Application Debugging Using GDB

V4.2 FOR BCA LINUX® ROUTERS

CPE-AN500-R

5300 California Avenue • Irvine, CA 92617 • Phone: 949-926-5000 • Fax: 949-926-5203

January 23, 2014

Revision History

Revision	*Date*	*Change Description*
CPE-AN500-R	01/23/14	Initial release

Broadcom Corporation
5300 California Avenue
Irvine, CA 92617

Table of Contents

List of Figures

About This Document

Purpose and Audience

This document provides information on debugging Linux® user space applications on the BCM96XXX Reference Design Platform using GDB. Both interactive and post-mortem methods are presented.

This document is intended for software engineers.

Acronyms and Abbreviations

In most cases, acronyms and abbreviations are defined on first use.

For a comprehensive list of acronyms and other terms used in Broadcom documents, go to: http://www.broadcom.com/press/glossary.php.

Document Conventions

The following conventions may be used in this document:

Convention	*Description*
Bold	User input and actions: for example, type **exit**, click **OK,** press **Alt+C**
`Monospace`	Code: `#include <iostream>` HTML: `<td rowspan = 3>` Command line commands and parameters: `wl [-l] <command>`
< >	Placeholders for *required* elements: enter your <username> or `wl <command>`
[ ]	Indicates *optional* command-line parameters: `wl [-l]` Indicates bit and byte ranges (inclusive): [0:3] or [7:0]

References

The references in this section may be used in conjunction with this document.

Note: Broadcom provides customer access to technical documentation and software through its Customer Support Portal (CSP) and Downloads and Support site (see Technical Support).

For Broadcom documents, replace the "xx" in the document number with the largest number available in the repository to ensure that you have the most current version of the document.

Document Name	*Number*	*Source*
[1] CMS Desktop Linux	CPE-AN6xx-R	CSP

Technical Support

Broadcom provides customer access to a wide range of information, including technical documentation, schematic diagrams, product bill of materials, PCB layout information, and software updates through its customer support portal (https://support.broadcom.com). For a CSP account, contact your Sales or Engineering support representative.

In addition, Broadcom provides other product support through its Downloads and Support site (http://www.broadcom.com/support/).

Introduction

This document provides information on debugging Linux® user space applications on the BCM96XXX Reference Design Platform using GDB. Two major methods are presented:

- ***Interactive*** debugging: using GDB at the host and GDBServer on the target.
- ***Post-mortem*** debugging: allowing the application to crash and create a core dump on the target, which is then transferred to the host, and then inspect the core dump using GDB on the host.

This document uses specific applications and build profiles as examples, but any user space application and build profile can be debugged using these methods.

Interactive Debugging with GDB and GDBServer

Building an Image

Follow these steps to build an image that will allow you to do interactive application debugging using GDB and GDBServer.

1. For releases prior to 4.12L.07, delete the following three lines from make.common:

   ```
   ifeq ($(strip $(BUILD_GDBSERVER)),y)
   SSTRIP = /bin/touch
   endif
   ```

