kgdb HOWTO

Shakthi Kannan, shaks_wants_no_spam_at_shakthimaan_dot_com

January 5 2007

Revision: 1.2

Abstract

This HOWTO describes the steps followed on setting up a kgdb environment with two machines, and also on how to debug a driver module. Two x86 PCs are used. The testing machine already has a GNU/Linux distribution installed with lilo/grub bootloader. They are connected to a LAN, and communicate with each other using a NULL modem cable. The Linux-2.4.23 kernel is used. This document is released under the GNU Free Documentation License.

1 kernel debugging HOWTO

1.1 Download the sources

The sources need to be downloaded to the development machine. The Linux-2.4.23 kernel sources are downloaded from kernel.org to say, /home/foo:

```
http://www.kernel.org/pub/linux/kernel/v2.4/
```

The kgdb 1.9 patch for the 2.4.23 Linux kernel is downloaded (to say, /home/foo) from:

```
http://kgdb.linsyssoft.com/downloads.htm
```

The gdbmod 1.9 version for module debugging is downloaded from:

```
http://kgdb.linsyssoft.com/downloads.htm
```

Install the gdb that comes with your distribution to use it for kernel debugging. gdbmod version of gdb is required to debug modules that are loaded dynamically on to the testing machine.

1.2 Apply the patch

Extract the kernel sources:

```
cd /home/foo
tar xjvf linux-2.4.23.tar.bz2
```

Change into the linux-2.4.23 directory and apply the patch:

```
cd linux-2.4.23
patch -p1 < /home/foo/linux-2.4.23-kgdb-1.9.patch
```

The patch will enable the -g option in the Makefile for gcc to compile with debug symbols.

1.3 Compile the kernel

Compile the kernel:

```
make dep
make menuconfig
```

Enable the following options for KGDB:

```
CONFIG_KGDB
CONFIG_KGDB_THREAD
CONFIG_GDB_CONSOLE

make dep
make bzImage
```

1.4 Setup the testing machine

Transfer the compiled bzImage and System.map to /boot on the testing machine. Since, I have the two machines connected to the LAN, I use:

```
cd /home/foo/linux-2.4.23
scp System.map root@<ip-address>:/boot/System-map-2.4.23
scp arch/i386/boot/bzImage root@<ip-address>:/boot/vmlinuz-2.4.23-
    kgdb
```

Replace <ip-address> with the IP address of the testing machine.

Update the grub bootloader on the testing machine with the following:

```
title Debian GNU/Linux, kernel-2.4.23-kgdb
root (hd0,1)
kernel /boot/vmlinuz-2.4.23-kgdb ro root=/dev/hdc2 gdb gdbbaud=115200
savedefault
boot
```

Replace hdXY entries appropriately as on your x86 testing machine.

Connect a NULL modem cable between the testing machine and the development machine. My testing machine has a serial port, while my development machine is a laptop, IBM T41, without any serial port. So, I use a USB-to-serial converter and use the pl2303 and usbserial drivers on the development machine. The device file is /dev/ttyUSB0 as can be seen in the "/bin/dmesg" output.

1.5 Starting the communication

Reboot the testing machine and choose the new kgdb-patched kernel. It will boot-up and halt at:

```
Waiting for connection from remote gdb ...
```

Open a terminal in the development machine, su to root, and set the line speeds:

```
su -
Password: <enter-root-password>
stty ispeed 115200 ospeed 115200 < /dev/ttyUSB0
```

Since my USB-to-serial is connected to /dev/ttyUSB0, I use this device file. If you have a serial port, it will be /dev/ttyS0 or /dev/ttyS1. The exact name of the device file can be checked from the output of "/bin/dmesg" after you plug in the USB-to-serial cable.

Now, start the gdb session in the development machine:

```
cd /home/foo/linux-2.4.23
gdb vmlinux
```

Set the target remote connection:

```
(gdb) target remote /dev/ttyUSB0
```

To continue the booting of the testing machine, type continue (or *c*) in the gdb prompt:

```
(gdb) continue
```

You can now use the testing machine. If there is any crash, control will be transferred to gdb. If you would like gdb to get control, you can hit Control+c in the gdb prompt.

You can use the gdb commands to step through or debug the kernel.

2 module debugging HOWTO

The gdbmod version of gdb needs to be used for debugging modules that are loaded dynamically on the testing machine. Follow steps 1-4 as mentioned above in the kernel debugging HOWTO. Only the gdb version used differs here.

Compile the driver sources with the -g option to gcc.

Boot the testing machine with the kgdb-patched kernel. It will boot-up and halt at:

```
Waiting for connection from remote gdb ...
```

The gdbmod-1.9 executable can be placed in /usr/local/bin.

Open a terminal in the development machine, su to root, and set the line speeds:

```
su -
Password: <enter-root-password>
stty ispeed 115200 ospeed 115200 < /dev/ttyUSB0
```

Since my USB-to-serial is connected to /dev/ttyUSB0, I use this device file. If you have a serial port, it will be /dev/ttyS0 or /dev/ttyS1. The exact name of the device file can be checked from the output of "/bin/dmesg" after you plug in the USB-to-serial cable.

Enter into the Linux kernel sources on the development machine, and run gdbmod on the vmlinux file:

```
gdbmod-1.9  vmlinux
```

Set the target remote connection:

```
(gdb) target remote /dev/ttyUSB0
```

You can set the source code search path for the driver sources using:

```
(gdb) set solib-search-path /path/to/driver/sources
```

Continue with the booting on the testing machine, type continue (or *c*) in the gdb prompt on the development machine:

```
(gdb) continue
```

Transfer the driver module to the testing machine. Since, the systems are connected to the LAN, I use scp:

```
scp /path/to/driver.o root@<ip-address>:/root
```

Replace <ip-address> with the IP address of the testing machine.

Now if you load the driver module on the testing machine, it will load. In case of any crash, control will be transferred to gdb. If you would like gdb to get control, you can hit Control+c in the gdb prompt.

Because gdb does not know of the module that is yet to be loaded, you can't put a breakpoint at module_init(). You need to put a breakpoint in the kernel, prior to module init. So, place a breakpoint at module_event().

```
(gdb) br module_event
```

Load the module on the testing machine and it will break at module_event(). Now you can place a breakpoint at your driver module_init (say hello_init), as the kernel now knows about your loaded module:

```
(gdb) br hello_init
```

You can now step through hello_init of your module. Data Display Debugger (DDD) can be used for both gdb and kgdb source-code stepping and debugging.

Have fun!

3 Bibliography

linsysoft . *KGDB Documentation, For versions upto 2.2.* LinSysSoft Technologies Pvt. Ltd. 2005.