ICT332 Project #1

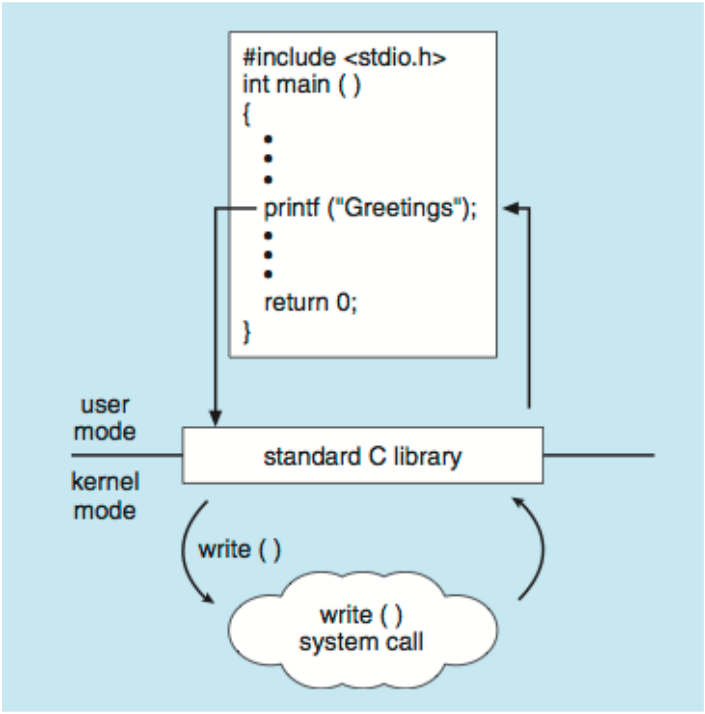
Instructor: Sangeun Oh

Project Goal

- Linux의 소스코드를 수정하고 컴파일하여 새로운 system call을 추가
- 이렇게 추가된 system call을 사용하는 user program 제작

System call

- User mode에서 kernel mode로 진입하기 위한 통로
  - 커널에서 제공하는 서비스를 이용하기 위해 필요
- User program은 보통 직접 system call을 호출하기 보단 high-level API를 이용
  - 사용자에게 편리한 interface 제공



TODO

- 시스템 콜 추가
  - 시스템 콜 코드를 통해 입력 받은 숫자를 출력하는 기능 구현
- 새 시스템 콜을 이용한 user program 작성
  - 추가된 시스템 콜을 호출하여 커널에게 어떤 임의의 숫자를 출력하도록 요청함
  - 터미널에서 dmesg 명령어로 커널이 출력하는 숫자를 확인할 수 있음
- 실행 결과

User program 입력 및 출력

```
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$
```

커널 로그 출력 (dmesg)

```
[    13.476315] 10:47:18.764014 main     6.1.16 r140961 started. Verbo
[    13.477458] 10:47:18.765115 main     vbglR3GuestCtrlDetectPeekGetC
)
[    13.490379] vboxsf: g_fHostFeatures=0x8000000f g_fSfFeatures=0x1 g
[    13.490629] vboxsf: Successfully loaded version 6.1.16
[    13.490672] vboxsf: Successfully loaded version 6.1.16 (LINUX_VERS
[    13.493075] 10:47:18.780727 automount vbsvcAutomounterMountIt: Suc
on '/media/sf_Shared'
[    19.248858] ISO 9660 Extensions: Microsoft Joliet Level 3
[    19.264919] ISO 9660 Extensions: RRIP_1991A
[    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$
```

커널 코드 수정

- syscall_64.tbl
  - 시스템 콜 함수들의 이름에 대한 심볼 정보를 모아 놓은 파일
  - 새로 추가할 시스템 콜 번호
- syscalls.h
  - 추가한 시스템 콜 함수들의 prototype 정의 및 테이블 등록
- my_syscall.c
  - /usr/src/linux-5.9.1/kernel/ 하위에 해당 파일을 추가
  - 새로 추가할 시스템 콜의 소스
- Makefile
  - /usr/src/linux-5.9.1/kernel/Makefile 수정
  - My_syscall.o 오브젝트 추가

syscall_64.tbl

- 리눅스에서 제공하는 모든 시스템 콜의 고유 번호를 저장
  - (linux)/arch/x86/entry/syscalls/syscall_64.tbl
    - 단, 32비트 가상 머신은 (linux)/arch/x86/entry/syscalls/syscall_32.tbl
    - ※ (linux) 는/usr/src/linux-5.9.1
- 시스템 콜의 symbol 정보 집합
  - 링커에 의해 관리되는 정보
    - Linux kernel source tree에 흩어져 있는 시스템 콜 함수의 주소들을 저장하는 테이블
  - 시스템 콜 주소는 링커가 자동으로 관리

