Assignment – Operating Systems

Implementing a new commands for JOSH operating system which prints hardware information about computer

Introduction to JOSH

JOSH operating system is a basic implementation of an operating system which runs from FAT12 media (e.g. floppy disk). It is an interrupt driven, single task operating system. Therefore JOSH is a good point to meet with both practical and theoretical aspects of implementing an operating system.

Here the focus is on implementing few commands into kernel of JOSH operating system in order to get hardware information of the computer.

The following information will reveal how I achieved the target

I downloaded the boot-strap loader and the kernel version 3.0 of JOSH operating system. The boot-strap loader supports to boot into the kernel using a FAT12 media. But here I used a USB pen drive which supports FAT32 file system and overrode it with a FAT12 floppy disk image in order to use it with the boot-strap loader. The kernel has been implemented using assembly language and the basic functionalities of the JOSH operating system were there. The following interrupt calls have been used in the course of implementation of the JOSH kernel.

1. 0x10 - BIOS interrupt call uses for invoke services like set video mode, clear/scroll screen, write character at cursor, write string and various video services.

   The int 0x10 call is used with 0x0e entry on AH registry in the JOSH kernel in order to write character in teletype (TTY) mode (writes one character at the current cursor location and advances the curser). Also few things about this interrupt call can be specified with the contents in few other registries. (AL - Character to write, BL - Foreground color, BH - Display page number)

2. 0x16 - this invokes the application-level interface to the keyboard. Keystrokes are processed asynchronously (in the background).

   The int 0x16 call is used with 0x10 entry on AH registry in the JOSH kernel in order to read extended keyboard input

3. 0x19 - supposed to restart the BIOS bootstrap loader

4. 0x21 - invokes various DOS services including keyboard input, video output, disk file access, executing programs, memory allocation and various other things. (Dispatcher for DOS services)

   The int 0x21 call is used with 0x01 entry on AH registry in the JOSH kernel in order to read character from STDIN (Standard Input which is data going into a program)

Functions to print hardware information were implemented in following manner

To retrieve hardware information, BIOS entries were used and few other interrupt calls were helpful. The followings will explain how these interrupt calls were used.

1. 0x11 - returns data stored at BIOS Data Area location 40:10 which are the information about installed hardware.

   On return following flags are contained in AX registry,

```
|F|E|D|C|B|A|9|8|7|6|5|4|3|2|1|0|  AX
 | | | | | | | | | | | | | | | `---- IPL diskette installed
 | | | | | | | | | | | | | | `----- math coprocessor
 | | | | | | | | | | | | | `-------- old PC system board RAM < 256K
 | | | | | | | | | | | | | `----- pointing device installed (PS/2)
 | | | | | | | | | | | | `------ not used on PS/2
 | | | | | | | | | | | `--------- initial video mode
 | | | | | | | | `------------ # of diskette drives, less 1
 | | | | | | | `------------- 0 if DMA installed
 | | | | `------------------ number of serial ports
 | | | `------------------- game adapter installed
 | | `--------------------- unused, internal modem (PS/2)
 `----------------------- number of printer ports
```

2. 0x15 – it returns extended memory size while 0xe801 is in registry AX. Gives results for memory size above 64 MB. Returning values are stored in AX or BX and CX or DX, where amount of configured memory 1MB to 16MB in AX or CX in KBs and amount of configured memory above 16MB in 64KB blocks in BX or DX.

   To grab details about CPU the cpuid opcode is used. The CPUID instruction takes no parameters as CPUID implicitly uses the EAX register. The EAX register should be loaded with a value specifying what information to return. The values need to be specified as follows to get certain return values.

   EAX=0: Get vendor ID : 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 (largest value that EAX can be set to before calling CPUID) is returned in EAX.

