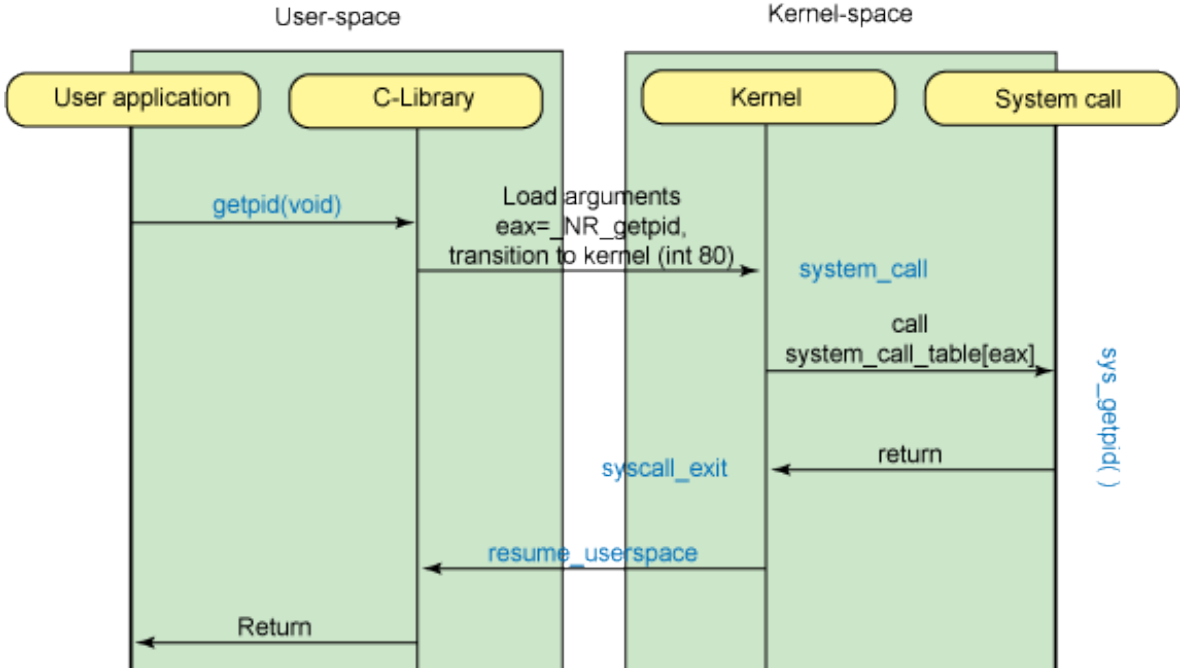
Computer Core Practice1: Operating System
Week3. System Call & Debugging Technique

진주영, 황인중
Embedded Software Lab.

System Call

- 응용 프로그램이 커널에 접근하기 위한 인터페이스
  - 사용자 모드에 있는 응용 프로그램이 커널의 기능을 사용할 수 있도록 함
  - 시스템 콜을 부르면 사용자 모드에서 커널 모드로 바뀜
  - 커널에서 시스템 콜을 처리를 끝내면 커널 모드에서 사용자 모드로 돌아가 작업을 계속함

