시스템 콜

목표

- Raspberry Pi에 자신이 제작한 System Call 함수를 커널에 빌드
- 자신이 제작한 System Call 함수를 직접 사용해보기

준비물

- 호스트 시스템에서 타겟 시스템의 Kernel 소스코드*(4.19.x 이상 버전 권장)*
  *호스트 시스템에 타겟 시스템의 Kernel 소스코드가 없다면 앞선 *"라즈베리파이 커널 컴파일"* 실습 내용 참조
- 호스트 시스템에서 타겟 시스템의 툴 체인 환경(Cross Compiler, Make 등)
  *호스트 시스템에 타겟 시스템의 툴 체인 환경이 없다면 앞선 *"라즈베리파이 커널 컴파일"* 실습 내용 참조

시스템 콜이란?

- 간단히 시스템 콜은 운영 체제의 커널이 제공하는 서비스에 대해, 응용 프로그램의 요청에 따라 커널에 접근하기 위한 인터페이스이다. 보통 고급 언어로 작성된 프로그램들은 직접 시스템을 사용할 수 없기 때문에 시스템 콜과 같은 API를 통해 시스템에 접근하게 하는 방법을 사용한다.

타겟 시스템의 커널에 자신의 시스템 콜 빌드

- *본 실습에서는 타겟 시스템 커널의 버전을 5.10.x로 사용하였다.
  하지만 이번 실습을 따라하기 위해서 본 실습의 타겟 시스템의 커널 버전을 반드시 동일하게 맞출 필요는 없다.
  단지 호스트 시스템에 타겟 시스템의 커널 소스만 존재하면 된다.
- 먼저 호스트 시스템에 존재하는 타겟 시스템의 커널 소스를 변경하는 것을 진행한다.
  또한, 실습을 설명하기 위해 [kernel]이라는 파일 경로를 사용한다.
  [kernel]은 타겟 시스템의 커널 소스 시작 디렉터리를 의미한다.
- 본 실습의 3.1. 까지는 호스트 시스템에서 진행되는 것이며, 3.2. 부턴 타겟 시스템에서 진행된다.

1. 시스템 콜 헤더(함수 원형) 추가

1.1. 파일 경로 : [kernel]/arch/arm/include/generated/uapi/asm/unistd-common.h

```
#ifndef _UAPI_ASM_ARM_UNISTD_COMMON_H
#define _UAPI_ASM_ARM_UNISTD_COMMON_H 1

#define __NR_restart_syscall (__NR_SYSCALL_BASE + 0)
#define __NR_exit (__NR_SYSCALL_BASE + 1)
#define __NR_fork (__NR_SYSCALL_BASE + 2)
#define __NR_read (__NR_SYSCALL_BASE + 3)
#define __NR_write (__NR_SYSCALL_BASE + 4)
                    •
                    •
                    •
#define __NR_pidfd_open (__NR_SYSCALL_BASE + 434)
#define __NR_clone3 (__NR_SYSCALL_BASE + 435)
#define __NR_close_range (__NR_SYSCALL_BASE + 436)
#define __NR_openat2 (__NR_SYSCALL_BASE + 437)
#define __NR_pidfd_getfd (__NR_SYSCALL_BASE + 438)
#define __NR_faccessat2 (__NR_SYSCALL_BASE + 439)
#define __NR_process_madvise (__NR_SYSCALL_BASE + 440)

#define __NR_mysyscall (__NR_SYSCALL_BASE + 441)

#endif /* _UAPI_ASM_ARM_UNISTD_COMMON_H */
```

>> 파일의 가장 끝 줄에 자신의 시스템 콜 추가 – 형식 : __NR_[시스템콜] (__NR_SYSCALL_BASE + N)

vi 에디터에서 파일의 가장 끝 줄로 한번에 갈려면 명령모드에서 대문자 G를 입력

>> 커널 버전마다 뒤의 숫자는 달라짐으로 새로운 시스템 콜의 번호는 기존의 마지막 번호보다 +1 높은 수로 설정

>> 본 실습에서는 기존에 440이라는 값이 존재함으로 +1하여 441의 번호로 새로운 시스템 콜 번호를 부여

1.2. 파일 경로 : [kernel]/arch/arm/include/generated/asm/unistd-nr.h

```
#ifndef _ASM_ARM_UNISTD_NR_H
#define _ASM_ARM_UNISTD_NR_H 1

/*
 * This needs to be greater than __NR_last_syscall+1 in order to account
 * for the padding in the syscall table.
 */

/* aligned to 4 */
#define __NR_syscalls 444

#endif /* _ASM_ARM_UNISTD_NR_H */
```