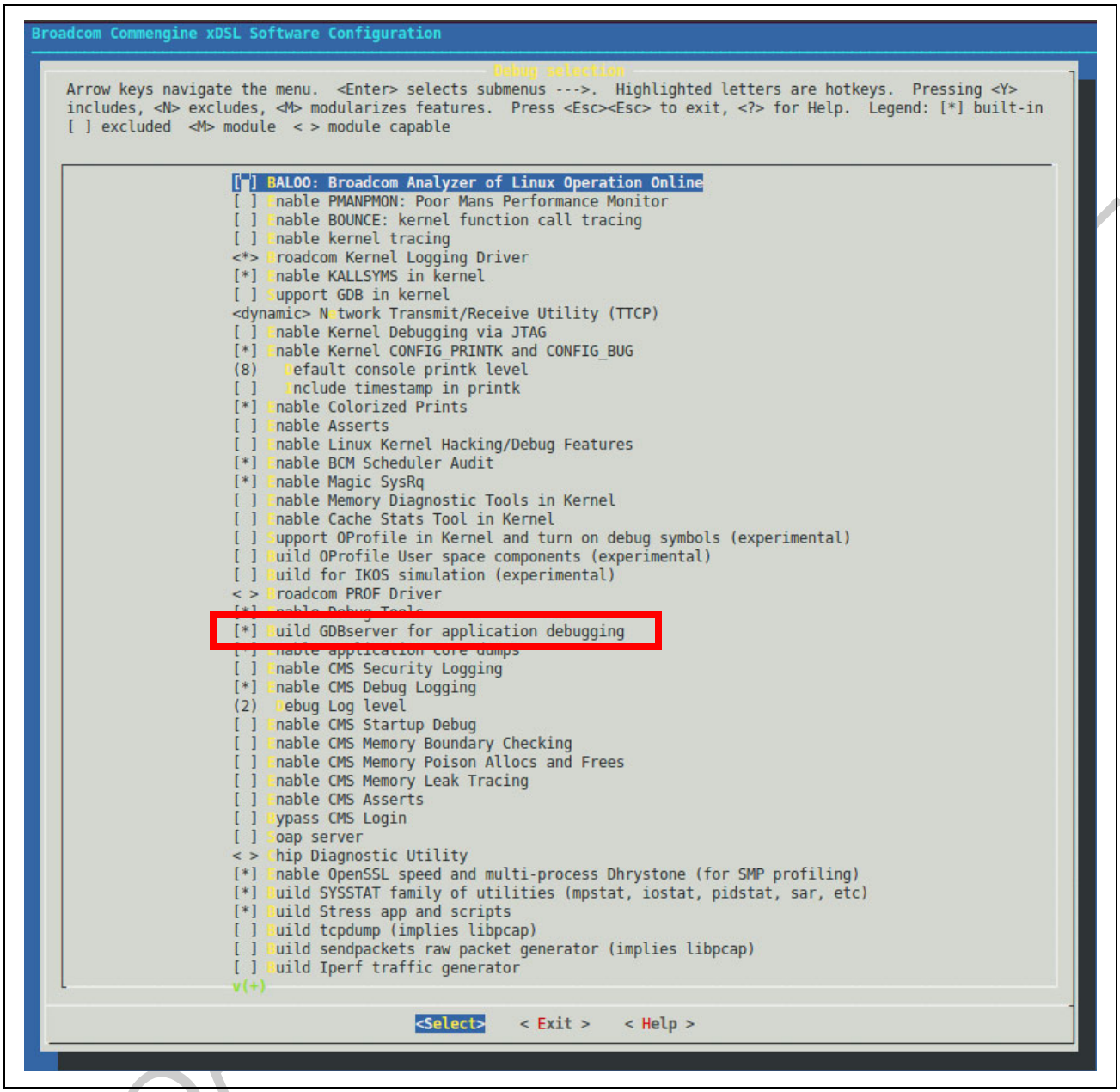
   For the 4.12L.07 release and beyond, no manual edit is needed.

2. At the top-level of source code directory, type **make menuconfig**.

   Select "**Load software build profile**" and load an existing profile, such as 96368GW, then select "**Debug selection**".

   Select "**Build GDBServer for application debugging**". See Figure 1 on page 8.

Figure 1: Main Configuration Menu



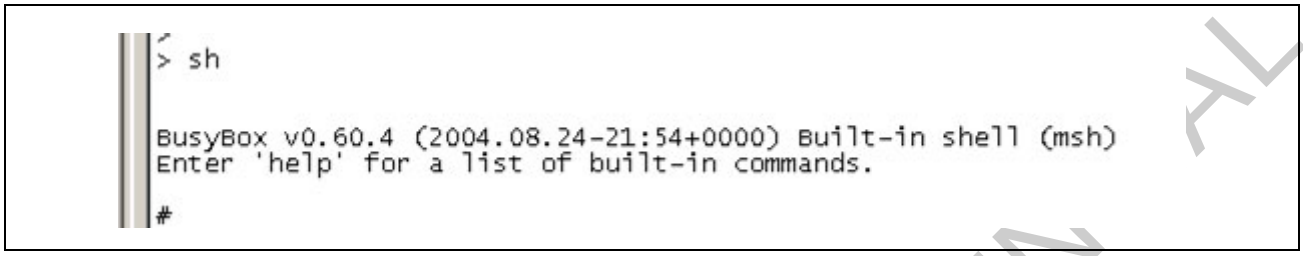
3. Save the profile and build the image using the modified profile.

