

# 운영체제론 실습 5주차

정보보호연구실 @ 한양대학교

# 운영체제론 실습 5주차

1. Module Programming 설명
2. 생일 목록 불러오는 모듈 프로그래밍

# 모듈(Module) 프로그래밍

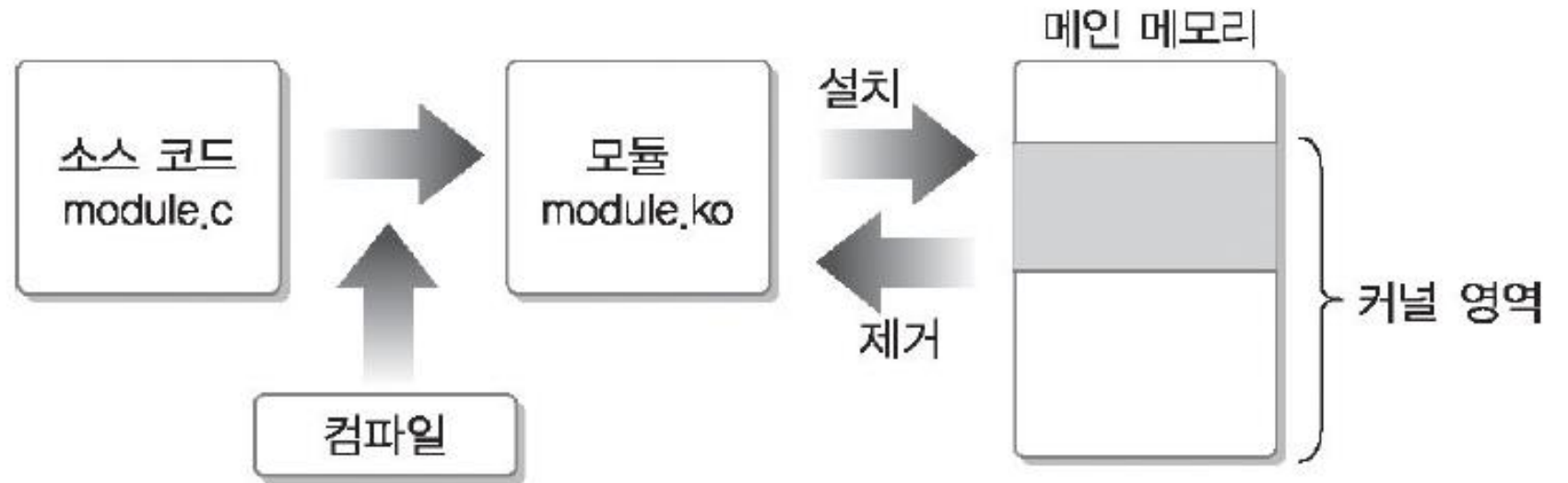
## 1) 모듈 프로그래밍

- 커널을 변경할 시 커널 전체를 다시 컴파일 해야함 ex) system call 등록
- 모듈 프로그램으로 개발하면, **해당 모듈만 컴파일** 후 **필요할 때만 동적으로 연결**하여 커널의 일부로 사용 가능
- 자주 사용하지 않는 커널 기능을 메모리에 상주시키지 않아도 됨
- **확장성과 재사용성**을 높일 수 있음
- **사건 구동형 (event-driven program)** 방식으로 작성
- 내부에 main() 함수가 존재하지 않음
- 커널에 적재/제거하기 위한 규칙과 유틸리티가 필요
- 커널에 적재된 모듈 프로그램은 시스템 내부에서 **모든 특권**을 가지게 되므로, 신중하게 작성해야 함

# 모듈(Module) 프로그래밍

## 2) 모듈 프로그래밍

- 모듈 프로그램 작성
- 모듈 프로그램 컴파일
- 모듈 설치
- 설치된 모듈 확인
- 모듈 제거



# 모듈(Module) 프로그래밍

## 3) 모듈 프로그래밍 코드 작성

```
GNU nano 2.9.3 mymodule.c

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

int simple_init(void)
{
    printk(KERN_INFO "Loading My Module.....\n");
    return 0;
}

void simple_exit(void)
{
    printk(KERN_INFO "Removing My Module.....\n");
}

module_init(simple_init);
module_exit(simple_exit);
```

초기화 루틴

종료 루틴

초기화, 종료 루틴 설정

# 모듈(Module) 프로그래밍

## 4) 모듈 Makefile 코드 작성

```
GNU nano 2.9.3          Makefile

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

obj-m := mymodule.o

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

clean:
    rm -rf *~
```

# 모듈(Module) 프로그래밍

## 5) 모듈 프로그램 컴파일

- \$ sudo make

```
dongmin1@dongmin1-VirtualBox:~/week4$ sudo make
make -C /lib/modules/5.5.13/build KBUILD_EXTMOD=/home/dongmin1/week4 modules
make[1]: Entering directory '/usr/src/linux-5.5.13'
  CC [M]  /home/dongmin1/week4/mymodule.o
  Building modules, stage 2.
  MODPOST 1 modules
WARNING: modpost: missing MODULE_LICENSE() in /home/dongmin1/week4/mymodule.o
see include/linux/module.h for more information
  CC [M]  /home/dongmin1/week4/mymodule.mod.o
  LD [M]  /home/dongmin1/week4/mymodule.ko
make[1]: Leaving directory '/usr/src/linux-5.5.13'
```

```
dongmin1@dongmin1-VirtualBox:~/week4$ ls
Makefile      Module.symvers  mymodule.ko     mymodule.mod.c  mymodule.o
modules.order mymodule.c      mymodule.mod    mymodule.mod.o
```

