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

강사 – **Innova Lee(이상훈)**
gcccompil3r@gmail.com
학생 hyungjun Yu(유형준)
love592946@nave.rcom

2개월차 시험 유 형 준

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

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

A:
369serv.c

```
#include "game.h"
#include <malloc.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>
#include <pthread.h>
#include <arpa/inet.h>
#include <sys/socket.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];
int data[MAX_CLNT];
int dat=0;
int thread_pid[MAX_CLNT];
int idx;
int cnt[MAX_CLNT];
int lum;
int totalcnt=0;
pthread_mutex_t mtx;

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

```
}

void sig_handler(int signo)
{
        int i;

        printf("Time Over!₩n");

        pthread_mutex_lock(&mtx);

        for(i = 0; i < clnt_cnt; i++)
                if(thread_pid[i] == getpid())
                {
                        cnt[i] += 1;
                        totalcnt +=1;
                }

        pthread_mutex_unlock(&mtx);

        alarm(3);
}



void proc_msg(Data *data, int k)
{
        int i;
        int cmp = atoi(data->ms);
        char smsg[64] = {0};
        char numc[1024];
        int ncnt=0;
        char clap[32]="clap! ";

        dat += 1;
        sprintf(numc,"%d",dat);

        pthread_mutex_lock(&mtx);
        totalcnt +=1;
        cnt[k] += 1;
        sprintf(smsg,"You Player[%d]₩n",k);
        write(clnt_socks[k],smsg,strlen(smsg));

        /*this logic count number 3 or 6 or 9 */
        for(i=0;numc[i];i++)
        {
                if(numc[i] =='3' || numc[i] == '6' || numc[i] == '9')
                {
                        ncnt+=1;
```

```
                }
        }

                /*lum is data->signal (ctrl+c count) */
                lum=atoi(data->signal);

/* in ncnt count of 3 or 6 or 9*/
        if(ncnt != 0)
        {

                printf("369 = %dWn",lum);
                if(lum == ncnt)
                {
                        sprintf(smsg,"Next turn %dWn",cmp);

                        /*lum = Ctrl + c count
                        this is a clap display*/
                        for(i=0; i<lum; i++)
                        {
                        write(clnt_socks[k],clap,strlen(clap));
                        }

                        printf("player[%d]cnt = %dWn",k, cnt[k]);
                        printf("totalcnt = %dWn",totalcnt);

                }
                else if(lum != ncnt)
                {
                        sprintf(smsg,"youloose %dWn",cmp);
                        for(i=0; i<lum; i++)
                        {
                        write(clnt_socks[k],clap,strlen(clap));
                        }
                        printf("player[%d]cnt = %dWn",k, cnt[k]);
                        printf("totalcnt = %dWn",totalcnt);
                        lum=0;
                }

        }
        else
        {
                for(i=0; i<lum; i++)
                {

                        write(clnt_socks[k],clap,strlen(clap));
                }

                if(dat > cmp){
```

```
                        sprintf(smsg, "youloose %dWn", cmp);
                        printf("player[%d]cnt = %dWn",k, cnt[k]);
                        printf("totalcnt = %dWn",totalcnt);
                }
                else if(dat < cmp){
                        sprintf(smsg, "you loose %dWn", cmp);
                        printf("player[%d]cnt = %dWn",k, cnt[k]);
                        printf("totalcnt = %dWn",totalcnt);
                }
                else
                {
                        strcpy(smsg, "Next turnWn");
                        printf("player[%d]cnt = %dWn",k, cnt[k]);
                        printf("totalcnt = %dWn",totalcnt);
                }
        }
        strcat(smsg, "Input Number: Wn");
        write(clnt_socks[k], smsg, strlen(smsg));



        pthread_mutex_unlock(&mtx);
}

void *clnt_handler(void *arg)
{
        int clnt_sock = *((int *)arg);
        int str_len = 0, i;
        char msg[BUF_SIZE] = {0};
        char pattern[BUF_SIZE] = "Input Number: Wn";
        signal(SIGALRM, sig_handler);

        pthread_mutex_lock(&mtx);
        thread_pid[idx++] = getpid();

        Data *data;
        data =(Data *)malloc(sizeof(Data));

        i = idx - 1;
        printf("i = %dWn", i);
        write(clnt_socks[i], pattern, strlen(pattern));
        pthread_mutex_unlock(&mtx);
        alarm(3);

        while(read(clnt_sock,data,sizeof(Data)) !=0)
        {
                alarm(0);
                proc_msg(data,i);
```

```
                alarm(2);
        }

        pthread_mutex_lock(&mtx);

