NAME

adb − debugger

SYNOPSIS

adb [**−w**] [objfil [corfil]]

DESCRIPTION

Adb is a general-purpose debugging program. It may be used to examine files and to provide a controlled environment for the execution of UNIX programs.

Objfil is normally an executable program file, preferably containing a symbol table; if not then the symbolic features of *adb* cannot be used although the file can still be examined. The default for *objfil* is **a.out.** *Corfil* is assumed to be a core image file produced after executing *objfil;* the default for *corfil* is **core.**

Requests to *adb* are read from the standard input and responses are to the standard output. If the **−w** flag is present then both *objfil* and *corfil* are created if necessary and opened for reading and writing so that files can be modified using *adb*. *Adb* ignores QUIT signals; INTERRUPT causes return to the next *adb* command.

In general, requests to *adb* are of the form

[*address*] [,*count*] [*command*] [;]

If *address* is present then *dot* is set to *address*. Initially*dot* is set to 0. F or most commands, *count* specifies how many times the command will be executed. The default *count* is 1; *Address* and *count* are expressions.

The interpretation of an address depends on the context it is used in. If a sub-process is being debugged then addresses are interpreted in the usual way in the address space of the sub-process. For further details of address mapping see **ADDRESSES**.

EXPRESSIONS

.	The value of *dot*.
+	The value of *dot* incremented by the current increment.
ˆ	The value of *dot* decremented by the current increment.
"	The last *address* typed.
integer	An octal number if *integer* begins with a 0; a hexadecimal number if preceded by ‘#’; otherwise a decimal number.
*integer***.***fraction*	A 32-bit floating point number.
´*cccc*´	The ASCII value of up to 4 characters. ‘\’ may be used to escape ‘´’.
< *name*	The value of *name*, which is either a variable name or a register name. *Adb* maintains a number of variables (q.v.) that are referred to by the letters **a** to **z** or the digits 0 to 9 (see **VARIABLES** below). If *name* is a register name, then the value of the register is obtained from the system header in *corfil*. The register names are **r0 ... r5 sp pc ps**.
symbol	A *symbol* is a sequence of upper or lower case letters, underscores or digits, not starting with a digit. ‘\’ may be used to escape other characters. The value of the *symbol* is taken from the symbol table in *objfil*. An initial ‘_’ or ‘~’ will be prepended to *symbol* if needed.
routine **.** *name*	The address of the variable *name* in the specified C routine. Both *routine* and *name* are *symbols*. If *name* is omitted, the value is the address of the most recently activated C stack frame corresponding to *routine*.

(*exp*)	The value of *exp*.

Monadic operators

∗ *exp*	The contents of the location addressed by *exp* in *corfil*.
@ *exp*	The contents of the location addressed by *exp* in *objfil*.
− *exp*	Integer negation.
~ *exp*	Bitwise complement.

Dyadic operators are left associative and are less binding than monadic operators.

e1 + *e2*	Integer addition.
e1 − *e2*	Integer subtraction.
e1 ∗ *e2*	Integer multiplication.
e1 % *e2*	Integer division.
e1 & *e2*	Bitwise conjunction.
e1 | *e2*	Bitwise disjunction.
e1 # *e2*	*e1* rounded up to the next multiple of *e2*.

COMMANDS

Most commands consist of a verb followed by a modifier or list of modifiers. The following verbs are available. (The commands '?' and '/' may be followed by '∗'; see **ADDRESSES** for further details.)

? *f*	Locations starting at *address* in *objfil* are printed according to the format *f*. *Dot* is incremented by the sum of the increments for each format letter (q.v.).
/ *f*	Locations starting at *address* in *corfil* are printed according to the format *f* and *dot* is incremented as for '?'.
= *f*	The value of *address* itself is printed in the styles indicated by the format *f*. (For **i** format '?' is printed for the parts of the instruction that reference subsequent words.)

