Semaphore, Shared Memory and IPC

Introduction

This module provides wrappers for the System V IPC family of functions. It includes semaphores, shared memory and inter-process messaging (IPC).

Semaphores may be used to provide exclusive access to resources on the current machine, or to limit the number of processes that may simultaneously use a resource.

This module provides also shared memory functions using System V shared memory. Shared memory may be used to provide access to global variables. Different httpd-daemons and even other programs (such as Perl, C, ...) are able to access this data to provide a global data-exchange. Remember, that shared memory is NOT safe against simultaneous access. Use semaphores for synchronization.

Limits of Shared Memory by the Unix OS

SHMMAX	max size of shared memory, normally 131072 bytes
SHMMIN	minimum size of shared memory, normally 1 byte
SHMMNI	max amount of shared memory segments on a system, normally 100
SHMSEG	max amount of shared memory segments per process, normally 6

The messaging functions may be used to send and receive messages to/from other processes. They provide a simple and effective means of exchanging data between processes, without the need for setting up an alternative using Unix domain sockets.

Note
This extension is not available on Windows platforms.

Installing/Configuring

Requirements

No external libraries are needed to build this extension.

Installation

Support for this functions are not enabled by default. To enable System V semaphore support compile PHP with the option *--enable-sysvsem*. To enable the System V shared memory support compile PHP with the option *--enable-sysvshm*. To enable the System V messages support compile PHP with the option *--enable-sysvmsg*.

Runtime Configuration

The behaviour of these functions is affected by settings in *php.ini*.

Semaphore Configuration Options

Name	Default	Changeable	Changelog
sysvmsg.value	"42"	PHP_INI_ALL	
sysvmsg.string	"foobar"	PHP_INI_ALL	

For further details and definitions of the PHP_INI_* constants, see the php.ini directives.

Resource Types

This extension has no resource types defined.

Predefined Constants

The constants below are defined by this extension, and will only be available when the extension has either been compiled into PHP or dynamically loaded at runtime.

System V message constants

Constant	Type	Changelog
MSG_IPC_NOWAIT	integer	
MSG_EAGAIN	integer	As of 5.2.0
MSG_ENOMSG	integer	As of 5.2.0
MSG_NOERROR	integer	
MSG_EXCEPT	integer	

Semaphore Functions

ftok

ftok -- Convert a pathname and a project identifier to a System V IPC key

Description

int **ftok** (string `$pathname`, string `$proj`)

The function converts the *`pathname`* of an existing accessible file and a project identifier into an *integer* for use with for example shmop_open() and other System V IPC keys.

Parameters

`pathname`
: Path to an accessible file.

`proj`
: Project identifier. This must be a one character string.

Return Values

On success the return value will be the created key value, otherwise *-1* is returned.

See Also

- shmop_open()
- sem_get()

msg_get_queue

msg_get_queue -- Create or attach to a message queue

Description

resource **msg_get_queue** (int `$key` [, int `$perms`])

msg_get_queue() returns an id that can be used to access the System V message queue with the given *`key`*. The first call creates the message queue with the optional *`perms`*. A second call to msg_get_queue() for the same *`key`* will return a different message queue identifier, but both identifiers access the same underlying message queue.

Parameters

`key`
: Message queue numeric ID

`perms`
: Queue permissions. Default to 0666. If the message queue already exists, the *`perms`* will be ignored.

Return Values

Returns a resource handle that can be used to access the System V message queue.

See Also

- msg_remove_queue()
- msg_receive()
- msg_send()
- msg_stat_queue()
- msg_set_queue()

msg_queue_exists

msg_queue_exists -- Check wether a message queue exists

Description

bool **msg_queue_exists** (int `$key`)

Checks wether the message queue *key* exists.

