

2016년 2학기 운영체제실습 5주차

System Call Programming

Dept. of Computer Engineering,
Kwangwoon Univ.

Contents

- ▶ **System Call**

- ▶ 실습 1. 새로운 System Call 작성

- ▶ **데이터 교환 함수**

- ▶ **Timer**

- ▶ Jiffies
 - ▶ `current_kernel_time()`

- ▶ **실습 2. 현재 커널 시간을 반환하는 함수 작성**

스케줄 변경

수업	9월						
	일	월	화(3,4)	수	목(3,4)	금(5,6)	토
1주차		8/29	8/30	8/31	9/1	2	3
2주차	4	5	6	7	8	9	10
3주차 (대체:10/1)	11	12	13	14	15	16	17
				추석 연휴			
4주차	18	19	20	21	22	23	24
5주차	25	26	27	28	29	30	

수업	10월						
	일	월	화(3,4)	수	목(3,4)	금(5,6)	토
5주차							1 1차 퀴즈
6주차	2	3	4	5	6	7	8
		개천절		월계축전			
7주차	9	10	11	12	13	14	15
8주차 (대체:10/15)	16	17	18	19	20	21	22 2차 퀴즈
		중간고사					20
9주차	23	24	25	26	27	28	
10주차	30	31					

수업	11월						
	일	월	화(3,4)	수	목(3,4)	금(5,6)	토
10주차			1	2	3	4	5 3차 퀴즈
11주차	6	7	8	9	10	11	12
12주차	13	14	15	16	17	18	19
13주차	20	21	22	23	24	25	26 4차 퀴즈
14주차	27	28	29	30			

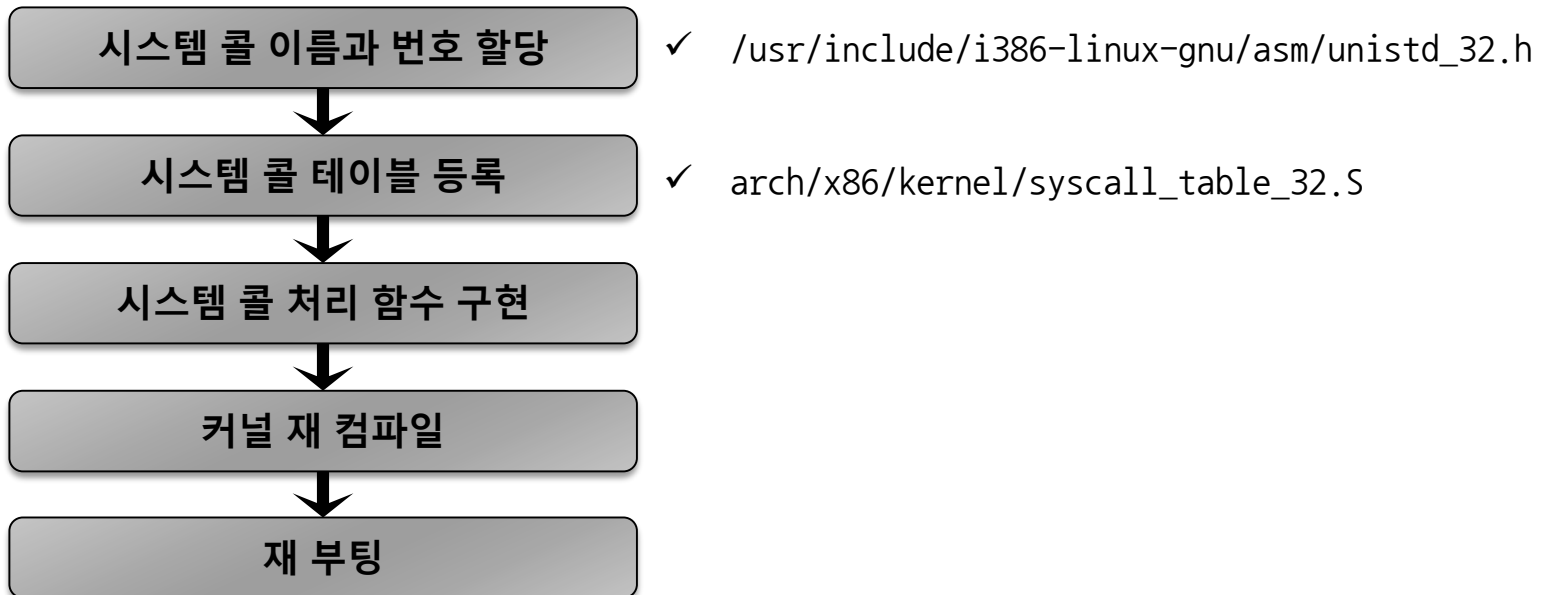
수업	12월						
	일	월	화(3,4)	수	목(3,4)	금(5,6)	토
14주차					1	2	3
15주차 (대체:12/10)	4	5	6	7	8	9	10 5차 퀴즈
		기말고사, 보강					

