Back-tracing in MIPS-based Linux Systems

Kim, Jong-Sung (jsungkim@lge.com)

LG Electronics

Agenda

- ❑ Backgrounds
- ❑ MIPS stack-frame structure
- ❑ Back-tracing in MIPS systems
- ❑ Back-tracing from the signal context
- ❑ Sample applications
- ❑ Summary
- ❑ References
- ❑ Appendix: Crash Report System applied to LGE products

Backgrounds

The MIPS Core

- Brief history
  - In 1981, a team led by John L. Hennessy at Stanford University started working on what would become the 1st MIPS processor
  - In 1984, Hennessy left Stanford to form MIPS Computer Systems
  - In 1992, SGI bought the company to guarantee the design would not be lost
  - The company became known as MIPS Technologies
- Key concepts
  - Deep instruction pipelines
  - One cycle for one instruction (eliminating interlocks)
- Core design licensing
  - Broadcom (SiByte), IDT, LSI Logic, NEC, Philips, Toshiba, …

- Very popular in developing CE products (BDP, DTV, PDA, STB, …)
- Known as rolling back stack-frames is not possible

Back-tracing

- In many cases, it's very hard and takes long time to reproduce an error
- Just-in-time debug information is very useful
  - Process/thread ID
  - Register dumps
  - Variable dumps
  - Programming language-level call-stack
  - Et cetera

- Back-tracing: extracting the function call-stack

Related Works

- __builtin_return_address function/macro inside GCC
  - Written by Richard Henderson (rth@redhat.com)
- Several just-in-time debug features inside Glibc
  - Including:
    - backtrace(3), backtrace_symbols(3), …
    - catchsegv(1), libSegFault.so
  - Written by Ulrich Drepper (drepper@redhat.com)

- However, they're not available for MIPS systems

MIPS Stack-frame Structure

MIPS Stack-frame Structure from ABI

❑ **Conceptual structure of a MIPS stack-frame**

Base	Offset	Contents	Frame
		unspecified … variable size	*High addresses*
	+16	(if present) incoming arguments passed in stack frame	Previous
old *$sp*	+0	space for incoming arguments 1-4	
		locals and temporaries	
		general register save area	Current
		floating-point register save area	
$sp	+0	argument build area	*Low addresses*

Real-world MIPS Stack-frame Structure

- Sample C function
  - Nested function
  - Two automatic variables

```
#include <dlfcn.h>
#include <stdio.h>

…

static int shared_local(void)
{
         void *dl_obj;
         int (*dl_fcn)(void);

         printf("%s\n", __FUNCTION__);

         dl_obj = dlopen("libdynamic.so", RTLD_NOW);
         dl_fcn = (int (*)(void))dlsym(dl_obj, "dynamic_global");

         return dl_fcn();
}
```

- Stack-frame structure
  - Reserved region for arguments
  - Old stack-frame pointer
  - Return address

Base	Offset	Contents	Frame
			High addresses
		unknown	Previous
Old *$sp*	+0		
	+36	return address ($ra)	
	+32	old frame pointer ($fp)	
		local variable	
	+24	local variable	
		not used	Current
	+16	old context register ($gp)	
		reserved for argument	
		reserved for argument	
		reserved for argument	
$sp	+0	reserved for argument	*Low addresses*

- Hmm.. what's the problem?
  - Variable offsets from the top of stack
  - This figure is not always true

Back-tracing in MIPS Systems

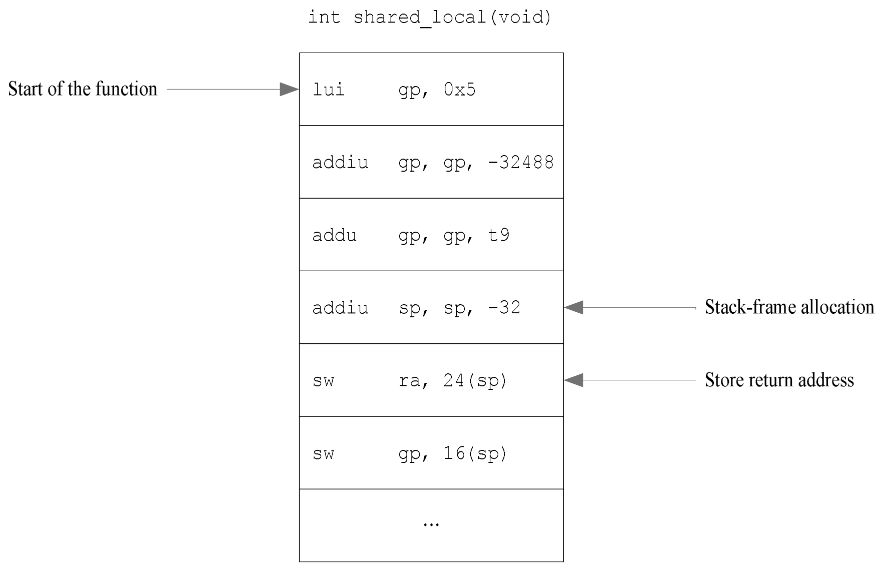
Binary Code Scanning

- The stack-frame is not enough for back-tracing
  - Previous stack-frame pointer
    - Offset from $sp is variable
    - Sometimes not saved
  - Return address
    - Offset from $sp is variable
    - Sometimes not saved (but, don’t care in this section)

- So, binary code scanning is required to acquire:
  - Current stack-frame size
  - Offset of stack-stored return address

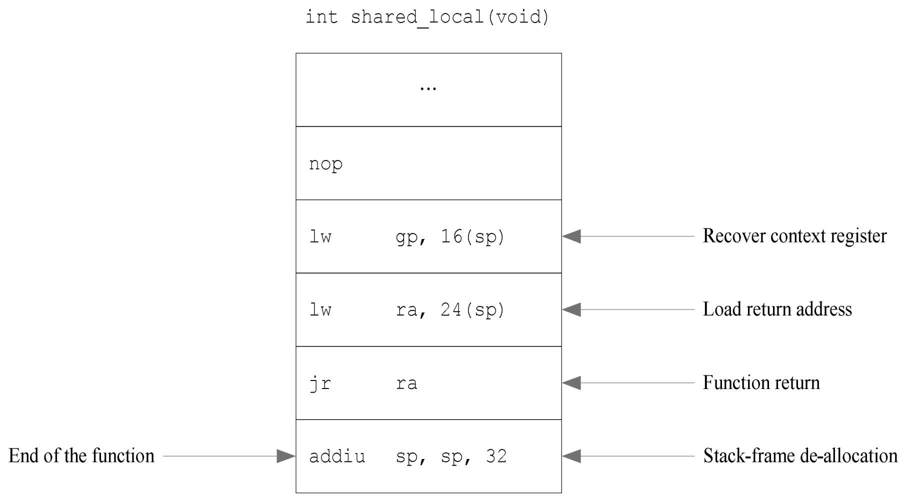
Function Prologue & Epilogue

- Prologue for a nested function
    - Context register setup
    - Current stack-frame allocation
    - Return address saving



```
lui     gp, 0x5
addiu   gp, gp, -32488
addu    gp, gp, t9
addiu   sp, sp, -32
sw      ra, 24(sp)
sw      gp, 16(sp)
...
```

- Epilogue for a nested function
    - Return address loading
    - Current stack-frame de-allocation
    - Function return



```
...
nop
lw      gp, 16(sp)
lw      ra, 24(sp)
jr      ra
addiu   sp, sp, 32
```

