Xilinx Zynq FPGA, TI DSP, MCU 기반의 프로그래밍 및 회로 설계 전문가 과정

강사 – Innova Lee(이상훈)

gcccompil3r@gmail.com

학생 – hoseong Lee(이호성)

hslee00001@naver.com

2개월차 시험

1번

1. 시스템 프로그래밍 5 점 문제 , 파이프 통신을 구현하고 c type.c라고 입력할 경우 현재 위치의 디렉토리에 type.c 파일을 생성하도록 프로그래밍하시오.

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

int main(void)
{
  int fd,ret,filedes;
  char buf[1024];
  char *buf2 = "type.c";
  mkfifo("myfifo");
  fd=open("myfifo",O_RDWR);
  for(;;)
  {
    ret = read(0,buf,sizeof(buf));
    buf[ret-1]=0;
    printf("Keyboard input: [%s]\n",buf);
    int het = strcmp(buf,buf2);
    if(het==0){
      if((filedes=open(buf,O_CREAT|O_RDWR|0644))==0)
      {
```

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

int main(void)
{
  int fd,ret,filedes;
  char buf[1024];
  char *buf2 = "type.c";
  mkfifo("myfifo");
 // fd=open("myfifo",O_RDWR);
  for(;;)
  {
    fd=open("myfifo",O_RDWR);
    ret = read(0,buf,sizeof(buf));
    buf[ret-1]=0;
    printf("Keyboard input:[%s]\n",buf);
    int het = strcmp(buf,buf2);
    if(het==0){
      if((filedes=open(buf,O_CREAT|O_RDWR|0644))==0)
      {
        printf("File Open Error!\n");
```

```
            printf("File Open Error!\n");
            exit(1);
          }
          close(filedes);
          break;
        }

        read(fd,buf,sizeof(buf));
        buf[ret -1] = 0;
        printf("Pipe input :[%s]\n",buf);
        het = strcmp(buf,buf2);

        if(het==0){
          if((filedes=open(buf,O_CREAT|O_RDWR|0644))==0)
          {
            printf("File Open Error!\n");
            exit(1);
          }
          close(filedes);
          break;
        }
      }
      return 0;
}
```

cat > myfifo

→ 컴파일한 창에서 입력시에 버퍼 사이즈를 6개 입력을 넣어줘야합니다..ㅠㅠ

```
            exit(1);
          }
          close(filedes);
          break;
        }
        close(fd);
        fd=open("myfifo",O_RDWR,O_TRUNC);
        read(fd,buf,sizeof(buf));
        buf[ret -1] = 0;
        printf("Pipe input :[%s]\n",buf);
        het = strcmp(buf,buf2);

        if(het==0){
          if((filedes=open(buf,O_CREAT|O_RDWR|0644))==0)
          {
            printf("File Open Error!\n");
            exit(1);
          }
          close(filedes);
          break;
        }
      }
      return 0;
}
```

→ 초기화해줘봤지만, 안됨

결국 컴파일창에서 입력: type.c 는 바로 type.c 만들어지지만, ,

다른 터미널창에서 입력은 컴파일한 창에서 입력시 버퍼사이즈 6개의 입력을 넣어줘야 만들어집니다.

2. 369 게임을 작성하시오. 2초내에 값을 입력하게 하시오. 박수를 쳐야 하는 경우를 Ctrl + C를 누르도록 한다. 2 초 내에 값을 입력하지 못할 경우 게임이 오버되게 한다. Ctrl + C를 누르면 "Clap!" 이라는 문자열이 출력되게 한다. (5)

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

   파일시스템, 메모리,네트워크,디바이스,프로세스 관리

4. Unix 계열의 모든 OS는 모든 것을 무엇으로 관리하는가 ?

   모든것은 파일이다. 태스크와 장소를 제공하는 파일이라는 두가지 객체로 모든 것을 지원한다.

5. 리눅스의 장점에 대한 각각의 상세한 기술 리눅스에는 여러 장점이 있다.(배점 0.2 점)

아래의 장점들 각각에 대해 기술하라.

* 사용자 임의대로 재구성이 가능하다 → **커스터 마이징이 가능, 버그가 발견 됬을 때 알아서 고친다.**

* 열약한 환경에서도 HW 자원을 적절히 활용하여 동작한다 → **성능이 좋지 않은 컴퓨터에서도 리눅스가 깔린다. 모놀리식 방식이기 때문이다**

* 커널의 크기가 작다. → **윈도우보다 커널이 가볍다. 그래서 호환이 좋고, 성능이 좋다.**

* 완벽한 멀티유저, 멀티태스킹 시스템 → **리눅스의 또 다른 특징으로는 이미 하나의 프로세스가 실행되고 있는 가운데 또 다른 프로세스가 진행될 수 있다.**

* 뛰어난 안정성 → **네트워크 장비나 기상 기지국에서 사용될 정도로 뛰어난 안정성을 가지고 있다.**

* 빠른 업그레이드 → **빠른 업그레이드로 기술자들에게 최신 기술을 사용할 수 있게 한다. 그들만의 리그이다.**

* 강력한 네트워크 지원 → **리눅스는 인터넷의 모든기능을 지원한다. 인터넷용 프로그램인 웹 브라우저, 메일(pine, elm) 뉴스(tin, nn) 등 외에도 웹서버 (Apache,CERN), 메일서버 (sendmail, Smail), 뉴스서버 (INND, C-News), DNS(Domain Name System) 서버, IRC 서버 등 거의 모든 인터넷서버의 기능을 갖추고 있고, 방화벽(Firewall) 으로도 사용할 수 있다. 인터넷용 시리얼 프로 토콜인 PPP, SLIP, CSLIP 등도 지원한다.**

* 풍부한 소프트웨어 → **리눅스의 개방성 또한 큰 장점이라고 할 수 있다. 많은 우수 인력이 확보되어 있기 때문에 우수한 소프트웨어 개발이 가능하고 여러 배포판 개발 업체들이 있기 때문에 사용**

자에게 선택권이 주어진다.

6. 32 bit System에서 User와 Kernel의 Space 구간을 적으시오.

컴퓨터운영 체제는 일반적으로가상 메모리를커널 공간과사용자 공간으로 분리시킨다. 커널 공간은커널, 커널 확장 기능, 대부분의장치 드라이버를 실행하기 위한 예비 공간이다. 반면 사용자 공간은 모든 사용자 모드 응용 프로그램들이 동작하는 메모리 영역으로, 이 메모리는 필요할 때페이징처리될 수 있다. 각 프로세스에게 총 4GB크기의 가상공간을 할당하고, 0~3GB의 공간을 사용자 공간으로 사용하고 나머지 3~4GB를 커널 공간으로 사용한다.

