

2024년 2학기 운영체제실습 3주차

# System Call Programming

**System Software Laboratory**

School of Computer and Information Engineering

Kwangwoon Univ.

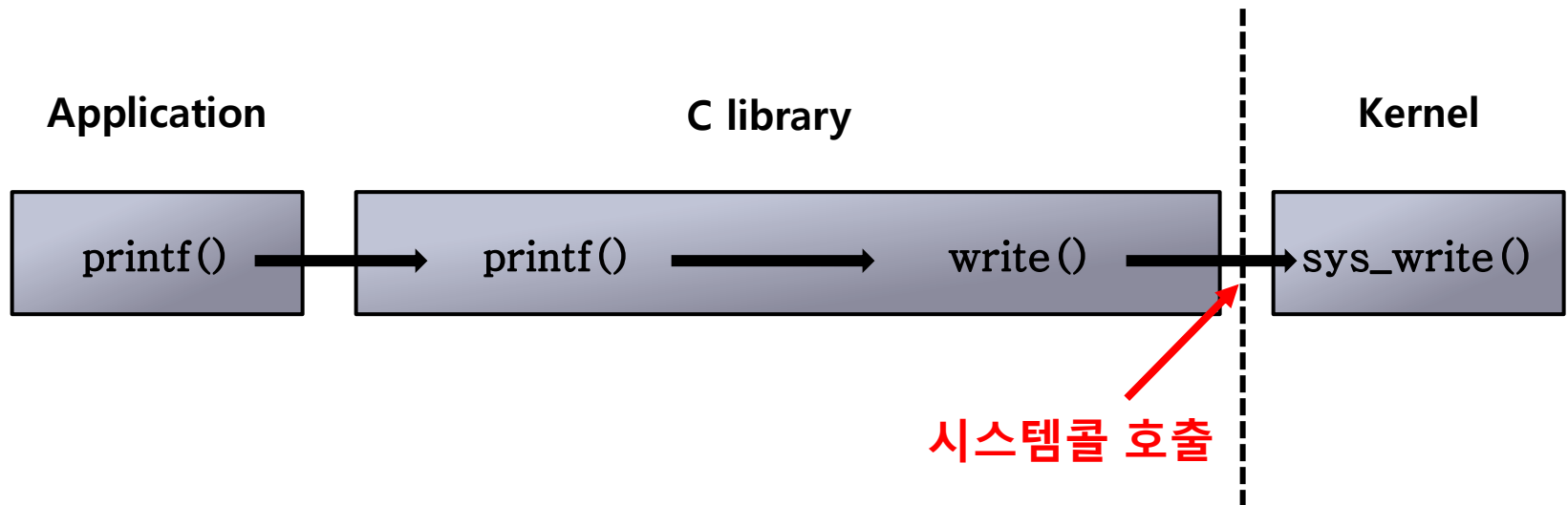
# Contents

- **System Call**
  - 실습 1. 새로운 System Call 작성
- **데이터 교환 함수**
- **Timer**
  - Jiffies
  - `current_kernel_time()`

# System Call

## System Call

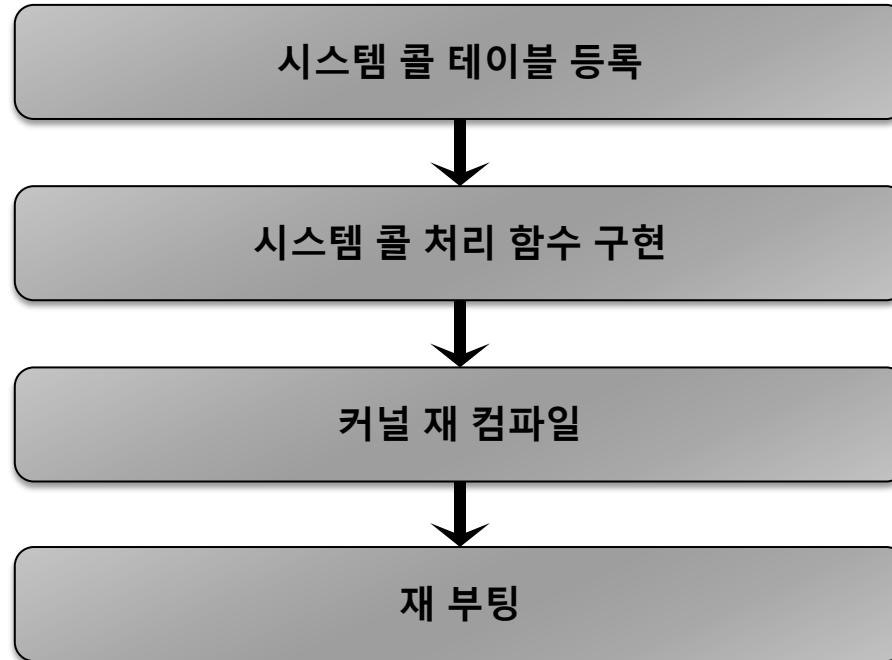
- 사용자 응용 프로그램에서 운영체제의 기능을 사용할 수 있게 해주는 통로
- 사용자 모드에 있는 프로세스가 CPU, disk, printer 등의 하드웨어 장치와 상호 작용할 수 있도록 기능을 제공하는 인터페이스
- 소프트웨어 인터럽트를 통해 사용자 프로그램에서 커널에게 보내는 서비스 요청



