I DSP,Xilinx zynq FPGA,MCU 및 Xilinx zynq FPGA 프로그래밍 전문가 과정

강사-INNOVA LEE(이상훈)
Gccompil3r@gmail.com
학생-윤지완
Yoonjw789 @naver.com

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.

<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);
}
void *send_msg(void *arg)
{
```

```
	int sock = *((int *)arg);
	char msg[BUF_SIZE];

	for(;;)
	{
		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(;;)
	{
		str_len = read(sock, msg, BUF_SIZE - 1);
		msg[str_len] = 0;
		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]));
    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;
}
```

<serv>

3.리눅스 커널은 운영체제(OS)다.

OS 가 관리해야 하는 제일 중요한 5 가지에 대해 기술하시오.

cpu 는 물리 적인 자원을 테스크라는 추상적인 자원으로 제공해주는 태스크 관리자

메모리를 세그먼트나 페이지 라는 개념으로 제공해주는 메모리 관리자.

디스크를 파일이라는 개념으로 제공해주는 파일 시스템

네트워크 장치를 소켓이라는 개념으로 제공해주는 네트워크 관리자

각종 자아치를 디바이스 드라이버를 통해 일관되게 접근하도록 해주는 디바이스 드라이버 이 5 가지가 있다.

4.Unix 계열의 모든 OS 는 모든 것을 무엇으로 관리하는가 ?

생명으로 제공한느 태스크와 장소를 제공하는 파일이라는 두 객체로 모든 지원을 한다.

5.리눅스에는 여러 장점이 있다.(배점 0.2 점)

아래의 장점들 각각에 대해 기술하라.

* 사용자 임의대로 재구성이 가능하다. :오픈 소스를 통해 일반 시민이 자기가 우너하는 프로그램을 설계할 수 있다.

* 열약한 환경에서도 HW 자원을 적절히 활용하여 동작한다. :최적화가 가능하다

* 커널의 크기가 작다.

* 완벽한 멀티유저, 멀티태스킹 시스템

* 뛰어난 안정성 :고가의 네트 워크 장비에서 사용될 정보로 안전하다

* 빠른 업그레이드 :

* 강력한 네트워크 지원

* 풍부한 소프트웨어

6. 32bit System 에서 User 와 Kernel 의 Space 구간을 적으시오

0~3GB :STACK, HEAP,DATA,TEXT 유져 구간

3~4GB:kernel 구간

7.Page Fault 가 발생했을때

운영체제가 어떻게 동작하는지 기술하시오.

:리눅스에서는 모든 것은 파일이다라는 마인드이기 때문에 kernel 도 예외는 아니다. kernel 도 사용장에게 서비스를 제공하기 위해서는 메모리를 할당을 받아야 한다. 그래서 kernel 도 페이지를 할당을 받을라 그러는데 페이지가 존재하지 않거나 다른 값들이 들어있어 할당을 받지 못하면 page 핸들러가 발동이 되서 유저는 세그먼트 fault kernel 은 핸들러가 페이지를 할당해주고 kernel 이 다시 페이지할당을 시도하고 할당을 받는 식이다.

8.리눅스 실행 파일 포맷이 무엇인지 적으시오

ELF (Executable and Linkable Format)

9.프로세스와 스레드를 구별하는 방법에 대해 기술하시오.

프로세스는 운영체제로부터 자원을 할당받는 작업의 단위. 쉽게 말해, 컴퓨터에서 실행중인 컴퓨터 프로그램을 의미합니다.

스레드 는 한개의 프로세스 내에서 동작되는 여러 실행의 흐름.

이것을 확인하기 위해서는 clone 을 사용하면 잘알수 있다. clone 은 프로세스 쓰레드 모두 커퍼가 가능하기때문이다.

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

task_struct 가 thread 가 생성될때마다 같이 생성 된다,

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

Current:현재 구동중인 프로세스 ID 등을 알 수 있다.

