4가지 연산기능 하는 시스템콜 추가

20172609
김시온

1. 서론

- 4 가지 연산
기능을 (덧셈(+), 뺄셈(-), 곱셈(*), 나머지(%)) 제공하는 신규 시스템 콜을 각각 추가하고,
추가된 시스템 콜을 호출하는 테스트 프로그램 구현

- 4 가지
이항연산을 (덧셈(+), 뺄셈(-), 곱셈(*), 나머지(%)) 제공하는 새로운 시스템 콜 함수 4 개를 커널에 등록하기

- 테스트 프로그램을 (4 가지 연산만을 수행하는 이항 계산기 프로그램) 작성하여
새롭게 등록된 시스템 콜 호출 확인하기

2. 본론

커널버전 5.11.22 를 설치하였다.

/usr/src/linux/linux-5.11.22/arch/x86/entry/syscalls 디렉터리에서 syscall_64.tbl 파일을 편집하였다.

시스템콜 테이블을 등록하였다..

```
442     common  add_sion                sys_add_sion
443     common  minus_sion              sys_minus_sion
444     common  multiply_sion           sys_multiply_sion
445     common  mod_sion                sys_mod_sion
#
```

/usr/src/linux/linux-5.11.22/include/linux 디렉터리에서 syscalls.h

어셈블리 코드에서 직접 호출 할 수있도록 네가지 함수의 프로토타입을 정의하였다.

```
int __sys_setsockopt(int fd, int level, int optname, char __user *optval,
                int optlen);
asmlinkage int sys_add_sion(int a, int b,int* c);
asmlinkage int sys_minus_sion(int a, int b,int* c);
asmlinkage int sys_multiply_sion(int a, int b,int* c);
asmlinkage int sys_mod_sion(int a, int b,int* c);

#endif
```

a, b 는 피연산자,

c 는 결과값을 저장하는 int 형 포인터변수

그 후

/usr/src/linux/linux-5.11.22/kernel 디렉터리로 이동 후 시스템 콜의 구현 파일을 편집하였다.

Kernel 메모리에서 User 메모리로 단순 변수값을 보내기 위해 put_user(입력할 값,입력할주소) 함수를 사용했다.

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

asmlinkage int sys_add_sion(int a, int b,int* c){
        int tmp;
        tmp = a+ b;
        put_user(tmp,c);
        return 0;
}

SYSCALL_DEFINE3(add_sion,int,a,int,b,int*,c)
{
        return sys_add_sion(a,b,c);
}
```

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

asmlinkage int sys_minus_sion(int a, int b,int* c){
        int tmp;
        tmp = a-b;
        put_user(tmp,c);
        return 0;

}

SYSCALL_DEFINE3(minus_sion,int,a,int,b,int*,c)
{
        return sys_minus_sion(a,b,c);
}
```

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

asmlinkage int sys_multiply_sion(int a, int b,int* c){
        int tmp;
        tmp = a*b;
        put_user(tmp,c);
        return 0;

}

SYSCALL_DEFINE3(multiply_sion,int,a,int,b,int*,c)
{
        return sys_multiply_sion(a,b,c);
}
```

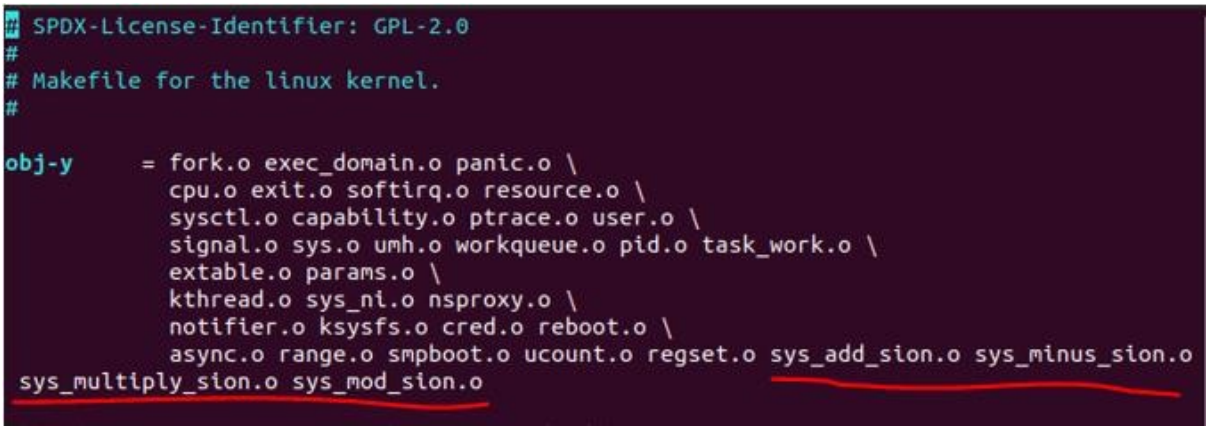
```c
#include <linux/kernel.h>
#include <linux/syscalls.h>

