| 과목명 | 운영체제 | 분반 | X | 담당교수 | 김화성 교수님 |
|---|---|---|---|---|---|
| 학과 | 전자통신공학과 | 학번 | 2016707079 | 이름 | 하상천 |
| 과제명: H/W 4 - 리눅스 커널 모듈 프로그래밍 실습 및 과제 2 | | | | | |

1. 과제설명 ( 사용자 요구사항 기술: 과제에 대한 설명 및 목표 )

# 운영체제 실습 및 과제 - 2

## (리눅스 커널 모듈 프로그래밍 실습 및 과제)

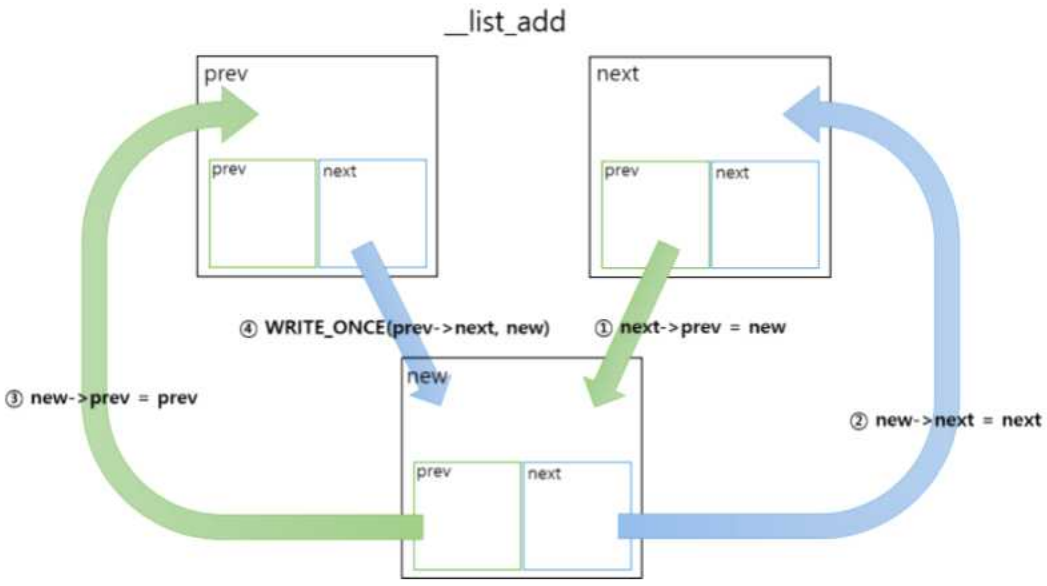
리눅스 커널 모듈 프로그래밍 실습 및 과제입니다. 실습과 과제를 수행하세요.
실습의 경우 수행 결과만 출력하여 보고서에 첨부
과제의 경우 관련이론, 코드, 코드에 대한 설명, 수행 결과를 첨부하세요.
고찰은 실습과 과제에 모두에 대해 작성하세요.