7. Page Fault 가 발생했을때 운영체제가 어떻게 동작하는지 기술하시오.

Page Fault 는 가상 메모리를 물리 메모리로 변환하는 도중

물리 메모리에 접근했더니 할당된 페이지가 없을 경우 발생한다.

이것이 발생하면 현재 수행중이던 ip(pc) 레지스터를 저장하고

페이지에 대한 쓰기 권한을 가지고 있다면

Page Fault Handler 를 구동시켜서 페이지를 할당하고

저장해놨던 ip(pc) 를 복원하여 다시 Page Fault 가 발생했던 루틴을 구동시킨다.

만약 쓰기 권한이 없다면 Segmentation Fault 를 발생시킨다.

8. 리눅스 실행 파일 포맷이 무엇인지 적으시오.

ELF(Executable and Linking Format) - 리눅스의 실행파일포맷, 말 그대로 실행 가능한 그리고 링크를 하는 형식
DWARF- 리눅스의 디버깅 파일 포맷

9. 프로세스와 스레드를 구별하는 방법에 대해 기술하시오.

프로세스와 스레드의 차이를 알면 된다. 프로그램은 어떤 작업을 위해 실행 하는 파일. 프로세스는 프로그램이 실행되고 있는 상태. 프로세스는 실행이 됐을때운영체제 (OS)로 부터 자원(메모리와 같은 주소 공간)을 할당을 받습니다. 스레드는 한개의프로세스 내에서 동작되는 여러 실행의 흐름이다. 멀티프로세서와 멀티 스레드 둘다 여러 흐름이 동시에 진행된다는 점은 같지만, 프로세스는 독립적으로 실행되며 각각 별개의 메모리를 차지하고 있지만, 멀티스레드는 프로세스 내의 메모리를 공유해 사용할 수 있다. 즉, 메모리를 공유할 수 있는가 이다.

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

task_struct 구조체를 생성한다. 이것으로 프로세스와 스레드의 관련된 정보를 알 수 있다. 둘 모두 task_struct 라는 동일한 자료구조를 생성하여 관리한다.

11. 리눅스 커널 소스에 보면 current라는 것이 보인다. 이것이 무엇을 의미하는 것인지 적으시오. 커널 소스 코드와 함께 기술하시오.

(current → get_current() → currnt_thread_info() → thread_info()
Thread info 에 task_struct가 보인다. 이것이 커널 스택이다. 커널 스택에 지금 task_struct, cpu_context (컨텍스트 스위칭하려고) 가 들어 있었다.
즉, thread_info를 정보를 가지고 현재 구동중인 프로세스가 어떤놈이고, 이녀석이 지금 현재 cpu레지스터를 어디까지 사용했는지 전부 알 수 있다.)
current는 현재 구동중인 task에 task_struct 포인터를 가져오는 것이다. 즉, 현재 태스크의 task_struct 구조체를 가리킬 수 있게 해주며, 이것을 얻어올 수 있으므로 current 하고 pid를 찍으면 현재 구동중인 프로세스 아이디가 나온다. (Current 하고 tgid 찍으면 현재 구동중인 쓰레드 그룹아이디 값이 나온다.)

```
register unsigned long current_stack_pointer asm ("sp");
```

assembly sp 로 되어 있다. 실제 sp레지스터 그 주소값이 여기 있는것.

```
return (struct thread_info *)
        (current_stack_pointer & ~(THREAD_SIZE - 1));
```

thread_size는 8196으로 되어 있다. → 우리가 사용하는 프로세스 하나당 커널에 주어지는 스택공간은 8k라는 것이다. 여기서는 8k 단위로 정렬을 하고 있다.

12. Memory Management 입장에서 Process와 Thread의 핵심적인 차이점은 무엇인가 ?

쓰레드는 메모리를 공유한다. 쓰레드를 생성하면, 자식스레드와 부모스레드는 서로 같은 주소공간을 공유한다. 부모스레드와 자식스레드의 주소 공간이 critical section이 되고, 이를 막기위해 mutex와 semaphore, spin lock을 걸어주게 된다. 스레드가 아닌 프로세스였다면 프로세스는 pid값이 자식 프로세스와는 다르며, 메모리를 공유하지 못한다.

13. Task가 관리해야하는 3가지 Context가 있다. System Context, Memory Context, HW Context가 있다. 이중 HW Context는 무엇을 하기 위한 구조인가 ?

Context switch을 할때, 태스크의 현재 실행 위치와 같은 정보를 cpu레지스터(h/w context)에 저장하고. thread_struct로 관리된다. 실행중이던 태스크가 대기상태나 준비상태로 전이할 때 이 태스크가 어디까지 실행했는지 기억해두는 공간으로, 이후 이 태스크가 다시 실행될때 기억해 두었던 곳부터 다시 시작하게된다. 즉, 레지스터에 태스크 정보를 저장하고, 그 정보를 관리하는 것.

14. 리눅스 커널의 스케쥴링 정책중 Deadline 방식에 대해 기술하시오.

실시간 태스크 스케줄링 기법 중 하나인 EDF(Earliest Deadline First)알고리즘을 구현한 것으로 가장 가까운 (가장 급한) 태스크를 스케줄링 대상으로 선정한다. 정책을 사용하는 태스크들은 deadline을 이용하여 RBTree에 정렬되어, DEADLINE을 사용하는 태스크의 경우 우선순위는 의미가 없다. 이에따라 우선순위 기반 스케줄링 정책대비, 기아현상등의 문제에 효율적이며, 주기성을 가지는 프로그램과 제약시간을 가지는 응용들에 효과적으로 적용 가능하다.

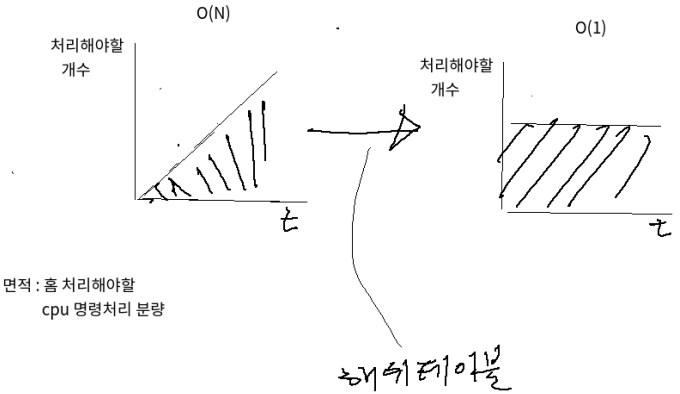
15. TASK_INTURRUPTIBLE과 TASK_UNINTERRUPTIBLE은 왜 필요한지 기술하시오.

