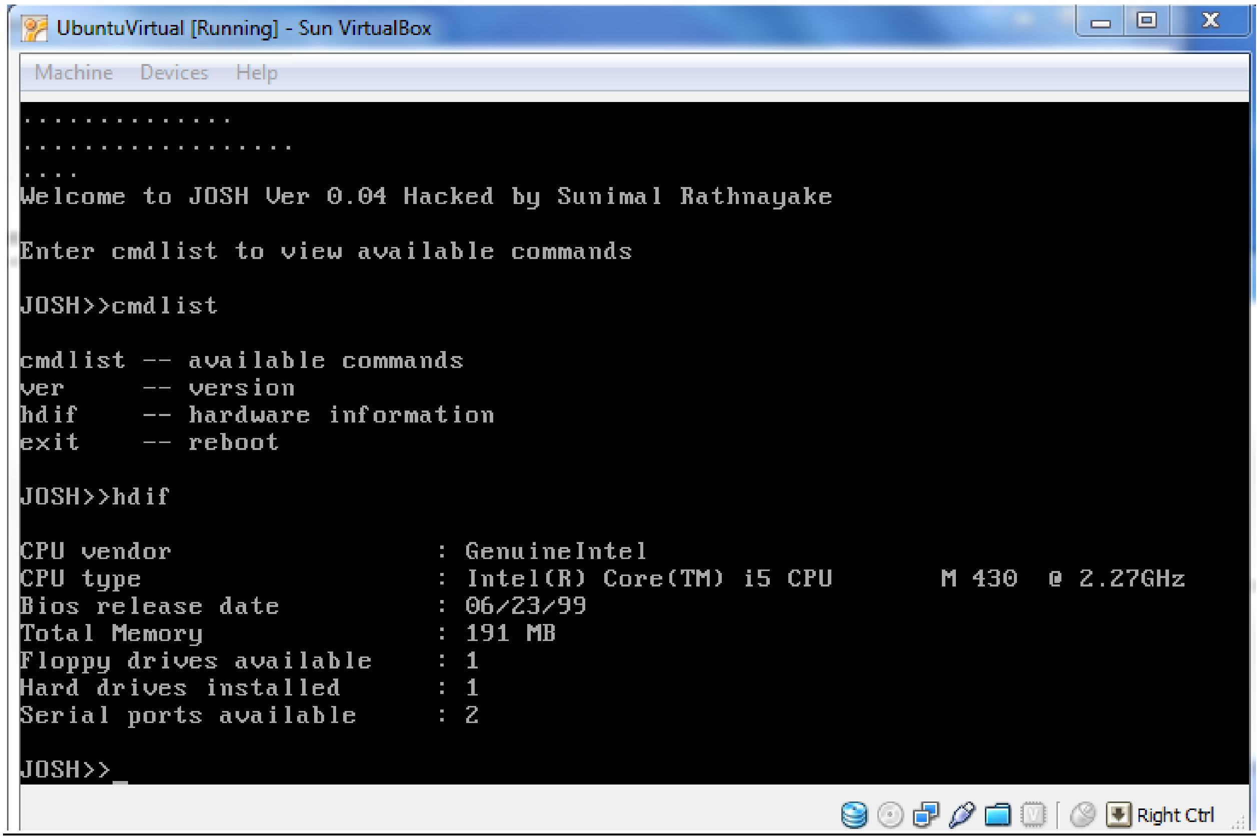
Operating Systems - CS2042

Programming Assignment 1 – Hacking JOSH OS

Implementing a command to show hardware information

R.M.S. Rathnayake – 090436X

Snapshot of completed OS



```
..............
..................
....
Welcome to JOSH Ver 0.04 Hacked by Sunimal Rathnayake

Enter cmdlist to view available commands

JOSH>>cmdlist

cmdlist -- available commands
ver     -- version
hdif    -- hardware information
exit    -- reboot

JOSH>>hdif

CPU vendor                   : GenuineIntel
CPU type                     : Intel(R) Core(TM) i5 CPU       M 430  @ 2.27GHz
Bios release date            : 06/23/99
Total Memory                 : 191 MB
Floppy drives available      : 1
Hard drives installed        : 1
Serial ports available       : 2

JOSH>>_
```

Figure : JOSH snapshot

How I achieved

JOSH operating system is designed to run on FAT12 formatted disk. Therefore First I formatted my USB flash drive Therefore first I converted my USB flash drive to FAT12 format by creating a floppy image and overriding the USB disk by that floppy image. This was done according to the steps given in the tutorial at http://asiri.rathnayake.org/articles/hacking-josh-operating-system-tutorial/ .

