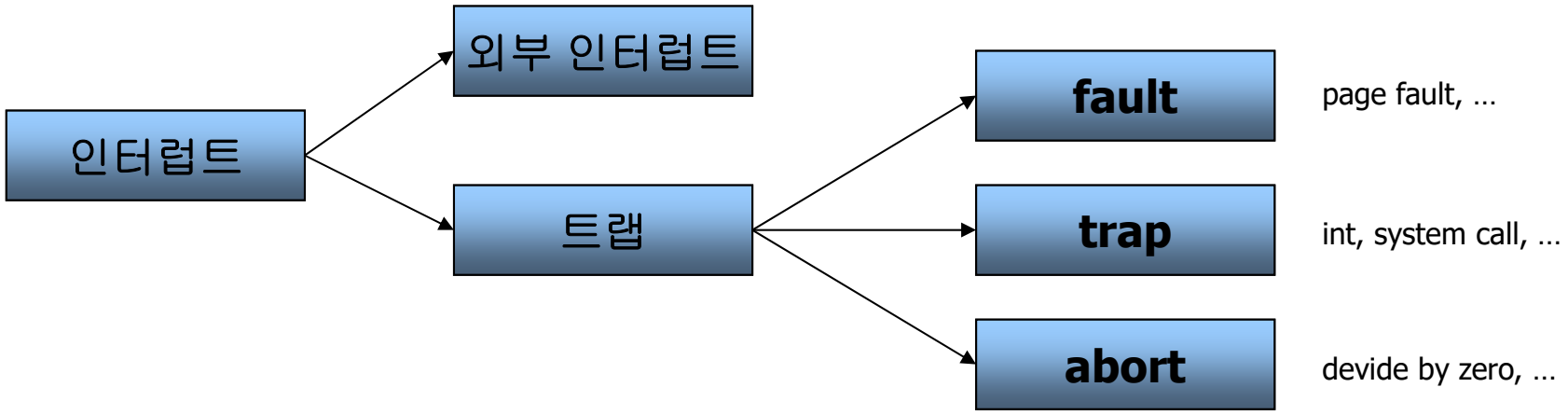
Interrupt, Trap and System call

May, 2016

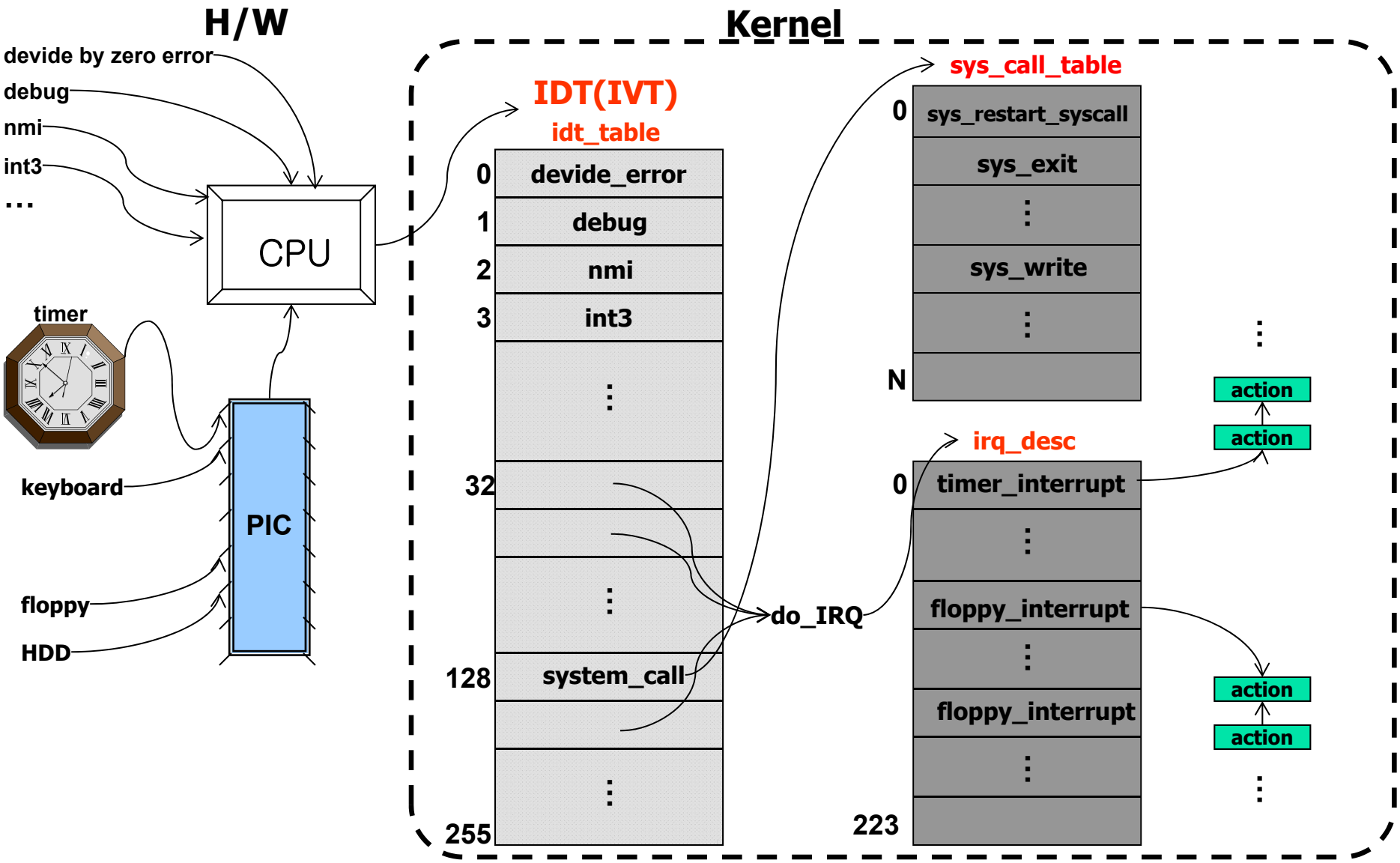
Seungjae Baek

Dept. of software
Dankook University
http://embedded.dankook.ac.kr/~baeksj

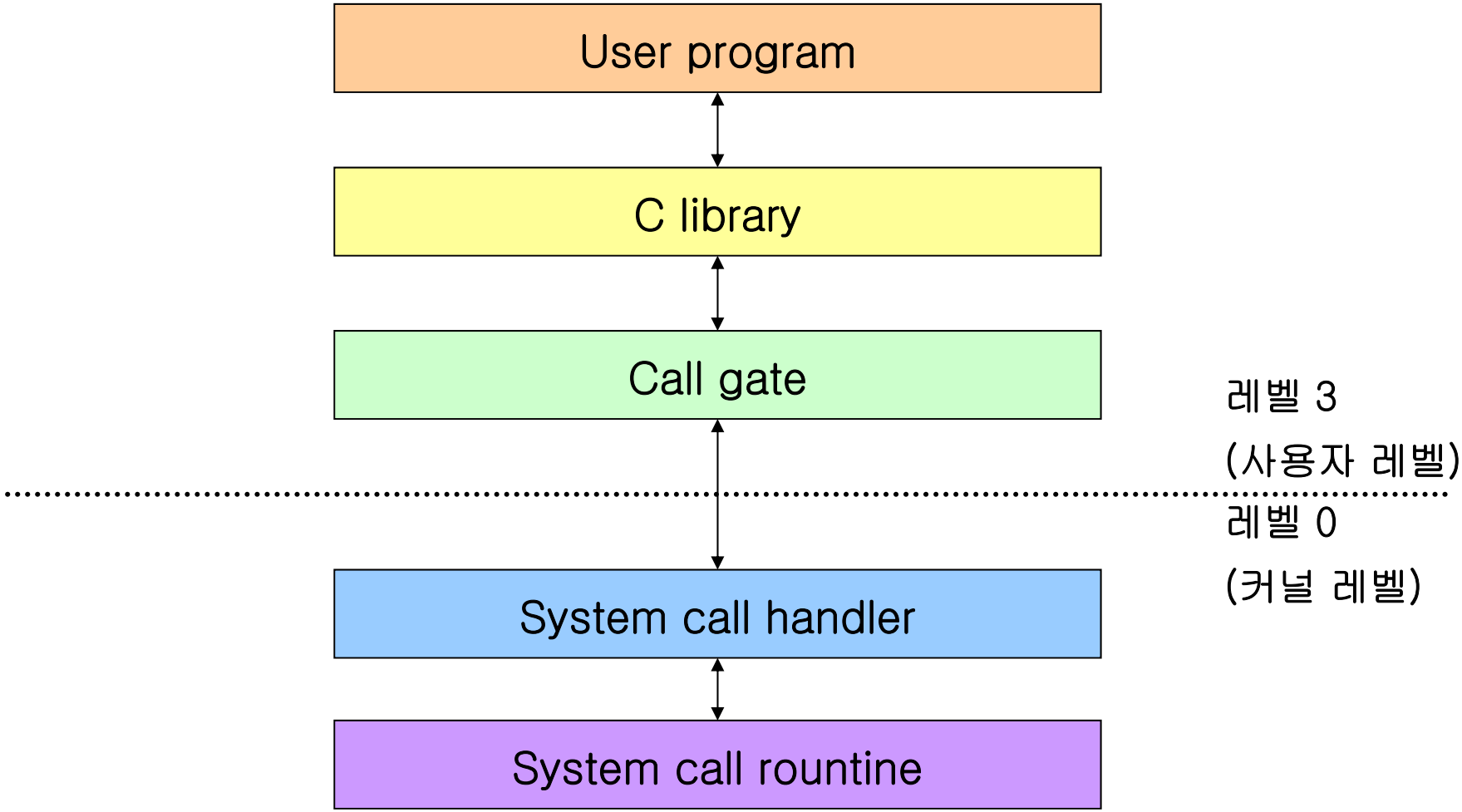
인터럽트의 분류


인터럽트
외부 인터럽트
트랩
fault
trap
abort
page fault, …
int, system call, …
devide by zero, …

인터럽트와 트랩의 처리

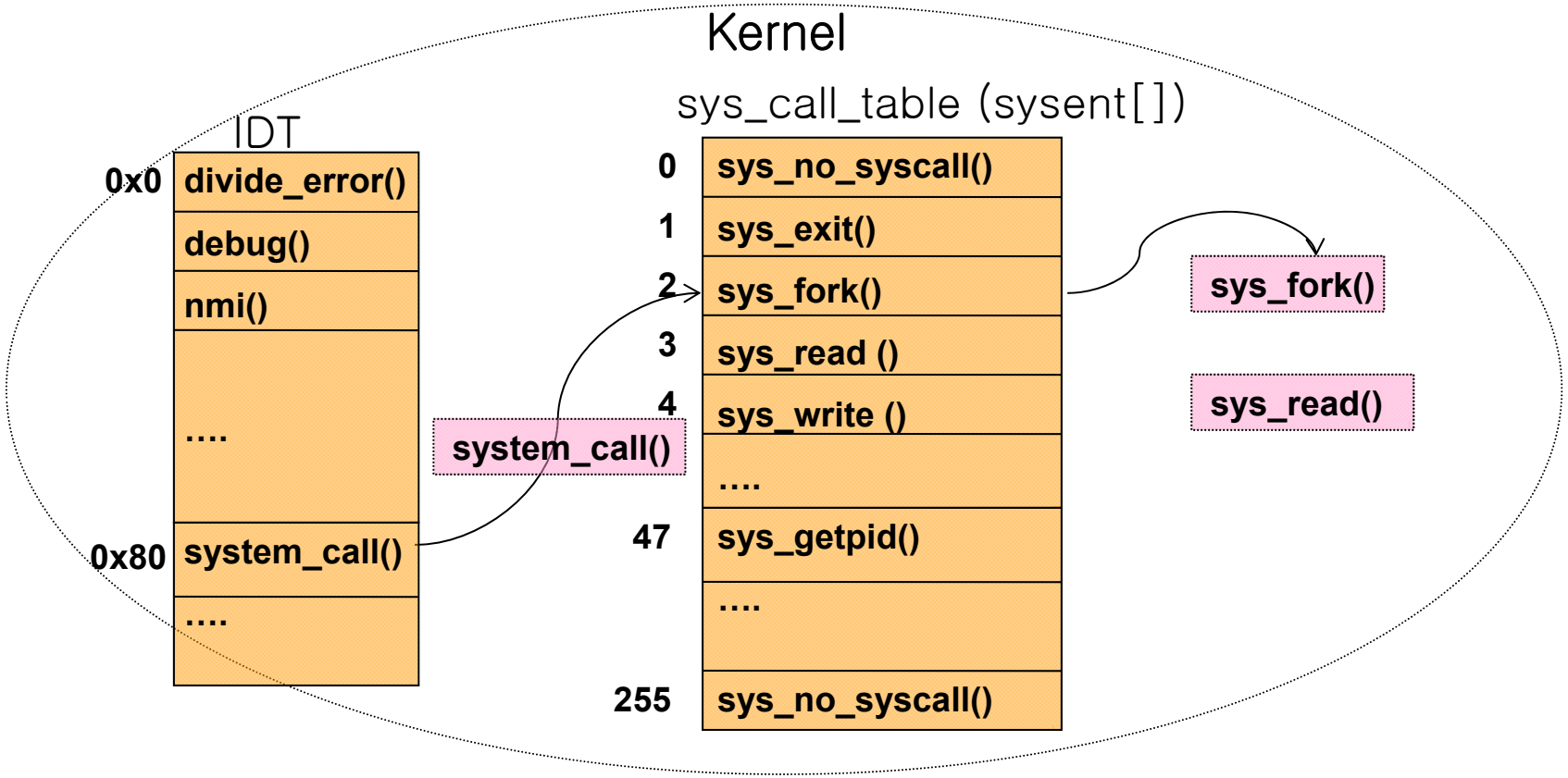

H/W
devide by zero error
debug
nmi
int3
...
CPU
timer
keyboard
PIC
floppy
HDD
Kernel
IDT(IVT)
idt_table
0 devide_error
1 debug
2 nmi
3 int3
32
128 system_call
255
do_IRQ
sys_call_table
0 sys_restart_syscall
sys_exit
sys_write
N
irq_desc
0 timer_interrupt
floppy_interrupt
floppy_interrupt
223
action
action
action
action

시스템 호출 처리 (1/14)


User program
C library
Call gate
System call handler
System call rountine
레벨 3
(사용자 레벨)
레벨 0
(커널 레벨)