<printf() 호출 시 어플리케이션과 커널의 관계>

# System Call

- System Call 구현



# 실습 1. 새로운 System Call 작성

- System Call 테이블 등록

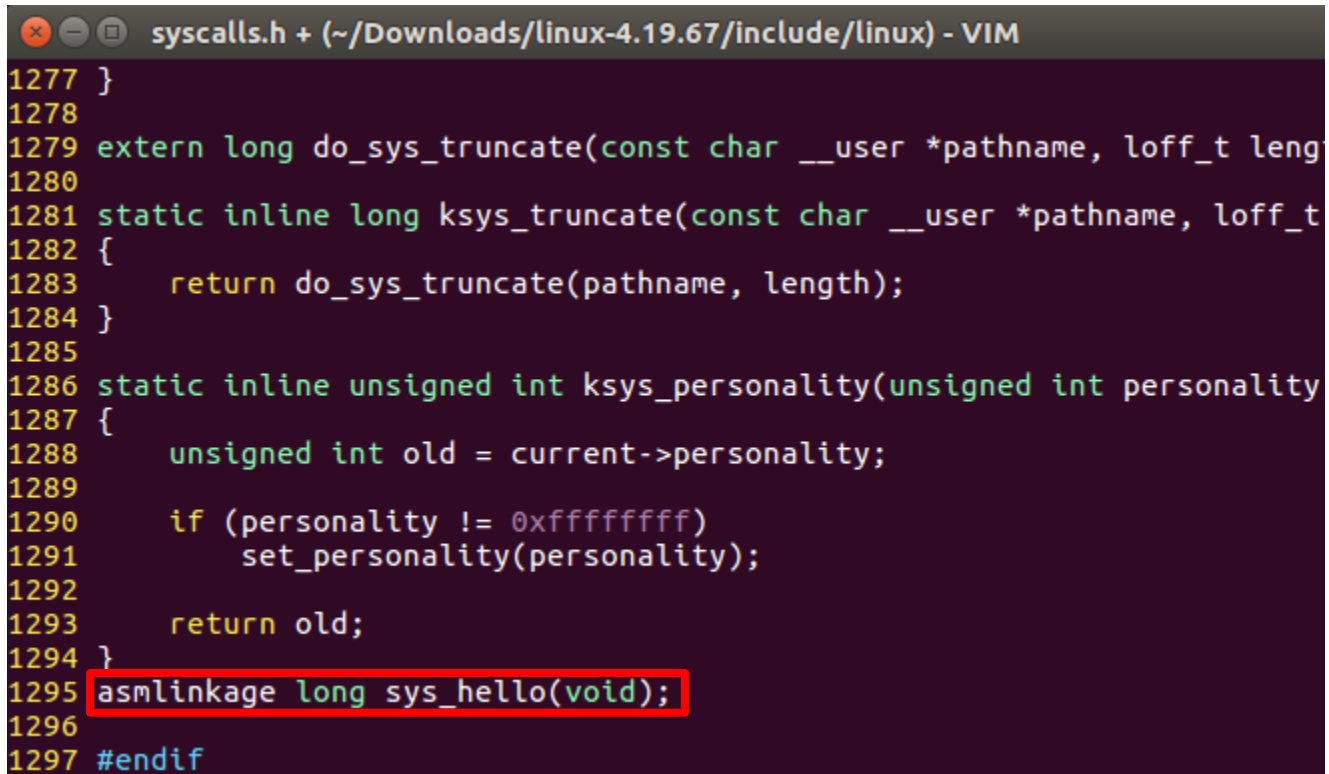
- `$ vi arch/x86/entry/syscalls/syscall_64.tbl`

