



*Full credit is given to the above companies including the OS that this PDF file was generated!*

## **Red Hat Enterprise Linux Release 9.2 Manual Pages on 'msgsnd.2' command**

**\$ man msgsnd.2**

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

NAME

msgrcv, msgsnd - System V message queue operations

SYNOPSIS

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);
ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp,
               int msgflg);
```

DESCRIPTION

The msgsnd() and msgrcv() system calls are used to send messages to, and receive messages from, a System V message queue. The calling process must have write permission on the message queue in order to send a message, and read permission to receive a message.

The msgp argument is a pointer to a caller-defined structure of the following general form:

```
struct msgbuf {
    long mtype;     /* message type, must be > 0 */
    char mtext[1]; /* message data */
};
```

The mtext field is an array (or other structure) whose size is speci-

fied by msgsz, a nonnegative integer value. Messages of zero length

(i.e., no mtext field) are permitted. The mtype field must have a strictly positive integer value. This value can be used by the receiving process for message selection (see the description of msgrcv() below).

## msgsnd()

The msgsnd() system call appends a copy of the message pointed to by msgp to the message queue whose identifier is specified by msqid.

If sufficient space is available in the queue, msgsnd() succeeds immediately. The queue capacity is governed by the msg\_qbytes field in the associated data structure for the message queue. During queue creation this field is initialized to MSGMNB bytes, but this limit can be modified using msgctl(2). A message queue is considered to be full if either of the following conditions is true:

? Adding a new message to the queue would cause the total number of bytes in the queue to exceed the queue's maximum size (the msg\_qbytes field).

? Adding another message to the queue would cause the total number of messages in the queue to exceed the queue's maximum size (the msg\_qbytes field). This check is necessary to prevent an unlimited number of zero-length messages being placed on the queue. Although such messages contain no data, they nevertheless consume (locked) kernel memory.

If insufficient space is available in the queue, then the default behavior of msgsnd() is to block until space becomes available. If IPC\_NOWAIT is specified in msgflg, then the call instead fails with the error EAGAIN.

A blocked msgsnd() call may also fail if:

? the queue is removed, in which case the system call fails with errno set to EIDRM; or

? a signal is caught, in which case the system call fails with errno set to EINTR; see signal(7). (msgsnd() is never automatically restarted after being interrupted by a signal handler, regardless of the setting of the SA\_RESTART flag when establishing a signal handler.)

der.)

Upon successful completion the message queue data structure is updated as follows:

? msg\_lspid is set to the process ID of the calling process.

? msg\_qnum is incremented by 1.

? msg\_stime is set to the current time.

msgrcv()

The msgrcv() system call removes a message from the queue specified by msqid and places it in the buffer pointed to by msgp.

The argument msgsz specifies the maximum size in bytes for the member mtext of the structure pointed to by the msgp argument. If the message text has length greater than msgsz, then the behavior depends on whether MSG\_NOERROR is specified in msgflg. If MSG\_NOERROR is specified, then the message text will be truncated (and the truncated part will be lost); if MSG\_NOERROR is not specified, then the message isn't removed from the queue and the system call fails returning -1 with errno set to E2BIG.

Unless MSG\_COPY is specified in msgflg (see below), the msgtyp argument specifies the type of message requested, as follows:

? If msgtyp is 0, then the first message in the queue is read.

? If msgtyp is greater than 0, then the first message in the queue of type msgtyp is read, unless MSG\_EXCEPT was specified in msgflg, in which case the first message in the queue of type not equal to msgtyp will be read.

? If msgtyp is less than 0, then the first message in the queue with the lowest type less than or equal to the absolute value of msgtyp will be read.

The msgflg argument is a bit mask constructed by ORing together zero or more of the following flags:

IPC\_NOWAIT

Return immediately if no message of the requested type is in the queue. The system call fails with errno set to ENOMSG.

MSG\_COPY (since Linux 3.8)

Nondestructively fetch a copy of the message at the ordinal position in the queue specified by msgtyp (messages are considered to be numbered starting at 0).

This flag must be specified in conjunction with IPC\_NOWAIT, with the result that, if there is no message available at the given position, the call fails immediately with the error ENOMSG. Because they alter the meaning of msgtyp in orthogonal ways, MSG\_COPY and MSG\_EXCEPT may not both be specified in msgflg. The MSG\_COPY flag was added for the implementation of the kernel checkpoint-restore facility and is available only if the kernel was built with the CONFIG\_CHECKPOINT\_RESTORE option.

#### MSG\_EXCEPT

Used with msgtyp greater than 0 to read the first message in the queue with message type that differs from msgtyp.

#### MSG\_NOERROR

To truncate the message text if longer than msgsz bytes.