Debugging the Application

Follow these steps to debug any application running on the target:

1. On the target, enter the shell prompt.

Figure 2: Entering the Shell Prompt



type:

ls /bin/gdbserver

to verify gdbserver has been installed in your image.

2. Start GDBServer using the following syntax:

gdbserver :port_number PROGRAM [ARGS ...].

For example, to debug your new application called myapp, use the following syntax:

/bin/gdbserver :2345 /bin/myapp

The user may choose another port number if desired. Many Broadcom applications, such as ssk and smd are already running. Some applications, such as httpd and dhcpd, must be started by the CMS framework and not on the command line. GDBServer can attach itself to an already running process if given the pid of the process. The syntax is:

/bin/gdbserver --attach :port_number pid

The 'ps' command can be used to list the PIDs of the executing processes. When the GDBServer exits, it will kill the process as well. For example, if you want to debug ssk, and from the ps listing you know ssk has pid 282, you would type:

/bin/gdbserver --attach :2345 282

3. On the host side, cd to the application source directory and start GDB. The following example uses ssk as the application to be debugged:

cd SRC*PATH*/userspace/private/apps/ssk

where "SRCPATH" is the top level of your source code directory.

TOOLCHAIN/usr/bin/mips-linux-uclibc-gdb ssk

where "TOOLCHAIN" is where you installed your toolchain.

Now specify where the target libraries are:

(gdb) set solib-absolute-prefix SRCPATH/targets/PROFILE/fs.install

(gdb) set solib-search-path SRCPATH/targets/PROFILE/fs.install/lib

where PROFILE is the name of the build profile you are using, for example 96368GW.

Now connect your GDB to the GDBServer running on the target:

(gdb) target remote 192.168.1.1:2345

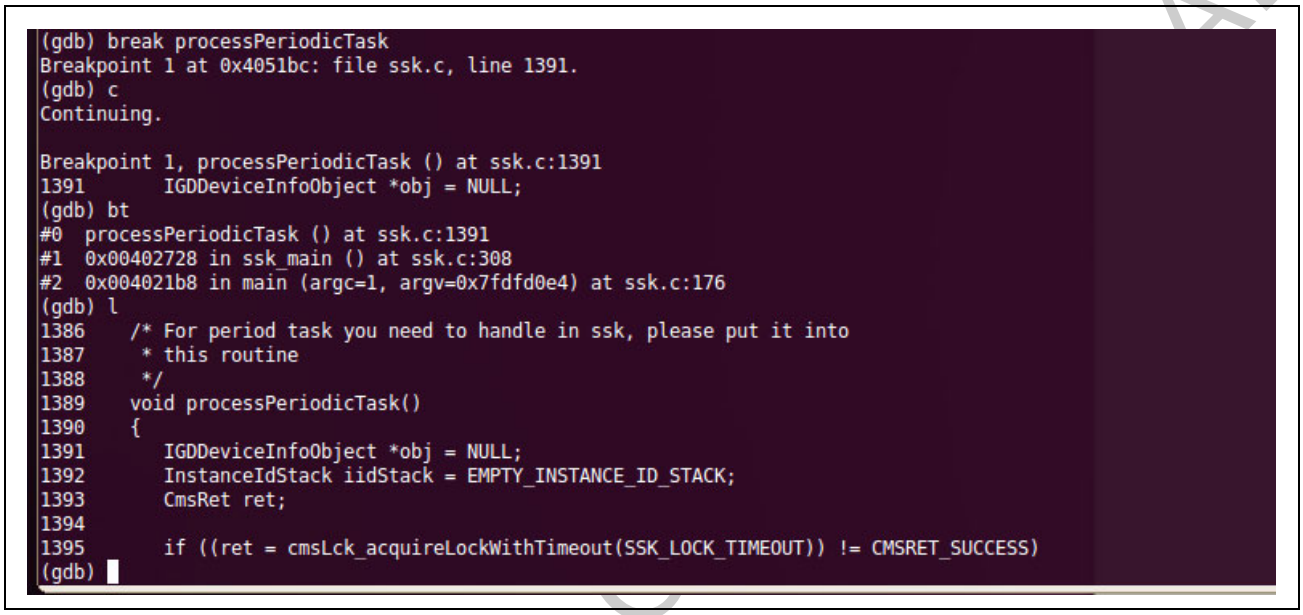
In this example, 192.168.1.1 is the target IP address.

