TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – 최대성
c3d4s19@naver.com

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

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.

<init_sock.h 파일>

```
#include "init_sock.h"

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

void init_sock_addr(int type, char** argv, si addr, int* sock){
       *sock = socket(PF_INET, SOCK_STREAM, 0);

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

          memset(&addr, 0, sizeof(addr));
             addr.sin_family = AF_INET;

   if(type == SERV){
          addr.sin_addr.s_addr = htonl(INADDR_ANY);
          addr.sin_port = htons(atoi(argv[1]));

          if(bind(*sock, (sap)&addr, sizeof(addr)) == -1)
                  err_handler("bind() error");

          if(listen(*sock, 15) == -1)
                  err_handler("listen() error");
   }
   else if(type == CLNT){
      addr.sin_addr.s_addr = inet_addr(argv[1]);
      addr.sin_port = htons(atoi(argv[2]));

      if(connect(*sock, (sap)&addr, sizeof(addr)) == -1)
         err_handler("connect() error");
   }
}
```

<init_sock.c 파일>

```
#include "init_sock.h"

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

void init_sock_addr(int type, char** argv, si addr, int* sock){

       *sock = socket(PF_INET, SOCK_STREAM, 0);

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

          memset(&addr, 0, sizeof(addr));
             addr.sin_family = AF_INET;

   if(type == SERV){
          addr.sin_addr.s_addr = htonl(INADDR_ANY);
          addr.sin_port = htons(atoi(argv[1]));

          if(bind(*sock, (sap)&addr, sizeof(addr)) == -1)
                  err_handler("bind() error");

          if(listen(*sock, 15) == -1)
                  err_handler("listen() error");
   }
   else if(type == CLNT){
      addr.sin_addr.s_addr = inet_addr(argv[1]);
      addr.sin_port = htons(atoi(argv[2]));

      if(connect(*sock, (sap)&addr, sizeof(addr)) == -1)
          err_handler("connect() error");
   }
}
```

<server.c 파일>

```
#include <pthread.h>
#include <signal.h>
#include "init_sock.h"

#define BUF_SIZE    1024

socklen_t clnt_addr_size = sizeof(si);
```

```
typedef struct _sock_thread{
	int cnt;
	int socks[20];
	pthread_t pids[20];
}sock_thread;
sock_thread c_thread;
pthread_mutex_t mtx;

int cnt369(char num[]){
	int result = 0;
	int numlen = strlen(num);
	for(int i = 0; i < numlen; i++){
		if(num[i] == '3' || num[i] == '6' || num[i] == '9'){
			result++;
		}
	}
	return result;
}

		//369 게임 함수로 만들기
		//1. 시간초과-> sigalarm //2. 잘못 입력  //3. 제대로 입력(박수 or 숫자) 박수는 clnt 에서 특정 문자(@) 보내는걸로 처리함
void* clnt_handler(void* arg){
	int clnt_sock = *((int*)arg);
	char buf[BUF_SIZE] = "you are clnt\n";
	char msg[BUF_SIZE];
	sprintf(msg, "%s %d\n", buf , clnt_sock);

	int num369, clap, alarm_flag;
	char answer[BUF_SIZE];	//clnt 에서 보낸 답
	char correct[BUF_SIZE]; //정답

regame:
	num369 = 0;
	memset(buf, 0x00, BUF_SIZE);
	sprintf(buf, "--------------------\n  369 Game Start!  \n--------------------\n\n");
	write(clnt_sock, buf, sizeof(buf));

	while(1){
		memset(buf, 0x00, BUF_SIZE);
		memset(correct, 0x00, BUF_SIZE);
		sprintf(correct, "%d\n", ++num369);

		//3, 6, 9 숫자가 있는 경우 박수(@)로 변경
		if(clap = cnt369(correct)){
			int i;
			memset(correct, 0x00, BUF_SIZE);
			for(i = 0; i < clap; i++){
				correct[i] = '@';
			}
```

```
                        strcat(correct,"\n\0");
                }

                read(clnt_sock, answer, sizeof(answer)); // -> Blocking

                //제대로 입력시
                if(!strcmp(correct, answer)){
                        memset(buf, 0x00, BUF_SIZE);
                        sprintf(buf,"\ncorrect!\n\n");
                        write(clnt_sock, buf, sizeof(buf));
                }
                //잘못 입력
                else{
                        memset(buf, 0x00, BUF_SIZE);
                        sprintf(buf,"\nincorrect!\n\n");
                        write(clnt_sock, buf, sizeof(buf));
                        goto regame;
                }

        }
        close(clnt_sock);
   return NULL;

}

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

   int serv_sock;
        int clnt_sock;
        si serv_addr, clnt_addr;
        socklen_t addr_size; //
        pthread_t t_id;          //
        c_thread.cnt = 0;

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

   init_sock_addr(SERV, argv, serv_addr, &serv_sock);


        printf("check!\n");
        while(1){
                addr_size = sizeof(clnt_addr);
                clnt_sock = accept(serv_sock, (struct sockaddr *)&clnt_addr, &addr_size);
                if(clnt_sock == -1)
                        err_handler("accept() error");

                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));

                pthread_mutex_lock(&mtx);
                c_thread.socks[c_thread.cnt] = clnt_sock;
                c_thread.pids[c_thread.cnt++] = t_id;
                pthread_mutex_unlock(&mtx);

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

<clnt.c 파일>

