시스템 콜 함수 구현 및 실행

2021년 10월

김상현

차 례

1장　프로젝트 동기/목적

시스템 콜 함수를 직접 구현해보고 커널 컴파일을 통해 실제 사용가능 할 수 있도록 알아본다. 특히 시스템 콜 테이블 등과의 관계도 파악한다.
설계 도중 발생하는 오류등을 파악하고 해결해본다.

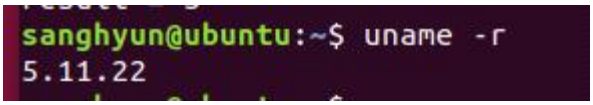
2장　설계/구현

구현 순서에 따라 보고서를 구성해보았다.

1. 커널 컴파일 (5.11.22 q버젼으로 초기화);

해당 내용은 캡처하지 않아 설명으로 대체한다.

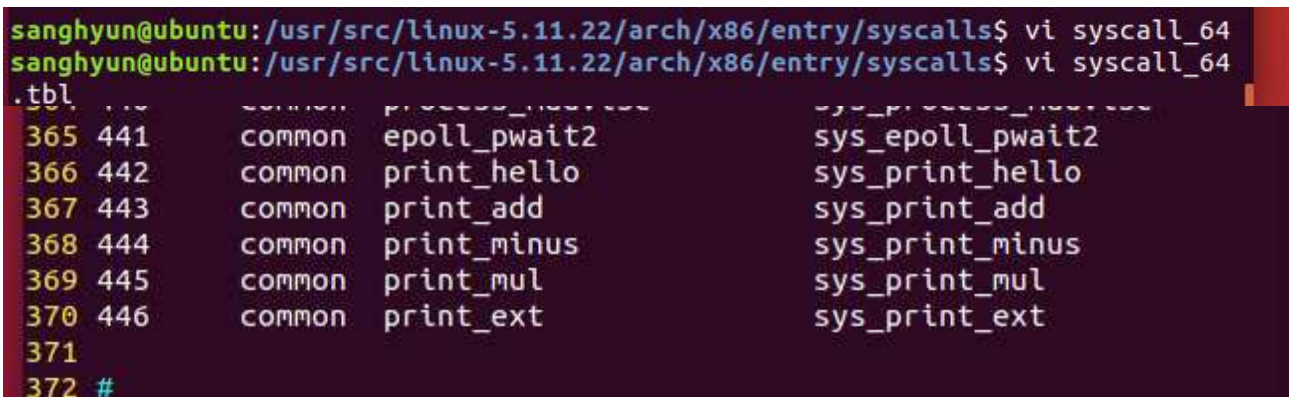
- wget https://www.kernel.org/pub/linux/kernel/v5.x/linux-5.13.12.tar.xz 을 통해 리눅스 커널 파일을 다운 받을 수 있다.
- 해당 파일을 압축 해제하고 리눅스 컴파일을 진행한다.



리눅스 컴파일을 성공적으로 수행한 뒤의 화면이다.

2. 추가하려는 시스템 콜을 시스템 콜 테이블 등록 ;

새롭게 추가하려는 시스템 콜을 모두 콜 테이블에 모두 등록해 주어야 한다. 이후 나오겠지만 새로운 함수 이름은 (sys_print_add / sys_print_minus / sys_print_mul / sys_print_ext) 이다.



총 네가지를 각각 443, 444, 445, 446 번에 넣었고 이는 이후 호출하기 위해 쓰이는 번호가 될 것이다.
● <num> <abi> <name> <entry point> 순으로 구성되어있다.

3. 시스템 콜 헤더파일에 등록

```
sanghyun@ubuntu:/usr/src/linux-5.11.22$ cd include/linux
sanghyun@ubuntu:/usr/src/linux-5.11.22/include/linux$ vi syscalls.h
1367 asmlinkage long sys_print_hello(void);
1368 asmlinkage int sys_print_add(int ,int);
1369 asmlinkage int sys_print_minus(int , int);
1370 asmlinkage int sys_print_mul(int ,int);
1371 asmlinkage int sys_print_ext(int,int);
                                                    1371,1        99%
```

헤더 파일에 등록할 때 앞에 쓰이는 asmlinkage 는 어셈블리 코드에서도 c 함수 호출을 위해 쓰였다.

4. 시스템 콜 함수 구현

```
sanghyun@ubuntu:/usr/src/linux-5.11.22$ cd kernel
```

```
 1 #include <linux/kernel.h>
 2 #include <linux/syscalls.h>
 3 #include <linux/unistd.h>
 4 #include <linux/errno.h>
 5 #include <linux/sched.h>
 6
 7 asmlinkage int sys_print_add(int a,int b){
 8         return a+b;
 9 }
10 SYSCALL_DEFINE2(print_add, int, a, int, b){
11         return sys_print_add(a,b);
12 }
```

덧셈에 해당하여 구현한 sys_print_add 함수 이다.

아래 SYSCALL_DEFINE2() 함수는 매크로 함수로써 인자가 2개인 것을 선언 할 때 사용한다. 사용 방법은 SYSCALL_DEFINE2(함수 이름, 자료형, 변수1, 자로형 , 변수2) 식으로 사용한다.

이때 당연하게도 인자가 없는 경우는 SYSCALL_DEFINE0() , 한 개라면 SYSCALL_DEFINE1() 등 으로 응용하여 사용할 수 있다.

