Title: A minimal GDB stub for embedded remote debugging.

Summary :

1. There are two types of stubs one with OS and other without OS support.
   1. With OS support we can ask OS to read/write to Reg Or mem.
   2. Without we need to read the correct location by using system stack or predefined location.
2. Implement a gdb server based on ptrace from linux os
3. This paper illustrate the debugging life cycle between the host machine debugger and the target machine simply.

target OS and features: Linux

---------------------------------------------------------------------------------------------------------------------------

communication medium : Tcp/IP

-------------------------------------------------

stub execution level : in User Space

---------------------------------------------------------------------------------------------------------------------------

stub structure : GDBServer

---------------------------------------------------------------------------------------------------------------------------

Stub features :

Basic Stub Basic commands(Mm,Gg,Zz,Cc,Ss,..).

Reading/writing regisetres via Ptrace helper in linux.

Reading writing memory via Ptrace helper in linux.

Setting breakpoint via INT3 instruction.

All-Stop single-process multithread debugging

---------------------------------------------------------------------------------------------------------------------------

Issues regarding portability: it depends on only linux as it needs build it’s structure based on ptrace.

Title: Research of “Stub” Remote Debugging Technique.

Summary:

1. This paper discussed how to build a stub for your OS .
2. FUNCTIONS OF GDB host machine SUPPORTED
   1. Symbol Table
   2. Communication Protocol
3. FUNCTIONS OF Stub Target machine SUPPORTED
   1. Main Commands in the Stub Module(Mm,Gg,Ss,..).
   2. Debugging environment initialization.

The stub module should not directly run on the bare machine, it needs the support of kernel.

* 1. Implement of the Remote Communication Protocol
  2. The Realization of a Breakpoint.
  3. The Realization of the Single-step

target OS and features: RTEMS (RealTime Executive for Multiprocessor Systems)

---------------------------------------------------------------------------------------------------------------------------

Communication medium: serial communication.

---------------------------------------------------------------------------------------------------------------------------

Stub execution level : not specified in paper ,but there is implementation online for RTEMS Stub

That support :

* Low Level Debugging (Kernel Space):
* Operates at 'ISR' level
* Maximal intrusion complete system is stopped
* Can do ISR debugging.
* High Level (User Space):
* Operates at 'task' level
* Minimal intrusion (only tasks hitting a breakpoint are stopped)
* Needs a functional system
* No ISR debugging

---------------------------------------------------------------------------------------------------------------------------

Stub structure : not specified in paper , but there is a structure on online on <https://github.com/epicsdeb/rtems-gdbstub>

---------------------------------------------------------------------------------------------------------------------------

Stub features :

1-Support Multithread Debugging.

2- Support Tcp/UDP and Serial Communication.

---------------------------------------------------------------------------------------------------------------------------

Issues regarding portability: depends on RTEMS Architecture and helper Function, so may need some time to understand what this helper function in rtems do to port these functionality to our OS and may to a lot of time.

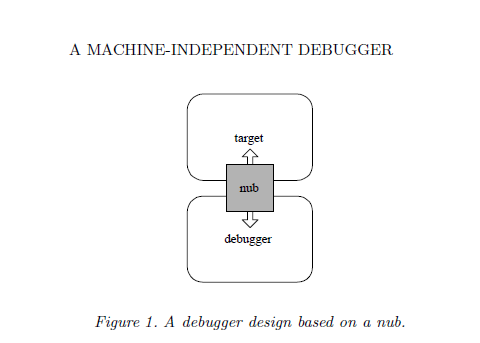
Their code not cleans enough for maintenance and porting to new OS.

**New Approach of debugging**

Title: A Machine-Independent Debugger.

Summary :

* This Paper a new idea of cross debugging that by generation a portable or independent on Target architecture and OS symbol table .
* And the dependence will be in the target by defining a nub interface on target and on host
* The nub on target will initialize the portable Symbol table on host with dependence data see the following Fig



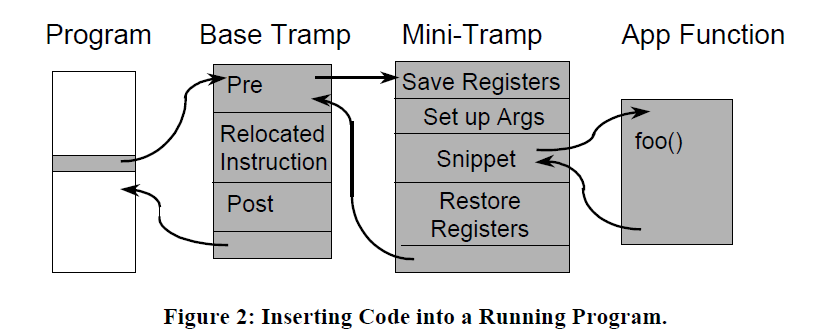
-the main advantage of this the host debugger portability only the dependence on the target , Which means that one program on the host can debug multiple Tragets.

- the Cons of this Idea is the overhead will be on the target to support the host debugger with the dependence information , so Cross debugging prefer that the overhead will be done on the host not to target and also the debugger helper in the target should be simple for maintenance and performance.

Title: An API for Runtime Code Patching.

Summary :

* This paper introduces how to debug in runtime without knowing the symbol table of the debugged program and without debugger also without commands.
* By injecting the debugging code in Run Time as in the following Fig



* example : if we need to put conditional breakpoint we at specified location

1- we will write in that location a branch command to the debugging code in our case called **Base Tramp** in the Base Tramp we will save the current context and write a code (**Snippt**)to check if the value achieved put a breakpoint ins in the location of the running program after that we post the control to the running program.

* Pros
  + It’s not need to a debugger and a symbol table, just you need an API to inject code in runtime
* Cons
  + It’s difficult to be used in cross debugging or remotely debug
  + Can’t be used to debug multi-thread or core environment.
  + User when debug need to write code to debug the running program like

If he need to print value of variable need to write a program that have the address of this variable and get the value from this location and print it and so on so debugging process is slow and un safe.