NAME

svipc – System V interprocess communication mechanisms

SYNOPSIS

#include <sys/msg.h>
#include <sys/sem.h>
#include <sys/shm.h>

DESCRIPTION

This manual page refers to the Linux implementation of the System V interprocess communication (IPC) mechanisms: message queues, semaphore sets, and shared memory segments. In the following, the word *resource* means an instantiation of one among such mechanisms.

Resource access permissions

For each resource, the system uses a common structure of type *struct ipc_perm* to store information needed in determining permissions to perform an IPC operation. The *ipc_perm* structure includes the following members:

```
struct ipc_perm {
    uid_t          cuid;   /* creator user ID */
    gid_t          cgid;   /* creator group ID */
    uid_t          uid;    /* owner user ID */
    gid_t          gid;    /* owner group ID */
    unsigned short mode;   /* r/w permissions */
};
```

The *mode* member of the *ipc_perm* structure defines, with its lower 9 bits, the access permissions to the resource for a process executing an IPC system call. The permissions are interpreted as follows:

0400 Read by user.
0200 Write by user.
0040 Read by group.
0020 Write by group.
0004 Read by others.
0002 Write by others.

Bits 0100, 0010, and 0001 (the execute bits) are unused by the system. Furthermore, "write" effectively means "alter" for a semaphore set.

The same system header file also defines the following symbolic constants:

IPC_CREAT Create entry if key doesn't exist.

IPC_EXCL Fail if key exists.

IPC_NOWAIT Error if request must wait.

IPC_PRIVATE Private key.

IPC_RMID Remove resource.

IPC_SET Set resource options.

IPC_STAT Get resource options.

Note that **IPC_PRIVATE** is a *key_t* type, while all the other symbolic constants are flag fields and can be OR'ed into an *int* type variable.

Message queues

A message queue is uniquely identified by a positive integer (its *msqid*) and has an associated data structure of type *struct msqid_ds*, defined in *<sys/msg.h>*, containing the following members:

```
struct msqid_ds {
    struct ipc_perm msg_perm;
    msgqnum_t       msg_qnum;    /* no of messages on queue */
```

```
    msglen_t        msg_qbytes;  /* bytes max on a queue */
    pid_t           msg_lspid;   /* PID of last msgsnd(2) call */
    pid_t           msg_lrpid;   /* PID of last msgrcv(2) call */
    time_t          msg_stime;   /* last msgsnd(2) time */
    time_t          msg_rtime;   /* last msgrcv(2) time */
    time_t          msg_ctime;   /* last change time */
};
```

msg_perm *ipc_perm* structure that specifies the access permissions on the message queue.

msg_qnum Number of messages currently on the message queue.

msg_qbytes Maximum number of bytes of message text allowed on the message queue.

msg_lspid ID of the process that performed the last **msgsnd**(2) system call.

msg_lrpid ID of the process that performed the last **msgrcv**(2) system call.

msg_stime Time of the last **msgsnd**(2) system call.

msg_rtime Time of the last **msgrcv**(2) system call.

msg_ctime Time of the last system call that changed a member of the *msqid_ds* structure.

Semaphore sets

A semaphore set is uniquely identified by a positive integer (its *semid*) and has an associated data structure of type *struct semid_ds*, defined in *<sys/sem.h>*, containing the following members:

```
struct semid_ds {
    struct ipc_perm sem_perm;
    time_t          sem_otime;   /* last operation time */
    time_t          sem_ctime;   /* last change time */
    unsigned long   sem_nsems;   /* count of sems in set */
};
```

sem_perm *ipc_perm* structure that specifies the access permissions on the semaphore set.

sem_otime Time of last **semop**(2) system call.

sem_ctime Time of last **semctl**(2) system call that changed a member of the above structure or of one semaphore belonging to the set.

sem_nsems Number of semaphores in the set. Each semaphore of the set is referenced by a nonnegative integer ranging from **0** to *sem_nsems−1*.

A semaphore is a data structure of type *struct sem* containing the following members:

```
struct sem {
    int semval;  /* semaphore value */
    int sempid;  /* PID of process that last modified */
};
```

semval Semaphore value: a nonnegative integer.

sempid PID of the last process that modified the value of this semaphore.

Shared memory segments

A shared memory segment is uniquely identified by a positive integer (its *shmid*) and has an associated data structure of type *struct shmid_ds*, defined in *<sys/shm.h>*, containing the following members:

```
struct shmid_ds {
    struct ipc_perm shm_perm;
    size_t          shm_segsz;   /* size of segment */
    pid_t           shm_cpid;    /* PID of creator */
    pid_t           shm_lpid;    /* PID, last operation */
    shmatt_t        shm_nattch;  /* no. of current attaches */
```

```
        time_t          shm_atime;   /* time of last attach */
        time_t          shm_dtime;   /* time of last detach */
        time_t          shm_ctime;   /* time of last change */
    };
```

shm_perm *ipc_perm* structure that specifies the access permissions on the shared memory segment.

shm_segsz Size in bytes of the shared memory segment.

shm_cpid ID of the process that created the shared memory segment.

shm_lpid ID of the last process that executed a **shmat**(2) or **shmdt**(2) system call.

shm_nattch Number of current alive attaches for this shared memory segment.

shm_atime Time of the last **shmat**(2) system call.

shm_dtime Time of the last **shmdt**(2) system call.

shm_ctime Time of the last **shmctl**(2) system call that changed *shmid_ds*.

IPC namespaces

For a discussion of the interaction of System V IPC objects and IPC namespaces, see **namespaces**(7).

SEE ALSO

ipcmk(1), **ipcrm**(1), **ipcs**(1), **lsipc**(1), **ipc**(2), **msgctl**(2), **msgget**(2), **msgrcv**(2), **msgsnd**(2), **semctl**(2), **semget**(2), **semop**(2), **shmat**(2), **shmctl**(2), **shmdt**(2), **shmget**(2), **ftok**(3), **namespaces**(7)

COLOPHON

This page is part of release 4.16 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/.

NAME

namespaces – overview of Linux namespaces

DESCRIPTION

A namespace wraps a global system resource in an abstraction that makes it appear to the processes within the namespace that they have their own isolated instance of the global resource. Changes to the global resource are visible to other processes that are members of the namespace, but are invisible to other processes. One use of namespaces is to implement containers.

