A Universal Dynamic Trace for Linux and Other Operating Systems

USENIX Annual Technical Conference 2001

Boston, USA

Richard J. Moore

richardj_moore@uk.ibm.com

IBM Linux Technology Centre

30th June 2001

0. Overview

1. Universal Dynamic Trace?
2. DProbes as a Trace Enabler
3. What is a Probepoint?
4. The Probepoint Specification
5. RPN Interpreter
6. RPN Command Categories
7. RPN Access to Logging Daemons
8. Manipulating the Log Buffer
9. Performance
10. Porting Considerations
11. Process Switching Example (1)
12. Process Switching Example (2)
13. What's Next?
14. Questions

Appendix

a. Watchpoint Probes
b. DProbes Components
c. DProbes Command
d. Example RPN Probe Program
e. Successful Employment of DProbes

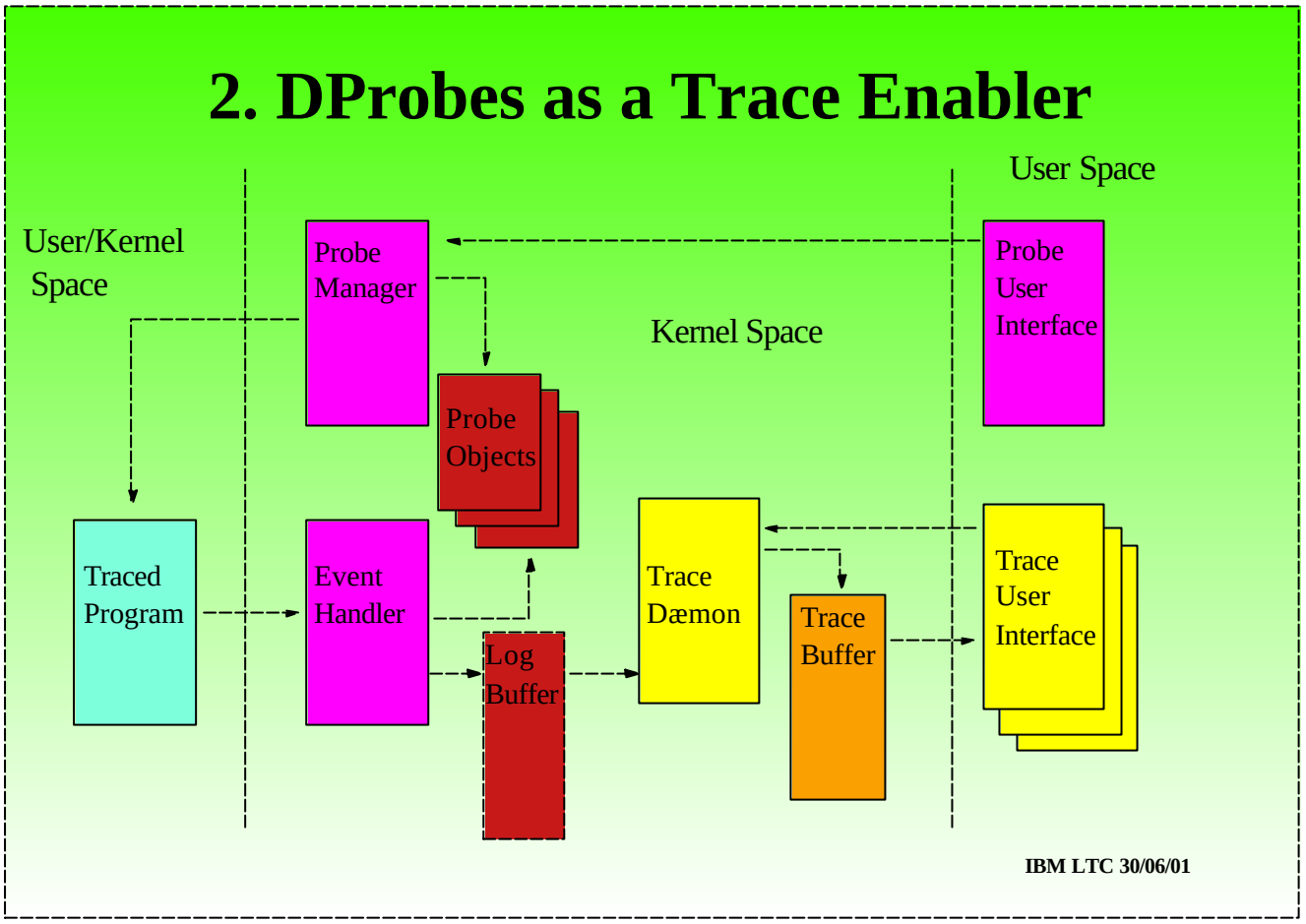
1. Universal Dynamic Trace?

