

UNIX-Type APIs (V5R2)

Interprocess Communication (IPC) APIs

Table of Contents

[Interprocess Communication \(IPC\) APIs](#)

- [Identifier Based Services](#)
- [Pointer Based Services](#)
- [Managing IPC Objects](#)
- APIs
 - [ftok\(\)](#) (Generate IPC Key from File Name)
 - [msgctl\(\)](#) (Perform Message Control Operations)
 - [msgget\(\)](#) (Get Message Queue)
 - [msgrcv\(\)](#) (Receive Message Operation)
 - [msgsnd\(\)](#) (Send Message Operation)
 - [QlgFtok\(\)](#) (Generate IPC Key from File Name (using NLS-enabled path name))
 - [QlgSem_open\(\)](#) (Open Named Semaphore (using NLS-enabled path name))
 - [QlgSem_open_np\(\)](#) (Open Named Semaphore with Maximum Value (using NLS-enabled path name))
 - [QlgSem_unlink\(\)](#) (Unlink Named Semaphore (using NLS-enabled path name))
 - [QP0ZDIPC](#) (Delete Interprocess Communication Objects)
 - [QP0ZOLIP](#) (Open List of Interprocess Communication Objects)
 - [QP0ZOLSM](#) (Open List of Semaphores)
 - [QP0ZRIPC](#) (Retrieve an Interprocess Communication Object)
 - [semctl\(\)](#) (Perform Semaphore Control Operations)
 - [semget\(\)](#) (Get Semaphore Set with Key)
 - [semop\(\)](#) (Perform Semaphore Operations on Semaphore Set)
 - [sem_close\(\)](#) (Close Named Semaphore)
 - [sem_destroy\(\)](#) (Destroy Unnamed Semaphore)
 - [sem_getvalue\(\)](#) (Get Semaphore Value)
 - [sem_init\(\)](#) (Initialize Unnamed Semaphore)
 - [sem_init_np\(\)](#) (Initialize Unnamed Semaphore with Maximum Value)
 - [sem_open\(\)](#) (Open Named Semaphore)
 - [sem_open_np\(\)](#) (Open Named Semaphore with Maximum Value)
 - [sem_post\(\)](#) (Post to Semaphore)

- [sem_post_np\(\)](#) (Post Value to Semaphore)
- [sem_trywait\(\)](#) (Try to Decrement Semaphore)
- [sem_unlink\(\)](#) (Unlink Named Semaphore)
- [sem_wait\(\)](#) (Wait for Semaphore)
- [sem_wait_np\(\)](#) (Wait for Semaphore with Timeout)
- [shmat\(\)](#) (Attach Shared Memory Segment to Current Process)
- [shmctl\(\)](#) (Perform Shared Memory Control Operations)
- [shmdt\(\)](#) (Detach Shared Memory Segment from Calling Process)
- [shmget\(\)](#) (Get ID of Shared Memory Segment with Key)

[Header Files for UNIX-Type Functions](#)

[Errno Values for UNIX-Type Functions](#)

Interprocess Communication (IPC) APIs

Interprocess communication (IPC) on OS/400 is made up of five services divided into the two categories of identifier-based services and pointer-based services. The identifier-based IPC services consist of message queues, semaphore sets, and shared memory. The pointer-based services consist of unnamed and named semaphores. The basic purpose of these services is to provide OS/400 processes and threads with a way to communicate with each other through a set of standard APIs. These functions are based on the definitions in the Single UNIX Specification.

For additional information on the Interprocess Communication APIs, see:

- [Identifier Based Services](#)
 - [Message Queues](#)
 - [Semaphore Sets](#)
 - [Shared Memory](#)

- [Pointer Based Services](#) (Named and Unnamed Semaphores)

- [Managing IPC Objects](#)

The interprocess communication functions and what they do are:

- [ftok\(\)](#) (Generate IPC Key from File Name) generates an IPC key based on the combination of path and id.
- [msgctl\(\)](#) (Perform Message Control Operations) provides message control operations as specified by cmd on the message queue specified by msqid.
- [msgget\(\)](#) (Get Message Queue) returns the message queue identifier associated with the parameter key.
- [msgrcv\(\)](#) (Receive Message Operation) reads a message from the queue associated with the message queue identifier specified by msqid and places it in the user-defined buffer pointed to by msgp.
- [msgsnd\(\)](#) (Send Message Operation) is used to send a message to the queue associated with the message queue identifier specified by msqid.
- [QlgFtok\(\)](#) (Generate IPC Key from File Name (using NLS-enabled path name)) generates an IPC key based on the combination of path and id.
- [QlgSem_open\(\)](#) (Open Named Semaphore (using NLS-enabled path name)) opens a named semaphore and returns a semaphore pointer that may be used on subsequent calls to [sem_post\(\)](#), [sem_post_np\(\)](#), [sem_wait\(\)](#), [sem_wait_np\(\)](#), [sem_trywait\(\)](#), [sem_getvalue\(\)](#), and [sem_close\(\)](#).
- [QlgSem_open_np\(\)](#) (Open Named Semaphore with Maximum Value (using NLS-enabled path name)) opens a named semaphore and returns a semaphore pointer that may be used on subsequent calls to [sem_post\(\)](#), [sem_post_np\(\)](#), [sem_wait\(\)](#), [sem_wait_np\(\)](#), [sem_trywait\(\)](#), [sem_getvalue\(\)](#), and [sem_close\(\)](#).
- [QlgSem_unlink\(\)](#) (Unlink Named Semaphore (using NLS-enabled path name)) unlinks a named semaphore.
- [QPOZDIPC](#) (Delete Interprocess Communication Objects) deletes one or more interprocess communication (IPC) objects as specified by the delete control parameter.
- [QPOZOLIP](#) (Open List of Interprocess Communication Objects) lets you generate a list of

interprocess communication (IPC) objects and descriptive information based on the selection parameters.

- [QP0ZOLSM](#) (Open List of Semaphores) lets you generate a list of description information about the semaphores within a semaphore set.
- [QP0ZRIPC](#) (Retrieve an Interprocess Communication Object) lets you generate detailed information about a single interprocess communication (IPC) object.
- [semctl\(\)](#) (Perform Semaphore Control Operations) provides semaphore control operations as specified by cmd on the semaphore specified by semnum in the semaphore set specified by semid.
- [semget\(\)](#) (Get Semaphore Set with Key) returns the semaphore ID associated with the specified semaphore key.
- [semop\(\)](#) (Perform Semaphore Operations on Semaphore Set) performs operations on semaphores in a semaphore set. These operations are supplied in a user-defined array of operations.
- [sem_close\(\)](#) (Close Named Semaphore) closes a named semaphore that was previously opened by a thread of the current process using [sem_open\(\)](#) or [sem_open_np\(\)](#).
- [sem_destroy\(\)](#) (Destroy Unnamed Semaphore) destroys an unnamed semaphore that was previously initialized using [sem_init\(\)](#) or [sem_init_np\(\)](#).
- [sem_getvalue\(\)](#) (Get Semaphore Value) retrieves the value of a named or unnamed semaphore.
- [sem_init\(\)](#) (Initialize Unnamed Semaphore) initializes an unnamed semaphore and sets its initial value.
- [sem_init_np\(\)](#) (Initialize Unnamed Semaphore with Maximum Value) initializes an unnamed semaphore and sets its initial value.
- [sem_open\(\)](#) (Open Named Semaphore) opens a named semaphore, returning a semaphore pointer that may be used on subsequent calls to [sem_post\(\)](#), [sem_post_np\(\)](#), [sem_wait\(\)](#), [sem_wait_np\(\)](#), [sem_trywait\(\)](#), [sem_getvalue\(\)](#), and [sem_close\(\)](#).
- [sem_open_np\(\)](#) (Open Named Semaphore with Maximum Value) opens a named semaphore, returning a semaphore pointer that may be used on subsequent calls to [sem_post\(\)](#), [sem_post_np\(\)](#), [sem_wait\(\)](#), [sem_wait_np\(\)](#), [sem_trywait\(\)](#), [sem_getvalue\(\)](#), and [sem_close\(\)](#).
- [sem_post\(\)](#) (Post to Semaphore) posts to a semaphore, incrementing its value by one.
- [sem_post_np\(\)](#) (Post Value to Semaphore) posts to a semaphore, incrementing its value by the increment specified in the options parameter.
- [sem_trywait\(\)](#) (Try to Decrement Semaphore) attempts to decrement the value of the semaphore.
- [sem_unlink\(\)](#) (Unlink Named Semaphore) unlinks a named semaphore.
- [sem_wait\(\)](#) (Wait for Semaphore) decrements by one the value of the semaphore.
- [sem_wait_np\(\)](#) (Wait for Semaphore with Timeout) attempts to decrement by one the value of the semaphore.
- [shmat\(\)](#) (Attach Shared Memory Segment to Current Process) returns the address of the shared memory segment associated with the specified shared memory identifier.
- [shmctl\(\)](#) (Perform Shared Memory Control Operations) provides shared memory control operations as specified by cmd on the shared memory segment specified by shmid.
- [shmdt\(\)](#) (Detach Shared Memory Segment from Calling Process) detaches the shared memory segment specified by shmaddr from the calling process.
- [shmget\(\)](#) (Get ID of Shared Memory Segment with Key) returns the shared memory ID associated with the specified shared memory key.

Note: These functions use header (include) files from the library QSYSINC, which is optionally installable. Make sure QSYSINC is installed on your system before using any of the functions. See [Header Files for UNIX-Type Functions](#) for the file and member name of each header file.

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

Identifier Based Services

Although each IPC service provides a specific type of interprocess communication, the three identifier based services share many similarities. Each service defines a mechanism through which its communications take place. For message queues, that mechanism is a message queue; for semaphore sets, it is a semaphore set; and for shared memory, it is a shared memory segment. These mechanisms are identified by a unique positive integer called, respectively, a message queue identifier (*msqid*), a semaphore set identifier (*semid*), and a shared memory identifier (*shmid*).

Note: Throughout the Interprocess Communication APIs, the term thread is used extensively. This does not mean that IPC objects can be used only between threads within one process, but rather that authorization checks and waits occur for the calling thread within a process.

Associated with each identifier is a data structure that contains state information for the IPC mechanism, as well as ownership and permissions information. The ownership and permissions information is defined in a structure in the `<sys/ipc.h>` header file as follows:

```
typedef struct ipc_perm {
    uid_t      uid;      /* Owner's user ID          */
    gid_t      gid;      /* Owner's group ID         */
    uid_t      cuid;     /* Creator's user ID        */
    gid_t      cgid;     /* Creator's group ID       */
    mode_t     mode;     /* Access modes             */
} ipc_perm_t;
```

This structure is similar to a file permissions structure, and is initialized by the thread that creates the IPC mechanism. It is then checked by all subsequent IPC operations to determine if the requesting thread has the required permissions to perform the operation.

To get an identifier, a thread must either create a new IPC mechanism or access an existing mechanism. This is done through the `msgget()`, `semget()`, and `shmget()` functions. Each get operation takes as input a *key* parameter and returns an identifier. Each get operation also takes a *flag* parameter. This *flag* parameter contains the IPC permissions for the mechanism as well as bits that determine whether or not a new mechanism is created. The rules for whether a new mechanism is created or an existing one is referred to are as follows:

- Specifying a *key* of `IPC_PRIVATE` guarantees a new mechanism is created.
- Setting the `IPC_CREAT` bit in the *flag* parameter creates a new mechanism for the specified *key* if one does not already exist. If an existing mechanism is found, its identifier is returned.
- Setting both `IPC_CREAT` and `IPC_EXCL` creates a new mechanism for the specified *key* only if a mechanism does not already exist. If an existing mechanism is found, an error is returned.

When a message queue, semaphore set, or shared memory segment is created, the thread that creates it determines how it can be accessed. The thread does this by passing mode information in the low-order 9 bits of the *flag* parameter on the `msgget()`, `semget()`, or `shmget()` function call. This information is used to initialize the mode field in the `ipc_perm` structure. The values of the bits are given below in hexadecimal notation:

Bit	Meaning
<code>X'0100'</code>	Read by user
<code>X'0080'</code>	Write by user

X'0020' Read by group
X'0010' Write by group
X'0004' Read by others
X'0002' Write by others

Subsequent IPC operations do a permission test on the calling thread before allowing the thread to perform the requested operation. This permission test is done in one of three forms:

- For the **msgget()**, **semget()**, or **shmget()** calls that are accessing an existing IPC mechanism, the caller's *flag* parameter is checked to make sure it does not specify access bits that are not in the mode field of the existing IPC mechanism's `ipc_perm` structure. If the *flag* parameter does not contain any bits that are not in the mode field, permission is granted.
- For most of the other IPC APIs, the effective user ID and effective group ID of the thread are retrieved, and these values are compared with the data in the `ipc_perm` structure as follows:
 - If the effective user ID equals either the `uid` or the `cuid` field for the IPC mechanism, and if the appropriate access bit is on in the mode field (either Read by user or Write by user, depending on the operation being requested), permission is granted.
 - If the effective group ID equals either the `gid` or the `cgid` field for the IPC mechanism, and if the appropriate access bit is on in the mode field (either Read by group or Write by group), permission is granted.
 - If none of the above tests are true, and if the appropriate access bit is on for others (either Read by others or Write by others), permission is granted.
- For the **msgctl()**, **semctl()**, or **shmctl()** APIs, some values of the *cmd* parameter require the caller to be the owner or creator of the IPC object, or have appropriate privileges. The values of *cmd* that this rule applies to depends on the API. This is shown in the API descriptions for **msgctl()**, **semctl()**, and **shmctl()**.

Message Queues

Message queues provide a form of message passing in which any process (given that it has the necessary permissions) can read a message from or write a message to any IPC message queue on the system. There are no requirements that a process be waiting to receive a message from a queue before another process sends one, or that a message exist on the queue before a process requests to receive one.

Every message on a queue has the following attributes:

- Message type
- Message length (length of data part of message)
- Message data (if length is greater than 0)

A thread gets a message queue identifier by calling the **msgget()** function. Depending on the *key* and *msgflg* parameters passed in, either a new message queue is created or an existing message queue is accessed. When a new message queue is created, a data structure is also created to contain information about the message queue. This structure is defined in the `<sys/msg.h>` header file as follows:

```

typedef struct msqid_ds {
    struct ipc_perm    msg_perm;    /* Operation permission struct */
    msgqnum_t         msg_qnum;    /* # msgs currently on queue */
    msglen_t          msg_qbytes;  /* Max # bytes allowed on queue*/
    pid_t             msg_lspid;   /* Process ID of last msgsnd() */
    pid_t             msg_lrpid;   /* Process ID of last msgrcv() */
    time_t            msg_stime;   /* Time of last msgsnd() */
    time_t            msg_rtime;   /* Time of last msgrcv() */
    time_t            msg_ctime;   /* Time of last change */
} msqid_ds_t;

```

A thread puts a message on a message queue by calling the **msgsnd()** function. The following parameters are passed in:

- Message queue ID
- Pointer to a buffer containing the message type and message data
- Length of the message
- Flag that specifies whether or not the thread is willing to wait to send the message.

A thread gets a message from a message queue by calling the **msgrcv()** function. The following parameters are passed in:

- Message queue ID
- Pointer to a buffer in which to receive the message
- Length of the buffer
- Type of message
- Flag that specifies whether or not the thread is willing to wait and whether or not the thread is willing to truncate a message to receive it

A thread removes a message queue ID by calling the **msgctl()** function. The thread also can use the **msgctl()** function to change the data structure values associated with the message queue ID or to retrieve the data structure values associated with the message queue ID. The following parameters are passed in:

- Message queue ID
- Command the thread wants to perform (remove ID, set data structure values, receive data structure values)
- Pointer to a buffer from which to set data structure values or in which to receive data structure values

Message Queue Differences and Restrictions

OS/400 message queues differ from the message queue definition in the Single UNIX Specification in the following ways:

- The maximum message size is 65535 bytes.
- The maximum number of bytes on a message queue is 16 777 216.
- The maximum number of message queues that can be created (system-wide) is 2 147 483 646.

The message queue functions are:

- [ftok\(\)](#) (Generate IPC Key from File Name) generates an IPC key based on the combination of path

and id.

- [`msgctl\(\)`](#) (Perform Message Control Operations) provides message control operations as specified by `cmd` on the message queue specified by `msqid`.
- [`msgget\(\)`](#) (Get Message Queue) returns the message queue identifier associated with the parameter `key`.
- [`msgrcv\(\)`](#) (Receive Message Operation) reads a message from the queue associated with the message queue identifier specified by `msqid` and places it in the user-defined buffer pointed to by `msgp`.
- [`msgsnd\(\)`](#) (Send Message Operation) is used to send a message to the queue associated with the message queue identifier specified by `msqid`.
- [`QlgFtok\(\)`](#) (Generate IPC Key from File Name (using NLS-enabled path name)) generates an IPC key based on the combination of path and id.

Semaphore Sets

A **semaphore** is a synchronization mechanism similar to a mutex or a machine interface (MI) lock. It can be used to control access to shared resources, or used to notify other threads of the availability of resources. It differs from a mutex in the following ways:

- A semaphore set is not a single value, but has a set of values. It is referred to through a **semaphore set** containing multiple semaphores. Each semaphore set is identified by a *semid*, which identifies the semaphore set, and a *semnum*, which identifies the semaphore within the set. Multiple semaphore operations may be specified on one **semop()** call. These operations are atomic on multiple semaphores within a semaphore set.
- Semaphore values can range from 0 to 65535.
- Semaphores have **permissions** associated with them. A thread must have appropriate authorities to perform an operation on a semaphore.
- A semaphore can have a **semaphore adjustment value** associated with it. This value represents resource allocations which can be automatically undone by the system when the thread ends, representing the releasing of resources. The adjustment value can range from -32767 to 32767.

Thus, a semaphore can be used as a resource counter or as a lock.

A process gets a semaphore set identifier by calling the **semget()** function. Depending on the *key* and *semflg* parameters passed in, either a new semaphore set is created or an existing semaphore set is accessed. When a new semaphore set is created, a data structure is also created to contain information about the semaphore set. This structure is defined in the `<sys/sem.h>` header file as follows:

```
typedef struct semid_ds {
    struct ipc_perm  sem_perm; /* Permissions structure */
    unsigned short  sem_nsems; /* Number of sems in set */
    time_t          sem_otime; /* Last sem op time */
    time_t          sem_ctime; /* Last change time */
} semtabentry_t;
```

A thread performs operations on one or more of the semaphores in a set by calling the **semop()** function. The following parameters are passed in:

- Semaphore ID
- Pointer to an array of sembuf structures
- Number of sembuf structures in the array.

The sembuf structure is defined in the `<sys/sem.h>` header file as follows:

```
struct sembuf {
    unsigned short sem_num; /* Semaphore number */
    short          sem_op;  /* Semaphore operation */
    short          sem_flg; /* Operation flags */
};
```

The operation performed on a semaphore is specified by the `sem_op` field, which can be positive, negative, or zero:

- If `sem_op` is positive, the value of `sem_op` is added to the semaphore's current value.
- If `sem_op` is zero, the caller will wait until the semaphore's value becomes zero.
- If `sem_op` is negative, the caller will wait until the semaphore's value is greater than or equal to the absolute value of `sem_op`. Then the absolute value of `sem_op` is subtracted from the semaphore's current value.

The `sem_flg` value specifies whether or not the thread is willing to wait, and also whether or not the thread wants the system to keep a semaphore adjustment value for the semaphore.

Semaphore waits are visible from the Work with Active Jobs display. A thread waiting on a semaphore in a semaphore set appears to be in a semaphore wait state (SEMW) on the Work with Threads display (requested using the `WRKJOB` command and taking option 20). Displaying the call stack of the thread shows the `semop()` function near the bottom of the stack.

A thread removes a semaphore set ID by calling the `semctl()` function. The thread also can use the `semctl()` function to change the data structure values associated with the semaphore set ID or to retrieve the data structure values associated with the semaphore set ID. The following parameters are passed in:

- Semaphore set ID
- Command the thread wants to perform (remove ID, set data structure values, receive data structure values),
- Pointer to a buffer from which to set data structure values, or in which to receive data structure values.

In addition, the `semctl()` function can perform various other control operations on a specific semaphore within a set, or on an entire semaphore set:

- Set or retrieve a semaphore value.
- Retrieve the process ID of the last thread to operate on a semaphore.
- Retrieve the number of threads waiting for a semaphore value to increase.
- Retrieve the number of threads waiting for a semaphore value to become zero.
- Retrieve the value of every semaphore in a semaphore set.
- Set the value of every semaphore in a semaphore set.

Semaphore Set Differences and Restrictions

OS/400 semaphore sets differ from the definition in the Single UNIX Specification in the following ways:

- The Single UNIX Specification does not define threads. Consequently, Single UNIX Specification semaphores are defined in terms of processes and the semaphore:
 - Causes the entire process to wait
 - Releases resources when the process ends

OS/400 handles semaphores at the thread level. An OS/400 semaphore:

- Causes only a single thread to wait
 - Releases resources when the thread ends
- The maximum number of semaphore sets that can be created (system-wide) is 2 147 483 646.
 - The maximum number of semaphores per semaphore set is 65535.
 - Semaphore values are limited to the range from 0 to 65535. Adjustment values associated with a semaphore are limited to the range -32767 to 32767.

The semaphore set functions are:

- [ftok\(\)](#) (Generate IPC Key from File Name) generates an IPC key based on the combination of path and id.
- [QlgFtok\(\)](#) (Generate IPC Key from File Name (using NLS-enabled path name)) generates an IPC key based on the combination of path and id.
- [semctl\(\)](#) (Perform Semaphore Control Operations) provides semaphore control operations as specified by cmd on the semaphore specified by semnum in the semaphore set specified by semid.
- [semget\(\)](#) (Get Semaphore Set with Key) returns the semaphore ID associated with the specified semaphore key.
- [semop\(\)](#) (Perform Semaphore Operations on Semaphore Set) performs operations on semaphores in a semaphore set. These operations are supplied in a user-defined array of operations.

Shared Memory

Processes and threads can communicate directly with one another by sharing parts of their memory space and then reading and writing the data stored in the **shared memory**. Synchronization of shared memory is the responsibility of the application program. Semaphores and mutexes provide ways to synchronize shared memory use across processes and threads.

A thread gets a shared memory identifier by calling the **shmget()** function. Depending on the *key* and *shmflg* parameters passed in, either a new shared memory segment is created or an existing shared memory segment is accessed. The size of the shared memory segment is specified by the *size* parameter. When a new shared memory segment is created, a data structure is also created to contain information about the shared memory segment. This structure is defined in the `<sys/shm.h>` header file as follows:

```
typedef struct shmid_ds {
    struct ipc_perm shm_perm; /* Operation permission struct */
    int shm_segsz; /* Segment size */
    pid_t shm_lpid; /* Process id of last shmop */
    pid_t shm_cpid; /* Process id of creator */
    int shm_nattch; /* Current # attached */
}
```

```

    time_t      shm_atime; /* Last shmat time          */
    time_t      shm_dtime; /* Last shmdt time         */
    time_t      shm_ctime; /* Last change time        */
} shmtableentry_t;

```

A process gets addressability to the shared memory segment by attaching to it using the **shmat()** function. The following parameters are passed in:

- Shared memory ID
- Pointer to an address
- Flag specifying how the shared memory segment is to be attached

A process detaches a shared memory segment by calling the **shmdt()** function. The only parameter passed in is the shared memory segment address. The process implicitly detaches from the shared memory when the process ends.

A thread removes a shared memory ID by calling the **shmctl()** function. The thread also can use the **shmctl()** function to change the data structure values associated with the shared memory ID or to retrieve the data structure values associated with the shared memory ID. The following parameters are passed in:

- Shared memory ID
- Command the thread wants to perform (remove ID, set data structure values, receive data structure values)
- Pointer to a buffer from which to set data structure values, or in which to receive data structure values.

Shared Memory Differences and Restrictions

Shared memory segments are created as teraspace-shared memory segments or as nonteraspace-shared memory segments. A teraspace shared memory segment is accessed by adding the shared memory segment to a process's teraspace. A teraspace is a space that has a much larger capacity than other OS/400 spaces and is addressable from only one process. A nonteraspace shared memory segment creates shared memory using OS/400 space objects.

A teraspace shared memory segment is created if SHM_TS_NP is specified on the *shmflag* parameter of **shmget()** or if a shared memory segment is created from a program that was compiled using the TERASPACE(*YES *TSIFC) option of CRTBNDC or CRTCMOD. The following capabilities and restrictions apply for teraspace shared memory segments.

- Teraspace shared memory objects may be attached in read-only mode.
- The address specified by *shmaddr* is only used when **shmat()** is called from a program that uses data model LLP64 and attaches to a teraspace shared memory segment. Otherwise it is not possible to specify the address in teraspace at which the shared memory is to be mapped. The *shmaddr* parameter on the **shmat()** function is ignored.
- After a teraspace shared memory segment is detached, it cannot be addressed through a pointer saved by the process.
- The maximum size of a teraspace shared memory segment is 4 294 967 295 bytes (4 GB minus 1).
- The maximum number of shared memory segments that can be created (system-wide) is 2 147 483 646.

- A teraspace shared memory segment may be created such that its size can be changed after it is created. The maximum size of this type of shared memory segment is 268 435 456 bytes (256 MB).

The OS/400 nonteraspace shared memory differs from the shared memory definition in the Single UNIX Specification in the following ways:

- The nonteraspace shared memory segments are OS/400 space objects and can be attached only in read/write mode, not in the read-only mode that the Single UNIX Specification allows. If the SHM_RDONLY flag is specified in the *shmflg* parameter on a **shmget()** call, the call fails and the *errno* variable is set to [EOPNOTSUPP].
- A nonteraspace shared memory segment can be attached only at the actual address of the OS/400 space object, not at an address specified by the thread. The *shmaddr* parameter on the **shmat()** function is ignored.
- After a nonteraspace shared memory segment is detached from a process, it still can be addressed through a pointer saved by the process. For nonteraspace shared memory segments, OS/400 does not "map" and "unmap" regions of storage to the address space of a process.
- The maximum size of a nonteraspace shared memory segment is 16 776 960 bytes. Although the maximum size of a shared memory segment is 16 776 960 bytes, shared memory segments larger than 16 773 120 bytes should be created as teraspace shared memory segments. When the operating system accesses a nonteraspace shared memory segment that has a size larger than 16 773 120 bytes, a performance degradation may be observed.
- The maximum number of shared memory segments that can be created (system-wide) is 2 147 483 646.
- The size of a nonteraspace shared memory segment may be changed using the SHM_RESIZE command of **shmctl()**, up to a maximum size of 16 773 120 bytes.

The shared memory functions are:

- [ftok\(\)](#) (Generate IPC Key from File Name) generates an IPC key based on the combination of path and id.
- [QlgFtok\(\)](#) (Generate IPC Key from File Name (using NLS-enabled path name)) generates an IPC key based on the combination of path and id.
- [shmat\(\)](#) (Attach Shared Memory Segment to Current Process) returns the address of the shared memory segment associated with the specified shared memory identifier.
- [shmctl\(\)](#) (Perform Shared Memory Control Operations) provides shared memory control operations as specified by cmd on the shared memory segment specified by shmid.
- [shmdt\(\)](#) (Detach Shared Memory Segment from Calling Process) detaches the shared memory segment specified by shmaddr from the calling process.
- [shmget\(\)](#) (Get ID of Shared Memory Segment with Key) returns the shared memory ID associated with the specified shared memory key.

Pointer Based Services

The pointer based services consist of named and unnamed semaphores. The named and unnamed semaphores on OS/400 differ from the other IPC mechanisms in that they do not have an IPC identifier associated with them. Instead, pointers to the semaphore are used to operate on the semaphore. Before using a semaphore, a process must obtain a pointer to the semaphore. Unlike a semaphore set, a named or unnamed semaphore refers to a single semaphore only. A semaphore contains a value, a maximum value, and a title.