>> 앞서 시스템 콜을 추가하였을 때의 번호가 해당 파일의 __NR_syscalls 번호보다 클 경우 값을 증가시켜 주기

>> 값을 증가 시켜줄 때는 **반드시 +4씩** 증가 시켜주기

Ex) 번호 441의 시스템 콜을 추가하였을 때 __NR_syscalls 440일 경우, __NR_syscalls 444로 증가

1.3. 파일 경로 : [kernel]/arch/arm/include/generated/calls-eabi.S

```
NATIVE(0, sys_restart_syscall)
NATIVE(1, sys_exit)
NATIVE(2, sys_fork)
NATIVE(3, sys_read)
NATIVE(4, sys_write)
                •
                •
                •
NATIVE(435, sys_clone3)
NATIVE(436, sys_close_range)
NATIVE(437, sys_openat2)
NATIVE(438, sys_pidfd_getfd)
NATIVE(439, sys_faccessat2)
NATIVE(440, sys_process_madvise)

NATIVE(441, sys_mysyscall)
```

>> 파일의 가장 끝 줄에 자신의 시스템 콜 번호와 호출할 때 사용할 함수 원형 이름 등록

vi 에디터에서 파일의 가장 끝 줄로 한번에 갈려면 명령모드에서 대문자 G를 입력

>> 여기에 등록된 함수 원형 이름을 통해서 유저 프로그램은 시스템 콜을 호출한다.

1.4. 파일 경로 : [kernel]/include/linux/syscalls.h

```
/* SPDX-License-Identifier: GPL-2.0-only */
/*
 * syscalls.h - Linux syscall interfaces (non-arch-specific)
 *
 * Copyright (c) 2004 Randy Dunlap
 * Copyright (c) 2004 Open Source Development Labs
 */

#ifndef _LINUX_SYSCALLS_H
#define _LINUX_SYSCALLS_H

struct __aio_sigset;
struct epoll_event;
struct iattr;
                •
                •
                •
/* obsolete: mm/ */
asmlinkage long sys_mmap_pgoff(unsigned long addr, unsigned long len,
                        unsigned long prot, unsigned long flags,
                        unsigned long fd, unsigned long pgoff);
asmlinkage long sys_old_mmap(struct mmap_arg_struct __user *arg);


/*
 * Not a real system call, but a placeholder for syscalls which are
 * not implemented -- see kernel/sys_ni.c
 */
asmlinkage long sys_ni_syscall(void);

asmlinkage long sys_mysyscall(void);

#endif /* CONFIG_ARCH_HAS_SYSCALL_WRAPPER */
                •
                •
                •
```

>> sys_ni_syscall() 함수 원형 밑에 자신의 시스템 콜 함수 원형 등록

vi 에디터의 라인 모드를 통해 문자열 검색기능을 사용하면 한번에 sys_ni_syscall()을 찾을 수 있다.

>> 앞서 1.3.에서 등록한 함수 원형 이름과 동일한 이름으로 등록

2. 시스템 콜 함수 작성 및 커널 빌드

2.1. 파일 경로 : [kernel]/kernel/mysyscall.c

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

asmlinkage int sys_mysyscall(void){
        printk("[mysyscall] Hello, World!!\n");
        return 0;
}
```

>> 유저 프로그램에서 본 실습에서 등록한 시스템 콜을 호출 시 시스템 로그가 찍히도록 하였다.

2.2. 파일 경로 : [kernel]/kernel/Makefile

```
# 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 mysyscall.o
```

•
•
•

>> Makefile의 obj-y의 가장 끝에 앞서 등록한 mysyscall.c의 오브젝트 파일이
>> 생성될 수 있도록 mysyscall.o를 작성해준다.

2.3. 파일 경로 : [kernel]/

