1 번

시스템 프로그래밍 5 점 문제

내용

파이프 통신을 구현하고 c type.c 라고 입력할 경우

현재 위치의 디렉토리에 type.c 파일을 생성하도록 프로그래밍하시오.

2 번

시스템 프로그래밍 5 점 문제 2

내용

369 게임을 작성하시오.

2 초내에 값을 입력하게 하시오.

박수를 쳐야 하는 경우를 Ctrl + C 를 누르도록 한다.

2 초 내에 값을 입력하지 못할 경우 게임이 오버되게 한다.

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

3 번

아래 물음에 답하시오.

내용

리눅스 커널은 운영체제(OS)다.

OS 가 관리해야 하는 제일 중요한 5 가지에 대해 기술하시오.

Filesystem manager , Memory manager , Task manager , Device manager , Network manager

4 번

아래 물음에 답하시오.

내용

Unix 계열의 모든 OS 는 모든 것을 무엇으로 관리하는가 ?

파일

5 번

눅스의 장점에 대한 각각의 상세한 기술	
내용	리눅스에는 여러 장점이 있다.(배점 0.2 점) 아래의 장점들 각각에 대해 기술하라. * 사용자 임의대로 재구성이 가능하다. * 열약한 환경에서도 HW 자원을 적절히 활용하여 동작한다. * 커널의 크기가 작다. * 완벽한 멀티유저, 멀티태스킹 시스템 * 뛰어난 안정성 * 빠른 업그레이드 * 강력한 네트워크 지원됨 * 풍부한 소프트웨어

* 사용자 임의대로 재구성이 가능하다.

C 언어 / 어셈블리어 공개 → 커스텀 가능(그들만의 리그..)

* 열약한 환경에서도 HW 자원을 적절히 활용하여 동작한다.

똥컴도 잘 돌아감 (최적화가 잘 되어있다.) 최고급 컴퓨터도 가능(안정적으로..)

* 커널의 크기가 작다.

윈도우는 크기가 크다. - 의미있나 ?

* 완벽한 멀티유저, 멀티태스킹 시스템

당연한 말.

* 뛰어난 안정성

네트워크가 되고 있다. (구글서버 생각하면 됨.)

* 빠른 업그레이드

업그레이드가 바로바로

* 강력한 네트워크 지원됨

TCP/IP

* 풍부한 소프트웨어

스톨만의 GNU(공개) 코드 몇 십만줄 공개

6 번

아래 물음에 답하시오.

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

4GB 로 1:3 비율로 커널은 위쪽 영역의 1 만큼 차지하고 유저 영역은 밑으로 3 만큼 차지 한다.

7 번

아래 물음에 답하시오.

내용 Page Fault 가 발생했을때

운영체제가 어떻게 동작하는지 기술하시오.

커널도 결국 프로그램이여서 메모리가 필요하다 그래서 메모리에 접근하여 메모리를 할당 받아야 하는데

접근 실패가 되어 Segmentation fault 가 뜨고 결국 그것이 Page Fault 다. 여기서 권한이 커널인 경우

Page Fault handling 이 다시 접근 권한을 주고 페이지를 할당할 수 있다

8 번

아래 물음에 답하시오.

내용 리눅스 실행 파일 포맷이 무엇인지 적으시오.

ELF

9 번

아래 물음에 답하시오.

내용 프로세스와 스레드를 구별하는 방법에 대해 기술하시오.

Pid 와 Tgid 값이 같은경우 프로세스 (쓰레드 리더)

Pid 와 Tgid 값이 다른 경우 쓰레드

10 번

아래 물음에 답하시오.

내용 Kernel 입장에서 Process 혹은 Thread 를 만들면 무엇을 생성하는가 ?

Task_sturct 구조체가 만들어 진다.

11 번

아래 물음에 답하시오.

내용

리눅스 커널 소스에 보면 current 라는 것이 보인다.

이것이 무엇을 의미하는 것인지 적으시오.

커널 소스 코드와 함께 기술하시오.

current 라는 매크로는 커널 내부에 정의되어 있는 매크로로써 현재 태스크의 task_struct 구조체를 가리킬 수 있게 해준다. task_tigd_vnr()은 해당 task_struct 구조체의 tgid 필드를 리턴한다. 따라서 이 함수는 task_struct 구조체의 tgid 필드 값을 리턴 하는 단순한 함수라고 볼 수 있다.