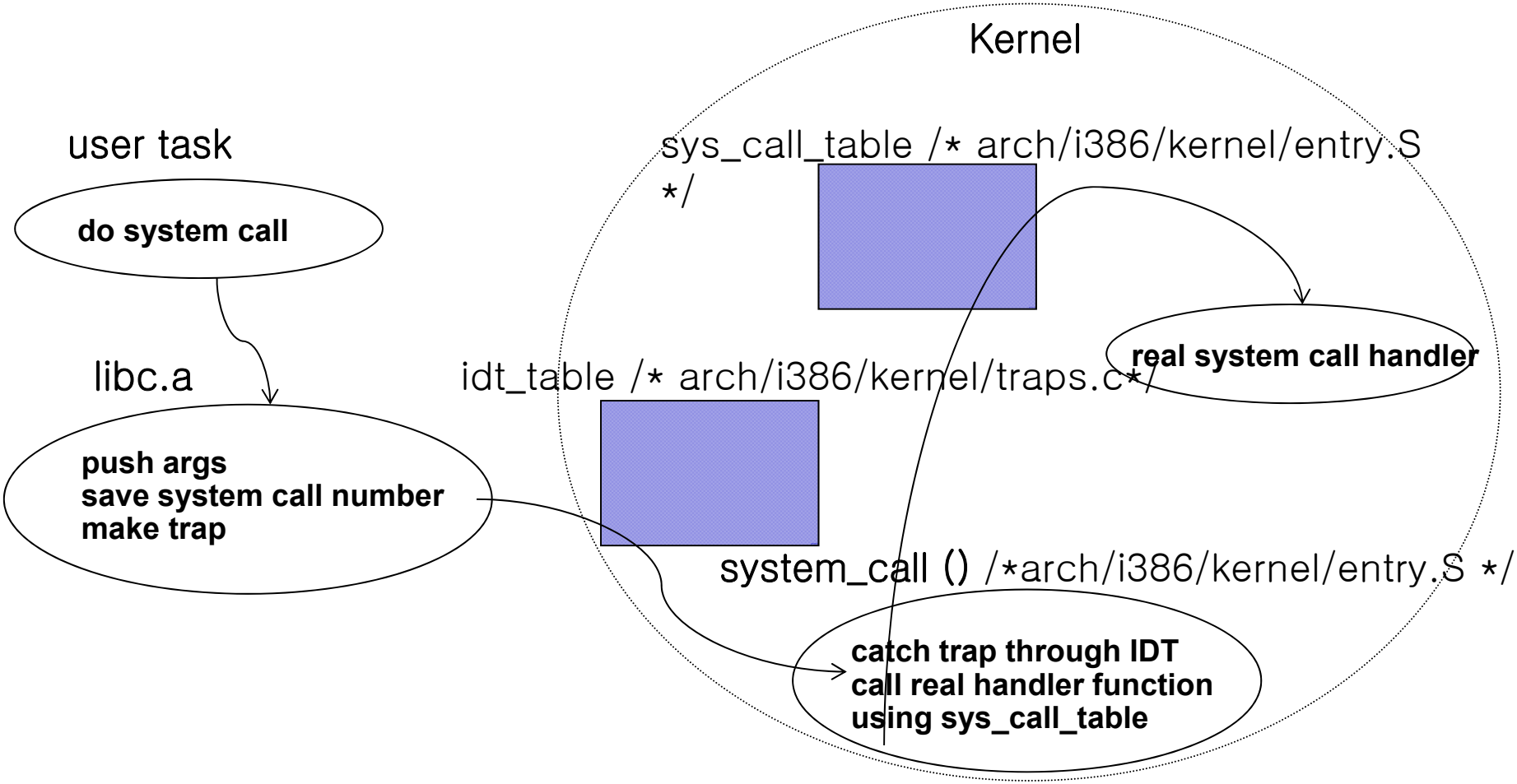
시스템 호출 처리 (3/14)

- 시스템 호출 (system call)
  - ✓ 태스크가 커널의 서비스 요청 (fork, read, socket, …)
  - ✓ 트랩 처리 메커니즘에 의해 처리 : 0x80
  - ✓ system call table 이용



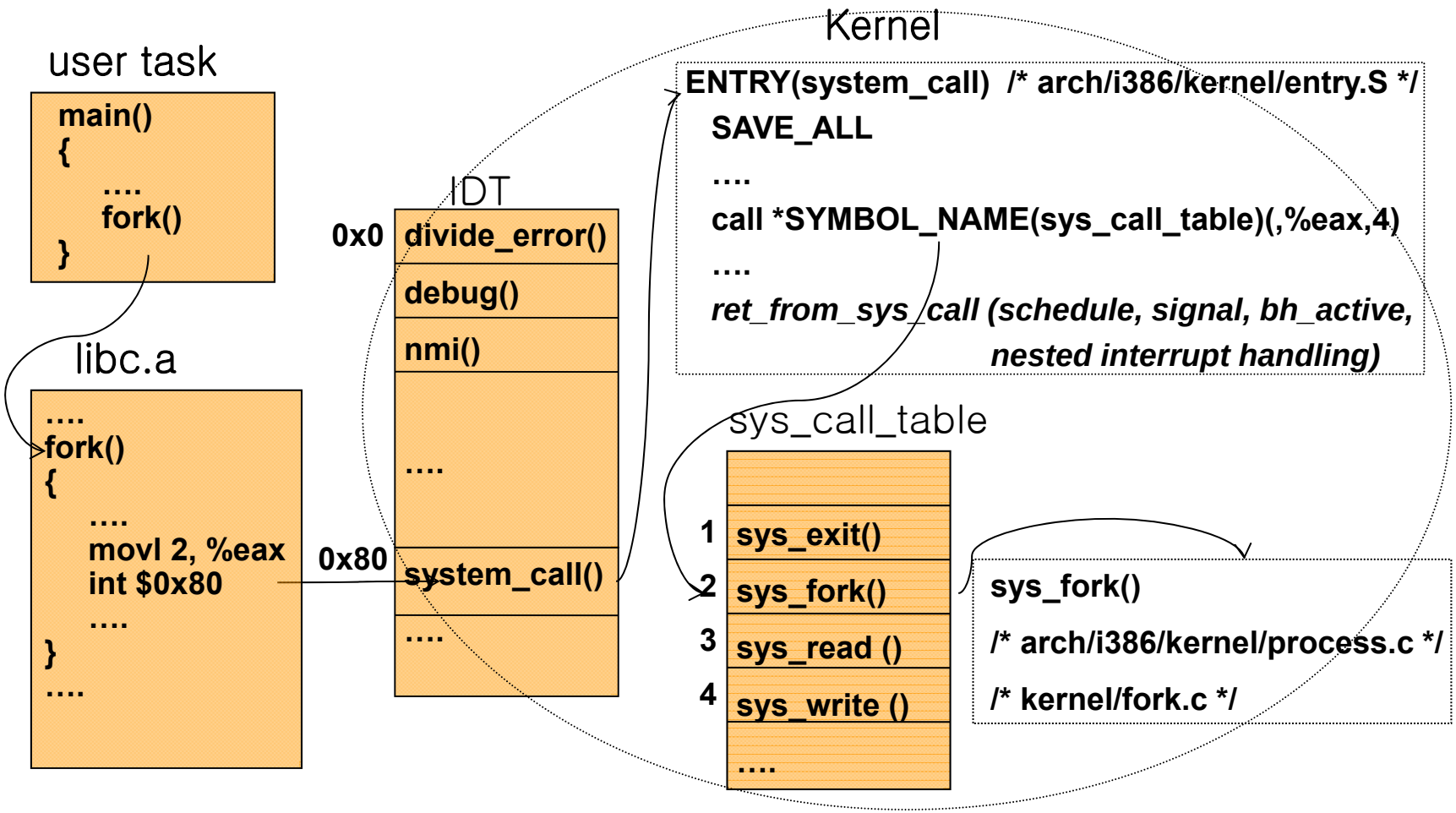
시스템 호출 처리 과정

- 시스템 호출 처리 과정



시스템 호출 처리 과정

- 시스템 호출 처리 과정 예 : fork



시스템 호출 처리 과정

```
ENTRY(system_call)
        pushl %eax                      # save orig_eax
    1   SAVE_ALL
    2   GET_CURRENT(%ebx)
    3   testb $0x02,tsk_ptrace(%ebx)    # PT_TRACESYS
        jne tracesys
    4   cmpl $(NR_syscalls),%eax
        jae badsys
    5   call *SYMBOL_NAME(sys_call_table)(,%eax,4)
        movl %eax,EAX(%esp)             # save the return
```

```
#define SAVE_ALL \
        cld; \
        pushl %es; \
        pushl %ds; \
        pushl %eax; \
        pushl %ebp; \
        pushl %edi; \
        pushl %esi; \
        pushl %edx; \
        pushl %ecx; \
        pushl %ebx; \
        movl $(__KERNEL_DS),%edx; \
        movl %edx,%ds; \
        movl %edx,%es;
```

