

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

강사 – Innova Lee(이상훈)

gcccompil3r@gmail.com

학생 – 장성환

redmk1025@gmail.com

1 문제 시스템 프로그래밍 5 점 문제 내용

파이프 통신을 구현하고 c type.c 라고 입력할 경우

현재 위치의 디렉토리에 type.c 파일을 생성하도록 프로그래밍하시오.

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

int main (void){

    int fd1, fd2,ret;
    char buf[1024];
    fd1 = open("myfifo",O_RDWR);

    fcntl(0, F_SETFL, O_NONBLOCK);
    fcntl(fd1, F_SETFL, O_NONBLOCK);

    for(;;){
        if((ret = read(fd1,buf,sizeof(buf)))>0){
            buf[ret-1] = 0;
            if(buf[0] == 'c'){
                char tmp[1024];
                for(int i=0; i<(strlen(buf)-2);i++){
                    tmp[i] = buf[i+2];
                }

                fd2 = open(tmp, O_CREAT | O_RDWR | O_EXCL, 0644);
```

```

    }
    else{
        printf("PIPE Input : [%s]\n", buf);
    }
}
}
close(fd1);
return 0;
}

```

결과.

```

hwan@HWAN:~/Documents/exam2$ cat > myfifo
qwe
asd
c abc.txt
^C

```

```

hwan@HWAN:~/Documents/exam2$ ls
1 1.c 2.c 78.c 79.c 92clnt.c 92serv.c a.out abc.txt clnt myfifo serv

```

다음과 같이 c abc.txt 를 누르자

파일이 생성되었다.

2 문제 시스템 프로그래밍 5 점 문제 2 내용

369 게임을 작성하시오.

2 초내에 값을 입력하게 하시오.

박수를 쳐야 하는 경우를 Ctrl + C 를 누르도록 한다.

2 초 내에 값을 입력하지 못할 경우 게임이 오버되게 한다.

Ctrl + C 를 누르면 "Clap!" 이라는 문자열이 출력되게 한다.

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

int cnt=1;
int clap_cnt;

void fail_game(int signo){
    printf("Fail!!!!!!!!!!!!!!\n");
    exit(0);
}

void clap(){
    printf("clap!\n");
    clap_cnt++;
}

int init_game(){
    char buf[1024];
    int tmp=0;

    sprintf(buf,"%d",cnt);
    for(int i=0; i<strlen(buf); i++){
        if(buf[i] == '3' || buf[i] == '6' || buf[i] == '9'){
```

```
hwan@HWAN:~/Documents/exam2$ ./a.out
1
correct!
2
correct!
^Cclap!
correct!
4
correct!
5
correct!
^Cclap!
correct!
7
correct!
8
correct!
Fail!!!!!!!!!!!!!!
```

```
        tmp++;
    }
}
return tmp;
}

int correct(int num){
    if(cnt == num)
        return 1;
    else
        return 0;
}

int main(void){

    int num;
    int ccnt;

    signal(SIGINT, clap);
    signal(SIGALRM, fail_game);

    while(1){
        ccnt = 0;
        ccnt = init_game();

        if(ccnt!=0){

            alarm(2);
            while(ccnt!=clap_cnt){
                sleep(0.2);
            }
            alarm(0);
```

```
    printf("correct!\n");
    cnt++;
    continue;
}

else{

    clap_cnt =0;
    alarm(2);
    scanf("%d", &num);
    alarm(0);

    if(correct(num) == 1){
        printf("correct!\n");
        cnt++;
        continue;
    }

    else{
        printf("idiot!!!\n");
        break;
    }
}

return 0;
}
```

3 문제 아래 물음에 답시오. 내용
리눅스 커널은 운영체제(OS)다.

OS 가 관리해야 하는 제일 중요한 5 가지에 대해 기술하시오.스

파일, 태스크, 메모리, 네트워크, 디바이

4 문제 아래 물음에 답하시오. 내용

Unix 계열의 모든 OS 는 모든 것을 무엇으로 관리하는가 ?

파일

5 문제 리눅스의 장점에 대한 각각의 상세한 기술 내용

리눅스에는 여러 장점이 있다.(배점 0.2 점) —

아래의 장점들 각각에 대해 기술하라.

* 사용자 임의대로 재구성이 가능하다. —

* 열악한 환경에서도 HW 자원을 적절히 활용하여 동작한다. —

* 커널의 크기가 작다. —

* 완벽한 멀티유저, 멀티태스킹 시스템 —

* 뛰어난 안정성 —

* 빠른 업그레이드 —

* 강력한 네트워크 지원 —

* 풍부한 소프트웨어

6 문제 아래 물음에 답하시오. 내용

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

0x00000000 ~ 0xC0000000 유저 스페이스 0xC0000001 ~ 0xFFFFFFFF 커널 스페이스

7 문제 아래 물음에 답하시오. 내용

Page Fault 가 발생했을때

운영체제가 어떻게 동작하는지 기술하시오.

페이지 폴트 발생시 이것이 커널단에서 발생한 것인지 유저 단에서 발생한 것인지 확인한다. 유저단에서 발생했을 경우 페이지가 폴트 났다는 메시지를 날리고, 커널단에서 발생하였다면 남는 공간의 페이지 프레임을 할당 받고 해당 실행파일에서 필요한 페이지를 프레임에 기록 후 페이지 테이블에 기록하고 주소변환 과정을 다시 수행한다.

8 문제 아래 물음에 답하시오. 내용

리눅스 실행 파일 포맷이 무엇인지 적으시오.

ELF

9 문제 아래 물음에 답하시오. 내용

프로세스와 스레드를 구별하는 방법에 대해 기술하시오.

pid 와 tgid 가 같다면 프로세스, pid 와 tgid 가 다르다면 스레드이다.

10 문제 아래 물음에 답하시오. 내용

Kernel 입장에서 Process 혹은 Thread 를 만들면 무엇을 생성하는가 ?

task_struct 구조체를 생성한다.

11 문제 아래 물음에 답하시오. 내용

리눅스 커널 소스에 보면 current 라는 것이 보인다.

이것이 무엇을 의미하는 것인지 적으시오.

커널 소스 코드와 함께 기술하시오.

current 는 현재 태스크의 task_struct 구조체를 가리킬 수 있게 해준다.

12 문제 아래 물음에 답하시오. 내용

Memory Management 입장에서 Process 와 Thread 의 핵심적인 차이점은 무엇인가 ?

자원공유가 되는지 안되는지가 핵심적인 차이점이다.

13 문제 아래 물음에 답하시오. 내용

Task 가 관리해야하는 3 가지 Context 가 있다.

System Context, Memory Context, HW Context 가 있다.

이중 HW Context 는 무엇을 하기 위한 구조인가 ?

thread_info 에 존재하며 문맥 교환시 태스크의 현재 실행위치에 대한 정보를 유지한다.

(이 정보를 유지해야 context switch 발생 후 다시 불러들일때, 정상적으로 이어질 수 있다.)

14 문제 아래 물음에 답하시오. 내용

리눅스 커널의 스케줄링 정책중 Deadline 방식에 대해 기술하시오.

완료시간, 작업량, 주기성을 가지고 수행이 되는 방식이다.

즉, 현재시간 + 런타임은 데드라인보다 작아야 한다.