Parameters

key
: Queue key.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

- msg_remove_queue()
- msg_receive()
- msg_stat_queue()

msg_receive

msg_receive -- Receive a message from a message queue

Description

bool **msg_receive** (resource `$queue`, int `$desiredmsgtype`, int `&$msgtype`, int `$maxsize`, mixed `&$message` [, bool `$unserialize` [, int `$flags` [, int `&$errorcode`]]])

msg_receive() will receive the first message from the specified *`queue`* of the type specified by *`desiredmsgtype`*.

Parameters

`queue`

`desiredmsgtype`
: If *`desiredmsgtype`* is 0, the message from the front of the queue is returned. If *`desiredmsgtype`* is greater than 0, then the first message of that type is returned. If *`desiredmsgtype`* is less than 0, the first message on the queue with the lowest type less than or equal to the absolute value of *`desiredmsgtype`* will be read. If no messages match the criteria, your script will wait until a suitable message arrives on the queue. You can prevent the script from blocking by specifying **MSG_IPC_NOWAIT** in the *`flags`* parameter.

`msgtype`
: The type of the message that was received will be stored in this parameter.

`maxsize`
: The maximum size of message to be accepted is specified by the *`maxsize`*; if the message in the queue is larger than this size the function will fail (unless you set *`flags`* as described below).

`message`
: The received message will be stored in *`message`*, unless there were errors receiving the message.

`unserialize`
: *`unserialize`* defaults to **TRUE**; if it is set to **TRUE**, the message is treated as though it was serialized using the same mechanism as the session module. The message will be unserialized and then returned to your script. This allows you to easily receive arrays or complex object structures from other PHP scripts, or if you are using the WDDX serializer, from any WDDX compatible source. If *`unserialize`* is **FALSE**, the message will be returned as a binary-safe string.

`flags`
: The optional *`flags`* allows you to pass flags to the low-level msgrcv system call. It

defaults to 0, but you may specify one or more of the following values (by adding or ORing them together).

Flag values for msg_receive

MSG_IPC_NOWAIT	If there are no messages of the *desiredmsgtype*, return immediately and do not wait. The function will fail and return an integer value corresponding to **MSG_ENOMSG**.
MSG_EXCEPT	Using this flag in combination with a *desiredmsgtype* greater than 0 will cause the function to receive the first message that is not equal to *desiredmsgtype*.
MSG_NOERROR	If the message is longer than *maxsize*, setting this flag will truncate the message to *maxsize* and will not signal an error.

errorcode
If the function fails, the optional *errorcode* will be set to the value of the system errno variable.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Upon successful completion the message queue data structure is updated as follows: *msg_lrpid* is set to the process-ID of the calling process, *msg_qnum* is decremented by 1 and *msg_rtime* is set to the current time.

See Also

- msg_remove_queue()
- msg_send()
- msg_stat_queue()
- msg_set_queue()

msg_remove_queue

msg_remove_queue -- Destroy a message queue

Description

bool **msg_remove_queue** (resource $queue)

msg_remove_queue() destroys the message queue specified by the *queue*. Only use this function when all processes have finished working with the message queue and you need to release the system resources held by it.

Parameters

queue
: Message queue resource handle

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

- msg_get_queue()
- msg_receive()
- msg_stat_queue()
- msg_set_queue()

msg_send

msg_send -- Send a message to a message queue

Description

bool **msg_send** (resource `$queue`, int `$msgtype`, mixed `$message` [, bool `$serialize` [, bool `$blocking` [, int `&$errorcode`]]])

msg_send() sends a *`message`* of type *`msgtype`* (which MUST be greater than 0) to the message queue specified by *`queue`*.

Parameters

`queue`

`msgtype`

`message`

`serialize`
The optional *`serialize`* controls how the *`message`* is sent. *`serialize`* defaults to **TRUE** which means that the *`message`* is serialized using the same mechanism as the session module before being sent to the queue. This allows complex arrays and objects to be sent to other PHP scripts, or if you are using the WDDX serializer, to any WDDX compatible client.

