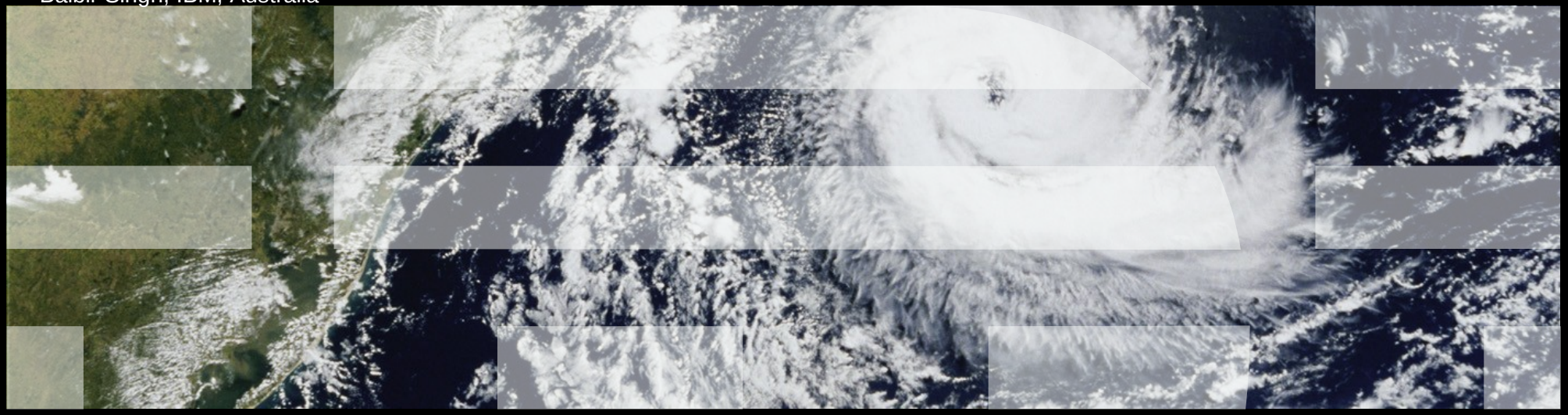
Live Patching – A review

Balbir Singh, IBM, Australia



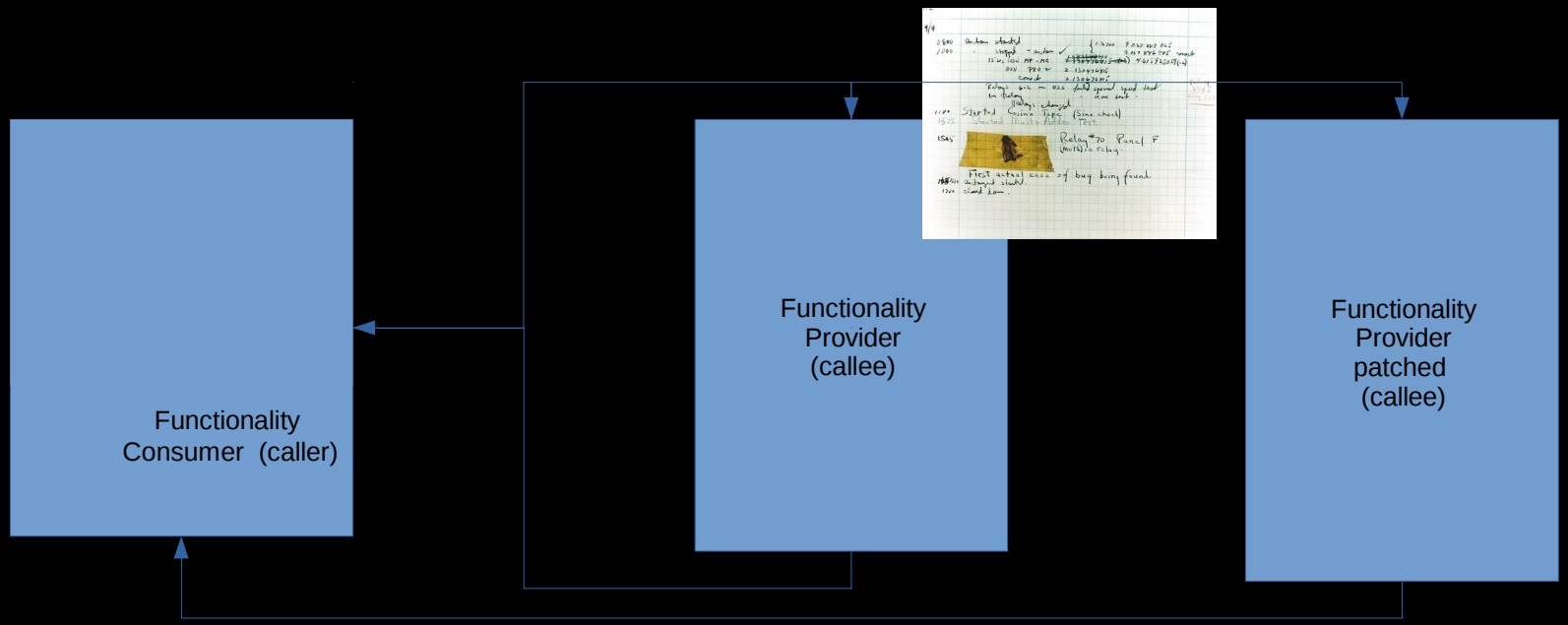
Agenda

- PPC64 port of live patching infrastructure
- Testing Live patching
- Thoughts on live patch life cycle management
- Security issues surrounding live patching

Acknowlegdements

- Michael Ellermen for all the discussions, code and design bits
- Torsten Duwe for the ppc64-le port
- Anton Blanchard for -mprofile-kernel
- Steven Rostedt for ftrace
