Papers

1. title: Implementing a GDB Stub in Lightweight Kitten OS
   1. Summary :
      1. they solved the problem of debugging user applications on HPC kitten OS.they implemented a kernel-module gdb-stub to mimic GDBServer due to lake of ptrace and partial-signals support in kitten OS

differences betweee KGDB and gdb stub

KGDB gdbstub

run in same address pace with kernel run in kenrnel address space

stop all threads once get control stop focused threads/processes

one debug target:Krenel many debug targets : processes

embedded system architecture : indeoendent from the architecture as it just interfaces with the OS level not the application low-level context

* 1. target OS and features: HPC Kitten OS supports multiprocess,multithread
  2. communication medium : virtual serial port emulated via Palacios VMM
  3. stub execution level : kernel module in kernel space,
  4. stub structure :
  5. stub features :
     1. seial Device IO COM
     2. basic RSP commands
     3. reading/writing regisetres via OS Task Control Block
     4. reading writing memory through OS address translation functions
     5. setting breakpoint via INT3 instruction
     6. single stepping via per-thread trap flag
     7. All-Stop single-process mulithread debugging
     8. handling concurrent threads debug-events through serializing
     9. global breakpoint list for each process
     10. support more than client gdb instance
  6. issues regarding portability: not specified --> source code
  7. other

1. title: Design and Implementation of Retargetable Software Debugger based on GDB
   1. Summary:

they implemented a basic Exception-lrvrl GDB Stub as part of their work to acheive a new Software debugger of Core-A architecture. they verified the debugger's functional operations on set of programsagainst ARM AXD debuggere on different HW platform

comparasion was about breakpoint usage, running target ,single stepping and variables inspection

* 1. embedded system architecture: ARM Core-A architecher based ISS
  2. target OS and features: none
  3. communication medium : not specified
  4. stub execution level :Exception-level
  5. stub structure :not specified
  6. stub feature:
     1. support for RSP commands g,G,m,M,z,Z,s,c
     2. issues regarding portability:
     3. they claim it can beeasily ported to other HW architectures like x86,ARM,MIPS
  7. other

1. title: Howto: GDB Remote Serial Protocol.Writing a RSP Server
   1. Summary:

They implemented a basic gdbserver for bare-metal application runs on or1Ksim OpenRISC-1000 Simulator

* 1. embedded system architecture : OpenRISC-100
  2. target OS and features: none
  3. communication medium : TCP/IP connection
  4. stub execution level : Exception Level
  5. stub structure :
     1. 1- External code interface (RSP initialization , RSP Exception Handler)
     2. 2- Packet interface (getting packet, processing packet)
     3. 3- IO Serial Interface
  6. stub features:
     1. reading/Writing Memory
     2. reading /writing register(s)
     3. binary memory write
     4. dummy query-packets support
     5. target restart and detach
     6. stepping
     7. continue
     8. reporting last exception signal
     9. Software /HWbreakpoints
     10. issues regarding portability
  7. other : source codedoes not actually make a real step /continue

1. title: Implementing a Remote Debugging Agent Using the GNU Debugger
   1. summary:

he based his paper on putting reader on the road of understanding basic

knowledge of Remote debugging and GDB RSP agent to achieve remote debugging for

embedded system development field. He introduces a good "gdb in action" start. Then,

he introduces a behavior explanation of basic GDB RSP command packets. he implemented basic gdb stub that supports HW single stepping ,SW-Single stepping ,SW breakpoints

* 1. embedded system architecture : SH2-architecture
  2. target OS and features : none
  3. communication medium : not specified
  4. stub execution level : Exception level
  5. stub structure : not specified --> source code
  6. stub features:
     1. HW single stepping
     2. SW-Single stepping
     3. SW breakpoints
  7. issues regarding portability: not specified
  8. other:

1. Title:A symbolic Debugger for powerPC-Based Hardware
   1. Summary :they implemented a gdbserver running on host machine to debug PowerPC-based bare-metal program. GDBserver splits into two level: highlevel that communicates with GDB on same host machine through TCP/IPport and low-level communicating with ESP-HW Software driver.
   2. Embedded architecture: PowerPC-based board
   3. target OS:none
   4. communication medium:ESPJTAG HW and TCP/IP port for host GDB
   5. stub execution level : Host side process
   6. stub structure:
   7. high-level portion : connects to Host GDB
   8. low-level portion : connects to ESP-HW SW driver interface
   9. stub features:
      1. target start/stop
      2. reading/writing registers
      3. reading/writing memory
   10. issues regarding portability: not specified
   11. other:

Implemented Stubs

1. OCDRemote and LPC2148
   1. OCDRemote is a commercial utility starts up on host machine and listens on a TCP/IP port for a connection from gdb and then translates gdb commands into JTAG/BDM actions for the target processor using the Macraigor hardware interface device. It supports all basic commands that gdb can issues to a bare-metal application(stop/resume targets, HW/SW Breakpoints,stepping/reading/writing memory) running on ARM,M68K,MIPS,PowerPC,X86-based targets.beside basic commands , it support a monitor command to extend GDB functionality to be server-specific. For monitor commands , it supports stop/run all processors , select/contol specific processor for debugging. limitations includes that it is working at exception level , it can not control application threads/tasks , it is not aware of the threads to processor core mapping and control.
   2. []made a board that runs a mini debugger GDB-server communicate with GDB vis RSP over UART. This debugger board accepts commands from GDB and translates it to JTAG commands sending them to that debuggee taret board via JTAG HW interface. According to their dubug agent , they suppor a very basic command packets like (qOffset,?,p,m,c,k,s) to allow step,stepi and info register gdb commands.
2. Linux GDB Server
3. openOCD has original purpose of provide debugging, in-system programming and boundary-scan testing for embedded target devices through HW debug adapter.it also allows subset of supported architectures to be debugged by GDB by providing a OpenOCD server process running on host listening to GDB command packets via TCP?IP or Pipe ports.Server provide 3 levels of debug functionalities for bare-metal applications
   * 1. basic debug commands and Reconfiguration for GDB behaviour
   1. Monitor commands for setting some architecture-specific features
   2. RTOS-awareness
      1. eCos,ThreadX, FreeRTOS,linux,ChibiOS embKernel and mqx.
      2. Require to export per-RTOS symbols from the program to be able to manipulate them
   3. Control processor cores for resuming program using defined <j>/<J> RSP Packets
      1. j smp status request
         1. jc packet: reading core id displayed by GDB connection
      2. J smp status set
         1. Jc XXXX : setting core id displayed at next GDB continue
      3. can be handled in gdb by either
         1. maintainance command to send extended packets to server
         2. using the special gdb variable $\_core inside gdb (set $\_core / print $\_core)
   4. multithread debugging
      1. need deep investigation in code
      2. it uses the rtos symbols to retrieve thread information
4. Ecos GDBStub section 8.2 and chapter 9 in ESdevelopment with Ecos
   1. Ecos RTOS implemented gdbstub as part of RedBoot. Gdbstub has layers
      1. eCos HAL layer
         1. serial port communication
         2. Exception handlers for breakpoint
      2. RSP layer
      3. Features
         1. Supported packets
            1. G,c,s,k,?,b,A,p,P,r,C,S,D,z,Z,deprecated q for thread information,H,T
         2. Multithread support ??
5. Qemu GDBStub is an internal part of Qemu emulator source code
   1. It starts bebore running the loaded program
   2. Qemu GDBStub is connected to host GDB vial local TCP/IP port and has the following features
      1. Supported packets
         1. ?,c,C,vCont,k,D,s,g,G,m,M,p,P,z,Z,H,q,Q
6. RTEMS GDBStub differs from RT-thread stub in two ways. First, it's high-level stub that runs at user-level to debug application tasks. Thus it's a daemon task that execute debug commands. Secondly, RTEMs stub is less intrusive in that GDB communication not stop all target tasks.It does this by maintaining internal list of stopped task that can inspected(stack frame,registers) and resumed. Other tasks not in focus of debug events (breakpoint ,stop,resum) runs normally according to their scheduling. A helper task also help in this intrusive reduction by being interrupted by scheduler and handles only non-task related work like inspect/change memory,disassemble. This helper task can be seen as a deffered ISR (to decrease interrupt latency).Stub accepts GDB connections over serial, TCP and UDP.
   1. Beside basic RSP commands(?,d,D,g,G,m,M,s,c,P,Z0,z0), RTEMS GDBStub supports GDB thread commands (and thread-related packets)which can be partially summarized. User commands are info threads, thread <rtems\_Task\_id>. RPS packets are Hc,Hg,T,QThreadExtraInfo,QThreadInfo.
   2. It has been ported to i386/pc386, m68k/coldfire and powerpc/shared.
7. Stlink GDBStub was designed to debug Cortex-M3/M4 STM32 discovery kits where it runs on host side.It accespts GDB commands and translates it into usb commands which on the target meet the STLink chip that translates commands into JTAG commands.Just as like Setup#3 remote debugging, stub uses TCP/IP as protocol medium
   1. Beside basic RPS commands(c,s,?,!,g,G,p,P, m,M,Z/z1,Z/z2,Z/z3,Z/z4)and optional commands(R, qSupported, qXfer, vKill) ,Stlink-Stub supports GDB Flash interface commands via packets(vFlashErase, vFlashWrite,vFlashDone)
8. RT-thread GDB stub is a low-level stub(Exception-level stub) that runs before RT-Thread Tasks starts. Its not a kernel task. It is one of the basic GDBstub that let GDB see the Whole debugee as single process single-thread program. Its uses UART as RSP medium. RT-Thread supports the basic GDBStub minimal effective RSP commands9(?,q,p,g,P,G,m,X,M,D,k,C,Z0/z0,Z1/Z1,s,c)

