The great troubleshooting encounter:

CDT meets Trace Compass

EclipseCon, March 2015

Marc Khouzam
Marc-André Laperle
ABOUT US

- Marc Khouzam
  - Software Developer at Ericsson since 1998
  - CDT project co-lead, focusing on Debugging
  - Working with CDT since 2009

- Marc-André Laperle
  - Software Developer at Ericsson since 2013
  - Committer for Trace Compass, CDT and Linux Tools
  - Contributor to other projects (Platform UI, SWT, EGit, Mylyn, PDE)
AGENDA

• A bit of background: Debug and Tracing

• CDT Debug and Trace Compass integration
  – An integration in 4 parts

• Conclusion
A LITTLE BACKGROUND: ADVANCED DEBUGGING
NON-STOP DEBUGGING

› Program continues execution while suspending some threads

› Reduced intrusiveness
Sometimes traces are necessary

Printf without recompiling or redeploying!
Access to system information while debugging

- Message Queues
- All Processes
- Loaded Kernel Modules
- Semaphores
- Sockets
- All Threads
- Process Groups
- Shared Memory Segments
- File Descriptors
MORE BACKGROUND: TRACING WITH TRACE COMPASS
TRACE COMPASS
COMMON FEATURES

› Data-driven state system and views
  - XML description of state changes to convert trace events to states
  - XML description of view representation of the computed state system
  - Can be created without changing source code or recompiling

› For example: 50 lines of XML created the view below
CONTROL FLOW VIEW

› Displays processes state changes (color-coded) over time
RESOURCES VIEW

- Displays **system resource states** (color-coded) over time
## CPU USAGE VIEW

### TID | Process       | %     | Time             
---|---------------|-------|------------------
4983| master_player | 0.508% | 507549 ns        
5079| java          | 0.172% | 172449 ns        
4980| challenger    | 0.469% | 469351 ns        
4986| master_player | 0.508% | 507759 ns        
4985| master_player | 66.658% | 66657523 ns       
5074| sh            | 0.041% | 41494 ns         
5077| java          | 0.138% | 137739 ns        
5076| java          | 0.013% | 13099 ns         
4847| java          | 0.020% | 20163 ns         
4988| kworker/3:1   | 0.595% | 595313 ns        
3837| gtk-window-deco | 0.074% | 73961 ns      
5045| java          | 0.029% | 28705 ns         

### CPU usage

![Graph showing CPU usage over time]
AGENDA

- A bit of background: Debug and Tracing

- CDT Debug and Trace Compass integration
  1. Enhanced Post-mortem troubleshooting
  2. Debugging with Trace snapshots
  3. Tracing with the (Multicore) Visualizer
  4. GDB Traces with Trace Compass

- Conclusion
ENHANCED POST-MORTEM TROUBLESHOOTING
POST-MORTEM DEBUG

- Use GDB to examine core file
- Variables, Registers, Memory
POST-MORTEM TRACE

› Standard visualization of traces taken upon a crash
Joint Debug/Tracing visualization for most flexibility
SYSTEM SETUP

1. Enable Tracing e.g., LTTng, UST, etc

2. Register crash handler with Linux kernel (man core)

3. Crash Handler collects/stores traces as well as core file
1) Use Post-Mortem launch

2) Specify location of Traces
CORE + TRACES
AGENDA

- A bit of background: Debug and Tracing

- CDT Debug and Trace Compass integration
  1. Enhanced Post-mortem troubleshooting
  2. **Debugging with Trace snapshots**
  3. Tracing with the (Multicore) Visualizer
  4. GDB Traces with Trace Compass

- Conclusion
DEBUGGING WITH TRACE SNAPSHOTS
Acquire snapshot and open on suspended debugger
Advantages:
- Very low overhead
- Minimal disk usage

Disadvantage:
- Limited data available (as big as buffer allows)
THE PROTOTYPE

1) Create a tracing session

2) Select session in Debug configuration

3) Suspend (or hit a breakpoint)
FUTURE IMPROVEMENTS

Configure session from Debug configuration
  • Choose tracer
  • Choose trace points
  • Tracer specific options
  • Persisted
FUTURE IMPROVEMENTS

› Callstacks of the last few seconds

Current stack frames (GDB)

Previous events with function entry and exit (LTTng snapshot)
Result (example)

Callstack can be visualized moments before suspend!
AGENDA

• A bit of background: Debug and Tracing

• CDT Debug and Trace Compass integration
  1. Enhanced Post-mortem troubleshooting
  2. Debugging with Trace snapshots
  3. Tracing with the (Multicore) Visualizer
  4. GDB Traces with Trace Compass

• Conclusion
Show all threads except sleeping
  • All of them could run

Coloured by kernel state

CPU Usage

We can have a better grasp of level of overload

Which processes are affected by the overload?
› Colouring by process

› Sorting as improvement
TRACE COMPASS AND TRACE VISUALIZER
Another example

Notice partial CPU usage even with overload

Could it be the Kernel using CPU?

