Kernel Debugging Techniques

Sep 19, 2006 By Christopher Hallinan (/user/801642)
in

Chapter 14: Kernel Debugging Techniques, reprinted with permission from Embedded Linux Primer: A Practical Real-World Approach By Christopher Hallinan, Published by Prentice Hall

Chapter 14: Kernel Debugging Techniques

Often the pivotal factor in achieving development timetables comes down to one's efficiency in finding and fixing bugs. Debugging inside the Linux kernel can be quite challenging. No matter how you approach it, kernel debugging will always be complex. This chapter examines some of the complexities and presents ideas and methods to improve your debugging skills inside the kernel and device drivers.

<several sections omitted>

Debugging Optimized Kernel Code

At the start of this chapter, we said that one of the challenges identified in debugging kernel code results from compiler optimization. We noted that the Linux kernel is compiled by default with optimization level -O2. In the examples up to this point, we used -O1 optimization to simplify the debugging task. Here we illustrate one of the many ways optimization can complicate debugging.

The related Internet mail lists are strewn with questions related to what appear to be broken tools. Sometimes the poster reports that his debugger is singlestepping backward or that his line numbers do not line up with his source code. Here we present an example to illustrate the complexities that optimizing compilers bring to source-level debugging. In this example, the line numbers that gdb reports when a breakpoint is hit do not match up with the line numbers in our source file due to function inlining.

For this demonstration, we use the same debug code snippet as shown in Listing 14-4. (Parts of this is reproduced in Listing 14-9, below.) However, for this example, we have compiled the kernel with the compiler optimization flag -O2. This is the default for the Linux kernel. Listing 14-7 shows the results of this debugging session.

Listing 14-7

Optimized Architecture-Setup Code

```
$ ppc_44x-gdb --silent vmlinux
(gdb) target remote /dev/ttyS0
Remote debugging using /dev/ttyS0
breakinst () at arch/ppc/kernel/ppc-stub.c:825
825 }
(gdb) b panic
Breakpoint 1 at 0xc0016b18: file kernel/panic.c, line 74.
(gdb) b sys_sync
Breakpoint 2 at 0xc005a8c8: file fs/buffer.c, line 296.
(gdb) b yosemite_setup_arch
Breakpoint 3 at 0xc020f438: file arch/ppc/platforms/4xx/yosemite.c, line
116.
(gdb) c
Continuing.
Breakpoint 3, yosemite_setup_arch ()
at arch/ppc/platforms/4xx/yosemite.c:116
```

```
116 def = ocp_get_one_device(OCP_VENDOR_IBM, OCP_FUNC_EMAC, 0);
(gdb) l
111 struct ocp_def *def;
112 struct ocp_func_emac_data *emacdata;
113
114 /* Set mac_addr and phy mode for each EMAC */
115
116 def = ocp_get_one_device(OCP_VENDOR_IBM, OCP_FUNC_EMAC, 0);
117 emacdata = def->additions;
118 memcpy(emacdata->mac_addr, __res.bi_enetaddr, 6);
119 emacdata->phy_mode = PHY_MODE_RMII;
120
(gdb) p yosemite_setup_arch
$1 = {void (void)} 0xc020f41c <yosemite_setup_arch>
```

Referring back to Listing 14-4, (this can also be seen in Listing 14-9 below) notice that the function yosemite_setup_arch() actually falls on line 306 of the file yosemite.c. Compare that with Listing 14-7. We hit the breakpoint, but gdb reports the breakpoint at file yosemite.c line 116. It appears at first glance to be a mismatch of line numbers between the debugger and the corresponding source code. Is this a bug? First let's confirm what the compiler produced for debug information. Using the readelf5 tool described in Chapter 13, "Development Tools", we can examine the debug information for this function produced by the compiler.

```
$ ppc_44x-readelf --debug-dump=info vmlinux | grep -u6 \
yosemite_setup_arch | tail -n 7
DW_AT_name : (indirect string, offset: 0x9c04): yosemite_setup_arch
DW_AT_decl_file : 1
DW_AT_decl_line : 307
DW_AT_prototyped : 1
DW_AT_low_pc : 0xc020f41c
DW_AT_high_pc : 0xc020f794
DW_AT_frame_base : 1 byte block: 51 (DW_OP_reg1)
```

We don't have to be experts at reading DWARF2 debug records6 to recognize that the function in question is reported at line 307 in our source file. We can confirm this using the addr2line utility, also introduced in Chapter 13. Using the address derived from gdb in Listing 14-7:

```
$ ppc_44x-addr2line -e vmlinux 0xc020f41c
arch/ppc/platforms/4xx/yosemite.c:307
```

At this point, gdb is reporting our breakpoint at line 116 of the yosemite.c file. To understand what is happening, we need to look at the assembler output of the function as reported by gdb. Listing 14-8 is the output from gdb after issuing the disassemble command on the `yosemite_setup_arch()` function.