asmlinkage int sys_mod_sion(int a, int b,int *c){
        int tmp;
        tmp = a%b;
        put_user(tmp,c);
        return 0;

}

SYSCALL_DEFINE3(mod_sion,int,a,int,b,int*,c)
{
        return sys_mod_sion(a,b,c);
}
```

추가한 시스템 콜이 다른 시스템 콜과 함께
컴파일 될 수 있도록 Makefile을 편집하였다.



```
# 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_add_sion.o sys_minus_sion.o
 sys_multiply_sion.o sys_mod_sion.o
```

그 후
make-kpkg -revision=2.0 kernel_image

명령어를 통해 새로 컴파일 하고,

/usr/src/linux 폴더에서

dpkg -I linux-image-5.11.22_2.0_amd64.deb

명령어를 입력하여 새로운 커널 이미지로 부팅하였다.

테스트

test.c

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

int main()
{
        int N1,N2;
        char operator;
        int result=0;
        printf(">> ");
        scanf("%d%c%d",&N1,&operator,&N2);
        switch(operator)
        {
                case '+' : syscall(442,N1,N2,&result); break;
                case '-' : syscall(443,N1,N2,&result); break;
                case '*' : syscall(444,N1,N2,&result); break;
                case '%' : syscall(445,N1,N2,&result); break;
                default : printf("wrong operator input. use +-
*%% only.\nDo not use blank in operation\n");
                          return 0;
        }
        printf("result : %d\n",result);
        return 0;
}
```

3. 덧셈

```
sion@ubuntu:~$ ./a.out
>> 10+20
result : 30
```

```
sion@ubuntu:~$ ./a.out
>> -50+50
result : 0
```

```
sion@ubuntu:~$ ./a.out
>> -100+20
result : -80
```

```
sion@ubuntu:~$ ./a.out
>> 2147483645+2
result : 2147483647
```

```
sion@ubuntu:~$ ./a.out
>> -2146483649+2
result : -2146483647
```

4. 뺄셈

```
sion@ubuntu:~$ ./a.out
>> 3000-4000
result : -1000
```

```
sion@ubuntu:~$ ./a.out
>> -10-20
result : -30
```

```
sion@ubuntu:~$ ./a.out
>> 5000-1234
result : 3766
```

```
sion@ubuntu:~$ ./a.out
>> -12315--1231156
result : 1218841
```

5. 곱셈

```
sion@ubuntu:~$ ./a.out
>> 19*19
result : 361
```

```
sion@ubuntu:~$ ./a.out
>> 123*123
result : 15129
```

```
sion@ubuntu:~$ ./a.out
>> -100*200
result : -20000
```

```
sion@ubuntu:~$ ./a.out
>> -100*-100
result : 10000
```

6. 나머지 연산

```
sion@ubuntu:~$ ./a.out
>> 30%7
result : 2
```

```
sion@ubuntu:~$ ./a.out
>> 100%10
result : 0
```

```
sion@ubuntu:~$ ./a.out
>> -30%7
result : -2
```

```
sion@ubuntu:~$ ./a.out
>> 30%-7
result : 2
```

C = a% b 에서 C의 부호는 a의 부호이다.

5)연산부호가 잘못되었을 경우.

```
sion@ubuntu:~$ ./a.out
>> 10^30
wrong operator input. use +-*% only.
Do not use blank in operation
```

7. 결론

+ - * % 연산을 지원하는 시스템 콜 함수를 만들었다.

과제를 수행하며, 시스템 콜 함수가 어떻게 작동되는지 확인할 수 있었다.

더 알아볼 점:

인자를 2개를 받아서 간단히 계산하는 함수를 만들었었으나, 출력값이 -1 ~ -4095 일 경우 -1로만 출력되었다. 이에 대한 원인을 찾아봐야겠다.