Compiling, Installing, and Running MIPS NDK Native Applications

Document Number: MD00885
Revision 01.00
September 1, 2011

Contents

1 Introduction

The MIPS Android Native Development Kit (NDK) provides a development environment that allows programmers to build Android applications in native C or C++. NDK for the MIPS® architecture streamlines the entire build process for fast native development. For detailed information, see `http://developer.android.com/sdk/ndk/index.html`.

1.1 NDK Versions

The Android NDK supports *mips* and *mips-r2* ABIs.

The following package was used for the example applications in this document:

```
android-mips-ndk-r5b-linux-x86.tar.bz2
```

mips: MIPS32 Release 1 instruction set, Little Endian, hardware floating-point.

mips-r2: MIPS32 Release 2 instruction set, Little Endian, hardware floating-point.

Big Endian and software floating-point are not supported.

The NDK package can be downloaded from `http://developer.mips.com/android/download-android-ndk`.

1.2 System and Software Requirements

The example applications in this document use the following software packages:

1. Ubuntu 32-bit 10.04
2. `JDK 5` (`jdk-1_5_0_22-linux-i586.bin`)
3. `android-mips-ndk-r5b-linux-x86.tar.bz2`
4. `android-sdk_linux-x86_froyo.tar`

All tools are installed in `/home/vlinux/tool`:

`/home/vlinux/tool`:	tool root directory
`/home/vlinux/tool/ndk`:	all versions of NDK
`/home/vlinux/tool/sdk`:	all versions of SDK
`/home/vlinux/tool/jdk`:	all versions of JDK

1.2.1 System packages

The example applications described in this document are tested on 32-bit Ubuntu 10.04. with the following packages installed:

```
sudo apt-get update
sudo apt-get install libgmp3-dev libgmpxx4ldbl
sudo apt-get install ant1.8
```

1.2.2 JDK 5

As of July 2011, Ubuntu no longer supports JDK 5. To install JDK 5 manually, download the JDK 5 update 22 from the following website:

```
http://www.oracle.com/technetwork/java/javase/downloads/index-jdk5-jsp-142662.html
```

`jdk-1_5_0_22-linux-i586.bin` is used for the applications in this document. To install the package:

```
$ mkdir -p /home/vlinux/tool/jdk
$ cd /home/vlinux/tool/jdk
$ ./ jdk-1_5_0_22-linux-i586.bin
```

Accept the agreement to start the installation process:

```
Sun Microsystems, Inc.  Binary Code License Agreement

for the JAVA 2 PLATFORM STANDARD EDITION DEVELOPMENT KIT 5.0

SUN  MICROSYSTEMS,  INC.  ("SUN") IS WILLING TO LICENSE  THE
SOFTWARE  IDENTIFIED  BELOW TO YOU ONLY  UPON THE  CONDITION
THAT YOU ACCEPT ALL OF THE TERMS  CONTAINED  IN THIS  BINARY
CODE  LICENSE  AGREEMENT  AND  SUPPLEMENTAL   LICENSE  TERMS

:
message cut
:

For inquiries please contact:  Sun Microsystems,  Inc., 4150
Network  Circle,  Santa  Clara,   California  95054,  U.S.A.
(LFI#143333/Form ID#011801)

Do you agree to the above license terms? [yes or no]
yes

Unpacking...
Checksumming...
0
0
Extracting...
UnZipSFX 5.50 of 17 February 2002, by Info-ZIP (Zip-Bugs@lists.wku.edu).
   creating: jdk1.5.0_22/
   creating: jdk1.5.0_22/jre/
   creating: jdk1.5.0_22/jre/bin/
  inflating: jdk1.5.0_22/jre/bin/java
  inflating: jdk1.5.0_22/jre/bin/keytool
```

By default, it creates a subdirectory named `jdk1.5.0.22`.

```
vlinux@ubuntu10:~/jdk$ ls
jdk1.5.0_22
vlinux@ubuntu10:~/jdk$ cd jdk1.5.0_22/
vlinux@ubuntu10:~/tool/jdk/jdk1.5.0_22$ pwd
/home/vlinux/tool/jdk/jdk1.5.0_22
```

1.3 SDK and NDK Installation

To install SDK and NDK, untar the package (this example tests Froyo PR5):

```
$ mkdir -p /home/vlinux/tool/ndk
$ cd /home/vlinux/tool/ndk
$ tar -xzf android-mips-ndk-r5b-linux-x86.tar.bz2

$ mkdir -p /home/vlinux/tool/sdk
$ cd /home/vlinux/tool/sdk
$ tar -xf android-sdk_linux-x86_froyo.tar
```

1.4 System Variables

To use the NDK, both the SDK and NDK system variables must be set. A `source.me` file is created in `/home/vlinux/tool` with the following setup:

```
#!/bin/sh
export NDK=`pwd`/ndk/android-mips-ndk-r5b-linux
export PATH=$NDK/toolchains/mips-linux-android-4.4.3/prebuilt/linux-x86/bin:$PATH
export SDK=`pwd`/sdk/android-sdk_eng.wipromips3_linux-x86
export PATH=$SDK/tools:$PATH
export PATH=$SDK/platform-tools:$PATH
export PATH=/home/vlinux/tool/jdk1.5.0_22/bin:$PATH
```

To setup the environment variables:

```
$ cd /home/vlinux/tool/
$ source source.me
```

1.5 Android Emulator Setup

A binary image included with SDK can be used with the emulator. The first step is to determine the available target:

```
$ android list targets
Available Android targets:
id: 1 or "android-8"
     Name: Android 2.2.1
     Type: Platform
     API level: 8
     Revision: 2
     Skins: WQVGA432, WVGA854, WQVGA400, HVGA, QVGA, WVGA800 (default)
```

This SDK is for Froyo and supports API=8. The target ID can be used when creating the new AVD configuration file.

The second step is to create an AVD configuration. In this example, an AVD configuration named `keng` is created in `/home/vlinux/img1`.

```
vlinux@ubuntu10:~$ android create avd -n keng -p /home/vlinux/img1 -t 1
Android 2.2.1 is a basic Android platform.
Do you wish to create a custom hardware profile [no] no
Created AVD 'keng' based on Android 2.2.1, with the following hardware config:
vm.heapSize=24
hw.lcd.density=240
```

```
vlinux@ubuntu10:~$ ls -l /home/vlinux/img1/*
-rw-r--r-- 1 vlinux vlinux     147 2011-08-20 02:58 /home/vlinux/img1/config.ini
-rw-r--r-- 1 vlinux vlinux 1522752 2011-08-20 02:58 /home/vlinux/img1/userdata.img
```

The following variables are used to create a new AVD configuration file:

`-n:` specifies configuration name
`-p:` specifies where the configuration is stored
`-t:` specifies the target

To check what AVD configurations are available in the system:

```
$ android list avd
Available Android Virtual Devices:
    Name: keng
    Path: /home/vlinux/img1
  Target: Android 2.2.1 (API level 8)
    Skin: WVGA800
```

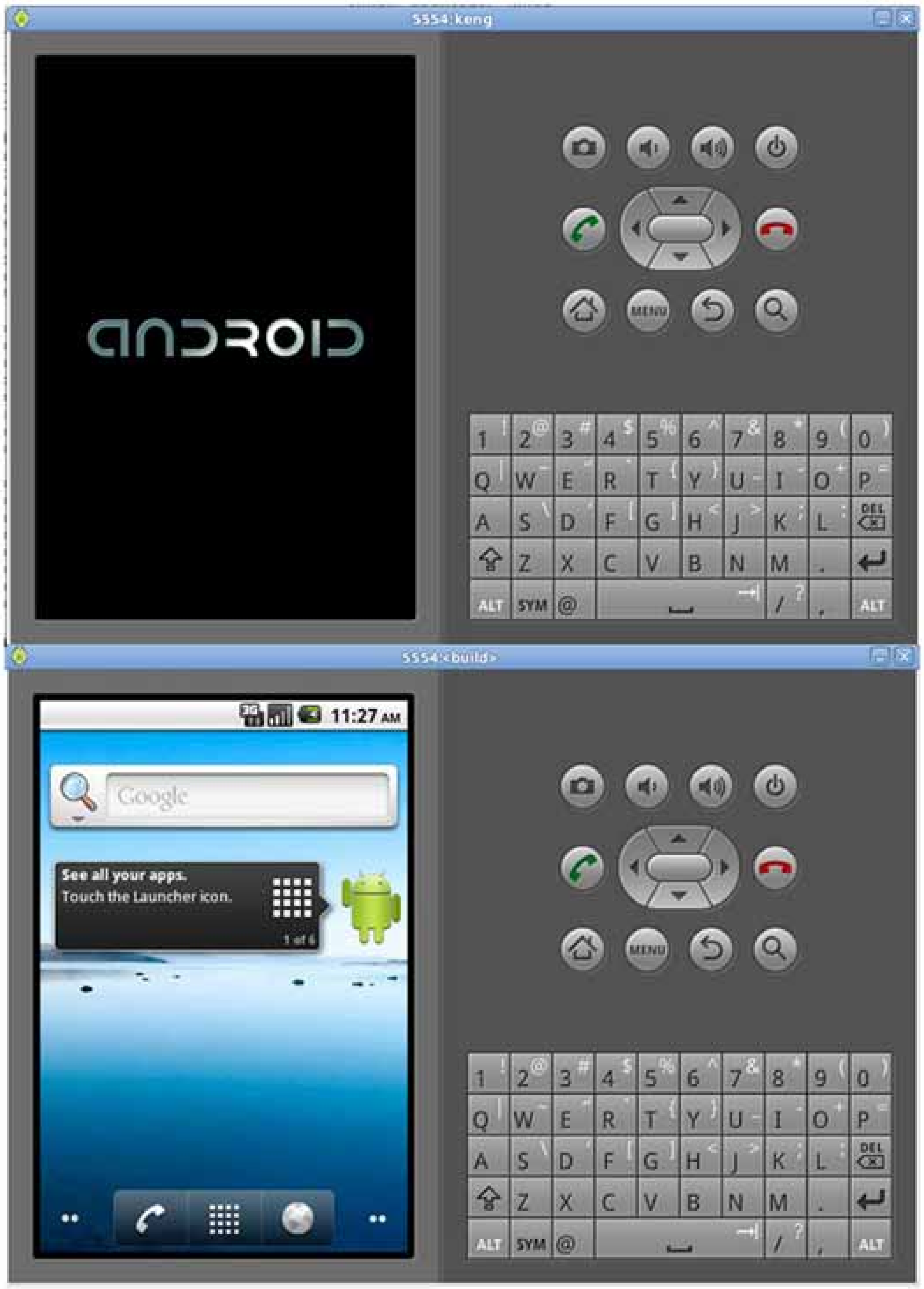
To delete ab AVD configuration:

```
$ android delete avd -n keng
Deleting file /home/vlinux/.android/avd/keng.ini
Deleting folder /home/vlinux/img1
AVD 'keng' deleted.
```

