Technical Discussion: Linux Kernel Tracer

Tohru Nojiri
Hitachi

Overview of the Session

1. Presentation of LKST (Linux Kernel State Tracer)
2. Demonstrations
3. Updates on SH port of LKST by Iijima-san of Lineo
4. Tutorials of LKST and lkstlogtools by Iijima-san
5. Summary of LKST Technical Meeting in Tokyo
6. **Technical Discussion on "Linux Kernel Tracer"**
   - What kind of bugs do we struggle with everyday?
   - How do we see System Problems and debug them?
   - Is Linux Kernel Tracer is helpful for us?
   - Requirement for Features of Linux Kernel Tracer
   - Suggestion on Implementation of the New Features

1. LKST: Linux Kernel State Tracer

What is LKST

- LKST
  - Event Tracer Keeps Track of Kernel State Transition for Linux Kernel
    - Process Management, Interrupt, Exceptions, System Calls, Memory Management, Networking, IPC, Locks, Timer, Oops, etc.
  - Helps Us to do System Failure Analysis and Performance Analysis
  - One of the Results of Collaborative Work of IBM, Fujitsu, NEC and Hitachi
  - Currently Maintained by Hitachi
  - Originally Implemented on IA-32 PC Server
  - SH-4 Port, MIPS Port and ARM Port Available for Embedded Systems
  - Available at `http://sourceforge.net/projects/lkst`

Features

- Hooks in Kernel Source Code to Trap Kernel Event
  - Place Hooks to Call Out Kernel to LKST Module (Event Handler)
  - Place Hooks in Arbitrary Kernel Source Code Locations
  - Low Overhead Hook Mechanism by using Kernel Hooks
- Activate/Deactivate Every Hook without Kernel Rebuild
  - Select Kernel Events Dynamically during System Behavior Analysis
- Event Handler to Write Kernel State in Buffer (Event Buffer)
  - Event Buffer is Allocated in Kernel Address Space
- Various Type of Data Structure and Control for Event Buffer
  - Keep Just Important Information in Small Event Buffer
- Everything is Customizable On-the-Fly

LKST Structure

Event Set

Event type [hex]	Categoly	Mnemonic	Description of events		where to hook
01	Process management	PROCESS_CONTEXTSWITCH	Process context switching		schedule()
02		PROCESS_WAKEUP	WAKEUP		try_to_wake_up()
03		PROCESS_SIGSEND	sending signal		specific_send_sig_info()
04		PROCESS_LTHREADGEN	creating a kernel thread		kernel_thread()
05		PROCESS_INIT_WQH	initialize wait_queue_head		init_waitqueue_head()
06		PROCESS_ADD_WQ	add to waitqueue		__add_wait_queue(),__add_wait_queue
07		PROCESS_REM_WQ	remove from waitqueue		__remove_wait_queue()
10	Interrupts	INT_HARDWARE_ENTRY	hardware	entrance	do_IRQ()
12		INT_TASKLETHI_ENTRY	software	entrance	tasklet_hi_action()
14		INT_TASKLET_ENTRY		entrance	tasklet_action()
16		INT_BH_ENTRY		entrance	bh_action()
20	Exceptions	EXCEPTION_ENTRY	de	entrance	error_code
			int3		
			overflow		
			bounds		
			invalid_op		
			double_fault		
			coprocessor_segment_overrun		
			invalid_TSS		
			segment_not_present		
			stack_segment		
			alignment_check		
			coprocessor_error		
			simd_coprocessor_error		
			debug		
			general_protection		
			page_fault		
			machine_check		
			sprious_interrupt_bug		
			device_not_available		device_not_available
			nmi		nmi
21		EXCEPTION_EXIT	device_not_available	exit	device_not_available
			nmi		nmi
			exceptions other than above two		error_code
30		SYSCALL_ENTRY	entrance		beginning of system_call()

Hook Point

- Kernel Location Corresponding to Event (State Transition)
  - Place Hook in the Kernel Source Code to Trap each Event
  - Event Takes Place when Kernel Execution Reaches Hook Point
  - Call Out of Kernel to Event Handler to Generate LKST Message

