Kernel Reference

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Table of Contents

Document serial number: 000004892117

Memory Manager

Document serial number: 000004892117

Datatypes

SceKernelAllocMemBlockOpt

`sceKernelAllocMemBlock()` optional data

Definition

```
#include <kernel.h>
typedef struct SceKernelAllocMemBlockOpt {
        SceSize size;
        SceUInt32 attr;
        SceSize alignment;
        SceUInt32 reserved;
        const char *strBaseBlockName;
} SceKernelAllocMemBlockOpt;
```

Members

size	Specify size of this structure
attr	Attribute, specify valid member
alignment	Specify alignment `SCE_KERNEL_ALLOC_MEMBLOCK_ATTR_HAS_ALIGNMENT` must be set in *attr*. The sizes that can be specified in alignment vary according to *type* of `sceKernelAllocMemBlock()`.
reserved	Reserved
strBaseBlockName	Reserved

Description

Optional structure of `sceKernelAllocMemBlock()`.

The following definition can be set in *attr*:

Value	Description
SCE_KERNEL_ALLOC_MEMBLOCK_ATTR_HAS_ALIGNMENT	Alignment member is valid

SceKernelFreeMemorySizeInfo

Data structure that receives the result of `sceKernelGetFreeMemorySize()`

Definition

```
#include <kernel.h>
typedef struct SceKernelFreeMemorySizeInfo {
        SceSize size;
        SceSize sizeMain;
        SceSize sizeCdram;
        SceSize sizePhycont;
} SceKernelFreeMemorySizeInfo;
```

Members

size	Specify size of this structure
sizeMain	Returns the free memory size of the main memory area on LPDDR2
sizeCdram	Returns the free memory size on CDRAM
sizePhycont	Returns the free memory size of the physical continuous memory area on LPDDR2

Description

This data structure is used to return the result of `sceKernelGetFreeMemorySize()`.

SceKernelMemoryType

Type representing memory type

Definition

```
#include <kernel.h>
typedef int SceKernelMemoryType;
```

Description

This type represents the memory block type allocated with `sceKernelAllocMemBlock()`.

SceKernelMemBlockInfo

Memory block information structure

Definition

```
#include <kernel.h>
typedef struct SceKernelMemBlockInfo {
        SceSize size;
        void *mappedBase;
        SceSize mappedSize;
        SceKernelMemoryType memoryType;
        SceUInt32 access;
} SceKernelMemBlockInfo;
```

Members

size	Size of structure
mappedBase	Map base address
mappedSize	Mapped size
memoryType	Memory type
access	Access privilege

Description

This structure is used to obtain information about a memory block.

Document serial number: 000004892117

Memory Block

sceKernelAllocMemBlock

Allocate new memory blocks

Definition

```
#include <kernel.h>
SceUID sceKernelAllocMemBlock(
        const char *name,
        SceKernelMemBlockType type,
        SceSize vsize,
        struct SceKernelAllocMemBlockOpt *pOpt
);
```

Arguments

name	Memory block name NULL cannot be specified.
type	Memory block type
vsize	Size of memory block in virtual space
pOpt	Optional structure

Return Values

Value	Description
id	ID of generated memory block
<SCE_OK	Error code

Description

This allocates new memory blocks. The following memory types can be specified to *type*.

Value	Description
SCE_KERNEL_MEMBLOCK_TYPE_USER_RW	Allocates a cache-enabled memory that can be read/written by the user from the main area on LPDDR2. The size must be a multiple of 4 KB. The alignment that can be specified in *pOpt* must be between 4 KB and 16 MB and a power of 2.
SCE_KERNEL_MEMBLOCK_TYPE_USER_RW_UNCACHE	Allocates a cache-disabled memory that can be read/written by the user from the main area on LPDDR2. The size must be a multiple of 4 KB. The alignment that can be specified in *pOpt* must be between 4 KB and 16 MB and a power of 2.

Value	Description
`SCE_KERNEL_MEMBLOCK_TYPE_USER_MAIN_PHYCONT_RW`	Allocates a cache-enabled memory that can be read/written by the user from the physical continuous memory area on LPDDR2. The allocated memory is always a physical continuous memory area. The size must be a multiple of 1MB. The alignment cannot be set with *pOpt*.
`SCE_KERNEL_MEMBLOCK_TYPE_USER_MAIN_PHYCONT_NC_RW`	Allocates a cache-disabled memory that can be read/written by the user from the physical continuous memory area on LPDDR2. The allocated memory is always a physical continuous memory area. The size must be a multiple of 1MB. The alignment cannot be set with *pOpt*.
`SCE_KERNEL_MEMBLOCK_TYPE_USER_CDRAM_RW`	Allocates a cache-disabled memory that can be read/written by the user from the CDRAM The size must be a multiple of 256 KB. The alignment that can be specified in *pOpt* must be between 256 KB and 16 MB and a power of 2.

sceKernelFreeMemBlock

Free memory blocks

Definition

```
#include <kernel.h>
int sceKernelFreeMemBlock(
        SceUID uid
);
```

Arguments

uid Memory block ID

Return Values

Value	Description
SCE_OK	Success
<SCE_OK	Error code

Description

This frees the memory blocks.

Document serial number: 000004892117

sceKernelGetMemBlockBase

Get mapped base address of memory block

Definition

```
#include <kernel.h>
int sceKernelGetMemBlockBase(
        SceUID uid,
        void **ppBase
);
```

Arguments

uid Memory block ID
ppBase Virtual address of memory block

Return Values

Value	Description
SCE_OK	Success
<SCE_OK	Error code

Description

This obtains the base address where a memory block is mapped.

sceKernelFindMemBlockByAddr

Get memory block identifier through virtual address

Definition

```
#include <kernel.h>
SceUID sceKernelFindMemBlockByAddr(
        void *uaddr,
        SceSize vsize
);
```

Arguments

uaddr Base address
vsize Virtual address size

Return Values

Value	Description
SCE_OK	Success
<SCE_OK	Error code

Description

This obtains the identifier of the memory block that includes the specified virtual address range.

Document serial number: 000004892117

sceKernelGetMemBlockInfoByAddr

Get memory block information

Definition

```
#include <kernel.h>
int sceKernelGetMemBlockInfoByAddr(
        void *vbase,
        SceKernelMemBlockInfo *pInfo
);
```

Arguments

vbase Base address
pInfo Information structure

Return Values

Value	Description
SCE_OK	Success
<SCE_OK	Error code

Description

This obtains the information of the memory block that includes the specified virtual address.

Memory Management

sceKernelGetFreeMemorySize

Get free memory information

Definition

```
#include <kernel.h>
int sceKernelGetFreeMemorySize(
        SceKernelFreeMemorySizeInfo*pInfo
);
```

Arguments

pInfo Information structure

Return Values

Value	Description
SCE_OK	Success
<SCE_OK	Error code

Description

This obtains the free memory information. The returned free memory size is the actual memory size minus a margin for system operation.

Thread Manager

Basic Thread Feature

SceKernelThreadEntry

Thread entry type

Definition

```
#include <kernel.h>
typedef SceInt32 (*SceKernelThreadEntry)(
        SceSize argSize,
        void *pArgBlock
);
```

Arguments

argSize	The argument size given with `sceKernelStartThread()` is passed.
pArgBlock	The address on the stack where the argument block given with `sceKernelStartThread()` is copied is passed.

Return Values

Thread exit status

Description

This is the type of the thread entry function. It is specified in `sceKernelCreateThread()`. The argument block of the thread given with the argument of `sceKernelStartThread()` is copied to the stack of the thread that has been started, and is received from the started thread through the argument of the entry function.

When the thread's entry function is terminated with return, the return value of the entry function will be the exit status of that thread. The exit status of the thread can be obtained either with `sceKernelWaitThreadEnd()` or, if the thread has not yet been deleted, with `sceKernelGetThreadExitStatus()`.

SceKernelChangeStackFunction

Type for the stack changing function

Definition

```
#include <kernel.h>
typedef SceInt32 (*SceKernelChangeStackFunction)(
        void *pArg
);
```

Arguments

pArg Arbitrary pointer to pass to stack changing function

Return Values

Becomes the return value of `sceKernelCallWithChangeStack()`

Description

This is the type for the function called upon changing the stack. It is specified to `sceKernelCallWithChangeStack()`.

`sceKernelCallWithChangeStack()` calls the stack changing function; and the stack is switched. When the stack changing function terminates with return, stack is switched back to the original stack.

SceKernelThreadOptParam

Thread optional data

Definition

```
#include <kernel.h>
typedef struct _SceKernelThreadOptParam {
        SceSize size;
        SceUInt32 attr;
} SceKernelThreadOptParam;
```

Members

`size` Size of this structure. (Value of `sizeof(SceKernelThreadOptParam)`)
`attr` Bit pattern specifying valid member within the structure

Description

This structure is used with `sceKernelCreateThread()` to store optional data that is provided when a thread is created.

It is provided for future expansion.

SceKernelThreadInfo

Thread information structure

Definition

```
#include <kernel.h>
typedef struct _SceKernelThreadInfo {
        SceSize size;
        SceUID processId;
        char name[SCE_UID_NAMELEN+1];
        SceUInt32 attr;
        SceUInt32 status;
        SceKernelThreadEntry entry;
        void *pStack;
        SceSize stackSize;
        SceInt32 initPriority;
        SceInt32 currentPriority;
        SceInt32 initCpuAffinityMask;
        SceInt32 currentCpuAffinityMask;
        SceInt32 currentCpuId;
        SceInt32 lastExecutedCpuId;
        SceUInt32 waitType;
        SceUID waitId;
        SceInt32 exitStatus;
        SceKernelSysClock runClocks;
        SceUInt32 intrPreemptCount;
        SceUInt32 threadPreemptCount;
        SceUInt32 threadReleaseCount;
        SceInt32 changeCpuCount;
        SceInt32 fNotifyCallback;
        SceInt32 reserved;
} SceKernelThreadInfo;
```

Members

size	Size of this structure. (Value of `sizeof(SceKernelThreadInfo)`)
processId	Process ID belonging to thread
name	Process name
attr	Thread attribute
status	Thread status
entry	Address of thread entry function
pStack	Address of thread stack (bottom)
stackSize	Thread stack size
initPriority	Initial priority of thread
currentPriority	Current priority of thread
initCpuAffinityMask	Initial CPU affinity mask of thread
currentCpuAffinityMask	Current CPU affinity mask of thread
currentCpuId	Current CPU number to which thread belongs
lastExecutedCpuId	Last CPU on which a thread is executed
waitType	Type when the thread is in wait state
waitId	Target UID of a synchronous object when the thread is in wait state
exitStatus	Thread exit status
runClocks	Time of execution from thread creation to present (in microseconds)
intrPreemptCount	Number of times thread is preempted due to interrupts
threadPreemptCount	Number of times thread is preempted due to other threads' operations
threadReleaseCount	Number of times thread has spontaneously released the right to execute

changeCpuCount	Number of times thread moved executed CPU
fNotifyCallback	Flag indicating whether callback has been reported to the thread
reserved	Reserved area

Description

This structure stores the thread information by `sceKernelGetThreadInfo()`.