- Universal:
  - Kernel or User Space
  - Interrupt time, Task Time
  - MP compliant
  - Code or Data
- Dynamic:
  - Tracepoints "inserted" at run-time
  - Doesn't require source modifications
  - Tracepoint actions are customisable at run-time

→ Debugging engine

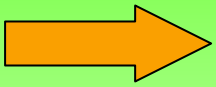
IBM LTC 30/06/01

- Universal dynamic trace - what is it?
-
  - It's universal because it can operate in kernel or user space
  - It will work under the most extreme conditions at interrupt time or task time or even between contexts
  - It operates in an MP environment (there are some caveats discussed later)
  - It can be used to trace code or data usage
-
- The Dynamic aspect refers to:
  - The ability to *insert* tracepoints at run-time
  - Without the need to modify or prepare traced code in advance
  - Actions taken when the tracepoint fires are customisable at run-time.
-
- This implies the need for a debugging engine to be interfacing with a standard system tracing mechanism.



- Dynamic Probes provides the debugging engine.
- It operates in kernel space as a self-contained debugger effectively encapsulated in an interrupt handler with minimal dependence on system services.
-
- It uses a breakpoint mechanism to insert its probes.
- There is a Log buffer for staging traced data.
-
- The only requirement of a system tracing mechanism is that its operates on a dæmon principle that provides an callable interface what can be driven from an interrupt handler.
-
- The Linux Trace Toolkit from Opersys meets these requirements.

3. What is a Probepoint?



Automated Breakpoint

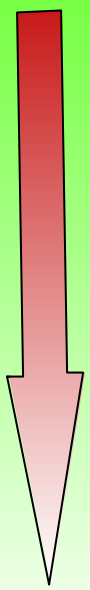
- Trapping Breakpoint (INT3):
  - Unlimited number in general
  - Usually generalises across platforms
  - Module-level Specification
  - Can miss events under MP
- Hardware Watchpoint (DRegs):
  - No missed events under MP
  - Limited number in general
  - Doesn't generalise across platforms
  - Virtual/Physical Storage Specification

IBM LTC 30/06/01

- A the heart of DProbes is the Probepoint, which is essentially an automated breakpoint.
-
- There are in general two way to implement breakpoints. Each has it merits:
-
- The trapping breakpoint is implemented using an instruction replacement technique (INT3 under IA32).
- There's no limit to the number of concurrently installed breakpoints.
- This mechanism generalises across other architectures
- We can canonically define probes with reference to a module rather than storage location - discussed next foil.
- However there is a theoretical exposure under MP: because we need unlimited access to system resources, the DPEH runs in privileged mode, which means we can't generally emulate the original instruction after the breakpoint fires. We have to single-step it in situ. This means temporarily removing the breakpoint and thus exposing us to a miss if the same probe was executed on another processor. In practice this is not a problem, since we tend to be concerned with races in two different pieces of code for the same data, rather than the same piece of code. However it's really a problem the stop-cpus switch does allow execution on other processors to be suspended during single-step
-
- The alternative mechanism is to use the inbuilt debugging H/W sometimes called the "watchpoint" mechanism.
- The MP miss problem doesn't occur
- However, the number of concurrent watchpoint can be severely limited (4 on IA32).
- Also it's very tied to a particular architecture and so doesn't generalise easily.
- Finally watchpoints are specified by storage location without reference to context or module which as the potential to cause unnecessary hits which would have to be filtered.
-
-
- We choose primarily the trapping breakpoint mechanism for probepoints in code. These are defined relative to a module.
-

4. Probepoint Specification

Local



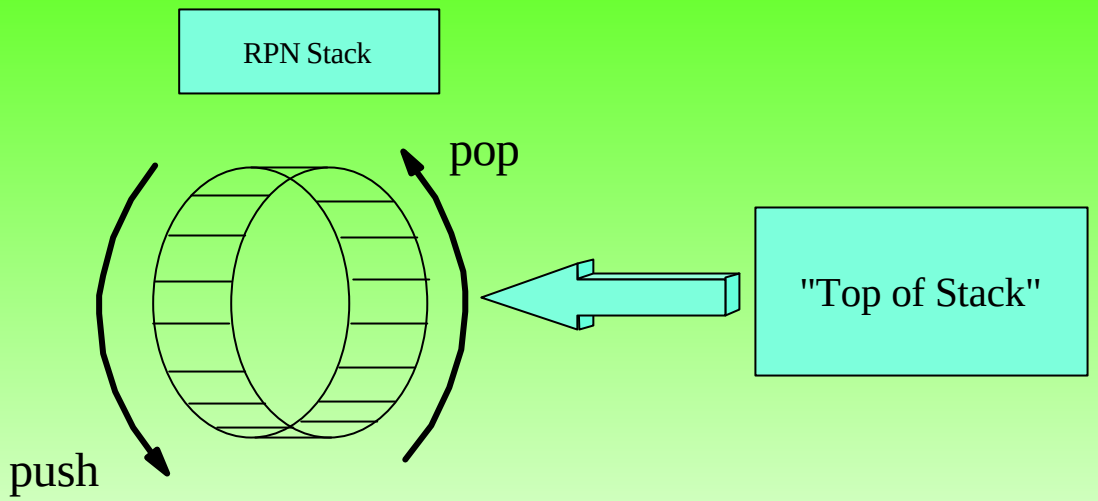
Locality	User Specification	Characteristics	Internal Specification	Typical Usage
Per-process	virtual address/ module-offset	Privatises shared pages via COW		GDB, ptrace
Per-module	module-offset	Global, inserted using aliased virtual address.	inode-offset for non-resident and user modules. Virtual address for resident kernel modules.	DProbes
Virtual Storage	virtual address	Limited to Kernel space or one process		Debug H/W kernel debuggers watchpoints
Physical Storage	physical address	Limited to resident modules		Debug H/W kernel debuggers watchpoints