```
#include <pthread.h>
#include <signal.h>
#include <setjmp.h>
#include "init_sock.h"

#define MSG_SIZE 1024

char s_msg[MSG_SIZE];

jmp_buf env;

//알람 시그널
void alarm_sig(int signo){
        longjmp(env, 1);
}

//박수 시그널 (Ctrl + c)
void clap_sig(int signo){
   printf("(clap) ");
   strcat(s_msg,"@");
}

void* recv_msg(void* arg){
   int clnt_sock = *((int*)arg);
   int str_len;
   char msg[MSG_SIZE];

   while(1){
      memset(msg, 0x00, MSG_SIZE);
      str_len = read(clnt_sock, msg, sizeof(msg)); // -> Blocking
      if(str_len == -1)
         err_handler("read() error!");
      write(1, msg, strlen(msg));
   }
   return NULL;
}

void* send_msg(void* arg){
```

```
    int clnt_sock = *((int*)arg);
    int str_len;
    char tmp[MSG_SIZE] = "";
    sigset_t t_sig_mask;
        signal(SIGALRM, alarm_sig);
    signal(SIGINT, clap_sig);
        sigemptyset(&t_sig_mask);
        sigaddset(&t_sig_mask, SIGALRM);

        if(setjmp(env) != 0){
      printf("    Time Over    ");
                memset(tmp, 0x00, MSG_SIZE);
                sprintf(tmp, "-\n");
                write(clnt_sock, tmp, sizeof(tmp));
        }
    pthread_sigmask(SIG_UNBLOCK, &t_sig_mask, NULL);

    while(1){
       memset(tmp, 0x00, MSG_SIZE);
       memset(s_msg, 0x00, MSG_SIZE);

       alarm(2);
       fgets(tmp, MSG_SIZE, stdin);    // -> Blocking
       strcat(s_msg, tmp);
       alarm(0);

       str_len = write(clnt_sock, s_msg, sizeof(s_msg));
       if(str_len == -1)
          err_handler("write() error!");
    }
    return NULL;
}

int main(int argc, char** argv){
    int clnt_sock;
    int str_len;
    char msg[32];
    si serv_addr;

    pthread_t read_pid, write_pid;
        sigset_t p_sig_mask;
        sigemptyset(&p_sig_mask);
        sigaddset(&p_sig_mask, SIGALRM);
        pthread_sigmask(SIG_BLOCK, &p_sig_mask, NULL);