SceKernelThreadRunStatus

Thread run status for each CPU

Definition

```
#include <kernel.h>
typedef struct _SceKernelThreadRunStatus {
        SceSize size;
        struct {
              SceUID processId;
              SceUID threadId;
              SceInt32 priority;
        } cpuInfo[SCE_KERNEL_MAX_CPU];
} SceKernelThreadRunStatus;
```

Members

size	Size of this structure. (Value of `sizeof(SceKernelThreadRunStatus)`)
processId	ID of the process to which the thread in execution belongs
threadId	ID of the thread in execution
priority	Priority of the thread in execution
cpuInfo	Individual CPU-related information

Description

This structure is used to obtain the thread run status for each CPU with `sceKernelGetThreadRunStatus()`.

SceKernelSystemInfo

System status

Definition

```
#include <kernel.h>
typedef struct _SceKernelSystemInfo {
        SceSize size;
        SceUInt32 activeCpuMask;
        struct {
              SceKernelSysClock idleClock;
              SceUInt32 comesOutOfIdleCount;
              SceUInt32 threadSwitchCount;
        } cpuInfo[SCE_KERNEL_MAX_CPU];
} SceKernelSystemInfo;
```

Members

size	Size of this structure. (Value of `sizeof(SceKernelSystemInfo)`)
activeCpuMask	CPU mask that can be used at present
idleClock	Time when there are no threads to be executed by the CPU
comesOutOfIdleCount	Number of transitions from the state in which the CPU has no executing threads to the state with executing threads
threadSwitchCount	Number of times context switching was performed by CPU
cpuInfo	Individual CPU-related information

Description

This structure is used to obtain the system status with `sceKernelGetSystemInfo()`.

sceKernelCreateThread

Create thread to run in address space of this process

Definition

```
#include <kernel.h>
SceUID sceKernelCreateThread(
        const char *pName,
        SceKernelThreadEntry entry,
        SceInt32 initPriority,
        SceSize stackSize,
        SceUInt32 attr,
        SceInt32 cpuAffinityMask,
        const SceKernelThreadOptParam *pOptParam
);
```

Arguments

pName	Specify the thread's name. Since the thread name is only used for the purpose of identification by the operator during debugging, it need not be unique. The name's maximum length is 31 bytes. NULL cannot be specified.
entry	Specify the thread's entry address. The function that is the thread's entry point can receive memory block data with two arguments. The arguments are given with `sceKernelStartThread()` (see below). The thread exits by returning from this function, and the function's return value will be the thread's exit status.
initPriority	Specify the thread's initial priority. Priority will be higher for smaller numbers. The range that can be used goes from `SCE_KERNEL_HIGHEST_PRIORITY_USER(=64)` to `SCE_KERNEL_LOWEST_PRIORITY_USER(=191)`. By specifying `SCE_KERNEL_DEFAULT_PRIORITY_USER`, it is possible to specify the default priority of the process to which the thread belongs. The default priority can also be specified with an offset (`SCE_KERNEL_DEFAULT_PRIORITY_USER` -31...+32). By specifying `SCE_KERNEL_CURRENT_THREAD_PRIORITY(=0)`, it is possible to specify the priority of the calling thread for this function. It is not possible to specify an offset for `SCE_KERNEL_CURRENT_THREAD_PRIORITY`.
stackSize	Specify the thread's necessary stack size in bytes. Since the argument block given with `sceKernelStartThread()` is copied to the stack, leave room for this when specifying stack size. When a stack size which is not multiples of 4 KiB is specified, the actual stack size will be rounded up in 4 KiB units. The maximum specifiable value is `SCE_KERNEL_THREAD_STACK_SIZE_MAX` (32 MiB). The minimum value is `SCE_KERNEL_THREAD_STACK_SIZE_MIN` (4 KiB).
attr	Multiple thread attributes can be specified with logical OR. Currently, there are no attributes that can be specified. (0x80000000 will be obtained when observing from the debugger due to the system's automatic addition of a flag (0x80000000) indicating that this is a user thread.)

cpuAffinityMask	Specify the thread's initial CPU affinity mask. It is possible to run threads only specified with the CPU affinity mask. The CPU affinity mask can be specified through logical OR of the following macros: - `SCE_KERNEL_CPU_MASK_USER_0` - `SCE_KERNEL_CPU_MASK_USER_1` - `SCE_KERNEL_CPU_MASK_USER_2` It is also possible to use the following macro in order to represent all usable CPUs. - `SCE_KERNEL_CPU_MASK_USER_ALL` Moreover, the following macro can be used as the default CPU affinity mask of the thread. - `SCE_KERNEL_THREAD_CPU_AFFINITY_MASK_DEFAULT` The default CPU affinity mask is set by the system for each process. In the case of game applications, CPUs capable of running for the thread are currently the same as when `SCE_KERNEL_CPU_MASK_USER_ALL` is specified.
pOptParam	Argument for future expansion. Specify NULL.

Return Values

Value	Description
Positive value	Thread identifier (UID)
Negative value	Error code

Description

This creates a thread operated within an address space of the current process.

This allocates the thread management area for the thread to be created, performs the initial settings, and reserves the stack area. Information about the thread to be created is specified in the arguments, and the function returns the thread identifier (UID). The created thread is placed in DORMANT state.

Note
For a more detailed description of thread priorities and CPU affinity masks, and notes on settings, refer to the "Kernel Overview" document.

When a thread is created, the following memories are removed from the application's memory budget in addition to the user mode stack memory specified with *stackSize*.

- User mode TLS: Size obtained by rounding up in 4 KiB units "reserved 2 KiB + Size used by module as *_thread* variable" (at least 4 KiB)

sceKernelStartThread

Start thread

Definition

```
#include <kernel.h>
SceInt32 sceKernelStartThread(
        SceUID threadId,
        SceSize argSize,
        const void *pArgBlock
);
```

Arguments

threadId	Specify the identifier of the thread to be started.
argSize	Specify the size (in bytes) of the argument to be passed to the thread.
pArgBlock	Specify the address of the argument to be passed to the thread.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This starts execution of the thread specified with *threadId*, and then places it in READY state.

An argument block specified by *argSize* and *argBlock* is copied to the thread's stack, *argSize* is directly passed to the first argument of the thread's entry function, and the address of the argument block that was copied to the stack is passed to the second argument.

This system call does not perform start request queuing. In other words, if the target thread is not in DORMANT state, an `SCE_KERNEL_ERROR_NOT_DORMANT` error is returned to the caller thread.

When staring a thread, the priority, CPU affinity mask, and attribute of the thread are initialized each time with the value specified to the arguments *initPriority*, *cpuAffinityMask*, and *attr* of `sceKernelCreateThread()` when respective threads were generated.

sceKernelExitThread

Exit this thread

Definition

```
#include <kernel.h>
SceInt32 sceKernelExitThread(
        SceInt32 exitStatus
);
```

Arguments

exitStatus Specify the value set to the exit status of calling thread.

Return Values

Value	Description
Negative value	Error code

Description

This causes the calling thread to terminate itself normally and transition to DORMANT state.

`sceKernelExitThread()` is a system call that does not return to the issuing context.

Note that resources (such as memory and semaphores) that were acquired by the thread to be terminated are not automatically released. However, in the case of a mutex, if a thread is terminated while a mutex is locked, the mutex is automatically unlocked. Note that the mutex is only unlocked and is not deleted.

If the terminated thread is restarted by `sceKernelStartThread()`, the information contained in the thread management area, such as the thread priority, is set again.

This information is not inherited when the thread is terminated.

When the thread is terminated, *exitStatus* is used as the exit status for that thread.

If a thread is terminated by returning from the entry function without using `sceKernelExitThread()`, the return value of the entry function is used as the exit status.

The exit status of a thread can be obtained by using `sceKernelGetThreadExitStatus()` or `sceKernelGetThreadInfo()`, and if another thread is waiting for that thread to terminate by `sceKernelWaitThreadEnd()`, the exit status can be obtained from the return value.

Although the exit status of a thread is saved when the thread terminates and is in DORMANT state, the value is lost when the thread is restarted by using `sceKernelStartThread()` or when the thread is deleted.

sceKernelDeleteThread

Delete thread

Definition

```
#include <kernel.h>
SceInt32 sceKernelDeleteThread(
        SceUID threadId
);
```

Arguments

threadId Specify the identifier of the thread to be deleted.

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This deletes the thread specified by *threadId*.

When the specified thread is deleted, the stack area and thread management area are freed. The specified thread must be in DORMANT state.

The calling thread cannot be specified as the thread to be deleted because it cannot be in DORMANT state. An SCE_KERNEL_ERROR_NOT_DORMANT error will occur. To delete the calling thread, use sceKernelExitDeleteThread().

sceKernelExitDeleteThread

Exit and delete this thread

Definition

```
#include <kernel.h>
SceInt32 sceKernelExitDeleteThread(
        SceInt32 exitStatus
);
```

Arguments

exitStatus Specify the value to be set to the exit status of calling thread.

Return Values

Value	Description
Negative value	Error code

Description

This causes the calling thread to terminate itself normally, and then deletes the thread. `sceKernelExitDeleteThread()` is a system call that does not return to the issuing context.

Note that resources (such as memory and semaphores) that were acquired by the thread to be terminated are not automatically released. However, in the case of a mutex, if a thread is terminated while a mutex is locked, the mutex is automatically unlocked. Note that the mutex is only unlocked and is not deleted.

When a thread terminates, *exitStatus* is used as the exit status of that thread. Because `sceKernelExitDeleteThread()` deletes a thread immediately without having the thread transition to DORMANT state, the exit status of a thread that was terminated by this system call cannot be obtained by using `sceKernelGetThreadExitStatus()` or `sceKernelGetThreadInfo()`. The exit status can be obtained from the return value only when `sceKernelWaitThreadEnd()` is used to wait for the termination of the thread.

sceKernelChangeThreadCpuAffinityMask

Set CPU affinity mask of thread

Definition

```
#include <kernel.h>
SceInt32 sceKernelChangeThreadCpuAffinityMask(
        SceUID threadId,
        SceInt32 cpuAffinityMask
);
```

Arguments

threadId Specify the identifier of the thread for which the CPU affinity mask is to be changed.
The calling thread can be specified with `SCE_KERNEL_THREAD_ID_SELF`.

cpuAffinityMask Specify the CPU affinity mask after change.
The CPU affinity mask is represented through logical OR of the following macros.
- `SCE_KERNEL_CPU_MASK_USER_0`
- `SCE_KERNEL_CPU_MASK_USER_1`
- `SCE_KERNEL_CPU_MASK_USER_2`

It is also possible to use the following macro in order to represent all usable CPUs.
- `SCE_KERNEL_CPU_MASK_USER_ALL`

Moreover, the following macro can be used as the default CPU affinity mask of the thread.
- `SCE_KERNEL_THREAD_CPU_AFFINITY_MASK_DEFAULT`

The default CPU affinity mask is set by the system for each process. In the case of game applications, CPUs capable of running for the thread are currently the same as when `SCE_KERNEL_CPU_MASK_USER_ALL` is specified.

Return Values

Value	Description
Positive value	CPU affinity mask before being changed
Negative value	Error code

Description

This changes the CPU affinity mask of the thread specified with *threadId* to *cpuAffinityMask*. The thread can run only on the CPU specified in the CPU affinity mask.

