

BLG413E System Programing

Project 2

Yunus Güngör – 150150701

Duhan Cem Karagöz – 150130066

**Objective**

This project helps to gain experience with Linux environment and drivers in Linux operating systems. Objective of this project is to implement a character device that acts as a message box between users in the system.

**Environment**

In this project Lubuntu 14.04 with GCC 4.7 is used with C programing language. Program created in this project runs in kernel space as a module.

**Task Distribution**

Required tasks has been distributed among two members of our team. Yunus Güngör handled with driver installation, driver initialization, message input outputs and message data. Duhan Cem handled with setting up environment for a basic driver, writing test program, ioctl commands and memory cleanup.

**Implementation**

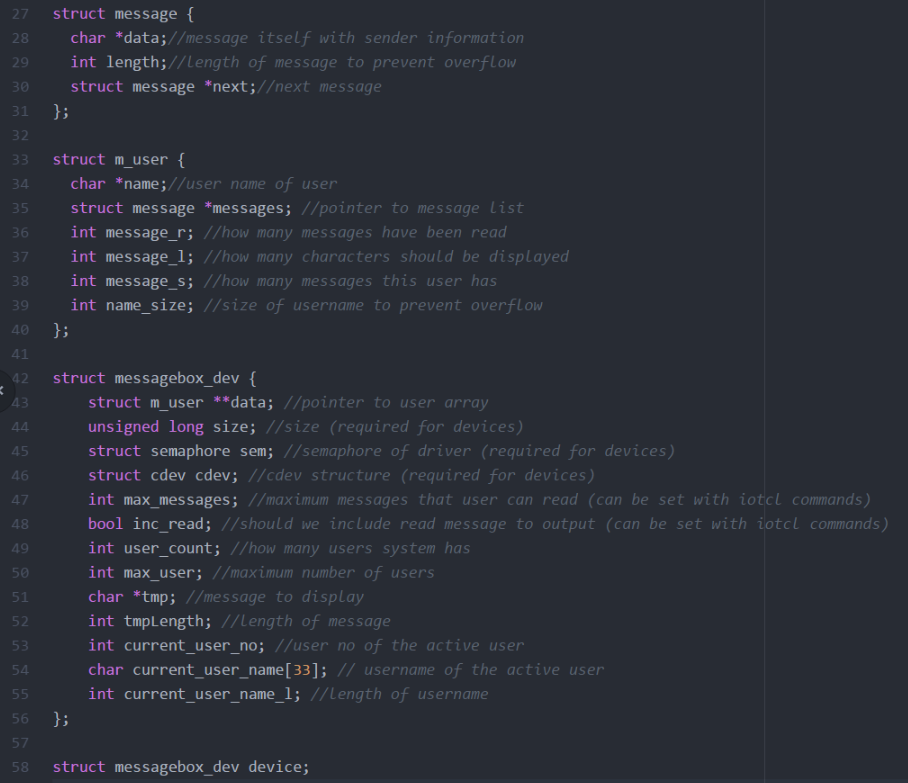
Implementing a message box requires a data structure to keep messages and users in. In this project we have implemented a general structure for keeping all the data we need. This structure called messagebox\_dev and used with device variable in the program. This structure contains, an user array and user array contains a linked list to keep all messages send to that user. Message’s sender information kept directly in the message text. For example, if root users send message “hello”, this message kept as “root: hello”. Data structure can be seen at figure 1. Also, user array used as quantum array and messages in message lists used as quantums.

Figure 1-Data Structure

After setting required information like major and minor numbers, required commands to initialize module has been send to operating system. Additionally, required commands for creating device and device node has been send to the operating system to provide automated creation of device. Setup process can be examined in figure 2 and figure 3.

Figure 3 – Device node initializing code

Figure 2-Module and device initializing code

Since all messages has been send with username only, we must implement a mechanism for acquiring username in kernel space. We have achieved that by processing /etc/passwd file and acquiring username section with the matching uid. Uid have been provided by current\_uid function in linux. After getting username, we must match user’s username within our data structure. To accomplish that program traverses user array until it finds a match. Matching uid, username and user id in our data structure has been done in the open function of the device. This function can be examined in figure 4.

Each time device opened which happens before every write and read process, current user’s uid, user name and user id in our data structure has been found and kept in device structure for easy access when write and read functions called. And a temporary string that has user’s unread massages or all messages created. Since read function can be called more than once and there is no way to know that open operation acquired for writing or reading, our implementation creates temporary string in open function. This temporary string only used in the read function. Adding unread messages only or all messages to the temporary string decided by inc\_read variable which can be set by iotcl commands. Maximum unread messages can read by a user implemented in this step too. This value can also be set by iotcl commands.

