SDB User's Guide

Version 1.0
Feb. 17, 1999

About Sdb

The Spruce Fir Moss Spider Debugger (SDB) is a debugger for programs developed with the Titanium reference compiler (tc/tcbuild). Based upon gdb v4.17, this debugger retains all of the original's functionality in debugging various languages, platforms, and thread packages. Additional support was added to simplify debugging Titanium programs, such as accessing Java and Titanium arrays as well as displaying Domains, RectDomains, and Points.

Currently some features of Titanium cannot be debugged with this implementation. These will be addressed soon. They include debugging wide pointers (NOW, smp-wide), modifying variables on the fly, calling functions in the executable from the command line, and runtime type casting when printing variables.

Features

Language Support

Gdb supports cross language debugging though language commands and vaious language specifications built into it. A new gdb language type called *titanium* has been added. When Titanium programs are debugged, the debugger language is automatically set to this value. Once the debugger language is set for Titanium, the command line behaves like a mini-interpreter. So, a subset of the commands in the Titanium language is available. One can access arrays and variables (formals, automatics, and object fields) in the same format as in the Titanium language itself. Additionally, the command line supports creating Titanium Point<N> classes in the debugger to more easily access arrays.

The debugger currently works with the uniproc (the default), smp-narrow, and pthread backends of the Titanium compiler's runtime systems. Sdb will debug the latter two only if gdb 4.17 supports the debugging of threads—currently includes Sparc Solaris. A version for Linux threads is forthcoming.

Commands

No additional commands have been added. Multi-programming is supported through gdb's thread commands. Some Titanium specific commands, however, will be added to better support multiprogramming.

Threads

Titanium processes on the supported backends use thread libraries. So, debugging multiple processes is available only if debugging threads is available on your platform. Sdb is based upon a version of Gdb 4.17 that supports debugging threads on (at least) Solaris and Linux platforms. On Linux platforms, please consult the URL:

- http://www.gr.opengroup.org/java/jdk/linux/debug.htm

to verify that you have the supported library versions.

Tutorial

Basics

First, you need to build your Titanium program with the *–g* option to turn on support for debugging and also (recommended) turn off optimizations with “dash Oh-Zero”.

```
$ tcbuild --tc-flags “-O0” -g myProgram.ti
```

This results in the executable `myProgram` being created using the *uniproc* backend. This backend allows single threaded programs to run. You can now run Sdb on this executable to debug it. Here is an example.

```
$ gdb myProgram
SPRUCE-FIR MOSS SPIDER Debugger sdb 0.01
[ … ]
(sdb)
```

Make sure that you are picking up the correct executable (the name in the file system for the Sdb program itself is still `gdb`, so make sure you see the first line stating that this is actually Sdb as well as the prompt). Verify that the language is correctly set to *titanium* by using the `show` command. Also, the `set` command can be used to manually set the language to the desired setting.

```
(sdb) show language
The current source language is “auto; currently titanium”.
(sdb) set language titanium
(sdb)
```

Now, set a breakpoint on the `main` function of the program and start the execution. Note that before execution starts, setting a breakpoint on *main* will find the entry point of the program as determined by the Titanium compiler.

```
(sdb) break main
Breakpoint 1 at 0x338b4: file myProgram.ti, line 16.
(sdb) run
Starting program: myProgram

Breakpoint 1, Main.main (args=0x197f80) at myProgram.ti:16
16          int aNumber;
(sdb)
```

Set a breakpoint based on the file name and line number and continue to skip over the initializations

```
(sdb) break myProgram.ti:33
Breakpoint 2 at 0x33dd0: file myProgram.ti, line 33.
(sdb) continue
Continuing.

Breakpoint 2, Main.main (args=0x197f80) at myProgram.ti:33
33          aPoint = [2];
(sdb)
```

Display some program values.

```
(sdb) print aNumber
$1 = 5
```

```
(sdb) print object
$2 = (MyClass) 0x1a16a8 (java.lang.Object.PrivateInfo = 0,
  MyClass.x = 1, MyClass.y = 2)
(sdb) print object.x
$3 = 1
(sdb)
```

Look at the Java array created and index its values.

```
(sdb) print java_array
$4 = (int []) 0x19dd98 int[5]
(sdb) print java_array[2]
$5 = 2
(sdb)
```

Do the same for the Titanium array.

```
(sdb) print ti_array
$6 = (MyClass [1d]) 0xeffff8a8 MyClass[[[1]:[5]:[1]]]
(sdb) print ti_array[[5]]
$7 = (MyClass) 0x1a1600 (java.lang.Object.PrivateInfo = 0,
  MyClass.x = 1, MyClass.y = 2)
(sdb)
```

Notice that if you give an out-of-bounds access, the debugger will catch it.

```
(sdb) print java_array[200]
Index value of 200 is out of bounds while accessing a Java array of
size 5
(sdb)
```

Step to give `aPoint` a value, then use it to index the Titanium array.

```
(sdb) step
35          System.out.println("Done.");
(sdb) print ti_array[aPoint]
$8 = (MyClass) 0x1a1648 (java.lang.Object.PrivateInfo = 0,
  MyClass.x = 1, MyClass.y = 2)
(sdb)
```

Finally, complete the execution and quit.

```
(sdb) continue
Continuing.
Done.

Program exited normally.
(sdb) quit
```

Threaded Backends

Currently, the *smp-narrow* (and likely the *pthread*) backends of the Titanium compiler are supported by Sdb for multiprogramming. These backends use either the Solaris or pthread packages for Titanium processes. Information about the threads can be gotten from the *info* command.

If, for example, a program had two processes, you may see (from a Solaris OS using the *smp-narrow* backend):

```
(sdb) info threads
  12 Thread 5 (LWP 4)  Main.main (args=0x1caf00) at myProgram.ti:16
  11 Thread 4           0xef77759c in _swtch ()
  10 Thread 3 (LWP 2)  0xef6b9838 in _signotifywait()
  9 Thread 2           0xef77759c in _swtch ()
  8 Thread 1           0xef6b96f0 in _lwp_sema_wait ()
* 7 Thread 6 (LWP 5)  Main.main (args=0x1caf10) at myProgram.ti:16
[ … stuff about LWPs … ]
(sdb)
```

This shows the two Titanium process threads—*Thread 5* and *Thread 6*. The other entries can be ignored. Because the threads are created in order of process numbers, *Thread 5* is Titanium process 0, while *Thread 6* is process 1. The * next to the *7* indicates that *Thread 6* (Titanium process 1) is the current thread being viewed.

To see what is happening with Titanium process 0, switch the current thread using the *thread* command.

```
(sdb) thread 12
[Switching to Thread 5 (LWP 4)]
#0  Main.main (args=0x1caf00) at myProgram.ti:16
16          int aNumber;
(sdb)
```

Future versions of Sdb will have specific command to help in this process.

Requests

Please send all requests to miyamoto@cs.berkeley.edu.