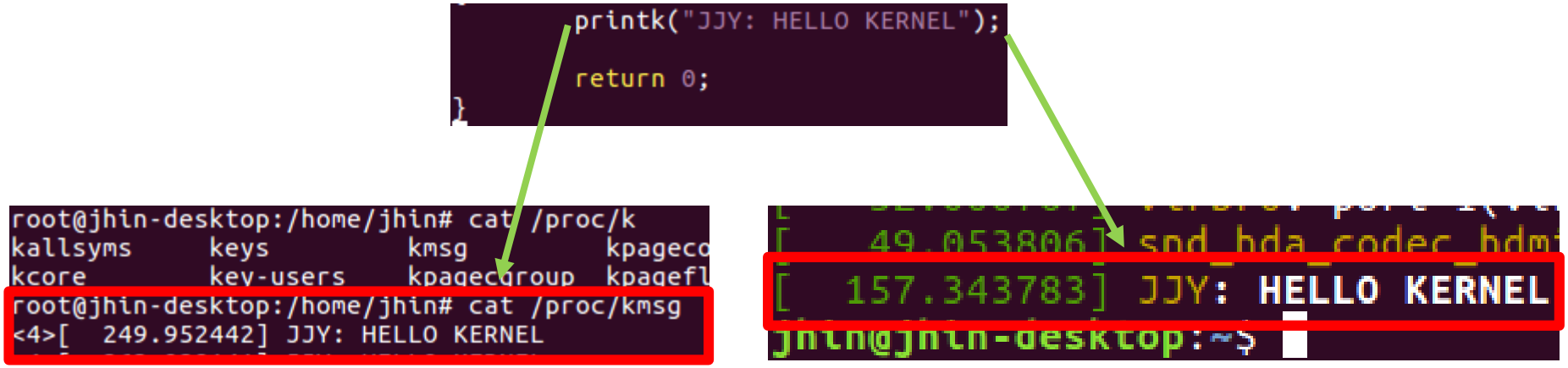


System Call - example


```
asmlinkage long sys_open(const char __user *filename,
                int flags, umode_t mode);
SYSCALL_DEFINE3(open, const char __user *, filename, int, flags, umode_t, mode)
{
    if (force_o_largefile())
        flags |= O_LARGEFILE;

    return do_sys_open(AT_FDCWD, filename, flags, mode);
}
long do_sys_open(int dfd, const char __user *filename, int flags, umode_t mode)
{
    struct open_flags op;
    int fd = build_open_flags(flags, mode, &op);
    struct filename *tmp;

    if (fd)
        return fd;

    tmp = getname(filename);
    if (IS_ERR(tmp))
        return PTR_ERR(tmp);

    fd = get_unused_fd_flags(flags);
    if (fd >= 0) {
        struct file *f = do_filp_open(dfd, tmp, &op);
        if (IS_ERR(f)) {
            put_unused_fd(fd);
            fd = PTR_ERR(f);
        } else {
            fsnotify_open(f);
            fd_install(fd, f);
        }
    }
    putname(tmp);
    return fd;
} ? end do_sys_open ?
```

Debugging Technique - printk

- Kernel Print 함수
- C 라이브러리의 printf() 함수와 유사하게 동작
- 커널에서 언제 어디서나 호출 가능
- /proc/kmsg, printk trace, dmesg 명령어 등으로 printk가 출력한 로그를 확인할 수 있음



/proc/kmsg

dmesg

Debugging Technique - strace

- Command로 인해 발생하는 system call과 parameter, return value 등을 사용자에게 보여줌
- 사용법: strace command
  - Example1: strace ls
  - Example2: strace ./application

```c
#include <stdio.h>

int main(void)
{
        FILE *fp = fopen("abc.txt", "w");
        fprintf(fp, "hi");
        fclose(fp);

        return 0;
}
```

```
jhin@jhin-desktop:~$ vim fileio.c
jhin@jhin-desktop:~$ gcc -o fileio fileio.c
jhin@jhin-desktop:~$ strace ./fileio
execve("./fileio", ["./fileio"], [/* 70 vars */]) = 0
brk(NULL)                                       = 0x24f6000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=128388, ...}) =
mmap(NULL, 128388, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f
close(3)                                        = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No
open("/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOE
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0
fstat(3, {st_mode=S_IFREG|0755, st_size=1868984, ...})
mmap(NULL, 3971488, PROT_READ|PROT_EXEC, MAP_PRIVATE|MA
mprotect(0x7fdd35224000, 2097152, PROT_NONE) = 0
mmap(0x7fdd35424000, 24576, PROT_READ|PROT_WRITE, MAP_P
mmap(0x7fdd3542a000, 14752, PROT_READ|PROT_WRITE, MAP_P
close(3)                                        = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_
arch_prctl(ARCH_SET_FS, 0x7fdd3562f700) = 0
mprotect(0x7fdd35424000, 16384, PROT_READ) = 0
mprotect(0x600000, 4096, PROT_READ)     = 0
mprotect(0x7fdd35653000, 4096, PROT_READ) = 0
munmap(0x7fdd35631000, 128388)          = 0
brk(NULL)                                       = 0x24f6000
brk(0x2517000)                                  = 0x2517000
open("abc.txt", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
fstat(3, {st_mode=S_IFREG|0664, st_size=0, ...}) = 0
write(3, "hi", 2)                               = 2
close(3)                                        = 0
exit_group(0)                                   = ?
+++ exited with 0 +++
```

Debugging Technique - ftrace

- 커널의 내부 동작을 trace 및 디버깅하기 위해 사용
  - Event tracing (interrupt, scheduling, filesystem, …)
  - Kernel function tracing (모든 커널 함수, stack usage, …)
  - Latency tracing (wakeup, wakeup_rt, irqsoft, preemptoff, …)

