# Module Programming

# Textbook

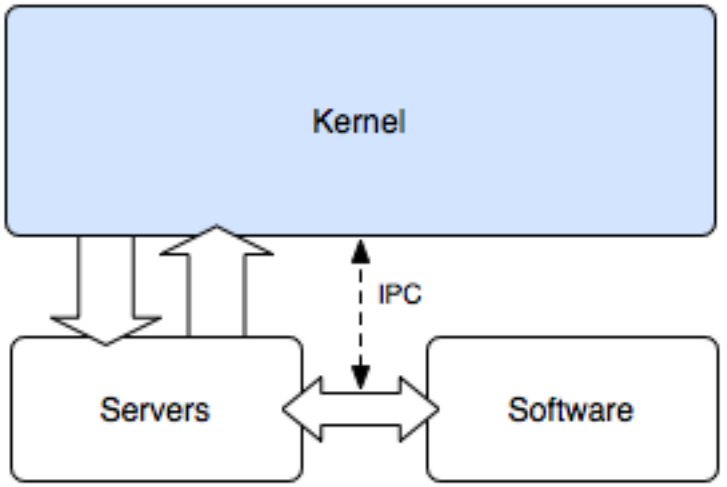
- IT EXPERT 리눅스 커널 프로그래밍, 한동훈,원일용,하홍준 저, 한빛미디어.

# Agenda

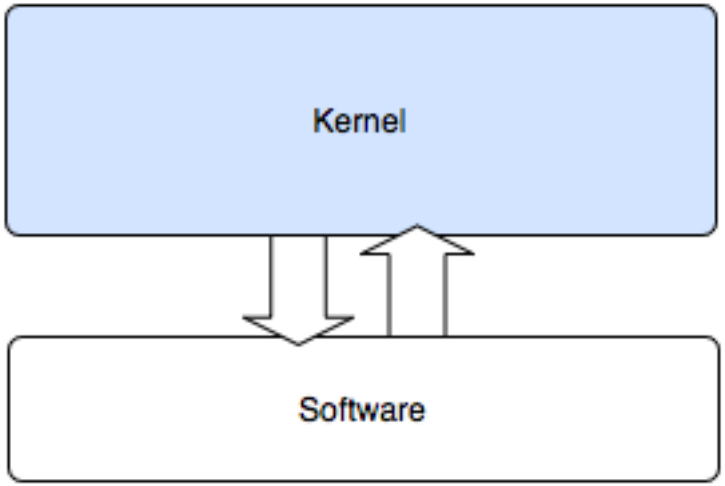
- Module and Monolithic Kernel
- Module development
- Caller and Callee
- Parameters
- Wrapping system calls

# 1. Monolithic Kernel

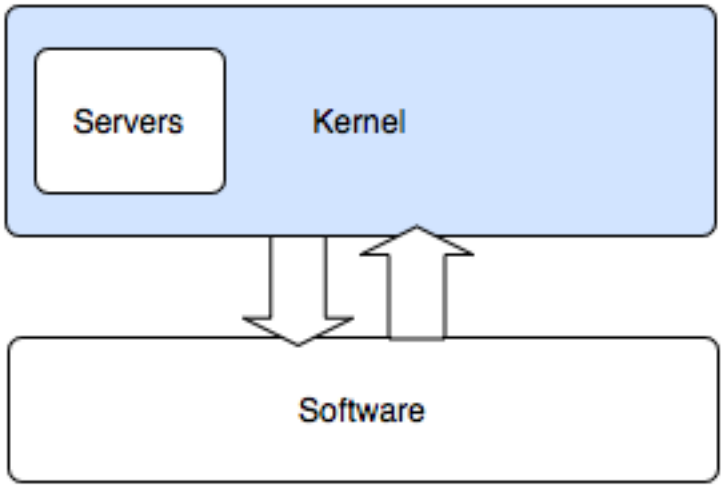
- Types
  - Micro Kernel
    - ex. Mach, GNU Hurd, Minix, K42
  - Monolithic Kernel
    - ex. Linux, BSD, Solaris, MS-DOS
  - Hybrid Kernel
    - ex. Windows NT, BeOS



