# Episodes of LKST
# for Embedded Linux Systems

LINEO Solutions

Lineo Solutions, Inc

## Presentation Overview

- Our Motivation & Objective, Focus of Interests
- LKST Tutorial
- “Porting to Embedded” Status
- Episodes acquired from the porting
  - Development of Cross Environments, and Porting to Various Architectures
  - Challenges with ideas and Benchmarking measurement
- Other tracing technologies
  - Kprobes for SH, SystemTap for SH
    - Lineo Experienced as cooperative works with Hitachi-san
- Summary

# Focus of Interests

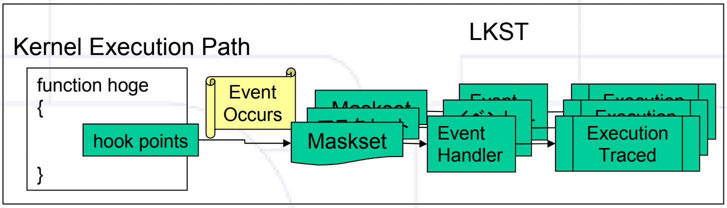
- Linux Kernel Tracing Technology
  - LKST ... Simple Mechanism with many advantages
  - (Rigid and) Static hookpoints, light overhead
    - Cf: (Flexible and ) dynamic tracer such as Kprobes
    - Relatively easy to maintain
- Potentially Possible to Contribute to Improve Linux in Numerical Quantification Aspect
  - Kernel behavior is apparently different from debugger
    - Trace data are collected during the kernel continues running.
  - For example, to Provide / Support Performance Evaluation (Plans, exams and analyses with Visualization)

# LKST Tutorial

- Software Framework of LKST
  - Required at the beginning of tracing
    - Kernel patches
      - Hookpoints are implemented in corresponding kernel codes
        - » (linux/, arch/xxx, etc.)
      - LKST core (in Kernel Space) in drivers/lkst
    - LKST packages
      - Event Handers are implemented in lkst drivers
      - User Commands
        - » To Control LKST core and Ring Buffers (lkst)
        - » To Contol Masksets (lkstm)
        - » To Control Buffer Operations (lkstbuf)
      - Analyzers (lkstlogtools, etc.)
  - Static Tracer Principles
    - Simple Mechanism such as “printk”
      - Lines of Patch is Proportional to Number of Hookpoints

# LKST Tutorial

- Mechanism of LKST
  - Embedded Hookpoints in Kernel Sources
  - Acts in Tracing is Configurable by Masksets and Event Handlers
  - Event-driven Tracing Processing
  - Configurable Without Stopping the Kernel
  - High Degree of Freedom to Customize
  - Light Kernel Overhead



Kernel Execution Path

LKST

function hoge
{
}

hook points

Event Occurs

Maskset

Event Handler

Execution Traced

# LKST Tutorial

Usage

The basic procedure for tracing lkst data is written in “howto.txt” under lkst-2.3.2.tar.gz

