What & Why?

- Low-level system debugging facility
  - Operates in extreme conditions
  - Live Systems vs. Development
  - Automated Kernel Debugger (SMP capable)
  - Dynamically Customisable Trace/Logger
  - Universal (User/Kernel/Interrupt mode code)
  - Fine grained Storage Profiling
  - Low System Overhead

- Proven technology from OS/2
- **Enabler for other RAS offerings**
  - Dynamic Complex Assertion Checking
    - Trigger analysis from first-point-of-failure
  - Ad-hoc Tracing
What is a Probepoint?

Automated Breakpoint

- Trapping Breakpoint (INT3, SVC 255):
  - Unlimited number in general
  - Usually generalises across platforms
  - Module-level Specification
  - Can miss events under MP

- Hardware Watchpoint (DRegs):
  - No missed events under MP
  - Limited number in general
  - Doesn't generalise across platforms
  - Virtual/Physical Storage Specification
## Probepoint Specification

<table>
<thead>
<tr>
<th>Locality</th>
<th>User Specification</th>
<th>Characteristics</th>
<th>Internal Specification</th>
<th>Typical Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Per-process</td>
<td>virtual address/module-offset</td>
<td>Privatises shared pages via COW</td>
<td></td>
<td>GDB, ptrace</td>
</tr>
<tr>
<td>Per-module</td>
<td>module-offset</td>
<td>Global, inserted using aliased virtual address.</td>
<td>inode-offset for non-resident and user modules. Virtual address for resident kernel modules.</td>
<td>DProbes (User probes)</td>
</tr>
<tr>
<td>Global</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Virtual Storage</td>
<td>virtual address</td>
<td>Limited to Kernel space or one process</td>
<td></td>
<td>Debug H/W kernel debuggers watchpoints (Kprobes)</td>
</tr>
<tr>
<td>Physical Storage</td>
<td>physical address</td>
<td>Limited to resident modules</td>
<td></td>
<td>Debug H/W kernel debuggers watchpoints</td>
</tr>
</tbody>
</table>
Basic Probe Components

- Trap 3
- Trap 1
- DPEH
- Probe Manager
- Probe Object
- Probing Routines (kernel module)
- Callback (pre-handler)
- Post Handler

(kernel space) (user space)
Probe Mechanism

- Breakpoint
  - Instruction Replacement (INT3)
- Single-stepping or Emulation
- Interrupts disabled
- Per-CPU probe context
- Log Buffer Committal
- Recursion
- SMP Serialization & Locking
- SMP probe misses window*
- Zombie breakpoints
Patch Organization

- **2.5**
  - Kernel Probes (*kprobes*) (< 400 LOC)
  - Debug Register Management (*dr_alloc*)
  - Watchpoint Probes (*kwatchpoints*)
  - User Space Probes (*uprobes*)
  - Dprobes driver (rest of dprobes function)

- **2.4**
  - Full Dprobes kernel patch using Kernel Hooks

**User Space**
- Dprobes command (RPN based probe language input)
- Dprobes C Compiler (*dpcc*) (C-like interface to RPN)
Kernel Probes

```c
struct kprobe {
    struct list_head list;

    /* location of the probepoint */
    kprobe_opcode_t * addr;
    /* Called before addr is executed. */
    kprobe_pre_handler_t pre_handler;

    /* Called after addr is executed, unless... */
    kprobe_post_handler_t post_handler;

    /* ... called if executing addr causes a fault (eg. page fault) */
    kprobe_fault_handler_t fault_handler;

    /* Saved opcode (which has been replaced with breakpoint) */
    kprobe_opcode_t opcode;
};
```
Kprobes Interfaces

```c
static void test_probe_handler(struct kprobe *p, struct pt_regs *regs)
{
    printk("p1 hit\n");
    count++;
    return;
}

struct kprobe p1 = {
    .addr = (u8 *)0xc0116120, /* do_fork */
    .pre_handler = test_probe_handler,
};

->  int register_kprobe(struct kprobe *);
->  int unregister_kprobe(struct kprobe *);
```
Watchpoint Probes

- Fired on specific types of memory accesses
  - Execute, Write, Read or Write, IO
  - Specified by virtual address, range (non-context specific)
- Exploits H/W debug registers
  - 4 on Intel x86, 1-4 byte range
  - Debug Reg. Alloc patch - co-ordinate w/ other Debug Facilities
- Enables fine-grained storage profiling with LTT
  - e.g. Monitoring specific kernel data structures
- Kwatches patch
- Pagepoints extension
User Probes

- System wide global
  - Common to all Instances of an object module
  - No Copy-On-Write!
  - Shared Library, Executable program
- Identified by `<inode, offset>` rather than addr
- On-demand insertion (no forced loading/pinning of pages)
  - low memory overhead (can have large no. of probes)
  - e.g. by intercepting readpage aops for the inode
- Uprobes patch
  - minimal support (physical placement; search on probe hit)
  - extends kprobes struct with `<inode, offset>`
Full DProbes Components

- readpage
- trap 3
- trap 1
- DPEH Callback
- Probe Object
- Probe Mgr
- RPN Command Interpreter
- Logging Daemons
- External Agents

(kernel space)   (user space)

dprobes cmd

IBM LTC 23/01/03
External Facilities

- Logging Daemons
  - Syslog (klogd) - default
  - COM1 and COM2
  - Universal Dynamic Trace - LTT (Opersys)
  - POSIX Event Logging

