Android+Eclipse GDB debugging for the Tegra 200 series devkit

Version 0.1

Contents

Introduction

This document describes a method of debugging Native Development Kit (NDK) based applications from within Eclipse. While only the gdbserver and arm-eabi-gdb binaries are required to debug native code on Android, we've found it useful to have debugging integrated with Eclipse.

It is assumed that the user has a basic familiarity with Android, GDB, Eclipse, the Tegra development kit and is already using Eclipse for creating Android NDK applications. Please read the other documents available at http://developer.nvidia.com/tegra/ and ask in the forums if this is not the case.

Note: On Windows, any time the word "shell" is mentioned this is referring to a Cygwin command shell.

Note: <install path> is referring to the directory where the archive was extracted to

Preparations

Installation

1. Unpack the compressed archive, make sure your adb connection is working and go to the created "gdb" directory in a command shell and run the install.sh script:

   ```
   cd <install path>/gdb
   ./install.sh
   ```

   This script will copy the gdbserver executable to the device and pull the device libraries to a local directory to enable symbol lookups where available.
2. Create or edit your ~/.gdbinit file and add this line:

   ```
   set solib-search-path <install path>/gdb/lib
   ```

   This will ensure that the debugger can find the device libraries for symbol lookups when available.

3. Ensure that you can run the host side debugger by executing the debugger stand alone. For example on windows:

   ```
   ./prebuilt/windows/arm-eabi-gdb.exe
   ```

If you have all the dependencies installed, you will be greeted by the gdb prompt, looking similar to the following:

```
GNU gdb (GDB) 7.0.50.20100202
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-pc-cygwin --target=arm-eabi-linux".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
(gdb)
```

If there are missing dependencies, you'll get an error such as the following:

```
./prebuilt/windows/arm-eabi-gdb.exe: error while loading shared libraries: cygncurses-
10.dll: cannot open shared object file: No such file or directory
```

This indicates that the ncurses-10 dependency is missing and will need to be installed before it's possible to execute the debugger. If there is no output at all, the command "ldd" can be used. Example output of the command is:

```
ldd ./prebuilt/windows/arm-eabi-gdb.exe | grep no
        cygncurses-10.dll => not found
```

Alternatively, the following can be executed:

```
which `objdump -p ./prebuilt/windows/arm-eabi-gdb.exe | grep DLL | awk '{print $3}'` |
grep no
```

Upgrading to the latest version of Cygwin is recommended in case there are dependency issues.

Eclipse debug target setup

1. Start up Eclipse

> **Note:** If using Windows as the host OS, this must be done from a Cygwin shell so that the Cygwin environment is properly set up. If this isn't done, the native debugger will be unable to start.

2. Select the menu item Run->Debug Configurations
3. If you haven't debugged your application as an Android application before:
   3.1. Select "Android Application" and press the new launch configuration button
   3.2. Select the "<project name>" configuration that just got created
   3.3. Press the "Browse" button and select your project
4. Select C/C++ Application and press the new launch configuration button
5. Select the "<project name> Default" configuration that just got created
6. Specify your project's .so in the "Main" tab under C/C++ Application.

7. In the tab "Debugger"
   7.1. Change the “Stop at startup at” function to something your application uses or disable the feature by using the checkbox
   7.2. Select the "gdbserver Debugger" in the drop down list
   7.3. Sub-tab "main"
      7.3.1. Set the “GDB debugger” to <install path>/gdb/prebuilt/<flavor>/arm-eabi-gdb, where <flavor> is the operating system you are using. Please see the <install path>/gdb/prebuilt/ for the available flavor options.
      7.3.2. Set the “GDB command file” to your previously created ~/.gdbinit
      7.3.3. If using Windows as the host OS, set the “GDB command set” to “CygWin”
   7.4. Sub-tab Connection
      7.4.1. Set type to TCP
      7.4.2. Set Hostname to localhost
      7.4.3. Set port number to 12345
   7.5. Apply the settings and close the Debug Configurations window
8. If using Windows, select the menu item “Window->Preferences”.
   8.1. Select “C/C++ -> Debug -> Common Source Lookup Path” and click “Add”
   8.2. Select “Path Mapping” and click “ok”
   8.3. Click add
   8.4. Under “Compilation path”, type “\cygdrive\c” and under “Local file system path” select “C:”. This ensures that gdb and Eclipse are on the same page with regards to source code. If your source code resides on other drives than C:, please add those drives too in a similar manner.

Debugging

Once everything has been set up, you are ready to start debugging.

1. Set a breakpoint somewhere in your java init code
2. Start debugging your app by right clicking on the project in the Project Explorer and selecting Debug As -> Android Application
3. When the java breakpoint is hit
   3.1. Go to <install path>/gdb directory in a command shell and run the debug.sh script to attach gdbserver to the already running process. The syntax of the script is

```
./debug.sh /path/to/your/library.so java.package.name.of.your.activity
```

   For example if you keep your project in /code/mydemo and the fully qualified class name of your activity is com.mycompany.mydemo.MyDemo, then you would run:

```
cd <install path>/gdb
./debug.sh /code/mydemo/libs/armeabi/libmydemo.so com.mycompany.mydemo
```

On a successful execution of the script, the output will look similar to the following

```
Attached; pid = 1095
Listening on port 12345
```

If this is not the case, make sure that the application is launched and running and that you are using the correct parameters to the script.

3.2. Once gdbserver has attached, use the "Run->Debug Configurations" menu item to go back to your previously created C/C++ Application debug configuration and press the debug button.
3.3. Wait, it'll take a while. After some time gdb will be attached and ready to rock. If you get the error "Error creating session", it is usually due to not starting eclipse from a cygwin shell on Windows. If you did and still get the error, try executing the arm-eabi-gdb binary in a shell and resolve any errors that might show up doing so.

Frequently Asked Questions

Why won't the debugger break into my C/C++ code?

Make sure that the GDB debugger is attached after your native library has been loaded with the System.loadLibrary call, your source code is compiled with debug information enabled (compile flag "-g") and that the debug information has not been stripped out of the binary.

Setting breakpoints in C/C++ code can only be done before the application has started or while the GDB debugger is examining a breakpoint, not while the application is running.

If you still cannot debug, go to your GDB Debug Configuration and in the "Debugging" tab, toggle the "Verbose Console Mode" option in the "Main" sub-tab and use the console output to solve your debugging problem or post it in the forums for further help.

NVIDIA Corporation
2701 San Tomas Expressway
Santa Clara, CA 95050
www.nvidia.com