# 모듈(Module) 프로그래밍

## 6) 모듈 설치

- `$ sudo insmod {modulename}.ko`
- `$ dmesg`

```
[ 89.714377] 04:41:36.586061 main OS Product: Linux
[ 89.714472] 04:41:36.586164 main OS Release: 5.5.13
[ 89.714573] 04:41:36.586258 main OS Version: #3 SMP Tue Apr 7 09:50:56 KS
T 2020
[ 89.714721] 04:41:36.586359 main Executable: /opt/VBoxGuestAdditions-6.1.
4/sbin/VBoxService
04:41:36.586363 main Process ID: 959
04:41:36.586366 main Package type: LINUX_64BITS_GENERIC
[ 89.718947] 04:41:36.590600 main 6.1.4 r136177 started. Verbose level = 0
[ 89.723660] 04:41:36.595298 main vbglR3GuestCtrlDetectPeekGetCancelSuppor
t: Supported (#1)
[ 120.096227] systemd-journald[275]: File /var/log/journal/fbf0c302b00f4680905c
0220a8f589b6/user-1000.journal corrupted or uncleanly shut down, renaming and re
placing.
[ 122.298655] VBGL_IOCTL_ACQUIRE_GUEST_CAPABILITIES failed rc=-138
[ 128.366872] rfkill: input handler disabled
[ 129.549405] VBGL_IOCTL_ACQUIRE_GUEST_CAPABILITIES failed rc=-138
[ 136.538498] ISO 9660 Extensions: Microsoft Joliet Level 3
[ 136.564392] ISO 9660 Extensions: RRIP_1991A
[ 155.205207] SYSCALL HELLO IS CALLED!
[19468.727579] mymodule: module license 'unspecified' taints kernel.
[19468.727581] Disabling lock debugging due to kernel taint
[19468.727900] Loading My Module.....
dongmin1@dongmin1-VirtualBox:~/week4$
```



# 모듈(Module) 프로그래밍

## 7) 설치된 모듈 확인

- \$ lsmod

```
dongmin1@dongmin1-VirtualBox:~/week4$ sudo insmod mymodule.ko
dongmin1@dongmin1-VirtualBox:~/week4$ lsmod
```

Module	Size	Used by
mymodule	16384	0
nls_utf8	16384	1
isofs	49152	1
vboxvideo	36864	0
intel_rapl_msr	20480	0
intel_rapl_common	24576	1 intel_rapl_msr
intel_powerclamp	20480	0
crct10dif_pclmul	16384	1
crc32_pclmul	16384	0
vmwgfx	303104	4