12 번

내용

아래 물음에 답하시오.

Memory Management 입장에서 Process 와 Thread 의 핵심적인 차이점은 무엇인가 ?

프로세스는 메모리를 공유하지 않지만 쓰레드는 메모리를 공유한다.

13 번

아래 물음에 답하시오.

내용

Task 가 관리해야하는 3 가지 Context 가 있다.

System Context, Memory Context, HW Context 가 있다.

이중 HW Context 는 무엇을 하기 위한 구조인가 ?

14 번

아래 물음에 답하시오.

내용 리눅스 커널의 스케쥴링 정책중 Deadline 방식에 대해 기술하시오.

*deadline 정책

기존 리눅스의 실시간 태스크 스케줄링 정책은 우선순위에 기반하여 스케줄링 대상을 선정하는데 반해 ,
deadline 정책은 잠시 후 설명될 deadline 이 가장 가까운 (즉 가장 급한) 태스크를 스케줄링 대상으로 선정한다.
예를 들어 동영상을 재생하는 태스크가 수행중 이라고 가정하자.
초당 30 프레임을 디코딩 하여 화면에 출력하면 태스크는 1 초당 30 번 씩 ‘해야하는 일’을 가지고 있는 것이다.
이 때, (1 초/ 30)보다는 적은 시간내에 수행해야할 ‘작업량’ 을 가질 수 있음을 의미한다.
여기서 ‘초당 30 회’는 period 라고 부르고 ‘작업량’은 runtime, ‘완료시간’을 DEADLINE 이라고 한다.
이 DEADLINE 정책을 사용하는 각 태스크들은 deadline 이용하여 RBTree 에 정렬되어 있으며, 스케줄러가 호
출되면 가장 가까운 deadline 을 가지는 태스크를 스케줄링 대상으로 선정한다. 즉 우선순위가 상관없다.
(영상, 음성, 스트리밍 등, 제약 시간을 가지는 수많은 응용들에 용이하다.)
*커널 내에서 (vi -t 로 들어간 직후, kernel 디렉토리에서) driving 하며 찾을 때, 명령어
:cs find 0 ‘파일이름’

15 번

다음 물음에 답하시오.

내용 TASK_INTURRUPTIBLE 과 TASK_UNINTERRUPTIBLE 은 왜 필요한지 기술하시오.

실행 상태에 있던 태스크가 특정한 사건을 기다려야 할 필요가 있으면 대기상태 (TASK_INTERRUPTIBLE,

TASK_UNINTERRUPTIBLE,TASK_KILLABLE)로 전이하기 때문에 특정 사건을 기다릴 때 T

ASK_INTURRUPTIBLE 과 TASK_UNINTERRUPTIBLE 가 필요하다 .

16

아래 물음에 답하시오.

내용

O(N)과 O(1) Algorithm 에 대해 기술하시오.

그리고 무엇이 어떤 경우에 더 좋은지 기술하시오.

태스크가의 개수가 늘어나면 그만큼 스케줄링에 걸리는 시간도 선형적으로

증가하며 (O(n)의 시간복잡도), 따라 스케줄링 시간의 예측이 불가능 하다.

O(N)은 데이터가 적을 때 좋다

O(1)은 데이터 많을 때 좋다.

아래 물음에 답하시오.

내용

현재 4 개의 CPU(0, 1, 2, 3)가 있고 각각의 RQ 에는 1, 2 개의 프로세스가 위치한다.

이 경우 2 번 CPU 에 있는 부모가 fork()를 수행하여 Task 를 만들어냈다.

이 Task 는 어디에 위치하는 것이 좋을까 ?

그리고 그 이유를 적으시오.

2 번 CPU

이유는 자식 태스크가 부모 태스크와 같은 CPU 에서 수행될 때 더 높은 캐시 친화력을 얻을 수 있기 때문이다.

18

아래 물음에 답하시오.

내용

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

이 경우 3 번에서 fork()를 수행하여 Task 를 만들었다.

이 Task 는 어디에 위치하는 것이 좋을까 ?

역시 이유를 적으시오.

2 번 CPU

부하균등 → 한가한 CPU 에 태스크를 이주시켜서 성능을 향상 시킨다.

19

아래 물음에 답하시오.

