ADVANCED C/C++
DEBUGGING (CDT PROJECT)
TRACING (LINUX TOOLS PROJECT)

DOMINIQUE DOT TOUPIN AT
ERICSSON DOT COM
FULL OPEN SOURCE SOLUTION

› ECLIPSE CDT
› ECLIPSE LINUX TOOLS
› GDB
› LTTNG
› LINUX
REVERSE DEBUGGING

› Allows to undo register and memory changes so as to move the execution backwards
› Uses recording and playback

Buttons to control reverse execution
Start recording and display execution buttons
- More execution in parallel in many processes
- Debugging related processes at the same time
- Dynamically attach and detach from processes
- Follow child process created with a fork, exec,
- Global Breakpoint, many processes can execute the same code, auto attached to the process only when the breakpoint is hit, also useful for short lived-process
- Core awareness, threads are running on which cores

Cores are shown for both threads and processes
2.4 View where users can see the cores and what is running on each core, in many cases users will use core affinity to put thread on specific core for execution locality. The threads don't move much between cores.
2.5 Define what are the process/threads for an application, attach to all of them;
2.6 Select one core and stop it or put a breakpoint on all the threads running on it; or a global breakpoint for the first process or thread hitting the breakpoint;
2.7 Continue/step for all thread running on a core;
2.8 Understand the interaction between process/threads of different cores, e.g. signal breakpoint;
2.9 A few use cases/features are almost the same as the one in PTP (process set, operations on set, etc.)


3) Systems with multiple cores (On Chip Debugging)
3.1 Hardware bring-up debugging
   User attaches a debugger to a multi-core system for purpose of hardware configuration, such as configuring registers, IO devices, etc. (no OS, no symbol data, no executable image)
3.2 No-OS debugging
   User attaches to a multi-core system and downloads simple executable image to each core and debugs them.
3.3 Debugging with OS-awareness
   User attaches to a multi-core system, downloads an OS image to each, starts debugging the entire OS on each board, or specific tasks/processes running on specific cores.
3.4 Synchronized run control operations
   Some multi-core debuggers allow for performing synchronized operations on multiple cores. Allowing the debugger or the hardware to operate on multiple cores minimizes the skid, or the time lag, in when the operations are carried out on the different cores. There needs to be dedicated UI and APIs to exercise these debugger features.
SPECIAL BREAKPOINTS

› Conditional Breakpoint
  - Stop only if the condition is true.
  - C assert condition, break when assertion is false

› Data Breakpoint or Watchpoint
  - Stop whenever the value of an expression change
  - Don't have to predict where this may happen
  - Can be a complex expression or just a single variable

› Program event breakpoint
  - Stop when a special event occurs
  - Throwing/catching C++ exception,
  - unhandled exception
  - call to exec, fork, syscal
• Display and editing of complex objects like Lists, Maps, Vectors

• Some programs have a deep interaction with OS resources DSF-GDB can show: process groups, file descriptors, internet-domain sockets, shared memory segments, semaphore, message queues, loaded kernel modules, etc.
NON-STOP

› Debugging a process by stopping its execution might cause the program to change its behavior drastically, or perhaps fail, even when the code itself is correct.
  - Troubleshooting in the lab
  - Chasing a race condition
  - Debugging problems happening only under heavy load
  - Investigating user interface issues

› Non-Stop allows to stop and examine one or more thread in the debugger while other threads continue to execute freely
Dynamic Tracepoints

- Tracepoint collects user-specified info and continues execution without stopping any thread, essential for live sites
- Dynamic i.e. inserted with a jump (in process) or a trap
- Data collection can be conditional to a user specified expression

Tracepoint actions:
- collect state trace data e.g. timestamp, and program data e.g. variables, register
- evaluate expressions, e.g. modify trace variables
- step (similar to breakpoint step) and collect data in each step

- A trace experiment can be stopped after the n'th hit
- Static tracepoint (LTTng UST) can be stored in the debug tracepoint buffer
- Debug tracepoint are good when no static tracepoint are available and for small quantity of data
Intuitive display using debugger views
TRACING

› Need to understand what is going on in a system without causing disturbance? → Tracing is for you

› Compared to logging, tracing typically records lower-level events that occur much more frequently

› Tracers are therefore optimized to handle a lot of data while having a small impact on the system

› Static Tracepoint
  − created by designer before compilation
  − represent wisdom of developers who are most familiar with the code
  − The rest of the world can use static tracepoint to extract a great deal of useful information without having to know the code
**LTtng Low-Overhead Tracing Architecture**

- **Host**
  - Eclipse Tracing and Monitoring Framework (EPL)
  - libltttraceapi (LGPL)
  - Shell command or scripting (GPLv2)

- **Target**
  - C/C++ Application
    - Tracepoint
    - ust/libust (LGPL)
    - Shared-Memory per-CPU Buffers
  - Java Application
    - Tracepoint
    - ust/libust (LGPL)
    - Shared-Memory per-CPU Buffers
  - Erlang Application
    - Tracepoint
    - ust/libust (LGPL)
    - Shared-Memory per-CPU Buffers