To start the emulator with the configuration:

```
$ emulator -avd keng
```

The emulator will be launched as shown below.


5554:keng
android
5554:<build>
11:27 AM
Google
See all your apps.
Touch the Launcher icon.
1 of 6

2 Simple NDK Example Applications

This section provides example applications that build and test a variety of NDK applications. In the code listings shown, text in blue is entered by the user, and text in red shows changes from previous code.

2.1 Hello World without Android makefile

The following example compiles a simple standalone application without using the Android makefile and GUI.

2.1.1 Build native standalone application

The example below assumes that the example code is located in `/home/vlinux/ex/ndk.hello`.

```
~/ex/ndk.hello$ ls
c1.sh  c2.sh  crtbegin_dynamic.o  crtbegin_static.o  crtend_android.o  lib1.c
README  test  test1.c  test.c
```

The NDK compiler functions similarly to the regular GNU C compiler. First create a simple HelloWorld in C code in `test.c`:

```
#include <stdio.h>

int main() {
   printf("hello\n");
   return 0;
}
```

To compile the code, use the following commands:

```
$ source ~/source.me
$NDK/toolchains/mips-linux-android-4.4.3/prebuilt/linux-x86/bin/mips-linux-android-
gcc \
-I$NDK/platforms/android-8/arch-mips-r2/usr/include \
-L$NDK/platforms/android-8/arch-mips-r2/usr/lib \
-Wl,-rpath-link=$NDK/platforms/android-8/arch-mips-r2/usr/lib \
--sysroot=$NDK/platforms/android-8/arch-mips-r2 \
-EL -g -mhard-float -o test test.c -llog -lm
```

These commands will generate a binary program named `test`.

2.1.2 Run native application

To run the example, two consoles are required. The first console will be used to start the emulator. The second console will be used to connect, upload, and run the application.

On the first console, do the following:

```
$ source ~/source.me
$ emulator -avd keng -shell
```

The `-avd` option will load the android virtual device configuration name `keng`.

The `-shell` option command directs the console to enter the emulator shell.

After Android has completed the loading process (and can activate the slide menu to launch the application), start the second console to connect to the Android debug service:

```
$ source ~/source.me
$ adb kill-server
$ adb start-server
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
vlinux@ubuntu10:~/tool$
```

To test the android debug service, do the following:

```
vlinux@ubuntu10:~/tool$ adb shell
#
# pwd
/
# ls
config
cache
sdcard
acct
mnt
d
etc
system
sys
sbin
proc
init.rc
init.goldfish.rc
init
default.prop
data
root
dev
#
# exit
```

After the debug service connection is setup correctly, the next step is to upload the code to the emulator and execute it:

```
vlinux@ubuntu10:~/ex/ndk.hello$ adb push test /data/local
89 KB/s (4893 bytes in 0.053s)
```

This will upload the program to the emulator at `/data/local`. When complete, enter the Android shell, change the execution permission, and execute the program:

```
vlinux@ubuntu10:~/ex/ndk.hello$ adb shell
# cd /data/local
# chmod 755 test
# ls
test
tmp
# ./test
hello
#
```

2.1.3 Debug native application

To debug the Android application with gdb, three consoles are needed. The first console will be used to start the emulator, the second console will be used to start the gdbserver service on the android emulator, and the third console will be used to run gdb, which will be used to control the debug process.

First start the emulator with the configuration name `keng` as follows:

```
$ source ~/tool/source.me
$ emulator -avd keng -shell
```

When the emulator has finished loading, start a new console. The next step is redirecting the TCP port 1234 to the emulator. This can be done by establishing a telnet connection to the emulator on port 5554. The second command can be used for port redirecting:

```
$ telnet localhost 5554
Trying ::1...
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Android Console: type 'help' for a list of commands
OK

redir add tcp:1234:1234
OK

exit
Connection closed by foreign host.
```

To see pre-configured port:

```
redir list
tcp:1234  => 1234
OK
```

To delete pre-configured port:

```
redir del tcp:1234
OK
```

To start the gdbserver:

```
$ source ~/tool/source.me
$ adb shell
#
# gdbserver 10.0.2.2:1234 /data/local/test
Process /data/local/test created; pid = 307
Listening on port 1234
```

To connect the mips-linux-android-gdb to gdbserver:

```
$ source ~/tool/source.me
$ cd ~/ex/ndk.hello
vlinux@ubuntu10:~/ex/ndk.hello$ mips-linux-android-gdb test
GNU gdb 6.6
Copyright (C) 2006 Free Software Foundation, Inc.
```

```
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or 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.
This GDB was configured as "--host=x86_64-linux-gnu --target=mips-linux-gnu"...
(gdb) target remote localhost:1234
Remote debugging using localhost:1234
[New Thread 307]
warning: Unable to find dynamic linker breakpoint function.
GDB will be unable to debug shared library initializers
and track explicitly loaded dynamic code.
0x7f000100 in ?? ()
(gdb)
```

When the connection has been made, the gdbserver's console will display the following:

```
Process /data/local/test created; pid = 307
Listening on port 1234
Remote debugging from host 10.0.2.2
```

Now the code can be debugged:

```
(gdb) b *main
Breakpoint 1 at 0x400450: file test.c, line 5.
(gdb) c
Continuing.
Error while mapping shared library sections:
/system/bin/linker: No such file or directory.
Error while mapping shared library sections:
libc.so: Success.
Error while mapping shared library sections:
libstdc++.so: Success.
Error while mapping shared library sections:
libm.so: Success.
Error while mapping shared library sections:
liblog.so: Success.
Breakpoint 1, main () at test.c:5
5   int main() {
(gdb)
(gdb) list
1
2   #include <stdio.h>
3
4
5   int main() {
6       printf("hello\n");
7       return 0;
8   }
9
(gdb) disassemble
Dump of assembler code for function main:
0x00400450 <main+0>:addiusp,sp,-32
0x00400454 <main+4>:swra,28(sp)
0x00400458 <main+8>:sws8,24(sp)
0x0040045c <main+12>:moves8,sp
0x00400460 <main+16>:luigp,0x42
0x00400464 <main+20>:addiugp,gp,-31472
0x00400468 <main+24>:swgp,16(sp)
```

```
0x0040046c <main+28>:luiv0,0x40
0x00400470 <main+32>:addiua0,v0,1248
0x00400474 <main+36>:lwv0,-32732(gp)
0x00400478 <main+40>:movet9,v0
0x0040047c <main+44>:jalrt9
0x00400480 <main+48>:nop
0x00400484 <main+52>:lwgp,16(s8)
0x00400488 <main+56>:movev0,zero
0x0040048c <main+60>:movesp,s8
0x00400490 <main+64>:lwra,28(sp)
0x00400494 <main+68>:lws8,24(sp)
0x00400498 <main+72>:addiusp,sp,32
0x0040049c <main+76>:jrra
0x004004a0 <main+80>:nop
End of assembler dump.
(gdb) c
Continuing.

Program exited normally.
```

The gdbserver console will display the following message:

```
Process /data/local/test created; pid = 307
Listening on port 1234
Remote debugging from host 10.0.2.2
hello

Child exited with retcode = 0

Child exited with status 0
GDBserver exiting
```

2.1.4 Debug example code together with system library

By default, SDK does not include the RFS image that has the debug symbols, which makes source-level debugging of applications with the system library impossible. The steps shown below assume that the debug symbols are available by building it from Android source.