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

    init_sock_addr(CLNT, argv, serv_addr, &clnt_sock);
```

```
    pthread_create(&read_pid, NULL, recv_msg, &clnt_sock);
    pthread_create(&write_pid, NULL, send_msg, &clnt_sock);
    pthread_join(read_pid, NULL);
    pthread_join(write_pid, NULL);

    close(clnt_sock);

    return 0;

}
```

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

Task Manage,
Filesystem Manage,
Memory Manage,
Device Manage,
Network Manage

4. Unix 계열의 모든 OS 는 모든 것을 무엇으로 관리하는가 ?

파일

5. 리눅스에는 여러 장점이 있다.(배점 0.2 점)
아래의 장점들 각각에 대해 기술하라.

* 사용자 임의대로 재구성이 가능하다.

→ 코드가 누구에게나 오픈되어있다.

* 열약한 환경에서도 HW 자원을 적절히 활용하여 동작한다.

→ 모놀리식 커널 구조로 되어있어서 자원 사용이 효율적이며
추가로 필요한 부분은 마이크로 커널 방식의 모듈을 부착하여 사용할 수 있다.

* 커널의 크기가 작다.

→ 하이브리드 커널 방식이라서 모듈을 사용하지 않을 경우 모놀리식 커널 방식과 비슷하여 작은 크기를 유지할 수 있다.

* 완벽한 멀티유저, 멀티태스킹 시스템

* 뛰어난 안정성

→ 소스코드가 오픈되어있어서 문제 발생시 사용자들이 빠르게 수정 가능하다.

* 빠른 업그레이드

→ 누구나 소스코드를 변경 할 수 있다.

* 강력한 네트워크 지원

* 풍부한 소프트웨어

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

User 가상공간: 0 ~ 3GB
Kernel 가상공간: 3GB ~ 4GB

7. Page Fault 가 발생했을때
운영체제가 어떻게 동작하는지 기술하시오.

유저의 경우는 segment fault 가 반환되지만 커널의 경우에는 Page handler 가 페이지를 할당받아서 다시 접근한다.

8. 리눅스 실행 파일 포맷이 무엇인지 적으시오.

ELF

9. 프로세스와 스레드를 구별하는 방법에 대해 기술하시오.

pid 와 tgid 를 비교하여 같으면 프로세스(리더) 다르면 tgid 의 스레드 맴버이다

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

task_struct 구조체 생성

11. 리눅스 커널 소스에 보면 current 라는 것이 보인다.
이것이 무엇을 의미하는 것인지 적으시오.

커널 소스 코드와 함께 기술하시오.

현재 진행중인 Task 를 가리키는 포인터

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

Process 간에는 기본적으로 서로 데이터 공유가 안되며 다른 IPC 방법을 사용하여 데이터를 교환한다
Thread 간에는 서로 데이터를 공유하며 데이터 동시 사용을 막기위해 뮤텍스를 이용한다

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 ?

System Context - Task 의 정보를 유지하기 위해 커널이 할당한 자료구조들
(task_struct, file descriptor, segment table, page table, file table, ...)
Memory Context - 가상 메모리(segment)의 Text, Data, Stack, Heap 영역, Swap 공간

Hardware Context - Task 의 현재 실행 위치 정보, 실행 중이던 Task 가 대기 상태나 준비 상태로 전이할 때 어디까지 실행됐는지 저장하는 공간

14. 리눅스 커널의 스케쥴링 정책중 Deadline 방식에 대해 기술하시오.

Task 마다 중요도에 따라 Deadline 을 정해주어서 스케줄링하는 방식

15. TASK_INTURRUPTIBLE 과 TASK_UNINTERRUPTIBLE 은 왜 필요한지 기술하시오.

Task 가 맡고 있는 일의 중요도에 따라서 인터럽트를 허용할지 말지 결정해주는 변수이다.

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

O(N)의 경우는 데이터의 양과 처리 속도가 비례한다는 의미이고
O(1)의 경우는 데이터의 양에 관계없이 처리 속도가 일정하다는 의미이다.
일반적으로 데이터의 양이 많을 경우 O(1)이 유리하고