# 모듈(Module) 프로그래밍

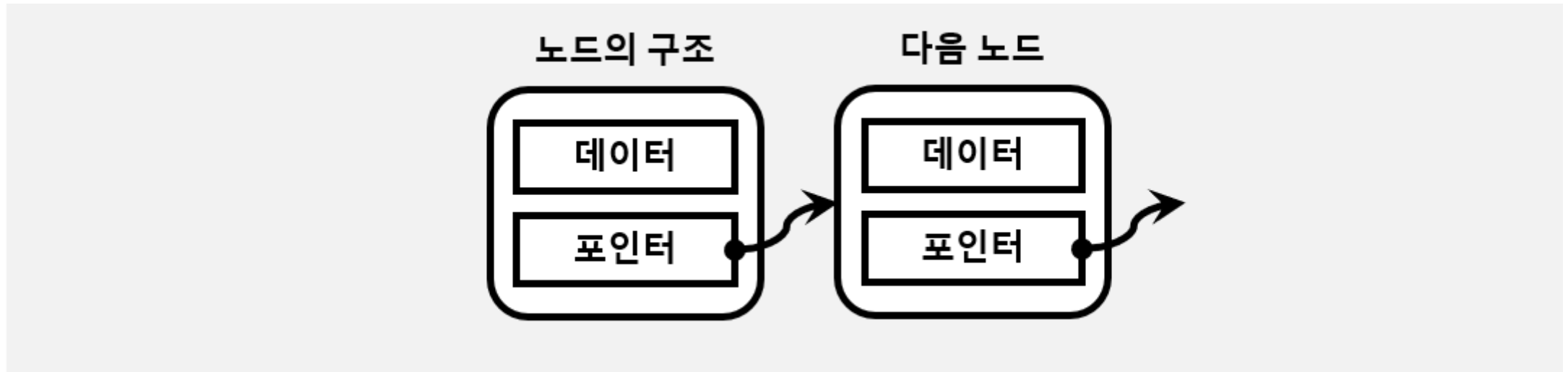
## 8) 모듈 제거

- `$ sudo rmmod {modulename}`
- `$ dmesg`

```
dongmin1@dongmin1-VirtualBox: ~/week4
File Edit View Search Terminal Help
[ 89.714472] 04:41:36.586164 main OS Release: 5.5.13
[ 89.714573] 04:41:36.586258 main OS Version: #3 SMP Tue Apr 7 09:50:56 KS
T 2020
[ 89.714721] 04:41:36.586359 main Executable: /opt/VBoxGuestAdditions-6.1.
4/sbin/VBoxService
04:41:36.586363 main Process ID: 959
04:41:36.586366 main Package type: LINUX_64BITS_GENERIC
[ 89.718947] 04:41:36.590600 main 6.1.4 r136177 started. Verbose level = 0
[ 89.723660] 04:41:36.595298 main vbglR3GuestCtrlDetectPeekGetCancelSupport: Supported (#1)
[ 120.096227] systemd-journald[275]: File /var/log/journal/fbf0c302b00f4680905c
0220a8f589b6/user-1000.journal corrupted or uncleanly shut down, renaming and re
placing.
[ 122.298655] VBGL_IOCTL_ACQUIRE_GUEST_CAPABILITIES failed rc=-138
[ 128.366872] rfkill: input handler disabled
[ 129.549405] VBGL_IOCTL_ACQUIRE_GUEST_CAPABILITIES failed rc=-138
[ 136.538498] ISO 9660 Extensions: Microsoft Joliet Level 3
[ 136.564392] ISO 9660 Extensions: RRIP_1991A
[ 155.205207] SYSCALL HELLO IS CALLED!
[19468.727579] mymodule: module license 'unspecified' taints kernel.
[19468.727581] Disabling lock debugging due to kernel taint
[19468.727900] Loading My Module.....
[19865.134974] Removing My Module.....
dongmin1@dongmin1-VirtualBox:~/week4$
```

# 연결 리스트(Linked List)

- 연결 리스트(Linked List): 각 데이터들을 포인터로 연결하여 관리하는 구조임
- 노드 : 데이터를 저장하는 데이터 영역과 다음 노드를 가리키는 포인터 영역으로 구성됨



# 연결 리스트(Linked List)

- Linked List를 사용해서 얻는 **이점**

- 동적 자료구조
- 쉬운 생성과 삭제
- 노드의 생성과 삭제가 자유롭기 때문에 **메모리 낭비가 적음**
- Linked List를 통해 **다른 자료구조들을 쉽게 구현 가능**

- Linked List의 **단점**

- 데이터 하나를 표현하기 위해 '**포인터**'라는 추가 메모리 사용
- 데이터 탐색하는 시간 복잡도가 매우 높음  $O(n)$

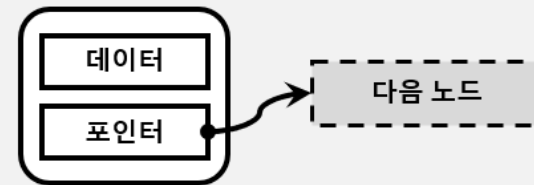
# 단순 연결 리스트(Singly Linked List)

- 단순 연결 리스트는 다음 노드만을 가리키는 단방향 연결 구조

s\_list.c

```
struct Node{  
    int data;  
    struct Node *next;  
};
```

노드의 구조



- 함수 예) 노드의 생성

s\_list.c

```
node createNode(){  
    node new_node;  
    new_node = (Node)malloc(sizeof(struct Node));  
    new_node->next = NULL;  
    return new_node;  
}
```