Micro Kernel



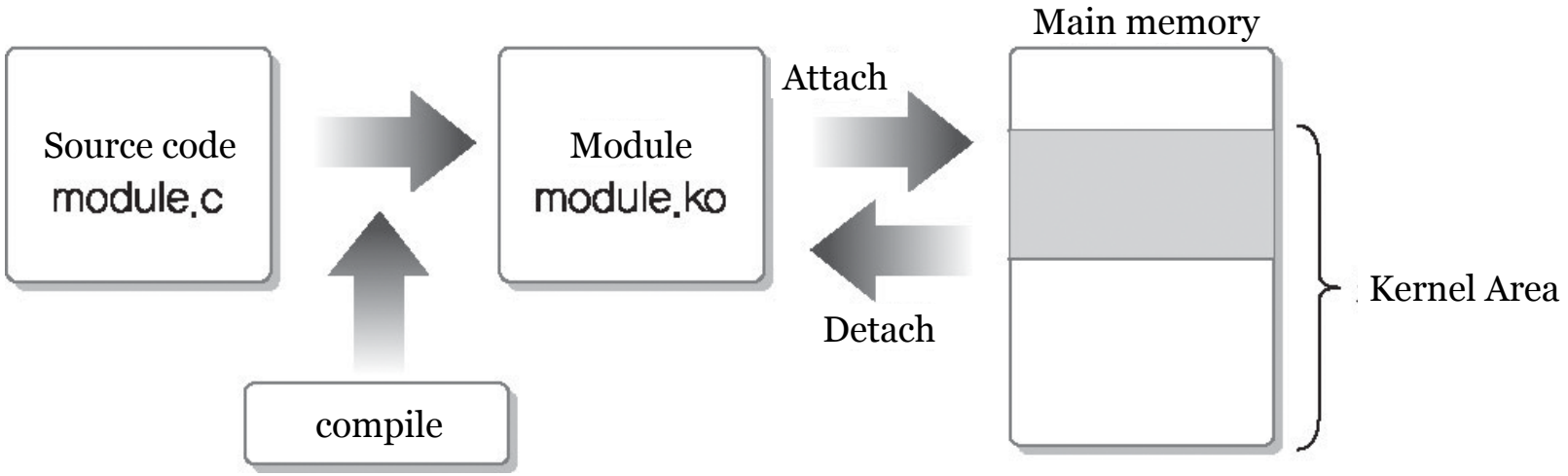
Monolithic Kernel



Hybrid Kernel

# 2. Module Development

- Process
  - Can use kernel functions only



# Module Development Process

- 1. Write a source code
- 2. Compile it
- 3. Insert the module (insmod())
- 4. Check the module (lsmod())
- 5. Remove the module (rmmod())

# Module Program

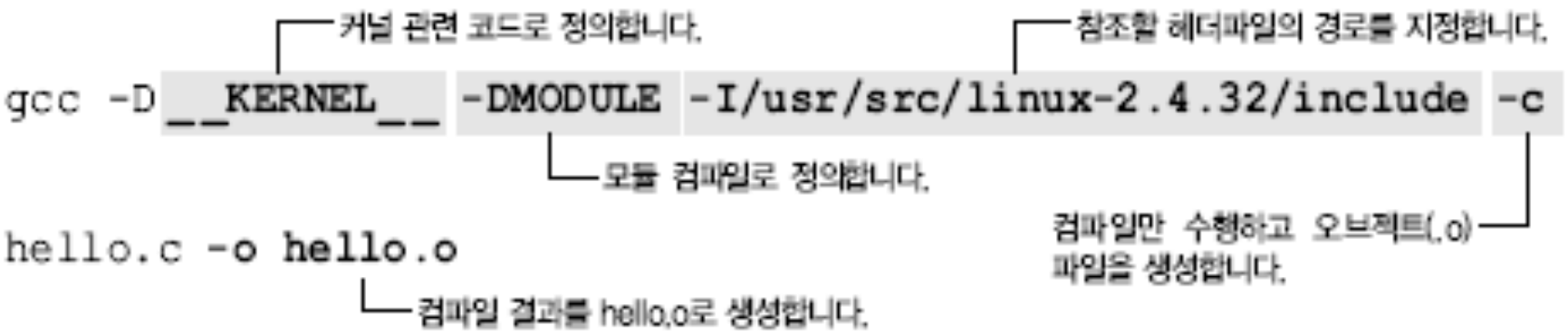
```
// header files
#include <linux/kernel.h>
#include <linux/module.h>

// initialization on module attachment
int __init init_module(void)
{
        ……// initialization code
        return 0;   // success
}
// initialization on module detachment
void __exit cleanup_module(void)
{
        ……// cleanup code
}
MODULE_LICENSE(“GPL”);
```

