1.파이프 통신을 구현하고 c type.c라고 입력할 경우

현재 위치의 디렉토리에 type.c 파일을 생성하도록 프로그래밍하시오.

```c
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
void create_file(void);
int main(void){
    int fd, ret;
    char buf[1024];
    fd = open("myfifo",O_RDWR);
//    fcntl(0, F_SETFL, O_NONBLOCK);
//    fcntl(fd, F_SETFL, O_NONBLOCK);
    int i = 0;
    for(i = 0;;i++){
    /*
        if((ret = read(0,buf,sizeof(buf))) > 0){
            buf[ret - 1] = 0;
            printf("Keyboard Input : [%s]\n",buf);
        }
    */
        if((ret = read(fd,buf,sizeof(buf))) > 0){
            buf[ret - 1] = 0;
            printf("Pipe Input : [%s]\n",buf);
        }
        if(!strcmp(buf,"c type.c"))
            create_file();
    }
    close(fd);
    return 0;
}void create_file(void){
    int fd;
    if((fd = open("type.c",O_CREAT|O_TRUNC,0644)) > 0)
        printf("type.c created!\n");
    else
        printf("create error\n");
    close(fd);
}
```



```
howard@ubuntu:~/HomeworkBackup/20th$ ./a.out
Pipe Input : [test message]
Pipe Input : [1]
Pipe Input : [2]
Pipe Input : [3]
Pipe Input : [c type.c]
type.c created!
```

```
howard@ubuntu:~/HomeworkBackup/20th$ cat > myfifo
test message
1
2
3
c type.c
```

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.

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

```c
#include <stdbool.h>
#include <setjmp.h>

jmp_buf env;
unsigned int clap = 0;
unsigned int count = 1;

void time_handler(int signo);
bool chk_right(int count, int input);
void int_handler(int signo);
int chk_369_num(int count);
void waits(void);
int main(void){
    //max user's input 999
    char input[4] = {'\0',};
    int usr, clap, ret;
    while(1){
redo:
        signal(SIGINT, int_handler);
        signal(SIGALRM,time_handler);
        alarm(3);
        //user input num
        system("clear");
        printf("num -> %d \n",count);
        clap = chk_369_num(count);
        if(clap)
            goto wait;
        read(0, input, sizeof(input));
        usr = atoi(input);
#if 0
    if(clap != 0){
            printf("You should've claped %d times\n",clap);
            exit(1);
        }
#endif

        chk_right(count,usr);
        count++;
    }
wait:
    while(1){
        pause();
        count++;
        goto redo;
    }

}
void time_handler(int signo){
    printf("3 sec flew, exit! :(\n");
    exit(1);
}
bool chk_right(int count, int input){
    if(count - input){
        printf("%d != %d , exit! :(",count,input);
        exit(1);
    }
    if(chk_369_num(count)){
        printf("should've claped!!");
        exit(1);
```

```
64      }
65      return true;
66  }
67  void int_handler(int signo){
68      clap++;
69      int chk = chk_369_num(count);
70      if(chk < clap){
71          printf("chk : %d, clap : %d\n",chk,clap);
72          printf("wrong clap!\n");
73          exit(1);
74      }
75      if(chk == clap)
76          clap = 0;
77      else
78          waits();
79  }
80  void waits(void){
81      printf("clap more\n");
82
83      signal(SIGINT, int_handler);
84      pause();
85  }
86  int chk_369_num(int count){
87      int chk = 0;
88      int buf;
89      buf = count / 100;
90      if(((count/100) % 3 == 0)&& (count / 100 != 0))
91          chk++;
92      if((((count%100) / 10) % 3 == 0)&&((count % 100) / 10 != 0))
93          chk++;
94      if(((count % 10) % 3 == 0)&&(count % 10 != 0))
95          chk++;
96      return chk;
97  }
98
```



```
howard@ubuntu: ~/HomeworkBackup/test2 ref
num -> 5
3 sec flew, exit! :(
```

3.리눅스 커널은 운영체제(OS)다.

OS가 관리해야 하는 제일 중요한 5가지에 대해 기술하시오.

file system manager
memory manager
task manager
device manager
network manager

4.Unix 계열의 모든 OS는 모든 것을 무엇으로 관리하는가 ?

하다.

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**

*** 사용자 임의대로 재구성이 가능하다.**
오픈소스이므로, 필요한 기능이 있다면 직접 추가해서 사용할 수도 있따.
*** 열약한 환경에서도 HW 자원을 적절히 활용하여 동작한다.**
많은 기법들을 통해 메모리를 효율적으로 활용한다.
*** 커널의 크기가 작다.**
monolithic과 micro 커널 두 개를 다 지원하는 하이브리트 형태이므로, 필요하다면 경량화가 가능하다.
*** 완벽한 멀티유저, 멀티태스킹 시스템**
실시간 스케줄링과 일반 스케줄링 방식을 지원하므로 멀티태스킹이 가능하다.
*** 빠른 업그레이드**
오픈소스이고, 수많은 개발자들이 개발에 참여하므로 업그레이드 속도가 빠르다, 하지만 리누스가 변수이름 바꾸는건 짜증난다.
*** 풍부한 소프트웨어**
GNU를 통한 풍부한 소프트웨어를 지원한다.

6.32 bit System에서 User와 Kernel의 Space 구간을 적으시오.

32 bit System.
커널 :3~4GB, 총 1GB
유저 : 0~3GB, 총 3GB

64 bit System.
커널 : 8~16EB, 총 8EB
유저 : 0~8EB, 총 8EB

7.Page Fault 가 발생했을때
운영체제가 어떻게 동작하는지 기술하시오

가상 메모리의 주소를 통해 물리 메모리에 접근할 때, paging 기법을 통해 접근한다. 처음 메모리에 접근할 때는 해당 물리 메모리가 할당되어 있지 않기 때문에, page fault 가 발생한다. 이 때, 권한이 커널 모드인 경우, page fault handler를 실행하여 해당 물리 메모리를 할당하여 다시 접근한다. 권한이 유저 모드일 경우, segmentation fault를 발생하여 태스크를 종료한다.

8.리눅스 실행 파일 포맷이 무엇인지 적으시오.

실행 파일 포맷: ELF
추가로 디버깅 파일 포맷 : DWARF

9.프로세스와 스레드를 구별하는 방법에 대해 기술하시오

스레드 라는 것은, 동일한 tgid를 가진 프로세스 내의 태스크를 말한다.
따라서, getpid()==gettgid() 일 경우 프로세스이고,
getpid() != gettgid() 일 경우 쓰레드이다.

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

```
task_struct{
   mm_struct{
      vm_area_struct{
            :
      }
   start_code, end_code
   start_brk, end_brk
   start_data, end_data
   start_stack
     :
  }
```

:
}
등을 생성하여 가상 메모리 레이아웃을 생성한다.

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

12. Memory Management 입장에서 Process와 Thread의 핵심적인 차이점은 무엇인가 ?

process는 process간에 가상 메모리 레이아웃이 독립적이기 때문에 메모리 공유를 하지 않고, Thread는 같은 tgid상의 task들끼리 stack을 제외한 메모리 레이아웃을 공유하기 때문에 critical section이 존재한다. 이는 IPC를 하는 데 있어 process보다 효율적인 면이 있다.

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

CPU의 문맥(레지스터들)을 저장하기 위한 구조로, context switching시 사용된다.

14.리눅스 커널의 스케쥴링 정책중 Deadline 방식에 대해 기술하시오.

