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

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – 장성환
redmk1025@gmail.com

5 문제 리눅스의 장점에 대한 각각의 상세한 기술 내용
리눅스에는 여러 장점이 있다.(배점 0.2 점)

*** 사용자 임의대로 재구성이 가능하다.**
***** 사용자 임의대로 재구성이 가능하다**.**

리눅스 커널이 **F/OSS** 에 속하여 있으므로 소스 코드를 원하는 대로 수정할 수 있다**.**
그러나 **License** 부분을 조심해야 한다**.**
*** 열약한 환경에서도 HW 자원을 적절히 활용하여 동작한다.**
리눅스 커널이 가볍고 좋을뿐만 아니라 소스가 공개되어 있어
다양한 분야의 사람들이 지속적으로 개발하여 어떠한 열약한 환경에서도 잘 동작한다**.**
*** 커널의 크기가 작다.**
최적화가 잘 되어 있다는 뜻
*** 완벽한 멀티유저, 멀티태스킹 시스템**
리눅스는 RT Scheduling **방식을 채택하여** Multi-Tasking **을 효율적으로 잘 해낸다**.
*** 뛰어난 안정성**
전 세계의 많은 개발자들이 지속적으로 유지보수하여 안정성이 뛰어남
*** 빠른 업그레이드**

전 세계의 많은 개발자들이 지속적으로 유지보수하여 안정성이 뛰어남
*** 강력한 네트워크 지원**
TCP/IP Stack **이 원채 잘 되어 있다보니** Router **및** Switch **등의 장비에서 사용함**
*** 풍부한 소프트웨어**
~~GNU(GNU is Not Unix) **정신에 입각하여 많은** Tool **들이 개발되어 있다**.~~

11 문제 아래 물음에 답하시오. 내용
리눅스 커널 소스에 보면 current 라는 것이 보인다.

이것이 무엇을 의미하는 것인지 적으시오.

커널 소스 코드와 함께 기술하시오.

current 는 현재 태스크의 task_struct 구조체를 가리킬 수 있게 해준다.

먼저 **vi -t current** 로 검색하면
아래 헤더 파일에 **x86** 에 한하여 관련 정보를 확인할 수 있다**.**

arch/x86/include/asm/current.h

여기서 **get_current()** 매크로를 살펴보면 **ARM** 의 경우에는 아래 파일에

include/asm-generic/current.h

thread_info->task 를 확인할 수 있다**.**
x86 의 경우에는 동일한 파일 위치에서
this_cpu_read_stable() 함수에 의해 동작한다**.**

이 부분을 살펴보면 아래 파일

arch/x86/include/asm/percpu.h 에서

percpu_stable_op("mov", var) 매크로를 통해 관리됨을 볼 수 있다**.**
Intel 방식의 특유의 세그먼트 레지스터를 사용하여
관리하는 것을 볼 수 있는 부분이다**.**

20 문제 아래 물음에 답하시오. 내용
Kernel 의 Scheduling Mechanism 에서 Static Priority 와 Dynamic Priority 번호가 어떻게 되는지 적으시오.

Static Priority : 1 ~ 99
Dynamic Priority : 100 ~ 139

30 문제 다음 물음에 답하라. 내용
MMU(Memory Management Unit)의 주요 핵심 기능을 모두 적고 간략히 설명하시오.

가상 주소로부터 물리 주소로의 변환을 담당하는 별도의 하드웨어

주소 변환을 위해서는 페이지 디렉터리와 페이지 테이블을 탐색해야 하는데, 이것을 하드웨어 적으로 처리한다. 또한 TLB 같은 페이지 테이블 엔트리 캐시를 사용하여 빠른 주소변환을 지원한다.

1. HAT(HW Address Translation) 는
  가상 메모리 공간을 실제 물리 메모리 공간으로 변환한다**.**
2. TLB(Translation Lookaside Buffer) 는
  가상 메모리 주소와 대응되는 물리 메모리 주소를 **Caching** 한다**.**

31 문제 물음에 답하라. 내용
하드디스크의 최소 단위를 무엇이라 부르고 그 크기는 얼마인가 ?

Sector, 512 byte

32 문제 다음 물음에 답하라. 내용
Character 디바이스 드라이버를 작성할 때 반드시 Wrapping 해야 하는 부분이 어디인가 ?

(Task 구조체에서 부터 연결된 부분까지를 쭉 이어서 작성하라)

task_struct->files_struct->file->file_operations 에 해당함

39 문제 아래 물음에 답하시오. 내용
예로 모니터 3 개를 쓰는 경우 양쪽에 모두 인터럽트를 공유해야 한다.

