Technical University of Moldova

SOMIPP

Laboratory work #2  
 Topic: Loader and Kernel

***Made by Timbalist Ana***

***Verified by Calin R.***

***Chisinau – 2018***

***Tasks:***

Create 2 programs and bind them to work together.

First program will be the Loader which should be placed into boot sector of your boot device (floppy or usb-stick). This "Loader" program should implement the following subtasks:

1. print message:  
***BootLoader developed by student <FullName>  
Press "1" to start loading the kernel to RAM***

2. If the user press other key (except "1") that on the screen should appear message "**Try again**" from the new line! and the readkey procedure should repeat.

3. If the user press the "**1**" key from the keyboard than the Loader program should start loading the Kernel code from boot device to RAM to some specific **memory\_address**. Addresses will be different for each student  
--> ( the Kernel code should be placed on the floppy image starting from sector Nr.**N** (where N = 3 \* your\_number\_fromstudents\_registry)  
--> ( **memory\_address** where the kernel will be loaded will be = **7c00h + 3 \* 256 \* your\_number\_fromstudents\_registry**)  
  
4. After Loader successfully will load Kernel to the memory you should print the message:  
***Kernel was loaded into RAM to address <complete student's specific ADDRESS>!  
Press "2" to start the Kernel***

5. If the user press other key (except "2") that on the screen should appear message "**Try again**" from the new line! and the readkey procedure should repeat.

6. If the user press the "**2**" key from the keyboard than the Kernel should start and execute!

\*\*\* Like Kernel program each student should use program from first laboratory Work, but Kernel code should be a single program bigger than 512Bytes.

**Implementation:**

*At the early start the BootLoader (MyKernelLoader.asm) will propose the user to start the kernel process pressing button ‘1’. If the user presses corresponding button, the kernel is loaded. Otherwise, the loop will not be ended until the ‘1’ button will be pressed.*

*Each string is printed with help of function ‘Teletype\_String\_procedure’ which uses the string as a parameter (bx points to corresponding one). Taking into consideration that all strings in this small program are implemented as NULL-terminated, it is simple to use the addres of the first character of string and iterate the corresponding char sequence (incrementing di). Initially, di will be set value 0.*

*Another thing implemented is that using int 16h / ah = 00h in the code we ar able to define what we want to get: either BIOS scan code, either ASCII character. Using different instruction sets it’s possible to create ‘while’ loop (until the favorable event will come).*

*How to link Loader and Kernel?*

* *Defining in both MyLoaderKernel.asm and MyKernel.asm the same constants which will indicate the Kernel sector which will be loaded.*

*My ID in the students’ register is 14. It means (by condition) that the Kernel Process should be defined at 3 \* 256 \* 14 (Also adding 7c00h). The kernel sector, respectively, will be defined as 3 \* (3 \* 256 \* 14 + 7c00h).*

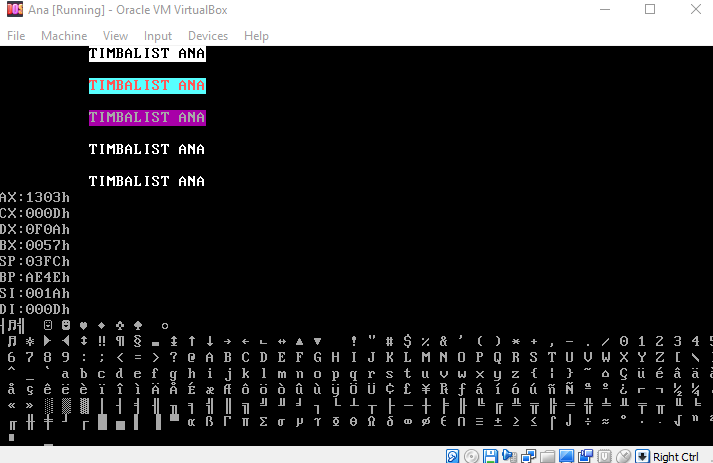
* *The following line in the code will load the Kernel:*

*jmp 0800h: START\_KERNEL\_PROCESS*

*But, first of all, will be integrity check (as it was showed in the emu8086 examples of a tiny OS). For this aim is used nop.*

*The idea of an infinite loop is the same as in the example above. The user will be forced either to press ‘2’, either to close the window. Pressing the button ‘2’, the user will accept starting the Kernel.*

*The Kernel has been started and the result of the previous labs is shown in one screen. For the previous laboratory work it was need to create 3 different floppy images with different tasks. This time I have unified them all into one floppy using a specific script (.bat).*

**