예를 들면 모니터가 30 hz 로 동작한다면, 0.033 초안에 작업이 완료 되어야 한다. 기존 태스크들의 런타임과 주기를 이용하여 데드라인 안으로 작동할 수 있도록 하는 방식이다.

15 문제 다음 물음에 답하시오. 내용

TASK_INTURRUPTIBLE 과 TASK_UNINTERRUPTIBLE 은 왜 필요한지 기술하시오.

시그널에 반응하는지 안하는지에 대한것이다.

사소한 시그널에 반응하여 태스크가 죽어버리면 안정성을 추구하는 프로그램에는 큰 문제일 것이다.

따라서 시그널에 반응하지 않도록 하려면 필요하다. 반대의 경우도 마찬가지이다.

16 문제 아래 물음에 답하시오. 내용

O(N)과 O(1) Algorithm 에 대해 기술하시오.

그리고 무엇이 어떤 경우에 더 좋은지 기술하시오.

O(N) 알고리즘은 처리에 있어서 입력에 비례하여 비율이 대략적으로 n 만큼 증가하는 것이고, O(1)알고리즘은 항상 알고리즘 처리에 있어서 일정한 비율의 시간으로 처

리가 가능하다.

두 알고리즘 관련하여 $O(1)$ 의 일정한 비율보다 $O(N)$ 이 더 빠르게 처리되는 경우가 있다.

이런 경우에는 $O(N)$ 을 사용하고 입력이 많은 경우에는 $O(1)$ 을 사용하면된다. (즉 상황에 따라 다르다.)

17 문제 아래 물음에 답하시오. 내용

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 에 위치하도록 하는것이 좋다. 캐시를 공유하기 때문에 성능상 저하를 최소화 시키기 위해서이다.

18 문제 아래 물음에 답하시오. 내용

앞선 문제에서 이번에는 0, 1, 3 에 매우 많은 프로세스가 존재한다.

이 경우 3 번에서 `fork()`를 수행하여 Task 를 만들었다.

이 Task 는 어디에 위치하는 것이 좋을까 ?

역시 이유를 적으시오.

2 번에 위치시키는 것이 좋다. 캐시 공유든 여러가지 상황을 고려하더라도 다른 cpu 가 매우 많은 작업을 수행하느라 바쁘다면, 한가한 cpu 로 이주 시키는 것이 성능상의 이점이기 때문이다.

19 문제 아래 물음에 답하시오. 내용

UMA 와 NUMA 에 대해 기술하고 Kernel 에서 이들을 어떠한 방식으로 관리하는지 기술하시오.

커널 내부의 소스 코드와 함께 기술하도록 하시오.

UMA - 복수개의 cpu 가 메모리 등의 자원을 공유

NUMA - 복수개의 cpu 가 그룹으로 나뉘어 각각에 그룹에게 별도의 지역 메모리를 주는 구조.

20 문제 아래 물음에 답하시오. 내용

Kernel 의 Scheduling Mechanism 에서 Static Priority 와 Dynamic Priority 번호가 어떻게 되는지 적으시오.

21 문제 아래 물음에 답하시오. 내용

ZONE_HIGHMEM 에 대해 아는대로 기술하시오.

32 비트 체제에서 실제 메모리가 큰 경우에 커널의 가상 메모리가 1GB 를 할당받아서 1:1 맵핑이 불가능하다. 따라서 896MB 이상의 메모리 공간을 간접참조 형식으로 실제 메모리 전부를 가리키게 할 수 있도록 한다.

22 문제 다음 물음에 답하시오. 내용

물리 메모리의 최소 단위를 무엇이라고 하며 크기가 얼마인가 ?

그리고 이러한 개념을 SW 적으로 구현한 구조체의 이름은 무엇인가 ?

페이지프레임 4KB

```
struct page{}
```

23 문제 다음 물음에 답하시오. 내용

Linux Kernel 은 외부 단편화와 메모리 부하를 최소화하기 위해 Buddy 할당자를 사용한다.

Buddy 할당자의 Algorithm 을 자세히 기술하시오.

버디 할당자는 zone 구조체의 free_area 구조체 배열을 통하여 구축된다.

버디 할당자는 요청된 크기를 만족하는 최소의 order 에서 페이지 프레임을 할당해준다. 만약에 그 order 에 가용된 페이지 프레임이 존재하지 않으면 상위 order 에서 페이지 프레임을 할당받아 두 부분으로 나누어 한 부분은 할당해주고 나머지 부분은 하위 order 에서 가용 페이지 프레임으로 관리하게 된다.

24 문제 다음 문제에 답을 작성하시오. 내용

앞선 문제에 이어 내부 단편화를 최소화 하기 위해 Buddy 에서 Page 를 받아 Slab 할당자를 사용한다.

Slab 할당자는 어떤식으로 관리되는지 기술하시오.

슬랩 할당자는 메모리 일부구간을 미리 나누어 캐시처럼 사용하려고 하는 것이다.

32byte ~ 4MB 크기까지 나누어 유지한다. 슬랩 할당자인 `kmem_cache_alloc()`으로 순서대로 저장한다. 만일 공간을 할당 받으면서 더이상 할당해줄 공간이 없으면 버디로부터 페이지 프레임을 더 할당 받아서 `kmem_cache_grow()` 같은 함수를 호출하여 슬랩을 확장 시킨다.

25 문제 아래 물음에 답하시오. 내용

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 -> mm_struct -> vm_area_struct`

`mm_struct` 에서 구조변수들인 `start_code`, `end_code`, `end_data`, `brk`, `start_brk`, `start_stack` 등등 과 같은 변수로 스택, 힙, 데이터, 텍스트 해당 영역의 위치를 가리킨다.

26 문제 다음 물음에 대해 답하시오. 내용

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 -> mm_struct -> vm_area_struct`

`vm_area_struct` 에서 공통의 속성을 갖는 각각의 region 을 관리한다. (`vm_end`, `vm_start` 로 region 의 시작과 끝이 나옴)

27 문제 다음 물음에 대해 답하시오. 내용

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 ?

`a.out` 파일의 헤더를 읽고 적절한 값을 채워준다.

28 문제 아래 물음에 답하시오. 내용

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.

커널도 하나의 프로그램이기 때문에 당연히 메모리의 기본 구성이 존재 하여야 한다.

29 문제 아래 물음에 답하시오. 내용

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

sys_execve() 시스템 호출이 남는 페이지 프레임을 할당 받은후에 파일의 일부 내용을 읽는다. 메모리에 실제로 몇 몇 장소에 올라가게 되고,

페이지 테이블을 통하여 가상 메모리의 주소가 페이지 테이블을 이용하여 실제 메모리를 가리킬 수 있도록 한다.

demend on page 로 일부 실행에 필수적인 부분만 지정해 놓는다. 그리고 이후에 물리메모리에 적재 되어 있지 않은 부분이 있다면 페이지 폴트 핸들러가 동작이 되어 (남는 페이지 프레임을 할당받고 페이지 테이블에 다시 기록하고 주소변환과정 수행) 물리 주소로 변환 할 수 있게 한다.

30 문제 다음 물음에 답하라. 내용

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

가상 주소로부터 물리 주소로의 변환을 담당하는 별도의 하드웨어

주소 변환을 위해서는 페이지 디렉터리와 페이지 테이블을 탐색해야 하는데, 이것을 하드웨어 적으로 처리한다. 또한 TLB 같은 페이지 테이블 엔트리 캐시를 사용하여 빠른 주소변환을 지원한다.

