

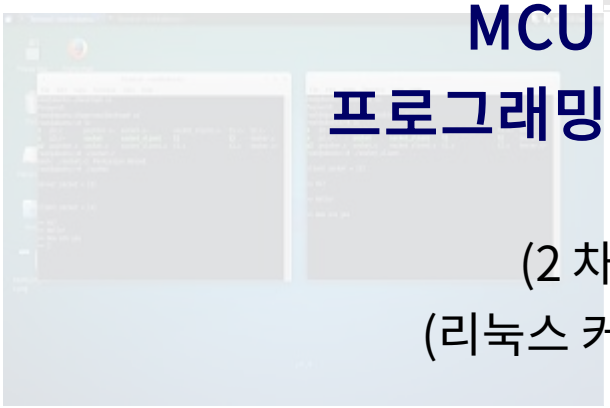


Xilinx Zynq FPGA,TI DSP, MCU 기반의 프로그래밍 전문가 과정

(2 차 시험)

(리눅스 커널 시스템)

날 짜 : 2018 . 4. 10

강사 – Innova Lee(이상훈)

gcccompil3r@gmail.com

학생 – 정한별
hanbulkr@gmail.com

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

```
int main(int argc , char **argv)
{
	int fd, ret;
	ssize_t nread;
	char buf[1024];
	fd = open(argv[1], O_WRONLY | O_CREAT ,0644);
	close(fd);

	return 0;
}
```

2.369 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.
만들다 실패..

서버.

```
// 박수 = Ctrl + c
// SIGALRM 은 2 초 이다.


#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>
#include <setjmp.h>


#define BUF_SIZE	256
#define MAX_CLNT	512


typedef struct sockaddr_in		si;
```

```
typedef struct sockaddr *                       sap;
int clnt_cnt = 0;
int clnt_socks[MAX_CLNT];
int data[MAX_CLNT];
int thread_pid[MAX_CLNT];
int idx;
int cnt[MAX_CLNT];
int preNum=0;

jmp_buf gogo;

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);

        // 현재 쓰레드 PID 를 가져온다.
        for(i = 0; i < clnt_cnt; i++)
                if(thread_pid[i] == getpid())
                        pthread_kill(thread_pid[i], SIGINT);
                        //cnt[i] += 1;      // 이게 현재 숫자라고 가정한다.
        pthread_mutex_unlock(&mtx);
        // 나중에 알람을 끄고 소켓이 꺼지게 해야한다.
        close(clnt_socks[idx-1]);
}

void sig_handler2(int signo)
{
        int i;
        printf(" 박수! \n");

        pthread_mutex_lock(&mtx);

        // 현재 쓰레드 PID 를 가져온다.
        for(i = 0; i < clnt_cnt; i++)
                if(thread_pid[i] == getpid())
                        cnt[i] += 1;
        longjmp(gogo, 1);
        pthread_mutex_unlock(&mtx);
```

```
        alarm(2);
}



void proc_msg(char *msg, int len, int k)
{
        int i;
        int cmp = atoi(msg);
        char smsg[64] = {0};

        pthread_mutex_lock(&mtx);
        // k 는 현재 플레이어 넘버 +1, idx 가 player 라 보면됨.
        cnt[k] += 1;
        preNum++;       // 현재 진행중인 값.

        // 누구턴인지 -> (턴인사람 , 아닌사람->끝)
        // 데이터가 지행중인 값이랑 같은지


        if(data[k] > cmp)
                sprintf(smsg, "big than %d\n", cmp);
        else if(data[k] < cmp)
                sprintf(smsg, "less than %d\n", cmp);
        else if(setjmp(gogo))
                sprintf(smsg, "박수 \n");
        else
        {
                strcpy(smsg, "You win!\n");
                printf("cnt = %d\n", cnt[k]);
        }
        // 스트링에 가져다가 붙이는것.
        // 사실 좋은 구조는 클라이언트한테 숫자 쏘고 클라이언트 종료되게 해야함.
        sprintf(smsg, "[%d] player Input Number: \n",k+1);
        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: \n";

        //
        signal(SIGALRM, sig_handler);
```

```
	signal(SIGINT, sig_handler2);

	// 쓰레드의 pid 값이 나온다 getpid 에서. 쓰레드를 통해 들어온거니까
	pthread_mutex_lock(&mtx);
	thread_pid[idx++] = getpid();
	i = idx - 1;
	printf("idx = %d\n",idx);
	printf("i = %d\n", i);

	// 락을 걸어준 것이다.
	write(clnt_socks[i], pattern, strlen(pattern));
	pthread_mutex_unlock(&mtx);

	alarm(2);
	//break point idx 2 명이 들어와야 시작한다고 표시한것.
	while((str_len = read(clnt_sock, msg, sizeof(msg))) != 0 && idx >= 1)
	{
		// 잘 들어왔으니 알람을 초기화 한다.
		alarm(0);

		proc_msg(msg, str_len, i);

		alarm(2);
	}


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

	return NULL;
}



int main(int argc , char **argv)
{
	int serv_sock, clnt_sock, fd;
	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, (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);
		// 클라이언트의 접근을 승인한다. 다음 클라이언트가 들어오기 전까지 블록한다.
		clnt_sock = accept(serv_sock, (sap)&clnt_addr, &addr_size);

		thread_pid[idx++] = getpid();
		// idx 는 사람의 수 clnt_cnt 도 사람의 수.


		pthread_mutex_lock(&mtx);

		clnt_socks[clnt_cnt++] = clnt_sock;

		pthread_mutex_unlock(&mtx);

		// 쓰레드에  전달되는 인자 (마지막꺼_fd 값)
		pthread_create(&t_id, NULL, clnt_handler, (void *)&clnt_sock);
		// 떼어내다... 쓰레드 t_id 는 식별자 혹은 id (cpu 에 새로 할당됨)
		pthread_detach(t_id);
		printf("Connected Client IP: %s\n", inet_ntoa(clnt_addr.sin_addr));
	}
	//close(clnt_sock);
	close(serv_sock);

	return 0;
}
////////////////////////////////////////////////////////////////////////////////
client

#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

typedef struct sockaddr_in      si;
typedef struct sockaddr *       sp;

char msg[BUF_SIZE];

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

// pthead_creat 의 마지막인자는 여기로 들어온다. 여러개 보낼려면 구조체 형으로 보내야한다.
// 3 번째 인자에는 void * 형으로 들어와야한다... 어떤게 올지 모르니까.
// 검색 속도를 올릴려면 , 예를 들어 이진트리를 병렬 검색을 한다
void *send_msg(void *arg)
{
        int sock = *((int *)arg);
        char msg[BUF_SIZE];

        for(;;)
        {
                // 무한루프를 돌면서 계속 읽고 전송해 준다.
                fgets(msg, BUF_SIZE, stdin);
                write(sock, msg, strlen(msg));
        }

        return NULL;
}

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

        for(;;)
        {
                // 서버로 부터 받아서
                str_len = read(sock, msg, BUF_SIZE - 1);
                // 0 으로 해주는 이유는 먼저 긴게 들어오고 나중에 짧은게 들어올때 덮어씌면서 뒤가 남음.
                // memset 을 해주는게 좋지만 널문자를 끝맺음을 함으로 동작에 이득을 취한다.
                msg[str_len] = 0;
                // 모니터의 출력을 한다.
                fputs(msg, stdout);
```

```
        }

        return NULL;
}

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

        // 커넥트 하는 순간 어셉트가 동작한다.
        // 위에서 설정한 (포트와 주소 정보(serv_adr)) 를 가지고 connect 를 시도한다.
        if(connect(sock, (sp)&serv_addr, sizeof(serv_addr)) == -1)
                err_handler("connect() error");

        // 송신과 수신을 분리하기 위해서 한다. fork 랑 비슷하다.
        // 전송.
        pthread_create(&snd_thread, NULL, send_msg, (void *)&sock);
        // 수신.
        pthread_create(&rcv_thread, NULL, recv_msg, (void *)&sock);
        // 본격적인 구동은 join 을 할때. 더이상 센드, 혹은 보낼게 없거나 하면 멈춤.
        pthread_join(snd_thread, &thread_ret);
        pthread_join(rcv_thread, &thread_ret);

        close(sock);

        return 0;
}
```