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

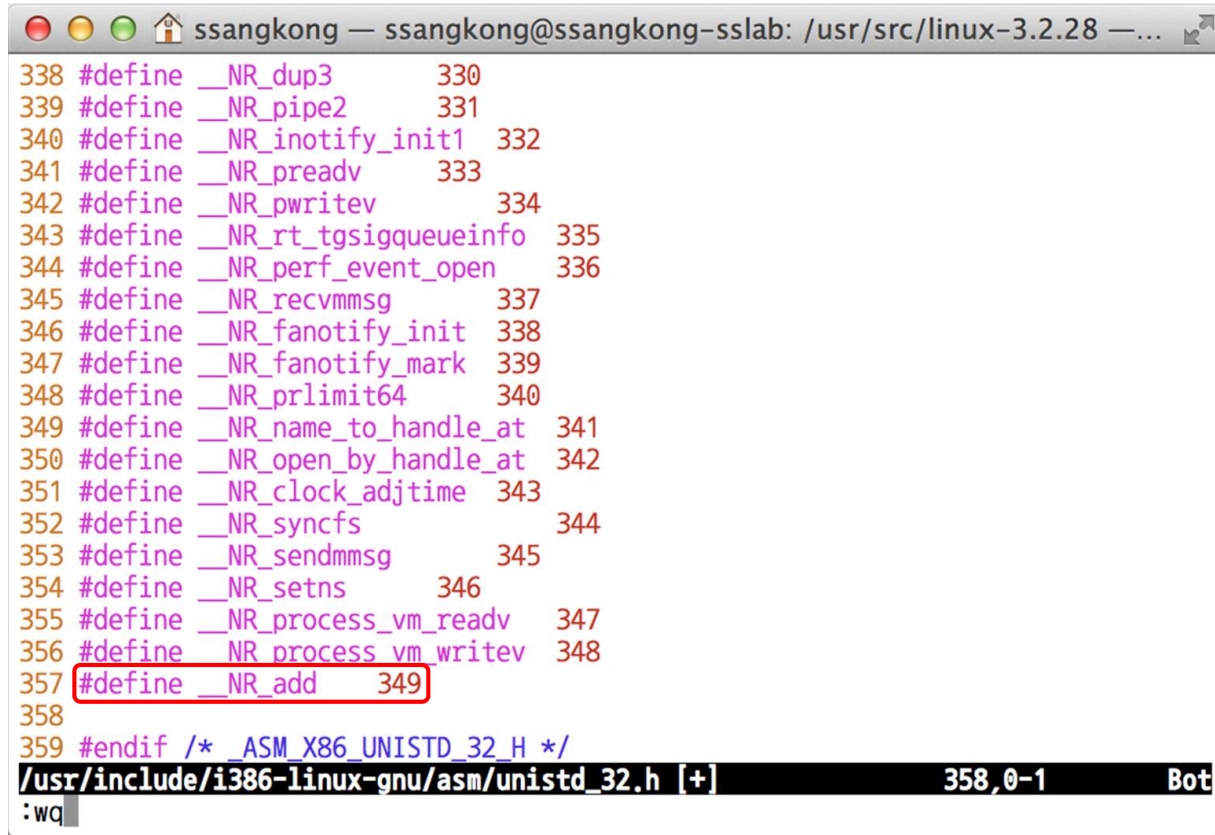
▶ System Call 구현



실습 1. 새로운 System Call 작성

▶ System Call 이름 및 번호 할당

- ▶ `$ cd /usr/include/i386-linux-gnu/asm`
- ▶ `$ vi unistd_32.h`



```
ssangkong — ssangkong@ssangkong-sslab: /usr/src/linux-3.2.28 —...
338 #define __NR_dup3      330
339 #define __NR_pipe2     331
340 #define __NR_inotify_init1 332
341 #define __NR_preadv     333
342 #define __NR_pwritev    334
343 #define __NR_rt_tsigqueueinfo 335
344 #define __NR_perf_event_open 336
345 #define __NR_recvmmsg   337
346 #define __NR_fanotify_init 338
347 #define __NR_fanotify_mark 339
348 #define __NR_prlimit64   340
349 #define __NR_name_to_handle_at 341
350 #define __NR_open_by_handle_at 342
351 #define __NR_clock_adjtime 343
352 #define __NR_syncfs      344
353 #define __NR_sendmmsg    345
354 #define __NR_setns       346
355 #define __NR_process_vm_readv 347
356 #define __NR_process_vm_writev 348
357 #define __NR_add         349
358
359 #endif /* ASM_X86_UNISTD_32_H */