Now you can set breakpoints on the app, for example by typing:

(gdb) break processPeriodicTask

and then type “c” to allow ssk to continue execution. Within 60 seconds, ssk should hit this breakpoint and control will be returned to the debugger. You can then type “bt” to get a backtrace, inspect variables, and so on. See Figure 3 on page 10.

Figure 3: Breakpoints



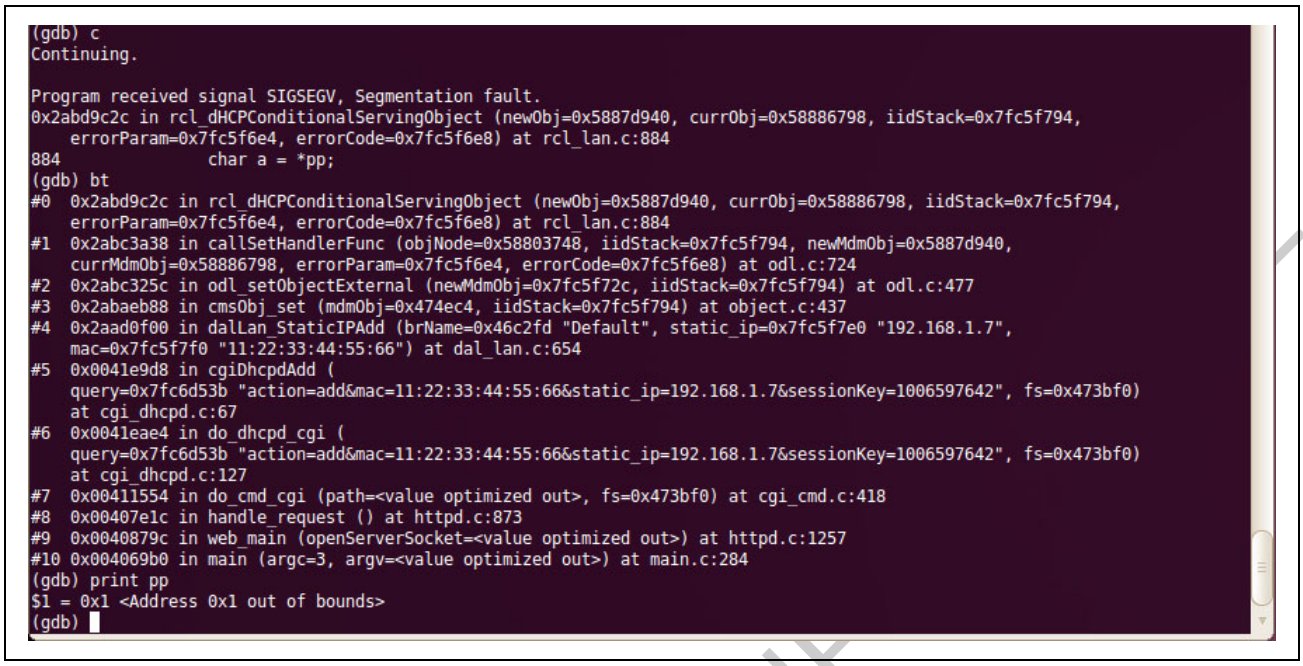
(TODO: include tips for debugging pthread apps.)