# Example) Hello Module (Kernel 2.4)

```
// header files
#include <linux/kernel.h>
#include <linux/module.h>

// initialization on module attachment
int __init init_module(void)
{
  printk(KERN_ALERT “[Module Message] Hello, Module\n”);
  return 0;   // success
}
// initialization on module detachment
void __exit cleanup_module(void)
{
  printk(KERN_ALERT “[Module Message] Do you really want to break
        up with me?\n”);
}
MODULE_LICENSE(“GPL”);
```

# Example) Hello Module Compile (Kernel 2.4)

- gcc –D__KERNEL__ -DMODULE –I/usr/src/linux-2.4.32/include –c hello.c –o hello.o



# Example) Hello Module (Kernel 2.6)-1

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

// 초기화 루틴
int __init init_hello(void)
{
  printk( KERN_ALERT "[Module Message] Hello, Module.\n" );
  return 0;
}
```

init.h를 추가합니다.

초기화 루틴 위치입니다.

# Example) Hello Module (Kernel 2.6)-2

```
// 종료 루틴
void __exit exit_hello(void)
{
  printk( KERN_ALERT "[ Module Message]  Do you really want to break
          up with me?\n" );
}

module_init( init_hello );

module_exit( exit_hello );

MODULE_LICENSE( "GPL" );
```

종료 루틴 위치입니다.

초기화 루틴 위치입니다.

종료 루틴 위치입니다.

# Example) Hello Module Compile (Kernel 2.6)-1

- Makefile

```
KERNELDIR = /lib/modules/$(shell uname -r)/build

obj-m = hello.o

KDIR  := /lib/modules/$(shell uname -r)/build
PWD := $(shell pwd)

default:
	$(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules

clean:
	rm -rf *.ko
	rm -rf *.mod.*
	rm -rf .*.cmd
	rm -rf *.o
```

탭으로 띄어쓰기 해야 합니다.

# Example) Hello Module Compile (Kernel 2.6)-2

- [make] command to build the module

```
coffee:~/works/chap04/2.6# make
make -C /lib/modules/2.6.14.6/build SUBDIRS=/root/works/chap04/2.6
modules
make[1]: Entering directory '/usr/src/linux-2.6.14.6'
  CC [M]  /root/works/chap04/2.6/hello.o
  Building modules, stage 2.
  MODPOST
  CC      /root/works/chap04/2.6/hello.mod.o
  LD [M]  /root/works/chap04/2.6/hello.ko
                                          └──── 2.6 커널의 hello 모듈
make[1]: Leaving directory '/usr/src/linux-2.6.14.6'
```

# Running the Hello Module

- Check
  - [lsmod] command
- Insert
  - insmod hello.o (Kernel 2.4)
  - insmod hello.ko (Kernel 2.6)
- Remove
  - rmmod hello

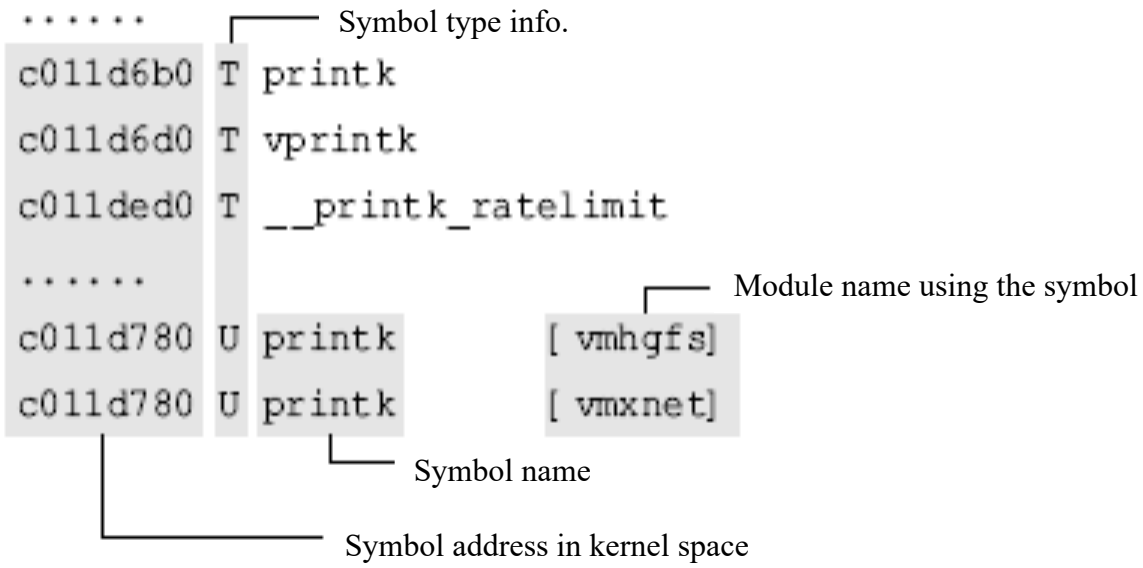
# lsmod() ex.

