

A Basic Operating System with Functionality for the End User

Menna Ahmed, Yin Leng,
Shane Loong and Muhammad A Qureshi

December 9th, 2016

1 Introduction

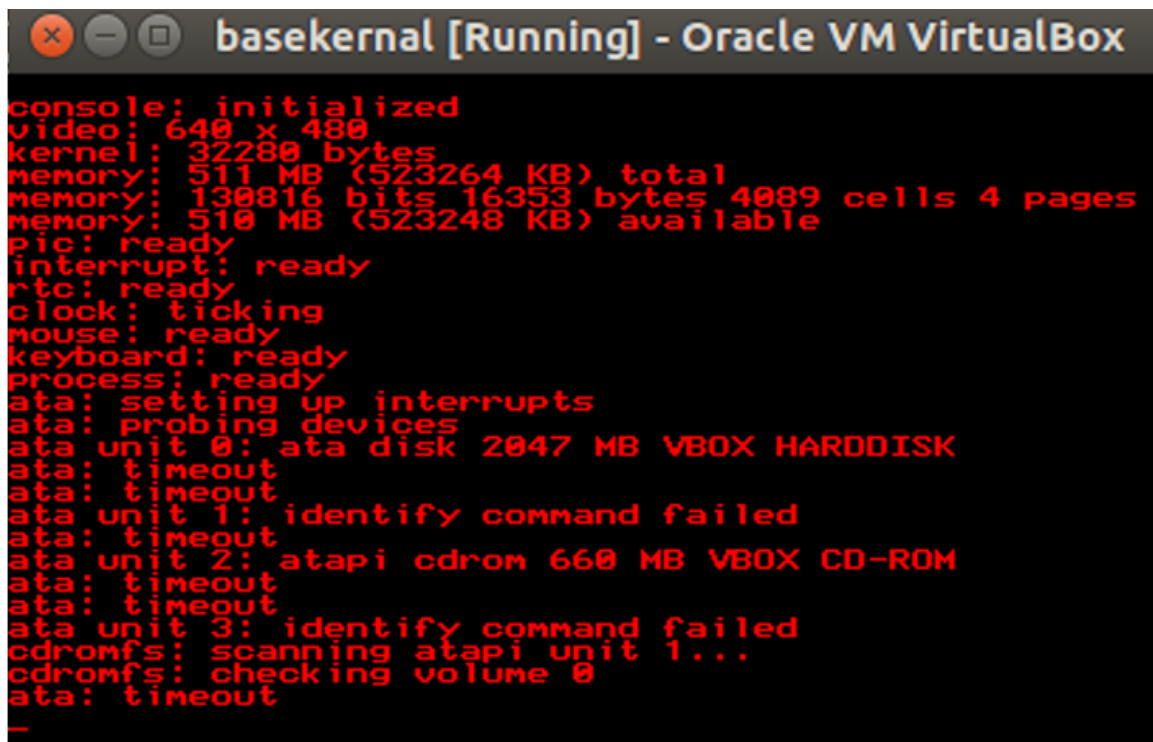
1.1 Context

The ‘basekernel’ by Prof. Douglas Thain [1] of the University of Notre Dame presents a model for a kernel which acts as a general base for a simple Operating System (OS). It acts as an extensible piece of work, which includes adequate booting, memory allocation, process management and a console to present system statistics. The underlying structure is written entirely in C and assembly, which makes it easily interpretable for many students with a generic programming background.

The existing work, when compiled, produces an ISO image which when mounted to a Virtual-Machine (VM), produces a new 640x480 window, that acts as the console. The console then displays statements which confirm to the user what system specifications have been initialized from the time the basekernel underwent booting.