태스크가 특정한 사건을 기다려야 할 필요가 있을 때, 태스크가 이미 lock이 걸려있는 자원에 대해 lock을 획득하려고 시도, 디스크같은 주변 장치에 요청을 보내고 그 요청이 완료되기까지 기다리는 것. 대기하는 동안 기다리는 그 사전 외에는 일체 방해받아선 안되는 경우가 TASK_UNINTERRUPTIBLE, 그렇지 않으면 TASK_INTERRUPTIBLE, 대기상태로 전이한 태스크는 기다리는 사건(event)에 따라 특정 큐(queue)에서 대기한다. 태스크가 기다리고 있던 사건(event)이 발생하면 대기 상태 태스크는 TASK_RUNNING(ready)상태로 전이한다.

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

시간복잡도: 어떤 알고리즘이 얼마나 걸리느냐(CPU사용량)
O(1) : 입력한 자료의 수에 관계없이 일정한 실행 시간을 갖는다.
O(N): 입력 자료의 수 N개에 따라 선형적인 실행 시간이 걸리는 경우-<u>입력자료 각각에 일정량(시간)이 할당되는 경우</u>

cpu에서 처리할 개수가 많으면 O(1) 가 좋고, 처리해야할 개수가 별로 없으면 Q(N) 을 사용하는 것이 좋다.



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

2번에 위치하는 것이 좋다.

fork()는 같은 Task를 복사하게 된다.

그리고 다시 그 코드를 사용할 것이라면

CPU의 Instruction Cache, Data Cache를 재활용하는 것이 최고다.

결론적으로 Cache Affinity를 적극 활용하자는 것이다.

18. 앞선 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.

2번에 위치하는 것이 좋다.
이번에는 주어진 시간내에 0, 1, 3 은 Scheduling 을 수행할 수 없기 때문에
오히려 상대적으로 Task가 적은 2번에 배치하는 것이 좋다.

Cache Affinity를 포기할지라도
아예 Task 를 동작시킬 기회가 없는것보다 좋기 때문이다.

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

복수개의 cpu를 가지고 있는 구조의 컴퓨터 시스템의 자원공유로 우마에서는 병목현상이 발생하였고, 이를 해결하기위해 누마구조가 나왔다.

UMA(Uniform Memory Access)- 메모리 접근하는 속도가 모두 같다. 모든 cpu들이 메모리를 공유한다. UMA구조의 시스템에서 리눅스가 수행된다면 한 개의 노드가 존재하고, 이 노드는 리눅스의 전역변수인 contig_page_data를 통해 접근 가능하다. 만약 NUMA구조의 시스템에서 리눅스가 수행된다면 복수 개의 노드가 존재 할 것이다.

NUMA(Non-uniform Memory Access)- 메모리 접근 속도가 다르다. NUMA에서 메모리 접근 타이밍이 다르다. 필요한 메모리가 반대쪽 메모리에 존재할 때, 네트워크 버스를 타고 접근함. 이기종 아키텍처에 사용되는 방식이다.

```
#ifdef CONFIG_NUMA
        int node;
#endif

        /*
         * The target ratio of ACTIVE_ANON to INACTIVE_ANON pages on
         * this zone's LRU.  Maintained by the pageout code.
         */
        unsigned int inactive_ratio;

        struct pglist_data      *zone_pgdat;
```

```
pg_data_t *pgdat_list[MAX_NUMNODES];
```

pglist_data → 메모리 집합 뱅크의 구조체이고, NUMA구조의 시스템에서 복수 개의 노드는 pgdat_list라는 이름의 배열에서 접근가능하다. 이 노드라는 일관된 자료구조를 통해 전체 물리메모리에 접근할 수 있다. UMA 구조와 NUMA 구조 둘 모두, 하나의 노드는 pg_data_t 구조체를 통해 표현된다.

20. Kernel의 Scheduling Mechanism에서 Static Priority와 Dynamic Priority 번호가 어떻게 되는지 적으시오

static priority – 0~99 번
dynimic priority – 100~139 번

21. ZONE_HIGHMEM 에 대해 아는대로 기술하시오.

메모리 영역의 실제적인 활용과 레이아웃은 아키텍쳐마다 다르다. 예를들면, 어떤 아키텍쳐에서는 모든 메모리 주소에 DMA를 사용할 수 있다. 이러한 아키텍쳐에서는 ZONE_DMA영역이 비게되며, 따라서 용도에 관계없이 ZONE_NORMAL를 할당하여 사용할 수 있다. 반면 x86아키텍쳐에서 ISA 디바이스는 모든 32비트 주소 공간으로 DMA를 할 수가 없는데(할 수 있다면 4GB), 그 이유는 ISA 장치(24비트까지만 접근가능) 가 물리적 메모리의 처음 16MB만을 접근할 수 있기 때문이다. 따라서 x86에서는 ZONE_DMA는 0~16MB 범위의 모든 메모리로 구성된다 ZONE_HIGHMEM 역시 동일한 방식으로 동작한다. 즉 아키텍처에 따라 직접 매핑 가능한 범위가 일정하지 않다. 리메모리가 1GB이상이라면 896MB까지를 커널의 가상주소공간과 1:1로 연결해주고, 나머지 부분은 필요할 때 동적으로 연결하여 사용하는 구조를 채택했는데, 이때 896MB 이상의 메모리 영역을 ZONE_HIGHMEM이라 부른다. 어떤 아키텍처에서는 모든 메모리가 직접 매핑되므로 ZONE_HIGHMEM 영역이 비어있는 경우도 있다.