There are two types of semaphores: named semaphores and unnamed semaphores. Once a semaphore is created and a pointer to the semaphore is obtained, the same operations are used to manipulate the values of both types of semaphores. Like the semaphores in a semaphore set, a named or unnamed semaphore has a nonzero value. A semaphore can be used as a resource counter or as a lock. A thread decrements a semaphore to obtain one or more associated resources, and increments the semaphore to release the resource. A semaphore also has a maximum value associated with it. An attempt to increment the value of a semaphore above its maximum value results in an error.

Besides a value, named and unnamed semaphores contain a maximum value and a title. The maximum value sets the highest value that the semaphore value may obtain. The title is a null-terminated string with a maximum size of 16 characters that are associated with the semaphore and may be used to contain debugging information. The titles associated with named and unnamed semaphores may be obtained by using the **QP0ZOLIP()** API.

A process obtains a pointer to a named semaphore by calling the **sem_open()** or **sem_open_np()** functions. These functions find the semaphore associated with a name. The name is a character string, interpreted in the CCSID of the job. The name may be structured so that it looks like a pathname. This name, however, has no relationship to any file system. If the semaphore exists and the process has permission to use the semaphore, then the system allocates memory for the semaphore and returns a pointer to the caller. If the semaphore does not exist, it will be created if the appropriate flags are set. When a new named semaphore is created, the permissions of the semaphore are set using the information provided by the *mode* parameter. These permissions are the same as those used by the identifier- based IPC services. The **sem_open_np()** function permits the caller to set the maximum value and title of a semaphore when creating a named semaphore. When a process is finished using a named semaphore, it should call **sem_close()** to close the semaphore. The semaphore is also explicitly closed when a process terminates. When a named semaphore will no longer be needed, it can be removed from the system using **sem_unlink()**.

A process obtains a pointer to an unnamed semaphore calling the **sem_init()** or **sem_init_np()** functions. These functions initialize a semaphore at the specified memory location. The **sem_init_np()** function permits the caller to set the maximum value and title of a unnamed semaphore when it is created. When a process is finished using an unnamed semaphore, it should call **sem_destroy()** to destroy the semaphore and release system resources associated with that semaphore.

A process decrements by one the value of a semaphore using the **sem_wait()** and **sem_wait_np()** functions. If the value of the semaphore is currently zero, then the thread is blocked until the value of the semaphore is incremented or until the time specified on the **sem_wait_np()** has expired. The **sem_trywait()** call may be used to decrement the value of the semaphore if it is greater than zero. If the current value of the semaphore is zero, then **sem_trywait()** will return an error. The **sem_post()** and **sem_post_np()** functions are used to increment the value of a semaphore. After the value of the semaphore is incremented, it may be decremented immediately by threads that have blocked trying to decrement the semaphore.

Named and unnamed semaphore waits are visible from the Work with Active Jobs display. A thread waiting on a named or unnamed semaphore will be in a semaphore wait state (SEMW).

The **sem_getvalue()** function returns the value of the semaphore if the value is greater than or equal to zero. If there are threads waiting on the semaphore, **sem_getvalue()** returns a negative number whose absolute value is the number of threads waiting on the semaphore.

For details on the semaphore functions, see the following:

- [OlgSem_open\(\)](#) (Open Named Semaphore (using NLS-enabled path name)) opens a named semaphore and returns a semaphore pointer that may be used on subsequent calls to `sem_post()`, `sem_post_np()`, `sem_wait()`, `sem_wait_np()`, `sem_trywait()`, `sem_getvalue()`, and `sem_close()`.
- [OlgSem_open_np\(\)](#) (Open Named Semaphore with Maximum Value (using NLS-enabled path name)) opens a named semaphore and returns a semaphore pointer that may be used on subsequent calls to `sem_post()`, `sem_post_np()`, `sem_wait()`, `sem_wait_np()`, `sem_trywait()`, `sem_getvalue()`, and `sem_close()`.
- [OlgSem_unlink\(\)](#) (Unlink Named Semaphore (using NLS-enabled path name)) unlinks a named semaphore.
- [sem_close\(\)](#) (Close Named Semaphore) closes a named semaphore that was previously opened by a thread of the current process using `sem_open()` or `sem_open_np()`.
- [sem_destroy\(\)](#) (Destroy Unnamed Semaphore) destroys an unnamed semaphore that was previously initialized using `sem_init()` or `sem_init_np()`.
- [sem_getvalue\(\)](#) (Get Semaphore Value) retrieves the value of a named or unnamed semaphore.
- [sem_init\(\)](#) (Initialize Unnamed Semaphore) initializes an unnamed semaphore and sets its initial value.
- [sem_init_np\(\)](#) (Initialize Unnamed Semaphore with Maximum Value) initializes an unnamed semaphore and sets its initial value.
- [sem_open\(\)](#) (Open Named Semaphore) opens a named semaphore, returning a semaphore pointer that may be used on subsequent calls to `sem_post()`, `sem_post_np()`, `sem_wait()`, `sem_wait_np()`, `sem_trywait()`, `sem_getvalue()`, and `sem_close()`.
- [sem_open_np\(\)](#) (Open Named Semaphore with Maximum Value) opens a named semaphore, returning a semaphore pointer that may be used on subsequent calls to `sem_post()`, `sem_post_np()`, `sem_wait()`, `sem_wait_np()`, `sem_trywait()`, `sem_getvalue()`, and `sem_close()`.
- [sem_post\(\)](#) (Post to Semaphore) posts to a semaphore, incrementing its value by one.
- [sem_post_np\(\)](#) (Post Value to Semaphore) posts to a semaphore, incrementing its value by the increment specified in the options parameter.
- [sem_trywait\(\)](#) (Try to Decrement Semaphore) attempts to decrement the value of the semaphore.
- [sem_unlink\(\)](#) (Unlink Named Semaphore) unlinks a named semaphore.
- [sem_wait\(\)](#) (Wait for Semaphore) decrements by one the value of the semaphore.
- [sem_wait_np\(\)](#) (Wait for Semaphore with Timeout) attempts to decrement by one the value of the semaphore.

Managing IPC Objects

Interprocess communication objects can be managed with the following APIs. The QP0ZOLIP API opens a list of message queue, shared memory, semaphore set, named semaphore or unnamed semaphore objects by type, by owner, by creator, or by key. The QP0ZOLSM API opens a list of semaphores in a semaphore set. Both APIs return a handle that can be used to get list entries with the QGYGTLE API, find entries by number with the QGYFNDE API, or close the list with the QGYCLST API.

The QP0ZRIPC API retrieves detailed information about message queue, shared memory, or semaphore set objects. The QP0ZDIPC API deletes message queue, shared memory, or semaphore set objects.

The IPC object management APIs are:

- [QP0ZDIPC](#) (Delete Interprocess Communication Objects) deletes one or more interprocess communication (IPC) objects as specified by the delete control parameter.
- [QP0ZOLIP](#) (Open List of Interprocess Communication Objects) lets you generate a list of interprocess communication (IPC) objects and descriptive information based on the selection parameters.
- [QP0ZOLSM](#) (Open List of Semaphores) lets you generate a list of description information about the semaphores within a semaphore set.
- [QP0ZRIPC](#) (Retrieve an Interprocess Communication Object) lets you generate detailed information about a single interprocess communication (IPC) object.

ftok()--Generate IPC Key from File Name

Syntax

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

key_t ftok(const char *path, int id);
```

Service Program Name: QP0ZCPA

Default Public Authority: *USE

Threadsafe: Conditional; see [Usage Notes](#).

The **ftok()** function generates an IPC key based on the combination of *path* and *id*.

Identifier-based interprocess communication facilities require you to supply a key to the **msgget()**, **semget()**, **shmget()** subroutines to obtain interprocess communication identifiers. The **ftok()** function is one mechanism to generate these keys.

If the values for *path* and *id* are the same as a previous call to **ftok()** and the file named by *path* was not deleted and re-created in between calls to **ftok()**, **ftok()** will return the same key.

The **ftok()** function returns different keys if different values of *path* and *id* are used.

Only the low-order 8-bits of *id* are significant. The remaining bits are ignored by **ftok()**.

Parameters

path

(Input) The path name of the file used in combination with *id* to generate the key.

See [QlgFtok--Generate IPC Key from File Name \(using NLS-enabled path name\)](#) for a description and an example of supplying the *path* in any CCSID.

id

(Input) The integer identifier used in combination with *path* to generate the key. Only the low order 8-bits of *id* are significant. The remaining bits will be ignored.

Authorities

Authorization Required for ftok() (excluding QOPT)

Object Referred to	Authority Required	errno
Each directory in the path name preceding the object	*X	EACCES

Object	None	None
--------	------	------

Authorization Required for `ftok()` in the QOPT File System

Object Referred to	Authority Required	errno
Volume containing directory or object	*USE	EACCES
Directory or object within volume	None	None

Return Value

value `ftok()` was successful.

(key_t)-1 `ftok()` was not successful. The *errno* variable is set to indicate the error.

Error Conditions

If `ftok()` is not successful, *errno* indicates one of the following errors.

[EACCES]

Permission denied.

An attempt was made to access an object in a way forbidden by its object access permissions.

The thread does not have access to the specified file, directory, component, or path.

If you are accessing a remote file through the Network File System, update operations to file permissions at the server are not reflected at the client until updates to data that is stored locally by the Network File System take place. (Several options on the Add Mounted File System (ADDMFS) command determine the time between refresh operations of local data.) Access to a remote file may also fail due to different mappings of user IDs (UID) or group IDs (GID) on the local and remote systems.

[EAGAIN]

Operation would have caused the process to be suspended.

[EBADFID]

A file ID could not be assigned when linking an object to a directory.

The file ID table is missing or damaged.

To recover from this error, run the Reclaim Storage (RCLSTG) command as soon as possible.

[EBADNAME]

The object name specified is not correct.

[EBUSY]

Resource busy.

An attempt was made to use a system resource that is not available at this time.

[ECONVERT]

Conversion error.

One or more characters could not be converted from the source CCSID to the target CCSID.

[EDAMAGE]

A damaged object was encountered.

A referenced object is damaged. The object cannot be used.

[EFAULT]

The address used for an argument is not correct.

In attempting to use an argument in a call, the system detected an address that is not valid.

While attempting to access a parameter passed to this function, the system detected an address that is not valid.

[EFILECVT]

File ID conversion of a directory failed.

Try to run the Reclaim Storage (RCLSTG) command to recover from this error.

[EINTR]

Interrupted function call.

[EINVAL]

The value specified for the argument is not correct.

A function was passed incorrect argument values, or an operation was attempted on an object and the operation specified is not supported for that type of object.

An argument value is not valid, out of range, or NULL.

[EIO]

Input/output error.

A physical I/O error occurred.

A referenced object may be damaged.

[ELOOP]

A loop exists in the symbolic links.

This error is issued if the number of symbolic links encountered is more than POSIX_SYMLINK_MAX (defined in the limits.h header file). Symbolic links are encountered during resolution of the directory or path name.

[ENAMETOOLONG]

A path name is too long.

A path name is longer than `PATH_MAX` characters or some component of the name is longer than `NAME_MAX` characters while `_POSIX_NO_TRUNC` is in effect. For symbolic links, the length of the name string substituted for a symbolic link exceeds `PATH_MAX`. The `PATH_MAX` and `NAME_MAX` values can be determined using the `pathconf()` function.

[ENOENT]

No such path or directory.

The directory or a component of the path name specified does not exist.

A named file or directory does not exist or is an empty string.

[ENOMEM]

Storage allocation request failed.

A function needed to allocate storage, but no storage is available.

There is not enough memory to perform the requested function.

[ENOSPC]

No space available.

The requested operations required additional space on the device and there is no space left. This could also be caused by exceeding the user profile storage limit when creating or transferring ownership of an object.

Insufficient space remains to hold the intended file, directory, or link.

[ENOTDIR]

Not a directory.

A component of the specified path name existed, but it was not a directory when a directory was expected.

Some component of the path name is not a directory, or is an empty string.

[ENOTSAFE]

Function is not allowed in a job that is running with multiple threads.

[ENOTSUP]

Operation not supported.

The operation, though supported in general, is not supported for the requested object or the requested arguments.

[EPERM]

Operation not permitted.

You must have appropriate privileges or be the owner of the object or other resource to do the requested operation.

[EROOBJ]

Object is read only.

You have attempted to update an object that can be read only.

[ESTALE]

File or object handle rejected by server.

If you are accessing a remote file through the Network File System, the file may have been deleted at the server.

[EUNKNOWN]

Unknown system state.

The operation failed because of an unknown system state. See any messages in the job log and correct any errors that are indicated, then retry the operation.

Usage Notes

1. This function will fail with error code [ENOTSAFE] when both of the following conditions occur:
 - Where multiple threads exist in the job.
 - The object this function is operating on resides in a file system that is not threadsafe. Only the following file systems are threadsafe for this function:
 - Root
 - QOpenSys
 - User-defined
 - QNTC
 - QSYS.LIB
 - >>Independent ASP QSYS.LIB<<
 - QOPT
2. If the values for *path* and *id* are the same as a previous call to **ftok()** and if the file named by *path* was deleted and re-created in between calls to **ftok()**, **ftok()** will return a different key.
3. The **ftok()** function will return the same key for different values of *path* if the path names refer to symbolic links or hard links whose target files are the same.
4. The **ftok()** function may return the same key for different values of *path* if the target files are in different file systems.
5. The **ftok()** function may return the same key for different values of *path* if the target file is in a file system that contains more than 2^{24} files.

Related Information

- [QlgFtok--Generate IPC Key from File Name \(using NLS-enabled path name\)](#)
- [msgget\(\)-Get Message Queue](#)

- [semget\(\)-Get Semaphore Set with Key](#)
- [shmget\(\)-Get ID of Shared Memory Segment with Key](#)

Example

The following example uses **ftok()** and **semget()** functions.

```
#include <sys/ipc.h>
#include <sys/sem.h>
#include <errno.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    key_t myKey;
    int    semid;

    /* Use ftok to generate a key associated with a file. */
    /* Every process will get the same key back if the */
    /* caller calls with the same parameters.          */
    myKey = ftok("/myApplication/myFile", 42);
    if(myKey == -1) {
        printf("ftok failed with errno = %d\n", errno);
        return -1;
    }

    /* Call an xxxget() API, where xxx is sem, shm, or msg. */
    /* This will create or reference an existing IPC object */
    /* with the 'well known' key associated with the file   */
    /* name used above.                                     */
    semid = semget(myKey, 1, 0666 | IPC_CREAT);
    if(semid == -1) {
        printf("semget failed with errno = %d\n", errno);
        return -1;
    }

    /* ... Use the semaphore as required ... */
    return 0;
}
```

API introduced: V4R3

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

msgctl()-Perform Message Control Operations

Syntax

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

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

Service Program Name: QP0ZUSHR

Default Public Authority: *USE

Threadsafe: Yes

The **msgctl()** function provides message control operations as specified by **cmd** on the message queue specified by **msqid**.

Parameters

msqid

(Input) Message queue identifier, a positive integer. It is created by the **msgget()** function and used to identify the message queue on which to perform the control operation.

cmd

(Input) Command, the control operation to perform on the message queue.

buf

(I/O) Pointer to the message queue data structure to be used to get or set message queue information.

The **cmd** parameter can have one of the following values:

IPC_STAT Put the current value of each member of the **msqid_ds** data structure associated with **msqid** into the structure pointed to by **buf**.

IPC_SET Set the value of the following members of the `msqid_ds` data structure associated with `msqid` to the corresponding value found in the structure pointed to by `buf`:

- `msg_perm.uid`
- `msg_perm.gid`
- `msg_perm.mode`
- `msg_qbytes`

`IPC_SET` can be run only by a thread with appropriate privileges or one that has an effective user ID equal to the value of `msg_perm.cuid` or `msg_perm.uid` in the `msqid_ds` data structure associated with `msqid`. Only a thread with appropriate privileges can raise the value of `msg_qbytes`.

IPC_RMID Remove the message queue identifier specified by `msqid` from the system and destroy the message queue and `msqid_ds` data structure associated with it. `IPC_RMID` can be run only by a thread with appropriate privileges or one that has an effective user ID equal to the value of `msg_perm.cuid` or `msg_perm.uid` in the `msqid_ds` data structure associated with `msqid`. The structure pointed to by `buf` is ignored and can be `NULL`.

Authorities

Figure 1-4. Authorization Required for `msgctl()`

Object Referred to	Authority Required	errno
Message queue for which state information is retrieved (<code>cmd = IPC_STAT</code>)	Read	EACCES
Message queue for which state information is set (<code>cmd = IPC_SET</code>)	See Note	EPERM
Message queue to be removed (<code>cmd = IPC_RMID</code>)	See Note	EPERM

Note: To set message queue information or to remove a message queue, the thread must be the owner or creator of the queue, or have appropriate privileges. To raise the value of `msg_qbytes`, a thread must have appropriate privileges.

Return Value

0 `msgctl()` was successful.

-1 `msgctl()` was not successful. The `errno` variable is set to indicate the error.

Error Conditions

If `msgctl()` is not successful, `errno` usually indicates one of the following errors. Under some conditions, `errno` could indicate an error other than those listed here.

[EACCES]

Permission denied.

An attempt was made to access an object in a way forbidden by its object access permissions.

The thread does not have access to the specified file, directory, component, or path.

If you are accessing a remote file through the Network File System, update operations to file permissions at the server are not reflected at the client until updates to data that is stored locally by the Network File System take place. (Several options on the Add Mounted File System (ADDMFS) command determine the time between refresh operations of local data.) Access to a remote file may also fail due to different mappings of user IDs (UID) or group IDs (GID) on the local and remote systems.

The parameter `cmd` is `IPC_STAT`, but the calling thread does not have read permission.

[EDAMAGE]

A damaged object was encountered.

A referenced object is damaged. The object cannot be used.

The message queue has been damaged by a previous message queue operation.

[EFAULT]

The address used for an argument is not correct.

In attempting to use an argument in a call, the system detected an address that is not valid.

While attempting to access a parameter passed to this function, the system detected an address that is not valid.

[EINVAL]

An invalid parameter was found.

A parameter passed to this function is not valid.

One of the following has occurred:

- The value of `cmd` is either `IPC_SET` or `IPC_STAT` and the value of `buf` is `NULL`.
- The value of `msqid` is not a valid message queue identifier.
- The value of `cmd` is not a valid command.
- The value of `cmd` is `IPC_SET` and the value of `msg_qbytes` exceeds the system-imposed limit.

[EPERM]

Operation not permitted.

You must have appropriate privileges or be the owner of the object or other resource to do the requested operation.

The parameter `cmd` is equal to `IPC_RMID` or `IPC_SET` and both of the following are true:

- the caller does not have the appropriate privileges.
- the effective user ID of the caller is not equal to the value of `msg_perm.cuid` or `msg_perm.uid` in the data structure associated with `msgid`.

The parameter `cmd` is `IPC_SET` and an attempt is being made to increase the value of `msg_qbytes`, but the caller does not have appropriate privileges.

[EUNKNOWN]

Unknown system state.

The operation failed because of an unknown system state. See any messages in the job log and correct any errors that are indicated, then retry the operation.

Error Messages

None.

Usage Notes

"Appropriate privileges" is defined to be `*ALLOBJ` special authority. If the user profile under which the thread is running does not have `*ALLOBJ` special authority, the caller does not have appropriate privileges.

Related Information

- The `<sys/msg.h>` file (see [Header Files for UNIX-Type Functions](#))
- [msgget\(\)-Get Message Queue](#)
- [msgrcv\(\)-Receive Message Operation](#)
- [msgsnd\(\)-Send Message Operation](#)

Example

The following example performs a control operation on a message queue:

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

main() {
    int msqid;
```

```
int rc;  
struct msqid_ds buf;  
  
rc = msgctl(msqid, IPC_STAT, &buf);  
}
```

API introduced: V3R6

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

msgget()-Get Message Queue

Syntax

```
#include <sys/msg.h>
#include <sys/stat.h>

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

Service Program Name: QP0ZUSHR

Default Public Authority: *USE

Threadsafe: Yes

The **msgget()** function returns the message queue identifier associated with the parameter key.

Parameters

key

(Input) Key associated with the message queue. Specifying a key of `IPC_PRIVATE` guarantees that a unique message queue is created. A key also can be generated by the caller or by calling the **ftok()** function.

msgflg

(Input) Operations and permissions flag.

The value of `msgflg` is either 0 or is obtained by performing an OR operation on one or more of the following constants:

<i>'0x0100' or S_IRUSR</i>	Permits the creator of the message queue to read it
<i>'0x0080' or S_IWUSR</i>	Permits the creator of the message queue to write it
<i>'0x0020' or S_IRGRP</i>	Permits the group associated with the message queue to read it
<i>'0x0010' or S_IWGRP</i>	Permits the group associated with the message queue to write it
<i>'0x0004' or S_IROTH</i>	Permits others to read the message queue
<i>'0x0002' or S_IWOTH</i>	Permits others to write the message queue
<i>'0x0200' or IPC_CREAT</i>	Creates the message queue if it does not exist already
<i>'0x0400' or IPC_EXCL</i>	Causes msgget() to fail if <code>IPC_CREAT</code> is set and the message queue already exists

Authorities

Figure 1-5. Authorization Required for msgget()

Object Referred to	Authority Required	errno
Message queue to be created	None	None
Existing message queue to be accessed	See Note	EACCES

Note: If the thread is accessing an existing message queue, the mode specified in the last 9 bits of msgflg must be a subset of the mode of the existing message queue.

Return Value

- value* **msgget()** was successful. The value returned is the message queue ID associated with the key parameter.
- 1* **msgget()** was not successful. The errno variable is set to indicate the error.

Error Conditions

If **msgget()** is not successful, errno usually indicates one of the following errors. Under some conditions, errno could indicate an error other than those listed here.

[EACCES]

Permission denied.

An attempt was made to access an object in a way forbidden by its object access permissions.

The thread does not have access to the specified file, directory, component, or path.

If you are accessing a remote file through the Network File System, update operations to file permissions at the server are not reflected at the client until updates to data that is stored locally by the Network File System take place. (Several options on the Add Mounted File System (ADDMFS) command determine the time between refresh operations of local data.) Access to a remote file may also fail due to different mappings of user IDs (UID) or group IDs (GID) on the local and remote systems.

A message queue identifier exists for the parameter key, but operation permission as specified by the low-order 9 bits of msgflg would not be granted.

[EDAMAGE]

A damaged object was encountered.

A referenced object is damaged. The object cannot be used.

The message queue has been damaged by a previous message queue operation.

[EEXIST]

File exists.

The file specified already exists and the specified operation requires that it not exist.

The named file, directory, or path already exists.

A message queue identifier exists for the parameter key, but $((\text{msgflg} \& \text{IPC_CREAT}) \&\& (\text{msgflg} \& \text{IPC_EXCL}))$ is not zero. (& is a bitwise AND; && is a logical AND.)

[ENOENT]

No such path or directory.

The directory or a component of the path name specified does not exist.

A named file or directory does not exist or is an empty string.

A message queue identifier does not exist for the parameter key, and $(\text{msgflg} \& \text{IPC_CREAT})$ is zero. (& is a bitwise AND.)

[ENOSPC]

No space available.

The requested operations required additional space on the device and there is no space left. This could also be caused by exceeding the user profile storage limit when creating or transferring ownership of an object.

Insufficient space remains to hold the intended file, directory, or link.

A message queue identifier is to be created, but the system-imposed limit on the maximum number of allowed message queue identifiers system-wide would be exceeded.

[EUNKNOWN]

Unknown system state.

The operation failed because of an unknown system state. See any messages in the job log and correct any errors that are indicated, then retry the operation.

Error Messages

None.

Usage Notes

1. A message queue identifier, associated message queue, and data structure (see the `<sys/msg.h>` header file) are created for the parameter key if one of the following is true:
 2. ○ The parameter key is equal to `IPC_PRIVATE`.
 - The parameter key does not already have a message queue identifier associated with it and $(\text{msgflg} \& \text{IPC_CREAT})$ is not zero.
3. On creation, the data structure associated with the new message queue identifier is initialized as

follows:

- `msg_perm.cuid` and `msg_perm.uid` are set equal to the effective user ID of the calling thread.
- `msg_perm.cgid` and `msg_perm.gid` are set equal to the effective group ID of the calling thread.
- The low-order 9 bits of `msg_perm.mode` are set equal to the low-order 9 bits of `msgflg`.
- `msg_qnum`, `msg_lspid`, `msg_lrpid`, `msg_stime`, and `msg_rtime` are set equal to 0.
- `msg_ctime` is set equal to the current time.
- `msg_qbytes` is set equal to the system limit.

Related Information

- The `<sys/msg.h>` file (see [Header Files for UNIX-Type Functions](#))
- [ftok\(\)--Generate IPC Key from File Name](#)
- [msgctl\(\)--Perform Message Control Operations](#)
- [msgrcv\(\)--Receive Message Operation](#)
- [msgsnd\(\)--Send Message Operation](#)

Example

The following example creates a message queue:

```
#include <sys/msg.h>
#include <sys/stat.h>

main() {
    int msgflg = 0;
    int msqid;

    msqid = msgget(IPC_PRIVATE, msgflg | IPC_CREAT | S_IRUSR | S_IWUSR);
}
```

API introduced: V3R6

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

msgrcv()--Receive Message Operation

Syntax

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

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

Service Program Name: QP0ZUSHR

Default Public Authority: *USE

Threadsafe: Yes

The **msgrcv()** function reads a message from the queue associated with the message queue identifier specified by **msqid** and places it in the user-defined buffer pointed to by **msgp**.

Parameters

msqid

(Input) Message queue identifier from which the message will be received.

msgp

(Output) Pointer to a buffer in which the received message will be stored. See the details below on the structure of the user-defined buffer.

msgsz

(Input) Length of the data portion of the buffer.

msgtyp

(Input) Type of message to be received.

msgflg

(Input) Action to be taken if a message of the desired type is not on the queue, or if the data portion of the message to be received is larger than **msgsz**.

The parameter **msgp** points to a user-defined buffer that must contain the following:

1. A field of type `long int` that will specify the type of the message.
2. A data part that will hold the data bytes of the message.

The following structure is an example of what this user-defined buffer might look like:

```
struct mymsg {
```



```

    long int    mtype;    /* message type */
    char        mtext[1]; /* message text */
}

```

The structure member `mtype` is the type of the received message, as specified by the sending thread. The structure member `mtext` is the text of the message.

The parameter `msgtyp` specifies the type of message requested as follows:

- If `msgtyp` is equal to zero, the first message on the queue is received.
- If `msgtyp` is greater than zero, the first message of type `msgtyp` is received.
- If `msgtyp` is less than zero, the first message of the lowest type that is less than or equal to the absolute value of `msgtyp` is received.

The parameter `msgsz` should include any bytes inserted by the compiler for padding or alignment purposes. These bytes are part of the message data and affect the total number of bytes in the message queue.

The following example shows pad data and how it affects the size of a message:

