컴퓨터학부 20172642 신형담

1-1. 구현개요

덧셈, 뺄셈, 곱셈, 나머지 연산을 제공하는 신규 시스템 콜을 추가하여 이를 호출하여 정상적으로 결과를 출력하는지 프로그램을 구현한다.

우선 시스템 콜 테이블에 해당 함수들을 등록한다. syscall_64.tbl 파일을 편집하여 442 ~ 445 사이에 각각 덧셈, 뺄셈, 곱셈, 나머지 연산을 등록했다.

```
442     common  print_add               sys_print_add
443     common  print_abs               sys_print_abs
444     common  print_mul               sys_print_mul
445     common  print_mod               sys_print_mod
```

이후 syscall.h 파일에 해당 함수들의 프로토타입을 등록한다. 4가지의 연산 모두 int형 피연산자 2개로 이루어진 수식을 입력 받아 수행할 것이다. 또한 함수에서 이 값을 계산하여 결과를 res변수에 저장하기 때문에 아래와 같이 프로토타입을 선언했다.

```
asmlinkage int sys_print_add(int op1, int op2, int* res);
asmlinkage int sys_print_abs(int op1, int op2, int* res);
asmlinkage int sys_print_mul(int op1, int op2, int* res);
asmlinkage int sys_print_mod(int op1, int op2, int* res);
```

다음으로 시스템 콜 함수를 구현한다. sys_print_add, sys_print_abs, sys_print_mul, sys_print_mod 함수는 모두 입력받은 수식의 int형 피연산자 2개(op1, op2)와 결과를 저장할 포인터 변수 res를 파라미터로 받아서 val이란 지역변수에 계산한 값을 저장하고 put_user 함수를 통해 커널에서 계산한 값 val을 res에 넘겨주고 0을 리턴하는 방식으로 동작한다. 여기서 예외를 고려해야 하는 함수는 sys_print_mod 함수인데, 나머지 연산의 경우 두번째 피연산자 op2의 값이 0이 될 수 없다. 따라서 op2 == 0 인 경우 -1을 리턴한다. 아래 캡처들은 해당 시스템 콜 함수들의 코드이다.

- sys_print_add.c

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

asmlinkage int sys_print_add(int op1, int op2, int* res) {
        int val;
        val = op1 + op2;
        put_user(val, res);
        return 0;
}

SYSCALL_DEFINE3(print_add, int, op1, int, op2, int*, res) {
        return sys_print_add(op1, op2, res);
}
```

- sys_print_abs.c

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

asmlinkage int sys_print_abs(int op1, int op2, int* res) {
        int val;
        val = op1 - op2;
        put_user(val, res);
        return 0;
}

SYSCALL_DEFINE3(print_abs, int, op1, int, op2, int*, res) {
        return sys_print_abs(op1, op2, res);
}
```

- sys_print_mul.c

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

asmlinkage int sys_print_mul(int op1, int op2, int* res) {
        int val;
        val = op1 * op2;
        put_user(val, res);
        return 0;
}

SYSCALL_DEFINE3(print_mul, int, op1, int, op2, int*, res) {
        return sys_print_mul(op1, op2, res);
}
```

- sys_print_mod.c

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

asmlinkage int sys_print_mod(int op1, int op2, int* res) {
        if (op2 == 0)
                return -1;
        int val;
        val = op1 % op2;
        put_user(val, res);
        return 0;
}

SYSCALL_DEFINE3(print_mod, int, op1, int, op2, int*, res) {
        return sys_print_mod(op1, op2, res);
}
```

if(op == 2) return -1; 은 op2가 0일 경우에 대한 처리이다.

다음으로 Makefile을 편집한 후 마지막으로 커널을 다시 컴파일 후 해당 커널로 재부팅 한다.

```
sys_print_add.o sys_print_abs.o sys_print_mul.o sys_print_mod.o
```

마지막으로 새로 등록한 시스템 콜 함수들이 정상적으로 동작되는지 테스트할 프로그램 test.c를 구현한다. test.c는 main 함수와 lex 함수로 구성된다.

lex 함수는 파라미터로 첫 번째 피연산자 값을 저장할 int형 포인터 op1, 두 번째 피연산자 값을 저장할 int형 포인터 op2, 연산의 종류를 저장할 int형 포인터 cal을 갖는다. lex 함수는 사용자로부터 입력받은 수식을 getchar 함수를 통해 읽어서 피연산자와 연산자를 값으로 저장하는 역할을 수행한다. 우선 while문을 통해 입력받은 수식의 시작에 오는 공백(' '이나 '₩t')을 변수 ch에 읽어들이면 getchar 함수를 수행하여 공백들을 건너뛴다. 이후 첫 번째 피연산자가 음수일 경우 '-'부터 전역변수 text 배열에 순서대로 getchar 함수를 이어서 진행하고 읽어들인 ch에 대해 do {} while(isdigit(ch))문을 통해 text 배열에 피연산자 숫자를 하나씩 이어서 저장한다. 이후 *op1 = atoi(text)를 통해 파라미터로 받은 변수 op에 값을 저장한다. 이후 연산자를 읽어들일 때 까지 while문을 통해 getchar 함수를 수행하여 공백들을 건너뛰고 ch에 연산자를 읽어들일 경우 if문을 통해 '+'인 경우는 0, '-'인 경우는 1, '*'인 경우는 2, '%'인 경우는 3을 cal에 할당한다. 이후 첫 번째 피연산자를 읽어들인 방식과 동일한 방식으로 두 번째 피연산자를 읽어들여 op2에 값을 할당한다.