- External Agents
  - KDB
  - SGI Kernel Crash Dump
  - Core Dump
RPN Interpreter

- Access to CPU (low-level) resources
- "Easy" to generate from a HLL (dpcc) - compare with Java
RPN Command Categories

- Arithmetical/Logical
- Program Flow
  - Conditional
  - Subroutine calls
- External Triggers
- Local (i.e. per-probe) and Global Variables
- Log Buffer (per-processor)
- Exception Handling
- System Resources:
  - Registers, Memory, IO
The DProbe High-Level Language Compiler

- Allows a C-like language to be used for probe definitions.
- Compiler (dpcc) generates RPN code.
- Allow variables from probed program to be referenced.
- Supports a set of simulated c-RTL routines.
- Needs original program to be compiled with debug opts.
- Language is fully described in dpcc.groff man pages.
- See numerous examples packaged with dpcc.
Example dpcc Probe Program

```c
#include <linux/vmlinux>

#define MODNAME("/usr/src/linux/vmlinux")
#define PROBEPOINT_LOCATION("fork.c:589")
#define MODTYPE(kernel)
#define PROBEPOINT_HANDLER("test")

/* Demonstrates logging local variable (2.4.18 kernel)
  do_fork:589 => blank line after *p = *current;
  Logs the task_struct pointed to by p.
  View output in system log e.g. /var/log/messages.
*/

void test()
{
    log_probe_expr("*p");
}
```
DProbes Command

- Dprobes C Compiler (dpcc)
- RPN Command File
- Program Symbolic Information
- gcc pre-processor
- DProbes Cmd
- dpcc Probe Program
- Predefined Probe Definition File
- API
Employment of DProbes

- Sysfs based driver using kprobes - dynamic printk insertion (Rusty Lynch)
  
  ```
  $ echo "add <address> <message>" > /sys/noisy/ctl
  ```

- Field problems (OS/2 dtrace):
  - Obscure cause-symptom relation & live system:
    - Page Manager Bugs, Probes in Context Switch!
    - Parcel Bomb Problems (async msg induced, OS/2 PM)
    - Device Driver/Device Interface Bugs (interrupt handler probes)

- During development
  - Large-scale (internal) instrumentation
  - Fault Injection
  - Ad-hoc Profiling
  - Debugging races, timing sensitive problems
Work-In-Progress/Future Work

- Dprobes driver for 2.5 using kprobes
- IA-64 port
- More instruction emulation?
- Single-step out-of-place
- Pagepoints
- Sampler probes?
- Init-time probes?

Architectures currently supported: ia32, s390, s390x, ppc, ppc64
16. Questions?

Mailing List: dprobes@oss.software.ibm.com
Web Page:
System RAS Projects page: http://systemras.sourceforge.net

Core Team:
Richard Moore (RAS Architect)
S. Vamsikrishna (Project Maintainer)
Bharata B. Rao
Gary Hade (ia64)
Michael Grundy (s390 & ppc)
Subodh Soni
Suparna Bhattacharya
Thomas Zanussi (dpcc)

Our Thanks to:
Rusty Russell (genesis of "kprobes")
Andi Kleen (SuSE)
Andrea Arcangeli (SuSE)
Karim Yaghmour (OperSys)
Maneesh Soni (IBM)
9. Performance

Quantitative measurements
Pentium 90Mhz (11ns cycle time)
order of 8-16µs

Qualitative results
Tracepoints on entry to pagefault routines - negligible
Tracepoints on kernel heap routines - negligible
Tracepoints on all kernel APIs - negligible
Tracepoints on all kernel routines (4000) - somewhat noticeable!
10. Porting Considerations

Linux on other H/W:
- Integer size
- RPN Instruction set - register set - endian issues
- Probepoint implementation - INT3 equivalent
- Single-step mechanism - atomicity with breakpoint
- Serialisation - Cache & MP
- Watchpoint implementation

To Other OS's:
- Module management
- Page management
- Symbolic support - ELF
- Memory aliasing
- Fault interception
11. Process Switching Example (1)

Out-going Process

In-coming Process

copy from system

zero difference

hi lower bound

lo upper bound

copy from system

zero difference

hi lower bound

lo upper bound
12. Process Switching Example (2)

Out-going Process

- lo upper bound
- hi lower bound
- copy from system
- zero difference
- copy from system

In-coming Process

- destroyed data
- hi lower bound
- lo upper bound

IBM LTC 23/01/03
RPN Program Storage

DPO

Hdr

Per-DPO Local Variable Array

Probe Definitions

Global Variable Array

Per-Processor Log Buffer

P0

P1

P2
Example RPN Probe Program

name = bzImage
modtype = kernel
major = 1
jmpmax = 32
logmax = 100
vars = 1

offset = kill_proc
opcode = 0x55
minor = 1
ignore = 0
maxhits = 1000
inc lv,0
push d,16
push r, esp
log mrf
exit

Further info => man ./dprobes.lang.groff
13. Command Invocations

INSERT:
dprobes -i <rpn_file_name>
dprobes -i <rpn_file_name> -s <symbol_file_name>

QUERY:
dprobes -q [-x] [-a]

REMOVE:
dprobes -r <rpn_file_name>
dprobes -r -a

GET LOCAL/GLOBAL VARS:
dprobes -g -a

Further info => man ./dprobes.groff