



Reverse, Multi-Process and Non-Stop Debugging come to the CDT

Or: *How I Learned to Stop Worrying and
Love GDB (and DSF)*

Marc Khouzam, François Chouinard

Agenda



- Ericsson: What are we doing here?
- DSF Overview
 - Demo 1
 - What's New
 - Multi-threading
 - Multi-process
 - Upcoming Debugging Features
 - Demo 2
 - Reverse Debugging
- Questions

Ericsson Context



- Not a tool vendor!
- 5 main proprietary platforms (sort of... 😊)
 - Some with proprietary OSes
- Each PF requires its own development environment
 - Heavy use of tools in all phases of SW development
 - Standard, everyday tools (e.g. editors, compilers, CM, ...)
 - Custom tools for “advanced” needs

- Eclipse: our tool integration platform of choice
- Push for an Eclipse-based IDE for each platform
 - Open source solutions when available
 - Vendor or in-house solutions for the rest

- Everything was cool...



*Then the sh*t hit the fan...*

- 'Difficulties' at integrating a debugger for one of our proprietary platforms
- In-house CDI-GDB based solution (incomplete)
 - Worked for the emulator, not the real target
 - Support for freeze-mode only (a.k.a. all-stop)
 - Lacking support for:
 - Multiple targets
 - Multiple processes
 - Non-stop multi-threading
 - Awkward support for:
 - Execution model
 - Data model
- Internal study concluded that:
 - GDB could be enhanced to bridge the gap
 - CDI-based solution would eventually be problematic
 - DD/DSF offered a better alternative



GDB Improvements

- Support for non-stop multi-threading
 - Run mode selection (all- or non-stop)
 - Run control - select/suspend/resume/... individual threads or groups of threads

- Support for multi-process
 - Direct support for multiple processes
 - Handling of execution contexts and aggregation in GDB
 - Global breakpoints
 - Auto-attach

- Some proprietary extensions for our specific needs (not contributed back)

CDI-GDB Challenges



- No direct support for proprietary platform features
 - Non-standard execution and data models
 - Multiple targets
 - Multiple processes
 - Non-stop multi-threading

- Overall performance issue
 - Synchronous debugger communication
 - Slow data retrieval (for large data structures)
 - Stepping-in for large projects
 - Scalability ~

- CDI could have been extended, but:
 - Difficult to integrate our changes in the code base without breaking compatibility ☹
 - Would likely have had to maintain our patches outside of CDI ☹

DSF Features



- Aimed at embedded systems ☺
 - Support for non-standard execution/data models
 - Asynchronous *everything*
 - Command caching and coalescing (in theory...)
 - Fetch only what is required by the UI
 - Scalability !
- Built-in support for concurrency ☺
 - Asynchronous interfaces
 - Threading model
 - Thread pool of size 1 (*de facto* global semaphore)
 - Low overhead

DSF Features



- Service oriented architecture ☺
 - OSGi-based
 - Can easily add/extend/substitute services
 - Run Control
 - Breakpoints
 - Variables, Registers, ...
 - MyExcellentService
 - ...

- Other nice UI features:
 - Update policies
 - Fast stepping
 - New memory rendering (ironically called “Traditional” ☺)
 - New disassembly view
 - Stack Frame partial display

Demo 1



- What's New
 - Launch
 - Views
 - GDB traces
 - As-needed back-end requests
 - Caching (memory)
 - Breadcrumb

- Non-stop multi-threading
 - Thread selection/control

- Multi-process
 - Connecting to running processes

Debug Perspective



The screenshot shows the Eclipse IDE in the Debug Perspective. The main editor displays the source code for `NonStopMultiThreading.cpp`. The code includes a `main` function and a `test_threads` function. The `test_threads` function contains a loop for creating worker threads and a loop for joining them. The `main` function prompts the user for a duration and then calls `test_threads`.