Kernel Execution Thread

Hook Point

```
static int functionA()
{
 unsigned int flags;

If ( 1 ) {
   LKST_HOOK(EVENT_1,arg1,arg2,...);
}

spin_lock_irqsave(&lockA, flags);
...
```

Branch

Event Handler

```
void handler_B(arg1,arg2,...)
{

lkst_evhandlerprim_entry_log(..
.);
}
```

Event Handler

- Function Called with Event Trapped
  - Calling Event Handler with PID and 4 Additional Args
  - System Defined Event Handler
    - DEFAULT (ID=1)
    - Nothing (ID=255)
  - User Defined Event Hander (Extended Event Handler)
    - Implemented and Installed As Kernel Modules
    - Adding Extended Event Handler Like Device Driver

“context_switch”, 00, 0001300, 10584453412, 214325555, ”pointer to task struct(prev)”,

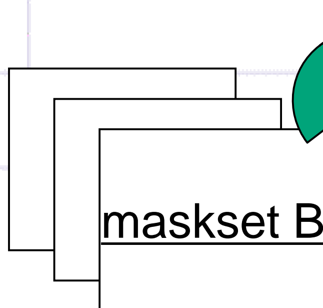
Event CPU PID TimeStamp(sec, nanosec)

0xda42800,0x00000000,”pointer to task struct(next)”, 0xda42400,0x00000000, ...

Arg1 Arg2

MaskSet

- Connecting Event With Event Handler
  - System Defined MaskSet
    - RDEFAULT: Primary Events Trapped Call Default Event Handler
    - RALL: All Events Trapped Call Default Event Handler
    - RNOTHING: No Event Trapped
  - User Defined MaskSet
    - LKST Utility Command



MaskSet

Event

LKST_HOOK(EVENT_1...);

maskset A
EVENT_1⇒handler_B
EVENT_2⇒handler_A
EVENT_3⇒handler_C
EVENT_4⇒Nothing

On-The-Fly

Event Handler

void handler_A(...)

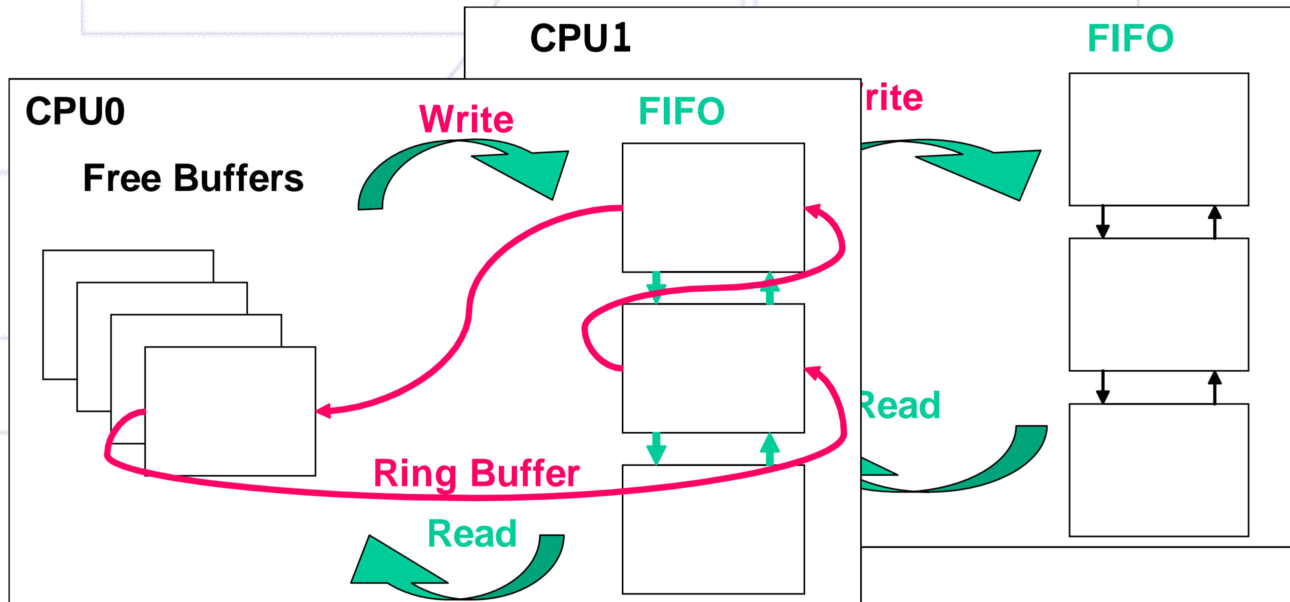
void handler_B(...)

