Report

Name: Bijon Setyawan Raya 林湧

Student ID: 104062181

The code below is a struct that consists of each student information from their ID and their birthday.

```
typedef struct _student_info {
  int student_id, day, month, year;
  struct list_head list; //list of all student structures
} student_info;
```

So what is `struct list_head list`? It's a list that consists of all student_info structures.

The next part will be `static LIST_HEAD(students)`. Since I want to iterate from one node to another, I need a special pointer that refers to my Linked List without being a list node itself. We can think that as a head pointer declaration for my Linked List called `students`.

```
int add_student(int student_id, int day, int month, int year) {
  student_info *student;
  student = kmalloc(sizeof(*student), GFP_KERNEL); // GFP_KERNEL: Normal
allocation of kernel memory
  student->student_id = student_id;
  student->day = day;
  student->month = month;
  student->year = year;

  // The most common way of initializing the linked list at running time
  INIT_LIST_HEAD(&student->list);
  list_add_tail(&student->list, &students);
  return 0;
}
```

For this section of code above, it's just a function to add each student one by one and using their information as the parameters of the functions.

As we can see in the third line, I use `GFP_KERNEL`. It's just a normal way of allocating kernel memory. That line of code means that *kmalloc* can put the current process to sleep waiting for a page when called in low-memory situations.

```
int init_student_list (void) {
  student_info *cursor;

  printk(KERN_INFO "Loading module");
  add_student(107062555, 1, 1, 1994);
  add_student(107065510, 8, 4, 1994);
  add_student(107062031, 15, 7, 1994);
  add_student(107065531, 22, 10, 1994);
  add_student(107065513, 29, 12, 1994);

  // list_for_each_entry (pos, head, member)
  // pos: the type * to use as a loop cursor
  // head: the head of the list
  // member: the name of the list_head in the struct
  list_for_each_entry(cursor, &students, list){
    printk(KERN_INFO "%d, %d-%d-%d", cursor->student_id, cursor->day, cursor-
>month, cursor->year);
  }

  return 0;
}
```

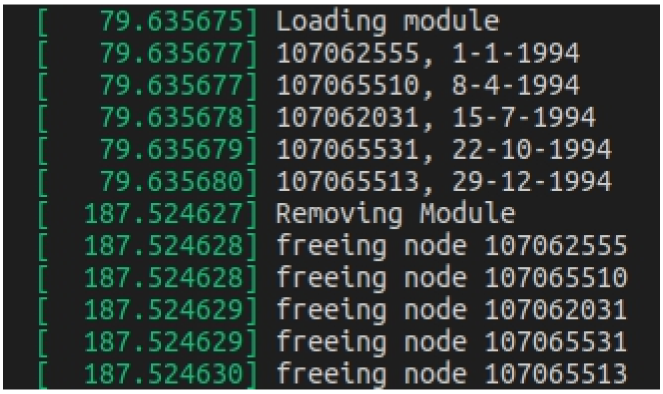
The code section above is the initialization module. Meaning I add the students I want in this function here. At the bottom part of it, I utilized the function that works like a for-loop called `list_for_each_entry` which has three parameters like: `list_for_each_entry(pos, head, member)`. While *pos* is a pointer that works as a cursor, *head* is the first node of the Linked List, and *member* is the list structure of `students_info`.

```
void exit_student_list (void) {
  student_info *cursor, *temp;
  printk(KERN_INFO "Removing Module");

  // list_for_each_entry will break if I delete something while
  // iterating.
  // Therefore, I use list_for_each_safe for deleting each
  // element in the list.
  list_for_each_entry_safe(cursor, temp, &students, list) {
    printk(KERN_INFO "freeing node %d", cursor->student_id);
    list_del(&cursor->list);
    kfree(cursor);
  }
  kfree(temp);
}
```

In this function, I chose `list_for_each_entry_safe()` over `list_for_each_entry()` because `list_for_each_entry()` breaks easily when someone tries to delete a node while iterating through a certain list of nodes. Therefore, I chose `list_for_each_entry_safe()` so I could remove each of the nodes without worrying about memory crash.