# 이중 연결 리스트(Doubly Linked List)

- 이중 연결 리스트는 이전과 다음 노드를 가리키는 양방향 연결 구조

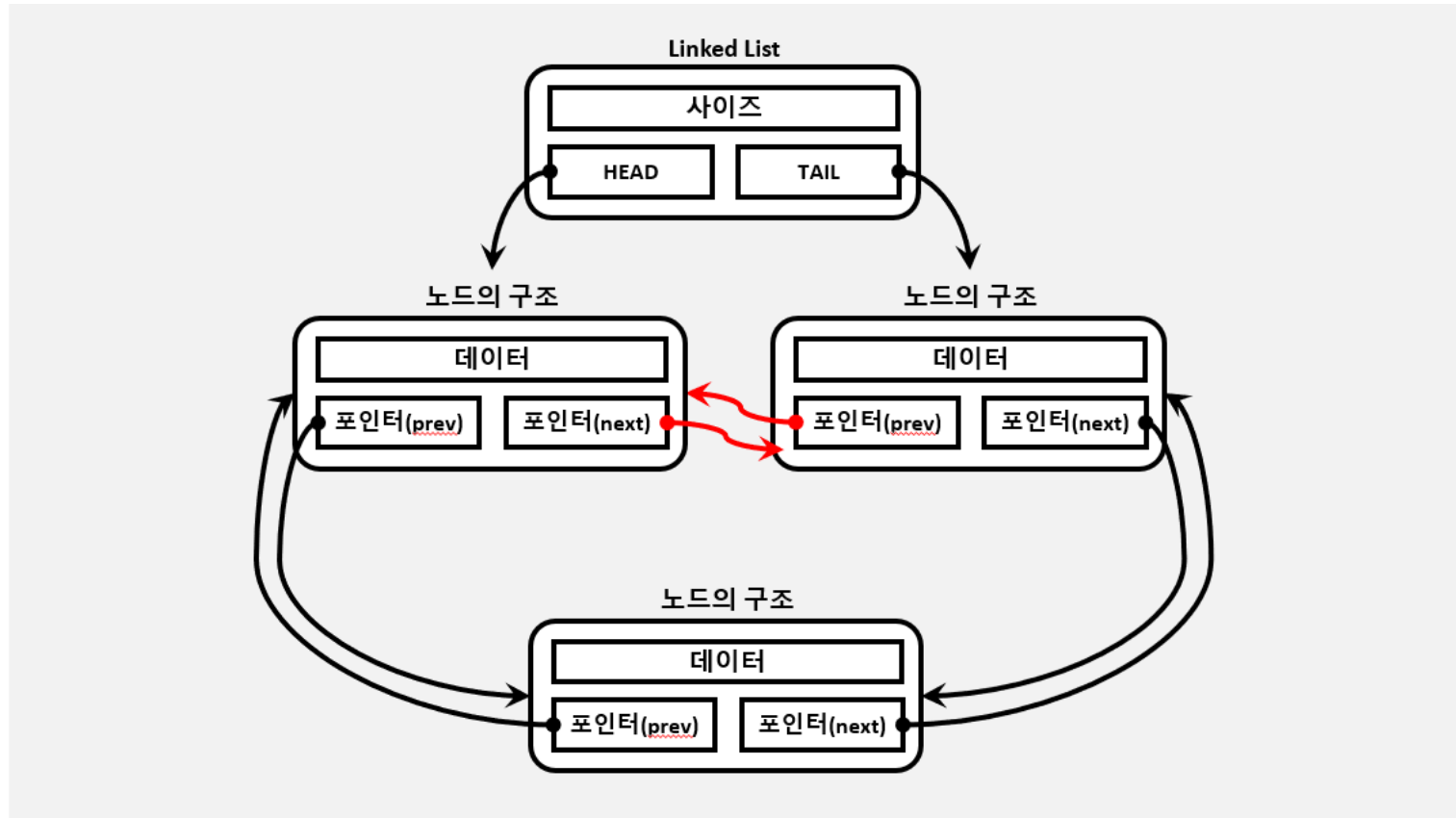


**d\_list.c**

```
struct Node{  
    int data;  
    struct Node *prev, *next;  
};
```

# 이중 원형 연결 리스트(Doubly Circular Linked List)

- 이중 원형 연결 리스트는 처음 노드와 마지막 노드가 연결되어 원형을 이루는 구조



# 커널에는 어떻게 구현되어 있을까?

- 커널에는 우리가 알고 있는 Linked List가 어떤 모습을 하고 있을까?
  - vi/nano /usr/src/linux-\$(uname -r)/include/linux/types.h

**types.h**

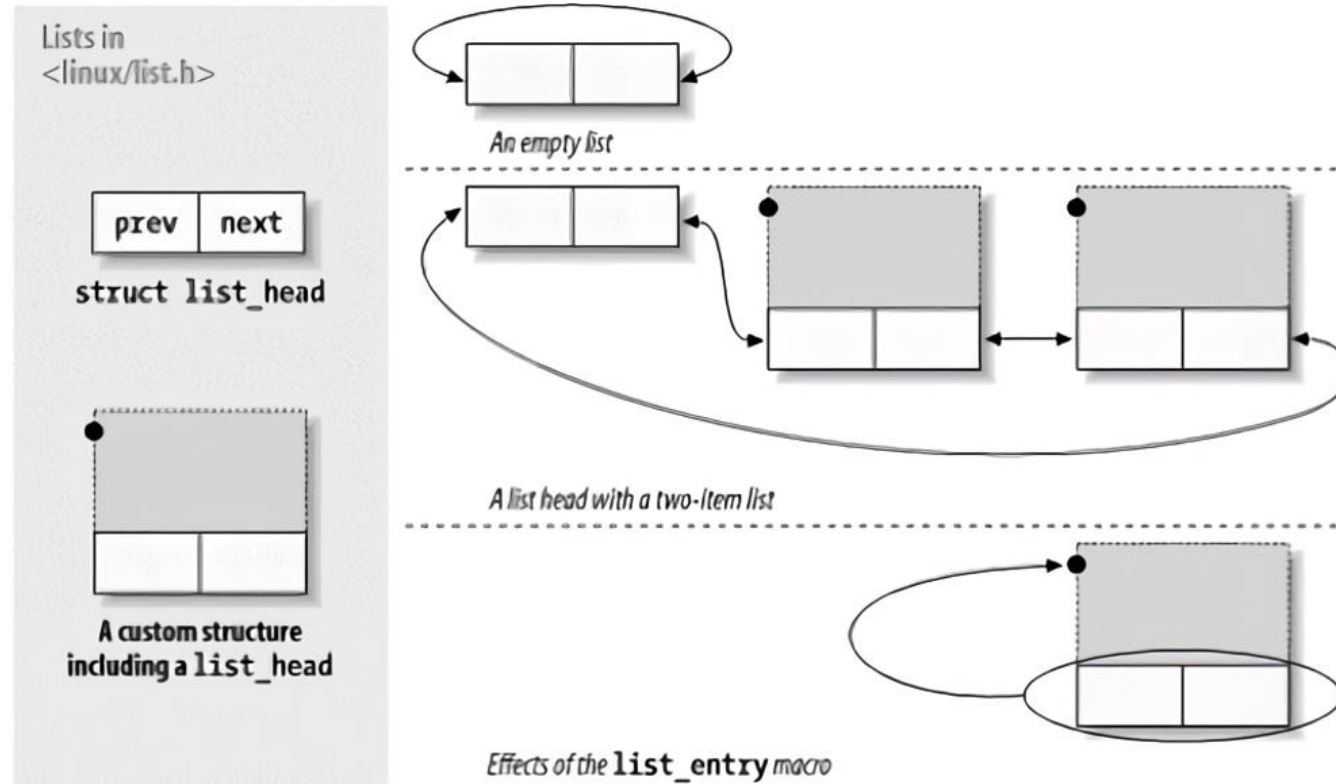
```
struct list_head{  
    struct list_head *prev, *next;  
};
```