31 문제 물음에 답하라. 내용

하드디스크의 최소 단위를 무엇이라 부르고 그 크기는 얼마인가?

디스크 블록, 4KB

32 문제 다음 물음에 답하라. 내용

Character 디바이스 드라이버를 작성할 때 반드시 Wrapping 해야 하는 부분이 어디인가?

(Task 구조체에서 부터 연결된 부분까지를 꼭 이어서 작성하라)

33 문제 다음 물음에 답하시오. 내용

예로 유저 영역에서 open 시스템 콜을 호출 했다고 가정할 때 커널에서는 어떤 함수가 동작하게 되는가 ?

실제 소스 코드 차원에서 이를 찾아서 기술하도록 한다.

sys_open()

34 문제 아래 물음에 답하라. 내용

task_struct 에서 super_block 이 하는 역할은 무엇인가 ?

'/'의 디렉터리 위치를 알 수 있게 한다.

35 문제 아래 물음에 답하시오. 내용

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

VFS 는 어떤 파일 a 에 대하여 무슨 파일시스템인지 판단한다. 다음으로 해당 파일의 정보를 담을 목적으로 구조체를 만들고, 이 구조체를 인자로 하여 해당 파일시스템에 구현되어있는 고유한 함수를 호출한다. 그러면 해당 파일시스템은 자신의 inode 테이블을 뒤져서 a 의 inode 를 찾아내고 inode 내에 담겨있는 정보를 VFS 가 넘긴 구조체에 적절히 넣어준 뒤 리턴한다. 그러면 VFS 는 이 구조체의 내용을 바탕으로 사용자 태스크에게 필요한 정보를 넘겨주게 된다.

36 문제 아래 물음에 답하시오. 내용

Linux Kernel 에서 Interrupt 를 크게 2 가지로 분류한다.

그 2 가지에 대해 각각 기술하고 간략히 설명하시오.

HW 인터럽트(태스크와 관련없이 주변 장치에서 발생된 비동기 하드웨어적인 사건)와 SW 인터럽트(태스크와 관련있는 동기적인 사건)로 나뉜다.

37 문제 아래 물음에 답하시오. 내용

내부 인터럽트는 다시 크게 3 분류로 나눌 수 있다.

3 가지를 분류하시오

fault, trap, abort

38 문제 아래 물음에 답하시오. 내용

앞선 문제에서 분류한 녀석들의 특징에 대해 기술하시오.

fault - fault 를 일으킨 명령어 주소를 eip 에 넣어 두었다가 해당 핸들러가 종료되고 나면 다시 eip 부터 수행을 시작한다.

trap - 시스템 콜과 같이 발생한 경우 명령어의 다음 주소를 eip 에 저장해 두었다가 이 이후에 그 다음부터 수행한다.
abort - 심각한 에러인 경우이므로 현재 태스크를 강제 종료한다.

39 문제 아래 물음에 답하시오. 내용

예로 모니터 3 개를 쓰는 경우 양쪽에 모두 인터럽트를 공유해야 한다.

Linux Kernel에서는 어떠한 방법을 통해 이들을 공유하는가 ?

디바이스 드라이버의 주 번호는 major 그리고 나머지인 minor 을 이용한다.

40 문제 다음 물음에 답하시오. 내용

System Call 호출시 Kernel 에서 실제 System Call 을 처리하기 위해 Indexing 을 수행하여 적절한 함수가 호출되도록 주소값을 저장해놓고 있다.

이 구조체의 이름을 적으시오.

41 문제 아래 물음에 답하시오. 내용

38 에서 User Space 에서 System Call 번호를 전달한다.

Intel Machine에서는 이를 어디에 저장하는가 ?

또한 ARM Machine에서는 이를 어디에 저장하는가 ?

__vectors_start :

42 문제 다음 물음에 대해 답하라. 내용

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

pgd_t * pgd;

43 문제 다음 물음에 답하라. 내용

또한 Page Directory 를 가르키는 Intel 전용 Register 가 존재한다.

이 Register 의 이름을 적고 ARM 에서 이 역할을 하는 레지스터의 이름을 적으시오.

IP , PC

44 문제 아래 물음에 답하라. 내용

커널 내부에서 메모리 할당이 필요한 이유는 무엇인가?

리눅스도 C 와 어셈블리어로 작성된 소프트웨어 이기 때문에 수행되기 위해서는 스택을 필요로 하기 때문이다.

45 문제 다음 물음에 답하라. 내용

메모리를 불연속적으로 할당하는 기법은 무엇인가?

`vmalloc()`

46 문제 아래 물음에 답하시오. 내용

메모리를 연속적으로 할당하는 기법은 무엇인가?

`kmalloc()`

47 문제 아래 물음에 답하시오. 내용

Mutex 와 Semaphore 의 차이점을 기술하시오.

뮤텍스는 잠금과 해제 개념이다. 공유자원에 접근할 수 있도록 하는 허가권과 같은 개념이다.

세마포어는 접근 허용 가능한 카운터의 갯수를 가진 카운터이다. 카운터가 1 개인 경우에는 뮤텍스와 같은 동작을 하고 그 이상인 경우에는 다르게 볼 수 있다.

48 문제 다음 물음에 답하라. 내용

`module_init()` 함수 호출은 언제 이루어지는가?

`insmod` 명령 이후에 모듈을 삽입 후 호출이 이루어진다.

49 문제 물음에 답하라. 내용

`module_exit()` 함수 호출은 언제 이루어지는가?

`rmmod` 명령 이후에 적재된 모듈을 해제한 후에 호출이 이루어진다.

50 문제 아래 물음에 답하시오. 내용

`thread_union` 에 대해 기술하시오.

리눅스도 소프트웨어 이기 때문에 수행되기 위해서는 스택을 필요로 하는데,

태스크당 할당되는 커널 스택을 `thread_union` 이라고 한다.

`thread_union` 은 문맥교환을 위한 `cpu_context_save` (레지스터 저장), `thread_info` 를 가지고 있어서 `task_struct` 포인터, `flag` (스케줄링의 여부) 등을 가지고 있다.

51 문제 다음 물음에 답하라. 내용

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 꼭 찾아오길 바람)

52 문제 다음 물음에 답하시오. 내용

예로 간단한 Character Device Driver 를 작성했다고 가정해본다.

그리고 insmod 를 통해 Driver 를 Kernel 내에 삽입했으며 mknod 를 이용하여 /dev/장치파일을 생성하였다.

그리고 이에 적절한 User 프로그램을 동작시켰다.

이 Driver 가 실제 Kernel 에서 어떻게 관리되고 사용되는지 내부 Mechanism 을 기술하시오.

53 문제 아래 문제에 대한 답변을 작성하시오. 내용

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.

일반적으로 메모리가 할당될 때, 순차적으로 저장되면 효율이 좋다. 따라서 속도를 우선시 하는 경우에 kmalloc()을 사용하는 것이 좋다.

하지만 단점이 있다. kmalloc 은 슬랩 할당자로 부터 임의의 크기의 메모리 공간을 할당 받는다. 따라서 슬랩에서 정의된 4MB 크기 이상을 한번에 할당 받을 수 없다.