```
inslab-test-server@computer:~/git/5.10_rpi_kernel/linux$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- zImage
  SYNC    include/generated/autoconf.h
  CC      arch/arm/kernel/asm-offsets.s
  CALL    scripts/checksyscalls.sh
  CALL    scripts/atomic/check-atomics.sh
  CC      init/main.o
  CHK     include/generated/compile.h
  CC      init/version.o
  CC      init/do_mounts.o
  CC      init/do_mounts_rd.o
  CC      init/do_mounts_initrd.o
  CC      init/initramfs.o
  CC      init/init_task.o
  AR      init/built-in.a
  CC      arch/arm/vfp/vfpmodule.o
  AR      arch/arm/vfp/built-in.a
  CC      arch/arm/vdso/vgettimeofday.o
  LD      arch/arm/vdso/vdso.so.raw
  MUNGE   arch/arm/vdso/vdso.so.dbg
  OBJCOPY arch/arm/vdso/vdso.so
  AS      arch/arm/vdso/vdso.o
  AR      arch/arm/vdso/built-in.a
  CC      arch/arm/kernel/elf.o
  AS      arch/arm/kernel/entry-common.o
  CC      arch/arm/kernel/irq.o
  CC      arch/arm/kernel/opcodes.o
  CC      arch/arm/kernel/process.o
  CC      arch/arm/kernel/ptrace.o
```

>> 시스템 콜 헤더(함수 원형)와 코드를 추가하였으면 타겟 시스템의 커널 이미지(zImage)를 생성 시켜준다.
>> 명령어 : make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- -j$(nproc) zImage

2.4. 파일 경로 : [kernel]/kernel/

```
insla-test-server@computer:~/git/5.10_rpi_kernel/linux$ ls kernel
Kconfig.freezer   configs.ko          groups.c          module-internal.h
Kconfig.hz        configs.mod         groups.o          module.c
Kconfig.locks     configs.mod.c       hung_task.c       module.o
Kconfig.preempt   configs.mod.o       hung_task.o       module_signature.c
Makefile          configs.o           iomem.c           module_signing.c
acct.c            context_tracking.c  iomem.o           modules.order
acct.o            cpu.c               irq               mysyscall.c
async.c           cpu.o               irq_work.c        mysyscall.o
async.o           cpu_pm.c            irq_work.o        notifier.c
audit.c           crash_core.c        jump_label.c      notifier.o
audit.h           crash_dump.c        jump_label.o      nsproxy.c
audit.o           cred.c              kallsyms.c        nsproxy.o
```

>> 앞서 작성한 소스코드 파일(.c)의 오브젝트 파일(.o)이 생성되어 있다면

>> 커널에 자신이 생성한 시스템 콜 함수가 정상적으로 빌드가 된 것 이다.

3. 생성된 커널 이미지를 타겟 시스템에 적용

3.1. 파일 경로 : [kernel]/arch/arm/boot/zImage

```
insla-test-server@computer:~/git/5.10_rpi_kernel/linux$ scp arch/arm/boot/zImage pi@192.168.1.56:/home/pi
pi@192.168.1.56's password:
zImage                                        100% 6632KB   3.9MB/s   00:01
insla-test-server@computer:~/git/5.10_rpi_kernel/linux$
```

>> 생성된 커널 이미지(zImage)를 타겟 시스템으로 이동

>> 본 실습에서는 scp 명령어를 사용하여 이동하였다.

>> scp 명령어 형식 : scp [전송할 파일] [타겟 호스트 이름]@[IP]:/[파일이 전달될 타겟 시스템의 경로]

Ex) scp arch/arm/boot/zImage pi@192.168.1.56:/home/pi

❖ 3.2. 부턴 타겟 시스템(라즈베리파이) 환경에서 실습이 진행된다.

3.2. 옮겨진 커널 이미지 파일를 타겟 시스템에 적용