Result:



```
[   79.635675] Loading module
[   79.635677] 107062555, 1-1-1994
[   79.635677] 107065510, 8-4-1994
[   79.635678] 107062031, 15-7-1994
[   79.635679] 107065531, 22-10-1994
[   79.635680] 107065513, 29-12-1994
[  187.524627] Removing Module
[  187.524628] freeing node 107062555
[  187.524628] freeing node 107065510
[  187.524629] freeing node 107062031
[  187.524629] freeing node 107065531
[  187.524630] freeing node 107065513
```

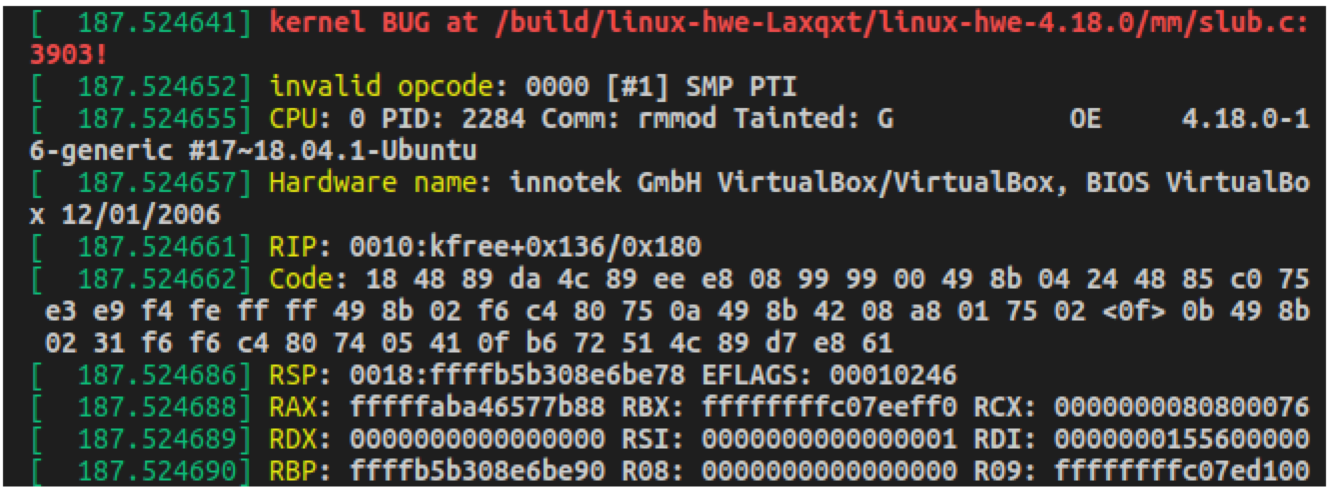
Challenges:

While working on the homework, I had no problems when `make` and `sudo insmod sample.ko` commands were inputted. Once I inputted `sudo rmmod sample.ko`, my computer showed me `Segmentation fault`.

```
bijon@bijon-VirtualBox:~/Downloads$ sudo rmmod sample.ko
Segmentation fault (core dumped)
```

Soon not long after the terminal showing `Segmentation fault (core dumped)`, my Ubuntu started showing me `System Error` and asking me whether I would want to report or not.

Then I realized that my Ubuntu on the VM machine has a bug that I have no idea to troubleshoot.



```
[  187.524641] kernel BUG at /build/linux-hwe-Laxqxt/linux-hwe-4.18.0/mm/slub.c:
3903!
[  187.524652] invalid opcode: 0000 [#1] SMP PTI
[  187.524655] CPU: 0 PID: 2284 Comm: rmmod Tainted: G           OE     4.18.0-1
6-generic #17~18.04.1-Ubuntu
[  187.524657] Hardware name: innotek GmbH VirtualBox/VirtualBox, BIOS VirtualBo
x 12/01/2006
[  187.524661] RIP: 0010:kfree+0x136/0x180
[  187.524662] Code: 18 48 89 da 4c 89 ee e8 08 99 99 00 49 8b 04 24 48 85 c0 75
 e3 e9 f4 fe ff ff 49 8b 02 f6 c4 80 75 0a 49 8b 42 08 a8 01 75 02 <0f> 0b 49 8b
 02 31 f6 f6 c4 80 74 05 41 0f b6 72 51 4c 89 d7 e8 61
[  187.524686] RSP: 0018:ffffb5b308e6be78 EFLAGS: 00010246
[  187.524688] RAX: fffffaba46577b88 RBX: ffffffffc07eeff0 RCX: 0000000080800076
[  187.524689] RDX: 0000000000000000 RSI: 0000000000000001 RDI: 0000000155600000
[  187.524690] RBP: ffffb5b308e6be90 R08: 0000000000000000 R09: ffffffffc07ed100
```

Fortunately, the result in the first picture was still shown right above all of the errors shown in the second and the third pictures.