실시간 태스크 스케줄링 정책 중 하나이다.
프로세스들이 어떤 시간 내에 반드시 수행해야 하는 일의 양이 있는 것이다.
30 frame / sec로 출력되는 동영상을 생각해보자. 이 동영상 프로세스가 1 frame을 출력하는 데 0.02 sec가 소요된다고 하자. 총 30frame을 출력하는 데에는 0.6sec가 필요할 것이다. 현재 시간이 0.4 sec가 넘었다면, 남은 시간동안 30 frame을 모두 출력하지 못하게 되기 때문에 문제가 생긴다. 즉, 모든 프로세스가 현재시간 + runtime < Deadline을 만족하도록 스케줄링 하는 방식이다.
구현 방법은 비교적 간단하다. BST 자료구조를 보면, 가장 작은 값은 항상 왼쪽 최하위 노드에 있다. Deadline 까지 여유 시간을 트리에 넣어, 가장 왼쪽 노드를 읽어 그 프로세스에게 우선권을 주면 된다. 스케줄링인 만큼, 삽입과 삭제가 둘 다 매우 빈번하기 때문에 BST나 AVL Tree보다는 RB Tree가 더욱 효과적이다.

15.TASK_INTURRUPTIBLE과 TASK_UNINTERRUPTIBLE은 왜 필요한지 기술하시오.

인터럽트 루틴 처리 중, 다시 인터럽트가 발생하는 것을 방지하기 위해 필요하다.

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

O(N)은 어떤 알고리즘을 수행하는 데 자료의 수에 비례하여 시간이 필요한 것을 말한다. 예를 들면, for문 등이 그러하다. O(C) 알고리즘은 자료의 수에 상관 없이 특정 시간이 소요되는 것을 말한다. stack의 pop 알고리즘이 그러하다.
스케줄링하는 데 있어, 우선순위를 검색하는 데 O(N)의 시간이 필요하다면, 스케줄링시 필요한 시간을 정확히 예측하기가 어렵다. O(1)시간의 경우, 태스크 개수가 많아질 때 속도도 빠르고 필요한 시간을 예측할 수 있기 때문에 더욱 좋다.

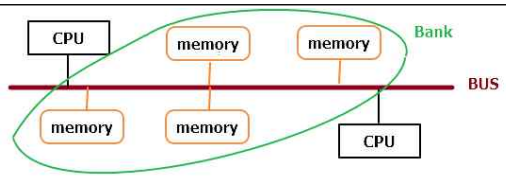
17.현재 4개의 CPU(0, 1, 2, 3)가 있고 각각의 RQ에는 1, 2개의 프로세스가 위치한다.
이 경우 2번 CPU에 있는 부모가 fork()를 수행하여 Task를 만들어냈다.
이 Task는 어디에 위치하는 것이 좋을까 ?
그리고 그 이유를 적으시오.

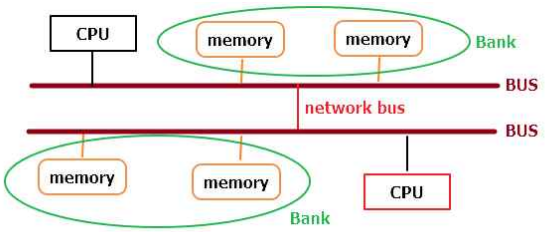
2번 CPU의 RQ에 위치하는 것이 좋다. fork를 통해 만든 자식 프로세스는 부모 프로세스와 매우 유사한 메모리 구조를 갖게 되는데, 이는 cache의 활용성을 높혀주기 때문이다.

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.

이미 매우 많은 프로세스가 존재한다면, cache를 활용할 수 있다고 보기 어렵기 때문에 프로세스 수를 분산하여 2번 CPU의 RQ에 위치하는 것이 좋다

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





UMA NUMA

CPU가 여러 개 있을 때, 메모리들이 CPU에 접근하는 데 여러 개의 버스가 있을 수 있다.
버스를 여러개 거치고 접근할 때, 속도가 느려진다. Bank라는 것은 cpu에 대해 같은 접근속도를 가진 메모리들의 집합을 의미한다. Bank가 1개일 때 UMA 구조라고 하고, Bank가 여러 개일 때 NUMA 구조라고 한다.
Bank는 커널 상에서 Node로 관리된다. 이 node는 pg_data_t 구조체를 통해 표현된다.

```
typedefstruct pglist_data {
        struct zone node_zones[MAX_NR_ZONES];
        struct zonelist node_zonelists[MAX_ZONELISTS];
        int nr_zones;
#ifdef CONFIG_FLAT_NODE_MEM_MAP
        struct page *node_mem_map;
#ifdef CONFIG_PAGE_EXTENSION
        struct page_ext *node_page_ext;
#endif
#endif
#ifndef CONFIG_NO_BOOTMEM
        struct bootmem_data *bdata;
#endif
#ifdef CONFIG_MEMORY_HOTPLUG
        spinlock_t node_size_lock;
#endif
        unsignedlong node_start_pfn;
        unsignedlong node_present_pages;
        unsignedlong node_spanned_pages;

        int node_id;
        wait_queue_head_t kswapd_wait;
        wait_queue_head_t pfmemalloc_wait;
        struct task_struct *kswapd;

        int kswapd_max_order;
        enum zone_type classzone_idx;
#ifdef CONFIG_NUMA_BALANCING

        spinlock_t numabalancing_migrate_lock;

        unsignedlong numabalancing_migrate_next_window;
```

```
        unsignedlong numabalancing_migrate_nr_pages;
#endif
#ifdef CONFIG_DEFERRED_STRUCT_PAGE_INIT
```

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

Static Priority : 0 ~ 99
Dynamic Priority : 100 ~ 139

21.ZONE_HIGHMEM 에 대해 아는대로 기술하시오.

Zone을 통한 메모리 접근 방식에는 크게 3가지가 있다.
ZONE_DMA, ZONE_NORMAL, ZONE_HIGHMEM.
x86 (32bit) 시스템에서 가상 메모리의 커널 영역은 1GB이기 때문에 물리 메모리가 그보다 클 경우, 1:1 mapping으로는 모든 메모리를 참조할 수가 없다. ZONE_DMA는 특수한 영역으로, 실시간 접근이 필요한 경우 사용하고, ZONE_NORMAL 방식은 1:1mapping하는 방식으로 최대한 많이 활용한다.(16~896M) 1:1mapping으로 커버가 불가능한 영역은 남아있는 128M만큼의 공간으로 ZONE_HIGHMEM을 통해 메모리 접근 테이블을 만들어 테이블을 참조하여 접근한다. 테이블을 참조하여 접근하는 만큼, 속도 면에서는 1:1방식보다 좋지 않다.

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

페이지 프레임, 4kbyte
struct page{ }

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

struct zone{ } 안에 struct free_area[MAX_ORDER]변수를 통해 관리한다.
메모리를 $4K*2^n$ 단위로 할당 및 해제한다. 예를 들면, 5K만큼 할당 요청이 들어오면 페이지 2개, 8K만큼의 공간을 할당해주고 병합한다. 필요한 공간 중 가장 적게 연속적으로 남은 메모리 공간을 할당해 주므로, 외부 단편화를 어느정도 방지할 수 있다. 한 메모리 공간을 계속해서 할당 및 해제를 할 시, 페이지 병합 및 분할하는 데 오버헤드가 발생할 수 있다. 이를 방지하기 위한 방법이 Lazy Buddy 알고리즘이다.

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

페이지의 단위는 4K이다. 하지만, 자료구조를 만든다고 할 때, malloc을 통해 할당하는 메모리는 12byte정도가 된다. 나머지 공간이 매우 비효율적으로 남는다. 이러한 것을 내부 단편화라고 한다. 내부 단편화를 최소화 하기 위해, 4KB만큼의 공간을 미리 할당받는다. 이 공간을 다시 64Byte 크기로 분할한다. 크기가 작은 메모리의 할당 요청이 들어오면, 이 공간을 활용한다.

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)

```
task_struct{
    mm_struct{
      start_code, end_code;
      start_brk, end_brk;
      start_data, end_data;
      start_stack;
```

의 7개의 변수, 28byte를 통해 표현된다.

