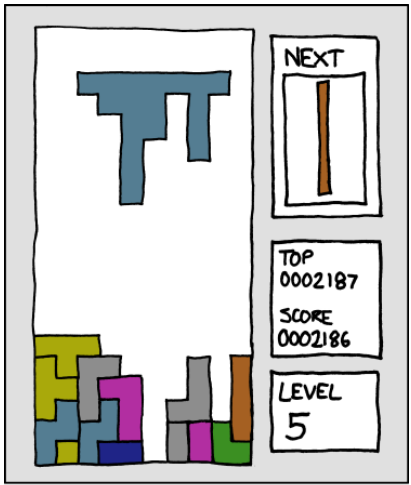
Introducción a Bochs y el Bootloader

Programación de Sistemas Operativos

David Alejandro González Márquez

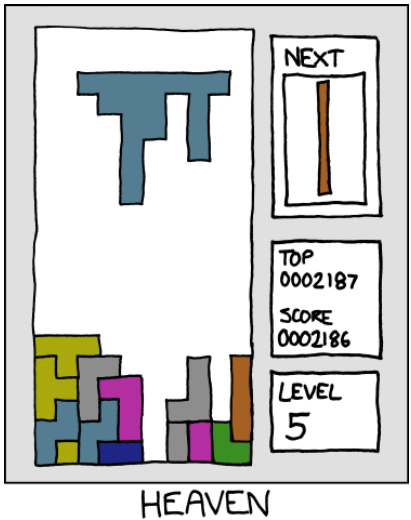
Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires



HEAVEN

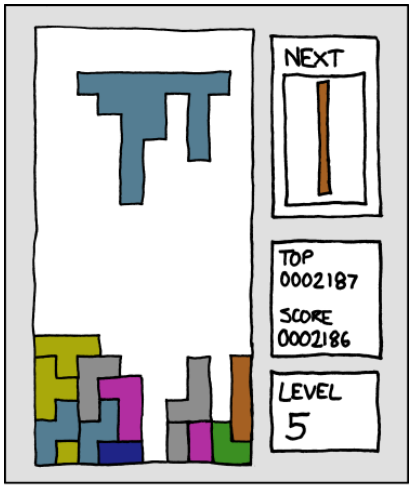
Fuente: https://xkcd.com/

Programación a nivel de Usuario

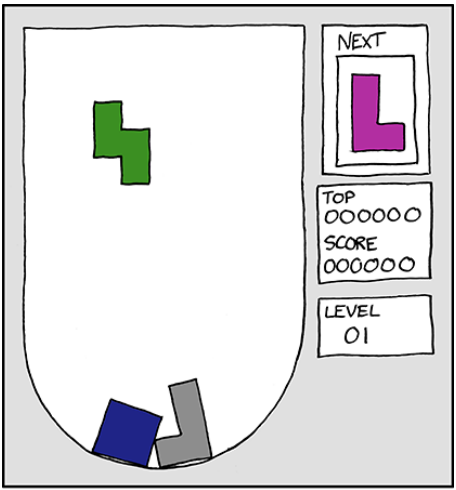


Fuente: https://xkcd.com/

Programación a nivel de Usuario



Programación de Sistemas Operativos



Fuente: https://xkcd.com/

Fuente: Marvel Comics

Fuente: Marvel Comics

Agenda

- Introducción a Bochs y su debugger
- Funcionamiento y responsabilidades del Bootloader
- Comandos para compilar y enlazar código de sistemas

¿Porque usar bochs?

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

¿Porque usar bochs?

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

Si queremos acceder a estos mecanismos debemos estar en lugar del **sistema operativo**.

¿Porque usar bochs?

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

Si queremos acceder a estos mecanismos debemos estar en lugar del **sistema operativo**.

- Utilizar instrucciones de nivel privilegiado

¿Porque usar bochs?

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

Si queremos acceder a estos mecanismos debemos estar en lugar del **sistema operativo**.

- Utilizar instrucciones de nivel privilegiado
- Ver estructuras del procesador

¿Porque usar bochs?

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

Si queremos acceder a estos mecanismos debemos estar en lugar del **sistema operativo**.

- Utilizar instrucciones de nivel privilegiado
- Ver estructuras del procesador
- Cambiar modos del procesador

¿Porque usar bochs?

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

Si queremos acceder a estos mecanismos debemos estar en lugar del **sistema operativo**.

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

¿Porque usar bochs?

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

Si queremos acceder a estos mecanismos debemos estar en lugar del **sistema operativo**.