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

1. 태스크 관리
2. 메모리 관리
3. 파일시스템

4. 디바이스 드라이버
5. 네트워크

4. Unix 계열의 모든 OS 는 모든 것을 무엇으로 관리하는가 ?
추상적인 자원을 통해 관리 한다.
(커널을 통해 태스크 관리자 , 메모리 관리자 , 파일시스템, 디바이스 드라이버 관리자, 네트워크 관리자를 이용해 관리 한다.)

5.리눅스에는 여러 장점이 있다.(배점 0.2 점) 아래의 장점들 각각에 대해 기술하라.
*** 사용자 임의대로 재구성이 가능하다.**

-open 소스이며 하이브리드 방식으로 디바이스 드라이브등 필요한것이 있으면 레고블럭처럼 떼었다가 붙였다가 하며 쓸수 있기 때문에 사용자가 직접 필요에 맞게 개량할 수 있다.

*** 열약한 환경에서도 HW 자원을 적절히 활용하여 동작한다.**

모노리틱, 하이브리드 방식을 통해 하드웨어 자원을 적절히 적재하며 사용하기 편리하다
모듈을 사용하면 커널 소스 없이 직접 컴파일이 가능하다.

*** 커널의 크기가 작다.**

마이크로 방식을 통해 커널 소스가 작고 관리 , 개선 , 유지 에 편하다. 핸드폰 노트북등 이동성 기기에 적합하다.

*** 완벽한 멀티유저, 멀티태스킹 시스템**

thread 를 이용해 멀티 태스킹이 가능하다.

*** 뛰어난 안정성**

어셈블리어등의 기계어를 사용하면 더욱 안전한 프로그램밍을 가능케 할 수 있다.

*** 빠른 업그레이드**

오픈 소스방식이기 때문에 많은 사용자들이 더 좋은 소스를 공유해서 필요한 부분을 같이 업그레이드 해 나가기 때문에 업그레이드 속도가 빠른다.

*** 강력한 네트워크 지원**

osi 7 계층의 이론을 통해 강력한 네트워크 연결 방식을 지원한다.

*** 풍부한 소프트웨어**

모든것을 적재하는 방식이기 때문에 필요에 따라 다른 소프트웨어를 사용하기에 용이하기 때문이다.

6. 32 bit System 에서 User 와 Kernel 의 Space 구간을 적으시오.
0xc0000000 밑으로는 유저 영역 위로는 커널 영역이다.

7.Page Fault 가 발생했을때 운영체제가 어떻게 동작하는지 기술하시오.

Page fault handler 가 동작해서 유저영역인지 커널 영역이지 확인후 커널이면(demand on paging) 페이즈 프레임을 할당해 준다.

8.리눅스 실행 파일 포맷이 무엇인지 적으시오.

ELF (Executable and Linkable Format)

9.프로세스와 스레드를 구별하는 방법에 대해 기술하시오.

프로세스는 동작중이 프로그램, 스레드는 프로세스 내에서 실행 된 tesk_struct 의 단위라고 보면된다.

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

task_struct 를 생성한다.

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

current 는 thead_stack 를 나타낸다.

```
/**
 * struct thread_stack - thread stack constructed from 'call' and 'return'
 *                       branch samples.
 * @stack: array that holds the stack
 * @cnt: number of entries in the stack
 * @sz: current maximum stack size
 * @trace_nr: current trace number
 * @branch_count: running branch count
 * @kernel_start: kernel start address
 * @last_time: last timestamp
 * @crp: call/return processor
 * @comm: current comm
 */
struct thread_stack {
    struct thread_stack_entry *stack;
    size_t cnt;
    size_t sz;
    u64 trace_nr;
    u64 branch_count;
    u64 kernel_start;
    u64 last_time;
    struct call_return_processor *crp;
    struct comm *comm;
};
```

12. Memory Management 입장에서 Process 와 Thread 의 핵심적인 차이점은 무엇인가 ?

process 는 다른 주소공간에 저장 된다. Thread 는 같은 주소 공간을 갖는다.

13. Task 가 관리해야하는 3 가지 Context 가 있다. System Context, Memory Context, HW Context 가 있다. 이중 HW Context 는 무엇을 하기 위한 구조인가 ?

실행중이던 태스크가 대기 상태나 준비 상태로 전이 할때 태스크가 어디까지 실행했는지 기억해 두는 공간이다.

14. 리눅스 커널의 스케쥴링 정책중 Deadline 방식에 대해 기술하시오.

기존 리눅스의 실시간 태스크 스케줄링 정책은 **우선순위에 기반**하여 스케줄링 대상을 선정하는데 **반해 , deadline 정책은 deadline 이 가장 가까운 (즉 가장 급한) 태스크를 스케줄링 대상으로 선정**한다.

15. TASK_INTURRUPTIBLE 과 TASK_UNINTERRUPTIBLE 은 왜 필요한지 기술하시오.
효과적이지만 잠재적으로 시그널을 받지 않는 **TASK_UNINTERRUPTIBLE** 과
쉽게 잠에서 깨어나지만 좀 더 안전한 **TASK_INTERRUPTIBLE**

등의 **프로세스 상태를 조절 할 때 쓴다.**

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

TASK_UNINTERRUPTIBLE: 프로세스 상태는 **TASK_INTERRUPTIBLE** 과 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) Algorithm 에 대해 기술하시오. 그리고 무엇이 어떤 경우에 더 좋은지 기술하시오.
(FIFO 혹은 RR 스케줄링 정책을 사용 시) 시스템에 존재하는 모든 태스크는 tasklist 라는 이중 연결 구조를 갖는다. 이 리스트를 사용하면 테스크 내에 모든 연결구조에 접근이 가능하다. 그리고 그중 가장 높은 우선 순위에 있는 것을 찾아내면 된다.
O(n)의 경우 y=ax 의 선형적 증가 형태로 시간복잡도가 증가한다. 처리할 개수가 작을 때 사용하기 좋다.
O(n)의 경우 y= a 의 형태로 일정한 개수의 상관없이 일정한 시간으로 동작하기 때문에 처리할 개수가 많을 때 사용하는 것이 용이하다.