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

내용

커널 내부의 소스 코드와 함께 기술하도록 하시오.

멀티 프로세싱 기법 - UMA , NUMA

UMA(Uniform Memory Access , 균일 기억 장치 접근) - 뱅크 1 개

-모든 프로세스 들이 상호간에 연결되어서 하나의 메모리를 공유하는 기술

- 프로세스들은 메모리의 어느 영역이든지 접근이 가능하며 , 모든 프로세스가 걸리는 시간은 동일하다.

-구조가 간단하고, 프로그래밍 하기가 쉽다.

-메모리에 한번에 하나씩 연결이 가능하기에 커질수록 효율성이 떨어진다.

NUMA(Non-Uniformed Memory Access,불균일 기억 장치 접근) - 뱅크 2 개 이

-UMA 가 가지고 있는 모델의 한계를 극복하고 , 더 큰 시스템을 만들기 위해 만들어졌다.

-메모리에 접근하는 시간이, 프로세서와 메모리의 상대적인 위치에 따라서 달라진다.

20

아래 물음에 답하시오.

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

내용

21

아래 물음에 답하시오.

내용 ZONE_HIGHMEM 에 대해 아는대로 기술하시오.

리눅스는 가상 주소 공간 중 3~4GB 영역을 차지한다 여기서 가상 주소공간과 물리메모리 공간을 1:1 로 연결한다면 아무리 메모리를 사용하려 해봐야 1GB 이상은 접근할 수가 없을 것이다. 수 GB 장착하는 현재의 시스템에서 너무 치명적인 약점이다 그래서 물리메모리가 1GB 이상이라면 896MB 까지를 커널의 가상 주소 공간과 1:1 로 연결해주고 나머지는 동적으로 연결하여 사용하는 구조를 채택하였다 이 때 896MB 이상의 메모리 영역을 ZONE_HIGHMEM 이라 부른다.

22

다음 물음에 답하시오.

내용 물리 메모리의 최소 단위를 무엇이라고 하며 크기가 얼마인가 ?

그리고 이러한 개념을 SW 적으로 구현한 구조체의 이름은 무엇인가 ?

Page frame , 4kb

struct page{

}

23

다음 물음에 답하시오.

내용

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

Buddy 할당자의 Algorithm 을 자세히 기술하시오.

사용자가 1byte 가 필요시 1byte 단위로 할당해 버리면 이를 관리하는 메타데이터가 방대해 지기

때문에 1byte 보다 큰 단위로 메모리를 할당해 주어야 한다

리눅스는 물리 메모리의 최소 관리 단위인 페이지 프레임 단위로 할당하도록 결정하였다.

4k*2^n 으로 움직임. -리눅스 구현상 최대 할당 크기는 2^9x4KB 이다

만약 작은 단위로 크기를 요청할 경우 4KB 를 할당해주면 내부 단편화문제가 발생한다.

24

다음 문제에 답을 작성하시오.

내용

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

Slab 할당자는 어떤식으로 관리되는지 기술하시오.

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

1. 페이지 프레임을 받아서 이 공간을 나눈다.

2.버디 할당자로 부터 받지 말고 나뉜 공간으로 받는다.

3.해제시에는 버디로 반납하는게 아닌 미리 할당 받은 공간에 그대로 가지고 있는다.

4. 일종의 캐시로 사용하는 것이다.

이런 캐시 집합을 통해 메모리를 관리하는 정책을 slab 할당자라고 한다.

슬랭 할당자는 외부 인터페이스 함수로 kmalloc()/kfree()를 제공한다.

25

아래 물음에 답하시오.

내용

Kernel 은 Memory Management 를 수행하기 위해 VM(가상 메모리)를 관리한다.

가상 메모리의 구조인 Stack, Heap, Data, Text 는 어디에 기록되는가 ?

(Task 구조체의 어떠한 구조체가 이를 표현하는지 기술하시오)

```
mm_struct {
    unsigned long start_code, end_code, start_data, end_data;   //text 시작과 끝 , data 시작과 끝
    unsigned long start_brk, brk, start_stack; // heap 시작과 끝 , stack 의 시작
    }
```

26

다음 물음에 대해 답하시오.

내용

23 번에서 Stack, Heap, Data, Text 등 서로 같은 속성을 가진 Page 를 모아서 관리하기 위한 구조체 무엇인가 ?