넘어온 매개변수를 커널 모드 스택에 저장

```
badsys:
        movl $-ENOSYS,EAX(%esp)
        jmp ret_from_sys_call
```

1. 제어 유닛이 자동으로 저장한 eflags, cs, eip, ss, esp제외한 모든 reg를 스택에 저장
2. ebx reg에 current P의 디스크립터를 저장
3. current의 ptrace필드에 PT_TRACESYS플래그가 들어 있는지, 즉 디버거가 프로그램의 시스템콜 호출을 추적 중인지 검사 ➔ syscall_trace()를 처음, 마지막 두번 호출하게 됨
4. 올바른 syscall번호인지 검사. 잘못된 번호이면 바로 종료
5. dispatch table의 각 엔트리는 4바이트이므로, 시스템 콜 번호에 4를 곱한 후 + sys_call_table 시작주소를 더해서 ➔ 서비스 루틴의 ptr얻어와서 호출함 . 호출 종료되면 리턴값을 저장한 스택(사용자 모드에서의 eax)에 저장해놓고, syscall핸들러를 종료하는 ret_from_sys_call로 점프

시스템 호출 처리 과정

```
ENTRY(ret_from_sys_call)
        cli                             # need_resched and signals atomic test
        cmpl $0,need_resched(%ebx)
        jne reschedule
        cmpl $0,sigpending(%ebx)
        jne signal_return
restore_all:
        RESTORE_ALL
```

```
tracesys:
        movl $-ENOSYS,EAX(%esp)
        call SYMBOL_NAME(syscall_trace)
        movl ORIG_EAX(%esp),%eax
        cmpl $(NR_syscalls),%eax
        jae tracesys_exit
        call *SYMBOL_NAME(sys_call_table)(,%eax,4)
        movl %eax,EAX(%esp)             # save the return value
```

```
tracesys_exit:
        call SYMBOL_NAME(syscall_trace)
        jmp ret_from_sys_call
```

```
signal_return:
        sti                             # we can get here from an interrupt handler
        testl $(VM_MASK),EFLAGS(%esp)
        movl %esp,%eax
        jne v86_signal_return
        xorl %edx,%edx
        call SYMBOL_NAME(do_signal)
        jmp restore_all
```

시스템 호출 구현

- system_call() → sys_system_call()
- asmlinkage
  - ✓ Asm내에서 C 함수를 호출할 수 있도록 함
- 시스템 콜 번호
  - ✓ 각 시스템 콜에 시스템콜 번호 부여. 고유한 숫자
  - ✓ sys_call_table에 저장하고 있다
- 시스템 콜 핸들러
  - ✓ int $0x80 ➔ 128번 vector의, 프로그래밍에 의한 예외발생
  - ✓ 순서
    - 시스템 콜 사용위해 커널에 인터럽트를 건다(eax에 syscall번호)
    - reg를 커널 모드 스택에 저장
    - 시스템콜 서비스 루틴 호출하여 처리
    - ret_from_sys_call()로 핸들러서 빠져 나옴
  - ✓ 적법한 매개변수, 퍼미션 인지 검사 필수
  - ✓ copy_to_user(), copy_from_user()사용

시스템 호출 구현

- 시스템 콜 컨텍스트
  - ✓ 이때 **current** 포인터는 시스템 콜을 호출한 프로세스를 가리킴
  - ✓ 시스템 콜을 실행하는 동안 커널은 프로세스 컨텍스트에 존재
  - ✓ 따라서 휴면가능, 완전히 선점 가능
    - 휴면 가능하므로, 시스템 콜은 커널의 대부분의 기능을 사용가능
    - 선점 가능하므로, **re-entrant**해야 함

- 최대한 빠르고, 더 이상 간단할 수 없도록 구현

```
/**
 * sys_getpid - return the thread group id of the current process
 *
 * Note, despite the name, this returns the tgid not the pid.  The tgid and
 * the pid are identical unless CLONE_THREAD was specified on clone() in
 * which case the tgid is the same in all threads of the same group.
 *
 * This is SMP safe as current->tgid does not change.
 */
SYSCALL_DEFINE0(getpid)
{
        return task_tgid_vnr(current);
}
```

시스템 호출 구현

- ✓ 매개 변수 확인
  - ▪ 매개 변수가 주소인 경우 검사 방법 두 가지
    - · 선형 주소가 P 주소 공간에 속하는지,속하면 접근 권한이 있는지 검사
    - · **선형 주소가 PAGE_OFFSET보다 낮은지 만 확인**
  - ▪ verify_area()함수이용 system call에 전달한 주소 검사(=access_ok 매크로)
- ✓ 프로세스 주소 공간 접근

Function	Action
get_user _ _get_user	Reads an integer value from user space (1, 2, or 4 bytes)
put_user _ _put_user	Writes an integer value to user space (1, 2, or 4 bytes)
copy_from_user _ _copy_from_user	Copies a block of arbitrary size from user space
copy_to_user _ _copy_to_user	Copies a block of arbitrary size to user space
strncpy_from_user _ _strncpy_from_user	Copies a null-terminated string from user space
strlen_user strnlen_user	Returns the length of a null-terminated string in user space
clear_user _ _clear_user	Fills a memory area in user space with zeros

새로운 시스템 호출 구현 (1/9)

- 새로운 시스템 호출 구현

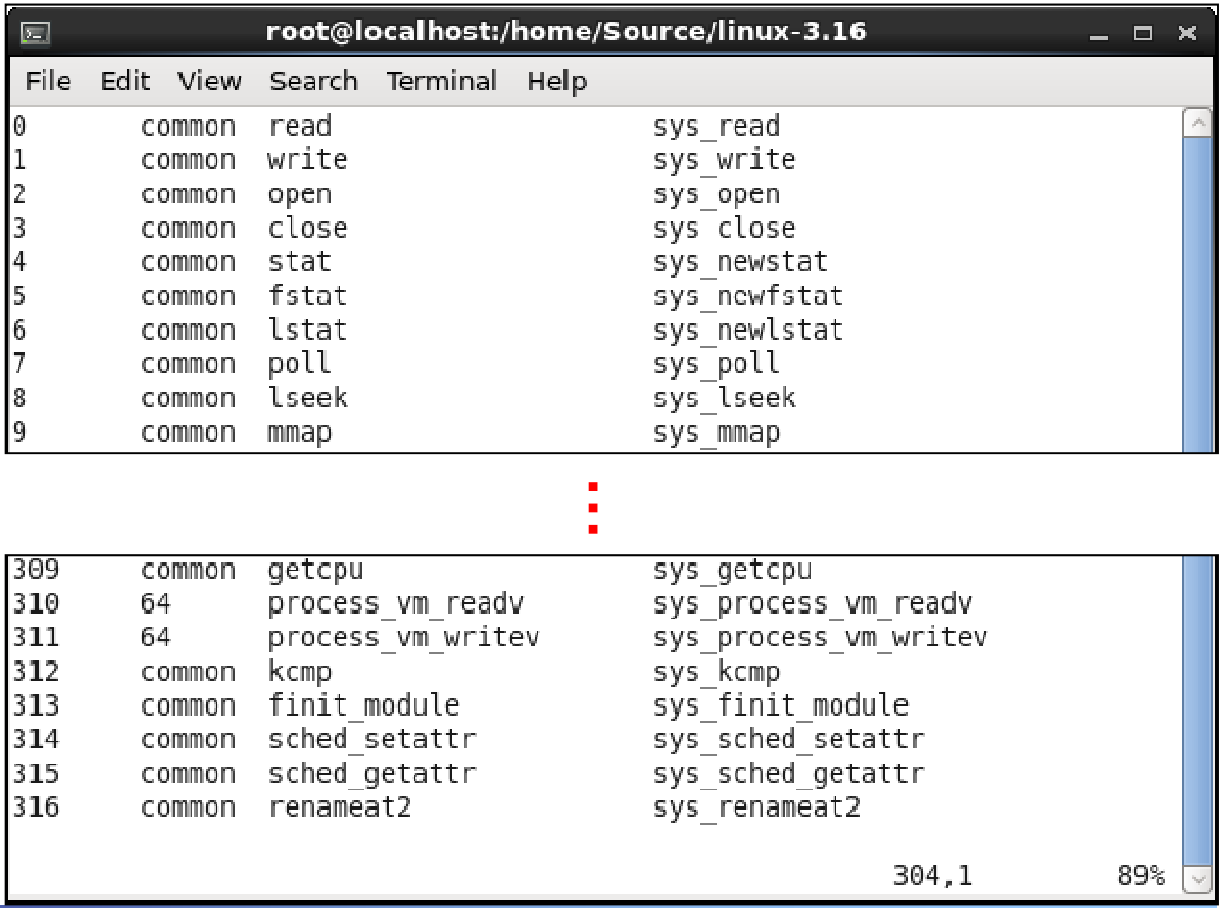
  - ✓ 커널 수정 : 4 단계
    1. 새로운 시스템 호출 번호 할당 (allocate syscall_number)
    2. 새로운 시스템 호출 함수 sys_call_table[]에 등록
    3. 새로운 시스템 호출 함수 커널에 구현
    4. 커널 컴파일 및 리부팅

