

# 운영체제 실습2

2021320314 고건영

## 시스템콜 작동 과정 분석

### 1. 사용자 공간에서의 호출

- 목적

복잡한 호출 과정은 라이브러리가 담당하고 사용자는 편리하게 함수를 사용할 수 있게 한다.

- 과정

- a. 일반적 사용자 프로그래밍 과정에서 **라이브러리 함수 호출**(e.g. printf())
- b. 내부적으로 동작하는 glibc의 **라이브러리 래퍼 함수 호출**(e.g. write() 래퍼 함수)
- c. 래퍼 함수가 어떤 시스템 콜인지 식별 번호를 설정하고 **시스템 콜 번호와 인자들(파일 디스크립터, 버퍼 주소 등)을 레지스터에 준비**
- d. **syscall(유저 모드 → 커널 모드 전환 전용 cpu 명령어) 실행**을 통해 커널에 요청

### 2. 커널 모드로 전환

- 목적

유저 영역에서 커널 영역으로 안전하게 권한 변경을 하기 위함이다.

- 과정

- a. **syscall 이 실행**되면 CPU는 **유저 모드에서 커널 모드로 전환**하고 미리 정해진 커널의 진입점부터 실행 시작
- b. **사용자 프로그램의 실행 상태(context)를 저장**

### 3. 커널 내부 처리

- 목적

커널에서 실제로 유저 영역에서 요청한 시스템 콜을 처리하기 위함이다.

- 과정
  - a. 커널은 **시스템 콜 테이블**을 참조해 시스템 콜 번에 해당하는 **커널 함수(핸들러)** 실행
  - b. 핸들러 함수는 **인자의 유효성 검사** 후 요청된 시스템 콜의 **실제 작업을 수행**(e.g. write는 파일 쓰기 로직)

#### 4. 사용자 공간으로의 복귀

- 목적

안전하게 커널 영역에서 유저 영역으로 복귀한다. 이로써 유저는 커널 영역의 처리를 고려하지 않고 작업을 이어서 처리할 수 있다.
- 과정
  - a. 핸들러가 종료되고 **결과값(성공/실패 정보)** 레지스터에 저장
  - b. 이전에 저장했던 **사용자 프로그램 상태(context)**를 복원
  - c. 커널에서 **sysret(커널 모드 → 유저 모드 전환 전용 cpu명령어)**를 실행
  - d. sysret이 실행되면 CPU는 커널 모드에서 **유저 모드로 전환**
  - e. 사용자 프로그램은 **syscall**을 호출했던 다음 지점부터 실행을 재개
  - f. 유저 공간(라이브러리 래퍼 함수)에서 **커널이 전달한 결과값**을 처리/확인

## 시스템콜 생성 및 커널 컴파일 연습

- 시스템 콜 테이블에 새로운 시스템 콜 등록한 스크린샷

```

GNU nano 7.2 /usr/src/linux-source-6.8.0/arch/x86/entry/syscalls/syscall_64.tbl
447 common memfd_secret sys_memfd_secret
448 common process_mrelease sys_process_mrelease
449 common futex_waitv sys_futex_waitv
450 common set_mempolicy_home_node sys_set_mempolicy_home_node
451 common cachestat sys_cachestat
452 common fchmodat2 sys_fchmodat2
453 64 map_shadow_stack sys_map_shadow_stack
454 common futex_wake sys_futex_wake
455 common futex_wait sys_futex_wait
456 common futex_requeue sys_futex_requeue
457 common statmount sys_statmount
458 common listmount sys_listmount
459 common lsm_get_self_attr sys_lsm_get_self_attr
460 common lsm_set_self_attr sys_lsm_set_self_attr
461 common lsm_list_modules sys_lsm_list_modules
462 common print_student_name sys_print_student_name
463 common print_student_id sys_print_student_id
464 common print_student_info sys_print_student_info
#
# Due to a historical design error, certain syscalls are numbered differently
# in x32 as compared to native x86_64. These syscalls have numbers 512-547.
# Do not add new syscalls to this range. Numbers 548 and above are available
# for non-x32 use.
#
512 x32 rt_sigaction compat_sys_rt_sigaction
513 x32 rt_sigreturn compat_sys_x32_rt_sigreturn
514 x32 ioctl compat_sys_ioctl

```

시스템콜 테이블(syscall\_64.tbl)에 다음과 같은 3개의 시스템 콜을 등록하였다. sys\_가 붙지 않은 것은 시스템 콜의 이름이고 뒤의 sys\_ 가 붙은 것은 실제 핸들러 함수 이름이다.