- Abhi Chatterjee/Dipankar Sarma/Paul Mckenney for help and inspiration

Big Picture



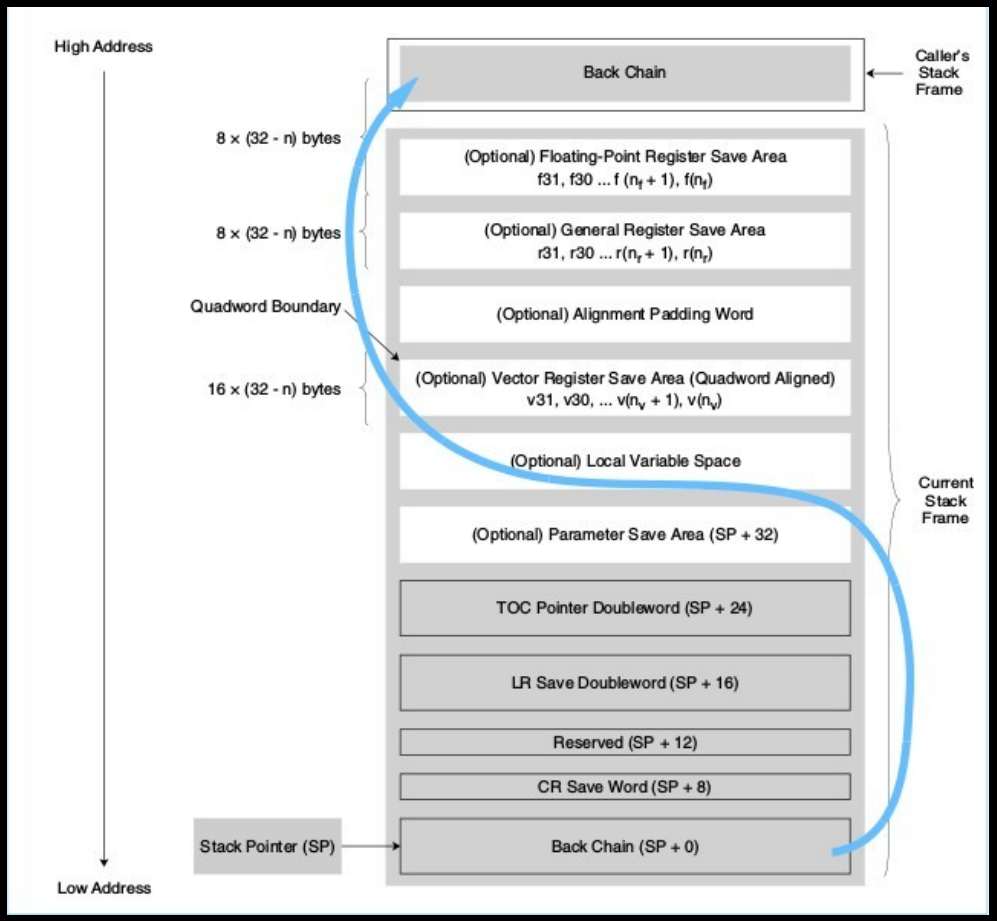
PowerPC Architecture Basics

- TOC (Table of contents)
- Call Frame (ABI)
  - Global and local entry points
  - Generated stubs
  - Role of the linker
- Modules (patches as modules)
  - Calling functions between kernel and module(s) and vice-versa

TOC

- RISC architecture – limited direct addressing
- Setup at global entry points
- Saved and restored before global function calls (next slide)

