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

-

2. 369 게임을 작성하시오.
2 초내에 값을 입력하게 하시오.
박수를 쳐야 하는 경우를 Ctrl + C 를 누르도록 한다.
2 초내에 값을 입력하지 못할 경우 게임이 오버되게 한다.
Ctrl + C 를 누르면 "Clap!"이라는 문자열이 출력되게 한다.
-

tewill@tewill-Z20NH-AS51B5U: ~/Desktop/test2

```c
#include <stdio.h>
#include <time.h>
#include <signal.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>

int count;

void game_over(int nosig)
{
    printf("game over\n");
    exit(1);
}

void clap(int nosig)
{
    printf("Clap!\n");
    if(count > 0)
        count--;
    else
        game_over(0);
}

void game_start()
{
    char buf[1024];
    int max = 0, num;

    signal(SIGALRM, game_over);
    signal(SIGINT, clap);
    printf("GAME START\n\n");
    for(;;){
        max ++;
        count = 0;

        if( max > 100 && ((max % 1000) / 100) % 3 == 0)
            count++;
        if( max > 10  && ((max % 100) / 10) % 3 == 0)
            count++;
        if((max % 10)% 3 == 0)
            count++;

        if(count > 0){
            alarm(2);
            sleep(2);
            if(count == 0)
                alarm(0);
        } else {
            alarm(2);
            read(0, buf, 1024);
            alarm(0);
            num = atoi(buf);
            if(num == max)
                continue;
            else
                game_over(0);
        }
    }
```

1,1 Top

tewill@tewill-Z20NH-AS51B5U: ~/Desktop/test2

```c

int main(void)
{
    game_start();
    return 0;
}
```

66,1 Bot

3.리눅스 커널은 운영체제(OS)다.
OS 가 관리해야 하는 제일 중요한 5 가지에 대해 기술하시오.
- filesystem manager / task manager / memory manager / device manager / network manager

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

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

*** 사용자 임의대로 재구성이 가능하다.**
- 리눅스는 운영체제 자체가 오픈되어 있다.

*** 열약한 환경에서도 HW 자원을 적절히 활용하여 동작한다.**
- **paging** 과 **context switching** 을 통하여 열약한 환경에서도 **HW** 자원을 적절히 활용하여 동작한다.

*** 커널의 크기가 작다.**
- 하이브리드 커널이며, 커널은 마이크로 커널을 기반으로 하여 작고, 추가로 모놀리식 커널 역시 갖고있어 추가 기능을 사용할 수 있다. 과거에는 커널의 크기가 작았지만 현재 그 이점은 줄어들었다.

*** 완벽한 멀티유저, 멀티태스킹 시스템**
- **paging** 과 **context switching** 을 통하여 완벽한 멀티태스킹 시스템을 지원해 준다.

*** 뛰어난 안정성**
- 오랜시간 사용되어왔으며 검증되어 있다. 네트워크와 기상 기지국 등 에서 리눅스를 사용하는것을 예로 볼 수 있다.

*** 빠른 업그레이드**
- **GNU** 제단을 통해 빠르게 업그레이드 되어왔다.

*** 강력한 네트워크 지원**
- **tcp/ip** 가 리눅스를 기반으로 만들었기 때문에 윈도우 등에 비하여 더 최적화 되어있다.

*** 풍부한 소프트웨어**
- **GNU** 제단을 통하여 많은 소프트웨어가 오픈되어 있다.

6. 32bit System 에서 User 와 Kernel 의 Space 구간을 적으시오.
- **32bit** 기준으로, 커널과 유저의 영역은 **1 : 3** 비율로 이루어져 있다.
즉, **0xC0000000** 을 기준으로 하여 커널과 유저 영역이 구분된다.

7. Page Fault 가 발생했을때 운영체제가 어떻게 동작하는지 기술하시오.
- **Page Fault** 발생 시, 물리 메모리에 비어있는 페이지 프레임이 있는지 확인한다.
없을 경우, 사용 빈도가 떨어지는 페이지 프레임을 비우는 작업이 추가되며, 있을 경우 해당 페이지 프레임에 해당 페이지를 로드하고, 페이지 테이블을 최신화 한다. 그 후 인터럽트로 중단되어 있던 작업을 다시 시작한다.

8. 리눅스 실행 파일 포맷이 무엇인지 적으시오.
- **ELF**

9. 프로세스와 스레드를 구별하는 방법에 대해 기술하시오.
- **pid** 와 **tpid** 가 동일하다면 프로세스이고, 다르다면 스레드 이다.