This linker warning message is shown when gdb on the host machine cannot find the linker that contains the debug symbol information:

```
(gdb) target remote localhost:1234
Remote debugging using localhost:1234
[New Thread 307]
warning: Unable to find dynamic linker breakpoint function.
GDB will be unable to debug shared library initializers
and track explicitly loaded dynamic code.
0x7f000100 in ?? ()
```

This can be resolved by entering the following command before connecting to the gdbserver:

```
set solib-search-path android_source_directory/out/target/product/generic/symbols/
system/bin
```

This error message is shown when gdb on the host machine cannot find the library that contains the debug symbol information:

```
(gdb) c
Continuing.
Error while mapping shared library sections:
/system/bin/linker: No such file or directory.
Error while mapping shared library sections:
libc.so: Success.
Error while mapping shared library sections:
libstdc++.so: Success.
Error while mapping shared library sections:
libm.so: Success.
Error while mapping shared library sections:
liblog.so: Success.
```

This problem can be resolved by entering the following command before connecting to the gdbserver:

```
set solib-search-path android_source_directory/out/target/product/generic/symbols/
system/lib
```

In the example, `android_source_directory` is located in `/home/vlinux/mips/mipsfroyo`, the path must be set before connecting to the gdbserver. If more than one directory is required, use a colon (“:”) to separate the paths:

```
set solib-search-path android_source_directory/out/target/product/generic/symbols/
system/bin: android_source_directory/out/target/product/generic/symbols/system/lib
```

The libraries and executables are in `system/lib` and `system/bin` respectively. The output will be similar to the following:

```
keng@linux-softcsd:~/project/ndk.hello$ mips-linux-android-gdb test
GNU gdb 6.6
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or 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.
This GDB was configured as "--host=x86_64-linux-gnu --target=mips-linux-gnu"...
(gdb) set solib-search-path /home/vlinux/mips/mipsfroyo/out/target/product/generic/
symbols/system/lib:/home/vlinux/mips/mipsfroyo/out/target/product/generic/symbols/
system/bin
(gdb) target remote localhost:1234
Remote debugging using localhost:1234
[New Thread 345]
__dl___start () at bionic/linker/arch/mips/begin.S:9
9               bal     1f
Current language:  auto; currently asm
```

2.2 Shared library without Android makefile

NDK is commonly used to develop shared libraries. Such shared libraries can be used in Android java applications via the JNI interface. For purposes of testing, NDK can be used to build the test program and shared library, and then debug the application via gdbserver. The following files are used in this example:

`test1.c`:	test program that calls the shared library
`libc1.c`:	shared library containing function `func1` that prints a message to the console output of shared library `libc1.so`

2.2.1 Building shared library

Building the shared library is similar to building the native application with NDK. First, define a simple shared library function in in `libc1.c` like the following:

```
#include <stdio.h>

int func1() {
        printf("hello from func1\n");
        return 0;
}
```

Modify the previous `test1.c` program as follows:

```
#include <stdio.h>
extern int func1();
int main() {
        func1();
        printf("hello\n");
        return 0;
}
```

When completed, compile the library and program using the following commands:

```
$NDK/toolchains/mips-linux-android-4.4.3/prebuilt/linux-x86/bin/mips-linux-android-
gcc \
-I$NDK/platforms/android-8/arch-mips-r2/usr/include \
-L$NDK/platforms/android-8/arch-mips-r2/usr/lib \
-EL-g -mhard-float -fPIC -shared -Wl,-soname -Wl,libc1.so  -o libc1.so lib1.c -llog
-lm

$NDK/toolchains/mips-linux-android-4.4.3/prebuilt/linux-x86/bin/mips-linux-android-
gcc \
-I$NDK/platforms/android-8/arch-mips-r2/usr/include \
-L$NDK/platforms/android-8/arch-mips-r2/usr/lib -L. \
-EL -mhard-float -o test1 test1.c -lc1 -llog -lm
```

2.2.2 Run test program with shared library

Running the test program with the shared library is the same as running the standalone program without the shared library, except that one extra step is required. Assuming that the emulator is up and running with ADB service, upload the program to the emulator:

```
$ adb push test1 /data/local
88 KB/s (4136 bytes in 0.045s)

$ adb push libc1.so /data/local
62 KB/s (3607 bytes in 0.056s)
```

To run the program:

```
$ adb shell
# cd /data/local
# ls -l
-rwxrwxrwx root     root         4136 2011-08-11 17:44 test1
-rwxrwxrwx root     root         2467 2011-08-11 17:44 libc1.so
```

```
drwxrwx--x shell    shell             2011-08-23 13:11 tmp
# chmod 755 test1
# ./test1
link_image[2329]: failed to link ./test1
CANNOT LINK EXECUTABLE
#
```

The error message “CANNOT LINK EXECUTABLE” indicates that LD_LIBRARY_PATH is not setup correctly. By default, LD_LIBRARY_PATH is set to /system/lib.

```
$ adb shell
# cd /data/local
# ls -l
-rwxrwxrwx root     root         4136 2011-08-11 17:44 test1
-rwxrwxrwx root     root         2467 2011-08-11 17:44 libc1.so
drwxrwx--x shell    shell             2011-08-23 13:11 tmp
# chmod 755 test1
# ./test1
link_image[2329]: failed to link ./test1
CANNOT LINK EXECUTABLE
#
```

To resolve this problem, update LD_LIBRARY_PATH using either of the following commands:

```
#LD_LIBRARY_PATH=/data/local:$LD_LIBRARY_PATH ./test1
hello from func1
hello

# export LD_LIBRARY_PATH=/data/local:$LD_LIBRARY_PATH
# ./test1
hello from func1
hello
```

2.2.3 Debug shared library

Debugging the shared library is the same as debugging the application, except that LD_LIBRARY_PATH must be set before starting the gdbserver:

```
# export LD_LIBRARY_PATH=/data/local:$LD_LIBRARY_PATH
# gdbserver 10.0.2.2:1234 /data/local/test1
Listening on port 1234
```

Below is the expected output:

```
$ mips-linux-android-gdb test1
GNU gdb 6.6
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or 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.
This GDB was configured as "--host=x86_64-linux-gnu --target=mips-linux-gnu"...
(gdb)  set solib-search-path /home/vlinux/mips/mipsfroyo/out/target/product/
generic/symbols/system/bin:/home/vlinux/mips/mipsfroyo/out/target/product/generic/
symbols/system/lib
(gdb) target remote localhost:1234
Remote debugging using localhost:1234
```

```
[New Thread 395]
__dl___start () at bionic/linker/arch/mips/begin.S:9
9               bal     1f
Current language:  auto; currently asm
(gdb) delete
(gdb) b *main
Breakpoint 1 at 0x400480
(gdb) b *func1
Breakpoint 2 at 0x400510
(gdb) c
Continuing.
Breakpoint 2 at 0x50000360: file lib1.c, line 5.

Breakpoint 1, 0x00400480 in main ()
Current language:  auto; currently c
(gdb) c
Continuing.

Breakpoint 2, func1 () at lib1.c:5
5       int func1() {
(gdb) list
1
2       #include <stdio.h>
3
4
5       int func1() {
6               printf("hello from func1\n");
7               return 0;
8       }
9
(gdb)
```

2.3 Sample program in NDK

2.3.1 Build hello-jni example

To begin building the NDK:

```
$ source source.me
$ cd $NDK/samples/hello-jni
$ ls
AndroidManifest.xml  default.properties  jni  res  src  tests
$ $NDK/ndk-build
Gdbserver      : [mips-linux-android-4.4.3] libs/mips/gdbserver
Gdbsetup       : libs/mips/gdb.setup
Compile   : hello-jni <= hello-jni.c
SharedLibrary  : libhello-jni.so
Install        : libhello-jni.so => libs/mips/libhello-jni.so
```