Linux provides the following namespaces:

lB lB lB l lB l. Namespace ConstantIsolates Cgroup CLONE_NEWCGROUP Cgroup root directory IPC CLONE_NEWIPC System V IPC, POSIX message queues Network CLONE_NEWNET Network devices, stacks, ports, etc. Mount CLONE_NEWNS Mount points PID CLONE_NEWPID Process IDs User CLONE_NEWUSER User and group IDs UTS CLONE_NEWUTS Hostname and NIS domain name

This page describes the various namespaces and the associated */proc* files, and summarizes the APIs for working with namespaces.

The namespaces API

As well as various */proc* files described below, the namespaces API includes the following system calls:

clone(2)
: The **clone**(2) system call creates a new process. If the *flags* argument of the call specifies one or more of the **CLONE_NEW*** flags listed below, then new namespaces are created for each flag, and the child process is made a member of those namespaces. (This system call also implements a number of features unrelated to namespaces.)

setns(2)
: The **setns**(2) system call allows the calling process to join an existing namespace. The namespace to join is specified via a file descriptor that refers to one of the */proc/[pid]/ns* files described below.

unshare(2)
: The **unshare**(2) system call moves the calling process to a new namespace. If the *flags* argument of the call specifies one or more of the **CLONE_NEW*** flags listed below, then new namespaces are created for each flag, and the calling process is made a member of those namespaces. (This system call also implements a number of features unrelated to namespaces.)

Creation of new namespaces using **clone**(2) and **unshare**(2) in most cases requires the **CAP_SYS_ADMIN** capability. User namespaces are the exception: since Linux 3.8, no privilege is required to create a user namespace.

The /proc/[pid]/ns/ directory

Each process has a */proc/[pid]/ns/* subdirectory containing one entry for each namespace that supports being manipulated by **setns**(2):

```
$ ls -l /proc/$$/ns
total 0
lrwxrwxrwx. 1 mtk mtk 0 Apr 28 12:46 cgroup -> cgroup:[4026531835]
lrwxrwxrwx. 1 mtk mtk 0 Apr 28 12:46 ipc -> ipc:[4026531839]
lrwxrwxrwx. 1 mtk mtk 0 Apr 28 12:46 mnt -> mnt:[4026531840]
lrwxrwxrwx. 1 mtk mtk 0 Apr 28 12:46 net -> net:[4026531969]
lrwxrwxrwx. 1 mtk mtk 0 Apr 28 12:46 pid -> pid:[4026531836]
lrwxrwxrwx. 1 mtk mtk 0 Apr 28 12:46 pid_for_children -> pid:[4026531834]
lrwxrwxrwx. 1 mtk mtk 0 Apr 28 12:46 user -> user:[4026531837]
lrwxrwxrwx. 1 mtk mtk 0 Apr 28 12:46 uts -> uts:[4026531838]
```

Bind mounting (see **mount**(2)) one of the files in this directory to somewhere else in the filesystem keeps the corresponding namespace of the process specified by *pid* alive even if all processes currently in the namespace terminate.

Opening one of the files in this directory (or a file that is bind mounted to one of these files) returns a file handle for the corresponding namespace of the process specified by *pid*. As long as this file descriptor remains open, the namespace will remain alive, even if all processes in the namespace terminate. The file descriptor can be passed to **setns**(2).

In Linux 3.7 and earlier, these files were visible as hard links. Since Linux 3.8, they appear as symbolic links. If two processes are in the same namespace, then the device IDs and inode numbers of their */proc/[pid]/ns/xxx* symbolic links will be the same; an application can check this using the *stat.st_dev* and *stat.st_ino* fields returned by **stat**(2). The content of this symbolic link is a string containing the namespace type and inode number as in the following example:

```
$ readlink /proc/$$/ns/uts
uts:[4026531838]
```

The symbolic links in this subdirectory are as follows:

/proc/[pid]/ns/cgroup (since Linux 4.6)
: This file is a handle for the cgroup namespace of the process.

/proc/[pid]/ns/ipc (since Linux 3.0)
: This file is a handle for the IPC namespace of the process.

/proc/[pid]/ns/mnt (since Linux 3.8)
: This file is a handle for the mount namespace of the process.

/proc/[pid]/ns/net (since Linux 3.0)
: This file is a handle for the network namespace of the process.

/proc/[pid]/ns/pid (since Linux 3.8)
: This file is a handle for the PID namespace of the process. This handle is permanent for the lifetime of the process (i.e., a process's PID namespace membership never changes).

/proc/[pid]/ns/pid_for_children (since Linux 4.12)
: This file is a handle for the PID namespace of child processes created by this process. This can change as a consequence of calls to **unshare**(2) and **setns**(2) (see **pid_namespaces**(7)), so the file may differ from */proc/[pid]/ns/pid*. The symbolic link gains a value only after the first child process is created in the namespace. (Beforehand, **readlink**(2) of the symbolic link will return an empty buffer.)

/proc/[pid]/ns/user (since Linux 3.8)
: This file is a handle for the user namespace of the process.

/proc/[pid]/ns/uts (since Linux 3.0)
: This file is a handle for the UTS namespace of the process.

Permission to dereference or read (**readlink**(2)) these symbolic links is governed by a ptrace access mode **PTRACE_MODE_READ_FSCREDS** check; see **ptrace**(2).

The /proc/sys/user directory

The files in the */proc/sys/user* directory (which is present since Linux 4.9) expose limits on the number of namespaces of various types that can be created. The files are as follows:

max_cgroup_namespaces
: The value in this file defines a per-user limit on the number of cgroup namespaces that may be created in the user namespace.

max_ipc_namespaces
: The value in this file defines a per-user limit on the number of ipc namespaces that may be created in the user namespace.

max_mnt_namespaces
: The value in this file defines a per-user limit on the number of mount namespaces that may be created in the user namespace.

max_net_namespaces
The value in this file defines a per-user limit on the number of network namespaces that may be created in the user namespace.

max_pid_namespaces
The value in this file defines a per-user limit on the number of pid namespaces that may be created in the user namespace.

max_user_namespaces
The value in this file defines a per-user limit on the number of user namespaces that may be created in the user namespace.

max_uts_namespaces
The value in this file defines a per-user limit on the number of user namespaces that may be created in the user namespace.

