운영체제 및 실습

-2차 과제-

학번: 201404377

이름: 진승언

#시스템 동작과정 (ppt참고)

시스템 콜이란 운영체제의 커널이 제공하는 서비스에 대해, 응용 프로그램의 요청에 따라 커널에 접근하기 위한 인터페이스이다.

먼저 ppt를 참고해서 간단하게 동작과정을 봤다.

```
#define __NR_proc_list 548

int proc_list(void)
{
        return syscall(__NR_proc_list);
}

void main()
{
        int n = proc_list();
        printf("proc_list() : return %d\n",n);
}
```

(1) 위 사진은 시스템 콜을 요청한다.(IDT를 통해 시스템 콜 테이블로 이동됨)(548번임을 암)

```
547     x32     pwritev2                compat_sys_pwritev64v2
548     x32     proc_list               sys_proc_list
```

(2) 시스템 콜 테이블에서 548번에 해당하는 엔트리를 찾아간다. 그리고 해당 시스템 콜을 호출한다. 여기서는 sys_proc_list 를 호출하는 것이다. (시스템 콜 테이블은 시스템 콜 번호, 이름, 함수이름으로 등록되어있다).

```
u201800000@u201800000: ~/linux-4.15.14/include/linux
u201800000@u201800000:~/linux-4.15.14/include/linux$ vi syscalls.h
```

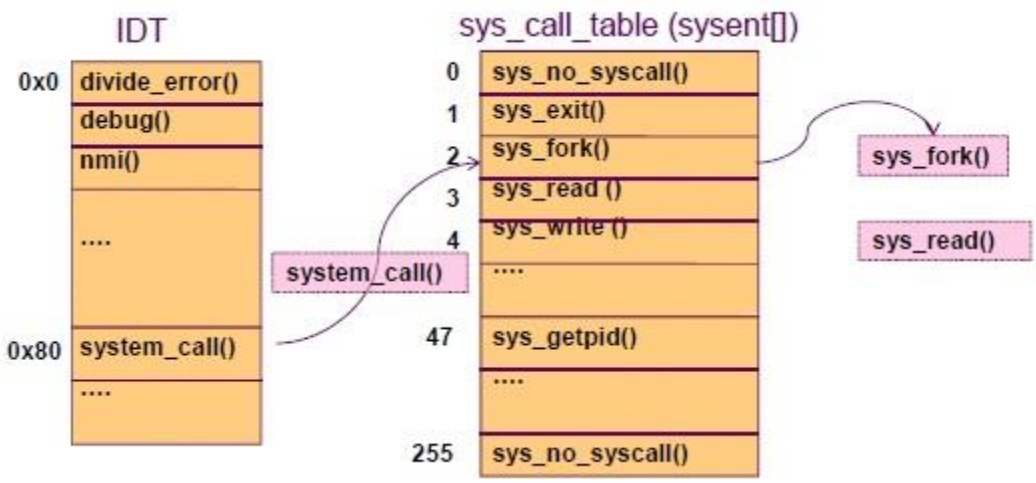
```
asmlinkage long sys_pkey_alloc(unsigned long flags, unsigned long init_val);
asmlinkage long sys_pkey_free(int pkey);
asmlinkage long sys_statx(int dfd, const char __user *path, unsigned flags,
                          unsigned mask, struct statx __user *buffer);
asmlinkage int sys_proc_list(void);
#endif
                                                              944,6         Bot
```

```
u201800000@u201800000: ~/linux-4.15.14/proc_list
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/sched.h>
#include <linux/syscalls.h>

asmlinkage int sys_proc_list(void)
{
        int n = 0;
        struct task_struct *p;
        printk("\tProcess\tpid\tstate\n");
        for_each_process(p)
        {
                printk("%15s\t%u\t%ld\n", p->comm, task_pid_nr(p), p->state);
                n++;
        }

        return n;
}
```

(3) 시스템 콜 테이블을 통해서 해당 시스템 콜 서비스 루틴이 실행된다.

#시스템 콜 동작과정(추가자료파일 및 인터넷 참고)