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

페이지 프레임, struct page

23. Linux Kernel은 외부 단편화와 메모리 부하를 최소화하기 위해 Buddy 할당자를 사용한다. Buddy 할당자의 Algorithm을 자세히 기술하시오.

버디할당자는 zone구조체에 존재하는 free_area배열을 통해 구축된다. 존하나당 버디가 존재한다. free_area는 10개의 엔트리를 가지고, 0~9까지의 엔트리는 free_area 가 관리하는 할당의 크기를 나타낸다 4MB의 크기이다. nr_free라는 free한 페이지 개수를 나타내는 벼수와 list_head라는 연결리스트가 존재한다. map이라는 변수의 비트맵에 페이지 프레임 단위로 봤을 때의 상태를 저장한다. free_area[order].map을 본다. 양쪽 둘 모두 alloc, free일 때는 0, 그리고 한쪽은 alooc이고, 한족은 free일때는 1로 한다. order(0)의 한 페이지 프레임 단위로 메모리를 관리하고, order(1) 은 두페이지 프레임 단위로 메모리를 관리한다. order(2)는 8~15사이에 물리메모리에 16kb만큼의 메모리를 할당 할 수 있기 때문에 주의 해야한다. 다른것들도 이와 마찬가지이다. free_area.free_list 라는 것은 빈공간을 나타내는 리스트이다. 만약 위의 물리메모리에 두페이지에 해당하는 메모리 요청이 온다면 free_list에 두페이지 크기의 메모리가 존재하지 않으니 상위 order에서 페이지 프레임을 할당 받아 두 부분으로 나누고, 한 부분은 할당, 다른 한부분은 하위 order에서 가용페이지 프레임으로 관리한다. 이를 버디할당자라고 부른다.

24. 앞선 문제에 이어 내부 단편화를 최소화 하기 위해 Buddy에서 Page를 받아 Slab 할당자를 사용한다. Slab 할당자는 어떤식으로 관리되는지 기술하시오

페이지 프레임 크기가 클수록 내부 단편화로 인한 낭비되는 공간이 큰데, 버디를 이용하여 공간을 쪼개게 되면 공간을 쪼개는 작업이 속도를 느리게 하는 문제점이 있다. 위의 문제를 해결하기 위한 것이 슬랩할당자이다. 페이지 프레임의 크기가 4KB 일 때, 64byte 크기의 메모리를 공간을 요청하면 미리 4KB 페이지 프레임을 할당 받은 뒤 이 공간을 64byte 분할해두어 이 공간을 떼어주도록 한다. 일종의 캐시로 사용하는 것인데 이런 메모리 관리하는 정책을 슬랩 할당자라 부른다. 자주 할당하고 해제되는 크기의 cache 를 가지고 있어야 내부 단편화를 최소화 시킬 수 있다. 2 의 승수크기의 cache 를 128KB 유지한다. 다양한 크기의 캐시를 관리하는 kmem_cache 라는 자료구조가 정의되어 있다. : 슬랩 할당자는 위의 함수 외에도 외부 인터페이스 함수로 kmalloc() / kfree()를 제공한다. kmalloc()함수를 이용ㅇ해 한번에 할당 받을 수 있는 최대 크기는 128K 혹은 4MB 이며 할당된 공간은 물리적으로 연속이라는 특징을 가진다.

25. 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)

task는 자신의 고유한 가상 메모리를 갖는데, 커널이 mm_struct 구조체를 관리해야한다. 즉, text,stack,hea,data 정보를 알아야한다. task_struct에서 태스크의 메모리와 관련된 내용은 mm_struct구조체가 관리한다. 리눅스 커널은 가상 메모리 공간 중 같은 속성을 가지며 연속인 영역을 region이라는 이름으로 부른다. 각각의 region을 vm_area_struct라는 자료구조를 통해 관리한다. 태스크는 텍스트, 스택, 힙, 데이터 등의 region으로 구성된다. 텍스트는 가상메모리의 0번지에서부터 자신의 크기만큼 차지하고, 텍스트의 끝부터 데이터가 차지한다. 그리고 데이터의 끝부터 힙이 차지하며, 힙의 끝은 brk라는 변수가 가리킨다. 태스크에 새로운 메모리 공간이 동적으로할당되면 brk부터 힙이 위쪽 방향으로 자라게 된다.

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

공통의 속성을 갖는 페이지들이 모여 vm_area를 구성하고, 이를 vm_area_struct라는 자료구조로 관리한다. 이 자료구조가 가르키는 세그먼트의 시작주소, 끝주소, 그리고 region의 접근제어 등을 기록하는 플래그 등의 변수를 갖는다. 같은 태스크에 속한 vm_area_struct들이 모여 하나의 mm_strcut내에서 관리되는 것이다.

27. 프로그램을 실행한다고 하면 fork(), execve()의 콤보로 이어지게 된다. 이때 실제 gcc *.c로 컴파일한 a.out을 ./a.out을 통해 실행한다고 가정한다. 실행을 한다고 하면 a.out File의 Text 영역에 해당하는 부분을 읽어야 한다.실제 Kernel은 무엇을 읽고 이 영역들을 적절한 값으로 채워주는가 ?

ELF Header 와 Program Headers 를 읽고 값을 적절하게 채운다.

28. 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.

C언어를 사용하기 위해서는 반드시 Stack이 필요하다.
Kernel 영역에서도 동작하는 코드가 올라가기 위한 Text 영역
전역 변수가 있는 Data 영역, 동적 할당하는 Heap,
지역 변수를 사용하는 Stack이 존재한다.
이는 역시 User 영역에서도 동일하므로 양쪽에 모두 메모리 공간이 구성된다.

29. VM(가상 메모리)와 PM(물리 메모리)를 관리하는데 있어 VM을 PM으로 변환시키는 Paging Mechanism에 대해 Kernel에 기반하여 서술하시오

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

VM 과 PM 연결시 사용하는 것은 페이지 테이블이다. 가상주소를 물리주소로 변환하는 주소 변환정보를 기록한 테이블이다. 이를 통해 가상메모리의 주소가 실제 메모리를 가르킬수 있도록 한다. 그리고 demend on page로 일부 실행에 필수적인 부분만 지정해놓는다. 이후 물리메모리에 적재되어 있지 않은 부분이 있다면 페이지 폴트 핸들러는 페이지 프레임을 할당받고 페이지 테이블에 다시 기록하고 물리 주소로 변환 할 수 있게 한다. 페이지 프레임을 할당받고 프레임 번호를 기록하는 것을 페이징이라한다.

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

1. HAT(HW Address Translation) 는 가상 메모리 공간을 실제 물리 메모리 공간으로 변환한다.

2. TLB(Translation Lookaside Buffer) 는 가상 메모리 주소와 대응되는 물리 메모리 주소를 Caching한다.

가상 주소로 부터 물리 주소로의 변환을 담당하는 별도의 하드웨어이다.
주소 변환을 위해서는 페이지 디렉터리와 페이지 테이블을 탐색해야 하는데, 이것을 하드웨어 적으로 처리한다. 또한 TLB 같은 페이지테이블 엔트리 캐시를 사용하여 빠른 주소변환을 담당한다.

31. 하드디스크의 최소 단위를 무엇이라 부르고 그 크기는 얼마인가 ?