(역시 Task 구조체의 어떠한 구조체에서 어떠한 식으로 연결되는지 기술하시오)

```
mm_struct{
        pgd   //페이지 테이블 관리 페이징 페이지 디렉토리 시작주소 들어있음(물리메모리)
        }
```

27. 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 ?

커널은 **inode** 로 파일의 속성 정보를 가지고 있는데 이것으로 파일의 내용을 접근가능하며
파일의 내용을 메모리에 올린다**.**

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.

커널도 결국 프로그램이라 메모리를 필요로 한다**.** 그러므로 **stack** 영역이 있다**.**

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 생성한 후 생성된 태스크에게 가상 주소공간을 제공해 주고 필요하다면 물리메모리의 일부를 할당해 준 뒤**(**페이징**)** 태스크가 원하는 디스크 상의 내용을 읽어 물리 메모리에 올려놓고 이 물리 메모리에 실제 주소와 태스크의 가상 주소공간을 연결해 주어야한다**.**

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

대부분의 **CPU** 는 가상 주소를 사용하는 운영체제가 원활히 수행될 수 있도록 하기 위해
가상 주소로부터 물리 주소로의 변환을 담당하는 별도의 하드웨어가 **MMU** 이다
이 하드웨어는 인텔 계열 **32bit CPU** 를 예로 들면 하드웨어적으로 **2** 단계의 페이지 테이블을 지원하며
알파 **CPU** 에서는 **3** 단계의 페이지 테이블을 지원한다**.**

31. 하드디스크의 최소 단위를 무엇이라 부르고 그 크기는 얼마인가 ?

Page frame , 4KB

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 ?

파일 시스템을 기술하는 정보를 저장한다**.** 파일 시스템마다 하나씩 존재

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

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

외부 인터럽트 **:** 현재 수행중인 태스크와 관련없는 주변장치에서 발생된 비동기적인 하드웨어적인 사건으 ㄹ의미 한다**.**

트랩 **:** 현재 수행중인 태스크와 관련있는 즉 동기적으로 발생하는 사건으로써 트랩이라고 부른다
트랩은 소프트웨어 적인 사건이며 예외 처리라고도 한다 **(ex 0** 으로 나누는 연산**,**세그멘테이션 결함**,**페이지 결함 보호 결함**,** 시스템 호출 등**)**

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

Fault , trap , abort

38. 35 번에서 분류한 녀석들의 특징에 대해 기술하시오.

Fault : 리눅스 커널은 **fault** 를 일으킨 명령어 주소를 **eip** 에 넣어 두었다가 해당 핸들러가 종료되고 나면 **eip** 에 저장되어 있는 주소부터 다시 수행을 시작한다

trap : 리눅스 커널은 **trap** 을 일으킨 명령어의 다음 주소를 **eip** 에 넣어 두었다가 그 다음부터 다시 수행한다

abort : 이는 심각한 에러인 경우이므로 **eip** 값을 저장해야할 필요가 없으며 현재 태스크를 강제 종료시키면 된다**.**

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

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.

sys_call_table

41. 38 에서 User Space 에서 System Call 번호를 전달한다.
Intel Machine 에서는 이를 어디에 저장하는가 ?
또한 ARM Machine 에서는 이를 어디에 저장하는가 ?

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

pgd

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

44. 커널 내부에서 메모리 할당이 필요한 이유는 무엇인가 ?

커널도 프로그램이기 때문에 메모리가 필요하다.

45. 메모리를 불연속적으로 할당하는 기법은 무엇인가 ?

블록체인 기법 **,** 인덱스 블록 기법 **, FAT**

46. 메모리를 연속적으로 할당하는 기법은 무엇인가 ?

47. Mutex 와 Semaphore 의 차이점을 기술하시오.

Spinlock 은 여러 프로세스에 적용할 수 없고 **semaphore** 는 프로세스에 여러 개 적용이 가능하다**.** 단순하고 간단할 땐 **spinlock** 이 좋고**, Context switching** 을 하는 것처럼 여러 프로세스를 실행해야 할 때에는 **semaphore lock** 형식이 좋다**. semaphore** 는 크고 대규모의 프로세스를 처리해야 할 때 사용된다**.**

48. module_init() 함수 호출은 언제 이루어지는가 ?

49. module_exit() 함수 호출은 언제 이루어지는가 ?