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.
커널 소스 코드와 함께 기술하시오.
- 즉, 자기 자신의 **task** 를 가르킨다.

tewill@tewill-Z20NH-AS51B5U: ~/my_proj/kernel0/linux-4.4

```
 1 #ifndef _M68K_CURRENT_H
 2 #define _M68K_CURRENT_H
 3
 4 #ifdef CONFIG_MMU
 5
 6 register struct task_struct *current __asm__("%a2");
 7
 8 #else
 9
10 /*
11  *      Rather than dedicate a register (as the m68k source does), we
12  *      just keep a global,  we should probably just change it all to be
13  *      current and lose _current_task.
14  */
15 #include <linux/thread_info.h>
16
17 struct task_struct;
18
19 static inline struct task_struct *get_current(void)
20 {
21         return(current_thread_info()->task);
22 }
23
24 #define current get_current()
25
26 #endif /* CONFNIG_MMU */
27
28 #endif /* !(_M68K_CURRENT_H) */
~
~
~
~
~
~
~
~
~
~
"arch/m68k/include/asm/current.h" 28L, 541C                   19,1          All
```

12. Memory Management 입장에서 Process 와 Thread 의 핵심적인 차이점은 무엇인가 ?
- **process** 의 경우 서로 다른 주소 공간을 갖게 되지만, **thread** 의 경우 서로 같은 주소 공간을 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 ?
- **context swiching** 을 할 때 **task** 의 현재 실행 위치에 대한 정보를 저장한다.

14. 리눅스 커널의 스케쥴링 정책중 Deadline 방식에 대해 기술하시오.
- 프로세스들에게 **deadline** 을 주어 동작하도록 한다. 이는 우선순위에 따라 값을 다르게 주어, 현실 시간에 따라서 가상의 시간이 일정하도록 한다. 이때 처리에 최소 시간을 계산하여, 남는 시간에 다른 프로세스의 동작을 처리하도록 하는데 이를 **deadline** 방식 이라 한다.

15. TASK_INTURRUPTIBLE 과 TASK_UNINTERRUPTIBLE 은 왜 필요한지 기술하시오.
- **task** 를 관리하기 위해서 필요하다. 프로세스의 상태를 설정하는 것으로서, 시그널에 대한 처리에 따라 나눠진다. 시그널 호출 시 깨어나는 것과 특정 시그널에 따라 깨어나는 것으로 나뉘어 있고, 이는 유저가 작성한 프로세스에 따라 적절히 설정하도록 한다.

16. O(N)과 O(1) Algorithm 에 대해 기술하시오.
그리고 무엇이 어떤 경우에 더 좋은지 기술하시오.
- 상황에 따라서 다르다. **Task** 숫자가 적을경우 **0(n)**이 더 좋을 수도 있지만, 많아질수록 **cpu** 가 처리해야 할 명령어의 분량이 증가하게 된다. 그에 반해 **0(1)** 의 경우 항상 일정하게 **cpu** 명령어의 분량이 정해져 있다. 이는 **task** 가 적을 경우 단점이 될 수도 있지만, 일반적인 환경(리눅스)에서는 기본적으로 많은 수의 **task** 가 동작하기 때문에 **0(n)** 보다 더 좋다.

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.
- 2 번 **CPU** 가 더 좋다. 복사를 처리하는 과정에서 다른 곳으로 옮기고 복사하는것 보다 해당 **cpu** 에서 바로 처리하는 것이 빠르기 때문이다.

18. 15 번 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.
- 2 번 CPU 가 더 좋다. 다른 3 개의 CPU 가 작업할 것들이 밀려있기 때문에 비어있는 2 번 CPU 에서 처리하는것이 더 빠르게 처리할 수 있기 때문이다.

19. UMA 와 NUMA 에 대해 기술하고 Kernel 에서 이들을 어떠한 방식으로 관리하는지 기술하시오.
커널 내부의 소스 코드와 함께 기술하도록 하시오.
- UMA 는 하나의 메모리만을 관리하지만, 이로인해 병목현상이 발생하게 된다. 이를 해소하고자 NUMA 가 만들어 졌는데, 복수의 CPU 를 사용할 때 메모리를 나누어 CPU 마다 별도의 메모리를 주는 방식이다.

20. Kernel 의 Scheduling Mechanism 에서 Static Priority 와 Dynamic Priority 번호가 어떻게 되는지 적으시오.
- Static Priority 는 0 에서 99 번 까지이고, Dynamic Priority 는 100 139 번 까지이다.

21. ZONE_HIGHMEM 에 대해 아는대로 기술하시오.
- 가상 주소와 물리 메모리를 1:1 로 맵핑할 경우 1GB 이상 사용이 불가능 하기 때문에, 1GB 이상일 경우 896MB 이상의 메모리 영역을 간접 참조를 통하여 만든다. 단 이건 과거에 주로 사용되었던 것이고, 현재는 대부분 ZONE_HIGHMEM 없이 사용되고 있다.