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)

task_struct{
mm_struct{
vm_area_struct{
의 vm_area_struct를 통해 관리한다.

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 ?

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.

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

task_struct : mm_strcut : pgd 변수를 통해 page global direct table의 시작 주소를 저장한다. 이 주소를 기반으로 MMU가 물리 메모리 주소로 mapping한다.

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

논리적 주소와 물리적 주소의 mapping
CPU의 L2 cache 관리

31.하드디스크의 최소 단위를 무엇이라 부르고 그 크기는 얼마인가 ?

Disk Block 이라고 부르고 보통 4KB이다.

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 ?

루트 파일 시스템의 위치 저장, 파일 시스템의 전체적인 정보 저장.

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

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

하드웨어적인 인터럽트인 외부 인터럽트와 소프트웨어적인 인터럽트인 내부 인터럽트가 있다. 외부 인터럽트에는 키보드, 마우스 등등의 동작이 있고, 내부 인터럽트에는 fault, trap, abort 등이 있다.

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

내부 인터럽트에는 fault, trap, abort 등이 있다.

38.앞선 문제에서 분류한 녀석들의 특징에 대해 기술하시오.

내부 인터럽트에는 fault, trap, abort 등이 있다.
fault 발생시에는 핸들러가 작동한 후 다시 수행해야 하기에, eip(rip) 레지스터에 fault를 일으킨 명령어를 저장한다.
trap은 trap을 일으킨 명령어의 다음 명령어를 저장 하고, 핸들러 동작 후 호출한다.
abort는 심각한 에러인 경우로, 그냥 태스크를 종료한다.

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

모니터를 3개 쓰는 경우, 3개의 디바이스는 같은 주번호를 갖고, 다른 부번호를 갖는다. 이

를 통해 관리한다.

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

41.38에서 User Space에서 System Call 번호를 전달한다.
Intel Machine에서는 이를 어디에 저장하는가 ?
또한 ARM Machine에서는 이를 어디에 저장하는가 ?

intel은 eax 레지스터에 System Call 번호를 저장한다.

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

pgd_t* pgd;

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

44.커널 내부에서 메모리 할당이 필요한 이유는 무엇인가 ?

45.메모리를 불연속적으로 할당하는 기법은 무엇인가 ?

kmalloc()

46.메모리를 연속적으로 할당하는 기법은 무엇인가 ?

vmalloc()

47. Mutex 와 Semaphore 의 차이점을 기술하시오.

Semaphore에는 최대 key값, 동시에 접근할 수 있는 task의 개수가 있다. 동시에 접근할 수 있는 task의 개수가 1개인 것이 mutex이다.

48.module_init() 함수 호출은 언제 이루어지는가 ?

insmod 명령어로 모듈 삽입시

49.module_exit() 함수 호출은 언제 이루어지는가 ?

rmmod 명령어로 모듈 제거시

50.thread_union 에 대해 기술하시오.

thread_union 안에는 thread_info 구조체와 kernel stack 변수가 있다. thread_info 구조체 안에는 스케줄링 flags 변수, task_struct 포인터, cpu_context_save 구조체 등이 있다.

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)

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.

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.

kmalloc을 통해 메모리를 할당하면, 메모리 중간중간에 빈 공간을 활용하기 때문에 공간을 효율적으로 활용할 수 있다. vmalloc을 통해 메모리를 할당할 경우, 공간 효율은 상대적으로 좋지 않지만, 연속된 메모리 공간을 할당받기 때문에 cache를 통한 접근이 가능해지므로 성능상 좋아질 여지가 있다.

54. Character Device Driver를 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.

명령어를 순차적으로 실행할 경우, 비교적 비효율적인 순서가 있을 수 있다. 명령어끼리 독립적인 처리가 가능한 경우, 순서를 바꿔도 문제가 없는데 순서를 바꿈으로서 효율이 좋다면 이런 명령어들의 나열에는 OoO를 사용하는 것이 좋다.

56.Compiler의 Instruction Scheduling에 대해 기술하라.

57.CISC Architecture와 RISC Architecture에 대한 차이점을 기술하라.

CISC Architecture에는 많은 수의 명령어가 있다. 그 중 실제로 자주 쓰이는 명령어가 몇개 안된다는 점을 착안해서 만든 구조가 RISC 구조이다. RISC는 자주 쓰이는 명령어로 구성되어 있고, 필요하다면 명령어들의 조합으로 다른 명령어를 사용한다. CISC 구조의 명령어들의 크기는 가변적이고, RISC 구조의 명령어들의 크기는 일정하다

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 어느정도의 Instruction Scheduling을 HW의 힘만으로 수행할 수 있다.
이러한 것을 무엇이라 부르는가 ?

59.Pipeline이 깨지는 경우에 대해 자세히 기술하시오

branch 명령어를 execute할 경우, 파이프라인이 깨진다.
branch 명령어가 execute될 때, 그 외의 다른 명령어들(fetch, discode상에 올라가 있는)이 버려지기 때문에, 깨지게 된다.

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 ?

레지스터

61.Intel의 Hyper Threading 기술에 대해 상세히 기술하시오.

cpu하나에 가상 cpu 회로를 만들어 실제로 2개처럼 동작하게 하는 기술이다.
간단하게 말하면, cpu가 사용하고 있지 않는 레지스터를 다른 스레드에서 활용하여 최대한의 cpu 성능을 내기 위한 방법이다. 그렇기 때문에, 프로세스에 따라 고성능을 낼 수도 있고, 아닐 수도 있다. 게임 같은 경우, 같은 연산을 반복하는 경우가 많이 때문에 Hyper Threading으로 인한 이득은 보기 힘들다.

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.

63.파일의 종류를 확인하는 프로그램을 작성하시도록 하시오.

```
1   #include <stdio.h>
2   #include <unistd.h>
3   #include <dirent.h>
4   #include <stdlib.h>
5   #include <sys/stat.h>
6   void chk_file(struct stat buf);
7   int main(int argc, char** argv){
8       struct stat buf;
```

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

    stat(argv[1], &buf);
    chk_file(buf);

    return 0;
}
void chk_file(struct stat buf){
    if(S_ISDIR(buf.st_mode))
        printf("directory\n");
    if(S_ISREG(buf.st_mode))
        printf("regular\n");
    if(S_ISFIFO(buf.st_mode))
        printf("pipe\n");
    if(S_ISSOCK(buf.st_mode))
        printf("socket\n");
    if(S_ISCHR(buf.st_mode))
        printf("character device\n");
    if(S_ISBLK(buf.st_mode))
        printf("block device\n");
}
```



```
howard@ubuntu:~/HomeworkBackup/test2_ref$ ./a.out 7.c
regular
howard@ubuntu:~/HomeworkBackup/test2_ref$ ./a.out pvge
directory
howard@ubuntu:~/HomeworkBackup/test2_ref$
```

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

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
typedef struct sockaddr_in si;
typedef struct sockaddr * sap;

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

}

int main(int argc,char** argv){
    int serv_sock;
    int clnt_sock;
    int ret;

    si serv_addr;
    si clnt_addr;
    socklen_t clnt_addr_size;
    char* send = "Hello";
    char recv[32] = {0,};
    if(argc != 2){
        printf("use : %s <port>\n",argv[0]);
```

```
            exit(1);
        }
        //sock file descriptor
        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]));
        //bind : server ip address setting
        if(bind(serv_sock, (sap)&serv_addr, sizeof(serv_addr)) == -1)
            err_handler("bind() error");
        //listen : max people , wait client.

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

        clnt_addr_size = sizeof(clnt_addr);
        //client permit.
        clnt_sock = accept(serv_sock, (struct sockaddr *)&clnt_addr,
&clnt_addr_size);
        if(clnt_sock == -1)
            err_handler("accept() error");
        while(1){
            write(clnt_sock, send, sizeof(send));
            ret = read(clnt_sock, recv, sizeof(recv));
            printf("message from clnt : %s\n",recv);
            sleep(1);
        }
        close(clnt_sock);
        close(serv_sock);

}
```