/usr/include/i386-linux-gnu/asm/unistd_32.h [+]  
358,0-1 Bot
:wq
```

실습 1. 새로운 System Call 작성

▶ System Call 테이블 등록

- ▶ `$ cd /usr/src/linux-3.2.28`
- ▶ `$ cd arch/x86/kernel`
- ▶ `$ vi syscall_table_32.S`

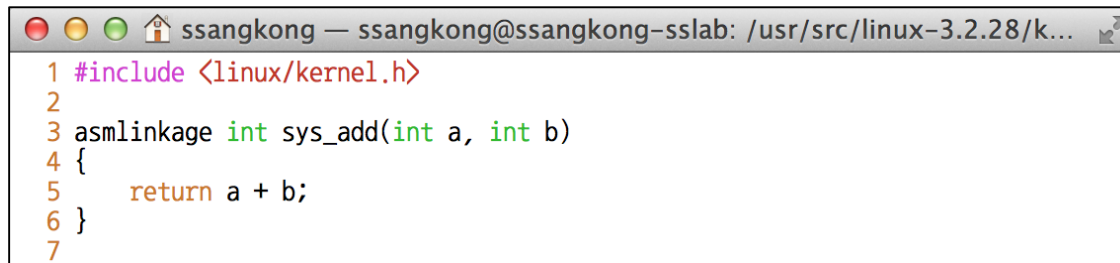


```
ssangkong — ssangkong@ssangkong-sslab: /usr/src/linux-3.2.28 — ...
330     .long sys_eventfd2
331     .long sys_epoll_create1
332     .long sys_dup3          /* 330 */
333     .long sys_pipe2
334     .long sys_inotify_init1
335     .long sys_preadv
336     .long sys_pwritev
337     .long sys_rt_tsigqueueinfo /* 335 */
338     .long sys_perf_event_open
339     .long sys_recvmmsg
340     .long sys_fanotify_init
341     .long sys_fanotify_mark
342     .long sys_prlimit64     /* 340 */
343     .long sys_name_to_handle_at
344     .long sys_open_by_handle_at
345     .long sys_clock_adjtime
346     .long sys_syncfs
347     .long sys_sendmmsg     /* 345 */
348     .long sys_setns
349     .long sys_process_vm_readv
350     .long sys_process_vm_writev
351     .long sys_add
arch/x86/kernel/syscall_table_32.S 351,14-17 Bot
:wq
```