If changes to the CPU affinity mask of the thread currently being executed prevents the thread from being executed on the current CPU, that thread is immediately dispatched. If a thread with a higher priority is running on a CPU that has become available for execution due to scheduling, the thread which CPU affinity mask has been changed is forced to wait in READY state.

However, if the thread was in the no-dispatch state, changes to the CPU affinity mask are performed immediately, but re-scheduling processing is delayed until dispatch is permitted.

When this function is executed for a thread where `SCE_KERNEL_THREAD_CPU_AFFINITY_MASK_DEFAULT` is specified as the CPU affinity mask, the CPU affinity mask preceding the change that is obtained as the return value will be `SCE_KERNEL_THREAD_CPU_AFFINITY_MASK_DEFAULT`.

sceKernelGetThreadCpuAffinityMask

Get CPU affinity mask of thread

Definition

```
#include <kernel.h>
SceInt32 sceKernelGetThreadCpuAffinityMask(
        SceUID threadId
);
```

Arguments

`threadId` Specify the identifier of the thread for obtaining the CPU affinity mask. It is possible to specify the calling thread with `SCE_KERNEL_THREAD_ID_SELF`.

Return Values

Value	Description
Positive value	CPU affinity mask
Negative value	Error code

Description

This obtains the CPU affinity mask of the thread specified with *threadId*.

The CPU affinity mask is represented through logical OR of the following macros.

- `SCE_KERNEL_CPU_MASK_USER_0`
- `SCE_KERNEL_CPU_MASK_USER_1`
- `SCE_KERNEL_CPU_MASK_USER_2`

The values of valid CPU affinity masks are always positive.

When this function is executed for a thread where `SCE_KERNEL_THREAD_CPU_AFFINITY_MASK_DEFAULT` is specified as the CPU affinity mask, `SCE_KERNEL_THREAD_CPU_AFFINITY_MASK_DEFAULT` will be obtained.

sceKernelChangeThreadPriority

Change thread priority

Definition

```
#include <kernel.h>
SceInt32 sceKernelChangeThreadPriority(
        SceUID threadId,
        SceInt32 priority
);
```

Arguments

threadId Specify the identifier of the thread whose priority is to be changed.
It is possible to specify the calling thread with `SCE_KERNEL_THREAD_ID_SELF`.

priority Specify priority after change.
Thread priority will be higher for smaller numbers.
The range that can be used goes from `SCE_KERNEL_HIGHEST_PRIORITY_USER` (=64) to `SCE_KERNEL_LOWEST_PRIORITY_USER` (=191).
The default priority of the process to which the thread belongs can be specified with `SCE_KERNEL_DEFAULT_PRIORITY_USER`. Also, default priority can be specified with `SCE_KERNEL_DEFAULT_PRIORITY_USER` ± offset.
With `SCE_KERNEL_CURRENT_THREAD_PRIORITY` (=0), it is possible to specify the priority of the calling thread for this function.
It is not possible to specify an offset for `SCE_KERNEL_CURRENT_THREAD_PRIORITY`.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This changes the priority of the thread specified with *threadId* to *priority*.

The new priority set by this system call is valid unless it is changed again, before the thread is terminated. If the thread is in DORMANT state, the priority of the thread when it was terminated will be discarded, and the priority when the thread is started the next time will be the startup priority (*initPriority*) that was specified when the thread was created.

If the target thread had been enqueued in the wait queue with priority, the queue order may change due to this system call.

sceKernelChangeThreadPriority2

Change thread priority

Definition

```
#include <kernel.h>
SceInt32 sceKernelChangeThreadPriority2(
        SceUID threadId,
        SceInt32 priority
);
```

Arguments

threadId	Specify the identifier of the thread whose priority is to be changed. It is possible to specify the calling thread with `SCE_KERNEL_THREAD_ID_SELF`.
priority	Specify priority after change. Thread priority will be higher for smaller numbers. The range that can be used goes from `SCE_KERNEL_HIGHEST_PRIORITY_USER` (=64) to `SCE_KERNEL_LOWEST_PRIORITY_USER` (=191). The default priority of the process to which the thread belongs can be specified with `SCE_KERNEL_DEFAULT_PRIORITY_USER`. Also, default priority can be specified with `SCE_KERNEL_DEFAULT_PRIORITY_USER` ± offset. With `SCE_KERNEL_CURRENT_THREAD_PRIORITY` (=0), it is possible to specify the priority of the calling thread for this function. It is not possible to specify an offset for `SCE_KERNEL_CURRENT_THREAD_PRIORITY`.

Return Values

Value	Description
Positive value	Priority before change
Negative value	Error code

Description

This changes the priority of the thread specified with *threadId* to *priority*.

The new priority set by this system call is valid unless it is changed again, before the thread is terminated. If the thread is in DORMANT state, the priority of the thread when it was terminated will be discarded, and the priority when the thread is started the next time will be the startup priority (*initPriority*) that was specified when the thread was created.

If the target thread had been enqueued in the wait queue with priority, the queue order may change due to this system call.

The difference from `sceKernelChangeThreadPriority()` is that this function returns the priority before it was changed upon successful processing completion.

sceKernelGetThreadCurrentPriority

Get current priority of this thread

Definition

```
#include <kernel.h>
SceInt32 sceKernelGetThreadCurrentPriority(
        void
);
```

Arguments

None

Return Values

Value	Description
Positive value	Current calling thread priority
Negative value	Error code

Description

This obtains the current priority of the calling thread.

sceKernelGetThreadId

Get this thread identifier (thread ID)

Definition

```
#include <kernel.h>
SceUID sceKernelGetThreadId(
        void
);
```

Arguments

None

Return Values

Value	Description
Positive value	Thread identifier
Negative value	Error code

Description

This obtains the identifier (UID) of the calling thread. The values of valid thread identifiers are always positive.

sceKernelChangeCurrentThreadAttr

Change current thread attribute

Definition

```
#include <kernel.h>
SceInt32 sceKernelChangeCurrentThreadAttr(
        SceUInt32 clearAttr,
        SceUInt32 setAttr
);
```

Arguments

clearAttr	Specify attribute bit pattern to be cleared
setAttr	Specify attribute bit pattern to be set

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This changes the attributes of the current thread. At present, there are no changeable attributes.

sceKernelGetThreadExitStatus

Get exit status of thread

Definition

```
#include <kernel.h>
SceInt32 sceKernelGetThreadExitStatus(
        SceUID threadId
        SceInt32 *pExitStatus
);
```

Arguments

threadId	Specify the identifier of the thread for which exit status is to be obtained.
pExitStatus	Specify the pointer to the `SceInt32` type variable receiving the exit status of the target thread.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This obtains the exit status of the thread specified with *threadId*.

The exit status can be obtained only for threads that were started, ended normally, and transitioned back to DORMANT state. If the target thread has not been started since it was created, `SCE_KERNEL_ERROR_DORMANT` is returned. If the target thread has not ended yet, `SCE_KERNEL_ERROR_NOT_DORMANT` is returned.

If the target thread is in DELETED state and is not referenced by any contexts, the thread is deleted, and therefore, the exit status can no longer be obtained by `sceKernelGetThreadExitStatus()`.

sceKernelGetProcessId

Get the identifier of the process (process ID) to which the calling thread belongs

Definition

```
#include <kernel.h>
SceUID sceKernelGetProcessId(
        void
);
```

Arguments

None

Return Values

Value	Description
Positive value	Identifier of the process to which the calling thread belongs
Negative value	Error code

Description

This obtains the identifier of the process to which the calling thread belongs. The values of valid thread identifiers are always positive.

Document serial number: 000004892117

Document serial number: 000004892117

sceKernelCheckWaitableStatus

Determine possibility of current context transitioning to wait state

Definition

```
#include <kernel.h>
SceInt32 sceKernelCheckWaitableStatus(
        void
);
```

Arguments

None

Return Values

Value	Description
SCE_OK	Callable
Negative value	Not callable (error code of the reason)

Description

This determines whether or not the current context can call a system call that can be in wait state. In the thread context, there are no situations in which system call that can be in wait state cannot be called.

Document serial number: 000004892117

Document serial number: 000004892117

sceKernelGetThreadInfo

Get thread status

Definition

```
#include <kernel.h>
SceInt32 sceKernelGetThreadInfo(
        SceUID threadId,
        SceKernelThreadInfo *pInfo
);
```

Arguments

threadId	Specify the identifier of the thread whose status is to be obtained. The calling thread can be specified with `SCE_KERNEL_THREAD_ID_SELF`.
pInfo	Specify the pointer to the structure variable receiving thread status. Always assign `sizeof(SceKernelThreadInfo)` to *pInfo*->*size* when calling.

Return Values

Value	Description
Positive value	Success
Negative value	Error code

Description

This obtains the thread status.

This system call is provided to aid debugging. Information that can be obtained by this system call changes moment by moment. Programming that issues this system call frequently, then changes the control flow according to the information that was received is not recommended.

sceKernelGetThreadRunStatus

Get current thread run status in each CPU

Definition

```
#include <kernel.h>
SceInt32 sceKernelGetThreadRunStatus(
        SceKernelThreadRunStatus *pStatus
);
```

Arguments

pStatus	Specify the pointer to the structure variable receiving thread run status in each CPU. Always assign `sizeof(SceKernelThreadRunStatus)` to *pStatus*->*size* when calling.

Return Values

Value	Description
Positive value	Success
Negative value	Error code

Description

This obtains the current thread run status for each CPU.

This system call is provided to aid debugging. Information that can be obtained by this system call changes moment by moment. Programming that issues this system call frequently, then changes the control flow according to the information that was received is not recommended.

sceKernelGetSystemInfo

Get system status

Definition

```
#include <kernel.h>
SceInt32 sceKernelGetSystemInfo(
        SceKernelSystemInfo *pInfo
);
```

Arguments

pInfo Specify the pointer to the structure variable receiving system status.
Always assign `sizeof(SceKernelSystemInfo)` to *pInfo*->*size* when calling.

Return Values

Value	Description
Positive value	Exit status
Negative value	Error code

Description

This obtains the current system status.

This system call is provided to aid debugging. Information that can be obtained by this system call changes moment by moment. Programming that issues this system call frequently, then changes the control flow according to the information that was received is not recommended.

sceKernelGetThreadmgrUIDClass

Get class of identifier (UID) managed by thread manager

Definition

```
#include <kernel.h>
SceInt32 sceKernelGetThreadmgrUIDClass(
        SceUID uid
);
```

Argument

uid Specify the identifier for obtaining UID class

Return Values

Value	Description
Positive value	UID class
Negative value	Error code

Description

This obtains the class of identifiers from the identifiers of threads, semaphores, event flags, etc., managed by the thread manager.

sceKernelCheckThreadStack

Get remaining stack size of own thread

Definition

```
#include <kernel.h>
SceSize sceKernelCheckThreadStack(
        void
);
```

Arguments

None

Return Values

Value	Description
Value other than 0	Remaining stack size
0	Stack overflow or underflow occurred

Description

This function obtains the remaining stack size of the thread that called this function at the timing of the function call.

sceKernelCallWithChangeStack

Change stack of own thread

Definition

```
#include <kernel.h>
SceInt32 sceKernelCallWithChangeStack(
        void *pBase,
        SceSize size,
        SceKernelChangeStackFunction changeStackFunc,
        void *pCommon
);
```

Arguments

pBase	Base address of stack to set
size	Size of stack to set
changeStackFunc	Pointer to function to call after stack change
pCommon	Address passed as argument in function specified in *changeStackFunc*

Return Values

Return value of *changeStackFunc*

Description

This function changes the stack address range of its own thread.

