

운영체제

System Call 추가와 이를 이용한 프로그램 구현

과제 정보

1. 파일 목록

커널 소스 경로: /usr/src/linux/linux-5.15.120/

파일명	설명
install.sh	커널 소스 폴더에 작업 파일을 자동으로 반영하는 shell command 파일.
main.c	추가된 system call을 이용한 응용프로그램 소스 파일.
Makefile	응용프로그램을 컴파일하기 위한 Makefile.
위의 세 파일은 경로 변경사항 없음.	
Makefile.def	추가된 system call을 위한 변경된 Makefile. <u>해당 경로로 이동 후 이름 변경 필요.</u>
/usr/src/linux/linux-5.15.120/kernel/Makefile	
sys_calc.c	입력된 두 정수를 계산하는 system call 함수 소스 파일.
/usr/src/linux/linux-5.15.120/kernel/sys_calc.c	
sys_rev.c	입력된 정수를 역순으로 출력하는 system call 함수 소스 파일.
/usr/src/linux/linux-5.15.120/kernel/sys_rev.c	
syscall_64.tbl	추가된 system call을 테이블에 추가한 테이블 파일.
/usr/src/linux/linux-5.15.120/arch/x86/entry/syscalls/syscall_64.tbl	
syscalls.h	추가된 system call을 헤더 파일에 적용한 C 헤더 파일.
/usr/src/linux/linux-5.15.120/include/linux/syscalls.h	



“install.sh” 파일을 이용하여 파일을 자동으로 이동하도록 제작하였습니다.
“3. 설치 방법”을 참조해 주시기 바랍니다.

1. 과제 설명

해당 과제는 입력된 자연수를 뒤집는 함수와 입력된 수식(+ 혹은 -)을 받아 계산하는 함수를 System Call에 추가한 후 추가한 System Call 함수를 이용하여 응용프로그램을 만드는 과제이다. 입력과 출력 예시는 다음과 같다.

```
Input: 123
Output: 321
Input: 123 + 123
Output: 0
Input: 123+123
Output: 0
Input: 123-123
Output: 246
Input: -123-123
Wrong Input!
Input:
exit
```

입력은 자연수만 받고, 정수 한 개만 들어온 경우 숫자를 뒤집어서 출력하도록 했다. 자연수 두 개와 부호(+ 혹은 -, 자연수 부호 자연수)가 들어온 경우 반대 부호로 계산하여 결과를 출력하도록 하였다. 이 때, 중간에 공백이 있는 경우 무시한다.

위의 규칙에 맞지 않는 입력은 “Wrong Input!”을 출력하고, 아무것도 입력하지 않으면 프로그램은 종료된다.

2. 파일별 설명

파일명	설명
install.sh	커널 소스 폴더에 작업 파일을 자동으로 반영하는 shell command 파일.
main.c	추가된 system call을 이용한 응용프로그램 소스 파일.
Makefile	응용프로그램을 컴파일하기 위한 Makefile.
Makefile.def	추가된 system call을 위한 변경된 Makefile.
sys_calc.c	입력된 두 정수를 계산하는 system call 함수 소스 파일.
sys_rev.c	입력된 정수를 역순으로 출력하는 system call 함수 소스 파일.
syscall_64.tbl	추가된 system call을 테이블에 추가한 테이블 파일.
syscalls.h	추가된 system call을 헤더 파일에 적용한 C 헤더 파일.

3. 설치 방법

A. Kernel Compile

“install.sh” 파일의 권한을 확인하고, 실행이 불가능한 권한인 경우 권한을 수정한다.

제출한 폴더에서, “install.sh”을 실행하여 작업물을 커널 소스에 반영한다.

커널 소스의 경로는 “/usr/src/linux/linux-5.15.120/”이며, 변경이 필요한 경우 셸파일 내의 “KERNEL_PATH” 변수를 수정한다. 해당 경로의 마지막은 “/”를 반드시 넣어야 한다.

커널 소스 경로로 이동하여, “.config” 파일을 “/boot” 디렉토리에서 가져와 “make menuconfig”를 이용하여 불러온다.

“make” 명령어를 이용하여 컴파일 한다.

“make modules_install” 명령어를 이용하여 컴파일 한다.

컴파일 이 끝난 상태에서 “make install” 명령어를 이용하여 커널을 설치한다.

설치가 끝난 후 “reboot” 명령어를 이용하여 다시 시작한 후 커널 버전을 확인한다.



Digital Ocean의 서버를 이용하여 과제를 수행하던 중 “kernel-package”가 설치되지 않아 기본적인 방식을 이용하여 컴파일 후 커널을 설치하였다.