```
pi@raspberrypi:~ $ ls
zImage
pi@raspberrypi:~ $ sudo mv zImage /boot/kernel71.img
mv: failed to preserve ownership for '/boot/kernel71.img': Operation not permitted
pi@raspberrypi:~ $
```

>> 본 실습에서는 커널 이미지를 호스트 시스템에서 타겟 시스템의 홈 디렉터리인 /home/pi로 옮겼다.

>> 해당 커널 이미지를 라즈베리파이 타겟 시스템의 커널 이미지로 변경해주었다.

명령어 : sudo mv zImage /boot/kenrel7l.img (라즈베리파이 4B 이상)

명령어 : sudo mv zImage /boot/kernel7.img (라즈베리파이 3B+ 이하)

타겟 시스템에서 자신의 시스템 콜 사용

- 앞선 단락에서 3.2.까지 진행하였다면 타겟 시스템(라즈베리파이)의 커널에 시스템 콜이 등록이 완료된 것이다. 이제 커널에 등록된 시스템 콜을 유저 프로그램에서 호출해보는 것을 진행해본다.

4. 타겟 시스템에서 시스템 콜을 사용하는 유저 프로그램 생성

4.1. 소스코드 생성

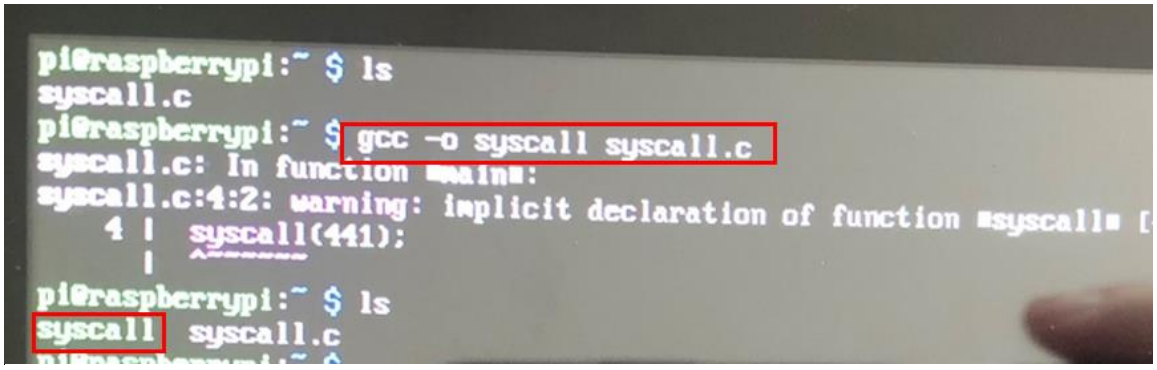
```
#include <sys/syscall.h>

int main() {
        syscall(441);
        return 0;
}
```

>> 앞서 등록한 시스템 콜 번호를 syscall() 함수에 입력

>> 본 실습에서 해당 소스코드 파일의 이름은 syscall.c로 하였다.

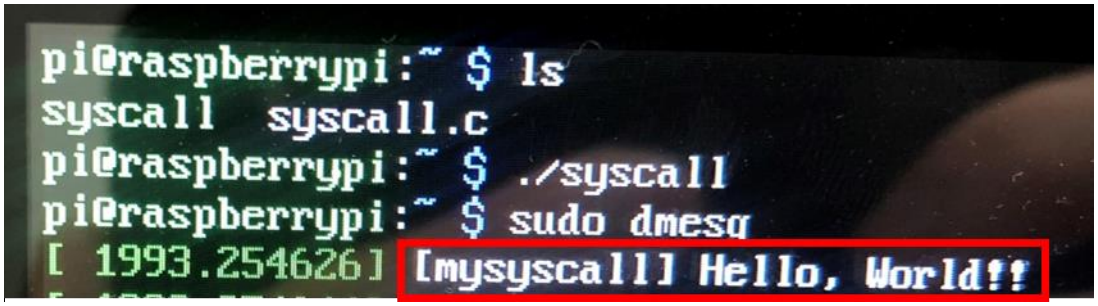
4.2. 소스코드 컴파일



>> gcc 컴파일러를 통해 위 소스코드(syscall.c)를 컴파일

>> 명령어 : gcc -o syscall syscall.c

4.3. 컴파일 된 실행파일의 실행 결과 확인



>> 생성된 syscall 파일을 실행하였을 시 앞서 등록한 시스템 콜의 코드대로 정상 작동하는 것을 볼 수 있다.