Global

IBM LTC 30/06/01

- We mentioned on the previous foil that the probepoint is defined relative to a module. This slide compares the merits and employment of various breakpoint tracking strategies.
- GDB defines breakpoints per process using ptrace. The placement of a breakpoint causes privatisation of a page (COW), with a resulting impact to the swap device.
- Debuggers using watchpoints - typically KDB place breakpoints in kernel space and have to filter unnecessary interrupts if the want a per-process view. It is difficult to relate such breakpoints to a user module since the virtual storage mapping may be different per process.
- On some architectures watchpoints may be defined by physical storage location (e.g. S/390) but again this is difficult to relate to a user's module because the physical mapping may change with paging activity.
- DProbes uses a module-relative approach. BPs are inserted using the physical address to avoid COW proliferation of privatised pages. We track the BP using inode-offset. This gives us a global context to the probe without the impact to the swap device.

5. RPN Interpreter



- Access to CPU (low-level) resources
- "Easy" to generate from a HLL - c.f. Java

IBM LTC 30/06/01

- The heart of the DPEH is the RPN command interpreter.
- Two questions:
- what is RPN
- why use RPN
-
- languages - such as List, use a stack on which to place operands then execute the operation, which the operands to be popped off the stack and the result pushed onto the stack. It's "Reverse" because syntactically one codes operands before operation. It's "Polish" probably because it was invented by a Pole.
-
- RPN interpreters are very easy to implement.
- They give easy access to low-level resources while generalising across architectures
- They permit high-level languages to be defined which generate RPN code - compare with Java and the JVM which is an RPN-based virtual machine.
- The use of an HLL provides an architecture-independent means of writing probe-programs.

6. RPN Command Categories

- Arithmetical/Logical
- Program Flow
  - ➡ Conditional
  - ➡ Subroutine calls
- External Triggers
- Local and Global Variables
- Log Buffer
- Exception Handling
- System Resources:
  - ➡ Registers, Memory, IO

IBM LTC 30/06/01

- The RPN command language comprises the following categories of command:
-
- Basic arithmetical and logical instructions
- Program flow including conditional logic and subroutine calls.
- Mechanisms for invoking external agents and daemon - External Triggers.
- Local and Global storage for use by the RPN program#
- Exception handling to recover from unexpected environmental conditions such as memory not accessible.
- Access to system resources. CPU regs, Memory, Kernel data items, IO ports etc..

7. RPN Access to Logging Daemons

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

- Externally triggered facilities come in two types:
- Logging Daemons, to which are passed a log buffer and control is returned to the DPEH. Examples include: syslog, com ports, LTT, Event Log (future enhancement).
- External Agents transfer control to the external facility without expectation of return. Examples of these are KDB,lkcd, code dump.
- These type types are handled slightly differently by the DPEH. Because Logging Daemons tend to want to record only one event per attempted execution of a code location - we avoid log replications by delaying logging until the original instruction executes without faulting - think about recoverable page faults and the effect it would have on a trace if an event were recorded per trial execution. This behaviour by the way can be overridden using the logonfault facility.
- With external agents we restore the original instruction and give control to the agent without single-stepping.

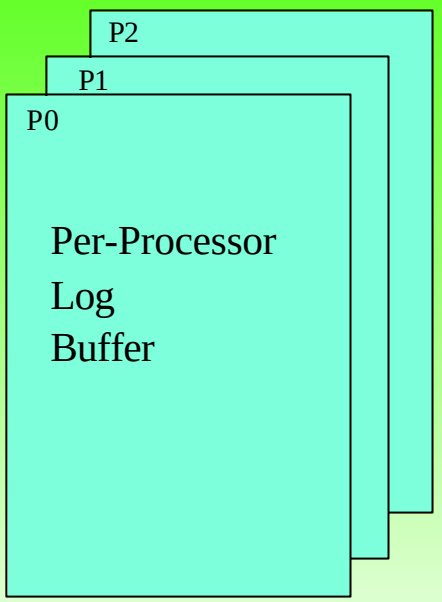
8. Manipulating the Log Buffer

log b,n
log w,n
log d,n

write data from the RPN stack

log mrf/mrs
log arf/ars

write data from memory with 3-byte prefix:



code	length

0: success
1: zero terminated ascii range
-1: flat address fault
-2: segmented address fault