By default, the `build` command will not appear on the screen. To turn on console messages:

```
vlinux@ubuntu10:~/tool/ndk/android-mips-ndk-r5b-linux/samples/hello-jni$ V=1 $NDK/
ndk-build
rm -f /home/vlinux/tool/ndk/android-mips-ndk-r5b-linux/samples/hello-jni/libs/mips/
lib*.so /home/vlinux/tool/ndk/android-mips-ndk-r5b-linux/samples/hello-jni/libs/
mips-r2/lib*.so
```

```
rm -f /home/vlinux/tool/ndk/android-mips-ndk-r5b-linux/samples/hello-jni/libs/mips/
gdbserver /home/vlinux/tool/ndk/android-mips-ndk-r5b-linux/samples/hello-jni/libs/
mips-r2/gdbserver
rm -f /home/vlinux/tool/ndk/android-mips-ndk-r5b-linux/samples/hello-jni/libs/mips/
gdb.setup /home/vlinux/tool/ndk/android-mips-ndk-r5b-linux/samples/hello-jni/libs/
mips-r2/gdb.setup
Gdbserver      : [mips-linux-android-4.4.3] libs/mips/gdbserver
mkdir -p /home/vlinux/tool/ndk/android-mips-ndk-r5b-linux/samples/hello-jni/libs/
mips
install -p /home/vlinux/tool/ndk/android-mips-ndk-r5b-linux/toolchains/mips-linux-
android-4.4.3/prebuilt/gdbserver /home/vlinux/tool/ndk/android-mips-ndk-r5b-linux/
samples/hello-jni/libs/mips/gdbserver
Gdbsetup       : libs/mips/gdb.setup
echo "set solib-search-path /home/vlinux/tool/ndk/android-mips-ndk-r5b-linux/
samples/hello-jni/obj/local/mips" > /home/vlinux/tool/ndk/android-mips-ndk-r5b-
linux/samples/hello-jni/libs/mips/gdb.setup
echo "directory /home/vlinux/tool/ndk/android-mips-ndk-r5b-linux/platforms/android-
8/arch-mips/usr/include /home/vlinux/tool/ndk/android-mips-ndk-r5b-linux/samples/
hello-jni/jni /home/vlinux/tool/ndk/android-mips-ndk-r5b-linux/sources/cxx-stl/
system" >> /home/vlinux/tool/ndk/android-mips-ndk-r5b-linux/samples/hello-jni/libs/
mips/gdb.setup
Compile   : hello-jni <= hello-jni.c
/home/vlinux/tool/ndk/android-mips-ndk-r5b-linux/toolchains/mips-linux-android-
4.4.3/prebuilt/linux-x86/bin/mips-linux-android-gcc -MMD -MP -MF /home/vlinux/tool/
ndk/android-mips-ndk-r5b-linux/samples/hello-jni/obj/local/mips/objs-debug/hello-
jni/hello-jni.o.d.org -fpic -fno-strict-aliasing -finline-functions -ffunction-
sections -funwind-tables -fmessage-length=0 -fno-inline-functions-called-once -
fgcse-after-reload -frerun-cse-after-loop -frename-registers  -D__ANDROID__ -Wno-
psabi -EL -march=mips32 -mtune=mips32 -mips32 -mhard-float -O2 -fomit-frame-pointer
-funswitch-loops -finline-limit=300 -I/home/vlinux/tool/ndk/android-mips-ndk-r5b-
linux/samples/hello-jni/jni -DANDROID  -Wa,--noexecstack -O0 -g -I/home/vlinux/
tool/ndk/android-mips-ndk-r5b-linux/platforms/android-8/arch-mips/usr/include -c  /
home/vlinux/tool/ndk/android-mips-ndk-r5b-linux/samples/hello-jni/jni/hello-jni.c -
o /home/vlinux/tool/ndk/android-mips-ndk-r5b-linux/samples/hello-jni/obj/local/
mips/objs-debug/hello-jni/hello-jni.o && rm -f /home/vlinux/tool/ndk/android-mips-
ndk-r5b-linux/samples/hello-jni/obj/local/mips/objs-debug/hello-jni/hello-jni.o.d
&& mv /home/vlinux/tool/ndk/android-mips-ndk-r5b-linux/samples/hello-jni/obj/local/
mips/objs-debug/hello-jni/hello-jni.o.d.org /home/vlinux/tool/ndk/android-mips-ndk-
r5b-linux/samples/hello-jni/obj/local/mips/objs-debug/hello-jni/hello-jni.o.d
SharedLibrary  : libhello-jni.so
/home/vlinux/tool/ndk/android-mips-ndk-r5b-linux/toolchains/mips-linux-android-
4.4.3/prebuilt/linux-x86/bin/mips-linux-android-gcc -nostdlib -Wl,-soname,libhello-
jni.so -Wl,-shared,-Bsymbolic -Wl,-T,ldscripts/mipself_android.xsc    /home/vlinux/
tool/ndk/android-mips-ndk-r5b-linux/samples/hello-jni/obj/local/mips/objs-debug/
hello-jni/hello-jni.o      /home/vlinux/tool/ndk/android-mips-ndk-r5b-linux/
platforms/android-8/arch-mips/usr/lib/libc.so /home/vlinux/tool/ndk/android-mips-
ndk-r5b-linux/platforms/android-8/arch-mips/usr/lib/libstdc++.so /home/vlinux/tool/
ndk/android-mips-ndk-r5b-linux/platforms/android-8/arch-mips/usr/lib/libm.so /home/
vlinux/tool/ndk/android-mips-ndk-r5b-linux/platforms/android-8/arch-mips/usr/lib/
libdl.so  -EL -march=mips32 -mips32 -mhard-float /home/vlinux/tool/ndk/android-
mips-ndk-r5b-linux/toolchains/mips-linux-android-4.4.3/prebuilt/linux-x86/bin/../
lib/gcc/mips-linux-android/4.4.3/el/mips32/libgcc_eh.a /home/vlinux/tool/ndk/
android-mips-ndk-r5b-linux/toolchains/mips-linux-android-4.4.3/prebuilt/linux-x86/
bin/../lib/gcc/mips-linux-android/4.4.3/el/mips32/libgcc.a   -Wl,--no-undefined -
Wl,-z,noexecstack  -Wl,-rpath-link=/home/vlinux/tool/ndk/android-mips-ndk-r5b-
linux/platforms/android-8/arch-mips/usr/lib   -o /home/vlinux/tool/ndk/android-
mips-ndk-r5b-linux/samples/hello-jni/obj/local/mips/libhello-jni.so
Install        : libhello-jni.so => libs/mips/libhello-jni.so
```

```
mkdir -p /home/vlinux/tool/ndk/android-mips-ndk-r5b-linux/samples/hello-jni/libs/
mips
install -p /home/vlinux/tool/ndk/android-mips-ndk-r5b-linux/samples/hello-jni/obj/
local/mips/libhello-jni.so /home/vlinux/tool/ndk/android-mips-ndk-r5b-linux/
samples/hello-jni/libs/mips/libhello-jni.so
/home/vlinux/tool/ndk/android-mips-ndk-r5b-linux/toolchains/mips-linux-android-
4.4.3/prebuilt/linux-x86/bin/mips-linux-android-strip --strip-unneeded  /home/
vlinux/tool/ndk/android-mips-ndk-r5b-linux/samples/hello-jni/libs
```

The procedure above builds the shared library.

Next we generate `build.xml`:

```
$ source source.me
$ cd $NDK/samples/hello-jni
$ ls
AndroidManifest.xml  default.properties  jni  res  src  tests
$ android update project -p . -s
Updated default.properties
Updated local.properties
Added file ./build.xml
Added file ./proguard.cfg
Updated default.properties
Updated local.properties
Added file ./tests/build.xml
Added file ./tests/proguard.cfg
```

After generating the `build.xml`, the following command can be used to generate the apk.

```
$ ant debug
vlinux@ubuntu10:~/tool/ndk/android-mips-ndk-r5b-linux/samples/hello-jni$ ant debug
Buildfile: /home/vlinux/tool/ndk/android-mips-ndk-r5b-linux/samples/hello-jni/
build.xml
    [setup] Android SDK Tools Revision 11
    [setup] Project Target: Android 2.2.1
    [setup] API level: 8
    [setup]
    [setup] ------------------
    [setup] Resolving library dependencies:
    [setup] No library dependencies.
    [setup]
    [setup] ------------------
    [setup]
    [setup] WARNING: Attribute minSdkVersion in AndroidManifest.xml (3) is lower
than the project target API level (8)
    [setup]
    [setup] Importing rules file: tools/ant/main_rules.xml

-debug-obfuscation-check:

-set-debug-mode:

-compile-tested-if-test:

-pre-build:

-dirs:
     [echo] Creating output directories if needed...
    [mkdir] Created dir: /home/vlinux/tool/ndk/android-mips-ndk-r5b-linux/samples/
hello-jni/bin
    [mkdir] Created dir: /home/vlinux/tool/ndk/android-mips-ndk-r5b-linux/samples/
hello-jni/gen
    [mkdir] Created dir: /home/vlinux/tool/ndk/android-mips-ndk-r5b-linux/samples/
hello-jni/bin/classes

-aidl:
     [echo] Compiling aidl files into Java classes...

-renderscript:
     [echo] Compiling RenderScript files into Java classes and RenderScript
bytecode...

-resource-src:
     [echo] Generating R.java / Manifest.java from the resources...

-pre-compile:

compile:
    [javac] /home/vlinux/tool/sdk/android-sdk_eng.wipromips3_linux-x86/tools/ant/
main_rules.xml:384: warning: 'includeantruntime' was not set, defaulting to
build.sysclasspath=last; set to false for repeatable builds
    [javac] Compiling 2 source files to /home/vlinux/tool/ndk/android-mips-ndk-r5b-
linux/samples/hello-jni/bin/classes

-post-compile:

-obfuscate:

-dex:
     [echo] Converting compiled files and external libraries into /home/vlinux/
tool/ndk/android-mips-ndk-r5b-linux/samples/hello-jni/bin/classes.dex...

-package-resources:
     [echo] Packaging resources
     [aapt] Creating full resource package...
     [aapt] Warning: AndroidManifest.xml already defines debuggable (in http://
schemas.android.com/apk/res/android); using existing value in manifest.

-package-debug-sign:
[apkbuilder] Creating HelloJni-debug-unaligned.apk and signing it with a debug
key...

debug:
     [echo] Running zip align on final apk...
     [echo] Debug Package: /home/vlinux/tool/ndk/android-mips-ndk-r5b-linux/
samples/hello-jni/bin/HelloJni-debug.apk

BUILD SUCCESSFUL
Total time: 5 seconds
```

When complete, the following apk will be generated:

```
$vlinux@ubuntu10:~/tool/ndk/android-mips-ndk-r5b-linux/samples/hello-jni$ find . -
name "*.apk"
./bin/HelloJni-debug.apk
./bin/HelloJni-debug-unaligned.apk
```

2.3.2 Running hello-jni example

Assume the emulator is up and running with ADB service available.

```
vlinux@ubuntu10:~/tool/ndk/android-mips-ndk-r5b-linux/samples/hello-jni$ adb kill-
server
vlinux@ubuntu10:~/tool/ndk/android-mips-ndk-r5b-linux/samples/hello-jni$ adb start-
server
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
vlinux@ubuntu10:~/tool/ndk/android-mips-ndk-r5b-linux/samples/hello-jni$ adb
install ./bin/HelloJni-debug.apk
468 KB/s (112242 bytes in 0.233s)
    pkg: /data/local/tmp/HelloJni-debug.apk
Failure [INSTALL_FAILED_INVALID_APK]
```

If installation failed with `[INSTALL_FAILED_INVALID_APK]`, the next step is to check that the emulator is configured to have the same ABI.

