

## 1. 시스템 콜 테이블 등록

```
root@ubuntu: /usr/src/linux/linux-5.11.22/arch/x86/entry/sy...
432    common    fsmount          sys_fsmount
433    common    fspick          sys_fspick
434    common    pidfd_open      sys_pidfd_open
435    common    clone3          sys_clone3
436    common    close_range     sys_close_range
437    common    openat2         sys_openat2
438    common    pidfd_getfd     sys_pidfd_getfd
439    common    faccessat2     sys_faccessat2
440    common    process_madvise sys_process_madvise
441    common    epoll_pwait2   sys_epoll_pwait2
442    common    print_add      sys_print_add
443    common    print_min      sys_print_min
444    common    print_mul      sys_print_mul
445    common    print_mod      sys_print_mod
#
# Due to a historical design error, certain syscalls are numbered differently
# in x32 as compared to native x86_64. These syscalls have numbers 512-547.
# Do not add new syscalls to this range. Numbers 548 and above are available
# for non-x32 use.
#
```

현재 시스템 콜 테이블이 441까지 사용 중 이므로 442부터 445까지를 사용하였다.

## 2. 시스템 콜 헤더 파일 등록

```
root@ubuntu: /usr/src/linux/linux-5.11.22/include/linux
long ksys_old_semctl(int semid, int semnum, int cmd, unsigned long arg);
long ksys_msgget(key_t key, int msgflg);
long ksys_old_msgctl(int msqid, int cmd, struct msqid_ds __user *buf);
long ksys_msgrcv(int msqid, struct msgbuf __user *msgp, size_t msgsz,
                long msgtyp, int msgflg);
long ksys_msgsnd(int msqid, struct msgbuf __user *msgp, size_t msgsz,
                int msgflg);
long ksys_shmget(key_t key, size_t size, int shmflg);
long ksys_shmctl(char __user *shmaddr);
long ksys_old_shmctl(int shmid, int cmd, struct shmid_ds __user *buf);
long compat_ksys_semtimedop(int semid, struct sembuf __user *tsems,
                            unsigned int nsops,
                            const struct old_timespec32 __user *timeout);

int __sys_getsockopt(int fd, int level, int optname, char __user *optval,
                    int __user *optlen);
int __sys_setsockopt(int fd, int level, int optname, char __user *optval,
                    int optlen);
asmlinkage long sys_print_add(int a, int b, int *ptr);
asmlinkage long sys_print_min(int a, int b, int *ptr);
asmlinkage long sys_print_mul(int a, int b, int *ptr);
asmlinkage long sys_print_mod(int a, int b, int *ptr);
#endif
"syscalls.h" 1371L, 55960C                                1370,52          Bot
```

헤더 파일에 함수의 프로토타입을 선언한다. asmlinkage를 앞에 붙임으로서 어셈블리 코드에서도 C 함수 호출이 가능하게 선언 하였다.

### 3. 시스템 콜 함수 구현 – sys\_print\_add

```
root@ubuntu: /usr/src/linux/linux-5.11.22/kernel
#include<linux/kernel.h>
#include<linux/syscalls.h>
#include<asm/uaccess.h>
asmlinkage long sys_print_add(int a, int b, int *ptr){
    int add = a + b;
    put_user(add,ptr);
    return 0;
}
SYSCALL_DEFINE3(print_add,int,a,int,b,int*,ptr){
    return sys_print_add(a,b,ptr);
}
~
~
```

덧셈을 구현한 sys\_print\_add.c 파일 이다. 인자로 받은 두 변수를 덧셈 연산 해 준 후, put\_user 함수를 사용해 사용자가 지정한 포인터에 값을 써 준다. 단순히 덧셈 결과를 리턴 하지 않고 put\_user 함수를 사용한 이유는 결과가 -1에서 -4096 사이일 경우 제대로 값이 리턴 되지 않기 때문이다. 따라서 포인터 변수를 사용해 그 값에 연산 결과를 써 주어야 하는데, 이때 커널의 내용을 사용자의 메모리 영역에 바로 쓰는 것은 불가능하다. 내용을 복사하기 위해서 put\_user 함수를 사용하였다.

SYSCALL\_DEFINE3의 경우 시스템 콜 핸들러 이다. N에는 호출할 함수의 인자의 개수가 들어간다. 현재 sys\_print\_add 함수에서 3개의 인자를 필요로 하기 때문에 SYSCALL\_DEFINE3으로 매크로를 사용하였다

### 4. 시스템 콜 함수 구현 – sys\_print\_min

```
root@ubuntu: /usr/src/linux/linux-5.11.22/kernel
#include<linux/kernel.h>
#include<linux/syscalls.h>
#include<asm/uaccess.h>
asmlinkage long sys_print_min(int a, int b, int *ptr){
    int min = a - b;
    put_user(min,ptr);
    return 0;
}
SYSCALL_DEFINE3(print_min,int,a,int,b,int*,ptr){
    return sys_print_min(a,b,ptr);
}
~
~
```

뺄셈 연산을 위한 sys\_print\_min.c 파일 이다. 덧셈과 마찬가지로, 뺄셈 연산을 한 결과를 put\_user 함수를 사용하여 ptr에 복사 하였다. 인자가 3개 이므로 SYSCALL\_DEFINE3 매크로를 사용하였다.