Commercial Debugging Tools

1. Isystem : In circuit debugger for Embedded System
   * Provide winIDEA : <http://www.isystem.com/products/software/winidea>
     + Provide an IDE and debugging feature for wide range of targets
       - Basic debug features
         * High/Low-level Debugging
         * Register/Memory views
         * MMU,cache inspection
         * HW and SW breakpoints
       - Trace , Performance analysis,code coverage
       - Multicore debugging : <http://www.isystem.com/downloads/winIDEA/help/> Debug->Multicore debugging
         * IDE instance for each core
         * Primary instance has initialized system and downloaded code file
         * Secondary instances load code file symbols and optionally start secondary cores
         * Cores may be synchronized according to target CPU
       - OS awareness (FreeRTOS, RTX, rcX, CMX, µC/OS-II,...)
     + Debugger
       - Provide basic/multicore debugging of bare-metal target programs using GDB toolchain and over JTAG and others OCD interfaces
       - Support its own HW Debugger and Third party HW debuggers
         * Renesas E1, Segger J-Link, ST-link
     + Communication between host and target
       - Using USB or bluetooth connection
2. Lauterbach TRACE32 — In circuit debugger for Embedded Systems
   * Support of wide range of different types and manufacturer of processing systems
   * It provides HW debugger/Trace solutions
   * Awareness of Very Wide Range of RTOSes
     + Allow performance metrics for RTOS resouurces
     + Displaying thread context
     + Task-related breakpoint for some RTOSes(RTEMS)
   * Interface with GDB
     + As backend
   * Multicore Debugging
     + Using target shared debug interface
     + One or more HW Debugger
     + One client debugger instance for each core
     + Host JTAG server in case of using one HW Debugger
3. UDE
   * Awareness of Multiple RTOSes(Viewing of system resources)
     + rcX
     + Keil RTX
     + CMX-RTX
     + Nucleus PLUS
     + MicroC/OS-II
     + PXROS
   * Multicore debugging : <http://www.pls-mc.com/multi-core-debugging-mca_support/features-a-994.html>
     + Use one interface for all cores
     + Use manager to select cores for debugging
     + Synchronization of cores for the debugging (common start and stop)
     + // visualization of context information after synchronized program execution.
   * Communication : <http://www.pls-mc.com/multi-core-mca_support-with-universal-debug-engine/multi_core_debugging-a-818.html>
     + JTAG
     + But could not understand (is it one or more HW JTAG or one for each core)
4. IAR
   * Uses C-spy Debugger into IDE to support Following features
     + Basic debugger funcrtionalities and view
     + Multicore debugging
       - Stop/Start all cores or individual cores
       - Symmetric multicore debugging
         * One debugger instance
       - Asymmetric multicore debugging
         * Master and slaves debugger instances
         * Instances are highly synchronized
     + RTOS Support
       - Display RTOS Resources
         * Task-specific breakpoints
         * Task-specific stepping
       - Eclipse-based AVIX-RT
       - CMX-RTX
       - CMX-Tiny+
       - eForce mC3/Compact
       - eSysTech X realtime kernel
       - Express Logic ThreadX
       - FreeRTOS, OpenRTOS, and SafeRTOS
       - Freescale MQX
       - Micrium uC/OS-II
       - Micro Digital SMX
       - MISPO NORTi
       - OSEK (ORTI)
       - RTXC Quadros
       - Segger embOS
       - unicoi Fusion.
5. ARM DSTREAM
6. Mentor CodeBench
7. Freescale CodeWarrior