17.현재 4 개의 CPU(0, 1, 2, 3)가 있고 각각의 RQ 에는 1, 2 개의 프로세스가 위치한다. 이 경우 2 번 CPU 에 있는 부모가 fork()를 수행하여 Task 를 만들어냈다. 이 Task 는 어디에 위치하는 것이 좋을까 ? 그리고 그 이유를 적으시오.
cpu 는 오직 한순간에 한 작업을 시행 할 수 있다. cpu 가 한클럭에 한번의 4 가지 일을 할 수 있다. 그렇기 때문에 rq 가 돌고 있지 않는 남는 자리가 많은 cpu 에서 수행 되도록 해야 한다.

18. 15 번 문제에서 이번에는 0, 1, 3 에 매우 많은 프로세스가 존재한다. 이 경우 3 번에서 fork()를 수행하여 Task 를 만들었다.이 Task 는 어디에 위치하는 것이 좋을까 ? 역시 이유를 적으시오.
일반 태스크의 스케쥴링을 생각했을때 … 2 번쪽에 task 가 생겨야 한다. 공정하게 일을 나누어 해야한다.

19.UMA 와 NUMA 에 대해 기술하고 Kernel 에서 이들을 어떠한 방식으로 관리하는지 기술하시오.
커널 내부의 소스 코드와 함께 기술하도록 하시오.

Uma - 메모리와 **cpu** 가 단일 버스에 물려 있다. 병목현상이 일어날 수도 있다.
- **UMA** 의 경우 한개의 **node** 가 존재할 것이다. 이 **node** 는 **contig_pagedata** 를 통해 접근 가능하다.

Numa – cpu 부하 뿐만아니라 메모리 접근 시간츼 차이 등도 고려 하여 부한 균등을 시도한다.
- NUMA 의 경우 복수개의 node 가 존재할 것이다. 복수개 node 는 list 를 통해 관리되며 pglist_data 라는배열을 통해 접근이 가능하다.

```
struct zone {
```

```
#ifdef CONFIG_NUMA
    int node;
```

```
struct node_data {
    struct pglist_data pglist;
    struct hub_data hub;
};
```

20.

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

Static priority = 0~99 번.

Dynamic priority = 100 ~ 139

22.ZONE_HIGHMEM 에 대해 아는대로 기술하시오.

- ZONE_HIGHMEM (896 ~ end) → 동적으로 연결하여 사용할 수 있는 간접 참조 구조

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

- page frame 이라 한다 **. SW** 는 **page** 구조체를 쓴다**.**

24.Linux Kernel 은 외부 단편화와 메모리 부하를 최소화하기 위해 Buddy 할당자를 사용한다.
Buddy 할당자의 Algorithm 을 자세히 기술하시오.

- 페이지 프레임 단위 (4kB) 로 할당 하도록 결정 한 방식이 Buddy 이다 . (4KB x 2^n 방식으로 동작)

25.21 번에 이어 내부 단편화를 최소화 하기 위해 Buddy 에서 Page 를 받아 Slab 할당자를 사용한다.
Slab 할당자는 어떤식으로 관리되는지 기술하시오.

1. 페이지 프레임 을 받아서 이 공간을 나눈다 .
2. 그리고 버디할당자로 부터 받지 말고 나뉜 공간으로 받는다 .
3. 해제시에는 버디로 반납하는게 아닌 미리 할당 받은 공간에 그대로 가지고 있는다 .
4. 일종의 캐시로 사용하는 것이다 .

26.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 에서 관리한다.

27.23 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

28.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 ?

Elf 포멧을 통해 읽게 되고 그와 연결되어 있는 가상 메모리 주소를 통해서 일게 된다.

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

태스크를 하나 생성하고 태스크에게 가상 주소 공간을 제공해 주고 , 메모리의 일부를 할당해 준다 (page), 태스크가 원하는 디스크 상의 내용을 읽어 물리 메모리에 올려놓고 (sys_execve) 물리 메모리의 실제 주소와 태스크의 가상 주소 공간을 연결 하여 적재 하게 된다 .

30. MMU(Memory Management Unit)의 주요 핵심 기능을 모두 적고 간략히 설명하시오.
페이징이 된 녀석을 미리 **cache** 해서 **page fault** 발생시에 **pagehandler** 에 서 일어나는 가상주소에서 물리 주소로 변환을 시켜주는 하드웨어 이다**.**

31. 하드디스크의 최소 단위를 무엇이라 부르고 그 크기는 얼마인가 ?
디스크 블록 **, 4kB** 이다**.**

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

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

SYSCALL 함수가 동작한다**.**
open() 시스템 콜을 한다. Vfs 는 어떤 파일시스템인지 확인을 한다.

34. task_struct 에서 super_block 이 하는 역할은 무엇인가 ?

- 파일시스템의 전체적인 정보와 '/'의 위치를 기억하는 것이다**.**

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

1) b.txt 파일을 인자로 **open()** 시스템 콜을 한다**. Vfs** 는 어떤 파일시스템인지 확인을 한다**.**
2) b.txt 를 구조체로 만들어 **ext2** 내부에 **open** 함수를 호출해 보낸다**.**
3) ext2 는 자신의 **inode** 테이블에서 **b.txt** 의 **inode** 를 찾아**,** 그 정보를 **vfs** 에 넘긴다**.**
4) 넘겨진 **b.txt** 의 **inode** 정보를 **vfs** 내에 만들어 둔 구조체에 담아 리턴을 한다**.**
5) inode 내에 담겨진 정보를 **vfs** 가 사용자 태스크에 넘겨준다**.**

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

외부 인터럽트
내부 인터럽트

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

1.fault
2.trap
3.abort

38. 35 번에서 분류한 녀석들의 특징에 대해 기술하시오.
1. fault
fualt 를 일으킨 명령어 주소를 **eip** 에 넣어 두었다가 해당 핸들러가 종료되고 나면 **eip** 에 저장되어 있는 주소부터 다시 수행한다**.**
2. trap
trap 을 일이큰 명령어의 다음주소 를 **eip** 에 넣어두었다가 해당 핸들러 종료 후 **eip** 에 저장되어 있는 주소 부터 수행**.**
3. abort
심각한 애러의 경우**, eip** 에 저장할 필요가 없으므로 현재 태스크를 강제 종료 한다**.**

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

PIC 를 통한 외부인터럽트를 사용하여 이들을 공유시킨다.

40.System Call 호출시 Kernel 에서 실제 System Call 을 처리하기 위해
Indexing 을 수행하여 적절한 함수가 호출되도록 주소값을 저장해놓고 있다.
이 구조체의 이름을 적으시오.
IDT 구조체 이다.

41.38 에서 User Space 에서 System Call 번호를 전달한다.
Intel Machine 에서는 이를 어디에 저장하는가 ?
또한 ARM Machine 에서는 이를 어디에 저장하는가 ?

systme_call : arch/x86/kernel/entry_32.S
sys_call_table : ~/arch/x86/entry/syscalls/syscall_64.tbl

42.Paging Mechanism 에서 핵심이 되는 Page Directory 는 mm_struct 의 어떤 변수가 가지고 있는가 ?
pgd 가 가지고 있다.
Pgd → pmd → pte 순이다

43.또한 Page Directory 를 가르키는 Intel 전용 Register 가 존재한다.
이 Register 의 이름을 적고 ARM 에서 이 역할을 하는 레지스터의 이름을 적으시오.
CR 이다. 암에서는 TTBR 이다.