Client.c

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

typedef struct sockaddr_in si;
typedef struct sockaddr * sap;

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

int main(int argc,char* argv[]){

    int sock;
    int str_len;
    char msg[32];
    char* send = "Hi";
```

```
23      si serv_addr;
24      if(argc != 3){
25          printf("err\n");
26          exit(1);
27      }
28
29      sock = socket(PF_INET, SOCK_STREAM, 0);
30
31      if(sock == -1)
32          err_handler("socket() error");
33
34      memset(&serv_addr, 0 , sizeof(serv_addr));
35      serv_addr.sin_family = AF_INET;
36      serv_addr.sin_addr.s_addr = inet_addr(argv[1]);
37      serv_addr.sin_port = htons(atoi(argv[2]));
38
39      if(connect(sock, (sap)&serv_addr, sizeof(serv_addr)) == -1)
40          err_handler("connect() error");
41      while(1){
42          str_len = read(sock, msg, sizeof(msg) -1 );
43          printf("msg from serv : %s\n",msg);
44
45          write(sock, send, strlen(send));
46          sleep(1);
47      }
48      close(sock);
49      return 0;
50
51  }
52
```



```
howard@ubuntu:~/HomeworkBackup/test2_ref$ ./serv 7777
message from clnt : Hi
message from clnt : Hi
message from clnt : Hi
message from clnt : Hi
message from clnt : Hi
message from clnt : Hi
message from clnt : Hi
message from clnt : Hi
message from clnt : Hi
message from clnt : Hi
message from clnt : Hi
message from clnt : Hi
message from clnt : Hi
message from clnt : Hi
message from clnt : Hi
```

```
howard@ubuntu:~/HomeworkBackup/test2_ref$ ./clnt 127.0.0.1 7777
msg from serv : Hello
msg from serv : Hello
msg from serv : Hello
msg from serv : Hello
msg from serv : Hello
msg from serv : Hello
msg from serv : Hello
msg from serv : Hello
msg from serv : Hello
msg from serv : Hello
msg from serv : Hello
msg from serv : Hello
msg from serv : Hello
msg from serv : Hello
msg from serv : Hello
```

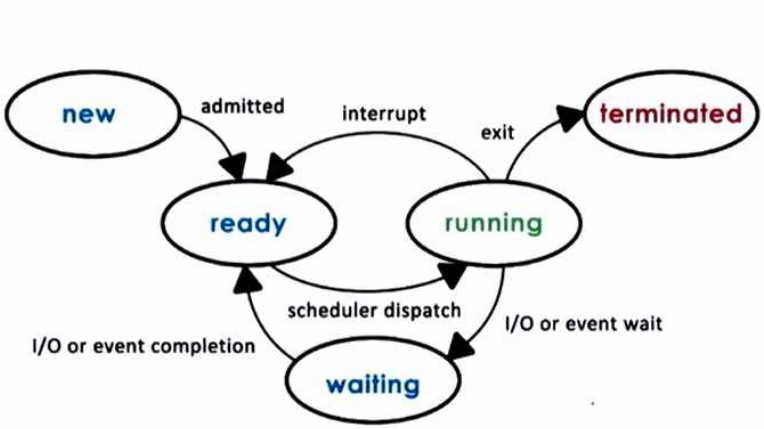
65.Shared Memory를 통해 임의의 파일을 읽고 그 내용을 공유하도록 프로그래밍하시오

66.자신이 사용하는 리눅스 커널의 버전을 확인해보고 https://mirrors.edge.kernel.org/pub/linux/kernel/v4.x/$(자신의 버전).tar.gz 를 다운받아보고

이 압축된 파일을 압축 해제하고 task_struct 를 찾아보도록 한다.
일련의 과정을 기술하면 된다.

uname -r을 통해 현재 리눅스 버전을 확인한다.
wget https://mirrors.edge.kernel.org/pub/linux/kernel/v4.x/$(자신의 버전).tar.gz 명령어로 커널 압축파일을 다운로드한다.
tar -zxvf (버전).tar.gz 명령어를 통해 압축을 해제한다.
커널 디렉토리 안에 들어가서 ctags -R 명령어를 통해 태그 파일을 생성한다.(vimrc,mkcscope.sh 설정은 이미 했다는 가정 하에.)
태그 검색을 통해 task_struct를 검색한다 : vi -t task_struct

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



일단 프로세스가 실행되면, task_struct 가 생성되고 vm_area_struct가 생성되어 가상 메모리 레이아웃, 즉 Segment들을 가지게 된다. 이 segment들의 주소를 CPU 내부에 있는 MMU를 통한 Paging기법을 통해 실제 물리메모리에 mapping된다. 메모리에 올라간 프로세스들은 일단 Run Queue(혹은 Ready Queue)에 저장된다. Run Queue에 있는 모든 프로세스들은 각각의 우선순위에 따라 OS가 스케쥴링을 통해 1개씩 CPU에 올려 실행상태가 된다.(싱글코어의 경우)

실행중인 프로세스가 A,B 두 개 있다고 하자. A프로세스가 running 상태라면, B프로세스는 ready상태일 것이다. wait queue에는 인터럽트 등을 통해 정지중인 프로세스가 이동한다.

태스크 스케쥴링 방식은 크게 2가지가 있다. 실시간 태스크 스케줄링(0~99) , 일반 태스크 스케줄링(100~139), 숫자는 우선순위. 실시간 태스크 스케줄링 방식의 정책에는 크게 FIFO, RR, DEADLINE 방식이 있고, 일반 태스크 스케줄링 방식의 정책에는 리눅스는 현재 CFS 정책을 채택하고 있다.

68. 현재 삽입된 디바이스 드라이버의 리스트를 보는 명령어는 무엇인가 ?

lsmod

69. System Call Mechanism에 대해 기술하시오.

시스템 콜을 간단히 정의하자면, 인터럽트 핸들러라고 할 수 있다.인터럽트에는 크게 2가지 종류가 있다. 키보드, 마우스 등 하드웨어로 인한 비동기적인 외부 인터럽트, 현재 수행중인 태스크와 관련있는 동기적인 내부 인터럽드 등이 있다. 외부 인터럽트의 예시를 들자면, 마우스를 움직이면 마우스 포인터가 움직이는 것, 키보드를 입력하면 글자가 출력되는 것 등 모두 외부 인터럽트이다. 내부 인터럽트는 소프트웨어의 예외처리라고도 볼 수 있다. fault계열의 오류, devide_by_zero,시스템 콜 등이 있다. 즉, 시스템 콜이라는 것은 소프트웨어 인터럽트를 말한다고도 할 수 있다.

시스템 콜이 호출되면, 현재 실행중인 프로세스의 정보는 context switching를 통해 저장된다. IDT 또는 IVT를 읽고, 입력받은 번호에 해당하는 인터럽트가 발생한다. 0~31번은 내부 인터럽트, 즉, 트랩 영역이고, 32~255까지는 외부 인터럽트 영역이다. 이 중 128번은

System call이다. 0x80이 들어오면 sys_call_table을 읽은 후 eax 레지스터에 있는 값을 읽어와서 시스템 콜을 실행한다.

70. Process와 VM과의 관계에 대해 기술하시오.

모든 프로세스는 메모리에 올라가는 순간(실행되는 순간) task_struct를 생성한다. 이 안에는 vm_area_struct가 있다. 그 안에는 mm_struct가 있어 가상 메모리 레이아웃을 7개의 변수, 28byte로 표현할 수 있다.즉, 모든 물리 메모리를 한번에 잡는 것이 아닌, 필요한 만큼만 할당할 수 있어 보다 효과적으로 메모리를 관리할 수 있다. VM(Virtual Machine)도 비슷한 개념에서 출발한다. 물리적으로 떨어져 있는 시스템들이 하나의 시스템에 접근함으로써, 자원의 낭비 없이 이용률을 높이고, 통합 관리가 가능하므로 관리 비용이 줄어든다.

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