하지만 vmalloc 을 이용하여 가상적으로만 연속인 메모리를 할당을 한다면, 크기에 상관없이 할당이 가능하다.

54 문제 디바이스 드라이버를 작성하시오 - 10 점 내용

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 문제 아래 물음에 답하시오. 내용

OoO(Out-of-Order)인 비순차 실행에 대해 기술하라.

순차적으로 cpu 의 명령을 수행하는 것이 아니라 명령들이 서로 새치기를 하면서 파이프 라인을 효율적으로 사용하는 것.

56 문제 아래 물음에 답하시오. 내용

Compiler 의 Instruction Scheduling 에 대해 기술하라.

57 문제 다음 물음에 답하시오. 내용

CISC Architecture 와 RISC Architecture 에 대한 차이점을 기술하라.

CISC

명령어의 갯수가 많고, 그 길이가 다양하며 실행 사이클도 명령어마다 다름.

x86: 1 ~ 17 byte 사이의 가변길이의 명령어 세트

다양한 명령어를 사용하므로 컴파일러가 복잡해짐 그리고 회로 구성도가 복잡하다.

RISC

명령어의 길이는 고정적이며, 워드와 데이터 버스 크기가 모두 동일하고 실행 사이클도 모두 동일하다.

ARM: 32bit 고정길이 명령어세트 (다른 비트도 있으나 기본적으로 32 비트가 기본 명령어)

파이프라인을 사용한다.

명령어의 종류가 적어서 단순한 컴파일러 구현이 가능하고 회로 구성이 간단하다.

58 문제 아래 질문에 적절한 답을 기술하시오. 내용

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 어느정도의 Instruction Scheduling 을 HW 의 힘만으로 수행할 수 있다.

이러한 것을 무엇이라 부르는가 ?

MMU

59 문제 아래 물음에 답하시오. 내용

Pipeline 이 깨지는 경우에 대해 자세히 기술하시오.

분기 명령이 떨어지면 깨지게 된다 . 파이프는 미리 cpu 의 명령어를 담아두게 되는데 분기 명령이 생기는 순간에 미리 담아두었던 명령어가 쓸모가 없어진다. 따라서 다시 명령어를 담아두게 되며 이를 파이프라인이 깨졌다고 표현한다.

60 문제 다음 물음에 답하시오. 내용

CPU 들은 각각 저마다 이것을 가지고 있다.

Compiler 개발자들은 이것을 고려해서 Compiler 를 만들어야 한다.

또한 HW 입장에서 이것을 고려해서 설계를 해야 한다.

여기서 말하는 이것이란 무엇인가 ?

레지스터

61 문제 아래 질문에 적절한 답을 작성하시오. 내용

Intel 의 Hyper Threading 기술에 대해 상세히 기술하시오.

물리적인 cpu 에 대하여 fork()함수의 동작을 회로로 구성하여 논리적인 cpu 로 만드는 것이다.
즉, 물리적인 cpu 가 4 개라면 하이퍼 스레딩 기술을 통하여 논리적인 cpu 8 개로 동작이 가능하도록 한다.

62 문제 아래 질문에 답하라 - 10 점 문제 내용

그동안 많은 것을 배웠을 것이다.

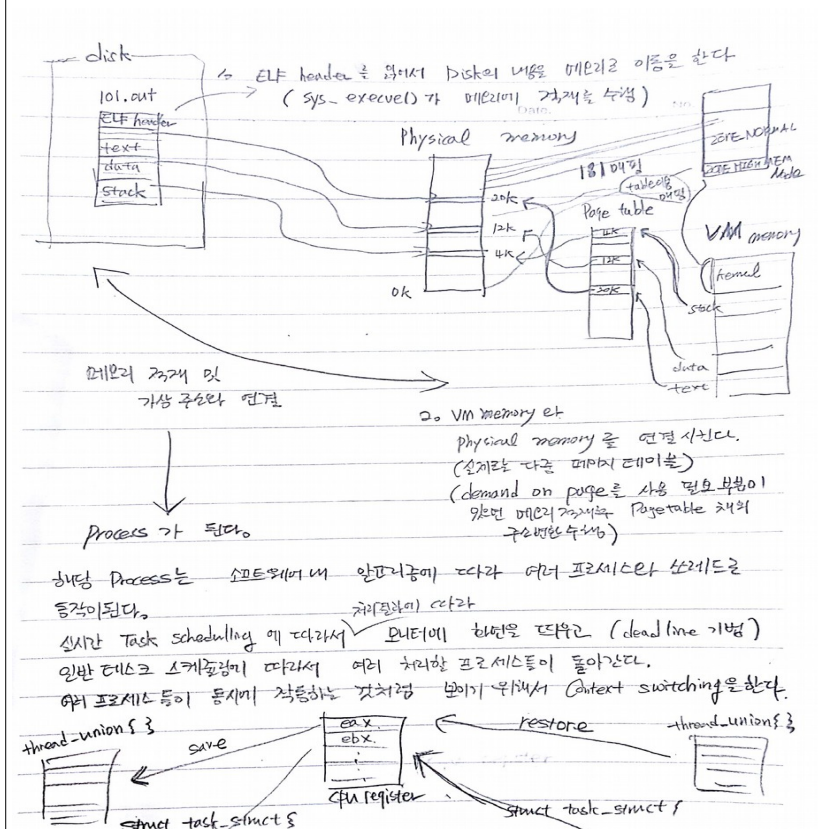
최종적으로 Kernel Map 을 그려보도록 한다. (Networking 부분은 생략해도 좋다)

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 문제 다음을 프로그래밍 하시오. 내용

파일의 종류를 확인하는 프로그램을 작성하시도록 하시오.

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

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

    struct stat buf;
    struct dirent *p;
    struct passwd *pw;
    struct group *gr;
    struct tm *tm;
    char ch;
    char perm[11] = "_____";
    char rwx[4] = "rwx";
    char sst[4] = "sst";
    int i;
    stat(argv[1], &buf);

    if(S_ISDIR(buf.st_mode)) //st_mode 에 해당 정보(디렉토리 인지)를 인식하여 리턴한다.
        perm[0] = 'd';
    if(S_ISREG(buf.st_mode))
        perm[0] = '-';
    if(S_ISFIFO(buf.st_mode))
        perm[0] = 'p';
```

```

if(S_ISLNK(buf.st_mode))
    perm[0] = 'l';
if(S_ISSOCK(buf.st_mode))
    perm[0] = 's';
if(S_ISCHR(buf.st_mode))
    perm[0] = 'c';
if(S_ISBLK(buf.st_mode))
    perm[0] = 'b';

for(i=0; i<9; i++)
    if((buf.st_mode >> (8-i)) & 1)
        perm[i + 1] = rwx[i % 3];
for(i=0; i<3; i++)
    if((buf.st_mode >> (11-i)) & 1)
        if(perm[(i+1)*3] = '-')
            perm[(i+1)*3] = sst[i] ^ 0x20;
        else
            perm[(i+1)*3] = sst[i];

printf("%s",perm);
printf("%lu",buf.st_nlink);
pw = getpwuid(buf.st_uid);
printf("%s",pw->pw_name);
gr = getgrgid(buf.st_gid);
printf("%s",gr->gr_name);
printf("%lu",buf.st_size);
tm = localtime(&buf.st_mtime);
printf("%d-%02d-%02d %02d %02d",tm->tm_year +1900, tm->tm_mon +1, tm->tm_mday, tm->tm_hour, tm->tm_min);
printf("\n");
return 0;
}

```

64 문제 다음을 프로그래밍 하시오. 내용

서버와 클라이언트가 1 초마다 Hi, Hello 를 주고 받게 만드시오.

*serv