Back-tracing Procedure

- ❑ Initialization
  - ➲ Registers latching (ra ← $ra, sp ← $sp)
  - ➲ Code scanning for current stack-frame size
  - ➲ Adjust sp to previous stack-frame (sp ← sp + stack_size)
- ❑ Repeat until maximum depth reached or ra is zero
  - ➲ Save ra in return address buffer
  - ➲ Code scanning for current stack-frame size and offset of saved return address
  - ➲ Load return address to ra (ra ← sp[ra_offset])
  - ➲ Adjust sp to previous stack-frame (sp ← sp + stack_size)
- ❑ Return the count of the return addresses found

Instruction Formats

31 26	25 21	20 16	15 0
ADDIU 001001	rs	rt	immediate
6	5	5	16

Format:	ADDIU rt, rs, immediate
Description:	GPR[rt] ← GPR[rs] + immediate

31 26	25 21	20 16	15 0
LW 100011	base	rt	offset
6	5	5	16

Format:	LW rt, offset(base)
Description:	GPR[rt] ← memory[GPR[base] + offset]

31 26	25 21	20 16	15 0
SW 101011	base	rt	offset
6	5	5	16

Format:	SW rt, offset(base)
Description:	memory[GPR[base] + offset] ← GPR[rt]

backtrace_mips32 Function

❑ Working source code of backtrace_mips32

```
#define abs(s)  ((s) < 0 ? -(s) : (s))

int backtrace_mips32(void **buffer, int size)
{
        unsigned long *addr;
        unsigned long *ra;
        unsigned long *sp;
        size_t ra_offset;
        size_t stack_size;
        int depth;

        if(!size)
                return 0;
        if(!buffer || size < 0)
                return -EINVAL;

        // get current $ra and $sp
        __asm__ __volatile__ (
        "       move    %0, $ra\n"
        "       move    %1, $sp\n"
        : "=r"(ra), "=r"(sp)
        );

        // scanning to find the size of the current stack-frame
        stack_size = 0;

        for(addr = (unsigned long *)backtrace_mips32; !stack_size; ++addr)
        {
                if((*addr & 0xffff0000) == 0x27bd0000)
                        stack_size = abs((short)(*addr & 0xffff));
                else if(*addr == 0x03e00008)
                        break;
        }

        sp = (unsigned long *)((unsigned long)sp + stack_size);
```

```
        // repeat backward scanning
        for(depth = 0; depth < size && ra; ++depth)
        {
                buffer[depth] = ra;

                ra_offset = 0;
                stack_size = 0;

                for(addr = ra; !ra_offset || !stack_size; --addr)
                {
                        switch(*addr & 0xffff0000)
                        {
                        case 0x27bd0000:
                                stack_size = abs((short)(*addr & 0xffff));
                                break;

                        case 0xafbf0000:
                                ra_offset = (short)(*addr & 0xffff);
                                break;

                        case 0x3c1c0000:
                                return depth + 1;

                        default:
                                break;
                        }
                }

                ra = *(unsigned long **)((unsigned long)sp + ra_offset);
                sp = (unsigned long *)((unsigned long)sp + stack_size);
        }

        return depth;
}
```

Back-tracing from The Signal Context

Signal Handler Context

- **backtrace_mips32 can't handle stack-frames from signal contexts**
- **In the signal handler context:**
  - **$ra points to the code block (by kernel) in the stack**
  - **backtrace_mips32 can't handle this non-function code block**

- **To back-trace from signal contexts:**
  - **Skip the kernel-inserted code/data block by referencing the signal context structure (ucontext_t) given to the signal handler**
  - **Handle the possible leaf function at the top of the call-stack**
    - No saved return address
    - No stack-frame

Back-tracing from The Signal Context

- ❑ Initialization
  - ➲ Find $pc, $ra, and $sp from the signal context structure
    (pc ← mcontext_t::pc, ra ← mcontext_t::gregs[31], sp ← mcontext_t::gregs[29])
  - ➲ Save pc in return address buffer
  - ➲ Code scanning from pc to find stack-frame size and stored ra offset
  - ➲ If return address was stored, load it to ra (ra ← sp[ra_offset])
  - ➲ Adjust sp to previous stack-frame (sp ← sp + stack_size)
- ❑ Repeat until maximum depth reached or ra is zero
  - ➲ Save ra in return address buffer
  - ➲ Code scanning for current stack-frame size and offset of saved return address
  - ➲ Load return address to ra (ra ← sp[ra_offset])
  - ➲ Adjust sp to previous stack-frame (sp ← sp + stack_size)
- ❑ Return the count of found return addresses

sigbacktrace_mips32 Function

❑ Working source code of sigbacktrace_mips32

```c
#define abs(s)  ((s) < 0 ? -(s) : (s))

int sigbacktrace_mips32(void **buffer, int size, ucontext_t const *uc)
{
        unsigned long *addr;
        unsigned long *pc, *ra, *sp;
        size_t ra_offset, stack_size;
        int depth;

        if(size == 0)
                return 0;
        if(!buffer || size < 0 || !uc)
                return -EINVAL;

        // get current $pc, $ra and $sp
        pc = (unsigned long *)(unsigned long)uc->uc_mcontext.pc;
        ra = (unsigned long *)(unsigned long)uc->uc_mcontext.gregs[31];
        sp = (unsigned long *)(unsigned long)uc->uc_mcontext.gregs[29];

        buffer[0] = pc;

        if(size == 1)
                return 1;

        // scanning to find the size of the current stack-frame
        ra_offset = stack_size = 0;

        for(addr = pc; !ra_offset || !stack_size; --addr)
        {
                switch(*addr & 0xffff0000)
                {
                case 0x27bd0000:
                        stack_size = abs((short)(*addr & 0xffff));
                        break;

                case 0xafbf0000:
                        ra_offset = (short)(*addr & 0xffff);
                        break;

                case 0x3c1c0000:
                        goto __out_of_loop;

                default:
                        break;
                }
        }
__out_of_loop:

        if(ra_offset)
                ra = *(unsigned long **)((unsigned long)sp + ra_offset);
        if(stack_size)
                sp = (unsigned long *)((unsigned long)sp + stack_size);

        // repeat backward scanning
        for(depth = 1; depth < size && ra; ++depth)
        {
                buffer[depth] = ra;

                ra_offset = stack_size = 0;

                for(addr = ra; !ra_offset || !stack_size; --addr)
                {
                        switch(*addr & 0xffff0000)
                        {
                        case 0x27bd0000:
                                stack_size = abs((short)(*addr & 0xffff));
                                break;

                        case 0xafbf0000:
                                ra_offset = (short)(*addr & 0xffff);
                                break;

                        case 0x3c1c0000:
                                return depth + 1;

                        default:
                                break;
                        }
                }

                ra = *(unsigned long **)((unsigned long)sp + ra_offset);
                sp = (unsigned long *)((unsigned long)sp + stack_size);
        }

        return depth;
}
```

More Considerations for The Safer Back-tracing

- Leaf functions
  - Leaf functions usually don’t save registers
  - Leaf functions can run with zero-size stack-frame
- Assembly-coded or hard-optimized functions
  - These functions may not save registers
  - These functions may run with zero-size stack-frame
  - These functions may not have normal function prologue and/or epilogue
- If a function without normal function prologue is located at the first place of a loaded object, sigbacktrace will dereference illegal addresses
- Therefore, back-tracing needs hands of the loaded object/symbol table