22. 물리 메모리의 최소 단위를 무엇이라고 하며 크기가 얼마인가 ?
그리고 이러한 개념을 SW 적으로 구현한 구조체의 이름은 무엇인가 ?
- 4KB, follow_pte

23. Linux Kernel 은 외부 단편화와 메모리 부하를 최소화하기 위해 Buddy 할당자를 사용한다.
Buddy 할당자의 Algorithm 을 자세히 기술하시오.
- free_area[] 배열을 통해 관리된다. 총 10 개의 엔트리를 갖고 있으며, 이는 각각 2 의 승수를 나타낸다.
페이지를 사용여부에 따라 0 과 1 로 관리를 하며, 요구하는 page 의 크기에 따라서 관리된다.
예를들어 2page 를 넣는다면 1 번 배열에서 1 을 찾고, 있을경우 해당 위치에 삽입 후 값을 0 으로 바꾸는 식이다.
없을 경우, 그 상위로 올가 찾아보고, 집어넣는 다음에 다시 free_area 의 값을 저장 상태에 맞게 정리한다.
뺄 경우는 반대로 동작하면 된다.

24. 21 번에 이어 내부 단편화를 최소화 하기 위해 Buddy 에서 Page 를 받아 Slab 할당자를 사용한다.
Slab 할당자는 어떤식으로 관리되는지 기술하시오.
- 가득 차 있는 stabs_full 과 비어있는 slabs_free, 분할되 있는 slabs_partial 에서 요구하는 공간에 따라 우선적으로 partial 을 탐색하고 들어갈 경우 저장한다. 해당 위치에 없을경우 free 공간에 저장을 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)
- task_struct 내부의 mm 이 가르키는 mm_struct 내부 mmap 에 의하여 기록된다.

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)
- task_struct 내부의 mm 이 가르키는 mm_struct 내부 mmap 이 가르키는 vm-area_struct 이다.

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 ?
- sys_execve() 호출을 통하여 free 한 페이지 프레임들을 할당받는다. 그 후 파일 시스템에게 수행하려는 파일의 일부 내용을 읽어달라고 요청한다. 그 후 읽어진 내용을 할당받은 페이지 프레임에 저장한다.

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.
- 커널 역시도 결국 프로그램이기 때문이다. 유저가 만든 프로세스가 stack, heap, data, text 를 사용하듯 커널 역시 메모리 기본 구성을 이용한다.

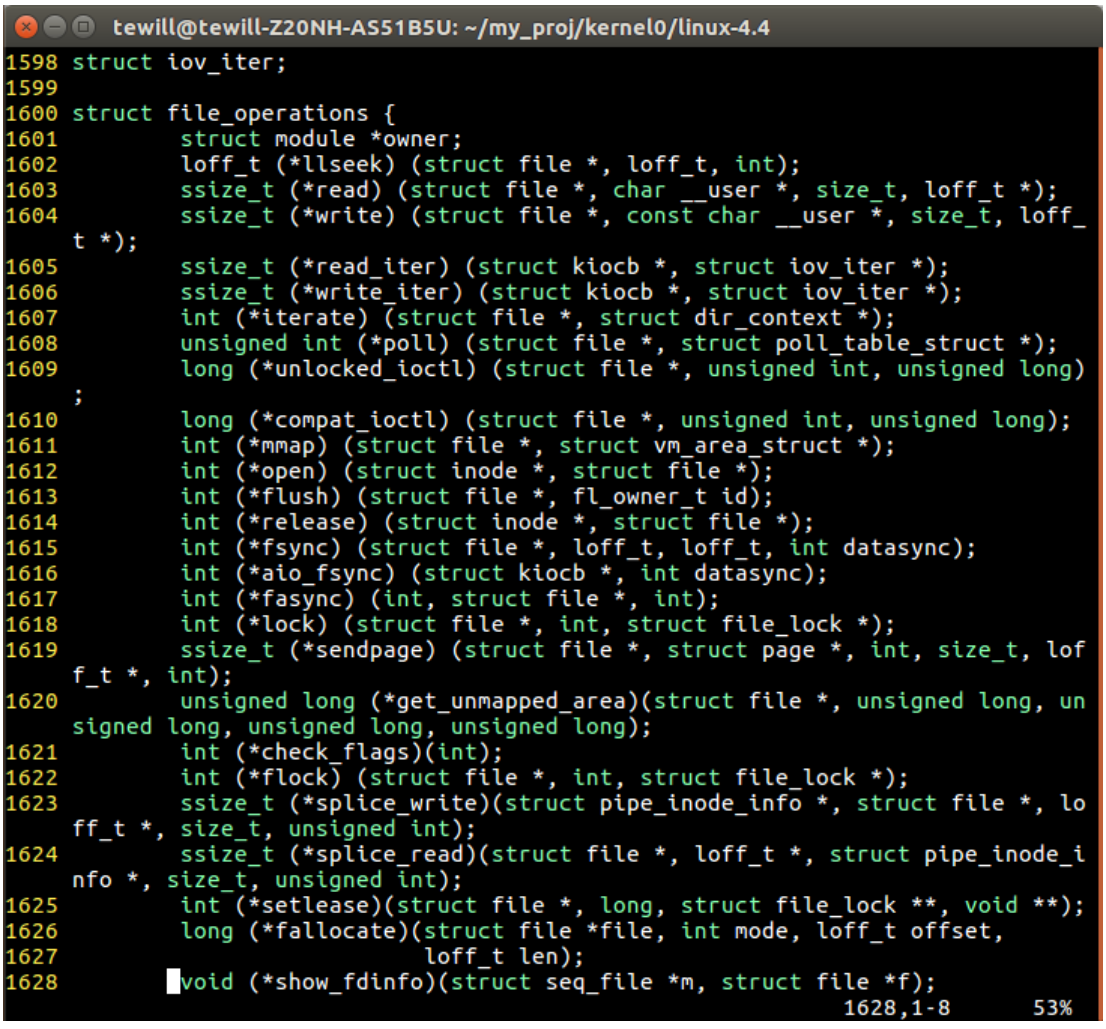
29. VM(가상 메모리)와 PM(물리 메모리)를 관리하는데 있어
VM 을 PM 으로 변환시키는 Paging Mechanism 에 대해 Kernel 에 기반하여 서술하시오.
- 페이징이란 페이지 프레임을 기반으로 데이더를 관리하는 기법이다. 가상 메모리 주소에 접근할 때, 페이지(4kb = 4096)로 나눈다. 그 후, 몫을 페이지 테이블의 엔트리를 탐색하는 인덱스로 사용하고, 나머지는 페이지 프레임 내에서 offset 으로 사용한다.