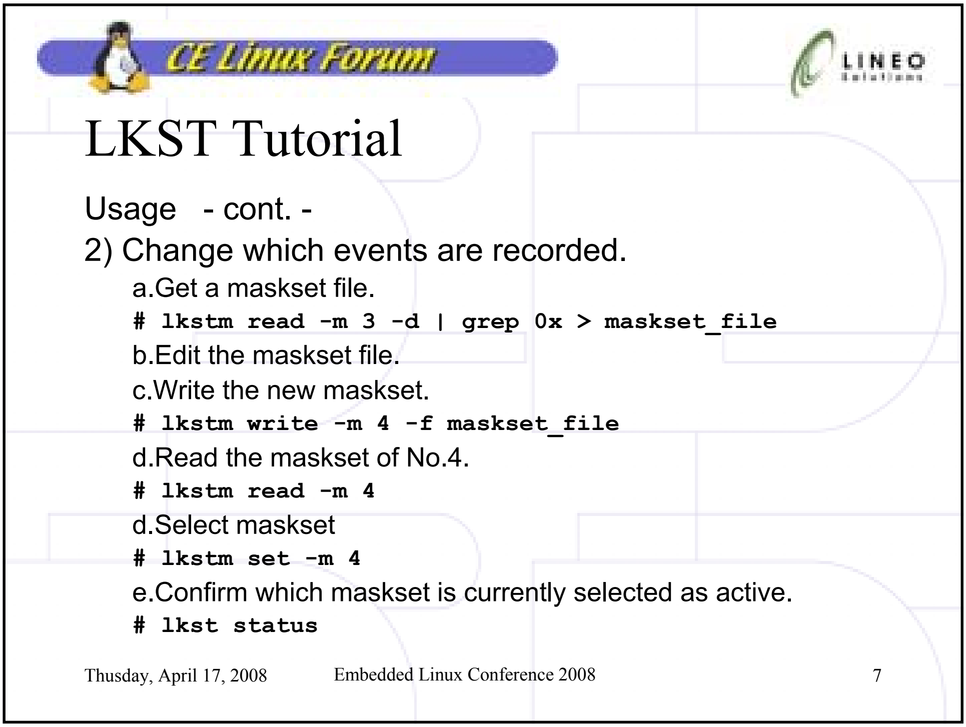
1) Display the present kernel trace data

a. Get a log buffer from kernel

```
% lkstbuf read -f logfile
```

b. Display the trace data

```
% lkstbuf print -f logfile
```



# LKST Tutorial

Usage - cont. -

2) Change which events are recorded.

a.Get a maskset file.

```
# lkstm read -m 3 -d | grep 0x > maskset_file
```

b.Edit the maskset file.

c.Write the new maskset.

```
# lkstm write -m 4 -f maskset_file
```

d.Read the maskset of No.4.

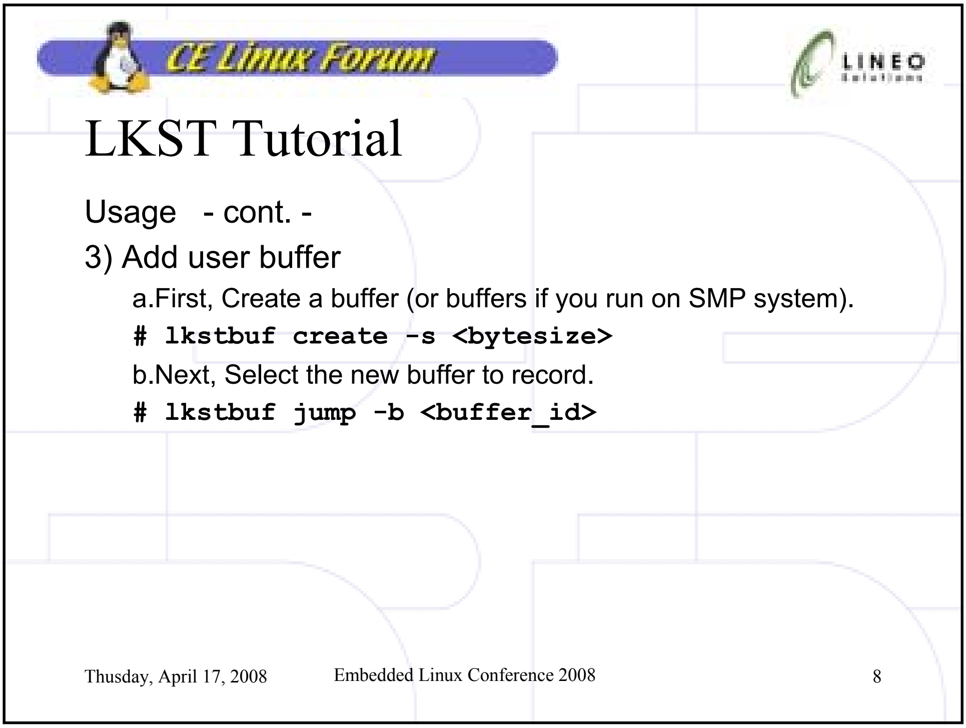
```
# lkstm read -m 4
```

d.Select maskset

```
# lkstm set -m 4
```

e.Confirm which maskset is currently selected as active.

```
# lkst status
```



# LKST Tutorial

Usage - cont. -

3) Add user buffer

a.First, Create a buffer (or buffers if you run on SMP system).

```
# lkstbuf create -s <bytesize>
```