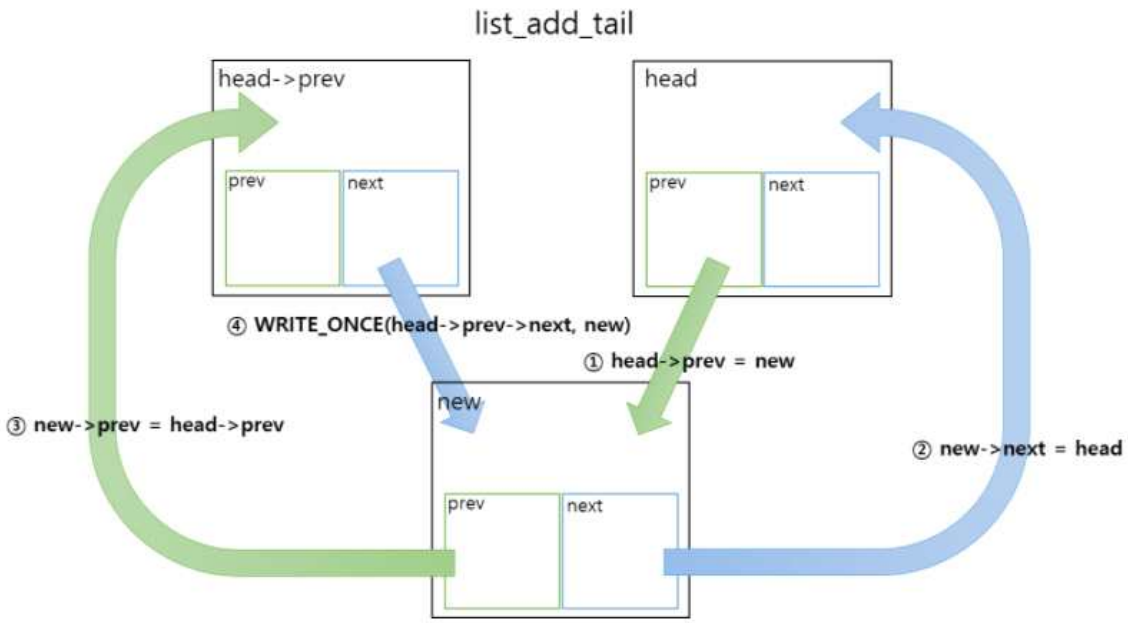
### 실습 2. 커널 모듈 linked list 삽입 실습

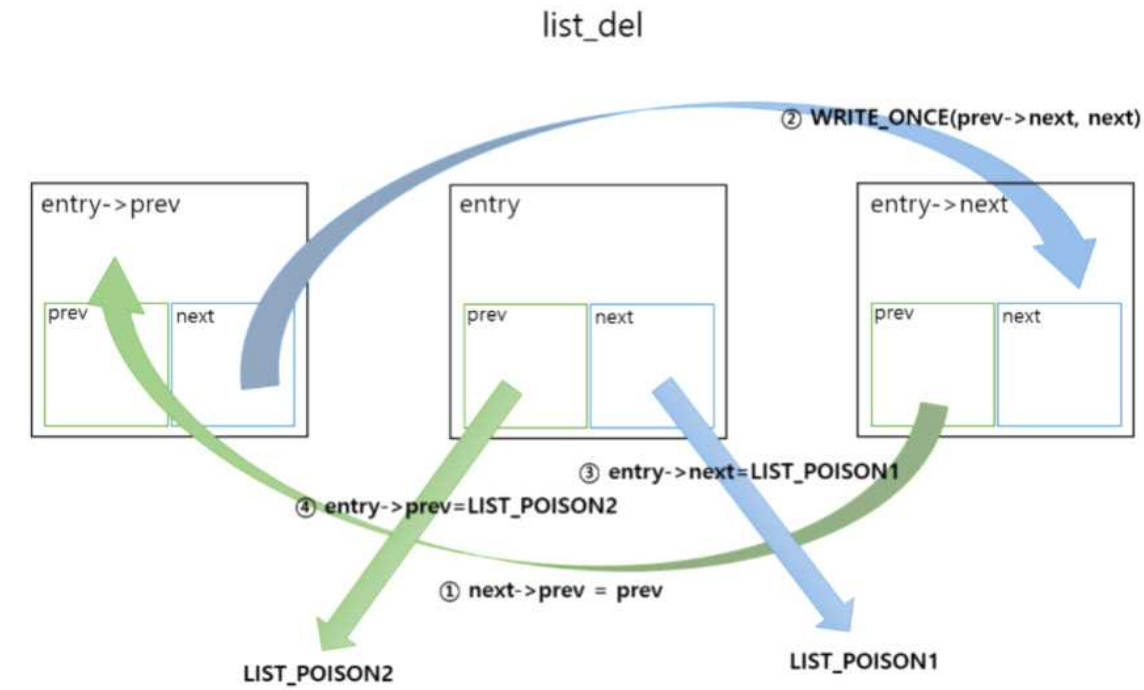
### 과제 2. 커널 모듈 프로그래밍에서의 linked list 사용

- 실습에서 진행한 list 삽입 외에 list.h 파일에서 리스트 삭제, 순환 매크로를 사용하여 각 기능별로 동작 결과를 출력하시오.

