

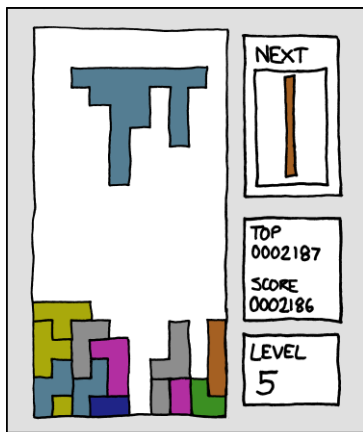
# Introducción a Bochs

## Organización del Computador II

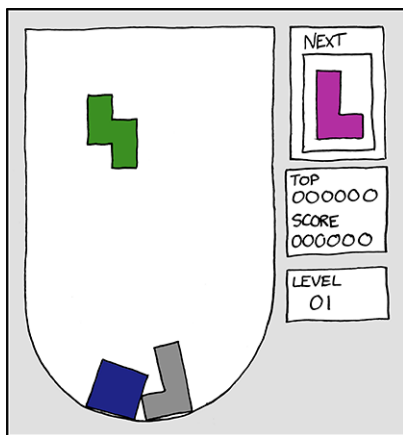
David Alejandro González Márquez

Departamento de Computación  
Facultad de Ciencias Exactas y Naturales  
Universidad de Buenos Aires

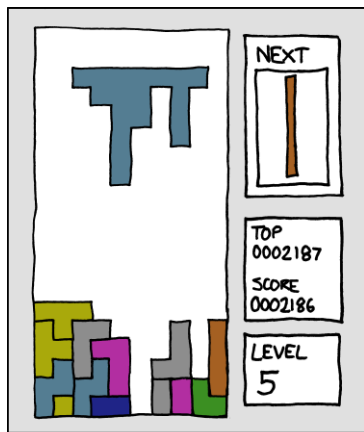
1er Cuatrimestre 2014  
22-05-14



HEAVEN

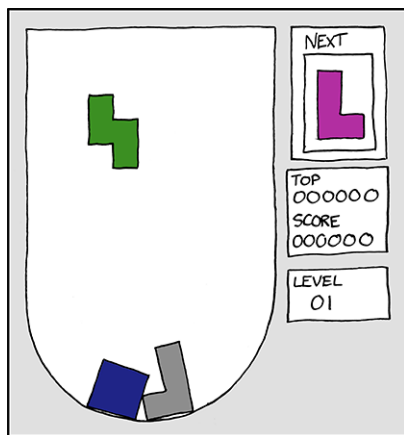


HELL



HEAVEN

Programación a nivel de Usuario



HELL

Programación de S. O.

## ¿Porque usar bochs?

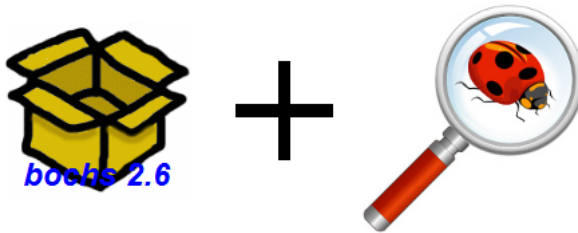
El procesador posee intrucciones que no se pueden usar a nivel de usuario.

Por lo tanto, si queremos acceder a estos mecanismos debemos estar en lugar del sistema operativo.

- Utilizar instrucciones de nivel privilegiado
- Acceder a los mecanismos de manejo de memoria
- Cambiar modos del procesador
- Controlar interrupciones

## VM con Debugger

Un debugger como gdb es un **proceso** más. No puede monitorear el sistema operativo.



Bochs + Debugger

Necesitamos un debugger **en** la Virtual Machine.

## Problemas del bochs

Bochs es un simulador de una computadora por esto nos permite correr instrucción por instrucción.

Pero su versión oficial no esta compilada con esta posibilidad.

Manos a la obra!

- bajar de:  
`http://sourceforge.net/projects/bochs/files/bochs/2.6.2/`  
el archivo `bochs-2.6.2.tar.gz`
- descomprimir: `tar -xvzf bochs-2.6.2.tar.gz`
- en la carpeta descomprimida hacer:
  - `./configure --enable-debugger --enable-disasm`  
`--disable-docbook`  
`--prefix=/home/< usuario >/bochs-2.6.2/`
  - `make`
  - `make install`

# Bochs

## ----- Bochs Configuration: Main Menu -----

This is the Bochs Configuration Interface, where you can describe the machine that you want to simulate. Bochs has already searched for a configuration file (typically called bochsrc.txt) and loaded it if it could be found. When you are satisfied with the configuration, go ahead and start the simulation.

You can also start bochs with the `-q` option to skip these menus.