Catching Crashes

The interactive debug method can also be used to analyze applications that crash. In this case, either start or attach to the application which is known to crash. Instead of setting a breakpoint, simply allow the application to continue running. When the application crashes, control will be transferred to the GDB on the host, where you can do a backtrace, inspect variables, and so on.

In Figure 4 on page 11, a bug was purposely introduced in rcl_dHCPConditionalServingObject(). Steps were taken on the webui to trigger this bug, and when httpd crashed, control was transferred to the GDB on the host.

Figure 4: Example of Application Crash



Debugger Graphical User Interfaces

DDD (Data Display Debugger)

DDD is a graphical front end for GDB and other debuggers. It can be invoked with the shell command ddd, provided that you have X server running on your PC host. The typical usage of the DDD on the BCM96XXX platforms is shown as follows:

% ddd --gdb --debugger *TOOLCHAIN*/usr/bin/mips-linux-uclibc-gdb *APP_NAME*

where “TOOLCHAIN” is where toolchain was installed.

More detailed information on DDD is available online via the DDD webpage, www.gnu.org/software/ddd.

XXGDB

xxgdb is an X-window graphical front end for GDB. xxgdb is part of the Mandrake installation and can be installed as a service on the Linux box.

To launch xxgdb, invoke it as follows:

% xxgdb

Post-Mortem Debugging with GDB

In some cases, it may be inconvenient or impossible to attach GDB to an application on the target. In these cases, the user can still configure the application to dump a core when it crashes, then transfer the core file to the host, where it can be debugged.