void handler_C(...)

Link Event with Handler

Event Buffer

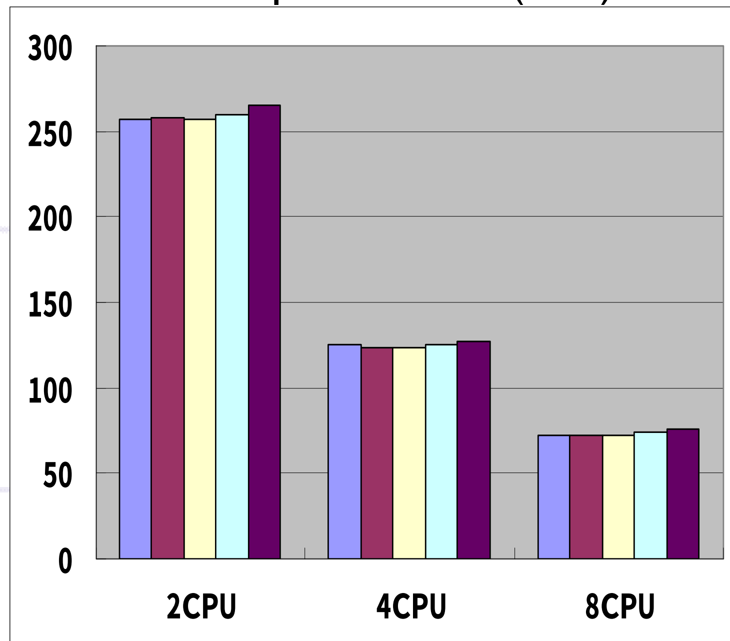
- Consists of Fixed Size of Mem Blocks Linked Together
  - Create and Adding a Block to Linked List On-the-Fly
  - Event Handler Writes Message to Event Buffer like Ring Buffer
  - LKST Utilities Reads data from Event Buffer like FIFO
- Event Buffer per CPU



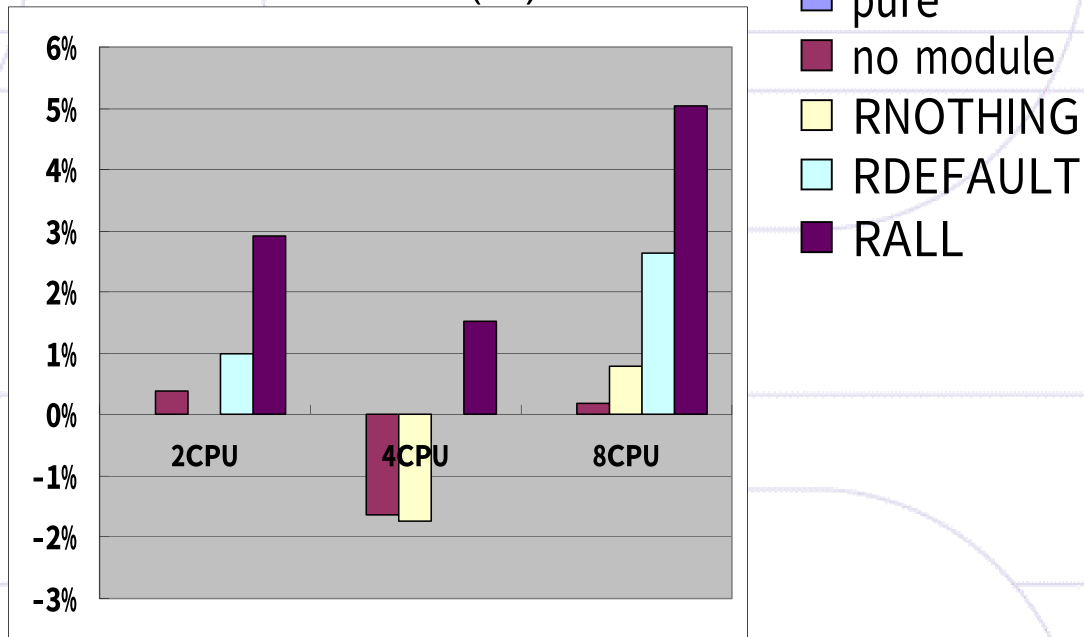
LKST Overhead (Kernel Build)