1.2 Problem Statement

In spite of the basekernel presenting a thorough back-ended structure to the OS, it does not present a usable OS. It does not address the end-user usability via the console. Instead, it displays a console with only the initializations of the Video Memory, Page-tables, Interrupt Handlers and other features. Some form of keyboard interaction for the OS and a File-System will definitely benefit this OS. Figure 1.1 shows the console produced after mounting and executing the ISO image of the basekernel.



```
basekernel [Running] - Oracle VM VirtualBox
console: initialized
video: 640 x 480
kernel: 32280 bytes
memory: 511 MB (523264 KB) total
memory: 130816 bits 16353 bytes 4089 cells 4 pages
memory: 510 MB (523248 KB) available
pic: ready
interrupt: ready
rtc: ready
clock: ticking
mouse: ready
keyboard: ready
process: ready
ata: setting up interrupts
ata: probing devices
ata unit 0: ata disk 2047 MB VBOX HARDDISK
ata: timeout
ata: timeout
ata unit 1: identify command failed
ata: timeout
ata unit 2: atapi cdrom 660 MB VBOX CD-ROM
ata: timeout
ata: timeout
ata unit 3: identify command failed
cdromfs: scanning atapi unit 1...
cdromfs: checking volume 0
ata: timeout
_
```

Figure 1.1 - The basekernel, with all initializations and no interaction with the user.

1.3 Result

The results obtained were: the implementation of a keyboard driver, which takes the SCAN codes directly from the key-presses. This driver then translates the numeric SCAN-codes to the corresponding ASCII character. For example, the 'R' key is interpreted to the numerical SCAN-code of 19, which is then translated to the 'r' character. The generic solutions that one thinks of after encountering the problems discussed in Section 1.2 would simply be to call a *scanf()* function in C to process keyboard input, however it is not that simple to address since the basekernel is its own environment. This means that all the libraries and header files cannot be included without defining them yourself. As a result, the only solution that can be finalized is the creation of an original keyboard driver.

The issue of no existing console commands was also addressed by giving the user a predefined set of usable instructions to communicate with the kernel. To add on, after the successful implementation of the keyboard, once the text reached the vertical end of the screen, the console would refresh, which posed the loss of information previously typed. Hence, a possible implementation of a scroll option was attempted as well. These changes together with the original Basekernel, is collectively known as MAGS-OS (Menna's, Awais', Gladys' and Shane's Operating System).

1.4 Outline

Section 2 will describe in detail the concepts and previous works related to the MAGS-OS. Section 3 will describe the actual work we were able to achieve and how we came about the solution. Section 4 will analyze the evaluations we had our OS go through in order to determine its usability and the understandability of the project as a whole.

2 Background Information

MAGS-OS is based on the Basekernel project by Prof. Douglas Thain of the University of Notre Dame. Basekernel is a simple OS kernel created for teaching and research purposes. It is an incomplete OS that uses raw low-level code as a base start for people who want to develop or study the functionality of an OS. This Basekernel can only boot a virtual machine that is PC-compatible with basic system calls and paged virtual memory. When this OS runs, the terminal will show a black screen with red text, and the user will not be able to enter any commands from the keyboard. This OS has a few functionalities that are similar to what users have on a normal everyday OS that most users use.

The Basekernel starts off its roots at the *bootblock*. This BIOS is responsible for the startup of the machine, loading the sectors into memory, and then transferring control off to the kernel. The bootblock calls to the BIOS to load the remaining sectors containing the kernel code and then leaves full control of the program to the kernel.

As all control is given to the kernel, the structure of memory that is to dictate the layout of memory is described in the *memorylayout*. The *memorylayout* describes how the stack is to start at 0xffff0 and grow downward. The total size of the kernel, and the mapping of memory for the *kmalloc()* function is defined as well.

The memory layout of the original OS is structured with a similar layout as discussed in class by Professor Michel Barbeau. Memory is stored in the form of pages and is mapped using a page-table. The base of the memory layout is defined in *memorylayout*, which initializes the startup of *kmalloc*. In terms of memory allocation, the main focus is to be placed on *kmalloc*, *pagetable*, and *memory*.

In particular, the *memory* file deals with initializing the memory for the kernel, the processes, the freeing, counting, and allocating memory for pages. Keeping count of the memory is important so that memory cannot be allocated beyond the limit of what is allocatable. *Pagetable* lays out the foundation of pages in the OS. It defines how the bits of each page are to be structured. Since the basekernel is a 32-bit OS, it allocates 12 bits for the physical address of the page, and the remaining 20 bits as the Page Frame Number, totaling to 32 bits per page. Figure 2.1 describes in detail the page layout set by the basekernel.