30. MMU(Memory Management Unit)의 주요 핵심 기능을 모두 적고 간략히 설명하시오.
- 주소 변환 과정이 필요하며, 메모리 접근시간에 대한 예측성이 떨어지는 점(캐시가 깨지는 것)을 보완해 준다. **TLB** 와 같은 캐시 메모리를 이용하여 주소변환을 따로 처리하도록 하여, 빠른 주소변환을 가능하게 한다.

31. 하드디스크의 최소 단위를 무엇이라 부르고 그 크기는 얼마인가 ?
- **sector** 라 불리우며 크기는 **512byte** 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.
- **file_operations** 을 통하여, 사용하고자 하는 시스템 콜을 동작시킨다.



34. task_struct 에서 super_block 이 하는 역할은 무엇인가 ?
- '/' 로서, 파일 시스템과 관련된 정보들을 관리한다. 즉 **inode** 들을 관리한다.

35. VFS(Virtual File System)이 동작하는 Mechanism 에 대해 서술하시오.
- 유저가 파일에 관련된 요청을 할 경우, **VFS** 는 유저가 요청하는 **file** 의 디렉토리를 확인하고, 사용자가 요청하는 파일의 디렉토리에 맞는 형식으로 해당 요청을 수행한다. 예를들어 다른 형식의 파일을 읽을 경우, 그것을 사용자가 원하는 형식으로 변환해 준다.

36. Linux Kernel 에서 Interrupt 를 크게 2 가지로 분류한다.
그 2 가지에 대해 각각 기술하고 간략히 설명하시오.
내부적 인터럽트와 외부적 인터럽트로 구분된다. 이중 외부적 인터렙트는 키보드 마우스 등 주변 장치에서 발생되는 비동기적인 하드웨어적인 것을 말한다. 내부적 인터럽트는 소프트웨어적인 것을 가르키며, **cpu** 내부에서 발생하는 것을 말한다. 대표적으로 세그멘테이션 폴트, 페이지 폴트, 시스템 콜 등이 있다.

37. 내부 인터럽트는 다시 크게 3 분류로 나눌 수 있다.
3 가지를 분류하시오.
- fault / trap / abort

38. 35 번에서 분류한 녀석들의 특징에 대해 기술하시오.
- fault : fault 를 일으킨 명령어 주소를 **eip** 에 넣어 두었다가 해당 핸들러가 종료되가 나면 **eip** 가 저장된 주소부터 다시 시작한다.
Trap : trap 을 일으킨 명령어를 **eip** 에 넣어두고 그 다음부터 다시 수행한다.
Abort : 심각한 에러로서 현재 테스크를 강제 종료한다.

39. 예로 모니터 3 개를 쓰는 경우 양쪽에 모두 인터럽트를 공유해야 한다.
Linux Kernel 에서는 어떠한 방법을 통해 이들을 공유하는가 ?
- **irq_table** 을 통해 **32~255** 까지 관리되며, **do_IRQ** 를 통하여 **irq_desc_t** 자료구조로 관리된다.