  - ✓ 사용자 응용 작성 : 2단계
    1. 새로운 시스템 호출을 사용하는 사용자 수준 응용 작성
    2. 라이브러리로 사용자 응용 작성 (optional) : ar, ranlib

새로운 시스템 호출 구현 (2/9)

- 새로운 시스템 호출 구현 예 : newsyscall() 이라는 이름의 새로운 시스템 호출 구현

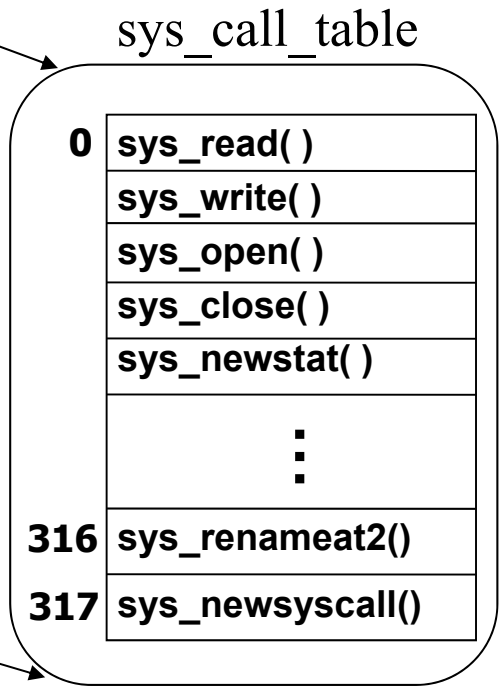
1. 새로운 시스템 호출 번호 할당(~/arch/x86/syscalls/syscall_64.tbl)



새로운 시스템 호출 구현 (3/9)

2. 새로운 시스템 호출 함수 sys_call_table[]에 등록

```
/* arch/x86/kernel/syscalls/syscall_64.tbl 파일의 내용 */
0    common      read          sys_read
1    common      write         sys_write
2    common      open          sys_open
3    common      close         sys_close
4    common      stat          sys_newstat
5    common      fstat         sys_newfstat
…
316  common      renameat2     sys_renameat2
317  common      newsyscall    sys_newsyscall
```



새로운 시스템 호출 구현 (5/9)

3. 새로운 시스템 호출 함수 커널에 구현

```
/* include/linux/syscalls.h 파일의 내용 */
…
asmlinkage long        sys_fork(void);
asmlinkage long        sys_vfork(void);
asmlinkage long        sys_newsyscall(void);
…
```

```
/* kernel/newfile.c 파일의 내용 */
#include <linux/unistd.h>
#include <linux/errno.h>
#include <linux/kernel.h>
#include <linux/sched.h>

asmlinkage long sys_newsyscall(void)
{
        printk("<0>Hello Linux, I'm in Kernel\n");
        return 0;
}
EXPORT_SYMBOL_GPL(sys_newsyscall);
```

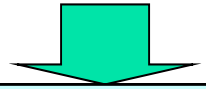
- printf()가 아니라 printk() 임에 주의

새로운 시스템 호출 구현 (6/9)

4. 커널 컴파일 및 리부팅

- 커널 컴파일 전에 makefile을 다음과 같이 수정해야 한다.

```
/* kernel/Makefile 의 변경 전 내용 */
obj-y    = fork.o exec_domain.o panic.o \
           cpu.o exit.o itimer.o time.o softirq.o resource.o \
           sysctl.o sysctl_binary.o capability.o ptrace.o timer.o user.o \
           signal.o sys.o kmod.o workqueue.o pid.o task_work.o \
           extable.o params.o posix-timers.o \
           kthread.o sys_ni.o posix-cpu-timers.o \
           hrtimer.o nsproxy.o \
           notifier.o ksysfs.o cred.o reboot.o \
           async.o range.o groups.o smpboot.o
```



```
/* kernel/Makefile 의 변경 전 내용 */
obj-y    = fork.o exec_domain.o panic.o \
           cpu.o exit.o itimer.o time.o softirq.o resource.o \
           sysctl.o sysctl_binary.o capability.o ptrace.o timer.o user.o \
           signal.o sys.o kmod.o workqueue.o pid.o task_work.o \
           extable.o params.o posix-timers.o \
           kthread.o sys_ni.o posix-cpu-timers.o \
           hrtimer.o nsproxy.o \
           notifier.o ksysfs.o cred.o reboot.o \
           async.o range.o groups.o smpboot.o newfile.o
```

- 커널 컴파일 후 재부팅

새로운 시스템 호출 구현 (7/9)

1. 새로운 시스템 호출을 사용하는 사용자 수준 응용 작성

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

int main(void)
{
        syscall(325);
        return 0;
}
```

```
301 #define __NR_rt_tgsigqueueinfo 297
302 #define __NR_perf_event_open 298
303 #define __NR_recvmmsg 299
304 #define __NR_fanotify_init 300
305 #define __NR_fanotify_mark 301
306 #define __NR_prlimit64 302
307 #define __NR_name_to_handle_at 303
308 #define __NR_open_by_handle_at 304
309 #define __NR_clock_adjtime 305
310 #define __NR_syncfs 306
311 #define __NR_sendmmsg 307
312 #define __NR_setns 308
313 #define __NR_getcpu 309
314 #define __NR_process_vm_readv 310
315 #define __NR_process_vm_writev 311
316 #define __NR_kcmp 312
317 #define __NR_finit_module 313
318 #define __NR_newsyscall 317
319
320 #endif /* _ASM_X86_UNISTD_64_H */
~
~
~
unistd_64.h
```

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

int main(void)
{
        syscall(__NR_newsyscall);
        return 0;
}
```