Listing 14-8

Disassemble Function yosemite_setup_arch

```
(gdb) disassemble yosemite_setup_arch
0xc020f41c <yosemite_setup_arch+0>: mflr r0
0xc020f420 <yosemite_setup_arch+4>: stwu r1,-48(r1)
0xc020f424 <yosemite_setup_arch+8>: li r4,512
0xc020f428 <yosemite_setup_arch+12>: li r5,0
0xc020f42c <yosemite_setup_arch+16>: li r3,4116
0xc020f430 <yosemite_setup_arch+20>: stmw r25,20(r1)
0xc020f434 <yosemite_setup_arch+24>: stw r0,52(r1)
0xc020f438 <yosemite_setup_arch+28>: bl 0xc000d344 <ocp_get_one_device>
0xc020f43c <yosemite_setup_arch+32>: lwz r31,32(r3)
```

```
   0xc020f440 <yosemite_setup_arch+36>: lis r4,-16350
   0xc020f444 <yosemite_setup_arch+40>: li r28,2
   0xc020f448 <yosemite_setup_arch+44>: addi r4,r4,21460
   0xc020f44c <yosemite_setup_arch+48>: li r5,6
   0xc020f450 <yosemite_setup_arch+52>: lis r29,-16350
   0xc020f454 <yosemite_setup_arch+56>: addi r3,r31,48
   0xc020f458 <yosemite_setup_arch+60>: lis r25,-16350
   0xc020f45c <yosemite_setup_arch+64>: bl 0xc000c708 <memcpy>
   0xc020f460 <yosemite_setup_arch+68>: stw r28,44(r31)
   0xc020f464 <yosemite_setup_arch+72>: li r4,512
   0xc020f468 <yosemite_setup_arch+76>: li r5,1
   0xc020f46c <yosemite_setup_arch+80>: li r3,4116
   0xc020f470 <yosemite_setup_arch+84>: addi r26,r25,15104
   0xc020f474 <yosemite_setup_arch+88>: bl 0xc000d344 <ocp_get_one_device>
   0xc020f478 <yosemite_setup_arch+92>: lis r4,-16350
   0xc020f47c <yosemite_setup_arch+96>: lwz r31,32(r3)
   0xc020f480 <yosemite_setup_arch+100>: addi r4,r4,21534
   0xc020f484 <yosemite_setup_arch+104>: li r5,6
   0xc020f488 <yosemite_setup_arch+108>: addi r3,r31,48
   0xc020f48c <yosemite_setup_arch+112>: bl 0xc000c708 <memcpy>
   0xc020f490 <yosemite_setup_arch+116>: lis r4,1017
   0xc020f494 <yosemite_setup_arch+120>: lis r5,168
   0xc020f498 <yosemite_setup_arch+124>: stw r28,44(r31)
   0xc020f49c <yosemite_setup_arch+128>: ori r4,r4,16554
   0xc020f4a0 <yosemite_setup_arch+132>: ori r5,r5,49152
   0xc020f4a4 <yosemite_setup_arch+136>: addi r3,r29,-15380
   0xc020f4a8 <yosemite_setup_arch+140>: addi r29,r29,-15380
   0xc020f4ac <yosemite_setup_arch+144>: bl 0xc020e338 <ibm440gx_get_clocks
>
   0xc020f4b0 <yosemite_setup_arch+148>: li r0,0
   0xc020f4b4 <yosemite_setup_arch+152>: lis r11,-16352
   0xc020f4b8 <yosemite_setup_arch+156>: ori r0,r0,50000
   0xc020f4bc <yosemite_setup_arch+160>: lwz r10,12(r29)
   0xc020f4c0 <yosemite_setup_arch+164>: lis r9,-16352
   0xc020f4c4 <yosemite_setup_arch+168>: stw r0,8068(r11)
   0xc020f4c8 <yosemite_setup_arch+172>: lwz r0,84(r26)
   0xc020f4cc <yosemite_setup_arch+176>: stw r10,8136(r9)
   0xc020f4d0 <yosemite_setup_arch+180>: mtctr r0
   0xc020f4d4 <yosemite_setup_arch+184>: bctrl
   0xc020f4d8 <yosemite_setup_arch+188>: li r5,64
   0xc020f4dc <yosemite_setup_arch+192>: mr r31,r3
   0xc020f4e0 <yosemite_setup_arch+196>: lis r4,-4288
   0xc020f4e4 <yosemite_setup_arch+200>: li r3,0
   0xc020f4e8 <yosemite_setup_arch+204>: bl 0xc000c0f8 <ioremap64>
   End of assembler dump.
   (gdb)
```

