더 열심히 해도 모자랄 판에,, 앞으로 제가 할 수 있는 최선은

다 해보려고 합니다. 노력은 배신하지 않는다고 하니까요 앞으로 파이팅 해 보겠습니다!!