syscall_64.tbl

```
# don't use numbers 387 through 423, add new calls after the last
# 'common' entry
424     common  pidfd_send_signal       sys_pidfd_send_signal
425     common  io_uring_setup          sys_io_uring_setup
426     common  io_uring_enter          sys_io_uring_enter
427     common  io_uring_register       sys_io_uring_register
428     common  open_tree               sys_open_tree
429     common  move_mount              sys_move_mount
430     common  fsopen                  sys_fsopen
431     common  fsconfig                sys_fsconfig
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  print_number            sys_print_number

#
# x32-specific system call numbers start at 512 to avoid cache impact
# for native 64-bit operation. The __x32_compat_sys stubs are created
# on-the-fly for compat_sys_*() compatibility system calls if X86_X32
# is defined.
#
512     x32     rt_sigaction            compat_sys_rt_sigaction
513     x32     rt_sigreturn            compat_sys_x32_rt_sigreturn
514     x32     ioctl                   compat_sys_ioctl
515     x32     readv                   compat_sys_readv
```

syscalls.h

- 시스템 콜 함수들의 prototype을 정의
  - (linux)/include/linux/syscalls.h 파일에 등록
  - asmlinkage void sys_print_number(int num)
- 왜 asmlinkage를 사용하는가?
  - 시스템 콜 호출은 int 80 인터럽트 핸들러에서 호출
  - 인터럽트 핸들러는 assembly 코드로 작성됨
  - asmlinkage를 함수 앞에 선언하면 assembly code에서도 C 함수 호출이 가능해짐

```
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 void sys_print_number(int num);
#endif
```

my_syscall.c

- 추가할 시스템 콜 소스
  - 시스템 콜이 실제로 할 일을 구현
    - /usr/src/linux-5.9.1/kernel/ 하위에 my_syscall.c 파일을 생성하고 작성
  - asmlinkage void sys_print_number(int num) 함수를 구현
    - Hint: 함수 구현 방법은 /usr/src/linux-5.9.1/kernel/ 하위에 있는 다른 시스템 콜 소스를 참고해 볼 것
    - Hint: SYSCALL_DEFINE0, SYSCALL_DEFINE1, …. 라는 키워드 찾아볼 것
- 헤더 파일 추가 (include)
  - <linux/linkage.h>
  - <linux/kernel.h>
  - <linux/syscalls.h>

Makefile

- /usr/src/linux-5.9.11/kernel/Makefile
- kernel make 시에 포함되도록 obj-y 부분에 추가
- .o 오브젝트 파일명은 자신의 c 파일 이름과 동일
  - 예) my_syscall.c ➔ my_syscall.o

```
# 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 my_syscall.o
```

커널 컴파일

- 위 내용들을 모두 완료하였으면 /usr/src/linux-5.9.1/ 에서
  - sudo make
    - -j 옵션 사용 가능
  - sudo make install
- 만약 안되면 make clean하고 다시 빌드할 것

User program 작성

- 추가한 시스템 콜을 사용하는 user program 작성
  - syscall() 이라는 매크로 함수를 이용하여 시스템 콜 호출
    - <unistd.h> 헤더 추가할 것
    - 사용법: 시스템 콜 번호와 인자를 넣어서 사용
      - » syscall(440, …);
      - » #define PRINT_NUMBER 440 // 시스템 콜 번호 선언 후에 syscall(PRINT_NUMBER, …); 으로 사용하면 더 편리
- 기본적으로 사용자로부터 integer 값을 계속해서 입력 받아야 함
  - -1을 입력 받으면 프로그램 종료
- Integer 값을 입력 받을 때마다 추가한 시스템 콜 호출
  - 이때, 인자는 입력 받은 integer 값
  - -1이 입력될 경우 시스템 콜을 호출하지 않고 프로그램 종료

User program 컴파일

- User program 소스 파일이 test.c라면
  - gcc test.c –o test
    - “test.c를 컴파일해서 test라는 이름의 실행파일을 만들어라”
  - ./test로실행 후, dmesg를 통해서 my_syscall.c의 printk로 원하던 출력이 나왔는지를 확인

과제 제출

1. 직접 작성하거나 수정한 소스 파일 제출
   - 수정하거나 추가한 커널 소스코드
     - 리눅스 소스 코드 전체 제출이 아님
   - User program 코드

2. dmesg로 확인한 시스템 콜 결과와 user program 입력에 대한 스크린샷 제출

```
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$
```

```
[   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$
```

3. 커널 로그 제출
   - dmesg > kernel.log 를 터미널에 입력
   - kernel.log 파일 제출

과제 제출

- 제출 기한: 2021. 06. 18. 금요일 23시 59분
  - 제출 파일들을 zip 압축파일로 압축하여 제출할 것
  - 파일 이름은 “Proj01_학번_이름.zip”
  - Ajou bb 프로젝트 제출란에 업로드