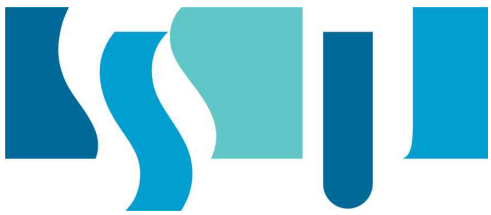


운영체제 과제 3

과목명	운영체제
교수명	김철홍 교수님
학 과	IT대학 컴퓨터학부
학 번	20172655
이 름	이강산
제출일	2021.10.12.

< 목차 >

1. 개요

2. 시스템 콜 함수 추가

2-0. 커널 환경 변경

2-1. 덧셈(+), 뺄셈(-), 곱셈(*), 나머지(%) 연산 시스템 콜 추가

2-1-1) 시스템 콜 테이블 등록

2-1-2) 시스템 콜 헤더 파일에 함수 프로토타입 정의

2-1-3) 시스템 콜 함수 구현

2-1-4) 작성한 시스템 콜 함수 Makefile에 등록

2-2. 시스템 콜 호출 테스트 프로그램

2-2-1) 소스 코드

2-2-2) 동작 예시

3. 과제를 통해 느낀 점

1. 개요

- 시스템 호출(system call)은 운영 체제의 커널이 제공하는 서비스에 대해, 응용 프로그램의 요청에 따라 커널에 접근하기 위한 인터페이스이다. 표준 C라이브러리나 사용자 어플리케이션의 작업을 수행하기 위해 내부적으로 시스템 호출이 이루어지는 경우가 많으며, 시스템 호출이 일어날 경우 커널은 다음과 같은 루틴을 통해 일을 처리하고 해당 작업을 요청한 프로세스에 반환한다. 먼저, 커널은 사용자가 시스템 호출을 함으로써 발생한 트랩의 번호를 통해 시스템 호출이 일어났음을 인지한다. 그 후, 시스템 콜 테이블을 참조함으로써 사용자가 원하는 기능을 수행한 후 그 결과를 반환한다.

시스템 호출은 사용자가 직접 추가할 수도 있으며, 과제를 통해 덧셈, 뺄셈, 곱셈, 나머지연산 총 네 가지 시스템 호출 함수를 커널에 추가하고 호출해본다.

2. 시스템 콜 함수 추가

2-0. 커널 환경 변경

- 리눅스 커널 5.11.22 버전을 설치하여 구현 및 테스트 환경을 구성하였다.

기존 리눅스 커널

```
san@linux:~$ uname -r
5.14.6-051406-generic
san@linux:~$
```

변경된 리눅스 커널

```
san@linux:/usr/src/linux$ ls
linux-5.11.22  linux-5.11.22.tar.xz  linux-image-5.11.22_1.0_amd64.deb
san@linux:/usr/src/linux$ uname -r
5.11.22
```

2-1. 덧셈(+), 뺄셈(-), 곱셈(*), 나머지(%) 연산 시스템 콜 추가

2-1-1) 시스템 콜 테이블 등록

- /usr/src/linux/linux-5.11.22/arch/x86/entry/syscalls 위치한 syscall_64.tbl 파일을 수정하여, 시스템 콜 테이블에 직접 작성한 시스템 콜 함수를 추가한다.

- 4개의 시스템 콜 함수는 각각 sys_my_add, sys_my_sub, sys_my_mul, sys_my_mod 의 이름을 가지며, 순서대로 443, 444, 445, 446번을 부여하였다.

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

```
441     common  epoll_pwait2             sys_epoll_pwait2

# my syscall functions START
442     common  print_hello             sys_print_hello
443     common  my_add                  sys_my_add
444     common  my_sub                  sys_my_sub
445     common  my_mul                  sys_my_mul
446     common  my_mod                  sys_my_mod
# my syscall functions END


# Due to a historical design error, certain syscalls are numbered differently
```

2-1-2) 시스템 콜 헤더 파일에 함수 프로토타입 정의

- /usr/src/linux/linux-5.11.22/include/linux에 위치한 syscalls.h 파일을 수정하여 추가할 4개의 시스템 콜 함수에 대한 함수 프로토타입을 정의한다.

