

Department of Computer Science and Engineering

Faculty of Engineering, University of Moratuwa

CS 2040 — Operating Systems

B. Sc. (Eng) Semester 3

A. L. Abeyweera

090007J

Programming Assignment 1 – Implementing a new command for JOSH

In programming assignment 1, I did the first part which is to implement a new command for operating system to print hardware information about the computer. Even though I had no prior knowledge about assembly language or operating system development, I started the project by following the instruction on the resource website [1].

I used a pen drive(128MB) to build the JOSH operating system. I had some problem with the unmounting process earlier. When I used the right click menu method to unmount the pen, errors were given for some commands saying no device found. Later I used the command line method (umount sdc) to unmount the pen drive. After following the rest of the tutorial I was able to complete building the bootable pen drive.

To get a better understanding of the JOSH operating system, I read the Tutorial written on JOSH by Dr. Mohan Raj Dhanagopal.[2] From the Tutorial I understood that I do not have to edit the boot.asm to implement a new command. Boot strap loader handles the loading of kernel to the memory. Therefore after the OS is loaded, the computer will be totally controlled by the kernel.

In the kernel-3.0.asm , they have implemented the interrupt 21 for printing a string. All we have to do to print a string is, setting string index to SI, set AX = 1 and call interrupt 21. As the initial step I changed the minor version to 4 by changing the strWelcomeMsg and strMinorVer strings.

Adding the new command

After we enter a command in the JOSH, it is read and the command is saved in the strCmd0 variable. We can compare the variable with the required command by using “repe cmpsb” command. Before that we have to set the CX register to the number of characters(with the null character). CX register is used to determine the number of characters that need to be compared. If comparison returns false, skip to the next command implementation. After the command implementation we need to jump to the _cmd_done label.

First I tested the command by printing a string on the screen. The command worked without any problem. To manage the code well, I needed to add a function to display the hardware information. I read the NASM documentation about the procedures.[3] From the documentation I found that we can define a procedures simply by adding the required code with “ret” under a label. When we need to invoke the procedure all we have to write is call “functionlabel”. We can pass the

required values for the function by storing them on variables or registers.

I started implementing my `_display_hardware_info` procedure by defining the label “`_display_hardware_info`” and adding `ret` with previously described string printing code.

Memory information

I found a wiki about detecting memory in the website osdev.org [4]. According to the source we cannot read the whole memory at once. It has lower memory part and the upper memory part stored separately in the computer. First I used the interrupt 12h to get number of base memory. After the interrupt 12 amount of base memory is stores in the ax register in Kb. Then I print the value using the `_print_dec` procedure.

I used a similar approach to find the memory between 1M and 16M. The only deference is interrupt is 15h and the function is `ax=e801h`.

```
mov ax, 0xE801
int 0x15
mov dx, ax
call _print_dec
```

To read the memory above 16M, I used the same interrupt but this time I used the value in DX register. Since DX register contains the number of 64k blocks, I divided the number by 16 to get the value in mega bytes.

```
mov ax, 0xE801
int 0x15
mov ax, dx
mov dx, 0
mov si, 16
div si
```

To show the total memory, I needed to get the sum of the three memory parts. First I stored them in variables. Then later I first added the 1st and 2nd parts of memory which are in kB. Then converted the result to MB by dividing it by 2¹⁰ (by shifting bits 10 times.) Then added the last memory part and printed the result in MB.

Procedure `_print_dec`

In assembly language there is no straight forward method to print a decimal number. Therefore I wrote a decimal printing procedure using some resources in the Internet. Argument to the function is passed through the DX register. Calculation is done by dividing the number by 10 and saving the remainder using “`push dx`”. This is done repetitively until the division results in zero. Each time I do a division , CX register is incremented, because it is later by the loop command to determine the number of iterations.

Decimal number is printed digit by digit by using the loop instruction. Before printing the

decimal digital 48 is added to get the ASCII value to represent the digit. Then using the BIOS teletype output, the number is printed.

CPU information

According to the wikipedia “cpuid” returns the CPU's manufacturer ID string (a twelve character ASCII string stored in EBX, EDX, ECX - in that order) when `eax = 0`. [5] . First I moved EBX and EDX to EAX and EBX. Then set EDX to null character. We now have the string in order. String is then saved to the `strVendorID` variable. Using the interrupt 21h string is printed on the screen.

When “cpuid” is called with `EAX=80000002h,80000003h,80000004h` , we get the entire 48-byte null-terminated ASCII processor brand string. After each call the string is put in the EAX, EBX, ECX, and EDX registers in order. Using the `save_string` function I saved the string in variable `strBrand`. Then using interrupt 21 the Brand string is printed.