Stack Frame and ABI



Register	Preservation Rules	Purpose
r0	Volatile	Optional use in function linkage. Used in function prologues.
r1	Nonvolatile	Stack frame pointer.
r2	Nonvolatile[1]	TOC pointer.
r3 - r10	Volatile	Parameter and return values.
r11	Volatile	Optional use in function linkage. Used as an environment pointer in languages that require environment pointers.
r12	Volatile	Optional use in function linkage. Function entry address at the global entry point.
r13	Reserved	Thread pointer (see *Section 3.7 Thread Local Storage ABI* on page 110).
r14 - r31[2]	Nonvolatile	Local variables.
LR	Volatile	Link register.
CTR	Volatile	Loop count register.
TAR	Reserved	Reserved for system use. This register should not be read or written by application software.
XER	Volatile	Fixed-point exception register.
CR0 - CR1	Volatile	Condition register fields.
CR2 - CR4	Nonvolatile	Condition register fields.
CR5 - CR7	Volatile	Condition register fields.
DSCR	Limited Access	Data stream prefetch control.
VRSAVE	Reserved	Reserved for system use. This register should not be read or written by application software.

1. Register r2 is nonvolatile with respect to calls between functions in the same compilation unit. It is saved and restored by code inserted by the linker resolving a call to an external function. For more information, see *TOC Pointer Usage* on page 41.
2. If a function needs a frame pointer, assigning r31 to the role of the frame pointer is recommended.

cmdline_proc_show

```
c000000000360fd0  3c4c00ed    addis r2,r12,237
c000000000360fd4  38422e30    addi  r2,r2,11824
c000000000360fd8  7c0802a6    mflr  r0
c000000000360fdc  f8010010    std   r0,16(r1)
c000000000360fe0  4bca9145    bl    c00000000000a124  #
ftrace_caller+0x0/0xd8
c000000000360fe4  7c0802a6    mflr  r0
c000000000360fe8  f8010010    std   r0,16(r1)
c000000000360fec  f821ffa1    stdu  r1,-96(r1)
c000000000360ff0  3c82ffa7    addis r4,r2,-89
c000000000360ff4  3d220007    addis r9,r2,7
c000000000360ff8  3929c228    addi  r9,r9,-15832
c000000000360ffc  38847b28    addi  r4,r4,31528
c000000000361000  e8a90000    ld    r5,0(r9)
c000000000361004  4bfa0495    bl    c000000000301498  #
seq_printf+0x8/0x50
c000000000361008  60000000    nop
c00000000036100c  38600000    li    r3,0
```

Stub

```
d0000000017e01d0  3d62ffff    addis r11,r2,-1
d0000000017e01d4  396b7c08    addi  r11,r11,31752
d0000000017e01d8  f8410018    std   r2,24(r1)
d0000000017e01dc  e98b0020    ld    r12,32(r11)
d0000000017e01e0  7d8903a6    mtctr r12
d0000000017e01e4  4e800420    bctr
d0000000017e01e8  00000000    .long 0x0
d0000000017e01ec  73747562    andi. r20,r27,30050
d0000000017e01f0  001465e0    .long 0x1465e0
d0000000017e01f4  c0000000    lfs   f0,0(0)
```

livepatch_cmdline_proc_show

```
d0000000017e0000  3c4c0001    addis r2,r12,1
d0000000017e0004  384285c8    addi  r2,r2,-31288
d0000000017e0008  7c0802a6    mflr  r0
d0000000017e000c  f8010010    std   r0,16(r1)
d0000000017e0010  60000000    nop
d0000000017e0014  7c0802a6    mflr  r0
d0000000017e0018  f8010010    std   r0,16(r1)
d0000000017e001c  f821ffa1    stdu  r1,-96(r1)
d0000000017e0020  3c820000    addis r4,r2,0
d0000000017e0024  e8848000    ld    r4,-32768(r4)
d0000000017e0028  3ca20000    addis r5,r2,0
d0000000017e002c  e8a58008    ld    r5,-32760(r5)
d0000000017e0030  48000179    bl    d0000000017e01a8  #
livepatch_exit+0x98/0x1b0 [livepatch_sample]
d0000000017e0034  e8410018    ld    r2,24(r1)
d0000000017e0038  38600000    li    r3,0
d0000000017e003c  38210060    addi  r1,r1,96
```