syscall 매크로

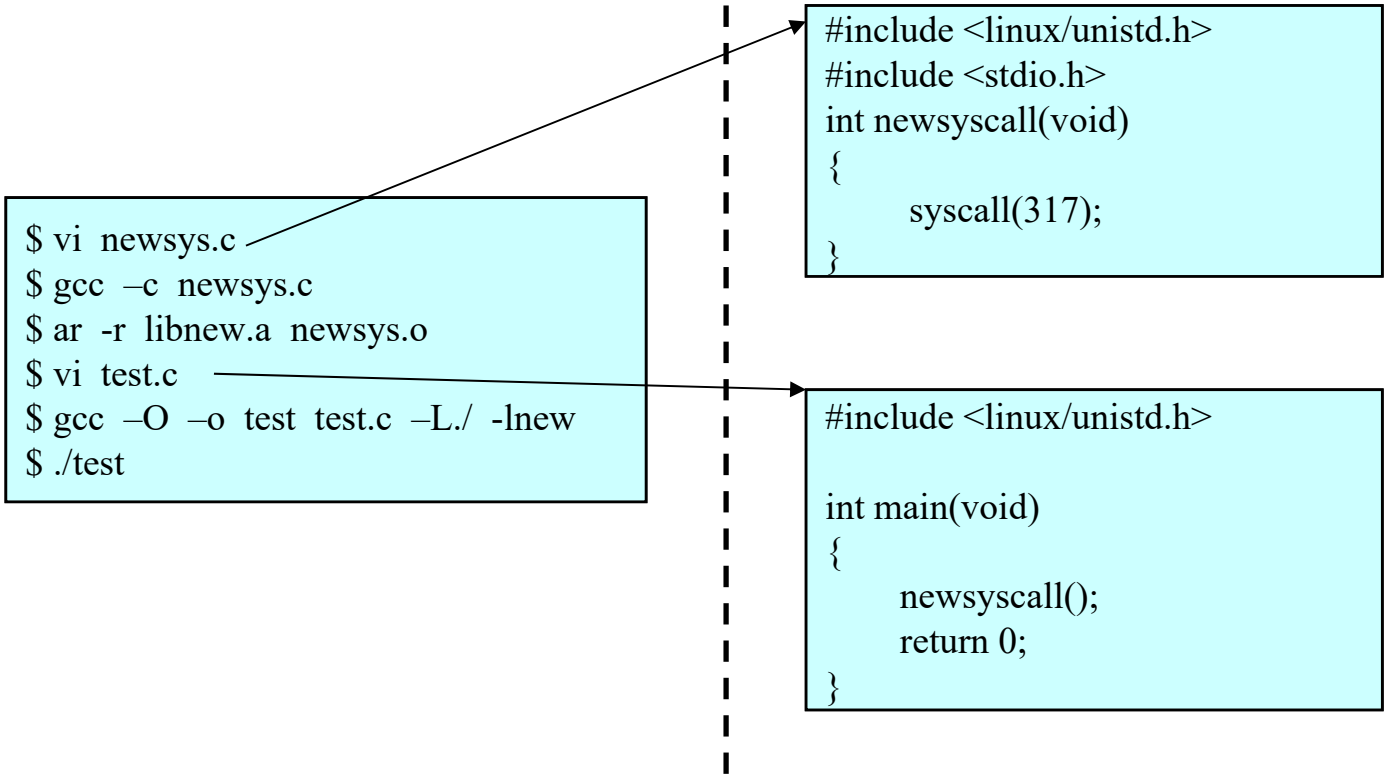
```
/* /usr/include/unistd.h */
extern long int syscall (long int __sysno, ...) __THROW;
```

```
/* glibc/sysdeps/unix/sysv/linux/i386/syscall.S */
 .text
ENTRY (syscall)

      PUSHARGS_6            /* Save register contents.  */
      _DOARGS_6(44)         /* Load arguments.  */
      movl 20(%esp), %eax   /* Load syscall number into %eax.  */
      ENTER_KERNEL          /* Do the system call.  */
      POPARGS_6             /* Restore register contents.  */
      cmpl $-4095, %eax     /* Check %eax for error.  */
      jae SYSCALL_ERROR_LABEL /* Jump to error handler if error.  */
L(pseudo_end):
      ret                   /* Return to caller.  */

PSEUDO_END (syscall)
```

```
/* glibc/sysdeps/unix/sysv/linux/i386/sysdep.h */
# define ENTER_KERNEL int $0x80
```

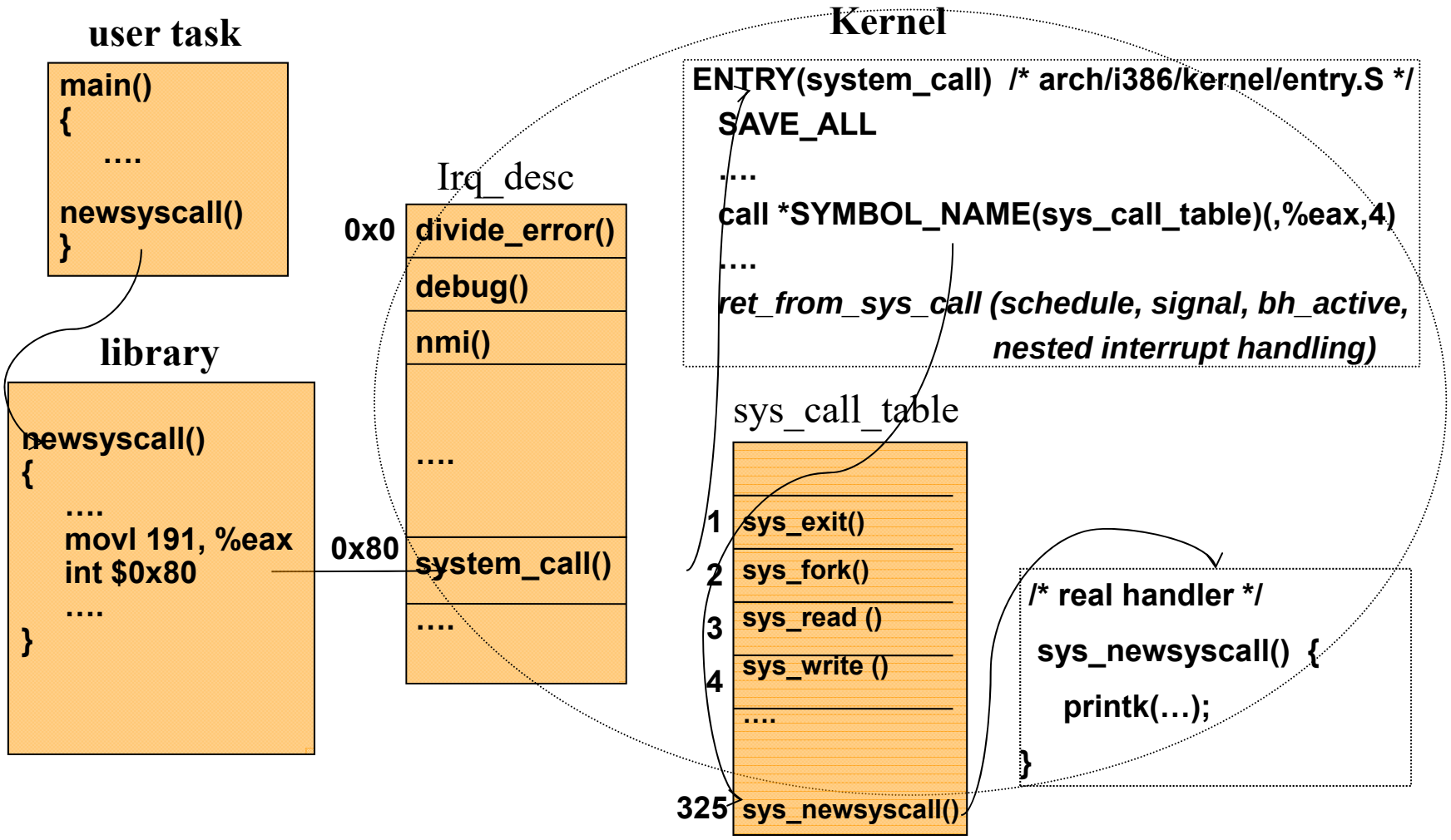
새로운 시스템 호출 구현 (8/9)

2. 라이브러리로 사용자 응용 작성



새로운 시스템 호출 구현 (9/9)