- **LTtng Daemon (LGPL)**
  - Concurrent trace sessions
  - Zero copy
  - Streaming or regular mode for network and local file
  - Flight recorder with save-on-demand
  - Self-describing binary format highly optimized for huge traces

- **Tracepoint Characteristics**
  - Low overhead, no trap, no system call
  - Signal, thread and NMI Safe
  - Wait-free read-copy update
  - Cycle-level time-stamp
  - Dynamic activation
  - Re-entrant kernel tracing
  - Non-blocking atomic operations
  - BSD license headers

- **Debugfs**
  - Local File System
  - LTtng/Ftrace

- **Trace-control and data-retrieval socket using TCF protocol**

- **Ltt-control/liblttctl (GPLv2)**
LTNG PERSPECTIVE
### LTTNG HISTOGRAM, STATISTICS

![Histogram and Statistics](image)

#### Statistics

<table>
<thead>
<tr>
<th>Level</th>
<th>Number of Events</th>
<th>CPU Time</th>
<th>Cumulative CPU Time</th>
<th>Elapsed Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trace1-15316</td>
<td>15316</td>
<td>0.060617118</td>
<td>0.87635906</td>
<td>0.762141301</td>
</tr>
<tr>
<td><strong>CPUers</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Event Types</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Modes</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IRQ</td>
<td>99</td>
<td>0.003638321</td>
<td>0.008165658</td>
<td>0.004082829</td>
</tr>
<tr>
<td>MODE_UNKNOWN</td>
<td>12432</td>
<td>0</td>
<td>8.497766916</td>
<td>0.762141301</td>
</tr>
<tr>
<td>SOFTIRQ</td>
<td>89</td>
<td>0.001467848</td>
<td>0.00309494</td>
<td>0.00154747</td>
</tr>
<tr>
<td>SYSCALL</td>
<td>1890</td>
<td>0.030586202</td>
<td>1.595780335</td>
<td>0.748304729</td>
</tr>
<tr>
<td>TRAP</td>
<td>319</td>
<td>0.006543252</td>
<td>0.031454274</td>
<td>0.008206273</td>
</tr>
<tr>
<td>USER_MODE</td>
<td>487</td>
<td>0.018381495</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

© Ericsson | LinuxCon Tracing Mini-Summit 2010
LT fancy font

CONTROL FLOW, RESOURCES

Image of a computer screen displaying a control flow diagram with processes and resources listed. The diagram shows various processes and their resource usage, including CPU time and memory usage. The bottom part of the image shows a resource usage chart, with details on CPU utilization over time. The chart includes timestamps and a breakdown of CPU activity by process and interrupt.
UPCOMING FEATURES

› General
  - Tracing tool control
  - Trace streaming
  - Heterogeneous traces
  - GDB Tracepoints
  - Source lookup
  - Performance tuning

› Other trace format
  - Linux User Space Tracing
  - Text format
  - De-facto standard format Multi-core association, Embedded Linux Forum, Samsung, Ericsson, Mentor Graphic, WindRiver, IBM, Freescale, TI, Nokia-Siemens Network, National Instruments, etc.

www.multicore-association.org/workgroup/tiwg.php

Common Trace Format Requirement:
http://lwn.net/Articles/408824/
Common Trace Format Implementation:
http://lwn.net/Articles/408825/

› Analyses
  - Time correction (traces synchronization)
    › Multi-core, multi-level, multi-node
  - Timing dependencies (between processes)
  - Latency Analysis
  - Pattern matching (security e.g. intrusion detection)
ADDITIONAL ONLINE RESOURCES

› Eclipse CDT DSF-GBD lead: marc DOT khouzam AT ericsson DOT com
cdt-dev@eclipse.org

› CDT Multi-core debugging http://wiki.eclipse.org/CDT/designs/MultiCoreDebug,

› http://gcc.gnu.org/wiki/summit2010

› Advanced Tracing Features using GDB and LTTng, Real-time debugging using GDB
Tracepoints, GDB Tracepoints: From Prototype to Production


Using Eclipse for Reverse, Multi-Process and Non-Stop Debugging with GDB p.65,
GDB Tracepoints, Redux p.105, Hybrid multi-architecture debugging with GDB p.137

Debugging in GDB p.117

› Eclipse LTTng plug-in lead: francois DOT chouinard AT ericsson DOT com
linuxtools-dev@eclipse.org

› http://www.eclipse.org/linuxtools/projectPages/Lttng

› http://www.lttng.org, http://lttng.org/content/success-stories

© Ericsson | Eclipse Summit 2010
ADVANCED C/C++
DEBUGGING (CDT PROJECT)
TRACING (LINUX TOOLS PROJECT)

DOMINIQUE DOT TOUPIN AT
ERICSSON DOT COM