- 462번 - print\_student\_name , sys\_print\_student\_name
- 463번 - print\_student\_id , sys\_print\_student\_id
- 464번 - print\_student\_info , sys\_print\_student\_info

커널은 사용자 프로그램이 넘겨준 시스템콜 식별 번호를 통해 시스템콜 테이블에 매핑된 함수중에서 대응되는 핸들러 함수를 찾아 실행한다.

이렇게 새로운 시스템콜을 테이블에 등록해서 직접 만든 함수를 커널이 호출할 수 있게 된다.

- **syscall.h에 새로운 시스템 콜을 등록한 스크린샷**

```

GNU nano 7.2 /usr/src/linux-source-6.8.0/include/linux/syscalls.h
/* for __ARCH_WANT_SYS_IPC */
long ksys_semtimedop(int semid, struct sembuf __user *tsops,
                    unsigned int nsops,
                    const struct __kernel_timespec __user *timeout);
long ksys_semget(key_t key, int nsems, int semflg);
long ksys_old_semctl(int semid, int semnum, int cmd, unsigned long arg);
long ksys_msgsget(key_t key, int msgflg);
long ksys_old_msgctl(int msqid, int cmd, struct msqid_ds __user *buf);
long ksys_msgrcv(int msqid, struct msgbuf __user *msgp, size_t msgsz,
                long msgtyp, int msgflg);
long ksys_msgsnd(int msqid, struct msgbuf __user *msgp, size_t msgsz,
                int msgflg);
long ksys_shmget(key_t key, size_t size, int shmflg);
long ksys_shmdt(char __user *shmaddr);
long ksys_old_shmctl(int shmid, int cmd, struct shmid_ds __user *buf);
long compat_ksys_semtimedop(int semid, struct sembuf __user *tsems,
                            unsigned int nsops,
                            const struct old_timespec32 __user *timeout);
long __do_semtimedop(int semid, struct sembuf *tsems, unsigned int nsops,
                    const struct timespec64 *timeout,
                    struct ipc_namespace *ns);

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 long sys_print_student_name(char __user *name);
asmlinkage long sys_print_student_id(char __user *id);
asmlinkage long sys_print_student_info(char __user *school, char __user *major);
#endif

```

syscall.h는 커널 빌드 시에 컴파일러가 시스템 콜 함수를 인식하게 해주는 선언용 헤더이다.  
여기에 프로토타입 함수 sys\_print\_student\_name , sys\_print\_student\_id ,  
sys\_print\_student\_info 를 등록했다.  
컴파일 할때 핸들러 함수들이 선언돼있어야 테이블에 매핑을 할 수 있다.

- Makefile에 새 시스템콜 함수 파일을 추가한 스크린샷

```

GNU nano 7.2 /usr/src/linux-source-6.8.0/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 reset.o ksyms_common.o new_syscall.o_

obj-$(CONFIG_USERMODE_DRIVER) += usermode_driver.o
obj-$(CONFIG_MULTIUSER) += groups.o
obj-$(CONFIG_VHOST_TASK) += vhost_task.o

ifdef CONFIG_FUNCTION_TRACER
# Do not trace internal ftrace files
CFLAGS_REMOVE_irq_work.o = $(CC_FLAGS_FTRACE)
endif

```

Makefile에 명시된 오브젝트 파일은 커널 빌드 대상이 되는데 여기에 new\_syscall.o를 추가  
해서 new\_syscall.c가 빌드 될 수 있도록 했다.