Note the following details about these files:

* The values in these files are modifiable by privileged processes.
* The values exposed by these files are the limits for the user namespace in which the opening process resides.
* The limits are per-user. Each user in the same user namespace can create namespaces up to the defined limit.
* The limits apply to all users, including UID 0.
* These limits apply in addition to any other per-namespace limits (such as those for PID and user namespaces) that may be enforced.
* Upon encountering these limits, **clone**(2) and **unshare**(2) fail with the error **ENOSPC**.
* For the initial user namespace, the default value in each of these files is half the limit on the number of threads that may be created (*/proc/sys/kernel/threads-max*). In all descendant user namespaces, the default value in each file is **MAXINT**.
* When a namespace is created, the object is also accounted against ancestor namespaces. More precisely:
  + Each user namespace has a creator UID.
  + When a namespace is created, it is accounted against the creator UIDs in each of the ancestor user namespaces, and the kernel ensures that the corresponding namespace limit for the creator UID in the ancestor namespace is not exceeded.
  + The aforementioned point ensures that creating a new user namespace cannot be used as a means to escape the limits in force in the current user namespace.

Cgroup namespaces (CLONE_NEWCGROUP)
See **cgroup_namespaces**(7).

IPC namespaces (CLONE_NEWIPC)
IPC namespaces isolate certain IPC resources, namely, System V IPC objects (see **svipc**(7)) and (since Linux 2.6.30) POSIX message queues (see **mq_overview**(7)). The common characteristic of these IPC mechanisms is that IPC objects are identified by mechanisms other than filesystem pathnames.

Each IPC namespace has its own set of System V IPC identifiers and its own POSIX message queue filesystem. Objects created in an IPC namespace are visible to all other processes that are members of that namespace, but are not visible to processes in other IPC namespaces.

The following */proc* interfaces are distinct in each IPC namespace:

* The POSIX message queue interfaces in */proc/sys/fs/mqueue*.
* The System V IPC interfaces in */proc/sys/kernel*, namely: *msgmax*, *msgmnb*, *msgmni*, *sem*, *shmall*, *shmmax*, *shmmni*, and *shm_rmid_forced*.

* The System V IPC interfaces in */proc/sysvipc*.

When an IPC namespace is destroyed (i.e., when the last process that is a member of the namespace terminates), all IPC objects in the namespace are automatically destroyed.

Use of IPC namespaces requires a kernel that is configured with the **CONFIG_IPC_NS** option.

Network namespaces (CLONE_NEWNET)
See **network_namespaces**(7).

Mount namespaces (CLONE_NEWNS)
See **mount_namespaces**(7).

PID namespaces (CLONE_NEWPID)
See **pid_namespaces**(7).

User namespaces (CLONE_NEWUSER)
See **user_namespaces**(7).

UTS namespaces (CLONE_NEWUTS)
UTS namespaces provide isolation of two system identifiers: the hostname and the NIS domain name. These identifiers are set using **sethostname**(2) and **setdomainname**(2), and can be retrieved using **uname**(2), **gethostname**(2), and **getdomainname**(2).

Use of UTS namespaces requires a kernel that is configured with the **CONFIG_UTS_NS** option.

EXAMPLE

See **clone**(2) and **user_namespaces**(7).

SEE ALSO

nsenter(1), **readlink**(1), **unshare**(1), **clone**(2), **ioctl_ns**(2), **setns**(2), **unshare**(2), **proc**(5), **capabilities**(7), **cgroup_namespaces**(7), **cgroups**(7), **credentials**(7), **network_namespaces**(7), **pid_namespaces**(7), **user_namespaces**(7), **lsns**(8), **switch_root**(8)

COLOPHON

This page is part of release 4.16 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/.

NAME

ftok − convert a pathname and a project identifier to a System V IPC key

SYNOPSIS

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

key_t ftok(const char ****pathname*, int** *proj_id***);**

DESCRIPTION

The **ftok**() function uses the identity of the file named by the given *pathname* (which must refer to an existing, accessible file) and the least significant 8 bits of *proj_id* (which must be nonzero) to generate a *key_t* type System V IPC key, suitable for use with **msgget**(2), **semget**(2), or **shmget**(2).

The resulting value is the same for all pathnames that name the same file, when the same value of *proj_id* is used. The value returned should be different when the (simultaneously existing) files or the project IDs differ.

RETURN VALUE

On success, the generated *key_t* value is returned. On failure −1 is returned, with *errno* indicating the error as for the **stat**(2) system call.

ATTRIBUTES

For an explanation of the terms used in this section, see **attributes**(7).

allbox; lb lb lb l l l. InterfaceAttributeValue T{ **ftok**() T} Thread safety MT-Safe

CONFORMING TO

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

NOTES

On some ancient systems, the prototype was:

```
key_t ftok(char *pathname, char proj_id);
```

Today, *proj_id* is an *int*, but still only 8 bits are used. Typical usage has an ASCII character *proj_id*, that is why the behavior is said to be undefined when *proj_id* is zero.

Of course, no guarantee can be given that the resulting *key_t* is unique. Typically, a best-effort attempt combines the given *proj_id* byte, the lower 16 bits of the inode number, and the lower 8 bits of the device number into a 32-bit result. Collisions may easily happen, for example between files on */dev/hda1* and files on */dev/sda1*.

SEE ALSO

msgget(2), **semget**(2), **shmget**(2), **stat**(2), **svipc**(7)

COLOPHON

This page is part of release 4.16 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/.

NAME

ipc − System V IPC system calls

SYNOPSIS

int ipc(unsigned int *call*, **int** *first*, **int** *second*, **int** *third*,
void * *ptr*, **long** *fifth*);

DESCRIPTION

ipc() is a common kernel entry point for the System V IPC calls for messages, semaphores, and shared memory. *call* determines which IPC function to invoke; the other arguments are passed through to the appropriate call.

User-space programs should call the appropriate functions by their usual names. Only standard library implementors and kernel hackers need to know about **ipc**().

CONFORMING TO

ipc() is Linux-specific, and should not be used in programs intended to be portable.