- 새로운 시스템 호출 처리 과정



가상주소 변환

```
#include <linux/unistd.h>
#include <linux/errno.h>
#include <linux/kernel.h>
#include <linux/sched.h>
#include <linux/mm_types.h>
#include <linux/hugetlb.h>
#include <linux/mm.h>

asmlinkage long sys_newsyscall(unsigned long address)
{
	struct vm_area_struct *vma = find_vma(current->mm, address);
	unsigned int flags = 0;
	unsigned char * pa;
	pgd_t *pgd;	pud_t *pud;	pmd_t *pmd;
	pte_t *ptep, pte;	spinlock_t *ptl;	struct page *page;
	struct mm_struct *mm = current->mm;
	page = follow_huge_addr(mm, address, flags & FOLL_WRITE);
	if (!IS_ERR(page)) {
		BUG_ON(flags & FOLL_GET);
		goto out;
	}
	page = NULL;
	pgd = pgd_offset(mm, address);
	if (pgd_none(*pgd) || unlikely(pgd_bad(*pgd)))
		goto no_page_table;
	pud = pud_offset(pgd, address);
	if (pud_none(*pud))
		goto no_page_table;
	if (pud_huge(*pud)) {
		BUG_ON(flags & FOLL_GET);
		page = follow_huge_pud(mm, address, pud, flags &
FOLL_WRITE);
		goto out;
	}
	if (unlikely(pud_bad(*pud)))
		goto no_page_table;
	pmd = pmd_offset(pud, address);
	if (pmd_none(*pmd))	goto no_page_table;
	if (pmd_huge(*pmd)) {
	   BUG_ON(flags & FOLL_GET);
	   page = follow_huge_pmd(mm, address, pmd, flags & FOLL_WRITE);
	   goto out;
	}
	if (unlikely(pmd_bad(*pmd)))	goto no_page_table;
	ptep = pte_offset_map_lock(mm, pmd, address, &ptl);
	pte = *ptep;
	if (!pte_present(pte))  goto no_page;
	if ((flags & FOLL_WRITE) && !pte_write(pte))  goto unlock;
	page = vm_normal_page(vma, address, pte);
	if (unlikely(!page)) {
		if ((flags & FOLL_DUMP) ) goto bad_page;
		page = pte_page(pte);
	}
	if (flags & FOLL_GET)	get_page(page);
	pa = (unsigned char *)page_address(page);
	pa += address & (PAGE_SIZE - 1);
	printk("<0>[%s]: arg VA=[%x]\n", __FUNCTION__, address);
	printk("<0>[%s]: PGD(%p)=%x\n", __FUNCTION__, pgd, *pgd);
	printk("<0>[%s]: PUD(%p)=%x\n", __FUNCTION__, pud, *pud);
	printk("<0>[%s]: PMD(%p)=%x\n", __FUNCTION__, pmd, *pmd);
	printk("<0>[%s]: PTE(%p)=%x\n", __FUNCTION__, ptep, *ptep);
	printk("<0>[%s]: result PA=[%x], data=%d\n", __FUNCTION__, pa,
*(int *)pa);
unlock:	pte_unmap_unlock(ptep, ptl);
out:	return page;
bad_page:
	pte_unmap_unlock(ptep, ptl);
	return ERR_PTR(-EFAULT);
no_page:
	pte_unmap_unlock(ptep, ptl);
	if (!pte_none(pte))	return page;
no_page_table:
	if ((flags & FOLL_DUMP) &&
		(!vma->vm_ops || !vma->vm_ops->fault))
		return ERR_PTR(-EFAULT);
	return page;
```

가상 주소 변환 결과


```c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
        int a;
        a = 1869;
        syscall(299, &a);
        return 0;
}
```

```
[root@localhost ~]# ./ts
[root@localhost ~]# dmesg | tail -n 5
<0>[sys_hycall]: PGD(ffff88086e9b17f8)=6e628067
<0>[sys_hycall]: PUD(ffff88086e628ff0)=6df38067
<0>[sys_hycall]: PMD(ffff88086df38740)=6e586067
<0>[sys_hycall]: PTE(ffff88086e586f58)=5e30a067
<0>[sys_hycall]: result PA=[5e30aba0], data=1383
[root@localhost ~]#
```

Offset in the page frame

시스템 호출 구현 확장 (1/9)

- 인자 전달
- 기존 시스템 호출 분석
- 커널 정보 얻기
- 모듈 프로그래밍을 이용한 시스템 호출 구현 **=>** 모듈 프로그래밍 장 참조

☞ ***Just Do It*** (百見不如一打)

시스템 호출 구현 확장 (2/9)

- 인자 전달 : show_mult(arg1, arg2, result)
  1. 새로운 시스템 호출 번호 할당 : 192번
  2. 새로운 시스템 호출 함수 sys_call_table[]에 등록
  3. 새로운 시스템 호출 함수 커널에 구현

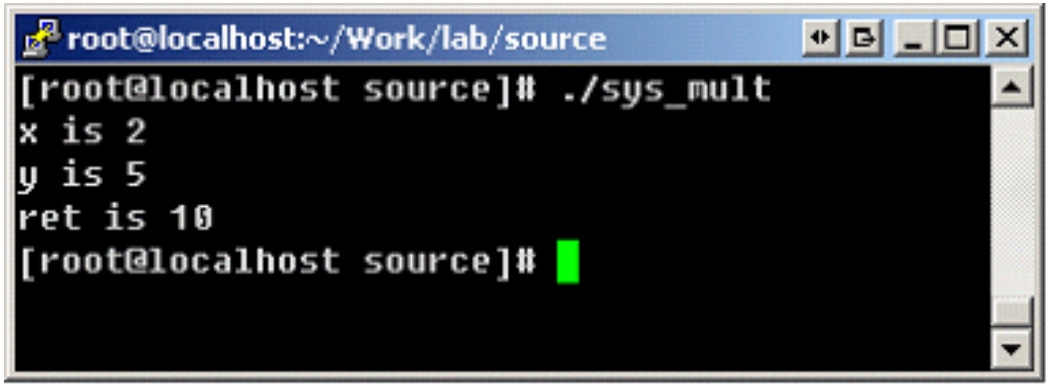
```
#include<linux/unistd.h>
#include<linux/kernel.h>
#include<asm-x86/uaccess.h>
asmlinkage int sys_show_mult(int x, int y, int* res)
{
   int error, compute;
   int i;
    error = access_ok(VERIFY_WRITE,res,sizeof(*res));
   if(error < 0)
   {
      printk("error in cdang\n");
      printk("error is %d\n",error);
      return error;
   }
   compute = x*y;
   printk("computeis %d\n",compute);
   i=  copy_to_user(res,&compute,sizeof(int));
   return 0;
}
```

  4. 커널 컴파일 및 리부팅

시스템 호출 구현 확장 (3/9)

- 인자 전달 : show_mult(arg1, arg2, result)
  1. 사용자 수준 응용

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

int main(void)
{
   int mult_ret = 0;
   int x = 2,y=5;
   int i;
   i=syscall(325,x,y,&mult_ret);
   printf("x is %d\ny is %d\nret is %d\n",x,y,mult_ret);

   return 0;
}
```



시스템 호출 구현 확장 (4/9)

- 커널 정보 얻기 : gettaskinfo()
  - ✓ header

```
#include<linux/kernel.h>
#include<linux/sched.h>
#include <linux/slab.h>
#include <linux/uaccess.h>
#include <linux/fs.h>
#include <linux/fdtable.h>
struct mystat
{
    pid_t pid;
    pid_t ppid;
    int stat;
    int priority;
    int policy;
    long utime;
    long stime;
    long starttime;
    unsigned long min_flt;
    unsigned long maj_flt;
    long open_files;
};
```

시스템 호출 구현 확장 (5/9)

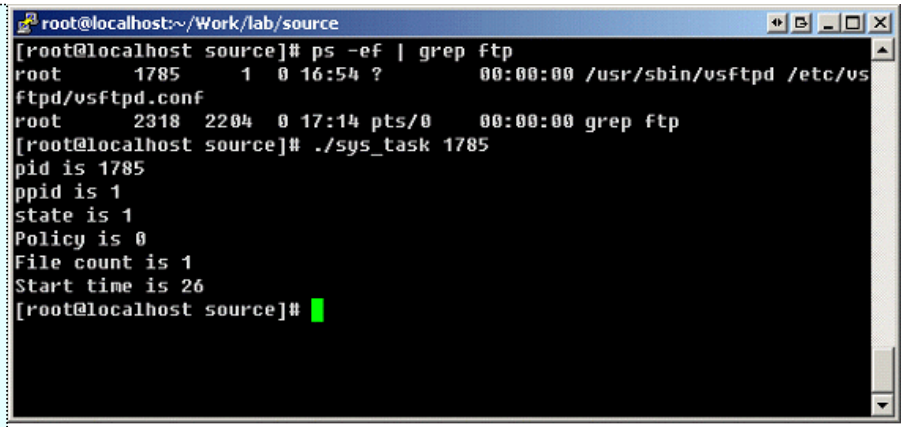
- 커널 정보 얻기 : gettaskinfo()
  - ✓ 커널 함수