Since it is hard to reboot the computer every time I need to execute the code, I installed a virtual machine software (Sun VirtualBox).

Functions Implemented

1) Display the number of Serial Ports

```
_display_serial_ports:
     call _display_endl

     mov si, strSerial
          mov al, 0x01
          int 0x21

     xor ax, ax
     int 0x11
     and ax, 0xe00
     shr ax, 9
     add ax, 48
     mov ah, 0x0e
     int 0x10
     ret
```

Note: Interrupt 0x21 with value 0x0e in register ah displays the contents in register si on the teletype output. Inturrupt 0x11 returns the equipment list data stored in in bios data area. [1][2]

strSerial is a string constant that is used to store a string to be displayed.
Xor ax,ax clears all the bits in the ax register. When interrupt 0x11 is the bios equipment list flags are stored in ax register.
And ax, 0xe00 is used to mask out bits 9,10 and 11 which are used to indicate the number of serial ports available in the computer.
Then ax is right shifted by 9 bit positions so that only required bits are remaining in the ax register.
48 is added to ax to get the ASCII value of the value stored in ax.
Then it is displayed.

2) Display the number of floppy drives

```
_diskette:
     call _display_endl
     mov si, strDiskette
     mov al, 0x01
        int 0x21
     xor ax, ax
     int 0x11
     and ax, 0x80
     shr ax, 6
     add ax, 49
     mov ah, 0x0e
     int 0x10
     ret
```

This is almost similar to the above mentioned function except that in this occasion, 8th bit of ax register is masked out.[2]

3) Number of hard drives installed

```
_hard_drive:
      call _display_endl

           mov si, strHardDrive
           mov al, 0x01
           int 0x21

           mov ax, 0x0040
           push es
           mov es,ax
           mov al,[es:0x0075]
           add al, 48
           pop es
           mov ah, 0x0e
           int 0x10
           ret
```

Number of hard drives installed in the system can be extracted from memory locations 0x0040 to 0x0075. Interrupt 0x10 with 0x0e in the ah register prints the text in al.[3]

Note :There might be values in registers we use for our funtions. They have to be restored after we release that register after out function is executed. This is done by pushing the register values to stack before we use it and popping them out of the stack after we use the register.

4) Bios date

```
_bios_date:
           call _display_endl
           mov si, strBios
           mov al, 0x01
           int 0x21
           push es
           mov ax, 0xf000

           mov es, ax
           mov si, 0xfff5
           mov bl,8
           _loop:
                mov al, [es:si]
                mov ah, 0x0e
                int 0x10
                inc si
                dec bl
                cmp bl, 0
                jne _loop
           pop es
           ret
```

Bios release date can be extracted from the memory locations 0xf000 to 0xfff5. Since there are 8 characters of date date (DD/MM/YY) I used a decrementing register bl in order to take contol out of the loop when the date string is printed.

Note: cmp is the assembly command that is used to compare two values. jne is a command that follows a cmp command which jumps to a given location if compared values are not equal.

5) CPU vendor and CPU model

```
_cpuid:
     call _display_endl
     mov si, strVendor
          mov al, 0x01
          int 0x21
     push eax
     push ebx
     push ecx
     push edx

     xor eax, eax
     mov eax, 0x00
     cpuid
     mov [strCPUID], ebx
     mov [strCPUID+4], edx
     mov [strCPUID+8], ecx

     pop edx
     pop ecx
     pop ebx
     pop eax
     mov si, strCPUID
     call _disp_str
     call _display_endl
     mov si, strCpuType
          mov al, 0x01
          int 0x21
     mov eax, 0x80000002
     mov si, strBrand
     cpuid
     call _string_store
     mov eax, 0x80000003
     cpuid
     call _string_store
     mov eax, 0x80000004
     cpuid
     call _string_store
     add si,16
     mov si,0x00


     mov si, strBrand
     mov al, 0x01
     int 0x21

     ret
```