- asmlinkage를 앞에 붙임으로써 어셈블리 코드에서도 C 함수 호출을 가능하게 하였다.

- 4개의 시스템 콜 함수 모두 두 개의 long 타입의 인자를 받아들여, 이름에 명시된 연산을 수행한 후, long 타입으로 반환한다.

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

```
                int optlen);

/* my system calls start */
asmlinkage long sys_print_hello(void);
asmlinkage long sys_my_add(long num1, long num2);
asmlinkage long sys_my_sub(long num1, long num2);
asmlinkage long sys_my_mul(long num1, long num2);
asmlinkage long sys_my_mod(long num1, long num2);
/*  my system calls end  */


#endif
```

2-1-3) 시스템 콜 함수 구현

- /usr/src/linux/linux-5.11.22/kernel 디렉토리에 추가될 시스템 콜 함수 구현 파일을 작성한다. (sys_my_add.c, sys_my_sub.c, sys_my_mul.c, sys_my_mod.c)

- 시스템 콜 함수가 정상적으로 호출되는지를 확인하기 위해 커널에 어떤 인자들을 넘겨받아 호출되었는지 출력하는 구문을 추가하였다. 각 시스템 콜 함수들은 두 개의 long 타입 인자를 넘겨받아 이름에 맞는 연산을 한 후 반환한다.

- SYSCALL_DEFINEn 꼴로 시작하는 코드는 리눅스 커널에서 제공하는 매크로 함수로, 시스템 콜 핸들러 함수이다. 시스템 콜 핸들러 함수는 다음과 같은 꼴로 작성된다.

- SYSCALL_DEFINEn(시스템 콜 함수 이름, 1번째 인자 타입, 1번째 인자 이름, 2번째 인자 타입, 2번째 인자 이름,...){...}

  (n은 핸들러의 대상인 시스템 콜의 인자 개수)

- 다음은 구현한 시스템 호출 함수들의 소스코드이다.

/usr/src/linux/linux-5.11.22/kernel/sys_my_add.c

```
#include <linux/kernel.h>
#include <linux/syscalls.h>
asmlinkage long sys_my_add(long num1, long num2)
{
        printk("sys_my_add CALLED, %ld + %ld\n", num1, num2);
        return (num1+num2);
}
SYSCALL_DEFINE2(my_add, long, num1, long, num2)
{
        return sys_my_add(num1, num2);
}
```

/usr/src/linux/linux-5.11.22/kernel/sys_my_sub.c

```
#include <linux/kernel.h>
#include <linux/syscalls.h>
asmlinkage long sys_my_sub(long num1, long num2)
{
        printk("sys_my_sub CALLED, %ld - %ld\n", num1, num2);
        return (num1-num2);
}
SYSCALL_DEFINE2(my_sub, long, num1, long, num2)
{
        return sys_my_sub(num1, num2);
}
```

/usr/src/linux/linux-5.11.22/kernel/sys_my_mul.c

```
#include <linux/kernel.h>
#include <linux/syscalls.h>
asmlinkage long sys_my_mul(long num1, long num2)
{
        printk("sys_my_mul CALLED, %ld * %ld\n", num1, num2);
        return (num1*num2);
}
SYSCALL_DEFINE2(my_mul, long, num1, long, num2)
{
        return sys_my_mul(num1, num2);
}
```

/usr/src/linux/linux-5.11.22/kernel/sys_my_mod.c

```
#include <linux/kernel.h>
#include <linux/syscalls.h>
asmlinkage long sys_my_mod(long num1, long num2)
{
        printk("sys_my_mod CALLED, %ld %% %ld\n", num1, num2);
        return (num1%num2);
}
SYSCALL_DEFINE2(my_mod, long, num1, long, num2)
{
        return sys_my_mod(num1, num2);
}
```

2-1-4) 작성한 시스템 콜 함수 Makefile에 등록

- /usr/src/linux/linux-5.11.22/kernel에 위치한 Makefile에 내가 추가한 시스템 콜이 다른 시스템 콜과 함께 컴파일 될 수 있도록, sys_my_add.o 등 목적 파일 이름들을 추가한다.