Annotations in yellow boxes point to specific lines of code:

- Breadcrumb activated when you reduce Debug view to 1 line**: Points to line 86, `pthread_create(&threads[i], NULL, worker_thread, (void*) thread_ids[i].c_str());`
- Worker thread creation loop for the multi-thread example**: Points to the loop starting at line 85, `for (unsigned i = 0; i < nb_threads; i++) {`
- Timer thread creation**: Points to line 82, `pthread_create(&timer_th, NULL, timer_thread, (void*) duration);`

The right-hand side of the IDE shows the Variables view with the following table:

Name	Type
	char *
	unsigned int
	int
	pthread_t

The bottom of the IDE shows the Console view with the following output:

```
NonStopMultiThreading [C/C++ Application] gdb
set target-async on
set pagination off
[Thread debugging using libthread_db enabled]
[New Thread 0xb7d37b90 (LWP 5469)]
[Thread 0xb7d37b90 (LWP 5469) exited]
```

Non-Stop Multi-Threading



The screenshot shows the Eclipse IDE's 'Debug Configurations' dialog. The configuration name is 'NonStopMultiThreading'. The 'Debugger' is set to 'gdb/mi'. The 'Stop on startup at' is set to 'main'. Under 'Debugger Options', the 'Main' tab is active, showing the GDB debugger path as '/home/franccis/Workspaces/GDB/GDB-G.8.50/gdb/gdb' and the GDB command file as '.gdbn.L'. The 'Non-stop mode' checkbox is checked, with a note: '(Note: Requires non-stop GDB)'. The 'Enable Reverse Debugging at startup' checkbox is unchecked. A yellow callout box points to the 'Non-stop mode' checkbox with the text: 'Multi-thread apps can be debugged in either all-stop (freeze) or non-stop mode'. The dialog also shows a list of other configurations on the left and buttons for 'Apply', 'Revert', 'Close', and 'Debug' at the bottom.

Multi-thread apps can be debugged in either all-stop (freeze) or non-stop mode

Non-Stop Multi-Threading



The screenshot shows the Eclipse IDE interface during a debug session. The top toolbar includes buttons for Run, Breakpoint, and other debugging actions. The Debug console on the left shows a tree view of threads: Thread [4] 6001 (Suspended: Breakpoint), Thread [3] 5999 (Suspended: Breakpoint), Thread [2] 5998 (Running), and Thread [1] 5995 (Running). The source code editor in the center shows the following code:

```
59 int nb_ iterations = 0;
60
61 cout << "Entering: " << thread_id << endl;
62
63 while (!is_finished) {
64     cout << "\t" << thread_id << " (" << ++nb_ iterations << ")" << endl;
65     sleep(nb_ threads);
66 }
67
68 cout << "Leaving: " << thread_id << endl;
69 pthread_exit(NULL);
70}
71
72//-----
73// test_thread: Starts the various threads and waits for completion
74//-----
75void test_threads(char* duration)
76{
```

Annotations in yellow boxes provide the following information:

- Worker threads 3 and 4 are stopped on the breakpoint (line 64)
- Threads 1 and 2 (main and timer thread) are still running

The console window at the bottom right shows the following output:

```
Entering test_threads()
Timer thread started...
Entering: Thread #3
Entering: Thread #4
```

Non-Stop Multi-Threading



The screenshot shows the Eclipse IDE in debug mode. The 'Properties for' dialog is open, showing the 'Filter' tab. Under 'Restrict to Selected Targets and Threads', the following threads are listed:

- Thread[4] 6001
- Thread[3] 5999
- Thread[2] 5998
- Thread[1] 5995

A callout box points to the 'Thread[3] 5999' entry with the text: "Breakpoint is filtered out for thread 3 i.e. only thread 4 will stop on the breakpoint now".

The main editor shows the following C++ code:

```
59 int nb_iterations = 0;
60
61 cout << "Entering: " << thread_id << endl;
62
63 while (!is_finished) {
64     cout << "\t" << thread_id << " (" << ++nb_i
65     sleep(nb_threads);
66 }
67
68 cout << "Leaving: " << thread_id << endl;
69 pthread_exit(NULL);
70}
71
72//-----
73// test_thread: Starts the various threads and waits for completion
74//-----
75void test_threads(char* duration)
76{
```

Non-Stop Multi-Threading



The screenshot shows the Eclipse IDE in a debug state. The main window displays the 'Debug' console with a tree view of threads. Thread [4] (ID 6001) is suspended at a breakpoint, while threads [1], [2], and [3] are running. A yellow callout box points to the thread list with the text: "You have full control over individual or all threads (suspend, resume, stop, ...)".