```
1   #include <stdio.h>
2   #include <fcntl.h>
3   #include <unistd.h>
4   #include <stdlib.h>
5   #define BUF_SIZE 1024
6   void err_handler(char* msg);
7   int chk_lines(char* file);
8   void print_five_lines(int nr, char* file);
9   int main(int argc, char** argv){
10      if(argc != 2){
11          printf("use : <%s> <filename>\n",argv[0]);
12          exit(-1);
13      }
14      int fd, nr = 0;
15      char buf[BUF_SIZE];
16
17      nr = chk_lines(argv[1]);
18
19      print_five_lines(nr, argv[1]);
20  }
21  void err_handler(char* msg){
22      fputs(msg,stderr);
23      fputc('\n',stderr);
24      exit(-1);
25  }
26  int chk_lines(char* file){
27      int nr = 0, fd;
28      char buf[32];
29      if((fd = open(file,O_RDONLY,0644))< 0)
30          err_handler("open error");
31      while(read(fd,buf,1) > 0)
32          if(buf[0] == '\n')
33              nr++;
34      close(fd);
35      return nr;
36  }
37  void print_five_lines(int nr, char* file){
38      int i, fd, ret, cur_nr = 0;
39      char buf[32];
40      if((fd = open(file,O_RDONLY,0644))< 0)
41          err_handler("open error");
42      printf("upper 5 lines\n");
43      while((ret = read(fd,buf,1)) > 0){
44          if(cur_nr < 5)
45              write(1, buf, ret);
46          else if(nr - cur_nr == 5)
```

```
47                  break;
48
49              if(buf[0] == '\n')
50                  cur_nr++;
51          }
52
53          printf("lower 5 lines\n");
54          while((ret = read(fd,buf,1)) > 0)
55              write(1, buf, ret);
56
57      }
58
```



```
howard@ubuntu:~/HomeworkBackup/test2_ref$ ./a.out test.txt
upper 5 lines
1111111111
22222222222
3333333333333
44444444444
5555555555
lower 5 lines
99999999
101010101010
11111111
12121212
13131313
```

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

```
1   #include <stdio.h>
2   #include <dirent.h>
3   #include <unistd.h>
4
5   int main(int argc,char** argv){
6       DIR* dp;
7       struct dirent* p;
8       int i = 1;
9       //select the directory to be listed
10      if(argc == 2)
11          dp = opendir("argv[1]");
12      else
13          dp = opendir(".");
14
15      while(p = readdir(dp)){
16          printf("%-16s",p->d_name);
17          if(!(i%5))
18              printf("\n");
19          i++;
20      }
21      printf("\n");
22  }
23
```



```
howard@ubuntu:~/HomeworkBackup/test2_ref$ ./a.out
1-6.c           aa.c            24.c            1-9.c           72.c
test.txt        7.c             1-1.c           9.c             a.out
..              shm.h           21.c            27.c            16.c
1-7.c           .               6.c             shmlib.c
```

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

74.C.O.W Architecture에 대해 기술하시오.

copy on write를 뜻한다. 본래 fork함수를 호출하게 되면, 자식 프로세스는 부모 프로세스의 모든 가상 메모리 레이아웃을 복제한다. COW라는 것은 미리 모든 메모리를 할당받지 않고, 실제로 쓰기 동작이 일어날 때 할당을 받는 구조를 말한다. 이런 기법을 사용하는 이

유는, 자식 프로세스가 부모 프로세스의 모든 메모리를 필요로 하지 않을 수도 있기 때문이다. 예를 들면, fork를 한 후, 자식 프로세스에서 exec을 하게 되면 부모 프로세스의 메모리를 복제하는 것은 불필요한 오버헤드를 발생시킨다.

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

Blocking 연산은 동기 방식이고, Non-Blocking 연산은 비동기 방식이다.
어떠한 연산을 하는 데 있어서 순서가 중요하다면 Blocking 방식을, 실시간 처리가 중요하다면 Non-Blocking 방식을 사용한다.

76. 자식이 정상 종료되었는지 비정상 종료되었는지 파악하는 프로그램을 작성하시오.

```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 child_process(void);
int main(void){
    pid_t pid, child_pid;
    int status;
    if((pid = fork()) > 0){
        printf("I'm parent process, waiting child signal\n");
        child_pid = wait(&status);
        if(WIFEXITED(status))
            printf("normal exit : pid(%d), signal(0x%x)\n",child_pid,status&0x7f);
        else
            printf("abnormal exit : pid(%d), signal(0x%x)\n",child_pid,status&0x7f);
    }
    else if(pid == 0)
        child_process();
    else{
        perror("fork()");
        exit(-1);
    }
    return 0;
}
void child_process(void){
    int i;
    sleep(1);
    printf("I'm child, I'll die after 5 sec\n");
    for(i=1;i<=5;i++){
        sleep(1);
        printf("%d sec flew\n",i);
    }
    printf("CHILD DIED\n");
//    abort();
    exit(1);
}
```



```
./howard@ubuntu:~/HomeworkBackup/test2_ref$ ./a.out
I'm parent process, waiting child signal
I'm child, I'll die after 5 sec
1 sec flew
2 sec flew
3 sec flew
4 sec flew
5 sec flew
CHILD DIED
normal exit : pid(3106), signal(0x0)
```

77.데몬 프로세스를 작성하시오.

잠시 동안 데몬이 아니고 영구히 데몬이 되게 하시오.

78.SIGINT는 무시하고 SIGQUIT을 맞으면 죽는 프로그램을 작성하시오.

```c
#include <signal.h>
#include <stdio.h>
void sig_handler(int signo);
int main(void){
	signal(SIGINT, SIG_IGN);
	signal(SIGQUIT, sig_handler);
	while(1);

}
void sig_handler(int signo){
	printf("QUIT!!\n");
	exit(-1);
}
```

```
howard@ubuntu:~/HomeworkBackup/test2_ref$ ./a.out
QUIT!!
```

```
howard    3153  2082 99 21:07 pts/1    00:00:02 ./a.out
howard    3155  3132  0 21:07 pts/19   00:00:00 grep --color=auto a.out
howard@ubuntu:~/HomeworkBackup/test2_ref$ kill -3 3153
```

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.

goto는 굉장히 유용하나, 다른 함수끼리의 이동, 즉, 다른 스택 공간으로의 점프는 불가능하다.이런 경우는 setjmp를 사용한다.

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

jmp_buf env;

void should_be_ignore(void);
int main(void){
	int ret;
	if((ret = setjmp(env)) == 0)
		should_be_ignore();
	else if(ret > 0)
		printf("execute this!\n");
	return 0;
}
void should_be_ignore(void){
	longjmp(env,1);
	printf("This sentence should be ignored(jumped)\n");

}
```



```
howard@ubuntu:~/HomeworkBackup/test2_ref$ ./a.out
execute this!
```

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

리눅스에서는 모든 것은 파일로 관리한다. file, socket, character device, block device, directory 등..
파일 디스크립터는 시스템 영역의 파일을 유저가 문제 없이 접근하기 위해 만든, 참조 변수 같은 것이다.

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

st_mode는 크게 2가지 정보를 가지고 있다. 파일의 형식과 권한 설정이다.
16비트 중 앞의 4 비트는 파일의 형식을 의미하고, 뒤의 12비트는 권한을 의미한다
12비트는 sstrwxrwxrwx로 이루어져 있고, 3쌍의 rwx는 순서대로 유저, 그룹, 제3자 권한을 의미한다.