```
#include <linux/export.h>
#include <linux/mm.h>
#include <linux/utsname.h>
#include <linux/mman.h>
#include <linux/reboot.h>
#include <linux/prctl.h>
#include <linux/highuid.h>
#include <linux/fs.h>
#include <linux/kmod.h>
#include <linux/perf_event.h>
#include <linux/resource.h>
#include <linux/kernel.h>
#include <linux/workqueue.h>
#include <linux/capability.h>
#include <linux/device.h>
#include <linux/key.h>
#include <linux/times.h>
#include <linux/posix-timers.h>
#include <linux/security.h>
#include <linux/dcookies.h>
#include <linux/suspend.h>
#include <linux/tty.h>
#include <linux/signal.h>
#include <linux/cn_proc.h>
#include <linux/getcpu.h>
#include <linux/task_io_accounting_ops.h>
#include <linux/seccomp.h>
#include <linux/cpu.h>
#include <linux/personality.h>
#include <linux/ptrace.h>
#include <linux/fs_struct.h>
#include <linux/file.h>
#include <linux/mount.h>
#include <linux/gfp.h>
#include <linux/syscore_ops.h>
#include <linux/version.h>
#include <linux/ctype.h>
:830
```

```
SYSCALL_DEFINE0(getpid)
{
        return task_tgid_vnr(current);
}

/* Thread ID - the internal kernel "pid" */
SYSCALL_DEFINE0(gettid)
{
        return task_pid_vnr(current);
}

/*
 * Accessing ->real_parent is not SMP-safe, it could
 * change from under us. However, we can use a stale
 * value of ->real_parent under rcu_read_lock(), see
 * release_task()->call_rcu(delayed_put_task_struct).
 */
SYSCALL_DEFINE0(getppid)
{
        int pid;