In this case, the ABI is misconfigured to `mipso32`, so we need to modify `/default.prop` in `SDK_ROOT/platforms/android-2.2.1/images/ramdisk.img`:

```
vlinux@ubuntu10:~/tool/ndk/android-mips-ndk-r5b-linux/samples/hello-jni$ adb shell
getprop | grep abi
[ro.product.cpu.abi]: [mipso32]
```

To modify the ABI setting, first we shutdown the emulator and then extract `ramdisk.img`:

```
$ mkdir test
$ cd test
$ cp $SDK/platforms/android-2.2.1/images/ramdisk.img ./ramdisk.cpio.gz
$ gzip -d ramdis.cpio.gz
$ cpio -i -F ./ramdisk.cpio
$ rm -rf ramdisk.cpio
$ ls
data  default.prop  dev  init  init.goldfish.rc  init.rc  proc  sbin  sys  system
```

Next we append the following line to the end of `default.prop`:

```
ro.product.cpu.abi=mips
```

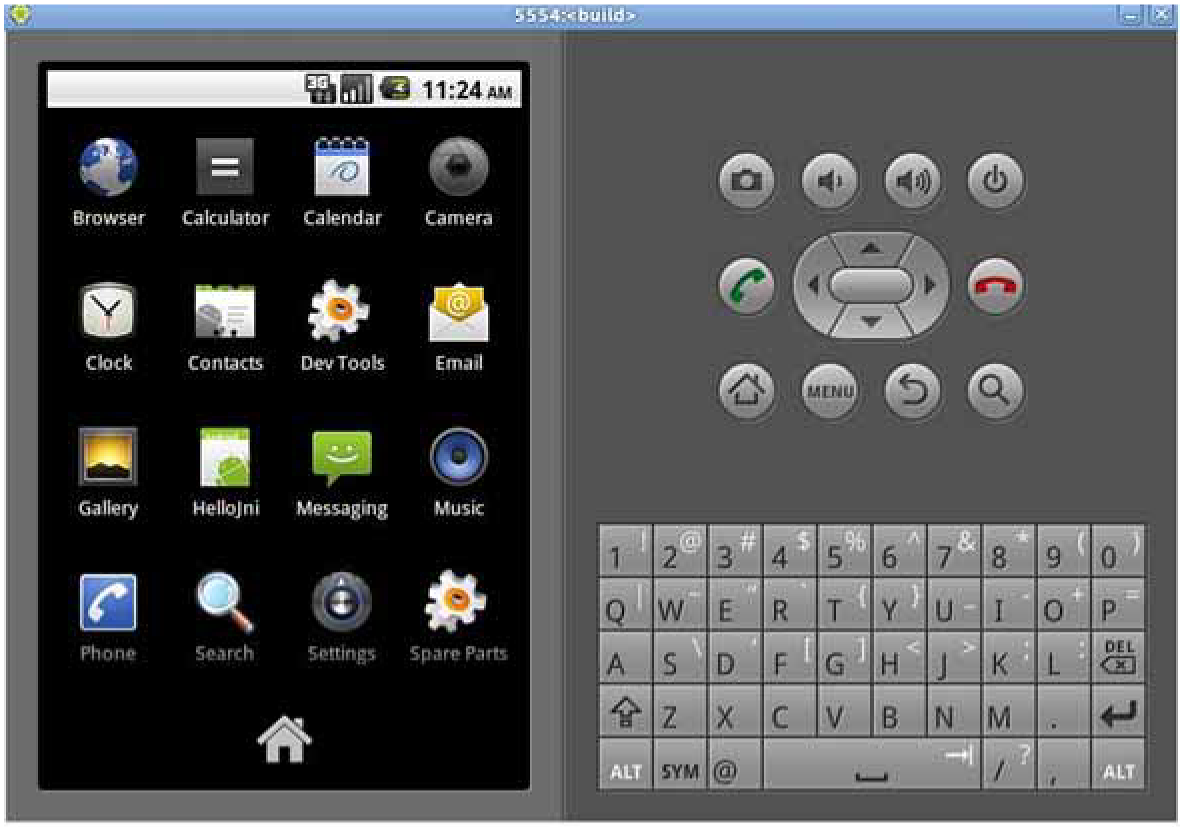
The third step is repacking the `ramdisk.img`. In the `ramdisk` directory:

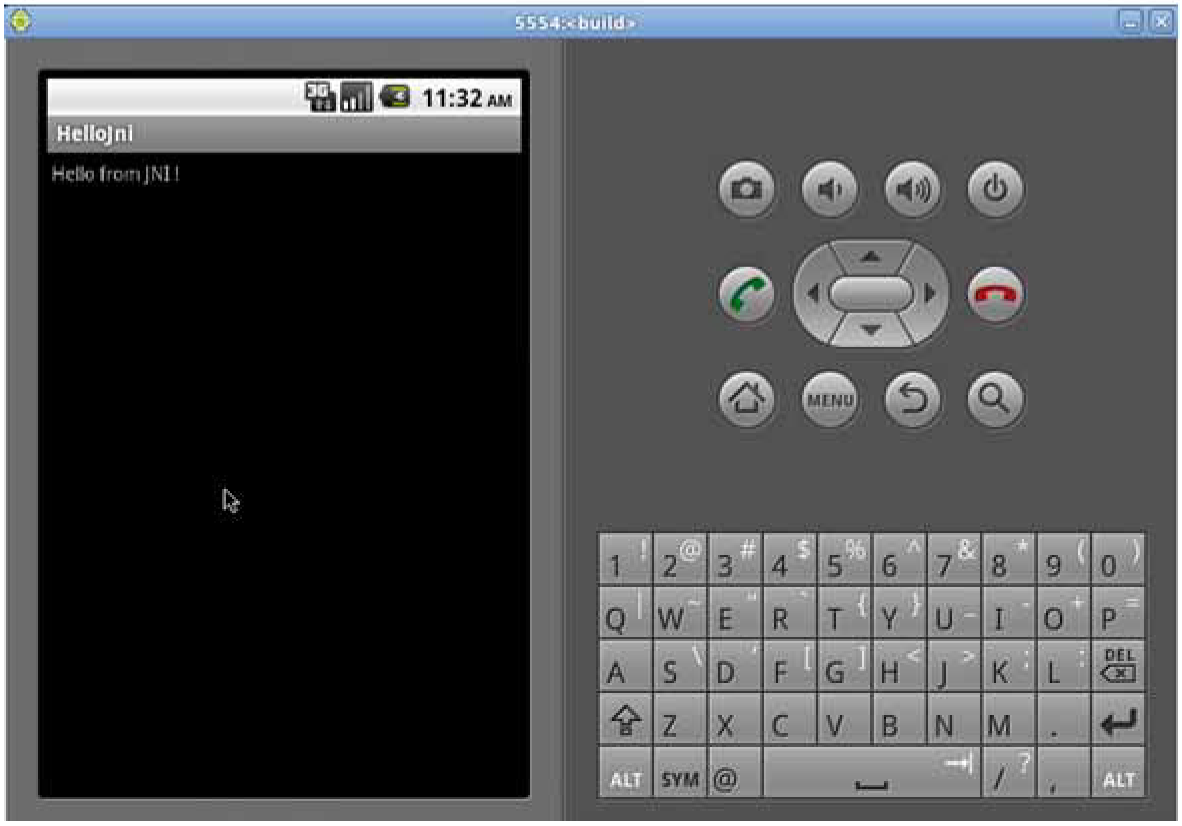
```
$ cd test
$ find . | cpio -o -H newc | gzip > ../ramdisk.img
762 blocks
$ cp ../ramdisk.img $SDK/platforms/android-2.2.1/images/ramdisk.img
```

When completed, start the emulator and repeat the installation.