1. Restore factory default configuration
2. Read options from...
3. Edit options
4. Save options to...
5. Restore the Bochs state from...
6. Begin simulation
7. Quit now

Please choose one: [6]

## Bochs: Config file

- Una imagen de linux de ejemplo:

<http://bochs.sourceforge.net/diskimages.html>

bochsrc

```
megs: 32
romimage: file=$BXSHARE/BIOS-bochs-latest
vgaromimage: file=$BXSHARE/VGABIOS-lgpl-latest
vga: extension=vbe
floppya: 1_44=a.img, status=inserted
floppyb: 1_44=b.img, status=inserted
ata0-master: type=disk, path=boot.img, cylinders=900, heads=15, spt=17
boot: c
log: bochsout.txt
mouse: enabled=0
clock: sync=slowdown
vga_update_interval: 150000
display_library: x, options="gui_debug" # use GTK debugger gui
# This enables the "magic breakpoint" feature when using the debugger.
# The instruction XCHG BX, BX causes Bochs to enter the debugger mode.
magic_break: enabled=1
```



## ¿Y el BIOS juega en primera?

- Cuando una computadora arranca solo existe el BIOS (Basic Input/Output system).
- El proceso de booteo comienza ejecutando el código del BIOS, ubicado en la posición 0xFFFF0, en modo real.
- El BIOS tiene el código en ROM, que realiza la inicialización (por ejemplo, la placa de video) y una verificación de la máquina (POST).
- Luego busca algún dispositivo de booteo: Disco Rígido, Floppy, USB, etc...

## ¿Y el BIOS juega en primera?

- Una vez localizado el dispositivo de arranque, carga el primer sector de 512 bytes (CDROM 2048 bytes) en la posición de memoria 0x07C00 y salta a esa dirección.
- Ahora la imagen de arranque es la encargada de cargar el kernel y luego pasarle el control.
- Para que una imagen sea de arranque debe ocupar exactamente 512 bytes (excepto en el CDROM), y estar firmada en los últimos dos bytes con 0x55AA.
- Una imagen de linux de ejemplo:  
<http://bochs.sourceforge.net/diskimages.html>

## ¿Qué podemos hacer?

- Creamos un breakpoint en la posición de memoria física donde comenzará a cargar el bootloader

```
<bochs:1> break 0x07C00
```

- Leemos la posición de memoria donde debería estar la firma del bootloader una vez que carga en memoria principal

```
<bochs:2> x/1x 0x07C00+510
```

```
[bochs]:
```

```
0x00007dfe <bogus+ 0>: 0x00000000
```

## ¿Qué podemos hacer?

- Continuamos la ejecución, hasta que llegamos al breakpoint

```
<bochs:3> c
```

```
(0) Breakpoint 1, 0x00007c00 in ?? ()
```

```
Next at t=49462126
```

```
(0) [0x00007c00] 0000:7c00 (unk. ctxt): cli ; fa
```

- Nuevamente, podemos leer y notar que el BIOS cargo los 512bytes pertenecientes al bootloader, primer sector de la unidad

```
<bochs:4> x/1x 0x07C00+510
```

```
[bochs]:
```

```
0x00007dfe <bogus+ 0>: 0x0000aa55
```

## Next y Step

### Opciones de debugging

- s | step | stepi [count] - ejecuta [count] instrucciones
- n | next | p - ejecuta instrucciones sin entrar a las subrutinas
- c | cont | continue - continua la ejecución
- q | quit | exit - sale del debugger y del emulador
- Ctrl-C Detiene la ejecución y retorna al prompt

## Registros de uso general

- `r | reg | regs | registers` - Lista los registros del CPU y sus contenidos

```
<bochs:12> registers
```

```
eax: 0x00000000 0
```

```
ecx: 0x00000000 0
```

```
edx: 0x00000543 1347
```

```
ebx: 0x00000000 0
```

```
esp: 0x00000000 0
```

```
ebp: 0x00000000 0
```

```
esi: 0x00000000 0
```

```
edi: 0x00000000 0
```

```
eip: 0x0000e05d
```

```
eflags 0x00000046
```

```
id vip vif ac vm rf nt IOPL=0 of df if tf sf ZF af PF cf
```

## FPU (deprecated)

- fp | fpu - Muestra el estado de la FPU

```
<bochs:13> fpu
status word: 0x0000: b c3 TOS0 c2 c1 c0 es sf pe ue oe ze de ie
control word: 0x0040: inf RC\_NEAREST PC\_32 pm um om zm dm im
tag word:      0x5555
operand:      0x0000
fip:          0x00000000
fcs:          0x0000
fdp:          0x00000000
fds:          0x0000
=>FP0 ST0(0):      raw 0x0000:0000000000000000 (0.0000000000) (ZERO)
  FP1 ST1(0):      raw 0x0000:0000000000000000 (0.0000000000) (ZERO)
  FP2 ST2(0):      raw 0x0000:0000000000000000 (0.0000000000) (ZERO)
  FP3 ST3(0):      raw 0x0000:0000000000000000 (0.0000000000) (ZERO)
  FP4 ST4(0):      raw 0x0000:0000000000000000 (0.0000000000) (ZERO)
  FP5 ST5(0):      raw 0x0000:0000000000000000 (0.0000000000) (ZERO)
  FP6 ST6(0):      raw 0x0000:0000000000000000 (0.0000000000) (ZERO)
  FP7 ST7(0):      raw 0x0000:0000000000000000 (0.0000000000) (ZERO)
```