Linux Kernel 에서는 어떠한 방법을 통해 이들을 공유하는가 ?

디바이스 드라이버의 주 번호는 major 그리고 나머지인 minor 을 이용한다.
외부 인터럽트의 경우 **32 ~ 255** 까지의 번호를 사용한다**.**
여기서 **128(0x80)**에 해당하는 **System Call** 만은 제외한다**.**
idt_table 에서 **128** 을 제외한 **32 ~ 255** 사이의 번호가 들어오면 실제 **HW Device** 다**.**
여기서 같은 종류의 **HW Device** 가 들어올 수 있는데
그들에 대해서 **Interrupt** 를 공유하게 하기 위해
irq_desc 라는 **Table** 을 추가로 두고 **active** 라는 것으로 **Interrupt** 를 공유하게끔한다**.**

40 문제 다음 물음에 답하시오. 내용
System Call 호출시 Kernel 에서 실제 System Call 을 처리하기 위해 Indexing 을 수행하여 적절한 함수가 호출되도록 주소값을 저장해놓고 있다.

이 구조체의 이름을 적으시오.

Intel 의 경우에 **sys_call_table**
ARM 의 경우에는 **__vectors_start + 0x1000** 에 해당함

41 문제 아래 물음에 답하시오. 내용
38 에서 User Space 에서 System Call 번호를 전달한다.

Intel Machine 에서는 이를 어디에 저장하는가 ?

또한 ARM Machine 에서는 이를 어디에 저장하는가 ?

Intel 의 경우에는 **ax** 레지스터에
ARM 의 경우에는 **r7** 레지스터에 해당한다**.**

43 문제 다음 물음에 답하라. 내용
또한 Page Directory 를 가르키는 Intel 전용 Register 가 존재한다.

이 Register 의 이름을 적고 ARM 에서 이 역할을 하는 레지스터의 이름을 적으시오.

Intel 의 경우엔 **CR3**
ARM 의 경우엔 **CP15**

51 문제 다음 물음에 답하라. 내용
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 내에 **files_struct** 내에 **file** 내에 **path** 내에 **dentry** 내에

inode 구조체에 존재하는 **i_rdev** 변수에 저장한다**.**

52 문제 다음 물음에 답하시오. 내용
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.

Device Driver 에서 **class_create, device_create** 를 이용해서
Character Device 인 **/dev/**장치파일을 만든다**.**
그리고 **Character Device Driver** 를 등록하기 위해서는
register_chrdev()가 동작해야 한다**.**

동적으로 할당할 경우에는 **alloc_chrdev_region()** 이 동작한다**.**
이때 **file_operations** 를 **Wrapping** 할 구조체를 전달하고
Major Number 와 장치명을 전달한다**.**

chrdevs 배열에 **Major Number** 에 해당하는 **Index** 에
file_operations 를 **Wrapping** 한 구조체를 저장해둔다**.**

그리고 이후에 실제 **User** 쪽에서 생성된 장치가 **open** 되면
그때는 **Major Number** 에 해당하는 장치의 **File Descriptor** 가 생성되었으므로
이 **fd_array** 즉**, file** 구조체에서 **i_mode** 를 보면
Character Device 임을 알 수 있고
i_rdev 를 보고 **Major Number** 와 **Minor Number** 를 파악할 수 있다**.**

그리고 **chrdevs** 에 등록한 **Major Number** 와 같음을 확인하고
이 **fd_array** 에 한해서는 **open, read, write, lseek, close** 등의
file_operations 구조체 내에 있는 함수 포인터들을
앞서 **Wrapping** 한 구조체로 대체한다**.**

그러면 **User** 에서 **read()**등을 호출할 경우 우리가 알고 있는 **read** 가 아닌
Device Driver 작성시 새로 만든 우리가 **Wrapping** 한 함수가 동작하게 된다**.**

54 문제 디바이스 드라이버를 작성하시오 - 10 점 내용
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!

