NAME

process_vm_readv, process_vm_writev − transfer data between process address spaces

LIBRARY

Standard C library (*libc*, *−lc*)

SYNOPSIS

```
#include <sys/uio.h>

ssize_t process_vm_readv(pid_t pid,
                         const struct iovec *local_iov,
                         unsigned long liovcnt,
                         const struct iovec *remote_iov,
                         unsigned long riovcnt,
                         unsigned long flags);
ssize_t process_vm_writev(pid_t pid,
                         const struct iovec *local_iov,
                         unsigned long liovcnt,
                         const struct iovec *remote_iov,
                         unsigned long riovcnt,
                         unsigned long flags);
```

Feature Test Macro Requirements for glibc (see **feature_test_macros**(7)):

process_vm_readv(), **process_vm_writev**():
_GNU_SOURCE

DESCRIPTION

These system calls transfer data between the address space of the calling process ("the local process") and the process identified by *pid* ("the remote process"). The data moves directly between the address spaces of the two processes, without passing through kernel space.

The **process_vm_readv**() system call transfers data from the remote process to the local process. The data to be transferred is identified by *remote_iov* and *riovcnt*: *remote_iov* is a pointer to an array describing address ranges in the process *pid*, and *riovcnt* specifies the number of elements in *remote_iov*. The data is transferred to the locations specified by *local_iov* and *liovcnt*: *local_iov* is a pointer to an array describing address ranges in the calling process, and *liovcnt* specifies the number of elements in *local_iov*.

The **process_vm_writev**() system call is the converse of **process_vm_readv**()—it transfers data from the local process to the remote process. Other than the direction of the transfer, the arguments *liovcnt*, *local_iov*, *riovcnt*, and *remote_iov* have the same meaning as for **process_vm_readv**().

The *local_iov* and *remote_iov* arguments point to an array of *iovec* structures, described in **iovec**(3type).

Buffers are processed in array order. This means that**pr ocess_vm_readv**() completely fills *local_iov[0]* before proceeding to *local_iov[1]*, and so on. Likewise, *remote_iov[0]* is completely read before proceeding to *remote_iov[1]*, and so on.

Similarly, **process_vm_writev**() writes out the entire contents of *local_iov[0]* before proceeding to *local_iov[1]*, and it completely fills *remote_iov[0]* before proceeding to *remote_iov[1]*.

The lengths of *remote_iov[i].iov_len* and *local_iov[i].iov_len* do not have to be the same. Thus, it is possible to split a single local buffer into multiple remote buffers, or vice versa.

The *flags* argument is currently unused and must be set to 0.

The values specified in the *liovcnt* and *riovcnt* arguments must be less than or equal to **IOV_MAX** (defined in <*limits.h*> or accessible via the call *sysconf(_SC_IOV_MAX)*).

The count arguments and *local_iov* are checked before doing any transfers. If the counts are too big, or *local_iov* is invalid, or the addresses refer to regions that are inaccessible to the local process, none of the vectors will be processed and an error will be returned immediately.

Note, however, that these system calls do not check the memory regions in the remote process until just

before doing the read/write. Consequently, a partial read/write (see RETURN VALUE) may result if one of the *remote_iov* elements points to an invalid memory region in the remote process. No further reads/writes will be attempted beyond that point. Keep this in mind when attempting to read data of unknown length (such as C strings that are null-terminated) from a remote process, by avoiding spanning memory pages (typically 4 KiB) in a single remote *iovec* element. (Instead, split the remote read into two *remote_iov* elements and have them merge back into a single write *local_iov* entry. The first read entry goes up to the page boundary, while the second starts on the next page boundary.)

Permission to read from or write to another process is governed by a ptrace access mode **PTRACE_MODE_ATTACH_REALCREDS** check; see **ptrace**(2).

RETURN VALUE

On success, **process_vm_readv**() returns the number of bytes read and **process_vm_writev**() returns the number of bytes written. This return value may be less than the total number of requested bytes, if a partial read/write occurred. (Partial transfers apply at the granularity of *iovec* elements. These system calls won't perform a partial transfer that splits a single *iovec* element.) The caller should check the return value to determine whether a partial read/write occurred.

On error, −1 is returned and *errno* is set to indicate the error.

ERRORS

EFAULT
The memory described by *local_iov* is outside the caller's accessible address space.

EFAULT
The memory described by *remote_iov* is outside the accessible address space of the process *pid*.

EINVAL
The sum of the *iov_len* values of either *local_iov* or *remote_iov* overflows a *ssize_t* value.

EINVAL
flags is not 0.

EINVAL
liovcnt or *riovcnt* is too large.

ENOMEM
Could not allocate memory for internal copies of the *iovec* structures.

EPERM
The caller does not have permission to access the address space of the process *pid*.

ESRCH
No process with ID *pid* exists.

VERSIONS

These system calls were added in Linux 3.2. Support is provided since glibc 2.15.

STANDARDS

These system calls are nonstandard Linux extensions.

NOTES

The data transfers performed by **process_vm_readv**() and **process_vm_writev**() are not guaranteed to be atomic in any way.

These system calls were designed to permit fast message passing by allowing messages to be exchanged with a single copy operation (rather than the double copy that would be required when using, for example, shared memory or pipes).

EXAMPLES

The following code sample demonstrates the use of **process_vm_readv**(). It reads 20 bytes at the address 0x10000 from the process with PID 10 and writes the first 10 bytes into *buf1* and the second 10 bytes into *buf2*.

```
#define _GNU_SOURCE
```

```
#include <stdlib.h>
#include <sys/types.h>
#include <sys/uio.h>

int
main(void)
{
    char          buf1[10];
    char          buf2[10];
    pid_t         pid = 10;    /* PID of remote process */
    ssize_t       nread;
    struct iovec  local[2];
    struct iovec  remote[1];

    local[0].iov_base = buf1;
    local[0].iov_len = 10;
    local[1].iov_base = buf2;
    local[1].iov_len = 10;
    remote[0].iov_base = (void *) 0x10000;
    remote[0].iov_len = 20;

    nread = process_vm_readv(pid, local, 2, remote, 1, 0);
    if (nread != 20)
        exit(EXIT_FAILURE);

    exit(EXIT_SUCCESS);
}
```

SEE ALSO

readv(2), **writev**(2)