44.커널 내부에서 메모리 할당이 필요한 이유는 무엇인가 ?
커널 내부에서 가상적으로 메모리를 만들어 더 많은 공간을 할당해서 하려고

45.메모리를 불연속적으로 할당하는 기법은 무엇인가 ?
세그먼테이션 기법

46.메모리를 연속적으로 할당하는 기법은 무엇인가 ?
버디 할당자

47.Mutex 와 Semaphore 의 차이점을 기술하시오.
mutext 는 **Critical section** 에 의해 공유 메모리가 침범할 경우가 있기에 접근을 막아주고 열어주고를 잘해야 하지만 세마포어는 그렇지 않다.

48.module_init() 함수 호출은 언제 이루어지는가 ?
터미널 창에서 **insmod** 를 입력 할 시에 호출이 된다.

49.module_exit() 함수 호출은 언제 이루어지는가 ?
터미널 창에서 **rmmod** 를 입력 할 시에 호출이 된다.

50.thread_union 에 대해 기술하시오.
thread_union 은 커널 스택 이라 불리고 **thread_info** 구조체를 포함한다.
thread_info 구조체를 **(**프로세스 디스크립터**(fd) → task** 마다 존재**)** 라 부른다.

51.Device Driver 는 Major Number 와 Minor Number 를 통해 Device 를 관리한다.
실제 Device 의 Major Number 와 Minor Number 를 저장하는 변수는 어떤 구조체의 어떤 변수인가 ?
(역시 Task 구조체에서부터 쭉 찾아오길 바람)
inode 구조체에 i_rdev 라는 변수이다.
task_struct → file_struct→ file→ dentry→ inode→ i_rdev

52.예로 간단한 Character Device Driver 를 작성했다고 가정해본다.

그리고 insmod 를 통해 Driver 를 Kernel 내에 삽입했으며 mknod 를 이용하여 /dev/장치파일을 생성하였다.그리고 이에 적절한 User 프로그램을 동작시켰다.
이 Driver 가 실제 Kernel 에서 어떻게 관리되고 사용되는지 내부 Mechanism 을 기술하시오.

결정한 이름의 드라이버의 주번호를 동적으로 할당 받아서 실행한다. file_operations 를 통해 자료 구현되어 있어 이 함수들의 시작 주소들이 등록된다.

53.Kernel 자체에 kmalloc(), vmalloc(), __get_free_pages()를 통해 메모리를 할당할 수 있다.
또한 kfree(), vfree(), free_pages()를 통해 할당한 메모리를 해제할 수 있다.
이러한 Mechanism 이 필요한 이유가 무엇인지 자세히 기술하라.

54.Character Device Driver 를 아래와 같이 동작하게 만드시오.
read(fd, buf, 10)을 동작시킬 경우 1 ~ 10 까지의 덧셈을 반환하도록 한다.
write(fd, buf, 5)를 동작시킬 경우 1 ~ 5 곱셈을 반환하도록 한다.
close(fd)를 수행하면 Kernel 내에서 "Finalize Device Driver"가 출력되게 하라!

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/slab.h>
#include <linux/fs.h>
#include <linux/cdev.h>
#include <linux/device.h>
#include <linux/uaccess.h>

#define DEVICE_NAME             "mydrv"
#define MYDRV_MAX_LENGTH        4096
#define MIN(a,b) (((a) < (b)) ?(a) :(b))

struct class *myclass;
struct cdev *mycdev;
struct device *mydevice;
dev_t mydev;

static char *mydrv_data;
static int mydrv_read_offset, mydrv_write_offset;

static int mydrv_open(struct inode *inode, struct file *file)
{
        printk("%s\n", __FUNCTION__);
        return 0;
}

static int mydrv_release(struct inode *inode, struct file *file)
{
        printk("%s\n",__FUNCTION__);
        return 0;
}
```

```
static ssize_t mydrv_read(struct file *file, char *buf, size_t count, loff_t *ppos)
{
	if( (buf == NULL) || (count <0))
		return -EINVAL;
	if( (mydrv_write_offset - mydrv_read_offset) <=0)
		return 0;
	count = MIN(( mydrv_write_offset -mydrv_read_offset), count);
	if(copy_to_user(buf, mydrv_data + mydrv_read_offset, count))
		return -EFAULT;
	else
		printk("Finalize Device Driver\n");
	mydrv_read_offset +=count;
	return count;

}

static ssize_t mydrv_write(struct file *file, const char *buf, size_t count, loff_t *ppos){
	if((buf == NULL) || (count<0))
		return -EINVAL;
	if(count +mydrv_write_offset >= MYDRV_MAX_LENGTH) {
		/* driver space is too small*/
		return 0;
	}
	if( copy_from_user(mydrv_data +mydrv_write_offset, buf, count))
		return -EFAULT;
	mydrv_write_offset +=count;
	return count;
}

struct file_operations mydrv_fops ={
	.owner = THIS_MODULE,
	.read = mydrv_read,
	.write= mydrv_write,
	.open = mydrv_open,
	.release = mydrv_release,
};


int mydrv_init(void)
{
	if(alloc_chrdev_region(&mydev, 0, 1, DEVICE_NAME) <0){
		return -EBUSY;
	}


	myclass = class_create(THIS_MODULE , "mycharclass");

	if (IS_ERR(myclass)){
		unregister_chrdev_region(mydev, 1);
		return PTR_ERR(myclass);
```

```
	}

	mydevice = device_create(myclass, NULL, mydev, NULL, "mydevicefile");
	if(IS_ERR(mydevice)){
		class_destroy(myclass);
		unregister_chrdev_region(mydev,1);
		return PTR_ERR(mydevice);
	}

	mycdev = cdev_alloc();
	mycdev -> ops = &mydrv_fops;
	mycdev -> owner = THIS_MODULE;
	if( cdev_add(mycdev, mydev, 1) <0){
		device_destroy(myclass, mydev);
		class_destroy(myclass);
		unregister_chrdev_region(mydev,1);
		return -EBUSY;
	}

	mydrv_data = (char *) kmalloc(MYDRV_MAX_LENGTH * sizeof(char),
GFP_KERNEL);
	mydrv_read_offset = mydrv_write_offset = 0;
	return 0;

}


void mydrv_cleanup(void)
{
	kfree(mydrv_data);
	cdev_del(mycdev);
	device_destroy(myclass, mydev);
	class_destroy(myclass);
	unregister_chrdev_region(mydev,1);
}

module_init(mydrv_init);
module_exit(mydrv_cleanup);
MODULE_LICENSE("GPL");
```

//

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

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

#define MAX_BUFFER      10
char buf_in[MAX_BUFFER];
char buf_out[MAX_BUFFER];

int main(void)
{
        int fd, i, c,res1 = 0;
        int res2 =1;
        char s[128];
        if( (fd = open("/dev/mydevicefile", O_RDWR)) <0){
                perror("open error");
                return -1;
        }

        for(i= 1 ; i<11; i++){
                res1 += i;
        }
        for(i= 1 ; i<6; i++){
                res2 *= i;
        }

        for(i =0; i<MAX_BUFFER; i++){
                fprintf(stderr, "%c\n",buf_in[i]);
        }

        fprintf(stderr, "\n");
        sprintf(s,"%d", res1);
        read(fd, s, MAX_BUFFER);
        write(fd, s, MAX_BUFFER);
        s[128] = 0;
        sprintf(s,"%d", res2);
        read(fd, s, MAX_BUFFER);
        write(fd, s, MAX_BUFFER);

        fprintf(stderr, "\n");

        close(fd);
        return 0;
}
```

55.OoO(Out-of-Order)인 비순차 실행에 대해 기술하라.