# MMX

- mmx - Muestra el estado de los registros de MMX

```
<bochs:3> mmx
```

```
MM[0] : 00000000:00000000
```

```
MM[1] : 00000000:00000000
```

```
MM[2] : 00000000:00000000
```

```
MM[3] : 00000000:00000000
```

```
MM[4] : 00000000:00000000
```

```
MM[5] : 00000000:00000000
```

```
MM[6] : 00000000:00000000
```

```
MM[7] : 00000000:00000000
```



# Memory Dump

- `x /nuf [addr]` - Muestra el contenido de la dirección `[addr]`
- `xp /nuf [addr]` - Muestra el contenido de la dirección física `[addr]` `nuf` es número que indica cuantos valores se mostrarán, seguido de uno o más de los indicadores de formato.
  - `x` : hex
  - `d` : decimal
  - `u` : sin signo
  - `o` : octal
  - `t` : binario
  - `c` : char
  - `s` : ascii
  - `i` : instrucción

select the size:

- `b` : byte
- `h` : word = half-word
- `w` : dobleword = word

## Memory Disassemble

- `u | disasm | disassemble [count] [start] [end]` - desensambla instrucciones desde la dirección lineal `[start]` hasta `[end]` exclusive.
- `u | disasm | disassemble switch-mode` - Selecciona la sintaxis Intel o AT&T de assembler
- `u | disasm | disassemble size = n` - Setea el tamaño del segmento a desensamblar

# Breakpoints

- `p | pb | break | pbreak [addr]` - Crea un breakpoint en la dirección física `[addr]`
- `vb | vbreak [seg:offset]` - Crea un breakpoint en la dirección virtual `[addr]`
- `lb | lbreak [addr]` - Crea un breakpoint en la dirección lineal `[addr]`
- `d | del | delete [n]` - Borra el breakpoint número `[n]`
- `bpe [n]` - Activa el breakpoint número `[n]`
- `bpd [n]` - Desactiva el breakpoint número `[n]`

# Watches

- `watch` - Muestra el estado actual de los watches
- `watch stop` - Detiene la simulación cuando un watch es encontrado
- `watch continue` - No detiene la simulación si un watch es encontrado
- `watch r | read [addr]` - Agrega un watch de lectura en la dirección física `[addr]`
- `watch w | write [addr]` - Agrega un watch de escritura en la dirección física `[addr]`

## Infos

- info break - Muestra los Breakpoint creados
- info eflags - Muestra el registro EEFLAGS
- info idt - Muestra el descriptor de interrupciones (idt)
- info ivt - Muestra la tabla de vectores de interrupción
- info gdt - Muestra la tabla global de descriptores (gdt)
- info tss - Muestra el segmento de estado de tarea actual (tss)
- info tab - Muestra la tabla de paginas

## Registros de Segmento

- sreg - Muestra los registros de segmento

```
<bochs:5> sreg
cs:s=0xf000, dh=0xff0093ff, dl=0x0000ffff, valid=7
ds:s=0x0000, dh=0x00009300, dl=0x0000ffff, valid=7
ss:s=0x0000, dh=0x00009300, dl=0x0000ffff, valid=7
es:s=0x0000, dh=0x00009300, dl=0x0000ffff, valid=7
fs:s=0x0000, dh=0x00009300, dl=0x0000ffff, valid=7
gs:s=0x0000, dh=0x00009300, dl=0x0000ffff, valid=7
ldtr:s=0x0000, dh=0x00008200, dl=0x0000ffff, valid=1
tr:s=0x0000, dh=0x00008b00, dl=0x0000ffff, valid=1
gdtr:base=0x00000000, limit=0xffff
idtr:base=0x00000000, limit=0xffff
```

## Registros de Control

- creg - Muestra los registros de control

```
<bochs:10> creg
```

```
CR0=0x60000010: pg CD NW ac wp ne ET ts em mp pe
```

```
CR2=page fault laddr=0x00000000
```

```
CR3=0x00000000
```

```
PCD=page-level cache disable=0
```

```
PWT=page-level writes transparent=0
```

```
CR4=0x00000000: osxsave smx vmx osxmmexcpt osfxsr pce pge mce pae pse de tsd pv
```

Gracias!!!

¿Preguntas?