1. To use GDB tools with Bochs, first we need to rebuild Bochs with gdb-stub enabled. That's because GDB stub is not active in standard Bochs binary package.

- Get Bochs package here:
  http://sourceforge.net/projects/bochs/files/bochs/2.6.8/
  Unzip it and we got the new source code.

- Before reinstall it, remove the existed Bochs in the Ubuntu by:
  sudo apt-get remove bochs

- Then configure Bochs with gdb stub enabled. Under the directory of the Bochs source code:
  sudo ./configure --enable-gdb-stub
  or, we can use some shortcut script by running:
  sudo sh .conf.linux --enable-gdb-stub

  Note: Enable debugging via gdb by adding --enable-gdb-stub instead of --enable-debugger and --enable-disasm (they are mutually exclusive). And remember to use 'sudo' all the time.

- After the configuration, run
  sudo make
  to compile bochs, and then run
  sudo make install
  to move it to /usr/local/bin

- When compile, if error related to plugins occurs, remove –enable-plugins from .conf.linux by uncommenting '*which_config=normal*' and commenting out '*which_config=plugins*' in *.conf.linux*

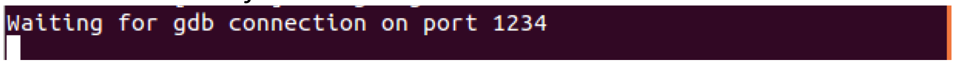
- Also, you may have error related to x11 (X window), just install libraries:
  sudo apt-get install xorg-dev

2. Enable debugging with gdb in the kernel project.

- Add the following line to bochsrc file:
  gdbstub: enabled=1, port=1234, text_base=0, data_base=0, bss_base=0
  This opens a remote gdb debug port.
- Edit the kernel project makefile and add '-g' in the compiler options to enable debugging symbols. Then recompile your project.

3. Now we can run and debug our project.

- Open the bochs tool and it will wait for gdb connection on port 1234
  Bochs –f bochsrc.txt



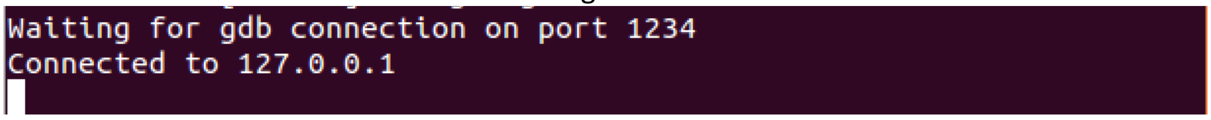
```
Waiting for gdb connection on port 1234
```

- Open another terminal window, run gdb and connect it to Bochs by:

  gdb –q kernel.o

  target remote :1234

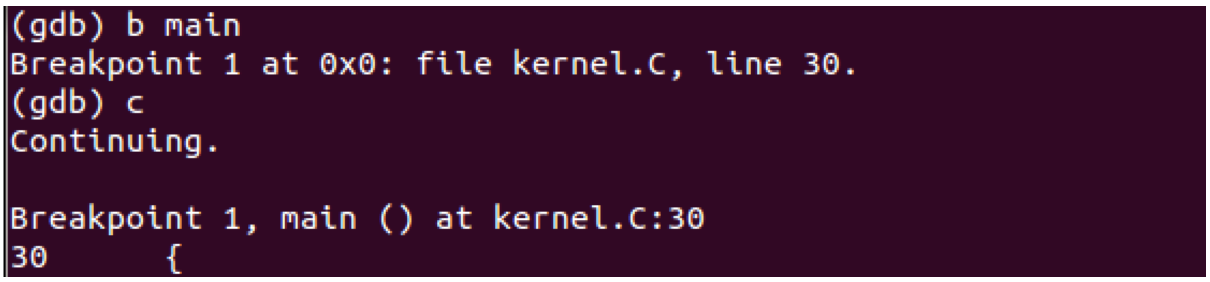
  Then the bochs will be connected to the gdb tool.



```
Waiting for gdb connection on port 1234
Connected to 127.0.0.1
```

