Rocky Enterprise Linux 9.2 Manual Pages on command 'process_vm_writev.2'

$ man process_vm_writev.2

PROCESS_VM_READV(2) Linux Programmer's Manual PROCESS_VM_READV(2)

NAME

process_vm_readv, process_vm_writev - transfer data between process ad?

dress spaces

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 call?ing 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 describ?ing 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 re?mote 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, defined in <sys/uio.h> as:

```
struct iovec {
    void  *iov_base;    /* Starting address */
    size_t iov_len;     /* Number of bytes to transfer */
};
```

Buffers are processed in array order. This means that process_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 lo?cal_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 buf? fer 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 trans? fers. If the counts are too big, or local_iov is invalid, or the ad? dresses refer to regions that are inaccessible to the local process, none of the vectors will be processed and an error will be returned im? mediately.

Note, however, that these system calls do not check the memory regions in the remote process until just before doing the read/write. Conse? quently, a partial read/write (see RETURN VALUE) may result if one of the remote_iov elements points to an invalid memory region in the re? mote 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 sin? gle remote iovec element. (Instead, split the remote read into two re? mote_iov elements and have them merge back into a single write lo? cal_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 par? tial 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 re?

turn value to determine whether a partial read/write occurred.

On error, -1 is returned and errno is set appropriately.

ERRORS

EFAULT The memory described by local_iov is outside the caller's acces?

sible address space.

EFAULT The memory described by remote_iov is outside the accessible ad?

dress 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 in

glibc since version 2.15.

CONFORMING TO

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 al?

lowing 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.

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

int
main(void)
{
    struct iovec local[2];
    struct iovec remote[1];
    char buf1[10];
    char buf2[10];
    ssize_t nread;
    pid_t pid = 10;             /* PID of remote process */
    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)
        return 1;
    else
        return 0;
}
```

SEE ALSO

readv(2), writev(2)

COLOPHON

This page is part of release 5.10 of the Linux man-pages project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at https://www.kernel.org/doc/man-pages/.