Sector, 512 byte

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

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

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

Kernel Source 에서 아래 파일에 위치한다. fs/open.c 에 위치하며 SYSCALL_DEFINE3(open, ~~~) 형태로 구동된다. 이 부분의 매크로를 분석하면 결국 sys_open() 이 됨을 알 수 있다.

Open() 함수가 호출 된다고 가정을 한다면, open함수가 user단에서 호출 되어 제일 먼저, ax레지스터에 5가 셋팅 되고, vector_start(IDT) 에서 80이 호출되면서 유저에서 커널로 제어권이 넘어간다. 다음으로 IDT에서 128번인 시스템콜이 불러지고 sys_call_table 에서 5번에 해당하는 함수가 최종적으로 동작하는 방식으로 작동한다.

34. **task_struct 에서 super_block 이 하는 역할은 무엇인가 ?**

super_block 은 루트 파일 시스템('/') 의 위치 정보를 가지고 있다. 또한 super_block은 파일시스템의 메타 데이터를 가지고 있다. 수퍼블록을 읽으면 메타 데이터가 읽히기 때문에 어떤 파일 시스템인지 알 수 있다.

35. **VFS(Virtual File System)이 동작하는 Mechanism 에 대해 서술하시오.**

일관된 사용하기 때문에 VFS(가상 파일 시스템) 은 유저에서 호출하는 함수의 인자에 담겨있는 파일이름을 보고, 해당 파일 시스템이 어떤것인지 판단하고, 사용자가 원하는 일을 해줄 수 있는 파일 시스템 고유의 함수를 호출해준다.

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

외부인터럽트 – 하드웨어들, 수행중인 태스크와 관련 없이 주변장치에서 발생된 비동기적 신호

내부인터럽트 – 소프트웨어, 현재 태스크와 관련있는 동기적신호, 트랩이라고 한다.

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

Fault→ page fault 등,
trap→ int, system call 등,
abort→ devide by zero 등

38. **35**번에서 분류한 녀석들의 특징에 대해 기술하시오.

Fault의 경우 **Page Fault**가 대표적이므로 발생시
현재 진행중인 주소를 복귀주소로 저장하고 **Fault**에 대한 처리를 진행하고
다시 돌아와서 **Fault**가 났던 부분을 다시 한 번 더 수행한다.

Trap의 경우 **System Call**이 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.

수퍼 블록객체 - 현재 사용중인 파일시스템 당 하나씩 주어진다.
아이노드 객체- 파일과 관련된 정보를 담기위한 구조체이다. 파일이 메타 데이터를 읽어서 아이노드 객체에 채워준다.
파일객체 – task가 open한 파일과 관련된 정보를 관리한다. 태스크가 여러개일때 오프셋 정보를 다르게 가지고,task가 inode에 접근하는 동안만 메모리에 유지된다.
디엔트리 객체 - 아이노드 객체를 자신의 태스크와 연결, 캐시역할을 한다.
Path - 가상파일이 뭘 물었는지 알려준다.

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

외부 인터럽트의 경우 **32 ~ 255**까지의 번호를 사용한다.
여기서 **128(0x80)**에 해당하는 **System Call**만은 제외한다.

idt_table에서 **128**을 제외한 **32 ~ 255** 사이의 번호가 들어오면 실제 **HW Device**다.
여기서 같은 종류의 **HW Device**가 들어올 수 있는데
그들에 대해서 **Interrupt**를 공유하게 하기 위해
irq_desc라는 **Table**을 추가로 두고 **active**라는 것으로 **Interrupt**를 공유하게끔한다.

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

Intel 의 경우에 sys_call_table
ARM 의 경우에는 __vectors_start + 0x1000 에 해당함

41. **38에서 User Space에서 System Call 번호를 전달한다. Intel Machine에서는 이를 어디에 저장하는가 ? 또한 ARM Machine에서는 이를 어디에 저장하는가 ?**

Intel 의 경우에는 ax 레지스터에 ARM 의 경우에는 r7 레지스터에 해당한다.

42. **Paging Mechanism에서 핵심이 되는 Page Directory 는 mm_struct의 어떤 변수가 가지고 있는가 ?**

pgd

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

Intel 의 경우엔 CR3
ARM 의 경우엔 CP15

44. **커널 내부에서 메모리 할당이 필요한 이유는 무엇인가 ?**

커널 스택으로 주어진 메모리 공간은 고작 **8K** 에 해당한다.
(물론 이 값은 **ARM** 이라고 가정하였을 때고 **Intel** 은 **16 K** 에 해당한다**)**
문제는 스택 공간이라 특정 작업을 수행하고
이후에 태스크가 종료되면 정보가 사라질 수도 있다.
이 정보가 없어지지 않고 유지될 필요가 있을 수도 있다.

뿐만 아니라 커널 자체도 프로그램이기 때문에 메모리 공간이 필요하다.
운영체제 또한 **C** 로 만든 프로그램에 불과하다는 것이다.
그러니 프로그램이 동작하기 위해 메모리를 요구하듯 커널도 필요하다.

45. **메모리를 불연속적으로 할당하는 기법은 무엇인가 ?**

vmalloc()함수이다.

46. **메모리를 연속적으로 할당하는 기법은 무엇인가 ?**

kmalloc()함수이다

47. **Mutex 와 Semaphore 의 차이점을 기술하시오.**

Mutex 와 **Semaphore** 모두 **Context Switching** 을 유발하게 된다.

차이점이라면 **Mutex** 는 공유된 자원의 데이터를
여러 쓰레드가 접근하는 것을 막는다.

Semaphore 는 공유된 자원의 데이터를
여러 프로세스가 접근하는 것을 막는 것이다.

세마포어는 뮤텍스가 될수 있지만, 뮤텍스는 세마포어가 될 수 없다.
세마포어는 소유할 수 없는 반면 뮤텍스는 소유가 가능하여 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.
뮤텍스는 동기화대상이 하나일때, 세마포어는 동기화 대상이 하나 이상일때 사용한다.

48. **module_init() 함수 호출은 언제 이루어지는가 ?**

Insmod 명령어를 통해 디바이스 드라이버를 커널에 적재하면 매크로에 지정된 함수가 호출됨.

49. **module_exit() 함수 호출은 언제 이루어지는가 ?**

rmmod가 명령어를 통해 디바이스 드라이버를 커널에 적재하면 매크로에 지정된 함수가 호출됨.

50. **thread_union 에 대해 기술하시오.**

태스크가 생성이 되면 각각의 태스크마다 task_struct 구조체와 thread_union을 할당한다.`
thread_union은 태스크당 할당되는 8KB의 스택이다. . thread_union도 리눅스가 프로세스를 관리하기 위한 커널 스택에 대한 자료구조이다.