b.Next, Select the new buffer to record.

```
# lkstbuf jump -b <buffer_id>
```

# LKST Tutorial

Hookpoint Code Example … kernel/sched.c (linux-2.6.18.8)

```
static int try_to_wake_up(task_t * p, unsigned int state, int
   sync)
{
        int cpu, this_cpu, success = 0;
        unsigned long flags;
        long old_state;
        runqueue_t *rq;
#ifdef CONFIG_SMP
        unsigned long load, this_load;
        struct sched_domain *sd;
        int new_cpu;
#endif
        LKST_HOOK(LKST_ETYPE_PROCESS_WAKEUP,
                  LKST_ARGP(p), LKST_ARG(state),
                  LKST_ARG(sync), LKST_ARG(0));
        rq = task_rq_lock(p, &flags);
        schedstat_inc(rq, ttwu_cnt);
        old_state = p->state;
```

# “Porting to Embedded” Status

- Patch submissions
  - MIPS(TX49) … Hitachi
  - ARM(OMAP1) … Hitachi
  - SH-4(RTS7751R2D) … Hitachi, Renesas, Lineo Solutions
    - http://sourceforge.net/tracker/?group_id=41854&atid=431465
- CELF presentations & demonstrations
  - Plenary Meeting, International Technical Jamboree (2005)
  - ELC - Kprobes for SH (2006), SystemTap for SH (2007)

    … Hitachi, Lineo Solutions

# “Porting to Embedded” Status

| ARCH | Board | Kernel | LKST |
|---|---|---|---|
| X86 |  | 2.6.9<br>2.6.12 | 2.2.1 -<br>2.3.2 |
|  | VIA EPIA ME6000 | 2.6.18.8 | 2.3.2 |
| SH-4 | Renesas RTS7751R2D<br>R0P751RLC0011RL<br>MS7763SE01 | 2.6.9<br>2.6.14.4<br>2.6.16.29 | 2.2.1<br>2.3.2<br>2.3.2 |
| ARM<br>(Ongoing) | PCIMX31ADS<br>KMC KZM-ARM11-01<br>M9328MX21 ADS | 2.6.16.19<br>2.6.16.19<br>2.6.16.34 | 2.3.2<br>2.3.2<br>2.3.2 |
| MIPS (Ongoing) | RBTX4938 | 2.6.18.8 | 2.3.2 |
| PA (Ongoing) | TD-BD-MPC8347EMB | 2.6.18.8 | 2.3.2 |

# Episodes acquired from the porting

Breaking Down “Apply to Embedded,” Numerous (essentially Challenging) “HURDLES” Were Found in Practical Tasks

“HURDLES”

1. Development of Cross Environments
   - Endian Conversion in Cross Environments
   - Experiences of Porting to Various Architectures
2. Challenges
   - Ideas to Improve Tracing System for Efficient “Analysis Circulation.”
   - Overhead of LKST by Measuring Benchmarking

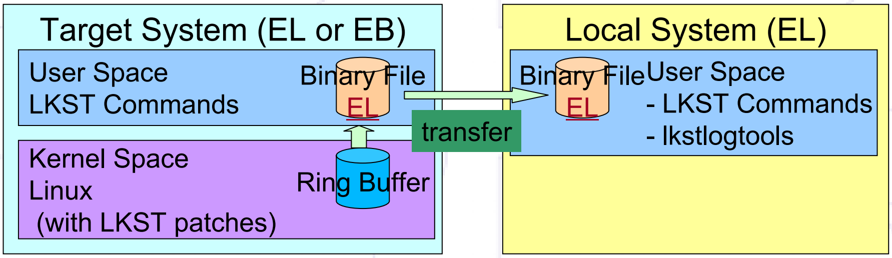
# Episodes acquired from the porting

## Development of Cross Environments

- Target System is on either Big or Little Endian.

  Entrusting the Analyze Function to Local System, the Target can Concentrate on Data-Collecting Tasks.