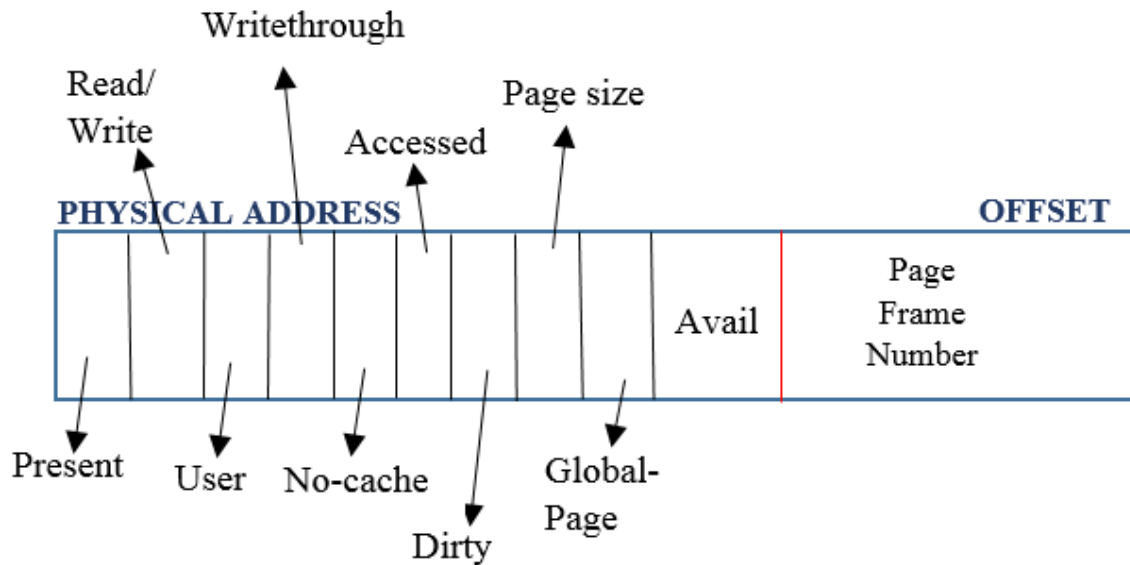


Figure 2.1 - The page layout set by the basekernel

Furthermore, *pagetable* declares an array of pages called *entry* which is to hold all the pages. It then, initializes functions to create, delete, allocate, and map (the process of mapping a physical address to a virtual address, via a mapping medium called a page-table) pages. The process of mapping is described in Figure 2.2.