main 함수는 if문을 통해 cal == 0(덧셈)일 경우 syscall(442, op1, op2, &res); 를 호출하고 "사용자로부터 입력 받은 수식 = res"의 형태로 값을 출력한다. 마찬가지로 cal == 1(뺄셈), cal == 2(곱셈), cal == 3(나머지)인 경우에도 각각 syscall(443, op1, op2, &res); , syscall(444, op1, op2, &res); , syscall(445, op1, op2, &res);를 호출하고 덧셈과 같은 형태로 결과를 출력한다. 추가적으로 나머지 연산인 경우 op2 == 0일 때 -1을 리턴하기 때문에 syscall(445, op1, op2, &res)의 리턴 값이 -1인 경우 fprintf 함수를 통해 "op2 can't be 0"라는 에러 메시지를 출력하도록 한다. 아래 캡처들은 test.c의 코드이다.

-test.c

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

char text[20];
void lex(int*, int*, int*);

int main() {
    int res;
    int op1, op2;
    int cal;

    lex(&op1, &op2, &cal);
    if(cal == 0) {
        syscall(442, op1, op2, &res);
        printf("%d + %d = %d\n", op1, op2, res);
        return 0;
    }
    if(cal == 1) {
        syscall(443, op1, op2, &res);
        printf("%d - %d = %d\n", op1, op2, res);
        return 0;
    }
    if(cal == 2) {
        syscall(444, op1, op2, &res);
        printf("%d * %d = %d\n", op1, op2, res);
        return 0;
    }
    if(cal == 3) {
        if(syscall(445, op1, op2, &res) == -1) {
            fprintf(stderr, "op2 can't be 0\n");
            return -1;
        }
        printf("%d %% %d = %d\n", op1, op2, res);
        return 0;
    }
}

void lex(int* op1, int* op2, int* cal) {
    char ch = ' ';
    int i = 0;
    memset(text, '\0', sizeof(text));
    while (ch == ' ' || ch =='\t')
        ch = getchar();
    if(ch == '-') {
        text[i++] = ch;
        ch = getchar();
    }
    if(isdigit(ch)) {
        do{
            text[i++] = ch;
            ch = getchar();
        } while(isdigit(ch));
        text[i] = 0;
        *op1 = atoi(text);
    }

    while (ch == ' ' || ch =='\t')
        ch = getchar();
    if(ch == '+') {
        ch = getchar();
        *cal = 0;
    }
    else if(ch == '-') {
```

```
69          ch = getchar();
70          *cal = 1;
71      }
72      else if(ch == '*') {
73          ch = getchar();
74          *cal = 2;
75      }
76      else {
77          ch = getchar();
78          *cal = 3;
79      }
80
81      i = 0;
82      memset(text, '\0', sizeof(text));
83      while (ch == ' ' || ch =='\t')
84          ch = getchar();
85      if(ch == '-') {
86          text[i++] = ch;
87          ch = getchar();
88      }
89      if(isdigit(ch)) {
90          do{
91              text[i++] = ch;
92              ch = getchar();
93          } while(isdigit(ch));
94          text[i] = 0;
95          *op2 = atoi(text);
96      }
97  }
```

1-2. 실행결과

- 덧셈

```
shin@shin-virtual-machine:~/shin/os$ ./a.out
10 + 10
10 + 10 = 20
```

```
shin@shin-virtual-machine:~/shin/os$ ./a.out
10 - 240
10 - 240 = -230
```

```
shin@shin-virtual-machine:~/shin/os$ ./a.out
-120 + 10
-120 + 10 = -110
```

결과가 음수인 경우에도 정상적으로 출력됨을 확인할 수 있다.

- 뺄셈

```
shin@shin-virtual-machine:~/shin/os$ ./a.out
10 - 7
10 - 7 = 3
```

```
shin@shin-virtual-machine:~/shin/os$ ./a.out
16 - 45
16 - 45 = -29
```

```
shin@shin-virtual-machine:~/shin/os$ ./a.out
16 - -40
16 - -40 = 56
```

결과가 음수인 경우와 두 번째 피연산자가 음수인 경우에도 정상적으로 출력되었다.

- 곱셈

```
shin@shin-virtual-machine:~/shin/os$ ./a.out
20 * -9
20 * -9 = -180
```

```
shin@shin-virtual-machine:~/shin/os$ ./a.out
-15 * -2
-15 * -2 = 30
```

```
shin@shin-virtual-machine:~/shin/os$ ./a.out
10 * 4
10 * 4 = 40
```

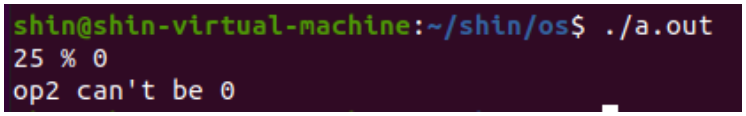
음수와 음수의 곱, 양수와 음수의 곱 역시 정상적으로 값이 출력되었다.

- 나머지

```
shin@shin-virtual-machine:~/shin/os$ ./a.out
10 % 3
10 % 3 = 1
```

```
shin@shin-virtual-machine:~/shin/os$ ./a.out
-15 % 2
-15 % 2 = -1
```

```
shin@shin-virtual-machine:~/shin/os$ ./a.out
20 % 4
20 % 4 = 0
```



결과가 정상적으로 출력되었고, 두번째 피연산자가 0일 때 에러메시지를 출력한다.