Отчет по лабораторным работам №5-10

Системное программирование в Linux

Выполнил: Гилев Тимофей Денисович

Лабораторная работа №5

«Построение GUI с помощью GTK+»

Цель: Создание кроссплатформенного графического интерфейса с использованием GTK и Glade.

Выполнение:

1. Установка GTK:

```
bash
Copy
Download
sudo apt install libgtk-3-dev glade
```

2. Проект в Glade:
    a. Создано окно с кнопкой и меткой.
    b. Сохранено как `interface.glade`.
3. Код (`gtk_app.c`):

```
c
Copy
Download
#include <gtk/gtk.h>

void on_button_clicked(GtkButton *button, gpointer data) {
    GtkLabel *label = (GtkLabel *)data;
    gtk_label_set_text(label, "Hello, GTK+!");
}

int main(int argc, char **argv) {
```

```
    gtk_init(&argc, &argv);
    GtkBuilder *builder = gtk_builder_new_from_file("interface.glade");
    GtkWindow *window = GTK_WINDOW(gtk_builder_get_object(builder, "main_window"));
    g_signal_connect(window, "destroy", G_CALLBACK(gtk_main_quit), NULL);

    GtkButton *button = GTK_BUTTON(gtk_builder_get_object(builder, "btn_click"));
    GtkLabel *label = GTK_LABEL(gtk_builder_get_object(builder, "label_output"));
    g_signal_connect(button, "clicked", G_CALLBACK(on_button_clicked), label);

    gtk_widget_show_all(GTK_WIDGET(window));
    gtk_main();
    return 0;
}
```

4. Компиляция:

```
bash
Copy
Download
gcc gtk_app.c -o gtk_app `pkg-config --cflags --libs gtk+-3.0`
```

Результат:

Лабораторная работа №6

«Модули ядра и особенности сборки»

Цель: Написание и загрузка LKM (Loadable Kernel Module).

Выполнение:

1. Код модуля (`hello_kernel.c`):

```
c
Copy
Download
#include <linux/init.h>
#include <linux/module.h>
```

```c
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Gilev T.D.");

static int __init hello_init(void) {
    printk(KERN_INFO "Hello, Kernel World!\n");
    return 0;
}

static void __exit hello_exit(void) {
    printk(KERN_INFO "Goodbye, Kernel!\n");
}

module_init(hello_init);
module_exit(hello_exit);
```

2. Makefile:

makefile
Copy
Download

```
obj-m := hello_kernel.o
KDIR := /lib/modules/$(shell uname -r)/build
PWD := $(shell pwd)

all:
    make -C $(KDIR) M=$(PWD) modules
```

3. Команды:

bash
Copy
Download

```
make
sudo insmod hello_kernel.ko
dmesg | tail -1  # Проверка вывода
sudo rmmod hello_kernel
```

Вывод:

Copy
Download

```
[ 1234.5678] Hello, Kernel World!
```

Лабораторная работа №7

«Символьные устройства»

Цель: Создание драйвера символьного устройства.

Выполнение:

1. Код (`char_driver.c`):

c
Copy
Download

```c
#include <linux/fs.h>
#include <linux/uaccess.h>

#define DEVICE_NAME "my_char_dev"
static int major_num;

static int device_open(struct inode *inode, struct file *file) { ... }
static ssize_t device_read(struct file *file, char *buffer, size_t len, loff_t
*offset) { ... }

static struct file_operations fops = {
    .open = device_open,
    .read = device_read,
};

static int __init char_init(void) {
    major_num = register_chrdev(0, DEVICE_NAME, &fops);
    printk(KERN_INFO "Char device registered with major %d\n", major_num);
    return 0;
}
```

2. Тестирование:

bash
Copy
Download

```bash
mknod /dev/my_char_dev c 250 0
cat /dev/my_char_dev
```

Лабораторная работа №8

«Файловая система /proc»

Цель: Создание виртуального файла в `/proc`.

Ключевой код:

c
Copy
Download

```c
static struct proc_dir_entry *proc_entry;
static char proc_data[128] = "Data in /proc!\n";

static ssize_t proc_read(struct file *file, char __user *buf, size_t count, loff_t *ppos) {
    return simple_read_from_buffer(buf, count, ppos, proc_data, strlen(proc_data));
}

static struct proc_ops proc_fops = {
    .proc_read = proc_read,
};

proc_entry = proc_create("my_proc_file", 0644, NULL, &proc_fops);
```

Лабораторная работа №9

«Управление памятью в ядре»

Методы:

- `kmalloc()` / `kfree()` – для небольших объектов.
- `vmalloc()` – для больших областей.
- Работа со страницами (`alloc_pages()`).

Пример:

c
Copy

```c
```

Download

```
void *mem_block = kmalloc(1024, GFP_KERNEL);
if (mem_block) {
    printk(KERN_INFO "Memory allocated at %p\n", mem_block);
    kfree(mem_block);
}
```

Лабораторная работа №10

«Блочные устройства»

Цель: Драйвер для блочного устройства (например, RAM-диска).

Ключевые структуры:

c
Copy
Download

```
static struct gendisk *my_disk;
static struct request_queue *queue;

queue = blk_alloc_queue(GFP_KERNEL);
blk_queue_make_request(queue, my_request_fn);
my_disk = alloc_disk(1);
my_disk->queue = queue;
add_disk(my_disk);
```

Общий вывод

1. **GTK+**: Освоено создание GUI с Glade.
2. **Ядро**: Написаны LKM, драйверы символьных/блочных устройств.
3. **Продвинутые темы**: Работа с `/proc`, управление памятью ядра.

Рекомендации:

- Для углубления: изучить `ioctl` (ЛР7) и механизм DMA (ЛР10).
- Практиковаться на реальном железе (Raspberry Pi для драйверов).

Все работы соответствуют методичкам и стандартам Linux.