   EAX=1: Processor Info and Feature Bits

EAX=2: Cache and TLB Descriptor information

EAX=3: Processor Serial Number

EAX=80000000h: Get Highest Extended Function Supported

EAX=80000001h: Extended Processor Info and Feature Bits

EAX=80000002h, 80000003h, 80000004h: Processor Brand String: 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.

EAX=80000005h: L1 Cache and TLB Identifiers

EAX=80000006h: Extended L2 Cache Features

EAX=80000007h: Advanced Power Management Information

EAX=80000008h: Virtual and Physical address Sizes

Function to grab CPU information:

```
_cpu:
        ……………
        mov si, strCPUInfo
        xor   eax,eax
        mov eax, 0x80000002
        cpuid
        call _capture
        mov eax, 0x80000003
        cpuid
        call _capture
        mov eax, 0x80000004
        cpuid
        call _capture
        add si,16
        mov si,0x00
        ……
```

"cpuid" opcode is used here with specifying values 80000002h, 80000003h, 80000004h in registry EAX in three steps and stored 16 byte portion of processor brand details which were loaded into registries EAX, EBX, ECX and EDX using "_capture" command in each step.

Function to grab memory information:

_memory:

```
……………
mov ax, 0xe801
int 0x15
jc _undetect
cmp ah, 0x86
je _undetect
cmp ah, 0x80
je _undetect
```

Interrupt call 0x15 is made while value 0xe801 is in registry AX to get memory size which is greater than 64MB. If it causes to set carry flag which is the case in an error then it jumps into "_undetect" function. Else if it causes to set the value of registry AH equals to 0x86 or 0x80 which is the case in an unsupported function or an invalid command it jumps into "_undetect" function. "_undetect" prints an error to display.

```
…….
cmp cx, 0x0000
je _copy
jmp _get_total
……
```

If value of CX has been set to 0 by the BIOS it means memory size information are contained in AX and BX. So values in AX and BX need to be transferred into CX and DX to simplicity of code by function "_copy". Then total memory amount is needed to be calculated by jumping into the function "_get_total".

```
……
_get_total:
shr cx, 10
shr dx, 4
add cx,dx
mov dx, cx
call _hex2dec
…….
```

Memory size 1MB to 16MB is configured in CX in KBs. So to get the size of that memory in MBs following calculation needs to be followed. (CX /1024 = DX * 2^{-10}) This is done by shifting right 10 bits from the content in CX.

Memory size above 16MB is configured in DX as the number of 64KB memory blocks. So to get the size of that memory in MBs the following calculation needs to be followed. (DX * 64/1024 = DX * 2^{-4}) This is done by shifting right 4 bits from the content in DX.

Then both CX and DX are added to get the total size of the memory. Then then function "_hex2dec" is called to print the total memory size in decimal form into the display.

Function to grab number of floppy disk drives:

```
_floppy:
        int 0x11
        and ax, 1
        cmp ax, 1
        je _floppy_present
        mov ah, 0x0e
        mov al, '0'
        int 0x10
        ……………
        _floppy_present:
        xor ax, ax
        int 0x11
        and ax, 0xc0
        shr ax, 6
        add ax, 48
        ……
```

Interrupt call 0x11 is made. AND operation to extract value of 0^{th} bit and then check whether it is 1 or 0. If it is 1 floppy drive is installed and then directs to command "_floppy_present" to count number of drives. If it is 0 no floppy drive is installed and then prints 0 to the display as floppy drive count.

Interrupt call 0x11 is made. AND operation to extract values of 6^{th}, 7^{th} bits (0xc0 =11000000) and shifting right by 6 bits to isolate the value of 6^{th}, 7^{th} bits. Then addition of 48 (pen drive which is used to boot the OS itself is not considered as a floppy drive) to get ASCII value into AX registry.