        rcu_read_lock();
```

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

쓰레드는 프로세스의 heap, static, code 영역 등을 공유한다는 사실이다. 각각의 프로세스가 독립적인 stack, heap, code, data 영역을 가진 반면에, 한 프로세스에 속한 쓰레드는 stack 영역을 제외한 메모리 영역은 공유한다.

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 ?
Hw context 으로 context switch 할 때 태스크으 히녀재 실행 위치에 대한 정보를 유지하며,쓰레드 구조 or 하드웨어 레지스터 문맥이라 불린다.

실행 중이던 태스크가 대기상태나 준비상태로 전이 할 때 테스크가 어디까지 실행됬는지 기억하는 공간

14. 리눅스 커널의 스케쥴링 정책중 Deadline 방식에 대해 기술하시오.

기한부(deadline) 스케줄링 - nonpreemptive
- 작업을 명시된 시간이나 기한내에 완료되도록 계획
- 작업시간이나 상황등 정보를 미리 예측하기가 어렵다

15. TASK_INTURRUPTIBLE 과 TASK_UNINTERRUPTIBLE 은 왜 필요한지 기술하시오.

하드웨어가 동작하는 동안에 interrupt 가 오면 기존에 동작하는 값을 저장해야 될때 그 값이 task_intteruptible 에 저장이 된다.

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

O(N):N 의 동작을 하면 N 의 시간이 걸리는 것이고,O(1)은 딱 한개의 동작에만 걸리는 시간이다.즉 몇개가 있던 그 시간만 동작하기 때문에 O(N)이 더 좋은거 같다.

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.

2 번 cpu 에 위치하는게 좋다.
이유는 자식 태스크가 부모 태스크와 같은 cpu 에서 수행이 될때가 더 높은 캐쉬 친화력이 높아지기때문이다

18. 17 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.
이 경우 부하 균등(load-balancing)을 통해서 cpu 에 너무많이 채워지면 가장 비어있는 cpu 에 보낸다 그래서 2 번으로 보낸다.

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

UMA=SMP:한개의 메모리가 있고 여러개의 CPU 가 있지만 버스를 한가지만 존재하기 때문에 모든 CPU 는 메모리를 공유
NUMA:CPU 부하뿐아니라 메모리 접근 시간의 차이등도 고려하여 부하 균등을 시도한다.메모리는 한개이고 CPU 가 여러개인건 UMA 랑 같지만 NUMA 는 버스가 한개가 더 있다.즉 UMA 는 직렬 연결,NUMA 는 병렬 연결이라 생각하면 된다.

21. ZONE_HIGHMEM 에 대해 아는대로 기술하시오.
리눅스는 가상 공간 중 3~4GB 만 차치하기 때문에 수십 GB 의 메모리를 장착하는데 이것을 어떻게 할 수 있을까 ? 이 부분을 보완하고자 ZONE_HIGHMEM 이 나온것이다.
물리 메모리가 1GB 이 상이라면 896MB 까지를 커널 아상 공간과 1:1 로 매칭 시키고 나머지 부분은 필요할 때 공적으로 연결하는 구조이다. 가상 메모리를 페이징해서 물리 메모리와 연결하고 896MB 가 되면 나머지 부분을 간접참조 하겠다는 뜻이다.

22. 물리 메모리의 최소 단위를 무엇이라고 하며 크기가 얼마인가 ?
그리고 이러한 개념을 SW 적으로 구현한 구조체의 이름은 무엇인가 ?
Page frame:물리 메모리의 최소 단위이며 4KB 이다.
구조체 이름:node_mem_map

23. Linux Kernel 은 외부 단편화와 메모리 부하를 최소화하기 위해 Buddy 할당자를 사용한다.
Buddy 할당자의 Algorithm 을 자세히 기술하시오.
Buddy 할당:버디 할당은 4K*2 의 N 승으로 할당을 한다.
free_area 라는 배열의 각 엔트리의 할등을 하는 방식으로 만약 물리 메모리의 페이지가 총 10 까지 있는데 0~4 까지는 다 차있고 5:free 6~7:alloc 이 면 처음 order 는 4*2 의 0 승이니까 4KB 으로 할당을 하는데 0~4 까지는 채울것이 없고 5 번만 채울 수 가있다. 그 다음 두번째 order:4*2 의 1 승=8KB 이기 때문에 0~3 까지는 다 채워져 있으니 0 ,4~7 을 보면 5

번만 안재워져 있다.하지만 여기서 8KB 를 할당하기 때문에 5 번만 비워있다 그래서 채울 수는 없기에 0 을 넣는다. 한 프레임은 4KB 을 가지고 있기 때문에 페이지 2 개씩 기준으로 잡는다.

24. 21 번에 이어 내부 단편화를 최소화 하기 위해 Buddy 에서 Page 를 받아 Slab 할당자를 사용한다.

Slab 할당자는 어떤식으로 관리되는지 기술하시오.

버디 할당자는 큰 메모리를 할당할 때 쓰이는 것이고 SLAB 할당자는 적은 메모리를 할당할 때 쓴다.

Slab 할당자는 미리 메모를 잘개 쪼개놓고 user 가 메모리를 요청 했을때 미리 쪼갠 메모리를 할당해 주는것이다. (미분으로 쪼개는 느낌이랑 같다.) 내부 단편화로 인한 낭비는되는 공간은 없지만 공간을 쪼개는 것에서 속도가 느리다.

구성:slab full,slab free slab partial(채워진 부분과 빈 부분을 모두 가지고 있다.)

25. 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)

mm_struct

26. 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)

vm_area_struct

28. 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

리눅스 앞에서 사용자나 커널이나 모두 같은 파일이다.그러기에 커널이 사용자에게 서비스를 하기 위해서 주소를 가지고 있어야 되며 각 동작을 하기 위해(프로세스) 메모리를 할당하여 값을 push 로 저장해 사용자에게 요청이 왔을 때 그 데이터를 꺼내어 서비를 제공하기 때문이다.

29. VM(가상 메모리)와 PM(물리 메모리)를 관리하는데 있어

VM 을 PM 으로 변환시키는 Paging Mechanism 에 대해 Kernel 에 기반하여 서술하시오.

30. MMU(Memory Management Unit)의 주요 핵심 기능을 모두 적고 간략히 설명하시오.
가상 메모리를 실제 메모리로 변환을 시켜준다.
TLB 같은 페이지 테이블 엔트리 캐시를 사용하여 빠른 주소 변환을 지원하고 있다.
TLB:MMU 가 가지고 있는 캐시

31. 하드디스크의 최소 단위를 무엇이라 부르고 그 크기는 얼마인가 ?
크기:512byte
이름:섹터

34. task_struct 에서 super_block 이 하는 역할은 무엇인가 ?
파일 시스템을 기술하는 정보를 저장한다.파일 시스템마다 하나씩 존재한다. 슈퍼 블록의 자료 구조, 파일 시스템의 크기, 블록의 수, 이용가능한 빈 블록 목록, 빈 블록 목록에서 그 다음의 빈 블록을 가리키는 인덱스, 아이노드 목록의 크기, 빈 아이노드의 수, 빈 아이노드 목록, 빈 아이노드 목록에서 그 다음의 빈 아이노드를 가리키는 인덱스, 빈 블록과 빈 아이노드 목록들에 대한 록 필드들, 슈퍼 블록들 수정되었는지 나타내는 플래그, 파일 시스템의 이름, 디스크 이름 정보들이 저장된다.

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

가상 파일 시스템(Virtual File System, 줄여서 VFS)은 실제 파일 시스템 위의 추상 계층이다. VFS 의 목적은 클라이언트 응용 프로그램이 여러 파일 시스템에 같은 방법으로 접근할 수 있게 하는 것이다. 이를테면 VFS 를 사용하면 클라이언트 응용 프로그램은 로컬인 기억 장치에도 네트워크 위의 기억 장치에 직접적으로 접근할 수 있기 때문에 로컬과 네트워크의 차이를 느끼지 못한다. 또, 마이크로소프트 윈도, 맥 오에스, 유닉스 운영 체제의 차이를 넘어 파일 시스템의 차이를 느끼는 일 없이 접근할 수 있다.

VFS 는 커널과 실제 파일 시스템과의 인터페이스나 규격을 정의하고 있다. 그러므로 그 규격에 따라 간단히 새로운 파일 시스템을 커널에 추가할 수 있다. 파일 시스템은 새로운 공개 버전과의 호환성을 위해 수정하거나 다시 컴파일해야 할 수도 있다. 아니면 운영 체제에서 규격 변경에 대해 하위 호환 모드로 동작한다면 각 파일 시스템은 새로운 버전의 운영 체제에서도 그대로 사용할 수 있다.

37. 내부 인터럽트는 다시 크게 3 분류로 나눌 수 있다.
3 가지를 분류하시오.
Trap,fault,abort

38. 37 번에서 분류한 녀석들의 특징에 대해 기술하시오.
Fault:커널은 **fault** 를 일으킨 명령어를 **eip** 에 넣어 두었다가 해당 핸들러가 종료가 되면 **eip** 에 저장했던 주소로부터 다시 수행
Trap:trap 일으킨 명령어를 다음주소 **eip** 에 넣고 그 다음부터 실행

abort: 이것은 심각한 명령어 이기때문에 **eip** 에 넣어두지 않고 강제 종료 시킨다**.**

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.
ia32_sys_call_table

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

Intel:r3
arm:r7
42. Paging Mechanism 에서 핵심이 되는 Page Directory 는 mm_struct 의 어떤 변수가 가지고 있는가 ?
struct vm_area_struct *mmap;
follow_page_pte
pgd_offset
pud_offset
pmd_offset

44. 커널 내부에서 메모리 할당이 필요한 이유는 무엇인가 ?

45. 메모리를 불연속적으로 할당하는 기법은 무엇인가 ?
index 기법, 블록체인 할당,FAT 기법

46. 메모리를 연속적으로 할당하는 기법은 무엇인가 ?
캐시

47. Mutex 와 Semaphore 의 차이점을 기술하시오.

가장 큰 차이점은 동기화하는 갯수이다.
mutex 는 동기화 대상이 무조건 1 개이지만
semaphore 는 여러 대상한테 동기활 할때 사용

48. module_init() 함수 호출은 언제 이루어지는가 ?
module 이 시작이 될때

49. module_exit() 함수 호출은 언제 이루어지는가 ?
module 이 종료될때

50. thread_union 에 대해 기술하시오.

