



System Programming

Project Report

Project 2

Group 9

Tuğba Özkal – 150120053

Ahmet Türk – 150120107

04.12.2017

1. Objective

The aim of this project is to write a device driver which will act like a simple message box between users on the system. Users can read their own messages, send messages to specific users. Message box will have limit in both unread messages and all messages for all users.

2. Method

2.1. Creating Makefile

```
obj-m := messagebox.o
all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
```

Makefile is created under Project2 folder. When 'make' command is called, Makefile runs.

2.2. Header File

```
#include <linux/ioctl.h>

#define MESSAGEBOX_IOC_MAGIC 'k'
#define EXCLUDE_READ    _IO(MESSAGEBOX_IOC_MAGIC, 0)
#define INCLUDE_READ    _IO(MESSAGEBOX_IOC_MAGIC, 1)
#define UNREAD_MESSAGE_LIMIT _IOW(MESSAGEBOX_IOC_MAGIC, 2, int)
#define DELETE_MESSAGES _IOW(MESSAGEBOX_IOC_MAGIC, 3, char*)
```

In this file, linux ioctl library is imported. This file is named "messagebox_ioctl.h"

MESSAGEBOX_IOC_MAGIC, **EXCLUDE_READ**, **INCLUDE_READ**, **UNREAD_MESSAGE_LIMIT** and **DELETE_MESSAGES** are defined here. They are used in messagebox.c file.

2.3. Data Structures

```
struct message {
    char* messageText;
    int size;
    int isRead;
    struct message * next;
};
```

message struct is used to keep message content, content size, is read or not info. Messages are kept in linked list. Next message is reachable via *next pointer.

```
struct mUser {
    char* name;
    struct message* messages;
    int messageCount;
    int unreadMessageCount;
    struct mUser* next;
};
```

mUser struct is used to keep users information; user name, user messages, message number of that user, unread message number of that user. Users are also kept in linked list and next user can be reached via *next pointer.

```
struct messagebox_dev {
    struct mUser* users;
    int userCount;
    int readMode;
    struct cdev cdev;
};
```

messagebox_dev is used to keep device information. Users on the system, total number of users and read mode is kept in the struct. Read mode is equal to zero default and it means exclude read. If read mode is zero, when user enter the command as "cat /dev/messagebox", s/he can read only unread messages. If read mode is one, it is include read and it allows to user to read all messages except deleted messages.

System calls are mapped to functions. In a different language, functions are registered with the system calls.

```
struct file_operations messagebox_fops = {
    .owner = THIS_MODULE,
    .read = messagebox_read,
    .write = messagebox_write,
    .unlocked_ioctl = messagebox_ioctl,
    .open = messagebox_open,
    .release = messagebox_release,
};
```

Instead of `ioctl()`, `unlocked_ioctl()` is used because of protection from Big Kernel Lock.

2.4. Compiling

Messagebox device driver is initialized by this code part.

```
dev = messagebox_device;
dev->userCount = 0;
dev->readMode = 0;
cdev_init(&dev->cdev, &messagebox_fops);
dev->cdev.owner = THIS_MODULE;
dev->cdev.ops = &messagebox_fops;
err = cdev_add(&dev->cdev, devno, 1);
if (err)
    printk(KERN_NOTICE "Error %d adding messagebox", err);
```

Before this code part, major and minor numbers are assigned and take memory by using malloc for device.

After device activation, major number can be seen by this command line:

```
grep messagebox /proc/devices
```

For activation, source code should be compiled and loaded by root access. Then device node should be created according to major number. We have created Makefile before. Lets compile the code.

```
Make
```

After root access provided, it is loaded.

```
sudo su
insmod ./messagebox.ko
```