- Hardware Configuration
  - 8 CPU PC Server
    - Pentium III Xeon 700MHz (L2: 1MB) x 8
    - Memory: 4GB

Elapsed Time (sec)



Overhead (%)

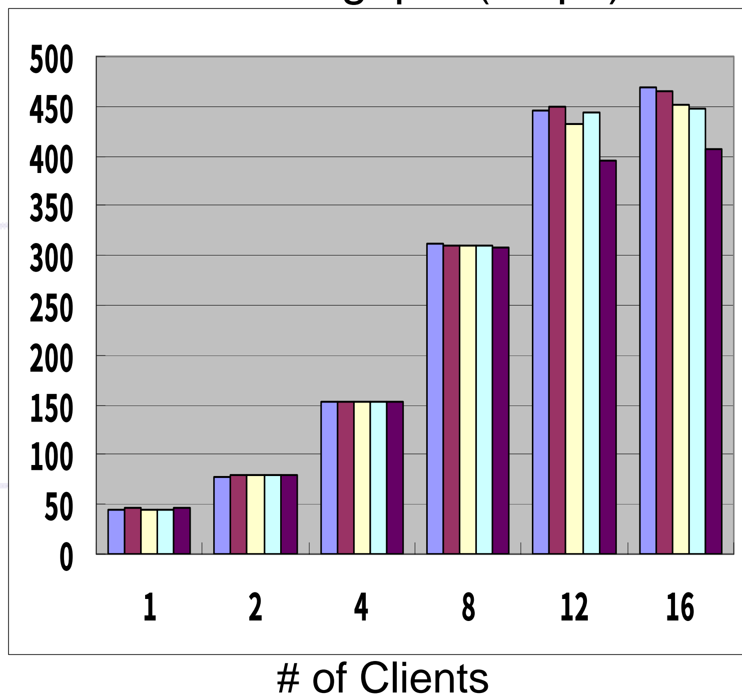


(測定環境：OSDLジャパン提供)

LKST Overhead (WebStone)

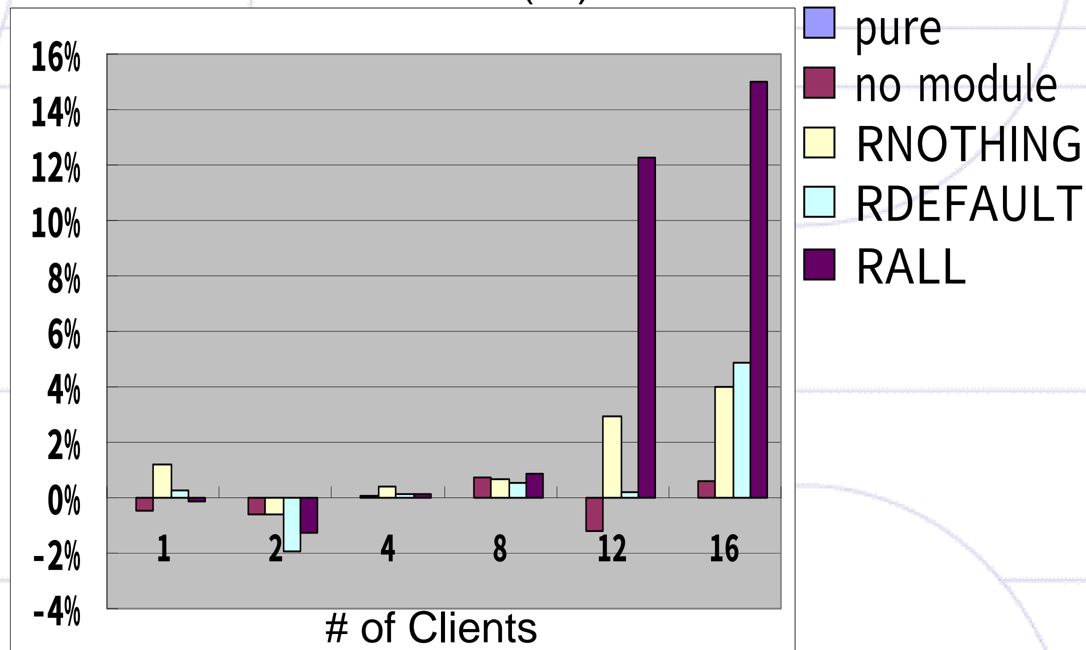
- Hardware Configuration
  - 8 CPU PC Server
  - 16 Client PCs (Pentium III 700MHz / 768MB RAM)
  - Gigabit Ethernet