51. **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)

118 584 include/linux/fs.h <<inode>>

```
struct inode {
dev_t                   i_rdev;
```

inode 구조체에 i_rdev 가 존재하고 i_rdev에 주번호와 부번호를 저장한다.

52. **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.**

커널에서 이 파일의 아이노드 객체를 통해서 주/부 번호를 얻고 장치 파일의 유형을 얻는다. 장치 파일의 유형을 통해 cdev_map 자료구조에 접근하여 주번호를 인자로 cdev 구조체를 검색한다. 검색 결과, 앞서 커널에 file_operations구조체를 찾게 되고, 따라서 이 드라이버에서 작성한함수로 file_operations 가 채워진다. 오픈한 파일이 있으면 거기에 대한 쓰기 나 일기가 동작하는 것을 볼 수 있다.

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 Mechanism이 필요한 이유가 무엇인지 자세히 기술하라.**

드라이버나 외부 인터페이스 함수의 메모리 배치 때문에 이 메커리즘 방식이 필요로하다.
Free_pages()함수나 get_free_pages()함수를 통해서 free_area에 비어 있는 곳을 찾고, 드라이버가 필요한 메모리공간을 kmalloc, kfree를 하면서 물리 메모리에 순차 배치를 시킨다.

54. **Character Device Driver를 아래와 같이 동작하게 만드시오 . read(fd, buf, 10)을 동작시킬 경우 1 ~ 10 까지의 뎃셈을 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.**

위의 OoO 개념으로 Compile-Time에 Compiler Level에서 직접 수행한다. Compiler가 직접 Compile-Time에 분석하므로 최적화의 수준이 높으면 높을수록 Compile Timing이 느려질 것이다. 그러나 성능만큼은 확실하게 보장할 수 있을 것이다.

56. **Compiler의 Instruction Scheduling에 대해 기술하라.**

57. **CISC Architecture와 RISC Architecture에 대한 차이점을 기술하라.**

CISC 프로세서란 프로세서 내부에 많은 명령어들을 담고 있는 프로세서를 말한다. 복잡한 명령어 컴퓨터(Complex Instruction Set Computer)라는 뜻으로 CISC라 부른다.
대개의 PC CPU들은 CISC 프로세서이다. CPU 자체에 많은 명령어가 있기 때문에 프로그래머들은 그것을 활용하여 쉽게 프로그램을 개발 할 수 있다.

반면 RISC(Reduced Instruction Set Computer)는 그에 비해 적은 명령어를 갖고 있다 스스로 처리하는 명령어가 적기 때문에 빠른 처리를 한다. 보통 서버용 CPU는 RISC 프로세서 이다.

→ CISC와 RISC의 가장 큰 차이점은 다이 사이즈다.

CISC는 다이 사이즈가 크다보니 여러가지 기능 유닛들을 포함할 수 있다.
그러다보니 상대적으로 전력을 많이 소비하게 되는 면모를 보인다.
RISC는 다이 사이즈가 작다보니 CISC처럼 여러 기능 유닛들을 집어넣을 수 없다.
그러한 이유로 전력 소모량도 상대적으로 적다.
또한 Dynamic하게 서포팅을 해주는 것이 없으므로
Compiler의 최적화 알고리즘은 RISC 쪽이 더 신경을 많이 써야 한다.
(CISC 쪽은 HW가 어느정도 커버 해주므로)

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

Dynamic Instruction Scheduling

59. **Pipeline이 깨지는 경우에 대해 자세히 기술하시오.**

분기가 발생할 때, 명령어들이 취소되면서 파이프라인이 깨지게된다. 분기 명령어는 파이프라인을 때려부순다. jmp 나 call 등의 분기명령어가 많을수록 힘들다.

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

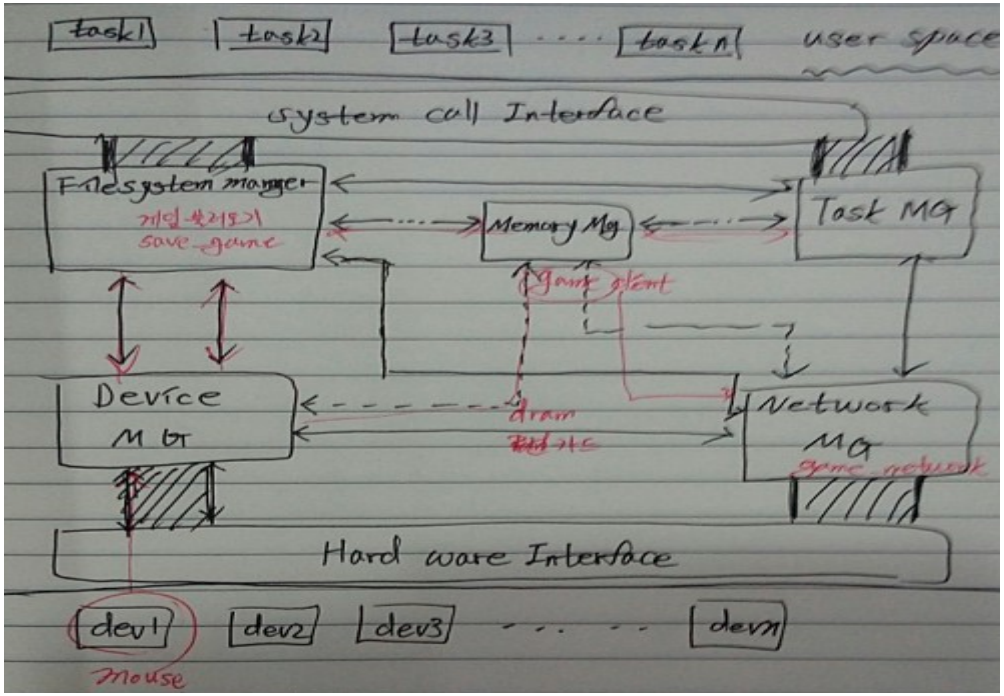
Task – 스케줄링

61. **Intel의 Hyper Threading 기술에 대해 상세히 기술하시오.**

Hyper Threading 은 Pentium 4 에서
최초로 구현된 SMT(Simultaneous Multi-Threading) 기술명이다.
Kernel 내에서 살펴봤던 Context를 HW 차원에서 지원하기 때문에
실제 1개 있는 CPU가 논리적으로 2개가 있는것처럼 보이게 된다.
HW상에서 회로로 이를 구현하는 것인데

Kernel에서 Context Switching에선 Register들을 저장했다면
이것을 HW 에서는 이러한 HW Context 들을 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 TLP 를 극대화하기에 좋은 기술이다.