- 작성한 new\_syscall.c 스크린샷

```

OS-tutorial [실행 중] - Oracle VM VirtualBox
파일  편집  보기  입력  장치  도움말
GNU nano 7.2 new_syscall.c
#include <linux/errno.h>
#include <linux/kernel.h>
#include <linux/syscalls.h>
#include <linux/uaccess.h>

SYSCALL_DEFINE1(print_student_name, char __user *, name)
{
    char buf[256];
    long ret;

    ret = strncpy_from_user(buf, name, sizeof(buf));
    if (ret < 0) {
        return -EFAULT;
    }
    if (ret == sizeof(buf)) {
        buf[sizeof(buf) - 1] = '\0';
    }

    printk(KERN_INFO "My Name is %s\n", buf);
    return 0;
}

SYSCALL_DEFINE1(print_student_id, char __user *, id)
{
    char buf[256];
    long ret = strncpy_from_user(buf, id, sizeof(buf));

    if (ret < 0) {
        return -EFAULT;
    }
    buf[sizeof(buf) - 1] = '\0';

    printk(KERN_INFO "My Student ID is %s\n", buf);
    return 0;
}

```

```

SYSCALL_DEFINE2(print_student_info, char __user *, school, char __user *, major)
{
    char school_buf[256];
    char major_buf[256];
    long ret;

    ret = strncpy_from_user(school_buf, school, sizeof(school_buf));
    if (ret < 0) {
        return -EFAULT;
    }
    school_buf[sizeof(school_buf) - 1] = '\0';

    ret = strncpy_from_user(major_buf, major, sizeof(major_buf));
    if (ret < 0) {
        return -EFAULT;
    }
    major_buf[sizeof(major_buf) - 1] = '\0';

    printk(KERN_INFO "I go to %s\n", school_buf);
    printk(KERN_INFO "I major in %s\n", major_buf);
    return 0;
}

```

SYSCALL\_DEFINE은 리눅스 커널에서 핸들러 함수를 정의할 때 쓰는 매크로이다. 뒤에 붙는 숫자는 인자의 개수를 의미한다. 이를 통해 3가지의 핸들러 함수를 구현했다.

- print\_student\_name

char \_\_user \*, name : 사용자 공간에서 문자열 포인터 name을 인자로 받음

ret = strncpy\_from\_user(buf, name, sizeof(buf)); : 전달받은 인자를 사용자 공간에서 커널 공간 버퍼 buf로 복사

복사 실패(ret < 0): -EFAULT 리턴

문자열이 길어 버퍼가 꽉 찼을 경우(ret == sizeof(buf)): buf[sizeof(buf) - 1] = '\0' 로 복사된 문자열 끝에 null문자를 넣어 꽉 찬 버퍼를 출력할 때의 오류를 방지