50. thread_union 에 대해 기술하시오.

Context Switching 을 수행할 때 태스크가 현재 어디까지 실행되었는지
기억해 놓는 공간이다

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)

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.

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!

55. OoO(Out-of-Order)인 비순차 실행에 대해 기술하라.
프로그래머는 프로그램을 짤 때 자신이 만든 코드가 차례대로(분기문이 없다면 위에서 아래로) 실행될 것으로 믿는다. 순차 프로세서는 이 순서를 그대로 지켜가며 실행한다. 그러나 어떤 메모리 로드 명령어가 캐시 미스를 겪고 있다고 하면 이를 뒤따르는 명령어 가운데 이 로드 명령어와 상관이 없는 것이라면 분명 미리 실행할 수 있을 것이다

56. Compiler 의 Instruction Scheduling 에 대해 기술하라.

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

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 ?

59. Pipeline 이 깨지는 경우에 대해 자세히 기술하시오.

call 이나 **jmp** 를 **cpu instrction** 레벨에서 분기 명령어라고 하고 이들은 **cpu** 파이프라인에 치명적인 손실을 가져다 준다 **.**
기본적으로 분기 명령어는 파이프라인을 부순다 **.**

Context switching 할 때마다 파이프라인이 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 ?

레지스터

61. Intel 의 Hyper Threading 기술에 대해 상세히 기술하시오.

Fork() 가 회로로 구현되었다**.(CPU n** 개면 **nx2** 개 처럼 동작하게 만드는 것**.**=공간 활용**)**

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.

63. 파일의 종류를 확인하는 프로그램을 작성하시도록 하시오.

yoosunglee@yoosunglee-Z20NH-AS51B5U: ~/Homework/yoosunglee/4.17

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

int main(int argc , char **argv)
{
        struct stat buf;
        char ch;
        stat(argv[1], &buf);
        if(S_ISDIR(buf.st_mode))
                ch ='d';
        if(S_ISREG(buf.st_mode))
                ch = '-';
        if(S_ISFIFO(buf.st_mode))
                ch ='P';
      if(S_ISLNK(buf.st_mode))
                ch ='l';
        if(S_ISSOCK(buf.st_mode))
                ch = 's';
        if(S_ISCHR(buf.st_mode))
                ch = 'c';
        if(S_ISBLK(buf.st_mode))
                ch = 'b';
        printf("%c\n",ch);

        return 0;
}
```

```
yoosunglee@yoosunglee-Z20NH-AS51B5U:~/Homework/yoosunglee/4.17$ ls
123  1.c  2.c  3.c  4.c  a.out  clnt  clnt.c  serv  serv.c
yoosunglee@yoosunglee-Z20NH-AS51B5U:~/Homework/yoosunglee/4.17$ ./a.out 123
d
yoosunglee@yoosunglee-Z20NH-AS51B5U:~/Homework/yoosunglee/4.17$ ./a.out 1.c
-
yoosunglee@yoosunglee-Z20NH-AS51B5U:~/Homework/yoosunglee/4.17$ ./a.out 2.c
-
yoosunglee@yoosunglee-Z20NH-AS51B5U:~/Homework/yoosunglee/4.17$
```

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

```
yoosunglee@yoosunglee-Z20NH-AS51B5U:~/Homework/yoosunglee/4.17$ vi clnt.c
yoosunglee@yoosunglee-Z20NH-AS51B5U:~/Homework/yoosunglee/4.17$ gcc -o clnt clnt.c
yoosunglee@yoosunglee-Z20NH-AS51B5U:~/Homework/yoosunglee/4.17$ ./clnt 127.0.0.1 7777
Connected ..........
Hi
msg form serv: hello
Hi
msg form serv: hello
Hi
msg form serv: hello
Hi
msg form serv: hello
Hi
msg form serv: hello
Hi
msg form serv: hello
Hi
^C
```

서버

#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() error");

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

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

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

   clnt_addr_size = sizeof(clnt_addr);

   for(i=0;i<5;i++){
```

```
      clnt_sock = accept(serv_sock,(struct sockaddr *)&clnt_addr, &clnt_addr_size);
      if(clnt_sock == -1)
          err_handler("accept() error");

       else
          printf("Conneted Client: %d\n",i+1);
       while((str_len = read(clnt_sock,msg,BUF_SIZE)) != 0)
            write(clnt_sock,"hello",6);
       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>

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[1024];
   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(;;){

        fputs("Hi\n",stdout);
            sleep(1);
        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 form serv: %s\n",msg);

    }

    close(sock);
    return 0;

}
```