B. Application Compile

1. 과제 폴더에서 “make” 명령어를 이용하여 소스 “main.c”를 컴파일한다.
2. 결과물인 “assignment2.out”를 실행한다.

4. 코드 설명

A. sys_calc.c

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

asmlinkage long sys_calc(long num_1, long num_2, char op) {
    if(op == '-') {
        return num_1 + num_2;
    }
    else if(op == '+') {
        return num_1 - num_2;
    }
    else {
        return -999999;
    }
}

SYSCALL_DEFINE3(calc, long, num_1, long, num_2, char, op) {
    return sys_calc(num_1, num_2, op);
}
```

해당 파일은 입력된 수식을 계산하기 위한 System call 함수를 정의해놓은 파일이다. 함수 “sys_calc”에서 두 자연수를 받아 변수 “op”의 부호에 따라 반대 부호를 이용하여 결과를 반환하도록 하였다. 응용프로그램에서 부호의 문법을 확인하지만, 안전사항을 위해 정해진 기호 문법 외의 기호가 들어오는 경우 “-999999”을 반환하도록 하였다.

그 후 SYSCALL_DEFINE를 이용하여 system call을 받도록 정의하였다. 함수 “sys_calc”는 3개의 변수를 받아야 하기에, SYSCALL_DEFINE3을 이용하여 정의했다. 결과값으로 함수 “sys_calc”의 결과를 반환한다.

B. sys_rev.c

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

asmlinkage long sys_rev(char *str) {
    int len = 0;
    int start = 0;
    int end = 0;

    while(str[len] != '\0') {
        len++;
    }

    end = len - 1;

    while(start < end) {
        char temp = str[start];
        str[start] = str[end];
        str[end] = temp;

        start++;
        end--;
    }

    return 0;
}

SYSCALL_DEFINE1(rev, char*, num) {
    return sys_rev(num);
}
```

해당 파일은 문자열 자연수를 뒤집어 출력하기 위한 System call 함수를 정의해놓은 파일이다. 입력된 문자열에서 문자열 끝의 위치를 알아낸 후 문자열에서 앞과 뒤를 스왑해준다. 스왑 작업은 start index와 end index가 겹치게 될때까지 진행한다. 작업이 끝나게 되면 0을 반환한다.

그 후 SYSCALL_DEFINE을 이용하여 system call을 받도록 정의하였다. 함수 “sys_rev”는 1개의 변수를 받아야 하기에, SYSCALL_DEFINE1을 이용하여 정의했다. 결과값으로 함수 “sys_rev”의 결과를 반환한다.

C. main.c

분량 문제로 인하여 실제 코드를 부분별로 나누어 설명하였다.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include <linux/kernel.h>
#include <sys/syscall.h>

// System call table
#define SYSCALL_CALC 449
#define SYSCALL_REV 450

#define OP_ERR 0
#define OP_PRINT 1
```

먼저, 해당 프로그램에서 사용될 라이브러리를 가져온 후 필요한 상수를 정의하였다. 상수명에서 SYSCALL로 시작하는 상수는 해당 system call의 정의된 주소값을 의미한다. 상수명에서 OP로 시작하는 상수는 동작 중 필요한 flag 상수다.

```
// 문자열 내의 숫자 여부를 판별하는 함수.
int isDigit(char *num) {
    int len = strlen(num);

    for(int i = 0; i < len; i++) {
        if(num[i] >= '0' && num[i] <= '9' || num[i] == '\n') continue;
        else return 0;
    }

    return 1;
}
```

해당 함수는 문자열이 오직 숫자로만 이루어진 문자열인지 판단하는 함수이다. 각 문자열의 문자마다 판단하여 모두 숫자인 경우 1을 반환하고, 아닌 경우 0을 반환한다.

```
// + 혹은 -가 있는 index를 반환하는 함수.
int whereOp(char *input) {
    for(int i = 0; i < strlen(input); i++) {
        if(input[i] == '+' || input[i] == '-') return i;
    }

    return -1;
}
```

해당 함수는 문자열 내에 어느 위치에 문법에 맞는 부호가 있는지 찾아 해당 위치 인덱스를 반환하는 함수다. 여기서 올바른 문법은 + 혹은 -를 말한다.

해당 부호가 문자열에 없는 경우 -1을 반환한다.

