

Re: How to implement the breakpoint in debugger?

Source: <http://linux.derkeiler.com/Newsgroups/comp.os.linux.misc/2004-06/1217.html>

From: Andy Baxter (news4_at_earthsonline.null.free-online.co.uk)

Date: 06/12/04

Date: Sat, 12 Jun 2004 17:44:54 +0100

Lee said:

> *Yes, english is my second language.:(*
>
> *Sorry for my poor english.*
>
> *What I meat is "if I were writing a debugger, how can it is stopped when*
> *it encounters a breakpoint". I am not focusing on any specific debugger.*
> *Just tell me the general rules please.*
>
> *Thanks all the same,*
>
> *Adrian*

I didn't know, but was interested enough to have a look...

This is from the gdb internals manual at:

http://sources.redhat.com/gdb/current/onlinedocs/gdbint_3.html#SEC6

3.2 Breakpoint Handling

In general, a breakpoint is a user-designated location in the program where the user wants to regain control if program execution ever reaches that location.

There are two main ways to implement breakpoints; either as "hardware" breakpoints or as "software" breakpoints.

Hardware breakpoints are sometimes available as a builtin debugging features with some chips. Typically these work by having dedicated register into which the breakpoint address may be stored. If the PC (shorthand for program counter) ever matches a value in a breakpoint registers, the CPU raises an exception and reports it to GDB.

Another possibility is when an emulator is in use; many emulators include circuitry that watches the address lines coming out from the processor, and force it to stop if the address matches a breakpoint's address.

comp.os.linux.misc: Re: How to implement the breakpoint in debugger?

A third possibility is that the target already has the ability to do breakpoints somehow; for instance, a ROM monitor may do its own software breakpoints. So although these are not literally "hardware breakpoints", from GDB's point of view they work the same; GDB need not do anything more than set the breakpoint and wait for something to happen.

Since they depend on hardware resources, hardware breakpoints may be limited in number; when the user asks for more, GDB will start trying to set software breakpoints. (On some architectures, notably the 32-bit x86 platforms, GDB cannot always know whether there's enough hardware resources to insert all the hardware breakpoints and watchpoints. On those platforms, GDB prints an error message only when the program being debugged is continued.)

Software breakpoints require GDB to do somewhat more work. The basic theory is that GDB will replace a program instruction with a trap, illegal divide, or some other instruction that will cause an exception, and then when it's encountered, GDB will take the exception and stop the program. When the user says to continue, GDB will restore the original instruction, single-step, re-insert the trap, and continue on.

Since it literally overwrites the program being tested, the program area must be writable, so this technique won't work on programs in ROM. It can also distort the behavior of programs that examine themselves, although such a situation would be highly unusual.

Also, the software breakpoint instruction should be the smallest size of instruction, so it doesn't overwrite an instruction that might be a jump target, and cause disaster when the program jumps into the middle of the breakpoint instruction. (Strictly speaking, the breakpoint must be no larger than the smallest interval between instructions that may be jump targets; perhaps there is an architecture where only even-numbered instructions may be jumped to.) Note that it's possible for an instruction set not to have any instructions usable for a software breakpoint, although in practice only the ARC has failed to define such an instruction.

The basic definition of the software breakpoint is the macro `BREAKPOINT`.

Basic breakpoint object handling is in ``breakpoint.c'`. However, much of the interesting breakpoint action is in ``infrun.c'`.

hope this helps.

andy.

--

<http://www.niftybits.ukfsn.org/>

remove 'n-u-l-l' to email me. html mail or attachments will go in the spam bin unless notified with [html] or [attachment] in the subject line.