2021년 1학기 운영체제 (ICT332) 1차 과제

1. 목적

이번 과제의 목적은 일반 배포판 리눅스에 새로운 Kernel을 컴파일하고 수정해서 새로운 System Call을 추가하는 것이다. System Call은 Kernel이 시스템이나 하드웨어에 특권이 있어야만 하는 동작들을 지정해둔 함수로써 User-Space에서 Kernel-Space로 전환하기 위해서 특별히 지정된 Trap Instruction을 거쳐야 한다. 본 과제에서는 Kernel을 직접 컴파일하고, 이후 기존의 Kernel에 새로운 System Call을 추가한 후에 이를 호출할 수 있는 user program까지를 구현하도록 한다.

새로 추가한 system call은 user program으로부터 입력 받은 integer 값을 출력하는 기능을 수행하도록 작성한다.

2. 과제 구현 목표

1. Kernel

   System Call시 전달받은 정수형 인자를 printk로 출력하도록 한다.

2. User Process

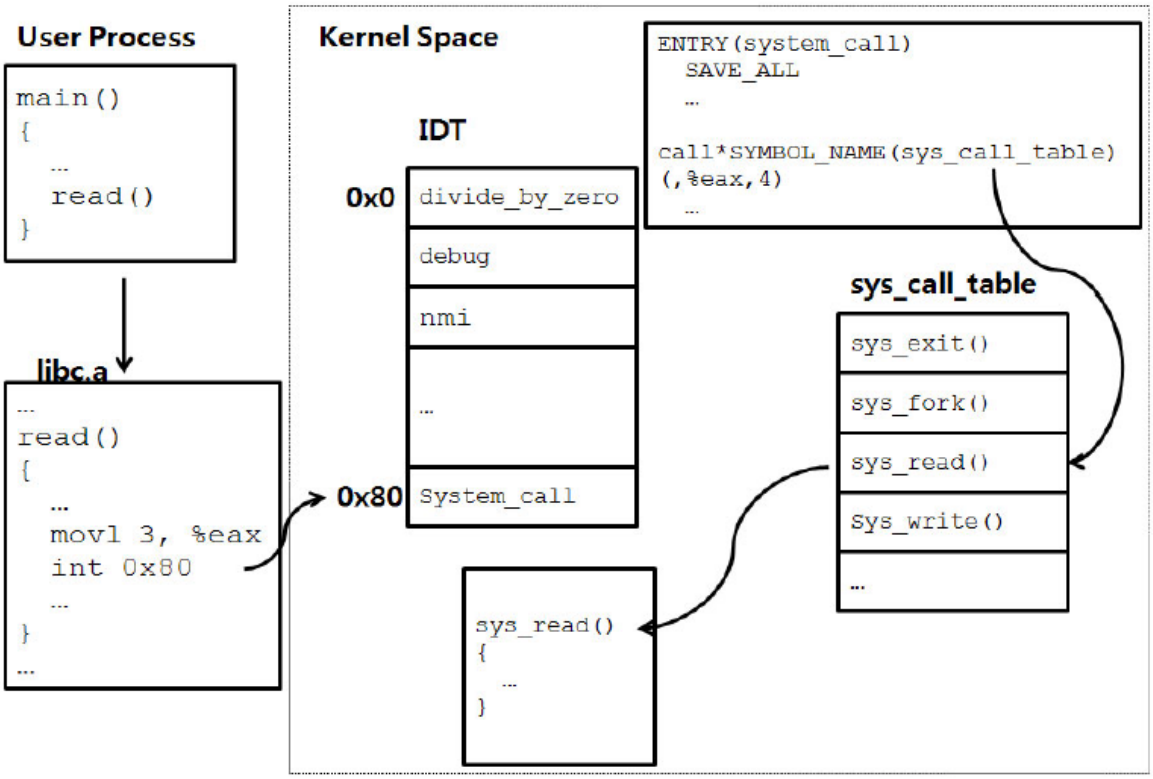
   A. syscall 함수를 이용해서 새로 작성한 System Call을 호출 하도록 한다.

   B. System Call의 인자로 정수형 숫자를 넘겨준다.

3. 세부사항

A. System Call

앞에서도 간략하게 설명했지만 System Call이란 리눅스 커널에 의해 제공되는 Kernel-level 서비스이다. 예를 들어 프로그래머가 파일을 읽는 서비스를 이용하고 싶다면 그에 해당하는 System Call을 이용해야 한다. C를 이용해서 프로그래밍할 경우 대부분의 System Call은 libc를 통한 Wrapper Function 형태로 제공받을 수 있다. 예를 들어 User Application에서 파일을 읽는 read 함수가 호출 되었다고 하면 다음 그림과 같은 호출이 일어나게 된다.