Function call variants

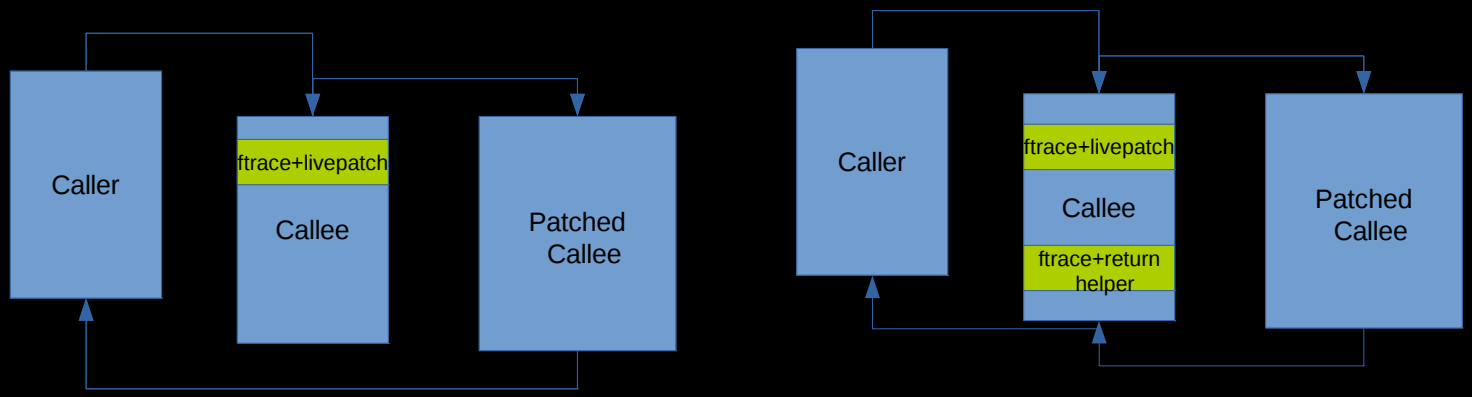
```
c0000000006b7c08  38800000    li    r4,0
c0000000006b7c0c  4bffdc8d    bl    c0000000006b5898  #
scsi_alloc_target+0x8/0x3a0
c0000000006b7c10  7c7d1b79    mr.   r29,r3

.........

c0000000006b7bc8  3be00000    li    r31,0
c0000000006b7bcc  4838ef1d    bl    c000000000a46ae8  #
mutex_unlock+0x8/0x70
c0000000006b7bd0  60000000    nop
```

Porting Livepatch

- Constraints
  - Implement FTRACE_WITH_REGS
- Local function calls become global
  - Compiler + linker can detect if toc save/restore is required in normal calls
  - With live patching, local functions become global
  - This means someone needs to save the TOC for local calls
- BE/LE considerations
- Compiler support (in our case -mprofile-kernel)
- Location of the call to mcount

Local to Global

- Local calls don't save/restore their TOC
  - Implies we have to do it for local calls turning global
- Desired call flow vs real call flow



Design alternatives

- Need for new stack frame
  - Created with livepatch_helper
  - Realized it breaks for args passed on stack
- No stack frame alternatives
  - Patch all callers (assuming the linker generated the NOP)
  - Racy, what happens if the function was called prior to patching, does it restore the right TOC?
- Hack the stack frame (4 bytes left in the frame)
- Patch hackiness (delegate saving to the patched function)
- Create an extra stack in thread_info (winner)
  - Limited length
  - Unlimited (prone to stack overflow, well, well..)

Testing

LCOV - code coverage report

Current view:	top level - kernel/livepatch		Hit	Total	Coverage
Test:	lp1.cov	**Lines:**	139	306	45.4 %
Date:	2016-10-27 22:47:19	**Functions:**	16	30	53.3 %

Filename	Line Coverage		Functions	
core.c	45.4 %	139 / 306	53.3 %	16 / 30

Generated by: LCOV version 1.12

Test Cases

- Most of the ftrace helper routines are in assembly
  - Hard to profile/validate
- We need a more comprehensive test suite
  - Live patching patched functions
  - Test local to global changes (arch specific)
  - Test with large number of arguments
  - Test with different types of arguments (VMX in the mix)
  - Functions that are patched, but there are still references to these functions (blocked inside a function to be patched)
- Can we get the patched function to do a quick self-test? Provide more data on how many times the bug was going to be hit?

Live patching design and issues?

- Ability to patch functions
  - Reuse ftrace
  - Nice/clean/easy and efficient
- Alternatives
  - Patch an individual instruction or block of instructions
  - Complex implementation, but restricts the amount of code to be validated. No stack frames
- Can the current framework be extended to user space for live patching?
- How do we decide what to live patch?
  - Dirty COW?
- Benefits of live-patching vs live cluster update?
- How long do users run their live-patches for – what is the expected life span?

Security implications

- Should we allow cross-signing for patching?
  - Can a patched binary be signed from someone other than who signed the base code?
- How do we prevent exploiters from using live-patching to load/inject changes?

Legal Statement

- This work represents the view of the author and does not necessarily represent the view of IBM.
- IBM and IBM (logo) are trademarks or registered trademarks of International Business Machines Corporation in the United States and/or other countries.
- Linux is a registered trademark of Linus Torvalds.
- Other company, product, and service names may be trademarks or service marks of others.

Discussion