```
struct thread_info {
        unsigned long           flags;          /* low level flags */
        int                     preempt_count;  /* 0 => preemptable, <0 => bug */
        mm_segment_t            addr_limit;     /* address limit */
        struct task_struct      *task;          /* main task structure */
        __u32                   cpu;            /* cpu */
        __u32                   cpu_domain;     /* cpu domain */
        struct cpu_context_save cpu_context;    /* cpu context */
        __u32                   syscall;        /* syscall number */
        __u8                    used_cp[16];    /* thread used copro */
        unsigned long           tp_value[2];    /* TLS registers */
#ifdef CONFIG_CRUNCH
```

cpu 의 추상화는 프로세스이다 이것듦을 값을 꼬이지 않게 하기 위해 레이스터에 집어넣는데 이것들을 넣고 빼고 수없이 하다보면 값이 꼬이고 전혀 다른값이 출력 되기때문에 이것을 막기 위해 cpu_context_save 어 값이 넣어놓는다. 그리고 flag 는 스케줄링을 해야 할지 말지를 판단하는 변수이다.

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.

Kmalloc:kmalloc 은 데이터를 순차적으로 메모리에 집어 넣을수가 있다.
Vmalloc:대용량 데이터를 수용할 수 있다.
즉 kmalloc 은 적은 데이터를 차례대로 넣을순 있지만 대용량이 들어오면 동작이 되지않는다.이것을 보완하고자 vmalloc 을 쓴다.

55. OoO(Out-of-Order)인 비순차 실행에 대해 기술하라.
고성능 마이크로프로세서가 특정한 종류의 지연으로 인해 낭비될 수 있는 명령 사이클을 이용하는 패러다임

> **중앙 처리 장치(CPU)가 명령어를 실행할 때 전달된 순서에 따라 처리하지 않고 CPU 의 효율을 높이는 방향으로 명령어를 처리하는 것**
> **명령 실행 효율을 높이기 위해 순서에 따라 처리하지 않는 기법**
> **순차적 명령어 처리 기법으로 생기는 불필요한 기다림으로 인한 CPU 의 효율 저하를 방지하기 위해 앞의 결과에 종속적이지 않은 명령어를 먼저 처리하는 명령어 처리 기법**