In addition to changing the stack, this function calls the function specified in *changeStackFunc*.

When using memory (other than the stack set upon thread generation) as stack, this function must always be used.

The entire set address range must be valid (read and write enabled).

VFP Exceptions

sceKernelChangeThreadVfpException

Change VFP exception trap setting of this thread

Definition

```
#include <kernel.h>
SceInt32 sceKernelChangeThreadVfpException(
        SceInt32 clearMask,
        SceInt32 setMask
);
```

Argument

clearMask	Specify VFP exception to disable trap with bit patterns. VFP exceptions are represented through logical OR of the following macros: - `SCE_KERNEL_VFP_EXCEPTION_MASK_QCE` - `SCE_KERNEL_VFP_EXCEPTION_MASK_IDE` - `SCE_KERNEL_VFP_EXCEPTION_MASK_IXE` - `SCE_KERNEL_VFP_EXCEPTION_MASK_UFE` - `SCE_KERNEL_VFP_EXCEPTION_MASK_OFE` - `SCE_KERNEL_VFP_EXCEPTION_MASK_DZE` - `SCE_KERNEL_VFP_EXCEPTION_MASK_IOE` In order to specify all VFP exceptions the following macro can be used: - `SCE_KERNEL_VFP_EXCEPTION_MASK_ALL`
setMask	Specify VFP exception to enable trap with bit patterns. Specifiable bit patterns are the same as *clearMask*.

Return Values

Value	Description
Value of 0 or higher	Setting before change
Negative value	Error code

Description

This changes VFP exception trap setting of this thread.

If an enabled VFP exception occurs in the thread, the process to which the thread belongs will stop running. However, unlike in the case of CPU exception, there is a slight delay between the occurrence of a VFP exception and the stopping of the process. For this reason, when the process stops the program will have progressed further than at the moment of the occurrence of the VFP exception.

sceKernelGetCurrentThreadVfpException

Get current VFP exception trap setting of this thread

Definition

```
#include <kernel.h>
SceInt32 sceKernelGetCurrentThreadVfpException(
        void
);
```

Argument

None

Return Values

Value	Description
Value of 0 or higher	Current VFP exception trap setting
Negative value	Error code

Description

This obtains the current VFP exception trap setting of this thread.

Direct Thread Synchronization Feature

sceKernelDelayThread

Delay thread

Definition

```
#include <kernel.h>
SceInt32 sceKernelDelayThread(
        SceUInt32 usec
);
```

Arguments

usec Specify delay time in a unit of microseconds

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This temporarily stops execution of the calling thread and enters an elapsed time wait state.

Specify the stop time in units of microseconds. When the time gets close to the finish time for multiple threads that have called sceKernelDelayThread() (200 microseconds or less), the threads will be collectively returned from their wait states at the latest time.

If this is executed by specifying 0 in *usec*, SCE_KERNEL_ERROR_INVALID_ARGUMENT error returns.

Note
This function serves to delay the thread that calls it by at least the time specified in *usec*.
Depending on the *usec* value, the CPU's right to execute might be transferred to another thread; however, this is not guaranteed.
Therefore, programming whereby the CPU's right to execute is expected to be transferred to another thread by using this function risks causing serious problem from the standpoint of future compatibility. Make sure to avoid this.
We strongly recommend controlling how threads run by making appropriate use of priority/CPU affinity mask settings as well as of the various synchronization objects provided by the kernel.

sceKernelDelayThreadCB

Delay thread (with callback check feature)

Definition

```
#include <kernel.h>
SceInt32 sceKernelDelayThreadCB(
        SceUInt32 usec
);
```

Arguments

usec Specify delay time in a unit of microseconds

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This temporarily stops execution of the calling thread and enters an elapsed time wait state.

Specify the stop time in units of microseconds. When the time gets close to the finish time for multiple threads that have called sceKernelDelayThread() (200 microseconds or less), the threads will be collectively returned from their wait states at the latest time.

If this function is executed by specifying 0 in *usec*, SCE_KERNEL_ERROR_INVALID_ARGUMENT error returns.

Note
This function serves to delay the thread that calls it by at least the time specified in *usec*.
Depending on the *usec* value, the CPU's right to execute might be transferred to another thread; however, this is not guaranteed.
Therefore, programming whereby the CPU's right to execute is expected to be transferred to another thread by using this function risks causing serious problem from the standpoint of future compatibility. Make sure to avoid this.
We strongly recommend controlling how threads run by making appropriate use of priority/CPU affinity mask settings as well as of the various synchronization objects provided by the kernel.

sceKernelDelayThreadCB() is a system call that adds a feature for checking whether or not a callback notification exists, to sceKernelDelayThread(). Callback notifications will be checked immediately when this function is called, regardless of whether calling this function causes the thread to enter wait state. If the thread subsequently enters wait state, checks will continue while waiting also. If a callback notification is received for the calling thread while waiting for elapsed time, the thread temporarily exits from the wait state, the callback function is executed, and then the thread enters the wait state again.

sceKernelWaitThreadEnd

Wait for other thread to end

Definition

```
#include <kernel.h>
SceInt32 sceKernelWaitThreadEnd(
        SceUID threadId,
        SceInt32 *pExitStatus,
        SceUInt32 *pTimeout
);
```

Arguments

threadId	Specify the identifier of the thread for which to wait for termination.
pExitStatus	Specify the pointer to the `SceInt32` type variable receiving the exit status of the target thread. Exit status will not be received if NULL is specified.
pTimeout	Specify the pointer to the `SceUInt32` type variable storing maximum waiting time (in microseconds). If NULL is specified waiting time will be infinite. When the wait condition is satisfied, the remaining time will return.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This waits for another thread specified by *threadId* to end.

When the wait for the thread to end is successfully completed, `SCE_OK` is returned. The exit status provided as the return value of the thread's entry function (or the argument of `sceKernelExitThread()` or `sceKernelExitDeleteThread()`) is received to *pExitStatus*.

When the wait for the thread to end fails, an error is returned.

sceKernelWaitThreadEndCB

Wait for other thread to end (with callback check feature)

Definition

```
#include <kernel.h>
SceInt32 sceKernelWaitThreadEndCB(
        SceUID threadId,
        SceInt32 *pExitStatus,
        SceUInt32 *pTimeout
);
```

Arguments

threadId	Specify the identifier of the thread for which to wait for termination.
pExitStatus	Specify the pointer to the `SceInt32` type variable receiving the exit status of the target thread. Exit status will not be received if NULL is specified.
pTimeout	Specify the pointer to the `SceUInt32` type variable storing maximum waiting time (in microseconds). If NULL is specified waiting time will be infinite. When waiting conditions are established, the remaining time will return.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This waits for another thread specified by *threadId* to end.

When the waiting of the thread to end is successful, `SCE_OK` is returned. The exit status provided as the return value of the thread's entry function (or the argument of `sceKernelExitThread()` or `sceKernelExitDeleteThread()`) is received to *pExitStatus*.

If the waiting of the thread to end failed, an error is returned.

`sceKernelWaitThreadEndCB()` is a system call that adds a feature for checking whether or not a callback notification exists, to `sceKernelWaitThreadEnd()`. Callback notifications will be checked immediately when this function is called, regardless of whether calling this function causes the thread to enter wait state. If the thread subsequently enters wait state, checks will continue while waiting also. If a callback notification is received for the calling thread while waiting for time to elapse, the thread temporarily exits from the wait state, the callback function is executed, and then the thread enters the wait state again.

Document serial number: 000004892117

Callbacks

SceKernelCallbackFunction

Callback function type

Definition

```
#include <kernel.h>
typedef SceInt32 (*SceKernelCallbackFunction)(
        SceUID notifyId,
        SceInt32 notifyCount,
        SceInt32 notifyArg,
        void *pCommon
);
```

Arguments

notifyId	In case of callback notification from a event to which callback was registered with `sceKernelRegisterCallbackToEvent()`, the identifier of the notifier event will be passed. If notification is not received from an event, `SCE_UID_INVALID_UID` will be passed.
notifyCount	The number of times callback is reported until this callback function is executed is passed.
notifyArg	The argument given by `sceKernelNotifyCallback()` is passed.
pCommon	The argument given by `sceKernelCreateCallback()` is passed.

Return Values

Value	Description
0	Normal
Not 0	Delete this callback

Description

A callback function is called by confirming that notification using `sceKernelCheckCallback()` after the occurrence of a callback is reported by `sceKernelNotifyCallback()`, or by executing a wait API such as `sceKernelWaitSemaCB()` that checks for the occurrence of a particular notification. By returning a value other than 0, the callback function can cancel its own registration.

Precautions

- The callback function is called in the context of the thread that created the callback. In other words, only a callback that was created by the calling thread can generate a callback function call either by an explicit callback check with `sceKernelCheckCallback()` or by a wait function with a callback check such as `sceKernelWaitSemaCB()`. To receive the callback notification, the thread that created the callback must use a function such as `sceKernelCheckCallback()` or `sceKernelWaitSemaCB()`.
- The `sceKernelCheckCallback()` and `sceKernelDeleteCallback()` functions cannot be used within a callback function. Also, if a wait function with callback, such as the `sceKernelWaitSemaCB()` function, is called within a callback function, nesting will occur and this may cause a stack overflow. Do not perform processing that causes an additional callback wait within a callback function.

SceKernelCallbackInfo

Callback status

Definition

```
#include <kernel.h>
typedef struct _SceKernelCallbackInfo {
        SceSize size;
        SceUID callbackId;
        char name[SCE_UID_NAMELEN+1];
        SceUInt32 attr;
        SceUID threadId;
        SceKernelCallbackFunction callbackFunc;
        SceUID notifyId;
        SceInt32 notifyCount;
        SceInt32 notifyArg;
        void *pCommon;
} SceKernelCallbackInfo;
```

Members

size	Size of this structure. (Value of `sizeof(SceKernelCallbackInfo)`)
callbackId	Callback identifier
name	Callback name specified with `sceKernelCreateCallback()`
attr	Callback attribute specified with `sceKernelCreateCallback()`
threadId	Identifier of thread callback belongs to
callbackFunc	Callback function registered by `sceKernelCreateCallback()`
notifyId	Identifier of callback notifier event
notifyCount	Number of times `sceKernelNotifyCallback()` is executed for this callback while the callback function has not yet been called and is delayed.
notifyArg	Argument given by `sceKernelNotifyCallback()` to this callback function
pCommon	*pCommon* argument registered by `sceKernelCreateCallback()`

Description

This structure is used to obtain the callback status with `sceKernelGetCallbackInfo()`.

sceKernelCreateCallback

Create callback

Definition

```
#include <kernel.h>
SceUID sceKernelCreateCallback(
        const char *pName,
        SceUInt32 attr,
        SceKernelCallbackFunction callbackFunc,
        void *pCommon
);
```

Arguments

pName	Specify callback name. Since the thread name is only used for the purpose of identification by the operator during debugging, it need not be unique. The name's maximum length is 31 bytes. NULL cannot be specified.
attr	Specify callback attribute through logical OR. At present, there are no specifiable attributes.
callbackFunc	Specify the entry address of the callback function
pCommon	Specify the argument to be passed to the callback function. It is passed to the callback function's *pCommon* without any changes.