Formats

A *format* consists of one or more characters that specify a style of printing. Each format character may be preceded by a decimal integer that is a repeat count for the format character. While stepping through a format,*dot* is incremented by the amount given for each format letter. If no format is given then the last format is used. The format letters available are as follows.

o	2	Print 2 bytes in octal. All octal numbers output by *adb* are preceded by 0.
O	4	Print 4 bytes in octal.
q	2	Print in signed octal.
Q	4	Print long signed octal.
d	2	Print in decimal.
D	4	Print long decimal.
x	2	Print 2 bytes in hexadecimal.
X	4	Print 4 bytes in hexadecimal.
u	2	Print as an unsigned decimal number.
U	4	Print long unsigned decimal.
f	4	Print the 32-bit value as a floating point number.

F	8	Print double floating point.
b	1	Print the addressed byte in octal.
c	1	Print the addressed character.
C	1	Print the addressed character using the following escape convention. Character values 000 to 040 are printed as @ followed by the corresponding character in the range 0100 to 0140. The character @ is printed as @@.
s	*n*	Print the addressed characters until a zero character is reached.
S	*n*	Print a string using the @ escape convention; *n* is the length of the string including its zero terminator.
Y	4	Print 4 bytes in date format (see *time*(II)).
i	*n*	Print as PDP-11 instructions; *n* is the number of bytes occupied by the instruction. This style of printing causes variables 1 and 2 to be set to the offset parts of the source and destination respectively.
a	0	Print the value of *dot* in symbolic form. Symbols are checked to ensure that they have an appropriate type as indicated below.

/	local or global data symbol
?	local or global text symbol
=	local or global absolute symbol

p	2	Print the addressed value in symbolic form using the same rules for symbol lookup as **a**.
t	0	When preceded by an integer, tabs to the next appropriate tab stop. For example, **8t** moves to the next 8 space tab stop.
r	0	Print a space.
n	0	Print a newline.
"..."	0	Print the enclosed string.
ˆ		*dot* is decremented by the current increment. Nothing is printed.
+		*dot* is incremented by 1. Nothing is printed.
−		*dot* is decremented by 1. Nothing is printed.

MORE COMMANDS

Here are a few more commands; '[?/]' means the command can start with either '?', for addresses in *objfil*, or '/', for addresses in *corfil.*

[**?/**] **l** *value mask*

Words starting at *dot* are masked with *mask* and compared with *value* until a match is found. If **L** is used, then the match is for 4 bytes at a time instead of 2. If no match is found, then *dot* is unchanged; otherwise *dot* is set to the matched location. If *mask* is omitted, then −1 is used.

[**?/**] **w** *value* **. . .**

value is written into the addressed location. If **W** is used then 4 bytes are written, otherwise 2 bytes are written. Odd addresses are not allowed when writing to the sub-process address space.

[**?/**] **m** *b1 e1 f1* [**?/**]

New values for (*b1* , *e1* , *f1*) are recorded. If less than three expressions are given then the remaining map parameters are left unchanged. If the '?' or '/' is followed by '∗' then the second segment (*b2* , *e2* , *f2*) of the mapping is changed. If the list is terminated by '?' or '/' then the file (*objfil* or *corfil* respectively) is used for subsequent requests. (So that, for example, '/m?' will cause '/' to refer to *objfil*.)

> *name*   *dot* is assigned to the variable or register named.

!   A shell is called to read the rest of the line following '!'.