- 이전 노드와 다음 노드를 가리키는 이중 연결 리스트임을 알 수 있음



# Linux 커널의 Linked List

- 리스트 노드(list\_head 구조체)를 사용자가 만든 데이터 안에 넣는 방식



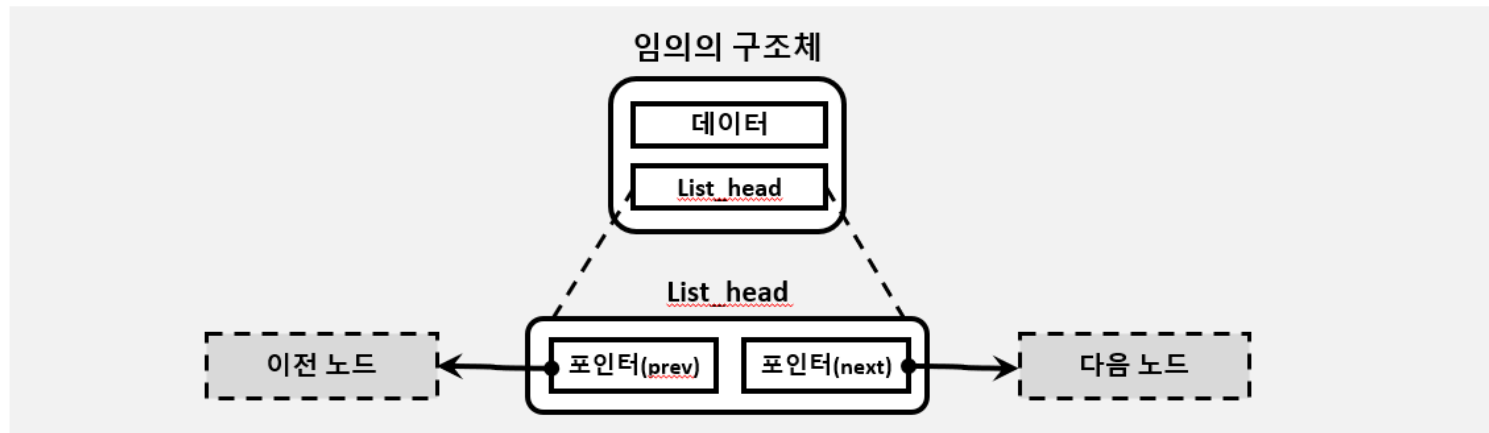
# 데이터 영역 구현

- **Linux는 Doubly Circular Linked List로 구현되어 있음**

1. 임의의 구조체(struct my\_struct) 선언
  - Struct list\_head를 멤버로 넣음
2. Head 선언
3. List.h 에서 제공하는 연산을 사용

