NAME

ipc − System V interprocess communication mechanisms

SYNOPSIS

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

DESCRIPTION

The manual page refers to the Linux implementation of the System V interprocess communication 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, defined by the *<sys/ipc.h>* system header file, includes the following members:

```
ushort cuid;    /* creator user id */
ushort cgid;    /* creator group id */
ushort uid;     /* owner user id */
ushort gid;     /* owner group id */
ushort 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 defines also the following symbolic constants:

IPC_CREAT Create entry if key doesn't exists.

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 others symbolic constants are flag fields or-able 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 msquid_ds**, defined in *<sys/msg.h>*, containing the following members:


```
struct ipc_perm msg_perm;
ushort msg_qnum;    /* no of messages on queue */
ushort msg_qbytes;  /* bytes max on a queue */
ushort msg_lspid;   /* pid of last msgsnd call */
ushort msg_lrpid;   /* pid of last msgrcv call */
time_t msg_stime;   /* last msgsnd time */
time_t msg_rtime;   /* last msgrcv 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** system call.

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

msg_stime Time of the last **msgsnd** system call.

msg_rtime Time of the last **msgcv** 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 ipc_perm sem_perm;
time_t sem_otime;   /* last operation time */
time_t sem_ctime;   /* last change time */
ushort 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** system call.

sem_ctime Time of last **semctl** 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 non-negative integer ranging from **0** to **sem_nsems−1**.

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

```
ushort semval;  /* semaphore value */
short sempid;   /* pid for last operation */
ushort semncnt; /* no. of awaiting semval to increase */
ushort semzcnt; /* no. of awaiting semval = 0 */
```

semval Semaphore value: a non-negative integer.

sempid ID of the last process that performed a semaphore operation on this semaphore.

semncnt Number of processes suspended awaiting for **semval** to increase.

semznt Number of processes suspended awaiting for **semval** to become zero.

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 ipc_perm shm_perm;
int shm_segsz;  /* size of segment */
ushort shm_cpid;    /* pid of creator */
```

```
ushort shm_lpid;    /* pid, last operation */
short 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** or **shmdt** system call.

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

shm_atime Time of the last **shmat** system call.

shm_dtime Time of the last **shmdt** system call.

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

SEE ALSO

ftok(3), **msgctl**(2), **msgget**(2), **msgrcv**(2), **msgsnd**(2), **semctl**(2), **semget**(2), **semop**(2), **shmat**(2), **shmctl**(2), **shmget**(2), **shmdt**(2).

NAME

msgget − get a 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 function returns the message queue identifier associated to the value of the *key* argument. A new message queue is created if *key* has value **IPC_PRIVATE** or *key* isn't **IPC_PRIVATE**, no existing message queue is associated to *key*, and **IPC_CREAT** is asserted in *msgflg* (i.e. *msgflg***&***IPC_CREAT* is nonzero). The presence in *msgflg* of the fields **IPC_CREAT** and **IPC_EXCL** plays the same role, with respect to the existence of the message queue, as the presence of **O_CREAT** and **O_EXCL** in the mode argument of the **open**(2) system call: i.e. the **msgget** function fails if *msgflg* asserts both **IPC_CREAT** and **IPC_EXCL** and a message queue already exists for *key*.

Upon creation, the lower 9 bits of the argument *msgflg* define the access permissions of the message queue. These permission bits have the same format and semantics as the access permissions parameter in **open**(2) or **creat**(2) system calls. (The execute permissions are not used.)

Furthermore, while creating, the system call initializes the system message queue data structure **msqid_ds** 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 lowest order 9 bits of **msg_perm.mode** are set to the lowest order 9 bit 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 access 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

For a failing return, **errno** will be set to one among the following values:

EACCES A message queue exists for *key*, but the calling process has no access permissions to the queue.

EEXIST A message queue exists for **key** and *msgflg* was asserting both **IPC_CREAT** and **IPC_EXCL**.

EIDRM The message queue is marked for removal.

ENOENT No message queue exists for *key* and *msgflg* wasn't asserting **IPC_CREAT**.

ENOMEM A message queue has to be created but the system has not 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.

NOTES

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 lowest order 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 maximum number of message queues: policy dependent.

BUGS

Use of **IPC_PRIVATE** does not actually prohibit other processes from getting access to the allocated message queue.

As for the files, there is currently no intrinsic way for a process to ensure exclusive access to a message queue. Asserting both **IPC_CREAT** and **IPC_EXCL** in *msgflg* only ensures (on success) that a new message queue will be created, it doesn't imply exclusive access to the message queue.

CONFORMING TO

SVr4, SVID. SVr4 does not document the EIDRM error code.

SEE ALSO

ftok(3), **ipc**(5), **msgctl**(2), **msgsnd**(2), **msgrcv**(2).