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

한개의 실행 파일을 실행할 경우 그 자체가 프로세스 **1** 개
프로세스들이 아주 빠른 속도로 **CPU** 제어권을 넘겨주면서 **CPU** 를 사용한다면 주기를 느끼지 못한 순간에 모든 작업이 완료된다 프로세스가 제어권을 얻어서 구동 중인것은 런큐에 있는데 런큐에 올라갈 수 있는 프로세스의 수는 **CPU** 수 만큼이다 제어권을 얻지 못하는 프로세스는 **wait** 큐에 있다 프로세스들은 서로 제어권을 얻으려고 경쟁하는데 이 것을 **context switching** 이라고 한다**.** 이렇게 제어권을 경쟁하면서 빠른 속도로 제어권이 왔다 갔다하는데 이것이 너무 빨라 멀티태스킹하는 것처럼 보임**.**

68. 현재 삽입된 디바이스 드라이버의 리스트를 보는 명령어는 무엇인가 ?

69. System Call Mechanism 에 대해 기술하시오.

70. Process 와 VM 과의 관계에 대해 기술하시오.

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

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

yoosunglee@yoosunglee-Z20NH-AS51B5U: ~/Homework/yoosunglee

```c
#include<sys/types.h>
#include<dirent.h>
#include<stdio.h>

int main(void)
{
        DIR *dp;
        int i = 0 ;
        struct dirent *p;

        dp = opendir(".");
        while(p =readdir(dp))
        {
                if(p->d_name[0] =='.')
                continue;
                printf("%-16s",p->d_name);
                if((i+1)%5==0)
                printf("\n");
                i++;
        }
        printf("\n");
        closedir(dp);
        return 0;
}
```

```
yoosunglee@yoosunglee-Z20NH-AS51B5U:~/Homework/yoosunglee/4.17$ gcc 5.c
yoosunglee@yoosunglee-Z20NH-AS51B5U:~/Homework/yoosunglee/4.17$ ./a.out
4.c             5.c             3.c             clnt            2.c
clnt.c          1.c             a.out           123             serv
serv.c
yoosunglee@yoosunglee-Z20NH-AS51B5U:~/Homework/yoosunglee/4.17$ vi 5.c
yoosunglee@yoosunglee-Z20NH-AS51B5U:~/Homework/yoosunglee/4.17$
```

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.

fork()는 부모의 모든 자원을 복제하여 자식 프로세스로 넘겨주는데 많은 자원을 복제한다는 점에서 단순하고 비효율적.
Copy-on-write 란 데이터의 복제를 지연 또는 방지하는 기법. 즉 프로세스 주소 공간을 복제하는 대신 부모와 자식이 하나의 공간을 공유. 만약 데이터를 써넣을 일이 발생하면, 그제서야 주소 공간을 복제하여 자식에게 넘겨줌. 결국 자원의 복제는 쓰기가 발생할 때에만 일어나게 되고 복제 전까지는 부모와 자식이 읽기 전용 주소공간을 공유 이 기법은 대량의 데이터 복제를 방지하여 성능을 최적화

75. Blocking 연산과 Non-Blocking 연산의 차이점에 대해 기술하시오.
Blocking
반드시 순차적으로 이루어져야 하는 상황에 사용

Non-Blocking
다수의 빠른 통신이 이루어져야 하는 상황에 사용
유연하게 대처 가능.

76. 자식이 정상 종료되었는지 비정상 종료되었는지 파악하는 프로그램을 작성하시오.

yoosunglee@yoosunglee-Z20NH-AS51B5U: ~/Homework/yoosunglee/4.17

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


void term_status(int status) //자식이 정상 종료인지 비정상인지 보려는 함수
{
        if(WIFEXITED(status)) //자식 프로세스가 정상종료면 참
                printf("(exit)status : 0x%x\n",WEXITSTATUS(status));
        else if(WTERMSIG(status)) // 비정상일 경우, 자식 프로세스를 종료하도록 한 시그널의 번호를 반환한다.
                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);
                term_status(status);
        }
        else if(pid ==0)
                abort();
        else{
                perror("fork() " );
                exit(-1);
        }
        return 0;
}
~
~
```