$ *modifier*

- **<** *f*   Read commands from the file *f* and return.
- **>** *f*   Send output to the file *f* which is created if it does not exist.
- **r**   Print the general registers and the instruction addressed by **pc**; *dot* is set to **pc**.
- **f**   Print the floating registers in single or double length. If the floating point status of **ps** is set to double (0200 bit) then double length is used anyway.
- **b**   Print all breakpoints and their associated counts and commands.
- **a**   ALGOL 68 stack backtrace. If *address* is given then it is taken to be the address of the current frame (instead of **r4**). If*count* is gi ven then only the first *count* frames are printed.
- **c**   C stack backtrace. If *address* is given then it is taken as the address of the current frame (instead of **r5**). If **C** is used then the names and (16-bit) values of all automatic and static variables are printed for each active function. If*count* is gi ven then only the first *count* frames are printed.
- **e**   The names and values of external variables are printed.
- **w**   Set the page width for output to *address* (default 80).
- **s**   Set the limit for symbol matches to *address* (default 255).
- **o**   All integers input are regarded as octal.
- **d**   Reset integer input as described in **EXPRESSIONS**.
- **q**   Exit from *adb*.
- **v**   Print all non-zero variables in octal.
- **m**   The values used for mapping addresses into file addresses are printed.

: *modifier*

- **b** *c*   Set breakpoint at *address*. The breakpoint is executed *c−1* times before causing a stop. Each time the breakpoint is encountered, the command *c* is executed. If this command sets*dot* to zero then the breakpoint causes a stop.
- **d**   Delete breakpoint at *address*.
- **r** *c*   Run *objfil* as a sub-process. If *address* is given explicitly, then the program is entered at this point; otherwise, the program is entered at its standard entry point; *c* specifies how many breakpoints are to be ignored before stopping. Arguments to the sub-process may be supplied on the same line as the command. An argument starting with < or > causes the standard input or output to be established for the command. All signals are turned on on entry to the sub-process.
- **c** *s*   The sub-process is continued with signal *s*. If*addr ess* is given then the sub-process is continued at this address. If no signal is specified then the signal that caused the sub-process to stop is sent. Breakpoint skipping is the same as for **r**.
- **s** *s*   As for **c** except that the sub-process is single stepped *count* times. If there is no current sub-process then *objfil* is run as a sub-process as for **r**. In this case no signal can be sent; the remainder of the line is treated as arguments to the sub-process.
- **k**   The current sub-process, if any, is terminated.

VARIABLES

Adb provides a number of variables. Named variables are set initially by *adb* but are not used subsequently. Numbered variables are reserved for communication as follows.

0	The last value printed.
1	The last offset part of an instruction source.
2	The previous value of variable 1.

On entry the following are set from the system header in the *corfil*. If*corfil* does not appear to be a **cor e** file then these values are set from *objfil*.

b	The base address of the data segment.
d	The data segment size.
e	The entry point.
m	The ‘magic’ number (0405, 0407, 0410 or 0411).
s	The stack segment size.
t	The text segment size.

ADDRESSES

The address in a file associated with a written address is determined by a mapping associated with that file. Each mapping is represented by two triples (*b1* , *e1* , *f1*) and (*b2* , *e2* , *f2*) and the *file address* corresponding to a written *address* is calculated as follows.

$$b1 \le address < e1 \Rightarrow file\ address = address + f1 - b1, \text{ otherwise,}$$

$$b2 \le address < e2 \Rightarrow file\ address = address + f2 - b2,$$

otherwise, the requested *address* is not legal. In some cases (e.g. for programs with separated I and D space) the two segments for a file may overlap. If a ‘?’ or ‘/’ is followed by an ‘∗’ then only the second triple is used.

The initial setting of both mappings is suitable for normal **a.out** and **core** files. If either file is not of the kind expected then, for that file, *b1* is set to 0, *e1* is set to the maximum file size and *f1* is set to 0; in this way the whole file can be examined with no address translation.

So that *adb* may be used on large files all appropriate values are kept as signed 32-bit integers.

EXIT STATUS

If the last command was successful then the exit status is zero; otherwise it is non-zero.

FILES

/dev/mem
/dev/swap

SEE ALSO

cdb(I), db(I), ptrace(II), a.out(V), core(V)

BUGS

a) A breakpoint set at the entry point is not effective on initial entry to the program.

b) When single stepping, system calls do not count as an executed instruction.