Integrating Bochs Environment with GDB

1. Start with installing GDB debugger: `sudo apt-get install gdb`
2. Debugging a file needs to access its symbol tables and the debugging information.

   The "–g" flag helps in generating the required debugging information to be used by the GDB debugger.

   SYNTAX: `$ gcc –g [options] [source files] [object files] [-o output file]`

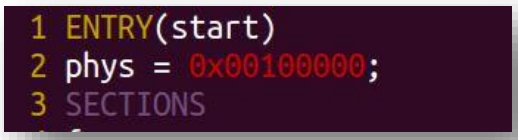
   - The "-g" flag can be added to our existing `makefile` at each of the compile steps to produce an object file containing the debug information. For example,

```
# ==== UTILITIES ====

utils.o: utils.H utils.C
        gcc $(GCC_OPTIONS) -g -c -o utils.o utils.C

# ==== DEVICES ====

console.o: console.H console.C
        gcc $(GCC_OPTIONS) -g -c -o console.o console.C

# ==== KERNEL MAIN FILE ====

kernel.o: kernel.C
        gcc $(GCC_OPTIONS) -g -c -o kernel.o kernel.C
```

3. These object files need to be linked together to produce an executable. Note that, this executable needs to retain the debugging symbols from the object files.
   - In our projects we use a custom linker file (`linker.ld`) to define the different memory segments (viz. Text, Data, BSS).
   - The first line of our linker file defines the Output Format of the executable being produced. It is by default defined to produce a "Binary" Output.

```
1 OUTPUT_FORMAT("binary")
2 ENTRY(start)
3 phys = 0x00100000;
4 SECTIONS
```

- 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. *Side Note: This is also the default output format of the "*`a.out`*" we get on compiling a file using GCC*.
- We can get an ELF output by not mentioning any output format in the linker file. That is, by removing the first line of our `linker.ld` file.



- Extra Notes:
  - To keep the naming of the output file relevant to its format, it is better to rename our output file from "`kernel.bin`" to "`kernel.elf`" or just "`kernel`".
  - This needs to be changed in the appropriate `makefile` and `copykernel` files being used.

4. We can now generate the desired kernel disk image by following the usual steps of compile and copy.

```
$ make –f <make file name>
$ sh copykernel.sh
```

5. 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 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
```

```
# what disk images will be used
floppya: 1_44=dev_kernel_grub.img, status=inserted
#floppyb: 1_44=floppyb.img, status=inserted

# GDB Debugging Stub
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 (more on this below).

6. Loading up Bochs
    - On running,

        `bochs –f bochsrc.bxrc`

        Bochs will load up and wait for a connection from GDB.

```
guest@TA-virtualbox:~/Desktop/proj1/P1$ bochs -f bochsrc.bxrc
========================================================================
                       Bochs x86 Emulator 2.4.6
            Build from CVS snapshot, on February 22, 2011
                  Compiled at Nov 11 2011, 09:31:18
========================================================================
00000000000i[     ] LTDL_LIBRARY_PATH not set. using compile time default '/usr/lib/bochs/plugins'
00000000000i[     ] BXSHARE not set. using compile time default '/usr/share/bochs'
00000000000i[     ] reading configuration from bochsrc.bxrc
00000000000i[     ] Enabled gdbstub
00000000000i[     ] lt_dlhandle is 0x2ec39b0
00000000000i[PLGIN] loaded plugin libbx_x.so
00000000000i[     ] installing x module as the Bochs GUI
00000000000i[     ] using log file bochsout.txt
Waiting for gdb connection on port 1234
```

7. Connecting from GDB
    - Open a new terminal and run,

        `gdb YOUR-KERNEL`

        YOUR-KERNEL is the ELF output file we compiled, i.e. “`kernel.elf`” or “`kernel`”

        It loads up the symbol information contained in your output.
    - GDB cannot automatically detect and support 64 bit architecture in Bochs. So we need to change the architecture in GDB command line using `set architecture` command as follows:

        `(gdb) set architecture i386:x86-64:intel`

- As we are going to debug the kernel running on Bochs, we need to connect to the Bochs target with the command,

  ```
  (gdb) target remote localhost:1234
  ```

  You can alternatively also connect by using the IP address of the computer running Bochs, for example,

  ```
  (gdb) target remote 127.0.0.1:1234
  ```

  1234 is the port number you have chosen to include in `bochsrc.bxrc` file in STEP 5

  This breaks at the first instruction step of your BIOS in Bochs.

- Now, we can set a breakpoint in any part of the kernel code, for example,

  ```
  (gdb) b main()
  (gdb) b kernel.C:35
  ```

- Use the continue (c) command to continue the Bochs simulation,

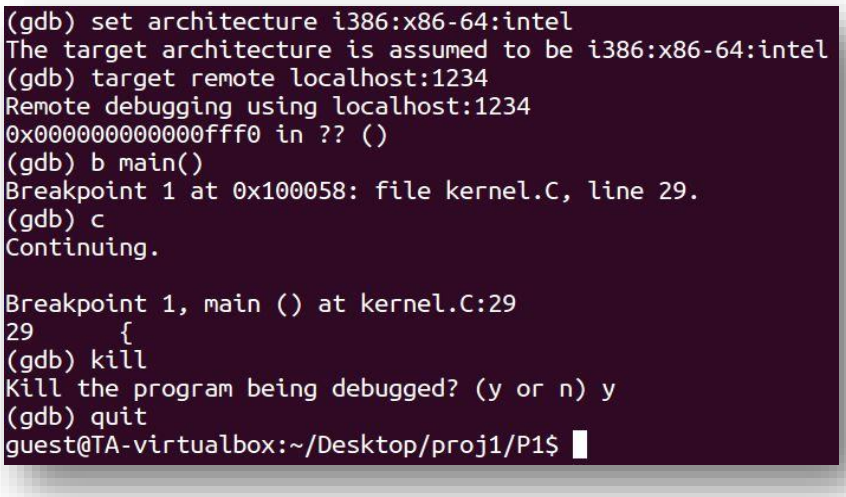
  ```
  (gdb) continue
  ```

  This shows up the main menu in your Bochs Emulator, press Enter to boot up your kernel. It now halts at the breakpoint set by you.

- Debug your kernel code as desired using the GDB debugger.

- You can end the debugging session by running the kill (k) command to terminate the debugging process and exit GDB with the quit (q) command

  ```
  (gdb) kill
  (gdb) quit
  ```



8. We can streamline the above GDB process by creating a file named '`.gdbinit`' in the project directory that contains the following steps:

```
file kernel
set architecture i386:x86-64:intel
target remote localhost:1234
```

   - Thus, running the command

     ```
     $ gdb
     ```

     in our terminal loads up the symbols and connects to the remote host. This helps us from typing these common commands every single time.