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

bochs: una Virtual Machine con Debugger

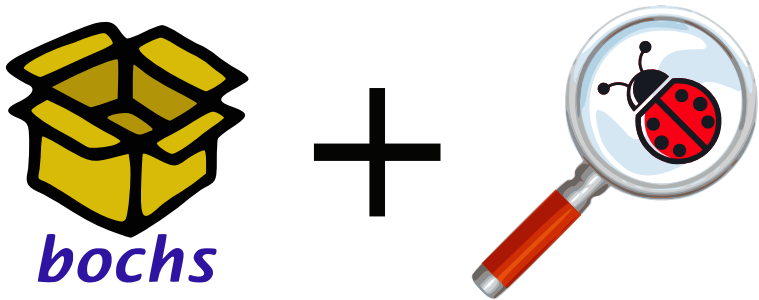
Un debugger como GDB es un proceso más $\Longrightarrow$ No puede monitorear el **Sistema Operativo**

bochs: una Virtual Machine con Debugger

Un debugger como GDB es un proceso más $\Rightarrow$ No puede monitorear el **Sistema Operativo**

bochs: una Virtual Machine con Debugger

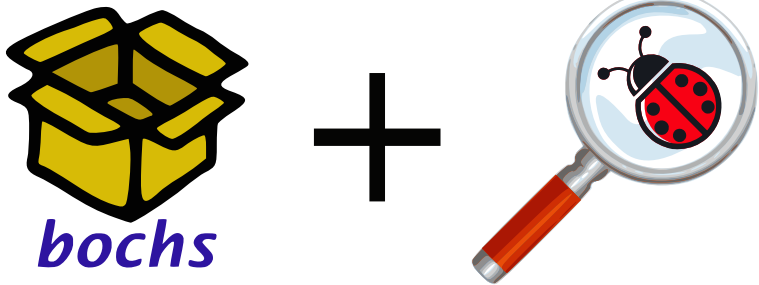
Un debugger como GDB es un proceso más $\Rightarrow$ No puede monitorear el **Sistema Operativo**



Bochs + Debugger

bochs: una Virtual Machine con Debugger

Un debugger como GDB es un proceso más $\Rightarrow$ No puede monitorear el **Sistema Operativo**



Bochs + Debugger

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

Problemas del 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.

Problemas del 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.

- bajar de: **`https://sourceforge.net/projects/bochs/files/bochs/2.6.9/`**
  el archivo: **`bochs-2.6.9.tar.gz`**

Problemas del 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.

- bajar de: **`https://sourceforge.net/projects/bochs/files/bochs/2.6.9/`**
  el archivo: **`bochs-2.6.9.tar.gz`**

- descomprimir: **`tar -xvvzf bochs-2.6.9.tar.gz`**

Problemas del 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.

- bajar de: **`https://sourceforge.net/projects/bochs/files/bochs/2.6.9/`**
  el archivo: **`bochs-2.6.9.tar.gz`**
- descomprimir: **`tar -xvvzf bochs-2.6.9.tar.gz`**
- en la carpeta descomprimida hacer:

```
$ ./configure --enable-debugger --enable-disasm --disable-docbook
  --enable-readline LDFLAGS='-pthread' --prefix=/home/< usuario >/bochs/
$ make
$ make install
```

Bonus para instalación

Paquetes que pueden faltar instalar:

```
$ sudo apt-get install xorg-dev
$ sudo apt-get install libx11-dev
$ sudo apt-get install libxrandr-dev
$ sudo apt-get install libgtk2.0-dev
```

Bonus para instalación

Paquetes que pueden faltar instalar:

```
$ sudo apt-get install xorg-dev
$ sudo apt-get install libx11-dev
$ sudo apt-get install libxrandr-dev
$ sudo apt-get install libgtk2.0-dev
```

Paquete adicional:

Permite tener un buffer de la consola de últimos comandos ejecutados. (aka “flechitas”)

```
$ sudo apt-get install libreadline-dev
```

Atajos útiles

Saltar el primer breakpoint y cargar sin menu

- Crear un archivo de nombre bochsdbg con el contenido:

```
continue
```

- Cargar el bochs usando:

```
$ bochs -q -rc bochsdbg
```

Atajos útiles

Saltar el primer breakpoint y cargar sin menu

- Crear un archivo de nombre bochsdbg con el contenido:

```
continue
```

- Cargar el bochs usando:

```
$ bochs -q -rc bochsdbg
```

Usar bochs desde cualquier path

- Agregar en el archivo /home/< *usuario* >/.bashrc:

```
export PATH+=":/home/< usuario >/bochs/bin/"
```

- Cargar cambios en la consola actual:

```
$ source ~/.bashrc
```

Configuraciones útiles

Activar log de eventos

- Descomentar la siguiente línea del bochsrc:

```
#log: bochs.log
```

- Comentar la siguiente:

```
log: /dev/null
```

- Para generar logs de todos los eventos reemplazar:

```
debug: action=ignore ⟶ debug: action=report
```

El tamaño del archivo resultante puede ser muy grande ya que registra todos los eventos.

Configuraciones útiles

Activar log de eventos

- Descomentar la siguiente línea del bochsrc:

```
#log: bochs.log
```

- Comentar la siguiente:

```
log: /dev/null
```

- Para generar logs de todos los eventos reemplazar:

```
debug: action=ignore ⟶ debug: action=report
```

El tamaño del archivo resultante puede ser muy grande ya que registra todos los eventos.