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

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

        return NULL;
}

int main(int argc, char **argv)
{
        int serv_sock, clnt_sock;
        si serv_addr, clnt_addr;
        socklen_t addr_size;
        pthread_t t_id;
        int idx = 0;

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

        srand(time(NULL));

        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, 2) == -1)
                err_handler("listen() error");

        for(;;)
        {

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


                thread_pid[idx++] = getpid();

                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: %sWn",
inet_ntoa(clnt_addr.sin_addr));

        }

        close(serv_sock);

        return 0;
}
```

369clnt.c

```
#include "game.h"
#include <signal.h>
```

```
#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 <malloc.h>

#define BUF_SIZE        128

typedef struct sockaddr_in    si;
typedef struct sockaddr *    sp;
int sign;
char msg[BUF_SIZE];
int sock;

void my_sig(int signo)
{

        char buf[BUF_SIZE];
        int len;
        sign +=1;
        len = strlen(buf);
}


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

void *send_msg(void *arg)
{
        sock = *((int *)arg);
        char msg[BUF_SIZE];
        Data *data;
        data = (Data *)malloc(sizeof(Data));

        for(;;)
        {
                signal(SIGINT,my_sig);
                read(0,data->ms,BUF_SIZE);
                sprintf(data->signal,"%d",sign);
                write(sock,data,sizeof(Data));
```

```
                sign=0;
        }

        return NULL;
}

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

        Data *data;
        data = (Data *)malloc(sizeof(Data));

        write(1,"369game start₩n",15);
        for(;;)
        {
                str_len = read(sock,data->ms, BUF_SIZE - 1);

                data->ms[str_len] = 0;
                fputs(data->ms,stdout);
        }

        return NULL;
}

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

        sock = socket(PF_INET, SOCK_STREAM, 0);

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

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

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

        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;
}
```

game.h

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

struct data
{
    char signal[128];
        char ms[128];
        };
        typedef struct data Data;
```

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

A: CPU 메모리 디스크 네트워크 터미널

4. Unix 계열의 모든 OS는 모든 것을 무엇으로 관리하는가 ?
A; FILE

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

* 사용자 임의대로 재구성이 가능하다.
  -> C언어로도 가능하고 오픈소스 , 커스터마이징이 가능하다.

* 열약한 환경에서도 HW 자원을 적절히 활용하여 동작한다.
  -> 마이크로커널식(탈부착)이라서 오래되거나 후진 컴퓨터에서도 작동한다

* 커널의 크기가 작다.
  -> 커널의 크기가 윈도우 커널보다 작다.

* 완벽한 멀티유저, 멀티태스킹 시스템
  -> 뛰어난 스케줄링 시스템으로 멀티가 된다.

* 뛰어난 안정성
  -> 고성능 네트워크 장비에 사용되는 것으로 증명. 구글도 쓴다.

* 빠른 업그레이드
  -> 많은 사람들이 만들고 수정하는 등 피드백이 빠르다.

* 강력한 네트워크 지원
  -> TCP/IP,BSP

* 풍부한 소프트웨어
  -> 굉장히 많은 업체에서 사용한다.

6. 32bit System에서 User와 Kernel의 Space 구간을 적으시오.

A:32비트 CPU의 경우 각 프로세스에게 4GB크기의 가상공간을 할당.
0~3GB의 공간을 User공간으로 사용하고, 나머지 3~4GB를 커널공간으로 사용.

7. Page Fault 가 발생했을때
운영체제가 어떻게 동작하는지 기술하시오.

8. 리눅스 실행 파일 포맷이 무엇인지 적으시오.

A: ELF

9. 프로세스와 스레드를 구별하는 방법에 대해 기술하시오.

A: 프로세스는 동작중인 프로그램
스레드는 수행의 단위

프로세스는 파일의 형태로 존재하고 있는 프로그램이 수행되기 위해서는
리눅스 커널로부터 CPU 등의 자원을 할당받을 수 있는 동적인 객체.

새로운 프로세스를 생성하면 자식프로세스와 부모프로세스는 서로 다른 주소 공간을
갖는다.
새로운 스레드를 생성하면 자식 스레드와 부모스레드는 서로 같은 주소공간을
공유한다.

Task_struct 자료 구조 중에서 수행 이미지를 공유하는가
같은 스레드 그룹에 속해 있는가 등의 여부에 따라 해석 차이가 있다.

10. Kernel 입장에서 Process 혹은 Thread를 만들면 무엇을 생성하는가 ?

A: task_struct 구조체

2개월차 시험 유 형 준

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

A: 현재 태스크의 task_struct 구조체를 가리킬 수 있게 해준다.