```
1: a = b + c;
2: d = a + b;
3: z = x * y;
```

위와 같은 경우 2 번 명령은 1 번이 끝나지 않으면 미리 실행될 수 없다. 그런데 덩달아 아무 상관없는 3 번도 기다려야한다. 만약, 1, 2 번에 있는 명령들이 지독하게 느린 것들이라면 아무 죄없는 무관한 뒤따르는 명령어들도 모두 기다려야만 하고 이는 성능 저하로 이어진다.

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

항 목	CISC	RISC
명령어	명령어 개수가 많고 그 길이가 다양 하며 실행사이클도 각각 다름	명령어 수가 적고 길이가 고정적, 워드와 데이터버스 크기가 동일, 실행사이클도 모두 동일
레지스터	소수레지스터	다중레지스터 사용 및 최적화
메모리	많은 명령어가 메모리를 참조	메모리는 Load, Store 명령만 처리
파이프라이닝	파이프라이닝 기법을 사용하기 어려움	파이프라이닝을 사용하기 쉽고 많이 사용, 슈퍼 스칼라
제어방식	마이크로 프로그래싱 제어	Hardwired control
컴파일러	컴파일러가 복잡함 컴파일작업 실행 시 RISC 와 비교하여 Binary 코드의 길이가 짧다	단순한 컴파일러 구현 가능 컴파일작업 실행 시 CISC 와 비교하여 Binary 코드의 길이가 길다
회로구성	복잡함	단순함
프로그램	비교적 적게 명령어를 사용하여 결과를 얻을 수 있음	명령어가 적고 단순하므로 많은 수의 명령어가 조합되어 사용
사용환경	호환성이 중요한 환경	단순한 명령이 반복적으로 많이 사용되고 빠르게 처리되는 환경

76. 자식이 정상 종료되었는지 비정상 종료되었는지 파악하는 프로그램을 작성하시오.

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<errno.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":"");
}
int main(void)
{
pid_t pid;
int status;
if((pid = fork())>0)
{
wait(&status);
```

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.

```
term_status(status);
}
else if(pid == 0)
abort();//signal abort
else
{
perror("fork()");
exit(-1);
}
return 0;
}
```

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.

<해결 코드>

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

jmp_buf env;

void test (void )
{

longjmp(env,1);
}

int main(void)
{
int ret;
if((ret=setjmp(env))==0)//이 부분이 goto err 의 동작이다.
처음 리턴값은 무조건 0 이다.longjmp 하고 다시 이 부분으로 돌아오고  1 을 가지고 오고 ret=1 의 값을 가지므로 error 이 동작이 된다.
test();
else if(ret>0)
printf("error\n");
return 0;
}
```

goto 는 스택을 해제할 권한이 없기때문에 main 으로 다시 돌아오지 못하는 경우

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

block 연산은 해당하는 명령이 오기전까지 기다리면서 아무것도 하지 않기에 값이 정확하게 나온다.

nonblockihng 은 대량의 연산을 할때 사용하며 대용량 서버에서도 사용된다.

i

77. 데몬 프로세스를 작성하시오.(죽음을 모르는 데몬 구현)

```
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<signal.h>
#include<unistd.h>
#include<stdlib.h>
int main(void)
{
siganl(SIGINT, SIG_IGN);//SIGINT = CTRL+C 를 무시한다.
siganl(SIGQUIT, SIG_IGN);
siganl(SIGKILL, SIG_IGN);
pause();
return 0;
}
```

88. 반 인원이 모두 참여할 수 있는 채팅 프로그램을 구현하시오.

<serv>

```
#include "load_test.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>
#include <stdbool.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 cnt[MAX_CLNT];
pthread_mutex_t mtx;
typedef struct __iot{
        int sock;
        char ip[16];
        int cnt;
} iot;

iot info[BUF_SIZE];