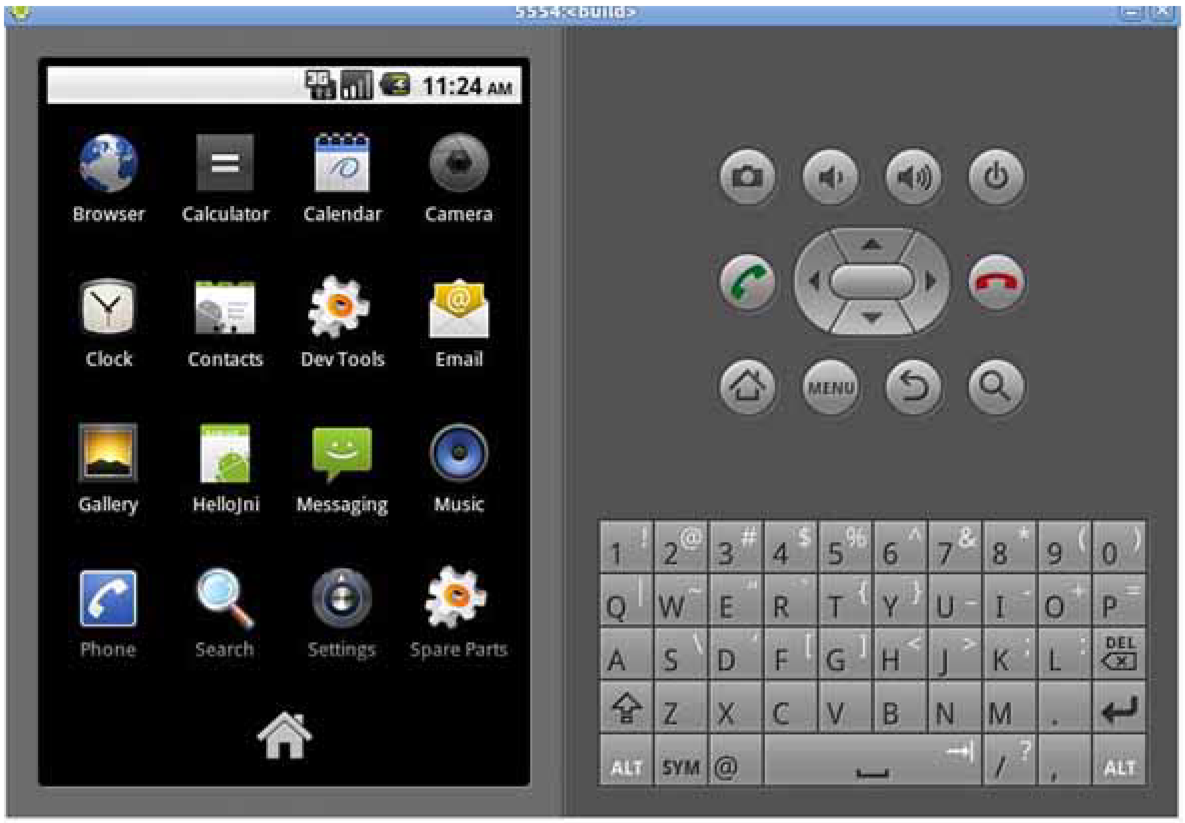
```
$ adb install ./bin/HelloJni-debug.apk
428 KB/s (112242 bytes in 0.256s)
    pkg: /data/local/tmp/HelloJni-debug.apk
Success
```

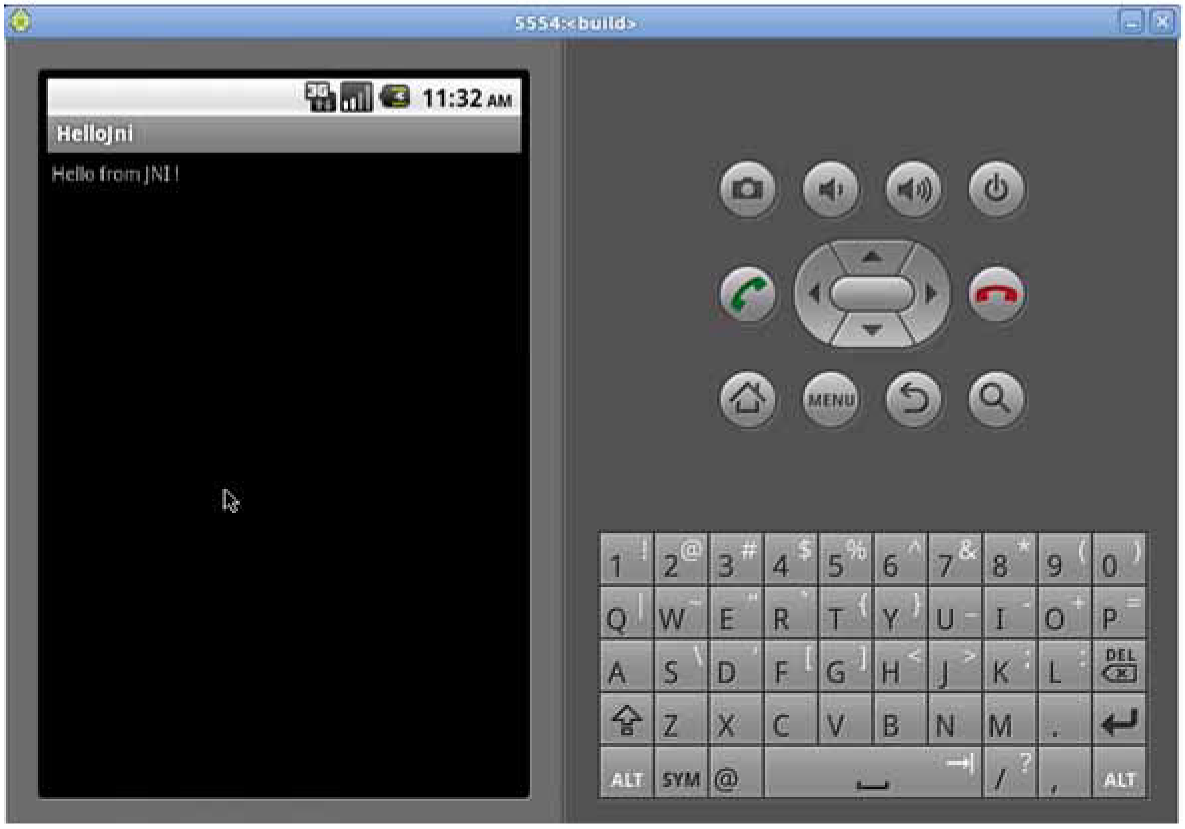
The `HelloJNI` icon should now appear on the Android desktop. Click on the icon to start the application.





Hello from JNI !