12. Memory Management 입장에서 Process와 Thread의 핵심적인 차이점은 무엇인가 ?

A; 새로운 프로세스를 생성하면 자식프로세스와 부모프로세스는 서로 다른 주소 공간을
    갖는다.
    새로운 스레드를 생성하면 자식 스레드와 부모스레드는 서로 같은 주소공간을
    공유한다.

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

A: CONTEXT SWITCHING을 하기 위한 구조이다.(레지스터 정보가 있다)

14. 리눅스 커널의 스케쥴링 정책중 Deadline 방식에 대해 기술하시오.

15. TASK_INTURRUPTIBLE과 TASK_UNINTERRUPTIBLE은 왜 필요한지 기술하시오.

TASK_INTERRUPTIBLE: 프로세스는 잠든 상태이며, 사건이 일어나기를 기다리고 있다. 프로세스는 시그널이 인터럽트하도록 열려 있다. 일단 시그널을 받거나 명시적인 깨어나기 호출로 깨어나면, 프로세스 상태가 TASK_RUNNING으로 전이한다.

TASK_UNINTERRUPTIBLE: 프로세스 상태는 TASK_INTERRUPTIBLE과 비슷하다. 하지만 이 상태에서 프로세스는 시그널을 처리하지 않는다. 몇몇 중요한 작업을 완료하는 중간에 있기에 이 상태에 있을 경우 프로세스를 인터럽트하는 상황이 그리 바람직하지 않을 가능성도 있다. 기다리고 있던 사건이 발생할 때, 프로세스는 명시적인 깨어나기 호출로 깨어난다.

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

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 만들어냈

다.
이 Task는 어디에 위치하는 것이 좋을까 ?
그리고 그 이유를 적으시오.

18. 15번 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.

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

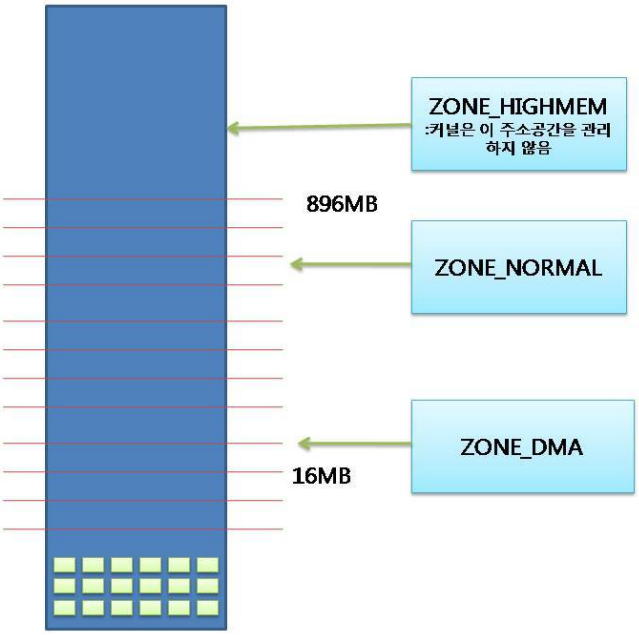
A: UMA구조라면 한개의 뱅크가 존재. NUMA구조라면 복수개의 뱅크가 존재.
(뱅크란 접근 속도가 같은 메모리의 집합 -> 뱅크를 표현하는 구조가 노드)
UMA구조는 한 개의 노드가 존재할 것이며, 이 노드는 리눅스의 전역변수인 contig_page_data를 통해 접근 가능하다. NUMA구조는 복수개의 노드가 존재할 것이며, 이를 관리하귀 위해서는 pgdat_list라는 이름의 배열을 통해 접근이 가능.

```
typedef struct pglist_data{
struct zone node_zones[MAX_NR_ZONES];
struct zonelist node_zonelists[GFP_ZONETYPES];
Int nr_zones;
struct page *node_mem_map;
struct bootmem_data *bdata;
unsigned long node_start_pfn;
unsigned long node_present_pages;
unsigned long node_spanned_pages;
int node_id;
wait_queue_head_t kswapd_wait;
wait_queue_head_t pfmemalloc_wait;
...
}pg_data_t;
```

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

21. ZONE_HIGHMEM 에 대해 아는대로 기술하시오.

A: 상위 메모리를 포함한다. 상위 메모리는 커널 주소 공간으로 영구적으로 매핑되지 않는다. 동적으로 메핑되는 페이지. X > 896MB.