```
kwon@kwon:~$ lsmod
Module                  Size  Used by
btrfs                 987136  0
xor                    24576  1 btrfs
raid6_pq              102400  1 btrfs
xfs                   970752  0
libcrc32c              16384  1 xfs
nls_iso8859_1          16384  1
snd_hda_codec_hdmi     53248  1
snd_hda_codec_realtek  86016  1
snd_hda_codec_generic  77824  1 snd_hda_codec_realtek
snd_hda_intel          36864  5
snd_hda_codec         135168  4 snd_hda_codec_realtek,snd_hda_codec_hdmi,snd_hda_codec_generic,snd_hda_intel
```

# Kernel Symbols

- Functions and variables available in Kernel area
- Symbol defined in
  - /proc/ksyms (Kernel 2.4)
  - /proc/kallsyms (Kernel 2.6)

```
coffee:~# grep printk /proc/kallsyms
......
c011d6b0 T printk
c011d6d0 T vprintk
c011ded0 T __printk_ratelimit
......
c011d780 U printk       [ vmhgfs]
c011d780 U printk       [ vmxnet]
```



# Symbol Types

- Upper case : global symbol
- Lower case : local symbol

| Symbol type | Description |
|---|---|
| A | Absolute address. Never modified. |
| B | bss symbol area |
| C | Uninitialized common symbol. |
| D | Initialized data symbol. |
| R | Read-only variable. (rdata area) |
| T | Text symbol (function). |
| U | Undefined symbol. |
| I | Indirect symbol (GNU extension). |
| N | Debugging symbol. |

# Symbol Macros

| Symbol Macro | Description |
| --- | --- |
| EXPORT_SYMBOL(var) | Export a symbol. |
| EXPORT_SYMBOL_NOVERS(var) | Export a symbol without version info. |
| EXPORT_SYMBOL_GPL(var) | Export a symbol to only a module with GPL license. |
| EXPORT_NO_SYMBOLS | Export no symbol. |

# Module License

- To denote the license of the module, use MODULE_LICENSE()
  - ex. MODULE_LICENSE( “GPL” );

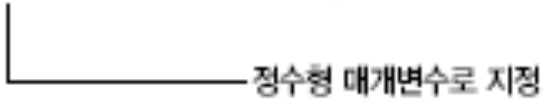
| License Type | Description |
|---|---|
| GPL | GNU Public License v2 또는 이상 |
| GPL v2 | GNU Public License v2 |
| GPL and additional rights | GNU Public License v2 rights and more |
| Dual BSD/GPL | GNU Public License v2 또는 BSD 라이선스 선택 |
| Dual MPL/GPL | GPL v2 또는 모질라 라이선스 선택 |
| Proprietary | Non-free S/W |

# Parameters

- Kernel 2.4
  - MODULE_PARM()
- Kernel 2.6.16
  - MODULE_PARM() or module_param()
- Kernel 2.6.17 or more
  - module_param()

# Example 1-1

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

int a;                          // 변수 a를 선언
MODULE_PARM( a, "i" );          // 변수 a를 int형 커널 매개변수로 사용

// 초기화 루틴
int __init init_param( void )
{
  printk( KERN_ALERT "[Module Message] parameter a = %d\n", a );
  return 0;
}
```



# Example 1-2

```
// 정리 루틴
void __exit exit_param( void )
{
}

module_init(  init_param   );
module_exit(  exit_param   );

MODULE_LICENSE( "GPL" );
```