```
#include “mystat.h”
asmlinkage int sys_gettaskinfo(int id, struct mystat *user_buf)
{
    long ret = 0;    struct mystat *buf;    int i, cnt ;    struct task_struct *search;    cnt = i = 0;
    search = pid_task(find_vpid(id), PIDTYPE_PID);
    if(search)
        printk(KERN_ERR "search pid: %d₩n", search->pid);
    if(!user_buf->starttime)
        return -1;
    buf = (char *)kmalloc(sizeof(struct mystat),GFP_KERNEL);
    if(buf == NULL)
    {
        printk("[SM] buf is NULL₩n");
        return -1;
    }
    buf->pid = search->pid;
    buf->ppid = search->parent->pid;
    buf->stat = search->state;
    buf->priority = search->prio;
    buf->policy = search->policy;
    buf->utime = search->utime;
    buf->stime = search->stime;
    buf->starttime = search->start_time.tv_sec;
    buf->min_flt = search->min_flt;
    buf->maj_flt = search->maj_flt;
    for(i=0; i<32; i++)
    {
        if((search->files->fd_array[i]) != NULL)
        {
            cnt++;
        }
    }
    buf->open_files = cnt;
    ret = copy_to_user(user_buf,buf,sizeof(struct mystat));
    printk(KERN_ERR "[SM] copy_to_user return: %ld₩n", ret);
    return 0;
}
```

시스템 호출 구현 확장 (6/9)

- 커널 정보 얻기 : gettaskinfo()
  - ✓ 사용자 수준 응용

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include "mystat.h"

int main(int argc, char * argv[])
{
    int task_number;
    struct mystat* mybuf;
    if(argc != 2)
    {
        printf("Usage : a.out pid₩n");
        exit(1);
    }
    task_number = atoi(argv[1]);
    printf("mybuf size : %d₩n",sizeof(mybuf));
    if(mybuf == NULL)
        exit(1);
    syscall(319,task_number,mybuf);
    printf("pid is %d ₩n",(int)mybuf->pid);
    printf("ppid is %d ₩n",(int)mybuf->ppid);
    printf("state is %d ₩n",(int)mybuf->stat);
    printf("Policy is %d ₩n",(int)mybuf->policy);
    printf("File count is %d ₩n",mybuf->open_files);
    printf("Start time is %d ₩n",mybuf->starttime);

    return 0;
}
```



시스템 호출 구현 확장 (7/9)

- 기존 시스템 호출 분석
  - ✓ getpid

```
asmlinkage int sys_getpid() {
    current->tgid;
}
```

all tasks connected using double linked list (next_task, next_run)
global variable: init_task, current
task[0]: init_task, task[1]: init process

  - ✓ nice

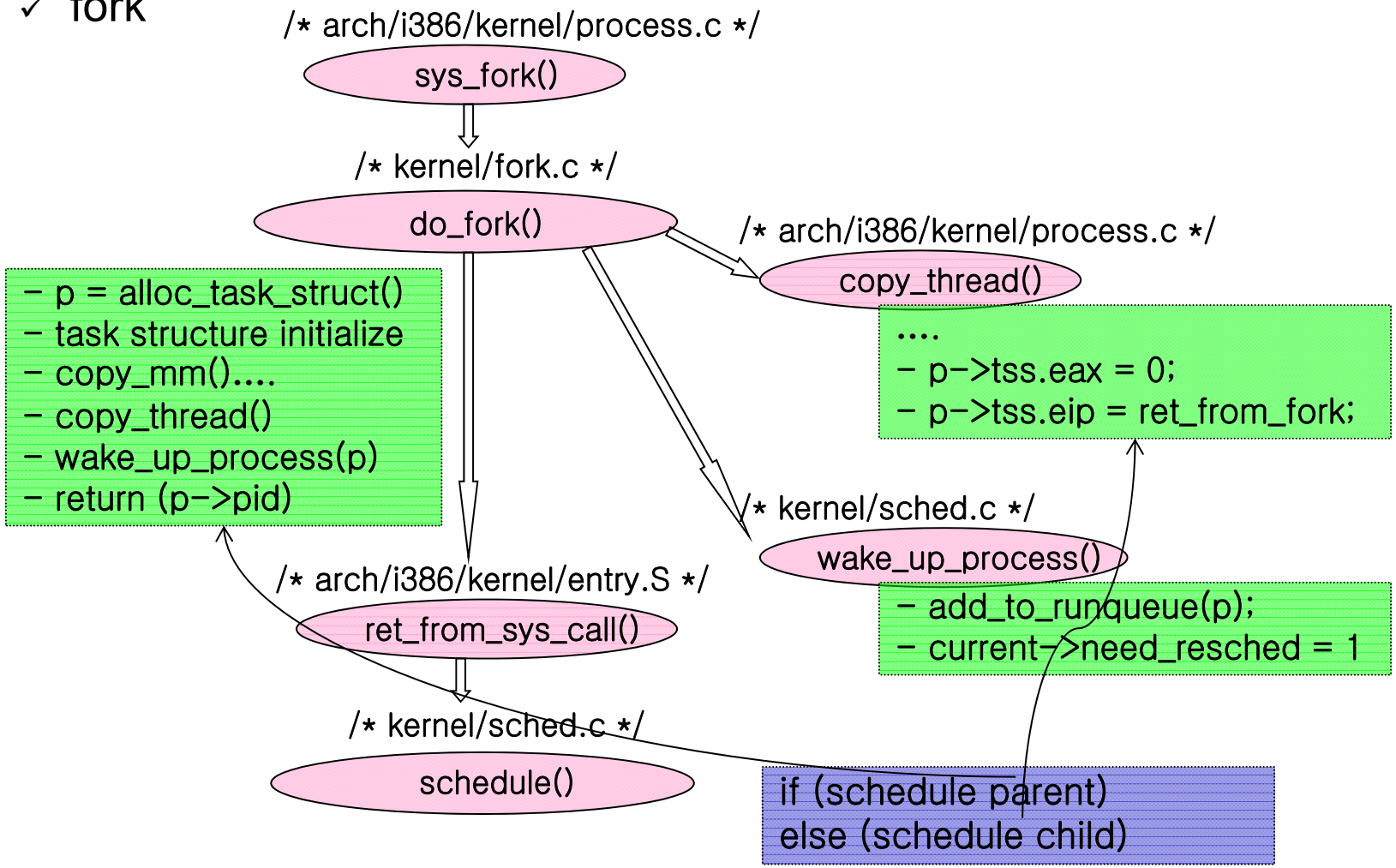
```
asmlinkage int sys_nice(new_priority) {
    ....
    current->priority = newpriority ;
}
```

  - ✓ pause

```
asmlinkage int sys_pause() {
    current->state = TASK_INTTERUPTIBLE;
    schedule();
}
```

시스템 호출 구현 확장 (8/9)

- 기존 시스템 호출 분석
  - ✓ fork



시스템 호출 구현 확장 (9/9)

- 기존 시스템 호출 분석
  - ✓ exit