Could indicate even stronger overload
VISUALIZER WITH XEON PHI

› Coloured by kernel state (RUNNING & SYSCALL)
VISUALIZER WITH XEON PHI

› Coloured by process
Filtering of cores to display
AGENDA

- A bit of background: Debug and Tracing
- CDT Debug and Trace Compass integration
  1. Enhanced Post-mortem troubleshooting
  2. Debugging with Trace snapshots
  3. Tracing with the (Multicore) Visualizer
  4. GDB Traces with Trace Compass
- Conclusion
GDB TRACES WITH TRACE COMPASS
Instrumentation, collection and visualization in CDT
Collected Data

Tracepoint that was hit

Line where trace was collected
GDB TRACES EVENT TABLE

Synchronized Trace Compass's Events Table
CONCLUSION
MULTICORE DEBUG GROUP

› Joint effort to bring multicore debugging to the CDT
  – Visualizer, Pin&Clone, Multiprocess, etc

› Support for those that want to add new features

› Monthly conference calls (open to all interested and free 😊)
  – http://wiki.eclipse.org/CDT/MultiCoreDebugWorkingGroup
MORE ON TRACING

› Learn more about tracing and Trace Compass:

› Thursday 12 noon in Harbour AB with Marc-Andre:

  ➔ “Analyzing Eclipse Applications with Trace Compass”

Thursday, March 12, 2015

<table>
<thead>
<tr>
<th></th>
<th>Grand Peninsula D</th>
<th>Grand Peninsula EFG</th>
<th>Harbour AB</th>
</tr>
</thead>
<tbody>
<tr>
<td>12:00 - 12:35</td>
<td>PolarSys Day</td>
<td>Marc-Andre Laperle [Ericsson]</td>
<td></td>
</tr>
</tbody>
</table>
SOME REFERENCES

› Integration on GitHub,
  https://github.com/MarkZ3/Trace-Compass/tree/dsf-mv-integration

› CDT Project, http://www.eclipse.org/cdt
› Trace Compass,
  https://projects.eclipse.org/projects/tools.tracecompass

› CDT Debug workgroup
  http://wiki.eclipse.org/CDT/MultiCoreDebugWorkingGroup
› CDT Wiki, http://wiki.eclipse.org/CDT
Evaluate the sessions

Sign in: www.eclipsecon.org
BONUS SLIDES
OTHER CDT DEBUG NEWS
DEBUG VIEW LABELS

- GDB binary name/version
- Thread Names
PER-ELEMENT FORMAT

› Ability to set format per element

› Variables, Expressions, Registers views
› Ability to create groups of registers
- Ability to pin a Multicore Visualizer to a session
- Allows to monitor multiple systems concurrently
MINI CORE DUMPS

› Effort of the Linux Diamon workgroup (diamon.org)

› Mini core dumps:
  – Configurable excerpt of full core dump
  – Space savings (good for embedded)
  – Storage of multiple mini core dumps

› Coming to a Linux distribution in the near future!
FUTURE PLANS
GLOBAL BREAKPOINTS

› Contribution to Linux Kernel ongoing

- Applies to every process
- Auto attach when hit
- Un-started or short lived process

(gdb) gbreak return.cc:14
Global Breakpoint 6 at 0x80484aa: file return.cc, line 14.
INTEGRATED GDB CONSOLE

› Coming in 2015!

ECLIPSE’S GDB-CONSOLE

- PROMPT
- EVENT REPORTING
- COMMAND HISTORY
- COMMAND COMPLETION
- SYNCHRONIZED WITH GUI
- INTEGRATED OR STAND-ALONE
Process Thread Core (PTC) sets control groups of debug elements:

- Step threads numbered between 34 and 59
  
  (gdb) step 34-59

- Step all threads running on core 2
  
  (gdb) step @2

- Stop everything running on cores 5 to 7, preventing new threads from being started
  
  (gdb) interrupt *,future@5-7