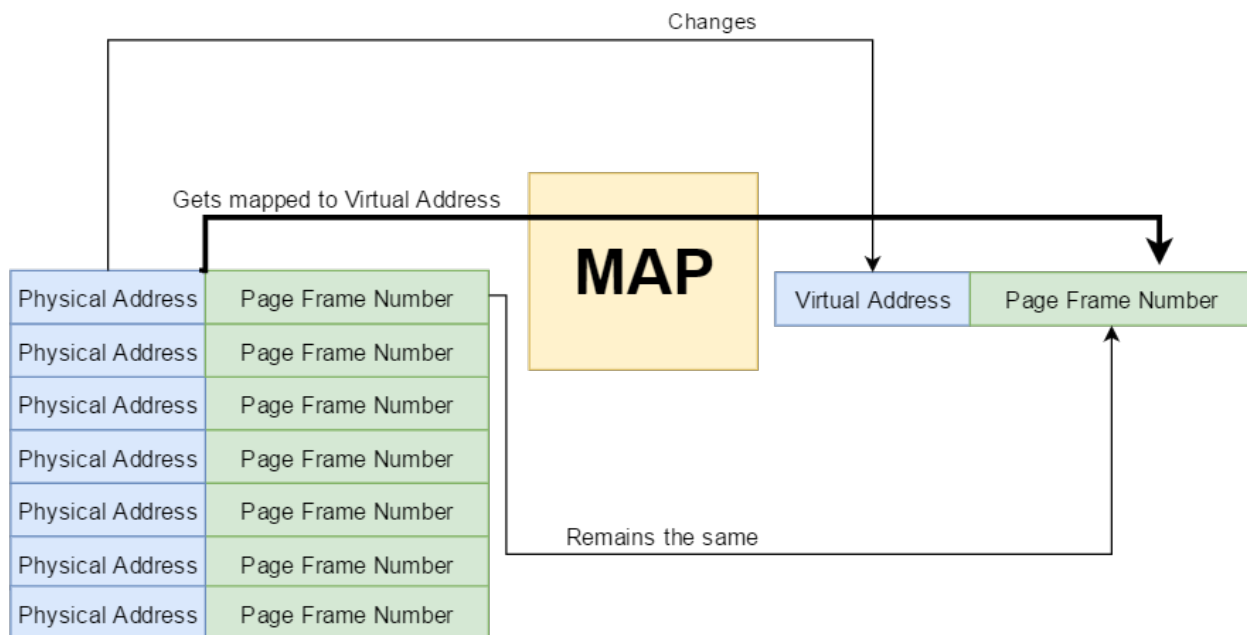


Figure 2.2 - The mapping of physical addresses to virtual addresses via a mapping medium called a page-table.

The last piece of the basekernel that deals with memory allocation is *kmalloc*. This is the actual kernel command which will be used to allocate memory. Memory is stored in this OS in the form of a doubly-linked list. When the *kmalloc* function is called to store some entity in memory, the corresponding *init* function is called in the background. This process creates a chunk of memory that is passed as a parameter. To avoid the fragmentation of memory, when memory is allocated a value that is not divisible by *KUNIT*, the value is rounded to the nearest multiple of *KUNIT*.

3 Result

MAGS-OS is capable of interacting with the user by modifying the original console. Input can be successfully read by the system through a self-created keyboard driver and the appropriate output will be generated. The I/O ports are included in any file that requires that functionality. It includes C code that wrapped around the assembly instructions to communicate with our hardware.

Keyboard implementation starts with the allocation of a buffer with memory size of 200. It is used to store input from the keyboard data port, at memory address 0x60. SCAN-codes are sent to the port and then translated to the appropriate ASCII character, e.g. when R is pressed, which has a scancode of 19, a switch statement in the driver will convert the scancode to the ASCII character and store it into the buffer that was allocated before. Then, the data from the buffer is printed to the console using a custom *printf* function in the *console*. Figure 3.1 describes this process taking place from the time a key is pressed to the output.