Activar GUI del debugger

- Descomentar la siguiente línea del bochsrc:

```
#display_library: x, options="gui_debug"# use GTK debugger gui
```

Bochs: Config file

Para 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
```

Debugger

Ejecución: Next y Step

- 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 promt

Registros de propósito 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
```

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	

y de tamaño

b : byte	h : word = half-word	w : doubleword = word

Memory Disassemble

- u | disasm | disassemble [count] [start] [end]
  Desensambla intrucciones desde la dirección lineal [start] hasta [end] exclusive.
- u | disasm | disassemble switch-mode
  Selecciona la sintaxis Intel o AT&T de asembler.
- 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]

Watchs

- watch — Muestra el estado actual de los watchs
- watch stop — Detiene la simulación cuando un watch es encontrado
- watch continue — No detiene la simulación si un wath 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 pvi vme
```

Magic Breakpoint

- xchg bx, bx Magic breakpoint

Esta instrucción detiene el flujo del programa y nos devuelve al prompt de bochs.

Magic Breakpoint

- xchg bx, bx Magic breakpoint

Esta instrucción detiene el flujo del programa y nos devuelve al prompt de bochs.

Fuente: Shia LaBeouf ‘Magic’ meme.

Más información

- Manual:
  `http://bochs.sourceforge.net/doc/docbook/user/`
- Información complementaria:
  `http://wiki.osdev.org/Bochs`
- Imagenes de discos:
  `http://bochs.sourceforge.net/diskimages.html`

Bootloader

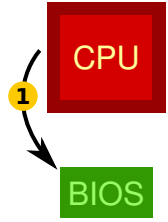
Inicio - Pasos de Boot

0 Presionamos el botón de encendido,
la circuitería del mother da alimentación
al microprocesador y arranca el sistema

Inicio - Pasos de Boot

0 Presionamos el botón de encendido, la circuitería del mother da alimentación al microprocesador y arranca el sistema

1 El CPU comienza a ejecutar el BIOS (Basic Input Output System), que consiste de una memoria ROM en el mother con las primeras instrucciones para el CPU

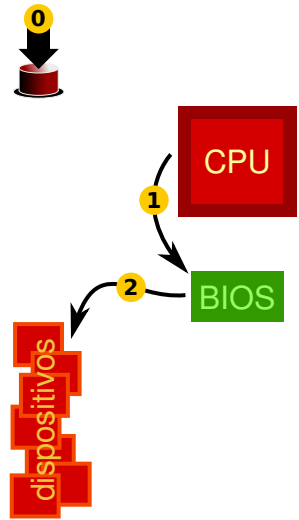


Inicio - Pasos de Boot

0 Presionamos el botón de encendido, la circuitería del mother da alimentación al microprocesador y arranca el sistema

1 El CPU comienza a ejecutar el BIOS (Basic Input Output System), que consiste de una memoria ROM en el mother con las primeras instrucciones para el CPU

2 El BIOS se encarga de correr una serie de diagnósticos llamados POST (Power On Self Test)



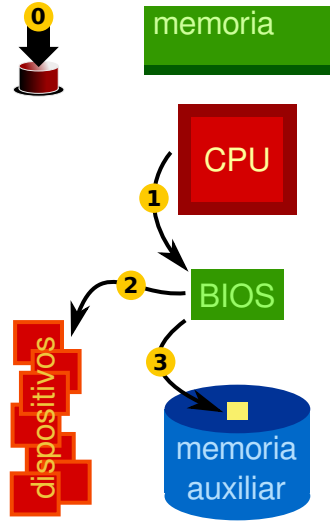
Inicio - Pasos de Boot

0 Presionamos el botón de encendido, la circuitería del mother da alimentación al microprocesador y arranca el sistema

1 El CPU comienza a ejecutar el BIOS (Basic Input Output System), que consiste de una memoria ROM en el mother con las primeras instrucciones para el CPU

2 El BIOS se encarga de correr una serie de diagnósticos llamados POST (Power On Self Test)

3 Busca un dispositivo "booteable" es decir, que en su sector de booteo los últimos dos bytes tengan la firma 0x55 y 0xAA respectivamente.



Inicio - Pasos de Boot

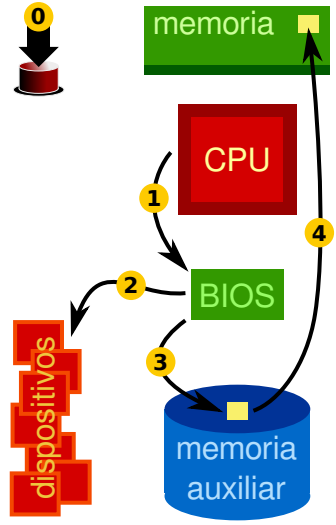
0 Presionamos el botón de encendido, la circuitería del mother da alimentación al microprocesador y arranca el sistema

1 El CPU comienza a ejecutar el BIOS (Basic Input Output System), que consiste de una memoria ROM en el mother con las primeras instrucciones para el CPU

2 El BIOS se encarga de correr una serie de diagnósticos llamados POST (Power On Self Test)

3 Busca un dispositivo "booteable" es decir, que en su sector de booteo los últimos dos bytes tengan la firma 0x55 y 0xAA respectivamente.

4 Se copia a memoria a partir de la dirección 0x7C00, el sector de booteo



Resumen

- 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 código en ROM que realiza la inicialización (por ejemplo, la placa de video) y una verificación inicial de la máquina: POST (Power on self-test).
- Luego busca algún dispositivo de booteo: Disco Rígido, Floppy, USB, etc...
- 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`

Ejemplo en 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]
```