```

struct mymsg {
    long int    mtype;    /* 12 bytes padding inserted after */
    char        *pointer; /* the mtype field by the compiler.*/
    char        c;        /* 15 bytes padding inserted after */
    char        *pointer2; /* the c field by the compiler. */
} msg;
/* After the mtype field, there are*/
/* 33 bytes of user data, but 60 */
/* bytes of data including padding.*/
msgsz = sizeof(msg) - sizeof(long int); /* 60 bytes. */

```

Authorities

Figure 1-6. Authorization Required for `msgrcv()`

Object Referred to	Authority Required	errno
Message queue from which message is received	Read	EACCES

Return Value

value **`msgrcv()`** was successful. The value returned is the number of bytes of data placed in the buffer `mtext`.

-1 **`msgrcv()`** was not successful. The `errno` variable is set to indicate the error.

Error Conditions

If `msgrev()` is not successful, `errno` usually indicates one of the following errors. Under some conditions, `errno` could indicate an error other than those listed here.

[E2BIG]

Argument list too long.

The size in bytes of `mtext` is greater than `msgsz` and `(msgflg & MSG_NOERROR)` is equal to zero. (& is a bitwise AND.)

[EACCES]

Permission denied.

An attempt was made to access an object in a way forbidden by its object access permissions.

The thread does not have access to the specified file, directory, component, or path.

If you are accessing a remote file through the Network File System, update operations to file permissions at the server are not reflected at the client until updates to data that is stored locally by the Network File System take place. (Several options on the Add Mounted File System (ADDMFS) command determine the time between refresh operations of local data.) Access to a remote file may also fail due to different mappings of user IDs (UID) or group IDs (GID) on the local and remote systems.

The calling thread does not have read permission.

[EDAMAGE]

A damaged object was encountered.

A referenced object is damaged. The object cannot be used.

The message queue has been damaged by a previous message queue operation.

[EFAULT]

The address used for an argument is not correct.

In attempting to use an argument in a call, the system detected an address that is not valid.

While attempting to access a parameter passed to this function, the system detected an address that is not valid.

[EIDRM]

ID has been removed.

The message queue identifier `msqid` was removed from the system.

[EINTR]

Interrupted function call.

The function `msgrev()` was interrupted by a signal.

[EINVAL]

An invalid parameter was found.

A parameter passed to this function is not valid.

One of the following has occurred:

- The value of msgp is NULL.
- The value of msqid is not a valid message queue identifier.

[ENOMSG]

Message does not exist.

The queue does not contain a message of the desired type and (msgflg & IPC_NOWAIT) is not zero.

[EUNKNOWN]

Unknown system state.

The operation failed because of an unknown system state. See any messages in the job log and correct any errors that are indicated, then retry the operation.

Error Messages

None.

Usage Notes

1. The parameter msgsz specifies the size in bytes of mtext. The received message is truncated to msgsz bytes if it is larger than msgsz and (msgflg & MSG_NOERROR) is not zero. The truncated part of the message is lost and no indication of the truncation is given to the calling thread.
2. The parameter msgflg specifies the action to be taken if a message of the desired type is not on the queue. These actions are as follows:
 - If (msgflg & IPC_NOWAIT) is not zero, the calling thread will return immediately with a return value of -1 and errno set to [ENOMSG].
 - If (msgflg & IPC_NOWAIT) is zero, the calling thread suspends processing until one of the following occurs:
 - A message of the desired type is placed on the queue.
 - The message queue identifier msqid is removed from the system. When this occurs, errno is set to [EIDRM] and a value of -1 is returned.
 - The calling thread receives a signal that is to be caught. In this case, a message is not received and the calling thread resumes processing in the manner prescribed in **sigaction()**.
3. The **msgrcv()** function does not tag message data with a CCSID (coded character set identifier)

value. If a CCSID value is required to correctly interpret the message data, it is the responsibility of the caller to include the CCSID value as part of the data.

4. On successful completion, the following actions are taken with respect to the data structure associated with `msqid`:
 - `msg_qnum` is decremented by 1.
 - `msg_lrpid` is set to the process ID of the calling thread.
 - `msg_rtime` is set to the current time.
5. If the `msgrcv()` function does not complete successfully, the requested message is not removed from the message queue.

Related Information

- The `<sys/msg.h>` file (see [Header Files for UNIX-Type Functions](#))
- [msgctl\(\)-Perform Message Control Operations](#)
- [msgget\(\)-Get Message Queue](#)
- [msgsnd\(\)-Send Message Operation](#)

Example

The following example receives a message from a message queue:

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

main() {
    int msqid = 0;
    int msgflg = 0;
    int rc;
    size_t msgsz;
    long int msgtyp;
    struct mymsg {
        long int mtype;
        char      mtext[256];
    };

    msgsz = 256;
    msgtyp = 1;
    rc = msgrcv(msqid, &mymsg, msgsz, msgtyp, msgflg | IPC_NOWAIT);
}
```

API introduced: V3R6

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

msgsnd()-Send Message Operation

Syntax

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

int msgsnd(int msqid, const void *msgp,
           size_t msgsz, int msgflg);
```

Service Program Name: QP0ZUSHR

Default Public Authority: *USE

Threadsafe: Yes

The **msgsnd()** function is used to send a message to the queue associated with the message queue identifier specified by **msqid**.

Parameters

msqid

(Input) Message queue identifier to which the message will be sent.

msgp

(Input) Pointer to the message to be sent.

msgsz

(Input) Length of the data part of the message to be sent.

msgflg

(Input) Action to be taken if the message cannot be immediately placed on the queue.

The parameter **msgp** points to a user-defined buffer that must contain the following:

1. A field of type long int that will specify the type of the message.
2. A data part that will hold the data bytes of the message.

The following structure is an example of what this user-defined buffer might look like:

```
struct mymsg {
    long int    mtype;    /* message type */
    char       mtext[1]; /* message text */
}
```

The structure member **mtype** is a long int that is greater than zero. It can be used by the receiving thread for message selection. The structure member **mtext** is any text of length **msgsz** bytes. The parameter **msgsz** can

range from zero to a system-imposed maximum.

The parameter `msgsz` should include any bytes inserted by the compiler for padding or alignment purposes. These bytes are part of the message data and affect the total number of bytes in the message queue.

The following example shows pad data and how it affects the size of a message:

```
struct mymsg {
    long int    mtype;        /* 12 bytes padding inserted after */
    char        *pointer;    /* the mtype field by the compiler.*/
    char        c;          /* 15 bytes padding inserted after */
    char        *pointer2;   /* the c field by the compiler.   */
} msg;
/* After the mtype field, there are*/
/* 33 bytes of user data, but 60  */
/* bytes of data including padding.*/
msgsz = sizeof(msg) - sizeof(long int);    /* 60 bytes.  */
```

Authorities

Figure 1-7. Authorization Required for `msgsnd()`

Object Referred to	Authority Required	errno
Message queue on which message is placed	Write	EACCES

Return Value

0 `msgsnd()` was successful.

-1 `msgsnd()` was not successful. The `errno` variable is set to indicate the error.

Error Conditions

If `msgsnd()` is not successful, `errno` usually indicates one of the following errors. Under some conditions, `errno` could indicate an error other than those listed here.

[EACCES]

Permission denied.

An attempt was made to access an object in a way forbidden by its object access permissions.

The thread does not have access to the specified file, directory, component, or path.

If you are accessing a remote file through the Network File System, update operations to file permissions at the server are not reflected at the client until updates to data that is stored locally by the Network File System take place. (Several options on the Add Mounted File System (ADDMFS))

command determine the time between refresh operations of local data.) Access to a remote file may also fail due to different mappings of user IDs (UID) or group IDs (GID) on the local and remote systems.

The calling thread does not have write permission.

[EAGAIN]

Operation would have caused the process to be suspended.

The message cannot be sent for one of the reasons cited above and (msgflg & IPC_NOWAIT) is not zero. (& is a bitwise AND.)

[EDAMAGE]

A damaged object was encountered.

A referenced object is damaged. The object cannot be used.

The message queue has been damaged by a previous message queue operation.

[EFAULT]

The address used for an argument is not correct.

In attempting to use an argument in a call, the system detected an address that is not valid.

While attempting to access a parameter passed to this function, the system detected an address that is not valid.

[EIDRM]

ID has been removed.

The message queue identifier msgqid was removed from the system.

[EINTR]

Interrupted function call.

The function **msgsnd()** was interrupted by a signal.

[EINVAL]

An invalid parameter was found.

A parameter passed to this function is not valid.

One of the following has occurred:

- The value of msgp is NULL.
- The value of msgqid is not a valid message queue identifier.
- The value of mtype is less than or equal to zero.
- The value of msgsz is greater than the system-imposed limit.

[EUNKNOWN]

Unknown system state.

The operation failed because of an unknown system state. See any messages in the job log and

correct any errors that are indicated, then retry the operation.

Error Messages

None.

Usage Notes

1. The parameter `msgflg` specifies the action to be taken if the number of bytes already on the queue is equal to `msg_qbytes` (see [Message Queues](#) or the `<sys/msg.h>` header file). The possible actions are as follows:
 - If `(msgflg & IPC_NOWAIT)` is not zero, the message is not sent. The calling thread will return immediately with a return value of -1 and `errno` set to `[EAGAIN]`.
 - If `(msgflg & IPC_NOWAIT)` is zero, the calling process suspends processing until one of the following occurs:
 - The condition responsible for the suspension no longer exists, in which case the message is sent.
 - The message queue identifier `msqid` is removed from the system. When this occurs, `errno` is set to `[EIDRM]` and a value of -1 is returned.
 - The calling thread receives a signal that is to be caught. In this case, a message is not sent and the calling thread resumes processing in the manner prescribed in `sigaction()`.
2. The `msgsnd()` function does not tag message data with a `CCSID` (coded character set identifier) value. If a `CCSID` value is required to correctly interpret the message data, it is the responsibility of the caller to include the `CCSID` value as part of the data.
3. On successful completion, the following actions are taken with respect to the data structure associated with `msqid`:
 - `msg_qnum` is incremented by 1.
 - `msg_lspid` is set to the process ID of the calling thread.
 - `msg_stime` is set to the current time.
4. If the `msgsnd()` function does not complete successfully, the requested message is not placed on the message queue.

Related Information

- The `<sys/msg.h>` file (see [Header Files for UNIX-Type Functions](#))
- [msgctl\(\)-Perform Message Control Operations](#)
- [msgget\(\)-Get Message Queue](#)
- [msgrcv\(\)-Receive Message Operation](#)

Example

The following example sends a message to a message queue:

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

main() {
    int msqid = 0;
    int msgflg = 0;
    int rc;
    size_t msgsz;
    struct mymsg {
        long int mtype;
        char      mtext[256];
    };

    msgsz = 256;
    mymsg.mtype = 1;
    rc = msgsnd(msqid, &mymsg, msgsz, msgflg | IPC_NOWAIT);
}
```

API introduced: V3R6

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

QlgFtok()--Generate IPC Key from File Name (using NLS-enabled path name)

Syntax

```
#include <sys/ipc.h>
#include <qlg.h>

key_t QlgFtok(const Qlg_Path_Name_T *path, int id);
```

Service Program Name: QP0ZCPA

Default Public Authority: *USE

Threadsafe: Conditional

The **QlgFtok()** function, like the **ftok()** function, generates an IPC key based on the combination of *path* and *id*. The difference is that the **QlgFtok()** function takes a pointer to a `Qlg_Path_Name_T` structure, while the **ftok()** function takes a pointer to a character string.

Limited information on the *path* parameter is provided here. For more information on the *path* parameter and for a discussion of other parameters, authorities required, returnvalues, and related information, see [ftok\(\)--Generate IPC Key from File Name](#).

Parameters

path

(Input) The path name of the file used in combination with *id* to generate the key. For more information on the `Qlg_Path_Name_T` structure, see [Path name format](#).

Related Information

[ftok\(\)--Generate IPC Key from File Name](#)

Example

The following example uses the **QlgFtok()** and **semget()** functions.

```
#include <sys/ipc.h>
#include <sys/sem.h>
#include <errno.h>
#include <stdio.h>
```

```

#include <qlg.h>

int main(int argc, char *argv[])
{
    key_t myKey;
    int    semid;

    #define mypath "/myApplication/myFile"
    const char US_const[3]= "US";
    const char Language_const[4]="ENU";
    const char Path_Name_Del_const[2]= "/";
    typedef struct pnstruct
    {
        Qlg_Path_Name_T qlg_struct;
        char[100] pn; /* This size must be >= the path */
                       /* name length or be a pointer */
                       /* to the path name. */

    };
    struct pnstruct path;

    /*****
    /* Initialize Qlg_Path_Name_T parameters */
    *****/
    memset((void*)path.name, 0x00, sizeof(struct pnstruct));
    path.qlg_struct.CCSID = 37;
    memcpy(path.qlg_struct.Country_ID,US_const,2);
    memcpy(path.qlg_struct.Language_ID,Language_const,3);
    path.qlg_struct.Path_Type = QLG_CHAR_SINGLE;
    path.qlg_struct.Path_Length = sizeof(mypath)-1;
    memcpy(path.qlg_struct.Path_Name_Delimiter,Path_Name_Del_const,1);
    memcpy(path.pn,mypath,sizeof(mypath));

    /* Use QlgFtok to generate a key associated with a file. */
    /* Every process will get the same key back if the caller */
    /* calls with the same parameters. */
    myKey = QlgFtok((Qlg_Path_Name_T *)path.name, 42);
    if(myKey == -1) {
        printf("QlgFtok failed with errno = %d\n", errno);
        return -1;
    }

    /* Call an xxxget() API, where xxx is sem, shm, or msg. */
    /* This will create or reference an existing IPC object */
    /* with the 'well known' key associated with the file */
    /* name used above. */
    semid = semget(myKey, 1, 0666 | IPC_CREAT);
    if(semid == -1) {
        printf("semget failed with errno = %d\n", errno);
        return -1;
    }

    /* ... Use the semaphore as required ... */
    return 0;
}

```

API introduced: V5R1

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

QlgSem_open()--Open Named Semaphore (using NLS-enabled path name)

Syntax

```
#include <semaphore.h>
#include <qlg.h>

sem_t * QlgSem_open(const Qlg_Path_Name_T *name,
                    int oflag, ...);
```

Service Program Name: QP0ZPSEM

Default Public Authority: *USE

Threadsafe: Yes

The **QlgSem_open()** function, like the **sem_open()** function, opens a named semaphore and returns a semaphore pointer that may be used on subsequent calls to **sem_post()**, **sem_post_np()**, **sem_wait()**, **sem_wait_np()**, **sem_trywait()**, **sem_getvalue()**, and **sem_close()**. The **QlgSem_open()** function takes a pointer to a **Qlg_Path_Name_T** structure, while the **sem_open()** function takes a pointer to a character string that is in the CCSID of the job.

Limited information on the *name* parameter is provided in this API. For additional information on the *name* parameter and a discussion of other parameters, authorities required, return values, and related information, see [sem_open\(\)--Open Named Semaphore](#).

Parameters

name

(Input) A pointer to a **Qlg_Path_Name_T** structure that contains a path name or a pointer to a path name of the semaphore to be opened. For more information on the **Qlg_Path_Name_T** structure, see [Path name format](#).

Error Conditions

If **QlgSem_open()** is not successful, *errno* usually indicates the following error or one of the errors identified in [sem_open\(\)--Open Named Semaphore](#).

[*ECONVERT*] A conversion error for the parameter *name*.

Related Information

- The <qlg.h> file (see [Header Files for UNIX-Type Functions](#))
- [sem_open\(\)](#)--Open Named Semaphore
- [QlgSem_open_np\(\)](#)--Open Named Semaphore with Maximum Value (using NLS-enabled path name)
- [QlgSem_unlink\(\)](#)--Unlink Named Semaphore (using NLS-enabled path name)

Note: All of the related information for [sem_open\(\)](#) applies to [QlgSem_open\(\)](#). See Related Information in [sem_open\(\)](#).

Example

The following example opens the named semaphore "/mysemaphore" and creates the semaphore with an initial value of 10 if it does not already exist. If the semaphore is created, the permissions are set such that only the current user has access to the semaphore.

```
#include <semaphore.h>
#include <qlg.h>
main() {

    sem_t * my_semaphore;
    int rc;

    #define mypath "/mysemaphore"
    const char US_const[3]= "US";
    const char Language_const[4]="ENU";
    const char Path_Name_Del_const[2]= "/";
    typedef struct pnstruct
    {
        Qlg_Path_Name_T qlg_struct;
        char[100] pn; /* This size must be >= the path */
                       /* name length or be a pointer */
                       /* to the path name. */
    };
    struct pnstruct path;

    /******
    /* Initialize Qlg_Path_Name_T parameters */
    /******
    memset((void*)path.name, 0x00, sizeof(struct pnstruct));
    path.qlg_struct.CCSID = 37;
    memcpy(path.qlg_struct.Country_ID,US_const,2);
    memcpy(path.qlg_struct.Language_ID,Language_const,3);
    path.qlg_struct.Path_Type = QLG_CHAR_SINGLE;
    path.qlg_struct.Path_Length = sizeof(mypath)-1;
    memcpy(path.qlg_struct.Path_Name_Delimiter,Path_Name_Del_const,1);
```

```
memcpy(path.pn, mypath, sizeof(mypath));  
  
my_semaphore = QlgSem_open((Qlg_Path_Name_T *)path name,  
                           O_CREAT, S_IRUSR | S_IWUSR, 10);  
  
}
```

API introduced: V5R1

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

QlgSem_open_np()--Open Named Semaphore with Maximum Value (using NLS-enabled path name)

Syntax

```
#include <semaphore.h>
#include <qlg.h>

sem_t * QlgSem_open_np(const Qlg_Path_Name_T *name, int oflag
                      mode_t mode, unsigned int value,
                      sem_attr_np_t * attr);
```

Service Program Name: QP0ZPSEM

Default Public Authority: *USE

Threadsafe: Yes

The **QlgSem_open_np()** function, like the **sem_open_np()** function, opens a named semaphore and returns a semaphore pointer that may be used on subsequent calls to **sem_post()**, **sem_post_np()**, **sem_wait()**, **sem_wait_np()**, **sem_trywait()**, **sem_getvalue()**, and **sem_close()**. The **QlgSem_open_np()** function takes a pointer to a **Qlg_Path_Name_T** structure, while the **sem_open_np()** function takes a pointer to a character string.

Limited information on the *name* parameter is provided in this API. For additional information on the *name* parameter and a discussion of other parameters, authorities required, return values, and related information, see [sem_open_np\(\)--Open Named Semaphore with Maximum Value](#).

Parameters

name

(Input) A pointer to a **Qlg_Path_Name_T** structure that contains a path name or a pointer to a path name of the semaphore to be opened. For more information on the **Qlg_Path_Name_T** structure, see [Path name format](#).

Error Conditions

If **QlgSem_open_np()** is not successful, *errno* usually indicates the following error or one of the errors identified in [sem_open_np\(\)--Open Named Semaphore with Maximum Value](#).

[*ECONVERT*]

A conversion error for the parameter *name*.

Related Information

- The <qlg.h> file (see [Header Files for UNIX-Type Functions](#))
- [sem_open_np\(\)](#)--Open Named Semaphore with Maximum Value
- [QlgSem_open\(\)](#)--Open Named Semaphore (using NLS-enabled path name)
- [QlgSem_unlink\(\)](#)--Unlink Named Semaphore (using NLS-enabled path name)

Note: All of the related information for **sem_open_np()** applies to **QlgSem_open_np()**. See Related Information in [sem_open\(\)](#).

Example

The following example opens the named semaphore "/mysemaphore" and creates the semaphore with an initial value of 10 and a maximum value of 11. The permissions are set such that only the current user has access to the semaphore.

```
#include <semaphore.h>
#include <qlg.h>
main() {

    sem_t * my_semaphore;
    int rc;
    sem_attr_np_t attr;

#define mypath "/mysemaphore"
    const char US_const[3]= "US";
    const char Language_const[4]="ENU";
    const char Path_Name_Del_const[2]= "/";
    typedef struct pnstruct
    {
        Qlg_Path_Name_T qlg_struct;
        char[100] pn; /* This size must be >= the path */
                        /* name length or be a pointer */
                        /* to the path name. */
    };
    struct pnstruct path;

    /******
    /* Initialize Qlg_Path_Name_T parameters */
    /******
    memset((void*)path.name, 0x00, sizeof(struct pnstruct));
    path.qlg_struct.CCSID = 37;
    memcpy(path.qlg_struct.Country_ID,US_const,2);
    memcpy(path.qlg_struct.Language_ID,Language_const,3);
    path.qlg_struct.Path_Type = QLG_CHAR_SINGLE;
```

```
path.qlg_struct.Path_Length = sizeof(mypath)-1;
memcpy(path.qlg_struct.Path_Name_Delimiter, Path_Name_Del_const, 1);
memcpy(path.pn, mypath, sizeof(mypath));

memset(&attr, 0, sizeof(attr));
attr.maxvalue=11;
my_semaphore = QlgSem_open_np((Qlg_Path_Name_T *)path name,
                             O_CREAT|O_EXCL,
                             S_IRUSR | S_IWUSR,
                             10,
                             &attr);
}
```

API introduced: V5R1

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

QlgSem_unlink()--Unlink Named Semaphore (using NLS-enabled path name)

Syntax

```
#include <semaphore.h>
#include <qlg.h>

int QlgSem_unlink(const Qlg_Path_Name_T *name);
```

Service Program Name: QP0ZPSEM

Default Public Authority: *USE

Threadsafe: Yes

The **QlgSem_unlink()** function, like the **sem_unlink()** function, unlinks a named semaphore. The **QlgSem_unlink()** function takes a pointer to a `Qlg_Path_Name_T` structure, while the **sem_unlink()** function takes a pointer to a character string.

Limited information on the *name* parameter is provided in this API. For additional information on the *name* parameter, authorities required, return values, and related information, see [sem_unlink\(\)--Unlink Named Semaphore](#).

Parameters

name

(Input) A pointer a `Qlg_Path_Name_T` structure that contains a path name or a pointer to a path name of the semaphore to be unlinked. For more information on the `Qlg_Path_Name_T` structure, see [Path name format](#).

Error Conditions

If **QlgSem_unlink()** is not successful, *errno* usually indicates the following error or one of the errors identified in [sem_unlink\(\)--Unlink Named Semaphore](#).

[ECONVERT]

A conversion error for the parameter *name*.

Related Information

- The <qlg.h> file (see [Header Files for UNIX-Type Functions](#))
- [sem_unlink\(\)](#)--Unlink Named Semaphore
- [QlgSem_open\(\)](#)--Open Named Semaphore (using NLS-enabled path name)
- [QlgSem_open_np\(\)](#)--Open Named Semaphore with Maximum Value (using NLS-enabled path name)

Note: All of the related information for `sem_unlink()` applies to `QlgSem_unlink()`. See Related Information in [sem_unlink\(\)](#).

Example

The following example unlinks the named semaphore "/mysem".

```
#include <semaphore.h>
#include <qlg.h>

main() {
    int rc;

    #define mypath "/mysem"
    const char US_const[3]= "US";
    const char Language_const[4]="ENU";
    const char Path_Name_Del_const[2]= "/";
    typedef struct pnstruct
    {
        Qlg_Path_Name_T qlg_struct;
        char[100] pn; /* This size must be >= the path */
                        /* name length or be a pointer */
                        /* to the path name. */
    };
    struct pnstruct path;

    /******
    /* Initialize Qlg_Path_Name_T parameters */
    /******
    memset((void*)path.name, 0x00, sizeof(struct pnstruct));
    path.qlg_struct.CCSID = 37;
    memcpy(path.qlg_struct.Country_ID,US_const,2);
    memcpy(path.qlg_struct.Language_ID,Language_const,3);
    path.qlg_struct.Path_Type = QLG_CHAR_SINGLE;
    path.qlg_struct.Path_Length = sizeof(mypath)-1;
    memcpy(path.qlg_struct.Path_Name_Delimiter,Path_Name_Del_const,1);
    memcpy(path.pn,mypath,sizeof(mypath));

    rc = QlgSem_unlink((Qlg_Path_Name_T *)path.name);
```

}

API introduced: V5R1

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

Delete Interprocess Communication Objects (QP0ZDIPC) API

Required Parameter Group:

1	Delete control	Input	Char(*)
2	Error code	I/O	Char(*)

Default Public Authority: *USE


Threadsafe: No

The Delete Interprocess Communication Objects (QP0ZDIPC) API deletes one or more interprocess communication (IPC) objects as specified by the delete control parameter.

Authorities and Locks

Job Authority

The calling thread must be the owner, must be the creator, or must have *ALLOBJ special authority.

For additional information on these authorities, see the [iSeries Security Reference](#)  book.

Required Parameter Group

Delete control

INPUT; CHAR(*)

Information about which IPC objects to delete. For the layout of this structure, see [Delete Control Format](#).

Error code

I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see [Error Code Parameter](#).

Delete Control Format

The following shows the format of the delete control parameter. For detailed descriptions of the fields in the table, see [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Number of objects to delete.
These fields repeat for each object to delete.		CHAR(1)	IPC type
		CHAR(3)	Reserved
		BINARY(4)	Identifier

Field Descriptions

Identifier. A unique IPC identifier that is used to specify which IPC object is to be deleted. The identifier is obtained from calling the APIs `semget()`, `shmget()`, `msgget()`, or `QPOZOLIP`.

IPC type. This value describes the type of IPC object to delete. Possible values follow:

- 1 Delete a semaphore set object.
- 2 Delete a shared memory object.
- 3 Delete a message queue object.

Number of objects to delete. The number of IPC objects in the delete control parameter.

Reserved. A reserved field. These characters must be set to '00'x.

Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF3C90 E	Literal value cannot be changed.
CPF3CF1 E	Error code parameter not valid.
CPFA986 E	&1 IPC objects deleted; &2 IPC object not deleted
CPDA981 D	Not authorized to delete IPC object &1.
CPDA982 D	IPC object &1 does not exist.
CPDA983 D	IPC object &1 is marked as damaged.
CPFA987 E	Delete control not valid.

API introduced: V4R2

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

Open List of Interprocess Communication Objects (QP0ZOLIP) API

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	List information	Output	Char(80)
4	Number of records to return	Input	Binary(4)
5	Format name	Input	Char(8)
6	Filter information	Input	Char(*)
7	Filter format name	Input	Char(8)
8	Error code	I/O	Char(*)

Default Public Authority: *USE

Threadsafe: No

The Open List of Interprocess Communication Objects (QP0ZOLIP) API lets you generate a list of interprocess communication (IPC) objects and descriptive information based on the selection parameters. The QP0ZOLIP API places the specified number of list entries in the receiver variable. You can access additional records by using the Get List Entries (QGYGTLE) API. On successful completion of the QP0ZOLIP API, a handle is returned in the list information parameter. You may use this handle on subsequent calls to the following APIs:

- Get List Entries (QGYGTLE)
- Find Entry Number in List (QGYFNDE)
- Close List (QGYCLST)

You can use the QP0ZOLIP API to:

- Open a list of all IPC objects of a specific type (semaphore sets, message queues, shared memory, named semaphores, or unnamed semaphores).
- Open a list of identifier-based IPC objects (semaphore sets, message queues, or shared memory) of a specific type with a key in a specified range.
- Open a list of identifier-based IPC objects of a specific type that are owned by one or more specified users.
- Open a list of IPC objects of a specific type (semaphore sets, message queues, shared memory, or named semaphores) that were created by one or more specified users.


Only one IPC type (either semaphore sets, message queue, shared memory, named semaphores, or unnamed semaphores) can be returned in one call to this API. The IPC type is determined by the format name parameter.

The records returned by QP0ZOLIP include an information status field that describes the completeness and validity of the information. Be sure to check the information status field before using any other information returned.

Authorities and Locks

Job Authority

Service special authority (*SERVICE) is needed to call this API.

For additional information on this authority, see the [iSeries Security Reference](#)  book.

Required Parameter Group

Receiver variable

OUTPUT; CHAR(*)

The variable that is used to return the IPC object information that you requested.

Length of receiver variable

INPUT; BINARY(4)

The length of the receiver variable.

List information

OUTPUT; CHAR(80)

Information about the list of IPC objects that were opened. For a description of the layout of this parameter, see [Format of Open List Information](#).

Number of records to return

INPUT; BINARY(4)

The number of records in the list to put into the receiver variable.

Format name

INPUT; CHAR(8)

The format of the information to be returned in the receiver variable. This parameter will determine the type of IPC mechanism to open the list for. You must use one of the following format names:

LSST0100 This format is described in [LSST0100 Format](#).