***driver.c**

```
#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 <asm/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 int *write_ret;
static int *read_ret;
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 int mydrv_read(struct file *file, char *buf, size_t count, loff_t *ppos)
{
#if 0
    if((buf == NULL) || (count < 0))
        return -EINVAL;
    if(mydrv_write_offset - mydrv_read_offset <= 0)
        return 0;
    count = MIN((mydrv_write_offset - mydrv_write_offset), count);
    if(copy_to_user(buf, mydrv_data + mydrv_read_offset, count))
        return -EFAULT;
    mydrv_read_offset += count;
#endif
    int i;

    read_ret = (int *)kmalloc(sizeof(int), GFP_KERNEL);
    read_ret[0] = 1;

    for(i = 1; i <= count; i++)
```

```
        read_ret[0] *= i;

    return read_ret[0];
}

static ssize_t mydrv_write(struct file *file, const char *buf, size_t count, loff_t *ppos)
{
#if 0
    if((buf == NULL) || (count < 0))
        return -EINVAL;
    if(count + mydrv_write_offset >= MYDRV_MAX_LENGTH)
        return 0;
    if(copy_from_user(mydrv_data + mydrv_write_offset, buf, count))
        return -EFAULT;
    mydrv_write_offset += count;
#endif
    int i;

    write_ret = (int *)kmalloc(sizeof(int), GFP_KERNEL);
    write_ret[0] = 0;

    for(i = 1; i <= count; i++)
        write_ret[0] += i;

    return write_ret[0];
}

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);

*user.c
#include <stdio.h>
#include <fcntl.h>

#define MAX_BUFFER      26

char bin[MAX_BUFFER];
char bout[MAX_BUFFER];

int main(void)
{
```

```
    int fd, i, c = 65;
    int write_ret, read_ret;

    if((fd = open("/dev/mydevicefile", O_RDWR)) < 0)
    {
        perror("open()");
        return -1;
    }

    write_ret = write(fd, bout, 10);
    read_ret = read(fd, bin, 5);

    printf("write_ret = %d, read_ret = %d\n", write_ret, read_ret);

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

56 문제 아래 물음에 답하시오. 내용
Compiler 의 Instruction Scheduling 에 대해 기술하라.

위의 **OoO** 개념으로 **Compile-Time** 에 **Compiler Level** 에서 직접 수행한다**.**
Compiler 가 직접 **Compile-Time** 에 분석하므로
최적화의 수준이 높으면 높을수록 **Compile Timing** 이 느려질 것이다**.**
그러나 성능만큼은 확실하게 보장할 수 있을 것이다**.**

58 문제 아래 질문에 적절한 답을 기술하시오. 내용
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 ?

Dynamic Instruction Scheduling

60 문제 다음 물음에 답하시오. 내용
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
ISA(Instruction Set Architecture) : 명령어 집합

62 문제 아래 질문에 답하라 - 10 점 문제 내용
그동안 많은 것을 배웠을 것이다.

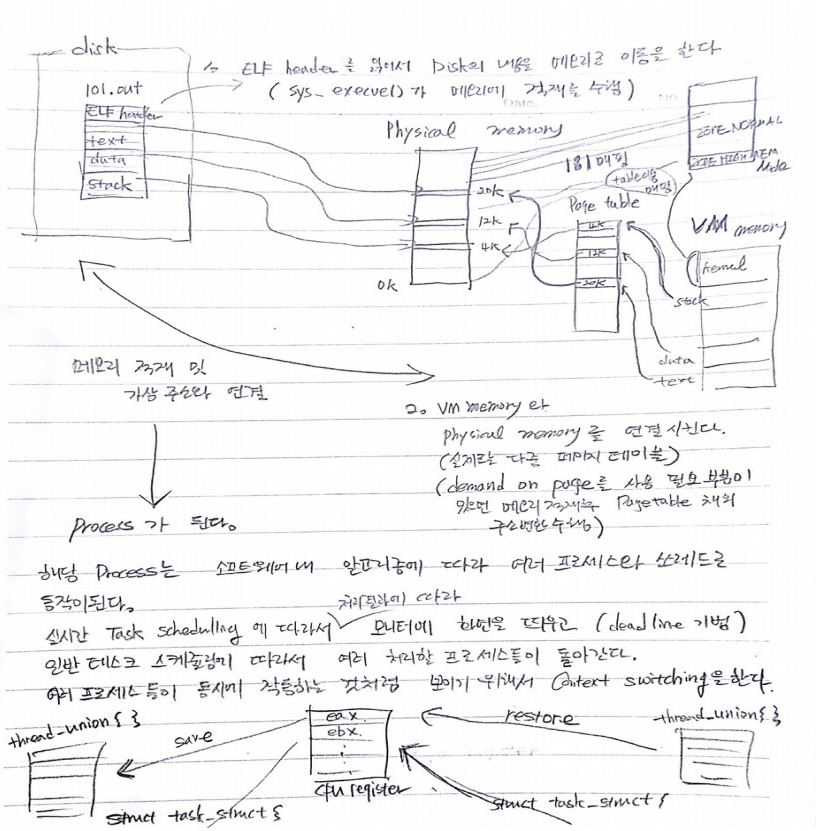
최종적으로 Kernel Map 을 그려보도록 한다. (Networking 부분은 생략해도 좋다)

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.



65 문제 다음을 프로그래밍 하시오. 내용
Shared Memory 를 통해 임의의 파일을 읽고 그 내용을 공유하도록 프로그래밍하시오.

***65_shm.h**

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

typedef struct
{
    char name[20];
    char buf[1024];
} SHM_t;

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

***65_recv.c**