Return Values

Value	Description
Positive value	Callback identifier
Negative value	Error code

Description

This creates a callback. The identifier (UID) of the created callback is returned as the return value.

sceKernelDeleteCallback

Delete callback

Definition

```
#include <kernel.h>
SceInt32 sceKernelDeleteCallback(
        SceUID callbackId
);
```

Arguments

callbackId Specify the identifier of the callback to be deleted.

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This deletes the callback specified with *callbackId*. A callback function can delete itself by returning a value of 1.

sceKernelNotifyCallback

Report callback

Definition

```
#include <kernel.h>
SceInt32 sceKernelNotifyCallback(
        SceUID callbackId,
        SceInt32 notifyArg
);
```

Arguments

callbackId	Specify the identifier of the callback to be reported
notifyArg	Specify the argument to be passed to the callback function.

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This reports the occurrence of the callback specified with *callbackId*.

The actual call of the associated callback function is delayed until either the thread that generated the callback confirms the callback notification by calling sceKernelCheckCallback() or until a wait function that checks the validity of the notification, such as sceKernelWaitSemaCB(), is called.

If the callback is reported multiple times before the callback function is called, the number of times it was reported up to the time that is called and the *notifyArg* argument provided by the last sceKernelNotifyCallback() are passed to the callback function.

sceKernelCancelCallback

Cancel callback notification

Definition

```
#include <kernel.h>
SceInt32 sceKernelCancelCallback(
        SceUID callbackId
);
```

Arguments

`callbackId` Specify the identifier of the callback for which notifications are to be canceled.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This cancels all notifications that were reported for the callback specified by *`callbackId`*.

sceKernelGetCallbackCount

Get callback notification count

Definition

```
#include <kernel.h>
SceInt32 sceKernelGetCallbackCount(
        SceUID callbackId
);
```

Arguments

callbackId Specify the identifier of the callback for which the f callback notification count is to be obtained.

Return Values

Value	Description
Positive value	Remaining callback notification count
Negative value	Error code

Description

This returns the number of times a callback notification has been delayed while the callback function still has not been called for the callback specified with *callbackId*.

sceKernelCheckCallback

Check existence of callback notification

Definition

```
#include <kernel.h>
SceInt32 sceKernelCheckCallback(
        void
);
```

Arguments

None

Return Values

Value	Description
Positive value	Callback notification count
Negative value	Error code

Description

This checks whether or not the occurrence of the callback has been reported for a callback created by the calling thread. If there is a callback the callback function registered for that callback is called. If no callback was reported, this function returns without doing anything.

The `sceKernelCheckCallback()` function cannot be called from within a callback function.

sceKernelGetCallbackInfo

Get callback status

Definition

```
#include <kernel.h>
SceInt32 sceKernelGetCallbackInfo(
        SceUID callbackId,
        SceKernelCallbackInfo *pInfo
);
```

Arguments

callbackId	Identifier of the callback whose status is to be obtained.
pInfo	Specify the pointer to the structure variable receiving callback status. Always assign `sizeof(SceKernelCallbackInfo)` to *pInfo*->*size* when calling.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This obtains the status of the callback specified with *callbackId*.

This system call is provided to aid debugging. Information that can be obtained by this system call changes moment by moment. Programming that issues this system call frequently, then changes the control flow according to the information that was received is not recommended.

sceKernelRegisterCallbackToEvent

Register callback to event

Definition

```
#include <kernel.h>
SceInt32 sceKernelRegisterCallbackToEvent(
        SceUID eventId,
        SceUID callbackId
);
```

Argument

eventId	Identifier of the event to which callback is to be registered
callbackId	Identifier of the callback to be registered to the event

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This registers a callback specified with *callbackId* to an event specified with *eventId*.

The callback is registered to a synchronization object (message pipe, timer etc.) implemented by the event. If the registered event satisfies callback notification conditions, the callback will be reported. If the callback is reported by the event, the identifier of the notifier event will be passed to *notifyId* of the callback function.

There is no limit to the number of callbacks that can simultaneously be registered to the same event.

sceKernelUnregisterCallbackFromEvent

Unregister callback from event

Definition

```
#include <kernel.h>
SceInt32 sceKernelUnregisterCallbackFromEvent(
        SceUID eventId,
        SceUID callbackId
);
```

Argument

eventId	Identifier of the event from which callback is to be unregistered
callbackId	Identifier of the callback to be unregistered from the event

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This unregisters a callback specified with *callbackId* from an event specified with *eventId*.

sceKernelUnregisterCallbackFromEventAll

Unregister all callbacks registered to event

Definition

```
#include <kernel.h>
SceInt32 sceKernelUnregisterCallbackFromEventAll(
        SceUID eventId
);
```

Argument

eventId Identifier of the event from which all callbacks are to be unregistered

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This unregisters all callbacks registered to an event specified with *eventId*.

Thread Events

SceKernelThreadEventHandler

Thread event handler type

Definition

```
#include <kernel.h>
typedef SceInt32 (*SceKernelThreadEventHandler)(
        SceInt32 type,
        SceUID threadId,
        SceInt32 arg,
        void *pCommon
);
```

Arguments

type	The cause of the call of the thread event handler is passed. - `SCE_KERNEL_THREAD_EVENT_TYPE_START`: the thread has started - `SCE_KERNEL_THREAD_EVENT_TYPE_EXIT`: the thread has exited
threadId	Identifier of the thread that has caused the call of the thread event handler is passed.
arg	Argument at the time of the call of the thread event handler is passed.
pCommon	*pCommon* specified with `sceKernelRegisterThreadEventHandler()` is passed without any changes.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Failure

Description

This is the thread event handler type.

Events

SceKernelWaitEvent

Structure for event wait condition list

Definition

```
#include <kernel.h>
typedef struct _SceKernelWaitEvent {
        SceUID eventId;
        SceUInt32 eventPattern;
} SceKernelWaitEvent;
```

Members

eventId	Event object identifier
eventPattern	Pattern of the waiting event

Description

This structure is for the event wait condition list passed to `sceKernelWaitMultipleEvents()`.

SceKernelResultEvent

Structure for get event wait result list

Definition

```
#include <kernel.h>
typedef struct _SceKernelResultEvent {
        SceUID eventId;
        SceInt32 result;
        SceUInt32 resultPattern;
        SceInt32 reserved[1];
        SceUInt64 userData;
} SceKernelResultEvent;
```

Members

eventId	Event object identifier
result	Waiting result (equivalent to the return value of `sceKernelWaitEvent()`)
resultPattern	Event pattern at the time of wait completion
reserved	Reserved area
userData	User data received at the time of wait completion

Description

This structure is for the get event wait result list passed to `sceKernelWaitMultipleEvents()`.

SceKernelEventInfo

Event object status

Definition

```
#include <kernel.h>
typedef struct _SceKernelEventInfo {
        SceSize size;
        SceUID eventId;
        char name[SCE_UID_NAMELEN+1];
        SceUInt32 attr;
        SceUInt32 eventPattern;
        SceUInt64 userData;
        SceUInt32 numWaitThreads;
        SceInt32 reserved[1];
} SceKernelEventInfo;
```

Members

size	Size of this structure. (Value of `sizeof(SceKernelEventInfo)`)
eventId	Event object identifier
name	Name specified when the event object is created
attr	Attribute specified when the event object is created
eventPattern	Current event's bit pattern
userData	User data attached to the event that was last notified
numWaitThreads	Number of threads waiting for the event object
reserved	Reserved area

Description

This is a structure used to obtain an event object's state with `sceKernelGetEventInfo()`.

sceKernelWaitEvent

Wait for event

Definition

```
#include <kernel.h>
SceInt32 sceKernelWaitEvent(
        SceUID eventId,
        SceUInt32 waitPattern,
        SceUInt32 *pResultPattern,
        SceUInt64 *pUserData,
        SceUInt32 *pTimeout
);
```

Arguments

eventId	Specify the identifier of waiting event object.
waitPattern	Specify the bit pattern indicating waiting event.
pResultPattern	Specify the pointer to the `SceUInt32` type variable receiving the bit pattern of the event when the wait condition is satisfied. Bit pattern will not be received if NULL is specified.
pUserData	Specify the pointer receiving optional data passed at the same time as event notification.
pTimeout	Specify the pointer to the `SceUInt32` type variable storing maximum waiting time (in microseconds). If NULL is specified waiting time will be infinite. When the wait condition is satisfied, the remaining time will return.

Return Values

Returns `SCE_OK` upon success.

Description

This system call waits for an event specified with *waitPattern* to be reported to an event object specified with *eventId*. If the logical AND of the event object specified with *eventId* with the event (wait condition) already specified with *waitPattern* is not 0, the issuing thread will continue without transition to WAITING state.

If the logical AND with the wait condition specified with *waitPattern* is 0, the issuing thread will change to WAITING state, and wait until an event satisfying the wait condition is reported.

The bit pattern of the event reported when the wait condition was met is returned to *pResultPattern*. It is possible to specify NULL if this value is not necessary. If exiting due to causes other than satisfaction of the wait condition, such as errors, the value indicated by *pResultPattern* will not be updated.

Specify *pTimeout* to set timeout operation (specified in microseconds). If *pTimeout* is NULL, timeout will not be performed. If *pTimeout* is specified, note that the value indicated by *pTimeout* will be updated when the system call terminates. If the condition is satisfied within the time limit, the value will be updated to the remaining time. If the condition is not satisfied, the value of *pTimeout* will be updated to 0, and the error `SCE_KERNEL_ERROR_WAIT_TIMEOUT` will return.

In case of an event with the `SCE_KERNEL_ATTR_TH_FIFO` attribute, threads will be placed in the wait queue in the order in which they have entered waiting state, with the thread that has entered waiting state first at the top. In the case of an event with the `SCE_KERNEL_ATTR_TH_PRIO` attribute, threads will be placed in the wait queue in accordance with their priority, with the thread with the highest priority at the top.

Based on event object attributes and notification type of reported events, "thread awakening rules at the time of event notification", "event object status after event notification" and "event object status after wait condition is satisfied" will present the following differences:

- When the event object attribute is `SCE_KERNEL_EVENT_ATTR_MANUAL_RESET`
  - Set notification of event
    All threads in waiting state are awakened by the reported event. The reported event is stored in notification state.
  - Pulse notification of event
    All threads in waiting state are awakened by the reported event. All reported events are cleared after thread awakening.
  - Satisfaction of event wait condition
    Events will be stored after wait completion.
- When the event object attribute is `SCE_KERNEL_EVENT_ATTR_AUTO_RESET`
  - Set notification of event
    One thread will be awakened by each event starting from the top of the thread wait queue. Events that have satisfied the wait condition and have awakened a thread will not be stored.
    If multiple events simultaneously satisfy the wait condition for a single waiting thread, all bits satisfying the condition at that point in time will be cleared.
    The events among those reported that have not awakened a thread will be stored until they either satisfy the wait condition at a later time or are overwritten by other notifications or clearing.
  - Pulse notification of event
    One thread will be awakened by each event starting from the top of the thread wait queue.
    If multiple events simultaneously satisfy the wait condition for a single waiting thread, all bits satisfying the condition at that point in time will be cleared.
    All reported events will be cleared after thread awakening.
  - Satisfaction of event wait condition
    Only the events for which the wait is completed will be cleared, and the thread will come out of the WAITING state.