Sample Applications

Build & Running Environment

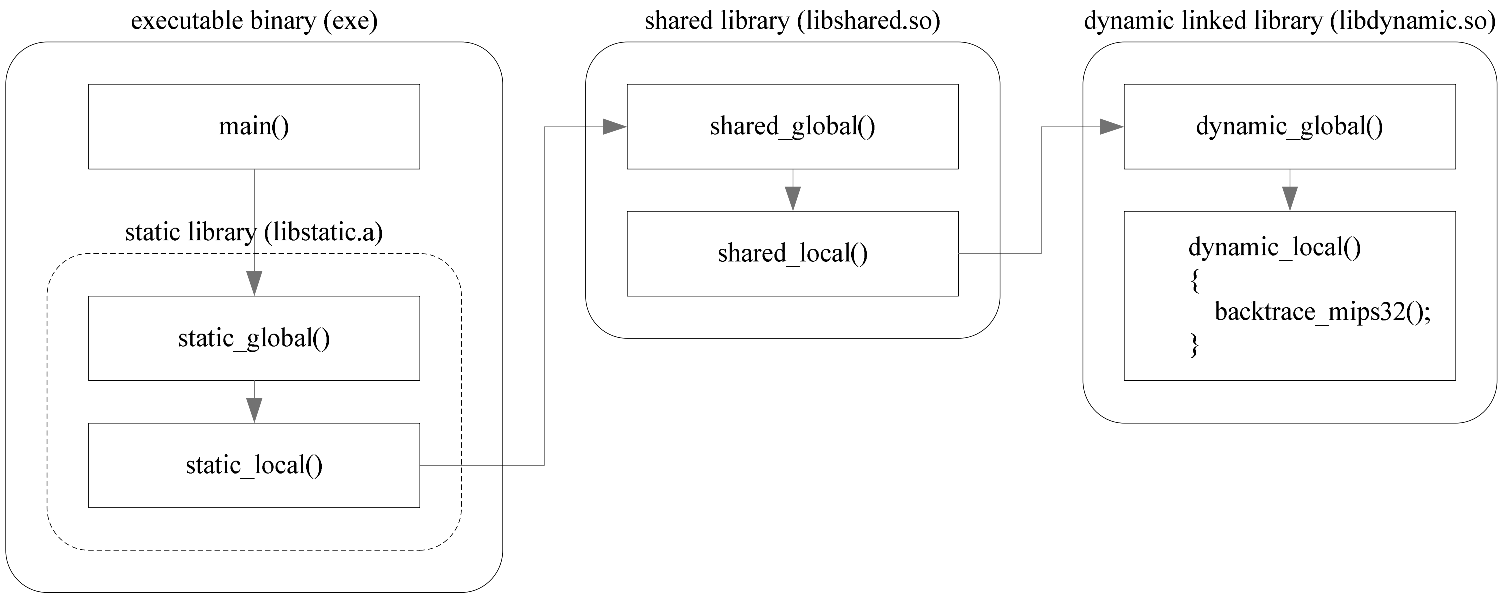
- Processor: Broadcom BCM7440P 266MHz

- Linux kernel: 2.6.12
- C library: uClibc 0.9.28
- GCC version: 3.4.6

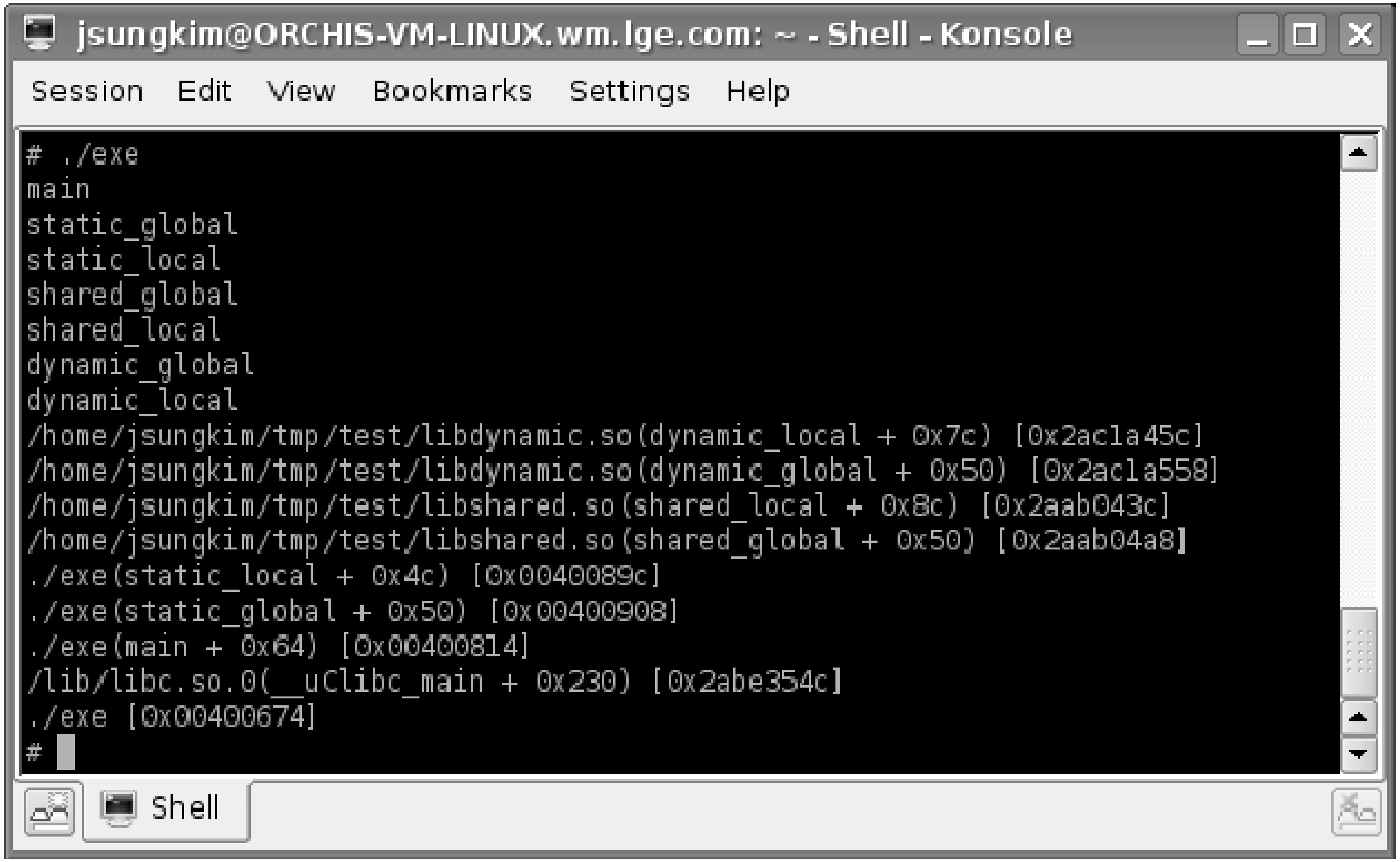
- CFLAGS: -O –W –Wall –export-dynamic –fPIC –fno-optimize-sibling-calls –g

Sample Application #1

- Simple application to test backtrace_mips32
  - Using static/shared/dynamic-loaded libraries
  - All functions print its name
  - dynamic_local dumps the call-stack using backtrace_mips32