LMSQ0100 This format is described in [LMSQ0100 Format](#).

LSHM0100 This format is described in [LSHM0100 Format](#).

LNSM0100 This format is described in [LNSM0100 Format](#).

LUSM0100 This format is described in [LUSM0100 Format](#).

Filter information

INPUT; CHAR(*)

The information in this parameter is used to filter the list of IPC objects. The format of this variable depends on the filter format name.

Filter format name

INPUT; CHAR(8)

The name of the format that is used to filter the list of IPC objects. You must use one of the following format names:

FIPC0100 This format is described in [FIPC0100 Format](#).

Error code

I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see [Error Code Parameter](#).

FIPC0100 Format

The following shows the format of the filter information for the FIPC0100 format. For detailed descriptions of the field in the table, see [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		
0	0	CHAR(1)	Filter on key
1	1	CHAR(3)	Reserved
4	4	BINARY(4)	Minimum key
8	8	BINARY(4)	Maximum key
12	C	BINARY(4)	Offset to owner profiles array
16	10	BINARY(4)	Number of owner profiles specified
20	14	BINARY(4)	Offset to creator profiles array
24	18	BINARY(4)	Number of creator profiles specified
This field repeats for each owner profile name.		CHAR(10)	Owner profile name
This field repeats for each creator profile name.		CHAR(10)	Creator profile name

Field Descriptions

Creator profile name. The user profile names that created the IPC objects being returned. These values are used only if the number of creator profiles specified field is greater than one. Possible special values follow:

**ALL* IPC objects created by any user profile are added to the list. The rest of the user profiles in the array are ignored.

**CURRENT* IPC objects created by the current user profile are added to the list.

Filter on key. Whether filtering will be done based on the key value of the IPC object. Possible values follow:

0 No filtering is done based on the key value. The values of minimum key field and maximum key field are ignored.

1 Filtering is done based on the values of minimum key field and maximum key field.

Maximum key. The maximum IPC object's key value. Only the IPC objects with a key greater than or equal to the minimum key and less than or equal to the maximum key will be added to the generated list. This value is only used if the filter on key field is set to one.

Minimum key. The minimum IPC object's key value. Only the IPC objects with a key greater than or equal to the minimum key and less than or equal to the maximum key will be added to the generated list. This value is only used if the filter on key field is set to one.

Number of creator profiles specified. The number of creator profiles specified in the creator profile names array. If this value is zero, no filtering is to be done for the creator user profile.

Number of owner profiles specified. The number of owner profiles specified in the owner profile names array. If this value is zero, no filtering is to be done for the owner user profile.

Offset to creator profiles array. The offset in characters (bytes) from the beginning of the filter information to the beginning of the array of creator profiles.

Offset to owner profiles array. The offset in characters (bytes) from the beginning of the filter information to the beginning of the array of owner profiles.

Owner profile name. The user profile names that own the IPC objects being returned. These values are used only if the number of owner profiles specified field is greater than one. Possible special values follow:

**ALL* IPC objects that are owned by any user profile are added to the list. The rest of the user profiles in the array are ignored.

**CURRENT* IPC objects that are owned by the current user profile are added to the list.

Reserved. These characters must be set to '00'x.

LSST0100 Format

This format name is used to return list information for semaphore sets. The following table shows the information returned in each record in the receiver variable for the LSST0100 format. For a detailed description of each field, see [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Identifier
4	4	BINARY(4)	Key
8	8	BINARY(4)	Number of semaphores
12	C	CHAR(1)	Damaged
13	D	CHAR(1)	Owner read permission
14	E	CHAR(1)	Owner write permission
15	F	CHAR(1)	Group read permission
16	10	CHAR(1)	Group write permission
17	11	CHAR(1)	General read permission
18	12	CHAR(1)	General write permission
19	13	CHAR(1)	Authorized to delete
20	14	CHAR(16)	Last semop() date and time
36	24	CHAR(16)	Last administration change date and time
52	34	CHAR(10)	Owner
62	3E	CHAR(10)	Group owner
72	48	CHAR(10)	Creator
82	52	CHAR(10)	Creator's group

LMSQ0100 Format

This format name is used to return list information for message queues. The following table shows the information returned in each record in the receiver variable for the LMSQ0100 format. For a detailed description of each field, see [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Identifier
4	4	BINARY(4)	Key
8	8	CHAR(1)	Damaged
9	9	CHAR(1)	Owner read permission
10	A	CHAR(1)	Owner write permission
11	B	CHAR(1)	Group read permission
12	C	CHAR(1)	Group write permission
13	D	CHAR(1)	General read permission
14	E	CHAR(1)	General write permission
15	F	CHAR(1)	Authorized to delete
16	10	BINARY(4)	Number of messages on queue
20	14	BINARY(4)	Size of all messages on queue
24	18	BINARY(4)	Maximum size of all messages on queue
28	1C	BINARY(4)	Number of threads to receive message

32	20	BINARY(4)	Number of threads to send message
36	24	CHAR(16)	Last msgrcv() date and time
52	34	CHAR(16)	Last msgsnd() date and time
68	44	CHAR(16)	Last administration change date and time
84	54	CHAR(10)	Owner
94	5E	CHAR(10)	Group owner
104	68	CHAR(10)	Creator
114	72	CHAR(10)	Creator's group

LSHM0100 Format

This format name is used to return list information for shared memory. The following table shows the information returned in each record in the receiver variable for the LSHM0100 format. For a detailed description of each field, see [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Identifier
4	4	BINARY(4)	Key
8	8	CHAR(1)	Damaged
9	9	CHAR(1)	Owner read permission
10	A	CHAR(1)	Owner write permission
11	B	CHAR(1)	Group read permission
12	C	CHAR(1)	Group write permission
13	D	CHAR(1)	General read permission
14	E	CHAR(1)	General write permission
15	F	CHAR(1)	Marked to be deleted
16	10	CHAR(1)	Authorized to delete
17	11	CHAR(1)	Teraspace
18	12	CHAR(1)	Resize
19	13	CHAR(1)	Reserved
20	14	BINARY(4)	Segment size
24	18	BINARY(4)	Number attached
28	1C	CHAR(16)	Last shmat() date and time
44	2C	CHAR(16)	Last detach date and time
60	3C	CHAR(16)	Last administration change date and time
76	4C	CHAR(10)	Owner
86	56	CHAR(10)	Group owner
96	60	CHAR(10)	Creator
106	6A	CHAR(10)	Creator's group

LNSM0100 Format

This format name is used to return list information for named semaphores. The following table shows the information returned in each record in the receiver variable for the LNSM0100 format. For a detailed description of each field, see [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Length of entry
4	4	BINARY(4)	Value
8	8	BINARY(4)	Maximum value
12	C	BINARY(4)	Offset to waiting threads
16	10	BINARY(4)	Number of waiting threads
20	14	BINARY(4)	Offset to name
24	18	BINARY(4)	Length of name
28	1C	CHAR(16)	Title
44	2C	CHAR(1)	Marked to be deleted
45	2D	CHAR(1)	Authorized to delete
46	2E	CHAR(10)	Creator
56	38	CHAR(10)	Creator's group
66	42	CHAR(1)	Owner read permission
67	43	CHAR(1)	Owner write permission
68	44	CHAR(1)	Group read permission
69	45	CHAR(1)	Group write permission
70	46	CHAR(1)	General read permission
71	47	CHAR(1)	General write permission
72	48	CHAR(26)	Last sem_post() qualified job identifier
98	62	CHAR(2)	Reserved
100	64	CHAR(16)	Last sem_post() thread identifier
116	74	CHAR(26)	Last sem_wait() qualified job identifier
142	8e	CHAR(2)	Reserved
144	90	CHAR(16)	Last sem_wait() thread identifier
These fields repeat for each thread waiting on the semaphore.		CHAR(26)	Waiting qualified job identifier
		CHAR(2)	Reserved
		CHAR(16)	Waiting thread identifier
This field follows the list of threads waiting on the semaphore.		CHAR(*)	Name of the semaphore

LUSM0100 Format

This format name is used to return list information for unnamed semaphores. The following table shows the information returned in each record in the receiver variable for the LUSM0100 format. For a detailed description of each field, see [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Length of entry
4	4	BINARY(4)	Value
8	8	BINARY(4)	Maximum value
12	C	BINARY(4)	Offset to waiting threads
16	10	BINARY(4)	Number of waiting threads
20	14	BINARY(4)	Reserved
24	18	CHAR(16)	Title
40	28	CHAR(26)	Last sem_post() qualified job identifier
66	42	CHAR(2)	Reserved
68	44	CHAR(16)	Last sem_post() thread identifier
84	54	CHAR(26)	Last sem_wait() qualified job identifier
110	6E	CHAR(2)	Reserved
112	70	CHAR(16)	Last sem_wait() thread identifier
These fields repeat for each thread waiting on the semaphore.		CHAR(26)	Waiting qualified job identifier
		CHAR(2)	Reserved
		CHAR(16)	Waiting thread identifier

Field Descriptions

Authorized to delete. This value determines if the caller has the authority to delete this IPC object. Possible values follow:

- 0 The calling thread cannot delete the IPC object.
- 1 The calling thread can delete the IPC object.

Creator. The name of the user profile that created this IPC object.

Creator's group. The name of the group profile that created this IPC object. A special value can be returned:

- *NONE The creator does not have a group profile.

Damaged. Whether the IPC object has suffered internal damage. Possible values follow:

- 0 The IPC object is not damaged.

1 The IPC object is damaged.

General read permission. Whether any user other than the owner and group owner has read authority to the IPC object. Possible values follow:

0 General read authority is not allowed to the IPC object.

1 General read authority is allowed for the IPC object.

General write permission. Whether if any user other than the owner and group owner has write authority to the IPC object. Possible values follow:

0 General write authority is not allowed to the IPC object.

1 General write authority is allowed to the IPC object.

Group owner. The name of the group profile that owns this IPC object. A special value can be returned:

**NONE* The IPC object does not have a group owner.

Group read permission. Whether the group owner has read authority to the IPC object. Possible values follow:

0 The group owner does not have read authority to the IPC object.

1 The group owner has read authority to the IPC object.

Group write permission. Whether the group owner has write authority to the IPC object. Possible values follow:

0 The group owner does not have write authority to the IPC object.

1 The group owner has write authority to the IPC object.

Identifier. The unique IPC object identifier.

Key. The key of the IPC object. If this value is zero, this IPC object has no key associated with it.

Last administration change date and time. The date and time of the last change to the IPC object for the owner, group owner, or permissions. The 16 characters are:

1 Century, where 0 indicates years 19xx and 1 indicates years 20xx.

2-7 Date, in YYMMDD (year, month, and day) format.

8-13 Time of day, in HHMMSS (hours, minutes, and seconds) format.

14-16 Milliseconds.

Last detach date and time. The date and time of the last detachment from the shared memory segment. If no thread has performed a successful detachment, this value will be set to all zeros. The 16 characters are:

1 Century, where 0 indicates years 19xx and 1 indicates years 20xx.

2-7 Date, in YYMMDD (year, month, and day) format.

8-13 Time of day, in HHMMSS (hours, minutes, and seconds) format.

14-16 Milliseconds.

Last msgrcv() date and time. The date and time of the last successful msgrcv() call. If no thread has performed a successful msgrcv() call, this value will be set to all zeros. The 16 characters are:

1 Century, where 0 indicates years 19xx and 1 indicates years 20xx.

2-7 Date, in YYMMDD (year, month, and day) format.

8-13 Time of day, in HHMMSS (hours, minutes, and seconds) format.

14-16 Milliseconds.

Last msgsnd() date and time. The date and time of the last successful msgsnd() call. If no thread has performed a successful msgsnd() call, this value will be set to all zeros. The 16 characters are:

1 Century, where 0 indicates years 19xx and 1 indicates years 20xx.

2-7 Date, in YYMMDD (year, month, and day) format.

8-13 Time of day, in HHMMSS (hours, minutes, and seconds) format.

14-16 Milliseconds.

Last sem_post() qualified job identifier. The job name, the job user profile, and the job number of the last thread that successfully called sem_post() or sem_post_np() if the job has not ended. The 26 characters are:

1-10 The job name

11-20 The user profile

21-26 The job number

If the thread has ended, then the first 16 characters contain 16 characters that uniquely identify the ended job, followed by 10 spaces. If no thread has used sem_post() to post to the semaphore, then the 26 characters will contain spaces.

Last sem_post() thread identifier. The thread ID of the last thread that successfully called sem_post() or sem_post_np() if the thread has not ended.

Last sem_wait() qualified job identifier. The job name, the job user profile, and the job number of the last thread that returned from a sem_wait(), sem_wait_np(), or sem_wait() call, if the job has not ended. The 26 characters are:

1-10 The job name

11-20 The user profile

21-26 The job number

If the thread has ended, then the first 16 characters contain 16 characters that uniquely identify the ended job, followed by 10 spaces. If no job has used sem_wait() to wait on the semaphore, then the 26 characters will contain spaces.

Last sem_wait() thread identifier. The thread ID of the last thread that returned from a sem_wait(), sem_wait_np(), or sem_wait() call, if the thread has not ended.

Last semop() date and time. The date and time of the last successful semop() call. If no thread has performed a successful semop() call, this value will be set to all zeros. The 16 characters are:

- 1* Century, where 0 indicates years 19xx and 1 indicates years 20xx.
- 2-7* Date, in YYMMDD (year, month, and day) format.
- 8-13* Time of day, in HHMMSS (hours, minutes, and seconds) format.
- 14-16* Milliseconds.

Last shmat() date and time. The date and time of the last successful shmat(). If no thread has performed a successful shmat() call, this value will be set to all zeros. The 16 characters are:

- 1* Century, where 0 indicates years 19xx and 1 indicates years 20xx.
- 2-7* Date, in YYMMDD (year, month, and day) format.
- 8-13* Time of day, in HHMMSS (hours, minutes, and seconds) format.
- 14-16* Milliseconds.

Length of entry. The length of this record in the list.

Length of name. The number of bytes in the name of the semaphore, not including the terminating null character.

Marked to be deleted. Whether the shared memory is marked to be deleted when the number attached becomes zero. Possible values follow:

- 0* The shared memory segment is not marked for deletion.
- 1* The shared memory segment is marked for deletion.

Maximum size of all messages on queue. The maximum byte size of all messages that can be on the queue at one time.

Maximum value. The maximum value of the semaphore.

Name of the semaphore. The null-terminated name of the semaphore.

Number attached. The number of times any thread has done a shmat() without doing a detach.

Number of messages on queue. The number of messages that are currently on the message queue.

Number of semaphores. The number of semaphores in the semaphore set.

Number of threads to receive message. The number of threads that are currently waiting to receive a message.

Number of threads to send message. The number of threads that are currently waiting to send a message.

Number of waiting threads. The total number of threads that are waiting for this semaphore to reach a certain value.

Offset to name. The offset to where the name field begins.

Offset to waiting threads. The offset to where the fields containing waiting threads begin.

Owner. The name of the user profile that owns this IPC object.

Owner read permission. Whether the owner has read authority to the IPC object. Possible values follow:

0 The owner does not have read authority to the IPC object.

1 The owner has read authority to the IPC object.

Owner write permission. Whether the owner has write authority to the IPC object. Possible values follow:

0 The owner does not have write authority to the IPC object.

1 The owner has write authority to the IPC object.

Reserved. An ignored field.

Resize. Whether the shared memory object may be resized. Possible values follow:

0 The shared memory object may not be resized.

1 The shared memory object may be resized.

Segment size. The size of the shared memory segment.

Size of all messages on queue. The byte size of all of the messages that are currently on the queue.

Teraspace. Whether the shared memory object is attachable only to a process's teraspace. Possible values follow:

0 The shared memory object is not attachable to a process's teraspace.

1 The shared memory object is attachable to a process's teraspace.

Title. The title of the semaphore. The title contains the 16 characters that are associated with the semaphore when it is created.

Value. The value of the semaphore.

Waiting qualified job identifier. The job name, the job user profile, and the job number of a thread waiting on the semaphore. The 26 characters are:

1-10 The job name

11-20 The user profile

21-26 The job number

Waiting thread identifier. The thread ID of a thread waiting on the semaphore.

Error Messages

Message ID	Error Message Text
CPF0F01 E	*SERVICE authority is required to run this program.
CPF2204 E	User profile &1 not found.
CPF24B4 E	Severe error while addressing parameter list.
CPF3C19 E	Error occurred with receiver variable specified.
CPF3C21 E	Format name &1 is not valid.
CPF3C90 E	Literal value cannot be changed.
CPF3CF1 E	Error code parameter not valid.
GUI0002 E	&2 is not valid for length of receiver variable.
GUI0027 E	&1 is not valid for number of records to return.
GUI0115 E	The list has been marked in error. See the previous messages.
GUI0118 E	Starting record cannot be 0 when records have been requested.
GUI0135 E	Filter key information is not valid.
GUI0136 E	Filter information is not valid.

API introduced: V4R2

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

Open List of Semaphores (QP0ZOLSM) API

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	List information	Output	Char(80)
4	Number of records to return	Input	Binary(4)
5	Format name	Input	Char(8)
6	Semaphore set identifier	Input	BINARY(4)
7	Error code	I/O	Char(*)

Default Public Authority: *USE

Threadsafe: No

The Open List of Semaphores (QP0ZOLSM) API lets you generate a list of description information about the semaphores within a semaphore set.

The QP0ZOLSM API places the specified number of list entries in the receiver variable. You can access additional records by using the Get List Entries (QGYGTLE) API. On successful completion of the QP0ZOLSM API, a handle is returned in the list information parameter. You may use this handle on subsequent calls to the following APIs:


- Get List Entries (QGYGTLE)
- Find Entry Number in List (QGYFNDE)
- Close List (QGYCLST)

The records returned by QP0ZOLSM include an information status field that describes the completeness and validity of the information. Be sure to check the information status field before using any other information returned.

Authorities and Locks

Job Authority

Service special authority (*SERVICE) is needed to call this API.

For additional information on this authority, see the [iSeries Security Reference](#)  book.

Required Parameter Group

Receiver variable

OUTPUT; CHAR(*)

The variable that is used to return the semaphore information that you requested.

Length of receiver variable

INPUT; BINARY(4)

The length of the receiver variable.

List information

OUTPUT; CHAR(80)

Information about the list of semaphores that were opened. For a description of the layout of this parameter, see [Format of Open List Information](#).

Number of records to return

INPUT; BINARY(4)

The number of records in the list to put into the receiver variable.

Format name

INPUT; CHAR(8)

The format of the information to be returned in the receiver variable. You must use the following format name:

LSEM0100 This format is described in [LSEM0100 Format](#).

Semaphore set identifier

INPUT; BINARY(4)

The semaphore set identifier of the semaphore set whose semaphores you would like the information about. The semaphore set identifier can be obtained from calling either the `semget()`, or `QPOZOLIP` API.

Error code

I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see [Error Code Parameter](#).

LSEM0100 Format

This format name is used to return list information for the semaphores in a semaphore set. The following table shows the information returned in each record in the receiver variable for the LSEM0100 format. For a detailed description of each field, see [Field Descriptions](#).

Offset		

Dec	Hex	Type	Field
0	0	BINARY(4)	Length of entry
4	4	BINARY(4)	Number
8	8	BINARY(4)	Value
12	C	BINARY(4)	Displacement to wait values
16	10	BINARY(4)	Number of waiters
20	14	BINARY(4)	Size of waiting information
24	18	BINARY(4)	Waiting for zero
28	1C	BINARY(4)	Waiting for positive value
32	20	CHAR(26)	Last changed qualified job identifier
58	3A	CHAR(2)	Reserved
60	3C	BINARY(4)	Process identifier
These fields repeat for each waiter on the semaphore value.		BINARY(4)	Wait value
		CHAR(26)	Waiting qualified job identifier
		CHAR(2)	Reserved

Field Descriptions

Displacement to wait values. The offset in characters (bytes) from the beginning of the semaphore record to the beginning of the array of wait values.

Last changed qualified job identifier. The job name, the job user profile, and the job number of the thread that last changed the value of the semaphore. The 26 characters are:

1-10 The job name

11-20 The user profile

21-26 The job number

These fields will be all blanks if any of the following are true:

- No thread has changed the semaphore value.
- The process that changed the semaphore has ended.
- The process that changed the semaphore has not been initialized for signals.

Length of entry. The length of this semaphore record in the list.

Number. The semaphore number in the semaphore set.

Number of waiters. The total number of threads that are waiting for this semaphore to reach a certain value.

Process identifier The process identifier of the last thread to change the value of the semaphore. If no thread has changed the semaphore value, this field will be zero.

Reserved. An ignored field.

Size of waiting information. The size, in bytes, of the record that is used to store information about a thread that is waiting for a semaphore value.

Value. The current value of the semaphore.

Wait value. The value that a thread is waiting for the semaphore to reach. If the value is zero, the thread is waiting for the semaphore value to equal zero. If the value is a positive number, the thread is waiting for the semaphore value to be greater than or equal to this value.

Waiting for positive value. The number of threads that are currently waiting for a semaphore value to reach a positive number.

Waiting for zero. The number of threads that are currently waiting for the semaphore value to reach zero.

Waiting qualified job identifier. The job name, the job user profile, and the job number of the thread that is currently waiting for the semaphore. The 26 characters are:

1-10 The job name

11-20 The user profile

21-26 The job number

Error Messages

Message ID	Error Message Text
GUI0002 E	&2 is not valid for length of receiver variable.
GUI0027 E	&1 is not valid for number of records to return.
GUI0115 E	The list has been marked in error. See the previous messages.
GUI0118 E	Starting record cannot be 0 when records have been requested.
CPF0F01 E	*SERVICE authority is required to run this program.
CPF2204 E	User profile &1 not found.
CPF24B4 E	Severe error while addressing parameter list.
CPF3C19 E	Error occurred with receiver variable specified.
CPF3C21 E	Format name &1 is not valid.
CPF3C90 E	Literal value cannot be changed.
CPF3CF1 E	Error code parameter not valid.
CPFA988 E	IPC object &1 does not exist.

API introduced: V4R2

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

Retrieve an Interprocess Communication Object (QP0ZRIPC) API

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Format name	Input	Char(8)
4	Identifier	Input	Binary(4)
5	Error code	I/O	Char(*)

Default Public Authority: *USE

Threadsafe: No


The Retrieve an Interprocess Communication Object (QP0ZRIPC) API lets you generate detailed information about a single interprocess communication (IPC) object. The object is identified by the format name and the identifier that is passed in.

The QP0ZRIPC API places the information about the object in the receiver variable. The information that is written to the receiver variable is dependent on the format name parameter.

Authorities and Locks

Job Authority

Service special authority (*SERVICE) is needed to call this API.

For additional information on this authority, see the [iSeries Security Reference](#)  book.

Required Parameter Group

Receiver variable

OUTPUT; CHAR(*)

The variable that is used to return the IPC object information that you requested.

Length of receiver variable

INPUT; BINARY(4)

The length of the receiver variable. The minimum length is 8 bytes.

Format name

INPUT; CHAR(8)

The format of the information to be returned in the receiver variable. This parameter will determine

the object type (either message queues, semaphore sets, or shared memory) to retrieve the list for. You must use one of the following format names:

RSST0100 This format is described in [RSST0100 Format](#).

RMSQ0100 This format is described in [RMSQ0100 Format](#).

RSHM0100 This format is described in [RSHM0100 Format](#).

Identifier

INPUT; BINARY(4)

The identifier of the IPC object that you would like to retrieve information about. This identifier is returned from the APIs `semget()`, `shmget()`, `msgget()`, or `QPOZOLIP`.

Error code

I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see [Error Code Parameter](#).

RSST0100 Format

This format name is used to return information for a single semaphore set. The following table shows the information returned in the receiver variable for the RSST0100 format. For a detailed description of each field, see [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Identifier
12	C	BINARY(4)	Key
16	10	BINARY(4)	Number of semaphores
20	14	CHAR(1)	Damaged
21	15	CHAR(1)	Owner read permission
22	16	CHAR(1)	Owner write permission
23	17	CHAR(1)	Group read permission
24	18	CHAR(1)	Group write permission
25	19	CHAR(1)	General read permission
26	1A	CHAR(1)	General write permission
27	1B	CHAR(1)	Authorized to delete
28	1C	CHAR(16)	Last semop() date and time
44	2C	CHAR(16)	Last administration change date and time
60	3C	CHAR(10)	Owner
70	46	CHAR(10)	Group owner

80	50	CHAR(10)	Creator
90	5A	CHAR(10)	Creator's group

RMSQ0100 Format

This format name is used to return information about a single message queue. The following table shows the information returned in the receiver variable for the RMSQ0100 format. For a detailed description of each field, see [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Identifier
12	C	BINARY(4)	Key
16	10	CHAR(1)	Damaged
17	11	CHAR(1)	Owner read permission
18	12	CHAR(1)	Owner write permission
19	13	CHAR(1)	Group read permission
20	14	CHAR(1)	Group write permission
21	15	CHAR(1)	General read permission
22	16	CHAR(1)	General write permission
23	17	CHAR(1)	Authorized to delete
24	18	BINARY(4)	Number of messages on queue
28	1C	BINARY(4)	Size of all messages on queue
32	20	BINARY(4)	Maximum size of all messages on queue
36	24	BINARY(4)	Number of threads to receive message
40	28	BINARY(4)	Number of threads to send message
44	2C	CHAR(16)	Last msgrcv() date and time
60	3C	CHAR(16)	Last msgsnd() date and time
76	4C	CHAR(16)	Last administration change date and time
92	5C	CHAR(10)	Owner
102	66	CHAR(10)	Group owner
112	70	CHAR(10)	Creator
122	7A	CHAR(10)	Creator's group
132	84	CHAR(26)	Last msgsnd() qualified job identifier
158	9E	CHAR(2)	Reserved
160	A0	BINARY(4)	Last msgsnd() process identifier
164	A4	CHAR(26)	Last msgrcv() qualified job identifier
190	BE	CHAR(2)	Reserved
192	C0	BINARY(4)	Last msgrcv() process identifier

196	C4	BINARY(4)	Offset to message type
200	C8	BINARY(4)	Size of message information record
204	CC	BINARY(4)	Offset to wait type
208	D0	BINARY(4)	Size of message receive record
212	D4	BINARY(4)	Offset to wait size
216	D8	BINARY(4)	Size of message send record
These fields repeat for each message on queue.		BINARY(4)	Message type
		BINARY(4)	Message size
These fields repeat for each thread waiting to receive a message.		BINARY(4)	Message wait type
		CHAR(26)	Message receive qualified job identifier
		CHAR(2)	Reserved
These fields repeat for each thread waiting to send a message.		BINARY(4)	Message wait size
		CHAR(26)	Message send qualified job identifier
		CHAR(2)	Reserved

RSHM0100 Format

This format name is used to return information for a single shared memory object. The following table shows the information returned in the receiver variable for the RSHM0100 format. For a detailed description of each field, see [Field Descriptions](#).