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

A: page frame(4KB) / 페이지 구조체

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

버디 메모리 할당 기술은 2의 거듭제곱 값(예를 들면, 2x. 여기서 x는 숫자)으로 메모리를 할당한다. 따라서, 프로그래머는 x값의 상한선을 결정하거나 구할 수 있는 코드를 작성해야 한다. 예를 들면, 시스템이 2000K의 물리적인 메모리를 가지고 있다면, 220(1024K)이 할당할 수 있는 가장 큰 블록이기 때문에 x값의 상한선은 20이 될 것이다. 이는 단일 청크에 물리적 메모리 전부를 할당하는 것이 불가능하기 때문이다. 남은 976K 메모리는 좀더 작은 블록들로 할당이 되어야 한다.
상한선(앞으로는 상한선을 u 로 표기하겠음)을 결정한 뒤에는, 프로그래머는 할당될 수 있는 가장 작은 메모리 블록인 하한선을 결정해야 한다. 이 하한선은 저장하는데 발생되는 오버헤드를 최소화하고 비어있는 메모리 공간을 최소화 하기 위해서 필요하다. 이 하한선이 없다면, 많은 프로그래머들은 1K나 2K 같은 작은 블록의 메모리를 요구할 것이고, 시스템은 할당되고 할당되지 않은 블록들을 기억하기 위해서 많은 공간을 낭비하게 될 것이다. 전형적으로 이 숫자(12 같은, 212 = 4K 블록에 할당된 메모리)는 공간의 낭비를 최소화할 수 있을만큼 충분히 작은 적당한 숫자이고, 과도한 오버헤드를 피할 수 있을만큼 충분히 큰 숫자이기도 하다. 앞으로는 이 하한선을 l 로 부르도록 하겠다.
이제 우리는 한계선을 갖게 되었으니, 프로그램이 메모리 요청을 하게 되면 무슨 일이 일어나는지 보도록 하겠다. 216 = 64K인 l = 16 을, 할당할 수 있는 가장 큰 블록인 220 = 1024K, u = 20 을 이 시스템에 요청한다.

다음의 표는 다양한 메모리 요청에 대한 시스템의 가능한 상태를 보여준다.

	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K	64K
$t=0$	1024K															
$t=1$	A-64K	64K	128K		256K				512K							
$t=2$	A-64K	64K	B-128K		256K				512K							
$t=3$	A-64K	C-64K	B-128K		256K				512K							
$t=4$	A-64K	C-64K	B-128K		D-128K		128K		512K							
$t=5$	A-64K	64K	B-128K		D-128K		128K		512K							
$t=6$	128K		B-128K		D-128K		128K		512K							
$t=7$	256K				D-128K		128K		512K							
$t=8$	1024K															

이 할당은 아래의 방식으로 발생할 수 있다.
1. 프로그램 A가 34K ~ 64K 크기의 메모리를 요청한다.
2. 프로그램 B가 66K ~ 128K 크기의 메모리를 요청한다.
3. 프로그램 C가 35K ~ 64K 크기의 메모리를 요청한다.
4. 프로그램 D가 67K ~ 128K 크기의 메모리를 요청한다.
5. 프로그램 C가 메모리를 해제한다.
6. 프로그램 A가 메모리를 해제한다.
7. 프로그램 B가 메모리를 해제한다.
8. 프로그램 D가 메모리를 해제한다.

보다시피, 메모리 요청이 발생되었을 때 다음과 같은 일들이 벌어졌다.
? 메모리가 할당되면
1. 적당한 크기의 메모리 슬롯을 찾는다.(최소한 요청된 메모리와 동일하거나 큰 2k 블록)
1. 적당한 크기의 메모리 슬롯이 발견되면, 프로그램에 할당한다.
2. 적당한 크기의 메모리 슬롯이 발견되지 않으면, 적당한 메모리 슬롯을 만들기를 시도한다. 시스템은 아래의 일들을 시도한다.
1. 요청된 메모리 크기보다 크게 절반씩 빈 메모리 슬롯을 자른다.
2. 하한선에 도착하게 되면, 해당 메모리(하한선 크기의 메모리)를 할당한다.

3. 다시 첫 번째 단계로 돌아간다.(적당한 크기의 메모리를 찾기 위해서)
4. 적당한 메모리 슬롯이 발견될 때 까지, 이 과정을 반복한다.
? 메모리가 해제되면
1. 메모리 블록을 해제한다.
2. 주변의 블록들을 살펴 본다 - 주변 블록들도 해제된 상태인가?
3. 만약 그렇다면, 두 메모리 블록을 조합하고 다시 두 번째 단계로 돌아간다. 그리고 해제된 모든 메모리들이 상한선에 도달할 때까지 이 과정을 반복하거나, 해제되지 않은 주변 블록들을 마주칠 때까지 반복한다.
메모리를 해제 하는 이 방법은 log2(u/l)과 같은 가장 효과적인 간결화 숫자를 이용하면 간결화가 상대적으로 빠르게 이루어져서 꽤 능률적이다. (log2(u)- log2(l))
전형적으로 버디 메모리 할당 시스템은 사용되거나 사용되지 않은 두 가지 상태로 메모리 블록들을 나누는 것을 뜻하는 이진 트리를 이용해서 구현된다.
하지만, 여전히 내부 단편화의 문제점은 남아 있다. 여러 경우에 있어서, 내부 단편화의 양을 최소화 하는 것은 필수적이다. 이 문제점은 slab allocation에 의해서 해결될 수 있다.