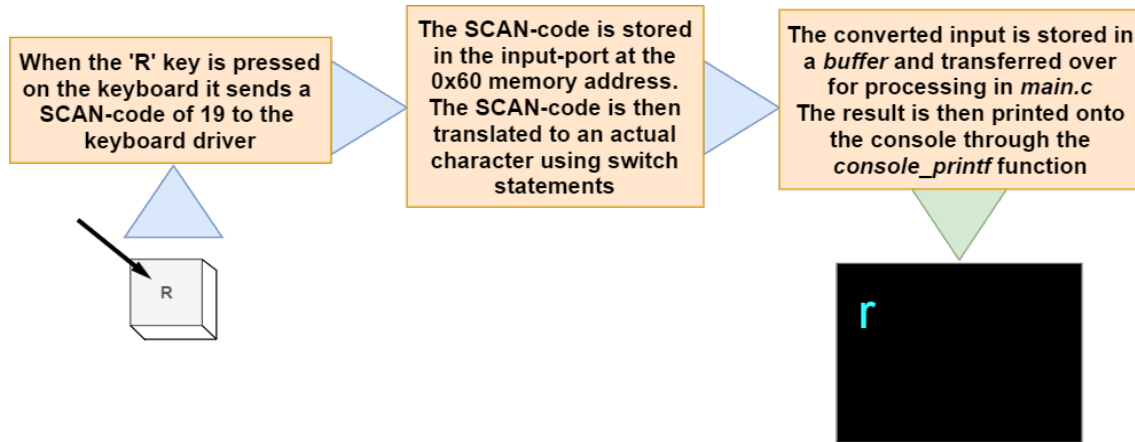


Figure 3.1: Keyboard interface

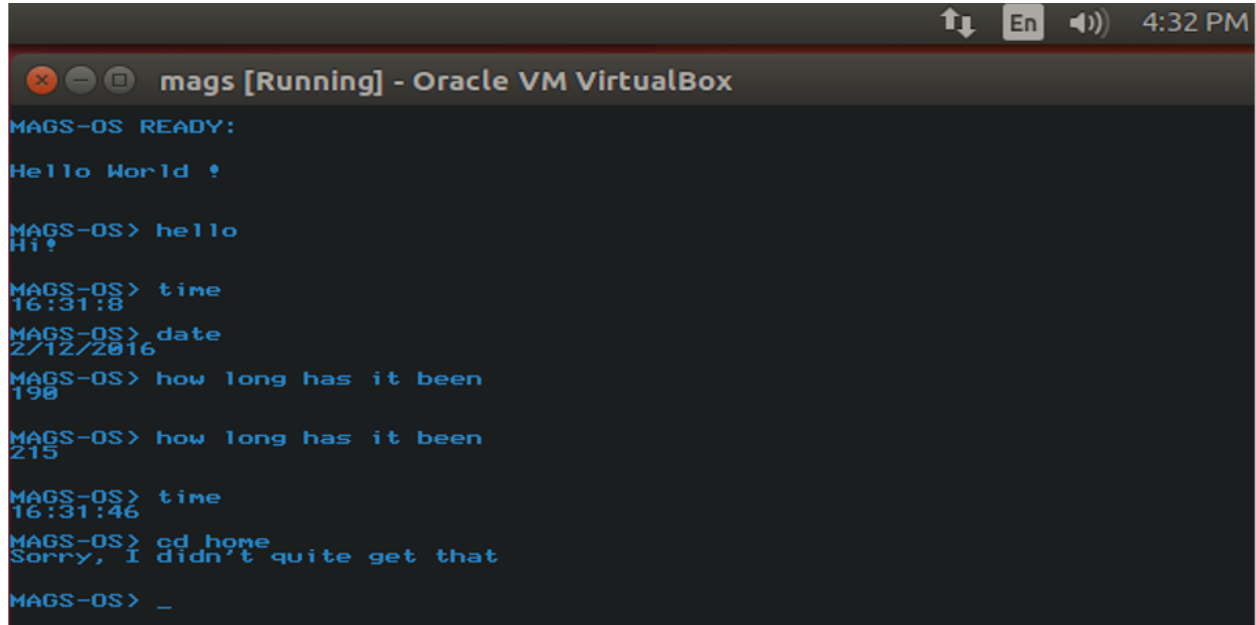
With the keyboard implemented, the opportunity to have user interaction with the system was opened up. After the initialization of basic operations such as memory allocation and interrupts, the modified OS displays a message to welcome the user. A command prompt was created from a do-while loop, which was added in *main* to read commands from the user. The command prompt was customized by having “MAGS-OS >”, to personalize the OS. At this point, because of the interrupts set up by the basekernel, any input entered was regarded as invalid. In order to tackle this issue, the disabling of the ‘Invalid Input’ interrupt had to be done. A corresponding interrupt handler was implemented, which came in the form of the do-while loop mentioned above. The command prompt’s background and font color was adjusted according to the numeric values present in the RGB color table to make it more presentable and soothing.

Moreover, a few commands are given the ability to trigger some functions to meet the user’s needs. For example, the *time* command allows the user to check the current time. When the user enters *time* in the command prompt, the Real Time Clock (RTC) function

rtc_fetch_time() is called, a time struct is created, and the RTC is accessed via I/O ports: the variables are read from their respective addresses, and converted to binary from binary coded decimal (BCD). A temporary string is then created using a custom *strcat()* (String-Concatenation) function, and the string is then returned and displayed (refer to Figure 3.2 for the final result). The command-prompt run time can also be checked by typing “*how long has it been*”. Upon receiving this command, the *clock_init()* function is called, which is basically a timer that starts ‘ticking’ once the shell runs. The time is stored in the 0x40 address and is updated every millisecond. Users can also get the current date by typing *date*. The *rtc_fetch_date()* function was implemented to get the day, month and year from the port by calling *rtc_read_port()*. It works similarly to the existent time command. The results are concatenated and formatted into a single string and returned. A formatted string of date is displayed on the console when date command is entered (refer to Figure 3.2 for the final result). When the user enters *clear* all previous video memory is cleared and a fresh terminal is started with the prompts “MAGS OS>”, therefore instilling the fact that the user has now ‘refreshed’ the screen and starts anew.