62. **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.



커널 공간에는 운영체제가 관리해야하는 5대요소가 있다. 이 5개의 블록들은 서로 상호작용하며 동작한다. 시스템 상에서 실행되는 응용 프로그램은 시스템 콜을 통하여 커널과 통신한다. 디스크에 저장되어있는 게임을 구동시킬 파일은 파일 시스템이 관리되어있고, 게임은 이미 실행된 상황이므로 메모리에 올라가있다. Memory manager는 dram을 사용하기에 device가 인식되어야 한다. 그리고, 프로세스 구동은 프로그램이 메모리에 올라갈 때, 메모리를 관리한다. 메모리에 올라가는 것은 스케줄러가 결정하는데, task manager가 스케줄러를 관리한다. 또한 게임은 클라이언트와 서버 등 여러가지 프로세스를 동작시킨다. 여러가지 프로세스간의 IPC통신 또한 task manager가 관리한다. 게임상에서 더블클릭을 하는 것은 진짜 물리적인 장치인 마우스의 입력으로 들어온다. 이들은 하드웨어를 관리하는 구조체dev1~ devn으로 표시 되어 있다. 마우스 입력은 하드웨어 인터페이스를 통해 연결된다. 또한 이 게임은 네트워크 프로그램이다. 네트워크 상에서 받아오는 정보는 메모리에 올라가야 한다. 또한 네트워크 프로그램은 랜카드가 없다면 ip통신이 불가능하다. 랜카드를 인식하려면 device manager가 필요하다. 즉 모든 것이 동시다발적으로 (순차적이지만) 상호작용을 하는 것이다.

63. 파일의 종류를 확인하는 프로그램을 작성하시도록 하시오.

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

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

Clinet

```
#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

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

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

Server

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

#include <signal.h>
#include <sys/wait.h>

typedef struct sockaddr_in          si;
typedef struct sockaddr *sp;

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

#define BUF_SIZE                        32

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);
```

```
	for(;;)
	{
		int len = read(sock, msg, BUF_SIZE);

		if(!len)
			return;

		printf("%s\n", msg);
	}
}

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

	for(;;)
	{
		write(sock, msg, strlen(msg));
		sleep(1);
	}
}

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)
	{
```

```
}

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)
```

```
		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;
}
```

```
		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, buf, BUF_SIZE)) != 0)
			{
				printf("%s\n", buf);
				write(clnt_sock, "Hello", strlen("Hello"));
				sleep(1);
			}

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

```
        }
        close(serv_sock);

        return 0;
}
```

65. **Shared Memory를 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/$(자신의버전).tar.gz 를 다운받아보고 이 압축된 파일을 압축 해제하고 task_struct 를 찾아보도록 한다. 일련의 과정을 기술하면 된다.**

67. **Multi-Tasking의 원리에 대해 서술하시오. (Run Queue, Wait Queue, CPU에 기초하여 서술하시오)**

다중 프로세스의 경우, A 프로세스가 cpu를 점유하다가 B프로세스에게 cpu의 점유권을 넘겨줄때 , context switching 기법을 사용한다. A프로세스가 B프로세스에게 cpu의 점유를 넘겨주는 순간에 A프로세스의 상태, 범용/특수 레지스터값, pc(프로그램카운터값), 스케줄링 정보, 주기억장치의 정보등을 어딘 가에 저장해 놓지 않는다면, 나중에 다시 A 프로세스가 cpu를 점유 했을때, 예전에 cpu의 점유를 넘겨줄 상태에서 부터 시작할 수 없게된다. 따라서 프로세스가 cpu의 점유를 다른 프로세스에게 넘겨 줄때, 해당 시점의 프로세스의 정보를 저장해 놓아야한다. 프로세스의 정보를 저장하는곳이 task_struct이다. 현재 cpu를 점유하고 있는 프로세스는 Run Queue에 저장이 되고, cpu를 점유하기 기다리고 있는 프로세스는 wait Queue에 저장이 된다.

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

Lsmod

69. **System Call Mechanism에 대해 기술하시오**

커널 영역의 기능을 사용자 모드가 사용 가능하게, 즉 프로세스가 하드웨어에 직접 접근해서 필요한 기능을 사용할 수 있게해준다.
각 시스템 콜에는 번호가 할당되고 시스템 콜 인터페이스는 이러한 번호에 따라 인덱스 되는 테이블을 유지된다. IDT에 의해 트랩을 시작하고, 시스템콜핸들러를 실행한다.
시스템 호출 함수를 사용하려면 함수 정보가 있는 라이브러리에서 대응하는 함수와 연결, 라이브러리 내부의 함수 호출하고, 호출번호에 해당하는 시스템 호출함수 실행 가상파일 시스템을 통해 커널 자원인 파일, 디렉토리, 디바이스 등을 하나의 파일처럼 접근한다.

70. **Process와 VM과의 관계에 대해 기술하시오.**

프로세스에서 프로세스로 vm을 공유 하려하는데, 권한을 주기에는 너무 일이 힘들다. 그래서 파이프라는 개념이 있다.

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

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

int save_area[1024];

int main(int argc, char **argv)
{
   int i;
   int ret;
   int index;
   int fd;
   char buf[1024];
   fd = open(argv[1], O_RDONLY);
   ret = read(fd, buf, sizeof(buf));
   for(i = 0, index = 1; buf[i]; i++)
   {
      if(buf[i] == '\n')
      {
         save_area[index] = i;
         index++;
      }
   }
   printf("Front 5 Lines\n");
   write(0, buf, save_area[5] + 1);
   printf("Back 5 Lines\n");
   printf("%s", &buf[save_area[index - 6] + 1]);
   return 0;
}
```

72. 디렉토리 내에 들어 있는 모든 File들을 출력하는 Program을 작성하시오.

```
#include <sys/types.h>
#include <dirent.h>
#include <stdio.h>

int main(void)
{
   DIR *dp;
   int i = 0;
   struct dirent *p;
   dp = opendir(".");
   while(p=readdir(dp))   // 디렉토리에 있는 리스트가 넘어온다.
   {
      if(p->d_name[0] == '.')
         continue;
      printf("%-16s ",p->d_name);
      if((i+1)%5==0)
         printf("\n");
      i++;
   }
   printf("\n");
   closedir(dp);
   return 0;
}
```

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

gdb [프로그램명] [부모프로세스 pid]

74. C.O.W Architecture에 대해 기술하시오.

C.ow : copy on writetext 기계어 먼저 복사한다. 그리고 전역변수로 스택에 다가가고 메모리에 쓰기 작업이 발생할 때 복사한다.