82.프로세스들은 최소 메모리를 관리하기 위한 mm, 파일 디스크립터인 fd_array, 그리고 signal을 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!

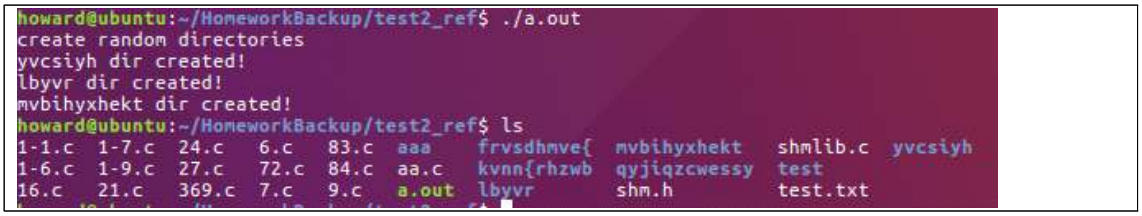
```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
int main(void){
    printf("create \"test\" directory\n");
    mkdir("./test",0644);
    return 0;
}
```



```
howard@ubuntu:~/HomeworkBackup/test2_ref$ ./a.out
create "test" directory
howard@ubuntu:~/HomeworkBackup/test2_ref$ ls
1-1.c  16.c   1-9.c  24.c  369.c  72.c  83.c  aaa   a.out  shmlib.c  test.txt
1-6.c  1-7.c  21.c   27.c  6.c    7.c   9.c   aa.c  shm.h  test
```

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

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <time.h>
#include <stdlib.h>
void make_rand_dir(void);
int main(void){
    printf("create random directories\n");
    int i;
    for(i=0;i<3;i++)
        make_rand_dir();
    return 0;
}
void make_rand_dir(void){
    srand(time(NULL));
    int nl = rand()%10 + 5;
    char* name = (char*)malloc(sizeof(char)*nl);
    int i;
    for(i=0;i<nl;i++)
        name[i] = 97 + rand()%26;
    mkdir(name,0644);
    printf("%s dir created!\n",name);
    sleep(1);
}
```



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 잡도록함)

```c
#include <stdio.h>
#include <unistd.h>
#include <dirent.h>
#include <sys/stat.h>
#include <stdbool.h>
#include <string.h>
#include <time.h>
#include <fcntl.h>
#include <stdlib.h>
void test(char* name);
bool chk_incld_abc(char* name);
int main(void){
    DIR* dp;
    struct dirent* p;
    struct stat buf;
    dp = opendir(".");
    while(p = readdir(dp)){
        stat(p->d_name,&buf);
        if(S_ISDIR(buf.st_mode)&&(p->d_name[0] != '.'))
            test(p->d_name);
    }
}
void test(char* name){
    srand(time(NULL));
    printf("DIR : %s\n",name);
    DIR* dp;
    char* file;
    int nl, i, fd;
    dp = opendir(name);
    chdir(name);
    nl = rand()%5 + 5;
    file = (char*)malloc(sizeof(char)*nl);

    for(i=0;i<nl;i++)
        file[i] = 97 + rand()%26;


    if((fd = creat(file,0644)) < 0)
        printf("create fail\n");
    else
        printf("%s created\n",file);

    sleep(1);
    closedir(dp);
    chdir("..");
}
bool chk_incld_abc(char* name){
    char* chk = NULL;
    if(strchr(name,'a'))
```

```
            return true;
        if(strchr(name,'b'))
            return true;
        if(strchr(name,'c'))
            return true;
        return false;
    }
```



```
DIR : gv
aqsknuaw created
DIR : yvsi
mnohxlurc created
DIR : pvge
llbsv created
```

86.앞선 문제까지 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.

```c
#include <stdio.h>
#include <unistd.h>
#include <dirent.h>
#include <sys/stat.h>
#include <stdbool.h>
#include <string.h>
void test(char* name);
bool chk_incld_abc(char* name);
int main(void){
    DIR* dp;
    struct dirent* p;
    struct stat buf;
    dp = opendir(".");
    while(p = readdir(dp)){
        stat(p->d_name,&buf);
        if(S_ISDIR(buf.st_mode)&&(p->d_name[0] != '.'))
            test(p->d_name);
    }
}
void test(char* name){
    printf("DIR : %s\n",name);
    DIR* dp;
    struct dirent* p;
    struct stat buf;
    dp = opendir(name);
    while(p = readdir(dp)){
        stat(p->d_name, &buf);
        if(!chk_incld_abc(p->d_name))
            continue;
        if(S_ISDIR(buf.st_mode))
            printf(" - Directory : %s\n",p->d_name);
        else
            printf(" - File : %s\n",p->d_name);
    }
    closedir(dp);
}
bool chk_incld_abc(char* name){
    char* chk = NULL;
    if(strchr(name,'a'))
        return true;
    if(strchr(name,'b'))
        return true;
```

```
43          if(strchr(name,'c'))
44              return true;
45          return false;
46      }
47
```



```
DIR : gv
 - File : a.txt
 - File : aqsknuaw
 - Directory : b.txt
DIR : yvsi
 - Directory : aifhrazmg
 - Directory : mnohxlurc
DIR : pvge
 - Directory : llbsv
```

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.

serv.c

```c
#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>
#include <time.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];
    int cnt = 0;
    double diff;

```

```
43          while((str_len = read(clnt_sock, msg, sizeof(msg))) != 0)
44              send_msg(msg, str_len);
45
46          pthread_mutex_lock(&mtx);
47          for(i=0;i<clnt_cnt;i++){
48              if(clnt_sock == clnt_socks[i]){
49                  while(i++ < clnt_cnt - 1)
50                      clnt_socks[i] = clnt_socks[i+1];
51                  break;
52              }
53          }
54          clnt_cnt--;
55          pthread_mutex_unlock(&mtx);
56          close(clnt_sock);
57          return NULL;
58      }
59
60      int main(int argc, char** argv){
61          int serv_sock, clnt_sock;
62          si serv_addr, clnt_addr;
63          socklen_t addr_size;
64          pthread_t t_id;
65
66          if(argc != 2){
67              printf("Usage : %s <port>\n",argv[0]);
68              exit(1);
69          }
70
71          pthread_mutex_init(&mtx, NULL);
72
73          serv_sock = socket(PF_INET, SOCK_STREAM, 0);
74
75          if(serv_sock == -1)
76              err_handler("socket() error");
77
78          memset(&serv_addr, 0, sizeof(serv_addr));
79          serv_addr.sin_family = AF_INET;
80          serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
81          serv_addr.sin_port = htons(atoi(argv[1]));
82
83          if(bind(serv_sock, (sp)&serv_addr, sizeof(serv_addr)) == -1)
84              err_handler("bind() error");
85
86          if(listen(serv_sock, 30) == -1)
87              err_handler("listen() error");
88
89          for(;;){
90              addr_size = sizeof(clnt_addr);
91              clnt_sock = accept(serv_sock, (sp)&clnt_addr, &addr_size);
92
93              pthread_mutex_lock(&mtx);
94              clnt_socks[clnt_cnt++] = clnt_sock;
95              pthread_mutex_unlock(&mtx);
96
97              pthread_create(&t_id, NULL, clnt_handler, (void*)&clnt_sock);
98              pthread_detach(t_id);
99              printf("Connected Client IP : %s\n",inet_ntoa(clnt_addr.sin_addr));
100         }
101         close(serv_sock);
```

```
    return 0;
}
```


clnt.c

```c
#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     128
#define NAME_SIZE     32

typedef struct sockaddr_in si;
typedef struct sockaddr * sp;

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];

    for(;;){
        fgets(msg, BUF_SIZE, stdin);
        if(!strcmp(msg, "q\n") || !strcmp(msg, "Q\n")){
            close(sock);
            exit(0);
        }
        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;
}
```