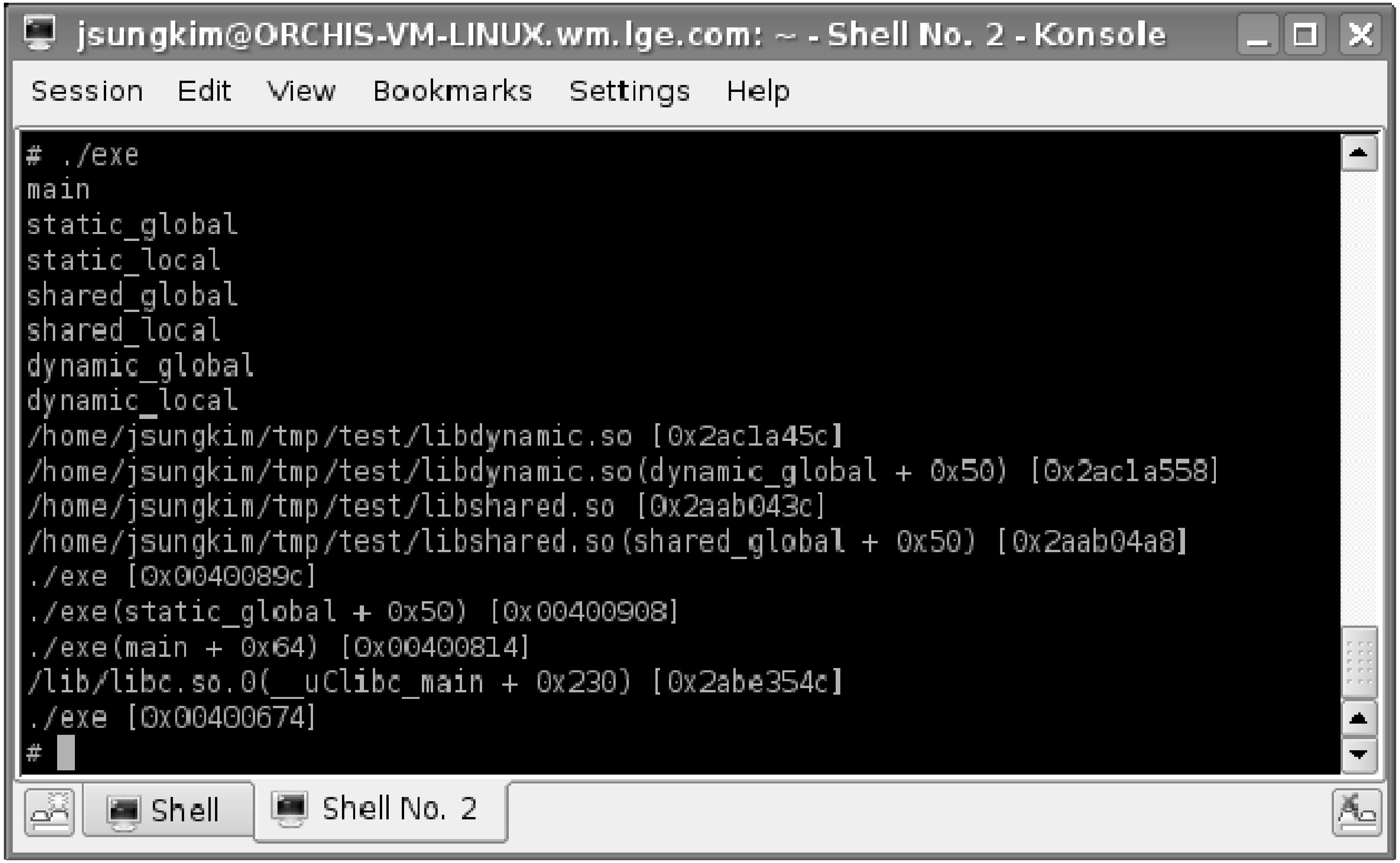
Outputs from The Application



jsungkim@ORCHIS-VM-LINUX.wm.lge.com: ~ - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

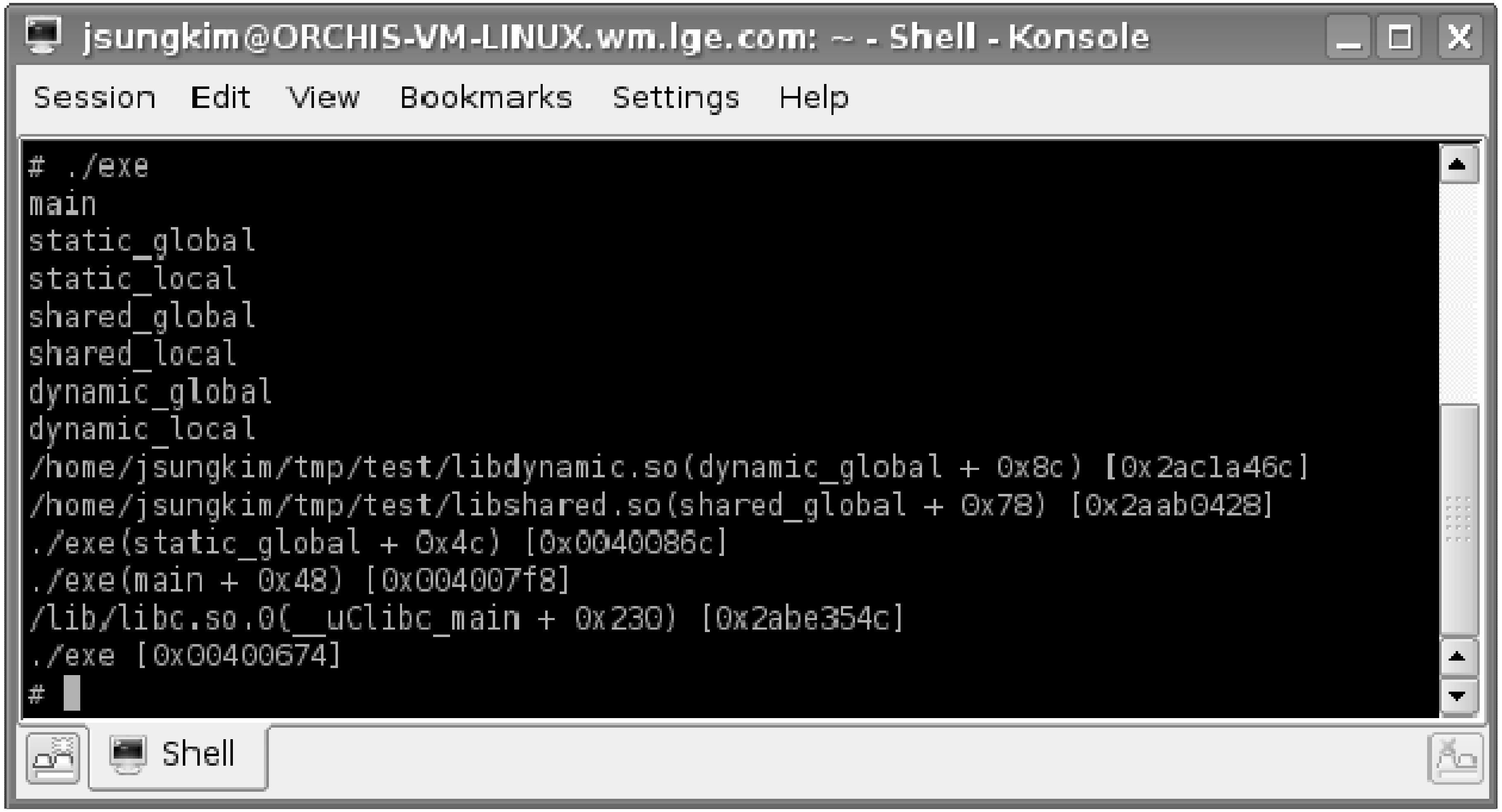
```
# ./exe
main
static_global
static_local
shared_global
shared_local
dynamic_global
dynamic_local
/home/jsungkim/tmp/test/libdynamic.so(dynamic_local + 0x7c) [0x2ac1a45c]
/home/jsungkim/tmp/test/libdynamic.so(dynamic_global + 0x50) [0x2ac1a558]
/home/jsungkim/tmp/test/libshared.so(shared_local + 0x8c) [0x2aab043c]
/home/jsungkim/tmp/test/libshared.so(shared_global + 0x50) [0x2aab04a8]
./exe(static_local + 0x4c) [0x0040089c]
./exe(static_global + 0x50) [0x00400908]
./exe(main + 0x64) [0x00400814]
/lib/libc.so.0(__uClibc_main + 0x230) [0x2abe354c]
./exe [0x00400674]
#
```

