Dynamic and Static Tracepoints using GDB (and LTTng)

Marc Khouzam, Ericsson Canada
marc.khouzam@ericsson.com

Agenda

- The context
- The need
- The solution
  - GDB's new tracepoint feature
  - Dynamic and static tracepoints
  - Tracing and visualizing trace data
  - Integration in Eclipse
- The timeline
- Questions

The Context

- Ericsson provides customers with large telecom applications
- These have strong constraints
  - Real-time
  - High-availability
  - Multi-process, multi-core, multi-processor
  - Highly complex
- They also use a wide variety of:
  - Hardware
  - Operating-systems (some proprietary)
  - Platforms (many proprietary)
  - Programming languages (some proprietary)

The Context

- Troubleshooting such apps is extremely difficult
  - Different phases (design, test, integration, live site)
  - Pin-pointing a problem is tedious and difficult
  - Debugging is often too intrusive
- Other complexities
  - Off-the-shelf components
  - Third-party tailored-made components
  - Customer-built contributions
  - …

The need

- Extremely low intrusiveness
  - For live sites
  - To be able to reproduce some race conditions
- Support for different architectures and operating systems
- Support for different programming languages
- Support for multi-processes, multi-cores, multi-processors

The need

- Can be used by designers, testers, integrators, network operators
  - Different training
  - Different environment
  - Different time constraints
- Can work with third-party components

The solution

- Highly efficient tracing tool
- Support for Dynamic Tracepoints
  - Added dynamically while code is executing
- Support for Static Tracepoints
  - Added in the source code, before compilation
- Support for disconnected tracing
  - Ability to set tracepoints then disconnect while data collection continues

The solution

- GDB (GNU Debugger)
  - Well-established and widely used
  - Open-source
  - Supports wide variety of CPUs
  - Supports wide variety of Operating Systems
  - Supports different programming languages
  - Provides live control of a running target
  - Provides control of remote targets
  - Can be easily be ported to new targets
  - Already provides a first dynamic tracepoint feature

The solution

- LTTng (Linux Tracing Toolkit)
  - Highly efficient
  - Open-source
  - Proven itself for Kernel tracing
  - Provides static tracepoints
  - Can be controlled at run-time by GDB
- Eclipse
  - Ericsson's chosen platform for tool integration
  - Open-source
  - Well-established and widely used
  - Already has a GDB integration

GDB's New Tracepoint Feature

- Supporting both Dynamic and Static tracepoints
- Tracepoint support using gdbserver
  - Tracing on the host can still be done using gdbserver
- Tracepoints implemented by
  - Breakpoints (slow dynamic tracepoints)
  - Jump-patching (fast dynamic tracepoints)
  - User-space LTTng (static tracepoints)
- Observer-mode to enforce tracing instead of debugging

Dynamic Tracepoints

- Creation of tracepoint as is done for breakpoints
- Enable/Disable tracepoints dynamically
- Dynamic condition can be assigned to a tracepoint
- Specification of data to be gathered using symbolic expressions and memory addresses (actions)
- Trace-state variables that can be used in conditions and actions
- Automatic timestamp collection on successful tracepoint hit

Dynamic Tracepoints

- Are only in effect if tracing is enabled
- Possible to define global actions (affecting all tracepoints)
- Hit count per tracepoint to stop tracing automatically
- Option to use a finite trace buffer or circular trace buffer
- Disconnected data gathering
- On-disk trace data storage for ‘small’ amounts of data

Two kinds of Dynamic Tracepoints

- Slow tracepoints using ptrace interface
  - using breakpoints and automatic resuming
  - handled by gdbserver itself
  - writes data to a gdbserver buffer
- Fast tracepoints using an in-process library
  - using jump patches
  - restricted to 5 bytes instructions
  - will give error or use slow tracepoint if installation fails
  - writes data to an in-process buffer
  - if condition of tracepoint is false, the tracepoint will take < 100nS

Static Tracepoints

- Creation of tracepoint is done by designer before compilation

As for Dynamic tracepoints:

- Enable/Disable tracepoints dynamically
- Dynamic condition can be assigned to a tracepoint
- Can additionally have dynamic tracing specified using symbolic expressions and memory addresses (actions)
- Trace-state variables that can be used in conditions and actions
- Automatic timestamp collection on successful tracepoint hit

Static Tracepoints

Also like Dynamic Tracepoints:

- Possible to define global actions (affecting all tracepoints)
- Hit count per tracepoint to stop tracing automatically
- Option to use a finite trace buffer or circular trace buffer
- Disconnected data gathering
- On-disk trace data storage for ‘small’ amounts of data

Static Tracepoints provided by LTTng

- Using User-space LTTng library
- Program to be traced is linked with LTTng library
- During tracing, user program calls LTTng library which calls GDB's in-process library
- Write data to the same in-process buffer as dynamic tracepoints
- Can be listed by GDB

- (More detailed presentation on Wednesday afternoon)

Tracing Experiment

- Time during which trace data is collected
- Triggers collection from enabled dynamic and static tracepoints
- Started and stopped by user
- Can also stop automatically due to hit count
- Runs independently of GDB connection
- Must be restarted to add/remove/modify tracepoints
- May allow for live examination of data

Trace Data Visualization

- Navigation through data records using GDB
  - From the start of the data
  - From a specific tracepoint
  - From a line or address in the code
- Each data record is a snapshot of debug information
- Records can be examined using standard GDB features
- Scripting can be used to run through trace data
- All collected data of a record can also be dumped as plain text
- Trace data can be saved to file
- Saved trace data can be examined offline

Eclipse Integration: Tracepoint Control

- First effort focuses on Dynamic Tracepoints
- Tracepoints to be handled like breakpoints with a visual differentiator
  - Create/Delete
  - Enable/Disable
  - Conditions
  - Actions
- Ability to start/stop tracing
- Synchronization of Tracepoints

Eclipse Integration: Trace Visualization

- Navigation through the trace data to change the point in time
  - across all trace data
  - across trace data of a specific tracepoint
- Navigation starting point
  - Start of data
  - Specified timestamp
  - Specified tracepoint ID
- Trace data information
  - Amount of data collected
  - Time span of the data collected
  - Hit count for each tracepoint

Eclipse Integration: Trace Visualization

- Data displayed in debugger view
  - As if debugger was attached at a specific point in time
  - Only collected information can be shown
  - Highlighting of the tracepoint of interest
- Data displayed as text
  - All collected data
  - Data from a specified starting point
  - Data of the current trace record
- Offline trace visualization in the exact same fashion

The Timeline

1. Dynamic tracepoints in GDB for a remote target (not Linux)
2. Dynamic tracepoint support in Eclipse
3. Dynamic and Static tracepoints in gdbserver (i.e., Linux)
4. Static tracepoint support in Eclipse

➔ All this targeted for end of 2009

Questions?