Furthermore, a scrolling feature was attempted, with some minor success. The *scrollup(...)* function was used to move pixels up by a factor of 50 vertical units. This function processes the memory (words) allocated for the strings printed on the console and then maps it directly above by ‘y’ units (which in this case was ‘5’). However, an error was encountered because of the memory layout of the new OS, since it causes memory to run out, and as a result, the screen does not get updated, and the console

eventually crashes. Figure 3.2 shows MAGS-OS, without the scrolling feature, as the VM would crash every time the scrolling feature would be tested.



```
MAGS-OS READY:
Hello World !

MAGS-OS> hello
Hi!

MAGS-OS> time
16:31:8
MAGS-OS> date
2/12/2016
MAGS-OS> how long has it been
190
MAGS-OS> how long has it been
215
MAGS-OS> time
16:31:46
MAGS-OS> cd home
Sorry, I didn't quite get that
MAGS-OS> _
```

Figure 3.2: Snapshot of interactive MAGS OS

4 Evaluation

MAGS-OS was evaluated by 10 surveyees with varying degrees of OS knowledge. The surveyees were shown the old basekernel, and the modified one, and were asked to interact with both of them.

All 10 surveyees (100%) preferred the new version of the OS (MAGS-OS) and were delighted to have the convenience and the privilege of some form of input. They were also pleased to not see any uninterpretable system statistics on the screen. However, all did also expect some form of use of this OS - including a file system and GUI. One of the surveyees also commented on the fact that there was no Upper-Case input in the OS which was quite unusual.

The overall notion gathered from the survey was that although MAGS-OS was an upgrade from basekernel, there were still many features that end-users would like to see implemented for their satisfaction.

5 Conclusion

5.1 Summary

Some of the modifications made to the basekernel were: displaying a kernel command prompt that reads some commands from the user via a self-made keyboard driver, modifying the console code so that it scrolls the display up when the user reaches the end of the screen, rather than clearing the screen and starting over, and implementing a user command that displays the process run time, the current time and date.

5.2 Relevance

The aspects related to the course dealt with in the MAGS-OS are: creation of a terminal emulator on a Virtual Machine, booting the operating system, loading the kernel into memory, implementing a keyboard driver into an operating system, augmenting an existing kernel with user interaction, and using a system clock.

5.3 Future Work

There is a lot that could be done in the future to extend the work. For example, when trying to write to an address somewhere into the (unmapped) user address space above 0x80000000, a page fault occurs and the system halts. The page fault handler could be modified to automatically allocate a fresh memory page and map it into the process

each time this happens. Also, the bug in the scroll up feature that was addressed in section 3 could also be fixed. Moreover, the OS currently lacks a file-system, which we tried to implement but could not achieve it, so that could also be implemented.

Contribution of Team Members

- A. Muhammad implemented the shell, keyboard, and wrote sections 1, 2, and 4 of the report. He also did the diagrams.
- B. Menna implemented the keyboard, and a user command feature that displays the current time and date. She also wrote the conclusion, and was responsible for the final editing of the report.
- C. Gladys implemented the keyboard, and added a scroll option as the output goes beyond the vertical limit of the screen. She also wrote section 3 of the report (result).
- D. Shane implemented the shell, the keyboard and did a part of section 2.

References

[1] D. Thain, *The Baskernel Operating System* -
<https://github.com/dthain/basekernel>. Accessed: October 30th 2016.

How to Mount an ISO in VirtualBox by Mark Pool. Available from:
<https://www.techwalla.com/articles/how-to-mount-an-iso-in-virtualbox>. Accessed: October 3rd 2016.

RGB Color Codes Chart.
http://www.rapidtables.com/web/color/RGB_Color.htm.
Accessed: November 1st, 2016.

Keyboard scancodes.
<https://www.win.tue.nl/~aeb/linux/kbd/scancodes-11.html>.
Accessed: November 3rd, 2016.