# List\_head 인터페이스 사용 방법

- 데이터 영역을 가지는 임의의 구조체를 만들고 list\_head를 가리키게 함

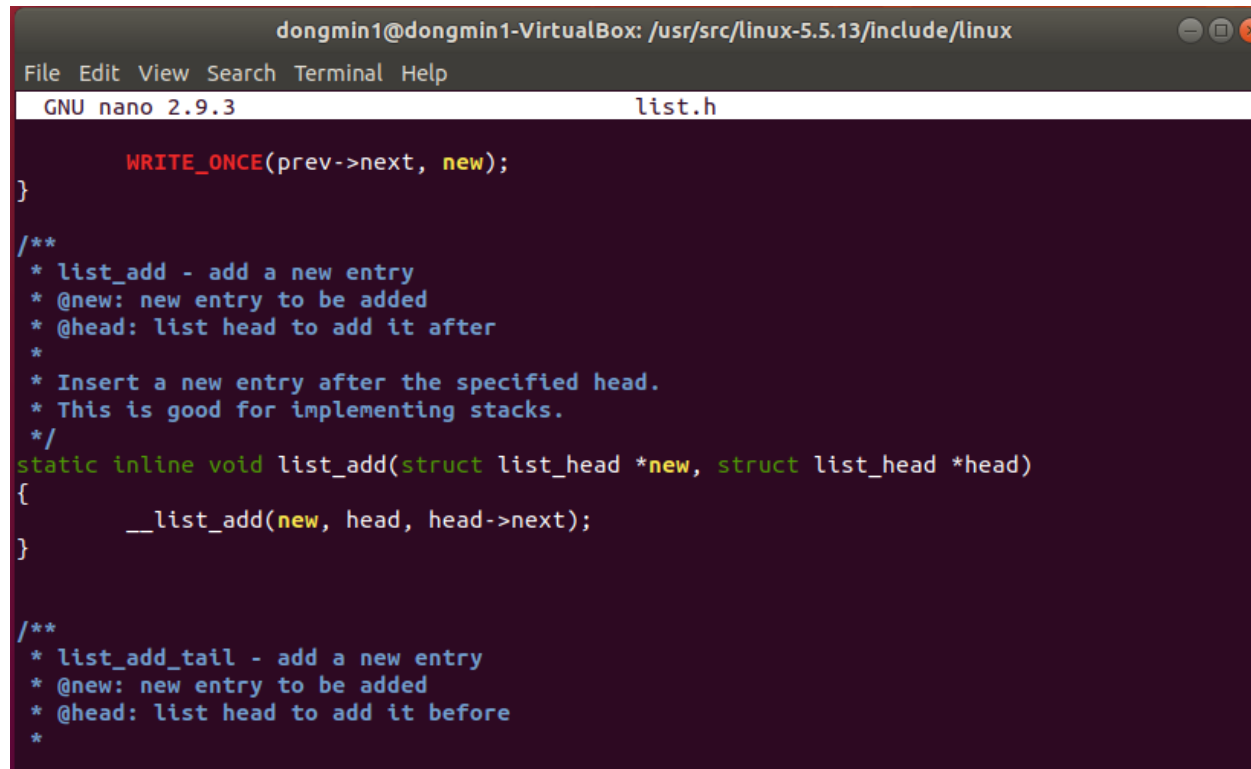


임의의 구조체의 예

```
struct my_struct{  
    void data; // 저장하고 싶은 데이터  
    struct list_head list;  
};
```

# List.h

- 연결리스트의 구조체, 함수 등이 구현되어 있는 헤더파일을 살펴보자
  - \$ vi /usr/src/linux-\$(uname -r)/include/linux/list.h



```
dongmin1@dongmin1-VirtualBox: /usr/src/linux-5.5.13/include/linux
File Edit View Search Terminal Help
GNU nano 2.9.3 list.h

    WRITE_ONCE(prev->next, new);
}

/**
 * list_add - add a new entry
 * @new: new entry to be added
 * @head: list head to add it after
 *
 * Insert a new entry after the specified head.
 * This is good for implementing stacks.
 */
static inline void list_add(struct list_head *new, struct list_head *head)
{
    __list_add(new, head, head->next);
}

/**
 * list_add_tail - add a new entry
 * @new: new entry to be added
 * @head: list head to add it before
 *
 */
```

# List.h

## ▪ 기본적인 함수

함수명	목 적
LIST_HEAD(ptr)	Ptr란 이름의 list_head를 정의 후 리스트 자료구조를 초기화
list_add( <b>struct</b> list_head *new, <b>struct</b> list_head *head);	이전에 만든 리스트에 새로운 entry(list_head)를 맨 앞에 추가
list_add_tail( <b>struct</b> list_head *new, <b>struct</b> list_head *head);	list_add와 동일하나 맨 뒤에 추가
list_del( <b>struct</b> list_head *entry);	원하는 entry(list_head)를 삭제
list_empty( <b>struct</b> list_head *head);	비어 있는지 체크 (비면 참)
list_for_each_entry(pos, head, member)	리스트 노드들을 한바퀴 순환하면서, 각 노드들을 참조하는 포인터를 시작주소 지점(entry)으로 옮기는 것
list_for_each_safe(pos, n, head)	entry 의 복사본을 사용함으로써 수행 시 해당 자료가 삭제되더라도 오류가 나지 않게 하는 것

# 데이터 생성

- Kmalloc을 통해 메모리 공간 할당

- Kmalloc은 커널 내부에 페이지 크기보다 작은 크기의 메모리 공간을 할당할 때 사용
- GFP\_KERNEL : 보통 커널 RAM 메모리를 할당함

