NAME

ptrace – allows a parent process to control the execution of a child process

SYNOPSIS

#include <unistd.h>
#include <sys/types.h>

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

DESCRIPTION

ptrace( ) allows a parent process to control the execution of a child process. Its primary use is for the implementation of breakpoint debugging. The child process behaves normally until it encounters a signal (see **signal**(5)), at which time it enters a stopped state and its parent is notified via the **wait**(2) function. When the child is in the stopped state, its parent can examine and modify its "core image" using **ptrace( )**. Also, the parent can cause the child either to terminate or continue, with the possibility of ignoring the signal that caused it to stop.

The *request* argument determines the action to be taken by **ptrace( )** and is one of the following:

0 This request must be issued by the child process if it is to be traced by its parent. It turns on the child's trace flag that stipulates that the child should be left in a stopped state on receipt of a signal rather than the state specified by *func* (see **signal**(3C)). The *pid*, *addr*, and *data* arguments are ignored, and a return value is not defined for this request. Peculiar results ensue if the parent does not expect to trace the child.

The remainder of the requests can only be used by the parent process. For each, *pid* is the process ID of the child. The child must be in a stopped state before these requests are made.

1, 2 With these requests, the word at location *addr* in the address space of the child is returned to the parent process. If instruction and data space are separated, request **1** returns a word from instruction space, and request **2** returns a word from data space. If instruction and data space are not separated, either request **1** or request **2** may be used with equal results. The *data* argument is ignored. These two requests fail if *addr* is not the start address of a word, in which case a value of –1 is returned to the parent process and the parent's **errno** is set to **EIO.**

3 With this request, the word at location *addr* in the child's user area in the system's address space (see **<sys/user.h>**) is returned to the parent process. The *data* argument is ignored. This request fails if *addr* is not the start address of a word or is outside the user area, in which case a value of –1 is returned to the parent process and the parent's **errno** is set to **EIO.**

4, 5 With these requests, the value given by the *data* argument is written into the address space of the child at location *addr*. If instruction and data space are separated, request **4** writes a word into instruction space, and request **5** writes a word into data space. If instruction and data space are not separated, either request **4** or request **5** may be used with equal results. On success, the value written into the address space of the child is returned to the parent. These two requests fail if *addr* is not the start address of a word. On failure a value of −1 is returned to the parent process and the parent's **errno** is set to **EIO.**

6 With this request, a few entries in the child's user area can be written. *data* gives the value that is to be written and *addr* is the location of the entry. The few entries that can be written are the general registers and the condition codes of the Processor Status Word.

7 This request causes the child to resume execution. If the *data* argument is 0, all pending signals including the one that caused the child to stop are canceled before it resumes execution. If the *data* argument is a valid signal number, the child resumes execution as if it had incurred that signal, and any other pending signals are canceled. The *addr* argument must be equal to 1 for this request. On success, the value of *data* is returned to the parent. This request fails if *data* is not 0 or a valid signal number, in which case a value of −1 is returned to the parent process and the parent's **errno** is set to **EIO.**

8 This request causes the child to terminate with the same consequences as **exit**(2).

9 This request sets the trace bit in the Processor Status Word of the child and then executes the same steps as listed above for request **7**. The trace bit causes an interrupt on completion of one machine instruction. This effectively allows single stepping of the child.

To forestall possible fraud, **ptrace( )** inhibits the set-user-ID facility on subsequent **exec**(2) calls. If a traced process calls **exec**(2), it stops before executing the first instruction of the new image showing signal **SIGTRAP**.

ERRORS

ptrace( ) in general fails if one or more of the following are true:

EIO *request is an illegal number.*

EPERM The effective user of the calling process is not super-user.

ESRCH *pid* identifies a child that does not exist or has not executed a **ptrace( )** with request **0**.

SEE ALSO

exec(2), **exit**(2), **wait**(2), **signal**(3C), **signal**(5)