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Identifier
12	C	BINARY(4)	Key
16	10	CHAR(1)	Damaged
17	11	CHAR(1)	Owner read permission
18	12	CHAR(1)	Owner write permission
19	13	CHAR(1)	Group read permission
20	14	CHAR(1)	Group write permission
21	15	CHAR(1)	General read permission
22	16	CHAR(1)	General write permission
23	17	CHAR(1)	Marked to be deleted
24	18	CHAR(1)	Authorized to delete
25	19	CHAR(1)	Teraspace
26	1A	CHAR(1)	Resize
27	1B	CHAR(1)	Reserved

28	1C	BINARY(4)	Segment size
32	20	BINARY(4)	Number attached
36	24	CHAR(16)	Last shmat() date and time
52	34	CHAR(16)	Last detach date and time
68	44	CHAR(16)	Last administration change date and time
84	54	CHAR(10)	Owner
94	5E	CHAR(10)	Group owner
104	68	CHAR(10)	Creator
114	72	CHAR(10)	Creator's group
124	7C	CHAR(26)	Last attach or detach qualified job identifier
150	96	CHAR(2)	Reserved
152	98	BINARY(4)	Last attach or detach process identifier
156	9C	BINARY(4)	Offset to times attached
160	A0	BINARY(4)	Number of attach entries
164	A4	BINARY(4)	Size of attach entry
These fields repeat for the number of attach entries.		BINARY(4)	Times attached
		CHAR(26)	Attached qualified job identifier
		CHAR(2)	Reserved

Field Descriptions

Attached qualified job identifier. The job name, the job user profile, and the job number of a job that is attached to the shared memory segment. The 26 characters are:

1-10 The job name

11-20 The user profile

21-26 The job number

Authorized to delete. This value determines if the caller has the authority to delete this IPC object. Possible values follow:

0 The current thread cannot delete the IPC object.

1 The current thread can delete the IPC object.

Bytes available. The number of bytes of data available to be returned. All available data is returned if enough space is provided.

Bytes returned. The number of bytes of data returned.

Creator. The name of the user profile that created this IPC object.

Creator's group. The name of the group profile that created this IPC object. A special value can be returned:

**NONE* The creator does not have a group profile.

Damaged. Whether the IPC object has suffered internal damage. Possible values follow:

0 The IPC object is not damaged.

1 The IPC object is damaged.

General read permission. Whether any user other than the owner and group owner has read authority to the IPC object. Possible values follow:

0 General read authority is not allowed to the IPC object.

1 General read authority is allowed to the IPC object.

General write permission. Whether any user other than the owner and group owner has write authority to the IPC object. Possible values follow:

0 General write authority is not allowed to the IPC object.

1 General write authority is allowed to the IPC object.

Group owner. The name of the group profile that owns this IPC object. A special value can be returned:

**NONE* The IPC object does not have a group owner.

Group read permission. Whether the group owner has read authority to the IPC object. Possible values follow:

0 The group owner does not have read authority to the IPC object.

1 The group owner has read authority to the IPC object.

Group write permission. Whether the group owner has write authority to the IPC object. Possible values follow:

0 The group owner does not have write authority to the IPC object.

1 The group owner has write authority to the IPC object.

Identifier. The unique IPC object identifier.

Key. The key of the IPC object. If this value is zero, this IPC object has no key associated with it.

Last administration change date and time. The date and time of the last change to the IPC object for the owner, group owner, or permissions. The 16 characters are:

1 Century, where 0 indicates years 19xx and 1 indicates years 20xx.

2-7 Date, in YYMMDD (year, month, and day) format.

8-13 Time of day, in HHMMSS (hours, minutes, and seconds) format.

14-16 Milliseconds.

Last attach or detach process identifier. The process identifier of the thread that performed the last successful attachment or detachment from the shared memory segment. If no thread has attached or detached from the shared memory segment, this field will be zero.

Last attach or detach qualified job identifier. The job name, the job user profile, and the job number of the thread that performed the last successful attachment or detachment from the shared memory segment. The 26 characters are:

1-10 The job name

11-20 The user profile

21-26 The job number

These fields will be all blanks if any of the following are true:

- No thread has performed an attachment or detachment on the shared memory.
- The last process that did an attachment or detachment on the shared memory has ended.
- The last process that did an attachment or detachment on the shared memory is not initialized for signals.

Last detach date and time. The date and time of the last detachment from the shared memory segment. If no thread has performed a successful detachment, this value will be set to all zeros. The 16 characters are:

1 Century, where 0 indicates years 19xx and 1 indicates years 20xx.

2-7 Date, in YYMMDD (year, month, and day) format.

8-13 Time of day, in HHMMSS (hours, minutes, and seconds) format.

14-16 Milliseconds.

Last msgrcv() date and time. The date and time of the last successful msgrcv() call. If no thread has performed a successful msgrcv() call, this value will be set to all zeros. The 16 characters are:

1 Century, where 0 indicates years 19xx and 1 indicates years 20xx.

2-7 Date, in YYMMDD (year, month, and day) format.

8-13 Time of day, in HHMMSS (hours, minutes, and seconds) format.

14-16 Milliseconds.

Last msgrcv() process identifier. The process identifier of the thread that performed the last successful msgrcv(). If no thread has done a msgrcv(), this field will be zero.

Last msgrcv() qualified job identifier. The job name, the job user profile, and the job number of the thread that performed the last successful msgrcv(). The 26 characters are:

1-10 The job name

11-20 The user profile

21-26 The job number

These fields will be all blanks if any of the following are true:

- No thread has received a message on this message queue.
- The last process to receive a message has ended.
- The last process to receive a message has not been initialized for signals.

Last msgsnd() date and time. The date and time of the last successful msgsnd() call. If no thread has performed a successful msgsnd() call, this value will be set to all zeros. The 16 characters are:

- 1* Century, where 0 indicates years 19xx and 1 indicates years 20xx.
- 2-7* Date, in YYMMDD (year, month, and day) format.
- 8-13* Time of day, in HHMMSS (hours, minutes, and seconds) format.
- 14-16* Milliseconds.

Last msgsnd() process identifier. The process identifier of the thread that performed the last successful msgsnd(). If no thread has done a msgsnd(), this field will be zero.

Last msgsnd() qualified job identifier. The job name, the job user profile, and the job number of the thread that performed the last successful msgsnd(). The 26 characters are:

- 1-10* The job name
- 11-20* The user profile
- 21-26* The job number

These fields will be all blanks if any of the following are true:

- No thread has sent a message to this message queue.
- The last process to send a message has ended.
- The last process to send a message has not been initialized for signals.

Last semop() date and time. The date and time of the last successful semop() call. If no thread has performed a successful semop() call, this value will be set to all zeros. The 16 characters are:

- 1* Century, where 0 indicates years 19xx and 1 indicates years 20xx.
- 2-7* Date, in YYMMDD (year, month, and day) format.
- 8-13* Time of day, in HHMMSS (hours, minutes, and seconds) format.
- 14-16* Milliseconds.

Last shmat() date and time. The date and time of the last successful shmat(). If no thread has performed a successful shmat() call, this value will be set to all zeros. The 16 characters are:

- 1* Century, where 0 indicates years 19xx and 1 indicates years 20xx.
- 2-7* Date, in YYMMDD (year, month, and day) format.

8-13 Time of day, in HHMMSS (hours, minutes, and seconds) format.

14-16 Milliseconds.

Marked to be deleted. Whether the shared memory is marked to be deleted when the number attached becomes zero. Possible values follow:

0 The shared memory segment is not marked for deletion.

1 The shared memory segment is marked for deletion.

Maximum size of all messages on queue. The maximum byte size of all messages that can be on the queue at one time.

Message receive qualified job identifier. The job name, the job user profile, and the job number of the thread that is waiting to receive a message. The 26 characters are:

1-10 The job name

11-20 The user profile

21-26 The job number

Message send qualified job identifier. The job name, the job user profile, and the job number of the thread that is waiting to send a message. The 26 characters are:

1-10 The job name

11-20 The user profile

21-26 The job number

Message size. The message size of a message that is currently on the queue.

Message type. The message type of a message that is currently on the queue.

Message wait size. The message size of a message that a thread is currently waiting to put on the queue.

Message wait type. The message type that a thread is currently waiting to receive.

Number attached. The number of times any thread has done a shmat() without doing a detach. One process can be attached multiple times to the same shared memory segment.

Number of attach entries. The number of entries in the variable length section of RSHM0100.

Number of threads to receive message. The number of threads that are currently waiting to receive a message.

Number of threads to send message. The number of threads that are currently waiting to send a message.

Number of messages on queue. The number of messages that are currently on the message queue.

Number of semaphores. The number of semaphores in the semaphore set.

Offset to message type. The offset in characters (bytes) from the beginning of the RMSQ0100 record to the message type field.

Offset to times attached. The offset in characters (bytes) from the beginning of the RSHM0100 record to the times attached field.

Offset to wait size. The offset in characters (bytes) from the beginning of the RMSQ0100 record to the wait size field.

Offset to wait type. The offset in characters (bytes) from the beginning of the RMSQ0100 record to the wait type field.

Owner. The name of the user profile that owns this IPC object.

Owner read permission. Whether the owner has read authority to the IPC object. Possible values follow:

0 The owner does not have read authority to the IPC object.

1 The owner has read authority to the IPC object.

Owner write permission. Whether the owner has write authority to the IPC object. Possible values follow:

0 The owner does not have write authority to the IPC object.

1 The owner has write authority to the IPC object.

Reserved. An ignored field.

Resize. Whether the shared memory object may be resized. Possible values follow:

0 The shared memory object may not be resized.

1 The shared memory object may be resized.

Segment size. The size of the shared memory segment.

Size of all messages on queue. The size, in bytes, of all of the messages that are currently on the queue.

Size of attach entry. The size, in bytes, of each attach entry in the array of attach entries.

Size of message information record. The size, in bytes, of each message information record.

Size of message receive record. The size, in bytes, of the record that is used to store information about a thread waiting to receive a message.

Size of message send record. The size, in bytes, of the record that is used to store information about a thread waiting to send a message.

Teraspace. Whether the shared memory object is attachable only to a process's teraspace. Possible values follow:

0 The shared memory object is not attachable to a process's teraspace.

1 The shared memory object is attachable to a process's teraspace.

Times attached. The number of times that this process is attached to the shared memory.

Error Messages

- GUI0002 E &2 is not valid for length of receiver variable.
- CPF0F01 E *SERVICE authority is required to run this program.
- CPF24B4 E Severe error while addressing parameter list.
- CPF3C19 E Error occurred with receiver variable specified.
- CPF3C21 E Format name &1 is not valid.
- CPF3C90 E Literal value cannot be changed.
- CPF3CF1 E Error code parameter not valid.
- CPFA988 E IPC object &1 does not exist.

API introduced: V4R2

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

semctl()-Perform Semaphore Control Operations

Syntax

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

int semctl(int semid, int semnum, int cmd, ...);
```

Service Program Name: QP0ZCPA

Default Public Authority: *USE

Threadsafe: Yes

The **semctl()** function provides semaphore control operations as specified by `cmd` on the semaphore specified by `semnum` in the semaphore set specified by `semid`.

Parameters

semid

(Input) Semaphore set identifier, a positive integer. It is created by the **semget()** function and used to identify the semaphore set on which to perform the control operation.

semnum

(Input) Semaphore number, a non-negative integer. It identifies a semaphore within the semaphore set on which to perform the control operation.

cmd

(Input) Command, the control operation to perform on the semaphore. See below for details.

...

(Input/output) An optional fourth parameter whose type depends on the value of `cmd`. For a `cmd` value of `SETVAL`, this parameter must be an integer. For a `cmd` value of `IPC_STAT` or `IPC_SET`, this parameter must be a pointer to a `semid_ds` structure. For a `cmd` value of `GETALL` or `SETALL`, this parameter must be a pointer to an array of type unsigned short. For all other values of `cmd`, this parameter is not required.

Note: Before Version 3 Release 6 of OS/400, the Common Programming APIs (CPA) Toolkit/400 support for **semctl()** required the use of the `semun` structure for this fourth parameter. With support for semaphores in OS/400, use of `semun` is not required and is not recommended when passing an integer value. If the optional fourth parameter is an integer, IPC expects it to directly follow the third parameter in storage. However, if the `semun` structure is used to pass the integer value, the value is aligned on a 16-byte boundary, which might not directly follow the third parameter. Therefore, the value used by IPC for the fourth parameter might not be the value intended by the caller, and unexpected results could occur.

The `cmd` parameter can have one of the following values:

- GETVAL* Return the `semval` value in the `semaphore_t` data structure associated with the specified semaphore. This command requires read permission.
- SETVAL* Set the `semval` value in the `semaphore_t` data structure associated with the specified semaphore to the integer value found in the fourth parameter and clear the associated per-thread semaphore adjustment value. This command requires write permission.
- GETPID* Return the `sempid` value in the `semaphore_t` data structure associated with the specified semaphore. This value is the process ID of the last thread to operate on the specified semaphore. This command requires read permission.
- GETNCNT* Return the `semncnt` value in the `semaphore_t` data structure associated with the specified semaphore. This value is the number of threads waiting for the specified semaphore's value to increase. This command requires read permission.
- GETZCNT* Return the `semzcnt` value in the `semaphore_t` data structure associated with the specified semaphore. This value is the number of threads waiting for the specified semaphore's value to reach zero. This command requires read permission.
- GETALL* Return the `semval` value in the `semaphore_t` data structure associated with each semaphore in the specified semaphore set. The `semval` values will be returned in the array pointed to by the fourth parameter, which will be a pointer to an array of type unsigned short. This command requires read permission.
- SETALL* Set the `semval` value in the `semaphore_t` data structure associated with each semaphore in the specified semaphore set and clear all associated per-thread semaphore-adjustment values. The `semval` values are set to the values contained in the array pointed to by the fourth parameter, which is a pointer to an array of type unsigned short. This command requires write permission.
- IPC_STAT* Place the current value of each member of the `semid_ds` data structure associated with `semid` into the structure pointed to by the fourth parameter, which is a pointer to a `semid_ds` structure. This command requires read permission.
- IPC_SET* Set the value of the following members of the `semid_ds` data structure associated with `semid` to the corresponding value found in the structure pointed to by the fourth parameter, which is a pointer to a `semid_ds` structure:
- `sem_perm.uid`
 - `sem_perm.gid`
 - `sem_perm.mode`
- `IPC_SET` can be performed only by a thread with appropriate privileges or one that has an effective user ID equal to the value of `sem_perm.cuid` or `sem_perm.uid` in the `semid_ds` data structure associated with `semid`.
- IPC_RMID* Remove the semaphore identifier specified by `semid` from the system and destroy the set of semaphores and `semid_ds` data structure associated with it. `IPC_RMID` can be performed only by a thread with appropriate privileges or one that has an effective user ID equal to the value of `sem_perm.cuid` or `sem_perm.uid` in the `semid_ds` data structure associated with `semid`.

Authorities

Figure 1-11. Authorization Required for semctl()

Object Referred to	Authority Required	errno
Semaphore, get the value of (cmd = GETVAL)	Read	EACCES
Semaphore, set the value of (cmd = SETVAL)	Write	EACCES
Semaphore, get last process to operate on (cmd = GETPID)	Read	EACCES
Semaphore, get number of threads waiting for value to increase (cmd = GETNCNT)	Read	EACCES
Semaphore, get number of threads waiting for value to reach zero (cmd = GETZCNT)	Read	EACCES
Semaphore set, get value of each semaphore (cmd = GETALL)	Read	EACCES
Semaphore set, set value of each semaphore (cmd = SETALL)	Write	EACCES
Semaphore set, retrieve state information (cmd = IPC_STAT)	Read	EACCES
Semaphore set, set state information (cmd = IPC_SET)	See Note	EPERM
Semaphore set, remove (cmd = IPC_RMID)	See Note	EPERM

Note: To set semaphore set information or to remove a semaphore set, the thread must be the owner or creator of the semaphore set, or have appropriate privileges.

Return Value

value **semctl()** was successful. Depending on the control operation specified in *cmd*, **semctl()** returns the following values:

<i>GETVAL</i>	The value of the specified semaphore.
<i>GETPID</i>	The process ID of the last thread that performed a semaphore operation on the specified semaphore.
<i>GETNCNT</i>	The number of threads waiting for the value of the specified semaphore to increase.
<i>GETZCNT</i>	The number of threads waiting for the value of the specified semaphore to reach zero.
<i>For all other values of cmd:</i>	The value is 0.

-1 **semctl()** was not successful. The *errno* variable is set to indicate the error.

Error Conditions

If **semctl()** is not successful, `errno` usually indicates one of the following errors. Under some conditions, `errno` could indicate an error other than those listed here.

[EACCES]

Permission denied.

An attempt was made to access an object in a way forbidden by its object access permissions.

The thread does not have access to the specified file, directory, component, or path.

If you are accessing a remote file through the Network File System, update operations to file permissions at the server are not reflected at the client until updates to data that is stored locally by the Network File System take place. (Several options on the Add Mounted File System (ADDMFS) command determine the time between refresh operations of local data.) Access to a remote file may also fail due to different mappings of user IDs (UID) or group IDs (GID) on the local and remote systems.

Operation permission is denied to the calling thread.

[EDAMAGE]

A damaged object was encountered.

A referenced object is damaged. The object cannot be used.

The value of `semid` corresponds to a semaphore set that has been marked as damaged by a previous semaphore operation.

[EFAULT]

The address used for an argument is not correct.

In attempting to use an argument in a call, the system detected an address that is not valid.

While attempting to access a parameter passed to this function, the system detected an address that is not valid.

[EINVAL]

An invalid parameter was found.

A parameter passed to this function is not valid.

One of the following has occurred:

- The value of `semid` is not a valid semaphore identifier.
- The value of `semnum` is less than zero or greater than or equal to `sem_nsems`.
- The value of `cmd` is not a valid command.

[EPERM]

Operation not permitted.

You must have appropriate privileges or be the owner of the object or other resource to do the requested operation.

The parameter `cmd` is equal to `IPC_RMID` or `IPC_SET` and both of the following are true:

- the calling thread does not have appropriate privileges.
- the effective user ID of the calling thread is not equal to the value of `sem_perm.cuid` or `sem_perm.uid` in the data structure associated with `semid`.

[ERANGE]

A range error occurred.

The value of an argument is too small, or a result too large.

The parameter `cmd` is equal to `SETVAL`, and the value to which `semval` is to be set is greater than the system-imposed maximum.

[EUNKNOWN]

Unknown system state.

The operation failed because of an unknown system state. See any messages in the job log and correct any errors that are indicated, then retry the operation.

Error Messages

None.

Usage Notes

"Appropriate privileges" is defined to be `*ALLOBJ` special authority. If the user profile under which the thread is running does not have `*ALLOBJ` special authority, the thread does not have appropriate privileges.

Related Information

- The `<sys/sem.h>` file (see [Header Files for UNIX-Type Functions](#))
- [semget\(\)-Get Semaphore Set with Key](#)
- [semop\(\)-Perform Semaphore Operations on Semaphore Set](#)

Example

For an example of using this function, see Using Semaphores and Shared Memory in [Examples](#).

API introduced: V3R6

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

semget()-Get Semaphore Set with Key

Syntax

```
#include <sys/sem.h>
#include <sys/stat.h>

int semget (key_t key, int nsems, int semflg);
```

Service Program Name: QP0ZCPA

Default Public Authority: *USE

Threadsafe: Yes

The **semget()** function returns the semaphore ID associated with the specified semaphore key.

Parameters

key

(Input) Key associated with the semaphore set. Specifying a key of `IPC_PRIVATE` guarantees that a unique semaphore set is created. A key also can be generated by the caller or by calling the **ftok()** function.

nsems

(Input) Number of semaphores in the semaphore set. The number of semaphores in the set cannot be changed after the semaphore set is created. If an existing semaphore set is being accessed, `nsems` can be zero.

semflg

(Input) Operations and permission flags.

The `semflg` parameter value is either 0, or is obtained by performing an OR operation on one or more of the following constants:

- | | |
|------------------|--|
| <i>S_IRUSR</i> | Permits the creator of the semaphore set to read it. |
| <i>S_IWUSR</i> | Permits the creator of the semaphore set to write it. |
| <i>S_IRGRP</i> | Permits the group associated with the semaphore set to read it. |
| <i>S_IWGRP</i> | Permits the group associated with the semaphore set to write it. |
| <i>S_IROTH</i> | Permits others to read the semaphore set. |
| <i>S_IWOTH</i> | Permits others to write the semaphore set. |
| <i>IPC_CREAT</i> | Creates the semaphore set if it does not already exist. |
| <i>IPC_EXCL</i> | Causes semget() to fail if <code>IPC_CREAT</code> is also set and the semaphore set already exists. |

Authorities

Figure 1-12. Authorization Required for `semget()`

Object Referred to	Authority Required	errno
Semaphore set to be created	None	None
Existing semaphore set to be accessed	See Note	EACCES

Note: If the thread is accessing a semaphore set that already exists, the mode specified in the last 9 bits of `semflg` must be a subset of the mode of the existing semaphore set.

Return Value

value `semget()` was successful. The value returned is the semaphore ID associated with the key parameter.

-1 `semget()` was not successful. The `errno` variable is set to indicate the error.

Error Conditions

If `semget()` is not successful, `errno` usually indicates one of the following errors. Under some conditions, `errno` could indicate an error other than those listed here.

[EACCES]

Permission denied.

An attempt was made to access an object in a way forbidden by its object access permissions.

The thread does not have access to the specified file, directory, component, or path.

If you are accessing a remote file through the Network File System, update operations to file permissions at the server are not reflected at the client until updates to data that is stored locally by the Network File System take place. (Several options on the Add Mounted File System (ADDMFS) command determine the time between refresh operations of local data.) Access to a remote file may also fail due to different mappings of user IDs (UID) or group IDs (GID) on the local and remote systems.

A semaphore identifier exists for the parameter key, but operation permission as specified by the low-order 9 bits of `semflg` would not be granted.

[EDAMAGE]

A damaged object was encountered.

A referenced object is damaged. The object cannot be used.

The value of key corresponds to a semaphore set that has been marked as damaged by a previous semaphore operation.

[EEXIST]

File exists.

The file specified already exists and the specified operation requires that it not exist.

The named file, directory, or path already exists.

A semaphore identifier exists for the parameter key, but $((\text{semflg} \ \& \ \text{IPC_CREAT}) \ \&\& \ (\text{semflg} \ \& \ \text{IPC_EXCL}))$ is not zero. (& is a bitwise AND; && is a logical AND.)

[EINVAL]

An invalid parameter was found.

A parameter passed to this function is not valid.

One of the following has occurred:

- The value of nsems is either less than or equal to zero or greater than the system-imposed limit.
- A semaphore identifier exists for the parameter key, but the number of semaphores in the set associated with it is less than nsems and nsems is not zero.

[ENOENT]

No such path or directory.

The directory or a component of the path name specified does not exist.

A named file or directory does not exist or is an empty string.

A semaphore identifier does not exist for the parameter key, and $(\text{semflg} \ \& \ \text{IPC_CREAT})$ is equal to zero.

[ENOSPC]

No space available.

The requested operations required additional space on the device and there is no space left. This could also be caused by exceeding the user profile storage limit when creating or transferring ownership of an object.

Insufficient space remains to hold the intended file, directory, or link.

A semaphore identifier is to be created but the system-imposed limit on the maximum number of allowed semaphores system-wide would be exceeded.

[EUNKNOWN]

Unknown system state.

The operation failed because of an unknown system state. See any messages in the job log and correct any errors that are indicated, then retry the operation.

Error Messages

None.

Usage Notes

1. **semget()** creates a semaphore set and its associated `semid_ds` data structure if one of the following is true:
 - The semaphore key is `IPC_PRIVATE`.
 - A semaphore set is not already associated with the semaphore key, and the `IPC_CREAT` flag is set.
2. When the semaphore set is created, the `semid_ds` data structure associated with the semaphore set is initialized as follows:
 - The `sem_perm.cuid` and `sem_perm.uid` values are set to the current user ID (uid) of the thread.
 - The `sem_perm.cgid` and `sem_perm.gid` values are set to the current group ID (gid) of the thread.
 - The `sem_perm.mode` is set according to the permissions specified in `semflg`.
 - The number of semaphores, `sem_nsems`, is set to the `nsems` parameter.
 - `sem_otime` is set to zero and `sem_ctime` is set to the current time.
3. A **semctl()** call specifying a `cmd` parameter of `SETALL` should be used to initialize the semaphore values after the semaphore set is created.

Related Information

- The `<sys/sem.h>` file (see [Header Files for UNIX-Type Functions](#))
- [ftok\(\)--Generate IPC Key from File Name](#)
- [semctl\(\)--Perform Semaphore Control Operations](#)
- [semop\(\)--Perform Semaphore Operations on Semaphore Set](#)

Example

For an example of using this function, see [Using Semaphores and Shared Memory in Appendix A, Examples](#).

API introduced: V4R4

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

semop()-Perform Semaphore Operations on Semaphore Set

Syntax

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

int semop(int semid, struct sembuf *sops,
          size_t nsops);
```

Service Program Name: QP0ZCPA

Default Public Authority: *USE

Threadsafe: Yes

The **semop()** function performs operations on semaphores in a semaphore set. These operations are supplied in a user-defined array of operations.

Parameters

semid

(Input) Semaphore set identifier.

sops

(Input) Pointer to array of semaphore operation (*sembuf*) structures.

nsops

(Input) Number of *sembuf* structures in *sops* array.

Following is an example of what one of the *sembuf* structures should look like:

```
struct sembuf {
    unsigned short sem_num; /* semaphore number */
    short sem_op; /* semaphore operation */
    short sem_flg; /* operation flags SEM_UNDO and IPC_NOWAIT */
}
```

Authorities

Figure 1-13. Authorization Required for *semop()*

Object Referred to	Authority Required	errno
Semaphore, <i>sem_op</i> is negative	Write	EACCES
Semaphore, <i>sem_op</i> is positive	Write	EACCES

Semaphore, sem_op is zero	Read	EACCES
---------------------------	------	--------

Return Value

0 **semop()** was successful.

-1 **semop()** was not successful. The `errno` variable is set to indicate the error.

Error Conditions

If **semop()** is not successful, `errno` usually indicates one of the following errors. Under some conditions, `errno` could indicate an error other than those listed here.

[EACCES]

Permission denied.

An attempt was made to access an object in a way forbidden by its object access permissions.

The thread does not have access to the specified file, directory, component, or path.

If you are accessing a remote file through the Network File System, update operations to file permissions at the server are not reflected at the client until updates to data that is stored locally by the Network File System take place. (Several options on the Add Mounted File System (ADDMFS) command determine the time between refresh operations of local data.) Access to a remote file may also fail due to different mappings of user IDs (UID) or group IDs (GID) on the local and remote systems.

Operation permission is denied to the calling thread.

[EAGAIN]

Operation would have caused the process to be suspended.

The operation would result in suspension of the calling thread but (`sem_flg & IPC_NOWAIT`) is not zero. (& is a bitwise AND.)

[EDAMAGE]

A damaged object was encountered.

A referenced object is damaged. The object cannot be used.

The value of `semid` corresponds to a semaphore set that has been marked as damaged by a previous semaphore operation.

[EFAULT]

The address used for an argument is not correct.

In attempting to use an argument in a call, the system detected an address that is not valid.

While attempting to access a parameter passed to this function, the system detected an address that

is not valid.

[EFBIG]

Object is too large.

The size of the object would exceed the system allowed maximum size.

The value of *sem_num* is less than 0 or greater than or equal to the number of semaphores in the set associated with *semid*.

[EIDRM]

ID has been removed.

The semaphore identifier *semid* has been removed from the system.

[EINTR]

Interrupted function call.

The *semop()* function was interrupted by a signal while the thread was in a wait state.

[EINVAL]

An invalid parameter was found.

A parameter passed to this function is not valid.

The value of *semid* is not a valid semaphore identifier.

[ENOSPC]

No space available.

The requested operations required additional space on the device and there is no space left. This could also be caused by exceeding the user profile storage limit when creating or transferring ownership of an object.

Insufficient space remains to hold the intended file, directory, or link.

The limit on the number of individual threads requesting a SEM_UNDO would be exceeded.

[ERANGE]

A range error occurred.

The value of an argument is too small, or a result too large.

An operation would cause a semval to overflow the system-imposed limit, or an operation would cause a semaphore adjustment value to overflow the system-imposed limit.

[EUNKNOWN]

Unknown system state.

The operation failed because of an unknown system state. See any messages in the job log and correct any errors that are indicated, then retry the operation.

Error Messages

None.

