Index No: 090248X

CS 2042- Operating Systems

Assignment 1: New Command for Hacking JOSH Operating System

Objective

Implement a new command for your operating system which prints hardware information about the computer.

Author

My name is *H.M.G.C.KARUNARATHNE* and my index number is 090248X. I'm a student of University of Moratuwa.

@. Name : H.M.G.C.Karunarathne

@ Index number : 090248X

Introduction

JOSH is an Operating System which can boot from a FAT12 disk (from a floppy disk). In this report, it includes that how am I implement a new command to the operating system to show Hardware Information.

These are the Information given by the Implemented command.
• Processor Family
• Processor Model Identifier
• Processor Stepping
• Processor Vendor Identifier
• Processor Model
• RAM size
• L2 Cache Size
• Number of Floppy Drives
• Number of Hard Disk Drives
• Number of Serial Ports
• Number of Parallel Ports
• Availability of Mouse
There were few available options to get the hardware information of the system. They were,
• CPUID command
• Calling BIOS Interrupts
I used two of them to get the above mentioned information of the system.

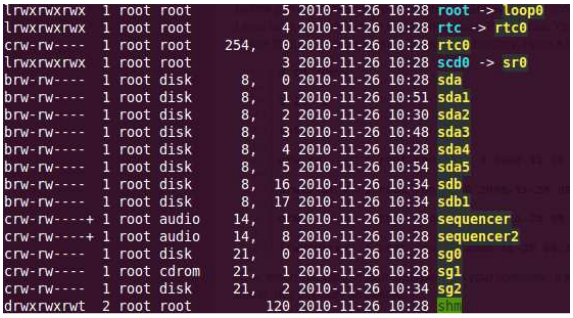
Preparing and making environment for create Operating system

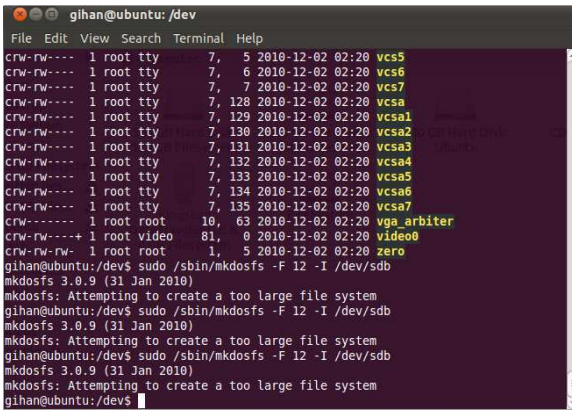
First I read the whole blog that was given as a source for this project.
[http://asiri.rathnayake.org/articles/hacking-josh-operating-system-tutorial/]

Then I followed the procedure according to this tutorial. When I start this project I had to face to some difficulties when preparing the environment. Those issues are mention as below and also how am I overcome those;

Difficulties of preparing environment

- In my computer the pen drive was shown as 'sdb' and it's differing from tutorial.
  Some time it didn't show the 'sdb1' that should show when flash drive was mounted.
  So, the command for format the pens drive doesn't work at that moment. Thus I used a technique to unmount the pen drive from a command and next try, it worked.





- Other problem that I faced in this scenario is I couldn't able connect to the network within Ubuntu. Because some driver are not found in Ubuntu. So I tend use the Windows 7 professional to make it. I used **nasm-2.07-installer** make the environment of nasm and use **Oracle VM Virtual Box** built a virtual computer to run the JOSH operating system.

Boot Loader

Boot-strap loader the system memory map will look as follows:

1. Memory block 0x0000 to 0x0040 (1-KB) -a list of addresses called the Interrupt Service Routine Vectors

2. Memory block 0x0040 to 0x0100 - BIOS data area and it is in this area where the BIOS saves data about the memory available, devices connected etc.

3. Memory block 0x0100 to 0x07C0 is free and is available for the OS and applications.

4. Memory block 0x07C0 to 0x07E0 - boot-strap loader is loaded by the BIOS.

5. Memory block 0x7CE0 to 0xA000 is free and is available for the OS and applications.

6. Memory block 0xA000 to 0xC000 - video sub-system.

7. Memory block 0xC000 to 0xF000 - ROM BIOS

8. Memory block 0xF000 to 0xFFFF - base ROM system ROM (the top of which is where the first instruction is present on a re-boot).
Provided web-site: http://www.mohanraj.info/boot.asm

Source of Previous Kernel 0.03.asm

I used the implemented kernel source code from following web side. It has only two instructions that are support by the kernel. Those are