```
// 시작 인덱스부터 말단 인덱스까지의 글자를 반환하는 함수.
char* getNumberString(char *input, int start_idx, int end_idx) {
    char *number = (char *)malloc(sizeof(char) * 1024);

    for(int i = start_idx; i < end_idx; i++) {
        number[i - start_idx] = input[i];
    }
    // 끝 자리에 NULL 삽입.
    number[strlen(input) - start_idx] = '\0';

    return number;
}
```

해당 함수는 문자열 내에서 숫자만을 가져와 반환하는 함수이다. 시작 인덱스인 start_idx에서 끝 인덱스인 end_idx까지의 문자열을 가져와 반환하는 방식이다.

```
// 입력된 문자열에 대해 공백 제거 후 반환하는 함수.
char* removeSpace(char *input) {
    char *input_removed = (char*)malloc(sizeof(char) * 1024);
    int len = strlen(input);
    int pos_removed = 0;

    for(int i = 0; i < len - 1; i++) {
        if(input[i] == ' ') continue;

        input_removed[pos_removed++] = input[i];
    }
    input_removed[pos_removed] = '\0';

    return input_removed;
}
```

해당 함수는 입력된 문자열에서 공백을 제거하기 위한 함수이다. 문자열을 순차적으로 확인하여 공백인 경우 메모리에 할당하지 않고, 공백이 아닌 경우 추가하여 해당 문자열을 반환한다.

```
int main(int argc, char const *argv[]) {

    while(1) {
        int idx = -1;
        int op_state = OP_PRINT;
        char input[2048];

        printf("Input: ");
        fgets(input, sizeof(input), stdin);
        strcpy(input, removeSpace(input));

        idx = whereOp(input);

        // 아무것도 입력되지 않을 시 종료.
        if(strlen(input) == 0) return 0;

        ...
    }
}
```

해당 부분은 함수 main의 시작 부분이다. 실행에 필요한 변수를 선언한 후 문자열을 입력받는다. 이 때, 공백을 포함하여 입력을 받기 위해 함수 fgets를 이용하였다. 입력을 받은 후 부호가 있는 위치를 찾고, 해당 문자열이 길이가 0인 경우 종료하도록 해놓았다.


```

int main(int argc, char const *argv[]) {

    while(1) {
        ...

        // 부호의 입력 여부에 따른 동작.
        if(idx == -1) {
            if(!isDigit(input))
                printf("Wrong Input!\n");
            else {
                // 에러가 발생하지 않는 경우 SYS Call
                syscall(SYS_CALL_REV, input);
                // sys_rev(input);
                printf("Output: %s\n", input);
            }
        }
        ...
    }
}

```

해당 부분에서는 문자열에 부호가 없는 경우 자연수가 입력된것으로 간주, 문자열을 뒤집어 출력하는것을 시도한다. 이 때, 해당 문자열이 숫자로만 이루어져 있는지 판별하고, 아닌경우 오류 메시지를 출력한다. 문자열이 숫자로만 이루어진 경우 함수 syscall을 이용하여 지정된 함수 번호와 변수를 넘겨준다. 이후 도출된 결과를 출력한다.

```

int main(int argc, char const *argv[]) {

    while(1) {
        ...

        else {
            char *num_1 = getNumberString(input, 0, idx);
            char *num_2 = getNumberString(input, idx + 1,
                                           (int)strlen(input));

            char op      = input[idx];

            // 예외 발생시 오류코드 저장.
            if(!(isDigit(num_1) && isDigit(num_2))) op_state = OP_ERR;
            if(!(op == '-' || op == '+'))          op_state = OP_ERR;

            // 에러가 발생하지 않는 경우 SYS Call
            if(op_state) {
                long int result = syscall(SYSCALL_CALC,
                                          atol(num_1), atol(num_2), op);
                // long int result = sys_calc(atol(num_1),
                //                             atol(num_2), op);
                printf("Output: %ld\n", result);
            }
            else printf("Wrong Input!\n");
        }
    }

    return 0;
}

```

해당 부분은 부호의 위치가 있는 경우 실행되는 부분이다. 부호의 위치를 기준으로 숫자를 두 부분으로 나누어 문자열을 가져온 후 두 문자열이 숫자인지와 부호가 정해진 문법에 맞는지 확인한다. 이 때, 문법이 맞지 않은 경우 변수 `op_state`에 오류임을 기록한다.

이후 오류 여부를 기준으로 오류가 없는 경우 함수 `syscall`을 이용하여 지정된 함수 번호와 변수를 넘겨준다. 실행 결과로 나온 변수를 출력한다. 오류가 발생한 경우 오류 메시지를 출력한다.

D. Makefile

```
assignment2.out: main.o
    gcc -o assignment2.out main.o

main.o: main.c
    gcc -c -o main.o main.c
```

해당 파일은 응용프로그램을 컴파일하기 위해 사용되는 Makefile이다. 요구사항에 맞추어 파일명을 “assignment2.out”로 저장하도록 하였다.

E. Makefile.def

```
...