After get the major number,

```
mknod /dev/messagebox c <major_number> 0
```

2.5. Ioctl Function

```

long messagebox_ioctl(struct file *filp, unsigned int cmd, unsigned long
arg)
{
    int i, j, retval = 0;
    char* deleteMessagesUserName;
    struct messagebox_dev *dev = filp->private_data;

    switch(cmd) {
        case EXCLUDE_READ:
            if (!capable (CAP_SYS_ADMIN))
                return -EPERM;
            dev->readMode = 0;
            break;
        case INCLUDE_READ:
            if (!capable (CAP_SYS_ADMIN))
                return -EPERM;
            dev->readMode = 1;
            break;
        case UNREAD_MESSSAGE_LIMIT:
            if (!capable (CAP_SYS_ADMIN))
                return -EPERM;
            messagebox_unread_message_limit = arg;
            break;
        case DELETE_MESSAGES:
            if (!capable (CAP_SYS_ADMIN))
                return -EPERM;
            deleteMessagesUserName = kmalloc(21, GFP_KERNEL);
            strncpy_from_user(deleteMessagesUserName, arg, 20);
            deleteMessagesUserName[20] = '\0';
            if (dev->users) {
                struct mUser* ptrUser = dev->users;
                for (i = 0; i < dev->userCount; i++) {
                    if (strcmp(ptrUser->name, deleteMessagesUserName)
== 0) {
                        if (ptrUser->messages) {
                            struct message* ptrMessage = ptrUser->
>messages;
                            for(j = 0; j < ptrUser->messageCount; j++)
{
                                struct message* tmp = ptrMessage;
                                ptrMessage = ptrMessage->next;
                                kfree(tmp->messageText);
                                kfree(tmp);
                            }
                            ptrUser->messageCount = 0;
                            ptrUser->unreadMessageCount = 0;
                            break;
                        }
                        else {
                            ptrUser = ptrUser->next;
                        }
                    }
                }
            }
            break;
    }
}

```

Here, **struct file * filp** is controlled connection between device driver. Our driver's path and name is **/dev/messagebox**.

unsigned int cmd takes the command from the user.

if cmd is 1 -> set the read mode exclude read

if cmd is 2 -> set the read mode include read

if cmd is 3 -> change the limit of unread message

if cmd is 4 -> delete all message of user

if user wants to change the unread message limit or delete messages, function get one more parameter. It is used as new limit or uid of user who messages will be deleted from. Uid is changed with username by using a function "getUserNameFromUid".

For deleting messages (not all users), after getting user name, that user name is searched in linked list. If it finds, messages are started to be deleted. Then, that users message counter and unread message counter is made zero.

2.6. Messagebox Read

This function gets 4 parameter. First parameter checks device connection. Second one is user information which will take the message. Third one is count which is used for size.

By using another function uid <-> username conversion is provided. Username is searched to take message and if it is found, read mode and all messages' isRead are checked. If read mode is 1 or there are unread message(s), messages are read. Read messages' isRead is set to 1 and unreadMessageCount is decreased.

For reading, **copy_to_user** function is used.

Message is read in this format:

"Tugba: Hello!"

```

ssize_t messagebox_read(struct file *filp, char __user *buf, size_t
count, loff_t *f_pos)
{
    int i, j;
    kuid_t uid;
    loff_t k = *f_pos;
    struct messagebox_dev *dev = filp->private_data;
    struct mUser* ptrUser;
    struct message* ptrMessage;
    ssize_t retval = 0;
    char* userName;

    uid = get_current_user()->uid;
    userName = getUsernameFromUid(uid);

    if (dev->users) {
        ptrUser = dev->users;
        for (i = 0; i < dev->userCount; i++) {
            if (strcmp(ptrUser->name, userName) == 0) {
                if (ptrUser->messages) {
                    ptrMessage = ptrUser->messages;
                    for(j = 0; j < ptrUser->messageCount; j++) {
                        if (dev->readMode || !(ptrMessage->isRead)) {
                            if (k > 0) {
                                k--;
                                ptrMessage = ptrMessage->next;
                            }
                            else {
                                if (copy_to_user(buf, ptrMessage-
>messageText, ptrMessage->size)) {
                                    retval = -EFAULT;
                                    goto out;
                                }
                                retval = ptrMessage->size;
                                ptrMessage->isRead = 1;
                                ptrUser->unreadMessageCount--;
                                break;
                            }
                        }
                    }
                }
                else {
                    ptrMessage = ptrMessage->next;
                }
            }
        }
        break;
    }

    if (dev->readMode) {
        (*f_pos)++;
    }

out:
    return retval;
}