Throughput (Mbps)



# of Clients

Overhead (%)

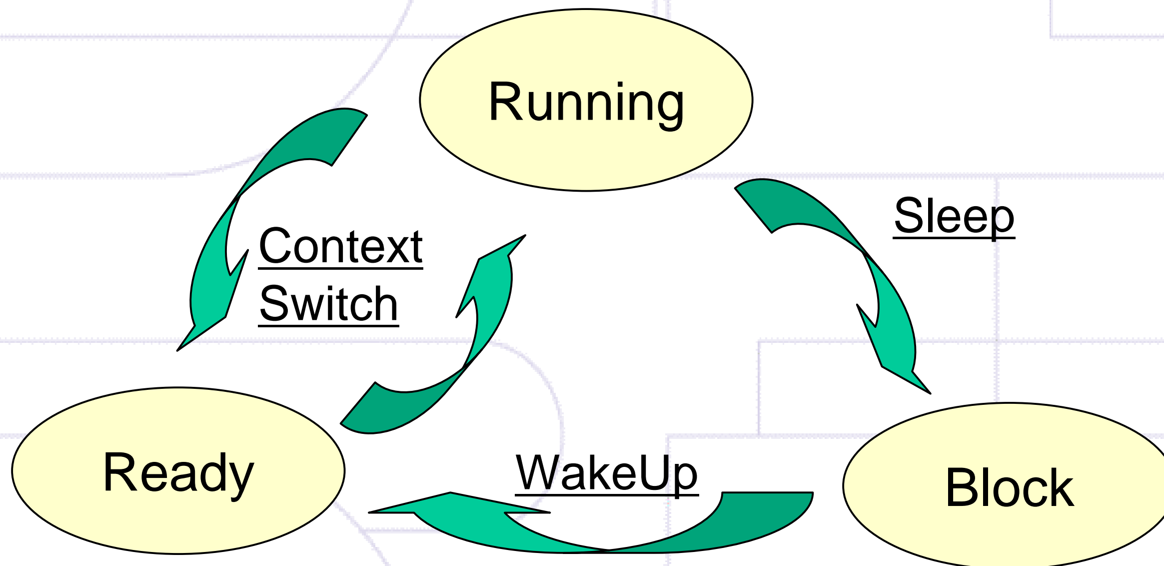


(測定環境：OSDLジャパン提供)