이후 -, *, % 의 경우 수식만 달라지기 때문에 아래 사진으로 첨부한다.

```
 1 #include <linux/kernel.h>
 2 #include <linux/syscalls.h>
 3 #include <linux/unistd.h>
 4 #include <linux/errno.h>
 5 #include <linux/sched.h>
 6
 7 asmlinkage int sys_print_minus(int a,int b){
 8         return a-b;
 9 }
10 SYSCALL_DEFINE2(print_minus, int, a, int, b){
11         return sys_print_minus(a,b);
12 }
13
```

sys_print_minus.c

```
 1 #include <linux/kernel.h>
 2 #include <linux/syscalls.h>
 3 #include <linux/unistd.h>
 4 #include <linux/errno.h>
 5 #include <linux/sched.h>
 6
 7 asmlinkage int sys_print_mul(int a,int b){
 8         return a*b;
 9 }
10 SYSCALL_DEFINE2(print_mul, int, a, int, b){
11         return sys_print_mul(a,b);
12 }
13
```

sys_print_mul.c

```c
#include <linux/kernel.h>
#include <linux/syscalls.h>
#include <linux/unistd.h>
#include <linux/errno.h>
#include <linux/sched.h>

asmlinkage int sys_print_ext(int a,int b){
        return a%b;
}
SYSCALL_DEFINE2(print_ext, int, a, int, b){
        return sys_print_ext(a,b);
}

```

sys_print_ext.c

(여기서 ext 는 extra 의 준말이다. 개인적으로 영단어의 부족함을 다시 느낀다.)

5. Makefile 에 등록

```
# SPDX-License-Identifier: GPL-2.0
#
# Makefile for the linux kernel.
#

obj-y     = fork.o exec_domain.o panic.o \
            cpu.o exit.o softirq.o resource.o \
            sysctl.o capability.o ptrace.o user.o \
            signal.o sys.o umh.o workqueue.o pid.o task_work.o \
            extable.o params.o \
            kthread.o sys_ni.o nsproxy.o \
            notifier.o ksysfs.o cred.o reboot.o \
            async.o range.o smpboot.o ucount.o regset.o sys_print_hello.o sys_print_add.o sys_print_minus.o sys_print_mul.o sys_print_ext.o

```

추가한 시스템 콜이 다른 시스템 콜과 함께 컴파일 될 수 있도록 object 파일을 등록시켜놓는다.

6. 커널 컴파일.

```
sanghyun@ubuntu:/usr/src/linux-5.11.22$ make-kpkg -j8 --initrd --revision=2.0 kernel_image
```

7. 시스템 콜 호출 (간단한 사칙연산 계산기)

가장 중요한 것은 표준 입력을 통한 수식 입력과 표준 출력을 통한 연산 결과 확인이다.

```c
#include<stdio.h>
#include <linux/kernel.h>
#include <sys/syscall.h>
#include <stdlib.h>
#include <unistd.h>

int main(){
        int a,b;
        char func;
        int result =0;
        printf("input func (+,-,*,%%) : ");
        func = getchar();
        printf("input a ,b : ");
        scanf("%d %d",&a,&b);
        if(func =='+')
                result = syscall(443,a,b);
        else if(func =='-')
                result = syscall(444,a,b);
        else if(func == '*')
                result = syscall(445,a,b);
        else if(func == '%')
                result = syscall(446,a,b);
        else{
                printf("no such func\n");
                return 1;
        }
        printf("result = %d\n",result);
        return 0;
}
```

syscall() 함수에서 번호로 해당 시스템 콜을 호출하고 뒤에 인자를 주어 해당 내용을 처리한다.

3장 수행결과

```
sanghyun@ubuntu:~$ ./a.out
input func (+,-,*,%) : +
input a ,b : 15 4
result = 19
sanghyun@ubuntu:~$ ./a.out
input func (+,-,*,%) : -
input a ,b : 15 4
result = 11
sanghyun@ubuntu:~$ ./a.out
input func (+,-,*,%) : *
input a ,b : 15 4
result = 60
sanghyun@ubuntu:~$ ./a.out
input func (+,-,*,%) : %
input a ,b : 15 4
result = 3
```

4장 결론 및 보충 할 점

본인은 같은 에러로 커널 컴파일이 되지 않아 많은 애를 먹었다.

```
  AR      drivers/firmware/built-in.a
  AR      drivers/built-in.a
make[1]: Leaving directory '/usr/src/linux/linux-5.11.22'
debian/ruleset/targets/common.mk:295: recipe for target 'debian/stamp/build/kernel' failed
make: *** [debian/stamp/build/kernel] Error 2
root@ubuntu:/usr/src/linux/linux-5.11.22#
```

위와 같은 에러의 해결 방법으로는 과제1 참고 사항에서 주어진 것처럼 .config 파일을 수정하는 방법이지만 이후에 컴파일 할 소스코드 등이 잘 못 되었을 때도 같은 내용이 출력된다.
(본인은 그걸 모르고 그냥 마냥 컴파일 문제라고 생각하여 하루를 허비하였다.)

또 현재 과제에서 중요한 부분이라고 생각한 점은 시스템 콜 테이블에 등록되는 <name> 과 <entry point> 의 차이이다.

<name> 은 말그대로 함수 이름 일 뿐 파일 이름이 아니다. 그렇기 때문에 /kernel/sys_print_add.c를 보면 직접적인 파일 이름과 함수를 연결 시켜주는 매크로 함수 SYSCALL_DEFINE2() 가 쓰인 것이다.