Ejemplo en Bochs

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

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

Ejemplo en Bochs

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

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

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

```
<bochs:2> x/1x 0x07C00+510
[bochs]:
0x00007dfe <bogus+ 0>: 0x00000000
```

Ejemplo en Bochs

- 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
```

Ejemplo en Bochs

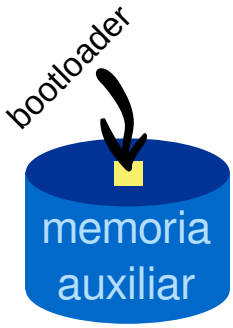
- Continuamos la ejecución, hasta que llegamos al breakpoint

```
<bochs:3> c
(0) Breakpoint 1, 0x00007c00 in ?? ()
Next at t=49462126
(0) [0x00007c00] 0000:7c00 (unk. ctxt): cli ; fa
```

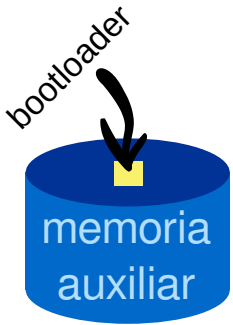
- Nuevamente, podemos leer y notar que el BIOS cargo los 512bytes pertenecientes al bootloader, primer sector de la unidad

```
<bochs:4> x/1x 0x07C00+510
[bochs]:
0x00007dfe <bogus+ 0>: 0x0000aa55
```

Bootloader - Pasos para indicar



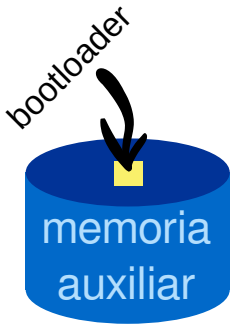
Bootloader - Pasos para indicar



1- Determinar el 'disco' y la partición a bootear

¿ USB drive SATA drive DVD drive FLOPPY drive ?

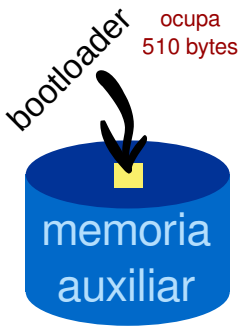
Bootloader - Pasos para indicar



1- Determinar el 'disco' y la partición a bootear

2- Determinar donde esta la imagen del kernel en ese 'disco'

Bootloader - Pasos para indicar



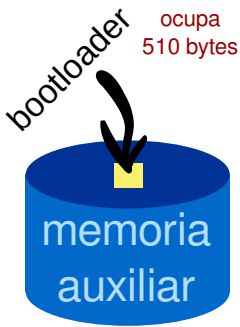
¡Todo esto en 510 bytes!