I. "ver" to show the version of the Operating system
II. "exit" to reboot the operating system

Provided web-site: http://www.mohanraj.info/kernel0.03.asm

Outcome Objectives after Finish

After reading and understanding the tutorials of Dr: Mohan Raj and much of web-site, I gain more knowledge about to dealing with assembly language such as NASM. I manage to understand the execution behavior of the Josh operating system. It has a shell which can get string arguments and call for particular functions. After that my target was to build a function which can show hardware information of a system. Using x86 Assembly language we can use BIOS interrupt calls, CPUID instruction, BIOS data area, SMBIOS –

Collection of tables which can provides you hardware information. In these methods I choose interrupt calls, CPUID instruction to display hardware information of the system. In my function it can show Processor Brand, Processor Type, Ram Size, system date and time, check the availability of serial ports, floppy drives, printers, co-processors and mouse. In this project, I try to implement a new command that will print the hardware details of the system [for more details look at "read.txt" file].

Procedure of Implement of Kernel GC 0.05

The kernel is the core of the OS performing most of the house-keeping work and providing the applications with a good set of functionality. In JOSH OS is provides all the functionality to applications through interrupts. Then I further study about how to handle those functionalities by the sake of interrupts (*the resources that I used to gain knowledge are mentioned in "References" below*). BIOS interrupts are very fast and also we can use them for access the shell to print details. Further we can also use routines to display text/graphics using the VGA RAM.

[from this site I enable to gain more idea about interrupt calling
http://www.delorie.com/djgpp/doc/ug/compiling/port16.html]

In NASM compiler, it uses three segments,

1. '.text' for code,
2. '.data' for initialized data (such as constants that never change),
3. '.bss' for un-initialized data

Get CPU details

[Accessed : http://en.wikipedia.org/wiki/CPUID]

CPUID implicitly uses the eax register. The eax register should be loaded with a value specifying what information to return. CPUID should be called with EAX = 0 ,This returns the CPU's manufacturer ID string - a twelve character ASCII string stored in ebx, edx, ecx - in that order. The highest basic calling parameter is returned in eax.

Processor manufacturer ID strings:

"AMDisbetter!" - Early engineering samples of AMD K5 processor
"AuthenticAMD" - AMD
"CentaurHauls" - Centaur
"CyrixInstead" - Cyrix
"GenuineIntel" - Intel
"TransmetaCPU" - Transmeta
"GenuineTMx86" - Transmeta
"Geode by NSC" - National Semiconductor
"NexGenDriven" - NexGen
"RiseRiseRise" - Rise
"SiS SiS SiS " - SiS
"UMC UMC UMC " - UMC
"VIA VIA VIA " – VIA

CPUID should be called with setting up eax register to eax=80000002h, 80000003h, 80000004h respectively. These return the processor brand string in eax, ebx,ecx and edx. CPUID must be issued with each parameter in sequence to get the entire 48-byte null-terminated ASCII processor brand string.

BIOS Interrupts

[Accessed : http://en.wikipedia.org/wiki/BIOS_interrupt_call ; http://www.bioscentral.com/misc/bda.htm]

The BIOS Interrupts are invoked as int 0xXX where XX is a hexadecimal number. The values returned by the interrupts are sometimes differs according to the input for some interrupts. To get the RAM size of the system, the interrupt call "int 0x15" is used with the input 0xE801 at the ax register (AH – 0xE8, AL – 0x01). This interrupt returns RAM size in 64 kb Blocks at DX register. Actually the value is 16 MB less than the actual RAM size

because of compatibility reasons. Therefore 16 MB should be added to the value returned by the interrupt call. To get the mouse information, the interrupt call “int 0x33” is used. The value of ax is 0 if mouse or a pointing device is absent. Thereby the availability of a pointing device can be determined. To get the number of Hard Disks, the interrupt call “int 0x13” is used with parameters AH=0x08 and DL=0x80. It returns the No. of Hard Disks at DL register. (No. of Floppy drives could also be gained by calling with arguments AH=0x08 and DL=0x00. But after using like that there was a problem at the shell.) One other useful interrupt call is “int 0x11” which return many information about Floppy Drives, Serial ports, Parallel ports, etc. The AX register is filled with the following information.

0 floppy disk(s) installed

1 80x87 coprocessor installed

3-2 number of 16K banks of RAM on motherboard

5-4 initial video mode.

00 EGA, VGA, or PGA.

01 40x25 color.

10 80x25 color.

11 80x25 monochrome

7-6 number of floppies installed less 1 (if bit 0 set)

8 DMA support installed

11-9 number of serial ports installed

12 game port installed

13 serial printer attached (PCjr).

15-14 number of parallel ports installed.

Using the above list number of floppy drives, serial ports, parallel ports can be displayed (Using the bit operations).