- 아래와 같이 입력

```
385 544 x32 io_submit      __x32_compat_sys_io_submit
386 545 x32 execveat      __x32_compat_sys_execveat/ptregs
387 546 x32 preadv2       __x32_compat_sys_preadv64v2
388 547 x32 pwritev2      __x32_compat_sys_pwritev64v2
389 548 common hello      __x64_sys_hello
```

# 실습 1. 새로운 System Call 작성

- System Call 테이블 등록
  - \$ vi include/linux/syscalls.h
    - #endif 이전에 작성



```
syscalls.h + (~/Downloads/linux-4.19.67/include/linux) - VIM
1277 }
1278
1279 extern long do_sys_truncate(const char __user *pathname, loff_t leng
1280
1281 static inline long ksys_truncate(const char __user *pathname, loff_t
1282 {
1283     return do_sys_truncate(pathname, length);
1284 }
1285
1286 static inline unsigned int ksys_personality(unsigned int personality
1287 {
1288     unsigned int old = current->personality;
1289
1290     if (personality != 0xffffffff)
1291         set_personality(personality);
1292
1293     return old;
1294 }
1295 asmlinkage long sys_hello(void);
1296
1297 #endif
```

# 실습 1. 새로운 System Call 작성

- System Call 함수 구현

- \$ mkdir hello
- \$ vi hello/hello.c

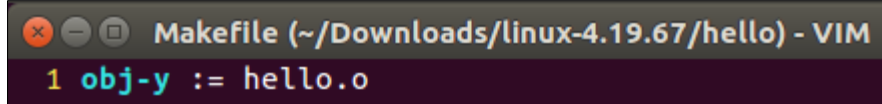
```
1 #include <linux/kernel.h>
2 #include <linux/syscalls.h>
3
4 SYSCALL_DEFINE0(hello)
5 {
6     printk("Hello world\n");
7
8     return 0;
9 }
```

- 4: SYSCALL\_DEFINE0(hello)
  - “hello” 라는 이름으로 시스템 콜을 생성하는 매크로
    - 이를 수행함으로써, sys\_hello 뿐만 아니라 다양한 함수들이 자동적으로 생성
    - 강의 자료 5페이지의 \_\_x64\_sys\_hello도 본 단계에서 자동으로 생성됨
  - 숫자 “0”은 해당 시스템 콜이 인자를 0개 받는 것을 의미
  - 인자 수에 따라 다음과 같은 것들이 존재
    - SYSCALL\_DEFINE1(), SYSCALL\_DEFINE2(), ... SYSCALL\_DEFINE6()

# 실습 1. 새로운 System Call 작성

- System Call 함수 구현

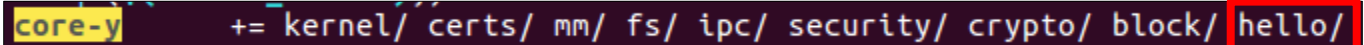
- \$ vi hello/Makefile



```
Makefile (~/Downloads/linux-4.19.67/hello) - VIM
1 obj-y := hello.o
```

- \$ vi Makefile

- core-y 패턴 검색 후 2번 째 결과에서



```
core-y += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/ hello/
```



# 실습 1. 새로운 System Call 작성

- 수정된 커널 컴파일
  - `$ sudo make`
  - `$ sudo make modules_install`
  - `$ sudo make install`
- `$ sudo reboot`
  - 방금 컴파일 한 커널로 부팅

# 실습 1. 새로운 System Call 작성

- 새로운 System Call을 테스트 할 프로그램 작성 및 동작
  - \$ mkdir ~/working
  - \$ cd ~/working
  - \$ vi hello\_test.c

```
1 #include <stdio.h> /* printf() */
2
3 #include <unistd.h> /* syscall() */
4 #include <sys/syscall.h> /* syscall() */
5
6 int main(void)
7 {
8     long ret;
9
10    ret = syscall(548);
11    printf("%ld\n", ret);
12
13    return 0;
14 }
```

# 실습 1. 새로운 System Call 작성

- 새로운 System Call을 테스트 할 프로그램 작성 및 동작 (cont'd)