IBM LTC 30/06/01

- There are three types of working storage available to a probe-handler:
- The local variable array that allows data to be share among probe handlers for a given module.
- The global variable array allows data to be shared among all probe handlers, whatever their module.
- The log buffer, which is defined per-processor to avoid unnecessary serialisation. This is used to stage the logged data which is eventually passed to the Logging Daemon.
- The set of **log** RPN commands provide the means to populate the Log Buffer. The are two varieties:
  - Those that log data directly from the RPN stack
  - Those that log data from memory.
- The latter category needs special processing:
  - A 3-byte prefix containing a code and length is placed at the head of the data in the Log Buffer. This enable variable length binary data to be logged. But it also allows for the case where an error needs to be logged when data is not accessible.

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!

- We used a simple loop to assess the performance overhead of a probe. Measurements were made on a 90Mz Pentium (and repeated later on a 200MHz chip - the results were consistent).
-
- Quantitatively the overhead of the Probe Event handler is 8µs if we emulate the original instruction and 16µs if we single-step it.
-
- Whether or not this is significant depends on the mean time to re-execute the same tracepoint.
-
- Practical examples show that when used for tracing system APIs or internal interfaces to major kernel components the effect is unnoticeable to the user.
- Only when ca. 4000 tracepoints were places on the entrypoints of all internal OS/2 kernel routines did we notice a significant effect.

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

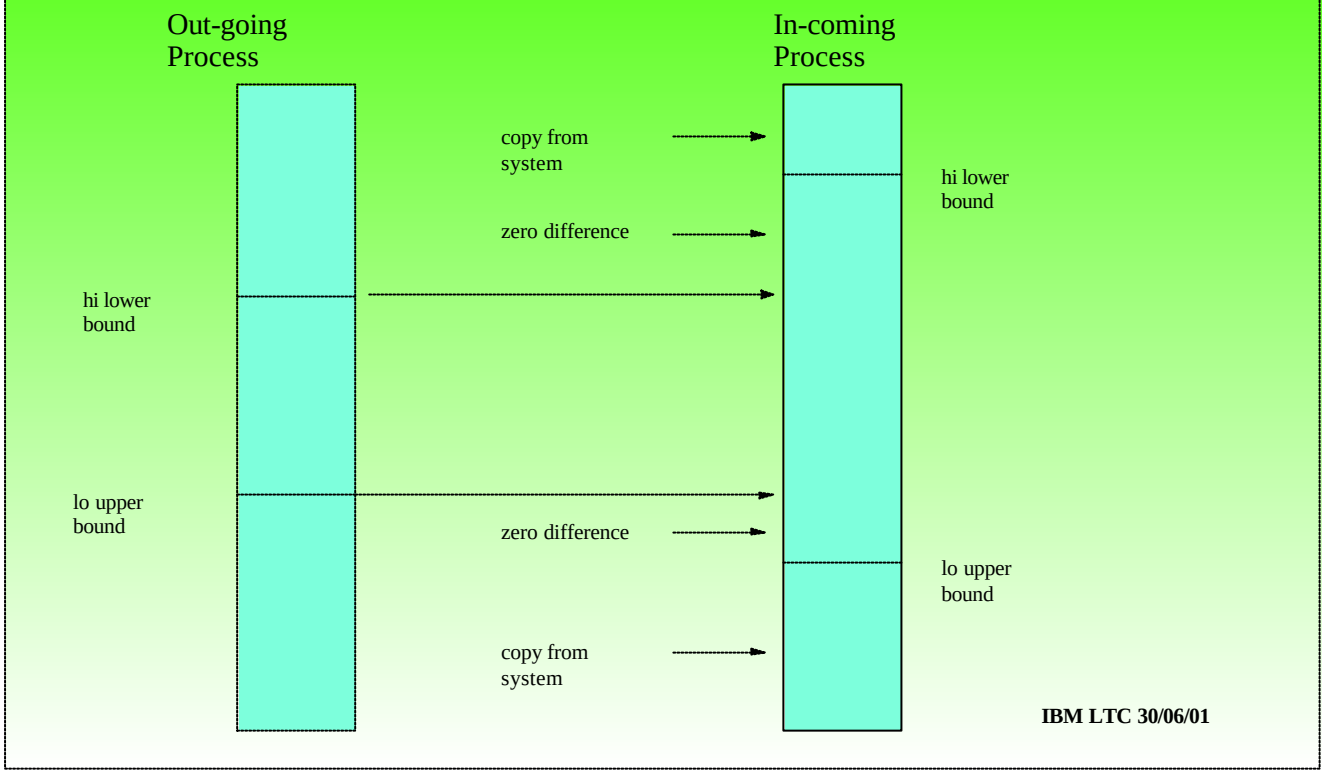
To Other OS's:

- Module management
- Page management
- Symbolic support - ELF
- Memory aliasing
- Fault interception

IBM LTC 30/06/01

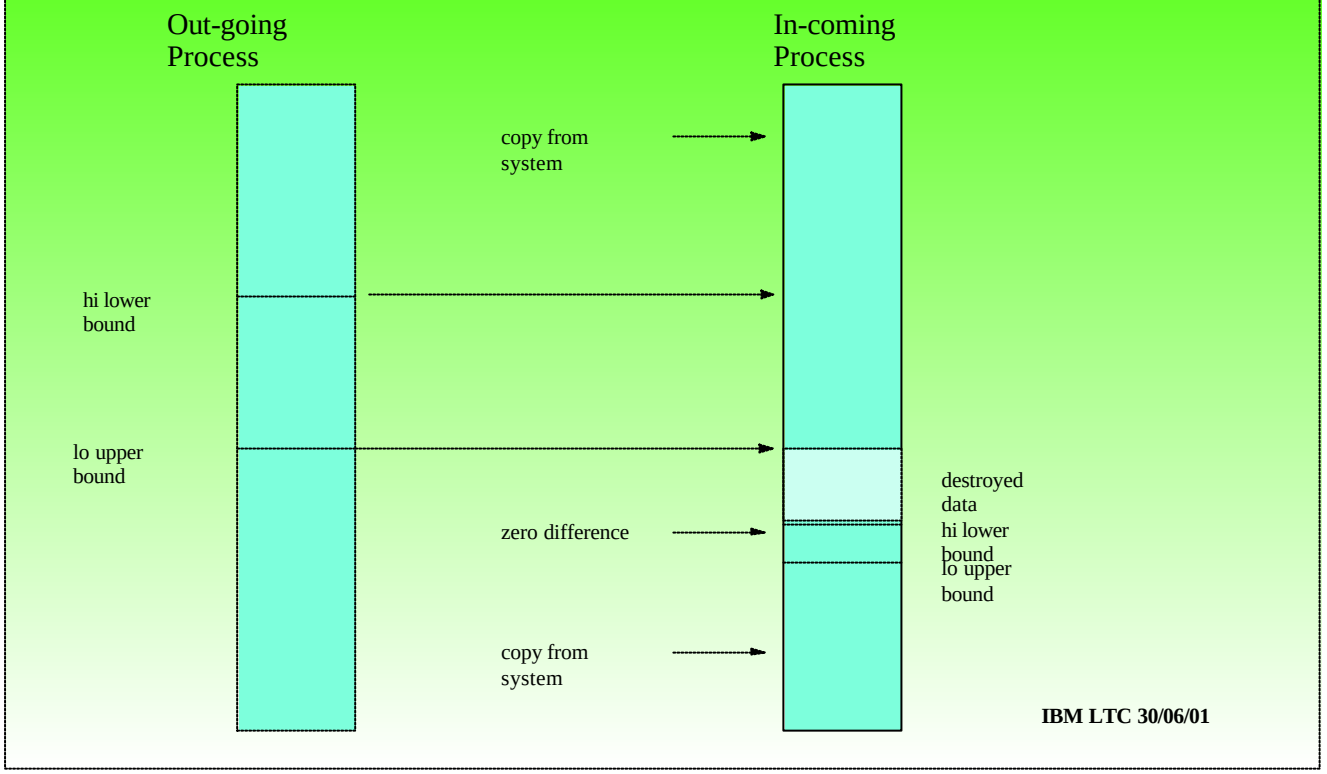
- When Porting to H/W platforms the considerations are principally:
  - RPN stack width, Local and Global arrays are dependent on the integer size. A #define ensures they a re modified consistently.
  - The RPN instruction set is assemble-like and relates to the underlying architecture. In particular the register mnemonics will need to be translates. There will also be some adjustments for endian issues. The HLL will hide these dependencies from the user.
  - The probepoint mechansim will need to use an appropriate trapping instruction.
  - Single-step will be specific to the architecture. DProbes imposes the requirement that breakpoint interrupt through to single-step completion is execute without interruption or recursion. Hence under IA32 Linux, both the trap 1 and trap 3 gates were change to be interrupt gates and the original instruction was stepped with interrupts disabled.
  - Watchpoint implementation is highly architecturally dependent. It can be omitted as it is an extension to the basic mechanism.
-
- In porting to other OS's a few more considerations apply:
  - How are modules managed - DProbes needs a convenient handle by which to define a probepoint. inode-offset works for Linux and may UNIX variants.
  - DProbes needs to intercept code page-in so that probepoint can be inserted. Hooking the readpage routine in Linux achieves this.
  - The DProbes command interprets ELF symbolic information. A DOS, Windows or OS/2 type of environment would require the equivalent information to be processed from either a module, linker map file or symbol file.
  - DProbes needs to alias memory to make the code modifications. Most OS's provide a kernel interface to achieve this. If they don't then direct manipulation a the page tables might be required.
  - The DProbes RPN Interpreter needs to intercept faults relating to access violations so that they can be silently handled.

