ЛАБОРАТОРНА РОБОТА 4

Технології розробки модулів ядра для операційної системи Linux.

Мета роботи полягає у оволодінні технологією розробки модулів ядра.

Робота виконується для POSIX ОС, на прикладі ОС Linux.

Робота виконується з використанням бібліотек мови С для POSIX ОС та автоматизації розробки.

Для виконання курсу ЛР потрібен комп'ютер, на якому встановлена POSIX операційна система (ОС). Особливих вимог до потужності персонального комп'ютера (ПК) не визначається.

Завдання для лабораторної роботи 4

Частина 1

Виконати технологічні кроки компіляції і завантаження і вивантаження модуля ядра.

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

MODULE_AUTHOR("Present <https://github.com/danya-psch>");  // Подарунок
MODULE_DESCRIPTION("Lab1 test kernel module"); // Опис фунціоналу
MODULE_LICENSE("MIT"); // ліцензія

/*
 (hello_init)
 (hello_exit).
 module_init i
module_exit використовують специальні макросы ядра
*/

static int __init hw_init(void)
{
        printk(KERN_ERR "Input , Kernel!\n");
        return 0;
}
```

```
static void __exit hw_exit(void)
{
        printk(KERN_ERR "It out from, Kernel!\n");
}
module_init(hw_init);
module_exit(hw_exit);
```

ifneq ($(KERNELRELEASE),)

```
# kbuild part of makefile
obj-m := hw_module.o
else
# normal makefile
KDIR ?= /lib/modules/`uname -r`/build

default:
        $(MAKE) -C $(KDIR) M=$$PWD
clean:
        $(MAKE) -C $(KDIR) M=$$PWD clean
endif
```

Завдання для лабораторної роботи 4

Частина 2

Наданий вихідний код модуля ядра.

#include <linux/kernel.h>

```
#include <linux/module.h>
#include <linux/random.h>
#include <linux/slab.h>
#include <linux/types.h>

MODULE_AUTHOR("");
MODULE_DESCRIPTION("Lab 1.2, using struct list_head");
MODULE_LICENSE("MIT");

#define LIST_LEN 10
#define MSG_PREF "TEST: "
```

```c
#define print_msg(msg, ...) printk(KERN_ERR MSG_PREF msg, ##__VA_ARGS__);

typedef struct {
        struct list_head lnode;
        uint32_t val;
} int_node_t;

// Макрос звільнення пам'яті
// do {}while(0)
#define ilfree(list_head) \
    do {                                                                    \
        int_node_t *__ptr, *__tmp;                                          \
        list_for_each_entry_safe(__ptr, __tmp, (list_head), lnode) {        \
            kfree(__ptr);                                                   \
        }                                                                   \
    } while(0)

// макрос Виводу значень до консоли
#define ilprint(list_head) \
    do {                                                                    \
        int_node_t *__ptr;                                                  \
        print_msg("List: {");                                               \
        list_for_each_entry(__ptr, (list_head), lnode) {                    \
            printk(KERN_ERR "\t%i ", __ptr->val);                           \
        }                                                                   \
        printk(KERN_ERR "}\n");                                             \
    } while(0)

// Ініціалізація списку
//https://elixir.bootlin.com/linux/latest/source/include/linux/list.h#L714
static struct list_head int_list = LIST_HEAD_INIT(int_list);

// Функція для зсуву списку праворуч
static void task(void) {
        int_node_t *ptr;
        list_for_each_entry(ptr, &int_list, lnode) {
                ptr->val <<= 1;
        }
}

static int __init list_module_init(void)
{
        int i;
        int ret = 0;
        print_msg("List allocation start...\n");
        // Виділення памяті під елементи списку
        for (i = 0; i < LIST_LEN; ++i) {
                int_node_t *ptr = (int_node_t *)kmalloc(sizeof(*ptr),
GFP_KERNEL);
                if (!ptr) {
                        print_msg("Can't alloc memory\n");
                        ret = -ENOMEM;
                        goto alloc_err;
                }
                // Функція для роботи
                get_random_bytes(&ptr->val, sizeof(ptr->val));
                list_add_tail(&ptr->lnode, &int_list);
        }
```

```
        print_msg("List allocation finish\n");
        ilprint(&int_list);

        task();

        ilprint(&int_list);

        return 0;
alloc_err:
        ilfree(&int_list);
        return ret;
}

static void __exit list_module_exit(void)
{
        ilfree(&int_list);
        print_msg("Hasta la vista, Kernel!\n");
}
module_init(list_module_init);
module_exit(list_module_exit);
```

Користуючись вихідним кодом, виконати завдання, що надані далі.

4.1 Порахувати кон'юнкцію (&) всіх елементів списку, N = 12

4.2 Знайти Найбільший значення в списку, N = 16

4.3 Знайти найеншее значення в списку, N = 13

4.4 Поділити елементи списку на число M = 25, N = 9

4.5 Зрушити елементи списку на 1 біт вліво, N = 12

4.6 Зрушити елементи списку на 1 біт вправо, N = 12

4.7 Інвертувати елементи списку, N = 10

1. Відповідно до наданих далі варіантів написати модуль ядра.
2. Завантажити модуль ядра. Переконатися у його коректній роботі.
3. Вивантажити модуль ядра. Написати звіт і висновки.
4. Варіанти завдань.

Програма проведення експерименту лабораторної роботи 4

1. Встановити та налаштувати середовище для розрбки модулів ядра.

2. Створити новий проект користуючись додатком. У разі появи помилок внести зміни до вихідного коду.

3. Після успішного виконання п. 1, 2 завантажити модуль ядра.

4. Переконатися у його коректній роботі.

5. Зробити висновки щодо особливостей виконання лабораторної роботи, наявності специфічних особливостей роботи програми.

Теоретичні відомості для ЛР 4

Модуль ядра Linux - це скомпільований двійковий код, який вставляється безпосередньо в ядро Linux, працюючи в кільці 0, внутрішньому і найменш захищеному кільці виконання команд в процесорі x86-64. Тут код виповнюється абсолютно без всяких перевірок, але зате на неймовірній швидкості і з доступом до будь-яких ресурсів системи.

Ядро Linux має модульну структуру. При завантаженні в пам'ять завантажується тільки мінімальне резидентное ядро. Після цього, якщо користувач викликає функцію, відсутню в резидентном ядрі, потрібний модуль ядра, іноді званий драйвером, динамічно завантажується в пам'ять.

Приступимо до написання коду. Підготуємо нашу середу:

mkdir ~ / src / lkm_example
cd ~ / src / lkm_example

запускаємо:

sudo insmod lkm_example.ko

Якщо все нормально, то функція printk забезпечує видачу не в консоль, а в журнал ядра. Для перегляду потрібно запустити:

sudo dmesg

Ви повинні побачити рядок "Hello, World!" з міткою часу на початку. Це означає, що наш модуль ядра завантажився і успішно зробив запис у журнал ядра. Ми можемо також перевірити, що модуль ще в пам'яті:

lsmod | grep "lkm_example"

Для видалення модуля запускаємо:

sudo rmmod lkm_example

Якщо ви знову запустіть dmesg, то побачите в журналі запис "Goodbye, World!". Можна знову запустити lsmod і переконатися, що модуль вивантажився.