# Example Test

```
coffee:~/works/chap04/2.6# insmod param1.ko
[Module Message] parameter a = 0
```

Case of not conveying the parameter value

```
coffee:~/works/chap04/2.6# insmod param1.ko a=300
[Module Message] parameter a = 300
```

Case of conveying the parameter value

# Example-1

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

int a;                          // 변수 a를 선언
char *str;                      // 문자열을 저장할 변수 str을 선언
MODULE_PARM( a, "i" );          // 변수 a를 int형 커널 매개변수로 사용
MODULE_PARM( str, "s" );        // 변수 str을 char * 형 커널 매개변수로 사용
```

# Example-2

```
// 초기화 루틴
int __init init_param( void )
{
  printk( KERN_ALERT "[Module Message] parameter a = %d\n", a );
  printk( KERN_ALERT "[Module Message] parameter str = %s\n", str );
  return 0;
}

// 정리 루틴
void __exit exit_param( void )
{
}

module_init( init_param );
module_exit( exit_param );

MODULE_LICENSE( "GPL" );
```

# Example Test

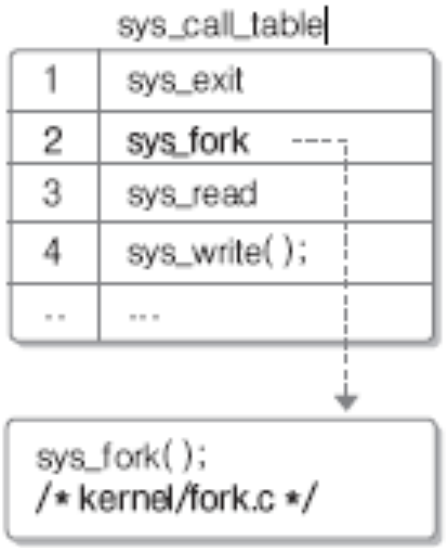
```
coffee:~/works/chap04/2.6# insmod param2.ko
[Module Message] parameter a = 0
[Module Message] parameter str = <NULL>

coffee:~/works/chap04/2.6# insmod param2.ko a=300 str=Hello
[Module Message] parameter a = 300
[Module Message] parameter str = Hello
```

# Argument Types

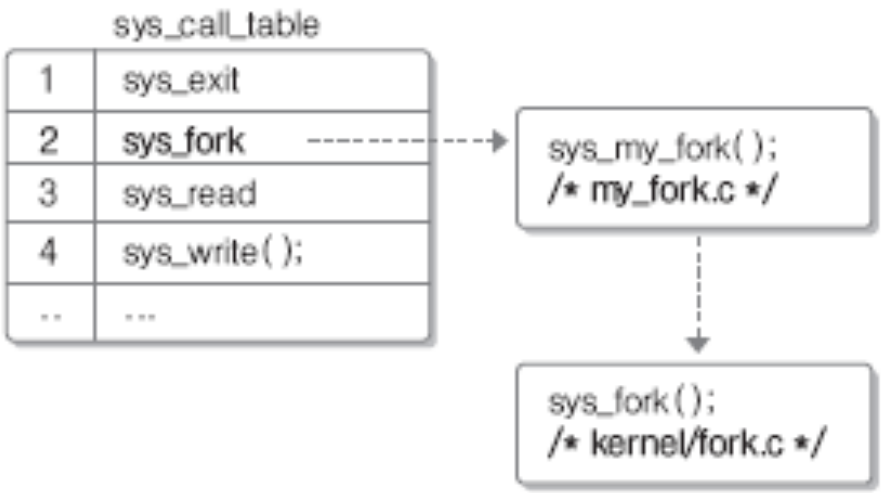
| C data type | Module parameter type | note |
|---|---|---|
| byte | b | |
| short | h | |
| ushort | h | |
| int | i | |
| uint | i | |
| long | l | |
| ulong | l | |
| bool | i | |
| char* | s | |
| char* | charp | 2.6 |
| char* | string | 2.6 |

# System Call Wrapping

- Change the systems call procedure
  - fork() -> sys_fork() : original system call procedure
  - fork() -> sys_my_fork() -> sys_fork() : modified system call procedure
- It changes sys_call_table.




Before system call wrapping




After system call wrapping

# Example-1 (Kernel 2.4)

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <sys/syscall.h>

// 시스템 호출 테이블을 참조하기 위한 선언
extern void *sys_call_table[];

// sys_open( )을 참조하기 위한 함수 포인터
asmlinkage int(*org_sys_open)(const char*, int, int );
// sys_getuid( )를 참조하기 위한 함수 포인터
asmlinkage int(*org_sys_getuid)( );
```