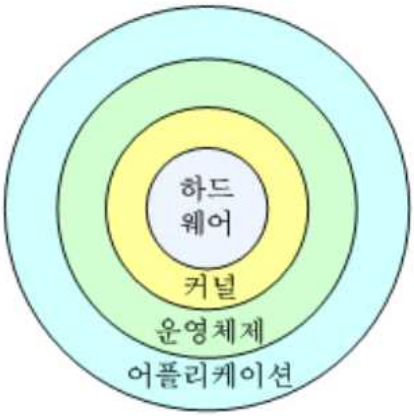
2. 사용자 요구사항을 정형적 방법으로 기술 (UML, Pseudo code, 그림등을 이용하여 기술)

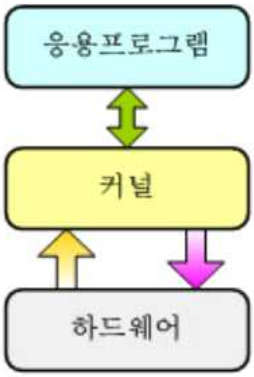


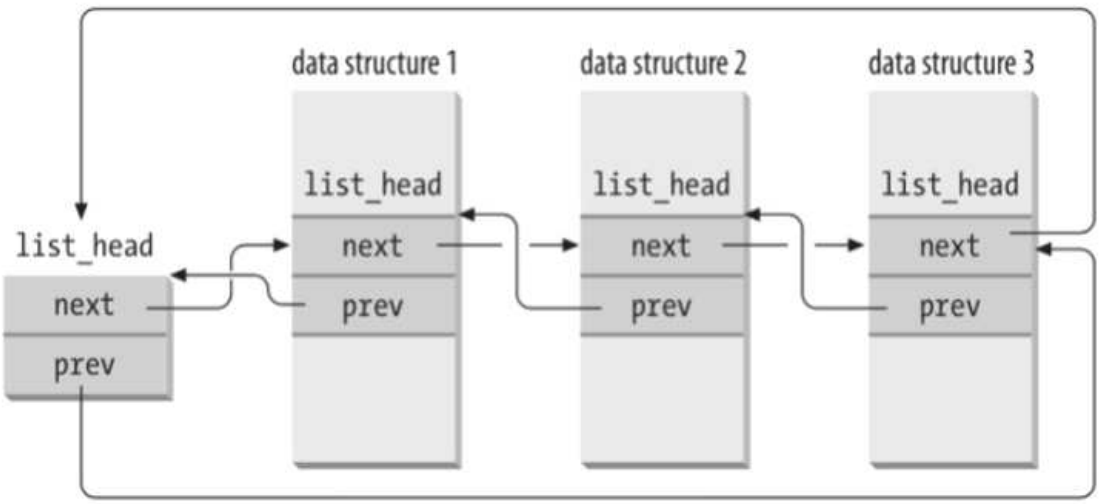

list_add_tail
head->prev
prev
next
head
prev
next
④ WRITE_ONCE(head->prev->next, new)
① head->prev = new
new
prev
next
③ new->prev = head->prev
② new->next = head


list_del
② WRITE_ONCE(prev->next, next)
entry->prev
prev
next
entry
prev
next
entry->next
prev
next
③ entry->next=LIST_POISON1
④ entry->prev=LIST_POISON2
① next->prev = prev
LIST_POISON2
LIST_POISON1

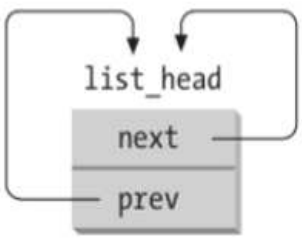
3. 알고리즘 및 자료구조 설계 내용




(a) a doubly linked listed with three elements




(b) an empty doubly linked list

4. 소스코드 설명 ( 직접 작성한 소스코드중에 핵심 부분을 발췌하여 설명 )

```
LIST_HEAD(Node_head);

static int list_insert(int data){
struct Node *new;
new = (struct Node *)kmalloc(sizeof(struct Node), GFP_KERNEL);
new->data = data
list_add_tail(&new->list, &Node_head);
return 0;
}
```