1- Determinar el 'disco' y la partición a bootear

2- Determinar donde esta la imagen del kernel en ese 'disco'

3- Cargar la imagen del kernel en memoria

Bootloader - Pasos para indicar



¡Todo esto en 510 bytes!

1- Determinar el 'disco' y la partición a bootear

2- Determinar donde esta la imagen del kernel en ese 'disco'

3- Cargar la imagen del kernel en memoria

4- Correr el "kernel"

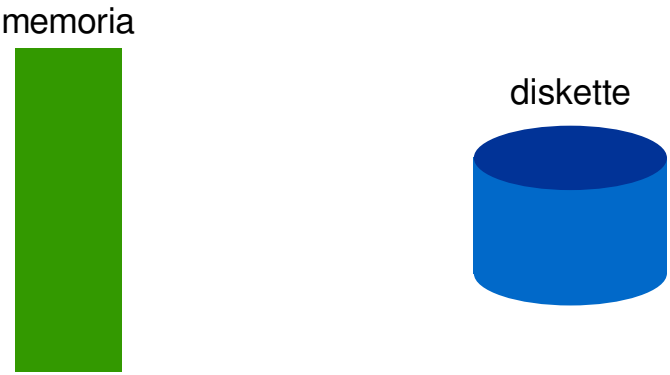
4.1- Pasar a modo protegido

4.2- Preparar las estructuras para administrar la memoria

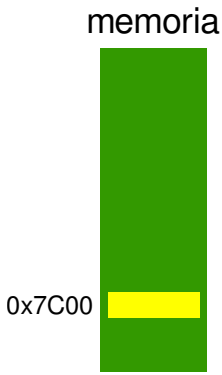
4.3- Preparar las estructuras del sistema

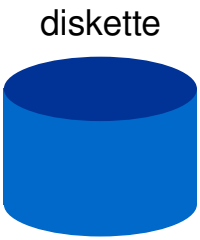
4.4- ¡Listo!

EL Bootloader de Orga2



EL Bootloader de Orga2

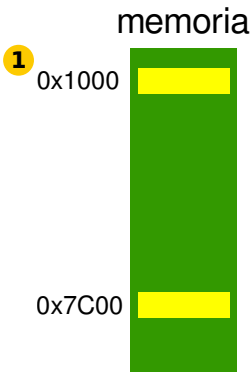




EL Bootloader de Orga2

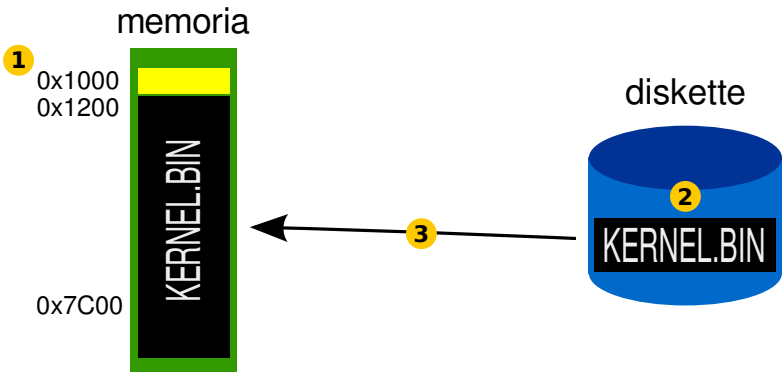
1 - Se copia el Bootloader en la posición **0x1000** de la memoria

EL Bootloader de Orga2



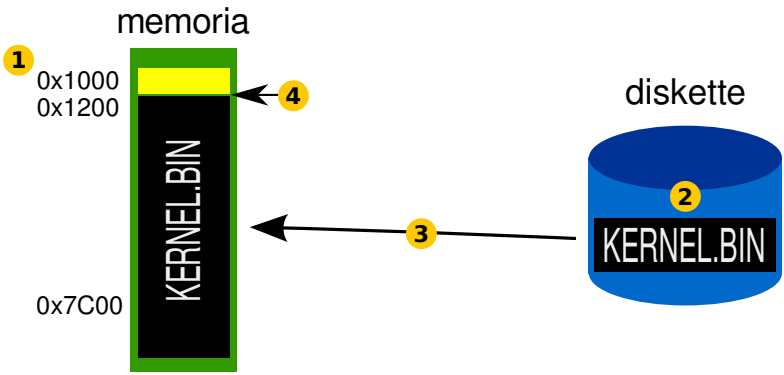
1- Se copia el Bootloader en la posición **0x1000** de la memoria
2- Se busca el archivo **KERNEL.BIN** en el diskette

EL Bootloader de Orga2



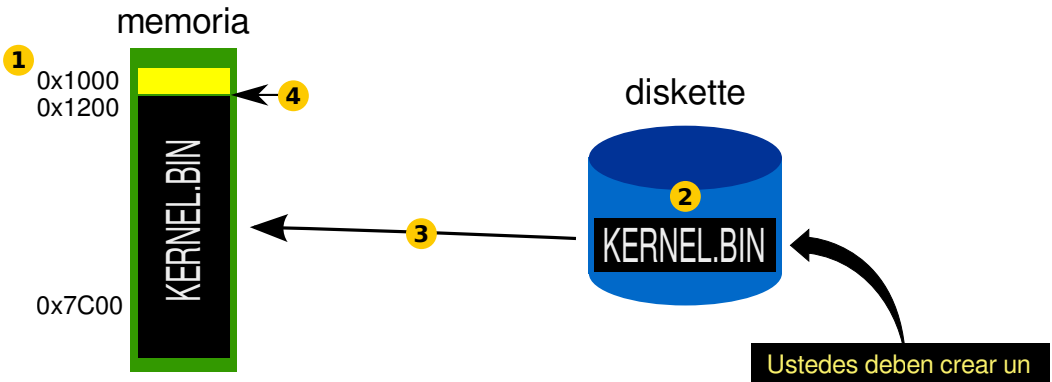
1 - Se copia el Bootloader en la posición **0x1000** de la memoria
2 - Se busca el archivo **KERNEL.BIN** en el diskette
3 - Se copia ese archivo en la posición **0x1200** de la memoria

EL Bootloader de Orga2



1 - Se copia el Bootloader en la posición **0x1000** de la memoria