printf()로 커널 로그에 버퍼의 내용과 함께 My Name is ~~출력

printf()는 커널 내부 함수로써 커널 로그 버퍼에 메시지를 기록해서 dmesg로 볼 수 있게 해줌

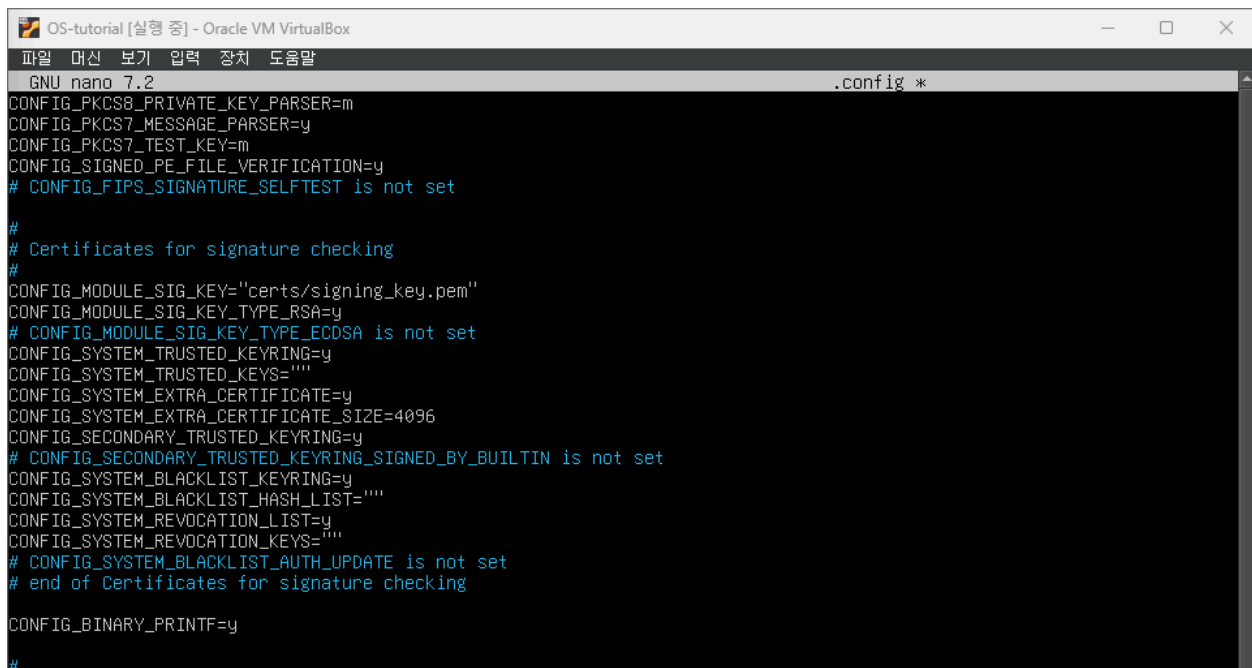
- **print\_student\_id**

포인터 id를 인자로 받는것과, 출력문이 My Student ID is ~~ 로 바뀐것을 제외하고 print\_student\_name과 유사

- **print\_student\_info**

school과 major 2개의 문자열 포인터를 받고 각각 ret을 사용하여 복사 실패 여부만 검증하고 버퍼가 꽉 찼는지 검증 없이 무조건 뒤에 null문자를 추가하는 것, printf를 두번 사용해, I go to ~~ 와 I major in ~~ 두 문장을 출력하는 점을 제외하고 앞선 함수들과 유사