## 5. 시스템 콜 함수 구현 – sys\_print\_mul

```
root@ubuntu: /usr/src/linux/linux-5.11.22/kernel
#include<linux/kernel.h>
#include<linux/syscalls.h>
#include<asm/uaccess.h>
asmlinkage long sys_print_mul(int a, int b, int *ptr){
    int mul = a * b;
    put_user(mul,ptr);
    return 0;
}
SYSCALL_DEFINE3(print_mul,int,a,int,b,int*,ptr){
    return (sys_print_mul(a,b,ptr));
}
```

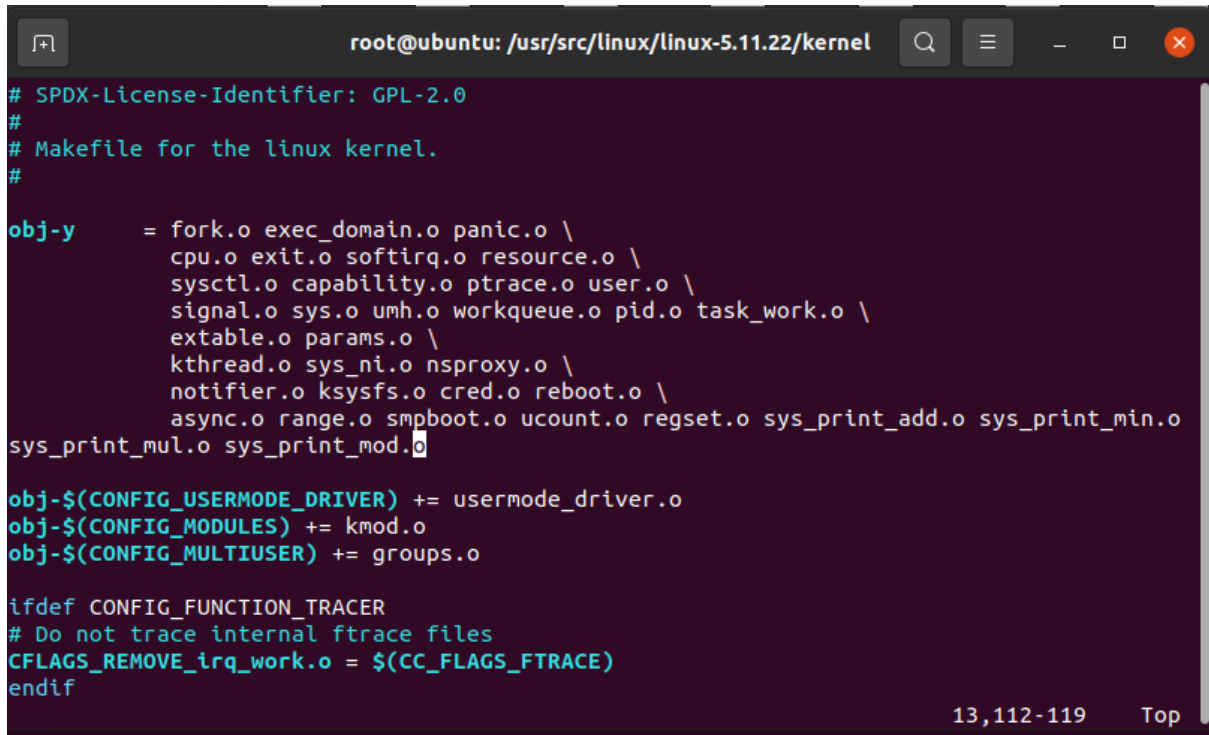
곱셈 연산을 위한 sys\_print\_mul.c 파일 이다. 덧셈과 마찬가지로, 곱셈 연산을 한 결과를 put\_user 함수를 사용하여 ptr에 복사 하였다. 인자가 3개 이므로 SYSCALL\_DEFINE3 매크로를 사용하였다.

## 6. 시스템 콜 함수 구현 – sys\_print\_mod

```
root@ubuntu: /usr/src/linux/linux-5.11.22/kernel
#include<linux/kernel.h>
#include<linux/syscalls.h>
#include<asm/uaccess.h>
asmlinkage long sys_print_mod(int a, int b, int *ptr){
    int mod = a % b;
    put_user(mod,ptr);
    return 0;
}
SYSCALL_DEFINE3(print_mod,int,a,int,b,int*,ptr){
    return sys_print_mod(a,b,ptr);
}
```

나머지 연산을 위한 sys\_print\_mod.c 파일 이다. 덧셈과 마찬가지로, 나머지 연산을 한 결과를 put\_user 함수를 사용하여 ptr에 복사 하였다. 인자가 3개 이므로 SYSCALL\_DEFINE3 매크로를 사용하였다.