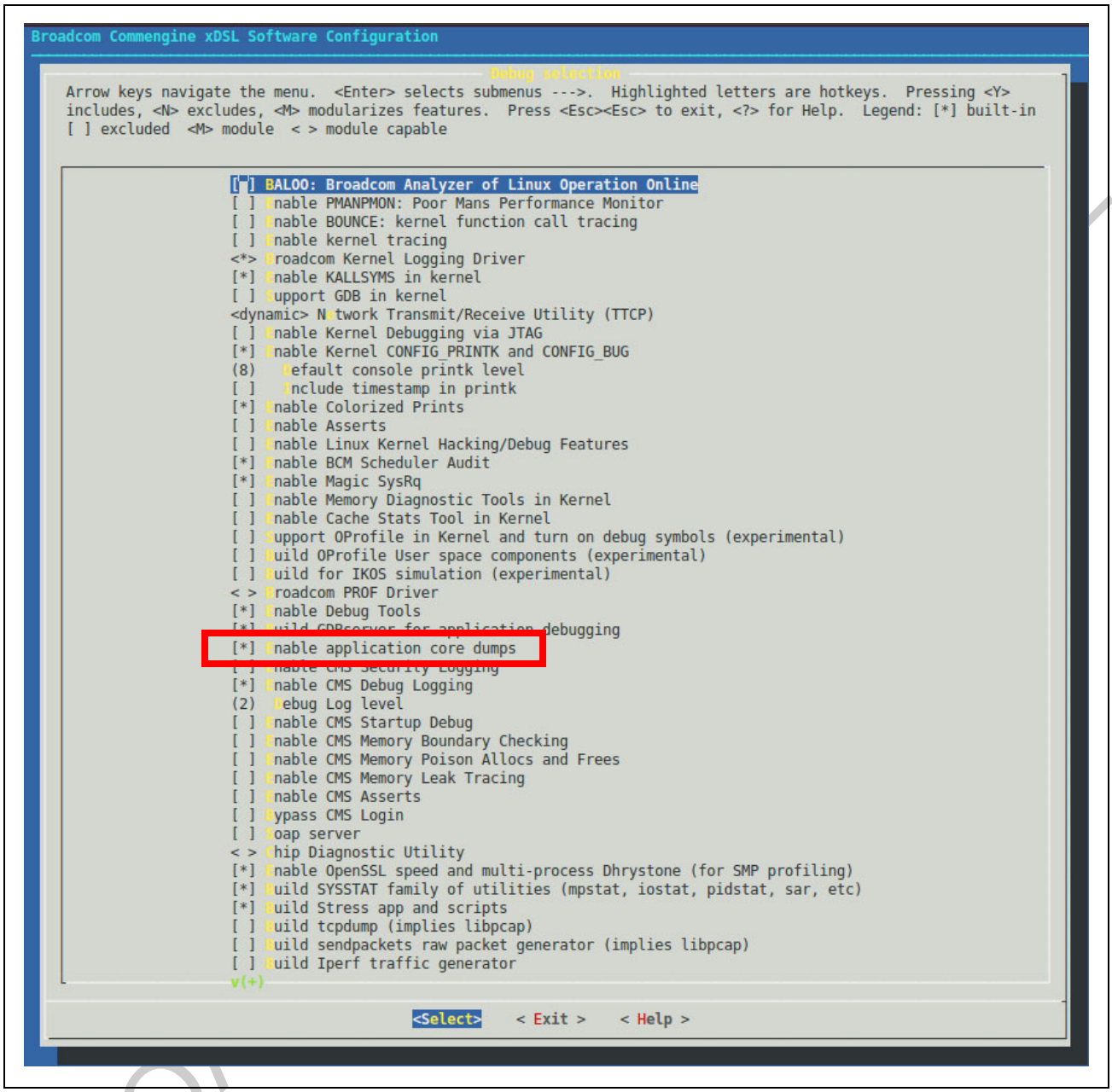
Setup

Prior to the 4.12L.07 release, make the following manual modifications:

- In make.common, delete the following three lines:
  ```
  ifeq ($(strip $(BUILD_GDBSERVER)),y)
  SSTRIP = /bin/touch
  endif
  ```
- In hostTools/scripts/defconfig-bcm.template, change this line:
  ```
  # CONFIG_ELF_CORE is not set
  ```
  into the following two lines:
  ```
  CONFIG_ELF_CORE=y
  CONFIG_CORE_DUMP_DEFAULT_ELF_HEADERS=y
  ```
- In userspace/gpl/apps/busybox/brcm.config, change this line:
  ```
  # CONFIG_FEATURE_INIT_COREDUMPS is not set
  ```
  into
  ```
  CONFIG_FEATURE_INIT_COREDUMPS=y
  ```
- In targets/buildFS, after the line:
  ```
  echo “Creating target root file system...”
  ```
  add this line:
  ```
  touch $ROOTFS/.init_enable_core
  ```
- In hostTools/libcreduction/Makefile, change:
  ```
  LIBOPT := y
  ```
  to
  ```
  LIBOPT := n
  ```
- Finally, rebuild the image and load it on the target.

With the 4.12L.07 release and beyond, run make menuconfig, and in the debug section, select “Enable application core dumps”. See Figure 5 on page 13.

Figure 5: Enable Application Core Dumps



Finally, rebuild the image and load it on the target.

Note: With the post-mortem debug approach, the user does not have to enable the GDB server (but it does not hurt to also enable it.)

Generating the Core on Target

The Linux kernel will now dump the core of any user space application that dies due to a core generating signal. See the man pages for signal(7) for a list of core generating signals. Typically, buggy applications will get a SIGSEV which will generate a core.

However, after the system boots, the user must still tell the Linux kernel where to dump the core and the filename of the core file. First the options will be listed, then some examples will be given.

There are several options for where to dump the core file:

- Systems which use a NAND based root filesystem can configure its root filesystem to be read/write via make menuconfig (typically it is configured as read/only).
- Some systems have a /data partition which is read/write.
- All systems have a /var partition which is read/write, but its size is limited to 420 KB. Typically, 420 KB is not big enough to hold a core file, but the size can be modified by editing SRCPATH/fs.src/etc/fstab.
- The core file can be transferred over the network using netcat (nc). This is a nice option if there are no read/write partitions big enough to hold the core file on the target.

The /proc file /proc/sys/kernel/core_pattern controls the location and filename of the core files. It takes several special characters, including:

- %e - will be transformed into the name of the application.
- %p - will be transformed into the pid of the application.
- %t - will be a timestamp (number of seconds since epoch).

See the man page core(5) for a complete list.