```
#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;
```

```
#define BUF_SIZE 1024
```

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

```
int main (int argc, char **argv){
    int i, str_len;
    int serv_sock, clnt_sock;

    char msg[BUF_SIZE];

    si serv_addr, clnt_addr;
    socklen_t clnt_addr_size;
```

*clnt

```
#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;
```

```
#define BUF_SIZE 1024
```

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

```
int main(int argc, char **argv){

    int sock;
    int str_len;
    si serv_addr;
    char *m = "Input Message(q to quit): ";
    char msg[BUF_SIZE];

    if(argc !=3){
```

```

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

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, (sap)&serv_addr, sizeof(serv_addr)) == -1)
    err_handler("bind() error");
if(listen(serv_sock, 5) == -1)
    err_handler("listen() error");

clnt_addr_size = sizeof(clnt_addr);

for(i = 0; i < 5; i++){
    clnt_sock = accept(serv_sock, (struct sockaddr *)&clnt_addr,
    &clnt_addr_size);

    if(clnt_sock == -1)
        err_handler("accept() error");

    else
        printf("Connected Client %d\n", i+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, (sap)&serv_addr, sizeof(serv_addr)) == -1)
    err_handler("connect() error");
else
    printf("Connected.....");


for(;;){
    //fputs("Input msg(q to QUIT): ", stdout);
    //fgets(msg, BUF_SIZE, stdin);

    //if(!strcmp(msg, "q\n") || !strcmp(msg, "Q\n"))
    //    break;

    sleep(1);
    write(sock, "Hi", 2);
    str_len = read(sock, msg, BUF_SIZE-1);

    if(str_len == -1)
        err_handler("read() error!");

```


<pre> while((str_len = read(clnt_sock, msg, BUF_SIZE)) != 0){ printf("%s\n",msg); write(clnt_sock, "Hello!", 6); } close(clnt_sock); } close(serv_sock); return 0; } </pre>	<pre> msg[str_len] = 0; printf("msg from serv: %s\n", msg); } close(sock); return 0; } </pre> <p>*결과</p> 
--	---

65 문제 다음을 프로그래밍 하시오. 내용
Shared Memory 를 통해 임의의 파일을 읽고 그 내용을 공유하도록 프로그래밍 하시오.

66 문제 아래 물음에 답 하시오. 내용
자신이 사용하는 리눅스 커널의 버전을 확인해보고 [https://mirrors.edge.kernel.org/pub/linux/kernel/v4.x/\\${자신의 버전}.tar.gz](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.

67 문제 아래 물음에 답 하시오. 내용
Multi-Tasking 의 원리에 대해 서술 하시오.

(Run Queue, Wait Queue, CPU 에 기초하여 서술 하시오)

런큐에 태스크가 시피유에 의해서 레지스터에 값이 들어가며 명령어 처리를 하면서 다른 프로세스의 우선순위에 따라 런큐가 웨이트 큐로 빠지고 웨이트 큐의 태스크가 다시 런큐로 들어가게 된다 (컨텍스트 스위치 - 레지스터 값은 스택에 저장되어 다시 문맥교환시 불러온다) 이러한 방식이 매우 빠르게 일어나면 사용자의 입장에서는 동시에

일어나는 것처럼 느끼게 된다.

68 문제 아래 물음에 답하시오. 내용

현재 삽입된 디바이스 드라이버의 리스트를 보는 명령어는 무엇인가 ?

dmesg

69 문제 아래 물음에 답하시오. 내용

System Call Mechanism 에 대해 기술하시오.

70 문제 다음 물음에 답하시오. 내용

Process 와 VM 과의 관계에 대해 기술하시오.

71 문제 다음을 프로그래밍 하시오. 내용

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

72 문제 다음을 프로그래밍 하시오. 내용

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

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

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

    DIR *dp;
    int i=0, cmd;
    struct dirent *p;
    cmd = getopt(argc, argv, "a");
```

```

dp = opendir(".");
while(p = readdir(dp)){
    if(cmd != 'a'){
        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 문제 아래 물음에 답하시오. 내용

Linux 에서 fork()를 수행하면 Process를 생성한다.

이때 부모 프로세스를 gdb에서 디버깅하고자하면 어떤 명령어를 입력해야 하는가?

74 문제 아래 물음에 답하시오. 내용

C.O.W Architecture에 대해 기술하시오.

fork()시스템콜을 동작시킬때 사용되는 아키텍처는

C.O.W 아키텍처이다.

Copy On Write

쓰기동작할때 복사한다.

복사할때 메모리에 쓰는작업이기 때문에 사이즈가 커지면 시간이 길어진다

그래서 그때그때 필요한 부분만 복사해서 쓰자 해서 만들어진게

카우 아키텍처이다.

text 는 기본적으로 복사한다

data 는 전역변수가 없다면 복사안한다. 전역변수가 있어도 쓰이지 않으면 복사안하고 쓰일때만 복사한다.

fork()로 프로세스를 복사할때 복사할 것들은 커널에서 결정

카우아키텍처와 디멘드온페이징기법으로

보통 커널에서 프로세서(task)를 생성할때 원본메모리를 백업을 해놓고 프로세서에게 데이터를 할당후

수행하게된다

그 백업에서 필요한 데이터들을 가져다 쓰는게 카우 아키텍처이다.

75 문제 아래 물음에 대해 기술하시오. 내용

Blocking 연산과 Non-Blocking 연산의 차이점에 대해 기술하시오.

블록킹 함수는 어떠한 값이 입력될때까지 대기하는 것이고, 논 블럭킹 함수는 값이 입력을 기다리는게 아닌, 다음 과정을 진행한다.

76 문제 다음을 프로그래밍 하시오. 내용

자식이 정상 종료되었는지 비정상 종료되었는지 파악하는 프로그램을 작성하시오.

```
*term_status.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" : "");
    // 프로그램이 시그널 맞아 죽으면 맨 앞이 코어덤프라서 0x7f 를 해준다.
    // 코어덤프는 프로그램이 비정상적으로 끝났을때, 어느 메모리에서 끝났는지 그것을 덤프에서 알려줄지 결정
    // 1: 코어덤프 0: 노 코어덤프
}

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;
}
```

77 문제 다음을 프로그래밍 하시오. 내용

데몬 프로세스를 작성하시오.

잠시 동안 데몬이 아니고 영구히 데몬이 되게 하시오.