Total memory available

Using interrupt 15 with ax=0xe801;

CF clear if successful
AX = extended memory between 1M and 16M, in K (max 3C00h = 15MB)
BX = extended memory above 16M, in 64K blocks
CX = configured memory 1M to 16M, in K
DX = configured memory above 16M, in 64K blocks
CF set on error

To show total extended memory ax, bx register values should be added. These register values (ax in kB and bx in 64kB blocks). are converted to MB and added to convert ax(in kB), bits in that register are shifted left by 10 bits and bx(in 64kB), bits in that register are shifted left by 4 bits.
ax only shows memory between 1MB and 16 MB, therefore by calling this interrupt, it shows 1MB less than actual memory available. Therefore Interrupt 12 has been used for show the base memory that was previously hidden from the interrupt 15. by just calling the interrupt 12 it will return the value into ax register

Number of serial ports available

When power is applied to the computer, the BIOS Data Area is created at memory location 0040:0000h with a typical size of 255 bytes. In offset 10h give the equipments available in the system
By shifting 9 bits left and using "and" operation with e00 hexadecimal value, function can filter-out the serial ports available in the system

Number of parallel ports available

In number 5 stated that in offset "10h" bit 15 and 14 gives parallel ports available in the system
By shifting 14 bits left and using "and" operation with ffffc000 hexadecimal value, function can filter-out the parallel ports.

SMBIOS – System Management BIOS

[Accessed : http://en.wikipedia.org/wiki/System_Management_BIOS ; http://www.dmtf.org/standards/smbios]

The SMBIOS provides numerous tables of data describing a computer's configuration. Available information includes items such as vendor name, BIOS version, installed components, CPU clock speed, etc. The SMBIOS Specification addresses how motherboard and system vendors present management information about their products in a standard format by extending the BIOS interface on x86 architecture systems. The information is intended to allow generic instrumentation to deliver this information to management applications that use DMI, CIM or direct access, eliminating the need for error prone operations like probing system hardware for presence detection.

Bit Operations Used

The output given by the above commands couldn't be used directly at most instances. It had to be manipulated to a useful format. These are some of the techniques I used to format the output.

Clearing a Register (Make the value 0)

In General, the MOV is used to change the value of a register. But to set a register value to zero the XOR operation of the register with itself is used since it's said to be efficient and faster than MOV.

E.g. To clear AX register XOR AX, AX

Extracting a bit/bits at a specific location of a register

Some operations give multiple output at the same register at separate sets of bits. Those bits needs to separated to get the output.

E.g. cupid with EAX – 0x00000001 gives 3 values at the same EAX register as follows

8..11 bits : Processor Family Number
4..7 bits : Processor Model Number
0..3 bits : Processor Stepping Number

To extract them the AND, SHR operations are used where AND does the bitwise AND operation of two values and SHR shifts the bit pattern right by a given quantity. To extract Family number these operations can be used as follows.

Output : yyyy yyyy yyyy yyyy yyyy zzzz yyyy yyyy
where z represent bits of Family ID and y represent other bits

AND EAX, 0x00000F00 gives
0000 0000 0000 0000 0000 zzzz 0000 0000
SHR AX, 8 gives
0000 0000 0000 zzzz

Note that AX is the Lower half of the EAX.
Likewise we can extract a specific bit or bits.

Getting Original size using the number of blocks

Some Outputs are given as number of blocks. For Example the RAM size is given in 64 kB blocks. So the output should be multiplied by 64. Usually a RAM size is expressed in MB not kB. Therefore the value should be divided by 1024 again. Those 2 can be combined as division by 16. The
multiplication and division by the numbers of 2's power can be achieved efficiently by shifting the bits. Therefore the final operation is to shift the register 4 bit right.

References

[1] Asiri's Laborotary - http://asiri.rathnayake.org/articles/hacking-josh-operatingsystem- tutorial/

[2] Dr. Mohan Raj's website - http://www.mohanraj.info/josh.jsp Accessed 30th October 2010

[3] NASM Manual (Online) http://www.nasm.us/doc/ Accessed 24th November 2010

[4] Details about cpuid assembly instruction http://www.gamedev.net/reference/articles/article1207.asp

[5] Wikipedia – BIOS Interrupts http://en.wikipedia.org/wiki/BIOS_interrupt_call Accessed 26th November 2010

[6] Wikipedia – CPUID http://en.wikipedia.org/wiki/CPUID Accessed 25th November 2010

[7] NASM Manual posix.nl – Accessed 27th November 2010

http://www.posix.nl/linuxassembly/nasmdochtml/nasmdoca.html

[8] CPUID in Assembly - http://www.richardcunningham.co.uk/labs/cpuid.php Accessed 27th November 2010.

[9] Sand Pile website http://www.sandpile.org/ia32/cpuid.htm Accessed 26th November 2010