Once again, we need not be PowerPC assembly language experts to understand what is happening here. Notice the labels associated with the PowerPC `bl` instruction. This is a function call in PowerPC mnemonics. The symbolic function labels are the important data points. After a cursory analysis, we see several function calls near the start of this assembler listing:

```
   Address  Function
   0xc020f438  ocp_get_one_device()
   0xc020f45c  memcpy()
   0xc020f474  ocp_get_one_device()
   0xc020f48c  memcpy()
   0xc020f4ac  ibm440gx_get_clocks()
```

Listing 14-9 reproduces portions of the source file yosemite.c. Correlating the functions we found in the gdb disassemble output, we see those labels occurring in the function `yosemite_set_emacdata()`, around the line numbers reported by gdb

when the breakpoint at `yosemite_setup_arch()` was encountered. The key to understanding the anomaly is to notice the subroutine call at the very start of `yosemite_setup_arch()`. The compiler has inlined the call to `yosemite_set_emacdata()`instead of generating a function call, as would be expected by simple inspection of the source code. This inlining produced the mismatch in the line numbers when gdb hit the breakpoint. Even though the `yosemite_set_emacdata()` function was not declared using the inline keyword, GCC inlined the function as a performance optimization.

Listing 14-9

Portions of Source File yosemite.c

```
109 static void __init yosemite_set_emacdata(void)
110 {
111 struct ocp_def *def;
112 struct ocp_func_emac_data *emacdata;
113
114 /* Set mac_addr and phy mode for each EMAC */
115
116 def = ocp_get_one_device(OCP_VENDOR_IBM, OCP_FUNC_EMAC, 0);
117 emacdata = def->additions;
118 memcpy(emacdata->mac_addr, __res.bi_enetaddr, 6);
119 emacdata->phy_mode = PHY_MODE_RMII;
120
121 def = ocp_get_one_device(OCP_VENDOR_IBM, OCP_FUNC_EMAC, 1);
122 emacdata = def->additions;
123 memcpy(emacdata->mac_addr, __res.bi_enet1addr, 6);
124 emacdata->phy_mode = PHY_MODE_RMII;
125 }
126
...
304
305 static void __init
306 yosemite_setup_arch(void)
307 {
308 yosemite_set_emacdata();
309
310 ibm440gx_get_clocks(&clocks, YOSEMITE_SYSCLK, 6 * 1843200);
311 ocp_sys_info.opb_bus_freq = clocks.opb;
312
313 /* init to some ~sane value until calibrate_delay() runs */
314 loops_per_jiffy = 50000000/HZ;
315
316 /* Setup PCI host bridge */
317 yosemite_setup_hose();
318
319 #ifdef CONFIG_BLK_DEV_INITRD
320 if (initrd_start)
321 ROOT_DEV = Root_RAM0;
322 else
323 #endif
324 #ifdef CONFIG_ROOT_NFS
325 ROOT_DEV = Root_NFS;
326 #else
327 ROOT_DEV = Root_HDA1;
328 #endif
368 Chapter 14 â
```

Comments

Post new comment

Subject:

Comment: *

Allowed HTML tags: <a>   <cite> <code> <pre>   <dl> <dt> <dd> <i> <b>
Lines and paragraphs break automatically.
Use to create page breaks.

More information about formatting options (/filter/tips)

Preview