- \$ ./app

```
sslab@ubuntu:~/test$ ./app
0
```

- \$ dmesg

```
sslab@ubuntu:~/test$ dmesg
[ 10.899900] audit: type=1400 audit(1568583870.725:11): apparmor="STATUS" operation="profile_load" profile="unconfined" name="webbrowser-app" pid=604 comm="apparmor_parser"
[ 11.088572] random: crng init done
[ 11.088573] random: 7 urandom warning(s) missed due to ratelimiting
[ 11.150644] Adding 998396k swap on /dev/sda5. Priority:-2 extents:1 across:998396k FS
[ 12.136088] IPv6: ADDRCONF(NETDEV_UP): ens33: link is not ready
[ 12.143996] IPv6: ADDRCONF(NETDEV_UP): ens33: link is not ready
[ 12.152331] e1000: ens33 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: None
[ 12.154231] IPv6: ADDRCONF(NETDEV_CHANGE): ens33: link becomes ready
[ 63.568999] Hello world
```

- 시스템 콜 호출을 실패한 경우, syscall() 함수에서 -1 반환

## 실습 2. 인자를 받는 System Call 작성

- System Call 테이블 등록

- \$ vi arch/x86/entry/syscalls/syscall\_64.tbl
  - 아래와 같이 작성

```
385 544 x32 io_submit      __x32_compat_sys_io_submit
386 545 x32 execveat      __x32_compat_sys_execveat/ptregs
387 546 x32 preadv2       __x32_compat_sys_preadv64v2
388 547 x32 pwritev2      __x32_compat_sys_pwritev64v2
389 548 common hello      x64 sys hello
390 549 common add        __x64_sys_add
```

## 실습 2. 인자를 받는 System Call 작성

- System Call 테이블 등록
  - \$ vi include/linux/syscalls.h
    - #endif 위에 작성

```
1286 static inline unsigned int ksys_personality(unsigned int personality)
1287 {
1288     unsigned int old = current->personality;
1289
1290     if (personality != 0xffffffff)
1291         set_personality(personality);
1292
1293     return old;
1294 }
1295
1296 asmlinkage long sys hello(void);
1297 asmlinkage long sys_add(int, int);
1298
1299 #endif
```

## 실습 2. 인자를 받는 System Call 작성

- System Call 함수 구현

- \$ mkdir add
- \$ vi add/add.c

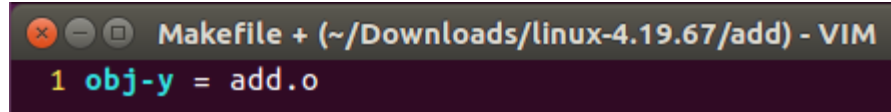
```
1 #include <linux/kernel.h>
2 #include <linux/syscalls.h>
3
4 SYSCALL_DEFINE2(add, int, a, int, b)
5 {
6     long ret;
7
8     ret = a+b;
9
10    return ret;
11 }
```

- 4: SYSCALL\_DEFINE2(add)
  - “hello” 라는 이름으로 시스템 콜을 생성하는 매크로
    - 이를 수행함으로써, sys\_add 뿐만 아니라 다양한 함수들이 자동적으로 생성
    - 강의 자료 12페이지의 \_\_x64\_sys\_hello도 본 단계에서 자동으로 생성됨
  - 숫자 “2”은 해당 시스템 콜이 인자를 2개 받는 것을 의미
    - int 형의 a, int 형의 b

## 실습 2. 인자를 받는 System Call 작성

- System Call 함수 구현

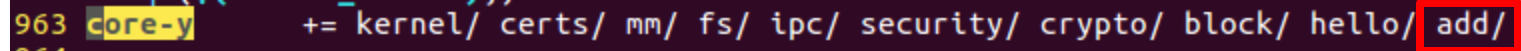
- \$ vi add/Makefile



A screenshot of a VIM editor window. The title bar reads 'Makefile + (~/Downloads/linux-4.19.67/add) - VIM'. The main text area shows a single line of code: '1 obj-y = add.o'.

- \$ vi Makefile

- core-y 패턴 검색 후 2번 째 결과에서



A screenshot of a VIM search result. The text '963 core-y' is highlighted in yellow. To its right, a list of paths is shown: '+= kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/ hello/ add/'. The 'add/' part at the end of the list is enclosed in a red rectangular box.