Write process invoked by a user using “echo @root message >/dev/messagebox” notation. User must have necessary permissions to successfully write on the device. Write function we have implemented gets message as an input from the operating system and process that input to store message. First of all, message in user space, copied to kernel space with necessary memory allocation. After that message processed to get username which is right after ‘@’ symbol. After acquiring user name, program searches on saved data to find user id in our data. If there is no such user, a new user created. Consider that, active user and user that receiver message are different users. They can be same user without a problem, but data structure we implemented keeps messages by message receiver. After finding user to send message, message added to that user’s message list with current user’s username and message. Current user’s name acquired from the data created in open function. If there is unallocated pointers for writing data, necessary allocation have been made. Write function can be examined in figure 5.

Each user can display own messages by simply using “cat /dev/messagebox”. This notation invokes read function. In the read function, we have used tmp pointer in device data structure. This pointer has user’s readable messages in a string form. Read function copies necessary characters to the given pointer with copy\_to\_user function. While coping, global value r\_index used for keeping a cursor, and count variable used for how much read function can copy to pointer. Pointer and count variable provided by operating system. Read function can be examined in figure 6.

When module is removed operating system calls cleanup\_module function. This function frees all the memory acquired before, removes device node, removes device and terminates the module.

More information can be found in the source code.

**Conculision**

In this project, we have learned to implement module and driver in linux operating system. Also we have gained experience with concepts like buffer, node and so on. Writing read function and llseek function helped us to understand linux’s stream logic. We have implemented memory allocation, file proces, basic data structures in kernel space which helped us gain experience with working in kernel space.