상황에 따라 데이터의 양이 매우 적을 경우 O(N)이 유리하기도 하다.

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.

부모 프로세스와 같은 위치인 2 번에 위치하는 것이 유리하다
fork()로 생성된 자식 프로세스는 부모 프로세스의 메모리를 복사했기 때문에 같은 CPU 에서 관리할 경우 캐시 적중률이 높아져서 속도가 빨라진다.

18. 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.

CPU 들의 Task 처리량 불균형이 심할 경우에는 Task 처리량이 적은 CPU 로 위치시켜야 속도 면에서 유리하다.

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

UMA – 모든 CPU 가 메모리와 입출력 버스 등을 공유하는 구조(=SMP) 이며 병목 현상으로 성능상 불리함
NUMA – CPU 들을 몇 개의 그룹으로 나누고 각 그룹에게 별도의 지역 메모리를 주는 구조이다
메모리의 위치에 따라서 성능 차이가 발생하므로 각 구조에 적합한 메모리 정책을 사용해야 한다.

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

Static Priority – 0 ~ 99   (커널 영역)
Dynamic Priority – 100 ~ 139 (유저 영역)

21. ZONE_HIGHMEM 에 대해 아는대로 기술하시오.

32bit 운영체제에서는 가상메모리의 크기가 최대 4GB 라서 그 보다 큰 용량의 메모리들을 커버하기 위해서 ZONE_HIGHMEM 을 이용한다. 64Bit 운영체제의 경우는 가상메모리의 크기가 매우 크기 때문에 ZONE_HIGHMEM 을 사용하지 않고 ZONE_NORMAL 만 사용해도 충분히 커버 가능하다. ZONE_HIGHMEM 은 Table index 를 이용하여 메모리 관리 영역을 증폭시킨다.

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

물리 메모리의 최소단위 - 페이지 프레임(4KB)
SW 적으로 구현한 구조체 이름 - 페이지

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

메모리 관리 부하와 외부 단편화를 줄이기 위해서 Buddy 할당자를 사용하는데 요청하는 메모리 크기에 따라서

2 의 정수 제곱 만큼의 페이지 프레임(4KB) 개수를 할당해준다. 리눅스 구현상 최대 할당 크기는 (2^9 * 4KB) 이다. 버디 할당자는 페이지로 메모리를 관리하며 10 개의 Order(n)을 이용하여 page 의 2 의 n 승 개수마다 페이지가 비었는지 사용중인지에 따라 같으면 0 다르면 1 로 표시해서 요청한 크기에 따라 빠르게 메모리 할당이 가능한 알고리즘으로 되어있다.

24. 앞선 문제에 이어 내부 단편화를 최소화 하기 위해 Buddy 에서 Page 를 받아 Slab 할당자를 사용한다.
Slab 할당자는 어떤식으로 관리되는지 기술하시오.

Slab 은내부 단편화를 줄이기 위해 미리 4KB 를 할당받아와서 32byte 단위로 나누어 관리한다. 관리 방식은 Buddy 와 유사하며 할당 공간이 부족한 경우 함수를 이용해 버디로부터 필요한 용량을 더 할당받는다.

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 아래 있는 vm_area_struct 에서 Stack, Heap, Data, Text 를 관리한다.

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)

task_struct → mm_struct → vm_area_struct 구조체에서
Stack, Heap, Data, Text 등의 가상메모리를 관리한다

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 ?

task_struct → mm_struct →start_code & end_code 를 보면 가상메모리상의 Text 영역의 위치를 알수 있다.

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 프로세스라서 메모리 기본 구성인 Stack, Heap, Data, Text 등이 필요하다.

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

task_struct → mm_struct → pgd 에 페이지 디렉토리의 시작주소가 들어있는데 페이지 테이블의 index 를 이용하여 따라가다 보면 물리메모리 주소가 나온다.

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

CPU 가 메모리에 접근하는 것을 관리하는 하드웨어 부품이다.
하드웨어에서 병렬적으로 가상 메모리 주소를 실제 메모리 주소로 변환한다.