Outputs from The Application (Stripped Binaries)


jsungkim@ORCHIS-VM-LINUX.wm.lge.com: ~ - Shell No. 2 - Konsole
Session Edit View Bookmarks Settings Help
Shell
Shell No. 2

```
# ./exe
main
static_global
static_local
shared_global
shared_local
dynamic_global
dynamic_local
/home/jsungkim/tmp/test/libdynamic.so [0x2ac1a45c]
/home/jsungkim/tmp/test/libdynamic.so(dynamic_global + 0x50) [0x2ac1a558]
/home/jsungkim/tmp/test/libshared.so [0x2aab043c]
/home/jsungkim/tmp/test/libshared.so(shared_global + 0x50) [0x2aab04a8]
./exe [0x0040089c]
./exe(static_global + 0x50) [0x00400908]
./exe(main + 0x64) [0x00400814]
/lib/libc.so.0(__uClibc_main + 0x230) [0x2abe354c]
./exe [0x00400674]
# 
```

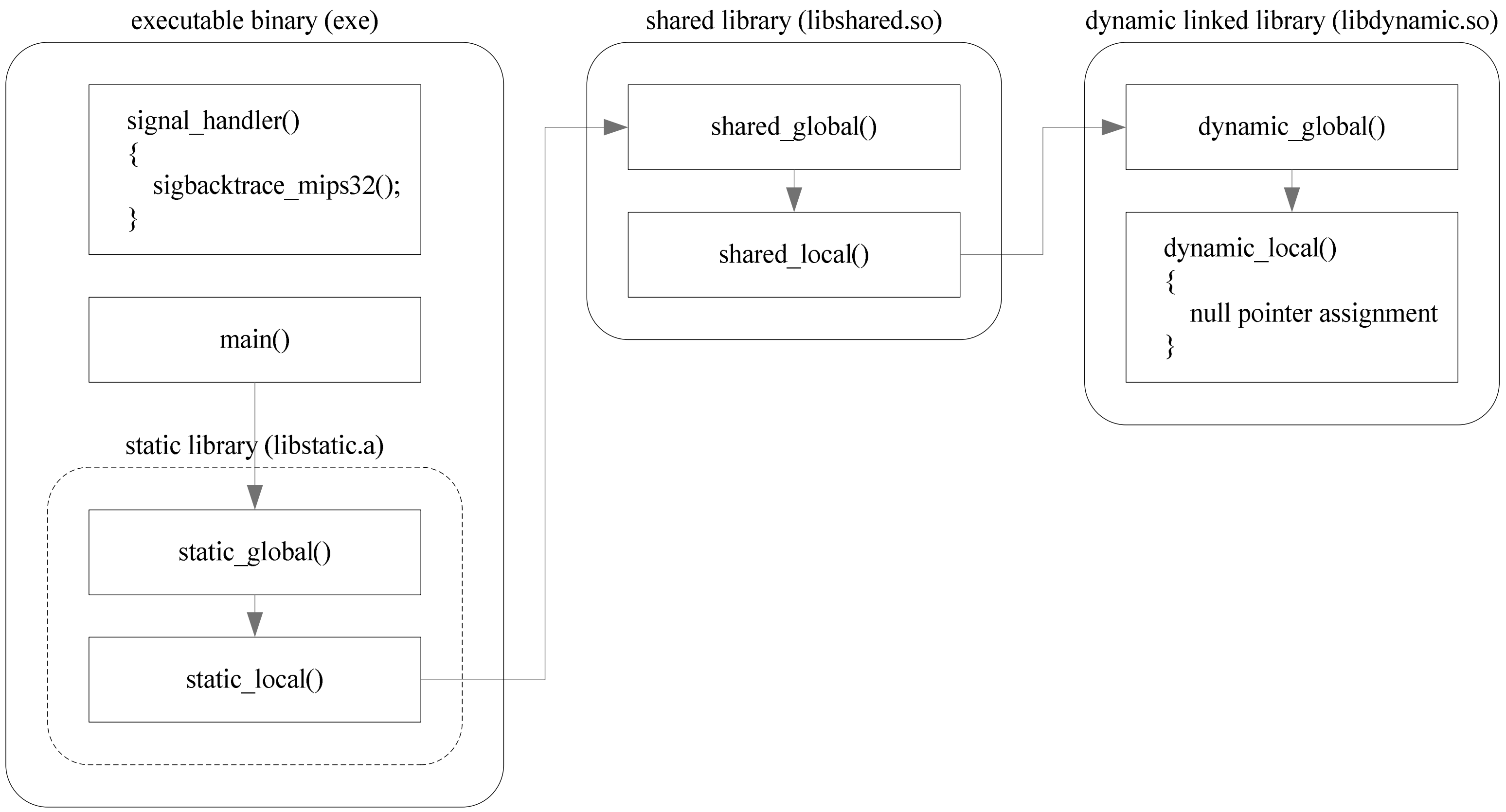
Outputs from The Application (Optimized Binaries by -O2 or -O3)


jsungkim@ORCHIS-VM-LINUX.wm.lge.com: ~ - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