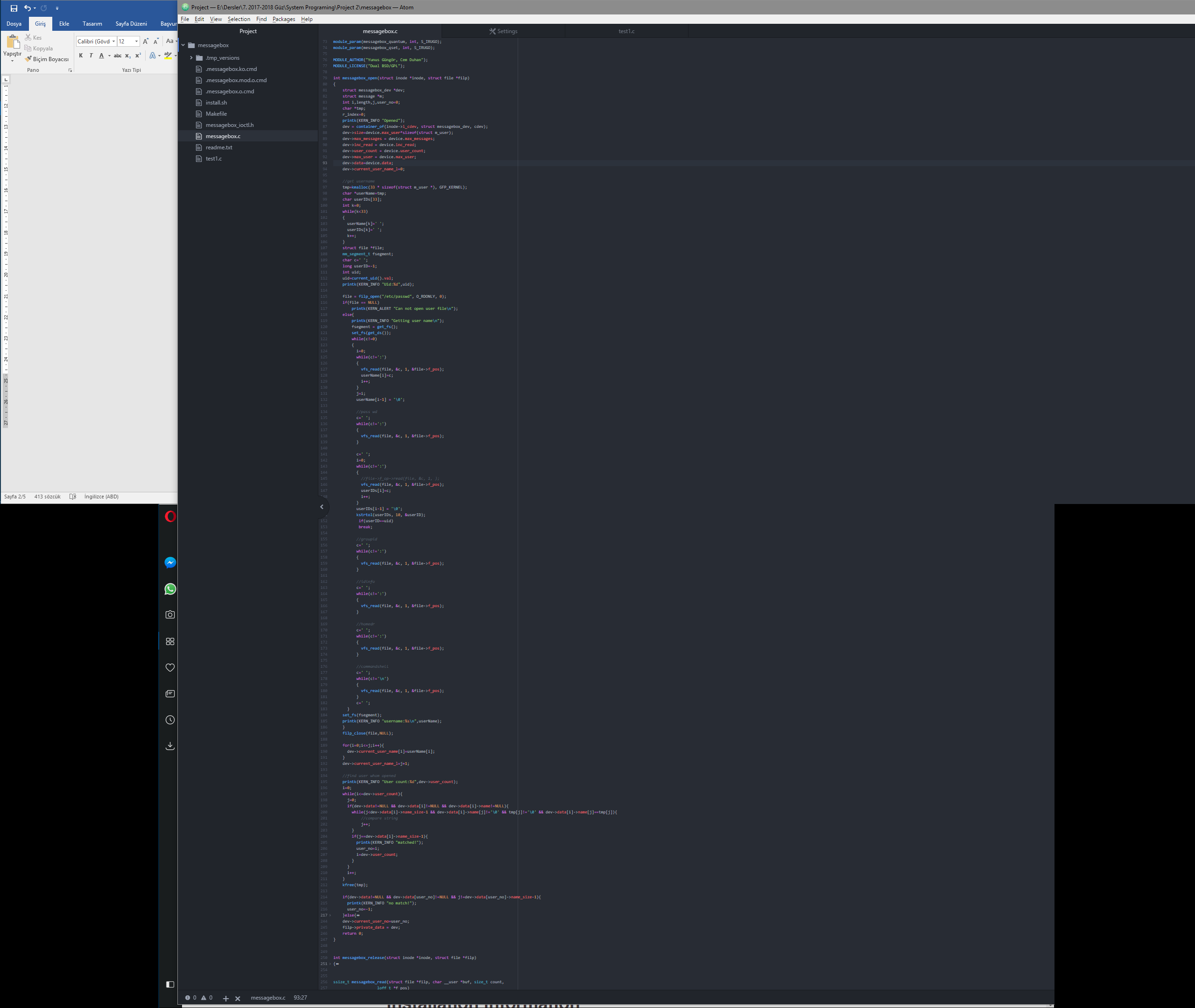


Figure 4 – Open Function handles with username, uid, user id and creates messages string