- **.config 파일 수정한 내용이 포함된 스크린샷**

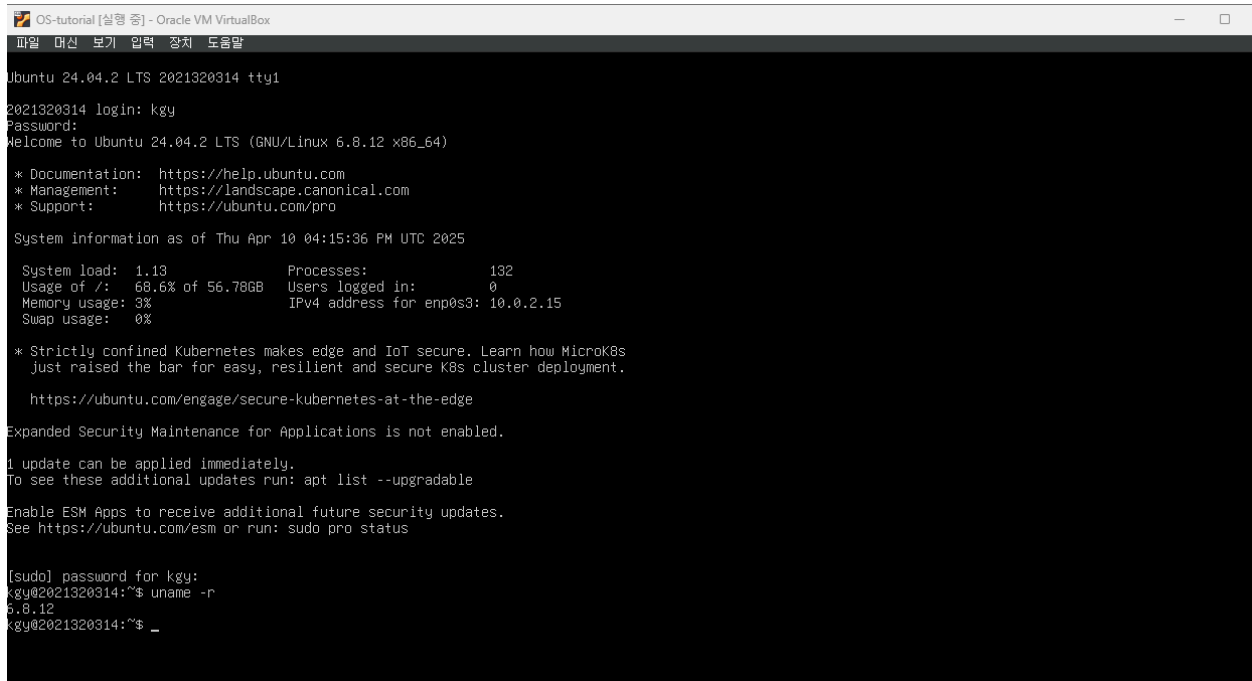


```
OS-tutorial [실행 중] - Oracle VM VirtualBox
파일  마진  보기  입력  장치  도움말
GNU nano 7.2                                .config *
CONFIG_PKCS8_PRIVATE_KEY_PARSER=m
CONFIG_PKCS7_MESSAGE_PARSER=y
CONFIG_PKCS7_TEST_KEY=m
CONFIG_SIGNED_PE_FILE_VERIFICATION=y
# CONFIG_FIPS_SIGNATURE_SELFTEST is not set
#
# Certificates for signature checking
#
CONFIG_MODULE_SIG_KEY="certs/signing_key.pem"
CONFIG_MODULE_SIG_KEY_TYPE_RSA=y
# CONFIG_MODULE_SIG_KEY_TYPE_ECDSA is not set
CONFIG_SYSTEM_TRUSTED_KEYRING=y
CONFIG_SYSTEM_TRUSTED_KEYS=""
CONFIG_SYSTEM_EXTRA_CERTIFICATE=y
CONFIG_SYSTEM_EXTRA_CERTIFICATE_SIZE=4096
CONFIG_SECONDARY_TRUSTED_KEYRING=y
# CONFIG_SECONDARY_TRUSTED_KEYRING_SIGNED_BY_BUILTIN is not set
CONFIG_SYSTEM_BLACKLIST_KEYRING=y
CONFIG_SYSTEM_BLACKLIST_HASH_LIST=""
CONFIG_SYSTEM_REVOCATION_LIST=y
CONFIG_SYSTEM_REVOCATION_KEYS=""
# CONFIG_SYSTEM_BLACKLIST_AUTH_UPDATE is not set
# end of Certificates for signature checking
CONFIG_BINARY_PRINTF=y
#
```

CONFIG\_SYSTEM\_TRUSTED\_KEYS는 신뢰할 수 있는 공개키들을 설정하는 항목이고,  
CONFIG\_SYSTEM\_REVOCATION\_KEYS는 신뢰하지 않는 키를 설정하는 항목이다.

이 두 항목을 "" 로 비움으로써 커널 빌드 및 부팅시 오류를 방지했다.

