The effect of a bug in the kernel is catastrophic and may lead to system crash. Kernel debugging is an important step for kernel developers to find out bugs or debugging kernel execution. Many developers use virtual machine while working on kernel level. QEMU is a famous virtualization tool among the kernel developers. In this tutorial we'll learn kernel debugging using GDB and QEMU. We'll use QEMU to run our guest kernel(the one to debug) and in the host machine we'll run GDB which will connect to QEMU. I'll use 32-bit Ubuntu-12.04 kernel in this tutorial.

For Kernel debugging using GDB & QEMU, we'll need the debug symbols. Default kernel installation doesn't include the debug symbols due to optimization. So we don't have access to everything in default installation. To debug a kernel we've to build it from source code so that the created binary will include the debug symbols. Obviously if you are a kernel programmer, you'll have to do it anyways.

Here I'll assume you know how to build kernel from the source code. So our first step will be to compile the kernel. It will create the kernel image file ***vmlinux*** inside the source directory. This image file will also contain the debug symbols. Copy the ***vmlinux*** file to host machine from the guest machine(QEMU VM). Also copy the **System.map** file which contains the name & address mapping from the kernel. We'll also need the source codes in the host machine. So keep a copy of the source directory in the host machine.

Now we are prepared to start debugging. Open a terminal and start QEMU using any of the command below based on your guest architecture.

```
qemu-system-i386 -hda ~/Ubuntu-12.04.img -s

or

qemu-system-x86_64 -hda ~/Ubuntu-12.04.img -s
```

Here I'm asking QEMU to run my VM image *Ubuntu-12.04.img* from **HOME** directory. Don't forget to add the "***-s***" parameter. It makes QEMU listen on port 1234, which we can connect to from GDB.

In another terminal we'll start GDB. We'll ask it to use the kernel image that we created during the compilation (the ***vmlinux*** file).

```
gdb ~/vmlinux
```

GDB will read and parse the debug symbol from ***vmlinux***. Now GDB is ready to connect to our VM instance. Connect using the command:

```
target remote localhost:1234
```

If everything goes well, you'll notice your VM instance is non-responsive which will mean GDB made a connection to the VM instance successfully. You can now test with some frequently used GDB commands.

Let's try an example. We'll disassemble the **do_fork** function using the **disassemble** command in GDB. This command takes an address to begin. But how do we know the starting address of **do_fork** function? Here comes the use of **System.map** file. This file contains a name-address mapping for all global variables and functions in the kernel. Open it up and search for "do_fork". Check out the a portion of System.map file for my kernel installation-

```
c1058d90 T sys_set_tid_address
c1058dc0 T mm_init_owner
c1058dd0 T fork_idle
c1058e50 T do_fork
c1059030 T kernel_thread
c1059070 T sys_fork
c10590a0 T sys_vfork
c10590d0 T SyS_clone
c10590d0 T sys_clone
```

Note that the starting address of **do_fork** for my VM is 0xC1058e50. So for my case I'll disassemble it using the command:

```
disassemble 0xc1058e50
```

We can also put a break point on this address using:

```
break 0xc1058e50
```

Now finding out addresses from System.map file is tedious. It would be convenient if we could do that using only the function name. In fact that's also possible. Try it:

```
disassemble do_fork
```

Now wouldn’t it be nice if you could execute the source code one line at a time? With GDB, that’s also possible. Add the source code path to GDB using the command:

```
directory ~/linux-kernel-src-directory/
```

Add a breakpoint at **do_fork** function and start debugging again. Once the breakpoint is hit, use **“step”** command for single line execution.

I hope this tutorial will help you start with kernel debugging using GDB and QEMU. For efficient debugging you should learn some useful GDB commands.