24. 21번에 이어 내부 단편화를 최소화 하기 위해 Buddy에서 Page를 받아 Slab 할당자를 사용한다.
Slab 할당자는 어떤식으로 관리되는지 기술하시오.

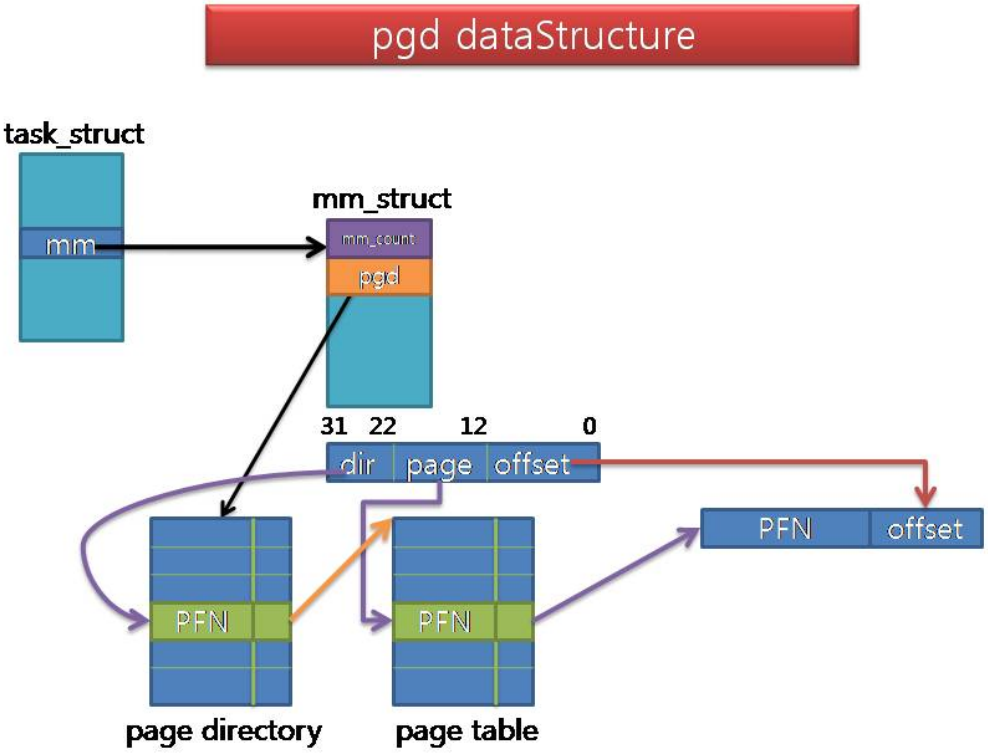
A: 1.만약 kmalloc을 이용해 kernel이 16byte를 할당받으려고 한다고 하자
2.page의 단위가 4kb이기 때문에 고작 16byte를 위해 4kb를 할당해야하나?
3.4096 -16 = 4080byte는 어떻게 될까 ? 내부 단편화의 문제가 발생.
4.이를 막기 위한 방법이 바로 슬랩할당자.
5.슬랩 할당자는 미리 할당해 놓은 작은 메모리 조각을 kmalloc의 요청에 따라서 요청한 양에 가장 가까운 메모리 조각을 반환
6.16byte를 요청했을 경우 가장 가까운 slab의 조각 32byte를 사용하수 있도록 반납해 준다.
7.slab은 kmem_cache_s 구조체와 kmem_list3에 의해 관리됨.
Slabs_partial: 슬랩내의 오브젝트가 사용 되는 것도 있고 아닌 것도 있는 혼재상황.
Slabs_full: 슬랩내의 오브젝트가 전부 사용 중
Slab_free: 슬랩내의 오브젝트가 empty인 것.
8.캐시는 관리가 필요한 오브젝트 종류별로(tast_struct,file,inode 등) 작성되고 그 오브
젝트의 슬랩을 관리
9.슬랩은 하나 이상의 연속된 물리 페이지로 구성되어 있으며 일반적으로 하나의 페이지로 구성됨. 캐시는 이러한 슬랩들의 복수개로 구성
10.커널이 시스템을 초기화 할때 kmem_cache_init 함수를 통하여 자주 사용되는 커널의 오브젝트들의 크기를 고려하여 일반적으로 사용할 목적으로 추가적인 캐시들생성.
11.커널은 위의 크기의 object로 이루어진 캐시를 미리 할당하여 메모리에 유지하고 있으며 사용자의 요구가 들어왔을 경우 위의 크기 중 가장 가깝게 큰 수로 올림하여 할당하게 되므로 페이지 단위로 관리하는 것 보다 내부 단편화를 줄이게 되며 시스템의 성능을 향상시키게 된다.