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.
- __vector_start

41. 38 에서 User Space 에서 System Call 번호를 전달한다.
Intel Machine 에서는 이를 어디에 저장하는가 ?
또한 ARM Machine 에서는 이를 어디에 저장하는가 ?
- Intel 0x80 / ARM 0x1000

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

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

44. 커널 내부에서 메모리 할당이 필요한 이유는 무엇인가 ?
- 커널 역시 프로그램이기 때문에, 동작하기 위해서는 메모리 할당이 필요하다. 처음부터 모든 메모리를 잡을경우 부팅이 느려지기 때문에 필요할때 할당받아서 사용한다.

45. 메모리를 불연속적으로 할당하는 기법은 무엇인가 ?
- 블록체인 기법 / 인덱스 블록 기법 / **FAT** 기법

46. 메모리를 연속적으로 할당하는 기법은 무엇인가 ?
- fifo

47. Mutex 와 Semaphore 의 차이점을 기술하시오.
- **Mutex** 는 스레드에서 크리티컬 섹션을 방지하기 위해 사용되고, **semaphore** 의 경우 **fork** 에서 사용된다.
mutex 는 **lock** 걸렸을 때, 다른 스레드가 사용하지 못하지만, **semaphore** 는 사용 가능한 데이터를 찾아 탐색하고, 있을 경우 해당 데이터를 사용한다.

48. module_init() 함수 호출은 언제 이루어지는가 ?
- 모듈이 붙었을 경우 사용된다. **insmod** 사용 시 호출된다.

49. module_exit() 함수 호출은 언제 이루어지는가 ?
- 모듈이 떨어질때 사용된다. **Rmmod** 사용시 호출된다.

50. thread_union 에 대해 기술하시오.
- 8KB 의 스택이다. 내부에 **thread_info** 구조체가 포함되어 있다.

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)
- **task_struct** 내부의 **file** 이 가르키는 **file_struct** 내부의 **inode** 내부의 **i_rdev** 에 저장된다.

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.
- 프로그램 실행 시 내부에서 등록만 되어 있는 상태이다. 그 후 호출이 될 때, 커널이 지정된 함수를 호출하여 실행된다. 함수 실행 후 커널은 원래 기능으로 다시 바꾸어 준다.

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.
- 등록 후 계속 유지될 경우, 다른 프로세스가 해당 시스템 콜 사용시 문제가 발생하기 때문이다.

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.
- 명령 실행 효율을 높이기 위해 순서에 따라 처리하지 않는 기법이다. 단, 명령어를 읽어들이는 단계와 마지막 명령어 완료 단계는 반드시 순차적으로 이루어진다.

56. Compiler 의 Instruction Scheduling 에 대해 기술하라.
- 코드를 프로세서에 맞게 최적화 시키는 것을 말한다. 불필요하게 코어가 낭비되는것을 수정하여 좀 더 최적화 되게 만들어 준다.

57. CISC Architecture 와 RISC Architecture 에 대한 차이점을 기술하라.
- **CISC** 의 경우 명령어의 종류가 많아서 실제 프로그램 작성시 명령어를 적게 사용한다. 하지만 컴파일러가 복잡해 지며, 회로역시 복잡해지고 비싸진다. **RISC** 는 반대로 명령어의 종류가 상대적으로 작고, 단순하여 컴파일러가 단순하다. 프로그램 작성 시 많은 명령어가 사용되며 회로가 단순한 편이다. 최근에는 서로가 서로의 장점을 흡수하여 많이 비슷해진 상태이다.

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 ?
-

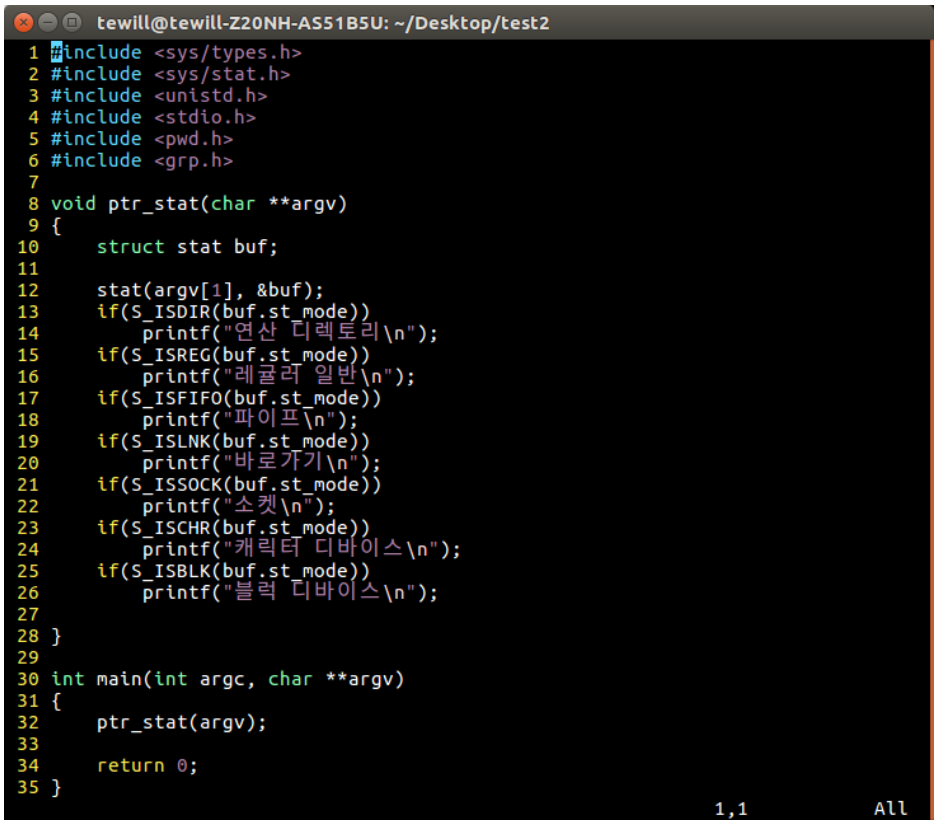
59. Pipeline 이 깨지는 경우에 대해 자세히 기술하시오.
- **jmp** 발생시 파이프 라인이 깨지게 된다. **cpu** 는 병렬로 처리를 하는데 **jmp** 발생 시 다른 위치로 가버리게 되서, 기존에 미리 진행중이던 연산이 쓸모 없어지게 된다. 또한 **push** 부터 다시 시작하게되어 여기서도 코어가 낭비된다.