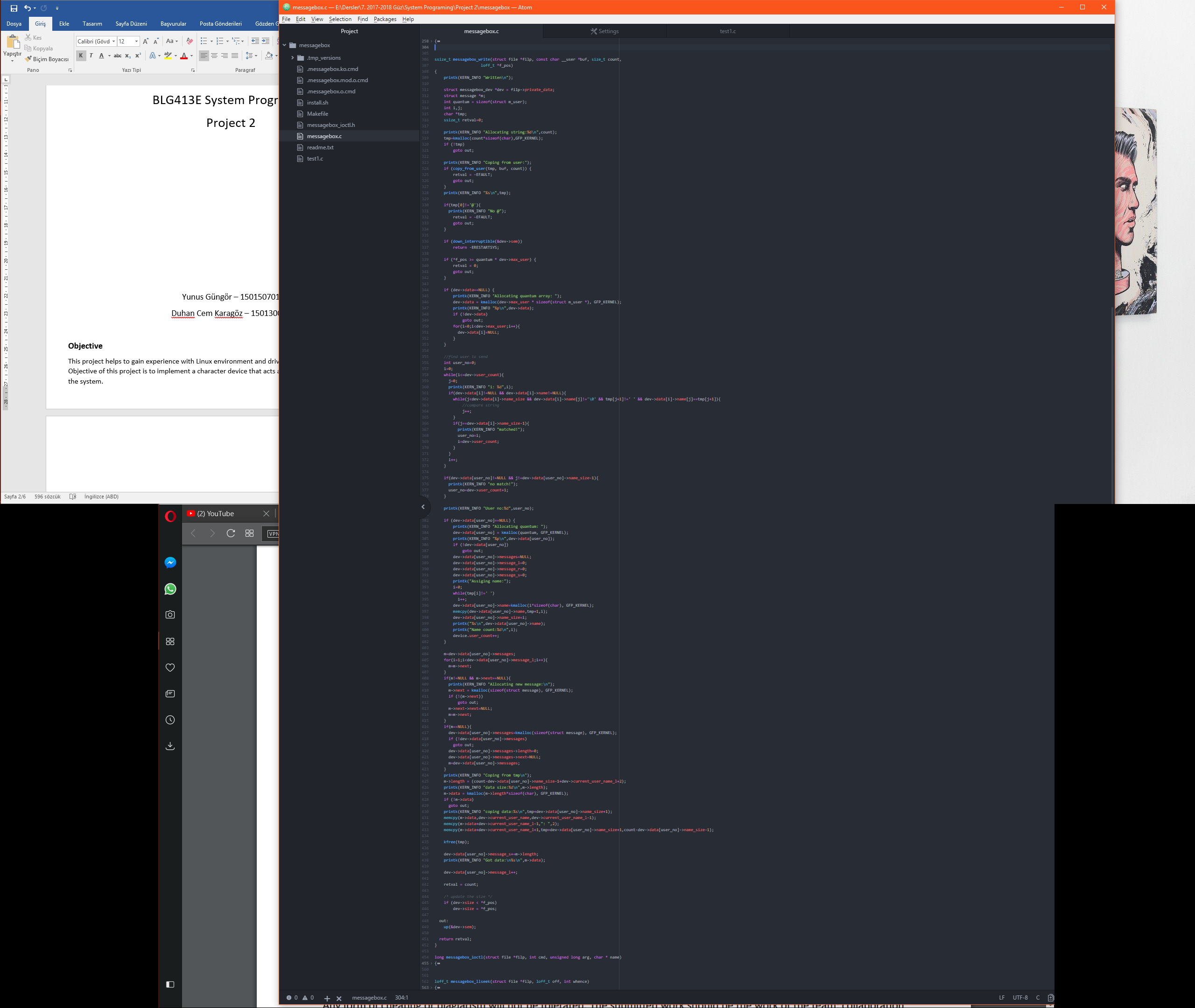


Figure 5 – Write Function handles with incoming messages and memory allocations

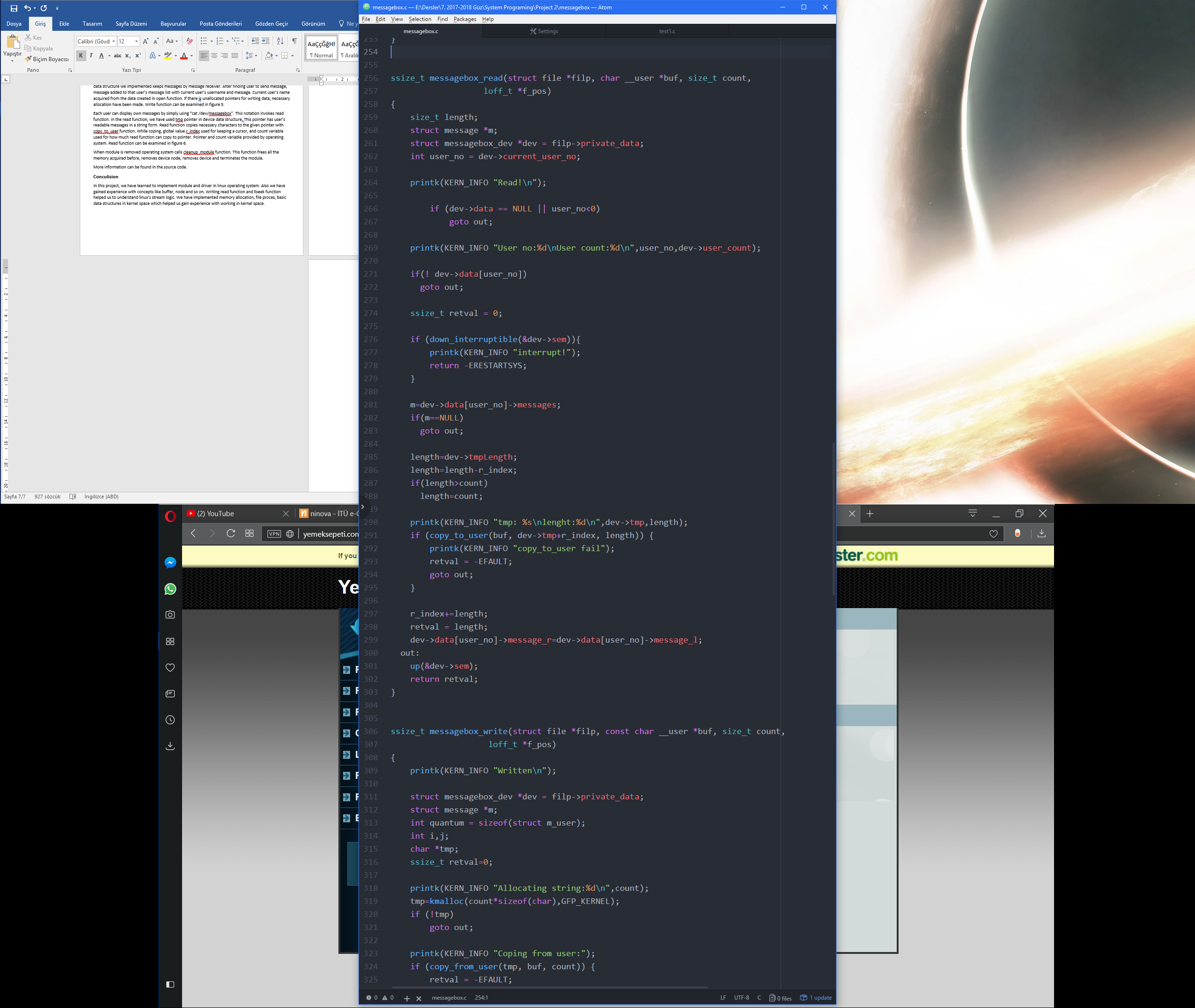


Figure 6 – Read function copies required part of message string