NAME

msgop − message operations

SYNOPSIS

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

int msgsnd (int *msqid*, **struct msgbuf ****msgp*, **int** *msgsz*, **int** *msgflg* **)**

int msgrcv (int *msqid*, **struct msgbuf ****msgp*, **int** *msgsz*, **long** *msgtyp*, **int** *msgflg* **)**

DESCRIPTION

To send or receive a message, the calling process allocates a structure that looks like the following

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

but with an array **mtext** of size *msgsz*, a non-negative integer value. The structure member **mtype** must have a strictly positive integer value that can be used by the receiving process for message selection (see the section about **msgrcv**).

The calling process must have write access permissions to send and read access permissions to receive a message on the queue.

The **msgsnd** system call enqueue a copy of the message pointed to by the *msgp* argument on the message queue whose identifier is specified by the value of the *msqid* argument.

The argument *msgflg* specifies the system call behaviour if enqueuing the new message will require more than **msg_qbytes** in the queue. Asserting **IPC_NOWAIT** the message will not be sent and the system call fails returning with **errno** set to **EAGAIN**. Otherwise the process is suspended until the condition for the suspension no longer exists (in which case the message is sent and the system call succeeds), or the queue is removed (in which case the system call fails with **errno** set to **EIDRM**), or the process receives a signal that has to be caught (in which case the system call fails with **errno** set to **EINTR**).

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.

The system call **msgrcv** reads a message from the message queue specified by *msqid* into the **msgbuf** pointed to by the *msgp* argument removing from the queue, on success, the read message.

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 if the *msgflg* argument asserts **MSG_NOERROR**, the message text will be truncated (and the truncated part will be lost), otherwise the message isn't removed from the queue and the system call fails returning with **errno** set to **E2BIG**.

The argument *msgtyp* specifies the type of message requested as follows:

If *msgtyp* is **0**, then the message on the queue's front is read.

If *msgtyp* is greater than **0**, then the first message on the queue of type *msgtyp* is read if **MSG_EXCEPT** isn't asserted by the *msgflg* argument, otherwise the first message on the queue of type not equal to *msgtyp* will be read.

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

The *msgflg* argument asserts none, one or more (or−ing them) among the following flags:

IPC_NOWAIT For immediate return if no message of the requested type is on the queue. The system call fails with errno set to **ENOMSG**.

MSG_EXCEPT Used with *msgtyp* greater than **0** to read the first message on 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 asserted in *msgflg*, the calling process is blocked until one of the following conditions occurs:

A message of the desired type is placed on the queue.

The message queue is removed from the system. In such a case the system call fails with **errno** set to **EIDRM**.

The calling process receives a signal that has to be caught. In such a case the system call fails with **errno** set to **EINTR**.

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 a failure both functions return **−1** with **errno** indicating the error, otherwise **msgsnd** returns **0** and **msgrvc** returns the number of bytes actually copied into the **mtext** array.

ERRORS

When **msgsnd** fails, at return **errno** will be set to one among the following values:

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

EACCES The calling process has no write access permissions on the message queue.

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 received a signal that had to be caught.

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 has not enough memory to make a copy of the supplied **msgbuf**.

When **msgrcv** fails, at return **errno** will be set to one among the following values:

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

EACCES The calling process has no read access permissions on the message queue.

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 received a signal that had to be caught.

EINVAL Illegal *msgqid* value, or *msgsz* less than **0**.

ENOMSG **IPC_NOWAIT** was asserted in *msgflg* and no message of the requested type existed on the message queue.

NOTES

The followings are system limits affecting a **msgsnd** system call:

MSGMAX Maximum size for a message text: the implementation set this value to 4080 bytes.

MSGMNB Default maximum size in bytes of a message queue: policy dependent. The super−user can increase the size of a message queue beyond **MSGMNB** by a **msgctl** system call.

The implementation has no intrinsic limits for the system wide maximum number of message headers (**MSGTQL**) and for the system wide maximum size in bytes of the message pool (**MSGPOOL**).

CONFORMING TO

SVr4, SVID.

SEE ALSO

ipc(5), **msgctl**(2), **msgget**(2), **msgrcv**(2), **msgsnd**(2).

NAME

msgctl − 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

The function performs the control operation specified by *cmd* on the message queue with identifier *msqid*. Legal values for *cmd* are:

IPC_STAT Copy info from the message queue data structure into the structure pointed to by *buf*. The user must have read access privileges on the message queue.

IPC_SET Write the values of some members of the **msqid_ds** structure pointed to by *buf* to the message queue data structure, updating also its **msg_ctime** member. Considered members from the user supplied **struct msqid_ds** pointed to by *buf* are

msg_perm.uid
msg_perm.gid
msg_perm.mode/* only lowest 9-bits */
msg_qbytes

The calling process effective user−ID must be one among super−user, creator or owner of the message queue. Only the super−user can raise the **msg_qbytes** value beyond the system parameter **MSGMNB**.