NOTES

On some architectures—for example x86-64 and ARM—there is no **ipc**() system call; instead, **msgctl**(2), **semctl**(2), **shmctl**(2), and so on really are implemented as separate system calls.

SEE ALSO

msgctl(2), **msgget**(2), **msgrcv**(2), **msgsnd**(2), **semctl**(2), **semget**(2), **semop**(2), **semtimedop**(2), **shmat**(2), **shmctl**(2), **shmdt**(2), **shmget**(2), **svipc**(7)

COLOPHON

This page is part of release 4.16 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/.

NAME

ipcmk − make various IPC resources

SYNOPSIS

ipcmk [options]

DESCRIPTION

ipcmk allows you to create shared memory segments, message queues, and semaphore arrays.

OPTIONS

Resources can be specified with these options:

−M, **−−shmem** *size*
: Create a shared memory segment of *size* bytes. The *size* argument may be followed by the multiplicative suffixes KiB (=1024), MiB (=1024*1024), and so on for GiB, etc. (the "iB" is optional, e.g., "K" has the same meaning as "KiB") or the suffixes KB (=1000), MB (=1000*1000), and so on for GB, etc.

−Q, **−−queue**
: Create a message queue.

−S, **−−semaphore** *number*
: Create a semaphore array with *number* of elements.

Other options are:

−p, **−−mode** *mode*
: Access permissions for the resource. Default is 0644.

−V, **−−version**
: Display version information and exit.

−h, **−−help**
: Display help text and exit.

SEE ALSO

ipcrm(1), **ipcs**(1)

AUTHOR

Hayden A. James ⟨hayden.james@gmail.com⟩

AVAILABILITY

The ipcmk command is part of the util-linux package and is available from Linux Kernel Archive ⟨https://www.kernel.org/pub/linux/utils/util-linux/⟩.

NAME

ipcrm − remove certain IPC resources

SYNOPSIS

ipcrm [options]

ipcrm {**shm**|**msg**|**sem**} *id*...

DESCRIPTION

ipcrm removes System V inter-process communication (IPC) objects and associated data structures from the system. In order to delete such objects, you must be superuser, or the creator or owner of the object.

System V IPC objects are of three types: shared memory, message queues, and semaphores. Deletion of a message queue or semaphore object is immediate (regardless of whether any process still holds an IPC identifier for the object). A shared memory object is only removed after all currently attached processes have detached (**shmdt**(2)) the object from their virtual address space.

Two syntax styles are supported. The old Linux historical syntax specifies a three-letter keyword indicating which class of object is to be deleted, followed by one or more IPC identifiers for objects of this type.

The SUS-compliant syntax allows the specification of zero or more objects of all three types in a single command line, with objects specified either by key or by identifier (see below). Both keys and identifiers may be specified in decimal, hexadecimal (specified with an initial '0x' or '0X'), or octal (specified with an initial '0').

The details of the removes are described in **shmctl**(2), **msgctl**(2), and **semctl**(2). The identifiers and keys can be found by using **ipcs**(1).

OPTIONS

-a, **−−all** [**shm**] [**msg**] [**sem**]
: Remove all resources. When an option argument is provided, the removal is performed only for the specified resource types. *Warning!* Do not use **−a** if you are unsure how the software using the resources might react to missing objects. Some programs create these resources at startup and may not have any code to deal with an unexpected disappearance.

−M, **−−shmem−key** *shmkey*
: Remove the shared memory segment created with *shmkey* after the last detach is performed.

−m, **−−shmem−id** *shmid*
: Remove the shared memory segment identified by *shmid* after the last detach is performed.

−Q, **−−queue−key** *msgkey*
: Remove the message queue created with *msgkey*.

−q, **−−queue−id** *msgid*
: Remove the message queue identified by *msgid*.

−S, **−−semaphore−key** *semkey*
: Remove the semaphore created with *semkey*.

−s, **−−semaphore−id** *semid*
: Remove the semaphore identified by *semid*.

−V, **−−version**
: Display version information and exit.

−h, **−−help**
: Display help text and exit.

NOTES

In its first Linux implementation, **ipcrm** used the deprecated syntax shown in the second line of the **SYNOPSIS**. Functionality present in other *nix implementations of **ipcrm** has since been added, namely the ability to delete resources by key (not just identifier), and to respect the same command-line syntax. For backward compatibility the previous syntax is still supported.

SEE ALSO

ipcmk(1), **ipcs**(1), **msgctl**(2), **msgget**(2), **semctl**(2), **semget**(2), **shmctl**(2), **shmdt**(2), **shmget**(2), **ftok**(3)

AVAILABILITY

The ipcrm command is part of the util-linux package and is available from Linux Kernel Archive ⟨https://www.kernel.org/pub/linux/utils/util-linux/⟩.

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

ipcrm — remove an XSI message queue, semaphore set, or shared memory segment identifier

SYNOPSIS

ipcrm **[**−q msgid**|**−Q msgkey**|**−s semid**|**−S semkey**|**−m shmid**|**−M shmkey**]**...

DESCRIPTION

The *ipcrm* utility shall remove zero or more message queues, semaphore sets, or shared memory segments. The interprocess communication facilities to be removed are specified by the options.

Only a user with appropriate privileges shall be allowed to remove an interprocess communication facility that was not created by or owned by the user invoking *ipcrm*.

OPTIONS

The *ipcrm* utility shall conform to the Base Definitions volume of POSIX.1-2008, *Section 12.2*, *Utility Syntax Guidelines*.

The following options shall be supported:

−q *msgid* Remove the message queue identifier *msgid* from the system and destroy the message queue and data structure associated with it.

−m *shmid* Remove the shared memory identifier *shmid* from the system. The shared memory segment and data structure associated with it shall be destroyed after the last detach.

−s *semid* Remove the semaphore identifier *semid* from the system and destroy the set of semaphores and data structure associated with it.

−Q *msgkey* Remove the message queue identifier, created with key *msgkey*, from the system and destroy the message queue and data structure associated with it.

−M *shmkey*
Remove the shared memory identifier, created with key *shmkey*, from the system. The shared memory segment and data structure associated with it shall be destroyed after the last detach.

−S *semkey* Remove the semaphore identifier, created with key *semkey*, from the system and destroy the set of semaphores and data structure associated with it.

