Driver Kit Example: Building, Loading, and Debugging

This example shows you how to compile a simple Driver Kit driver, load it, and debug it with GDB. (GDB is a source-level debugger that's described in Chapter 9 of *NEXTSTEP Operating System Software* and in *NEXTSTEP Development Tools and Techniques*.)

You need superuser access to two Intel-based computers to use GDB on your driver. The driver will run on one (the *slave* computer), and GDB will run on the other (the *master* computer).

For information about the Driver Kit, use the Driver Kit target of **/NextLibrary/Bookshelves/NextDeveloper.bshlf**.

Building the example driver

To build the driver, follow these steps:

1) Copy the **myTestDriver** directory to the place you'd like to work in, such as your home directory.

2) Open your copy of **myTestDriver/PB.project**. This opens the driver project in ProjectBuilder. You can browse the Files view to see how the driver files are organized.

3) Change to the Builder view by clicking the rightmost button in the top of the ProjectBuilder window.

4) Make sure the Target pop-up list is set to **bundle**, and click the Build button. ProjectBuilder builds the driver and the user-level **tester** tool, and puts them into a configuration bundle at **myTestDriver/myTestDriver.config**.

You must perform one more step to build a debuggable version of the driver:

5) In a Terminal window, change to the directory **myTestDriver/myTestDriver_reloc.tproj**, make a **debug** version of the driver, and copy the relocatable file into **myTestDriver/myTestDriver.config**.

```
slave> make debug
slave> rm ../myTestDriver.config/myTestDriver_reloc
slave> cp myTestDriver_reloc ../myTestDriver.config
```

Loading the example driver

To load the driver:

1) Copy the driver bundle into the **/usr/Devices** directory of the slave computer.

```
slave> su
slave# "rm" -r /usr/Devices/myTestDriver.config
slave# cp -r ../myTestDriver.config /usr/Devices
```

2) Load the driver and initialize it using **driverLoader** and, optionally, **kl_util**.

   To load and initialize the driver the simple way:

```
slave# driverLoader d=myTestDriver
Answer queries with 'y' for 'yes', anything else is 'no'.
 Load driver myTestDriver? y
Using Default table for myTestDriver
Configure driver myTestDriver unit 0? y
slave#
```

   To load and initialize the driver so that you can set breakpoints in initialization code (setting breakpoints is described later):

```
slave# kl_util -a /usr/Devices/myTestDriver.config/myTestDriver_reloc
Adding server with relocatable
   /usr/Devices/myTestDriver.config/myTestDriver_reloc
/usr/Devices/myTestDriver.config/myTestDriver_reloc is thin mach-o
Allocating server myTestDriver
Server myTestDriver linking
   /usr/Devices/myTestDriver.config/myTestDriver_reloc against /mach
Server myTestDriver linking relocatable
   "/usr/Devices/myTestDriver.config/myTestDriver_reloc" into loadable
   "/usr/Devices/myTestDriver.config/myTestDriver_loadable"
Server myTestDriver Allocated
slave# kl_util -l myTestDriver
Server myTestDriver loading
state.eip = 15cd80
Server myTestDriver download complete
Server myTestDriver starting up
Server myTestDriver Loaded

...Attach the debugger and establish breakpoints, as described later...

slave# driverLoader d=myTestDriver
```

Important: Driver Kit drivers can't be unloaded and then reloaded. To reload a Driver Kit driver, you must restart the computer.

4) The driver's output appears in the system message log.

```
slave> tail -2 /usr/adm/messages
Aug 18 18:02:50 slave mach: myTestDriver: interrupt 2 channel 1
```

```
Aug 18 18:02:50 slave mach: Registering: myTestDriver0
```

If your driver didn't load, make sure you followed all the above steps properly. **/usr/adm/messages** may contain hints at the reason the driver failed to load. Specifying the verbose option to **driverLoader** (**driverLoader d=myTestDriver v**) may produce some useful output, as well.

Debugging the example driver with GDB

Note: In 3.2, to use GDB to debug a kernel, the slave machine needs to have an Ethernet card that's controlled by a NeXT driver. Non-NeXT drivers don't currently contain support for GDB.