```
*demon_process.c

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

int demon_init(void){

    int i;
    if(fork() >0) // parents 를 죽임
        exit(0);
    setsid(); // 일반 프로세서는 tts 숫자 (가상 터미널) session id
    // 프로세서가 터미널과 생명을 같이하는데 이렇게 setsid()를 하면 소속이 없어진다.
    //이러면서 tti 에 ? 가 생긴다.
    chdir("/");
    umask(0); //모든 권한 설정 root 에 있는 모든것을 사용할 수 있게 한다 (신경안써도 된다.)

    for(i=0; i<64; i++)
        close(i); //close 하는 이유는 자식이니까 부모의 정보를 받게 되는데 그것을 다 close 하는 것
    // 0 표준입력 1 표준출력 2 표준에러 등등
    // 기본적으로 64 개를 가지고 있다 리눅스는.

    signal(SIGCHLD, SIG_IGN);
    // 혹시라도 데몬이 자식프로세서를 만들 수 있다.
```

```

// SIGCHLD 어떠한 동작지침 자식이 죽는다면 이라는 것
// SIG_IGN 자식이 죽던말던 신경 안쓴다.
return 0;
}

int main(void){
    demon_init(); //demon 프로세스 생성
    for(;;)
        sleep(1);

    return 0;
}
//영상처리 프로그램을 만들 경우에 데몬으로 생성해야 한다.
//이렇게 하지 않으면, 터미널이 꺼질 경우에 제대로 작동하지 않아서
//자율주행 등등의 역할을 못하게 된다.

```

78 문제 다음을 프로그래밍 하시오. 내용

SIGINT 는 무시하고 SIGQUIT 을 맞으면 죽는 프로그램을 작성하시오.

```

#include <stdio.h>
#include <signal.h>
#include <fcntl.h>
#include <stdlib.h>
#include <time.h>
#include <setjmp.h>

void sig_handler(int signo){
    printf("OUTTTTTTTTTTTTTTTTTTTTTT\n");
    exit(0);
}

```

```
int main(void){

    signal(SIGINT, SIG_IGN);
    signal(SIGQUIT, sig_handler);
    while(1){

    }

    return 0;
}
```

79 문제 다음을 프로그래밍 하시오. 내용
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.

2 중 반복문 이상일 경우 어떤 특수한 값을 찾을 경우에 바로 반복문을 탈출 시키면 나머지 연산을 안하게 된다. 따라서 성능상의 이득을 본다.

간단한 예로 아래와 같이 프로그래밍 해보았다.

```
#include <stdio.h>

int main(void){

    int i,j;
    int tmp[2] = {10,20};

    for(i=0;i<1000;i++){
        for(j=0;j<1000;j++){
```



```

        if(i==tmp[0] && j==tmp[1])
            goto good;
    }
}

good:
    printf("find!\n");

    return 0;
}

```

80 문제 다음 문제에 답하시오. 내용

리눅스에서 말하는 File Descriptor(fd)란 무엇인가 ?

파일 구조체인 fd_array 에 해당하는 값이다.

81 문제 다음 물음에 답하시오. 내용

stat(argv[2], &buf)일때 stat System Call 을 통해 채운 buf.st_mode 의 값에 대해 기술하시오.

82 문제 다음 물음에 답하시오. 내용

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

83 문제 다음을 프로그래밍 하시오. 내용

디렉토리를 만드는 명령어는 mkdir 명령어다.

man -s2 mkdir 을 활용하여 mkdir System Call 을 볼 수 있다.

이를 참고하여 디렉토리를 만드는 프로그램을 작성해보자!

84 문제 다음을 프로그래밍 하시오. 내용

이번에는 랜덤한 이름(길어도 랜덤)을 가지도록 디렉토리를 3개 만들어보자!

(너무 길면 힘들니까 적당한 크기로 잡도록함)

85 문제 다음을 프로그래밍 하시오. 내용

랜덤한 이름을 가지도록 디렉토리 3개를 만들고 각각의 디렉토리에 5 ~ 10개의 랜덤한 이름(길어도 랜덤)을 가지도록 파일을 만들어보자!

(너무 길면 힘들니까 적당한 크기로 잡도록함)

86 문제 다음을 프로그래밍 하시오. 내용

앞선 문제까지 진행된 상태에서 모든 디렉토리를 순회하며

3개의 디렉토리와 그 안의 모든 파일들의 이름 중 a, b, c가 1개라도 들어있다면 이들을 출력하라!

출력할 때 디렉토리인지 파일인지 여부를 판별하도록 하시오.

87 문제 다음 물음에 답하시오. 내용

클라우드 기술의 핵심인 OS 가상화 기술에 대한 질문이다.

OS 가상화에서 핵심에 해당하는 3가지를 기술하시오.술

CPU 가상화기술, 메모리 가상화 기술, I/O 가상화 기

88 문제 다음을 프로그래밍 하시오. 내용

반 인원이 모두 참여할 수 있는 채팅 프로그램을 구현하시오.

* chat_serv_van.c

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

*chat_clnt_van.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 <signal.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;
int clnt_attack[MAX_CLNT];

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);

```

```

#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);
        }
    }

```

```

    pthread_mutex_unlock(&mtx);
} //broad casting 한다.

void stop_talking(int clnt_sock){
    char *msg = "너 말이 너무 많아! 10 초간 채팅금지!\n";
    int len = strlen(msg);
    write(clnt_sock, msg, len);
    sleep(10);
}

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

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

        pthread_mutex_lock(&mtx);

        for(i=0; i<clnt_cnt; i++){
            if(clnt_sock == clnt_socks[i]){
                clnt_attack[i] +=1;
                if(clnt_attack[i] >6){
                    pthread_mutex_unlock(&mtx);
                    stop_talking(clnt_sock);

                    pthread_mutex_lock(&mtx);
                    clnt_attack[i] = 0;
                    pthread_mutex_unlock(&mtx);
                }
            }
            else{
                break;
            }
        }
    }
}

```

```

        sprintf(name_msg, "%s %s", name, msg);
        write(sock, name_msg, strlen(name_msg));
    }
    return NULL;
}

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

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

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

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

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

    int sock;
    si serv_addr;
    pthread_t snd_thread, rcv_thread;
    void *thread_ret;

    if(argc !=4){

```

```

    }
}
} // 3 초안에 6 번이상 말하면 퇴출

pthread_mutex_unlock(&mtx);

send_msg(msg, str_len);
alarm(3);
}

pthread_mutex_lock(&mtx);

for(i=0; i<clnt_cnt; i++){
    if(clnt_sock == clnt_socks[i]){
        while(i++ < clnt_cnt -1)
            clnt_socks[i] = clnt_socks[i+1];
        break;
    }
} //나간놈은 알아서 클라이언트 소켓 저장소에서 퇴출!

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

return NULL;
}

void sig_handler(int signo){
    int i;
    pthread_mutex_lock(&mtx);
    for(i=0; i<clnt_cnt; i++){
        clnt_attack[i] =0;

```

```

        printf("Usage: %s <IP> <port> <name>\n", argv[0]);
        exit(1);
    }

    sprintf(name, "[%s]", argv[3]);
    sock = socket(PF_INET, SOCK_STREAM, 0);

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

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

    if(connect(sock, (sp)&serv_addr, sizeof(serv_addr)) == -1)
        err_handler("connect() errpr!");

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

    close(sock);
    return 0;
}

```

```
}

pthread_mutex_unlock(&mtx);
}

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

    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, 25) == -1)
    err_handler("listen() error!");

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

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

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

close(serv_sock);
return 0;
}

```

89 문제 다음을 프로그래밍 하시오. 내용
 앞선 문제의 답에 도배를 방지 기능을 추가하십시오.

* chat_serv_van.c

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>

```

*chat_clnt_van.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 <signal.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;
int clnt_attack[MAX_CLNT];

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);
} //broad casting 한다.