- 커널 컴파일 후 커널 버전을 확인한 스크린샷



```
OS-tutorial [실행 중] - Oracle VM VirtualBox
파일  편집  보기  입력  장치  도움말

Ubuntu 24.04.2 LTS 2021320314 tty1
2021320314 login: kgy
Password:
Welcome to Ubuntu 24.04.2 LTS (GNU/Linux 6.8.12 x86_64)

* Documentation:  https://help.ubuntu.com
* Management:    https://landscape.canonical.com
* Support:       https://ubuntu.com/pro

System information as of Thu Apr 10 04:15:36 PM UTC 2025

System load:  1.13           Processes:    132
Usage of /:   68.6% of 56.78GB Users logged in: 0
Memory usage: 3%           IPv4 address for enp0s3: 10.0.2.15
Swap usage:   0%

* Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
  just raised the bar for easy, resilient and secure K8s cluster deployment.

https://ubuntu.com/engage/secure-kubernetes-at-the-edge

Expanded Security Maintenance for Applications is not enabled.

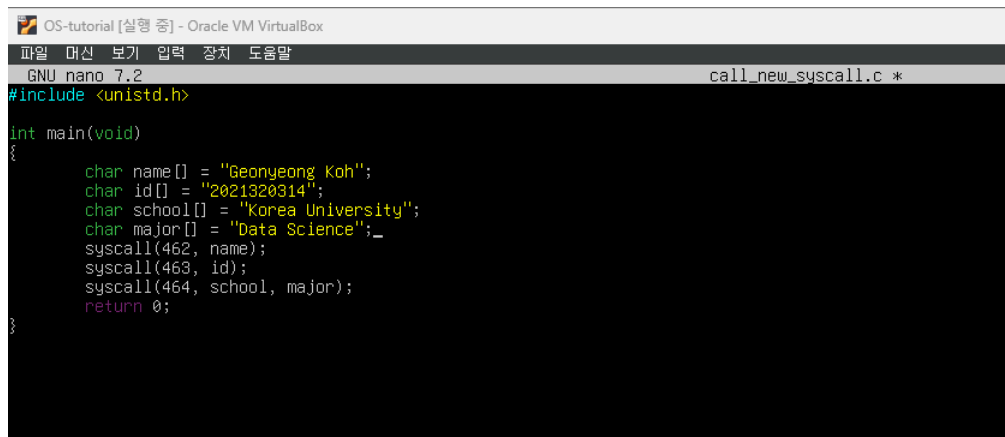
1 update can be applied immediately.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

[sudo] password for kgy:
kgy@2021320314:~$ uname -r
6.8.12
kgy@2021320314:~$ _
```

uname -r을 통해 커널 버전을 확인했고, 6.8.12버전으로 부팅되었음을 확인할 수 있다. 이는 시스템이 성공적으로 빌드가 완료되었고, 정상적으로 부팅되었음을 의미한다.

- 작성한 call\_new\_syscall.c 스크린샷



```
OS-tutorial [실행 중] - Oracle VM VirtualBox
파일  편집  보기  입력  장치  도움말

GNU nano 7.2                                call_new_syscall.c *
#include <unistd.h>

int main(void)
{
    char name[] = "Geonyeong Koh";
    char id[] = "2021320314";
    char school[] = "Korea University";
    char major[] = "Data Science";
    syscall(462, name);
    syscall(463, id);
    syscall(464, school, major);
    return 0;
}
```

syscall()함수는 시스템 콜 번호를 직접 호출하는 방식이다. 이 코드는 syscall()함수를 통해 462번 시스템콜에 name인자, 463번 시스템콜에 id 인자, 464번 시스템콜에 school과 major인자를 전달해서 직접 시스템콜을 호출하는 코드를 작성하였다.