# Example-2 (Kernel 2.4)

```
asmlinkage int sys_our_open( char *fname, int flags, int mode )
{
  // 전처리
  printk( KERN_ALERT "%s file is opened by %d\n", fname,
                      org_sys_getuid( ) );
  // 원본 sys_open( )을 호출
  return(org_sys_open( fname, flags, mode ) );
}
```

# Example-3

```
int __init init_module( )
{
  // 시스템 호출 테이블에서 sys_open( )의 위치를 알아내서 저장
  org_sys_open = sys_call_table[  __NR_open ];
  // 시스템 호출 테이블에서 sys_open( )을 sys_our_open( )의 위치로 변경
  sys_call_table[ __NR_open]  = sys_our_open;
  // 시스템 호출 테이블에서 sys_getuid( ) 시스템 호출 위치 가져오기
  org_sys_getuid = sys_call_table[  __NR_getuid ];
  printk( KERN_ALERT "[ Module Message]  Init...\n" );
  return 0;
}
```

# Example-4

```
int __exit cleanup_module( )
{
  // sys_our_open( )을 원본 sys_open( )으로 변경
  sys_call_table[ __NR_open ] = org_sys_open;
  printk( KERN_ALERT "[Module Message] Cleanup...\n" );
}

MODULE_LICENSE( "GPL" );
```

# Test

```
coffee:~/works/chap04/wrapping# gcc -D__KERNEL__ -DMODULE -
       I/usr/src/linux-2.4.32/include -O2 -c wrapping.c -o
       wrapping.o
coffee:~/works/chap04/wrapping# insmod wrapping.o
[Module Message] Init...
/etc/localtime file is opened by 0
20060212024819.ksyms file is opened by 0
/proc/ksyms file is opened by 0
20060212024819.modules file is opened by 0
/proc/modules file is opened by 0
......
coffee:~/works/chap04/wrapping# rmmod wrapping
[Module Message] Cleanup...
```

래핑 모듈 로드

open된 파일들

래핑 모듈 제거

# Find System Calls Table (Kernel 2.6)

```
static void **find_system_call_table( void )
{
  unsigned long ptr;
  extern int  loops_per_jiffy;
  unsigned long *p;

  for ( ptr = ( unsigned long )&loops_per_jiffy;
      ptr < ( unsigned long )&boot_cpu_data; ptr += sizeof( void* ) )
  {
      p = ( unsigned long * )ptr;
      if ( p[ 6 ] == ( unsigned long )sys_close )
      {

      return ( void** )p;
      }
  }

  return NULL;
}
```

커널 심볼로 공개하지 않습니다.

loops_per_jiffy와 boot_cpu_data 사이를 검색합니다.

ptr을 포인터로 변환합니다.

ptr의 6번째 주소가 sys_close와 같은지 비교합니다.

만약 주소가 같다면 p가 sys_call_table의 주소입니다.

# 참고 - gcc 최적화 옵션

- 커널 컴파일 시 최적화 옵션 –O2만 사용하는 이유(-O0~3)
  - 커널은 인라인 함수를 많이 사용하고 있는데 –O3 최적화는 컴파일러가 판단해서 인라인이 빠른 것은 인라인으로, 함수가 빠른 것은 함수로 바꿔버린다. 커널은 최적화된 수행 속도를 위해 의도적으로 인라인 함수를 사용하고 있어서 컴파일러에 의해서 자의적으로 함수로 바뀌는 것을 막기 위해 –O2 옵션을 사용한다.

| 최적화옵션 | 의미 |
|---|---|
| -O0(기본값) | 최적화를 수행하지 않는다 |
| -O<br>-O1 | 코드 크기와 실행 시간을 줄이는 것을 제외한 최적화는 실행하지 않는다 |
| -O2 | 메모리 공간과 속도를 희생하지 않는 범위내의 모든 최적화를 수행한다<br>loop unrolling과 function inlining에 대한 최적화를 수행하지 않는다 |
| -O3 | O2 최적화에 인라인 함수와 레지스터에 대한 최적화를 추가로 수행한다 |
| -Os | -O2 최적화와 함께 코드 크기에 대한 최적화를 수행한다 |