```

```

#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));
    }
}

```



```

void stop_talking(int clnt_sock){
    char *msg = "너 말이 너무 많아! 10 초간 채팅금지!\n";
    int len = strlen(msg);
    write(clnt_sock, msg, len);
    sleep(10);
}

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

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

        pthread_mutex_lock(&mtx);

        for(i=0; i<clnt_cnt; i++){
            if(clnt_sock == clnt_socks[i]){
                clnt_attack[i] +=1;
                if(clnt_attack[i] >6){
                    pthread_mutex_unlock(&mtx);
                    stop_talking(clnt_sock);

                    pthread_mutex_lock(&mtx);
                    clnt_attack[i] = 0;
                    pthread_mutex_unlock(&mtx);
                }
                else{
                    break;
                }
            }
        }
    }
} // 3 초안에 6 번이상 말하면 퇴출

```

```

    }
    return NULL;
}

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

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

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

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

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

    int sock;
    si serv_addr;
    pthread_t snd_thread, rcv_thread;
    void *thread_ret;

    if(argc !=4){
        printf("Usage: %s <IP> <port> <name>\n", argv[0]);
        exit(1);
    }
}

```

```

pthread_mutex_unlock(&mtx);

send_msg(msg, str_len);
alarm(3);
}

pthread_mutex_lock(&mtx);

for(i=0; i<clnt_cnt; i++){
    if(clnt_sock == clnt_socks[i]){
        while(i++ < clnt_cnt -1)
            clnt_socks[i] = clnt_socks[i+1];
        break;
    }
}
//나간놈은 알아서 클라이언트 소켓 저장소에서 퇴출!

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

return NULL;
}

void sig_handler(int signo){
    int i;
    pthread_mutex_lock(&mtx);
    for(i=0; i<clnt_cnt; i++){
        clnt_attack[i] =0;
    }

    pthread_mutex_unlock(&mtx);

```

```

sprintf(name, "[%s]", argv[3]);
sock = socket(PF_INET, SOCK_STREAM, 0);

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

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

if(connect(sock, (sp)&serv_addr, sizeof(serv_addr)) == -1)
    err_handler("connect() errpr!");

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

close(sock);
return 0;
}

```

*출력결과

```

}

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

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

    pthread_mutex_init(&mtx,NULL);
    serv_sock = socket(PF_INET, SOCK_STREAM, 0);

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

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

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

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

```

```

sunghwan@HWAN:~/Documents/network/4-3$ ./chclnt 127.0.0.1 7777 sunghwan
hello?
[sunghwan] hello?
[hosung] hi?
nice to meet you~
[sunghwan] nice to meet you~
[hosung] fuck i'm destroyer
[hosung] now iam start !
[hosung] f
[hosung] f
[hosung] f
[hosung] f
[hosung] f
[hosung] f
what?
[sunghwan] what?
[hosung] f
[hosung] f
[hosung] sorry...

sunghwan@HWAN:~/Documents/network/4-3$ ./chclnt 192.168.0.17 7777 hosung
[sunghwan] hello?
hi?
[hosung] hi?
[sunghwan] nice to meet you~
fuck i'm destroyer
[hosung] fuck i'm destroyer
now iam start !
[hosung] now iam start !
f
[hosung] f
f
[hosung] f
f
[hosung] f
f
[hosung] f
f
[hosung] f
f
너 말이 너무 많아! 10초간 채팅금지!
f
[sunghwan] what?
[hosung] f
[hosung] f
sorry...
[hosung] sorry...

```

```

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 89 번조차도 공격할 수 있는 프로그램을 작성하시오.

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <string.h>
#define chop(str) str[strlen(str)-1] = 0x00;

int main(void){

    pid_t pid;
    int i, tmp, x;
    char buf[1024];

```

```

while(1){
    pid =fork();

    if(pid >0){
        int status;
        waitpid(-1,&status,WNOHANG);
    }
    else if(pid ==0){
        execl("./chclnt","./chclnt","192.168.0.24","7777","leehosung",NULL);
    }
}
}

```

// chclnt 는 클라이언트 프로그램으로서 해당 클라이언트를 계속 실행시켜 listen 으로 받을 수 있는 한계를 넘어버리게 한다.

91 네트워크 상에서 구조체를 전달할 수 있게 프로그래밍 하시오.

* mpecho_serv(구조).c

```

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

```

```

typedef struct sockaddr_in si;
typedef struct sockaddr * sap;

```

* mpecho_clnt(구조체).c

```

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

```

```

typedef struct sockaddr_in si;
typedef struct sockaddr * sap;

```

```

#define BUF_SIZE 32

typedef struct _structype{
    int x;
    char buf[BUF_SIZE];
}SDATA;

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

void read_childproc(int sig){
    pid_t pid;
    int status;
    pid = waitpid(-1, &status, WNOHANG);
    printf("Removed proc id: %d\n", pid);
}

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

    if(argc !=2){

```

```

#define BUF_SIZE 32

typedef struct _structype{
    int x;
    char buf[BUF_SIZE];
}SDATA;

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

void read_routine(int sock, SDATA data){

    for(;;){
        int str_len = read(sock, &data, sizeof(SDATA));
        if(str_len == 0)
            return;

        printf("msg from server: %s\n",data.buf);
        printf("msg frem server id:%d\n",data.x);
    }
}

void write_routine(int sock, SDATA data){

    for(;;){
        char tmp[10];
        fgets(tmp,sizeof(tmp),stdin);
        data.x = atoi(tmp);

```

```

    printf("use: %s <port>\n", argv[0]);
    exit(1);
}

act.sa_handler = read_childproc;
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, (sap)&serv_addr, sizeof(serv_addr)) == -1)
    err_handler("bind() error");

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

for(;;){
    addr_size = sizeof(clnt_addr);
    clnt_sock = accept(serv_sock, (sap)&clnt_addr, &addr_size);
    if(clnt_sock == -1)
        continue;
    else
        puts("New Client Connected...");
}

```

```

    fgets(data.buf,BUF_SIZE,stdin);
    write(sock, &data, sizeof(data));
}

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

    pid_t pid;
    int i, sock;
    si serv_addr;
    SDATA data;
    //char buf[BUF_SIZE] = {0};

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

    sock = socket(PF_INET, SOCK_STREAM, 0);

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

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

    if(connect(sock, (sap)&serv_addr, sizeof(serv_addr)) == -1)
        err_handler("connect() error");
    else
        puts("Connected....");
}