IPC_RMID Remove immediately the message queue and its data structures awakening all waiting reader and writer processes (with an error return and **errno** set to **EIDRM**). The calling process effective user−ID must be one among super−user, creator or owner of the message queue.

RETURN VALUE

If successful, the return value will be **0**, otherwise **−1** with **errno** indicating the error.

ERRORS

For a failing return, **errno** will be set to one among the following values:

EACCES The argument *cmd* is equal to **IPC_STAT** but the calling process has no read access permissions on the message queue *msqid*.

EFAULT The argument *cmd* has 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*.

EPERM The argument *cmd* has value **IPC_SET** or **IPC_RMID** but the calling process effective user−ID has insufficient privileges to execute the command. Note this is also the case of a non super−user process trying to increase the **msg_qbytes** value beyond the value specified by the system parameter **MSGMNB**.

NOTES

The **IPC_INFO**, **MSG_STAT** and **MSG_INFO** control calls are used by the **ipcs**(8) program to provide information on allocated resources. In the future these can be modified as needed or moved to a proc file system interface.

CONFORMING TO

SVr4, SVID. SVID dies not document the EIDRM error condition.

SEE ALSO

ipc(5), **msgget**(2), **msgsnd**(2), **msgrcv**(2).

NAME

semget − get a semaphore set identifier

SYNOPSIS

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

int semget (key_t *key*, **int** *nsems*, **int** *semflg* **)**

DESCRIPTION

The function returns the semaphore set identifier associated to the value of the argument *key*. A new set of *nsems* semaphores is created if *key* has value **IPC_PRIVATE** or *key* isn't **IPC_PRIVATE**, no existing message queue is associated to *key*, and **IPC_CREAT** is asserted in *semflg* (i.e. *semflg***&IPC_CREAT** isn't zero). The presence in *semflg* of the fields **IPC_CREAT** and **IPC_EXCL** plays the same role, with respect to the existence of the semaphore set, as the presence of **O_CREAT** and **O_EXCL** in the mode argument of the **open**(2) system call: i.e. the **msgget** function fails if *semflg* asserts both **IPC_CREAT** and **IPC_EXCL** and a semaphore set already exists for *key*.

Upon creation, the lower 9 bits of the argument *semflg* define the access permissions (for owner, group and others) to the semaphore set in the same format, and with the same meaning, as for the access permissions parameter in the **open**(2) or **creat**(2) system calls (though the execute permissions are not used by the system, and write permissions, for a semaphore set, effectively means alter permissions).

Furthermore, while creating, the system call initializes the system semaphore set data structure **semid_ds** as follows:

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

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

The lowest order 9 bits of **sem_perm.mode** are set to the lowest order 9 bit of *semflg*.

sem_nsems is set to the value of *nsems*.

sem_otime is set to 0.

sem_ctime is set to the current time.

The argument *nsems* can be **0** (a don't care) when the system call isn't a create one. Otherwise *nsems* must be greater than **0** and less or equal to the maximum number of semaphores per semid, (**SEMMSL**).

If the semaphore set already exists, the access 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 semaphore set identifier (a positive integer), otherwise **−1** with **errno** indicating the error.

ERRORS

For a failing return, **errno** will be set to one among the following values:

EACCES A semaphore set exists for *key*, but the calling process has no access permissions to the set.

EEXIST A semaphore set exists for **key** and *semflg* was asserting both **IPC_CREAT** and **IPC_EXCL**.

EIDRM The semaphore set is marked as to be deleted.

ENOENT No semaphore set exists for *key* and *semflg* wasn't asserting **IPC_CREAT**.

ENOMEM A semaphore set has to be created but the system has not enough memory for the new data structure.

ENOSPC A semaphore set has to be created but the system limit for the maximum number of semaphore sets (**SEMMNI**), or the system wide maximum number of semaphores (**SEMMNS**), would be exceeded.

NOTES

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 lowest order 9 bits of *semflg* and creates a new semaphore set (on success).

The followings are limits on semaphore set resources affecting a **semget** call:

SEMMNI System wide maximum number of semaphore sets: policy dependent.

SEMMSL Maximum number of semaphores per semid: implementation dependent (500 currently).

SEMMNS System wide maximum number of semaphores: policy dependent. Values greater than **SEMMSL * SEMMNI** makes it irrelevant.

BUGS

Use of **IPC_PRIVATE** doesn't inhibit to other processes the access to the allocated semaphore set.

As for the files, there is currently no intrinsic way for a process to ensure exclusive access to a semaphore set. Asserting both **IPC_CREAT** and **IPC_EXCL** in *semflg* only ensures (on success) that a new semaphore set will be created, it doesn't imply exclusive access to the semaphore set.

The data structure associated with each semaphore in the set isn't initialized by the system call. In order to initialize those data structures, one has to execute a subsequent call to **semctl**(2) to perform a **SETVAL** or a **SETALL** command on the semaphore set.