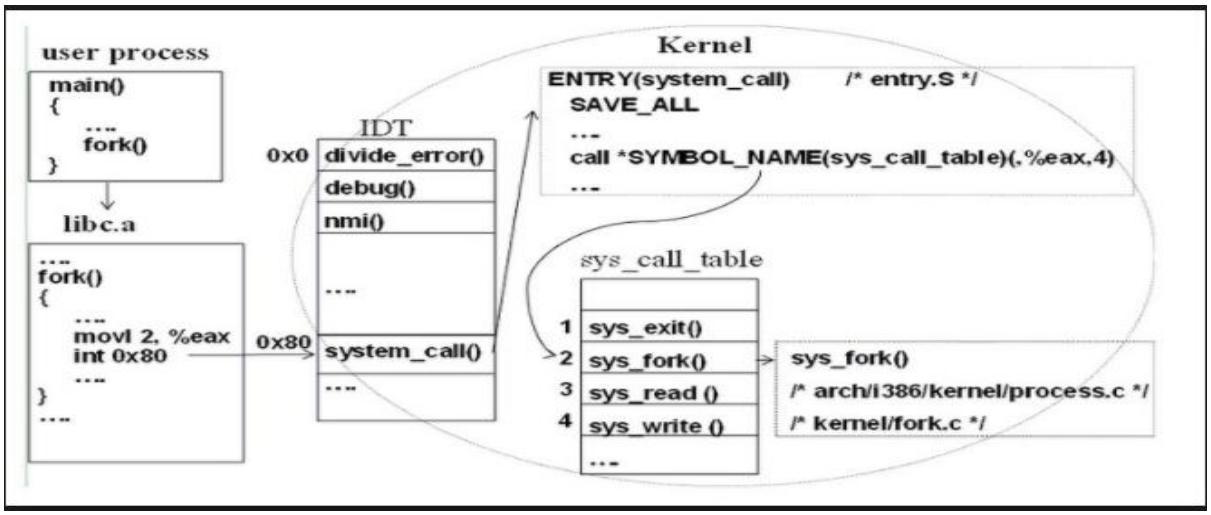
좀 더 자세히 설명하자면, 앞에서 말한 것처럼 시스템 콜은 사용자 공간(user space)의 응용 프로그램과 커널이 제공하는 서비스 사이의 인터페이스다. 즉, 응용 프로그램이(프로세스) 커널의 기능을 이용하기 위해 운영 체제에게 서비스를 요청하는 것을 말한다. 응용 프로그램은 사용자 공간에서 실행되기 때문에, CPU의 권한 수준과 하드웨어 접근이 제한되어 있다. 따라서 응용 프로그램이 하드웨어 제어 또는 프로세스 제어 등의 작업을 수행하기 위해서는 시스템 콜을 통해 커널 서비스를 제공받아야 한다.



구체적으로 시스템 호출이 서비스 되는 과정은 다음과 같다. 우선 사용자 수준 응용이 fork()라는 시스템 호출을 요청했다고 가정하자. 그럼 /usr/lib/libc.a라는 이름의 표준 C 라이브러리에 구현되어 있는 라이브러리 함수가 호출된다. 이 함수는 사용자 대신 트랩을 요청하는 일종의 대리인(agent)이라고 볼 수 있다. 이 라이브러리 함수는 시스템 호출 번호를 저장하고 트랩을 건다. 일단 트랩이 걸리면 제어가 커널로 넘겨지고 수행 모드가 커널 모드로 변화된다. 그리고 IDT 테이블과 sys_call_table, 그리고 넘겨진 인자들을 이용해 커널에서 구현된 시스템 호출 핸들러(system call handler)를 호출한다. System call handler 함수는 시스템 호출 번호에 해당하는 sys_call_table 엔트리를 찾아간다. 그 엔트리에는 시스템 호출을 처리하는 실제 함수인 system call service routine 의 주소가 들어가 있어서 결국 그 routine을 호출한다.

#기존 리눅스 커널 내부 시스템 콜 예시

리눅스에서의 시스템 콜의 예로 sys_fork(), sys_exit(), sys_read(), sys_write()등이 있지만 그 중에서 sys_fork()로 예시로 **시스템 호출과정을 자세히 살펴보았다**.



사용자 수준 응용이 fork()라는 시스템 호출을 요청했다고 가정하자. 그럼 /usr/lib/libc.a라는 이름의 표준 C 라이브러리에 구현되어 있는 fork()라는 이름의 라이브러리 함수가 호출된다. 이 라이브러리 함수는 CPU내에 있는 범용 레지스터 중에 하나인 eax 레지스터에 2라는 값을 저장하고, 0x80을 인자로 트랩을 건다 (다르게 표현하면 트랩의 소스가 0x80이 된다). 인텔 프로세서에서 트랩을 거는 명령이 "int"이다. 트랩이 걸리면 제어가 커널로 넘겨지고 수행 모드가 커널 모드로 변화된다. 그리고 커널은 현재 실행 중이던 태스크의 문맥을 stack에 저장하고 트랩의 소스에 대응되는 엔트리에 등록되어 있는 함수를 호출한다. 지금 이 설명에서 트랩 소스는 0x80이며 따라서 호출되는 함수는 system_call()이다. 이 함수는 arch/i386/kernel/entry.S 파일에 구현되어 있다. 이 함수는 eax의 값을 인덱스로 sys_call_table을 탐색한다. eax 레지스터에는 어떤 값 이 들어 있었는가? 2이다. 따라서 sys_call_table의 2라는 인덱스로 접근할 수 있는 엔트리에 등록되어 있는 함 수인 sys_fork()가 호출되게 되는 것이다. 결국 사용자 수준 응용은 fork()라는 시스템 호출을 요청했으며, IDT 테이블과 sys_call_table을 이용해 커널에서 구현된 sys_fork()함수가 호출되는 것이다