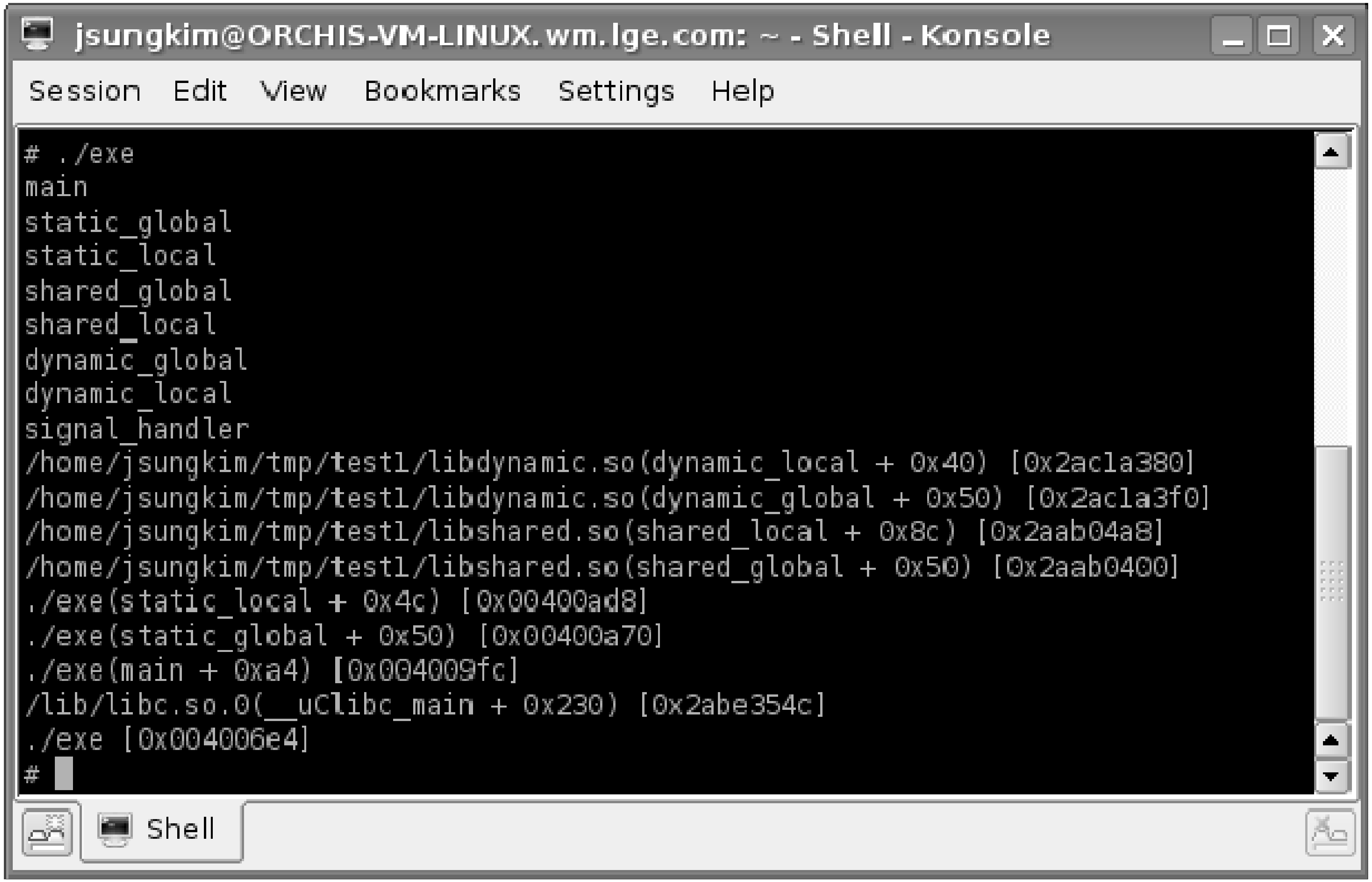
```
# ./exe
main
static_global
static_local
shared_global
shared_local
dynamic_global
dynamic_local
/home/jsungkim/tmp/test/libdynamic.so(dynamic_global + 0x8c) [0x2ac1a46c]
/home/jsungkim/tmp/test/libshared.so(shared_global + 0x78) [0x2aab0428]
./exe(static_global + 0x4c) [0x0040086c]
./exe(main + 0x48) [0x004007f8]
/lib/libc.so.0(__uClibc_main + 0x230) [0x2abe354c]
./exe [0x00400674]
#
```

Sample Application #2

- Same with sample application #1, except:
  - dynamic_local tries null-pointer assignment
  - sigbacktrace_mips32 is called from the (SIGSEGV handling) signal context


executable binary (exe)
signal_handler() { sigbacktrace_mips32(); }
main()
static library (libstatic.a)
static_global()
static_local()
shared library (libshared.so)
shared_global()
shared_local()
dynamic linked library (libdynamic.so)
dynamic_global()
dynamic_local() { null pointer assignment }

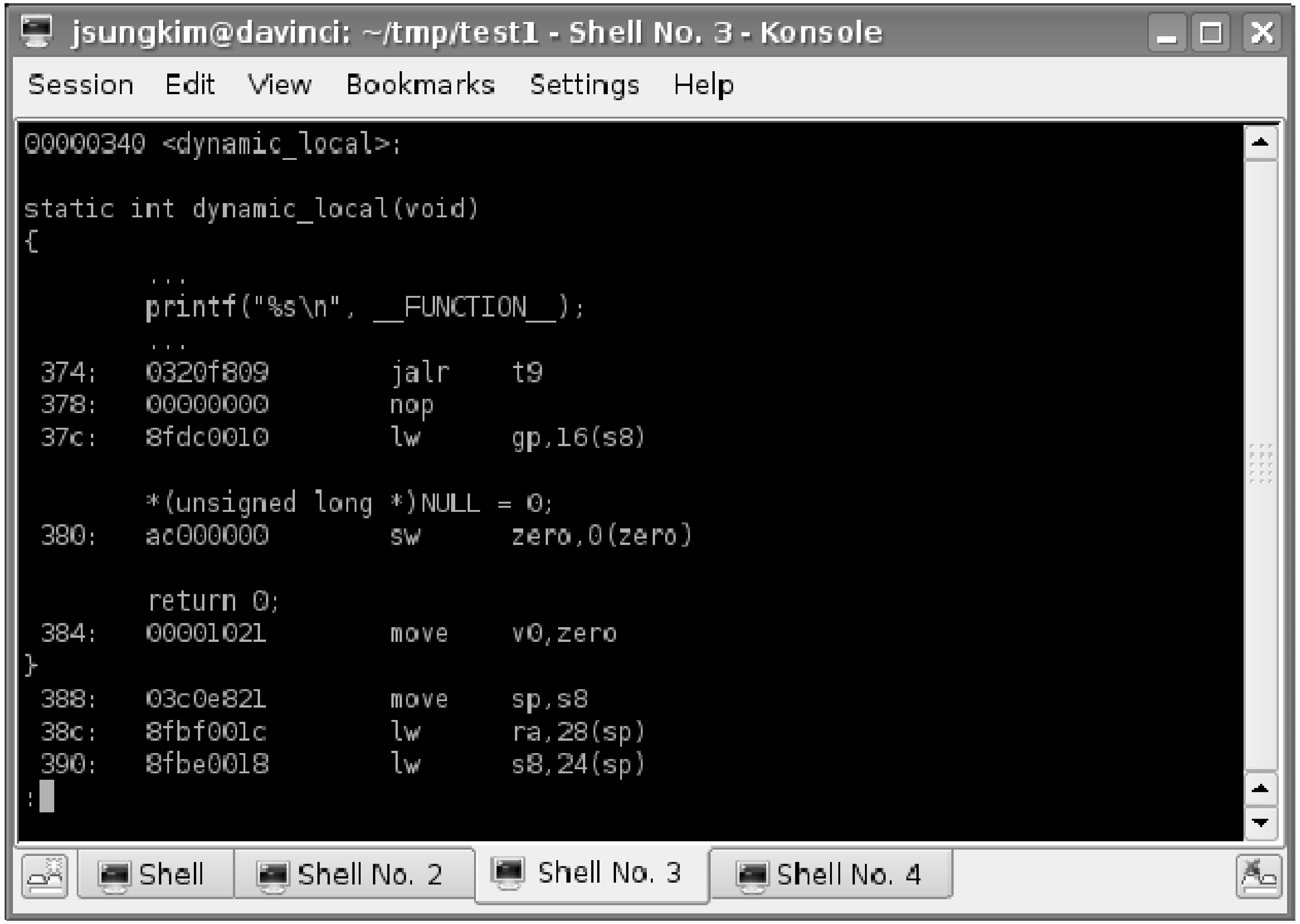
Outputs from The Application


jsungkim@ORCHIS-VM-LINUX.wm.lge.com: ~ - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell

```
# ./exe
main
static_global
static_local
shared_global
shared_local
dynamic_global
dynamic_local
signal_handler
/home/jsungkim/tmp/test1/libdynamic.so(dynamic_local + 0x40) [0x2ac1a380]
/home/jsungkim/tmp/test1/libdynamic.so(dynamic_global + 0x50) [0x2ac1a3f0]
/home/jsungkim/tmp/test1/libshared.so(shared_local + 0x8c) [0x2aab04a8]
/home/jsungkim/tmp/test1/libshared.so(shared_global + 0x50) [0x2aab0400]
./exe(static_local + 0x4c) [0x00400ad8]
./exe(static_global + 0x50) [0x00400a70]
./exe(main + 0xa4) [0x004009fc]
/lib/libc.so.0(__uClibc_main + 0x230) [0x2abe354c]
./exe [0x004006e4]
#
```