The CPUID opcode is a processor supplementary instruction for the x86 architecture that gives information and features about the CPU. By using the CPUID opcode, software can determine processor type and the presence of features. The value in the EAX register specifies what information to return when CPUID is executed.[4]

With 0x00 in the EAX register, CPUID stores the vendor name of the CPU in registers EBX, EDX and ECX. Then the values of those three registers must be stored as a single string and printed in order to print the vendor name.[4]

I have also included the code to obtain the processor model string in the above function. It was takn by three steps, calling cpuid thrice while changing the value of eax to 0x80000002, 0x80000003 and 0x80000004 respectively. Each time the value returned was stored by calling _string_store function which will be explained later in this document.

6) <u>Memory detect</u>

```
_mem_detect:
     call _display_endl

     xor ax,ax
     xor bx,bx
     xor cx,cx
     xor dx,dx

     mov ax, 0xe801
     int 0x15
     jc _error
     cmp ah, 0x86
     je _error
     cmp ah, 0x80
           je _error
    mov si, strMemory
     mov al, 0x01
     int 0x21
     cmp cx, 0x0000
     je _cx_zero
     jmp _mem_calc

_cx_zero:
     mov cx,ax
     mov dx,bx
```

```
_mem_calc:

     shr dx, 4
     shr cx, 10
     add cx,dx
     mov dx, cx
     call _hex2dec
     mov si, strMB
     mov al, 0x01
     int 0x21


     jmp
_memory_detected

_error:
     mov si, strMemError
     mov al, 0x01
     int 0x21

_memory_detected:
     ret
```

Memory can be detected by using the INT 0x15, EAX = 0xE801 command. This stores the extended memory between 1MB and 16MB in kilobytes in AX or BX and extended memory >16MB as no of pages of 64KB in CX or DX. [5]

The purpose of comparing ah with 0x86 is to check whether the function is supported on the system and comparing ah with 0x80 is to check whether it is an invalid command.

Some systems, when called the interrupt 0x15, returns that CX=DX=0. If so we have to copy AX and BX to CX and DX. It is don't by _cx_zero.

Detected memory has to be calculated MB and has to be converted to decimal from hexadecimal. This is done by _mem_calc. Function _hex2dec will be explained later. cx is divide by 2^10 to convert the value to MB and dx is divided by 2^4 convert it to MB because dx has the number of 64K pages.

Additional functions

- String store

```
_string_store:
      mov dword [si], eax
      mov dword [si+4], ebx
      mov dword [si+8], ecx
      mov dword [si+12], edx
      add si, 16
      ret
```

This function stores value in eax, ebx, ecx and edx as a single string in register si.

- Hex to dec

```
_hex2dec:

      push ax
      push bx
      push cx
      push si
      mov ax,dx
      mov si,10
      xor cx,cx
```

This function converts the hex stored in dx to decimal and prints it.

References

[1] INT 21 - DOS Function Dispatcher, http://stanislavs.org/helppc/int_21.html
[2] INT 11 - BIOS Equipment Determination / BIOS Equipment Flags,
http://stanislavs.org/helppc/int_11.html
[3] BDA - BIOS Data Area - PC Memory Map, http://stanislavs.org/helppc/bios_data_area.html
[5] CPUID, http://en.wikipedia.org/wiki/CPUID#EAX.3D0:_Get_vendor_ID
[4] Memory Map (x86), http://wiki.osdev.org/Memory_Map_%28x86%29

Other references

- http://www.bioscentral.com/misc/bda.htm
- http://www.on-time.com/rtos-32-docs/rttarget-32/reference-manual/smbios/
- http://www.dmtf.org/sites/default/files/standards/documents/DSP0134_2.7.0.pdf
- http://cyberasylum.wordpress.com/2010/11/19/assembly-tips-and-tricks/
- http://obahamema.blogspot.com/
- http://www.ctyme.com/hosting/index.htm
- http://datasheets.chipdb.org/Intel/x86/CPUID/24161821.pdf