비 순차적으로 실행하게 하는 기법이다.

예를 들며 **deadline** 기법 이다. 자신의 **dealine** 에 맞추어 우선순위 상관없이 **process** 가 돌아가게 된다.

56.Compiler 의 Instruction Scheduling 에 대해 기술하라.

컴파일러가 compile-time 에 의해 스케줄링 하는 것.

Static Instruction Scheduling 이라 할 수 있다

57.CISC Architecture 와 RISC Architecture 에 대한 차이점을 기술하라.

RISC 의 경우에는 소형 로직으로 충분한데, 이는 각각의 명령어가 단순한 프로세스에 상응하기 때문이다. 그러나 CISC 의 경우에는 각각의 명령어에 대해 실행되는 프로세스들이 복잡하므로 대형 로직이 필요하다. 이러한 이유 때문에 MICRO-ROM 방식을 채택한다

58.Compiler 의 Instruction Scheduling 은 Run-Time 이 아닌 Compile-Time 에 결정된다.
고로 이를 Static Instruction Scheduling 이라 할 수 있다.
Intel 계열의 Machine 에서는 Compiler 의 힘을 빌리지 않고도
어느저도의 Instruction Scheduling 을 HW 의 힘만으로 수행할 수 있다.
이러한 것을 무엇이라 부르는가 ?

디바이스 드라이버

59.Pipeline 이 깨지는 경우에 대해 자세히 기술하시오.

분기문명령어 등이 오면 파이프 라인이 깨진다. 실행까지 몇클럭을 더 낭비 하기 때문에 파이프 라인이 깨지게 된다. 기본적으로 **jmp , call, cpu instruction** 명령어 레벨 등 에서 깨진다.

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

cpu 타입에 따라서.

61.Intel 의 Hyper Threading 기술에 대해 상세히 기술하시오.

하이퍼 쓰레딩이란 **cpu** 가 쿼드라고 할때 하이퍼 쓰레딩을 쓰면 마치 **8** 개 처럼 동작이 가능한데 이것은 **fork** 방식을 하드웨어적으로 구현해 놓았기 때문에 가능한 것이다.

62.그동안 많은 것을 배웠을 것이다.
최종적으로 Kernel Map 을 그려보도록 한다.
(Networking 부분은 생략해도 좋다)
예로는 다음을 생각해보도록 한다.
여러분이 좋아하는 게임을 더블 클릭하여 실행한다고 할 때
그 과정 자체를 Linux Kernel 에 입각하여 기술하도록 하시오.
(그림과 설명을 같이 넣어서 해석하도록 한다)
소스 코드도 함께 추가하여 설명해야 한다.

우리가 더블 클릭을 외부 인터럽트가 발생하고 하드웨어 **pic** 에 의해 처리가 된다.
리눅스 커널에서는 이런 인터럽트 신호를 대기 하고 있다가 **(sigaction)** 실행을 하게 된다.

63. 파일의 종류를 확인하는 프로그램을 작성하시도록 하시오.

```
#include<stdio.h>
#include<unistd.h>
#include<fcntl.h>
#include<dirent.h>
#include<pwd.h>
#include<grp.h>
#include<time.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<string.h>
void read_l(char *name);

// R 일 때 이다.
void recursive_dir(char *dname, char **argv)
{
	struct dirent *p;
	struct stat buf;
	DIR *dp;
	chdir(dname);
	dp = opendir(".");
	printf("\t%s:\n",dname);
	while(p = readdir(dp)){

		if(argv[0] == 'l' || argv[1] == 'l' ||argv[2] == 'l')
			read_l(p->d_name);
		else
			printf("%s\n",p->d_name);


	}
	rewinddir(dp);
	while(p = readdir(dp))
	{
		stat(p->d_name, &buf);
		if(S_ISDIR(buf.st_mode))
			if(strcmp(p->d_name,".") && strcmp(p->d_name,".."))

				recursive_dir(p->d_name, argv);
	}

	chdir("..");
	closedir(dp);
}

void read_l(char *name)
{
	int i;
	char perm[11]="----------";
	char rwx[4] ="rwx";
	char sst[4] ="sst";
```

```
	char *arr="alRi";
	struct stat buf;
	struct passwd *pw;
	struct group *gr;
	struct tm *tm;

	stat(name,&buf);
	if(S_ISDIR(buf.st_mode))
		perm[0] = 'd';
	if(S_ISREG(buf.st_mode))
		perm[0] = '-';
	if(S_ISFIFO(buf.st_mode))
		perm[0] = 'p';
	if(S_ISSOCK(buf.st_mode))
		perm[0] = 's';
	if(S_ISCHR(buf.st_mode))
		perm[0] = 'c';
	if(S_ISLNK(buf.st_mode))
		perm[0]=  'i';
	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("%2lu ",buf.st_nlink);
	pw = getpwuid(buf.st_uid);
	printf("%3s ",pw->pw_name);
	gr = getgrgid(buf.st_gid);
	printf("%3s ",gr->gr_name);
	printf("%5lu ", buf.st_size);
	tm = localtime(&buf.st_mtime);
	printf("%5d -> %02d-%02d %02d:%02d ",
			tm->tm_year + 1900, tm->tm_mon +1,tm->tm_mday, tm->tm_hour, tm->tm_min);
	printf("%3s",name);
	printf("\n");
}

void ls_program(int flag,int argc, char **argv)
{
	int cmd, i=0, x=1;
	char ch ;
	char arr[] = "alRi\0";
	struct dirent *p;
	struct stat buf;
```

```
	DIR *dp;
	dp = opendir(".");
	stat(argv[1],&buf);

	while((cmd = getopt(argc, argv, arr))> 0)
	{
		while(p=readdir(dp))
		{
			switch(cmd)
			{
				case'a':
					flag |= 1<<0;
					printf("%-14s", p->d_name);
					if((i+1)%5==0)
						printf("\n");
					i++;

					break;
				case'l':
					flag |= 1<<1;
					read_l((p->d_name));
					break;
				case'R':
					flag |= 1<<2;
					recursive_dir(".",argv);
					goto a;
					break;
				default:
					printf("unknown iption\n");
			}
			if(cmd != 'a')
			{
				if(p->d_name[0] == '.')
					continue;
				if(cmd != 'l' && (i+1)%5==0 && cmd != 'R' )
					printf("\n");
				i++;
			}

		}
		a:
		printf("\n");
		closedir(dp);
	}
}


int check_flag(int flag)
{
	int i, tmp = flag;
```

```
        for(i = 0; i < 7; i++)
        {
                switch(tmp & (1 << i))
                {
                        case 1:
                                printf("a\n");
                                break;
                        case 2:
                                printf("l\n");
                                break;
                        case 4:
                                printf("R\n");
                                break;
                        case 8:
                                printf("i\n");
                                break;
                        case 16:
                                printf("E\n");
                                break;
                        case 32:
                                printf("F\n");
                                break;
                        case 64:
                                printf("G\n");
                                break;
                }
        }
}
int main(int argc, char **argv)
{

        struct stat buf;
        struct dirent *p;

        DIR *dp;
        int cmd, i=0, x=1;
        int flag=0;
        char ch ;
        char arr[] = "alRi";

        dp = opendir(".");
        stat(argv[1],&buf);
        ls_program(flag, argc, argv);
        check_flag(flag);


        return 0;
}
```