LIST_HEAD 매크로는 list_head 구조체를 선언하고, next와 prev를 자기 자신으로 초기화한다.
그리고 kmalloc() 함수를 통해 커널에서 동적 메모리를 할당 받는다. 파라미터로는 malloc()함수와 동일하게 바이트 단위로 메모리 크기를 지정한다. 두 번째 파라미터는 메모리 할당 세부 옵션 플래그인데 보통 커널 드라이버에선 GFP_KERNEL을 많이 사용한다. list_add_tail() 함수는 list의 마지막에 새로운 node를 추가한다. 만약 list_add() 함수를 사용했다면 list의 head 뒤에 새로운 node를 추가한다.

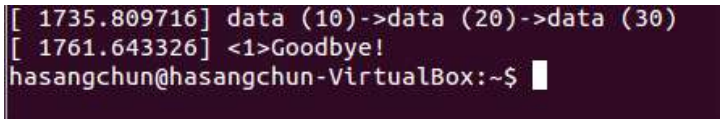
```
static void list_all_delete(void){
struct Node *p, *q;
list_for_each_entry_safe(p, q, &Node_head, list){
list_del(&p->list);
kfree(p);
}
}
```

list_all_delete() 함수는 리스트를 순환하면서 모든 리스트 멤버를 삭제하는 함수이다.
순환 매크로로 list_for_each_entry_safe() 함수를 사용한 이유는 list_for_each_entry() 함수는 리스트를 순환하다가 리스트를 지우면 리스트 순환을 멈추기 때문이다. 하지만 list_for_each_entry_safe() 함수는 리스트 멤버를 지워도 계속 순환한다.

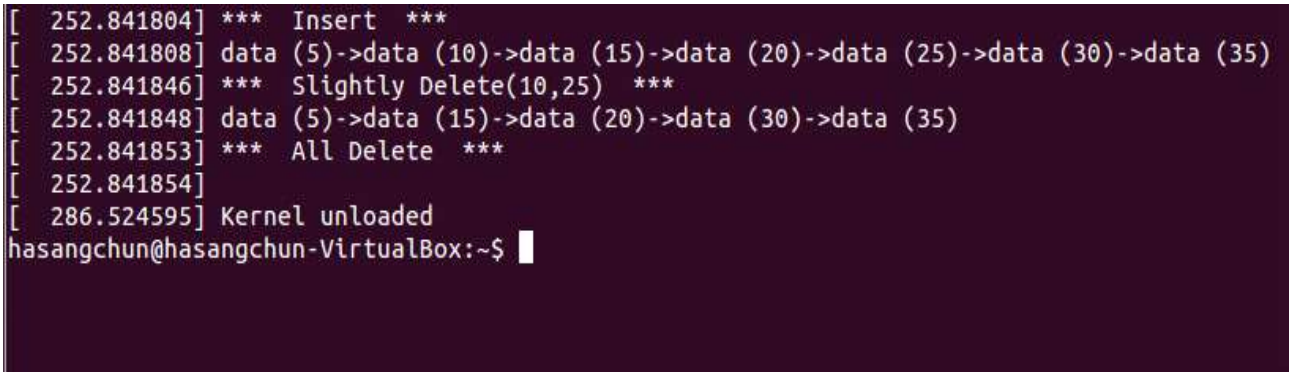
```
static void list_little_delete(void){
struct Node *p, *q;
list_for_each_entry_safe(p, q, &Node_head, list){
if ((p->data == 10) || (p->data == 25)) {
list_del(&p->list);
kfree(p);
}
}
}
```

list_little_delete() 함수는 리스트를 순환하면서 해당되는 리스트 멤버를 삭제하는 함수이다. data가 10이거나 25이면 그 리스트 멤버를 삭제하는 함수이다.