The source code editor shows the following code in `NonStopMultiThreading.cpp`:

```
59 int nb_iterations = 0;
60
61 cout << "Entering: " << thread_id << endl;
62
63 while (!is_finished) {
64     cout << "\t" << thread_id << " (" << ++nb_iterations << ")" << endl;
65     sleep(nb_threads);
66 }
67
68 cout << "Leaving: " << thread_id << endl;
69 pthread_exit(NULL);
70}
71
72//-----
73// test_thread: Starts the various threads and waits for completion
74//-----
75void test_threads(char* duration)
76{
```

The Console window on the right shows the output of the application, including "Timer thread started...", "Entering: Thread #3", "Entering: Thread #4", and a list of thread activity for Thread #4 (1) through (9) and Thread #3 (1) through (9).

A second yellow callout box points to the Console window with the text: "And voila! Thread 4 is stopped while thread 3 happily does its job".

Multi-Process



Dynamically connect to and disconnect from processes

Name	Type	Value
a	int	8
pa	int *	0xaa62f88
b	int [2]	0xaa62f7c
b[0]	int	5
b[1]	int	4
ptr	int *	0x100

Multiple processes running in the same launch. They can be individually controlled and inspected.

```
int mainExpressionTestApp() {
    printf("Running ExpressionTest App\n");
    int a = 8;
    int* pa = &a;
    int b[2] = {3, 4};
    b[0] = 5;
    b[1] = 6;
    int* ptr = new int;
    printf("ptr points to 0x%X\n", ptr);
    testLocals();
    testChildren();
    testWrite();
    testName1(1);
    testName2(2);
    testName1(3);
    testSameName();
    testConcurrent();
    testSubblock();
    testAddress();
    testUpdateChildren(0);
    testUpdateChildren(100);
    testUpdateChildren2(200);
    testDeleteChildren();
    testUpdateGDBBug();
    testUpdateIssue();
    testUpdateIssue2();
    testConcurrentReadAndUpdateChild();
    testConcurrentUpdateOutOfScopeChildThenParent();
    testUpdateOfPointer();
}
```

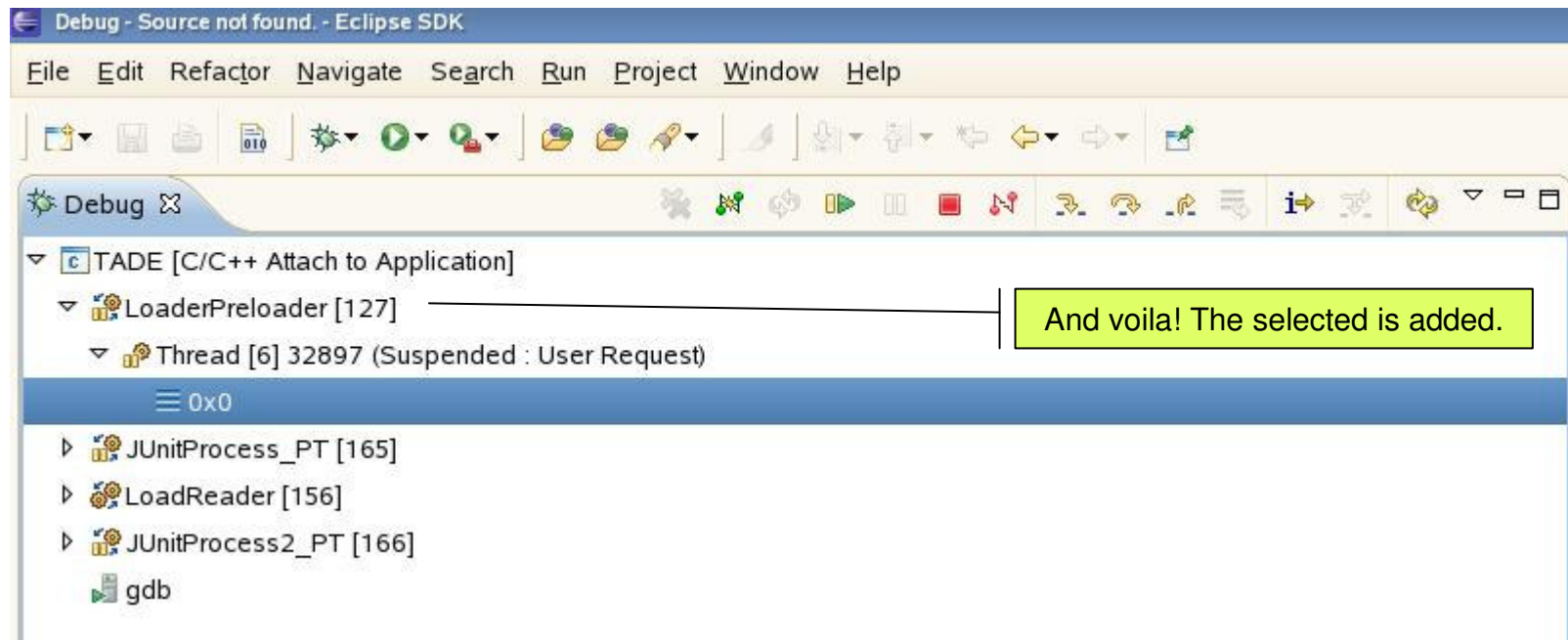
```
TADE [C/C++ Attach to Application] gdb
[New Thread 165.32960]
Current language: auto; currently c++
[New Thread 166.32961]
[New Thread 95.32865]
[New Thread 156.32936]
[New Thread 165.32960]
```