그림과 같이 User Process에서 read를 호출하면 Wrapper Function을 통해서 0x80의 Trap Instruction이 발생한다. 0x80 번째 Trap은 System Call로 예약이 되어 있고 이 Trap Handler는 Sys_call_table의 번호를 확인해서 3번째 Table에 저장되어 있는 sys_read를 호출 하게 된다. 이렇게 커널에서의 서비스가 끝나면 다시 User Process로 결과값과 함께 return하게 된다. 이번 과제에서는 sys_call_table에 1개의 Entry를 더 추가해서 sys_print_number 을 User Process에서 호출하고 dmesg를 통해서 prink로 출력한 메시지를 확인하도록 한다.

B. Kernel에서 문자열 출력 (printk 함수)

Kernel 영역에서는 기존에 사용하던 printf를 사용하지 못한다. 커널 영역에서 메시지를 출력하려면 printk를 사용해야 하며, 함수의 사용법은 기본적으로 printf와 같다. printk는 특별히 중요도를 지정해서 출력할 수도 있지만 이번 과제에서는 사용하지 않도록 한다. System Call에서 printk를 통해 메시지를 출력했다면 일반 Terminal 화면으로는 볼 수 없고, 최근에 발생한 커널 메시지를 확인하는 dmesg 명령어를 통해서 확인할 수 있다

4. 커널에서 수정해야 하는 주요 파일 목록

A. (linux)/arch/x86/entry/syscalls/syscall_64.tbl (64비트 가상머신)

(linux)/arch/x86/entry/syscalls/syscall_32.tbl (32비트 가상머신)

→ 시스템 콜 함수들의 이름에 대한 심볼 정보를 모아 놓은 파일

B. (linux)/include/linux/syscalls.h → 추가한 시스템 콜 함수들의 prototype 정의

C. (linux)/kernel/my_queue_syscall.c → 새로 추가할 시스템 콜의 소스

D. (linux)/kernel/Makefile → my_queue_syscall.o 오브젝트 추가

5. 추가한 System Call을 호출하는 User Application

System Call도 함수로 되어 있고 User Application에서 호출해야 실행된다. System Call을 호출하는 방법에도 여러 가지가 있는데 그 중 하나가 syscall 매크로를 사용하는 것이다. syscall 매크로는 인자의 개수에 따라서 syscall0 ~ syscall6까지 있는데 그 세부 구현은 (4-A 파일)에서 확인할 수 있다. syscall을 사용하는 방법은 syscall(시스템콜 번호, args...) 이다.

User application은 기본적으로 사용자로부터 integer 값을 계속해서 입력 받으며, -1을 입력 받을 경우 프로그램이 종료된다. Integer 값을 입력 받을 때마다 추가한 시스템 콜을 호출한다. 이때, 인자는 입력 받은 integer 값이다. 단, -1이 입력될 경우 시스템 콜을 호출하지 않는다.

6. 과제 결과 출력 형식

```
ohsang1213@ohsang1213-VirtualBox:~/Test$ ./test
Input an integer number (-1 for exit): 1
Input an integer number (-1 for exit): 5
Input an integer number (-1 for exit): 10
Input an integer number (-1 for exit): 20
Input an integer number (-1 for exit): 100
Input an integer number (-1 for exit): -1
ohsang1213@ohsang1213-VirtualBox:~/Test$
```

그림 1. User program 입력 및 출력

```
[   13.493075] 10:47:18.780727 automount vbsvcAutomounterMo
on '/media/sf_Shared'
[   19.248858] ISO 9660 Extensions: Microsoft Joliet Level
[   19.264919] ISO 9660 Extensions: RRIP_1991A
[   19.629548] rfkill: input handler disabled
[   38.093007] Hello, kernel! num = 1
[   39.025011] Hello, kernel! num = 5
[   40.355828] Hello, kernel! num = 10
[   41.467577] Hello, kernel! num = 20
[   42.498124] Hello, kernel! num = 100
ohsang1213@ohsang1213-VirtualBox:~/Test$
```

그림 2. 커널 로그 출력 (dmesg)