2 - Se busca el archivo **KERNEL.BIN** en el diskette
3 - Se copia ese archivo en la posición **0x1200** de la memoria
4 - Se salta y se ejecuta la instrucción en la posición **0x1200** de la memoria

EL Bootloader de Orga2



1 - Se copia el Bootloader en la posición **0x1000** de la memoria
2 - Se busca el archivo **KERNEL.BIN** en el diskette
3 - Se copia ese archivo en la posición **0x1200** de la memoria
4 - Se salta y se ejecuta la instrucción en la posición **0x1200** de la memoria

Compilando y Enlazando

- Un compilador construye un programa de forma que pueda correr sobre un **sistema operativo** determinado

Compilando y Enlazando

- Un compilador construye un programa de forma que pueda correr sobre un **sistema operativo** determinado
- Para resolver direcciones, toma una **dirección de inicio**, por ejemplo `0x00000000`

Compilando y Enlazando

- Un compilador construye un programa de forma que pueda correr sobre un **sistema operativo** determinado
- Para resolver direcciones, toma una **dirección de inicio**, por ejemplo `0x00000000`
- Cada etiqueta se traduce a una dirección contando **bytes** desde la dirección de inicio

Compilando y Enlazando

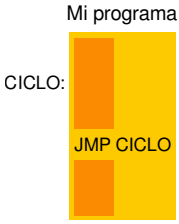
- Un compilador construye un programa de forma que pueda correr sobre un **sistema operativo** determinado
- Para resolver direcciones, toma una **dirección de inicio**, por ejemplo `0x00000000`
- Cada etiqueta se traduce a una dirección contando **bytes** desde la dirección de inicio