/usr/src/linux/linux-5.11.22/kernel/Makefile

```
# 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_my_add.o sys_my_sub.o \
            sys_my_mul.o sys_my_mod.o \
```

2-2. 시스템 콜 호출 테스트 프로그램

2-2-1) 소스 코드

- 테스트 파일은 다음 명세를 바탕으로 작성되었다.
- 프로그램 실행 후 +, -, *, % 만을 연산자로 사용한 수식을 입력받는다.
- 입력 시 숫자, 연산자 간 공백은 무시하며, 단항연산자(양수, 음수)를 지원한다.
- 잘못된 입력을 받았을 때(숫자가 아닌 피연산자, 피연산자 부족, 연산자 과다 등) 오류 메시지 출력 후 다음 입력을 받는다.
- x 입력 시 프로그램 종료한다.

- 수식의 연산은 앞서 작성한 시스템 콜 함수들을 사용한다.

test.c

```
/* 20172655 LEE KANG SAN */
#include <stdio.h>
#include <linux/kernel.h>
#include <sys/syscall.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
int main() {
        long length, num1, num2, result; 수식의 길이, 숫자1, 숫자2, 계산결과
        char exp_ori[101]; //수식 입력 버퍼
        char* exp; //입력된 수식(exp_ori)에서 공백문자 제거한 string 저장
        char n1[50]; //num1 string
        char n2[50]; //num2 string
        char op; //수식 내 operator 저장
        char* tok; //strtok() 리턴 받는 포인터 변수
        int opcheck; //수식 내 유효한 operator가 있는지 체크하는 변수

        printf("< ********** + - * %% Calculator **********
>\n");
        printf("input expression as following format : (num1) (op)
(num2)\n");
        printf("input character 'x' to exit                                    \n");

printf("============================================================\n");

        while(1) {
                opcheck=0;
                printf(">> ");
                fgets(exp_ori, 100, stdin); //표준입력으로부터 수식 입력받음
                exp_ori[strlen(exp_ori)-1]='\0'; //개행 제거
                length=strlen(exp_ori);
                exp=(char*)malloc(sizeof(length));
                //입력 받은 수식의 공백문자 제거하여 exp에 저장
                for(int i=0, j=0; i<length; i++)
                        if(exp_ori[i]!=' ') exp[j++]=exp_ori[i];
```