To debug the example driver with GDB, follow these steps:

1) Copy the loadable object file from **/usr/Devices** into a place accessible to the master. You need to do this again every time the driver is loaded.
   `slave> cp /usr/Devices/myTestDriver.config/myTestDriver_loadable ~`

2) If the master computer is running the same kernel as the slave, change to the root directory. Otherwise, copy the slave's kernel to a place where the master can read it, and change to that directory.

```
master> cd /
```

3) Update the network entry for the slave by running the **ping** command. Press Control-C after a few seconds. If the output of **ping** shows 100% packet loss, contact your system administrator.

```
master> /etc/ping slave
PING slave: 56 data bytes
64 bytes from 129.18.2.98: icmp_seq=0. time=34. ms
64 bytes from 129.18.2.98: icmp_seq=1. time=3. ms
<Control-C>
----slave PING Statistics----
2 packets transmitted, 2 packets received, 0% packet loss
round-trip (ms)  min/avg/max = 3/18/34
```

4) Start GDB, specifying the kernel's file.

```
master> gdb mach
GDB is free software and you are welcome to distribute copies of it
 under certain conditions; type "show copying" to see the conditions.
There is absolutely no warranty for GDB; type "show warranty" for details.
GDB 4.7 (NeXT 3.1), Copyright 1992 Free Software Foundation, Inc...
Reading symbols from /mach...(no debugging symbols found)...done.
```

5) On the slave computer, open the Kernel Debugger window by holding down both Alternate keys and pressing the NumLock key.

6) On the master computer, enter the **kattach** command.

```
(gdb) kattach slave
Attaching program: /mach to kernel on slave.
```

```
0x16e6eb in thread_io_bmap ()
(gdb)
```

Tip: If **kattach** fails, see the troubleshooting section below for help.

7) Bring the symbol information from the loadable file into GDB.

```
(gdb) add-file /Net/users/jsmith/myTestDriver_loadable
Reading symbols from   /Net/users/jsmith/myTestDriver_loadable...done.
(gdb)
```

8) Tell GDB where the driver's source files are.

```
(gdb) dir /Net/users/jsmith/myTestDriver
Source directories searched:     /Net/users/jsmith/myTestDriver://:$cdir:
$cwd
(gdb)
```

9) Set breakpoints and then continue the slave kernel's execution.

```
(gdb) b panic
Breakpoint 1 at 0x10d63c
(gdb) b getCharValues:forParameter:count:
The following classes implement getCharValues:forParameter:count::
1) -EventDriver           4) -IODevice              7) -IOVPCodeDisplay
2) -EventSrcPCKeyboard    5) -IODisplay             8) -myTestDriver
3) -EventSrcPCPointer     6) -IOFrameBufferDisplay
Which one do you want? 8
Reading in symbols for myTestDriver.m...done.
Breakpoint 2 at 0x1f90996: myTestDriver.m:85.
(gdb) c
```

```
Continuing.
```

On the slave machine, you can run the user-level program to exercise the driver (**/usr/Devices/myTestDriver.config/tester**).

10) To stop the slave's kernel (and get a "(gdb)" prompt) without hitting a breakpoint, generate a nonmaskable interrupt on the slave computer by pressing both Alternate keys and the NumLock key.
```
Program received signal 5, Trace/BPT trap
0x156e7a in idle_thread ()
(gdb)
```

11) To detach from the slave machine, first get a "(gdb)" prompt. Then enter the **kill** command and quit GDB.
```
(gdb) kill
Kill the inferior process? (y or n) y
(gdb) quit
master>
```

Note: The **kill** command does *not* kill the slave's kernel; in the future, another GDB command will likely be used for detaching from the slave kernel.

Troubleshooting

This section describes problems you might encounter when following the instructions in this document. Some of these problems might already be corrected in this release.

GDB won't attach to the slave computer.

This is usually caused by one of these three reasons:

- The slave computer doesn't have the Kernel Debugger window open. To open the Kernel Debugger window, hold down both Alternate keys and press the NumLock key.

- Another copy of GDB is attached to the slave. Use **ps -aux | grep gdb** to find this other copy; you should terminate it with the **kill** command.

- The network entry for the slave has expired. To update the network entry, exit GDB as described below, and use the **ping** command as described in step 3 of the debugging section. You can retain the network entry until the master reboots by using the **arp** command, as follows:

```
master# arp slave
slave (129.18.2.98) at 0:aa:0:18:5c:3d
master# arp -s slave 0:aa:0:18:5c:3d
```

  You can make sure the **arp -s** command worked by running **arp *host*** again and checking for "permanent" in the output:

```
master# arp slave
slave (129.18.2.98) at 0:aa:0:18:5c:3d permanent
```

GDB is attached to the slave computer, but you can't get a "(gdb)" prompt.

If the slave computer is unresponsive to Alternate-Alternate-NumLock, restart it by turning its power off and then on again. (Checking the disks will take a few minutes.) To quit GDB, type Control-Z, enter **bg** to see which background process GDB is, and then enter **kill %*n***, where ***n*** is the number shown by **bg**:

```
Control-Z
Stopped
master> bg
[1]    gdb mach &
master> kill %1
```