2. LKST Demonstrations

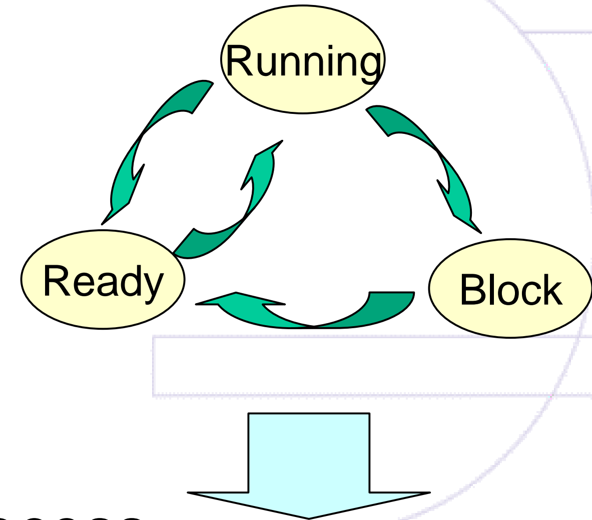
ProcessTrace: Outline

- Visualizing State Transition of a Process
  - State of Process: Running, Ready, Block
  - Picking up Events, “PROCESS_CONTEXTSWITH” and “PROCESS_WAKEUP” to See State Transition



ProcessTrace：Implementation

- Pick up Process State Transition
  - Create MaskSet to Pick Up the Events
    - “PROCESS_CONTEXTSWITH”
    - “PROCESS_WAKEUP”
  - Read Trace Data from Event Buffer

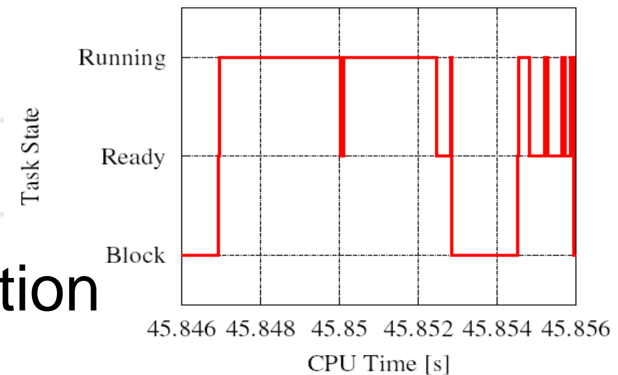


↓

- Trace State Transition of a Particular Process
  - Convert of Address of “task_struct” to PID
  - Trace State Transition of the Process

↓

- Plot Trace Data of Process State Transition



ProcessTrace: Creation of MaskSet

- Event and Args of Event Handler
  - PROCESS_CONTEXTSWITCH (Event ID=1)
    - Arg1： Address of task_struct of the Previous Process
    - Arg2： Address of task_struct of the Target Process
    - Arg3： State of the Previous Process after the Context Switch
  - PROCESS_WAKEUP (Event ID=2)
    - Arg1： Address of task_struct of the Target Process

RNOTHING

```
$ lkstm read -m 0 | lkstm write -m 3     Create Null MaskSet
$ lkstm config -m 3  1  1                Connect EventID=1 with default Handler
$ lkstm config -m 3  2  1                Connect EventID=2 with default Handler
$ lkstm set -m 3                         Switch to the new MaskSet
```

Event ID   Handler ID

ProcessTrace: TraceData

- lkstbuf Command
  - Read the TraceData from Event Buffer

```
$ lkstbuf read -f trace.log
```

LKST Format (trace.log)

  - Print in CSV Format

```
$ lkstbuf print -r -C -S -V -f trace.log > trace.csv
```

CSV (-C) Sec Resolution (-S)

trace.csv

"context_switch", 00, 0001300, 10584453412, 214325555, "pointer to task struct(prev)",

Event CPU PID TimeStamp(sec, nanosec)

0xda42800,0x00000000,"pointer to task struct(next)", 0xda42400,0x00000000, ...

Arg1 Arg2

ProcessTrace: PID and Task_Struct

- Conversion Table of address of “Task_Struct” to PID
  - From Trace Data of “PROCESS_CONTEXTSWITCH”

trace.csv

“context_switch”,00,0001300, 10584453412,214325555,”pointer to task struct(prev)”,
0xda42800,0x00000000,”pointer to task struct(next)”, 0xda42400,0x00000000, ...