OPERANDS

None.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *ipcrm*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of POSIX.1-2008, *Section 8.2*, *Internationalization Variables* for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS
Default.

STDOUT
Not used.

STDERR
The standard error shall be used only for diagnostic messages.

OUTPUT FILES
None.

EXTENDED DESCRIPTION
None.

EXIT STATUS
The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS
Default.

The following sections are informative.

APPLICATION USAGE
None.

EXAMPLES
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
ipcs

The Base Definitions volume of POSIX.1-2008, *Chapter 8*, *Environment Variables*, *Section 12.2*, *Utility Syntax Guidelines*

The System Interfaces volume of POSIX.1-2008, *msgctl*( ), *semctl*( ), *shmctl*( )

COPYRIGHT

NAME

ipcs − show information on IPC facilities

SYNOPSIS

ipcs [options]

DESCRIPTION

ipcs shows information on the inter-process communication facilities for which the calling process has read access. By default it shows information about all three resources: shared memory segments, message queues, and semaphore arrays.

OPTIONS

−i, **−−id** *id*
Show full details on just the one resource element identified by *id*. This option needs to be combined with one of the three resource options: **−m**, **−q** or **−s**.

−h, **−−help**
Display help text and exit.

−V, **−−version**
Display version information and exit.

Resource options

−m, **−−shmems**
Write information about active shared memory segments.

−q, **−−queues**
Write information about active message queues.

−s, **−−semaphores**
Write information about active semaphore sets.

−a, **−−all**
Write information about all three resources (default).

Output formats

Of these options only one takes effect: the last one specified.

−c, **−−creator**
Show creator and owner.

−l, **−−limits**
Show resource limits.

−p, **−−pid**
Show PIDs of creator and last operator.

−t, **−−time**
Write time information. The time of the last control operation that changed the access permissions for all facilities, the time of the last **msgsnd**(2) and **msgrcv**(2) operations on message queues, the time of the last **shmat**(2) and **shmdt**(2) operations on shared memory, and the time of the last **semop**(2) operation on semaphores.

−u, **−−summary**
Show status summary.

Representation

These affect only the **−l** (**−−limits**) option.

−b, **−−bytes**
Print sizes in bytes.

−−human
Print sizes in human-readable format.

SEE ALSO

ipcmk(1), **ipcrm**(1), **msgrcv**(2), **msgsnd**(2), **semget**(2), **semop**(2), **shmat**(2), **shmdt**(2), **shmget**(2)

CONFORMING TO

The Linux ipcs utility is not fully compatible to the POSIX ipcs utility. The Linux version does not support the POSIX **−a**, **−b** and **−o** options, but does support the **−l** and **−u** options not defined by POSIX. A portable application shall not use the **−a**, **−b**, **−o**, **−l**, and **−u** options.

AUTHOR

Krishna Balasubramanian ⟨balasub@cis.ohio-state.edu⟩

AVAILABILITY

The ipcs command is part of the util-linux package and is available from Linux Kernel Archive ⟨https://www.kernel.org/pub/linux/utils/util-linux/⟩.

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

ipcs — report XSI interprocess communication facilities status

SYNOPSIS

ipcs **[**−qms**]** **[**−a|−bcopt**]**

DESCRIPTION

The *ipcs* utility shall write information about active interprocess communication facilities.

Without options, information shall be written in short format for message queues, shared memory segments, and semaphore sets that are currently active in the system. Otherwise, the information that is displayed is controlled by the options specified.

OPTIONS

The *ipcs* utility shall conform to the Base Definitions volume of POSIX.1-2008, *Section 12.2*, *Utility Syntax Guidelines*.

The *ipcs* utility accepts the following options:

−q Write information about active message queues.

−m Write information about active shared memory segments.

−s Write information about active semaphore sets.

If **−q**, **−m**, or **−s** are specified, only information about those facilities shall be written. If none of these three are specified, information about all three shall be written subject to the following options:

−a Use all print options. (This is a shorthand notation for **−b**, **−c**, **−o**, **−p**, and **−t**.)

−b Write information on maximum allowable size. (Maximum number of bytes in messages on queue for message queues, size of segments for shared memory, and number of semaphores in each set for semaphores.)

−c Write creator's user name and group name; see below.

−o Write information on outstanding usage. (Number of messages on queue and total number of bytes in messages on queue for message queues, and number of processes attached to shared memory segments.)

−p Write process number information. (Process ID of the last process to send a message and process ID of the last process to receive a message on message queues, process ID of the creating process, and process ID of the last process to attach or detach on shared memory segments.)

−t Write time information. (Time of the last control operation that changed the access permissions for all facilities, time of the last *msgsnd*() and *msgrcv*() operations on message queues, time of the last *shmat*() and *shmdt*() operations on shared memory, and time of the last *semop*() operation on semaphores.)

OPERANDS

None.

STDIN

Not used.

INPUT FILES

* The group database

* The user database

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *ipcs*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of POSIX.1-2008, *Section 8.2*, *Internationalization Variables* for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE
Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH Determine the location of message catalogs for the processing of *LC_MESSAGES*.

TZ Determine the timezone for the date and time strings written by *ipcs*. If *TZ* is unset or null, an unspecified default timezone shall be used.

ASYNCHRONOUS EVENTS

Default.

STDOUT

An introductory line shall be written with the format:

"IPC status from %s as of %s\n", <*source***>, <***date***>**

where <*source*> indicates the source used to gather the statistics and <*date*> is the information that would be produced by the *date* command when invoked in the POSIX locale.

The *ipcs* utility then shall create up to three reports depending upon the **−q**, **−m**, and **−s** options. The first report shall indicate the status of message queues, the second report shall indicate the status of shared memory segments, and the third report shall indicate the status of semaphore sets.

If the corresponding facility is not installed or has not been used since the last reboot, then the report shall be written out in the format:

"%s facility not in system.\n", <*facility***>**

where <*facility*> is *Message Queue*, *Shared Memory*, or *Semaphore*, as appropriate. If the facility has been installed and has been used since the last reboot, column headings separated by one or more <space> characters and followed by a <newline> shall be written as indicated below followed by the facility name written out using the format:

"%s:\n", <*facility***>**

where <*facility*> is *Message Queues*, *Shared Memory*, or *Semaphores*, as appropriate. On the second and third reports the column headings need not be written if the last column headings written already provide column headings for all information in that report.