31. 하드디스크의 최소 단위를 무엇이라 부르고 그 크기는 얼마인가 ?

디스크 블록(4KB)이고 섹터 크기가 512byte 인 경우 섹터 8 개에 대응한다

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

task_struct → files-struct → file_operations 구조체

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

INLINE_SYS 함수에서 context switching 하고 커널로 제어권을 넘긴다

34. task_struct 에서 super_block 이 하는 역할은 무엇인가 ?

root 의 inode 정보 보관

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

디스크 파티션마다 블록디스크를 관리하는 방식이 다른데 관리 방식에 따라서 유저가 직접 다른방식의 함수들을 사용하는 것이 아니라 VFS 를 이용하여 같은 기능이지만 관리방식에 따라 다른 함수들을 커널과 유저 사이에서 하나로 묶어서 관리해준다.

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

내부 인터럽트: 소프트웨어적으로 발생하는 인터럽트이다.
외부 인터럽트: 하드웨어에서 발생하는 인터럽트를 말한다

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

System call
Trap
프로그램 검사 인터럽트

38. 앞선 문제에서 분류한 녀석들의 특징에 대해 기술하시오.

System call
→ 응용 프로그램에서 커널로 접근
Trap
→ 잘못된 명령이나 잘못된 데이터 사용시 발생
프로그램 검사 인터럽트
→ Division by zero, Exception 등

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

장치파일의 주번호는 같지만 부번호를 다르게 붙여주어서 공유한다.

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 구조체

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 → 페이지 테이블 관리 (페이지 디렉토리 시작주소가 들어있음)

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 프로세스이기 때문에 메모리가 필요하다

45. 메모리를 불연속적으로 할당하는 기법은 무엇인가 ?
vmalloc()

46. 메모리를 연속적으로 할당하는 기법은 무엇인가 ?
kmalloc()

47. Mutex 와 Semaphore 의 차이점을 기술하시오.

Mutex 는 스레드끼리 공유하는 자원관리에 사용
Semaphore 는 프로세스끼리 공유하는 자원 관리에 사용

48. module_init() 함수 호출은 언제 이루어지는가 ?
insmod 명령어를 실행할 때

49. module_exit() 함수 호출은 언제 이루어지는가 ?
rmmod 명령어를 실행할 때

50. thread_union 에 대해 기술하시오.
current 의 task_struct 정보를 가리키는 thread_info 구조체를 가지고 있다

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)

task_struct → files_struct → file → inode → i_rdev (주번호 부번호 저장)

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.

/dev 디렉토리 밑에 장치파일을 생성하고 i_name 에는 파일명을, i_rdev 에는 주번호와 부번호를, i_mode 에는 장치 파일 유형을 저장한다.

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()의 경우는 연속적인 메모리를 가져오는데 이럴 경우 성능은 빨라지지만 메모리 공간 관리에 불리하다
vmalloc()의 경우는 불연속적인 메모리를 가져와서 공간관리에 효율적이지만 성능이 낮아진다.
__get_free_pages()는 페이지를 할당해주는 함수이고 나머지 kfree(), vfree(), free_pages()는 위의 할당받은 함수에 따라서 메모리를 해제해준다. 효율적으로 메모리를 관리하기 위해 할당받는 함수가 다양하게 존재한다

54. Character Device Driver 를 아래와 같이 동작하게 만드시오.(10)
read(fd, buf, 10)을 동작시킬 경우 1 ~ 10 까지의 덧셈을 반환하도록 한다.

write(fd, buf, 5)를 동작시킬 경우 1 ~ 5 곱셈을 반환하도록 한다.

close(fd)를 수행하면 Kernel 내에서 "Finalize Device Driver"가 출력되게 하라!