Function to grab number of serial ports:

```
_serial_ports:
        ……………
        xor ax, ax
        int 0x11
        and ax, 0xe00
        shr ax, 9
        add ax, 48
        ……
```

Interrupt call 0x11 is made. AND operation to extract values of 9^{th}, 10^{th}, 11^{th} bits (0xe00 = 111000000000) and shifting right by 9 bits to isolate the value of 9^{th}, 10^{th}, 11^{th} bits. Then addition of 48 to get ASCII value into AX registry.

Function to grab number of parallel ports:

```
_parallel_ports:
    ……
    xor ax, ax
    int 0x11
    and ax, 0xc000
    shr ax, 14
    add ax, 48
    ……
```

Interrupt call 0x11 is made. AND operation to extract values of 14^{th}, 15^{th} bits (0xe00 = 111000000000) and shifting right by 14 bits to isolate the value of 9^{th}, 10^{th}, 11^{th} bits. Then addition of 48 to get ASCII value into AX registry.

Function to grab number of hard drives:

```
_harddisk:
    ……
    mov ax, 0x0040
    push es
    mov es,ax
    mov al,[es:0x0075]
    add al, 48
    pop es
    ……
```

Prevailed value in ES registry is pushed into the top of the stack to use in future operations if require. Then segment registry ES is used to read data at location 40:75 (offset is 0x75) which contain information about number of installed hard drives loaded into AL. Then 48 is added to value in AL in order to get ASCII value. After the operation ES is loaded back with values which contained in it at the beginning of the operation by performing a pop operation.

Function to check keyboard type:

```
_keyboard:
        ……………
        mov ax, 0x0040
        push es
        mov es,ax
        mov bx,[es:0x0096]
        and bx,0x10
        jnz _keyboard_Present
        mov si, strNKPresent
        mov al, 0x01
        int 0x21
        pop es
        ……
```

Prevailed value in ES registry is pushed into the top of the stack to use in future operations if require. Then segment registry ES is used to read data at location 40:96 (offset is 0x96) which contain information about number of installed hard drives loaded into BX. Then AND operation is carried on the value in BX to determine whether the 4^{th} bit (0x10 = 10000) is 1 or 0. If it is 1 non-zero flag is set to 1 and it is jumped into command "_keyboard_Present" which prints "Present" into the display. Otherwise "Not present" is printed into the display. After the operation ES is loaded back with values which contained in it at the beginning of the operation by performing a pop operation.

je <command_A>	- Jump to command_A if previous comparison (cmp) equals
jc <command_A>	- Jump to command_A if carry flag is set
xor <register_A>, <register_A>	- Clear register_A

References

Help on JOSH OS	: http://www.cs.bgu.ac.il/~yagel/dolev/MinimalOS/josh/index.htm
	http://asiri.rathnayake.org/articles/hacking-josh-operating-system-tutorial/
Grab hardware information:	http://stanislavs.org/helppc/bios_data_area.html
	http://en.wikipedia.org/wiki/CPUID
	http://www-ivs.cs.uni-magdeburg.de/~zbrog/asm/memory.html
	http://en.wikibooks.org/wiki/X86_Assembly/X86_Architecture
	http://arhiva.elitesecurity.org/t166386-Jedan-primjer-simle-OS
Assembly language help	: http://en.wikibooks.org/wiki/X86_Assembly/X86_Architecture
	http://en.wikipedia.org/wiki/X86_assembly_language
	http://www.ousob.com/ng/masm/ngf8a8.php
Information about registries	: http://www.eecg.toronto.edu/~amza/www.mindsec.com/files/x86regs.html
	http://en.wikipedia.org/wiki/X86_memory_segmentation
Help on interrupt calls	: http://www.datadoctor.biz/data_recovery_programming_book_chapter6-page21.html
	http://en.wikipedia.org/wiki/INT_21H
	http://www.syntax-example.com/Code/show-system-configuration-interrupt-11h-749.aspx
	http://members.tripod.com/~Vitaly_Filatov/ng/asm/asm_023.15.html

Name : Jayaweera W.J.A.I.U.

Index NO : 100227D