CONFORMING TO

SVr4, SVID. SVr4 documents additional error conditions EINVAL, EFBIG, E2BIG, EAGAIN, ERANGE, EFAULT.

SEE ALSO

ftok(3), **ipc**(5), **semctl**(2), **semop**(2).

NAME

semop − semaphore operations

SYNOPSIS

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

int semop (int *semid***, struct sembuf ****sops***, unsigned** *nsops* **)**

DESCRIPTION

The function performs operations on selected members of the semaphore set indicated by *semid*. Each of the *nsops* elements in the array pointed to by *sops* specify an operation to be performed on a semaphore by a **struct sembuf** including the following members:

```
short sem_num;  /* semaphore number: 0 = first */
short sem_op;   /* semaphore operation */
short sem_flg;  /* operation flags */
```

Flags recognized in **sem_flg** are **IPC_NOWAIT** and **SEM_UNDO**. If an operation asserts **SEM_UNDO**, it will be undone when the process exits.

The system call semantic assures that the operations will be performed if and only if all of them will succeed. Each operation is performed on the **sem_num**−th semaphore of the semaphore set − where the first semaphore of the set is semaphore **0** − and is one among the following three.

If **sem_op** is a positive integer, the operation adds this value to **semval**. Furthermore, if **SEM_UNDO** is asserted for this operation, the system updates the process undo count for this semaphore. The operation always goes through, so no process sleeping can happen. The calling process must have alter permissions on the semaphore set.

If **sem_op** is zero, the process must have read access permissions on the semaphore set. If **semval** is zero, the operation goes through. Otherwise, if **IPC_NOWAIT** is asserted in **sem_flg**, the system call fails (undoing all previous actions performed) with **errno** set to **EAGAIN**. Otherwise **semzcnt** is incremented by one and the process sleeps until one of the following occurs:

- **semval** becomes 0, at which time the value of **semzcnt** is decremented.
- The semaphore set is removed: the system call fails with **errno** set to **EIDRM**.
- The calling process receives a signal that has to be caught: the value of **semzcnt** is decremented and the system call fails with **errno** set to **EINTR**.

If **sem_op** is less than zero, the process must have alter permissions on the semaphore set. If **semval** is greater than or equal to the absolute value of **sem_op**, the absolute value of **sem_op** is subtracted by **semval**. Furthermore, if **SEM_UNDO** is asserted for this operation, the system updates the process undo count for this semaphore. Then the operation goes through. Otherwise, if **IPC_NOWAIT** is asserted in **sem_flg**, the system call fails (undoing all previous actions performed) with **errno** set to **EAGAIN**. Otherwise **semncnt** is incremented by one and the process sleeps until one of the following occurs:

- **semval** becomes greater or equal to the absolute value of **sem_op**, at which time the value of **semncnt** is decremented, the absolute value of **sem_op** is subtracted from **semval** and, if **SEM_UNDO** is asserted for this operation, the system updates the process undo count for this semaphore.
- The semaphore set is removed from the system: the system call fails with **errno** set to **EIDRM**.
- The calling process receives a signal that has to be caught: the value of **semncnt** is decremented and the system call fails with **errno** set to **EINTR**.

In case of success, the **sempid** member of the structure **sem** for each semaphore specified in the array pointed to by *sops* is set to the process−ID of the calling process. Furthermore both **sem_otime** and **sem_ctime** are set to the current time.

RETURN VALUE

If successful the system call returns **0**, otherwise it returns **−1** with **errno** indicating the error.

ERRORS

For a failing return, **errno** will be set to one among the following values:

E2BIG The argument *nsops* is greater than **SEMOPM**, the maximum number of operations allowed per system call.

EACCES The calling process has no access permissions on the semaphore set as required by one of the specified operations.

EAGAIN An operation could not go through and **IPC_NOWAIT** was asserted in its *sem_flg*.

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

EFBIG For some operation the value of **sem_num** is less than 0 or greater than or equal to the number of semaphores in the set.

EIDRM The semaphore set was removed.

EINTR Sleeping on a wait queue, the process received a signal that had to be caught.

EINVAL The semaphore set doesn't exist, or *semid* is less than zero, or *nsops* has a non-positive value.

ENOMEM The **sem_flg** of some operation asserted **SEM_UNDO** and the system has not enough memory to allocate the undo structure.

ERANGE For some operation **semop+semval** is greater than **SEMVMX**, the implementation dependent maximum value for **semval**.

NOTES

The **sem_undo** structures of a process aren't inherited by its child on execution of a **fork**(2) system call. They are instead inherited by the substituting process resulting by the execution of the **execve**(2) system call.

The followings are limits on semaphore set resources affecting a **semop** call:

SEMOPM Maximum number of operations allowed for one **semop** call: policy dependent.

SEMVMX Maximum allowable value for **semval**: implementation dependent (32767).