Before printing the CPU brand function availability is check and an error is printed if the CPU is not supported.

```
mov eax, 0x80000000
cpuid
cmp eax, 0x80000004
jne _cpu_not_supported
mov eax, 0x80000002
mov si, strBrand
cpuid
call _save_string
add si, 16
mov eax, 0x80000003
cpuid
call _save_string
add si, 16
mov eax, 0x80000004
cpuid
call _save_string
add si, 16
mov si, 0x00
```

Hard disk number

Memory block 0x0040 to 0x0100 is called BIOS data area. BIOS uses this area to store some information/ status of the hardware in the computer. [2] We can read this area to get lot of information about the present hardware. But modifying data inappropriately may cause system failures. Therefore we must be carefully deal with the BIOS data area. In NASM when we need to refer to a address , first we have to set the base address in a segment register. Then using the required offset we can fetch or store in the location.

To get the Number of hard disk drives we have to refer to the offset 75h in 40h base address. The implementation is given below.

```
mov ax,0040h      ; look at 0040:0075 for a number
mov es,ax
mov dl,[es:0075h] ; move the number into DL register
```

In a similar manner we can print any data present in the BIOS data area. The following table shows the layout of the BIOS data area. [6]

In my modified kernel, I have used this method to view the number of serial ports and serial port 1's address if it exists. We can effectively use this method to get the information about existence of many hardware devices. List of details which can be shown from this method is mentioned in the below table.

<i>Offset</i>	<i>Size (bytes)</i>	<i>Description</i>
0x00	2	Base I/O address of serial port 1
0x02	2	Base I/O address of serial port 2
0x04	2	Base I/O address of serial port 3
0x06	2	Base I/O address of serial port 4
0x08	2	Base I/O address of parallel port 1
0x0A	2	Base I/O address of parallel port 2
0x0C	2	Base I/O address of parallel port 3
0x0E	2	Base I/O address of parallel port 4
0x10	2	Equipment word
0x12	1	Manufacturing test data
0x13	2	Memory size in Kb
0x15	2	Manufacturing test data
0x17	2	Keyboard status flag
0x19	1	Alt + Numpad data
0x1A	2	Keyboard buffer head
0x1C	2	Keyboard buffer tail
0x1E	32	Keyboard buffer
0x3E	11	Disk controller information
0x49	30	Graphics adapter information
0x67	5	Unknown
0x6C	4	Counter

0x70	1	Counter 24 hour overflow
0x71	1	Keyboard control flag
0x72	2	Soft reset flag
0x74	4	Disk controller information
0x78	4	Parallel timeout values
0x7C	4	Serial timeout values
0x80	2	Start of PS/2 keyboard buffer
0x82	2	End of PS/2 keyboard buffer
0x84	7	Graphics controller information
0x8B	13	Disk controller information
0x96	2	Keyboard status flag
0x98	4	Pointer to user flag
0x9C	4	User wait count
0xA0	1	User wait flag
0xA1	7	Reserved for network adapters
0xA8	4	Graphics adapter information
0xAC	68	Reserved
0xF0	16	Inter-application communication area