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

```
void proc_msg(char *msg, int len, int sock)
{
	int i;

	pthread_mutex_lock(&mtx);

	for(i = 0; i < clnt_cnt; i++)
	{
		if(info[i].sock == sock)
			continue;
		write(info[i].sock, msg, len);
	}

	pthread_mutex_unlock(&mtx);
}

void *clnt_handler(void *arg)
{
	iot thread_info = *((iot *)arg);
	int len = 0, i;
	char msg[BUF_SIZE] = {0};

	tv start, end;
	double runtime = 0.0;
	double load_ratio;

	for(;;)
```

```
{
        gettimeofday(&start, NULL);
        len = read(thread_info.sock, msg, sizeof(msg));
        proc_msg(msg, len, thread_info.sock);
        gettimeofday(&end, NULL);

        runtime = get_runtime(start, end);

        load_ratio = 1.0 / runtime;
        printf("load_ratio = %lf\n", load_ratio);

        if(load_ratio > 1.5)
                thread_info.cnt++;

        if(thread_info.cnt > 10)
        {
                write(thread_info.sock, "You're Fired!!!\n", 16);
                add_black_list(thread_info.ip);
                goto end;
        }
}
pthread_mutex_lock(&mtx);

for(i = 0; i < clnt_cnt; i++)
{
        if(thread_info.sock == info[i].sock)
        {
                while(i++ < clnt_cnt - 1)
```

```
                        info[i].sock = info[i + 1].sock;
                    break;
                }
        }
        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;
                }
        }

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);
        }
```

```
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, MAX_CLNT) == -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);

	info[clnt_cnt].sock = clnt_sock;
	strcpy(info[clnt_cnt].ip, inet_ntoa(clnt_addr.sin_addr));
	info[clnt_cnt++].cnt = 0;

	pthread_mutex_unlock(&mtx);
```

```
            pthread_create(&t_id, NULL, clnt_handler, (void *)&info[clnt_cnt -
1]);
            pthread_detach(t_id);
            printf("Connected Client IP: %s\n", inet_ntoa(clnt_addr.sin_addr));
        }

        close(serv_sock);

        return 0;
}       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, MAX_CLNT) == -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);

            info[clnt_cnt].sock = clnt_sock;
            strcpy(info[clnt_cnt].ip, inet_ntoa(clnt_addr.sin_addr));
            info[clnt_cnt++].cnt = 0;

            pthread_mutex_unlock(&mtx);


            pthread_create(&t_id, NULL, clnt_handler, (void *)&info[clnt_cnt -
1]);
            pthread_detach(t_id);
            printf("Connected Client IP: %s\n", inet_ntoa(clnt_addr.sin_addr));
        }

        close(serv_sock);

        return 0;
}
<clnt>
```

```
#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] = "[ㄱㄱ]";
char msg[2048];

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

void make_rand_str(char *tmp)
{
```

```
        int i, end = rand() % 7 + 3;

        for(i = 0; i < end; i++)
                tmp[i] = rand() % 26 + 65;
}

void *send_msg(void *arg)
{
        int sock = *((int *)arg);
        char msg2[] = "https://kr.battle.net/heroes/ko/  <<== 지금 당장 접속하세
요!!\n";
        srand(time(NULL));

        char tmp1[32] = {0};

        for(;;)
        {
#if PASSIVE
                fgets(msg, BUF_SIZE, stdin);

                write(sock, msg, strlen(msg));
#endif
#if ATTACK
                make_rand_str(tmp1);

                printf("%s\n", msg);
                sprintf(msg, "%s %s %s", name, tmp1, msg2);
```

```
                printf("tmp1 = %s\n", tmp1);
                write(sock, msg, strlen(msg));
                sleep(5);
#endif
        }

        return NULL;
}

void *recv_msg(void *arg)
{
        int sock = *((int *)arg);
        char msg[NAME_SIZE + 2048];
        int str_len;

        for(;;)
        {
                str_len = read(sock, msg, NAME_SIZE + 2047);

                msg[str_len] = 0;
                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]));

    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.

<SERV>

```
#include "load_test.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>
#include <stdbool.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 cnt[MAX_CLNT];
pthread_mutex_t mtx;



typedef struct __iot{
```

```
        int sock;
        char ip[16];
        int cnt;
} iot;

iot info[BUF_SIZE];