- /sys/kernel/debug/tracing

```
root@jhin-desktop:/home/jhin# cd /sys/kernel/debug/tracing/
root@jhin-desktop:/sys/kernel/debug/tracing# ls
available_events            current_tracer          function_profile_enabled  max_graph_depth
available_filter_functions  dyn_ftrace_total_info   hwlat_detector            options
available_tracers           enabled_functions       instances                 per_cpu
buffer_size_kb              events                  kprobe_events             printk_formats
buffer_total_size_kb        free_buffer             kprobe_profile            README
```

- /sys/kernel/debug/tracing/events

```
root@jhin-desktop:/sys/kernel/debug/tracing# cd events/
root@jhin-desktop:/sys/kernel/debug/tracing/events# ls
block       cpuhp       fence     ftrace          header_event  irq          kvmmmu   mmc
cgroup      drm         fib       gpio            header_page   irq_vectors  libata   module
clk         enable      fib6      hda             huge_memory   jbd2         mce      mpx
cma         exceptions  filelock  hda_controller  i2c           kmem         mei      msr
compaction  ext4        filemap   hda_intel       iommu         kvm          migrate  napi
```

Debugging Technique - ftrace

- example

```
root@jhin-desktop:/sys/kernel/debug/tracing# ls
available_events            current_tracer         function_profile_enabled  max_graph_depth
available_filter_functions  dyn_ftrace_total_info  hwlat_detector            options
available_tracers           enabled_functions      instances                 per_cpu
buffer_size_kb              events                 kprobe_events             printk_formats
buffer_total_size_kb        free_buffer            kprobe_profile            README
root@jhin-desktop:/sys/kernel/debug/tracing# cat current_tracer
nop
root@jhin-desktop:/sys/kernel/debug/tracing# cat trace_pipe
^C
root@jhin-desktop:/sys/kernel/debug/tracing# cat available_tracers
hwlat blk mmiotrace function_graph wakeup_dl wakeup_rt wakeup function nop
root@jhin-desktop:/sys/kernel/debug/tracing# echo function > current_tracer
```

$ cat trace_pipe

```
irq/126-nvidia-1077  [007] ....   980.227551: os_acquire_spinlock <-_nv018056rm
irq/126-nvidia-1077  [007] ....   980.227551: _raw_spin_lock_irqsave <-os_acquire_spinlock
irq/126-nvidia-1077  [007] d..1   980.227552: do_raw_spin_lock <-_raw_spin_lock_irqsave
irq/126-nvidia-1077  [007] d..1   980.227552: os_release_spinlock <-_nv018056rm
irq/126-nvidia-1077  [007] d..1   980.227552: _raw_spin_unlock_irqrestore <-os_release_spinlock
irq/126-nvidia-1077  [007] d..1   980.227552: do_raw_spin_unlock <-_raw_spin_unlock_irqrestore
irq/126-nvidia-1077  [007] ....   980.227552: os_get_current_thread <-_nv026241rm
irq/126-nvidia-1077  [007] ....   980.227553: os_get_cpu_number <-_nv007141rm
irq/126-nvidia-1077  [007] ....   980.227553: os_get_current_time <-_nv026242rm
irq/126-nvidia-1077  [007] ....   980.227553: do_gettimeofday <-os_get_current_time
```

System Call - Practice

- 3 Steps for implementation of your own system call
  - 정의 (Define a system call)

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

asmlinkage long sys_jjycall(void)
{
```

  - 등록 (Register the system call to SYSCALL table)

```
asmlinkage long sys_pkey_mprotect(ur
                                  ur
asmlinkage long sys_pkey_alloc(unsig
asmlinkage long sys_pkey_free(int pk
asmlinkage long sys_jjycall(void);
#endif
```

```
325     common  mlock2          sys_mlock2
326     common  copy_file_range sys_copy_file_range
327     64      preadv2         sys_preadv2
328     64      pwritev2        sys_pwritev2
329     common  pkey_mprotect   sys_pkey_mprotect
330     common  pkey_alloc      sys_pkey_alloc
331     common  pkey_free       sys_pkey_free
332     common  jjycall         sys_jjycall
```

  - 사용 (Call it in a user-level application)

```c
#include <sys/syscall.h>
#define SYSCALL_NUMBER 332
int main(void)
{
        syscall(SYSCALL_NUMBER);
        return 0;
}
```

System Call - Practice

- cd ${KERNEL_SRC_DIR}/kernel
- vim sysjjy.c

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

asmlinkage long sys_jjycall(void)
{
        printk("JJY: HELLO KERNEL");

        return 0;
}
```