Note that level-triggered events are unrelated to the attributes of event objects, and will always remain in notification state as long as the conditions for notification are satisfied.

sceKernelWaitEventCB

Wait for event (with callback check feature)

Definition

```
#include <kernel.h>
SceInt32 sceKernelWaitEventCB(
        SceUID eventId,
        SceUInt32 waitPattern,
        SceUInt32 *pResultPattern,
        SceUInt64 *pUserData,
        SceUInt32 *pTimeout
);
```

Arguments

eventId	Specify the identifier of waiting event object
waitPattern	Specify the bit pattern indicating waiting event
pResultPattern	Specify the pointer to the `SceUInt32` type variable receiving the bit pattern of the event reported when the wait condition is satisfied. Bit pattern will not be received if NULL is specified.
pUserData	Specify the pointer receiving optional data passed at the same time as event notification
pTimeout	Specify the pointer to the `SceUInt32` type variable storing maximum waiting time (in microseconds). If NULL is specified waiting time will be infinite. When the wait condition is satisfied, the remaining time will return.

Return Values

Returns `SCE_OK` upon success.

Description

This system call waits for an event specified with *waitPattern* to be reported to an event object specified with *eventId*. If the logical AND of the event object specified with *eventId* with the event (wait condition) already specified with *waitPattern* is not 0, the issuing thread will continue without transition to WAITING state.

If the logical AND with the wait condition specified with *waitPattern* is 0, the issuing thread will change to WAITING state, and wait until an event satisfying the wait condition is reported.

The bit pattern of the event reported when the wait condition was met is returned to *pResultPattern*. It is possible to specify NULL if this value is not necessary. If exiting due to causes other than satisfaction of the wait condition, such as errors, the value indicated by *pResultPattern* will not be updated.

Specify *pTimeout* to set timeout operation (specified in microseconds). If *pTimeout* is NULL, timeout will not be performed. If *pTimeout* is specified, note that the value indicated by *pTimeout* will be updated when the system call terminates. If the condition is satisfied within the time limit, the value will be updated to the remaining time. If the condition is not satisfied, the value of *pTimeout* will be updated to 0, and the error `SCE_KERNEL_ERROR_WAIT_TIMEOUT` will return.

In case of an event with the `SCE_KERNEL_ATTR_TH_FIFO` attribute, threads will be placed in the wait queue in the order in which they have entered waiting state, with the thread that has entered waiting state first at the top. In the case of an event with the `SCE_KERNEL_ATTR_TH_PRIO` attribute, threads will be placed in the wait queue in accordance with their priority, with the thread with the highest priority at the top.

Document serial number: 000004892117

000004892117

Based on event object attributes and notification type of reported events, “thread awakening rules at the time of event notification”, “event object status after event notification” and “event object status after event wait condition is satisfied” will present the following differences:

- When the event object attribute is `SCE_KERNEL_EVENT_ATTR_MANUAL_RESET`
  - Set notification of event
    All threads in waiting state are awakened by the reported event. The reported event is stored in notification state.
  - Pulse notification of event
    All threads in waiting state are awakened by the reported event. All reported events are cleared after thread awakening.
  - Satisfaction of event wait condition
    Events will be stored after wait completion.
- When the event object attribute is `SCE_KERNEL_EVENT_ATTR_AUTO_RESET`
  - Set notification of event
    One thread will be awakened by each event starting from the top of the thread wait queue. Events that have satisfied the wait condition and have awakened a thread will not be stored.
    If multiple events simultaneously satisfy the wait condition for a single waiting thread, all bits satisfying the condition at that point in time will be cleared.
    The events among those reported that have not awakened a thread will be stored until they either satisfy the wait condition at a later time or are overwritten by other notifications or clearing.
  - Pulse notification of event
    One thread will be awakened by each event starting from the top of the thread wait queue.
    If multiple events simultaneously satisfy the wait condition for a single waiting thread, all bits satisfying the condition at that point in time will be cleared.
    All reported events will be cleared after thread awakening.
  - Satisfaction of event wait condition
    Only the events for which the wait is completed will be cleared, and the thread will come out of the WAITING state.

Note that level-triggered events are unrelated to the attributes of event objects, and will always remain in notification state as long as the conditions for notification are satisfied.

`sceKernelWaitEventCB()` is a system call that adds a feature for checking for callback notifications to `sceKernelWaitEvent()`. Callback notifications will be checked immediately when this function is called, regardless of whether calling this function causes the thread to enter wait state. If the thread subsequently enters wait state, checks will continue while waiting also. If notification is performed for callback of the calling thread while waiting for time to elapse, the thread will momentarily exit wait state to execute the callback function before returning to wait state.

sceKernelPollEvent

Polling of event

Definition

```
#include <kernel.h>
SceInt32 sceKernelPollEvent(
        SceUID eventId,
        SceUInt32 bitPattern,
        SceUInt32 *pResultPattern,
        SceUInt64 *pUserData
);
```

Arguments

eventId	Specify the identifier of the event object to be polled
bitPattern	Specify the bit pattern indicating polling condition
pResultPattern	Specify the pointer to the `SceUInt32` type variable receiving the bit pattern of event when polling is successful
pUserData	Specify the pointer receiving optional passed data when polling is successful

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This performs polling of an event object specified with *eventId*.

`sceKernelPollEvent()` is a system call without `sceKernelWaitEvent()`'s feature for entering wait state. Unlike `sceKernelWaitEvent()`, When condition to cancel wait is not satisfied an error (`SCE_KERNEL_ERROR_EVENT_COND`) will immediately returned.

sceKernelWaitMultipleEvents

Wait for multiple events object simultaneously

Definition

```
#include <kernel.h>
SceInt32 sceKernelWaitMultipleEvents(
        SceKernelWaitEvent *pWaitEventList,
        SceUInt32 numEvents,
        SceUInt32 waitMode,
        SceKernelResultEvent *pResultEventList,
        SceUInt32 *pTimeout
);
```

Argument

pWaitEventList	Specify the pointer to the structure variable storing the list of event wait conditions. The list of event wait conditions is composed of pairs of identifiers of event objects and event patterns to wait for.
numEvents	Specify the number of event wait conditions in *pWaitEventList*.
waitMode	Specify waiting mode. Specify one of the following: - `SCE_KERNEL_EVENT_WAIT_MODE_OR`: waits until one of the wait conditions specified on the event wait condition list is satisfied - `SCE_KERNEL_EVENT_WAIT_MODE_AND`: waits until all of the wait conditions specified on the event wait condition list are satisfied
pResultEventList	Specify the pointer to the structure receiving detailed information about events of which waiting conditions are satisfied. Since data of a maximum size equivalent to `SceKernelResultEvent *` *numEvents* may be returned, it is necessary to have this buffer size ready. It is not possible to specify a location whose address overlaps with *pWaitEventList*. If NULL is specified, no information will be received.
pTimeout	Specify the pointer to the `SceUInt32` type variable storing maximum waiting time (in microseconds). If NULL is specified waiting time will be infinite. When the wait condition is satisfied, the remaining time will return.

Return Values

Value	Description
Positive value	The number of times returning results to *pResultEventList* (If NULL is specified to *pResultEventList*, `SCE_OK` returns.)
Negative value	Error code

Description

This is a system call for waiting simultaneously for a combination of multiple event objects and wait conditions specified with *pWaitEventList* in accordance with the wait mode specified with *waitMode*. If the wait canceling conditions specified with *waitMode* have already been satisfied, the issuing thread will continue without transition to WAITING state.

If `SCE_KERNEL_EVENT_WAIT_MODE_OR` has been specified in *waitMode*, the thread will wait until one of the wait conditions on the wait condition list is satisfied, or until an error occurs for one of the wait conditions. If `SCE_KERNEL_EVENT_WAIT_MODE_AND` has been specified in *waitMode*, the thread will wait until all of the wait conditions on the wait condition list are satisfied, or error for all of the wait conditions occur.

If the thread awakens due to satisfaction of the wait condition, the return value will be the number of waits for which conditions have been satisfied or that have resulted in an error. At this time, the same quantity of detailed information on each wait result as the return values to *pResultEventList* will be stored. It is possible to specify NULL if detailed information is not necessary.

Specify *pTimeout* to set timeout operation (specified in microseconds). If *pTimeout* is NULL, timeout will not be performed. If *pTimeout* is specified, note that the value indicated by *pTimeout* will be updated when the system call terminates. If the condition is satisfied within the time limit, the value will be updated to the remaining time. If the condition is not satisfied, the value of *pTimeout* will be updated to 0, and the error `SCE_KERNEL_ERROR_WAIT_TIMEOUT` will return.

Document serial number: 000004892117

sceKernelWaitMultipleEventsCB

Wait for multiple event object simultaneously (with callback check feature)

Definition

```
#include <kernel.h>
SceInt32 sceKernelWaitMultipleEventsCB(
        SceKernelWaitEvent *pWaitEventList,
        SceUInt32 numEvents,
        SceUInt32 waitMode,
        SceKernelResultEvent *pResultEventList,
        SceUInt32 *pTimeout
);
```

Argument

pWaitEventList	Specify the pointer to structure variable storing the list of event wait conditions. The list of event wait conditions is composed of pairs of identifiers of event objects and event patterns to wait for.
numEvents	Specify the number of event wait conditions in *pWaitEventList*.
waitMode	Specify waiting mode. Specify either of the following: - `SCE_KERNEL_EVENT_WAIT_MODE_OR`: waits until one of the wait conditions specified on the event wait condition list is satisfied - `SCE_KERNEL_EVENT_WAIT_MODE_AND`: waits until all of the wait conditions specified on the event wait condition list are satisfied
pResultEventList	Specify the pointer to the structure receiving detailed information about events of which waiting conditions are satisfied. Since data of a maximum size equivalent to `SceKernelResultEvent` * *numEvents* may be returned, it is necessary to have this buffer size ready. It is not possible to specify a location whose address overlaps with *pWaitEventList*. If NULL is specified, no information will be received.
pTimeout	Specify the pointer to the `SceUInt32` type variable storing maximum waiting time (in microseconds). If NULL is specified waiting time will be infinite. When the wait condition is satisfied, the remaining time will return.

Return Values

Value	Description
Positive value	The number of times returning results to *pResultEventList* (If NULL is specified to *pResultEventList*, `SCE_OK` returns.)
Negative value	Error code

Document serial number: 000004892117

Description

This is a system call for waiting simultaneously for a combination of multiple event objects and wait conditions specified with *pWaitEventList* in accordance with the wait mode specified with *waitMode*. If the wait canceling conditions specified with *waitMode* have already been satisfied, the issuing thread will continue without transition to WAITING state.

If SCE_KERNEL_EVENT_WAIT_MODE_OR has been specified in *waitMode*, the thread will wait until one of the wait conditions on the wait condition list is satisfied, or until an error occurs for one of the wait conditions. If SCE_KERNEL_EVENT_WAIT_MODE_AND has been specified in *waitMode*, the thread will wait until all of the wait conditions on the wait condition list are satisfied, or error for all of the wait conditions occur.