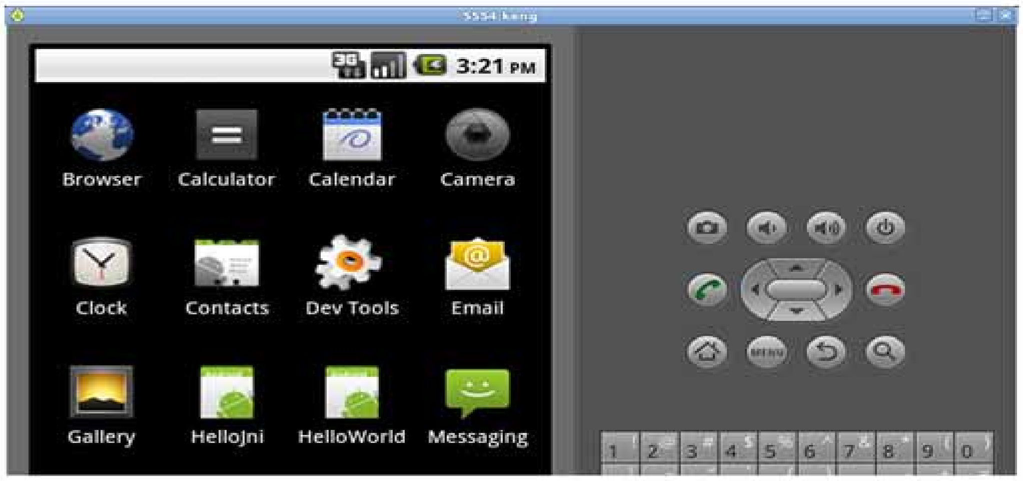
```
Hello from JNI !
```

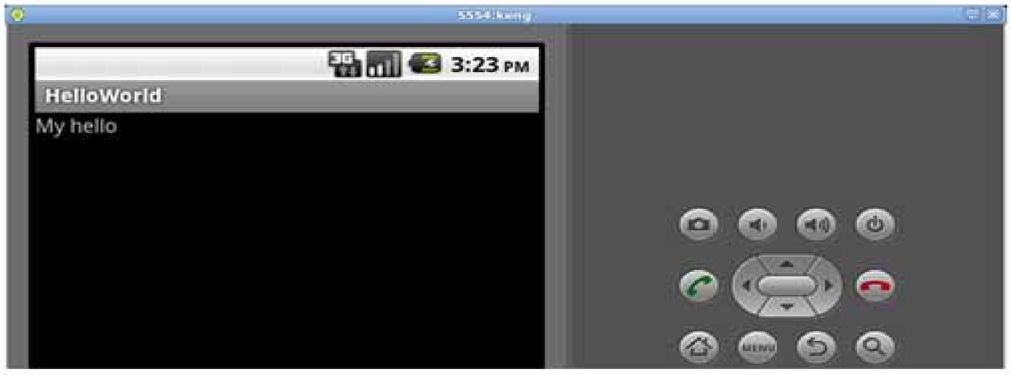
2.4 Android project with NDK

2.4.1 Sample program without native code

```
vlinux@ubuntu10:~/ex$ android create project --target 1 --name Hello \
--path ./helloworld --activity HelloWorld\
--package com.example.HelloWorld
Created project directory: ./helloworld
Created directory /home/vlinux/ex/helloworld/src/com/example/HelloWorld
Added file ./helloworld/src/com/example/HelloWorld/HelloWorld.java
Created directory /home/vlinux/ex/helloworld/res
Created directory /home/vlinux/ex/helloworld/bin
Created directory /home/vlinux/ex/helloworld/libs
Created directory /home/vlinux/ex/helloworld/res/values
Added file ./helloworld/res/values/strings.xml
Created directory /home/vlinux/ex/helloworld/res/layout
Added file ./helloworld/res/layout/main.xml
Created directory /home/vlinux/ex/helloworld/res/drawable-hdpi
Created directory /home/vlinux/ex/helloworld/res/drawable-mdpi
Created directory /home/vlinux/ex/helloworld/res/drawable-ldpi
Added file ./helloworld/AndroidManifest.xml
Added file ./helloworld/build.xml
Added file ./helloworld/proguard.cfg

vlinux@ubuntu10:~/ex$ ls -l helloworld/
total 40
-rw-r--r-- 1 vlinux vlinux  642 2011-08-25 15:03 AndroidManifest.xml
drwxr-xr-x 2 vlinux vlinux 4096 2011-08-25 15:03 bin
-rw-r--r-- 1 vlinux vlinux  696 2011-08-25 15:03 build.properties
-rw-r--r-- 1 vlinux vlinux 2889 2011-08-25 15:03 build.xml
-rw-r--r-- 1 vlinux vlinux  362 2011-08-25 15:03 default.properties
drwxr-xr-x 2 vlinux vlinux 4096 2011-08-25 15:03 libs
-rw-r--r-- 1 vlinux vlinux  450 2011-08-25 15:03 local.properties
-rw-r--r-- 1 vlinux vlinux 1159 2011-08-25 15:03 proguard.cfg
drwxr-xr-x 7 vlinux vlinux 4096 2011-08-25 15:03 res
drwxr-xr-x 3 vlinux vlinux 4096 2011-08-25 15:03 src

vlinux@ubuntu10:~/ex$ cd helloworld
vlinux@ubuntu10:~/ex/helloworld$ vi src/com/example/HelloWorld/HelloWorld.java

package com.example.HelloWorld;
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class HelloWorld extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        TextView tv = new TextView(this);
        tv.setText("My hello");
        setContentView(tv);
    }
}
```

```
vlinux@ubuntu10:~/ex/helloworld$ ant debug
Buildfile: /home/vlinux/ex/helloworld/build.xml

...

BUILD SUCCESSFUL
Total time: 4 seconds
vlinux@ubuntu10:~/ex/helloworld$ $ adb install ./bin/Hello-debug.apk
146 KB/s (13346 bytes in 0.088s)
    pkg: /data/local/tmp/Hello-debug.apk
Success
```





My hello

2.5 Sample program with native code

The following example creates an Android project from scratch.

```
$ source source.me
$ cd ~/ex
vlinux@ubuntu10:~/ex$ android create project --target 1 --name Hello\
--path ./myjnidir --activity HelloWorldJNI \
--package com.example.MyJNI
Created project directory: ./myjnidir
Created directory /home/vlinux/ex/myjnidir/src/com/example/MyJNI
Added file ./myjnidir/src/com/example/MyJNI/HelloWorldJNI.java
Created directory /home/vlinux/ex/myjnidir/res
Created directory /home/vlinux/ex/myjnidir/bin
Created directory /home/vlinux/ex/myjnidir/libs
Created directory /home/vlinux/ex/myjnidir/res/values
Added file ./myjnidir/res/values/strings.xml
Created directory /home/vlinux/ex/myjnidir/res/layout
Added file ./myjnidir/res/layout/main.xml
Created directory /home/vlinux/ex/myjnidir/res/drawable-hdpi
Created directory /home/vlinux/ex/myjnidir/res/drawable-mdpi
Created directory /home/vlinux/ex/myjnidir/res/drawable-ldpi
Added file ./myjnidir/AndroidManifest.xml
Added file ./myjnidir/build.xml
Added file ./myjnidir/proguard.cfg

vlinux@ubuntu10:~/ex$ ls -l ./myjnidir/
total 40
-rw-r--r-- 1 vlinux vlinux  640 2011-08-25 16:02 AndroidManifest.xml
drwxr-xr-x 2 vlinux vlinux 4096 2011-08-25 16:02 bin
-rw-r--r-- 1 vlinux vlinux  696 2011-08-25 16:02 build.properties
-rw-r--r-- 1 vlinux vlinux 2889 2011-08-25 16:02 build.xml
-rw-r--r-- 1 vlinux vlinux  362 2011-08-25 16:02 default.properties
drwxr-xr-x 2 vlinux vlinux 4096 2011-08-25 16:02 libs
-rw-r--r-- 1 vlinux vlinux  450 2011-08-25 16:02 local.properties
-rw-r--r-- 1 vlinux vlinux 1159 2011-08-25 16:02 proguard.cfg
drwxr-xr-x 7 vlinux vlinux 4096 2011-08-25 16:02 res
drwxr-xr-x 3 vlinux vlinux 4096 2011-08-25 16:02 src

vlinux@ubuntu10:~/ex$ ls -l ./myjnidir/src/com/example/MyJNI/
total 4
-rw-r--r-- 1 vlinux vlinux 347 2011-08-25 16:02 HelloWorldJNI.java
vlinux@ubuntu10:~/ex$ vi ./myjnidir/src/com/example/MyJNI/HelloWorldJNI.java
```

Modify the HelloworldJNI.java file with the following:

```
package com.example.MyJNI;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class HelloWorldJNI extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        TextView tv = new TextView(this);
        tv.setText( stringFromJNI() );
        setContentView(tv);
    }

    public native String stringFromJNI();
    static {
        System.loadLibrary("myjnilib");
    }

}
```

Test that the java code compiles correctly:

```
vlinux@ubuntu10:~/ex/myjnidir$ ant compile
Buildfile: /home/vlinux/ex/myjnidir/build.xml

....

BUILD SUCCESSFUL
Total time: 3 seconds
```

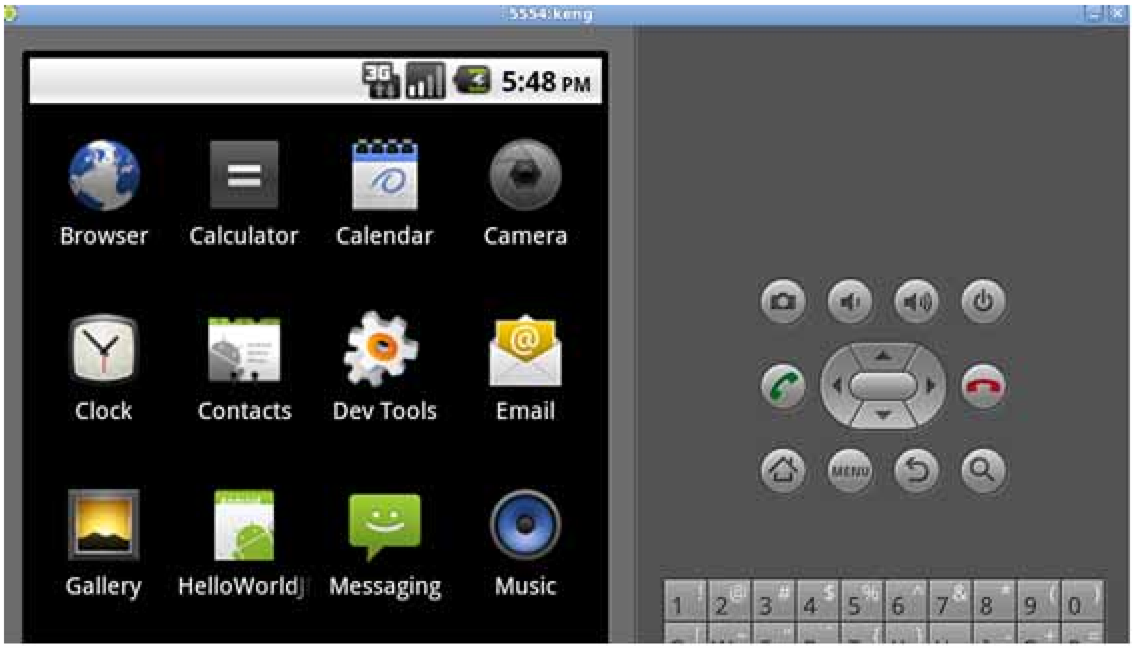
Create shared C library and Android Makefile:

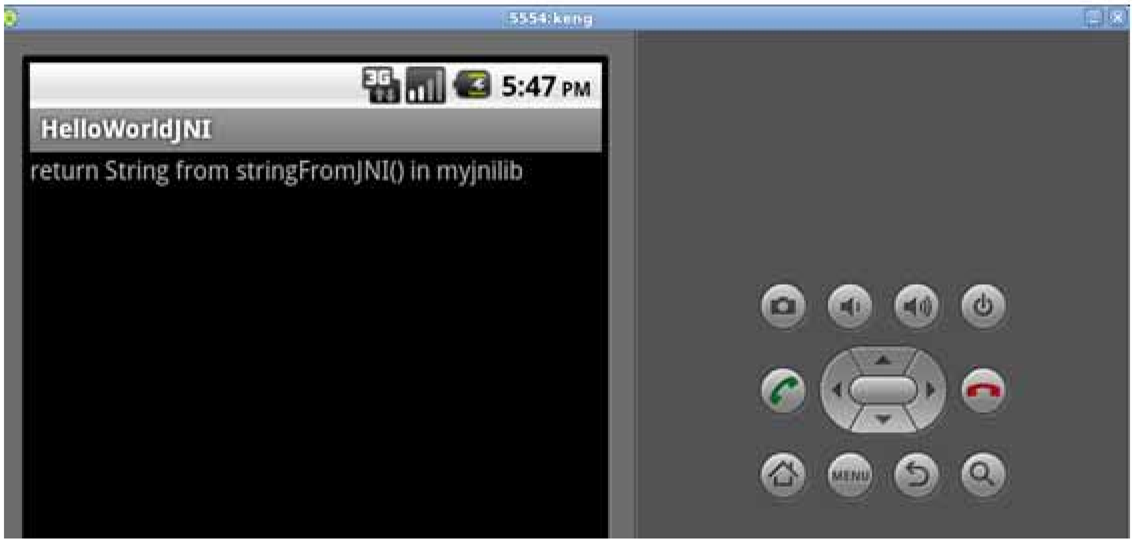
```
vlinux@ubuntu10:~/ex/myjnidir$ mkdir jni
vlinux@ubuntu10:~/ex/myjnidir$ vi jni/myjnilib.c

#include <string.h>
#include <jni.h>

jstring
Java_com_example_MyJNI_HelloWorldJNI_stringFromJNI( JNIEnv* env,
                                                  jobject thiz )
{
    return (*env)->NewStringUTF(env, "return String from stringFromJNI() in
myjnilib ");
}

vlinux@ubuntu10:~/ex/myjnidir/jni$ vi Android.mk

LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := myjnilib
LOCAL_SRC_FILES := myjnilib.c
include $(BUILD_SHARED_LIBRARY)
```

Compile and build the apk:

```
vlinux@ubuntu10:~/ex/myjnidir$ $NDK/ndk-build clean
Clean: myjnilib [mips]
Clean: stdc++ [mips]
vlinux@ubuntu10:~/ex/myjnidir$ $NDK/ndk-build
Compile     : myjnilib <= myjnilib.c
SharedLibrary  : libmyjnilib.so
Install        : libmyjnilib.so => libs/mips/libmyjnilib.so

vlinux@ubuntu10:~/ex/myjnidir$ ant clean
```

```
Buildfile: /home/vlinux/ex/myjnidir/build.xml

...

clean:
   [delete] Deleting directory /home/vlinux/ex/myjnidir/bin
   [delete] Deleting directory /home/vlinux/ex/myjnidir/gen

BUILD SUCCESSFUL
Total time: 1 second

vlinux@ubuntu10:~/ex/myjnidir$ ant debug
Buildfile: /home/vlinux/ex/myjnidir/build.xml

...

BUILD SUCCESSFUL
Total time: 2 seconds

vlinux@ubuntu10:~/ex/myjnidir$ adb install ./bin/Hello-debug.apk
255 KB/s (14541 bytes in 0.055s)
   pkg: /data/local/tmp/Hello-debug.apk
Success
```





```
return String from stringFromJNI() in myjnilib
```

3 Q & A

3.1 Execution Errors

This section explains the meaning of a number of error messages and how the errors can be resolved.

3.1.1 SDL init failure, reason is: No available video device

If the following message appears when you attempt to run the emulator,

```
SDL init failure, reason is: No available video device
```

you must install the following package:

```
sudo apt-get install ia32-libs lib32stdc++6
```

3.1.2 exec: 113: java: not found

When you start the emulator with the following command,

```
emulator
```

and receive the following message,

```
/home/vlinux/tool/sdk/android-sdk_eng.wipromips3_linux-x86/tools/android: 83: java:
not found
/home/vlinux/tool/sdk/android-sdk_eng.wipromips3_linux-x86/tools/android: 100:
java: not found
Starting Android SDK and AVD Manager
exec: 113: java: not found
```

the problem is caused by not adding `JDK 5 bin` directory to the path.

```
export PATH= /home/vlinux/tool/jdk1.5.0_22/bin:$PATH
```

3.1.3 mips-linux-android-gdb: command not found

```
$ mips-linux-android-gdb
mips-linux-android-gdb: command not found
```

`mips-linux-android-gdb` is missing in `NDK r5b. mips-linux-android-gdb` in `NDK r6m` can be used for this purpose. An alternative choice is to use the gdb in the Code Sourcery toolchain.

3.1.4 mips-linux-android-gdb: localhost:1234: Connection refused

If the port redirection is not setup correctly, the following message will appear:

```
vlinux@ubuntu10:~/ex/ndk.hello$ mips-linux-android-gdb test
GNU gdb 6.6
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or 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.
This GDB was configured as "--host=x86_64-linux-gnu --target=mips-linux-gnu"...
(gdb) set solib-search-path /home/keng/mips/mipsfroyo/out/target/product/generic/
symbols/system/lib
(gdb) target remote localhost:1234
localhost:1234: Connection refused.
```

To solve the problem, do the following:

```
$ telnet localhost 5554
Trying ::1...
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Android Console: type 'help' for a list of commands
OK

redir add tcp:1234:1234
OK

exit
Connection closed by foreign host.
```

3.1.5 warning: Unable to find dynamic linker breakpoint function

This warning message indicates that gdb cannot find the Android linker debug symbol. To solve the problem:

```
set solib-search-path android_source_directory/out/target/product/generic/symbols/
system/bin
```

See Section 2.1.4, “Debug example code together with system library” for more detailed information.

3.1.6 warning: Unable to find dynamic linker breakpoint function

This warning message indicates that gdb cannot find the Android linker debug symbol. To solve the problem:

```
set solib-search-path android_source_directory/out/target/product/generic/symbols/
system/bin
```

See Section 2.1.4, “Debug example code together with system library” for more detailed information.

3.1.7 Error while mapping shared library sections

This warning message indicates that gdb cannot find the Android system library debug symbol. To solve the problem:

```
set solib-search-path android_source_directory/out/target/product/generic/symbols/
system/lib

Error while mapping shared library sections:
/system/bin/linker: No such file or directory.
Error while mapping shared library sections:
libc.so: Success.
Error while mapping shared library sections:
libstdc++.so: Success.
Error while mapping shared library sections:
```

```
libm.so: Success.
Error while mapping shared library sections:
liblog.so: Success.
```

See Section 2.1.4, “Debug example code together with system library” for more detailed information.

3.2 Compilation Errors

3.2.1 error while loading shared libraries: libgmpxx.so.4

This compiler error message indicates that the `libgmp` library is missing. To install the library:

```
sudo apt-get install libgmp3-dev

vlinux@ubuntu10:~/ndk/android-mips-ndk-r5b-linux/samples/hello-jni$ $NDK/ndk-build
Gdbserver      : [mips-linux-android-4.4.3] libs/mips/gdbserver
Gdbsetup       : libs/mips/gdb.setup
Compile    : hello-jni <= hello-jni.c
/home/vlinux/ndk/android-mips-ndk-r5b-linux/toolchains/mips-linux-android-4.4.3/
prebuilt/linux-x86/bin/../libexec/gcc/mips-linux-android/4.4.3/cc1: error while
loading shared libraries: libgmpxx.so.4: cannot open shared object file: No such
file or directory
make: *** [/home/vlinux/ndk/android-mips-ndk-r5b-linux/samples/hello-jni/obj/local/
mips/objs-debug/hello-jni/hello-jni.o] Error 1
```

3.2.2 warning: libstdc++.so, needed by liblog.so, not found

This linker error message indicates that the linker cannot find the `share` library.

```
warning: libstdc++.so, needed by /home/vlinux/tool/ndk/android-mips-ndk-r5b-linux/
platforms/android-8/arch-mips-r2/usr/lib/liblog.so, not found
```

There are two ways to fix this problem. The first uses the `-Wl,-rpath-link` flag:

```
-Wl,-rpath-link=$NDK/platforms/android-8/arch-mips-r2/usr/lib
```

This forces the linker to search for the library from `$NDK/platforms/android-8/arch-mips-r2/usr/lib` at link time. `-rpath-link` will not affect the runtime search path. Without the `-rpath-link` option, the linker by default searches the `/lib` and `/usr/lib` host directories at link time.

The second way uses `-rpath`. Unlike the`-rpath-link` flag, `-rpath` will hardcode the runtime library search path when linking the `library/application`. The `-rpath` option must be used with the`-sysroot` option:

```
-Wl,-rpath --sysroot
```

3.2.3 ld: crtbegin_dynamic.o: No such file: No such file or directory

This error message indicates that the linker cannot find `crtbegin_dynamic.o`. This is a problem with r5b NDK. The simplest workaround is to copy the file to the current directory. If API 8 is used:

```
cp $NDK/platforms/android-8/arch-mips/usr/lib/*.o
```

3.2.4 ld: crtend_android.o: No such file: No such file or directory

Resolve with the workaround used in the preceding section.

4 Document Revision History

Revision	Date	Description
01.00	September 1, 2011	• Initial release