`blocking`
If the message is too large to fit in the queue, your script will wait until another process reads messages from the queue and frees enough space for your message to be sent. This is called blocking; you can prevent blocking by setting the optional *`blocking`* parameter to **FALSE**, in which case msg_send() will immediately return **FALSE** if the message is too big for the queue, and set the optional *`errorcode`* to **MSG_EAGAIN**, indicating that you should try to send your message again a little later on.

`errorcode`

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Upon successful completion the message queue data structure is updated as follows: *`msg_lspid`* is set to the process-ID of the calling process, *`msg_qnum`* is incremented by 1 and

msg_stime is set to the current time.

See Also

- msg_remove_queue()
- msg_receive()
- msg_stat_queue()
- msg_set_queue()

msg_set_queue

msg_set_queue -- Set information in the message queue data structure

Description

bool **msg_set_queue** (resource `$queue`, array `$data`)

msg_set_queue() allows you to change the values of the msg_perm.uid, msg_perm.gid, msg_perm.mode and msg_qbytes fields of the underlying message queue data structure.

Changing the data structure will require that PHP be running as the same user that created the queue, owns the queue (as determined by the existing msg_perm.xxx fields), or be running with root privileges. root privileges are required to raise the msg_qbytes values above the system defined limit.

Parameters

queue
: Message queue resource handle

data
: You specify the values you require by setting the value of the keys that you require in the *data* array.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

- msg_remove_queue()
- msg_receive()
- msg_stat_queue()
- msg_get_queue()

msg_stat_queue

msg_stat_queue -- Returns information from the message queue data structure

Description

array **msg_stat_queue** (resource $queue)

msg_stat_queue() returns the message queue meta data for the message queue specified by the *queue*. This is useful, for example, to determine which process sent the message that was just received.

Parameters

queue
Message queue resource handle

Return Values

The return value is an array whose keys and values have the following meanings:

Array structure for msg_stat_queue

msg_perm.uid	The uid of the owner of the queue.
msg_perm.gid	The gid of the owner of the queue.
msg_perm.mode	The file access mode of the queue.
msg_stime	The time that the last message was sent to the queue.
msg_rtime	The time that the last message was received from the queue.
msg_ctime	The time that the queue was last changed.
msg_qnum	The number of messages waiting to be read from the queue.
msg_qbytes	The number of bytes of space currently available in the queue to hold sent messages until they are received.
msg_lspid	The pid of the process that sent the last message to the queue.

msg_lrpid	The pid of the process that received the last message from the queue.

See Also

- msg_remove_queue()
- msg_receive()
- msg_get_queue()
- msg_set_queue()

sem_acquire

sem_acquire -- Acquire a semaphore

Description

bool **sem_acquire** (resource `$sem_identifier`)

sem_acquire() blocks (if necessary) until the semaphore can be acquired. A process attempting to acquire a semaphore which it has already acquired will block forever if acquiring the semaphore would cause its maximum number of semaphore to be exceeded.

After processing a request, any semaphores acquired by the process but not explicitly released will be released automatically and a warning will be generated.

Parameters

`sem_identifier`
: *`sem_identifier`* is a semaphore resource, obtained from sem_get().

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

- sem_get()
- sem_release()

sem_get

sem_get -- Get a semaphore id

Description

resource **sem_get** (int `$key` [, int `$max_acquire` [, int `$perm` [, int `$auto_release`]]])

sem_get() returns an id that can be used to access the System V semaphore with the given *`key`*.

A second call to sem_get() for the same key will return a different semaphore identifier, but both identifiers access the same underlying semaphore.

Parameters

`key`

`max_acquire`
: The number of processes that can acquire the semaphore simultaneously is set to *`max_acquire`* (defaults to 1).

`perm`
: The semaphore permissions. Defaults to 0666. Actually this value is set only if the process finds it is the only process currently attached to the semaphore.

`auto_release`
: Specifies if the semaphore should be automatically released on request shutdown.

Return Values

Returns a positive semaphore identifier on success, or **FALSE** on error.

ChangeLog

Version	Description
4.3.0	The *`auto_release`* parameter was added.