  … Smart as System Configuration
- Local System is on Little Endian (assuming X86 PCs)



# Episodes acquired from the porting

## Development of Cross Environments

Endian flag is in header of binary log data

```
include/linux/lkst_buffer.h
struct lkst_log_buffer {
        int cpu;                        /* cpu number */
        size_t read_size;               /* size of event records to read */
        size_t result_read_size;        /* size of read event records */
        struct timeval xtime;           /* xtime */
        lkst_tsc_t tsc;                 /* machine cycle */
                                        /* These two will be used to calculate
                                         * time of events in real time. */
        lkst_cpu_freq_t cpu_freq;       /* cpu clockspeed in kHz */
        struct lkst_log_record *buffer;
                        /* address of a buffer to store event records */

        int endian_big;                 /* byte order, 0 if little endian */
        int buf_ver;                    /* LKST buffer version */
        char arch[LKST_ARCH_NAME_LEN];  /* Architecture name */
        lkst_buffer_id id;              /* event buffer ID */
};
```

# Episodes acquired from the porting

## Development of Cross Environments

- Proposal of that “Binary log file is unified on Little Endian.”
  - As for the format of the binary log for example, please refer to `struct log_header_t` in `include/linux/lkst_buf.h`.
- Newly proposed “Endian free version of lkstbuf command” always writes binary log on Little Endian, regardless of the endianness of lkstbuf itself.
  - “BSWAP” function introduced in `lkst-2.3.2/lkstutils/buffer.c`

```
#if (LKST_BIG_ENDIAN == 1)
#define BSWAP(a){ ¥
  int s = sizeof(a); ¥
  if (s == 2) { ¥
    a = bswap_16(a); ¥
  } else if (s == 4) { ¥
    a = bswap_32(a); ¥
  } else if (s >= 8) { ¥
    a = bswap_64(a); ¥
  } ¥
}
#else
#define BSWAP(a)
#endif
```

# Episodes acquired from the porting

## Challenges

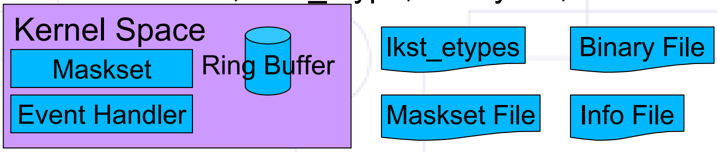
1. Ideas to Improve Tracing System for Efficient “Analysis Circulation.”
   - Management Mechanism for parameter Files (binary log, lkst_etypes, mask)
   - Categorizing of the Patch Files
   - Static Tracing for amount of data
2. Overhead of LKST by Measuring Benchmarking

# Episodes acquired from the porting

## Challenges

Idea of Management Mechanism for parameter Files (binary log, lkst_etypes, mask)

- "Info File" would Integrate the Tracing System, Making Easy to Manage the Data Collected.
- The "Info File" may contain Target Board Info, Maskset File, Lkst_etype, binary file, etc.



# Episodes acquired from the porting

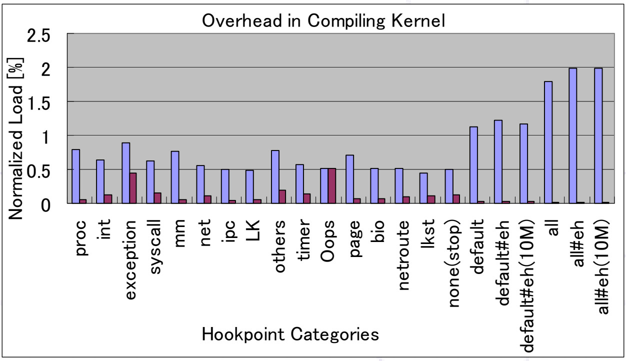
## Challenges

- Other Ideas
  - Categorizing of the Patch Files
    - Fast implement (light-weight LKST) - fast evaluation – full implementation (full LKST) cycle
    - Aiming Efficient Development of Kernel Patches
    - Major/Arch-independent/Common Parts with High Priorities (such as Context Switching, Memory Management).
  - Static Tracing
    - Current lkst Driver reads Ring Buffer From Starting Position to Current Position
      - File size written in User space changes in size every time due to the dynamic starting/current positions of Ring Buffer.
    - Entire Ring Buffer writing mechanism would be optionally appreciated.