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

서버.

```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;

#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;

	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(0 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");
	// 받을 사람 수.
	if(listen(serv_sock, 5) == -1)
		err_handler("lister() error");

	clnt_addr_size = sizeof(clnt_addr);

	for(i = 0 ; i<5; i ++)
	{	// 실질적인 클라이언트  승인
		clnt_sock = accept(serv_sock , (struct sockaddr *)&clnt_addr, &clnt_addr_size);
		if(clnt_sock == -1)
			err_handler("accept() error");
		else
			printf("Connected Clinet %d\n", i +1);
		// read 가 블록킹이라 글자가 전달이 안된다.
		while((str_len = read(clnt_sock, msg, BUF_SIZE)) !=0)
			write(clnt_sock , "hello", str_len);

		close(clnt_sock);
```

```
        }
        close(serv_sock);

        return 0;
}
```

클라이언트.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>
#include<arpa/inet.h>
#include<sys/socket.h>
#include<string.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 , str_len;
        si serv_addr;
        char msg[32];
        char *m = "Input Message(q to quit): ";

        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 ....");

        for(;;)
        {
                if(!strcmp(msg, "q\n") || !strcmp(msg, "g\n"))
                        break;

                write(sock, "hi", strlen(msg));
                str_len = read(sock, msg, BUF_SIZE -1);
```

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

                msg[str_len] = 0;
                printf("msg from serv: %s\n", msg);
                sleep(10);

        }
        close(sock);
        return 0;
}
```

65. Shared Memory 를 통해 임의의 파일을 읽고 그 내용을 공유하도록 프로그래밍하시오.

헤더파일.

```
#include<sys/ipc.h>
#include<sys/shm.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<errno.h>


typedef struct
{
        char name[20];
        int score;
}SHM_t;

int CreateSHM(long key);
int OpenSHM(long key);
SHM_t *GetPtrSHM(int shmid);
int FreePtrSHM(SHM_t *shmptr);



////////////////////////////////////////////////////////////////////////////////////////////////
```

shm .c 파일

```
#include "shm.h"

int CreateSHM(long key)
{
        //      0777 의 권한을 준다.
        return shmget(key, sizeof(SHM_t), IPC_CREAT | 0777);
}

int OpenSHM(long key)
{
        return shmget(key, sizeof(SHM_t),0);
        // 나는 이포인터를 쉐어드 메모리로 지정할 것이다.
        // 그럼 공유된 페이지 주소를 얻는다. 물리주소에 있는...
}

SHM_t *GetPtrSHM(int shmid)
{
        // at 는 공유메모리의 장소라고 생각하면 된다. 공유메로리 물리주소.
        // id 값을 가지고 찾는다. 맨 첫번째 주소부터 찾겠다.
        return (SHM_t *)shmat(shmid, (char *)0,0);
}
int FreePtrSHM(SHM_t *shmptr)
{
        return shmdt((char*)shmptr);
}
```

//
read 하기.**(**이미 공유가 되게 리는 하는 것임**)**

```
#include "shm.h"

int main(void)
{
        // 메모리 아이디
        int mid;
        // 쉐어드 메모리
        SHM_t *p;

        mid = OpenSHM(0x888);
        //쉐어드 메모리의 값을 얻어와라.
        p = GetPtrSHM(mid);

        //엔터를 치면
        getchar();
        // 아무개를 얻어옴 구조체의 공간을 가져온거임.
        strcpy(p->name,"아무개");
        //
        p->score = 93;
        // 쉐어드 메모리 다 썼으니 해제 한다.
        FreePtrSHM(p);
        return 0;
}
```

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-Tasking 의 원리에 대해 서술하시오.
(Run Queue, Wait Queue, CPU 에 기초하여 서술하시오)

멀티 태스킹이랑 여러가지 태스크가 마치 동시에 실행하듯이 동작하는 것인데…
cpu 가 할당해주는 **time slice** 에 따라 여러 태스크들이 여러가지 우선순위 기법들에 의해 **run queue** 로 **process** 를 돌리며 나머지는 **wait queue** 에서 대기 하며 동작하는 방식이다**.**

68.현재 삽입된 디바이스 드라이버의 리스트를 보는 명령어는 무엇인가 ?

lsblk

69.System Call Mechanism 에 대해 기술하시오.

시스템 커널 내부에 있는 SYSCALL 에서 명령어를 불러서 필요시 랩핑 시켜 사용한다.

70.Process 와 VM 과의 관계에 대해 기술하시오.

Vm (가상메모리**)** 와 **process** 의 관계는 프로세스가 실행될 시 **tack_sturct**→ **mm_struct** 에 의해 스텍에 메모리를 주소가 올라가게 된다 이때 간접 참조를 통해 여러 테이블을 할당 받고 **page frame** 단위로 기록이 된다**.**

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

명령어를 쓴다면

cat -n 5 file 이름

tail -n 5 file 이름

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

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

void recursive_dir(char *dname);

int main(int argc, char *argv[])
{
        recursive_dir(".");
        return 0;
}

void recursive_dir(char *dname)
{
        struct dirent *p;
        struct stat buf;
        DIR *dp;
        chdir(dname);
        dp = opendir(".");
        printf("\t%s:\n",dname);
        while(p = readdir(dp))
                printf("%s\n",p->d_name);
        rewinddir(dp); // 되감기 한다. 포인터의 위치를 맨앞으로 가져다 준다.
        while(p = readdir(dp))
        {
                stat(p->d_name, &buf);
                if(S_ISDIR(buf.st_mode))
                        if(strcmp(p->d_name,".") && strcmp(p->d_name,".."))
                        // 0 일때 돌아가는거니 둘다 아닐 때만 리컬시브로 어가란 뜻이다.
                                recursive_dir(p->d_name);
        }
        // 만약에 chdir 하기전에 마지막이.이면 상위로 가면 안되니까 if 로 .이 아닐때만.
        // <시험 문제> 상위돌다가 바꿔야하는 프로그램.
        chdir("..");
        closedir(dp);
}
```

73.Linux 에서 fork()를 수행하면 Process 를 생성한다.
이때 부모 프로세스를 gdb 에서 디버깅하고자하면 어떤 명령어를 입력해야 하는가 ?

74.C.O.W Architecture 에 대해 기술하시오.

Copy on write 기술이다. 쓰기를 할 때만 복사를 한다.
한번에 복사 하는 것이 아니라 단계별로 복사할때 쓴다.

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

블록을 하면 대기를 하게 된다.**wait** 과 **read** 의 경우를 **block** 이라고 할수 있으며 **blocking** 이 없을 시에 통신을 하기 어렵다고 보면 된다.

논블럭을 하면 대기를 하지 않고 넘어가게 된다.

76.자식이 정상 종료되었는지 비정상 종료되었는지 파악하는 프로그램을 작성하시오.

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

void term_status(int status)
{
        if(WIFEXITED(status))
                printf("(exit)status :0x%x\n",WEXITSTATUS(status));

        else if(WTERMSIG(status))
                printf("(siganl)status : 0x%x,%s\n",
                                status & 0x7f, WCOREDUMP(status) ? "core dumped" :"");
}

void my_sig(int signo)
{
        int status;
        wait(&status);
        term_status(status);
}

int main(void)
{

        pid_t pid;
```

```
        int i;

        signal(SIGCHLD, my_sig);
        if((pid = fork())>0)
                for(i=0;i<10000;i++)
                {
                        usleep(50000);
                        printf("%d\n",i+1);
                }
        else if(pid == 0)
                sleep(5);
        else
        {
                perror("fork()");
                exit(-1);
        }
        return 0;
}
```

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

78.SIGINT 는 무시하고 SIGQUIT 을 맞으면 죽는 프로그램을 작성하시오.
79.goto 는 굉장히 유용한 C 언어 문법이다.
   그러나 어떤 경우에는 goto 를 쓰기가 힘든 경우가 존재한다.
   이 경우가 언제인지 기술하고 해당하는 경우
   문제를 어떤식으로 해결 해야 하는지 프로그래밍 해보시오.