If no message of the requested type is available and IPC\_NOWAIT isn't specified in msgflg, the calling process is blocked until one of the following conditions occurs:

- ? A message of the desired type is placed in the queue.
- ? The message queue is removed from the system. In this case, the system call fails with errno set to EIDRM.
- ? The calling process catches a signal. In this case, the system call fails with errno set to EINTR. (msgrcv() is never automatically restarted after being interrupted by a signal handler, regardless of the setting of the SA\_RESTART flag when establishing a signal handler.)

Upon successful completion the message queue data structure is updated as follows:

- msg\_lrpid is set to the process ID of the calling process.
- msg\_qnum is decremented by 1.
- msg\_rtime is set to the current time.

On failure both functions return -1 with `errno` indicating the error, otherwise `msgsnd()` returns 0 and `msgrcv()` returns the number of bytes actually copied into the `mtext` array.

## ERRORS

When `msgsnd()` fails, `errno` will be set to one among the following values:

ues:

**EACCES** The calling process does not have write permission on the message queue, and does not have the `CAP_IPC_OWNER` capability in the user namespace that governs its IPC namespace.

**EAGAIN** The message can't be sent due to the `msg_qbytes` limit for the queue and `IPC_NOWAIT` was specified in `msgflg`.

**EFAULT** The address pointed to by `msgp` isn't accessible.

**EIDRM** The message queue was removed.

**EINTR** Sleeping on a full message queue condition, the process caught a signal.

**EINVAL** Invalid `msqid` value, or nonpositive `mtype` value, or invalid `msgsz` value (less than 0 or greater than the system value `MSGMAX`).

**ENOMEM** The system does not have enough memory to make a copy of the message pointed to by `msgp`.

When `msgrcv()` fails, `errno` will be set to one among the following values:

ues:

**E2BIG** The message text length is greater than `msgsz` and `MSG_NOERROR` isn't specified in `msgflg`.

**EACCES** The calling process does not have read permission on the message queue, and does not have the `CAP_IPC_OWNER` capability in the user namespace that governs its IPC namespace.

**EFAULT** The address pointed to by `msgp` isn't accessible.

**EIDRM** While the process was sleeping to receive a message, the message queue was removed.

**EINTR** While the process was sleeping to receive a message, the process caught a signal; see `signal(7)`.

**EINVAL** `msqid` was invalid, or `msgsz` was less than 0.

**EINVAL** (since Linux 3.14)

msgflg specified MSG\_COPY, but not IPC\_NOWAIT.

EINVAL (since Linux 3.14)

msgflg specified both MSG\_COPY and MSG\_EXCEPT.

ENOMSG IPC\_NOWAIT was specified in msgflg and no message of the re?

quested type existed on the message queue.

ENOMSG IPC\_NOWAIT and MSG\_COPY were specified in msgflg and the queue

contains less than msgtyp messages.

ENOSYS (since Linux 3.8)

Both MSG\_COPY and IPC\_NOWAIT were specified in msgflg, and this

kernel was configured without CONFIG\_CHECKPOINT\_RESTORE.

## CONFORMING TO

POSIX.1-2001, POSIX.1-2008, SVr4.

The MSG\_EXCEPT and MSG\_COPY flags are Linux-specific; their definitions

can be obtained by defining the \_GNU\_SOURCE feature test macro.

## NOTES

The inclusion of <sys/types.h> and <sys/ipc.h> isn't required on Linux or by any version of POSIX. However, some old implementations required the inclusion of these header files, and the SVID also documented their inclusion. Applications intended to be portable to such old systems may need to include these header files.

The msgp argument is declared as struct msgbuf \* in glibc 2.0 and 2.1.

It is declared as void \* in glibc 2.2 and later, as required by SUSv2 and SUSv3.

The following limits on message queue resources affect the msgsnd() call:

**MSGMAX** Maximum size of a message text, in bytes (default value: 8192 bytes). On Linux, this limit can be read and modified via /proc/sys/kernel/msgmax.

**MSGMNB** Maximum number of bytes that can be held in a message queue (default value: 16384 bytes). On Linux, this limit can be read and modified via /proc/sys/kernel/msgmnb. A privileged process (Linux: a process with the CAP\_SYS\_RESOURCE capability) can increase the size of a message queue beyond MSGMNB using the ms?

gctl(2) IPC\_SET operation.

The implementation has no intrinsic system-wide limits on the number of message headers (MSGTQL) and the number of bytes in the message pool (MSGPOOL).

## BUGS

In Linux 3.13 and earlier, if `msgrcv()` was called with the `MSG_COPY` flag, but without `IPC_NOWAIT`, and the message queue contained less than `msgtyp` messages, then the call would block until the next message is written to the queue. At that point, the call would return a copy of the message, regardless of whether that message was at the ordinal position `msgtyp`. This bug is fixed in Linux 3.14.

Specifying both `MSG_COPY` and `MSG_EXCEPT` in `msgflg` is a logical error (since these flags impose different interpretations on `msgtyp`). In Linux 3.13 and earlier, this error was not diagnosed by `msgrcv()`. This bug is fixed in Linux 3.14.

## EXAMPLES

The program below demonstrates the use of `msgsnd()` and `msgrcv()`.

The example program is first run with the `-s` option to send a message and then run again with the `-r` option to receive a message.

The following shell session shows a sample run of the program:

```
$ ./a.out -s
sent: a message at Wed Mar 4 16:25:45 2015
$ ./a.out -r
message received: a message at Wed Mar 4 16:25:45 2015
```

### Program source

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/ipc.h>
```

```

#include <sys/msg.h>

struct msgbuf {
    long mtype;
    char mtext[80];
};

static void
usage(char *prog_name, char *msg)
{
    if (msg != NULL)
        fputs(msg, stderr);

    fprintf(stderr, "Usage: %s [options]\n", prog_name);
    fprintf(stderr, "Options are:\n");
    fprintf(stderr, "-s    send message using msgsnd()\n");
    fprintf(stderr, "-r    read message using msgrcv()\n");
    fprintf(stderr, "-t    message type (default is 1)\n");
    fprintf(stderr, "-k    message queue key (default is 1234)\n");
    exit(EXIT_FAILURE);
}

static void
send_msg(int qid, int msgtype)
{
    struct msgbuf msg;
    time_t t;
    msg.mtype = msgtype;
    time(&t);
    snprintf(msg.mtext, sizeof(msg.mtext), "a message at %s",
             ctime(&t));
    if (msgsnd(qid, &msg, sizeof(msg.mtext),
              IPC_NOWAIT) == -1) {
        perror("msgsnd error");
        exit(EXIT_FAILURE);
    }
    printf("sent: %s\n", msg.mtext);
}

```



```

}

static void
get_msg(int qid, int msgtype)
{
    struct msgbuf msg;
    if (msgrcv(qid, &msg, sizeof(msg.mtext), msgtype,
        MSG_NOERROR | IPC_NOWAIT) == -1) {
        if (errno != ENOMSG) {
            perror("msgrcv");
            exit(EXIT_FAILURE);
        }
        printf("No message available for msgrcv()\n");
    } else
        printf("message received: %s\n", msg.mtext);
}

int
main(int argc, char *argv[])
{
    int qid, opt;
    int mode = 0;          /* 1 = send, 2 = receive */
    int msgtype = 1;
    int msgkey = 1234;
    while ((opt = getopt(argc, argv, "srt:k:")) != -1) {
        switch (opt) {
            case 's':
                mode = 1;
                break;
            case 'r':
                mode = 2;
                break;
            case 't':
                msgtype = atoi(optarg);
                if (msgtype <= 0)

```

```

        usage(argv[0], "-t option must be greater than 0\n");
    break;
case 'k':
    msgkey = atoi(optarg);
    break;
default:
    usage(argv[0], "Unrecognized option\n");
}
}
if (mode == 0)
    usage(argv[0], "must use either -s or -r option\n");
qid = msgget(msgkey, IPC_CREAT | 0666);
if (qid == -1) {
    perror("msgget");
    exit(EXIT_FAILURE);
}
if (mode == 2)
    get_msg(qid, msgtype);
else
    send_msg(qid, msgtype);
exit(EXIT_SUCCESS);
}

```

#### SEE ALSO

msgctl(2), msgget(2), capabilities(7), mq\_overview(7), sysvipc(7)

#### 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/>.