Examples

If you are using option (a) above and want the core file to have the format name-pid-timestamp.core, then type the following:

```
echo “%e-%p-%t.core” > /proc/sys/kernel/core_pattern
```

In this case, the core file will be in the same directory as the application.

If you are using option (b) above and want the core file to have the format core.appname.pid, then type the following:

```
echo “/data/core.%e.%p” > /proc/sys/kernel/core_pattern
```

If you are using option (c) above and want the core file to have the format core.appname.pid, then type the following:

```
echo “/var/core.%e.%p” > /proc/sys/kernel/core_pattern
```

If you are using option (d) above, then type the following:

```
echo “|/usr/bin/nc 192.168.1.100 30555” > /proc/sys/kernel/core_pattern
```

where 192.168.1.100 is the IP address of the development host, and 30555 is some arbitrary port number on the host. Both of these numbers may change depending on the environment. For this option, you must also have a nc server running on the development host:

```
nc -l 30555 > exe.core
```

The advantage of using nc to transfer the core image to the development host is that you do not need to write anything on the target. However, the disadvantage is that you must set up the server before the core dump occurs, and the server only handles one core dump at a time.

Transfer to Host

If the core file is initially written on the target (options a-c above), the user will need to transfer it to the development host. This can be done with nc. First, on the development host, type:

```
nc -l 30555 > core.out
```

where 30555 is an arbitrary port number. Choose another port number if desired.

core.out is the name of the core file. Choose a more descriptive name if you know the name of the application that generated the core.

Then on the target, type:

```
cat COREFILE | nc 192.168.1.100 30555
```

where COREFILE is the name of the corefile on the target, 192.168.1.100 is the IP address of the development host (may be different in your environment), and 30555 is some port number (may be customized for your environment).

Debug

On the development host, cd into the directory of the application that generated the core file, and make sure the core file is there. The following example uses userspace/private/apps/httpd and a core file name of core.out. Type:

```
/opt/toolchains/uclibc-crosstools-4.4.2-1/usr/bin/mips-linux-uclibc-gdb httpd core.out
```

initially, you will see a lot of error messages. They can be ignored. Now on the GDB prompt, type:

```
(gdb) set solib-absolute-prefix SRCPATH/targets/PROFILE/fs.install
(gdb) bt
```

where SRCPATH is top-level of your source code directory, and PROFILE is the profile name of the build (for example, 963268GW.

Typically, just seeing the full backtrace is enough to determine the cause of the crash. But the user can use GDB to dump out additional variables in the application. See the example in Figure 6 on page 16.

Figure 6: Variables Dump

Building a Custom GDBServer and GDB

In some special scenarios, the user may want or need to build his/her own GDBServer and GDB. This may also require rebuilding of the toolchain. Instructions for rebuilding the toolchain are outside of the scope of this document, however, below are the steps for building GDB.

Note: These instructions are from the 4.04 release, but they should still be basically correct.

1. Download the GDB source code.
2. Configure the source code for cross compile by typing:

   >./configure --target=mips-linux-uclibc --prefix=/opt/toolchains/uclibc-crosstools-gcc-4.2.3-3/usr --disable-werror
3. Build GDB and install it at the correct place in the toolchain, usually /opt/toolchains/uclibc-crosstools-gcc-4.4.2-1/usr/bin/mips-linux-uclibc-gdb.
4. Build GDBServer.

```
>cd gdb/gdbserver/

>export CC=/opt/toolchains/uclibc-crosstools-gcc-4.4.2-1/usr/bin/mips-linux-uclibc-gcc

>./configure --host=mips-linux-uclibc --target=mips-linux-uclibc --with-sysroot=/opt/
toolchains/uclibc-crosstools-gcc-4.4.2-1/usr/lib --prefix=/opt/toolchains/uclibc-crosstools-
gcc-4.4.2-1/usr

>make
```