## 실습 2. 인자를 받는 System Call 작성

- 수정된 커널 컴파일
  - `$ sudo make`
  - `$ sudo make modules_install`
  - `$ sudo make install`
- `$ sudo reboot`
  - 방금 컴파일 한 커널로 부팅



## 실습 2. 인자를 받는 System Call 작성

- 새로운 System Call을 테스트 할 프로그램 작성 및 동작
  - \$ mkdir ~/working
  - \$ cd ~/working
  - \$ vi add\_test.c

```
1 #include <stdio.h>
2
3 #include <unistd.h> /* syscall() */
4 #include <sys/syscall.h> /* syscall() */
5
6 int main(void)
7 {
8     int a, b;
9     long ret;
10
11     a = 7;
12     b = 4;
13
14     ret = syscall(549, a, b);
15     printf("%d op. %d = %ld\n", a, b, ret);
16
17     return 0;
18 }
```

# 데이터 교환 함수

- **System call을 실행하기 전 모든 매개변수가 유효한지 확인**
  - 사용자가 잘못된 영역의 포인터를 커널로 전달할 수 있다면 보안과 안정성에 문제가 생김
  - 사용자 포인터 참조 전, 커널은 다음 사항을 확인
    - 포인터가 사용자 공간의 메모리 영역을 가리키는가?
    - 요청한 프로세스의 주소 공간을 가리키는가?
    - 프로세스의 메모리 접근 제한을 준수하는가?
  - 커널은 위 확인 작업을 수행하고 원하는 내용을 제공하는 함수를 제공
    - `copy_to_user()`
    - `copy_from_user()`

# 데이터 교환 함수

- 사용자 영역과 커널 영역 사이에서 값을 교환하는 커널 내 함수
  - Included in `<asm/uaccess.h>`
  - Copy data from **user space** to **kernel space**
    - A block of data: `copy_from_user` (void \*to, void \*from, unsigned long n);
      - to : destination address, **in kernel space**
      - from : source address, **in user space**
      - n : # of bytes to copy
    - A simple variable: `get_user` (void \*x, void \*ptr);
      - x : variable to store, **in kernel space**
      - ptr : source address, **in user space**
  - Copy data from **kernel space** to **user space**
    - A block of data: `copy_to_user` (void \*to, void \*from, unsigned long n);
      - to : destination address, **in user space**
      - from : source address, **in kernel space**
      - n : # of bytes to copy
    - A simple variable: `put_user` (void \*x, void \*ptr);
      - x : variable to copy, **in user space**
      - ptr : source address, **in kernel space**

# Timer

- 시간 정보에 관한 전역 변수

- Jiffies, HZ, xtime

- Jiffies

- 시스템에 내장된 타이머에서 주기적으로 발생시키는 인터럽트
- 시스템이 시작된 이후 경과된 타이머 Tick의 수를 저장

# Timer

- **HZ (진동 수)**

- 초당 몇 번 tick이 발생하는가를 의미
- Tick: 인터럽트 사이의 시간 주기 ( $\text{tick} = 1/\text{HZ}$ )
  - i386의 경우 커널 2.4에서는 100, 2.6.10 이전 버전의 커널에서는 1000, 2.6.10 이후 커널에서는 250을 기본값으로 함
- 진동수가 높을 수록 시스템의 정확도가 높아지나, 타이머 인터럽트의 처리 비용이 높아짐

- **xtime**

- 현재 시각(wall time)이 xtime변수로 정의되어 있음
- `xtime.tv_sec`은 1970년 1월 1일 이후부터 지금까지의 시간(초 단위)을 의미
- `xtime.tv_nsec`은 마지막 초 이후에 경과된 나노 초를 의미

# jiffies

- **jiffies**
  - 시스템이 부팅된 이후 발생한 tick 수를 저장
- **사용법**
  - 초를 jiffies로 변환하는 방법
    - $\text{Second} * \text{HZ}$
  - Jiffies를 초로 변환하는 방법
    - $\text{Jiffies} / \text{HZ}$

# jiffies

- **사용 예제**

- 현재 시각 : `unsigned long time_stamp = jiffies;`
- 현재로부터 1tick 후 : `unsigned long next_tick = jiffies + 1;`
- 현재로부터 5초 후 : `unsigned long later = jiffies + 5 * HZ;`

- **jiffies in Kernel 5.15**

- Jiffies 값을 64비트로 변경
  - 64비트 jiffies값은 `jiffies_64`변수로 선언
- `jiffies_64`값을 참조하기 위한 함수
  - `get_jiffies_64()`

# current\_kernel\_time()

- `current_kernel_time()`

- 현재 시각(wall time)

```
struct timespec current_kernel_time(void);  
  
struct timespec {  
    time_t    tv_sec;           /* seconds */  
    long      tv_nsec;         /* nanoseconds */  
};
```

- `current_kernel_time().tv_sec`
    - 1970년 1월 1일 이후 지금까지의 시간(epoch time)을 초단위로 저장
  - `current_kernel_time().tv_nsec`
    - 마지막 초 이후에 경과된 나노 초