```
56
57  int main(int argc,char** argv){
58      int sock;
59      si serv_addr;
60      pthread_t snd_thread, rcv_thread;
61      void* thread_ret;
62      if(argc != 4){
63          printf("Usage: %s <IP> <port><name>\n",argv[0]);
64          exit(1);
65      }
66      sprintf(name, "[%s]", argv[3]);
67      sock = socket(PF_INET, SOCK_STREAM, 0);
68
69      if(sock == -1)
70          err_handler("socket() error");
71
72      memset(&serv_addr, 0, sizeof(serv_addr));
73      serv_addr.sin_family = AF_INET;
74      serv_addr.sin_addr.s_addr = inet_addr(argv[1]);
75      serv_addr.sin_port = htons(atoi(argv[2]));
76
77      if(connect(sock, (sp)&serv_addr, sizeof(serv_addr)) == -1)
78          err_handler("connect() error");
79
80      pthread_create(&snd_thread, NULL, send_msg, (void*)&sock);
81      pthread_create(&rcv_thread, NULL, recv_msg, (void*)&sock);
82
83      pthread_join(snd_thread, &thread_ret);
84      pthread_join(rcv_thread, &thread_ret);
85
86      close(sock);
87      return 0;
88  }
89
```



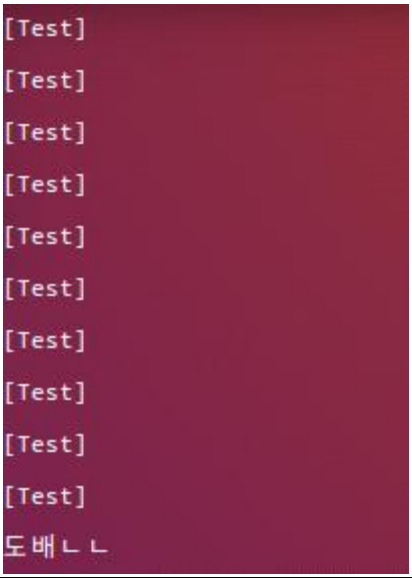
```
test message
[Test] test message
```

89.앞선 문제의 답에 도배를 방지 기능을 추가하시오.

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <string.h>
4   #include <unistd.h>
5   #include <pthread.h>
6   #include <arpa/inet.h>
7   #include <sys/socket.h>
8   #include <sys/epoll.h>
9   #include <time.h>
10
11  #define BUF_SIZE     128
12  #define MAX_CLNT     256
13
14  typedef struct sockaddr_in si;
15  typedef struct sockaddr *  sp;
16
17  int clnt_cnt = 0;
18  int clnt_socks[MAX_CLNT];
19  pthread_mutex_t mtx;
```

```
20
21  void err_handler(char* msg){
22      fputs(msg, stderr);
23      fputc('\n', stderr);
24      exit(1);
25  }
26
27  void send_msg(char* msg,int len){
28      int i;
29
30      pthread_mutex_lock(&mtx);
31
32      for(i=0;i<clnt_cnt;i++)
33          write(clnt_socks[i], msg, len);
34      pthread_mutex_unlock(&mtx);
35  }
36  void* clnt_handler(void* arg){
37      int clnt_sock = *((int*)arg);
38      int str_len = 0, i;
39      char msg[BUF_SIZE];
40      int cnt = 0;
41      clock_t clk_start;
42      double diff;
43      char kick_msg[256]="도배ㄴㄴ\n";
44      clk_start = clock();
45
46      while((str_len = read(clnt_sock, msg, sizeof(msg))) != 0){
47          redo:
48          diff = (double)(clock() - clk_start) / CLOCKS_PER_SEC;
49          diff *= 20000;
50          if((cnt > diff)&&(cnt > 4)){
51              send_msg(kick_msg, strlen(kick_msg));
52              goto tmp;
53          }
54          //initialize kick counter
55          if(diff > 5){
56              clk_start = clock();
57              cnt = 0;
58          }
59
60          send_msg(msg, str_len);
61          cnt++;
62      }
63  tmp:
64      sleep(3);
65      goto redo;
66      pthread_mutex_lock(&mtx);
67
68      for(i=0;i<clnt_cnt;i++){
69          if(clnt_sock == clnt_socks[i]){
70              while(i++ < clnt_cnt - 1)
71                  clnt_socks[i] = clnt_socks[i+1];
72              break;
73          }
74      }
75      clnt_cnt--;
76      pthread_mutex_unlock(&mtx);
77      close(clnt_sock);
78      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, 30) == -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;
}
```



90.앞서 만든 프로그램 조차도 공격할 수 있는 프로그램을 작성하시오.

```c
#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>
#include <time.h>
#define BUF_SIZE     128
#define NAME_SIZE     32

typedef struct sockaddr_in si;
typedef struct sockaddr * sp;

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

void err_handler(char* msg){
    fputs(msg, stderr);
    fputc('\n',stderr);
    exit(1);
}
char* create_rand_msg(void){
    srand(time(NULL));
    int str_len = rand()%20 + 10;
    int i;
    char* msg = (char*)malloc(sizeof(char)*str_len);
    for(i=0;i<str_len - 1;i++)
        msg[i] = 65 + rand()%26;
    msg[str_len - 1] = '\n';
    msg[str_len] = '\0';
    return msg;
}

void* send_msg(void* arg){
    int sock = *((int*)arg);
    char name_msg[NAME_SIZE + BUF_SIZE];
    char* msg;
    int time_rand;
    srand(time(NULL));
```

```
42          for(;;){
43              //period : 1~6 sec
44              time_rand = rand()%6 + 1;
45              //random msg
46              msg = create_rand_msg();
47
48              if(!strcmp(msg, "q\n") || !strcmp(msg, "Q\n")){
49                  close(sock);
50                  exit(0);
51              }
52              write(sock, msg, strlen(msg));
53              sleep(time_rand);
54          }
55          return NULL;
56      }
57
58      void* recv_msg(void* arg){
59          int sock = *((int*)arg);
60          char name_msg[NAME_SIZE + BUF_SIZE];
61          int str_len;
62
63          for(;;){
64              str_len = read(sock, name_msg, NAME_SIZE + BUF_SIZE - 1);
65              if(str_len == -1)
66                  return (void*)-1;
67
68              name_msg[str_len] = 0;
69              fputs(name_msg, stdout);
70          }
71          return NULL;
72      }
73
74      int main(int argc,char** argv){
75          int sock;
76          si serv_addr;
77          pthread_t snd_thread, rcv_thread;
78          void* thread_ret;
79          if(argc != 4){
80              printf("Usage: %s <IP> <port><name>\n",argv[0]);
81              exit(1);
82          }
83          sprintf(name, "[%s]", argv[3]);
84          sock = socket(PF_INET, SOCK_STREAM, 0);
85
86          if(sock == -1)
87              err_handler("socket() error");
88
89          memset(&serv_addr, 0, sizeof(serv_addr));
90          serv_addr.sin_family = AF_INET;
91          serv_addr.sin_addr.s_addr = inet_addr(argv[1]);
92          serv_addr.sin_port = htons(atoi(argv[2]));
93
94          if(connect(sock, (sp)&serv_addr, sizeof(serv_addr)) == -1)
95              err_handler("connect() error");
96
97          pthread_create(&snd_thread, NULL, send_msg, (void*)&sock);
98          pthread_create(&rcv_thread, NULL, recv_msg, (void*)&sock);
99
100         pthread_join(snd_thread, &thread_ret);
```

```
101         pthread_join(rcv_thread, &thread_ret);
102
103         close(sock);
104         return 0;
105     }
106
```



```
howard@ubuntu:~/HomeworkBackup/29th$ ./clnt 127.0.0.1 7777 D-DOS
XNMCBFBAZRLCHAYYXV
LRBBPCTCHBTLLLZDPOYMPTZXHJF
JIOOWDEKPVIUBBPWCXOKHMNILVDQ
VMKOAZBQRAIRJPMUYOZBQUPUR
QARBLEDHEJATCNGOGEOHV
F[HYEHBWQDURDGT
WQKOBDJQNH
LKBDII[SURZCGVINZOMGUXBGND
IHTF[QTDIN
NAKFHL[LSQONMF
```