The implementation has no intrinsic limits for the adjust on exit maximum value (**SEMAEM**), the system wide maximum number of undo structures (**SEMMNU**) and the per process maximum number of undo entries system parameters.

BUGS

The system maintains a per process **sem_undo** structure for each semaphore altered by the process with undo requests. Those structures are free at process exit. One major cause for unhappiness with the undo mechanism is that it does not fit in with the notion of having an atomic set of operations an array of semaphores. The undo requests for an array and each semaphore therein may have been accumulated over many **semopt** calls. Should the process sleep when exiting, or should all undo operations be applied with the **IPC_NOWAIT** flag in effect? Currently those undo operations which go through immediately are applied, and those that require a wait are ignored silently. Thus harmless undo usage is guaranteed with private semaphores only.

CONFORMING TO

SVr4, SVID. SVr4 documents additional error conditions EINVAL, EFBIG, ENOSPC.

SEE ALSO

ipc(5), **semctl**(2), **semget**(2).

NAME

semctl − semaphore control operations

SYNOPSIS

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

#if defined(__GNU_LIBRARY__) && !defined(_SEM_SEMUN_UNDEFINED)
/* union semun is defined by including <sys/sem.h> */
#else
/* according to X/OPEN we have to define it ourselves */
union semun {
    int val;                  /* value for SETVAL */
    struct semid_ds *buf;     /* buffer for IPC_STAT, IPC_SET */
    unsigned short int *array;  /* array for GETALL, SETALL */
    struct seminfo *__buf;    /* buffer for IPC_INFO */
};
#endif
```

int semctl (int *semid***, int** *semnum***, int** *cmd***, union semun** *arg***)**

DESCRIPTION

The function performs the control operation specified by *cmd* on the semaphore set (or on the *semnum*-th semaphore of the set) identified by *semid*. The first semaphore of the set is indicated by a value **0** for *semnum*.

Legal values for *cmd* are

IPC_STAT Copy info from the semaphore set data structure into the structure pointed to by *arg***.buf**. The argument *semnum* is ignored. The calling process must have read access privileges on the semaphore set.

IPC_SET Write the values of some members of the **semid_ds** structure pointed to by *arg***.buf** to the semaphore set data structure, updating also its **sem_ctime** member. Considered members from the user supplied **struct semid_ds** pointed to by *arg***.buf** are

sem_perm.uid
sem_perm.gid
sem_perm.mode /* only lowest 9-bits */

The calling process effective user−ID must be one among super−user, creator or owner of the semaphore set. The argument *semnum* is ignored.

IPC_RMID Remove immediately the semaphore set and its data structures awakening all waiting processes (with an error return and **errno** set to **EIDRM**). The calling process effective user−ID must be one among super−user, creator or owner of the semaphore set. The argument *semnum* is ignored.

GETALL Return **semval** for all semaphores of the set into *arg***.array**. The argument *semnum* is ignored. The calling process must have read access privileges on the semaphore set.

GETNCNT The system call returns the value of **semncnt** for the *semnum*−th semaphore of the set (i.e. the number of processes waiting for an increase of **semval** for the *semnum*−th semaphore of the set). The calling process must have read access privileges on the semaphore set.

GETPID The system call returns the value of **sempid** for the *semnum*−th semaphore of the set (i.e. the pid of the process that executed the last **semop** call for the *semnum*−th semaphore of the set). The calling process must have read access privileges on the semaphore set.

GETVAL The system call returns the value of **semval** for the *semnum*−th semaphore of the set. The calling process must have read access privileges on the semaphore set.

GETZCNT The system call returns the value of **semzcnt** for the *semnum*−th semaphore of the set (i.e. the number of processes waiting for **semval** of the *semnum*−th semaphore of the set to become 0). The calling process must have read access privileges on the semaphore set.

SETALL Set **semval** for all semaphores of the set using *arg***.array**, updating also the **sem_ctime** member of the **semid_ds** structure associated to the set. Undo entries are cleared for altered semaphores in all processes. Processes sleeping on the wait queue are awakened if some **semval** becomes 0 or increases. The argument *semnum* is ignored. The calling process must have alter access privileges on the semaphore set.

SETVAL Set the value of **semval** to *arg***.val** for the *semnum*−th semaphore of the set, updating also the **sem_ctime** member of the **semid_ds** structure associated to the set. Undo entry is cleared for altered semaphore in all processes. Processes sleeping on the wait queue are awakened if **semval** becomes 0 or increases. The calling process must have alter access privileges on the semaphore set.

RETURN VALUE

On fail the system call returns **−1** with **errno** indicating the error. Otherwise the system call returns a non-negative value depending on *cmd* as follows:

GETNCNT the value of **semncnt**.

GETPID the value of **sempid**.

GETVAL the value of **semval**.

GETZCNT the value of **semzcnt**.

ERRORS

For a failing return, **errno** will be set to one among the following values:

EACCES The calling process has no access permissions needed to execute *cmd*.

EFAULT The address pointed to by *arg***.buf** or *arg***.array** isn't accessible.

EIDRM The semaphore set was removed.

EINVAL Invalid value for *cmd* or *semid*.

EPERM The argument *cmd* has value **IPC_SET** or **IPC_RMID** but the calling process effective user−ID has insufficient privileges to execute the command.

ERANGE The argument *cmd* has value **SETALL** or **SETVAL** and the value to which **semval** has to be set (for some semaphore of the set) is less than 0 or greater than the implementation value **SEMVMX**.

NOTES

The **IPC_INFO**, **SEM_STAT** and **SEM_INFO** control calls are used by the **ipcs**(8) program to provide information on allocated resources. In the future these can be modified as needed or moved to a proc file system interface.

The following system limit on semaphore sets affects a **semctl** call:

SEMVMX Maximum value for **semval**: implementation dependent (32767).

CONFORMING TO

SVr4, SVID. SVr4 documents more error conditions EINVAL and EOVERFLOW.

SEE ALSO

ipc(5), **shmget**(2), **shmat**(2), **shmdt**(2).

NAME

shmget − allocates a shared memory segment

SYNOPSIS

#include <sys/ipc.h>

#include <sys/shm.h>

int shmget(key_t *key***, int** *size***, int** *shmflg***);**

DESCRIPTION

shmget() returns the identifier of the shared memory segment associated to the value of the argument *key*. A new shared memory segment, with size equal to the round up of *size* to a multiple of **PAGE_SIZE**, is created if *key* has value **IPC_PRIVATE** or *key* isn't **IPC_PRIVATE**, no shared memory segment is associated to *key*, and **IPC_CREAT** is asserted in *shmflg* (i.e. *shmflg***&IPC_CREAT** isn't zero). The presence in *shmflg* is composed of:

IPC_CREAT
to create a new segment. If this flag is not used, then **shmget()** will find the segment associated with *key*, check to see if the user has permission to receive the *shmid* associated with the segment, and ensure the segment is not marked for destruction.

IPC_EXCL used with **IPC_CREAT** to ensure failure if the segment exists.

mode_flags (lowest 9 bits)
specifying the permissions granted to the owner, group, and world. Presently, the execute permissions are not used by the system.

If a new segment is created, the access permissions from *shmflg* are copied into the *shm_perm* member of the *shmid_ds* structure that defines the segment. The *shmid_ds* structure:

```
struct shmid_ds {
    struct    ipc_perm shm_perm;    /* operation perms */
    int       shm_segsz;            /* size of segment (bytes) */
    time_t    shm_atime;            /* last attach time */
    time_t    shm_dtime;            /* last detach time */
    time_t    shm_ctime;            /* last change time */
    unsigned short    shm_cpid;     /* pid of creator */
    unsigned short    shm_lpid;     /* pid of last operator */
    short     shm_nattch;           /* no. of current attaches */
};

struct ipc_perm
{
  key_t  key;
  ushort uid;   /* owner euid and egid */
  ushort gid;
  ushort cuid;  /* creator euid and egid */
  ushort cgid;
  ushort mode;  /* lower 9 bits of shmflg */
  ushort seq;   /* sequence number */
};
```

Furthermore, while creating, the system call initializes the system shared memory segment data structure **shmid_ds** as follows:

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

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

The lowest order 9 bits of **shm_perm.mode** are set to the lowest order 9 bit of *shmflg*.

shm_segsz is set to the value of *size*.

shm_lpid, **shm_nattch**, **shm_atime** and **shm_dtime** are set to **0.**

shm_ctime is set to the current time.

If the shared memory segment already exists, the access permissions are verified, and a check is made to see if it is marked for destruction.

SYSTEM CALLS

fork() After a **fork()** the child inherits the attached shared memory segments.

exec() After an **exec()** all attached shared memory segments are detached (not destroyed).

exit() Upon **exit()** all attached shared memory segments are detached (not destroyed).

RETURN VALUE

A valid segment identifier, *shmid*, is returned on success, −1 on error.

ERRORS

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

EINVAL is returned if **SHMMIN** > *size* or *size* > **SHMMAX**, or *size* is greater than size of segment.

EEXIST is returned if **IPC_CREAT | IPC_EXCL** was specified and the segment exists.

EIDRM is returned if the segment is marked as destroyed, or was removed.

ENOSPC is returned if all possible shared memory id's have been taken (**SHMMNI**) or if allocating a segment of the requested *size* would cause the system to exceed the system-wide limit on shared memory (**SHMALL**).

ENOENT is returned if no segment exists for the given *key*, and **IPC_CREAT** was not specified.

EACCES is returned if the user does not have permission to access the shared memory segment.

ENOMEM is returned if no memory could be allocated for segment overhead.

NOTES

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 lowest order 9 bits of *shmflg* and creates a new shared memory segment (on success).

The followings are limits on shared memory segment resources affecting a **shmget** call:

SHMALL System wide maximum of shared memory pages: policy dependent.

SHMMAX Maximum size in bytes for a shared memory segment: implementation dependent (currently 4M).

SHMMIN Minimum size in bytes for a shared memory segment: implementation dependent (currently 1 byte, though **PAGE_SIZE** is the effective minimum size).

SHMMNI System wide maximum number of shared memory segments: implementation dependent (currently 4096).

The implementation has no specific limits for the per process maximum number of shared memory segments (**SHMSEG**).

BUGS

Use of **IPC_PRIVATE** doesn't inhibit to other processes the access to the allocated shared memory segment.

As for the files, there is currently no intrinsic way for a process to ensure exclusive access to a shared memory segment. Asserting both **IPC_CREAT** and **IPC_EXCL** in *shmflg* only ensures (on success) that a new shared memory segment will be created, it doesn't imply exclusive access to the segment.

CONFORMING TO

SVr4, SVID. SVr4 documents an additional error condition EEXIST. Neither SVr4 nor SVID documents an EIDRM condition.

SEE ALSO

ftok(3), **ipc**(5), **shmctl**(2), **shmat**(2), **shmdt**(2).

NAME

shmop − shared memory operations

SYNOPSIS

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

char *shmat (int *shmid*, **char *** *shmaddr*, **int** *shmflg* **)**

int shmdt (char * *shmaddr* **)**

DESCRIPTION

The function **shmat** attaches the shared memory segment identified by **shmid** to the data segment of the calling process. The attaching address is specified by *shmaddr* with one of the following criteria:

If *shmaddr* is **0**, the system tries to find an unmapped region in the range 1 − 1.5G starting from the upper value and coming down from there.

If *shmaddr* isn't **0** and **SHM_RND** is asserted in *shmflg*, the attach occurs at address equal to the rounding down of *shmaddr* to a multiple of **SHMLBA**. Otherwise *shmaddr* must be a page aligned address at which the attach occurs.

If **SHM_RDONLY** is asserted in *shmflg*, the segment is attached for reading and the process must have read access permissions to the segment. Otherwise the segment is attached for read and write and the process must have read and write access permissions to the segment. There is no notion of write-only shared memory segment.

The **brk** value of the calling process is not altered by the attach. The segment will automatically detached at process exit. The same segment may be attached as a read and as a read-write one, and more than once, in the process's address space.

On a successful **shmat** call the system updates the members of the structure **shmid_ds** associated to the shared memory segment as follows:

shm_atime is set to the current time.

shm_lpid is set to the process-ID of the calling process.

shm_nattch is incremented by one.

Note that the attach succeeds also if the shared memory segment is marked as to be deleted.

The function **shmdt** detaches from the calling process's data segment the shared memory segment located at the address specified by *shmaddr*. The detaching shared memory segment must be one among the currently attached ones (to the process's address space) with *shmaddr* equal to the value returned by the its attaching **shat** call.

On a successful **shmdt** call the system updates the members of the structure **shmid_ds** associated to the shared memory segment as follows:

shm_dtime is set to the current time.

shm_lpid is set to the process-ID of the calling process.

shm_nattch is decremented by one. If it becomes 0 and the segment is marked for deletion, the segment is deleted.

The occupied region in the user space of the calling process is unmapped.

SYSTEM CALLS

fork() After a **fork()** the child inherits the attached shared memory segments.

exec() After an **exec()** all attached shared memory segments are detached (not destroyed).

exit() Upon **exit()** all attached shared memory segments are detached (not destroyed).

RETURN VALUE

On a failure both functions return **−1** with **errno** indicating the error, otherwise **shmat** returns the address of the attached shared memory segment, and **shmdt** returns **0**.

ERRORS

When **shmat** fails, at return **errno** will be set to one among the following values:

EACCES The calling process has no access permissions for the requested attach type.

EINVAL Invalid *shmid* value, unaligned (i.e., not page-aligned and **SHM_RND** was not specified) or invalid *shmaddr* value, or failing attach at **brk**.

ENOMEM Could not allocate memory for the descriptor or for the page tables.

The function **shmdt** can fails only if there is no shared memory segment attached at *shmaddr*, in such a case at return **errno** will be set to **EINVAL**.

NOTES

On executing a **fork**(2) system call, the child inherits all the attached shared memory segments.

The shared memory segments attached to a process executing an **execve**(2) system call will not be attached to the resulting process.

The following is a system parameter affecting a **shmat** system call:

SHMLBA Segment low boundary address multiple. Must be page aligned. For the current implementation the **SHMBLA** value is **PAGE_SIZE**.

The implementation has no intrinsic limit to the per process maximum number of shared memory segments (**SHMSEG**)

CONFORMING TO

SVr4, SVID. SVr4 documents an additional error condition EMFILE.

SEE ALSO

ipc(5), **shmctl**(2), **shmget**(2).

NAME

shmctl − shared memory control

SYNOPSIS

#include <sys/ipc.h>

#include <sys/shm.h>

int shmctl(int *shmid*, **int** *cmd*, **struct shmid_ds *****buf*);

DESCRIPTION

shmctl() allows the user to receive information on a shared memory segment, set the owner, group, and permissions of a shared memory segment, or destroy a segment. The information about the segment identified by *shmid* is returned in a *shmid_ds* structure:

```
struct shmid_ds {
        struct  ipc_perm shm_perm;      /* operation perms */
        int     shm_segsz;              /* size of segment (bytes) */
        time_t  shm_atime;              /* last attach time */
        time_t  shm_dtime;              /* last detach time */
        time_t  shm_ctime;              /* last change time */
        unsigned short  shm_cpid;       /* pid of creator */
        unsigned short  shm_lpid;       /* pid of last operator */
        short   shm_nattch;             /* no. of current attaches */
        /* the following are private */
        unsigned short  shm_npages; /* size of segment (pages) */
        unsigned long   *shm_pages;
        struct shm_desc *attaches;    /* descriptors for attaches */
};
```

The fields in the member *shm_perm* can be set:

```
struct ipc_perm
{
  key_t  key;
  ushort uid;   /* owner euid and egid */
  ushort gid;
  ushort cuid;  /* creator euid and egid */
  ushort cgid;
  ushort mode;  /* lower 9 bits of access modes */
  ushort seq;   /* sequence number */
};
```

The following *cmds* are available:

IPC_STAT is used to copy the information about the shared memory segment into the buffer *buf*. The user must have **read** access to the shared memory segment.

IPC_SET is used to apply the changes the user has made to the *uid*, *gid*, or *mode* members of the *shm_perms* field. Only the lowest 9 bits of *mode* are used. The *shm_ctime* member is also updated. The user must be the owner, creator, or the super-user.

IPC_RMID is used to mark the segment as destroyed. It will actually be destroyed after the last detach. (I.e., when the *shm_nattch* member of the associated structure *shmid_ds* is zero.) The user must be the owner, creator, or the super-user.

The user *must* ensure that a segment is eventually destroyed; otherwise its pages that were faulted in will remain in memory or swap.

In addition, the **super-user** can prevent or allow swapping of a shared memory segment with the following *cmds*: (Linux only)

SHM_LOCK
prevents swapping of a shared memory segment. The user must fault in any pages that are required to be present after locking is enabled.

SHM_UNLOCK
allows the shared memory segment to be swapped out.

The **IPC_INFO**, **SHM_STAT** and **SHM_INFO** control calls are used by the **ipcs**(8) program to provide information on allocated resources. In the future, these man be modified as needed or moved to a proc file system interface.

SYSTEM CALLS

fork() After a **fork()** the child inherits the attached shared memory segments.

exec() After an **exec()** all attached shared memory segments are detached (not destroyed).

exit() Upon **exit()** all attached shared memory segments are detached (not destroyed).

RETURN VALUE

0 is returned on success, −1 on error.

ERRORS

On error, **errno** will be set to one of the following:

EACCES is returned if **IPC_STAT** is requested and *shm_perm.modes* does not allow read access for *msqid*.

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

EINVAL is returned if *shmid* is not a valid identifier, or *cmd* is not a valid command.

EIDRM is returned if *shmid* points to a removed identifier.

EPERM is returned if **IPC_SET** or **IPC_RMID** is attempted, and the user is not the creator, the owner, or the super-user, and the user does not have permission granted to their group or to the world.

CONFORMING TO

SVr4, SVID. SVr4 documents additional error conditions EINVAL, ENOENT, ENOSPC, ENOMEM, EEXIST. Neither SVr4 nor SVID documents an EIDRM error condition.

SEE ALSO

shmget(2), **shmop**(2)

NAME

ipcs − provide information on ipc facilities

SYNOPSIS

ipcs [−asmq] [−tclup]
ipcs [−smq] −i *id*
ipcs −h

DESCRIPTION

ipcs provides information on the ipc facilities for which the calling process has read acccess.

The **−i** option allows a specific resource *id* to be specified. Only information on this *id* will be printed.

Resources may be specified as follows:

−m shared memory segments

−q message queues

−s semaphore arrays

−a all (this is the default)

The output format may be specified as follows:

−t time

−p pid

−c creator

−l limits

−u summary

SEE ALSO

ipcrm(8)

AUTHOR

krishna balasubramanian (balasub@cis.ohio-state.edu)

NAME

ipcs − provide information on ipc facilities

SYNOPSIS

ipcrm [shm | msg | sem] *id*

DESCRIPTION

ipcrm will remove the resource speccified by *id*.

SEE ALSO

ipcs(8)

AUTHOR

krishna balasubramanian (balasub@cis.ohio-state.edu)