# Episodes acquired from the porting

## Challenges

- Benchmarking measurement (1)



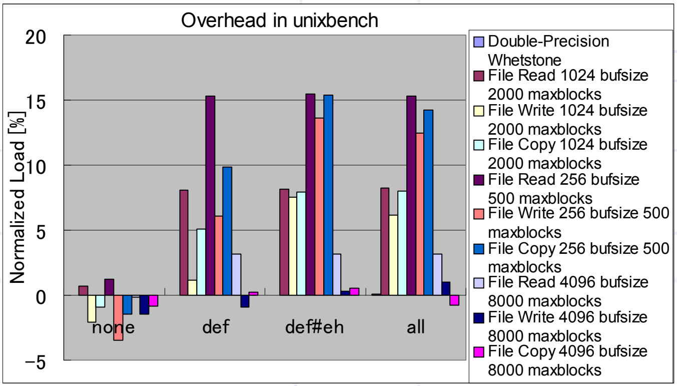
$$\text{Load} = \frac{(\text{dur}_{\text{LKST}} - \text{dur}_{\text{NO}})}{\text{dur}_{\text{NO}}} \text{X } 100[\%]$$

Target Board : VIA ME6000 (VIA C3-600MHz, RAM-128MB)
Linux-2.6.18.8_lkst-2.3.2
RootFS: NFS
Compiling Environments: Native toolchain installed in RootFS

# Episodes acquired from the porting

## Challenges

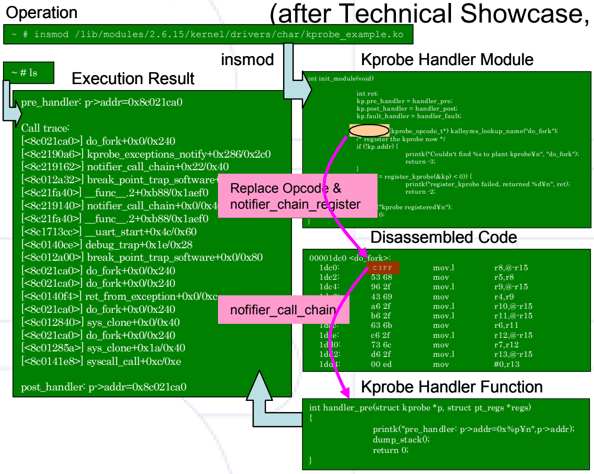
- Benchmarking measurement (2)
  - Filesystem … Standalone (HDD)