## 7. Makefile 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_add.o sys_print_min.o
sys_print_mul.o sys_print_mod.o

obj-$(CONFIG_USERMODE_DRIVER) += usermode_driver.o
obj-$(CONFIG_MODULES) += kmod.o
obj-$(CONFIG_MULTIUSER) += groups.o

ifdef CONFIG_FUNCTION_TRACER
# Do not trace internal ftrace files
CFLAGS_REMOVE_irq_work.o = $(CC_FLAGS_FTRACE)
endif
```

13,112-119 Top

추가한 시스템 콜들이 다른 시스템 콜과 함께 컴파일 될 수 있도록 Makefile에 sys\_print\_add, sys\_print\_min, sys\_print\_mul, sys\_print\_mod 4개를 모두 등록 하였다.

## 8. 테스트 파일

```
jess@ubuntu: ~  
#include<stdio.h>  
#include<linux/kernel.h>  
#include<sys/syscall.h>  
#include<unistd.h>  
#include<string.h>  
#include<ctype.h>  
#include<stdbool.h>  
  
void parse_token(char*,int*,int*,char*);  
  
int main(){  
    int a,b,ans,*ptr;  
    char ch;  
    char buffer[1024];  
    int n;  
    printf("input number : ");  
    scanf("%d\n",&n);  
    while(n--){  
        memset(buffer,0,sizeof(buffer));  
        gets(buffer);  
        parse_token(buffer,&a,&b,&ch);  
        switch(ch){  
            case '+':  
                syscall(442,a,b,ptr);  
                break;  
            case '-':  
                syscall(443,a,b,ptr);  
                break;  
            case '*':  
                syscall(444,a,b,ptr);  
                break;  
            case '%':  
                syscall(445,a,b,ptr);  
                break;  
        }  
        printf("%d %c %d = %d\n",a,ch,b,*ptr);  
    }  
    return 0;  
}
```

```
void parse_token(char *buffer,int *a,int *b,char *ch){  
    int index = -1;  
    bool check = false;  
    *a = *b = 0;  
    for(int i=0;i<strlen(buffer);i++){  
        if(i == 0 && buffer[i] == '-'){  
            check = true;  
            continue;  
        }  
        if(!isdigit(buffer[i])){  
            *ch = buffer[i];  
            index = i + 1;  
            break;  
        }  
        else{  
            *a = *a * 10 + buffer[i] - '0';  
        }  
    }  
    if(check)  
        *a *= -1;  
    check = false;  
    if(buffer[index] == '-'){  
        check = true;  
        index++;  
    }  
    for(int i=index;i<strlen(buffer);i++)  
        *b = *b * 10 + buffer[i] - '0';  
    if(check)  
        *b *= -1;  
}
```

n에 입력 받은 수만큼 반복문을 돌며 새로 추가한 시스템 콜 함수를 호출 한다. 반복문에서는 buffer 배열을 0으로 초기화 한 후, 한 줄을 입력 받는다. 그리고 입력 받은 한 줄을 pars\_token 함수를 이용하여 나누어 주고, 나누어 진 값들을 이용하여 시스템 콜 함수를 호출한다. 덧셈은 442, 뺄셈은 443, 곱셈은 444, 나머지는 445를 이용하여 호출한다. 그리고 시스템 콜 함수를 포인터에 저장하는 방식으로 구현 하였으므로, int \* 형 ptr 변수도 함수 호출에 같이 넣어준다.

parse\_token 함수에서는 인자로 받은 buffer 배열을 각각 포인터 변수에 넣어 주는 일을 한다. 우선 첫 번째 반복문에서는 음수인지 아닌지 부터 검사하여 -10+1 처럼 첫번째 수가 음수인 경우 따로 처리 해 줄 수 있도록 한다. 그리고 나머지는 다른 연산자가 나올 때 까지 10을 곱해주고 자릿수를 더해 주는 연산을 반복하여 포인터변수 a가 가리키는 주소에 알맞은 값을 할당하게 된다. 그리고 맨 처음에 음수인지 검사했던 결과에 따라 음수였다면 -1을 곱해준다. 그 다음 문자는 연산자가 나오므로, 그 내용은 ch가 가리키는 곳에 넣어 준다. 이제 남은 int \*형 b가 가리키는 곳에 두번째 값을 넣어 주면 된다. a에 값을 넣었을 때와 같이 음수인지 검사부터 먼저 하고, 배열의 끝까지 가며 b의 값을 계산한다.

## 9. 테스트 파일 실행 결과

```
jess@ubuntu: ~  
jess@ubuntu:~$ ./test  
input number : 10  
100+30  
100 + 30 = 130  
1000-256  
1000 - 256 = 744  
24*13  
24 * 13 = 312  
17%5  
17 % 5 = 2  
100+-31  
100 + -31 = 69  
-2--50  
-2 - -50 = 48  
-13*7  
-13 * 7 = -91  
13%-2  
13 % -2 = 1  
2123456789+-2156  
2123456789 + -2156 = 2123454633  
-10*-6  
-10 * -6 = 60  
jess@ubuntu:~$
```