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 ?
- 레지스터

61. Intel 의 Hyper Threading 기술에 대해 상세히 기술하시오.
- 실제로는 하나의 **cpu** 지만 가상으로 **2** 개로 인식하게 하는 기술이다.

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.
-




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

void ptr_stat(char **argv)
{
    struct stat buf;

    stat(argv[1], &buf);
    if(S_ISDIR(buf.st_mode))
        printf("연산 디렉토리\n");
    if(S_ISREG(buf.st_mode))
        printf("레귤러 일반\n");
    if(S_ISFIFO(buf.st_mode))
        printf("파이프\n");
    if(S_ISLNK(buf.st_mode))
        printf("바로가기\n");
    if(S_ISSOCK(buf.st_mode))
        printf("소켓\n");
    if(S_ISCHR(buf.st_mode))
        printf("캐릭터 디바이스\n");
    if(S_ISBLK(buf.st_mode))
        printf("블럭 디바이스\n");

}

int main(int argc, char **argv)
{
    ptr_stat(argv);

    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.

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.
- vi -t task_struct 를 통해 탐색을 하면, 그 중 **include** 내부의 **linux** 폴더의 **sched.h** 를 볼 수있다.
거기서 **task_struct** 내용을 보면 구조체로 되어있는 것을 볼 수 있다**.** 즉 **144** 번이 우리가 찾는 **task_struct** 이다**.**

67. Multi-Tasking 의 원리에 대해 서술하시오.
(Run Queue, Wait Queue, CPU 에 기초하여 서술하시오)
- 빠른 속도로 연산하는 것을 기반으로하여 페이징 기법과 컨테스트 스위칭을 통하여 멀티 테스킹을 구현한다.
우선순위에 따라 연산을 처리 순서가 정해진다. 이를 통해 **Run Queue** 에서 실행중이던 프로세스는 페이징을 통하여 **CPU** 에서 메모리로 저장되고**, Wait Queue** 에 위치하던 프로세스가 **Run Queue** 로 올라와 메모리에서 **CPU** 옮겨와 동작하게 된다**.**

68. 현재 삽입된 디바이스 드라이버의 리스트를 보는 명령어는 무엇인가 ?
- devs

69. System Call Mechanism 에 대해 기술하시오.
- 유저가 기능을 요청할 때 시스템 콜이 커널을 통해 처리하고, 그 결과를 유저에게 전달해준다.

70. Process 와 VM 과의 관계에 대해 기술하시오.
- 프로세스의 데이터는 가상 메모리로 관리된다.

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.
-

tewill@tewill-Z20NH-AS51B5U: ~/my_proj/linux5

```c
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <dirent.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);
}
```

-- INSERT -- 1,23 All

73. Linux 에서 fork()를 수행하면 Process 를 생성한다.
이때 부모 프로세스를 gdb 에서 디버깅하고자하면 어떤 명령어를 입력해야 하는가 ?
- **gdb** 에서 **set fork-follow-mode parent** 를 입력한다.

74. C.O.W Architecture 에 대해 기술하시오.
- 복사할 때 쓴다. **fork** 를 하여도 바로 모든 데이터를 복사하는게 아니라, 해당 데이터가 사용될 때 그 데이터만 복사하여 처리한다. 이를 통하여 불필요한 복사 과정을 줄일 수 있다.