Usage Notes

1. Each semaphore operation specified by the sops array is performed on the semaphore set specified by semid. The entire array of operations is performed atomically; no other thread will operate on the semaphore set until all of the operations are done or it is determined that they cannot be done. If the entire set of operations cannot be performed, none of the operations are done, and the thread waits until all of the operations can be done.
2. **semop()** changes each semaphore specified by sem_num according to the value of sem_op:
 - If sem_op is positive, **semop()** increments the value of the semaphore and awakens any threads waiting for the semaphore to increase. This corresponds to releasing resources controlled by the semaphore.
 - If sem_op is negative, **semop()** attempts to decrement the value of the semaphore. If the result would be negative, it waits for the semaphore value to increase. If the result would be positive, it decrements the semaphore. If the result would be zero, it decrements the semaphore and awakens any threads waiting for the semaphore to be zero. This corresponds to the allocation of resources.
 - If sem_op is zero, the thread waits for the semaphore's value to be zero.
3. If IPC_NOWAIT is set and the operation cannot be completed, **semop()** returns an [EAGAIN] error instead of causing the thread to wait.
4. If SEM_UNDO is set, **semop()** causes IPC to reverse the effect of this semaphore operation when the thread ends, effectively releasing the resources or request for resources controlled by the semaphore. This value is known as the semaphore adjustment value.
5. A **semop()** is interruptible by an asynchronous signal when the thread is waiting for a semaphore to reach a value.

Related Information

- The <sys/sem.h> file (see [Header Files for UNIX-Type Functions](#))
- [semget\(\)-Get Semaphore Set with Key](#)
- [semctl\(\)-Perform Semaphore Control Operations](#)

Example

For an example of using this function, see Using Semaphores and Shared Memory in [Examples](#).

API introduced: V3R6

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

sem_close()-Close Named Semaphore

Syntax

```
#include <semaphore.h>

int sem_close(sem_t * sem);
```

Service Program Name: QP0ZPSEM

Default Public Authority: *USE

Threadsafe: Yes

The **sem_close()** function closes a named semaphore that was previously opened by a thread of the current process using **sem_open()** or **sem_open_np()**. The **sem_close()** function frees system resources associated with the semaphore on behalf of the process. Using a semaphore after it has been closed will result in an error. A semaphore should be closed when it is no longer used. If a **sem_unlink()** was performed previously for the semaphore and the current process holds the last reference to the semaphore, then the named semaphore will be deleted and removed from the system.

Parameters

sem

(Input) A pointer to an opened named semaphore. This semaphore is closed for this process.

Authorities

No authorization is required. Authorization is verified during **sem_open()**.

Return Value

0 **sem_close()** was successful.

-1 **sem_close()** was not successful. The `errno` variable is set to indicate the error.

Error Conditions

If **sem_close()** is not successful, `errno` usually indicates one of the following errors. Under some conditions, `errno` could indicate an error other than those listed here.

[EINVAL]

The value specified for the argument is not correct.

A function was passed incorrect argument values, or an operation was attempted on an object and the operation specified is not supported for that type of object.

An argument value is not valid, out of range, or NULL.

The sem parameter is not a valid semaphore.

Error Messages

None.

Related Information

- The `<semaphore.h>` file (see [Header Files for UNIX-Type Functions](#))
- [sem_getvalue\(\)-Get Semaphore Value](#)
- [sem_open\(\)-Open Named Semaphore](#)
- [sem_open_np\(\)-Open Named Semaphore with Maximum Value](#)
- [sem_post\(\)-Post to Semaphore](#)
- [sem_post_np\(\)-Post Value to Semaphore](#)
- [sem_trywait\(\)-Try to Decrement Semaphore](#)
- [sem_unlink\(\)-Unlink Named Semaphore](#)
- [sem_wait\(\)-Wait for Semaphore](#)
- [sem_wait_np\(\)-Wait for Semaphore with Timeout](#)

Example

The following example opens a named semaphore with an initial value of 10 and then closes it.

```
#include <semaphore.h>
main() {
    sem_t * my_semaphore;
    int rc;
```



```
my_semaphore = sem_open("/mysemaphore",
                        O_CREAT, S_IRUSR | S_IWUSR,
                        10);

sem_close(my_semaphore);
}
```

API introduced: V4R4

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

sem_destroy()-Destroy Unnamed Semaphore

Syntax

```
#include <semaphore.h>

int sem_destroy(sem_t * sem);
```

Service Program Name: QP0ZPSEM

Default Public Authority: *USE

Threadsafe: Yes

The **sem_destroy()** function destroys an unnamed semaphore that was previously initialized using **sem_init()** or **sem_init_np()**. Any threads that have blocked from calling **sem_wait()** or **sem_wait_np()** on the semaphore will unblock and return an [EINVAL] or [EDESTROYED] error.

Parameters

sem

(Input) A pointer to an initialized unnamed semaphore. The semaphore is destroyed.

Authorities

None

Return Value

0 **sem_destroy()** was successful.

-1 **sem_destroy()** was not successful. The errno variable is set to indicate the error.

Error Conditions

If **sem_destroy()** is not successful, errno usually indicates one of the following errors. Under some conditions, errno could indicate an error other than those listed here.

[EBUSY]

Resource busy.

An attempt was made to use a system resource that is not available at this time.

The semaphore is being destroyed by another thread.

[EINVAL]

The value specified for the argument is not correct.

A function was passed incorrect argument values, or an operation was attempted on an object and the operation specified is not supported for that type of object.

An argument value is not valid, out of range, or NULL.

The sem parameter is not a valid semaphore.

Error Messages

None.

Related Information

- The <semaphore.h> file (see [Header Files for UNIX-Type Functions](#))
- [sem_getvalue\(\)-Get Semaphore Value](#)
- [sem_init\(\)-Initialize Unnamed Semaphore](#)
- [sem_init_np\(\)-Initialize Unnamed Semaphore with Maximum Value](#)
- [sem_post\(\)-Post to Semaphore](#)
- [sem_post_np\(\)-Post Value to Semaphore](#)
- [sem_trywait\(\)-Try to Decrement Semaphore](#)
- [sem_unlink\(\)-Unlink Named Semaphore](#)
- [sem_wait\(\)-Wait for Semaphore](#)
- [sem_wait_np\(\)-Wait for Semaphore with Timeout](#)

Example

The following example initializes an unnamed semaphore, `my_semaphore`, that will be used by threads of the current process and sets its value to 10. The semaphore is then destroyed using `sem_destroy()`.

```
#include <semaphore.h>
main() {
    sem_t my_semaphore;
    int rc;

    rc = sem_init(&my_semaphore, 0, 10);
    rc = sem_destroy(&my_semaphore);
}
```

API introduced: V4R4

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

sem_getvalue()-Get Semaphore Value

Syntax

```
#include <semaphore.h>

int sem_getvalue(sem_t * sem, int * value);
```

Service Program Name: QP0ZPSEM

Default Public Authority: *USE

Threadsafe: Yes

The **sem_getvalue()** function retrieves the value of a named or unnamed semaphore. If the current value of the semaphore is zero and there are threads waiting on the semaphore, a negative value is returned. The absolute value of this negative value is the number of threads waiting on the semaphore.

Parameters

sem

(Input) A pointer to an initialized unnamed semaphore or an opened named semaphore.

value

(Output) A pointer to the integer that contains the value of the semaphore.

Authorities

None

Return Value

0 **sem_getvalue()** was successful.

-1 **sem_getvalue()** was not successful. The `errno` variable is set to indicate the error.

Error Conditions

If **sem_getvalue()** is not successful, `errno` usually indicates one of the following errors. Under some conditions, `errno` could indicate an error other than those listed here.

[EINVAL]

The value specified for the argument is not correct.

A function was passed incorrect argument values, or an operation was attempted on an object and the operation specified is not supported for that type of object.

An argument value is not valid, out of range, or NULL.

Error Messages

None.

Related Information

- The `<semaphore.h>` file (see [Header Files for UNIX-Type Functions](#))
- [sem_close\(\)-Close Named Semaphore](#)
- [sem_destroy\(\)-Destroy Unnamed Semaphore](#)
- [sem_init\(\)-Initialize Unnamed Semaphore](#)
- [sem_init_np\(\)-Initialize Unnamed Semaphore with Maximum Value](#)
- [sem_open\(\)-Open Named Semaphore](#)
- [sem_open_np\(\)-Open Named Semaphore with Maximum Value](#)
- [sem_post\(\)-Post to Semaphore](#)
- [sem_post_np\(\)-Post Value to Semaphore](#)
- [sem_trywait\(\)-Try to Decrement Semaphore](#)
- [sem_unlink\(\)-Unlink Named Semaphore](#)
- [sem_wait\(\)-Wait for Semaphore](#)
- [sem_wait_np\(\)-Wait for Semaphore with Timeout](#)

Example

The following example retrieves the value of a semaphore before and after it is decremented by `sem_wait()`.

```
#include <stdio.h>
#include <semaphore.h>
main() {
    sem_t my_semaphore;
    int value;

    sem_init(&my_semaphore, 0, 10);
    sem_getvalue(&my_semaphore, &value);
    printf("The initial value of the semaphore is %d\n", value);
    sem_wait(&my_semaphore);
    sem_getvalue(&my_semaphore, &value);
    printf("The value of the semaphore after the wait is %d\n", value);
}
```

Output:

```
The initial value of the semaphore is 10
The value of the semaphore after the wait is 9
```

API introduced: V4R4

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

sem_init()-Initialize Unnamed Semaphore

Syntax

```
#include <semaphore.h>

int sem_init(sem_t * sem, int shared,
             unsigned int value);
```

Service Program Name: QP0ZPSEM

Default Public Authority: *USE

Threadsafe: Yes

The **sem_init()** function initializes an unnamed semaphore and sets its initial value. The maximum value of the semaphore is set to `SEM_VALUE_MAX`. The title for the semaphore is set to the character representation of the address of the semaphore. If an unnamed semaphore already exists at `sem`, then it will be destroyed and a new semaphore will be initialized.

Parameters

sem

(Input) A pointer to the storage of an uninitialized unnamed semaphore. The pointer must be aligned on a 16-byte boundary. This semaphore is initialized.

shared

(Input) An indication to the system of how the semaphore is going to be used. A value of zero indicates that the semaphore will be used only by threads within the current process. A nonzero value indicates that the semaphore may be used by threads from other processes.

value

(Input) The value used to initialize the value of the semaphore.

Authorities

None

Return Value

0 **sem_init()** was successful.

-1 **sem_init()** was not successful. The `errno` variable is set to indicate the error.

Error Conditions

If `sem_init()` is not successful, `errno` usually indicates one of the following errors. Under some conditions, `errno` could indicate an error other than those listed here.

[EINVAL]

The value specified for the argument is not correct.

A function was passed incorrect argument values, or an operation was attempted on an object and the operation specified is not supported for that type of object.

An argument value is not valid, out of range, or NULL.

The value parameter is greater than `SEM_VALUE_MAX`.

[ENOSPC]

No space available.

System semaphore resources have been exhausted.

Error Messages

None.

Related Information

- The `<semaphore.h>` file (see)
- [sem_destroy\(\)-Destroy Unnamed Semaphore](#)
- [sem_getvalue\(\)-Get Semaphore Value](#)
- [sem_init_np\(\)-Initialize Unnamed Semaphore with Maximum Value](#)
- [sem_post\(\)-Post to Semaphore](#)
- [sem_post_np\(\)-Post Value to Semaphore](#)
- [sem_trywait\(\)-Try to Decrement Semaphore](#)
- [sem_wait\(\)-Wait for Semaphore](#)
- [sem_wait_np\(\)-Wait for Semaphore with Timeout](#)

Example

The following example initializes an unnamed semaphore, `my_semaphore`, that will be used by threads of the current process. Its value is set to 10.

```
#include <semaphore.h>
main() {
    sem_t my_semaphore;
    int rc;

    rc = sem_init(&my_semaphore, 0, 10);
}
```

API introduced: V4R4

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

sem_init_np()-Initialize Unnamed Semaphore with Maximum Value

Syntax

```
#include <semaphore.h>

int sem_init_np(sem_t * sem, int shared,
                unsigned int value,
                sem_attr_np_t * attr);
```

Service Program Name: QP0ZPSEM

Default Public Authority: *USE

Threadsafe: Yes

The **sem_init_np()** function initializes an unnamed semaphore and sets its initial value. The **sem_init_np()** function uses the **attr** parameter to set the maximum value and title of the semaphore. If an unnamed semaphore already exists at **sem**, then it will be destroyed and a new semaphore will be initialized.

Parameters

sem

(Input) A pointer to the storage of an uninitialized unnamed semaphore. The pointer must be aligned on a 16-byte boundary. This semaphore is initialized.

shared

(Input) An indication to the system of how the semaphore is going to be used. A value of zero indicates that the semaphore will be used only by threads within the current process. A nonzero value indicates that the semaphore may be used by threads from other processes.

value

(Input) The value used to initialize the value of the semaphore.

attr

(Input) Attributes for the semaphore.

The members of the **sem_attr_np_t** structure are as follows.

unsigned int reserved1[1] A reserved field that must be set to zero.

unsigned int maxvalue The maximum value that the semaphore may obtain. **maxvalue** must be greater than zero. If a **sem_post()** or **sem_post_np()** operation would cause the value of a semaphore to exceed its maximum value, the operation will fail, returning **EINVAL**.

<i>unsigned int reserved2[2]</i>	A reserved field that must be set to zero.
<i>char title[16]</i>	The title of the semaphore. The title is a null-terminated string that contains up to 16 bytes. Any bytes after the null character are ignored. The title is retrieved using the Open List of Interprocess Communication Objects (QP0ZOLIP) API.
<i>void * reserved3[2]</i>	A reserved field that must be set to zero.

Authorities

None

Return Value

- 0* **sem_init_np()** was successful.
- 1* **sem_init_np()** was not successful. The `errno` variable is set to indicate the error.

Error Conditions

If **sem_init_np()** is not successful, `errno` usually indicates one of the following errors. Under some conditions, `errno` could indicate an error other than those listed here.

[EINVAL]

The value specified for the argument is not correct.

A function was passed incorrect argument values, or an operation was attempted on an object and the operation specified is not supported for that type of object.

An argument value is not valid, out of range, or NULL.

The value parameter is greater than the `maxvalue` field of the `attr` parameter.

The `maxvalue` field of the `attr` parameter is greater than `SEM_VALUE_MAX`.

The `maxvalue` field of the `attr` parameter is equal to zero.

The reserved fields of the `attr` argument are not set to zero.

[EFAULT]

The address used for an argument is not correct.

In attempting to use an argument in a call, the system detected an address that is not valid.

While attempting to access a parameter passed to this function, the system detected an address that is not valid.

[ENOSPC]

No space available.

The requested operations required additional space on the device and there is no space left. This could also be caused by exceeding the user profile storage limit when creating or transferring ownership of an object.

Insufficient space remains to hold the intended file, directory, or link.

System semaphore resources have been exhausted.

Error Messages

None.

Related Information

- The `<semaphore.h>` file (see [Header Files for UNIX-Type Functions](#))
- [sem_destroy\(\)-Destroy Unnamed Semaphore](#)
- [sem_getvalue\(\)-Get Semaphore Value](#)
- [sem_init\(\)-Initialize Unnamed Semaphore](#)
- [sem_post\(\)-Post to Semaphore](#)
- [sem_post_np\(\)-Post Value to Semaphore](#)
- [sem_trywait\(\)-Try to Decrement Semaphore](#)
- [sem_wait\(\)-Wait for Semaphore](#)
- [sem_wait_np\(\)-Wait for Semaphore with Timeout](#)

Example

The following example initializes an unnamed semaphore, `my_semaphore`, that will be used by threads of the current process and sets its value to 10. The maximum value and title of the semaphore are set to 10 and "MYSEM".

```
#include <semaphore.h>
main() {
    sem_t my_semaphore;
    sem_attr_np_t attr;
    int rc;
```

```
memset(&attr, 0, sizeof(attr));
attr.maxvalue = 10;
strcpy(attr.title, "MYSEM");
rc = sem_init_np(&my_semaphore, 0, 10, &attr);
}
```

API introduced: V4R4

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

sem_open()--Open Named Semaphore

Syntax

```
#include <semaphore.h>

sem_t * sem_open(const char *name, int oflag, ...);
```

Service Program Name: QP0ZPSEM

Default Public Authority: *USE

Threadsafe: Yes

The **sem_open()** function opens a named semaphore, returning a semaphore pointer that may be used on subsequent calls to **sem_post()**, **sem_post_np()**, **sem_wait()**, **sem_wait_np()**, **sem_trywait()**, **sem_getvalue()**, and **sem_close()**. When a semaphore is being created, the parameters *mode* and *value* must be specified on the call to **sem_open()**. If a semaphore is created, then the maximum value of the semaphore is set to SEM_VALUE_MAX and the title of the semaphore is set to the last 16 characters of the name.

If **sem_open()** is called multiple times within the same process using the same name, **sem_open()** will return a pointer to the same semaphore, as long as another process has not used **sem_unlink()** to unlink the semaphore.

If **sem_open()** is called from a program using data model LLP64, the returned semaphore pointer must be declared as a sem_t *__ptr128.

Parameters

name

(Input) A pointer to the null-terminated name of the semaphore to be opened. The name should begin with a slash (/) character. If the name does not begin with a slash (/) character, the system adds a slash to the beginning of the name.

This parameter is assumed to be represented in the CCSID (coded character set identifier) currently in effect for the job. If the CCSID of the job is 65535, this parameter is assumed to be represented in the default CCSID of the job.

The name is added to a set of names that is used only by named semaphores. The name has no relationship to any file system path names. The maximum length of the name is SEM_NAME_MAX.

See [QlgSem_open\(\)--Open Named Semaphore \(using NLS-enabled path name\)](#) for a description and an example of supplying the *name* in any CCSID.

oflag

(Input) Option flags.

The *oflag* parameter value is either zero or is obtained by performing an OR operation on one or more of the following constants:

'0x0008' or O_CREAT Creates the named semaphore if it does not already exist.

'0x0010' or O_EXCL Causes **sem_open()** to fail if O_CREAT is also set and the named semaphore already exists.

mode

(input) Permission flags.

The *mode* parameter value is either zero or is obtained by performing an OR operation on one or more of the following list of constants. For another process to open the semaphore, the process's effective UIDd must be able to open the semaphore in both read and write mode.

'0x0100' or S_IRUSR Permits the creator of the named semaphore to open the semaphore in read mode.

'0x0080' or S_IWUSR Permits the creator of the named semaphore to open the semaphore in write mode.

'0x0020' or S_IRGRP Permits the group associated with the named semaphore to open the semaphore in read mode.

'0x0010' or S_IWGRP Permits the group associated with the named semaphore to open the semaphore in write mode.

'0x0004' or S_IROTH Permits others to open the named semaphore in read mode.

'0x0002' or S_IWOTH Permits others to open the named semaphore in write mode.

value

(Input) Initial value of the named semaphore.

Authorities

Authorization required for sem_open()

Object Referred to	Authority Required	errno
Named semaphore to be created	None	None
Existing named semaphore to be accessed	*RW	EACCES

Return Value

value **sem_open()** was successful. The value returned is a pointer to the open named semaphore.

SEM_FAILED **sem_open()** was not successful. The *errno* variable is set to indicate the error.

Error Conditions

If **sem_open()** is not successful, *errno* usually indicates one of the following errors. Under some conditions, *errno* could indicate an error other than those listed here.

[EACCES]

Permission denied.

An attempt was made to access an object in a way forbidden by its object access permissions.

[EEXIST]

Semaphore exists.

A named semaphore exists for the parameter *name*, but O_CREAT and O_EXCL are both set in *oflag*.

[EINVAL]

The value specified for the argument is not correct.

A function was passed incorrect argument values, or an operation was attempted on an object and the operation specified is not supported for that type of object.

An argument value is not valid, out of range, or NULL.

The *value* parameter is greater than SEM_VALUE_MAX.

[ENAMETOOLONG]

The name is too long. The name is longer than the SEM_NAME_MAX characters.

[ENOENT]

No such path or directory.

The name specified on the **sem_open()** call does not refer to an existing named semaphore and O_CREAT was not set in *oflag*.

[ENOSPC]

No space available.

The requested operations required additional space on the device and there is no space left. This also could be caused by exceeding the user profile storage limit when creating or transferring ownership of an object.

Insufficient space remains to hold the intended file, directory, or link.

API introduced: V4R4

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

sem_open_np()--Open Named Semaphore with Maximum Value

Syntax

```
#include <semaphore.h>

sem_t * sem_open_np(const char *name, int oflag,
                   mode_t mode, unsigned int value,
                   sem_attr_np_t * attr);
```

Service Program Name: QP0ZPSEM

Default Public Authority: *USE

Threadsafe: Yes

The **sem_open_np()** function opens a named semaphore, returning a semaphore pointer that may be used on subsequent calls to **sem_post()**, **sem_post_np()**, **sem_wait()**, **sem_wait_np()**, **sem_trywait()**, **sem_getvalue()**, and **sem_close()**. If a named semaphore is being created, the parameters *mode*, *value*, and *attr* are used to set the permissions, value, and maximum value of the created semaphore.

If **sem_open_np()** is called multiple times within the same process using the same name, **sem_open_np()** will return a pointer to the same semaphore, as long as another process has not used **sem_unlink()** to unlink the semaphore.

If **sem_open_np()** is called from a program using data model LLP64, the returned semaphore pointer must be declared as a `sem_t * __ptr128`.

Parameters

name

(Input) A pointer to the null-terminated name of the semaphore to be opened. The name should begin with a slash (/) character. If the name does not begin with a slash (/) character, the system adds a slash to the beginning of the name.

This parameter is assumed to be represented in the CCSID (coded character set identifier) currently in effect for the job. If the CCSID of the job is 65535, this parameter is assumed to be represented in the default CCSID of the job.

The name is added to a set of names used by named semaphores only. The name has no relationship to any file system path names. The maximum length of the name is SEM_NAME_MAX.

See [QlgSem_open_np\(\)--Open Named Semaphore with Maximum Value](#) (using NLS-enabled path name) for a description and an example of supplying the *name* in any CCSID.

oflag

(Input) Option flags.

The *oflag* parameter value is either zero or is obtained by performing an OR operation on one or more of the following constants:

'0x0008' or O_CREAT Creates the named semaphore if it does not already exist.

'0x0010' or O_EXCL Causes **sem_open_np()** to fail if O_CREAT is also set and the named semaphore already exists.

mode

(input) Permission flags.

The *mode* parameter value is either zero or is obtained by performing an OR operation on one or more of the list of constants. For another process to open the semaphore, the process's effective UID must be able to open the semaphore in both read and write mode.

'0x0100' or S_IRUSR Permits the creator of the named semaphore to open the semaphore in read mode.

'0x0080' or S_IWUSR Permits the creator of the named semaphore to open the semaphore in write mode.

'0x0020' or S_IRGRP Permits the group associated with the named semaphore to open the semaphore in read mode.

'0x0010' or S_IWGRP Permits the group associated with the named semaphore to open the semaphore in write mode.

'0x0004' or S_IROTH Permits others to open the named semaphore in read mode.

'0x0002' or S_IWOTH Permits others to open the named semaphore in write mode.

value

(Input) The initial value of the named semaphore.

attr

(Input) Attributes for the semaphore.

The members of the `sem_attr_np_t` structure are as follows:

unsigned int reserved1[1] A reserved field that must be set to zero.

unsigned int maxvalue The maximum value that the semaphore may obtain. *maxvalue* must be greater than zero. If a **sem_post()** or **sem_post_np()** operation would cause the value of a semaphore to exceed its maximum value, the operation will fail, returning EINVAL.

unsigned int reserved2[1] A reserved field that must be set to zero.

char title[16] The title of the semaphore. The title is a null-terminated string that has a maximum length of 16 bytes. The string is associated with the semaphore. If the first byte is zero, then the system assigns a title to the semaphore that is based on the semaphore name. The title is retrieved using the Open List of Interprocess Communication Objects (QP0ZOLIP) API.

*void * reserved3[2]* A reserved field that must be set to zero.

Authorities

Authorization required for sem_open_np()

Object Referred to	Authority Required	errno
Named semaphore to be created	None	None
Existing named semaphore to be accessed	*RW	EACCES

Return Value

value **sem_open_np()** was successful. The value returned is a pointer to the opened named semaphore.

SEM_FAILED **sem_open_np()** was not successful. The *errno* variable is set to indicate the error.

Error Conditions

If **sem_open_np()** is not successful, *errno* usually indicates one of the following errors. Under some conditions, *errno* could indicate an error other than those listed here.

[EACCES]

Permission denied.

An attempt was made to access an object in a way forbidden by its object access permissions.

[EEXIST]

A named semaphore exists for the parameter *name*, but `O_CREAT` and `O_EXCL` are both set in *oflag*.

[EFAULT]

The address used for an argument is not correct.

In attempting to use an argument in a call, the system detected an address that is not valid.

While attempting to access a parameter passed to this function, the system detected an address that is not valid.

[EINVAL]

The value specified for the argument is not correct.

A function was passed incorrect argument values, or an operation was attempted on an object and the operation specified is not supported for that type of object.

An argument value is not valid, out of range, or NULL.

The *maxvalue* field of the *attr* argument is greater than `SEM_VALUE_MAX`.

The *maxvalue* field of the *attr* argument is equal to zero.

The *value* argument is greater than the *maxvalue* field of the *attr* argument.

The reserved fields of the *attr* argument are not set to zero.

[ENAMETOOLONG]

The name is too long. The name is longer than the `SEM_NAME_MAX` characters.

[ENOENT]

No such path or directory.

The directory or a component of the path name specified does not exist.

A named file or directory does not exist or is an empty string.

The name specified on the `sem_open_np()` call does not refer to an existing named semaphore and `O_CREAT` was not set in *oflag*.

[ENOSPC]

No space available.

The requested operations required additional space on the device and there is no space left. This also could be caused by exceeding the user profile storage limit when creating or transferring ownership of an object.

Insufficient space remains to hold the intended file, directory, or link.

System semaphore resources have been exhausted.

Error Messages

None.

Related Information

- The `<semaphore.h>` file (see [Header Files for UNIX-Type Functions](#))
- [OlgSem_open_np\(\)](#)--Open Named Semaphore with Maximum Value (using NLS-enabled path name)
- [sem_close\(\)](#)--Close Named Semaphore
- [sem_getvalue\(\)](#)--Get Semaphore Value
- [sem_open\(\)](#)--Open Named Semaphore
- [sem_post\(\)](#)--Post to Semaphore
- [sem_post_np\(\)](#)--Post Value to Semaphore
- [sem_trywait\(\)](#)--Try to Decrement Semaphore
- [sem_unlink\(\)](#)--Unlink Named Semaphore
- [sem_wait\(\)](#)--Wait for Semaphore
- [sem_wait_np\(\)](#)--Wait for Semaphore with Timeout>

Example

The following example opens the named semaphore `"/mysemaphore"` and creates the semaphore with an initial value of 10 and a maximum value of 11. The permissions are set such that only the current user has access to the semaphore.

```
#include <semaphore.h>
main() {
    sem_t * my_semaphore;
    int rc;
    sem_attr_np_t attr;

    memset(&attr, 0, sizeof(attr));
    attr.maxvalue=11;
    my_semaphore = sem_open_np("/mysemaphore",
                              O_CREAT|O_EXCL,
                              S_IRUSR | S_IWUSR,
```



```
    10,  
    &attr);  
}
```

API introduced: V4R4

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

sem_post()-Post to Semaphore

Syntax

```
#include <semaphore.h>

int sem_post(sem_t * sem);
```

Service Program Name: QP0ZPSEM

Default Public Authority: *USE

Threadsafe: Yes

The **sem_post()** function posts to a semaphore, incrementing its value by one. If the resulting value is greater than zero and if there is a thread waiting on the semaphore, the waiting thread decrements the semaphore value by one and continues running.