If the thread awakens due to satisfaction of the wait condition, the return value will be the number of waits for which conditions have been satisfied or that have resulted in an error. At this time, the same quantity of detailed information on each wait result as the return values to *pResultEventList* will be stored. It is possible to specify NULL if detailed information is not necessary.

Specify *pTimeout* to set timeout operation (specified in microseconds). If *pTimeout* is NULL, timeout will not be performed. If *pTimeout* is specified, note that the value indicated by *pTimeout* will be updated when the system call terminates. If the condition is satisfied within the time limit, the value will be updated to the remaining time. If the condition is not satisfied, the value of *pTimeout* will be updated to 0, and the error SCE_KERNEL_ERROR_WAIT_TIMEOUT will return.

sceKernelWaitMultipleEventsCB() is a system call that adds a feature for checking for callback notifications to sceKernelWaitMultipleEvents(). Callback notifications will be checked immediately when this function is called, regardless of whether calling this function causes the thread to enter wait state. If the thread subsequently enters wait state, checks will continue while waiting also. If notification is performed for callback of the calling thread while waiting for time to elapse, the thread will momentarily exit wait state to execute the callback function before returning to wait state.

sceKernelSetEvent

Perform set notification of event

Definition

```
#include <threadmgr.h>
SceInt32 sceKernelSetEvent(
        SceUID eventId,
        SceUInt32 setPattern,
        SceUInt64 userData
);
```

Arguments

eventId	Specify the identifier of the event object for which event notification is to be done
setPattern	Specify the bit pattern of the event to be set
userData	Specify the arbitrary data to be passed to the event object

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This performs set notification of the event specified in *setPattern* to the event object specified with *eventId*. The arbitrary data specified with *userData* is passed to the event object. The threads waiting for events that meet the wait conditions are awakened by event notification.

If the attribute of the event object is SCE_KERNEL_EVENT_ATTR_MANUAL_RESET, all the threads waiting for the bits (event) included in *setPattern* are awakened. The bits included in *setPattern* are all held even after return from this system call.

If the attribute of the event object is SCE_KERNEL_EVENT_ATTR_AUTO_RESET, the threads waiting for the bits (events) included in *setPattern* are awakened 1 for each bit. Of the bits included in *setPattern*, the bits that awakened a tread upon wait completion are cleared and are not held following return from this system call. Of the bits included in *setPattern*, the bits that did not awaken a thread are held until either wait completion occurs thereafter or they are overwritten with sceKernelPulseEvent()/sceKernelClearEvent().

The events that can be notified from the user mode are only those in the range of the user definition event (SCE_KERNEL_EVENT_USER_DEFINED_MASK).

sceKernelSetEventWithNotifyCallback

Perform set notification of event and callback notification

Definition

```
#include <threadmgr.h>
SceInt32 sceKernelSetEventWithNotifyCallback(
        SceUID eventId,
        SceUInt32 setPattern,
        SceUInt64 userData,
        SceInt32 notifyArg
);
```

Arguments

eventId	Specify the identifier of the event object for which event notification is to be done
setPattern	Specify the bit pattern of the event to be set
userData	Specify arbitrary data to be passed to event object
notifyArg	Specify *notifyArg* to be passed to callback function

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This performs set notification of the event specified in *setPattern* to the event object specified with *eventId*. At the same time, this performs notification of the callback registered to the corresponding event object. The arbitrary data specified with *userData* is also passed to the event object. The threads waiting for events that meet the wait conditions are awakened by event notification.

sceKernelSetEventWithNotifyCallback() is a system call that adds to sceKernelSetEvent() the notification feature of the callback registered to the event object.

If the attribute of the target event object is SCE_KERNEL_ATTR_NOTIFY_CB_ALL, all the registered callbacks are notified. If the attribute of the target event object is SCE_KERNEL_ATTR_NOTIFY_CB_WAKEUP_ONLY, of the registered callbacks, only the callbacks of the threads to be awakened through event notification are notified.

If callback is to be notified also to a thread awakened through event notification and that threads is waiting with a wait function with a callback notification check feature, the thread to be awakened ends the wait function after executing the callback function.

The events that can be notified from the user mode are only in the range of the user definition event (SCE_KERNEL_EVENT_USER_DEFINED_MASK).

sceKernelPulseEvent

Perform pulse notification of event

Definition

```
#include <threadmgr.h>
SceInt32 sceKernelPulseEvent(
        SceUID eventId,
        SceUInt32 pulsePattern,
        SceUInt64 userData
);
```

Arguments

eventId	Specify the identifier of the event object for which event notification is to be done
pulsePattern	Specify the bit pattern of the event to be set
userData	Specify the arbitrary data to be passed to the event object

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This performs pulse notification of the event specified with *pulsePattern* to the event object specified with *eventId*. The arbitrary data specified with *userData* is passed to the event object. The threads waiting for events that meet the wait conditions are awakened by event notification. The events that have been notified are cleared atomically after awakening thread.

If the attribute of the event object is SCE_KERNEL_EVENT_ATTR_MANUAL_RESET, all the threads waiting for the bits (events) included in *pulsePattern* are awakened. The bits included in *pulsePattern* are all cleared upon return from this system call.

If the attribute of the event object is SCE_KERNEL_EVENT_ATTR_AUTO_RESET, the threads waiting for the bits (events) included in *pulsePattern* are awakened 1 for each bit. The bits included in *pulsePattern* are all cleared upon return from this system call.

The events that can be notified from the user mode are only in the range of the user definition event (SCE_KERNEL_EVENT_USER_DEFINED_MASK).

sceKernelPulseEventWithNotifyCallback

Perform pulse notification of event and callback notification

Definition

```
#include <threadmgr.h>
SceInt32 sceKernelPulseEventWithNotifyCallback(
        SceUID eventId,
        SceUInt32 pulsePattern,
        SceUInt64 userData,
        SceInt32 notifyArg
);
```

Arguments

eventId	Specify the identifier of the event object for which event notification is to be done
pulsePattern	Specify the bit pattern of the event to be set
userData	Specify the arbitrary data to be passed to the event object
notifyArg	Specify *notifyArg* to be passed to the callback function

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This performs pulse notification of the event specified with *pulsePattern* to the event object specified with *eventId*. At the same time, this performs notification of the callback registered to the corresponding event object. The arbitrary data specified with *userData* is passed to the event object. The threads waiting for events that meet the wait conditions are awakened by event notification. The notified events are cleared atomically after awakening thread.

`sceKernelPulseEventWithNotifyCallback()` is a system call that adds to `sceKernelPulseEvent()` the notification feature of the callback registered to the event object.

If the attribute of the target event object is `SCE_KERNEL_ATTR_NOTIFY_CB_ALL`, all the registered callbacks are notified. If the attribute of the target event object is `SCE_KERNEL_ATTR_NOTIFY_CB_WAKEUP_ONLY`, of the registered callbacks, only the callbacks of the threads to be awakened through event notification are notified.

If callback is to be notified also to a thread awakened through event notification and that threads is waiting with a wait function with a callback notification check feature, the thread to be awakened ends the wait function after executing the callback function.

The events that can be notified from the user mode are only in the range of the user definition event (`SCE_KERNEL_EVENT_USER_DEFINED_MASK`).

sceKernelClearEvent

Clear event

Definition

```
#include <threadmgr.h>
SceInt32 sceKernelClearEvent(
        SceUID eventId,
        SceUInt32 clearPattern
);
```

Arguments

eventId	Specify the identifier of the event object for which the event is to be cleared
clearPattern	Clear the event corresponding to the bit whose value is 0 in *clearPattern*. In other words, the logical AND of the event value and *clearPattern* is used as the new event value.

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This clears the event of the event object specified with *eventId*.
This system call will not cancel the waiting state of threads that have been waiting for the event object.

The events that can be cleared from the user mode are only those in the range of the user definition event (SCE_KERNEL_EVENT_USER_DEFINED_MASK).

sceKernelCancelEvent

Cancel event

Definition

```
#include <kernel.h>
SceInt32 sceKernelCancelEvent(
        SceUID eventId,
        SceUInt32 *pNumWaitThreads
);
```

Arguments

eventId	Specify the identifier of the event object to be canceled
pNumWaitThreads	Specify the pointer to the `SceUInt32` type variable receiving the number of threads waiting for the event object at the time of cancelling The number of threads waiting for the event object will not be received if NULL is specified.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This cancels the threads waiting state for event notification by an event object specified with *eventId*. Canceled threads receive `SCE_KERNEL_ERROR_WAIT_CANCEL` with the return value of `sceKernelWaitEvent()`, and it is possible to determine that they have been canceled.

sceKernelCancelEventWithSetPattern

Cancel event and notify event

Definition

```
#include <threadmgr.h>
SceInt32 sceKernelCancelEventWithSetPattern(
        SceUID eventId,
        SceUInt32 setPattern,
        SceUInt64 userData,
        SceUInt32 *pNumWaitThreads
);
```

Arguments

eventId	Specify the identifier of the event object to be canceled
setPattern	Specify the event to be notified following cancellation
userData	Specify the arbitrary data to be passed to the event object following cancellation
pNumWaitThreads	Specify the pointer to the `SceUInt32` type variable receiving the number of threads waiting for the event object at the time of cancelling The number of threads waiting for the event object will not be received if NULL is specified.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This cancels the event object specified with *eventId* and then performs set notification of the event specified with *setPattern* atomically. All threads waiting for the event object are awakened.

Threads that have been awakened will receive `SCE_KERNEL_ERROR_WAIT_CANCEL` as the return value of `sceKernelWaitEvent()`, allowing determination that the waiting state has been canceled. Thereafter, set notification of the event value specified with *setPattern* to the event object is performed.

The events that can be notified from the user mode are only in the range of the user definition event (`SCE_KERNEL_EVENT_USER_DEFINED_MASK`).

Document serial number: 000004892117

sceKernelGetEventPattern

Get notification pattern of event

Definition

```
#include <threadmgr.h>
SceInt32 sceKernelGetEventPattern(
        SceUID eventId,
        SceUInt32 *pPattern
);
```

Arguments

eventId Specify the identifier of the event object for which to get the event bit pattern
pPattern Specify the pointer for receiving the event bit pattern

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This gets the event (bit pattern) notified to the event object specified with *eventId*.

sceKernelGetEventInfo

Get event object status

Definition

```
#include <kernel.h>
SceInt32 sceKernelGetEventInfo(
        SceUID eventId,
        SceKernelEventInfo *pInfo
);
```

Arguments

eventId Specify the identifier of the event object whose status is to be obtained.

pInfo Specify the pointer to the structure valuable receiving event object status. Always assign `sizeof(SceKernelEventInfo)` to *pInfo*->*size* when calling.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This obtains the status of event object specified with *eventId*.

This system call is provided to aid debugging. Information that can be obtained by this system call changes moment by moment. Programming that issues this system call frequently, then changes the control flow according to the information that was received is not recommended.

Event Flags

SceKernelEventFlagOptParam

Event flag optional data

Definition

```
#include <kernel.h>
typedef struct _SceKernelEventFlagOptParam {
        SceSize size;
} SceKernelEventFlagOptParam;
```

Members

size Size of this structure. (Value of `sizeof(SceKernelEventFlagOptParam)`)

Description

This structure is used with `sceKernelCreateEventFlag()` to store optional data that is provided when an event flag is created.