75. Blocking 연산과 Non-Blocking 연산의 차이점에 대해 기술하시오.
- 블로킹은 해당 위치에서 입력받을때까지 대기상태로 있지만 논 블로킹 상태일 경우 이를 넘기고 계속 진행한다.

76. 자식이 정상 종료되었는지 비정상 종료되었는지 파악하는 프로그램을 작성하시오.

-

tewill@tewill-Z20NH-AS51B5U: ~/my_proj/linux6

```c
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/wait.h>

void term_status(int status)
{
    if(WIFEXITED(status))
        printf("(exit)status : 0x%x\n", WEXITSTATUS(status));
    else if(WTERMSIG(status))
        printf("(signal)status : 0x%x, %s\n", status & 0x7f,
            WCOREDUMP(status) ? "core dumped" : "");
}

int main(void)
{
    pid_t pid;
    int status;
    if((pid = fork()) > 0)
    {
        wait(&status);
        term_status(status);
    }
    else if(pid == 0)
        abort();
    else
    {
        perror("fork() ");
        exit(-1);
    }
    return 0;
}
```

1,1 All

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

-

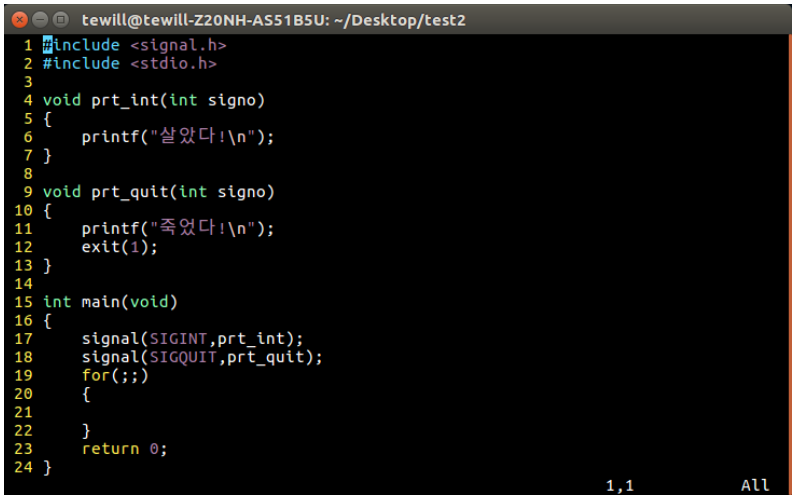
tewill@tewill-Z20NH-AS51B5U: ~/my_proj/linux6

```c
#include <sys/types.h>
#include <stdio.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();
    for(;;)
    {

    }

    return 0;
}
```

-- INSERT -- 32,2 All

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

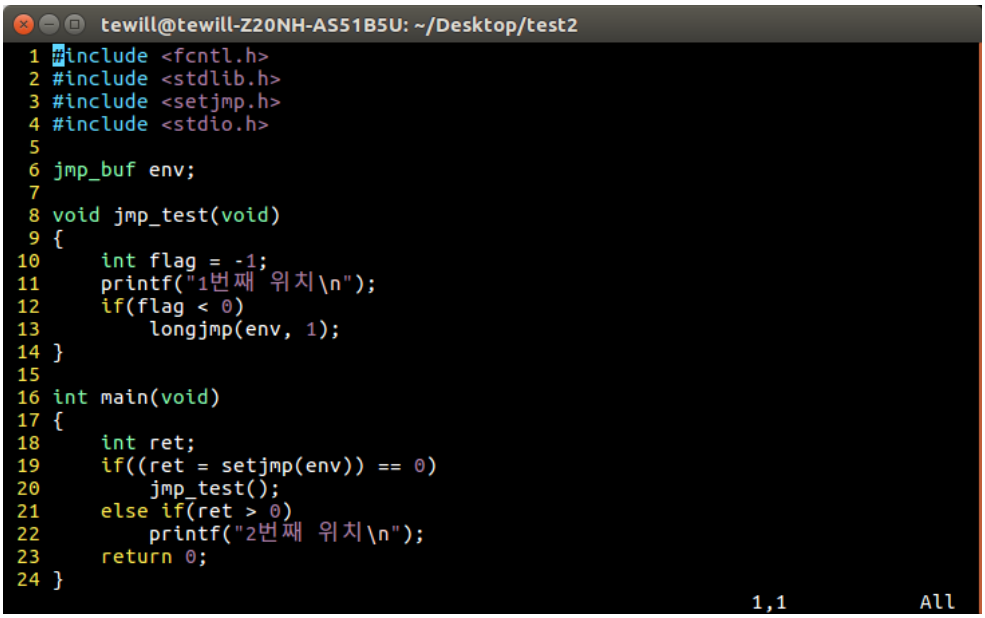
-




```c
#include <signal.h>
#include <stdio.h>

void prt_int(int signo)
{
    printf("살았다!\n");
}

void prt_quit(int signo)
{
    printf("죽었다!\n");
    exit(1);
}

int main(void)
{
    signal(SIGINT,prt_int);
    signal(SIGQUIT,prt_quit);
    for(;;)
    {

    }
    return 0;
}
```

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.