```
#include "65_shm.h"

int main(void)
{
```

```
    int mid;
    SHM_t *p;

    mid = CreateSHM(0x888);

    p = GetPtrSHM(mid);

    getchar();
    printf("name : %s\nbuf : %s\n", p -> name, p -> buf);

    FreePtrSHM(p);

    return 0;
}

*65_send.c
#include "65_shm.h"
#include <fcntl.h>

int main(int argc, char **argv)
{
    int fd;
    int ret;
    int mid;
    char buf[1024];
    SHM_t *p;

    mid = OpenSHM(0x888);

    p = GetPtrSHM(mid);

    getchar();
```

```
    fd = open(argv[1], O_RDONLY);
    ret = read(fd, buf, sizeof(buf));
    buf[ret - 1] = 0;
    strcpy(p -> name, argv[1]);
    strcpy(p -> buf, buf);

    FreePtrSHM(p);

    return 0;
}

*65_shmlib.c

#include "65_shm.h"

int CreateSHM(long key)
{
    return shmget(key, sizeof(SHM_t), IPC_CREAT | 0777);
}

int OpenSHM(long key)
{
    return shmget(key, sizeof(SHM_t), 0);
}

SHM_t *GetPtrSHM(int shmid)
{
    return (SHM_t *)shmat(shmid, (char *)0, 0);
}

int FreePtrSHM(SHM_t *shmptr)
{
    return shmdt((char *)shmptr);
```

}

68 문제 아래 물음에 답하시오. 내용
현재 삽입된 디바이스 드라이버의 리스트를 보는 명령어는 무엇인가 ?

lsmod

69 문제 아래 물음에 답하시오. 내용
System Call Mechanism 에 대해 기술하시오.

System Call 은 **User** 가 **Kernel** 에 요청하여 작업을 처리할 수 있도록 지원한다**.**
내부적으로 굉장히 복잡한 과정을 거치지만 **User** 가 **read()**등만 호출하면
실질적인 복잡한 모든 과정은 **Kernel** 이 수행하게 되어 편의성을 제공해준다**.**

70 문제 다음 물음에 답하시오. 내용
Process 와 VM 과의 관계에 대해 기술하시오.

Process 는 자신의 고유한 공간으로 가상의 **4GB** 를 가지고 있다**.**
실제 이 공간을 모두 사용하지 않으며 **Demand Paging** 에 따라 필요한 경우에만
실제 물리 메모리 공간을 **Paging Mechanism** 을 통해 할당받아 사용한다**.**

71 문제 다음을 프로그래밍 하시오. 내용
인자로 파일을 입력 받아 해당 파일의 앞 부분 5 줄을 출력하고 추가적으로 뒷 부분의 5 줄을 출력하는 프로그램을 작성하시오.

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

int save_area[1024];

int main(int argc, char **argv)
{
   int i;
   int ret;
   int index;
```

```
  int fd;
  char buf[1024];
  fd = open(argv[1], O_RDONLY);
  ret = read(fd, buf, sizeof(buf));
  for(i = 0, index = 1; buf[i]; i++)
  {
    if(buf[i] == '\n')
    {
      save_area[index] = i;
      index++;
    }
  }
  printf("Front 5 Lines\n");
  write(0, buf, save_area[5] + 1);
  printf("Back 5 Lines\n");
  printf("%s", &buf[save_area[index - 6] + 1]);
  return 0;
}
```

73 문제 아래 물음에 답하시오. 내용
Linux 에서 fork()를 수행하면 Process 를 생성한다.

이때 부모 프로세스를 gdb 에서 디버깅하고자하면 어떤 명령어를 입력해야 하는가 ?

set follow-fork-mode child

자식을 디버깅 하고자 하는 경우에는 **set follow-fork-mode child** 지만
부모의 경우는 디폴트 설정 값이므로 특별한 명령어를 줄 필요가 없다**.**

79 문제 다음을 프로그래밍 하시오. 내용
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.