The column headings provided in the first column below and the meaning of the information in those columns shall be given in order below; the letters in parentheses indicate the options that shall cause the corresponding column to appear; ''all'' means that the column shall always appear. Each column is separated by one or more <space> characters. Note that these options only determine what information is provided for each report; they do not determine which reports are written.

T (all) Type of facility:

q Message queue.

m Shared memory segment.

s Semaphore.

This field is a single character written using the format **%c**.

ID (all) The identifier for the facility entry. This field shall be written using the format **%d**.

KEY (all) The key used as an argument to *msgget*(), *semget*(), or *shmget*() to create the facility entry.

Note: The key of a shared memory segment is changed to IPC_PRIVATE when the segment has been removed until all processes attached to the segment detach it.

This field shall be written using the format 0x%x.

MODE (all) The facility access modes and flags. The mode shall consist of 11 characters that are interpreted as follows.

The first character shall be:

S If a process is waiting on a *msgsnd*() operation.

– If the above is not true.

The second character shall be:

R If a process is waiting on a *msgrcv*() operation.

C or – If the associated shared memory segment is to be cleared when the first attach operation is executed.

– If none of the above is true.

The next nine characters shall be interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the usergroup of the facility entry; and the last to all others. Within each set, the first character indicates permission to read, the second character indicates permission to write or alter the facility entry, and the last character is a minus-sign ('–').

The permissions shall be indicated as follows:

r If read permission is granted.

w If write permission is granted.

a If alter permission is granted.

– If the indicated permission is not granted.

The first character following the permissions specifies if there is an alternate or additional access control method associated with the facility. If there is no alternate or additional access control method associated with the facility, a single <space> shall be written; otherwise, another printable character is written.

OWNER (all)

The user name of the owner of the facility entry. If the user name of the owner is found in the user database, at least the first eight column positions of the name shall be written using the format **%s**. Otherwise, the user ID of the owner shall be written using the format **%d**.

GROUP (all) The group name of the owner of the facility entry. If the group name of the owner is found in the group database, at least the first eight column positions of the name shall be written using the format **%s**. Otherwise, the group ID of the owner shall be written using the format **%d**.

The following nine columns shall be only written out for message queues:

CREATOR (**a**,**c**)
: The user name of the creator of the facility entry. If the user name of the creator is found in the user database, at least the first eight column positions of the name shall be written using the format **%s**. Otherwise, the user ID of the creator shall be written using the format **%d**.

CGROUP (**a**,**c**)
: The group name of the creator of the facility entry. If the group name of the creator is found in the group database, at least the first eight column positions of the name shall be written using the format **%s**. Otherwise, the group ID of the creator shall be written using the format **%d**.

CBYTES (**a**,**o**)
: The number of bytes in messages currently outstanding on the associated message queue. This field shall be written using the format **%d**.

QNUM (**a**,**o**)
: The number of messages currently outstanding on the associated message queue. This field shall be written using the format **%d**.

QBYTES (**a**,**b**)
: The maximum number of bytes allowed in messages outstanding on the associated message queue. This field shall be written using the format **%d**.

LSPID (**a**,**p**)
: The process ID of the last process to send a message to the associated queue. This field shall be written using the format:

  "%d", <*pid***>**

  where <*pid*> is 0 if no message has been sent to the corresponding message queue; otherwise, <*pid*> shall be the process ID of the last process to send a message to the queue.

LRPID (**a**,**p**)
: The process ID of the last process to receive a message from the associated queue. This field shall be written using the format:

  "%d", <*pid***>**

  where <*pid*> is 0 if no message has been received from the corresponding message queue; otherwise, <*pid*> shall be the process ID of the last process to receive a message from the queue.

STIME (**a**,**t**)
: The time the last message was sent to the associated queue. If a message has been sent to the corresponding message queue, the hour, minute, and second of the last time a message was sent to the queue shall be written using the format **%d**:**%2.2d**:**%2.2d**. Otherwise, the format **" no-entry"** shall be written.

RTIME (**a**,**t**)
: The time the last message was received from the associated queue. If a message has been received from the corresponding message queue, the hour, minute, and second of the last time a message was received from the queue shall be written using the format **%d**:**%2.2d**:**%2.2d**. Otherwise, the format **" no-entry"** shall be written.

The following eight columns shall be only written out for shared memory segments.

CREATOR (**a**,**c**)
: The user of the creator of the facility entry. If the user name of the creator is found in the user database, at least the first eight column positions of the name shall be written using the format **%s**. Otherwise, the user ID of the creator shall be written using the format **%d**.

CGROUP (**a**,**c**)
: The group name of the creator of the facility entry. If the group name of the creator is found in the group database, at least the first eight column positions of the name shall be written using the format **%s**. Otherwise, the group ID of the creator shall be written using the format **%d**.

NATTCH (**a**,**o**)
The number of processes attached to the associated shared memory segment. This field shall be written using the format **%d**.

SEGSZ (**a**,**b**) The size of the associated shared memory segment. This field shall be written using the format **%d**.

CPID (**a**,**p**) The process ID of the creator of the shared memory entry. This field shall be written using the format **%d**.

LPID (**a**,**p**) The process ID of the last process to attach or detach the shared memory segment. This field shall be written using the format:

"%d", <*pid***>**

where <*pid*> is 0 if no process has attached the corresponding shared memory segment; otherwise, <*pid*> shall be the process ID of the last process to attach or detach the segment.

ATIME (**a**,**t**) The time the last attach on the associated shared memory segment was completed. If the corresponding shared memory segment has ever been attached, the hour, minute, and second of the last time the segment was attached shall be written using the format **%d**:**%2.2d**:**%2.2d**. Otherwise, the format **" no-entry"** shall be written.

DTIME (**a**,**t**) The time the last detach on the associated shared memory segment was completed. If the corresponding shared memory segment has ever been detached, the hour, minute, and second of the last time the segment was detached shall be written using the format **%d**:**%2.2d**:**%2.2d**. Otherwise, the format **" no-entry"** shall be written.

The following four columns shall be only written out for semaphore sets:

CREATOR (**a**,**c**)
The user of the creator of the facility entry. If the user name of the creator is found in the user database, at least the first eight column positions of the name shall be written using the format **%s**. Otherwise, the user ID of the creator shall be written using the format **%d**.