See Also

- sem_acquire()
- sem_release()
- ftok()

sem_release

sem_release -- Release a semaphore

Description

bool **sem_release** (resource `$sem_identifier`)

sem_release() releases the semaphore if it is currently acquired by the calling process, otherwise a warning is generated.

After releasing the semaphore, sem_acquire() may be called to re-acquire it.

Parameters

`sem_identifier`
A Semaphore resource handle as returned by sem_get().

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

- sem_get()
- sem_acquire()

sem_remove

sem_remove -- Remove a semaphore

Description

bool **sem_remove** (resource `$sem_identifier`)

sem_remove() removes the given semaphore.

After removing the semaphore, it is no more accessible.

Parameters

`sem_identifier`
A semaphore resource identifier as returned by sem_get().

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

- sem_get()
- sem_release()
- sem_acquire()

shm_attach

shm_attach -- Creates or open a shared memory segment

Description

int **shm_attach** (int `$key` [, int `$memsize` [, int `$perm`]])

shm_attach() returns an id that can be used to access the System V shared memory with the given *`key`*, the first call creates the shared memory segment with *`memsize`* and the optional perm-bits *`perm`*.

A second call to shm_attach() for the same *`key`* will return a different shared memory identifier, but both identifiers access the same underlying shared memory. *`memsize`* and *`perm`* will be ignored.

Parameters

`key`
: A numeric shared memory segment ID

`memsize`
: The memory size. If not provided, default to the *sysvshm.init_mem* in the *php.ini*, otherwise 10000 bytes.

`perm`
: The optional permission bits. Default to 0666.

Return Values

Returns a shared memory segment identifier.

See Also

- shm_detach()
- ftok()

shm_detach

shm_detach -- Disconnects from shared memory segment

Description

bool **shm_detach** (int `$shm_identifier`)

shm_detach() disconnects from the shared memory given by the *`shm_identifier`* created by shm_attach(). Remember, that shared memory still exist in the Unix system and the data is still present.

Parameters

`shm_identifier`
: A shared memory resource handle as returned by shm_attach()

Return Values

shm_detach() always returns **TRUE**.

See Also

- shm_attach()
- shm_remove()
- shm_remove_var()

shm_get_var

shm_get_var -- Returns a variable from shared memory

Description

mixed **shm_get_var** (int `$shm_identifier`, int `$variable_key`)

shm_get_var() returns the variable with a given *`variable_key`*, in the given shared memory segment. The variable is still present in the shared memory.

Parameters

`shm_identifier`
: Shared memory segment, obtained from shm_attach().

`variable_key`
: The variable key.

Return Values

Returns the variable with the given key.

shm_put_var

shm_put_var -- Inserts or updates a variable in shared memory

Description

bool **shm_put_var** (int `$shm_identifier`, int `$variable_key`, mixed `$variable`)

shm_put_var() inserts or updates the *`variable`* with the given *`variable_key`*.

Warnings (*E_WARNING* level) will be issued if *`shm_identifier`* is not a valid SysV shared memory index or if there was not enough shared memory remaining to complete your request.

Parameters

`shm_identifier`
: A shared memory resource handle as returned by shm_attach()

`variable_key`
: The variable key.

`variable`
: The variable. All variable-types are supported.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

shm_remove_var

shm_remove_var -- Removes a variable from shared memory

Description

bool **shm_remove_var** (int `$shm_identifier`, int `$variable_key`)

Removes a variable with a given *`variable_key`* and frees the occupied memory.

Parameters

`shm_identifier`
: The shared memory identifier as returned by shm_attach()

`variable_key`
: The variable key.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

- shm_remove()

shm_remove

shm_remove -- Removes shared memory from Unix systems

Description

bool **shm_remove** (int `$shm_identifier`)

shm_remove() removes the shared memory *`shm_identifier`*. All data will be destroyed.

Parameters

`shm_identifier`
: The shared memory identifier as returned by shm_attach()

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

- shm_remove_var()