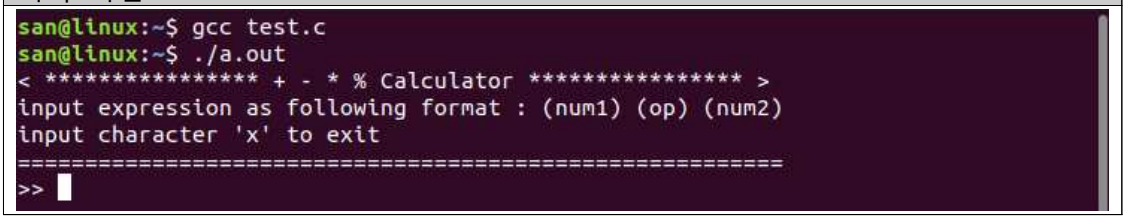
```
		length=strlen(exp);

		//x 입력시 프로그램 종료
		if(exp[0]=='x') {
			printf("inputed 'x', exit program.\n");
			return 0;
		}

		//num1의 마지막 자리 숫자 찾기
		//num1의 마지막 자리 숫자 다음은 무조건 유효한 연산자
		for(int i=0; i<length; i++) {
			if('0'<=exp[i] && exp[i]<='9') {

if(exp[i+1]=='+'||exp[i+1]=='-'||exp[i+1]=='*'||exp[i+1]=='%') {
					op=exp[i+1];
					opcheck=1; //found valid operator : +, -, *, % after num1
					exp[i+1]=' '; //replace operator with ' ' for tokenizing
					break;
				}
			}
		}

		//유효한 연산자 없으므로 에러 출력 후 다시 입력받음
		if(opcheck==0) {
			printf("Wrong expression.\n");
			continue;
		}

		//토크나이징 : n1, n2
		tok=strtok(exp, " ");
		strcpy(n1, tok);
		tok=strtok(NULL, " ");
		if(tok==NULL) {
			printf("Wrong expression.\n");
			continue;
```

```
                }
                strcpy(n2, tok);

                //토큰 n1, n2가 숫자인지 확인 atoi()
                if(0==(num1=atol(n1))) {
                        if(n1[0]=='0') ; //실제 숫자 0인 경우 예외처리
                        else { printf("num1 is not a number.\n"); continue; }
                }
                if(0==(num2=atol(n2))) {
                        if(n2[0]=='0') ; //실제 숫자 0인 경우 예외처리
                        else { printf("num2 is not a number.\n"); continue; }
                }

                printf("[%ld] %c [%ld] = ", num1, op, num2);

                switch(op) //연산자에 따라 서로 다른 시스템 콜 호출
                {
                        case '+': //sys_my_add
                                result=syscall(443, num1, num2);
                                break;
                        case '-': //sys_my_sub
                                result=syscall(444, num1, num2);
                                break;
                        case '*': //sys_my_mul
                                result=syscall(445, num1, num2);
                                break;
                        case '%': //sys_my_mod
                                result=syscall(446, num1, num2);
                                break;
                }
                printf("[%ld]\n", result); //연산 결과 출력

        }
        return 0;
}
```

2-2-2) 동작 예시

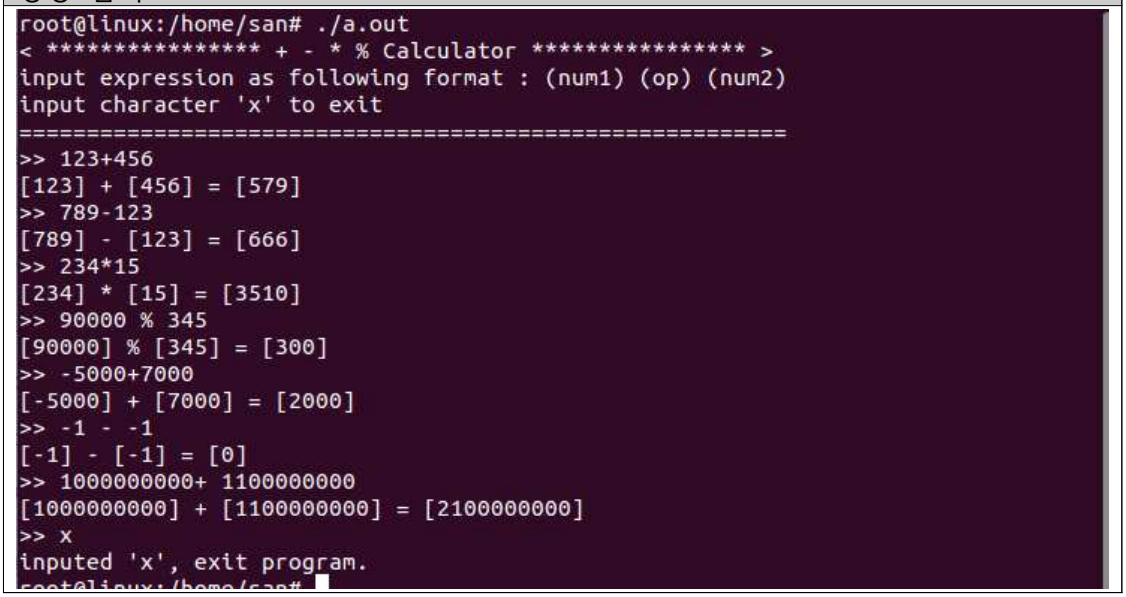
- 프로그램 시작 시 수식의 입력을 기다린다.