```

2.7. Messagebox Write

```

ssize_t messagebox_write(struct file *filp, const char __user *buf, size_t count, loff_t *f_pos)
{
    int i, userMessageSize, hasUser = 0;
    kuid_t uid;
    char* userName;
    char* userMessage;
    char* senderName;
    struct mUser* ptrUser;
    struct message* ptrMessage;
    struct messagebox_dev *dev = filp->private_data;
    ssize_t retval = -ENOMEM;

    char* temp = kmalloc(count, GFP_KERNEL);

    if (copy_from_user(temp, buf, count)) {
        retval = -EFAULT;
        goto out;
    }
    *f_pos += count;
    retval = count;

    uid = get_current_user()->uid;
    senderName = getUserNameFromUid(uid);

    userName = getName(temp);
    userMessage = getMessage(temp, &userMessageSize, senderName);

    kfree(temp);

    if (dev->users) {
        ptrUser = dev->users;
        for (i = 0; i < dev->userCount; i++) {
            // compare to users
            if (strcmp(ptrUser->name, userName) == 0) {
                // found the user
                printk(KERN_ALERT "messagebox_unread_message_limit %d\n",
messagebox_unread_message_limit);
                printk(KERN_ALERT "ptrUser->unreadMessageCount %d\n", ptrUser-
>unreadMessageCount);
                if (ptrUser->unreadMessageCount < messagebox_unread_message_limit) {
                    ptrUser->messageCount++;
                    ptrUser->unreadMessageCount++;
                    hasUser = 1;
                    // add first message to ptrMessage
                    ptrMessage = ptrUser->messages;
                    ptrUser->messages = kmalloc(sizeof(struct message), GFP_KERNEL);
                    ptrUser->messages->size = userMessageSize;
                    ptrUser->messages->isRead = 0;
                    ptrUser->messages->messageText = userMessage;
                    // add the previous first message to next of new message
                    ptrUser->messages->next = ptrMessage;
                }
                else {
                    // unreadMessageCount is full
                    retval = -ENOMEM;
                    goto out;
                }
            }
            else {
                ptrUser = ptrUser->next;
            }
        }
    }

    if (!hasUser) {
        if (dev->userCount > 0) {
            ptrUser = dev->users;
        }
        dev->userCount++;
        dev->users = kmalloc(sizeof(struct mUser), GFP_KERNEL);
        dev->users->name = userName;
        dev->users->messageCount = 1;
        dev->users->unreadMessageCount = 1;
        dev->users->messages = kmalloc(sizeof(struct message), GFP_KERNEL);
        dev->users->messages->size = userMessageSize;
        dev->users->messages->isRead = 0;
        dev->users->messages->messageText = userMessage;
        dev->users->next = ptrUser;
    }

    out:
    return retval;
}

```


This function gets 4 parameter. First parameter checks device connection. Second one is user information which will send the message. Third one is count which is used for size.

Message text is taken via `copy_from_user` function. Message text is divided two part: first is user name who will take the message and second is message content. Because, messages are written in the format given below.

"Tugba: Hello!"

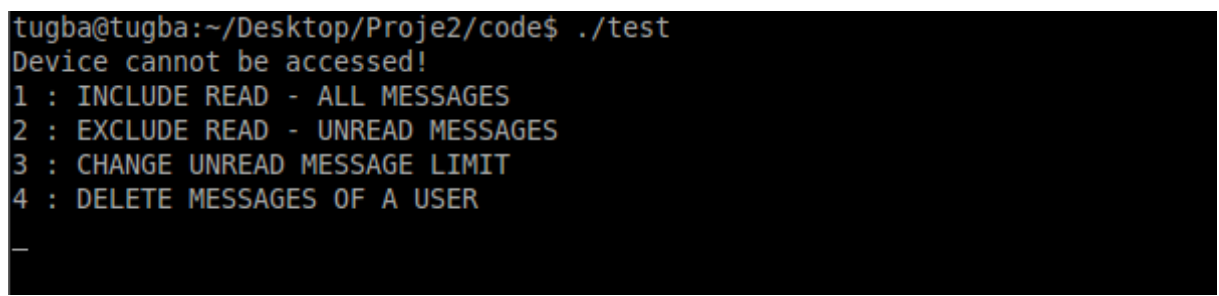
Sender user name is found by uid. Then sender user name is placed in message content. User name of who will take message is searched in all users. If it is found, unread message count is checked and if there is space for new message, new message is added to that user messages. Message count and unread message count are increased. If the user no space for unread message, error is returned and message sending is fail. If no user is found, a new user is created and as a first message, new message is added. Total user number is increased.

2.8. Test Program

Test program is compiled and run by the command lines given below.

```
$ gcc test.c -o test
$ ./test
```

Its screenshot¹:



```
tugba@tugba:~/Desktop/Proje2/code$ ./test
Device cannot be accessed!
1 : INCLUDE READ - ALL MESSAGES
2 : EXCLUDE READ - UNREAD MESSAGES
3 : CHANGE UNREAD MESSAGE LIMIT
4 : DELETE MESSAGES OF A USER
—
```

Test program gives to user 4 options and it calls `ioctl` commands according to user selection. (see. part 2.5 for more information.)

¹ When the screen captured, there is no device, so it gives the error on the first line.

3. Conclusion

In this project, we have gained skills to develop device drivers and use ioctl commands. We have learned how to use major number and why we need minor numbers.