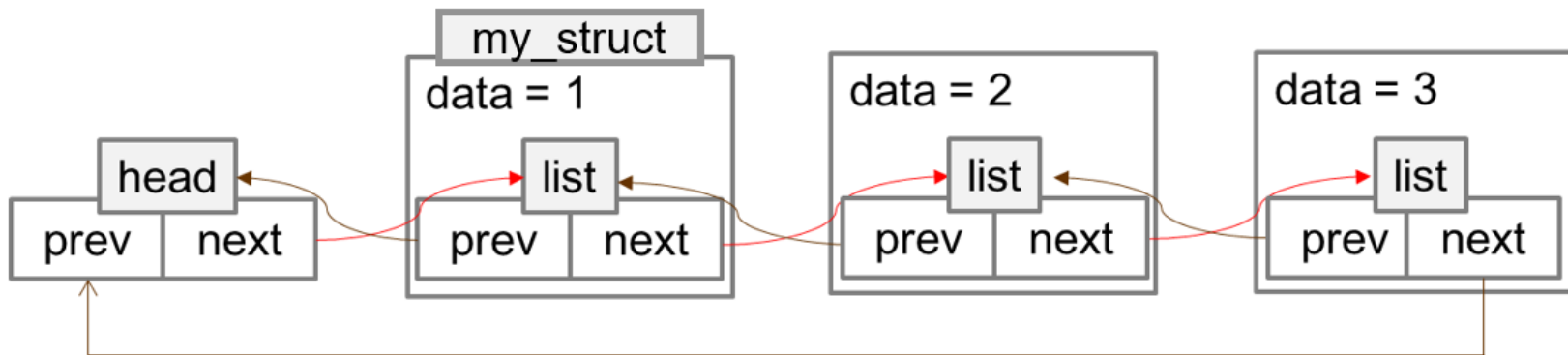
- 사용방법

```
struct my_struct *struct1;  
struct1 = kmalloc(sizeof(struct my_struct), GFP_KERNEL);
```

# 데이터 삽입

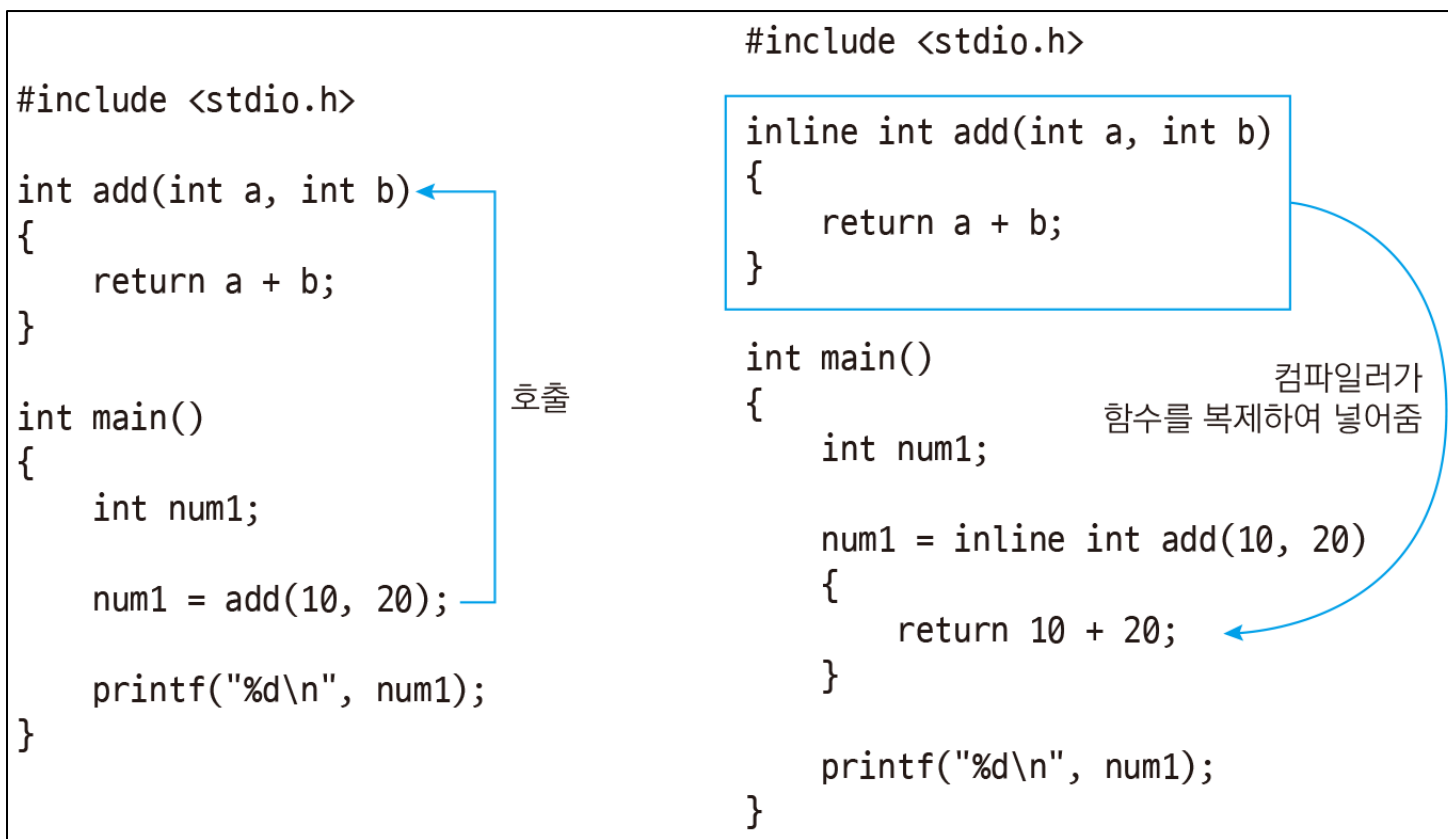
## ▪ List\_add\_tail

```
/**
 * list_add_tail - add a new entry
 * @new: new entry to be added
 * @head: list head to add it before
 *
 * Insert a new entry before the specified head.
 * This is useful for implementing queues.
 */
static inline void list_add_tail(struct list_head *new, struct list_head *head)
{
    __list_add(new, head->prev, head);
}
```



