GDB on ARM
Linaro Contributions

Dr. Ulrich Weigand
<ulrich.weigand@linaro.org>
<ulrich.weigand@de.ibm.com>
Agenda

• What is Linaro?
• Native debugging enhancements
  • Back-trace support
  • Hardware break-/watchpoint support
  • NEON vector register support
• Remote debugging enhancements
• Debugging native Android code
What is Linaro?

- An open source development consortium for ARM and the embedded community
- Founded in June 2010. Mission:
  - “to make it easier for ARM partners to deploy the latest technology into optimized Linux based products”
- Not-for-profit software engineering company
  - 130 Engineers as of end 2011
What does Linaro do?

- Delivers an optimized code base
  - Kernel and vital middleware
  - Applied across all member SoCs
- Tools
  - Best compiler, debugger, profiler
- Enabled on the latest SoCs
  - Cortex A8, A9, & A15 processors
- Delivered upstream
  - Evaluation builds for key distributions – Android, Chrome, MeeGo, Ubuntu
  - Test & Validation framework for member SoCs
Where does Linaro fit?
Linaro Toolchain Working Group

• Linaro Toolchain releases
  • Monthly GCC, GDB, QEMU releases

• Major focus areas
  • Compiler performance improvements
    – e.g. auto-vectorizer, scheduler, ARM back-end ...
  • Exploit latest ARM architecture features
    – e.g. STT_GNU_IFUNC support, optimized string routines
  • Support current hardware in QEMU
  • Reliable debugging support
Native debugging enhancements

• Linaro focus areas
  • Enhanced back-trace support
    – Prologue parsing for Thumb-2 code
    – Backtrace using ARM exception tables
    – Backtrace out of kernel vector page
    – Backtrace out of glibc system call stubs
  • Support ARM hardware break-/watchpoints
  • Support VFP/NEON registers in core files
  • Fix numerous GDB test suite failures on ARM
Basic algorithm

- Start with initial register set (frame #0)
- Extract PC from register set
- Determine register unwind instructions at PC
  - “Restore PC from LR”
  - “Add 128 to SP”
  - “Restore R8 from memory at location (old) SP + 80”
  - “Register R10 is unchanged”
  - “Register R2 cannot be unwound; its prior value is lost”
- Given old register set and memory contents, apply unwind instructions to construct register set at next frame (frame #1)
- Repeat until uppermost frame is reached
Back-trace support on ARM

- How to determine unwind instructions at PC
  - Use DWARF-2 Call Frame Instructions (.debug_frame; on non-ARM also .eh_frame)
  - Use ARM exception table information (.ARM.exidx / .ARM.extbl)
  - Disassemble start of function containing PC and interpret prologue
  - Hard-coded special cases (e.g. signal return trampolines, kernel vector page stubs)

- Challenges on ARM
  - No .eh_frame section means no DWARF CFI in the absence of debug info
  - ARM exception tables were not supported in GDB
  - Glibc assembler code was not (always) annotated with ARM exception tables
  - Prologue parsing did not handle the Thumb-2 instruction set
    - Note that Thumb-2 is the default on current Ubuntu distributions

- Current status
  - Support for all missing features added
  - No GDB test case fails due to unwind problems
    - This is true even in the absence of system library debug info packages
ARM hardware watchpoints

- Feature set
  - Hardware watchpoints
    - Trap when a pre-defined memory locations is modified
    - Used to implement “watch” family of commands in GDB
  - Hardware breakpoints
    - Trap when execution reaches a specified address
    - Used to implement “hbreak” family of commands in GDB
    - Useful in particular to set breakpoints in non-modifyable code (e.g. ROM)

- Current status
  - Hardware breakpoint/watchpoint support added to Linux kernel 2.6.37
  - Support exploited by GDB 7.3

- Hardware pre-requisites
  - Cortex-A8: limited HW support, not currently exploited by Linux kernel
  - Cortex-A9: improved HW support, Linux kernel supports one single HW watchpoint
  - Cortex-A15: full HW support, Linux (3.2) supports multiple HW watchpoints
VFP/NEON register sets