Arg1 PID — PID and Task_Struct of the Process

```
$ grep context trace.csv | cut -d, -f3,7 | sort | uniq > trace.db
```

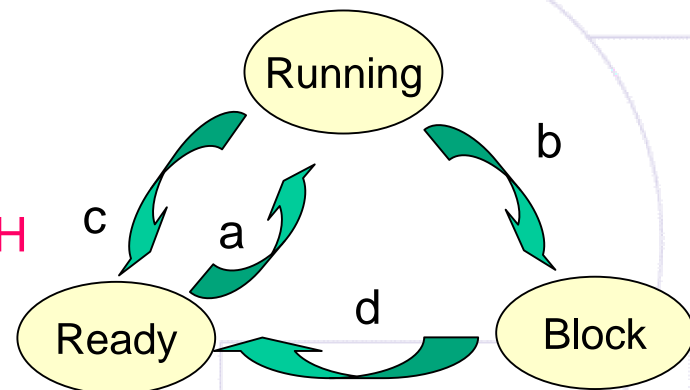
trace.db

```
00000000,0xc0422000
00000001,0xdc85c000
00000002,0xdd864000
00000007,0xdf46e000
•
•
•
```

ProcessTrace: State Transition

- State Transition
  - running
    a) Target Process of Context Switch
      - Arg2 of PROCESS_CONTEXTSWITCH
  - block
    b) Previous Process of Context Switch
      - Arg1 of PROCESS_CONTEXTSWITCH
      - Arg3 ***is NOT*** “TASK_RUNNING”
  - ready
    c) Previous Process of Context Switch
      - Arg1 of PROCESS_CONTEXTSWITCH
      - Arg3 ***is*** “TASK_RUNNING”

    d) Process Waked up
      - Arg1 of PROCESS_WAKEUP



Memory Allocation Viewer: Outline

- Visualizing Physical Page Allocation for a Particular Process

1. Get Info. of Current Allocated Physical Pages for the Process
   - Get Physical Page# of the Pages from vm_area_struct
   - Implemented as Kernel Module
2. Pick up Events of Physical Page Allocation
   - No Hook Point to Pick up this Event Provided by LKST
   - Placed Additional Hook Point to Pick it up
3. Write Physical Page # of Allocated One into Event Buffer
4. Get and Visualize the Physical Page # Every 1 sec.

5. Summary of LKST Technical Meeting

Summary of the Meeting

- Hitachi Hosted Technical Meeting on Kernel Tracer
- 10 People Joined the Meeting from 8 Companies
- We Discussed:
  - Q & A about LKST
  - Debug Style
  - Requirements for Kernel Tracer from the view of Embedded System Development

Summary of Q&A

Q1: How many HookPoints does LKST provides?

A1: Around 100 default HookPoints. We can place additional one in any location of Linux Kernel Source Code.

Q2: How many Events does Event Buffer Preserve?

A2: Event Buffer preserves about 1000 events with 64KB. It works like Ring Buffer and it is estimated that it keeps track of last few milliseconds at every moment with default Mask Set.

Q3: Tracer tends to Generate Huge Amount of Trace Data for the First Stage of Debug.

A3: Profiler is very useful to narrow the Scope of Analysis of Bug and Optimize Tracer to Pickup just useful Trace Data for its Debug

Summary of Requirements

- Reliable Extraction of Trace Data from Event Buffer at Any Serious System Problem like System Crash
  - JTAG, Crash Dump, ….
- Collaboration between Debugger/ICE and Kernel Tracer
  - Hook Point Insertion with Debugger/ICE User Interface
- Analyzing Relationship of Cause and Effect with Kernel Events and Application Events
  - Same Time Stamp between Kernel logs and Application logs?
  - PID info that LKST log data provides?
- Finding Buggy Modules Like Accessing Kernel Resources with Illegal Code Sequence Relating to Locking and Releasing Shared Resources
- Visualization Tools for Trace Data Analysis

6. Technical Discussion

Technical Discussion

Let's Talk and Discuss about the Following Things:

- What kind of bugs do we struggle with everyday?
- How do we see System Problems and debug them?
- Is Linux Kernel Tracer is helpful for us?
- Requirements for Features of Linux Kernel Tracer from the view of Embedded System Development
- Suggestion of Implementation of the Requirements