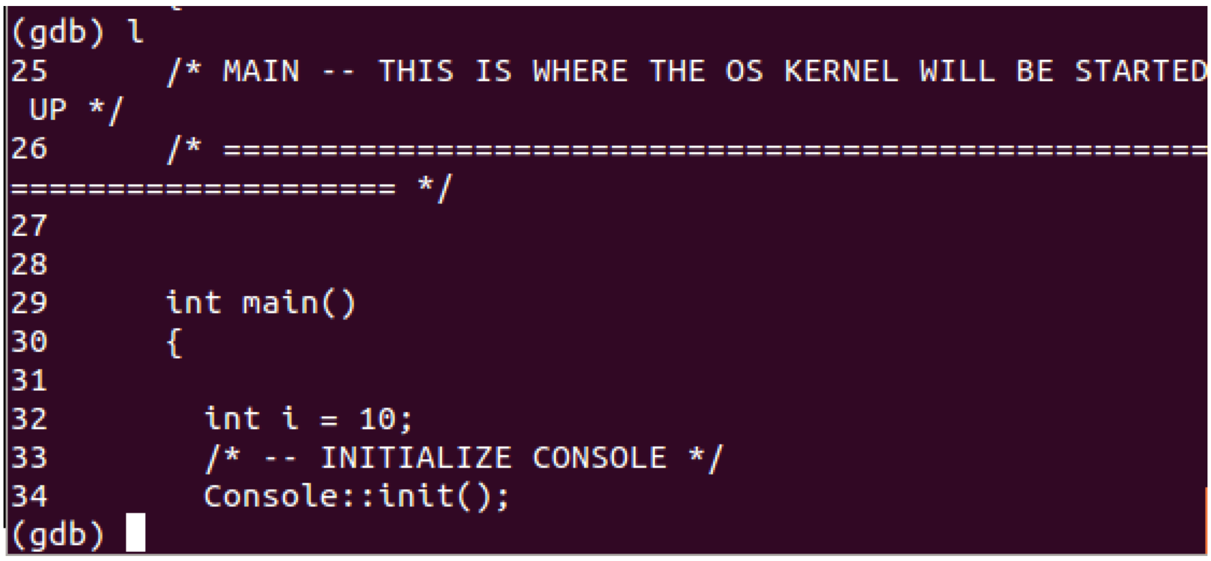
- Now we can "**try**" to debug our project through gdb.

  Upon a successful connection, bochs will break at the first instruction in the BIOS (not the bootloader nor the kernel). Notice that you won't be able to inspect kernel data at this point because your kernel has not yet been loaded by the bootloader.

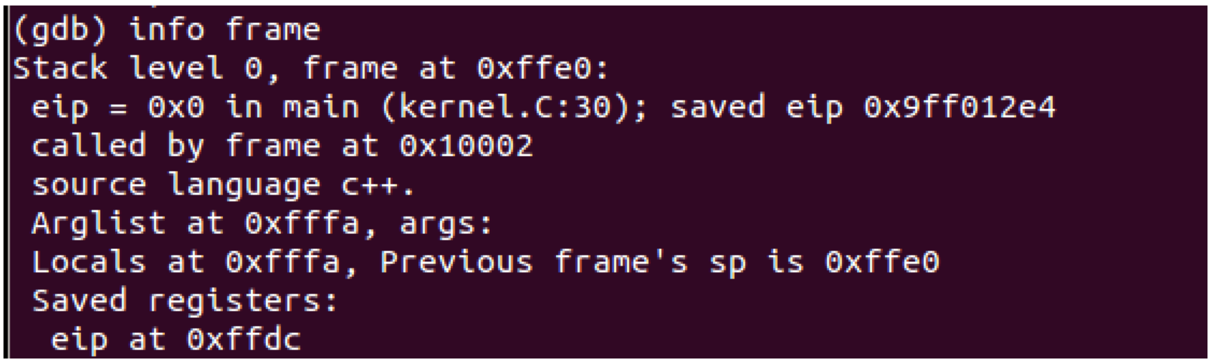


```
(gdb) b main
Breakpoint 1 at 0x0: file kernel.C, line 30.
(gdb) c
Continuing.

Breakpoint 1, main () at kernel.C:30
30        {
```



```
(gdb) l
25        /* MAIN -- THIS IS WHERE THE OS KERNEL WILL BE STARTED
 UP */
26        /* ===========================================================
==================== */
27
28
29        int main()
30        {
31
32          int i = 10;
33          /* -- INITIALIZE CONSOLE */
34          Console::init();
(gdb)
```



```
(gdb) info frame
Stack level 0, frame at 0xffe0:
 eip = 0x0 in main (kernel.C:30); saved eip 0x9ff012e4
 called by frame at 0x10002
 source language c++.
 Arglist at 0xfffa, args:
 Locals at 0xfffa, Previous frame's sp is 0xffe0
 Saved registers:
  eip at 0xffdc
```

Here is the situation:
Our kernel.bin file seems lose symbols which are needed for breakpoint insertion. According to the tutorial http://wiki.yak.net/746 , maybe we have to use ELF(Executable and Linking Format) as the kernel image format.
Still trying to figure it out.

References:
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