11. Process Switching Example (1)



- This example is taken from a live system problem on OS2.
-
- The symptom was that intermittently a page fault was generated for memory that had been locked.
-
- It was traced using DProbes to dump that page table entries before and after a context switch.
-
- Because OS2 employed a high address shared user memory pool - shared at the physical memory level it avoided updating PTEs that were in common to both the out-going and in-coming tasks.
-
- The page manager tracks the lower bound of the high (shared) memory and the upper bound of the lower (private) memory. The algorithm when as follows:
-
- Copy the high memory PTEs for the in-coming task
- Copy the low memory PTEs for the in-coming task
- If the out-going high memory lower bound was lower than the in-coming high memory lower bound then zero the difference
- If the out-going low memory upper bound was higher then the in-coming low-memory upper bound then zero the difference.

12. Process Switching Example (2)



- This algorithm is flawed the trace showed:
-
- When the in-coming low bound for high-memory is lower than the out-going low memory upper bound the different between these is zeroed erroneously.

13. What's Next?

- HLL compiler
- RPN extensions for HLL
- Port to zSeries (S/390)
- Sampler Probe type for Profiling
- Port to 64-bit xSeries (IA64)
- Custom Trace Formatting for Dynamic Trace events
- Instrumentation of Kernel and other key modules
- Data/Request Trace for selected drivers

IBM LTC 30/06/01

- We are working on the following new features:
-
- A HLL interface - initially C-like - may be later Java-like
- A number of RPN extensions to support the HLL: e.g. exception handling, working storage enhancements. More RPN commands for manipulating the RPN stack.
- We are working with the Poughkeepsie tools team on a port to Linux on zSeries (S/390)
- We are implementing a new probe type for profiling purposes - the sampler probe along with per-processor instance data variables.
- We are providing a custom trace formatting mechanism for Dynamic Trace = (DProbes + LTT)
- We employ dynamic trace to instrument drivers and the kernel.

14. Questions?

End of Presentation

Mailing List: *dprobes@oss.software.ibm.com*
Web Page: *http://oss.software.ibm.com/developerworks/opensource/linux/projects/dprobes*

Core Team:

Richard Moore (RAS Architect)
S. Vamsikrishna
Subodh Soni
Bharata B. Rao
Suparna Bhattacharya

Contributions From:

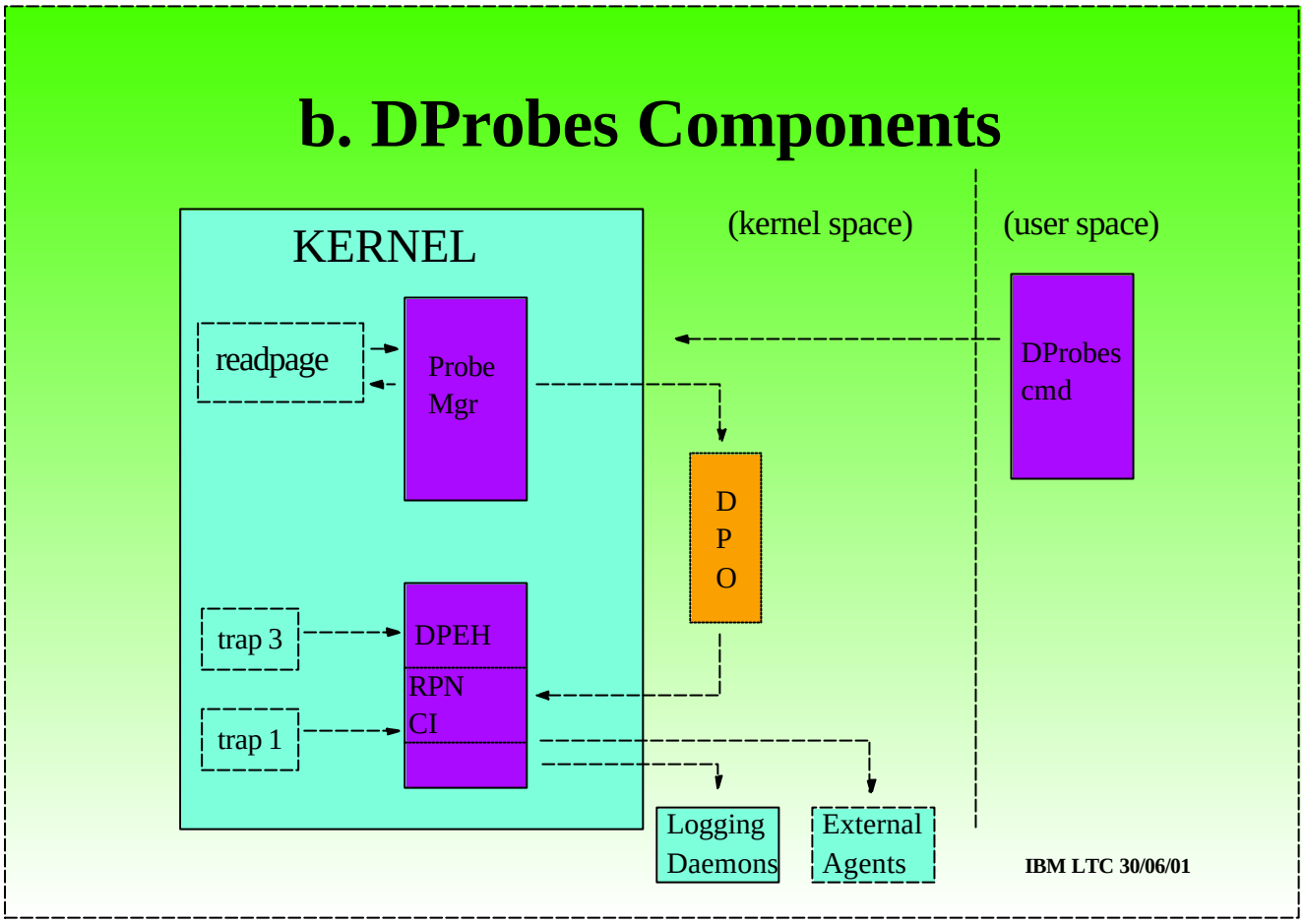
Maneesh Soni

Andi Kleen (SuSE)
Andrea Arcangeli (SuSE)
Karim Yaghmour (OperSys)