Parameters

sem

(Input) A pointer to an initialized unnamed semaphore or opened named semaphore.

Authorities

None

Return Value

0 **sem_post()** was successful.

-1 **sem_post()** was not successful. The `errno` variable is set to indicate the error.

Error Conditions

If **sem_post()** is not successful, `errno` usually indicates one of the following errors. Under some conditions, `errno` could indicate an error other than those listed here.

[EINVAL]

The value specified for the argument is not correct.

A function was passed incorrect argument values, or an operation was attempted on an object and

the operation specified is not supported for that type of object.

An argument value is not valid, out of range, or NULL.

Posting to the semaphore would cause its value to exceed its maximum value. The maximum value is SEM_VALUE_MAX or was set using `sem_open_np()` or `sem_init_np()`.

Error Messages

None.

Related Information

- The `<semaphore.h>` file (see [Header Files for UNIX-Type Functions](#))
- [sem_close\(\)-Close Named Semaphore](#)
- [sem_destroy\(\)-Destroy Unnamed Semaphore](#)
- [sem_getvalue\(\)-Get Semaphore Value](#)
- [sem_init\(\)-Initialize Unnamed Semaphore](#)
- [sem_open\(\)-Open Named Semaphore](#)
- [sem_open_np\(\)-Open Named Semaphore with Maximum Value](#)
- [sem_post_np\(\)-Post Value to Semaphore](#)
- [sem_trywait\(\)-Try to Decrement Semaphore](#)
- [sem_unlink\(\)-Unlink Named Semaphore](#)
- [sem_wait\(\)-Wait for Semaphore](#)
- [sem_wait_np\(\)-Wait for Semaphore with Timeout](#)

Example

The following example initializes an unnamed semaphore and posts to it, incrementing its value by 1.

```
#include <stdio.h>
#include <semaphore.h>
```

```
main() {
    sem_t my_semaphore;
    int value;

    sem_init(&my_semaphore, 0, 10);
    sem_getvalue(&my_semaphore, &value);
    printf("The initial value of the semaphore is %d\n", value);
    sem_post(&my_semaphore);
    sem_getvalue(&my_semaphore, &value);
    printf("The value of the semaphore after the post is %d\n", value);
}
```

Output:

```
The initial value of the semaphore is 10
The value of the semaphore after the post is 11
```

API introduced: V4R4

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

sem_post_np()-Post Value to Semaphore

Syntax

```
#include <semaphore.h>

int sem_post_np(sem_t * sem,
                sem_post_options_np_t *options);
```

Service Program Name: QP0ZPSEM

Default Public Authority: *USE

Threadsafe: Yes

The **sem_post_np()** function posts to a semaphore, incrementing its value by the increment specified in the options parameter. If the resulting value is greater than zero and if there are threads waiting on the semaphore, the waiting threads decrement the semaphore and continue running.

Parameters

sem

(Input) A pointer to an initialized unnamed semaphore or opened named semaphore.

options

(Input) Post options.

The members of the `sem_post_options_np_t` structure are as follows.

unsigned int reserved1[1] A reserved field that must be set to zero.

unsigned int increment The value, greater than zero, used to increment the semaphore. If the value specified causes the value of a semaphore to exceed its maximum value, **sem_post_np()** will fail by returning [EINVAL].

unsigned int reserved2[2] A reserved field that must be set to zero.

Authorities

None

Return Value

0 `sem_post_np()` was successful.

-1 `sem_post_np()` was not successful. The `errno` variable is set to indicate the error.

Error Conditions

If `sem_post_np()` is not successful, `errno` usually indicates one of the following errors. Under some conditions, `errno` could indicate an error other than those listed here.

[EINVAL]

The value specified for the argument is not correct.

A function was passed incorrect argument values, or an operation was attempted on an object and the operation specified is not supported for that type of object.

An argument value is not valid, out of range, or NULL.

[EOVERFLOW]

Maximum value exceeded.

Posting to the semaphore would cause its value to exceed its maximum value. The maximum value is `SEM_VALUE_MAX` or was set using `sem_open_np()` or `sem_init_np()`.

The reserved fields of the `attr` argument are not set to zero.

Error Messages

None.

Related Information

- The `<semaphore.h>` file (see [Header Files for UNIX-Type Functions](#))
- [sem_close\(\)-Close Named Semaphore](#)
- [sem_destroy\(\)-Destroy Unnamed Semaphore](#)
- [sem_getvalue\(\)-Get Semaphore Value](#)
- [sem_init\(\)-Initialize Unnamed Semaphore](#)
- [sem_init_np\(\)-Initialize Unnamed Semaphore with Maximum Value](#)
- [sem_open\(\)-Open Named Semaphore](#)

- [sem_open_np\(\)-Open Named Semaphore with Maximum Value](#)
- [sem_post\(\)-Post to Semaphore](#)
- [sem_trywait\(\)-Try to Decrement Semaphore](#)
- [sem_unlink\(\)-Unlink Named Semaphore](#)
- [sem_wait\(\)-Wait for Semaphore](#)
- [sem_wait_np\(\)-Wait for Semaphore with Timeout](#)

Example

The following example initializes an unnamed semaphore and posts to it, incrementing its value by 2.

```
#include <stdio.h>
#include <semaphore.h>
main() {
    sem_t my_semaphore;
    sem_post_options_np_t options;
    int value;

    sem_init(&my_semaphore, 0, 10);
    sem_getvalue(&my_semaphore, &value);
    printf("The initial value of the semaphore is %d.\n", value);
    memset(&options, 0, sizeof(options));
    options.increment=2;
    sem_post_np(&my_semaphore,&options);
    sem_getvalue(&my_semaphore, &value);
    printf("The value of the semaphore after the post is %d.\n", value);
}
```

Output:

The initial value of the semaphore is 10.
The value of the semaphore after the post is 12.

API introduced: V4R4

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

sem_trywait()-Try to Decrement Semaphore

Syntax

```
#include <semaphore.h>

int sem_trywait(sem_t * sem);
```

Service Program Name: QP0ZPSEM

Default Public Authority: *USE

Threadsafe: Yes

The **sem_trywait()** function attempts to decrement the value of the semaphore. The semaphore will be decremented if its value is greater than zero. If the value of the semaphore is zero, then **sem_trywait()** will return -1 and set errno to EAGAIN.

Parameters

sem

(Input) A pointer to an initialized unnamed semaphore or opened named semaphore.

Authorities

None

Return Value

0 **sem_trywait()** was successful.

-1 **sem_trywait()** was not successful. The errno variable is set to indicate the error.

Error Conditions

If **sem_trywait()** is not successful, errno usually indicates one of the following errors. Under some conditions, errno could indicate an error other than those listed here.

[EAGAIN]

Operation would have caused the process to be suspended.

The value of the semaphore is currently zero and cannot be decremented.

[EINTR]

Interrupted function call.

[EINVAL]

The value specified for the argument is not correct.

A function was passed incorrect argument values, or an operation was attempted on an object and the operation specified is not supported for that type of object.

An argument value is not valid, out of range, or NULL.

Error Messages

None.

Related Information

- The <[semaphore.h](#)> file (see [Header Files for UNIX-Type Functions](#))
- [sem_close\(\)-Close Named Semaphore](#)
- [sem_destroy\(\)-Destroy Unnamed Semaphore](#)
- [sem_getvalue\(\)-Get Semaphore Value](#)
- [sem_init\(\)-Initialize Unnamed Semaphore](#)
- [sem_init_np\(\)-Initialize Unnamed Semaphore with Maximum Value](#)
- [sem_open\(\)-Open Named Semaphore](#)
- [sem_open_np\(\)-Open Named Semaphore with Maximum Value](#)
- [sem_post\(\)-Post to Semaphore](#)
- [sem_post_np\(\)-Post Value to Semaphore](#)
- [sem_unlink\(\)-Unlink Named Semaphore](#)
- [sem_wait\(\)-Wait for Semaphore](#)
- [sem_wait_np\(\)-Wait for Semaphore with Timeout](#)

Example

The following example attempts to decrement a semaphore with a current value of zero.

```
#include <stdio.h>
#include <errno.h>
#include <semaphore.h>
main() {
    sem_t my_semaphore;
    int value;
    int rc;

    sem_init(&my_semaphore, 0, 1);
    sem_getvalue(&my_semaphore, &value);
    printf("The initial value of the semaphore is %d\n", value);
    sem_wait(&my_semaphore);
    sem_getvalue(&my_semaphore, &value);
    printf("The value of the semaphore after the wait is %d\n", value);
    rc = sem_trywait(&my_semaphore);
    if ((rc == -1) && (errno == EAGAIN)) {
        printf("sem_trywait did not decrement the semaphore\n");
    }
}
```

Output:

```
The initial value of the semaphore is 1
The value of the semaphore after the wait is 0
sem_trywait did not decrement the semaphore
```

API introduced: V4R4

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

sem_unlink()--Unlink Named Semaphore

Syntax

```
#include <semaphore.h>

int sem_unlink(const char *name);
```

Service Program Name: QP0ZPSEM

Default Public Authority: *USE

Threadsafe: Yes

The **sem_unlink()** function unlinks a named semaphore. The name of the semaphore is removed from the set of names used by named semaphores. If the semaphore is still in use, the semaphore is not deleted until all processes using the semaphore have ended or have called **sem_close()**. Using the name of an unlinked semaphore in subsequent calls to **sem_open()** or **sem_open_np()** will result in the creation of a new semaphore with the same name if the `O_CREAT` flag of the *oflag* parameter has been set.

Parameters

name

(Input) A pointer to the null-terminated name of the semaphore to be unlinked. The name should begin with a slash (/) character. If the name does not begin with a slash (/) character, the system adds a slash to the beginning of the name.

This parameter is assumed to be represented in the coded character set identifier (CCSID) currently in effect for the job. If the CCSID of the job is 65535, this parameter is assumed to be represented in the default CCSID of the job.

The name is present in a set of names used only by named semaphores. The name has no relation to any file system path names. The maximum length of the name is `SEM_NAME_MAX`.

See [QlgSem_unlink\(\)--Unlink Named Semaphore \(using NLS-enabled path name\)](#) for a description and an example of supplying the *name* in any CCSID.

Authorities

Authorization required for sem_unlink()

Object Referred to	Authority Required	errno
Named semaphore to be deleted	See note	EACCES

Note: To unlink a named semaphore, the effective UID of the process must be the creator of the semaphore

or the process must have *ALLOBJ authority.

Return Value

0 **sem_unlink()** was successful.

-1 **sem_unlink()** was not successful. The *errno* variable is set to indicate the error.

Error Conditions

If **sem_unlink()** is not successful, *errno* usually indicates one of the following errors. Under some conditions, *errno* could indicate an error other than those listed here.

[EACCES]

Permission denied.

An attempt was made to access an object in a way forbidden by its object access permissions.

[ENOENT]

No such path or directory.

The specified name doesnot refer to an existing named semaphore.

[EFAULT]

The address used for an argument is not correct.

In attempting to use an argument in a call, the system detected an address that is not valid.

While attempting to access a parameter passed to this function, the system detected an address that is not valid.

[ENAMETOOLONG]

The name is too long. The name is longer than the SEM_NAME_MAX characters.

Error Messages

None.

Related Information

- The <semaphore.h> file (see [Header Files for UNIX-Type Functions](#))
- [QlgSem_unlink\(\)](#)--Unlink Named Semaphore (using NLS-enabled path name)
- [sem_open\(\)](#)--Open Named Semaphore

- [sem_open_np\(\)](#)--Open Named Semaphore with Maximum Value

Example

The following example unlinks the named semaphore "/mysem".

```
#include <semaphore.h>
main() {
    int rc;

    rc = sem_unlink("/mysem");
}
```

API introduced: V4R4

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

sem_wait()-Wait for Semaphore

Syntax

```
#include <semaphore.h>

int sem_wait(sem_t * sem);
```

Service Program Name: QP0ZPSEM

Default Public Authority: *USE

Threadsafe: Yes

The **sem_wait()** function decrements by one the value of the semaphore. The semaphore will be decremented when its value is greater than zero. If the value of the semaphore is zero, then the current thread will block until the semaphore's value becomes greater than zero.

Parameters

sem

(Input) A pointer to an initialized unnamed semaphore or opened named semaphore.

Authorities

None

Return Value

0 **sem_wait()** was successful.

-1 **sem_wait()** was not successful. The errno variable is set to indicate the error.

Error Conditions

If **sem_wait()** is not successful, errno usually indicates one of the following errors. Under some conditions, errno could indicate an error other than those listed here.

[EINTR]

Interrupted function call.

[EINVAL]

The value specified for the argument is not correct.

A function was passed incorrect argument values, or an operation was attempted on an object and the operation specified is not supported for that type of object.

An argument value is not valid, out of range, or NULL.

Error Messages

None.

Related Information

- The `<semaphore.h>` file (see [Header Files for UNIX-Type Functions](#))
- [sem_close\(\)-Close Named Semaphore](#)
- [sem_destroy\(\)-Destroy Unnamed Semaphore](#)
- [sem_getvalue\(\)-Get Semaphore Value](#)
- [sem_init\(\)-Initialize Unnamed Semaphore](#)
- [sem_init_np\(\)-Initialize Unnamed Semaphore with Maximum Value](#)
- [sem_open\(\)-Open Named Semaphore](#)
- [sem_open_np\(\)-Open Named Semaphore with Maximum Value](#)
- [sem_post\(\)-Post to Semaphore](#)
- [sem_post_np\(\)-Post Value to Semaphore](#)
- [sem_trywait\(\)-Try to Decrement Semaphore](#)
- [sem_unlink\(\)-Unlink Named Semaphore](#)
- [sem_wait_np\(\)-Wait for Semaphore with Timeout](#)

Example

The following example creates a semaphore with an initial value of 10. The value is decremented by calling `sem_wait()`.

```
#include <stdio.h>
#include <semaphore.h>
main() {
    sem_t my_semaphore;
    int value;

    sem_init(&my_semaphore, 0, 1);
    sem_getvalue(&my_semaphore, &value);
    printf("The initial value of the semaphore is %d\n", value);
    sem_wait(&my_semaphore);
    sem_getvalue(&my_semaphore, &value);
    printf("The value of the semaphore after the wait is %d\n", value);
}
```

Output:

```
The initial value of the semaphore is 1
The value of the semaphore after the wait is 0
```

API introduced: V4R4

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

sem_wait_np()-Wait for Semaphore with Timeout

Syntax

```
#include <semaphore.h>

int sem_wait_np(sem_t * sem,
                sem_wait_options_np_t * options);
```

Service Program Name: QP0ZPSEM

Default Public Authority: *USE

Threadsafe: Yes

The **sem_wait_np()** function attempts to decrement by one the value of the semaphore. The semaphore will be decremented by one when its value is greater than zero. If the value of the semaphore is zero, then the current thread will block until the semaphore's value becomes greater than zero or until the timeout period specified on the options parameter has ended. If the semaphore is not decremented before the timeout ends, **sem_wait_np()** will return with an error, setting `errno` to `[ETIMEDOUT]`.

Parameters

sem

(Input) A pointer to an initialized unnamed semaphore or opened named semaphore.

options

(Input) A pointer to a semaphore wait (`sem_wait_options_np_t`) structure. The members of the `sem_wait_options_np_t` structure are as follows:

unsigned int reserved1[2] A reserved field that must be set to zero.

struct sem_timeout_t timeout The time, in MI time, that **sem_wait_np()** should wait for the semaphore. If the timeout is zero, **sem_wait_np()** will return immediately with `errno` set to `[ETIMEDOUT]` if the semaphore cannot be decremented. If a timeout value of `0xFFFFFFFF FFFFFFFF` is specified, then **sem_wait_np()** will wait indefinitely. The maximum timeout that may be specified is `281 272 976 710 655 (2 ** 48 - 1)` microseconds. Any value larger than this, other than `0xFFFFFFFF FFFFFFFF`, will cause **sem_wait_np()** to wait for the maximum timeout (`281 272 976 710 655` microseconds). The `Qp0zCvtToMITime()` may be used to convert a `timeval` structure to the corresponding MI time.

Authorities

None

Return Value

0 `sem_wait_np()` was successful.

-1 `sem_wait_np()` was not successful. The `errno` variable is set to indicate the error.

Error Conditions

If `sem_wait_np()` is not successful, `errno` usually indicates one of the following errors. Under some conditions, `errno` could indicate an error other than those listed here.

[ECANCEL]

Operation canceled.

[EDESTROYED]

The semaphore was destroyed.

[EINTR]

Interrupted function call.

[EINVAL]

The value specified for the argument is not correct.

A function was passed incorrect argument values, or an operation was attempted on an object and the operation specified is not supported for that type of object.

An argument value is not valid, out of range, or NULL.

[ETIMEDOUT]

A remote host did not respond within the timeout period.

Error Messages

None.

Related Information

- The `<semaphore.h>` file (see [Header Files for UNIX-Type Functions](#))
- [sem_close\(\)-Close Named Semaphore](#)
- [sem_destroy\(\)-Destroy Unnamed Semaphore](#)
- [sem_getvalue\(\)-Get Semaphore Value](#)
- [sem_init\(\)-Initialize Unnamed Semaphore](#)
- [sem_init_np\(\)-Initialize Unnamed Semaphore with Maximum Value](#)
- [sem_open\(\)-Open Named Semaphore](#)
- [sem_open_np\(\)-Open Named Semaphore with Maximum Value](#)
- [sem_post\(\)-Post to Semaphore](#)
- [sem_post_np\(\)-Post Value to Semaphore](#)
- [sem_trywait\(\)-Try to Decrement Semaphore](#)
- [sem_unlink\(\)-Unlink Named Semaphore](#)
- [sem_wait_np\(\)-Wait for Semaphore with Timeout](#)

Example

The following example creates a semaphore with an initial value of 1. The value is decremented using `sem_wait()`. The program then attempts to decrement the semaphore using `sem_wait_np()` with a timeout of 2 seconds. This will fail with `ETIMEDOUT` because the semaphore's value is currently zero.

```
#include <stdio.h>
#include <errno.h>
#include <semaphore.h>
#include <time.h>
#include <qp0z1170.h>

main() {
    sem_t my_semaphore;
    int value;
    sem_wait_options_np_t options;
    int rc;
    struct timeval waittime;
```

```

time_t start_time;
time_t end_time;

sem_init(&my_semaphore, 0, 1);
sem_getvalue(&my_semaphore, &value);
printf("The initial value of the semaphore is %d\n", value);
sem_wait(&my_semaphore);
sem_getvalue(&my_semaphore, &value);
printf("The value of the semaphore after the wait is %d\n", value);
memset(&options, 0, sizeof(options));
waittime.tv_sec = 2;
waittime.tv_usec = 0;
Qp0zCvtToMITime((unsigned char *) &options.timeout,
                wait_time,
                QP0Z_CVTTIME_TO_OFFSET);
time(&start_time);
rc = sem_wait_np(&my_semaphore, &options);
time(&end_time);
if ((rc == -1) && (errno == ETIMEDOUT)) {
    printf("sem_wait_np timed out after %d seconds\n",
          end_time - start_time);
}
}

```

Output:

```

The initial value of the semaphore is 1
The value of the semaphore after the wait is 0
sem_wait_np timed out after 2 seconds

```

API introduced: V4R4

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

shmat()-Attach Shared Memory Segment to Current Process

Syntax

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

void *shmat(int shmid, const void *shmaddr,
            int shmflg);
```

Service Program Name: QP0ZUSHR

Default Public Authority: *USE

Threadsafe: Yes

The **shmat()** function returns the address of the shared memory segment associated with the specified shared memory identifier.

Parameters

shmid

(Input) Shared memory identifier.

shmaddr

(Input) Shared memory address. The address at which the calling thread would like the shared memory segment attached.

shmflg

(Input) Operations flags.

The value of the shmflg parameter is either zero or the following constant:

'0x1000' or SHM_RDONLY Places the shared memory segment in read-only memory. This flag is valid only for teraspace shared memory segments.

Authorities

Figure 1-14. Authorization Required for shmat()

Object Referred to	Authority Required	errno
Shared memory segment to be attached in read/write memory	Read and Write	EACCES

Shared memory segment to be attached in read-only memory in a process's teraspace.	Read	EACCES
--	------	--------

Return Value

value **shmat()** was successful. The value returned is a pointer to the shared memory segment associated with the specified identifier.

NULL **shmat()** was not successful. The *errno* variable is set to indicate the error.

Error Conditions

If **shmat()** is not successful, *errno* usually indicates one of the following errors. Under some conditions, *errno* could indicate an error other than those listed here.

[EACCES]

Permission denied.

An attempt was made to access an object in a way forbidden by its object access permissions.

The thread does not have access to the specified file, directory, component, or path.

Operation permission is denied to the calling thread.

Shared memory operations are not permitted because the QSHRMEMCTL system value is set to 0.

[EADDRINUSE]

A damaged object was encountered.

Address already in use.

An attempt was made to attach to a SHM_MAP_FIXED_NP teraspace shared memory segment, but the address range is not available in the teraspace of the current process.

[EDAMAGE]

A damaged object was encountered.

The value of *shmid* corresponds to a shared memory ID that has been marked as damaged by a previous shared memory operation.

[EFAULT]

The address used for an argument is not correct.

In attempting to use an argument in a call, the system detected an address that is not valid.

While attempting to access a parameter passed to this function, the system detected an address that is not valid.

[EINVAL]

The value specified for the argument is not correct.

A function was passed incorrect argument values, or an operation was attempted on an object and

the operation specified is not supported for that type of object.

An argument value is not valid, out of range, or NULL.

The value of `shmid` is not a valid shared memory identifier.

[EOPNOTSUPP]

Operation not supported.

The operation, though supported in general, is not supported for the requested object or the requested arguments.

The value of `(shmflg & SHM_RDONLY)` is not zero. (& is a bitwise AND.) Read-only shared memory segments are not supported for nonteraspace shared memory segments. Read-only shared memory segments are not supported for shared memory segments created using the `SHM_MAP_FIXED_NP` option of `shmget()`.

[ENOMEM]

Storage allocation request failed.

A function needed to allocate storage, but no storage is available.

The available data space is not large enough to accommodate the shared memory segment.

[EUNKNOWN]

Unknown system state.

The operation failed because of an unknown system state. See any messages in the job log and correct any errors that are indicated, then retry the operation.

Error Messages

None.

Usage Notes

The IPC implementation has the following restrictions over the X/Open single UNIX specification (formerly Spec 1170) definition:

1. The address specified by `shmaddr` is only used when `shmat()` is called from a program that uses data model LLP64 and attaches to a teraspace shared memory segment. Otherwise the address specified by `shmaddr` is ignored and the actual shared memory segment address is returned regardless of the value of `shmaddr`.
2. The only supported operation flag is `SHM_READONLY`. This operation flag is supported only when you attach to a teraspace shared memory segment. If `shmflg` specifies `SHM_RDONLY` for a nonteraspace shared memory segment, then an `[EOPNOTSUPP]` error is returned. All other values for `shmflg` are ignored.
3. A module that was not created with teraspace memory enabled should not attach to a teraspace shared memory segment. The call to `shmat()` will succeed and return a pointer. Any attempt, however, by a module not created with teraspace memory enabled to use the returned pointer will result in an `MCH3601` (Pointer not set for location referenced) exception.

4. When a process attaches to a shared memory segment that was created using `SHM_MAP_FIXED_NP`, an address range within the process's teraspace is used for the shared memory mapping. When a subsequent process attaches to the shared memory segment, the same address range within its teraspace must be available. If the address range is not available, the call to `shmat()` will fail with an `[EADDRINUSE]` error.
5. The storage for a shared memory segment is allocated when the first process attaches to the shared memory segment. The storage is charged against the process's temporary storage limit. If the process does not have enough temporary storage to satisfy the request, the call to `shmat()` will fail with an `[ENOMEM]` error.

Related Information

- The `<sys/shm.h>` file (see [Header Files for UNIX-Type Functions](#))
- [shmctl\(\)-Perform Shared Memory Control Operations](#)
- [shmget\(\)-Get ID of Shared Memory Segment with Key](#)
- [shmdt\(\)-Detach Shared Memory Segment from Calling Process](#)

Example

For an example of using this function, see Using Semaphores and Shared Memory in [Examples](#).

API introduced: V3R6

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

shmctl()-Perform Shared Memory Control Operations

Syntax

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

```
int shmctl(int shmid, int cmd, struct shm_id *buf);
```

Service Program Name: QP0ZUSHR

Default Public Authority: *USE

Threadsafe: Yes

The **shmctl()** function provides shared memory control operations as specified by **cmd** on the shared memory segment specified by **shmid**.

Parameters

shmid

(Input) Shared memory identifier, a positive integer. It is created by the **shmget()** function and used to identify the shared memory segment on which to perform the control operation.

cmd

(Input) Command, the control operation to perform on the shared memory segment.

buf

(I/O) Pointer to the **shm_id** structure to be used to get or set shared memory information.

The **cmd** parameter can have one of the following values:

'0x0002' or IPC_STAT Place the current value of each member of the **shm_id** data structure associated with **shmid** into the structure pointed to by **buf**. This command requires read permission.

'0x0001' or *IPC_SET* Set the value of the following members of the `shmid_ds` data structure associated with `shmid` to the corresponding value found in the structure pointed to by `buf`:

- `shm_perm.uid`
- `shm_perm.gid`
- `shm_perm.mode`

IPC_SET can be performed only by a thread with appropriate privileges or one that has an effective user ID equal to the value of `shm_perm.cuid` or `shm_perm.uid` in the `shmid_ds` data structure associated with `shmid`.

'0x0000' or *IPC_RMID* Remove the shared memory identifier specified by `shmid` from the system and destroy the shared memory segment and the `shmid_ds` data structure associated with it. *IPC_RMID* can be performed only by a thread with appropriate privileges or one that has an effective user ID equal to the value of `shm_perm.cuid` or `shm_perm.uid` in the `shmid_ds` data structure associated with `shmid`. The structure pointed to by `buf` is ignored and a NULL pointer is valid.

'0x0006' or *SHM_SIZE* Set the size of the shared memory segment using the `shm_segsz` member of the `shmid_ds` data structure pointed to by `buf`. This value may be larger or smaller than the current size. This function is valid for nonteraspaceshared memory segments and for teraspaceshared memory segments created using the *SHM_RESIZE_NP* option of `shmget()`. The maximum size to which a nonteraspaceshared memory segment may be expanded is 16 773 120 bytes (16 MB minus 4096 bytes). The maximum size of a resizeable teraspaceshared memory segment is 268 435 456 bytes (256 MB). *SHM_SIZE* can be performed only by a thread with appropriate privileges or a thread that has an effective user ID equal to the value of `shm_perm.cuid` or `shm_perm.uid` in the `shmid_ds` data structure associated with `shmid`.

If a shared memory segment is resized to a smaller size, other threads using the shared memory that is being removed from the shared memory segment may experience memory exceptions when accessing that memory.

Authorities

Figure 1-15. Authorization Required for `shmctl()`

Object Referred to	Authority Required	errno
Shared memory segment for which state information is retrieved (cmd = <i>IPC_STAT</i>)	Read	EACCES
Shared memory segment for which state information is set (cmd = <i>IPC_SET</i>)	See Note	EPERM
Shared memory segment to be removed (cmd = <i>IPC_RMID</i>)	See Note	EPERM
Shared memory segment to be resized (cmd = <i>SHM_SIZE</i>)	See Note	EPERM

Note: To set shared memory segment information, to remove a shared memory segment, or to resize a shared memory segment, the thread must be the owner or creator of the shared memory segment or have appropriate privileges.

Return Value

0 **shmctl()** was successful.

-1 **shmctl()** was not successful. The `errno` variable is set to indicate the error.

Error Conditions

If **shmctl()** is not successful, `errno` usually indicates one of the following errors. Under some conditions, `errno` could indicate an error other than those listed here.

[EACCES]

Permission denied.

An attempt was made to access an object in a way forbidden by its object access permissions.

The thread does not have access to the specified file, directory, component, or path.

The parameter `cmd` is equal to `IPC_STAT` and the calling thread does not have read permission.

[EDAMAGE]

A damaged object was encountered.

The value of `shmid` corresponds to a shared memory ID that has been marked as damaged by a previous shared memory operation.

[EFAULT]

The address used for an argument is not correct.

In attempting to use an argument in a call, the system detected an address that is not valid.

While attempting to access a parameter passed to this function, the system detected an address that is not valid.

[EINVAL]

The value specified for the argument is not correct.

A function was passed incorrect argument values, or an operation was attempted on an object and the operation specified is not supported for that type of object.

An argument value is not valid, out of range, or `NULL`.

One of the following has occurred:

- The value of `shmid` is not a valid shared memory identifier.
- The value of `cmd` is not a valid command.

- The value of cmd is equal to SHM_SIZE, and the shared memory segment cannot be resized because it was not created by specifying SHM_RESIZE_NP on the shmflg parameter of **shmget()**.
- The value of cmd is equal to SHM_SIZE, and the new size is not valid for the shared memory segment.

[ENOMEM]

Storage allocation request failed.

A function needed to allocate storage, but no storage is available.

A shared memory identifier segment is to be resized, but the amount of available physical memory is not sufficient to fulfill the request.

[EPERM]

Operation not permitted.

You must have appropriate privileges or be the owner of the object or other resource to do the requested operation.

The parameter cmd is equal to IPC_RMID or IPC_SET and both of the following are true:

- the calling thread does not have the appropriate privileges.
- the effective user ID of the calling thread is not equal to the value of shm_perm.cuid or shm_perm.uid in the data structure associated with shmid.

[EUNKNOWN]

Unknown system state.

The operation failed because of an unknown system state. See any messages in the job log and correct any errors that are indicated, then retry the operation.

Error Messages

None.

Usage Notes

"Appropriate privileges" is defined to be *ALLOBJ special authority. If the user profile under which the thread is running does not have *ALLOBJ special authority, the thread does not have appropriate privileges.

Related Information

- The `<sys/shm.h>` file (see [Header Files for UNIX-Type Functions](#))
- [shmat\(\)-Attach Shared Memory Segment to Current Process](#)
- [shmdt\(\)-Detach Shared Memory Segment from Calling Process](#)
- [shmget\(\)-Get ID of Shared Memory Segment with Key](#)

Example

For an example of using this function, see Using Semaphores and Shared Memory in [Examples](#).

API introduced: V3R6

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

shmdt()-Detach Shared Memory Segment from Calling Process

Syntax

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

int shmdt(const void *shmaddr);
```

Service Program Name: QP0ZUSHR

Default Public Authority: *USE

Threadsafe: Yes

The **shmdt()** function detaches the shared memory segment specified by **shmaddr** from the calling process.

