INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

COMPUTER SYSTEMS LABORATORY

Fall 2010

Stage -1

External software

- VMware Player *http://www.vmware.com/download/player/*
- LinuxVM-0910-1 *http://www.deetc.isel.ipl.pt/programacao/psc/Ferramentas/LinuxVM-0910-1.zip*
- Bochs 2.4.5 *http://sourceforge.net/projects/bochs/files/bochs/2.4.5/bochs-2.4.5.tar.gz/download*

Setting up the basic environment

The following notes indicate how to setup the recommended environment. You may use a different Linux system if you want to, but the more you deviate from the recommended environment, the less we'll be able to help you with configuration issues. In any case, recent Ubuntu variants should cause less trouble than other options.

If you don't have *LinuxVM-0910-1* already installed:

- Install VMware Player
- Unzip *LinuxVM-0910-1.zip* into some convenient location (e.g.: `C:\LinuxVM\`)
- Run *LinuxVM.vmx* and select the option "*I copied it*" when the question pops up
- VMware Player will complain about VMware Tools not being up to date:
  - Press the button "*Update Tools*"
  - Right-click on the desktop background, open a terminal window, and execute:
    - `tar xzvf /media/cdrom0/VMwareTools-*.tar.gz`
    - `cd vmware-tools-distrib`
    - `sudo ./vmware-install.pl`
      NOTE: the system will probably ask you for the password of user *isel,* which is *isel* too
    - (accept all the default options – this will take a while)
    - `cd ..`
    - `rm -rf vmware-tools-distrib`
  - In the end, go to *Applications* > *Log Out* > *Shut Down*
  - Finally, launch LinuxVM again and check that you are able to:
    - resize the VMware Player window
    - move the mouse in and out of the VMware Player window
    - drag-and-drop files between your host system and LinuxVM

Irrespectively of what *LinuxVM-0910-1* you're using:

- Create a new empty folder in your base system (not in LinuxVM) for your work in/for this course (e.g.: `C:\LSC\`). In this document, this folder will be known as `%LSC_HOME%`
- Create a new empty folder under `%LSC_HOME%` with the name `Disks`
- Place the attached file `lsc.fd` in `%LSC_HOME%\Disks`

- In VMware Player, go to *Virtual Machine* > *Virtual Machine Settings…* > *Options* > *Shared Folders* and:
  - select the option *Always enabled*
  - [OPTIONAL] configure the shared folder *work* to point at some useful place in the base system (this may be used as an alternative way to exchange files between host and guest systems)
  - create a new shared folder named *disks* and point it at `%LSC_HOME%\Disks`
  - depending on your previous settings, you may need to restart your LinuxVM now (oh well, just do it…)
- In LinuxVM, open up a terminal window and execute:
  - `cd`
  - mkdir lsc
  - cd lsc
  - ln -s /mnt/hgfs/disks disks
    NOTE: `~/lsc/disks/` in LinuxVM is a link to `%LSC_HOME%\Disks\` in the host system
  - sudo apt-get update

With this setup, all virtual disk files created during this course will be accessible both from the host and the guest systems, allowing them to be tested with several emulators and virtualizers, even those installed at the host. Virtual disk files will reside at `%LSC_HOME%\Disks`, which is accessible from the guest as `~/lsc/disks`

Installing Bochs

Bochs is an open source IA-32 PC emulator with an integrated low-level debugger. Bochs debugging capabilities will be particularly useful during the first stages of this course, where the setup is too hostile for a common debugger.
Ubuntu already has a pre-packaged version of Bochs, but the internal debugger was left out of it, so we need to follow the hard path. In the unlikely event of something going wrong with the instructions we're providing you, another option is to proceed with Ubuntu's default Bochs package. You'll probably miss the debugger, though.

- Place `bochs-2.4.5.tar.gz` at `~/lsc/`, open a terminal window, and execute:
  - `cd ~/lsc`
  - `tar xzvf bochs-2.4.5.tar.gz`
  - `cd bochs-2.4.5`
  - `geany lsc-conf &`
- In the open text editor, place the following text, then save the file and leave the editor:

```
#!/bin/sh
./configure --enable-smp \
            --enable-cpu-level=6 \
            --enable-acpi \
            --enable-all-optimizations \
            --enable-x86-64 \
            --enable-pci \
            --enable-vmx \
            --enable-debugger \
            --enable-disasm \
            --enable-debugger-gui \
            --enable-logging \
            --enable-vbe \
```

```
            --enable-fpu \
            --enable-sb16 \
            --enable-cdrom \
            --enable-x86-debugger \
            --enable-iodebug \
            --disable-plugins \
            --disable-docbook \
            --with-x11
```