CGROUP (**a**,**c**)
The group name of the creator of the facility entry. If the group name of the creator is found in the group database, at least the first eight column positions of the name shall be written using the format **%s**. Otherwise, the group ID of the creator shall be written using the format **%d**.

NSEMS (**a**,**b**)
The number of semaphores in the set associated with the semaphore entry. This field shall be written using the format **%d**.

OTIME (**a**,**t**) The time the last semaphore operation on the set associated with the semaphore entry was completed. If a semaphore operation has ever been performed on the corresponding semaphore set, the hour, minute, and second of the last semaphore operation on the semaphore set shall be written using the format **%d**:**%2.2d**:**%2.2d**. Otherwise, the format **" no-entry"** shall be written.

The following column shall be written for all three reports when it is requested:

CTIME (**a**,**t**) The time the associated entry was created or changed. The hour, minute, and second of the time when the associated entry was created shall be written using the format **%d**:**%2.2d**:**%2.2d**.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

The following sections are informative.

APPLICATION USAGE

Things can change while *ipcs* is running; the information it gives is guaranteed to be accurate only when it was retrieved.

EXAMPLES

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

ipcrm

The Base Definitions volume of POSIX.1-2008, *Chapter 8*, *Environment Variables*, *Section 12.2*, *Utility Syntax Guidelines*

The System Interfaces volume of POSIX.1-2008, *msgrcv*( ), *msgsnd*( ), *semget*( ), *semop*( ), *shmat*( ), *shmdt*( ), *shmget*( )

COPYRIGHT

NAME

msgctl – System V message control operations

SYNOPSIS

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

int msgctl(int *msqid*, **int** *cmd*, **struct msqid_ds** **buf*);

DESCRIPTION

msgctl() performs the control operation specified by *cmd* on the System V message queue with identifier *msqid*.

The *msqid_ds* data structure is defined in *<sys/msg.h>* as follows:

```
struct msqid_ds {
    struct ipc_perm msg_perm;     /* Ownership and permissions */
    time_t          msg_stime;    /* Time of last msgsnd(2) */
    time_t          msg_rtime;    /* Time of last msgrcv(2) */
    time_t          msg_ctime;    /* Time of last change */
    unsigned long   __msg_cbytes; /* Current number of bytes in
                                     queue (nonstandard) */
    msgqnum_t       msg_qnum;     /* Current number of messages
                                     in queue */
    msglen_t        msg_qbytes;   /* Maximum number of bytes
                                     allowed in queue */
    pid_t           msg_lspid;    /* PID of last msgsnd(2) */
    pid_t           msg_lrpid;    /* PID of last msgrcv(2) */
};
```

The *ipc_perm* structure is defined as follows (the highlighted fields are settable using **IPC_SET**):

```
struct ipc_perm {
    key_t          __key;       /* Key supplied to msgget(2) */
    uid_t          uid;         /* Effective UID of owner */
    gid_t          gid;         /* Effective GID of owner */
    uid_t          cuid;        /* Effective UID of creator */
    gid_t          cgid;        /* Effective GID of creator */
    unsigned short mode;        /* Permissions */
    unsigned short __seq;       /* Sequence number */
};
```

Valid values for *cmd* are:

IPC_STAT
Copy information from the kernel data structure associated with *msqid* into the *msqid_ds* structure pointed to by *buf*. The caller must have read permission on the message queue.

IPC_SET
Write the values of some members of the *msqid_ds* structure pointed to by *buf* to the kernel data structure associated with this message queue, updating also its *msg_ctime* member. The following members of the structure are updated: *msg_qbytes*, *msg_perm.uid*, *msg_perm.gid*, and (the least significant 9 bits of) *msg_perm.mode*. The effective UID of the calling process must match the owner (*msg_perm.uid*) or creator (*msg_perm.cuid*) of the message queue, or the caller must be privileged. Appropriate privilege (Linux: the **CAP_SYS_RESOURCE** capability) is required to raise the *msg_qbytes* value beyond the system parameter **MSGMNB**.

IPC_RMID
Immediately remove the message queue, awakening all waiting reader and writer processes (with an error return and *errno* set to **EIDRM**). The calling process must have appropriate privileges or

its effective user ID must be either that of the creator or owner of the message queue. The third argument to **msgctl**() is ignored in this case.

IPC_INFO (Linux-specific)
Return information about system-wide message queue limits and parameters in the structure pointed to by *buf*. This structure is of type *msginfo* (thus, a cast is required), defined in *<sys/msg.h>* if the **_GNU_SOURCE** feature test macro is defined:

```
struct msginfo {
    int msgpool; /* Size in kibibytes of buffer pool
                    used to hold message data;
                    unused within kernel */
    int msgmap;  /* Maximum number of entries in message
                    map; unused within kernel */
    int msgmax;  /* Maximum number of bytes that can be
                    written in a single message */
    int msgmnb;  /* Maximum number of bytes that can be
                    written to queue; used to initialize
                    msg_qbytes during queue creation
                    (msgget(2)) */
    int msgmni;  /* Maximum number of message queues */
    int msgssz;  /* Message segment size;
                    unused within kernel */
    int msgtql;  /* Maximum number of messages on all queues
                    in system; unused within kernel */
    unsigned short int msgseg;
                 /* Maximum number of segments;
                    unused within kernel */
};
```

The *msgmni*, *msgmax*, and *msgmnb* settings can be changed via */proc* files of the same name; see **proc**(5) for details.

MSG_INFO (Linux-specific)
Return a *msginfo* structure containing the same information as for **IPC_INFO**, except that the following fields are returned with information about system resources consumed by message queues: the *msgpool* field returns the number of message queues that currently exist on the system; the *msgmap* field returns the total number of messages in all queues on the system; and the *msgtql* field returns the total number of bytes in all messages in all queues on the system.

MSG_STAT (Linux-specific)
Return a *msqid_ds* structure as for **IPC_STAT**. However, the *msqid* argument is not a queue identifier, but instead an index into the kernel's internal array that maintains information about all message queues on the system.