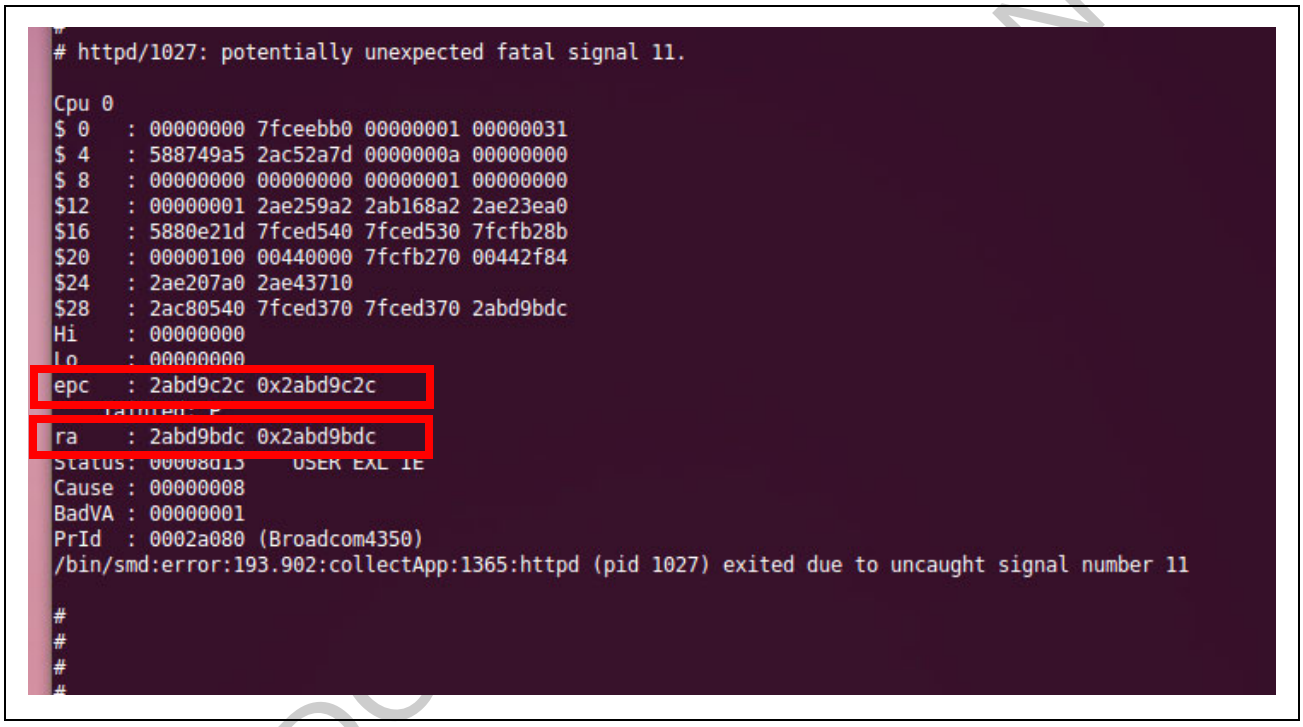
5. Install GDB server at the correct place in the toolchain, usually /opt/toolchains/uclibc-crosstools-gcc-4.4.2-1/usr/mips-linux-uclibc/target_utils/gdbserver.

Analyzing Crashes Without GDB

Even though the techniques described in this document are very effective at identifying the bug in an application, they do require some setup. Sometimes an application will crash on a system where GDBServer and core dumps have not been set up.

In recent releases, CMS configures /proc/sys/kernel/print-fatal-signals with "1". This means that when an application dies from an unexpected signal, the kernel will print out some information about the application, including the "EPC" (program counter address where the error occurred) and "RA" (return address). An example is shown in Figure 7.

Figure 7: Crash Results Without GDB



With some manual steps (getting the shared library maps of the application, dumping the application/ library symbol tables, and calculating the offsets), the user can identify the lines of code which correspond to the EPC and RA. However, the user will not be able to get a backtrace. So if this is not enough, then the user will need to use one of the techniques described in this document to debug the problem.

Connecting everything®

BROADCOM CORPORATION
5300 California Avenue
Irvine, CA 92617

Phone: 949-926-5000
Fax: 949-926-5203
E-mail: info@broadcom.com
Web: www.broadcom.com