Multi-Process



The screenshot shows the Eclipse IDE interface. In the background, there is a project explorer on the left showing a file structure with paths like 'ADE/example/JUnitProcess_OU/src/ExpressionTestApp.cc'. The main editor area shows a code snippet with variables: 'pa int *', 'b int [2]', and 'b[0] int'. A 'Select Process' dialog box is open in the center, titled 'Select a Process to attach the debugger to:'. It contains a list of processes: JVM - 135, JVM Launcher - 129, KernelUtilityDaemon - 13, LoaderPreloader - 127 (highlighted), LoadReader - 156, NetServer - 155, NtpManagement_PT - 106, ProfilerDaemon - 14, PSM-agent - 29, and PSM-master - 30. Below the list is a search field and 'OK' and 'Cancel' buttons. A callout box with a yellow background and black border points to the dialog with the text: 'Connect: you can attach to any running process'. The background also shows a 'Memory' view with addresses like 2960, 2961, 865, 2936, and 2960.

Multi-Process



Upcoming Debugging Features



- Reverse Debugging
- Tracepoints
- Multi-exec
- True GDB Server

Reverse Debugging



- Allows to step program backwards and forwards
 - Avoids repeating test over and over to gradually hone in on bug
 - Allows to replay a bug
 - Allows to go back and change program execution without having to recompile and repeat the test (some targets)

- Often available in simulators/emulators
 - VMWare
 - Simics

Reverse Debugging



- IDE front-end support now part of the CDT
 - Buttons, menus and key bindings for
 - Reverse Resume
 - Reverse StepIn
 - Reverse StepOver
 - Uncall
 - Perspective customization to show/hide these new UI features

- Availability for extension
 - Currently in DSF-GDB
 - Will eventually migrate to DSF
 - May even go to Debug Platform

GDB and Reverse Debugging



- Reverse Debugging infrastructure in GDB HEAD
 - Allows to hook to target that support reverse

