The Bochs Environment

You could use only Bochs (without VirtualBox); in such case this document would help you to build and run it in your environment. If you are using a different emulator or a virtual machine monitor (such as *VirtualBox*), the bochs environment is ready for you, and you will not be using the files described below, except for the floppy image file. You are still running bochs to emulate a system, as described in Step 6. Even if you do not need details about Bochs, I encourage you to read this document to get an idea of the components present in a bare metal (i.e., no virtualization) hardware emulator.

The Bochs Environment

Bochs is a highly portable open source IA-32 (x86) PC emulator written in C++, that runs on most popular platforms. It includes emulation of the Intel x86 CPU, common I/O devices, and a custom BIOS. Bochs can be compiled to emulate many different x86 CPUs, from early 386 to the most recent x86-64 Intel and AMD processors, even those not in the market yet. The ZIP archive that comes with Project 1 (P1) also contains a set of files that define the Bochs emulation environment:

`dev_kernel_grub.img`: This file contains the image of the boot "floppy disk". It contains the GRUB bootloader and a dummy kernel.The boot loader is the first software that runs when a computer testing and hardware initialization has been done. It loads the kernel and then transfers control to it.

`BIOS-bochs-latest`: This file contains the BIOS. The BIOS (more generally known as firmware) is the code that runs on power-on startup. Its main task is to initialize and test the hardware. Every particular computer has a BIOS specifically designed for it. The BIOS also provides basic operations such as I/O until the operating system is ready to take over.

`VGABIOS-lpgl-latest`: This file contains a basic VGA BIOS to manage the graphics card.

`bochsrc.bxrc`: This text file contains the configuration of the emulated machine. If bochs is correctly installed, double-clicking on this file should start the emulator. You can also run Bochs by invoking the program with the configuration file as an argument, as follows:

```
bochs -f bochsrc.bxrc
```

The Bochs Environment with GDB Integration

It is recommended to use an external debugger (`gdb`, the GNU debugger) with the Bochs environment. It would allow you to debug your program more efficiently. `gdb` is the standard debugger on Linux. This is a source level debugger.

It can be used over a serial line by implementing a Remote Serial Protocol stub in your operating system. Bochs can be installed as `/usr/bin/bochs`, `/usr/bin/bochsdbg`, and `/usr/bin/bochs-gdb`. The first one is the non-debugging version (faster emulation). The second version has an internal debugger, allowing you to step through your code in assembly. The third version is configured to use the gdb debugger which allow you to debug in source level and set breakpoints etc.
To use GDB with Bochs, you need to build Bochs with `gdb-stub` enabled. In addition to the information below, the Spring'20 teaching assistant, Osama Toor, recorded a **how-to video** that you may find useful: link to video in google drive.

Step 1: Download Bochs with `gdb` sourcecode from the following location: `https://sourceforge.net/projects/bochs/files/bochs/2.6.8/`

Step 2: Configure Bochs with `gdb` stub enabled, under the directory of the Bochs source code:

```
sudo ./configure --enable-gdb-stub
```

or

```
sudo sh .conf.linux --enable-gdb-stub
```

Step 3: After the configuration, to make it and move it to `/usr/local/bin`, run

```
sudo make
sudo make install
```

If errors related to plugins occur during compilation, remove `-enable-plugins` from `.conf.linux` by uncommenting `which_config=normal` and commenting out `which_config=plugins` in `.conf.linux`.

Step 4: `gdb` can be used to remotely debug the Bochs Environment, where the Bochs emulator acts as a remote host and our Linux machine as the local host. For this to work, we need to enable the `gdb` stub in the Bochs configuration file. This can be done by adding the following line in `bochsrc.brxc` file as shown:

```
gdbstub: enabled=1, port=1234, text_base=0, data_base=0, bss_base=0
```

The port number mentioned can be any number above 1024, as long you connect to the remote target using the same number.

Step 5: A flat binary output file is usually stripped off its debugging information and thus contains only the data part. This makes it impossible to debug using `gdb`. An alternative is to produce an ELF (Executable and Linkable Format) output that retains the debugging information. We do make documentation available with details on how to produce an ELF output.

Step 6: To load up Bochs, run:

```
bochs -f bochsrc.bxrc
```

Bochs will load up and wait for a connection from GDB. To connect from GDB, open a new terminal and run:

```
  gdb YOUR-KERNEL
```

where `YOUR-KERNEL` is the ELF output file.

Helpful Links

- https://www.cs.princeton.edu/courses/archive/fall09/cos318/precepts/bochs_gdb.html
- https://www.cs.princeton.edu/courses/archive/fall09/cos318/precepts/bochs_setup.html
- http://bochs.sourceforge.net/doc/docbook/user/debugging-with-gdb.html
- http://heim.ifi.uio.no/~inf3150/doc/tips_n_tricks/bochsd.html
- http://wiki.yak.net/746