- Los **destinos** de saltos o llamadas a funciones se debe conocer en tiempo de enlazado

Compilando y Enlazando

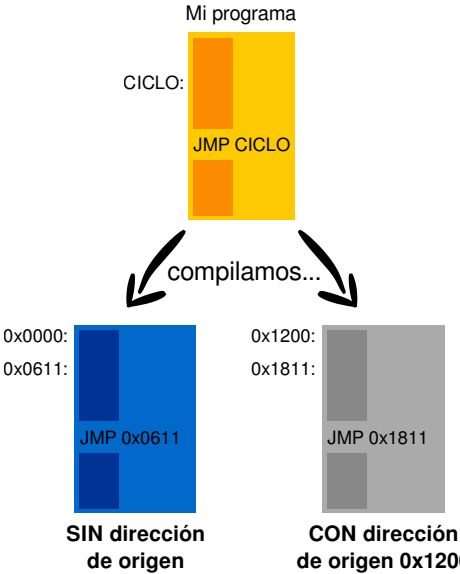
- Un compilador construye un programa de forma que pueda correr sobre un **sistema operativo** determinado
- Para resolver direcciones, toma una **dirección de inicio**, por ejemplo `0x00000000`
- Cada etiqueta se traduce a una dirección contando **bytes** desde la dirección de inicio
- Los **destinos** de saltos o llamadas a funciones se debe conocer en tiempo de enlazado
- ¿Y si no estamos en un sistema operativo?

  Ej: el archivo `KERNEL.BIN` se carga en la dirección `0x1200`

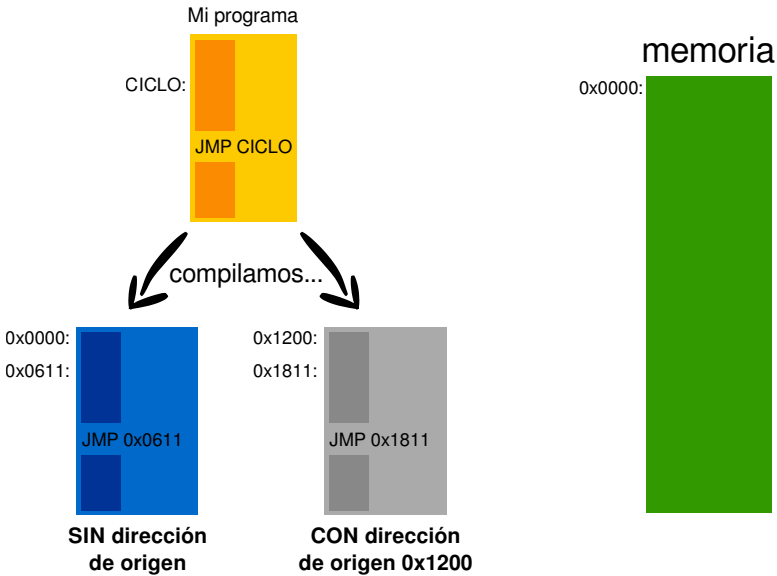
Compilando y Enlazando



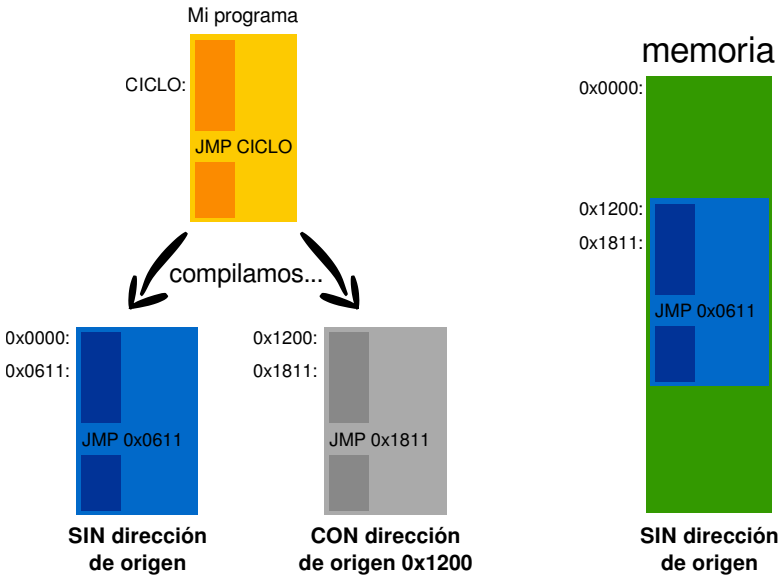
Compilando y Enlazando



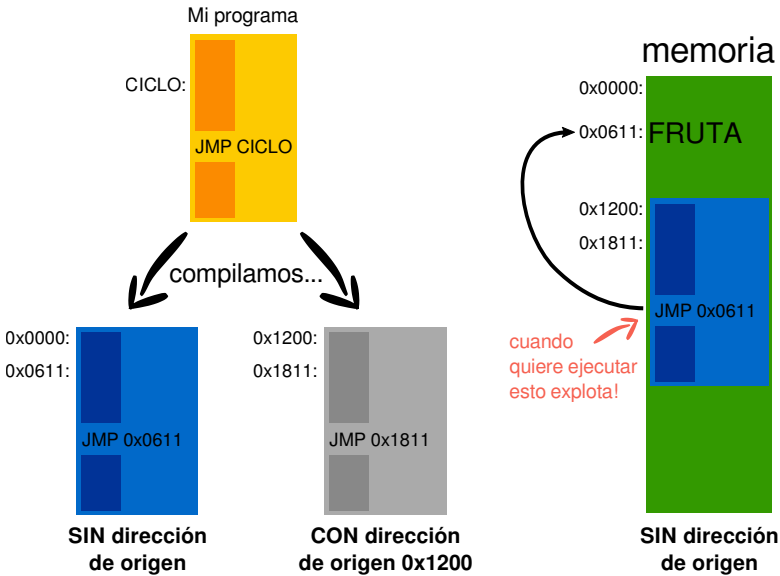
Compilando y Enlazando



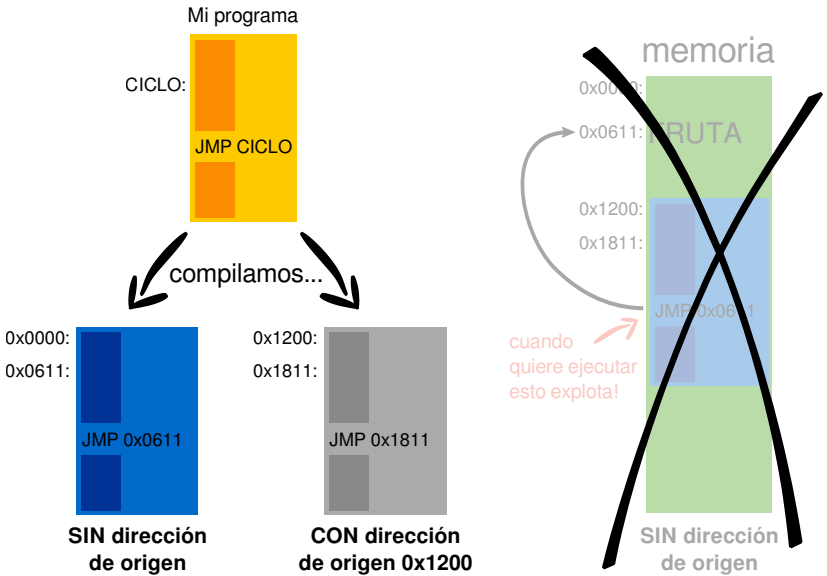
Compilando y Enlazando



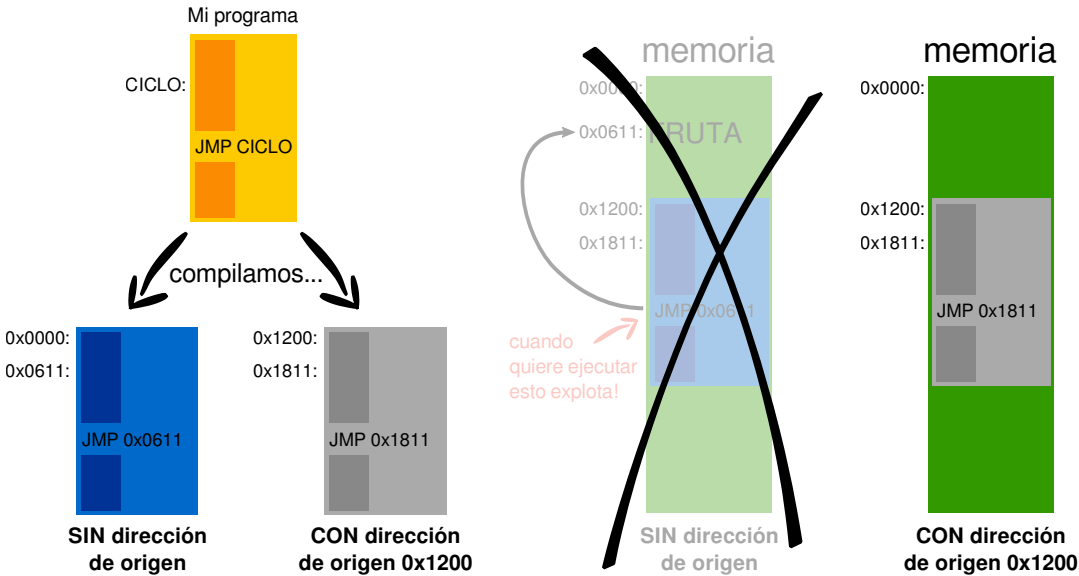
Compilando y Enlazando



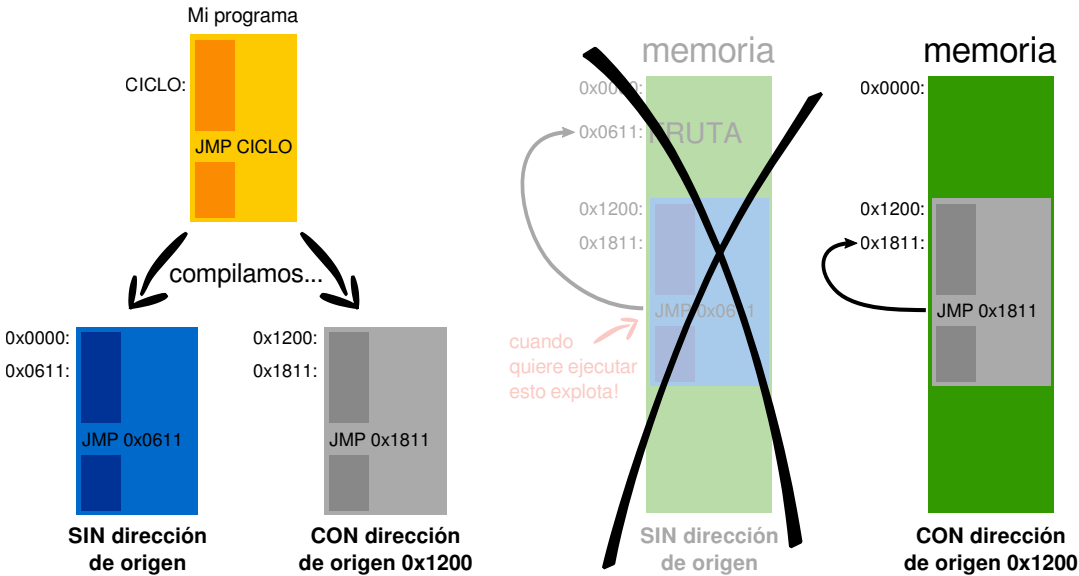
Compilando y Enlazando



Compilando y Enlazando



Compilando y Enlazando



Compilando y Enlazando

Indicar la dirección de origen

Hay 2 formas de hacerlo, según como se compile:

Compilando y Enlazando

Indicar la dirección de origen

Hay 2 formas de hacerlo, según como se compile:

- De **ensamblador a binario**, usamos la directiva `ORG` al inicio del `archivo.asm` para indicar la direccion de origen