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)

A: task_struct에서 태스크의 메모리와 관련된 내용은 mm이라는 이름의 필드에 담겨있다.
    mm_struct(~/include/linux/mm_types.h)라는 구조체를 가리킨다.

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)

A: vm_area_struct.



27. 프로그램을 실행한다고 하면 fork(), execve()의 콤보로 이어지게 된다.
    이때 실제 gcc *.c로 컴파일한 a.out을 ./a.out을 통해 실행한다고 가

정한다.
    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.

30. MMU(Memory Management Unit)의 주요 핵심 기능을 모두 적고 간략히 설명하시오.
MMU는 가상 메모리(virtural memory)를 활성화시키기 위해 필요한 자원들을 제공하고 있기 때문에 어플리케이션의 프로그래밍을 단순화 시켜준다.
가상 메모리란 시스템에 붙어 있는 물리 메모리에 독립적인 추가의 메모리 공간을 말한다.
MMU는 가상 공간에서 컴파일된 프로그램과 데이터의 주소를 그 프로그램이 실제 주메모리안에 저장되어 있는 물리 주소로 바꾸어 주는 변환 장치처럼 동작한다.

31. 하드디스크의 최소 단위를 무엇이라 부르고 그 크기는 얼마인가 ?

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 ?

 A:디스크의 실린더, 헤드의 수, inode 목록의 헤더, 사용하지 않는 블록에 대한 정보, 화일시스템의 기본 크기, 형태에 대한 정보가 들어있는 디스크 블록

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.

TASK_INTERRUPTIBLE: 프로세스는 잠든 상태이며, 사건이 일어나기를 기다리고 있다. 프로세스는 시그널이 인터럽트하도록 열려 있다. 일단 시그널을 받거나 명시적인 깨어나기 호출로 깨어나면, 프로세스 상태가

TASK_RUNNING으로 전이한다.

TASK_UNINTERRUPTIBLE: 프로세스 상태는 TASK_INTERRUPTIBLE과 비슷하다. 하지만 이 상태에서 프로세스는 시그널을 처리하지 않는다. 몇몇 중요한 작업을 완료하는 중간에 있기에 이 상태에 있을 경우 프로세스를 인터럽트 하는 상황이 그리 바람직하지 않을 가능성도 있다. 기다리고 있던 사건이 발생할 때, 프로세스는 명시적인 깨어나기 호출로 깨어난다.

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

A: 1.잘못된 명령이나 데이터를 사용할 때 발생하며, 트랩(Trap)이라고도 부름
   2.명령어 잘못에 의한 인터럽트: 프로그램에서 명령어를 잘 못 사용한 경우
   3.프로그램 인터럽트: 0으로 나누거나 over flwo가 발생한 경우

38. 35번에서 분류한 녀석들의 특징에 대해 기술하시오.

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

A: 소켓통신..?

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

A: inode 구조체

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

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

A: vm_area(vm_file, vm_offset 등)

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

A: union

44. 커널 내부에서 메모리 할당이 필요한 이유는 무엇인가 ?

A:커널도 프로그램이기 때문에

45. 메모리를 불연속적으로 할당하는 기법은 무엇인가 ?

A: 블록체인기법,인덱스 블록 기법, FAT 등

46. 메모리를 연속적으로 할당하는 기법은 무엇인가 ?

A:페이징

47. Mutex 와 Semaphore 의 차이점을 기술하시오.

A: Mutex - 상호배제라고도 한다.
Critical section을 가진 스레드들의 running time이 서로 겹치지 않게 각각 단독으로 실행되게 하는 기술. 다중 프로세스들의 공유 리소스에 대한 접근을 조율하기 위해 locking과 unlocking을 사용. 즉, 쉽게 말해서 뮤텍스 객체를 두 스레드가 동시에 사용 할 수 없다는 의미.