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

81.stat(argv[2], &buf)일때 stat System Call 을 통해 채운 buf.st_mode 의 값에 대해 기술하시오.
82.프로세스들은 최소 메모리를 관리하기 위한 mm, 파일 디스크립터인 fd_array,
   그리고 signal 을 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 3 개 만들어보자!

(너무 길면 힘드니까 적당한 크기로 잡도록함)

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

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

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

88. 반 인원이 모두 참여할 수 있는 채팅 프로그램을 구현하시오.
Serv.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 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];

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

	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;

	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;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
clnt.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;
}

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

	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;
}
```

89. 88 번 답에 도배를 방지 기능을 추가하시오.

벙어리를 3 번 시키고 그후엔 나가게 한다.

<blocking_server.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/time.h>
#include<sys/epoll.h>
#include<malloc.h>

#define BUF_SIZE                128
#define MAX_CLNT                256
#define SET_TIMES               6

typedef struct timeval    tv;
typedef struct sockaddr_in              si;
typedef struct sockaddr *               sp;

int cnt[MAX_CLNT];
int clnt_cnt = 0;
int idx;
int clnt_socks[MAX_CLNT];
int thread_pid[MAX_CLNT];
double runtime=0.0;
double load_ratio;
int flag;

pthread_mutex_t mtx;
tv start, end;

typedef struct __count
{
        int send_client[0];
}count;


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

count *ct = 0;

double get_runtime(tv start, tv end)
{
        end.tv_usec = end.tv_usec - start.tv_usec;
        end.tv_sec = end.tv_sec - start.tv_sec;
        end.tv_usec += end.tv_sec * 1000000;

        if((end.tv_usec / 1000000.0) > 10)
        {
                gettimeofday(&start, NULL);
                //cnt[]
```

```
	}

	//printf("runtime = %lf sec\n", end.tv_usec / 1000000.0);
	return end.tv_usec / 1000000.0;
}

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 proc_msg(char *msg, int len, int k)
{
	int i;
	//int cmp = atoi(msg);
	char smsg[256] = {0};
	char clnt_count[BUF_SIZE];

	pthread_mutex_lock(&mtx);

	cnt[k] += 1;

	sprintf(smsg,"[<말한 횟수 : %d>]\n",cnt[k]);
	write(clnt_socks[k], smsg, strlen(smsg));

	printf("cnt = %d\n", cnt[k]);

/*
	if(data[k] > cmp)
		sprintf(smsg, "greater than %d\n", cmp);
	else if(data[k] < cmp)
		sprintf(smsg, "less than %d\n", cmp);
	else
	{
		sprintf(clnt_count,"[<말한 횟수(1.6):%d>]\n",cnt[k]);
		write(clnt_socks[k], clnt_count, 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];
	char clnt_count[BUF_SIZE];
	i = clnt_cnt - 1;
```

```
flag = 0;
//tv start, end;


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

gettimeofday(&start, NULL);
while((str_len = read(clnt_sock , msg, sizeof(msg))) != 0){
        // 현재 말한 횟수 세는것은 한명만 된다.

        //ct->send_client[clnt_cnt-1] += 1;
        proc_msg(msg, str_len, i);
        send_msg(msg, str_len);

        // 끝나는 시간을 구함.
        gettimeofday(&end, NULL);

        //pthread_mutex_unlock(&mtx);

        runtime = get_runtime(start, end);
        if(runtime > 3)
        {
                gettimeofday(&start, NULL);
                cnt[i]=0;
        }
        load_ratio = cnt[i]/runtime;
        pthread_mutex_lock(&mtx);
        if(load_ratio > 5.7 || cnt[i]>15)
        {
                flag++;
                if(flag == 3){
                        write(clnt_socks[i], "너 이제 진짜 말못함\n", 128);
                        shutdown(clnt_socks[i], SHUT_WR);
                        break ;
                }
                write(clnt_socks[i], "당신은 잠시 벙어리\n", 128);
                sleep(6);
                while(read(clnt_sock, msg, sizeof(msg)) > BUF_SIZE-1){
                        memset(msg, 0, sizeof(msg));
                }
                memset(msg, 0, sizeof(msg));
                gettimeofday(&start, NULL);
                write(clnt_socks[i], "당신은 기적으로 말할 수 있게 되었습니다. \n", 128);
                cnt[i]=0;

        }
        pthread_mutex_unlock(&mtx);

        printf("runtime = %lf sec\n", runtime);
        printf("{load_ratio} = %lf sec\n", load_ratio);

}
```

```
	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;

	unsigned int i, cnt = 0;


	ct = (count*)malloc(1024);

	gettimeofday(&start, NULL);

	for(i = 0; i < 777777777; i++)
		cnt++;

	gettimeofday(&end, NULL);

	// 여기서 시간을 구한다.
	runtime = get_runtime(start, end);


	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);
		ct->send_client[clnt_cnt] = 0;
		clnt_socks[clnt_cnt++] = clnt_sock;
		pthread_mutex_unlock(&mtx);

		printf("clnt_cnt (사람수) : %d\n", clnt_cnt);

		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;
}
////////////////////////////////////////////////////////////////////////////
```

clnt.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_insi;
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;
}

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

```
        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;
}
```

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.