시작 화면



- 여러 가지 정상 입력을 통해 작성한 시스템 콜 함수를 호출하였다.
- 표준입력(stdin)으로 입력하며 표준출력(stdout)에 출력한다.
- +, =, *, % 연산을 수행한다.
- 입력 시 추가된 공백을 무시하며, 단항연산자(음수, 양수)를 지원한다.
- 대괄호를 사용하여 수식 및 결과를 명시적으로 표현하였다.
- x 입력을 통해 프로그램을 종료한다.

정상 입력



- 표준입력(stdin)으로 올바르지 못한 수식 입력 시 표준출력(stdout)에 오류 메시지를 출력한다.

- 피연산자가 부족한 경우, 피연산자가 정수가 아닌 경우, 연산자가 잘못 사용된 경우, 문자열이 입력된 경우 등

- 대괄호를 사용하여 수식 및 결과를 명시적으로 표현하였다.

- x 입력을 통해 프로그램을 종료한다.

비정상 입력 (예외처리)

```
< *************** + - * % Calculator *************** >
input expression as following format : (num1) (op) (num2)
input character 'x' to exit
=========================================================
>> 123+456
[123] + [456] = [579]
>> 123+
Wrong expression.
>> +456
Wrong expression.
>> a1+123
num1 is not a number.
>> 123+a1
num2 is not a number.
>> --123+123
num1 is not a number.
>> 123+-+345345
num2 is not a number.
>> abc
Wrong expression.
>> x
inputed 'x', exit program.
```

- 시스템 콜 작성 시 추가한 printk()를 통해 수식 연산을 위한 시스템 호출이 정상적으로 이루어지고 있는 지를 확인하였다.

시스템 콜 호출 확인(dmesg)

< 커널 로그 확인 >

```
[  762.399814] sys_my_add CALLED, 10 + 3
[  765.125035] sys_my_sub CALLED, 10 - 3
[  768.722451] sys_my_mul CALLED, 10 * 3
[  771.916170] sys_my_mod CALLED, 10 % 3
```

< 결과 출력 >

```
root@linux:/home/san# ./a.out
< *************** + - * % Calculator *************** >
input expression as following format : (num1) (op) (num2)
input character 'x' to exit
=========================================================
>> 10+3
[10] + [3] = [13]
>> 10-3
[10] - [3] = [7]
>> 10*3
[10] * [3] = [30]
>> 10%3
[10] % [3] = [1]
>> x
inputed 'x', exit program.
```

3. 과제를 통해 배운 점

- 과제를 통해 시스템 호출을 커널에 추가하는 법을 알게 되었다. 헤더에 함수 선언을 추가하고, 별도의 함수 정의 파일을 생성 및 Makefile에 추가하는 것은 기존에 하던 것과 다르지 않았지만, 시스템 호출을 추가하기 위해서는 시스템 콜 테이블에 등록까지 해야 한다는 것을 알게 되었다.

과제를 진행하며 커널 컴파일 중 오류가 발생하였는데, 시스템 콜 테이블에 작성한 이름에 언더바(_)가 하나를 빠트려 벌어진 일이었다. 덕분에 커널 컴파일만 수차례 해 볼 수 있었고, 커널 컴파일에 좀 더 능숙하게 되었다.