-

NAME

ptrace – process trace

SYNOPSIS

(ptrace = 26.; not in assembler)
(data in r0)
sys ptrace; pid; addr; request
(value in r0)

ptrace(request, pid, addr, data);

DESCRIPTION

Ptrace provides a means by which a parent process may control the execution of a child process, and examine and change its core image. Its primary use is for the implementation of breakpoint debugging, but it should be adaptable for simulation of non-UNIX environments. There are four arguments whose interpretation depends on a *request* argument. Generally, *pid* is the process ID of the traced process, which must be a child (no more distant descendant) of the tracing process. A process being traced behaves normally until it encounters some signal whether internally generated like ‘‘illegal instruction’’ or externally generated like ‘‘interrupt.’’ See signal (II) for the list. Then the traced process enters a stopped state and its parent is notified via *wait* (II). When the child is in the stopped state, its core image can be examined and modified using *ptrace.* If desired, another *ptrace* request can then cause the child either to terminate or to continue, possibly ignoring the signal.

The value of the *request* argument determines the precise action of the call:

0 This request is the only one used by the child process; it declares that the process is to be traced by its parent. All the other arguments are ignored. Peculiar results will ensue if the parent does not expect to trace the child.

1,2 The word in the child process’s address space at *addr* is returned (in r0). Request 1 indicates the data space (normally used); 2 indicates the instruction space (when I and D space are separated). *addr* must be even. The child must be stopped. The input *data* is ignored.

3 The word of the system’s per-process data area corresponding to *addr* is returned. *Addr* must be even and less than 512. This space contains the registers and other information about the process; its layout corresponds to the *user* structure in the system.

4,5 The given *data* is written at the word in the process’s address space corresponding to *addr,* which must be even. No useful value is returned. Request 4 specifies data space (normally used), 5 specifies instruction space. Attempts to write in pure procedure result in termination of the child, instead of going through or causing an error for the parent.

6 The process’s system data is written, as it is read with request 3. Only a few locations can be written in this way: the general registers, the floating point status and registers, and certain bits of the processor status word.

7 The *data* argument is taken as a signal number and the child’s execution continues as if it had incurred that signal. Normally the signal number will be either 0 to indicate that the signal which caused the stop should be ignored, or that value fetched out of the process’s image indicating which signal caused the stop.

8 The traced process terminates.

As indicated, these calls (except for request 0) can be used only when the subject process has stopped. The *wait* call is used to determine when a process stops; in such a case the ‘‘termination’’ status returned by *wait* has the value 0177 to indicate stoppage rather than genuine termination.

To forestall possible fraud, *ptrace* inhibits the set-user-id facility on subsequent *exec* (II)
calls.

SEE ALSO

wait (II), signal (II), cdb (I)

DIAGNOSTICS

From assembler, the c-bit (error bit) is set on errors; from C, −1 is returned and *errno* has the error code.

BUGS

The request 0 call should be able to specify signals which are to be treated normally and not cause a stop. In this way, for example, programs with simulated floating point (which use ‘‘illegal instruction’’ signals at a very high rate) could be efficiently debugged.

Also, it should be possible to stop a process on occurrence of a system call; in this way a completely controlled environment could be provided.