```
yoosunglee@yoosunglee-Z20NH-AS51B5U:~/Homework/yoosunglee/4.17$ ./a.out
(signal)status : 0x6 , core dumped
yoosunglee@yoosunglee-Z20NH-AS51B5U:~/Homework/yoosunglee/4.17$ vi 6.c
```

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.

goto 가 함수내에서만 작용함 **..** **goto** 는 스택을 해제할 능력이 없다**..(**다른 함수를 만들면 스택이 만들어짐**.)**
→**setjmp** 랑 **longjmp** 이용한다**.**

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

jmp_buf env;

void test(void)
{
        longjmp(env,1); //longjmp->setjmp
}

int main(void)
{
         int ret;
         if((ret = setjmp(env)) == 0){

        printf("test\n");
        test();
```

```
        }

        else if(ret >0)
        printf("error\n");

        return 0;
}
```

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

파일 식별자

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.

89. 88 번 답에 도배를 방지 기능을 추가하시오.

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.

임계영역
여러 태스크를 동시에 접근해서 정보가 꼬일 수 있는 구간이다
그래서 안전하게 사용하려면 **semaphore** 와 **spinlock** 으로 락을 걸어서 다른정보가 진입하지 못하도록 해야한다**.**

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

Fork()가 수행되면 **stack(**지역**) heap(**동적**) data(**전역**) text(**기계어**) (vm** 레이아웃**)**이 그대로 복사 됨**.**

95. 리눅스 커널의 arch 디렉토리에 대해서 설명하시오.

리눅스 커널 기능 중 하드웨어 종속적인 부분들이 구현된 디렉터리이다**.**

이 디렉터리는 **CPU** 의 타입에 따라 하위 디렉터리로 다시 구분된다**.**

96. 95 번 문제에서 arm 디렉토리 내부에 대해 설명하도록 하시오.

ARM 계열**(**저전력 특징이 있어 휴대폰 등 모바일 장치에 많이 사용됨**)**

97. 리눅스 커널 arch 디렉토리에서 c6x 가 무엇인지 기술하시오.

Ti dsp

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

99. ARM 에서 System Call 을 사용할 때 사용하는 레지스터를 적으시오.

100. 벌써 2 개월째에 접어들었다.
그동안 굉장히 많은 것들을 배웠을 것이다.
상당수가 새벽 3 ~ 5 시에 자서 2 ~ 4 시간 자면서 다녔다.

또한 수업 이후 저녁 시간에 남아서 9 시 ~ 10 시까지 공부를 한 사람들도 있다.
하루 하루에 대한 자기 자신의 반성과 그 날 해야할 일을 미루지는 않았는지 성찰할 필요가 있다.
그 날 해야할 일들이 쌓이고 쌓여서 결국에는 수습하지 못할정도로 많은 양이 쌓였을 수도 있다.
사람이란 것이 서 있으면 앉고 싶고 앉으면 눕고 싶고 누우면 자고 싶고 자면 일어나기 싫은 법이다.

내가 정말 죽을둥 살둥 이것을 이해하기 위해 열심히 했는지 고찰해보자!
2 개월간 자기 자신에 대한 반성과 성찰을 수행해보도록 한다.
또한 앞으로는 어떠한 자세로 임할 것인지 작성하시오.

내가 정말 죽을둥 살둥 이것을 이해하기 위해 열심히 했는가.. 내 모습을 보면 열심히 하지 않았던 것 같다.
어려운 내용인것은 잘 알고 있지만 너무 어려워서 멘붕이 많이 왔는데 수업에 못 따라간 것을 남아서 해도 채워지지 않는게 하루하루가 찝찝하고 잘 때도 편히 못 잤던 것 같다;; 하루를 못 따라가니 이 갭을 따라갈 수 가 없었던 것이 너무 힘들었다; 그래서 그런지 더 놓게 되는게 많았다..
최근 일주일에서 이주일 동안 느낀 것이 정말 노력을 하지 않으면 따라 갈 수도 없고 남는게 없다라는 느낌이 많이 들었다 다시 이런 생각이 들지 않도록 열심히 할 생각이다.
앞으로 남은 과정 화이팅 !ㅎㅎ