NAME

kadb – a kernel debugger

SPARC SYNOPSIS

ok boot *device_specifier* **kadb**

> b kadb [**–d**] [*boot-flags*]

x86 SYNOPSIS

> run kadb [**–d**] [*b-flags*]

AVAILABILITY

SUNWcar

DESCRIPTION

kadb is an interactive debugger with a user interface similar to that of **adb**(1), the assembly language debugger. **kadb** must be loaded prior to the standalone program it is to debug. **kadb** runs with the same environment as the standalone program, so it shares many resources with that program. The debugger is cognizant of and able to control multiple processors, should they be present in a system.

Unlike **adb**, **kadb** runs in the same supervisor virtual address space as the program being debugged — although it maintains a separate context. The debugger runs as a *co-process* that cannot be killed (no '**:k**') or rerun (no '**:r**'). There is no signal control (no '**:i**', '**:t**', or '**$i**'), although the keyboard facilities (CTRL–C, CTRL–S, and CTRL–Q) are simulated.

In the case of the UNIX system kernel, the keyboard abort sequence will suspend kernel operations and break into the debugger. On a SPARC system, the keyboard abort sequence is **L1-A** for the console, and BREAK for a serial line. On an x86 system, the keyboard abort sequence is CTRL-ALT-D for the console. **kadb** gives the prompt:

kadb[#]

where # is the CPU it is currently executing on.

Since the kernel is composed of the core image file (**kernel/unix**) and the set of loadable modules already in memory, **kadb** has the capability of debugging all of these by traversing special data structures. **kadb** makes use of this feature to allow it to reference any kernel data structure, even if it resides in a loadable module. **kadb** will set the **'–d'** flag by default so the program being debugged can tell it is being watched. If this flag is not given, **kadb** will load and immediately run the default kernel .

On a SPARC system, *device-specifier* specifies the device to load from (see **monitor**(1M)).

Most **adb** commands function in **kadb** as expected. As with '**adb –k**', **$p** works when debugging kernels. The verbs **?** and **/** are equivalent in **kadb**, since there is only one address space in use.

Additional Commands

kadb has the following additional commands. (Note: for the general syntax of **adb** commands, see **adb**(1).)

[Like **:e** in **adb**(1), but requires only one keystroke and no RETURN character.

] Like **:s** in **adb**(1), but requires only one keystroke and no RETURN character.

Additional x86 Commands

:a Sets a hardware access (read or write) breakpoint using the processor hardware facilities. The syntax and action for this is the same as **:b** in **adb**(1), with the following exceptions:

- the breakpoint will trigger if any bytes from the breakpoint for length bytes are being accessed. See **$l** below for setting the length of a hardware breakpoint.
- breakpoints should be aligned for the length specified. Any address is valid for length 1. Addresses divisible by 2 should be used for length 2 (short). Addresses divisible by 4 should be used for length 4 (long).
- for an access to be detected it must complete the instruction that caused the access. This means that this breakpoint will stop after the access not before.
- there are a limited number of hardware breakpoint registers (4) and when set this will use one.
- since this breakpoint does not modify memory locations this will work on locations that are not in core at the time the breakpoint is set.

:w Like **:a** above, but will only breakpoint on a write to the specified address.

:p Like **:a** above, but will only breakpoint when an instruction at the specified address is run. Note that the **$l** operation has no effect on this type of breakpoint. This breakpoint will occur before the instruction is executed.

$l Sets the default data length for an access or write breakpoint. The syntax for this command is **[length]$l**. Length can be set to one for byte, two for short and 4 for long word accesses. If length is left off byte will be assumed. Once set this value will affect any newly set access or write breakpoints. But will not affect ones set before this operation.

$b Has two additional columns that **adb**(1) does not have. The first is the **type** column which indicates soft for a normal breakpoint, access for an access hardware breakpoint, write for a write hardware breakpoint, and inst for an instruction hardware breakpoint. The second is the **len** column which for access and write breakpoints indicate the length of the operation to break on.

:i Will input a byte for display from the address specified io port. For example .B 330:i will input from address port **330**.

:o Will output a byte to the address specified io port. **[address],[data]:o** will output data byte to address io port. For example **330,80:o** will output 80 to address port **330**.

OPTIONS

–d Interactive startup. If this flag is given, **kadb** will stop after it is loaded and display the prompt as **kadb:** and the name of the default program to debug. For example:

kadb: kernel/unix

The user may either press Return to use the default, or backspace and type in the name of another program to debug.

boot-flags
You can specify boot flags as arguments when invoking **kadb**. Boot flags are passed to the program being debugged. See the manual page **boot**(1M) for relevant boot flags.

USAGE

Kernel Macros

As with **adb**, macros can be used with **kadb,** but they cannot be read from a file at run-time. The **kadb** command **$M** lists all the macros built into **kadb**.

FILES

/platform/*platform-name***/kadb** the debugger itself

/platform/*platform-name***/kernel/unix**
the default kernel

SEE ALSO

adb(1), **boot**(1M)

SPARC Only

kernel(1M), **monitor**(1M), **obpsym**(1M)

DIAGNOSTICS

kadb gives the same cryptic response as **adb**.

WARNINGS

SPARC: **kadb** cannot reliably single-step over instructions which change the processor status register.

NOTES

platform-name can be found using the **–i** option of **uname**(1).