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

void proc_msg(char *msg, int len, int sock)
{
        int i;

        pthread_mutex_lock(&mtx);

        for(i = 0; i < clnt_cnt; i++)
        {
                if(info[i].sock == sock)
                        continue;
                write(info[i].sock, msg, len);
        }

        pthread_mutex_unlock(&mtx);
```

```
}



void *clnt_handler(void *arg)
{
        iot thread_info = *((iot *)arg);
        int len = 0, i;
        char msg[BUF_SIZE] = {0};

        tv start, end;
        double runtime = 0.0;
        double load_ratio;

        for(;;)
        {
                gettimeofday(&start, NULL);
                len = read(thread_info.sock, msg, sizeof(msg));
                proc_msg(msg, len, thread_info.sock);
                gettimeofday(&end, NULL);

                runtime = get_runtime(start, end);

                load_ratio = 1.0 / runtime;
                printf("load_ratio = %lf\n", load_ratio);

                if(load_ratio > 1.5)
                        thread_info.cnt++;
```

```
            if(thread_info.cnt > 10)
            {
                write(thread_info.sock, "You're Fired!!!\n", 16);
                add_black_list(thread_info.ip);
                goto end;
            }
        }


#endif

end:
        pthread_mutex_lock(&mtx);

        for(i = 0; i < clnt_cnt; i++)
        {
            if(thread_info.sock == info[i].sock)
            {
                while(i++ < clnt_cnt - 1)
                    info[i].sock = info[i + 1].sock;
                break;
            }
        }

#if 0
        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;
            }
        }
#endif

        clnt_cnt--;
        pthread_mutex_unlock(&mtx);
        close(thread_info.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, MAX_CLNT) == -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);

            info[clnt_cnt].sock = clnt_sock;
            strcpy(info[clnt_cnt].ip, inet_ntoa(clnt_addr.sin_addr));
            info[clnt_cnt++].cnt = 0;

            pthread_mutex_unlock(&mtx);



            pthread_create(&t_id, NULL, clnt_handler, (void *)&info[clnt_cnt -
1]);
            pthread_detach(t_id);
            printf("Connected Client IP: %s\n", inet_ntoa(clnt_addr.sin_addr));
        }

        close(serv_sock);

        return 0;
}
```

i
<CLNT>
클라이언트는 위와 동일 하다.

90. 89 번 조차도 공격할 수 있는 프로그램을 작성하시오.
이번 프로그램은 serv 쪽은 같지만 클라이언트가 달라진다.

<clnt>

91. 네트워크 상에서 구조체를 전달할 수 있게 프로그래밍 하시오.

```
<clnt>
#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>

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

void read_proc(int sock, d *buf)
{
	for(;;)
	{
		int len = read(sock, buf, BUF_SIZE);

		if(!len)
			return;

		printf("msg from serv: %d, %f\n", buf->data, buf->fdata);
	}
}
```

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

	for(;;)
	{
		fgets(msg, BUF_SIZE, stdin);

		if(!strcmp(msg, "q\n") || !strcmp(msg, "Q\n"))
		{
			shutdown(sock, SHUT_WR);
			return;
		}

		buf->data = 3;
		buf->fdata = 7.7;

		write(sock, buf, sizeof(d));
	}
}

int main(int argc, char **argv)
{
	pid_t pid;
	int i, sock;
	si serv_addr;
	d struct_data;
	char buf[BUF_SIZE] = {0};

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

	sock = socket(PF_INET, SOCK_STREAM, 0);

	if(sock == -1)
```

```
            err_handler("socket() error");

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

        if(connect(sock, (sp)&serv_addr, sizeof(serv_addr)) == -1)
            err_handler("connect() error");
        else
            puts("Connected!\n");

        pid = fork();

        if(!pid)
            write_proc(sock, (d *)&struct_data);
        else
            read_proc(sock, (d *)&struct_data);

        close(sock);

        return 0;
}
```

93. Critical Section 이 무엇인지 기술하시오.

Critical section
여러 테스트가 동시에 접근해서 데이터 결과값이 달라지는 구간.코드 구현을 어셈블리어로 되어있다.
만약 A 프로세스와 B 프로세스가 동작을 하는데 A 의 데이터를 공유를 한다.그럼 A 의 동작이 끝나고 제어권이 B 프로세스로 넘어가서 B 프로세스가 동작을 해야되는 그 때 B 프로세스의 동작씹히고 다시 제어권이 A 프로세스로 오게되고 결국 최종값이 변하게 되는 상황이다.
그래서 이 상황을 방지하기 위해서 A 프로세스의 동작을 할 때 LOCK 을 걸어주는 것이다. 그럼 A 프로세스가 끝나기 전까지

는 아무도 건들일수 없기때문에 정상적인 값이 나온다.