실습 1. 새로운 System Call 작성

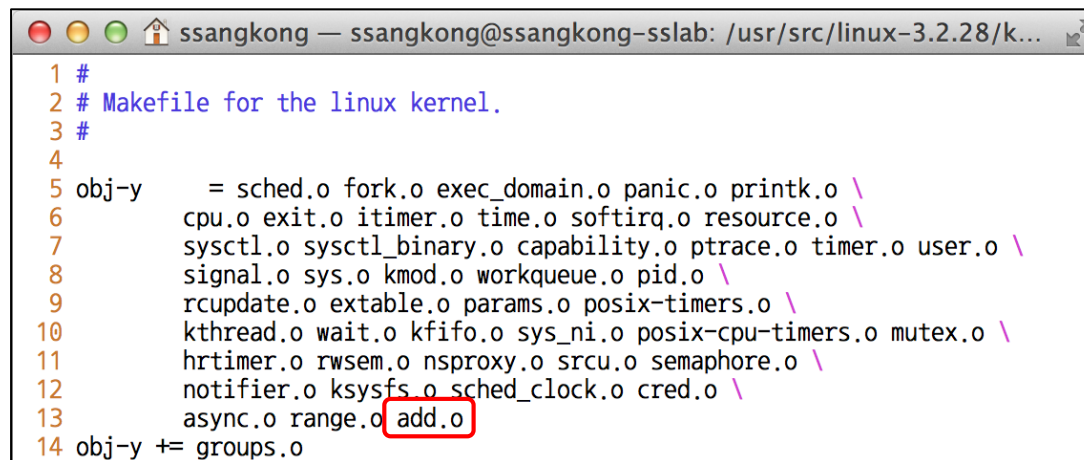
▶ System Call 함수 구현

- ▶ `$ cd /usr/src/linux-3.2.28/kernel`
- ▶ `$ vi add.c`



```
ssangkong — ssangkong@ssangkong-sslab: /usr/src/linux-3.2.28/k...  
1 #include <linux/kernel.h>  
2  
3 asmlinkage int sys_add(int a, int b)  
4 {  
5     return a + b;  
6 }  
7
```

- ▶ `$ vi Makefile`



```
ssangkong — ssangkong@ssangkong-sslab: /usr/src/linux-3.2.28/k...  
1 #  
2 # Makefile for the linux kernel.  
3 #  
4  
5 obj-y      = sched.o fork.o exec_domain.o panic.o printk.o \  
6             cpu.o exit.o itimer.o time.o softirq.o resource.o \  
7             sysctl.o sysctl_binary.o capability.o ptrace.o timer.o user.o \  
8             signal.o sys.o kmod.o workqueue.o pid.o \  
9             rcupdate.o extable.o params.o posix-timers.o \  
10            kthread.o wait.o kfifo.o sys_ni.o posix-cpu-timers.o mutex.o \  
11            hrtimer.o rwsem.o nsproxy.o srcu.o semaphore.o \  
12            notifier.o ksysfs.o sched_clock.o cred.o \  
13            async.o range.o add.o  
14 obj-y += groups.o
```

실습 1. 새로운 System Call 작성

▶ 수정된 커널 컴파일

- ▶ `$ cd /usr/src/linux-3.2.28`
- ▶ `$ make`
- ▶ `$ make modules_install`
- ▶ `$ make install`
- ▶ `$ 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 add_test.c`

```
ssangkong — ssangkong@ssangkong-sslab: ~/working — ssh — 80x24
1 include <stdio.h>
2 #include <linux/unistd.h>
3
4 int main(int argc, const char* argv[])
5 {
6     int a, b;
7     a=7;
8     b=4;
9     printf("%d add %d = %d\n", a, b, syscall(__NR_add, a, b));
10    a=2;
11    b=5;
12    printf("%d add %d = %d\n", a, b, syscall(__NR_add, a, b));
13    return 0;
14 }
```