2 중 반복문 이상일 경우 어떤 특수한 값을 찾을 경우에 바로 반복문을 탈출 시키면 나머지 연산을 안하게 된다. 따라서 성능상의 이득을 본다.

간단한 예로 아래와 같이 프로그래밍 해보았다. - 문제를 잘못 읽음.

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

jmp_buf env;

void error_label(int flag)
{
   if(flag == 1)
      longjmp(env, 1);
}

int main(void)
{
   int ret;
   int error_flag = 1;
   if(ret = setjmp(env) == 0)
      error_label(error_flag);
   else
      printf("Error\n");
   return 0;
}
```

81 문제 다음 물음에 답하시오. 내용

stat(argv[2], &buf)일때 stat System Call 을 통해 채운 buf.st_mode 의 값에 대해 기술하시오.

buf.st_mode 에는 리눅스 커널 **inode** 의 **i_mode** 와 같은 값이 들어가 있다**.**
파일의 종류 **4** 비트와 **setuid, setgid, sticky bit,** 그리고 **rwx** 가 **3** 개씩 존재한다**.**

82 문제 다음 물음에 답하시오. 내용
프로세스들은 최소 메모리를 관리하기 위한 mm, 파일 디스크립터인 fd_array, 그리고 signal 을 포함하고 있는데 그 이유에 대해 기술하시오.

자신이 실제 메모리 어디에 위치하는지에 대한 정보가 필요하고
또 자신이 하드 디스크의 어떤 파일이 메모리에 로드되어
프로세스가 되었는지의 정보가 필요하며
마지막으로 프로세스들 간에
signal 을 주고 받을 필요가 있기 때문에 **signal** 이 필요하다**.**

83 문제 다음을 프로그래밍 하시오. 내용
디렉토리를 만드는 명령어는 mkdir 명령어다.

man -s2 mkdir 을 활용하여 mkdir System Call 을 볼 수 있다.

이를 참고하여 디렉토리를 만드는 프로그램을 작성해보자!

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

int main(int argc, char *argv[])
{
    if(argc != 2)
    {
        printf("Usage: exe dir_name\n");
        exit(-1);
    }
```

```
    mkdir(argv[1], 0755);

    return 0;
}
```

84 문제 다음을 프로그래밍 하시오. 내용
이번에는 랜덤한 이름(길이도 랜덤)을 가지도록 디렉토리를 3 개 만들어보자!

(너무 길면 힘드니까 적당한 크기로 잡도록함)

```
#include <sys/stat.h>
#include <sys/types.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <time.h>

char *rand_name(void)
{
    int i, len;
    static char buf[8] = "\0";

    memset(buf, 0, sizeof(buf));
    len = rand() % 4 + 4;

    for(i = 0; i < len; i++)
        buf[i] = rand() % 26 + 97;

    printf("buf = %s\n", buf);

    return buf;
}
```

```
void make_rand_dname(char **dname)
{
    int i;

    for(i = 0; i < 3; i++)
    {
        dname[i] = (char *)malloc(sizeof(char) * 8);
        strcpy(dname[i], rand_name());
        //dname[i] = rand_name();
    }
}

int main(void)
{
    int i;
    char *dname[3] = {0};

    srand(time(NULL));
    make_rand_dname(dname);

    for(i = 0; i < 3; i++)
    {
        printf("dname[%d] = %s\n", i, dname[i]);
        mkdir(dname[i], 0755);
    }

    return 0;
}
```

85 문제 다음을 프로그래밍 하시오. 내용
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)

```
#include <sys/stat.h>
#include <sys/types.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <stdio.h>
#include <fcntl.h>
#include <time.h>

char *rand_name(void)
{
    int i, len;
    static char buf[8] = "\0";

    memset(buf, 0, sizeof(buf));
    len = rand() % 4 + 4;

    for(i = 0; i < len; i++)
        buf[i] = rand() % 26 + 97;

    printf("buf = %s\n", buf);

    return buf;
}

void make_rand_dname(char **dname)
{
    int i;
```

```
    for(i = 0; i < 3; i++)
    {
        dname[i] = (char *)malloc(sizeof(char) * 8);
        strcpy(dname[i], rand_name());
    }
}

void make_rand_file(void)
{
    int i, j, fd, len, cnt;
    char buf[11] = "\0";

    len = rand() % 6 + 5;
    cnt = rand() % 4 + 2;

    for(i = 0; i < cnt; i++)
    {
        for(j = 0; j < len; j++)
            buf[j] = rand() % 26 + 97;

        fd = open(buf, O_CREAT, 0644);
        close(fd);

        memset(buf, 0, sizeof(buf));
    }
}

void lookup_dname(char **dname)
{
    int i;
```

```
    for(i = 0; i < 3; i++)
    {
        chdir(dname[i]);
        make_rand_file();
        chdir("..");
    }
}

int main(void)
{
    int i;
    char *dname[3] = {0};

    srand(time(NULL));
    make_rand_dname(dname);

    for(i = 0; i < 3; i++)
        mkdir(dname[i], 0755);

    lookup_dname(dname);

    return 0;
}
```

