Guía de instalación y uso de Bochs

Organización del Computador II

Compilando Bochs

Bochs es un emulador de x86 con el cual vamos a poder ejecutar instrucciones privilegiadas y diseñar un sistema desde cero en un ambiente seguro y depurable. Para dotar a Bochs de las capacidades de debugging, tenemos que compilarlo manualmente, ya que las versiones distribuidas oficialmente vienen con esa opción desactivada.

El proceso de compilación se realiza en una terminal y se describe a continuación: libxrandr-dev

- Instalar primero dependencias que puedan faltar:

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

- Bajar el código fuente `bochs-2.7.tar.gz` de **https://sourceforge.net/projects/bochs/files/bochs/2.7/**

- Descomprimir haciendo:

  ```
  $ tar -xvf bochs-2.7.tar.gz
  ```

- En la carpeta descomprimida hacer:

  ```
  $ ./configure --enable-debugger --disable-docbook --enable-readline \
  --prefix=$HOME/bochs/
  $ make
  $ make install
  ```

- Para hacer accesible `bochs` desde cualquier ubicación, agregar la siguiente línea al final del archivo `.bachrc` (localizado en nuestro *home*)

  ```
  export PATH="$HOME/bochs/bin/:$PATH"
  ```

- Para releer al archivo `.bashrc` en la terminal actual:

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

- Listo! ya se puede ejecutar bochs

  ```
  $ bochs
  ```

Ejecutando Bochs

Bochs, al iniciarse, busca un archivo llamado **bochsrc**, que dispone de muchas opciones de configuración tanto del microprocesador como de los periféricos, el debugger, la imagen de booteo, entre otras muchas cosas. En el contexto de trabajo de la materia, la cátedra ya les brinda un archivo **bochsrc** con una configuración suficiente para arrancar a trabajar. Si desean más información sobre las configuraciones disponibles, les referimos a la documentación oficial.

En general, dicho archivo se encuentra en el directorio desde donde estamos llamando a **bochs**. De no encontrarse allí, se inicia un orden de búsqueda.

Al iniciar bochs, nos presenta un menú y en general, vamos a querer entrar en la opción 6, para comenzar con la simulación. Para que no nos presente el menú e ir directo a la simulación, ejecutamos:

```
$ bochs -q
```

A continuación, veremos los comandos más útiles del debugger[1]

Debugger

Magic breakpoint

Para insertar un breakpoint podemos poner la siguiente instrucción en nuestro código:

```
xchg bx,bx
```

En la simulación esta instrucción no tiene efecto, ya que intercambia el valor del registro `bx` consigo mismo. Para que esto funcione, debe estar en el archivo `bochsrc` la siguiente línea:

```
magic_break: enabled=1
```

Control de ejecución

- `s | step [count]`
  Ejecuta `[count]` instrucciones. Por default `[count]` es 1
- `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 | Ctrl-D`
  Detiene la ejecución y retorna al prompt

Registros

- `r | reg | regs | register`
  Lista los registros del CPU y sus contenidos
- `sreg`
  Muestra los registros de segmento
- `creg`
  Muestra los registros de control

[1] Más información en https://bochs.sourceforge.io/doc/docbook/user/internal-debugger.html

info

- `info break`
  Muestra los Breakpoint creados
- `info eflags`
  Muestra el registro EFLAGS
- `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
- `page [laddr]`
  Muestra el mapeo de la dirección lineal `laddr` a dirección física

Memory Dump

- `x /nuf [addr]`
  Muestra el contenido de la dirección `[addr]`
- `xp /nuf [addr]`
  Muestra el contenido de la dirección física `[addr]`
  `n` es el número que indica cuantos valores se mostrarán (default 1)
  `u` es el tamaño de la unidad, puede ser[2]:

```
b : byte                h : word (half-word)
w : doubleword(word)    g : quadword(giant word)
```

  `f` es el formato del número, puede ser:

```
x : hex      d : decimal       u : sin signo
o : octal    t : binario       c : char
s : ascii    i : instrucción
```

Memory Disassembly

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

[2]Entre paréntesis el nombre consistente con gdb

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]`