Accompanied to objdump Utility

❑ **If we have binaries compiled with “-g” option…**


jsungkim@davinci: ~/tmp/test1 - Shell No. 3 - Konsole
Session Edit View Bookmarks Settings Help
Shell
Shell No. 2
Shell No. 3
Shell No. 4

```
00000340 <dynamic_local>:

static int dynamic_local(void)
{
        ...
        printf("%s\n", __FUNCTION__);
        ...
 374:   0320f809        jalr    t9
 378:   00000000        nop
 37c:   8fdc0010        lw      gp,16(s8)

        *(unsigned long *)NULL = 0;
 380:   ac000000        sw      zero,0(zero)

        return 0;
 384:   00001021        move    v0,zero
}
 388:   03c0e821        move    sp,s8
 38c:   8fbf001c        lw      ra,28(sp)
 390:   8fbe0018        lw      s8,24(sp)
:
```

Wrap-up

Summary

- ❑ Back-tracing in the MIPS needs some code inspections
- ❑ Back-tracing from the signal context needs some more handlings
- ❑ Working backtrace/sigbacktrace functions are presented

- ❑ Now I’m working on making these functions as an open-source library or inside MIPS-ports of C libraries

References

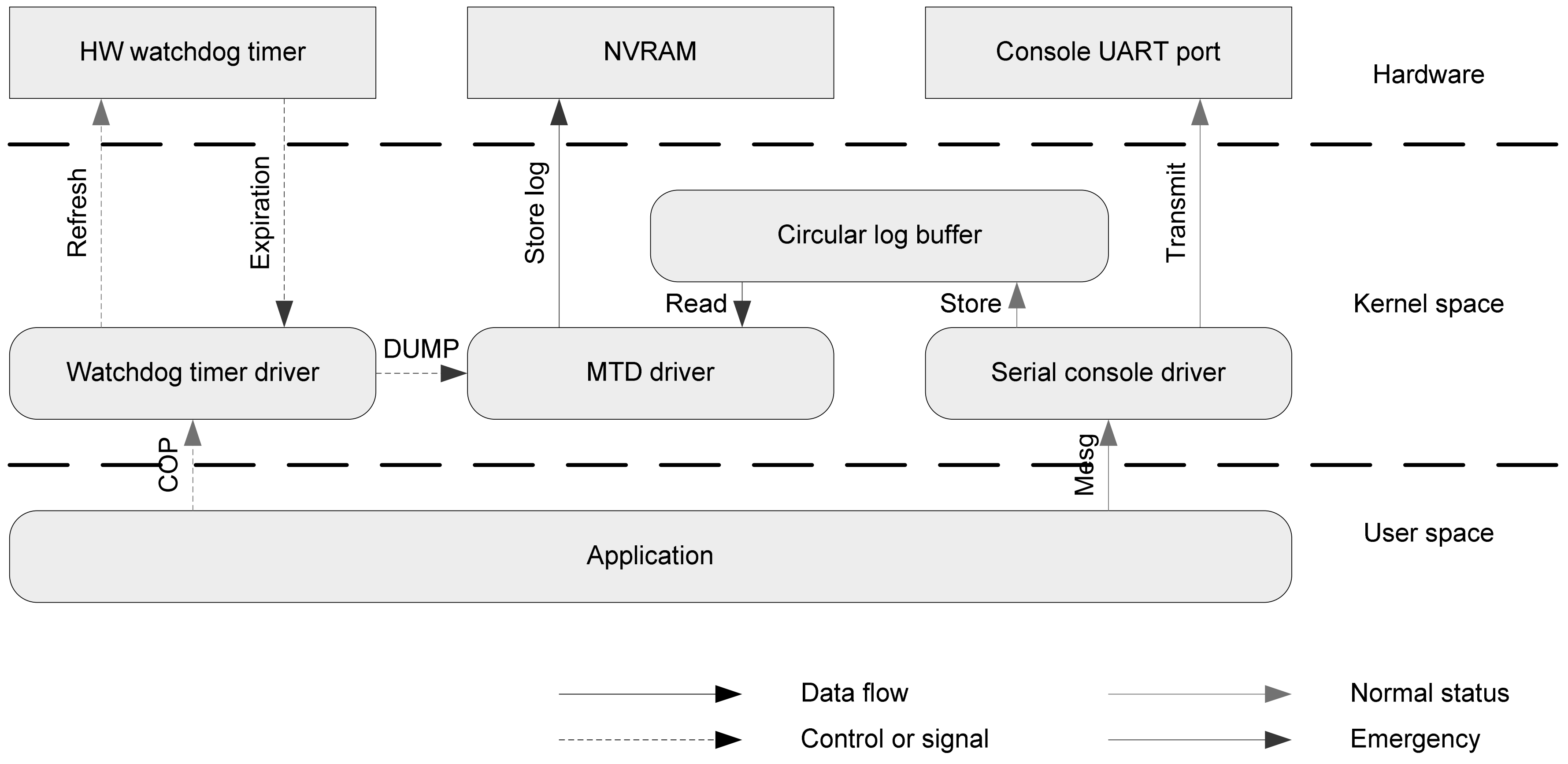
- Documents
  - MIPS32® Architecture For Programmers – Volume I: Introduction to the MIPS32® Architecture
  - MIPS32® Architecture For Programmers – Volume II: The MIPS32® Instruction Set
  - System V Application Binary Interface – MIPS® RISC Processor Supplement, 3rd Edition
  - Using the GNU Compiler Collection
- Internet resources
  - MIPS Architecture – History