Semaphore - 리소스의 상태를 나타내는 카운터. 일반적으로 긴 시간을 확보하는 리소스에 대해 이용. 경쟁적으로 사용하는 다중프로세스에서 행동을 조정하거나 동기화 시키는 기술. 커널의 한 지정된 저장장치 내 값으로서,각 프로세스는 이를 확인과 변경할 수있음.
->프로세스간 메시지 전송하거나 공유 메모리를 통해서 특정 데이터를 공유하게 될 경우 발생하는 문제는 공유된 자원에 여러 프로세스가 접근하면서 발생.하나의 프로세스만 접근을 가능하도록 해야할 때 세마포어를 사용.

즉 둘의 차이점은
-세마포어는 뮤텍스가 될 수 있지만 뮤텍스는 세마포어가 될 수 없음.
-세마포어는 소유할 수 없는 반면 뮤텍스는 소유가 가능하며 소유주가 이에 대한 책임짐.
-뮤텍스의 경우 뮤텍스를 소유하고 있는 쓰레드가 이 뮤텍스를 해제 할 수 있음. 하지만 세마포어를 소유하지 않는 스레드가 세마포어를 해제할 수 있다.
-세마포어는 시스템 범위에 걸쳐있고 파일시스템상의 파일 형태로 존재. 반면 뮤텍스는 프로세스 범위를 가지며 프로세스가 종료 될 때 자동으로 clean up 된다.
-뮤텍스는 동기화 대상이 오직 하나일 때 , 세마포어는 동기화 대상이 하나 이상일 때.
48. module_init() 함수 호출은 언제 이루어지는가 ?

49. module_exit() 함수 호출은 언제 이루어지는가 ?

50. thread_union 에 대해 기술하시오.

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.

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.

56. Compiler의 Instruction Scheduling에 대해 기술하라.

57. CISC Architecture와 RISC Architecture에 대한 차이점을 기술하라.

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

59. Pipeline이 깨지는 경우에 대해 자세히 기술하시오.

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 ?

A: OS

61. Intel의 Hyper Threading 기술에 대해 상세히 기술하시오.

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.

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

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.

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

68. 현재 삽입된 디바이스 드라이버의 리스트를 보는 명령어는 무엇인가 ?

69. System Call Mechanism에 대해 기술하시오.

70. Process와 VM과의 관계에 대해 기술하시오.

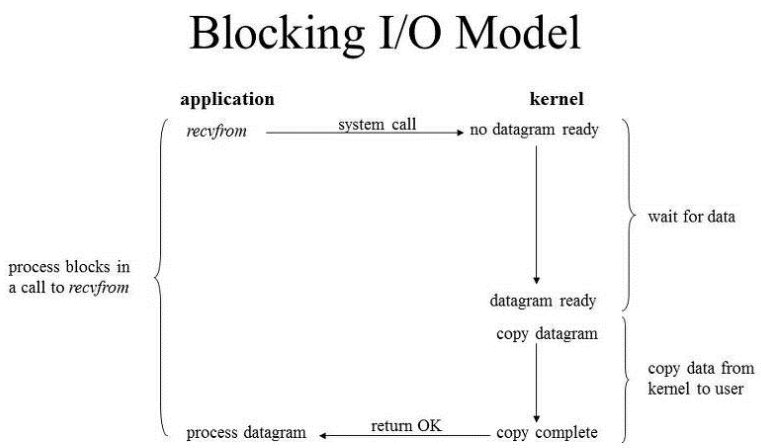
71. 인자로 파일을 입력 받아 해당 파일의 앞 부분 5줄을 출력하고
    추가적으로 뒷 부분의 5줄을 출력하는 프로그램을 작성하시오.

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

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

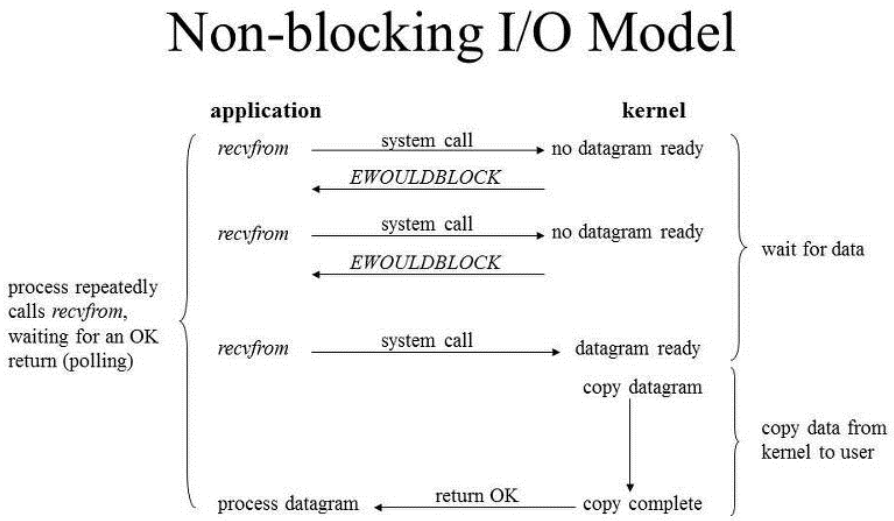
74. C.O.W Architecture에 대해 기술하시오.