# Inline 함수

- 실행 과정이 일반 함수와 크게 다르지 않음
- 컴파일러는 함수를 사용하는 부분에 함수의 코드를 복제해서 넣음





# 데이터 출력

- `list_for_each_entry`라는 매크로 함수를 사용(반복적으로 탐색하며 주어진 타입을 확인)

```
/**
 * list_for_each_entry - iterate over list of given type
 * @pos:      the type * to use as a loop cursor.
 * @head:     the head for your list.
 * @member:   the name of the list_head within the struct.
 */
#define list_for_each_entry(pos, head, member) \
    for (pos = list_first_entry(head, typeof(*pos), member); \
         &pos->member != (head); \
         pos = list_next_entry(pos, member))
```

# 데이터 삭제

- List\_for\_each\_safe 매크로 함수 사용 (반복적으로 탐색하며 노드마다 함수 수행)

```
/**
 * list_for_each_safe - iterate over a list safe against removal of list entry
 * @pos:      the &struct list_head to use as a loop cursor.
 * @n:      another &struct list_head to use as temporary storage
 * @head:    the head for your list.
 */
#define list_for_each_safe(pos, n, head) \
    for (pos = (head)->next, n = pos->next; pos != (head); \
         pos = n, n = pos->next)
```

- 본 매크로 함수에 추가할 내용
  - Printk(출력할 구조체의 내용)
  - List\_del(삭제할 구조체의 list\_head의 주소값)
  - Kfree(삭제할 구조체 메모리의 포인터)

# 매크로 함수

- 매크로 함수 예제

```
#define ADD(a , b) a + b
```

- 코드 내부에 매크로 함수를 사용했을 경우

```
...  
int result = ADD(2,3);  
...
```

- 연산을 수행하기 직전 컴파일러에 의해 코드가 그대로 치환됨

```
...  
int result = 2 + 3;  
...
```

# 실습: 생일 목록을 불러오는 모듈 프로그래밍

## TODO:

- 생일 데이터를 가지는 구조체를 만든다.
- 생일 데이터들끼리 커널의 연결리스트를 통해 연결한다.
- 연결된 데이터들을 전부 출력한다.

# 실습: 스키텐 코드(bdlist.c)

```
struct birthday {
    int day;
    int month;
    int year;
    struct list_head list;
};

static LIST_HEAD(birthday_list);

struct birthday *createBirthday(int day, int month, int year) {
    /* TODO: 생일을 위한 메모리를 할당하고, 인자들을 채워 생일을 완성하세요. */
}

int simple_init(void) {
    printk("INSTALL MODULE: bdlist\n");
    /* TODO: 생일 목록을 하나씩 생성하는대로 연결리스트에 연결시키세요 (노드 삽입). */
    /* TODO: 완성된 연결리스트를 탐색하는 커널 함수를 사용하여 출력하세요. */

    return 0;
}

void simple_exit(void) {
    /* 모듈을 제거할 때는 생성한 연결 리스트도 하나씩 제거하며 끝내도록 하세요. */
    /* 제거를 하기 전에 리스트가 "비어있을 경우"에 대한 예외처리를 하는게 좋겠죠? */
    if(list_empty(&birthday_list)) {
        printk("List is Empty\n");
        return;
    }

    /* TODO: 이제 본격적으로 연결리스트를 탐색하면서 하나씩 제거하도록 하시면 됩니다. */
    printk("REMOVE MODULE: bdlist\n");
}

module_init(simple_init);
module_exit(simple_exit);

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("make a list of birthdays and print");
MODULE_AUTHOR("이름_학번");
```

## Makefile

```
obj-m+=bdlist.o

all:
    make -C /lib/modules/$(shell uname -r)/build/ KBUILD_EXTMOD=$(PWD) modules
clean:
    make -C /lib/modules/$(shell uname -r)/build/ KBUILD_EXTMOD=$(PWD) clean
```

만약 컴파일 실패 시 KBUILD\_EXTMOD를 **M**으로 변경

# 실습 결과화면

- 모듈 생성 후 제거
- \$ dmesg

```
[ 1158.633714] INSTALL MODULE: bdlist
[ 1158.633716] INSTALL MODULE: bdlist
[ 1158.633717] OS Module : DAY 23, 2, 1995
[ 1158.633719] OS Module : DAY 19, 4, 1967
[ 1158.633720] OS Module : DAY 7, 2, 1964
[ 1213.797934] OS Module : Removing 23, 2, 1995
[ 1213.797936] OS Module : Removing 19, 4, 1967
[ 1213.797937] OS Module : Removing 7, 2, 1964
[ 1213.797937] REMOVE MODULE: bdlist
```