RETURN VALUE

On success, **IPC_STAT**, **IPC_SET**, and **IPC_RMID** return 0. A successful **IPC_INFO** or **MSG_INFO** operation returns the index of the highest used entry in the kernel's internal array recording information about all message queues. (This information can be used with repeated **MSG_STAT** operations to obtain information about all queues on the system.) A successful **MSG_STAT** operation returns the identifier of the queue whose index was given in *msqid*.

On error, −1 is returned with *errno* indicating the error.

ERRORS

On failure, *errno* is set to one of the following:

EACCES
The argument *cmd* is equal to **IPC_STAT** or **MSG_STAT**, but the calling process does not have read permission on the message queue *msqid*, and does not have the **CAP_IPC_OWNER**

capability in the user namespace that governs its IPC namespace.

EFAULT
The argument *cmd* has the value **IPC_SET** or **IPC_STAT**, but the address pointed to by *buf* isn't accessible.

EIDRM
The message queue was removed.

EINVAL
Invalid value for *cmd* or *msqid*. Or: for a **MSG_STAT** operation, the index value specified in *msqid* referred to an array slot that is currently unused.

EPERM
The argument *cmd* has the value **IPC_SET** or **IPC_RMID**, but the effective user ID of the calling process is not the creator (as found in *msg_perm.cuid*) or the owner (as found in *msg_perm.uid*) of the message queue, and the caller is not privileged (Linux: does not have the **CAP_SYS_ADMIN** capability).

EPERM
An attempt (**IPC_SET**) was made to increase *msg_qbytes* beyond the system parameter **MSGMNB**, but the caller is not privileged (Linux: does not have the **CAP_SYS_RESOURCE** capability).

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

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 **IPC_INFO**, **MSG_STAT** and **MSG_INFO** operations are used by the **ipcs**(1) program to provide information on allocated resources. In the future these may modified or moved to a */proc* filesystem interface.

Various fields in the *struct msqid_ds* were typed as *short* under Linux 2.2 and have become *long* under Linux 2.4. To take advantage of this, a recompilation under glibc-2.1.91 or later should suffice. (The kernel distinguishes old and new calls by an **IPC_64** flag in *cmd*.)

SEE ALSO
msgget(2), **msgrcv**(2), **msgsnd**(2), **capabilities**(7), **mq_overview**(7), **svipc**(7)

COLOPHON
This page is part of release 4.16 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/.

NAME

msgget − get a System V message queue identifier

SYNOPSIS

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

int msgget(key_t *key*, **int** *msgflg*);

DESCRIPTION

The **msgget**() system call returns the System V message queue identifier associated with the value of the *key* argument. It may be used either to obtain the identifier of a previously created message queue (when *msgflg* is zero and *key* does not have the value **IPC_PRIVATE**), or to create a new set.

A new message queue is created if *key* has the value **IPC_PRIVATE** or *key* isn't **IPC_PRIVATE**, no message queue with the given key *key* exists, and **IPC_CREAT** is specified in *msgflg*.

If *msgflg* specifies both **IPC_CREAT** and **IPC_EXCL** and a message queue already exists for *key*, then **msgget**() fails with *errno* set to **EEXIST**. (This is analogous to the effect of the combination **O_CREAT | O_EXCL** for **open**(2).)

Upon creation, the least significant bits of the argument *msgflg* define the permissions of the message queue. These permission bits have the same format and semantics as the permissions specified for the *mode* argument of **open**(2). (The execute permissions are not used.)

If a new message queue is created, then its associated data structure *msqid_ds* (see **msgctl**(2)) is initialized as follows:

msg_perm.cuid and *msg_perm.uid* are set to the effective user ID of the calling process.

msg_perm.cgid and *msg_perm.gid* are set to the effective group ID of the calling process.

The least significant 9 bits of *msg_perm.mode* are set to the least significant 9 bits of *msgflg*.

msg_qnum, *msg_lspid*, *msg_lrpid*, *msg_stime*, and *msg_rtime* are set to 0.

msg_ctime is set to the current time.

msg_qbytes is set to the system limit **MSGMNB**.

If the message queue already exists the permissions are verified, and a check is made to see if it is marked for destruction.

RETURN VALUE

If successful, the return value will be the message queue identifier (a nonnegative integer), otherwise −1 with *errno* indicating the error.

ERRORS

On failure, *errno* is set to one of the following values:

EACCES
A message queue exists for *key*, but the calling process does not have permission to access the queue, and does not have the **CAP_IPC_OWNER** capability in the user namespace that governs its IPC namespace.

EEXIST
IPC_CREAT and **IPC_EXCL** were specified in *msgflg*, but a message queue already exists for *key*.

ENOENT
No message queue exists for *key* and *msgflg* did not specify **IPC_CREAT**.

ENOMEM
A message queue has to be created but the system does not have enough memory for the new data structure.

ENOSPC
A message queue has to be created but the system limit for the maximum number of message queues (**MSGMNI**) would be exceeded.

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

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.

IPC_PRIVATE isn't a flag field but a *key_t* type. If this special value is used for *key*, the system call ignores everything but the least significant 9 bits of *msgflg* and creates a new message queue (on success).

The following is a system limit on message queue resources affecting a **msgget**() call:

MSGMNI
System-wide limit on the number of message queues. Before Linux 3.19, the default value for this limit was calculated using a formula based on available system memory. Since Linux 3.19, the default value is 32,000. On Linux, this limit can be read and modified via */proc/sys/kernel/msgmni*.

Linux notes
Until version 2.3.20, Linux would return **EIDRM** for a **msgget**() on a message queue scheduled for deletion.

BUGS
The name choice **IPC_PRIVATE** was perhaps unfortunate, **IPC_NEW** would more clearly show its function.

SEE ALSO
msgctl(2), **msgrcv**(2), **msgsnd**(2), **ftok**(3), **capabilities**(7), **mq_overview**(7), **svipc**(7)

COLOPHON
This page is part of release 4.16 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/.

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, respectively, 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 specified 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.)

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.

RETURN VALUE

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:

EACCES
The calling process does not have write permission on the message queue, and does not have the **CAP_IPC_OWNER** capability.

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:

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 requested 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)
MSG_COPY was 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 **msgctl**(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 **MSC_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.

EXAMPLE

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, (void *) &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, (void *) &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), **svipc**(7)

COLOPHON

This page is part of release 4.16 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/.