NAME

ptrace − process trace

SYNOPSIS

#include <sys/types.h>
#include <sys/signal.h>
#include <sys/ptrace.h>

int ptrace(int *request***, pid_t** *pid***, long** *addr***, long** *data***)**

DESCRIPTION

Note: This manual page has no relation to Minix. Someone who knows ptrace() has to check, or rewrite, this page. (kjb)

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. 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 **sigaction**(2) for the list. Then the traced process enters a stopped state and its parent is notified via **wait**(2). 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:

PT_TRACE_ME
: 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.

PT_READ_I, PT_READ_D
: The word in the child process's address space at *addr* is returned. If I and D space are separated (e.g. historically on a pdp-11), request PT_READ_I indicates I space, PT_READ_D D space. *Addr* must be even on some machines. The child must be stopped. The input *data* is ignored.

PT_READ_U
: The word of the system's per-process data area corresponding to *addr* is returned. *Addr* must be even on some machines 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.

PT_WRITE_I, PT_WRITE_D
: The given *data* is written at the word in the process's address space corresponding to *addr*, which must be even on some machines. No useful value is returned. If I and D space are separated, request PT_WRITE_I indicates I space, PT_WRITE_D D space. Attempts to write in pure procedure fail if another process is executing the same file.

PT_WRITE_U
: The process's system data is written, as it is read with request PT_READ_U. 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.

PT_CONTINUE
: The *data* argument is taken as a signal number and the child's execution continues at location *addr* as if it had incurred that signal. Normally the signal number will be either 0 to indicate that the signal that caused the stop should be ignored, or that value fetched out of the process's image indicating which signal caused the stop. If *addr* is (int *)1 then execution continues from where it stopped.

PT_KILL
: The traced process terminates.

PT_STEP
Execution continues as in request PT_CONTINUE; however, as soon as possible after execution of at least one instruction, execution stops again. The signal number from the stop is SIGTRAP. (On the VAX-11 the T-bit is used and just one instruction is executed.) This is part of the mechanism for implementing breakpoints.

As indicated, these calls (except for request PT_TRACE_ME) 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 and set-group-id facilities on subsequent **execve**(2) calls. If a traced process calls **execve**, it will stop before executing the first instruction of the new image showing signal SIGTRAP.

On a VAX-11, word also means a 32-bit integer, but the even restriction does not apply.

RETURN VALUE

A 0 value is returned if the call succeeds. If the call fails then a −1 is returned and the global variable *errno* is set to indicate the error.

ERRORS

[EIO]	The request code is invalid.
[ESRCH]	The specified process does not exist.
[EIO]	The given signal number is invalid.
[EIO]	The specified address is out of bounds.
[EPERM]	The specified process cannot be traced.

SEE ALSO

wait(2), **sigaction**(2), **mdb**(1).

BUGS

Ptrace is unique and arcane; it should be replaced with a special file that can be opened and read and written. The control functions could then be implemented with **ioctl**(2) calls on this file. This would be simpler to understand and have much higher performance.

The request PT_TRACE_ME call should be able to specify signals that 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.

The error indication, −1, is a legitimate function value; **errno**, (see **intro**(2)), can be used to disambiguate.

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