# Other tracing technologies

## Mechanism of kprobes for SH

(after Technical Showcase, ELC 2006)



```
~ # insmod /lib/modules/2.6.15/kernel/drivers/char/kprobe_example.ko
```

```
~ # ls
```

```
pre_handler: p->addr=0x8c021ca0

Call trace:
[<8c021ca0>] do_fork+0x0/0x240
[<8c2190a6>] kprobe_exceptions_notify+0x286/0x2c0
[<8c219162>] notifier_call_chain+0x22/0x40
[<8c012a32>] break_point_trap_software+
[<8c21fa40>] __func__.2+0xb88/0x1aef0
[<8c219140>] notifier_call_chain+0x0/0x4
[<8c21fa40>] __func__.2+0xb88/0x1aef0
[<8c1713cc>] __uart_start+0x4c/0x60
[<8c0140ce>] debug_trap+0x1e/0x28
[<8c012a00>] break_point_trap_software+0x0/0x80
[<8c021ca0>] do_fork+0x0/0x240
[<8c021ca0>] do_fork+0x0/0x240
[<8c0140f4>] ret_from_exception+0x0/0xc
[<8c021ca0>] do_fork+0x0/0x240
[<8c012840>] sys_clone+0x0/0x40
[<8c021ca0>] do_fork+0x0/0x240
[<8c01285a>] sys_clone+0x1a/0x40
[<8c0141e8>] syscall_call+0xc/0xe

post_handler: p->addr=0x8c021ca0
```

```
int init_module(void)

        int ret;
        kp.pre_handler = handler_pre;
        kp.post_handler = handler_post;
        kp.fault_handler = handler_fault;

                (kprobe_opcode_t*) kallsyms_lookup_name("do_fork");
        /* register the kprobe now */
        if (!kp.addr) {
                printk("Couldn't find %s to plant kprobe¥n", "do_fork");
                return -3;
        }
             = register_kprobe(&kp) < 0)) {
                printk("register_kprobe failed, returned %d¥n", ret);
                return -2;

              ("kprobe registered¥n");
              0;
}
```

```
00001dc0 <do_fork>:
   1dc0:      C3FF      mov.l     r8,@-r15
   1dc2:      53 68     mov       r5,r8
   1dc4:      96 2f     mov.l     r9,@-r15
              43 69     mov       r4,r9
              a6 2f     mov.l     r10,@-r15
              b6 2f     mov.l     r11,@-r15
   1dcc:      63 6b     mov       r6,r11
   1dce:      c6 2f     mov.l     r12,@-r15
   1dd0:      73 6c     mov       r7,r12
   1dd2:      d6 2f     mov.l     r13,@-r15
   1dd4:      00 ed     mov       #0,r13
```

```
int handler_pre(struct kprobe *p, struct pt_regs *regs)
{
        printk("pre_handler: p->addr=0x%p¥n",p->addr);
        dump_stack();
        return 0;
}
```

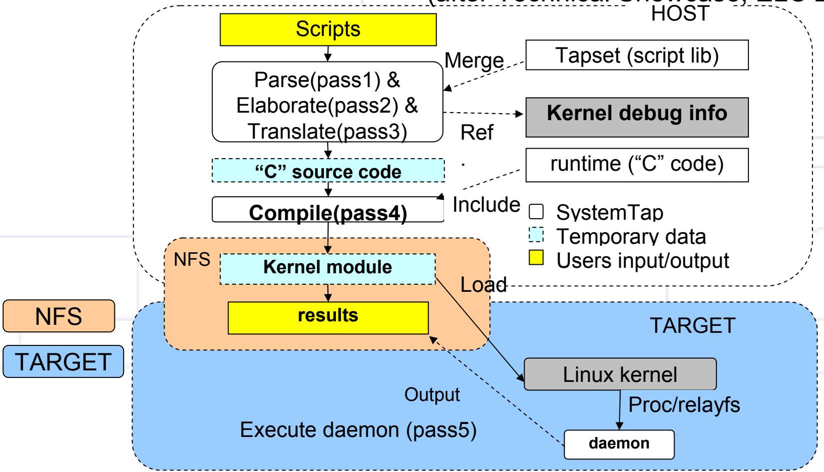
# Other tracing technologies

- SystemTap for SH … Hitachi, Lineo Solutions (Demo at ELC2007)
- What is SystemTap ?
  - software to simplify the gathering of information about the running Linux kernel.
- Configuration of SystemTap
  - Commands (stap) … Frontend of SystemTap, following:
    - Parse(pass1)
    - Elaborate(pass2)
    - Translate(pass3)
    - Compile(pass4)
  - Daemon (staprun)
    - Started from Stap, insmod the probed modules, combined to kernel and write results.
- Resources
  - Tapset … Library of Scripts
  - Runtime … C Library

# Other tracing technologies

## SystemTap (Idea for Embedded)

(after Technical Showcase, ELC 2007)



# Summary

- LKST Tutorial & Porting Updates are shown.
- Introduces Episodes acquired from the porting.
  - For Development of Cross Environments, mechanisms of endian exchange are required
  - Porting to Various Architectures are shown
  - Some Ideas for improving practicality of LKST are shown
    - Management Mechanism for Internal Files (binary log, lkst_etypes, mask)
    - Categorizing of the Patch Files
    - Static Tracing
  - Overhead of LKST by Measuring Benchmarking are shown
- Other tracing technologies Lineo Experienced are shown
  - Kprobes for SH, SystemTap for SH are shown

Thank You!