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

파일에 접근하기 위해 추상화 시켜놓은 장치를 애기한다.
리눅스는 모든것을 파일로 보고있기 때문에 fd 를 이용해 핸들링을 할 수 가 있다.
100 번문제 자아성찰의 시간
그전에 시험보다 잘보기 위해서 나름 어머니 간병하면서 조금식 봤는데 드래도 다른분들의 하는양은 못쫓아가서 너무 아쉬웠고 thread 쓰는 부분이나 signal 의 대해서 좀이해가잘 가지 않았습니다.그래서 인터넷 찾아보고 그 전보다는 이해가 충분히 잘갔습니다 앞으로 집에 있는 노트북을 고쳐서 집에서 프로그램을 할 수 있도록 만들고 그날 해본건 그날 노트북으로 하겠습니다.

78. SIGINT 는 무시하고 SIGQUIT 을 맞으면 죽는 프로그램을 작성하시오.

```
#include<stdio.h>
#include<sys/stat.h>
#include<sys/typed.h>
#include<fcntl.h>
#include<unistd.h>
#include<stdlib.h>
#include<signal.h>


void my_signal(void signo)
{
printf(“죽었다\n”);
exit(0);
}
int main(void)
{
```

```
signal(SIGINT ,SIG_IGN);//SIGINT 는 무시한다.
signal(SIGQUIT,my_signal);
pause();
return 0;
}
```

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

사용자가 fork 를 호출을 했다면 라이브러리 함수에서 fork 라는 라이브러리 함수가 호출이 되고 이 함수를 cpu 에 법용 레지스터(r3)에 저장이 되어있고 라이브러리 fork 는 어셈블리어로 되어있다. Movl 2,%eax 이 코드의 뜻은 eax 가 범용 레지스터이고 고유 번호 2 번을 넣겠다는 것이다.(movl:법용 레지스터를 이동시켜준다, mov 랑 다르다)
그리고 int $0x80trap 을 거는 명령어가 int 이고 trap 이 걸리면 제어권은 kernel 이 가지고 가고 ,kernel 은 trap 의 번호와 맞는 엔트리를 찾아 system_call 을 호출을 하고 이 함수를 arch 에 위치해있다. 그럼 sysem_call_table 에서 eax 값을 통해서 탐색을 시작하고 sys_call_table 의 고유 번호 2 번을 찾고 sys_fork 를 얻고 사용자에게 보내 줄것이다.

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

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

<clnt>

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

```
#define BUF_SIZE 4

typedef struct sockaddr_in      si;
typedef struct sockaddr * sad;
void err_handler(char *msg)
{
fputs(msg , stderr);
fputc("\n",stderr);
exit(1);
}
void my_sig(void signo)
{
printf(“1 초후에 보낸다\n”);
}
int main(void)
{
char buff[BUF_SIZE]="hi";
char buff1[7]="0";
int sock;
si serv_addr;
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, (sad)&serv_addr, sizeof(serv_addr)) == -1)
err_handler("connect() error");
siganl(SIGALRM,my_sig);
for(;;)
{

read(sock,buff1,sizeof(buff1));
alarm(1);
write(sock,buff,sizeof(buff));
}
return 0;
}


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

```
#include <pthread.h>
#include <arpa/inet.h>
#include <sys/socket.h>

#define BUF_SIZE       7

typedef struct sockaddr_in            si;
typedef struct sockaddr *             sp;

void err_handler(char *msg)
{
fputs(msg,stderr);
fputc("\n",stderr);
exit(1);
}
void my_sig(void signo)
{
printf(“1 초후에 보낸다\n”);
}

int main(void)
{
si serv_addr,clnt_addr;
char buf[BUF_SIZE]="hello";
char buf1[3]="0";
int serv_sock,clnt_sock;


       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");

       addr_size = sizeof(clnt_addr);
       clnt_sock = accept(serv_sock, (sp)&clnt_addr, &addr_size);
         err_handler("accept()\n");
signal(SIGALRM,my_sig);
```

```
for(;;)
{
read(clnt_sock,buf1,sizeof(buf1));
alarm(1);
write(clnt_sock,buf,sizeof(buf));
}
return 0;
}
```