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

56. 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 어느정도의 Instruction Scheduling 을 HW 의 힘만으로 수행할 수 있다.

이러한 것을 무엇이라 부르는가 ?

59. Pipeline 이 깨지는 경우에 대해 자세히 기술하시오.

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 ?

61. Intel 의 Hyper Threading 기술에 대해 상세히 기술하시오.

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. (10)

63. 파일의 종류를 확인하는 프로그램을 작성하시도록 하시오.

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

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)

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.

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.

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

76. 자식이 정상 종료되었는지 비정상 종료되었는지 파악하는 프로그램을 작성하시오.

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. 앞선 문제까지 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. 앞선 문제의 답에 도배를 방지 기능을 추가하시오.

90. 앞서 만든 프로그램 조차도 공격할 수 있는 프로그램을 작성하시오.

91. 네트워크 상에서 구조체를 전달할 수 있게 프로그래밍 하시오.

92. 앞선 문제를 응용하여 Queue 와 Network 프로그래밍을 연동하시오.

93. Critical Section 이 무엇인지 기술하시오.

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

95. 리눅스 커널의 arch 디렉토리에 대해서 설명하시오.

96. 앞선 문제에서 arm 디렉토리 내부에 대해 설명하도록 하시오.

97. 리눅스 커널 arch 디렉토리에서 c6x 가 무엇인지 기술하시오.

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

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

100. 벌써 2 개월째에 접어들었다.
그동안 굉장히 많은 것들을 배웠을 것이다.

상당수가 새벽 3 ~ 5 시에 자서 2 ~ 4 시간 자면서 다녔다.

또한 수업 이후 저녁 시간에 남아서 9 시 ~ 10 시까지 공부를 한 사람들도 있다.

하루 하루에 대한 자기 자신의 반성과 그 날 해야할 일을 미루지는 않았는지 성찰할 필요가 있다.

그 날 해야할 일들이 쌓이고 쌓여서 결국에는 수습하지 못할정도로 많은 양이 쌓였을 수도 있다.

사람이란 것이 서 있으면 앉고 싶고 앉으면 눕고 싶고 누우면 자고 싶고 자면 일어나기 싫은 법이다.

내가 정말 죽을둥 살둥 이것을 이해하기 위해 열심히 했는지 고찰해보자!

2 개월간 자기 자신에 대한 반성과 성찰을 수행해보도록 한다.

(맨 처음 과정에 들어왔을 때의 모습과 현재의 모습에 어떠한 발전이 있었는지 등등을 자세히 기술하도록 한다)

또한 앞으로는 어떠한 자세로 임할 것인지 작성하시오.

솔직히 교육과정의 커리큘럼 내용들이 많이 어렵고 벅찬 느낌이 강해서 왠만한 노력으로는 지식을 모두 흡수하는 게 불가능 하겠다는 생각이 듭니다. 그래도 매일 노력해서 지식의 일부분이라도 차근차근 알아가게 된다면 수업 내용을 모두 이해하지는 못하더라도 교육과정을 듣기 전의 모습과 비교해서는 매우 큰 성장이 있을 것이라는 확신이 듭니다. 그래서 매일 집으로 돌아가는 길에 모든 내용을 이해하지 못했다는 약간의 좌절감이 들지만 교육 과정을 성실히 수료하고 크게 성장해 있을 제 모습을 상상하며 매번 다시 마음을 다집니다. 2 개월이라는 시간 동안 치열하게 정말 많은 내용을 배워서 그런지 지금까지의 시간들이 매우 길게 느껴집니다. 교육 과정을 듣기 전에 리눅스 스펠링도 모르던 제가 리눅스 커널 시스템 프로그래밍을 하고 있다는 것이 지금도 신기하게 느껴집니다. 사실 지금까지 수업이 많이 힘들었고 앞으로 남은 과정도 많아서 수업 내용을 잘 이해할 수 있을까 하는 걱정이 앞서긴 하지만 이해하지 못하는 것보다 성실함을 잃는 것을 두려워하며 꾸준히 배워나가겠습니다.

(풀지 못한 문제 굵게 표시)