91.네트워크 상에서 구조체를 전달할 수 있게 프로그래밍 하시오.

st_server.c

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <string.h>
4   #include <unistd.h>
5   #include <arpa/inet.h>
6   #include <sys/socket.h>
7   typedef struct sockaddr_in si;
8   typedef struct sockaddr * sap;
9   typedef struct __st{
10      char name[30];
11      int score;
12  }st;
13  void err_handler(char* msg){
14      fputs(msg, stderr);
15      fputc('\n', stderr);
16      exit(1);
17
18  }
19
20  int main(int argc,char** argv){
21      int serv_sock;
22      int clnt_sock;
23      int ret;
24      st st_send = {"KKK",100};
25      si serv_addr;
26      si clnt_addr;
27      socklen_t clnt_addr_size;
28      char* send = "Hello";
29      char recv[32] = {0,};
30      if(argc != 2){
31          printf("use : %s <port>\n",argv[0]);
32          exit(1);
33      }
34      //sock file descriptor
35      serv_sock = socket(PF_INET, SOCK_STREAM, 0);
36      if(serv_sock == -1)
37          err_handler("socket() error");
38
39      memset(&serv_addr, 0, sizeof(serv_addr));
40      serv_addr.sin_family = AF_INET;
41      serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
42      serv_addr.sin_port = htons(atoi(argv[1]));
```

```
43      //bind : server ip address setting
44      if(bind(serv_sock, (sap)&serv_addr, sizeof(serv_addr)) == -1)
45          err_handler("bind() error");
46      //listen : max people , wait client.
47
48      if(listen(serv_sock, 5) == -1)
49          err_handler("listen() error");
50
51      clnt_addr_size = sizeof(clnt_addr);
52      //client permit.
53      clnt_sock = accept(serv_sock, (struct sockaddr *)&clnt_addr,
    &clnt_addr_size);
54      if(clnt_sock == -1)
55          err_handler("accept() error");
56      write(clnt_sock, &st_send, sizeof(&st_send));
57
58      close(clnt_sock);
59      close(serv_sock);
60
61  }
62
```


st_client.c

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <string.h>
4   #include <unistd.h>
5   #include <arpa/inet.h>
6   #include <sys/socket.h>
7
8   typedef struct sockaddr_in si;
9   typedef struct sockaddr * sap;
10  typedef struct __st{
11      char name[30];
12      int score;
13  }st;
14
15
16  void err_handler(char* msg){
17      fputs(msg, stderr);
18      fputc('\n',stderr);
19      exit(1);
20  }
21
22  int main(int argc,char* argv[]){
23
24      st st_recv;
25      int sock;
26      int str_len;
27      char msg[32];
28      char* send = "Hi";
29      si serv_addr;
30      if(argc != 3){
31          printf("err\n");
32          exit(1);
33      }
34
35      sock = socket(PF_INET, SOCK_STREAM, 0);
36
37      if(sock == -1)
```

```
38              err_handler("socket() error");
39
40          memset(&serv_addr, 0 , sizeof(serv_addr));
41          serv_addr.sin_family = AF_INET;
42          serv_addr.sin_addr.s_addr = inet_addr(argv[1]);
43          serv_addr.sin_port = htons(atoi(argv[2]));
44
45          if(connect(sock, (sap)&serv_addr, sizeof(serv_addr)) == -1)
46              err_handler("connect() error");
47
48          read(sock, &st_recv, sizeof(&st_recv) );
49          printf("name : %s, score : %d\n",st_recv.name,st_recv.score);
50
51          close(sock);
52          return 0;
53
54      }
55
```

Colored by Color Scripter

```
howard@ubuntu:~/HomeworkBackup/test2_ref$ ./st_clnt 127.0.0.1 7777
name : KKK, score : 1
```

92.앞선 문제를 응용하여 Queue 와 Network 프로그래밍을 연동하시오.

93. Critical Section 이 무엇인지 기술하시오.

process 내에서 thread를 생성하면, stack을 제외한 나머지 segment를 공유하는 task가 생성된다. 공유하는 segment가 있기 때문에, 접근 순서에 따라 해당 데이터가 원하지 않는 값이 저장될 수 있다. 이러한 segment 영역을 Critical Section이라고 한다.

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

95.리눅스 커널의 arch 디렉토리에 대해서 설명하시오.

CPU 아키텍처에 종속적인 부분이 담긴 디렉토리이다. 여러 가지 지원 하드웨어에 대해 각각 만들어져 있다. 대표적으로 x86(인텔)과 arm이 있다.

96.앞선 문제에서 arm 디렉토리 내부에 대해 설명하도록 하시오

97.리눅스 커널 arch 디렉토리에서 c6x 가 무엇인지 기술하시오.

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.
(맨 처음 과정에 들어왔을 때의 모습과 현재의 모습에 어떠한 발전이 있었는지 등등을 자세히 기술하도록 한다)
또한 앞으로는 어떠한 자세로 임할 것인지 작성하시오.

2개월차 수업을 들었지만, 벌써 대학 4년동안 배운 것보다도 더 많이 배운 듯한 기분이 듭니다. 매일매일 노력을 게을리 하지 않으려고 했습니다. 1개월 조금 지나고 부터는 거의 매일 남아 열심히 한다고 했지만, 공부한다고 노트북 잡고 있으면서 딴짓하는 시간, 핸드폰 보면서 뺏긴 시간 등 너무 아까운 시간들이 많았습니다. 그래도 잘한 것부터 말하자면, 자료구조나 정렬 알고리즘 부분 등 logic에 있어서는 매주 감을 잃지 않도록 노력했다고 생각합니다.

반면 현재 공부중인 kernel part와 system programming, network programming의 수련은 조금 부족했지 않나 생각합니다. 특히 커널부분은 다음주 쉬는 기간동안 무슨 방법을 써도 꼭 마스터하고 싶습니다. 향후 DSP와 FPGA, 특히 FPGA에 욕심이 조금 나기 때문에, 추상적이라 이해하기 어려운 OS라는 개념을 드라이빙을 통해 낱낱이 파헤쳐 볼 계획입니다.

앞으로의 공부 자세에 있어 목표는 2가지 정도일 것 같습니다. 먼저, 낭비되는 시간을 조금 줄여야 될 것 같습니다. 아무래도 혼자 공부할 때, 이어폰 끼고 공부하다보니 자꾸 딴짓을 하게 되는 경우가 있는데, 리눅스의 터미널, 구글링을 위한 인터넷페이지를 제외하고는 아무것도 키지 않고 공부할 생각입니다. 두 번째는 주변을 잘 활용할 계획입니다. 커널 공부하다보니, 모르는 개념 하나 나오면 오기가 생겨 그것을 해결하기 전까지는 그것만 죽어라 파는 경우가 있었습니다. 찾아봐도 잘 모르는 개념이 나오면, 일단 보류하고 난 후 다음 단계로 넘어가고 친구들과 선생님을 통한 해결을 하도록 할 것입니다.

OS는 이상적인 인간의 모습을 보는 듯 합니다. OS의 동작을 본받아 앞으로의 공부에 더욱 전념할 것입니다. 요약하자면, 앞으로의 제 공부 방향은 FPGA를 위한 공부를 초점으로 둘 것이며, 공부 외에 낭비되는 시간을 줄이고(외부 단편화 제거) 삽질하는 시간을 줄여(내부 단편화 제거) 남은 기간을 최대한 효율적으로 활용할 것입니다.