75. Blocking 연산과 Non-Blocking 연산의 차이점에 대해 기술하시오.
"A: 블로킹 - 네트워크 통신에서 요청이 발생하고 완료 될때까지 모든 일을 중단한 상태로 #대기



논블로킹 - 통신이 완료될 때까지 기다리지 않고 다른 작업을 수행할 수 있도록.

%



76. 자식이 정상 종료되었는지 비정상 종료되었는지 파악하는 프로그램을 작성하시오.

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

78. SIGINT는 무시하고 SIGQUIT을 맞으면 죽는 프로그램을 작성하시오.

79. goto는 굉장히 유용한 C언어 문법이다.
    그러나 어떤 경우에는 goto를 쓰기가 힘든 경우가 존재한다.
    이 경우가 언제인지 기술하고 해당하는 경우
    문제를 어떤식으로 해결 해야 하는지 프로그래밍 해보시오.

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

'A: 컴퓨터 프로그래밍에서 파일디스크립터란 파일에 접근하기 위해 추상화 시켜놓은 장치를 표현함. 윈도우의 핸들과 유닉스계열의 파일디스크립터는 같은 의미.
리눅스는 모든장치를 파일로 관리하기 때문에 파일 디스크립터를 이용하여 장치에 접근 할 수 있게(핸들링) 됨.

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

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!

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

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

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

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

A; 고립 집합 이동성

88. 반 인원이 모두 참여할 수 있는 채팅 프로그램을 구현하시오.

89. 88 번 답에 도배를 방지 기능을 추가하시오.

2개월차 시험 유 형 준

90. 89 번 조차도 공격할 수 있는 프로그램을 작성하시오.

91. 네트워크 상에서 구조체를 전달할 수 있게 프로그래밍 하시오.

92. 91 번을 응용하여 Queue 와 Network 프로그래밍을 연동하시오.

93. Critical Section 이 무엇인지 기술하시오.

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

95. 리눅스 커널의 arch 디렉토리에 대해서 설명하시오.

96. 95 번 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 고찰해 보자!
2 개월간 자기 자신에 대한 반성과 성찰을 수행해보도록 한다.
또한 앞으로는 어떠한 자세로 임할 것인지 작성하시오.

A: 모르는게 당연히 많을거라 생각해서 이 과정의 첫날부터 혼자서 10시까지 남았던 적이 있었습니다. 모르니까 어떻게든 따라가기 위해서 그 다음날도 그 다음날도 남기 시작했고 그러면서 같이 남는 친구들이 생기더니 나중에는 많아져서 든든하기도 합니다.
개인적인 사정이 4월달에 몰아쳐서 사실상 집중이 많이 힘들었습니다.
그 개인적인 사정이 노는 것도 있지만 아닌 것도 있는 굉장히 머리아픈 일이였거니와
통신과 커널을 본격적으로 들어가니 과정도 어려워지고 두개의 조합이 저를 ‘멍멍’한 상태로 만들었습니다.

2개월차 시험 유 형 준

그렇게 하루이틀 모르는것을 알고 남어가지 못하게 되니 쌓이고 쌓여서 나중에는 엄두도 안나는 상황이 되었습니다. 죽을둥 살둥 주말에도 공부했다면 지금보다 나았을 것 입니다.
하지만 그때 상황이 너무 머리아프니까 스트레스를 풀 생각만으로 주말을 지내다 보고 등등
초반의 기합이 다 날라가는 상황이 왔습니다.
그나마 좋은소식은 딱 오늘 4월17일에 개인적인 사정이 다 해결 되었습니다.
그래서 시험도 늦은거구요.
공부하는 방법을 바꿔야한다는 생각이 들었습니다.
쉬운과정이야 이해가 되니까 남아서 쉽게쉽게 정리만 하면 되지만
이렇게 어려운과정은 남아서 어떻게 복습할것인지 복습할 방법?도 달라야 할 것 같습니다.
모르는걸 따로 적는데 어떤개념을 모르는지, 이 부분은 언제 끝내야 할지 등
고통이 따르겠지만 조금씩 해보려고 합니다.
시험이후로 다시 집중할 수 있도록 하겠습니다. (근데 제 자리 너무 졸림. 뭔가 이상함)
감사합니다.