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_calc.o sys_rev.o

...
```

해당 파일은 추가된 system call을 추가한 커널 컴파일을 위한 Makefile이다. 응용프로그램을 위한 Makefile과 구분하기 위해 이름을 변경하였다. 커널 소스 디렉토리로 이동 후 이름 변경이 필요하다.

실제 수정된 부분만 보고서에 첨부하였다.

F. syscall_64.tbl

```
...  
449 common  calc          sys_calc  
450 common  rev           sys_rev  
...
```

해당 파일은 추가한 system call을 system call에 추가 반영한 테이블 파일이다. 계산 함수는 449번, 자연수 뒤집기 함수는 450번을 할당하였다.

실제 수정된 부분만 보고서에 첨부하였다.

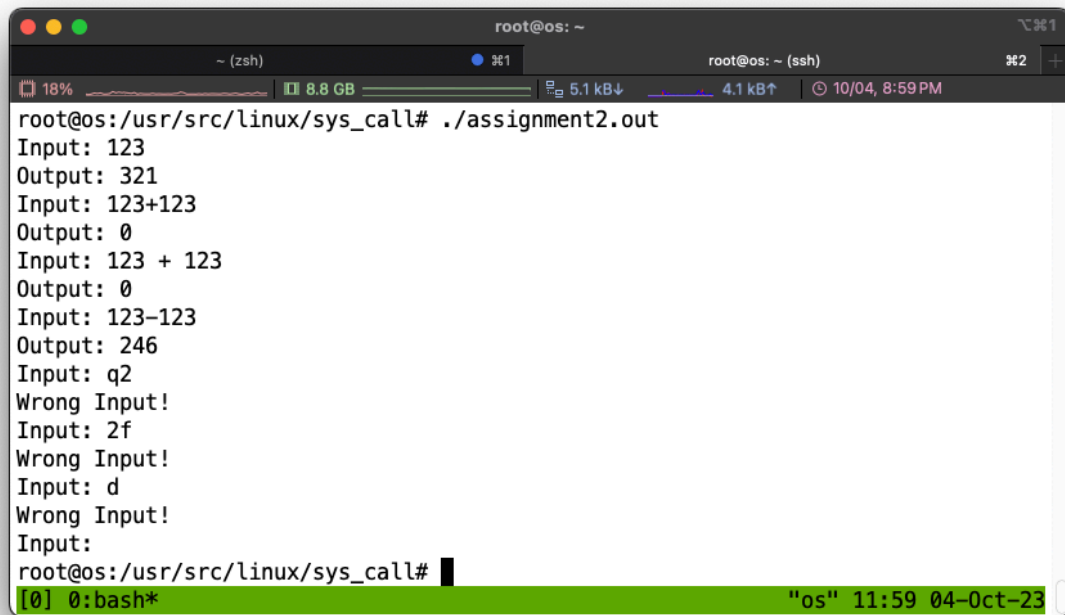
G. syscalls.h

```
...  
  
// sys_calc define.  
asmlinkage long sys_calc(long num_1, long num_2, char op);  
// sys_rev define.  
asmlinkage long sys_rev(char *str);  
  
#endif
```

해당 파일은 추가한 system call을 헤더파일에 적용한 파일이다.

실제 수정된 부분만 보고서에 첨부하였다.

5. 실행 결과



```
root@os: ~  
~ (zsh) 8.8 GB 5.1 kB↓ 4.1 kB↑ 10/04, 8:59 PM  
root@os:/usr/src/linux/sys_call# ./assignment2.out  
Input: 123  
Output: 321  
Input: 123+123  
Output: 0  
Input: 123 + 123  
Output: 0  
Input: 123-123  
Output: 246  
Input: q2  
Wrong Input!  
Input: 2f  
Wrong Input!  
Input: d  
Wrong Input!  
Input:  
root@os:/usr/src/linux/sys_call#  
[0] 0: bash* "os" 11:59 04-Oct-23
```

Fig 1. System call을 이용한 응용프로그램 실행시 결과.

위의 입력 예시와 같이, 요구되는 두 기능과 공백 무시 그리고 문법에 맞지 않는 입력이 들어올 시 “Wrong Input!”이 출력되도록 제작하였다.