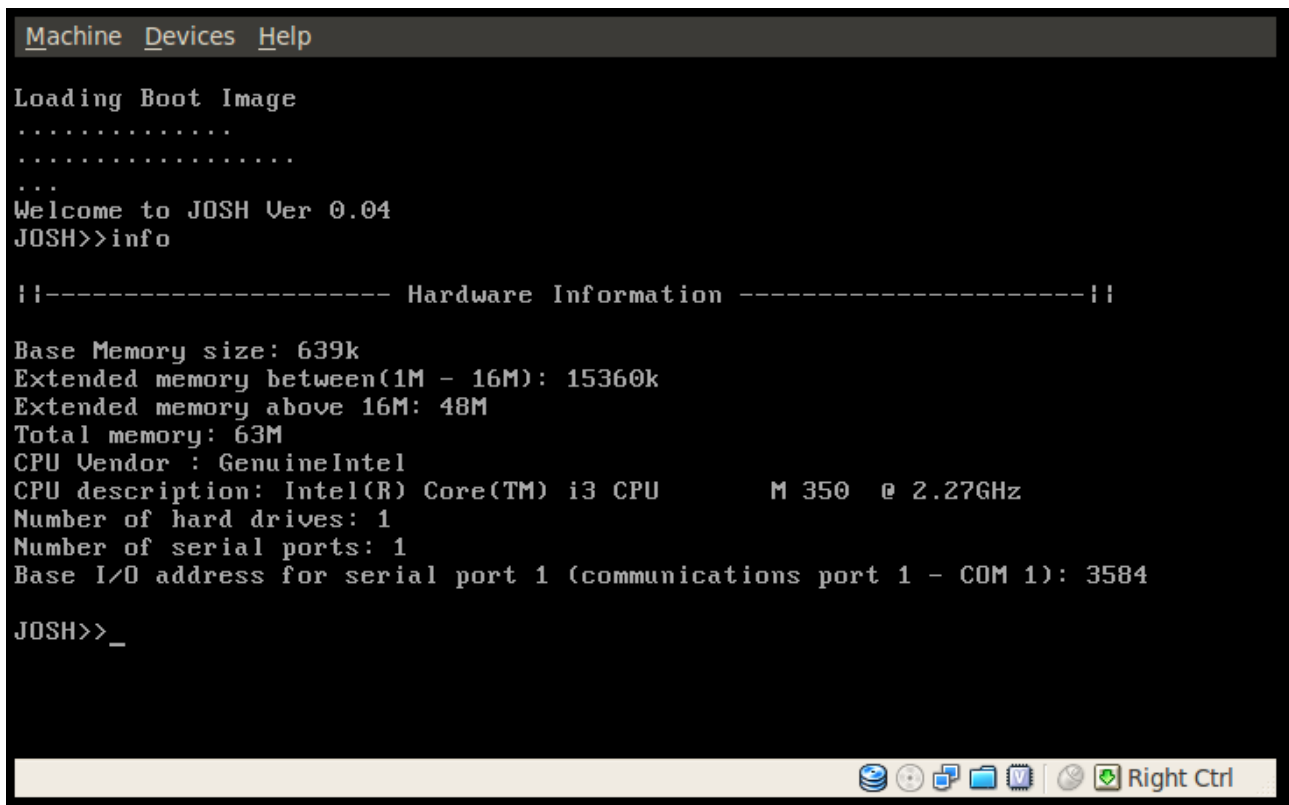
Final barrier

After I have finished all the above steps, I boot the pen drive again. The booting was successful, except that the info command works only one time. After the first execution, it was giving a message saying “Unknown command or bad file name!”. To solve the problem I added 6 push and pop commands at the start and end of the `_display_hardware_info` function to save and retrieve the register status.

In my implementation of the new JOSH command, I used mainly two methods to get the information. First one is **using interrupt calls**. After the interrupt call, the result will be stored in a register. According the format the data is represented, we may have to do some calculations to get the desired output.

Second method is by **reading the BIOS data area** we can get some hardware information. As previously described, some calculations will be needed to get the desired output.

Final output from the info command is shown below.

A screenshot of a VirtualBox window titled "Machine Devices Help". The window shows a terminal interface for the JOSH operating system. The text in the terminal is as follows:

```
Loading Boot Image
.....
...
Welcome to JOSH Ver 0.04
JOSH>>info

!!----- Hardware Information -----!!

Base Memory size: 639k
Extended memory between(1M - 16M): 15360k
Extended memory above 16M: 48M
Total memory: 63M
CPU Vendor : GenuineIntel
CPU description: Intel(R) Core(TM) i3 CPU          M 350  @ 2.27GHz
Number of hard drives: 1
Number of serial ports: 1
Base I/O address for serial port 1 (communications port 1 - COM 1): 3584

JOSH>>_
```

At the bottom of the window, there is a taskbar with several icons and the text "Right Ctrl".

Output generated from the JOSH command in VirtualBox

Reference

- [1].Asiri Rathnayake. "Hacking JOSH - Operating System Tutorial" Internet: <http://asiri.rathnayake.org/articles/hacking-josh-operating-system-tutorial/>, [Oct. 2, 2010].
- [2]. Dr. Mohan Raj Dhanagopal. "Operating System design & implementation Tutorial, JOSH" Internet: <http://www.mohanraj.info/josh.jsp>, [Oct. 2, 2010].
- [3]. "NASM Manual" Internet:<http://www.nasm.us/doc/>, [Oct. 9, 2010].
- [4]. "Detecting Memory (x86) " Internet: http://wiki.osdev.org/Detecting_Memory_%28x86%29, [Oct. 16, 2010].
- [5]. "CPUID" Internet: <http://en.wikipedia.org/wiki/CPUID> , [Oct. 30, 2010].
- [6.] "BiosCentral - BIOS Data Area" Internet: <http://www.bioscentral.com/misc/bda.htm>, [Nov 6, 2010].
- [7.] "BIOS Data Area" Internet: <http://alien.dowling.edu/~rohit/rg010.html>, [Nov 6, 2010].