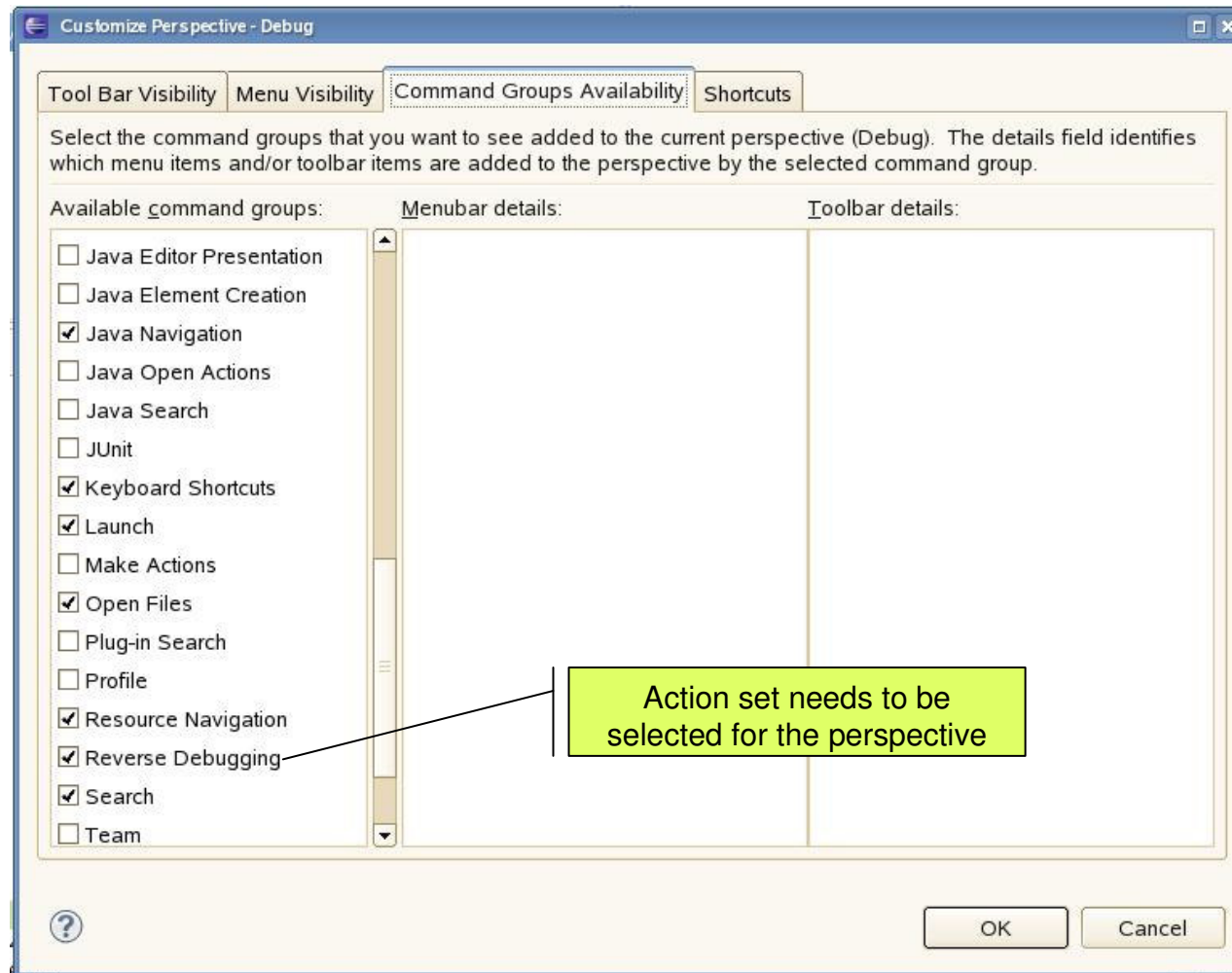
- Linux PRecord (Process Record and Replay) in GDB HEAD
 - By Hui Zhu (Teawater)
 - For Linux target
 - Records memory and register changes
 - 3 minor fixes are still awaiting approval