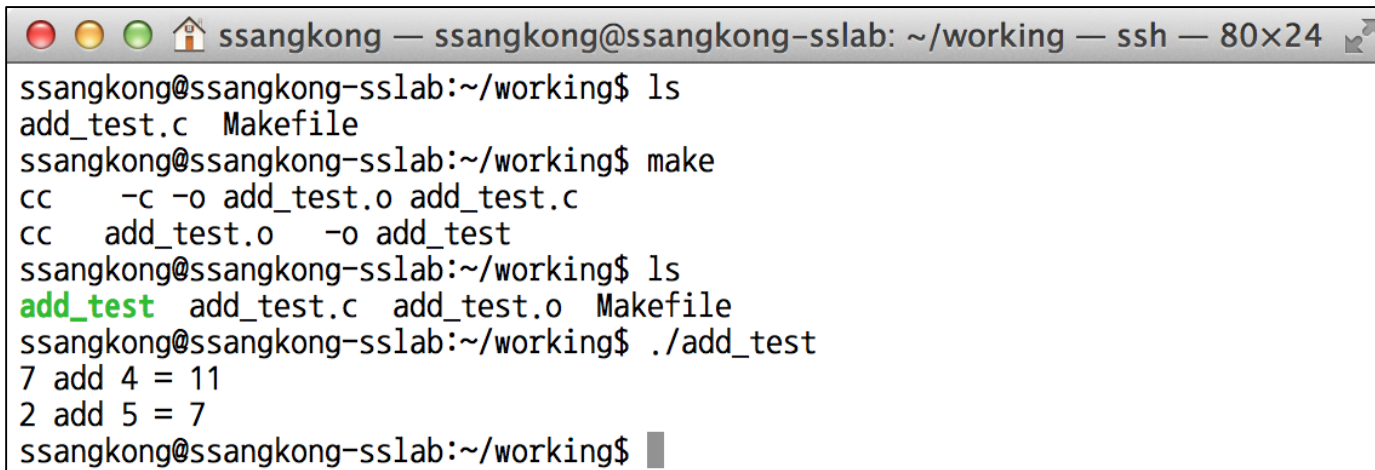
- ▶ `$ vi Makefile`

```
ssangkong — ssangkong@ssar
1 RCS=add_test.c
2
3 $(SRCS:.c=):$(SRCS:.c=.o)
4
5 clean:
6     $(RM) $(SRCS:.c=.o) $(SRCS:.c=)
```

실습 1. 새로운 System Call 작성

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

- ▶ \$ make
- ▶ \$./add_test



```
ssangkong@ssangkong-sslslab: ~/working — ssh — 80x24
ssangkong@ssangkong-sslslab:~/working$ ls
add_test.c  Makefile
ssangkong@ssangkong-sslslab:~/working$ make
cc -c -o add_test.o add_test.c
cc add_test.o -o add_test
ssangkong@ssangkong-sslslab:~/working$ ls
add_test add_test.c add_test.o Makefile
ssangkong@ssangkong-sslslab:~/working$ ./add_test
7 add 4 = 11
2 add 5 = 7
ssangkong@ssangkong-sslslab:~/working$
```

- ▶ 잘못된 시스템 호출이 발생하면 오류 코드 -1을 반환하고 실행을 중단함

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 높아지고 시스템의 성능이 좋아지나, 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 2.6

- ▶ jiffies 값을 64비트로 변경
 - ▶ 64비트 jiffies값은 jiffies_64변수로 선언
- ▶ jiffies_64값을 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

▶ 이전 커널에서는 xtime이라는 변수로 제공하였으나,

- ▶ tv_sec와 tv_nsec의 atomicity를 보장하기 위해
- ▶ current_kernel_time() 함수로 값을 받아온 뒤 사용

실습 2. 현재 커널 시간을 반환하는 함수 작성

▶ 조건

▶ System call 추가

- ▶ 이름 : `get_kt (__NR_getkt, sys_getkt)`
- ▶ 번호 : 350
- ▶ 함수 원형 : `asmlinkage int sys_getkt(int *sec, int *nsec);`

▶ 동작

- ▶ 현재 커널 시간을 계산
- ▶ 초 단위(int *sec)와 나노 초 단위(int *nsec)로 나누어 user space로 전달
- ▶ 데이터 전달 함수를 활용할 것