- Go back to the terminal window, make sure you're at `~/lsc/bochs-2.4.5` and execute:
  - `chmod 755 lsc-conf`
  - `sudo apt-get install xserver-xorg-dev xorg-dev libgtk2.0-dev`
  - `lsc-conf`
  - `make`
  - `sudo make install`
  - `cp .bochsrc ..`
  - `cd ..`
  - `geany .bochsrc &`
- Make sure that the following options are set accordingly in `.bochsrc` :
  - `display_library: x, options="gui_debug"`
  - `memory: guest=32, host=4`
  - `floppya: 1_44=/mnt/hgfs/disks/lsc.fd, status=inserted`
    NOTE: don't use the soft link as a path for bochs
  - `#ata0: enabled=0, […]`
  - `#ata1: enabled=0, […]`
  - `#ata2: enabled=0, […]`
  - `#ata3: enabled=0, […]`
  - `#ata0-master: […]`
  - `#ata0-slave: […]`
  - `boot: floppy`
  - `#boot: disk`
  - `debugger_log: debugger.out`
- Save the file (`.bochsrc`) and close the editor
- Back at the terminal window, make sure the current directory is `~/lsc` and execute:
  - `bochs`

Writing the code for the "Starting LSC…" boot floppy

In this exercise, a tiny program will be placed in the very first sector of an otherwise empty 1.41MiB floppy disk. This sector will be loaded at physical address 0x07C00 by the BIOS. The program starts by setting a few processor registers with known values, then prints "Starting LSC…" on the screen, and terminates.

The following is the skeleton of the program's source file, `~/lsc/lscboot.s` :

```
    .equ START_ADDR, 0x7C00  # .equ defines a textual substitution

    .text                    # code section starts here
    .code16                  # this is real mode (16 bit) code
```

```
      cli                       # no interrupts while initializing
      # ... init ...            # initialization code...
      sti                       # interrupts enabled after initializing

      # ... prog ...            # main program body...

      # ... term ...            # end of execution...

      .section .rodata          # program constants (no real protection)
msgb: .asciz "Starting LSC..."
msge:

      .data                     # program variables (probably not needed)
lsc:  .long 2010

      .end
```

To use `cs` with a value of 0x0000 and have the boot code start at offset 0x7C00 of the code segment, the program should enforce these addresses in its early steps. A simple way to do it is by issuing a *long jump* instruction (a *far jump*, in Intel's parlance), which simultaneously sets `cs` and `ip` with the specified values:

```
          ...
          ljmp $0, $norm_cs
    norm_cs:
          ...
```

Next, all other segment registers are set to zero. As segment registers can only appear in register-register instructions, they have to be set to zero with code like:

```
          ...
          xorw %ax, %ax
          movw %ax, %ds
          ...
```

It is suggested that the stack pointer be set to 0x7C00, as it can then grow downwards from there.

To write the message on the screen, you should use the BIOS service INT 0x10 / AH=0x0E, which prints a single character in teletype mode. In this mode, the BIOS service will take care of line breaks and scrolling for you. The code to write a single char is:

```
          ...
          movb  $'@', %al    # The character: '@'
          movb  $7  , %bl    # Light gray on black
          xorb  %bh , %bh    # Using page 0

          movb $0x0E, %ah    # Identifying the service
          int  $0x10         # Invoking the BIOS service

          ...
```

After presenting the message, there's nothing else to do, which means that the processor can be stopped. To stop it until the next interrupt, use the halt instruction.

```
        ...
stop:
        hlt
        jmp stop
        ...
```

Building the "Starting LSC..." boot floppy

Generate object code from the assembly source in ~/lsc/lscboot.s:

```
as -o lscboot.o lscboot.s
```

Create the file ~/lsc/bootrec.ld with the following content:

```
OUTPUT_FORMAT("binary")
SECTIONS
{
    .bootrec 0x7C00 : {
       *(.text)
       *(.rodata)
       *(.data)
       . = 510;
       SHORT(0xAA55)
    }
}
```

The boot sector can now be generated with:

```
ld -T bootrec.ld -o lscboot.bin lscboot.o
```

Finally, we use dd to produce the empty floppy space and to copy the boot sector:

dd if=/dev/zero of=lsc.fd bs=512 count=2880

dd if=lscboot.bin of=lsc.fd bs=512 count=1 conv=notrunc

Place the resulting lsc.fd file in ~/lsc/disks/ and run bochs.

To set up a breakpoint at your code, enter "b 0x7c00" (without the quotes) in the command box of the Bochs internal debugger – that's the text box at the bottom of the debug window.