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

아주빠르게 통신해야할 때는 논블록킹이 좋다. 반드시 순차적으로 진행되어야하는 것은 블록킹이좋다.

76. 자식이 정상 종료되었는지 비정상 종료되었는지 파악하는 프로그램을 작성하시오

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

int main(void)
{
  pid_t pid;
  int status;
  if((pid=fork())>0)
  {
    wait(&status);
    printf("status : %d\n",status);
  }
  else if(pid == 0)
    exit(7);
  else
  {
    perror("fork() ");
    exit(-1);
  }
  return 0;
}
```

정상종료 되었다면 **status : 1792** 이다.
비정상 종료된다면 **status**의 하위 **8**비트에는 프로세스를 종료한 시그널의 번호가 저장되며 상위 **8**비트에 **0**이 저장된다.

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

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

int main(void)
{
  pid_t pid;
  if((pid = fork()) > 0)
        sleep(1000);
  else if(pid == 0)
        ;
  else
  {
        perror("fork() ");
        exit(-1);
  }
  return 0;
}
```

78. **SIGINT는 무시하고 SIGQUIT을 맞으면 죽는 프로그램을 작성하시오.**

```
#include <signal.h>
#include <unistd.h>

int main(void)
{
  signal(SIGINT, SIG_IGN);
  for(;;)
    pause();
  return 0;
}
```

79. **goto는 굉장히 유용한 C언어 문법이다. 그러나 어떤 경우에는 goto를 쓰기가 힘든 경우가 존재한다. 이 경우가 언제인지 기술하고 해당하는 경우,**

문제를 어떤식으로 해결 해야 하는지 프로그래밍 해보시오.

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

시스템으로부터 할당받은 파일이나 소켓을 대표하는 정수를 의미 또한 표준 입력과, 표준출력도 파일 디스크립터로 표현 가능, 이들은 프로그램이 시작되자마자 기본적으로 열리고, 종료 시에 자동적으로 닫히게 된다.
표준입력 0 , 표준출력 1, 표준에러 2 따라서 파일디스크립터는 3부터 시작한다.

81. **stat(argv[2], &buf)일때 stat System Call을 통해 채운 buf.st_mode의 값에 대해 기술하시오.**

buf.st_mode에는 리눅스 커널 inode의 i_mode와 같은 값이 들어가 있다.
파일의 종류 4비트와 setuid, setgid, sticky bit, 그리고 rwx가 3개씩 존재한다.

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

자신이 실제 메모리 어디에 위치하는지에 대한 정보가 필요하고, 또 자신이 하드 디스크의 어떤 파일이 메모리에 로드 되어 프로세스가 되었는지의 정보가 필요하며 마지막으로 프로세스들 간에 **signal**을 주고 받을 필요가 있기 때문에 **signal**이 필요하다.

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

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

int main(int argc, char *argv[])
{
	if(argc != 2)
	{
		printf("Usage: exe dir_name\n");
		exit(-1);
	}
```

```
    mkdir(argv[1], 0755);

    return 0;
}
```

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

85. 랜덤한 이름을 가지도록 디렉토리 3개를 만들고 각각의 디렉토리에 5 ~ 10개 사이의 랜덤한 이름(길이도 랜덤)을 가지도록 파일을 만들어보자! (너무 길면 힘드니까 적당한 크기로 잡도록 함)

86. 85 번까지 진행된 상태에서 모든 디렉토리를 순회하며 3 개의 디렉토리와 그 안의 모든 파일들의 이름 중 a, b, c 가 1개라도 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.

CPU 가상화, 메모리 가상화, I/O 가상화

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.

여러개의 쓰레드에서 동시에 사용하면 안돼는 구간.

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

user app - fork() 를 구동 시키면 → c liberary 가 돌아간다. fork (커널개발자가 만든것) 에 어셈블리 코드가 들어간다. 그러면 int 0x80 으로 적혀 있다. 인터럽트 128번을 호출하라는 뜻. 인터럽트 128번을 호출하는 순간 제어권이 커널로 된다. (sys_call이 보임) 시스템 콜 처리를 하는데, Sys_ 붙은 애들을 모두 처리하게 된다. sys_clone, sys_vfork, sys_fork은 모두 do_fork에 연결되어 있고, 이들은 모두 커널 쓰레드를 생성한다.
= 결론적으로 vfork,fork, clone이든 뭐든 커널 내부에서는 커널 쓰레드로 관리한다는 것이고, 이들을 구동 시키는것은 옵션의 차이를 둔 do_fork이다. fork()와 clone() 둘 모두 커널 입장에서는 모두 태스크를 생성하기 때문이다. fork는 비교적 부모 태스크와 덜 공유 하는 태스크이고, clone()으로 생성되는 태스크는 비교적 부모 태스크와 많이 공유하는 태스크이다. 즉, do_fork()를 호출할 때 이 함수의 인자로 부모 태스크와 얼마나 공유할 지를 정해 줌으로써 fork()와 clone()함수 둘 다를 지원 할 수 있는 것이다.

95. **리눅스 커널의 arch 디렉토리에 대해서 설명하시오.**

리눅스 커널 기능중 하드웨어 종속적인 부분들이 구현된 디렉터리이다. 이디렉터리는 cpu의 타입에 따라 하위 디렉터리로 다시 구분된다.

96. **95 번 문제에서 arm 디렉토리 내부에 대해 설명하도록 하시오.**

ARM 은 하위 호환이 안되고 다양한 반도체 벤더들이 개발을 하고 있기 때문에 해당 디렉토리에 들어가면 회사별 주요 제품들의 이름이 보이는 것을 확인할 수 있다.

97. **리눅스 커널 arch 디렉토리에서 c6x 가 무엇인지 기술하시오.**

TI DSP 에 해당하는 Architecture

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

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

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.

저번 시험을 통해 다짐했던 것들을 또 같은 실수로 반복했습니다. 오늘 시험을 통해 느꼈습니다. 정말 열심히 안했구나를 느꼈습니다. 하루 수업을 못 따라갔으면 그 날 복습으로 따라갔어야 했습니다. 하루가 모여서 정말 산처럼 쌓이니까 더 못하게 됬습니다. 앞으로 정말 주말에 하겠다는 생각하지말고, 하루하루 그날 배운것 열심히 따라가도록 하겠습니다. 이번 쉬는 것을 계기로 여태까지 배운 c언어, 어셈블리, 자료구조, 시스템프로그래밍, 커널 복습 열심히 하도록 하겠습니다. 다음은 없다는 것을 느꼈습니다. 열심히 하겠습니다.