- vim Makefile

```
obj-y     = fork.o exec_domain.o panic.o \
            cpu.o exit.o softirq.o resource.o \
            sysctl.o sysctl_binary.o capability.o ptrace.o user.o \
            signal.o sys.o kmod.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 sysjjy.o

obj-$(CONFIG_MULTIUSER) += groups.o

ifdef CONFIG_FUNCTION_TRACER
```

System Call - Practice

- cd ${KERNEL_SRC_DIR}/arch/x86/entry/syscalls
- vim syscall_64.tbl

```
jhin@jhin-desktop:~/mtftl_total/mtftl_lightnvm$ cd arch/x86/entry/syscalls/
jhin@jhin-desktop:~/mtftl_total/mtftl_lightnvm/arch/x86/entry/syscalls$ ls
Makefile  syscall_32.tbl  syscall_64.tbl  syscallhdr.sh  syscalltbl.sh
jhin@jhin-desktop:~/mtftl_total/mtftl_lightnvm/arch/x86/entry/syscalls$ vim syscall_64.tbl
```

```
323     common  userfaultfd             sys_userfaultfd
324     common  membarrier              sys_membarrier
325     common  mlock2                  sys_mlock2
326     common  copy_file_range         sys_copy_file_range
327     64      preadv2                 sys_preadv2
328     64      pwritev2                sys_pwritev2
329     common  pkey_mprotect           sys_pkey_mprotect
330     common  pkey_alloc              sys_pkey_alloc
331     common  pkey_free               sys_pkey_free
332     common  jjycall                 sys_jjycall

#
# x32-specific system call numbers start at 512 to avoid cac
# for native 64-bit operation.
#
512     x32     rt_sigaction            compat_sys_rt_sigact
```

System Call 번호

앞서 정의했던 System Call 함수명

System Call - Practice

- cd ${KERNEL_SRC_DIR}/include/linux
- vim syscalls.h

jhin@jhin-desktop: ~/mtftl_total/mtftl_lightnvm

```
asmlinkage long sys_seccomp(unsigned int op, unsigned int flags,
                            const char __user *uargs);
asmlinkage long sys_getrandom(char __user *buf, size_t count,
                              unsigned int flags);
asmlinkage long sys_bpf(int cmd, union bpf_attr *attr, unsigned int size);

asmlinkage long sys_execveat(int dfd, const char __user *filename,
                        const char __user *const __user *argv,
                        const char __user *const __user *envp, int flags);

asmlinkage long sys_membarrier(int cmd, int flags);
asmlinkage long sys_copy_file_range(int fd_in, loff_t __user *off_in,
                                    int fd_out, loff_t __user *off_out,
                                    size_t len, unsigned int flags);

asmlinkage long sys_mlock2(unsigned long start, size_t len, int flags);

asmlinkage long sys_pkey_mprotect(unsigned long start, size_t len,
                                  unsigned long prot, int pkey);
asmlinkage long sys_pkey_alloc(unsigned long flags, unsigned long init_val);
asmlinkage long sys_pkey_free(int pkey);
asmlinkage long sys_jjycall(void);
#endif
                                                              906,6         Bot
```

System Call - Practice


```c
#include <sys/syscall.h>
#define SYSCALL_NUMBER 332
int main(void)
{
        syscall(SYSCALL_NUMBER);
        return 0;
}
```

```
[   30.694111] Bluetooth: BNEP socket layer
[   32.000758] device virbr0-nic left promi
[   32.000767] virbr0: port 1(virbr0-nic) e
[   49.053806] snd_hda_codec_hdmi hdaudioC1
[  157.343783] JJY: HELLO KERNEL
jnin@jnin-desktop:~$
```

Homework #1

- **아무 동작이나 수행하는 유저 프로그램 작성**
  - 간단한 파일 입출력 프로그램부터
  - 소켓을 이용한 통신 프로그램까지 모두 가능

- **strace, ftrace를 이용하여 작성한 프로그램이 어떻게 동작하는 지 관찰**
  - 수업시간에 예시로 본 function tracer 뿐만 아니라 다른 tracer도 사용 권장
  - Tracer on 뿐 아니라 events 디렉토리에서 다양한 event를 on 시킨 상태로 trace 출력을 수행해볼 것을 권장 ex. ext4 event 등..

- **제출물**
  - 과제 수행을 위하여 실행한 command를 순서대로 작성(또는 캡처)한 것
  - strace 수행 결과 출력 파일 (shell에서 redirection으로)
  - ftrace 수행 결과 출력 파일 (tracer 종류 또는 event 종류 별로 2가지)

- **기한**
  - 9/17 23:59:59 까지
  - 기한이 지나 제출한 것은 채점하지 않겠습니다.