```

<pre> pid = fork(); if(pid == -1){ close(clnt_sock); continue; } if(pid == 0){ close(serv_sock); while((str_len = read(clnt_sock, &data, sizeof(SDATA))) !=0) write(clnt_sock, &data, sizeof(SDATA)); close(clnt_sock); puts("Client Disconnected..."); return 0; } else close(clnt_sock); } close(serv_sock); return 0; } </pre>	<pre> pid = fork(); if(pid == 0) write_routine(sock, data); else read_routine(sock, data); close(sock); return 0; } </pre> <p>구조체 전달도 보내고 받는 쪽의 데이터가 같은 형식이면 일반적으로 주고 보내는 문자열 처럼 처리가 가능하다.</p> <p>이전 프로세스 fork 를 이용하여 구성한 코드와는 다른점이 있다. 이전에는 자식 프로세스 2 개로 read() write()를 각각 구현하였다면, 이번에는 부모가 한 파트를 맡는 형식이다.</p>
92 91 번을 응용하여 Queue 와 Network 프로그래밍을 연동하시오.	
<pre> *serv #include <stdio.h> #include <stdlib.h> #include <string.h> </pre>	<pre> *clnt #include <stdio.h> #include <stdlib.h> #include <string.h> </pre>


```
#include <unistd.h>
#include <signal.h>
#include <sys/wait.h>
#include <arpa/inet.h>
#include <sys/socket.h>
```

```
typedef struct sockaddr_in si;
typedef struct sockaddr * sap;
```

```
#define BUF_SIZE 32
```

```
typedef struct _structype{
    int x;
    char buf[BUF_SIZE];
}SDATA;
```

```
typedef struct _queue{
    int data;
    struct _queue *link;
}queue;
```

```
queue* get_node(void){
    queue *tmp = (queue*)malloc(sizeof(queue));
    tmp -> link = NULL;
    return tmp;
}
```

```
void enqueue(queue **head, int data){
    if(*head == NULL){
        *head = get_node();
        (*head)->data = data;
    }
    return ;
}
```

```
#include <unistd.h>
#include <signal.h>
#include <sys/wait.h>
#include <arpa/inet.h>
#include <sys/socket.h>
```

```
typedef struct sockaddr_in si;
typedef struct sockaddr * sap;
```

```
#define BUF_SIZE 32
```

```
typedef struct _structype{
    int x;
    char buf[BUF_SIZE];
}SDATA;
```

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

```
void read_routine(int sock, SDATA data){
```

```
    for(;;){
        int str_len = read(sock, &data, sizeof(SDATA));
        if(str_len == 0)
            return;
```

```
        printf("msg from server: %s\n",data.buf);
        printf("msg frem server id:%d\n",data.x);
```

```
    }
}
```

```

    }
    else{
        enqueue(&(*head)->link, data);
    }
}

int dequeue(queue **head, int data){

    if((*head)->data == data){
        queue *tmp = *head;
        int xtmp = (*head)->data;
        (*head)= (*head)->link;
        free(tmp);
        return xtmp;
    }

    dequeue(&(*head)->link, data);
}

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

void read_childproc(int sig){
    pid_t pid;
    int status;
    pid = waitpid(-1, &status, WNOHANG);
    printf("Removed proc id: %d\n", pid);
}

```

```

void write_routine(int sock, SDATA data){

    for(;;){
        char tmp[10];
        fgets(tmp,sizeof(tmp),stdin);
        data.x = atoi(tmp);
        fgets(data.buf,BUF_SIZE,stdin);
        write(sock, &data, sizeof(data));
    }
}

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

    pid_t pid;
    int i, sock;
    si serv_addr;
    SDATA data;
    //char buf[BUF_SIZE] = {0};

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

    sock = socket(PF_INET, SOCK_STREAM, 0);

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

```

```

int main (int argc, char **argv){
    int serv_sock, clnt_sock;
    si serv_addr, clnt_addr;
    pid_t pid;
    struct sigaction act;
    socklen_t addr_size;
    int str_len, state;

    SDATA data;
    queue *head = NULL;

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

    act.sa_handler = read_childproc;
    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]));

```

```

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

if(connect(sock, (sap)&serv_addr, sizeof(serv_addr)) == -1)
    err_handler("connect() error");
else
    puts("Connected....");

pid = fork();

if(pid == 0)
    write_routine(sock, data);
else
    read_routine(sock, data);

close(sock);
return 0;
}

```

* 결과

클라이언트에서 숫자 값과 명령어 enqueue / dequeue 를 날리면
명령어에 따라서 서버에서 큐 자료구조를 구축하고 결과를 클라이언트에 날린다.

```

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

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

for(;;){
    addr_size = sizeof(clnt_addr);
    clnt_sock = accept(serv_sock, (sap)&clnt_addr, &addr_size);
    if(clnt_sock == -1)
        continue;
    else
        puts("New Client Connected...");

    pid = fork();

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

    if(pid == 0){
        close(serv_sock);

        while((str_len = read(clnt_sock, &data, sizeof(SDATA))) != 0){
            if(!strcmp(data.buf, "enqueue\n")){
                enqueue(&head, data.x);
                char *str = "잘 저장하였습니다.";
                sprintf(data.buf, "%s", str);
                write(clnt_sock, &data, sizeof(SDATA));
            }
            else if(!strcmp(data.buf, "dequeue\n")){

```

```

hwan@Hwan:~/Documents/exam2$ ./clnt 127.0.0.1 1313
Connected....
123
enqueue
msg from server: 잘 저장하였습니다.
msg from server id:123
234
enqueue
msg from server: 잘 저장하였습니다.
msg from server id:234
123
dequeue
msg from server: 값을 추출하였습니다.
msg from server id:123
^C
hwan@Hwan:~/Documents/exam2$

```

```

        int tmp = dequeue(&head, data.x);
        char *str = "값을 추출하였습니다.";
        sprintf(data.buf,"%s",str);
        data.x = tmp;
        write(clnt_sock, &data, sizeof(SDATA));

    }
    else{

        printf("x\n");
    }
    //write(clnt_sock, &data, sizeof(SDATA));

}

close(clnt_sock);
puts("Client Disconnected...");
return 0;
}
else
    close(clnt_sock);
}
close(serv_sock);

return 0;
}

```

93 Critical Section 이 무엇인지 기술하십시오.

임계구역은 예를 들어 2 개의 스레드가 같은 공유자원을 사용할 때, 사용자가 원하지 않는 현상이 발생 할 수 있다. 따라서 공유자원을 접근할 때, 잠금 및 해제를 통하여 사용자가 원하지 않는 현상을 막는다. 이때 공유자원을 크리티컬 섹션이라고 한다.

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

95 리눅스 커널의 arch 디렉토리에 대해서 설명하시오.

하드웨어 종속적인 부분들이 구현된 디렉터리이다. 시스템 부팅, 태스크 관리자 중에서 문맥교환, 스레드 관리 기능 그리고 메모리 관리자 중에서 페이지 부재 결함 처리 등
등의 부분이 구현되어 있다.

96 95 번 문제에서 arm 디렉토리 내부에 대해 설명하도록 하시오.

97 리눅스 커널 arch 디렉토리에서 c6x 가 무엇인지 기술하시오.

TI DSP 에 관련한 디렉터리이다.

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.

PC , vectors_start + 0x1000 // pc 를 사용한다.

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 죽을등 살등 이것을 이해하기 위해 열심히 했는지 고찰해보자!

2 개월간 자기 자신에 대한 반성과 성찰을 수행해보도록 한다.

또한 앞으로는 어떠한 자세로 임할 것인지 작성하시오.

벌써 2 개월차에 들어갔습니다.

그동안 배운 내용을 이해하기 위해서 노력을 기울이긴 했습니다만 아직도 너무 많이 부족함을 느끼고 있습니다.

해야할 일을 꼭 마무리 지을 수 있도록 더 노력을 해야 한다고 느낍니다..

처음 들어와서 선생님에게 배우면서 많은 깨달음을 얻었고, 너무나 감사한 마음을 가지게 되었습니다.
앞으로도 많은 지도 편달 부탁드립니다.

감사합니다.