-




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

jmp_buf env;

void jmp_test(void)
{
    int flag = -1;
    printf("1번째 위치\n");
    if(flag < 0)
        longjmp(env, 1);
}

int main(void)
{
    int ret;
    if((ret = setjmp(env)) == 0)
        jmp_test();
    else if(ret > 0)
        printf("2번째 위치\n");
    return 0;
}
```

80. 리눅스에서 말하는 File Descriptor(fd)란 무엇인가 ?
- 리눅스에서 파일 정보를 보호하기 위하여 파일 디스크립터로 관리한다. 이는 현재 열려있는 파일들의 배열 인덱스를 나타낸다. **0** 번은 표준 입력, **1** 번은 표준 출력, **2** 번은 표준 에러이며 그 외에 **3** 번부터 열려있는 파일을 나타낸다.

81. stat(argv[2], &buf)일때 stat System Call 을 통해 채운 buf.st_mode 의 값에 대해 기술하시오.
- 해당 파일의 종류 및 권한을 나타낸다. 앞에서부터 **4hit** 는 파일 종류 그 뒤로 **3** 개 단위로 일시적 권한 부여, 사용자 권한, 그룹 권한, 제 **3** 자의 권한 이다.

82. 프로세스들은 최소 메모리를 관리하기 위한 mm, 파일 디스크립터인 fd_array,
그리고 signal 을 포함하고 있는데 그 이유에 대해 기술하시오.
- 프로세스의 데이터를 관리해야 하기때문에 **mm** 과 프로세스의 파일에 접근해야 하기 때문에 파일디스크립터, 그리고 **signal** 처리를 위해 필수적으로 이 **3** 가지가 들어간다.

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.
- **hypervisor / vme /** 심드 인스트럭션

88. 반 인원이 모두 참여할 수 있는 채팅 프로그램을 구현하시오.
-

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.

92. 91 번을 응용하여 Queue 와 Network 프로그래밍을 연동하시오.

93. Critical Section 이 무엇인지 기술하시오.
- 단순하게 설명하면 **data** 영역을 나타낸다. 스레드간에 공유가 일어나는 부분이기 때문에 해당 영역을 사용할 때 주의가 필요하다.

94. 유저에서 fork() 를 수행할때 벌어지는 일들 전부를 실제 소스 코드 차원에서 해석하도록 하시오.
- **fork** 호출 시, **clone()** 을 통하여 커널에게 전달한다. 그걸 전달받은 커널은 **sys_clone()** 을통해 옵션값으로 **do_fork()** 를 실행시키고, 이는 **kernel_thread()**를 통하여 프로세스를 생성한다. 생성 후 역순을 통해 결과값을 유저에게 전달한다.

95. 리눅스 커널의 arch 디렉토리에 대해서 설명하시오.
- 리눅스에서 지원되는 **CPU** 들의 정보가 저장되어있는 디렉토리이다.

96. 95 번 문제에서 arm 디렉토리 내부에 대해 설명하도록 하시오.
- **ARM CPU** 에 대한 파일들이 보관되어 있다.

97. 리눅스 커널 arch 디렉토리에서 c6x 가 무엇인지 기술하시오.
- **TI** 사의 **DSP** 내용이 들어있다.

98. Intel 아키텍처에서 실제 HW 인터럽트를 어떤 함수를 가지고 처리하게 되는지 코드와 함께 설명하시오.
- **irq_besc**
99. ARM 에서 System Call 을 사용할 때 사용하는 레지스터를 적으시오.
- **r7**

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.
- 알고 있다고 생각했던 것들을 막상 시험으로 보니 대부분 모르겠다는 사실을 알게 되었습니다.
저번 시험도 잘 못봤는데 이번 시험도 이러한 결과를 갖게 되어서,
그동안 너무 대충 한게 아닌가 싶어서 반성하게 되었습니다.
원래 계획했었던 다음주 쉬는 시간을 활용하여 학업 포트폴리오를 작성하는것에 있어서 좀 더 신경쓰도록 하겠고,
이번 시험을 통해 제대로 모르고 있던것들을 확인하였으니, 그 부분들을 좀 더 집중적으로 공부하도록 하겠습니다.
이번 시험은 저번 시험과는 다르게 쉬운것 부터 진행하긴 하였는데, 프로그램 코딩하는 부분에 있어서
시간을 많이 쓴다는 사실을 알게 되었습니다. 좀 더 코드 짜는 연습도 필요하다고 생각이 들었고,
이 부분에 있어서도 연습하도록 하겠습니다. 다음 시험은 이번보다 더 높은 점수를 받을 수 있도록 노력하겠습니다.