Appendix: Crash Report System Applied to LGE Products

Crash Report System

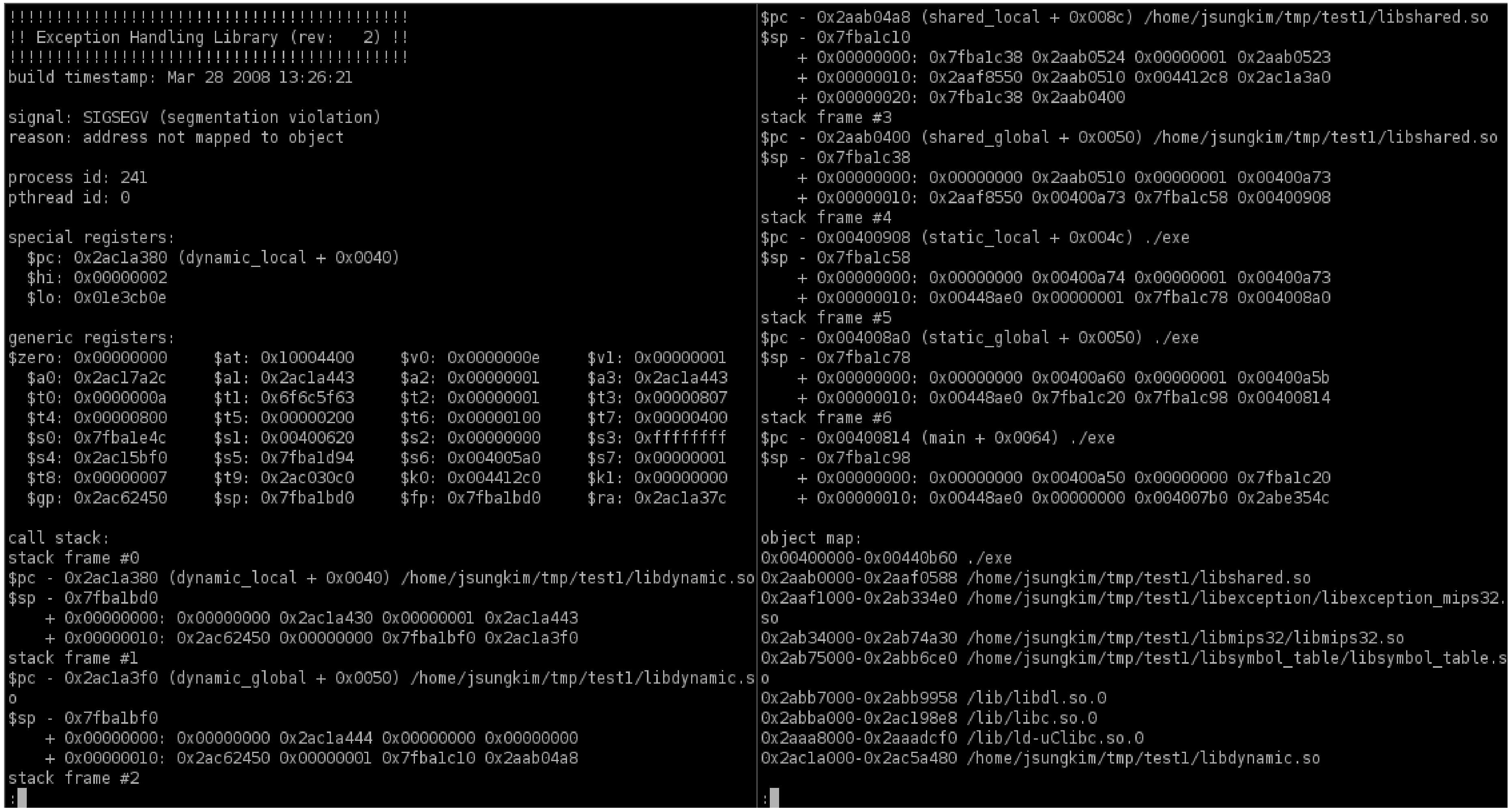
- ❑ **Purpose**
  - ➲ **Guarantee not to lose in-time information of system crashes**
  - ➲ **Easy extraction of in-time information**
    - /proc filesystem entry
    - Extractable to a USB drive
- ❑ **With Crash Report System…**
  - ➲ **All console output is stored on a circular log buffer**
  - ➲ **On watchdog expiration, the captured log is stored to an NVRAM**
  - ➲ **Developers can extract the stored log later**
  - ➲ **The stored log includes the just-in-time debug information**

Block Diagram



In-time Debug Information

- In-time debug information by the sample application



```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!! Exception Handling Library (rev:   2) !!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
build timestamp: Mar 28 2008 13:26:21

signal: SIGSEGV (segmentation violation)
reason: address not mapped to object

process id: 241
pthread id: 0

special registers:
  $pc: 0x2ac1a380 (dynamic_local + 0x0040)
  $hi: 0x00000002
  $lo: 0x01e3cb0e

generic registers:
$zero: 0x00000000     $at: 0x10004400     $v0: 0x0000000e     $v1: 0x00000001
  $a0: 0x2ac17a2c     $a1: 0x2ac1a443     $a2: 0x00000001     $a3: 0x2ac1a443
  $t0: 0x0000000a     $t1: 0x6f6c5f63     $t2: 0x00000001     $t3: 0x00000807
  $t4: 0x00000800     $t5: 0x00000200     $t6: 0x00000100     $t7: 0x00000400
  $s0: 0x7fba1e4c     $s1: 0x00400620     $s2: 0x00000000     $s3: 0xffffffff
  $s4: 0x2ac15bf0     $s5: 0x7fba1d94     $s6: 0x004005a0     $s7: 0x00000001
  $t8: 0x00000007     $t9: 0x2ac030c0     $k0: 0x004412c0     $k1: 0x00000000
  $gp: 0x2ac62450     $sp: 0x7fba1bd0     $fp: 0x7fba1bd0     $ra: 0x2ac1a37c

call stack:
stack frame #0
$pc - 0x2ac1a380 (dynamic_local + 0x0040) /home/jsungkim/tmp/test1/libdynamic.so
$sp - 0x7fba1bd0
    + 0x00000000: 0x00000000 0x2ac1a430 0x00000001 0x2ac1a443
    + 0x00000010: 0x2ac62450 0x00000000 0x7fba1bf0 0x2ac1a3f0
stack frame #1
$pc - 0x2ac1a3f0 (dynamic_global + 0x0050) /home/jsungkim/tmp/test1/libdynamic.s
o
$sp - 0x7fba1bf0
    + 0x00000000: 0x00000000 0x2ac1a444 0x00000000 0x00000000
    + 0x00000010: 0x2ac62450 0x00000001 0x7fba1c10 0x2aab04a8
stack frame #2
:
```

```
$pc - 0x2aab04a8 (shared_local + 0x008c) /home/jsungkim/tmp/test1/libshared.so
$sp - 0x7fba1c10
    + 0x00000000: 0x7fba1c38 0x2aab0524 0x00000001 0x2aab0523
    + 0x00000010: 0x2aaf8550 0x2aab0510 0x004412c8 0x2ac1a3a0
    + 0x00000020: 0x7fba1c38 0x2aab0400
stack frame #3
$pc - 0x2aab0400 (shared_global + 0x0050) /home/jsungkim/tmp/test1/libshared.so
$sp - 0x7fba1c38
    + 0x00000000: 0x00000000 0x2aab0510 0x00000001 0x00400a73
    + 0x00000010: 0x2aaf8550 0x00400a73 0x7fba1c58 0x00400908
stack frame #4
$pc - 0x00400908 (static_local + 0x004c) ./exe
$sp - 0x7fba1c58
    + 0x00000000: 0x00000000 0x00400a74 0x00000001 0x00400a73
    + 0x00000010: 0x00448ae0 0x00000001 0x7fba1c78 0x004008a0
stack frame #5
$pc - 0x004008a0 (static_global + 0x0050) ./exe
$sp - 0x7fba1c78
    + 0x00000000: 0x00000000 0x00400a60 0x00000001 0x00400a5b
    + 0x00000010: 0x00448ae0 0x7fba1c20 0x7fba1c98 0x00400814
stack frame #6
$pc - 0x00400814 (main + 0x0064) ./exe
$sp - 0x7fba1c98
    + 0x00000000: 0x00000000 0x00400a50 0x00000000 0x7fba1c20
    + 0x00000010: 0x00448ae0 0x00000000 0x004007b0 0x2abe354c

object map:
0x00400000-0x00440b60 ./exe
0x2aab0000-0x2aaf0588 /home/jsungkim/tmp/test1/libshared.so
0x2aaf1000-0x2ab334e0 /home/jsungkim/tmp/test1/libexception/libexception_mips32.
so
0x2ab34000-0x2ab74a30 /home/jsungkim/tmp/test1/libmips32/libmips32.so
0x2ab75000-0x2abb6ce0 /home/jsungkim/tmp/test1/libsymbol_table/libsymbol_table.s
o
0x2abb7000-0x2abb9958 /lib/libdl.so.0
0x2abba000-0x2ac198e8 /lib/libc.so.0
0x2aaa8000-0x2aaadcf0 /lib/ld-uClibc.so.0
0x2ac1a000-0x2ac5a480 /home/jsungkim/tmp/test1/libdynamic.so

:
```