Demo 2

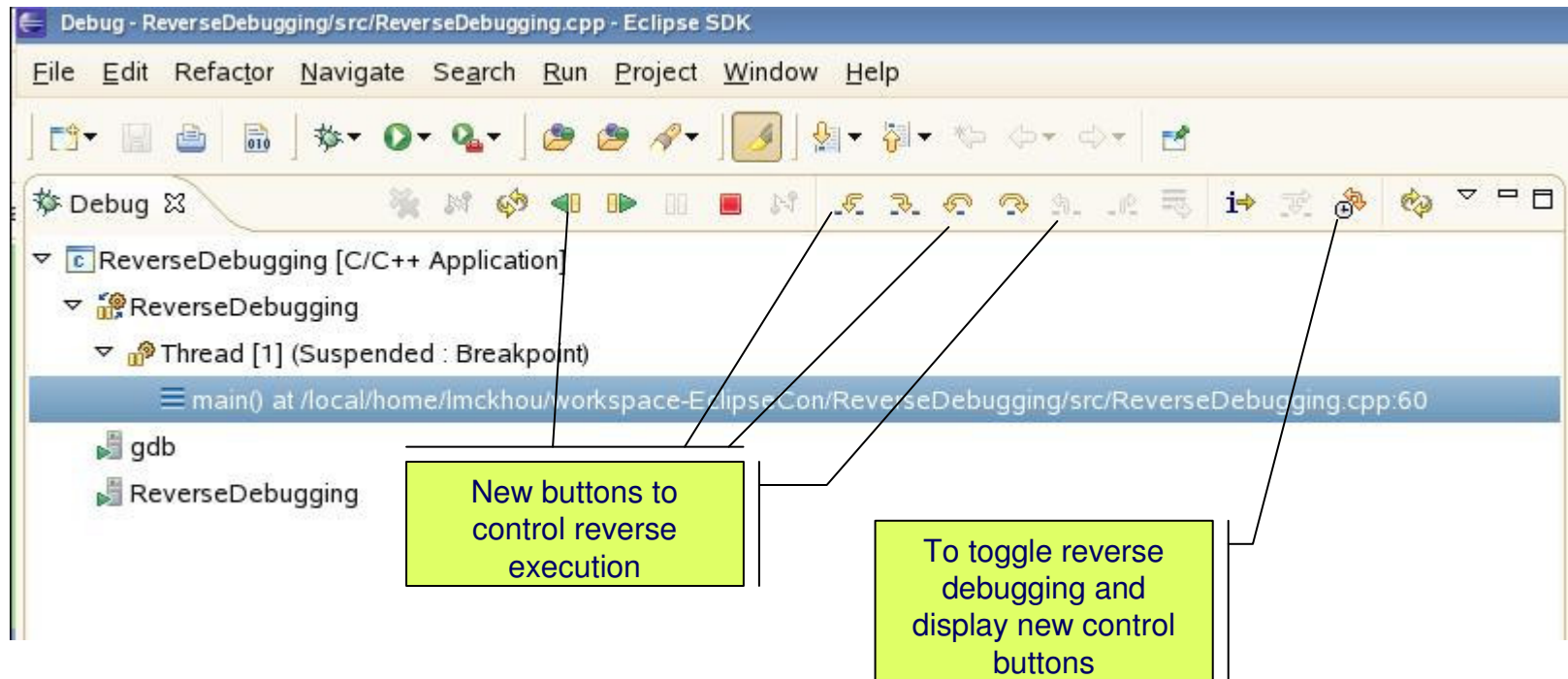


- Reverse Debugging
 - New reverse debugging action set
 - Reverse step-in, reverse step-over, resume, uncall
 - Buttons, key bindings and menus
 - Launch option
 - Views in reverse
 - Change execution path

Reverse Debugging



Reverse Debugging





Tracepoints

- Dynamic tracepoints
 - Add instrumentation to running code
 - Low-overhead
 - Enable/Disable dynamically
 - Trigger on user-defined condition
 - Off-line tracing
 - Trace collection from target

Multi-Exec



- In CDT 6.0
 - Multi-process support in DSF
 - Attaching to multiple processes in DSF-GDB

- Next steps:
 - DSF-GDB support for launching multiple processes in the same debug session
 - GDB support for Multi-process on Linux (should be part of the next release of GDB)



True GDB Server

- GDB provides gdbserver
 - Allows to debug remote program on Linux
 - Can be used as a basis to write a new debug server for your own OS
 - Accepts a single GDB connection
 - Usually started manually before beginning debugging session

- True gdbserver
 - Daemon
 - Accepts multiple GDB connections
 - Ready for debugging at any time

Additional Information



- **Contacts**

- Marc Khouzam, marc.khouzam@ericsson.com
- François Chouinard, francois.chouinard@ericsson.com

Questions?