Server.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>
#include<malloc.h>

typedef struct sockaddr_in              si;
typedef struct sockaddr *               sap;

#define BUF_SIZE                32
#define EMPTY                   0


typedef struct __stack
{
        int data;
        struct __stack *link;
        //int fdata[0];
}Stack;


Stack *get_node()
{
        Stack *tmp;
        tmp = (Stack *)malloc(sizeof(Stack));
        tmp-> link = 0;
```

```
	return tmp ;
}

void push(Stack **top, int data)
{
	Stack *tmp;
	tmp = *top;
	*top = get_node();
	(*top) -> data = data;
	(*top) -> link = tmp;

}

int pop(Stack **top)
{
	Stack *tmp;
	int num;
	tmp = *top;
	if(tmp == NULL){
			printf("값이 없다\n");
			return 0;
	}
	num = tmp -> data;
	*top = (*top)->link;
	free(tmp);
	return num;
}

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

void read_childoroc(int sig)
{
	pid_t pid;
	int status;
	pid = waitpid(-1, &status , WNOHANG);
	printf("REMOVED oroc 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 , fd;
	char buf[BUF_SIZE] = {0};
	pthread_t t_id;

	Stack *top;
	top = EMPTY;
	push(&top, 10);
	push(&top, 20);
	push(&top, 30);
	//printf("%d\n", pop(&top));
```

```
//printf("%d\n", pop(&top));
//printf("%d\n", pop(&top));
//printf("%d\n", pop(&top));

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

	//write(clnt_sock , (Stack *)top, sizeof(Stack));

	pid = fork();
	write(clnt_sock , (Stack *)top, sizeof(Stack));
	if(pid == -1)
	{
		close(clnt_sock);
		continue;
	}
	if(pid == 0)
	{
		close(serv_sock);

		while((str_len = read(clnt_sock , (Stack *)top, BUF_SIZE)) != 0)
		write(clnt_sock , (Stack *)top, sizeof(Stack));

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

```
        }
        close(serv_sock);
        return 0;
}
```

//

clnt.c

```
#include<stdio.h>
#include<stdlib.h>
#include<malloc.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                32
#define EMPTY                   0

typedef struct __stack
{
        int data;
        struct __stack *link;
        //int fdata[0];
}Stack;


Stack *get_node()
{
        Stack *tmp;
        tmp = (Stack *)malloc(sizeof(Stack));
        tmp-> link = 0;
        return tmp ;
}
void err_handler(char *msg)
{
        fputs(msg, stderr);
        fputc('\n', stderr);
        exit(1);
}

void push(Stack **top, int data)
{
        Stack *tmp;
        tmp = *top;
        *top = get_node();
        (*top) -> data = data;
        (*top) -> link = tmp;

}
int pop(Stack **top)
{
        Stack *tmp;
        int num;
        *top = get_node();
        tmp = *top;
        if(tmp == NULL){
```

```
                printf("값이 없다\n");
                return 0;
        }
        if((*top)){
        num = (*top)-> data;
                *top = (*top)->link;
        }
        free(tmp);
        return num;
}

int main(int argc, char **argv)
{
        pid_t pid;
        int i, sock;
        si serv_addr;
        char buf[1024] = {0};
        char smsg[256];

        Stack *top=get_node();


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


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

        sock = socket(PF_INET, SOCK_STREAM, 0);


        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]));

        // 커넥트 하는 순간 어셉트가 동작한다.
        // 위에서 설정한 (포트와 주소 정보(serv_adr)) 를 가지고 connect 를 시도한다.
        if(connect(sock, (sap)&serv_addr, sizeof(serv_addr)) == -1)
                err_handler("connect() error");

        else
                puts("Connected .....");

        read(sock, (Stack *)top, sizeof(buf));

        if(!pid){

                //top = (Stack *)buf;
//              printf("%d\n", pop(&top));
//              printf("%d\n", pop(&top));
//              printf("%d\n", pop(&top));
                sprintf(smsg,"%d\n", pop(&top));
                printf("%s", smsg);
                write(sock , smsg, sizeof(Stack));
```

```
                sprintf(smsg,"%d\n", pop(&top));
                printf("%s", smsg);
                write(sock , smsg, sizeof(Stack));
                sprintf(smsg,"%d\n", pop(&top));
                printf("%s", smsg);
                write(sock , smsg, sizeof(Stack));
//              sprintf(smsg,"%d\n", pop(&top));
//              write(clnt_sock , (Stack *)&top, sizeof(Stack));
        }
        else
                //read_routine(sock, (Stack *)buf);

        close(sock);

        return 0;
}
```

93.Critical Section 이 무엇인지 기술하시오.

여러 태스크 들이 동시에 접근해서 꼬일 수 있는 구간이다**. thread** 를 사용할 때 각별히 주의를 해야 하며 해결 방법으로는 **mutex** 를 이용해서 메모리 가 공유 될수 있는 구간에서**, lock ,unlock** 을 해주어야 한다**.**

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

우리가 **fork()**를 수행 시키면 **c library** 라고 커널 개발자가 만든 라이브러리를 불러 **systemcall** 을 한다**.** 그리고 **syscall** 에 의해서 커널레벨로 가서 **_do_fork()**가 불러지지고 어셈블리어를 실행 하여 다시 **syscall** 이 온 지점으로 돌려준다**.**

95.리눅스 커널의 arch 디렉토리에 대해서 설명하시오.

하드웨어에 종속적인 부분들이 구현된 디렉토리이다**.** 아키택쳐를 가리킨다**.** 한마디로 **cpu** 의 타입에 따라 하우위 디렉토리로 구분된다**.**
ARM Ti 인텔 등으로 나뉘어져 있다**.**

96.95 번 문제에서 arm 디렉토리 내부에 대해 설명하도록 하시오.

Arm 을 기반으로 하는 **cpu** 의 필요한 함수들 드라이버들 등이 들어있다**.**

97.리눅스 커널 arch 디렉토리에서 c6x 가 무엇인지 기술하시오.
TI DSP 를 가르킨다**.**우리가 사용할 것이다 앞으로**...**

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

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 해야 할 일들이 쌓이고 쌓여서 결국에는 수습하지 못할정도로 많은 양이 쌓였을 수도 있다. 사람이란 것이 서 있으면 앉고 싶고 앉으면 눕고 싶고 누우면 자고 싶고 자면 일어나기 싫은 법이다.
내가 정말 죽을둥 살둥 이것을 이해하기 위해 열심히 했는지 고찰해보자! 2 개월간 자기 자신에 대한 반성과 성찰을 수행해보도록 한다. 또한 앞으로는 어떠한 자세로 임할 것인지 작성하시오.

지난 2 개월, 솔직히 말해 저녁에 남기도 하고 집에가서 나머지 부분을 채우느라 고생을 많이 했었다 . 처음 시작하는 마음은 가벼웠지만 지금은 조금 무거워진 기분이다.
그저 지금보다 나은 실력으로 어떤 회사를 가더라도 기본이 있고 내 스스로 답답한 사람, 부족한 사람으로 있기 싫었다. 그렇기 때문에 일은 그만두고 약 1 년의 시간을 포기 하고 공부하기로 마음을 먹게 되었다.
약 2 달이 흘렀고 벌써 많은 지식들을 알게 되어 가고 있다. 매일이 그 방대한 양의 지식을 흡수하려고 노력하며 아둥 버둥 하고 있지만 사실은 내가 흡수하는 양은 일부 이며 내일이 오면 바로 그 지식들은 썰물처럼 자국만 남는 것 같다.
그래도 나는 이 시간이 좋다. 내가 스스로 배우고자 나와 하나하나 알아가는 것이 좋다. 물론 부족한 머리로 새로 얻은 지식을 응용하고 발전 하기에는 시간이 매우 걸리고 수업 중간 이해 못하고 넘어가야하는 것들도 있지만 마냥 뒤쳐지고 앉아서 물러서게 되는 도태의 형태가 아닌 하루하루 열심이라는 형태가 나를 만족 시키고 있는 것 같다.
앞으로도 부족한 부분속에서 헤메고 넘어지고 힘들겠지만 넘어서려고 노력해보겠다. 게으름과 나의 다른 취미들이 나의 감정을 유혹 하지만 그래도 이 시간만큼은 노력해 보겠다.
꾸준하게 꾸준하게 이 모습으로 1 년을 이어갈 것이다.