NAME

recvmmsg − receive multiple messages on a socket

LIBRARY

Standard C library (*libc*, *−lc*)

SYNOPSIS

#define _GNU_SOURCE /* See feature_test_macros(7) */
#include <sys/socket.h>

int recvmmsg(int *sockfd***, struct mmsghdr ****msgvec***, unsigned int** *vlen***,**
int *flags***, struct timespec ****timeout***);**

DESCRIPTION

The **recvmmsg**() system call is an extension of **recvmsg**(2) that allows the caller to receive multiple messages from a socket using a single system call. (This has performance benefits for some applications.) A further extension over **recvmsg**(2) is support for a timeout on the receive operation.

The *sockfd* argument is the file descriptor of the socket to receive data from.

The *msgvec* argument is a pointer to an array of *mmsghdr* structures. The size of this array is specified in *vlen*.

The *mmsghdr* structure is defined in <*sys/socket.h*> as:

```
struct mmsghdr {
    struct msghdr msg_hdr;  /* Message header */
    unsigned int  msg_len;  /* Number of received bytes for header */
};
```

The *msg_hdr* field is a *msghdr* structure, as described in **recvmsg**(2). The*msg_len* field is the number of bytes returned for the message in the entry. This field has the same value as the return value of a single **recvmsg**(2) on the header.

The *flags* argument contains flags ORed together. The flags are the same as documented for**r ecvmsg**(2), with the following addition:

MSG_WAITFORONE (since Linux 2.6.34)
Turns on **MSG_DONTWAIT** after the first message has been received.

The *timeout* argument points to a *struct timespec* (see **clock_gettime**(2)) defining a timeout (seconds plus nanoseconds) for the receive operation (*but see BUGS!*). (This interval will be rounded up to the system clock granularity, and kernel scheduling delays mean that the blocking interval may overrun by a small amount.) If*timeout* is NULL, then the operation blocks indefinitely .

A blocking **recvmmsg**() call blocks until *vlen* messages have been received or until the timeout expires. A nonblocking call reads as many messages as are available (up to the limit specified by *vlen*) and returns immediately.

On return from **recvmmsg**(), successive elements of *msgvec* are updated to contain information about each received message: *msg_len* contains the size of the received message; the subfields of *msg_hdr* are updated as described in **recvmsg**(2). The return value of the call indicates the number of elements of *msgvec* that have been updated.

RETURN VALUE

On success, **recvmmsg**() returns the number of messages received in *msgvec*; on error, −1 is returned, and *errno* is set to indicate the error.

ERRORS

Errors are as for **recvmsg**(2). In addition, the following error can occur:

EINVAL
timeout is invalid.

See also BUGS.

VERSIONS

The **recvmmsg**() system call was added in Linux 2.6.33. Support in glibc was added in glibc 2.12.

STANDARDS

recvmmsg() is Linux-specific.

BUGS

The *timeout* argument does not work as intended. The timeout is checked only after the receipt of each datagram, so that if up to *vlen−1* datagrams are received before the timeout expires, but then no further datagrams are received, the call will block forever.

If an error occurs after at least one message has been received, the call succeeds, and returns the number of messages received. The error code is expected to be returned on a subsequent call to **recvmmsg**(). In the current implementation, however, the error code can be overwritten in the meantime by an unrelated network event on a socket, for example an incoming ICMP packet.

EXAMPLES

The following program uses **recvmmsg**() to receive multiple messages on a socket and stores them in multiple buffers. The call returns if all buffers are filled or if the timeout specified has expired.

The following snippet periodically generates UDP datagrams containing a random number:

```
$ while true; do echo $RANDOM > /dev/udp/127.0.0.1/1234;
      sleep 0.25; done
```

These datagrams are read by the example application, which can give the following output:

```
$ ./a.out
5 messages received
1 11782
2 11345
3 304
4 13514
5 28421
```

Program source

```
#define _GNU_SOURCE
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <time.h>

int
main(void)
{
#define VLEN 10
#define BUFSIZE 200
#define TIMEOUT 1
    int                 sockfd, retval;
    char                bufs[VLEN][BUFSIZE+1];
    struct iovec        iovecs[VLEN];
    struct mmsghdr      msgs[VLEN];
    struct timespec     timeout;
    struct sockaddr_in  addr;

    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
```

```
        if (sockfd == -1) {
            perror("socket()");
            exit(EXIT_FAILURE);
        }

        addr.sin_family = AF_INET;
        addr.sin_addr.s_addr = htonl(INADDR_LOOPBACK);
        addr.sin_port = htons(1234);
        if (bind(sockfd, (struct sockaddr *) &addr, sizeof(addr)) == -1) {
            perror("bind()");
            exit(EXIT_FAILURE);
        }

        memset(msgs, 0, sizeof(msgs));
        for (size_t i = 0; i < VLEN; i++) {
            iovecs[i].iov_base         = bufs[i];
            iovecs[i].iov_len          = BUFSIZE;
            msgs[i].msg_hdr.msg_iov    = &iovecs[i];
            msgs[i].msg_hdr.msg_iovlen = 1;
        }

        timeout.tv_sec = TIMEOUT;
        timeout.tv_nsec = 0;

        retval = recvmmsg(sockfd, msgs, VLEN, 0, &timeout);
        if (retval == -1) {
            perror("recvmmsg()");
            exit(EXIT_FAILURE);
        }

        printf("%d messages received\n", retval);
        for (size_t i = 0; i < retval; i++) {
            bufs[i][msgs[i].msg_len] = 0;
            printf("%zu %s", i+1, bufs[i]);
        }
        exit(EXIT_SUCCESS);
    }
```

SEE ALSO

clock_gettime(2), **recvmsg**(2), **sendmmsg**(2), **sendmsg**(2), **socket**(2), **socket**(7)