- Floating-point / vector registers on ARM
  - Past architectures did not specify floating-point or vector registers; some implementations provided those via co-processor extensions
  - ARMv7 specifies VFPv3 and Advanced SIMD ("NEON") extensions
    - VFPv3-D16: 16 64-bit registers / 32 32-bit registers
    - VFPv3-D32: 32 64-bit registers
    - NEON: VFPv3-D32 registers re-interpreted as 16 128-bit registers

- Current status
  - Access VFP/NEON register sets in native/remote debugging: Supported with Linux kernel 2.6.30 / GDB 7.0
  - Access VFP/NEON registers sets in core files: Supported with Linux kernel 3.0 / GDB 7.3
Remote debugging enhancements

• Linaro focus areas
  • Basic remote debugging via gdbserver
    – Miscellaneous test suite fixes and enhancements
    – Fix problems with “remote:” sysroot access
  • Additional gdbserver / remote debugging features
    – Support hardware break-/watchpoints
    – Disable address space randomization
    – Core file generation and “info proc”
Background: Native debugging

- Application
- Source code
- System Header files
- System Libraries
- Application Executable
- Compiler
- GDB
- ptrace

Linaro
Background: Remote debugging

- Cross-GDB
- Application Source code
- Target Header files
- Target Libraries
- Application Executable
- Cross-Compiler
- GDBserver
- System Libraries
- Application
- Remote protocol
- ptrace
- copy
- Host
- Target
Remote debugging challenges

- **GDB accesses application binary / target libraries on host**
  - Assumes these are identical copies of files on target
    - Debugging will (silently) fail if that assumption is violated
  - Solution: Have gdbserver access files on target
    - Contents forwarded via remote protocol
  - Status: Implemented; enable via “set sysroot remote:”

- **Native target and gdbserver target feature sets differ**
  - Both implement similar functionality but do not share code
  - Some native features missing from remote debugging (and vice versa)
  - Long-term solution: Code re-factoring to allow re-use of identical code base
  - For now: Narrow gap by re-implementing missing gdbserver features
    - Support hardware break-/watchpoints
    - Disable address space randomization
    - Core file generation and “info proc”
Debugging Android

- Android applications
  - Usually implemented in Java, running on Dalvik
  - “Native” components can be implemented via NDK
  - GDB used to debug native Android code, usually configured as cross-debugger with gdbserver running on the Android device

- “Android vs. Linux” differences visible to GDB
  - Mostly caused by different libc (Bionic vs. glibc)
  - ABI issues: jmp_buf layout (setjmp/longjmp); signal frames
  - Debugging multi-threaded applications
    - GDB relies on “libthread_db” shared library provided by C library to inspect internal libc/libpthread data structure
    - Bionic does not have libthread_db; Android “fakes” it by looking at kernel data exported via /proc etc.
Debugging Android

• Current status
  • Android NDK provides patched version of GDB and gdbserver
  • Some third parties likewise provide versions with extra patches

• Goal
  • Have Android as fully supported target OS
    – Automatically detect whether target uses Bionic or glibc
  • Support both GDB native and gdbserver remote debugging
  • Everything available in upstream GDB without patches

• Work on this goal currently under way in Linaro
Summary

- Linaro is a not-for-profit engineering organization consolidating and optimizing open source Linux software and tools for the ARM architecture.
- One of the ongoing focus areas of Linaro's Toolchain Working Group is enhancing the GNU Debugger to provide a first-class debugging experience on current ARM-based devices.
- In GDB 7.3 Linaro contributed enhancements to bring the native debugging experience on ARM on par with other platforms.
- In GDB 7.4 Linaro contributed enhancements to bring the remote debugging experience, in particular on ARM, closer to the native feature set.
- Currently, work is under way to help fully integrate support for debugging Android native code into mainline GDB.
Questions?