```
ORG 0x1200
```

Compilando y Enlazando

Indicar la dirección de origen

Hay 2 formas de hacerlo, según como se compile:

- De **ensamblador a binario**, usamos la directiva ORG al inicio del `archivo.asm` para indicar la direccion de origen

```
ORG 0x1200
```

- De **ensamblador a elf**, usamos el parámetro `-Ttext` en el linker

```
-Ttext 0x1200
```

Compilando y Enlazando

Compilación de ensamblador a binario

- Ensamblado:

```
nasm -fbin archivo.asm -o archivo.bin
```

- Consideraciones:
  - Todo el código ejecutable tiene que estar incluido (No hay bibliotecas)
  - Se ejecuta tal cual se escribió, no hay entry point.

Compilando y Enlazando

Compilación de C en formato elf32 y linkeo

- Compilación:

```
gcc -m32 -fno-zero-initialized-in-bss -fno-stack-protector
-ffreestanding -c -o archivo.elf archivo.c
```

- Linkeo:

```
ld -static -m elf i386 -nostdlib -N -b elf32-i386 -e start
-Ttext 0x1200 -o archivo.elf archivo.o
```

- Lo convertimos en binario:

```
objcopy -S -O binary archivo.elf archivo.bin
```

Compilando y Enlazando

Compilación de assembly en formato elf32 y linkeo

- Compilación:

```
nasm -felf32 archivo.asm -o archivo.o
```

- Linkeo:

```
ld -static -m elf i386 --oformat binary -b elf32-i386 -e start
-Ttext 0x1200 archivo.o -o archivo.bin
```

- Consideraciones:
  - OJO código de 32 bits (en modo protegido)
  - Se pueden usar biblotecas. No se respeta el entry point.
  - El parámetro Ttext da el orígen de la sección .text.
  - Si usan el Bootloader de Orga 2, deben usar 0x1200 como origen de la sección .text.

Compilando y Enlazando

Compilación de un bootloader y creacion de diskette

- Creamos un diskette vacio:

```
dd bs=512 count=2880 if=/dev/zero of=diskette.img
```

- Formateamos la imagen en FAT12:

```
sudo mkfs.msdos -F 12 diskette.img -n ETIQUETA
```

- Escribimos en el sector de booteo:

```
dd if=bootloader.bin of=diskette.img count=1 seek=0 conv=notrunc
```

- Copiado del KERNEL.BIN dentro del diskette

```
mcopy -i diskette.img kernel.bin ::/
```

Bibliografía: Fuentes y material adicional

- Convenciones de llamados a función en x86:
  `https://en.wikipedia.org/wiki/X86_calling_conventions`
- Notas sobre System V ABI:
  `https://wiki.osdev.org/System_V_ABI`
- Documentación de NASM:
  `https://nasm.us/doc/`
- Artículo sobre el flag `-pie`:
  `https://eklitzke.org/position-independent-executables`
- Documentación de System V ABI:
  `https://uclibc.org/docs/psABI-x86_64.pdf`
- Manuales de Intel:
  `https://software.intel.com/en-us/articles/intel-sdm`

¡Gracias!

Recuerden leer los comentarios al final de este video por aclaraciones o fe de erratas.