Parameters

shmaddr

(Input) Address of the shared memory segment to be detached.

Authorities

Figure 1-16. Authorization Required for **shmdt()**

Object Referred to	Authority Required	errno
Shared memory segment to be detached	None	None

Return Value

- 0* **shmdt()** was successful.
- 1* **shmdt()** was not successful. The **errno** variable is set to indicate the error.

Error Conditions

If **shmdt()** is not successful, **errno** usually indicates one of the following errors. Under some conditions, **errno** could indicate an error other than those listed here.

[EDAMAGE]

A damaged object was encountered.

The value of `shmid` corresponds to a shared memory ID that has been marked as damaged by a previous shared memory operation.

[EFAULT]

The address used for an argument is not correct.

In attempting to use an argument in a call, the system detected an address that is not valid.

While attempting to access a parameter passed to this function, the system detected an address that is not valid.

[EINVAL]

The value specified for the argument is not correct.

A function was passed incorrect argument values, or an operation was attempted on an object and the operation specified is not supported for that type of object.

An argument value is not valid, out of range, or NULL.

The value of `shmaddr` is not the data segment start address of a shared memory segment.

[ENOSYS]

Function not implemented.

An attempt was made to use a function that is not available in this implementation for any object or any arguments.

The function is not implemented.

[EUNKNOWN]

Unknown system state.

The operation failed because of an unknown system state. See any messages in the job log and correct any errors that are indicated, then retry the operation.

Error Messages

None.

Usage Notes

This function does not delete the shared memory segment. To delete a shared memory segment, a `shmctl()` call specifying a `cmd` parameter of `IPC_RMID` must be used.

Related Information

- The `<sys/shm.h>` file (see [Header Files for UNIX-Type Functions](#))
- [shmat\(\)-Attach Shared Memory Segment to Current Process](#)
- [shmctl\(\)-Perform Shared Memory Control Operations](#)
- [shmget\(\)-Get ID of Shared Memory Segment with Key](#)

Example

For an example of using this function, see Using Semaphores and Shared Memory in [Examples](#).

API introduced: V3R6

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

shmget()-Get ID of Shared Memory Segment with Key

Syntax

```
#include <sys/shm.h>
#include <sys/stat.h>

int shmget(key_t key, size_t size, int shmflg);
```

Service Program Name: QP0ZUSHR

Default Public Authority: *USE

Threadsafe: Yes

The **shmget()** function returns the shared memory ID associated with the specified shared memory key.

Parameters

key

(Input) The key associated with the shared memory ID. Specifying a key of `IPC_PRIVATE` guarantees that a unique shared memory ID and shared memory segment are created. A key may also be generated by the caller or by calling the **ftok()** function.

size

(Input) The size of the shared memory segment being created. The size of the segment may be changed using the **shmctl()** API if it is nonteraspace shared memory segment or if it was created by specifying `SHM_RESIZE_NP` on the `shmflg` parameter of **shmget()**. If an existing shared memory ID is being accessed, the size may be zero.

shmflg

(Input) Operation and permission flags

The value of the `shmflg` parameter is either zero or is obtained by performing an OR operation on one or more of the constants listed below. If an existing shared memory ID is being accessed, then the permissions specified must be a subset of the existing permissions of the shared memory segment. If an existing shared memory ID is being accessed, then the `SHM_TS_NP` and `SHM_MAP_FIXED_NP` flags must match the existing attributes of the shared memory segment.

'0x0100' or S_IRUSR

Permits the creator of the shared memory ID to attach to it in read mode.

'0x0080' or S_IWUSR

Permits the creator of the shared memory ID to attach to it in write mode.

<i>'0x0020' or S_IRGRP</i>	Permits the group associated with the shared memory ID to attach to it in read mode
<i>'0x0010' or S_IWGRP</i>	Permits the group associated with the shared memory ID to attach to it in write mode
<i>'0x0004' or S_IROTH</i>	Permits others to attach to the shared memory ID in read mode.
<i>'0x0002' or S_IWOTH</i>	Permits others to attach to the shared memory ID in write mode.
<i>'0x0200' or IPC_CREAT</i>	Creates the shared memory segment ID if it does not exist already.
<i>'0x0400' or IPC_EXCL</i>	Causes shmget() to fail if IPC_CREAT is also set and the shared memory ID already exists.
<i>'0x10000' or SHM_TS_NP</i>	If shmget() creates a new shared memory segment, then the new shared memory segment will be created as a teraspace shared memory segment. When a process attaches to this shared memory segment, the shared memory segment will be added to the process's teraspace. Some compilers permit the user to indicate that the teraspace versions of storage functions should be used. For example, if a C module is compiled using CRTCMOD TERASPACE(*YES *TSIFC), this flag will be set automatically.
<i>'0x40000' or SHM_RESIZE_NP</i>	If shmget() creates a new teraspace shared memory segment, then the size of the shared memory segment may be changed using the shmctl() API. The maximum size of this teraspace shared memory segment is 268 435 456 bytes (256 MB). This flag is ignored for nonteraspace shared memory segments. A nonteraspace shared memory segment may always be resized up to 16 773 120 bytes (16 MB - 4096 bytes).
<i>'0x100000' or SHM_MAP_FIXED_NP</i>	If shmget() creates a new teraspace shared memory segment, then all processes that successfully attach to the shared memory segment will attach to the shared memory segment at the same address. The shared memory segment may not be attached in read-only mode. This flag is ignored for nonteraspace shared memory segments.

Authorities

Figure 1-17. Authorization Required for shmget()

Object Referred to	Authority Required	errno
Shared memory segment to be created	None	None
Existing shared memory segment to be accessed	See Note	EACCES

Note: If the thread is accessing a shared memory segment that already exists, the mode specified in the last 9 bits of shmflg must be a subset of the mode of the existing shared memory segment.

Return Value

value **shmget()** was successful. The value returned is the shared memory ID associated with the key parameter.

-1 **shmget()** was not successful. The `errno` variable is set to indicate the error.

Error Conditions

If **shmget()** is not successful, `errno` usually indicates one of the following errors. Under some conditions, `errno` could indicate an error other than those listed here.

[EACCES]

Permission denied.

An attempt was made to access an object in a way forbidden by its object access permissions.

The thread does not have access to the specified file, directory, component, or path.

A shared memory identifier exists for the parameter `key`, but operation permission as specified by the low-order 9 bits of `shmflg` would not be granted.

Shared memory operations are not permitted because the `QSHRMEMCTL` system value is set to 0.

[EDAMAGE]

A damaged object was encountered.

The value of `key` corresponds to shared memory that has been marked as damaged by a previous shared memory operation.

[EEXIST]

File exists.

The file specified already exists and the specified operation requires that it not exist.

The named file, directory, or path already exists.

A shared memory identifier exists for the parameter `key`, but $((\text{shmflg} \& \text{IPC_CREAT}) \&\& (\text{shmflg} \& \text{IPC_EXCL}))$ is not zero. ($\&$ is a bitwise AND; $\&\&$ is a logical AND.)

[EINVAL]

The value specified for the argument is not correct.

A function was passed incorrect argument values, or an operation was attempted on an object and the operation specified is not supported for that type of object.

An argument value is not valid, out of range, or NULL.

One of the following has occurred:

- The value of the parameter size is less than the system-imposed minimum or greater than the system-imposed maximum.

- A shared memory identifier exists for the parameter key, but the size of the segment associated with it is less than size and size is not zero.
- A shared memory identifier exists for the parameter key, but the SHM_MAP_FIXED_NP or SHM_TS_NP attributes of the shared memory segment do not match the attributes specified by the parameter shmflg.

[ENOENT]

No such path or directory.

The directory or a component of the path name specified does not exist.

A named file or directory does not exist or is an empty string.

A shared memory identifier does not exist for the parameter key, and (shmflg & IPC_CREAT) is zero.

[ENOMEM]

Storage allocation request failed.

A function needed to allocate storage, but no storage is available.

A shared memory identifier and associated shared memory segment are to be created, but the amount of available physical memory is not sufficient to fulfill the request.

[ENOSPC]

No space available.

The requested operations required additional space on the device and there is no space left. This could also be caused by exceeding the user profile storage limit when creating or transferring ownership of an object.

Insufficient space remains to hold the intended file, directory, or link.

A shared memory identifier is to be created, but the system-imposed limit on the maximum number of allowed shared memory identifiers system-wide would be exceeded.

[EUNKNOWN]

Unknown system state.

The operation failed because of an unknown system state. See any messages in the job log and correct any errors that are indicated, then retry the operation.

Error Messages

None.

Usage Notes

1. **shmget()** creates a shared memory ID, its associated `shmid_ds` data structure, and a shared memory segment of at least `size` bytes if one of the following is true:
 - The shared memory key is `IPC_PRIVATE`
 - There is no shared memory ID already associated with the shared memory key and the `IPC_CREAT` flag is set.
2. When the shared memory ID is created, the `shmid_ds` structure (defined in the `<sys/shm.h>` header file) that is associated with the shared memory ID is initialized as follows:
 - The `shm_perm.cuid` and `shm_perm.uid` values are set equal to the effective user ID (uid) of the calling thread.
 - The `shm_perm.cgid` and `shm_perm.gid` values are set equal to the effective group ID (gid) of the calling thread.
 - The low-order 9 bits of `shm_perm.mode` are set equal to the low-order 9 bits of `shmflg`.
 - `shm_segsz` is set to the value specified in `size`.
 - `shm_lpid`, `shm_nattch`, `shm_atime`, and `shm_dtime` are set to zero.
 - `shm_ctime` is set to the current time.
3. **shmat()** should be used to set or gain pointer addressability to the shared memory segment associated with the shared memory ID after the shared memory ID is obtained.
4. The maximum size of a teraspace shared memory segment is 4 294 967 295 bytes (4GB - 1). The maximum size of a resizable teraspace shared memory segment is 268 435 456 bytes (256 MB). The maximum shared memory segment size for nonteraspace shared memory segments is 16 776 960 bytes (16 MB - 256 bytes).
5. The storage for a shared memory segment is not allocated until it is attached to a process. A process will not be able to attach to a shared memory segment that is larger than the amount of storage available on the system.
6. Processes cannot attach a nonteraspace shared memory segment in read-only or write-only mode. Consequently, permissions that specify read-only or write-only will always result in **shmat()** failure. Processes are permitted to attach a teraspace shared memory segment in read-only mode.
7. Shared memory segments larger than 16 773 120 bytes (16 MB minus 4096 bytes) should be created as teraspace shared memory segments. When the operating system accesses a nonteraspace shared memory segment that has a size in the range 16 773 120 bytes (16 MB minus 4096 bytes) to 16 776 960 bytes (16 MB minus 256 bytes), a performance degradation may be observed.

Related Information

- The `<sys/shm.h>` file (see [Header Files for UNIX-Type Functions](#))
- [ftok\(\)](#)--Generate IPC Key from File Name
- [shmat\(\)](#)--Attach Shared Memory Segment to Current Process
- [shmctl\(\)](#)--Perform Shared Memory Control Operations
- [shmdt\(\)](#)--Detach Shared Memory Segment from Calling Process

Example

For an example of using this function, see Using Semaphores and Shared Memory in [Examples](#).

API introduced: V3R6

[Top](#) | [UNIX-Type APIs](#) | [APIs by category](#)

Header Files for UNIX-Type Functions

Programs using the UNIX-type functions must include one or more header files that contain information needed by the functions, such as:

- Macro definitions
- Data type definitions
- Structure definitions
- Function prototypes

The header files are provided in the QSYSINC library, which is optionally installable. Make sure QSYSINC is on your system before compiling programs that use these header files. For information on installing the QSYSINC library, see [Data structures and the QSYSINC Library](#).

The table below shows the file and member name in the QSYSINC library for each header file used by the UNIX-type APIs in this publication.

Name of Header File	Name of File in QSYSINC	Name of Member
arpa/inet.h	ARPA	INET
arpa/nameser.h	ARPA	NAMESER
bse.h	H	BSE
bsedos.h	H	BSEDOS
bseerr.h	H	BSEERR
dirent.h	H	DIRENT
errno.h	H	ERRNO
fcntl.h	H	FCNTL
grp.h	H	GRP
»inttypes.h	H	INTTYPES«
limits.h	H	LIMITS
»mman.h	H	MMAN«
netdb.h	H	NETDB
»netinet/icmp6.h	NETINET	ICMP6«
net/if.h	NET	IF
netinet/in.h	NETINET	IN
netinet/ip_icmp.h	NETINET	IP_ICMP
netinet/ip.h	NETINET	IP
»netinet/ip6.h	NETINET	IP6«
netinet/tcp.h	NETINET	TCP
netinet/udp.h	NETINET	UDP
netns/idp.h	NETNS	IDP
netns/ipx.h	NETNS	IPX
netns/ns.h	NETNS	NS
netns/sp.h	NETNS	SP
net/route.h	NET	ROUTE
nettel/tel.h	NETTEL	TEL

os2.h	H	OS2
os2def.h	H	OS2DEF
pwd.h	H	PWD
Qlg.h	H	QLG
qp0lflop.h	H	QP0LFLOP
»qp0ljrnl.h	H	QP0LJRNL«
»qp0lrord.h	H	QP0LRORD«
Qp0lstdi.h	H	QP0LSTDI
qp0wpid.h	H	QP0WPID
qp0zdipc.h	H	QP0ZDIPC
qp0zipc.h	H	QP0ZIPC
qp0zolip.h	H	QP0ZOLIP
qp0zolsm.h	H	QP0ZOLSM
qp0zripc.h	H	QP0ZRIPC
qp0ztrc.h	H	QP0ZTRC
qp0ztrml.h	H	QP0ZTRML
qp0z1170.h	H	QP0Z1170
»qsoasync.h	H	QSOASYNC«
qtnxaapi.h	H	QTNXAAPI
qtnxadtp.h	H	QTNXADTP
qtomeapi.h	H	QTOMEAPI
qtossapi.h	H	QTOSSAPI
resolv.h	H	RESOLVE
semaphore.h	H	SEMAPHORE
signal.h	H	SIGNAL
spawn.h	H	SPAWN
ssl.h	H	SSL
sys/errno.h	H	ERRNO
sys/ioctl.h	SYS	IOCTL
sys/ipc.h	SYS	IPC
sys/layout.h	H	LAYOUT
sys/limits.h	H	LIMITS
sys/msg.h	SYS	MSG
sys/param.h	SYS	PARAM
»sys/resource.h	SYS	RESOURCE«
sys/sem.h	SYS	SEM
sys/setjmp.h	SYS	SETJMP
sys/shm.h	SYS	SHM
sys/signal.h	SYS	SIGNAL
sys/socket.h	SYS	SOCKET
sys/stat.h	SYS	STAT
sys/statvfs.h	SYS	STATVFS

sys/time.h	SYS	TIME
sys/types.h	SYS	TYPES
sys/uio.h	SYS	UIO
sys/un.h	SYS	UN
sys/wait.h	SYS	WAIT
» ulimit.h	H	ULIMIT «
unistd.h	H	UNISTD
utime.h	H	UTIME

You can display a header file in QSYSINC by using one of the following methods:

- Using your editor. For example, to display the **unistd.h** header file using the Source Entry Utility editor, enter the following command:

```
STRSEU SRCFILE(QSYSINC/H) SRCMBR(UNISTD) OPTION(5)
```

- Using the Display Physical File Member command. For example, to display the **sys/stat.h** header file, enter the following command:

```
DSPPFM FILE(QSYSINC/SYS) MBR(STAT)
```

You can print a header file in QSYSINC by using one of the following methods:

- Using your editor. For example, to print the **unistd.h** header file using the Source Entry Utility editor, enter the following command:

```
STRSEU SRCFILE(QSYSINC/H) SRCMBR(UNISTD) OPTION(6)
```

- Using the Copy File command. For example, to print the **sys/stat.h** header file, enter the following command:

```
CPYF FROMFILE(QSYSINC/SYS) TOFILE(*PRINT) FROMMBR(STAT)
```

Symbolic links to these header files are also provided in directory /QIBM/include.

Errno Values for UNIX-Type Functions

Programs using the UNIX-type functions may receive error information as *errno* values. The possible values returned are listed here in ascending *errno* value sequence.

Name	Value	Text
EDOM	3001	A domain error occurred in a math function.
ERANGE	3002	A range error occurred.
ETRUNC	3003	Data was truncated on an input, output, or update operation.
ENOTOPEN	3004	File is not open.
ENOTREAD	3005	File is not opened for read operations.
EIO	3006	Input/output error.
ENODEV	3007	No such device.
ERECIO	3008	Cannot get single character for files opened for record I/O.
ENOTWRITE	3009	File is not opened for write operations.
ESTDIN	3010	The stdin stream cannot be opened.
ESTDOUT	3011	The stdout stream cannot be opened.
ESTDERR	3012	The stderr stream cannot be opened.
EBADSEEK	3013	The positioning parameter in fseek is not correct.
EBADNAME	3014	The object name specified is not correct.
EBADMODE	3015	The type variable specified on the open function is not correct.
EBADPOS	3017	The position specifier is not correct.
ENOPOS	3018	There is no record at the specified position.
ENUMMBRS	3019	Attempted to use ftell on multiple members.
ENUMRECS	3020	The current record position is too long for ftell.
EINVAL	3021	The value specified for the argument is not correct.
EBADFUNC	3022	Function parameter in the signal function is not set.
ENOENT	3025	No such path or directory.
ENOREC	3026	Record is not found.
EPERM	3027	The operation is not permitted.
EBADDATA	3028	Message data is not valid.
EBUSY	3029	Resource busy.
EBADOPT	3040	Option specified is not valid.
ENOTUPD	3041	File is not opened for update operations.
ENOTDLT	3042	File is not opened for delete operations.

EPAD	3043	The number of characters written is shorter than the expected record length.
EBADKEYLN	3044	A length that was not valid was specified for the key.
EPUTANDGET	3080	A read operation should not immediately follow a write operation.
EGETANDPUT	3081	A write operation should not immediately follow a read operation.
EIOERROR	3101	A nonrecoverable I/O error occurred.
EIORECERR	3102	A recoverable I/O error occurred.
EACCES	3401	Permission denied.
ENOTDIR	3403	Not a directory.
ENOSPC	3404	No space is available.
EXDEV	3405	Improper link.
EAGAIN	3406	Operation would have caused the process to be suspended.
EWOULDBLOCK	3406	Operation would have caused the process to be suspended.
EINTR	3407	Interrupted function call.
EFAULT	3408	The address used for an argument was not correct.
ETIME	3409	Operation timed out.
ENXIO	3415	No such device or address.
EAPAR	3418	Possible APAR condition or hardware failure.
ERECURSE	3419	Recursive attempt rejected.
EADDRINUSE	3420	Address already in use.
EADDRNOTAVAIL	3421	Address is not available.
EAFNOSUPPORT	3422	The type of socket is not supported in this protocol family.
EALREADY	3423	Operation is already in progress.
ECONNABORTED	3424	Connection ended abnormally.
ECONNREFUSED	3425	A remote host refused an attempted connect operation.
ECONNRESET	3426	A connection with a remote socket was reset by that socket.
EDESTADDRREQ	3427	Operation requires destination address.
EHOSTDOWN	3428	A remote host is not available.
EHOSTUNREACH	3429	A route to the remote host is not available.
EINPROGRESS	3430	Operation in progress.
EISCONN	3431	A connection has already been established.
EMSGSIZE	3432	Message size is out of range.
ENETDOWN	3433	The network currently is not available.
ENETRESET	3434	A socket is connected to a host that is no longer available.

ENETUNREACH	3435	Cannot reach the destination network.
ENOBUFS	3436	There is not enough buffer space for the requested operation.
ENOPROTOPT	3437	The protocol does not support the specified option.
ENOTCONN	3438	Requested operation requires a connection.
ENOTSOCK	3439	The specified descriptor does not reference a socket.
ENOTSUP	3440	Operation is not supported.
EOPNOTSUPP	3440	Operation is not supported.
EPFNOSUPPORT	3441	The socket protocol family is not supported.
EPROTONOSUPPORT	3442	No protocol of the specified type and domain exists.
EPROTOTYPE	3443	The socket type or protocols are not compatible.
ERCVDERR	3444	An error indication was sent by the peer program.
ESHUTDOWN	3445	Cannot send data after a shutdown.
ESOCKTNOSUPPORT	3446	The specified socket type is not supported.
ETIMEDOUT	3447	A remote host did not respond within the timeout period.
EUNATCH	3448	The protocol required to support the specified address family is not available at this time.
EBADF	3450	Descriptor is not valid.
EMFILE	3452	Too many open files for this process.
ENFILE	3453	Too many open files in the system.
EPIPE	3455	Broken pipe.
ECANCEL	3456	Operation cancelled.
EEXIST	3457	File exists.
EDEADLK	3459	Resource deadlock avoided.
ENOMEM	3460	Storage allocation request failed.
EOWNERTERM	3462	The synchronization object no longer exists because the owner is no longer running.
EDESTROYED	3463	The synchronization object was destroyed, or the object no longer exists.
ETERM	3464	Operation was terminated.
ENOENT1	3465	No such file or directory.
ENOEQFLOG	3466	Object is already linked to a dead directory.
EEMPTYDIR	3467	Directory is empty.
EMLINK	3468	Maximum link count for a file was exceeded.

ESPIPE	3469	Seek request is not supported for object.
ENOSYS	3470	Function not implemented.
EISDIR	3471	Specified target is a directory.
EROFS	3472	Read-only file system.
EUNKNOWN	3474	Unknown system state.
EITERBAD	3475	Iterator is not valid.
EITERSTE	3476	Iterator is in wrong state for operation.
EHRICLSBAD	3477	HRI class is not valid.
EHRICLBAD	3478	HRI subclass is not valid.
EHRITYPBAD	3479	HRI type is not valid.
ENOTAPPL	3480	Data requested is not applicable.
EHRIREQTYP	3481	HRI request type is not valid.
EHRINAMEBAD	3482	HRI resource name is not valid.
EDAMAGE	3484	A damaged object was encountered.
ELOOP	3485	A loop exists in the symbolic links.
ENAMETOOLONG	3486	A path name is too long.
ENOLCK	3487	No locks are available.
ENOTEMPTY	3488	Directory is not empty.
ENOSYSRSC	3489	System resources are not available.
ECONVERT	3490	Conversion error.
E2BIG	3491	Argument list is too long.
EILSEQ	3492	Conversion stopped due to input character that does not belong to the input codeset.
ETYPE	3493	Object type mismatch.
EBADDIR	3494	Attempted to reference a directory that was not found or was destroyed.
EBADOBJ	3495	Attempted to reference an object that was not found, was destroyed, or was damaged.
EIDXINVAL	3496	Data space index used as a directory is not valid.
ESOFTDAMAGE	3497	Object has soft damage.
ENOTENROLL	3498	User is not enrolled in system distribution directory.
EOffline	3499	Object is suspended.
EROOBJ	3500	Object is a read-only object.
EEAHDDSI	3501	Hard damage on extended attribute data space index.
EEASDDSI	3502	Soft damage on extended attribute data space index.
EEAHDDS	3503	Hard damage on extended attribute data space.
EEASDDS	3504	Soft damage on extended attribute data space.
EEADUPRC	3505	Duplicate extended attribute record.

ELOCKED	3506	Area being read from or written to is locked.
EFBIG	3507	Object too large.
EIDRM	3509	The semaphore, shared memory, or message queue identifier is removed from the system.
ENOMSG	3510	The queue does not contain a message of the desired type and (msgflg logically ANDed with IPC_NOWAIT).
EFILECVT	3511	File ID conversion of a directory failed.
EBADFID	3512	A file ID could not be assigned when linking an object to a directory.
ESTALE	3513	File handle was rejected by server.
ESRCH	3515	No such process.
ENOTSIGINIT	3516	Process is not enabled for signals.
ECHILD	3517	No child process.
EBADH	3520	Handle is not valid.
ETOOMANYREFS	3523	The operation would have exceeded the maximum number of references allowed for a descriptor.
ENOTSAFE	3524	Function is not allowed.
E_OVERFLOW	3525	Object is too large to process.
EJRNDDAMAGE	3526	Journal is damaged.
EJRNINACTIVE	3527	Journal is inactive.
EJRNRCVSPC	3528	Journal space or system storage error.
EJRNRMNT	3529	Journal is remote.
ENEWJRNRCV	3530	New journal receiver is needed.
ENEWJRN	3531	New journal is needed.
EJOURNALED	3532	Object already journaled.
EJRNENTTOOLONG	3533	Entry is too large to send.
EDATALINK	3534	Object is a datalink object.
ENOTAVAIL	3535	IASP is not available.
ENOTTY	3536	I/O control operation is not appropriate.
EFBIG2	3540	Attempt to write or truncate file past its sort file size limit.
ETXTBSY	3543	Text file busy.
EASPGRPNOTSET	3544	ASP group not set for thread.
ERESTART	3545	A system call was interrupted and may be restarted.