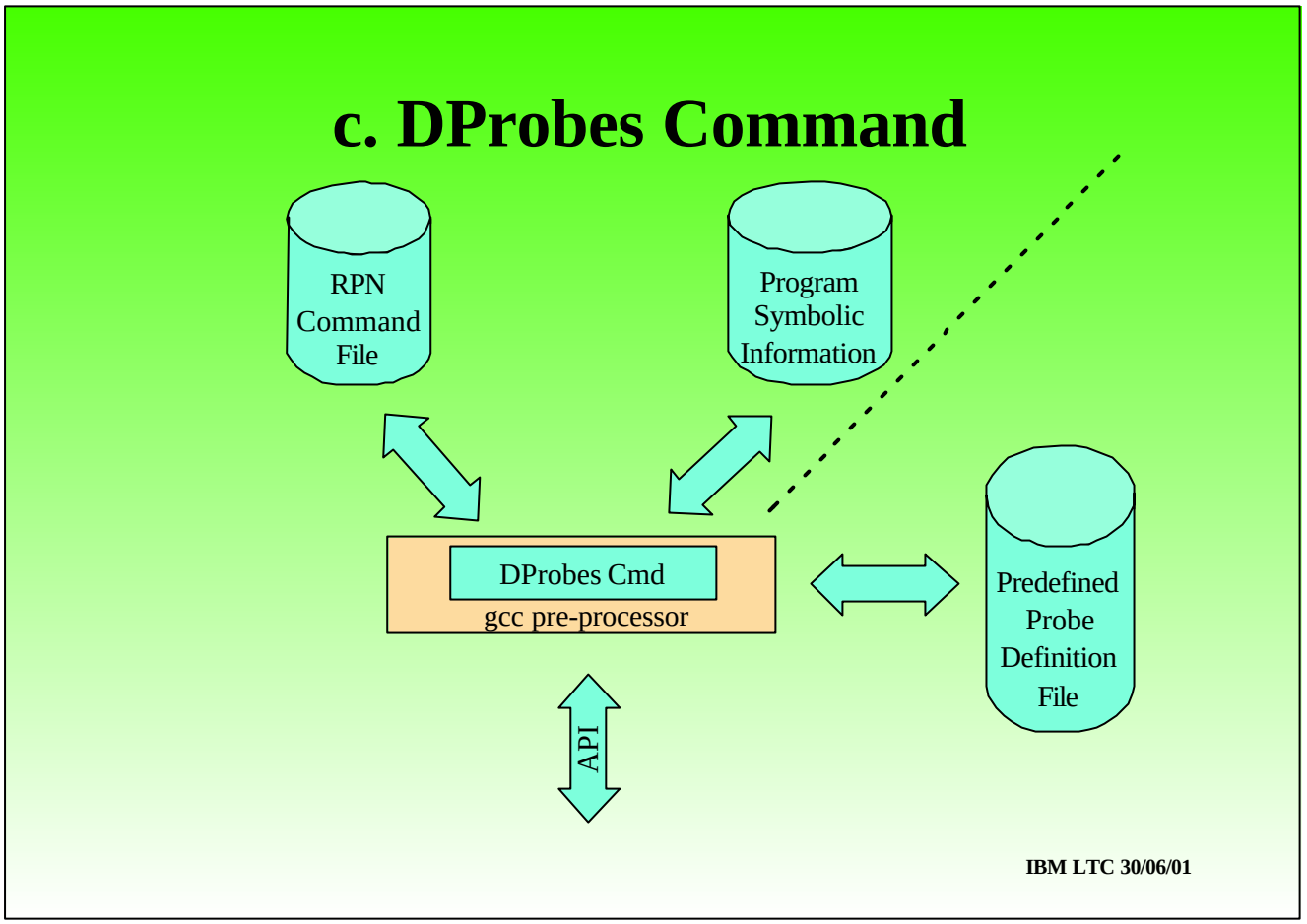
a. Watchpoint Probes

- Fired on specific types of memory accesses
  - Execute, Write, Read or Write, IO
  - Specified by virtual address, range
  - Not limited to any process context
- Exploits H/W debug registers (4 on Intel x86)
  - Debug Register Allocation patch for co-ordinating with other Debug Facilities
- Enables fine-grained storage profiling with LTT
  - e.g. Monitoring specific kernel data structures

- We do however also exploit watchpoints in the watchpoint probes (as opposed to the breakpoint probes).
- These probes are virtual-storage based and global without a module or process context.
- They permit memory accesses to be probed whether read/write/execute or IO
-
- IA32 limits us to a maximum of 4 WPs. (However we have devise a mechanism for simulating a generalised extension to this - details cannot be revealed at this stage but essentially it hooked into the paging mechanism).
- Linux does not cater well for multiple users of debugging registers - so we have also provided a DR allocation patch.



- In essence this is what DProbes looks like:
-
- Most of it's function is in kernel space. There are two key components:
- 1) The probe manager that looks after the installation and track of whether probes are places (probepoint locations).
- There's a data object that describes a probe: the Dynamic Probe Object.
- The probe mgr essentially hooks the readpage routine to install probepoints whenever a page of code is brought into memory. (it is more complex than this - but avoid going into details here - wait for questions).
-
- 2) The other components is the Dynamic Probe Event Handler, which responds to a breakpoint interrupt.
- It hooks the single-step and breakpoint interrupt vectors under IA32 architecture.
- Part of the DPO is a probe handler program, written in a Reverse Polish Notation language. This contains the instructions, per probepoint, that need to be interpreted when a probepoint "fires". Part of the DPEH is the RPN Command Interpreter (RPN CI) which performs this function.
- The RPN CI may call external facilities such as logging daemons - for tracing purposes
- The RPN CI may transfer control to external agents (no return guaranteed) such as lkcd, kdb, core dump, etc..
-
- Finally, there is a command line interface to control the activation and deactivation of probes.



- The DProbes command takes as input either:
-
- The RPN file and optionally Symbolic program information
-
- Or, a predefined probed definition file. This is essentially a pre-compiled version of the RPN file, in a form ready to pass to the DProbes API. It permits probes to be defined using symbolic information present only when the probed program is compiled with debugging options, then to have the debugging information stripped,
-
- The DProbes command invokes the gcc pre-processor - this allows standard C-like pre-processor constructs in the RPN file and for symbols to be substituted from the command line.

d. Example PRN Probe Program

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

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


- This is an example RPN probe handler definition.
-
- There's a header section that defines the module, number of local variables and other controls.
-
- Then there are a number of probe definition that follow. Each has a header followed by the RPN program.
-
- The probe header gives the location, in this case kill_proc and we also specify the original opcode for sanity reasons particularly where an address is given instead of a symbol.
- The program increments local variable 0, and logs the parameters pointed to by the ESP register on entry to kill_proc.

e. Successful Employment of DProbes

- Development of DProbes
- Kernel Development (SuSE)
- Page Manager Bugs
- Parcel Bomb Problems
- Device Driver/Device Interface Bugs
- Prototype System Modification
- Profiling
- Large-scale (internal) instrumentation

IBM LTC 30/06/01

- This is where we and others have successfully used DProbes:
-
- We used it to debug itself
- SuSE use it to debug the linux kernel - its easier than recompiling in printk statements.
- RJM used in to solve a number of OS page manager problems that were impossible to re-create at will and only occurred in a customer's production environment. Probes were placed in the context switching code path!!!
- The parcel bomb problem is where work request are enqueued asynchronously to some facility. If a problem occurs when a request is processed, the cause - the enquing process is long since gone. DProbes can be used to intercept and monitor enquing actions and catch the culprit. An example of this occurred with OS/2 Presentation Manager.
- We can trace requests to device drivers
- It has been used to prototype system modifications - by intercepting an API and dynamically changing one or more parameters.
- The local and global variables facilitate profiling. However we are enhancing DProbes to specifically for profiling.
- As a tracing mechanism DProbes can be used extensively with minimal system impact - e.g. 400 - 1000 concurrent tracepoints.