5. 실행결과 및 설명 ( 실행 결과를 캡쳐하여 첨부한 후 설명 )




실습 실행결과




과제 실행결과

(그림을 문서에 포함, 글자처럼 취급 옵션, 잉크 절약과 잘 보이게 하기위해 그림 반전)

6. 고찰 ( 과제를 진행하면서 배운점 이나, 시행 착오 내용, 기타 느낀점 )

이번 실습과 과제도 리눅스 커널 모듈 프로그래밍에 관한 것이었다. 실습으로는 커널 모듈 linked list를 삽입하는 것이었다. 저번 과제에서 커널 모듈은 모듈이 커널에 로딩될 때 호출되는 module_init() 함수와 커널에서 언로딩 될 때 호출되는 module_exit() 함수를 지녀야한다는 것을 배웠다. 그것을 기반으로 이번 실습과 과제를 진행하였다. linux/list 헤더파일에는 기존 자료구조에서 사용하는 list의 기능이 매크로로 정의되어있었다. 삽입, 삭제 하는 것뿐만 아니라 초기화하는 것, 이동시키는 것, 비어있는지 마지막인지 검사하는 것도 있고 for문처럼 각각의 node를 순서대로 접근 하는 것도 있었다. 삽입을 할 수 있는 방법은 list_add_tail()함수와 list_add()함수가 있다. 어느 쪽으로 삽입할 것인가에 대한 차이가 있지만 이번 실습과 과제에서는 list_add_tail() 함수를 사용했다.

그리고 순환 매크로로 list_for_each_entry_safe() 함수를 사용한 이유는 list_for_each_entry() 함수는 리스트를 순환하다가 리스트를 지우면 리스트 순환을 멈추기 때문이다.

실습에서는 모듈을 load할 때 10, 20, 30을 삽입하고 dmesg 명령어를 통해 결과를 확인하였다.

과제에서는 모듈을 load할 때 5, 10, 15, 20, 25, 30, 35를 삽입하고 조건문으로 10, 25를 삭제하고 그 다음 전체 삭제를 해준다. 그리고 dmesg 명령어를 통해 결과를 확인하였다.

처음에 linux/list 헤더파일을 잘 몰라서 리눅스 커널을 최초로 개발한 사람인 리누스 토르발스 깃허브를 보면서 공부했다.

참고자료 : https://github.com/torvalds/linux/blob/master/include/linux/list.h

7. 전체 소스코드 ( 글자크기 9에 줄간격을 120%로 유지하고 한 줄이 너무 길지 않게 작성 )

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

```
struct Node{
int data;
struct list_head list;
};

LIST_HEAD(Node_head);

static int list_insert(int data){
struct Node *new;
new = (struct Node *)kmalloc(sizeof(struct Node), GFP_KERNEL);
new->data = data
list_add_tail(&new->list, &Node_head);
return 0;
}

static void list_all_delete(void){
struct Node *p, *q;
list_for_each_entry_safe(p, q, &Node_head, list){
list_del(&p->list);
kfree(p);
}
}

static void list_little_delete(void){
struct Node *p, *q;
list_for_each_entry_safe(p, q, &Node_head, list){
if ((p->data == 10) || (p->data == 25)) {
list_del(&p->list);
kfree(p);
}
}
}

static void list_print(void){
struct Node *p;
list_for_each_entry(p, &Node_head, list){
printk("data (%d)", p->data);
if ((p->list).next != &Node_head)
printk("->");
}
printk("₩n");
}

static int list_init(void) {
LIST_HEAD(Node_head);
```

```
printk("*** Insert ***₩n");
list_insert(5);
list_insert(10);
list_insert(15);
list_insert(20);
list_insert(25);
list_insert(30);
list_insert(35);
list_print();

printk("*** Slightly Delete(10,25) ***₩n");
list_little_delete();
list_print();

printk("*** All Delete ***₩n");
list_all_delete();
list_print();

return 0;
}

static void list_exit(void) {
printk("Kernel unloaded₩n");
}

module_init(list_init);
module_exit(list_exit);

MODULE_LICENSE("GPL");
```

(글자크기는 10으로 유지하고 줄간격도 160%를 유지할 것)