86 문제 다음을 프로그래밍 하시오. 내용
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.

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

char *rand_name(void)
{
    int i, len;
    static char buf[8] = "\0";

    memset(buf, 0, sizeof(buf));
    len = rand() % 4 + 4;

    for(i = 0; i < len; i++)
        buf[i] = rand() % 26 + 97;

    printf("buf = %s\n", buf);

    return buf;
}

void make_rand_dname(char **dname)
{
    int i;

    for(i = 0; i < 3; i++)
    {
```

```
        dname[i] = (char *)malloc(sizeof(char) * 8);
        strcpy(dname[i], rand_name());
    }
}

void make_rand_file(void)
{
    int i, j, fd, len, cnt;
    char buf[11] = "\0";

    len = rand() % 6 + 5;
    cnt = rand() % 4 + 2;

    for(i = 0; i < cnt; i++)
    {
        for(j = 0; j < len; j++)
            buf[j] = rand() % 26 + 97;

        fd = open(buf, O_CREAT, 0644);
        close(fd);

        memset(buf, 0, sizeof(buf));
    }
}

void lookup_dname(char **dname)
{
    int i;

    for(i = 0; i < 3; i++)
    {
```

```
            chdir(dname[i]);
            make_rand_file();
            chdir("..");
        }
}

void find_abc(void)
{
    struct dirent *p;
    struct stat buf;
    DIR *dp;
    int i, len;

    dp = opendir(".");

    while(p = readdir(dp))
    {
        stat(p->d_name, &buf);
        len = strlen(p->d_name);

        for(i = 0; i < len; i++)
        {
            if(!strncmp(&p->d_name[i], "a", 1) ||
              !strncmp(&p->d_name[i], "b", 1) ||
              !strncmp(&p->d_name[i], "c", 1))
            {
                printf("name = %s\n", p->d_name);
            }
        }
    }
```

```
    closedir(dp);
}

void recur_find(char **dn)
{
    int i;

    for(i = 0; i < 3; i++)
    {
        chdir(dn[i]);
        find_abc();
        chdir("..");
    }
}

int main(void)
{
    int i;
    char *dname[3] = {0};

    srand(time(NULL));
    make_rand_dname(dname);

    for(i = 0; i < 3; i++)
        mkdir(dname[i], 0755);

    lookup_dname(dname);
    recur_find(dname);

    return 0;
}
```

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

우리가 만든 프로그램에서 **fork()** 가 호출되면
C Library 에 해당하는 **glibc** 의 **__libc_fork()** 가 호출됨
이 안에서 **ax** 레지스터에 시스템 콜 번호가 기록된다.
즉 **sys_fork()** 에 해당하는 시스템 콜 테이블의 번호가 들어가고
이후에 **int 0x80** 을 통해서 **128** 번 시스템 콜을 후출하게 된다.
그러면 제어권이 커널로 넘어가서 **idt_table(Interrupt Descriptor Table)**로 가고
여기서 시스템 콜은 **128** 번으로 **sys_call_table** 로 가서
ax 레지스터에 들어간 **sys_call_table[**번호**]** 의 위치에 있는
함수 포인터를 동작시키면 **sys_fork()** 가 구동이 된다.
sys_fork() 는 **SYSCALL_DEFINE0(fork)** 와 같이 구성되어 있다.

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

ARM 은 하위 호환이 안되고 다양한 반도체 벤더들이 개발을 하고 있기 때문에
해당 디렉토리에 들어가면 회사별 주요 제품들의 이름이 보이는 것을 확인할 수 있다.

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

일반적인 **HW** 인터럽트는 어셈블리 루틴 **common_interrupt** 레이블에서 처리한다.
이 안에서는 **do_IRQ()** 라는 함수가 같이 함께
일반적인 **HW** 인터럽트를 처리하기 위해 분발한다.

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

r7