- call\_new\_syscall.c 컴파일 및 실행 후, dmesg를 사용한 커널 메시지 출력 스크린샷

```

[ 12.811821] vmwgfx 0000:00:02.0: [drm] Max GMR ids is 8192
[ 12.811823] vmwgfx 0000:00:02.0: [drm] Max number of GMR pages is 1048576
[ 12.811825] vmwgfx 0000:00:02.0: [drm] Maximum display memory size is 16384 KiB
[ 12.819881] vmwgfx 0000:00:02.0: [drm] Screen Object display unit initialized
[ 12.820358] vmwgfx 0000:00:02.0: [drm] Fifo max 0x00200000 min 0x00001000 cap 0x00000355
[ 12.820654] vmwgfx 0000:00:02.0: [drm] Using command buffers with DMA pool.
[ 12.820672] vmwgfx 0000:00:02.0: [drm] Available shader model: Legacy.
[ 12.822509] [drm] Initialized vmwgfx 2.20.0 20211206 for 0000:00:02.0 on minor 0
[ 12.824047] fbcon: vmwgfxdrmfb (fb0) is primary device
[ 12.827793] Console: switching to colour frame buffer device 160x50
[ 12.943791] vmwgfx 0000:00:02.0: [drm] fb0: vmwgfxdrmfb frame buffer device
[ 12.964374] snd_intel8x0 0000:00:05.0: allow list rate for 1028:0177 is 48000
[ 13.058518] workqueue: drm_fb_helper_damage_work hogged CPU for >10000us 4 times, consider switching to WQ_UNBOUND
[ 13.126464] EXT4-fs (sda2): mounted filesystem cb545c07-80cd-4224-a41f-964b7b60aa25 r/w with ordered data mode. Quota mode: none.
[ 13.292554] audit: type=1400 audit(1744301728.640:2): apparmor="STATUS" operation="profile_load" profile="unconfined" name="Discord" pid=571 comm="apparmor_p
anser"
[ 13.292987] audit: type=1400 audit(1744301728.640:3): apparmor="STATUS" operation="profile_load" profile="unconfined" name="QtWebEngineProcess" pid=573 comm=
"apparmor_parser"
[ 13.292993] audit: type=1400 audit(1744301728.640:4): apparmor="STATUS" operation="profile_load" profile="unconfined" name="1password" pid=570 comm="apparmor
_parser"
[ 13.293166] audit: type=1400 audit(1744301728.640:5): apparmor="STATUS" operation="profile_load" profile="unconfined" name="4D6F6E676F44220436F6D70617373 pid
=572 comm="apparmor_parser"
[ 13.303712] audit: type=1400 audit(1744301728.650:6): apparmor="STATUS" operation="profile_load" profile="unconfined" name="brave" pid=578 comm="apparmor_par
ser"
[ 13.305470] audit: type=1400 audit(1744301728.650:7): apparmor="STATUS" operation="profile_load" profile="unconfined" name="buildah" pid=579 comm="apparmor_p
anser"
[ 13.306155] audit: type=1400 audit(1744301728.653:8): apparmor="STATUS" operation="profile_load" profile="unconfined" name="balena-etcher" pid=577 comm="appa
rmor_parser"
[ 13.310371] audit: type=1400 audit(1744301728.657:9): apparmor="STATUS" operation="profile_load" profile="unconfined" name="busybox" pid=581 comm="apparmor_p
anser"
[ 13.311879] audit: type=1400 audit(1744301728.659:10): apparmor="STATUS" operation="profile_load" profile="unconfined" name="ch-checkns" pid=583 comm="apparm
on_parser"
[ 13.312349] audit: type=1400 audit(1744301728.659:11): apparmor="STATUS" operation="profile_load" profile="unconfined" name="cam" pid=582 comm="apparmor_pars
er"
[ 13.370803] workqueue: drm_fb_helper_damage_work hogged CPU for >10000us 8 times, consider switching to WQ_UNBOUND
[ 14.260112] cfg80211: Loading compiled-in X.509 certificates for regulatory database
[ 14.263724] Loaded X.509 cert 'sforshee: 00b28ddf47aef9cea7'
[ 14.263864] Loaded X.509 cert 'wens: 61c038651aabdcf94bd0ac7ff06c7248db18c600'
[ 14.337276] e1000: enp0s3 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: RX
[ 14.542535] workqueue: drm_fb_helper_damage_work hogged CPU for >10000us 16 times, consider switching to WQ_UNBOUND
[ 15.201940] NET: Registered PF_QIPCRTR protocol family
[ 15.430908] loop0: detected capacity change from 0 to 8
[ 22.420641] systemd-journald[328]: /var/log/journal/205719a89dd24a78912e3aad37c541f6/user-1000.journal: Journal file uses a different sequence number ID, not
ating.
[ 133.491565] workqueue: drm_fb_helper_damage_work hogged CPU for >10000us 32 times, consider switching to WQ_UNBOUND
[ 344.600538] My Name is Geonyeong Koh
[ 344.600543] My Student ID is 2021320314
[ 344.600545] I go to Korea University
[ 344.600546] I major in Data Science
xgy@2021320314:~/workspace$ _

```

스크린샷 아래를 보면 정상적으로 call\_new\_syscall.c의 인자 정의했던 이름, 학번, 대학, 학과가 잘 출력되고 있다. 이를 통해 new\_syscall.c에 정의한 시스템 콜 함수들이 정상적으로 호출되고 실행되었음을 알 수 있다.