It is provided for future expansion.

SceKernelEventFlagInfo

Event flag status

Definition

```
#include <kernel.h>
typedef struct _SceKernelEventFlagInfo {
        SceSize size;
        SceUID evfId;
        char name[SCE_UID_NAMELEN+1];
        SceUInt32 attr;
        SceUInt32 initPattern;
        SceUInt32 currentPattern;
        SceUInt32 numWaitThreads;
} SceKernelEventFlagInfo;
```

Members

size	Size of this structure. (Value of `sizeof(SceKernelEventFlagInfo)`)
evfId	Identifier of event flag
name	Name of event flag specified with `sceKernelCreateEventFlag()`
attr	Attribute of event flag specified with `sceKernelCreateEventFlag()`
initPattern	Initial bit pattern of event flag
currentPattern	Current bit pattern of event flag
numWaitThreads	Number of threads waiting for event

Description

This structure is used to obtain the status of the event flag with `sceKernelGetEventFlagInfo()`.

sceKernelCreateEventFlag

Create event flag

Definition

```
#include <kernel.h>
SceUID sceKernelCreateEventFlag(
        const char *pName,
        SceUInt32 attr,
        SceUInt32 initPattern,
        const SceKernelEventFlagOptParam *pOptParam
);
```

Argument

pName	Specify the name of the event flag. Since the name of the event flag is only used for the purpose of identification by the operator during debugging when inter-process communication is not performed, it need not be unique. The name's maximum length is 31 bytes. NULL cannot be specified.
attr	Specify the attribute of the event flag. Specify one of the following as the order of the waiting queue: - `SCE_KERNEL_EVF_ATTR_TH_FIFO`: waiting threads are queued on a FIFO basis - `SCE_KERNEL_EVF_ATTR_TH_PRIO`: waiting threads are queued based on thread priority Specify whether it is possible for multiple threads to wait simultaneously with one of the following: - `SCE_KERNEL_EVF_ATTR_SINGLE`: Multiple threads cannot wait simultaneously - `SCE_KERNEL_EVF_ATTR_MULTI`: Multiple threads can wait simultaneously To create an event flag that can be referenced with `sceKernelOpenEventFlag()`, specify the following: - `SCE_KERNEL_ATTR_OPENABLE`
initPattern	Specify initial value of event flag
pOptParam	Argument for future expansion. Specify NULL.

Return Values

Value	Description
Positive value	Event flag identifier (UID)
Negative value	Error code

Description

This creates an event flag, and sets its initial value. The identifier of the event flag that is created is returned as a return value.

When specifying a name exceeding 31 bytes to *pName*, an error (`SCE_KERNEL_ERROR_UID_NAME_TOO_LONG`) will return when `SCE_KERNEL_ATTR_OPENABLE` is added to *attr*. An error will not occur if `SCE_KERNEL_ATTR_OPENABLE` is not added.

sceKernelDeleteEventFlag

Delete event flag

Definition

```
#include <kernel.h>
SceInt32 sceKernelDeleteEventFlag(
        SceUID evfId
);
```

Argument

`evfId` Specify the identifier of the event flag to be deleted.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This deletes the event flag specified with *evfId*, and created with `sceKernelCreateEventFlag()`. When `sceKernelDeleteEventFlag()` is executed, *evfId* will be invalidated and become unusable. However, if other CPUs are in the midst of processing in relation to the target event flag or references from other processes are being performed, actual object deletion will be delayed until referencing is over.

When the event flag is deleted, an error (`SCE_KERNEL_ERROR_WAIT_DELETE`) will return to the threads that were waiting for the target event flag.

The identifier obtained with `sceKernelOpenEventFlag()` cannot be specified to *evfId*.

When specified, `SCE_KERNEL_ERROR_UNKNOWN_EVF_ID` will return.

sceKernelOpenEventFlag

Reference event flag

Definition

```
#include <kernel.h>
SceUID sceKernelOpenEventFlag(
        const char *pName,
);
```

Arguments

pName Specify the event flag name to be referenced.

Return Values

Value	Description
Positive value	Event flag identifier
Negative value	Error code

Description

This references an event flag specified with *pName*. The new identifier of the event flag is returned as a return value. This system call is provided for the purpose of performing inter-process communication.

In order to reference an event flag, the `SCE_KERNEL_ATTR_OPENABLE` attribute must be specified as the *attr* argument of `sceKernelCreateEventFlag()` when creating the event flag. Note that it is necessary to define a unique name for an event flag within the system if specifying the attribute.

sceKernelCloseEventFlag

Close reference of event flag

Definition

```
#include <kernel.h>
SceInt32 sceKernelCloseEventFlag(
        SceUID evfId
);
```

Arguments

evfId Specify the identifier of the event flag whose reference is to be terminated.

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This terminates reference of the event flag specified with *evfId* and referenced with sceKernelOpenEventFlag(). This system call is provided for the purpose of inter-process communication.

When sceKernelCloseEventFlag() is executed, *evfId* is invalidated. Also, when there are no more references to the target event flag, the event flag will be deleted.

When the event flag is deleted, an error (SCE_KERNEL_ERROR_WAIT_DELETE) will return to the threads that had been waiting for the target event flag.

The identifier obtained with sceKernelCreateEventFlag() cannot be specified to *evfId*.

When specified, SCE_KERNEL_ERROR_UNKNOWN_EVF_ID will return.

sceKernelWaitEventFlag

Wait for event flag

Definition

```
#include <kernel.h>
SceInt32 sceKernelWaitEventFlag(
        SceUID evfId,
        SceUInt32 bitPattern,
        SceUInt32 waitMode,
        SceUInt32 *pResultPat,
        SceUInt32 *pTimeout
);
```

Arguments

evfId	Specify the identifier of the event flag to wait for.
bitPattern	Specify the value to be compared with event flag.
waitMode	Specify the wait mode. Specify one of the following: - `SCE_KERNEL_EVF_WAITMODE_AND`: AND wait - `SCE_KERNEL_EVF_WAITMODE_OR`: OR wait Also, as an option it is possible to add one of the following specifications through logical OR: - `SCE_KERNEL_EVF_WAITMODE_CLEAR_ALL`: all bits are cleared after wait condition is satisfied - `SCE_KERNEL_EVF_WAITMODE_CLEAR_PAT`: the bits specified in *bitPattern* are cleared after wait condition is satisfied
pResultPat	Specify the pointer to the `SceInt32` type variable receiving the value of the event flag when the wait condition is satisfied. Event flag value will not be received if NULL is specified.
pTimeout	Specify the pointer to the `SceUInt32` type variable storing maximum waiting time (in microseconds). If NULL is specified waiting time will be infinite. When the wait condition is satisfied, the remaining time will return

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This system call waits for the event flag specified with *evfId* to be set, in accordance with the wait canceling condition specified with *waitMode*. If the event flag specified with *evfId* already satisfies the wait canceling condition specified with *waitMode*, the issuing thread will continue without transition to WAITING mode.

Document serial number: 000004892117

If `SCE_KERNEL_EVF_WAITMODE_AND` is specified in *waitMode*, the thread will wait until all bits specified in *bitPattern* become 1. If `SCE_KERNEL_EVF_WAITMODE_OR` is specified in *waitMode*, the thread will wait until one of the bits specified in *bitPattern* becomes 1. If `SCE_KERNEL_EVF_WAITMODE_CLEAR_ALL` is additionally specified in *waitMode*, all bits of the event flag will become 0 when the wait canceling condition is satisfied and the thread waiting is canceled. If `SCE_KERNEL_EVF_WAITMODE_CLEAR_PAT` is additionally specified in *waitMode*, the bits specified in *bitPattern* will become 0 when the wait canceling condition is satisfied and the thread waiting is canceled.

Event flag values will return to *pResultPat* in one of the following cases. It is possible to specify NULL if this value is not necessary.

- If the wait canceling condition is satisfied, (that is, if `SCE_OK` is returned), immediately after the condition is satisfied, the event flag value before clearing with `SCE_KERNEL_EVF_WAITMODE_CLEAR_ALL` or `SCE_KERNEL_EVF_WAITMODE_CLEAR_PAT` will be returned.
- After the thread enters wait state, and if exiting for causes other than satisfaction of the wait canceling condition (that is, if the errors `SCE_KERNEL_ERROR_WAIT_CANCEL` or `SCE_KERNEL_ERROR_WAIT_DELETE` or `SCE_KERNEL_ERROR_WAIT_TIMEOUT` are returned), the event flag value immediately before exiting wait state will be returned. If cancelling has been performed with `sceKernelCancelEventFlag()`, the event flag value set at the time of cancelling will be returned.

Specify *pTimeout* to set timeout operation (specified in microseconds). If *pTimeout* is NULL, timeout will not be performed. If *pTimeout* is specified, note that the value indicated by *pTimeout* will be updated when the system call terminates. If the condition is satisfied within the time limit, the value will be updated to the remaining time. If the condition is not satisfied, the value of *pTimeout* will be updated to 0, and the error `SCE_KERNEL_ERROR_WAIT_TIMEOUT` will return.

A thread cannot execute `sceKernelWaitEventFlag()` or `sceKernelPollEventFlag()` for an event flag with the attribute `SCE_KERNEL_EVF_ATTR_SINGLE` that another thread is already waiting for. In this case, an error will immediately return to threads that subsequently execute `sceKernelWaitEventFlag()` or `sceKernelPollEventFlag()`.

If multiple threads enter wait state for an event flag with the attribute `SCE_KERNEL_EVF_ATTR_MULTI`, a thread waiting queue will be created. In this case, waiting state of multiple threads may be canceled by calling `sceKernelSetEventFlag()` once.

In case of an event flag with the `SCE_KERNEL_EVF_ATTR_TH_FIFO` attribute, threads will be placed in the wait queue in the order in which they have entered waiting state, with the thread that has entered waiting state first at the top. In the case of an event flag with the `SCE_KERNEL_EVF_ATTR_TH_PRIO` attribute, threads will be placed in the wait queue in accordance with their priority, with the thread with the highest priority at the top.

When `SCE_KERNEL_EVF_WAITMODE_CLEAR_ALL` or `SCE_KERNEL_EVF_WAITMODE_CLEAR_PAT` is specified for a thread during waiting, and the wait canceling condition for that thread is satisfied, the event flag will be cleared at the time of wait canceling. Waiting state of threads that were queued behind the thread for which `SCE_KERNEL_EVF_WAITMODE_CLEAR_ALL` or `SCE_KERNEL_EVF_WAITMODE_CLEAR_PAT` had been specified might not be canceled, since canceling waiting state of these threads depends on the event flag after clearing.

sceKernelWaitEventFlagCB

Wait for event flag (with callback check feature)

Definition

```
#include <kernel.h>
SceInt32 sceKernelWaitEventFlagCB(
        SceUID evfId,
        SceUInt32 bitPattern,
        SceUInt32 waitMode,
        SceUInt32 *pResultPat,
        SceUInt32 *pTimeout
);
```

Arguments

evfId	Specify the identifier of the event flag to wait for.
bitPattern	Specify the value to be compared with event flag.
waitMode	Specify the wait mode. Specify one of the following: - `SCE_KERNEL_EVF_WAITMODE_AND`: AND wait - `SCE_KERNEL_EVF_WAITMODE_OR`: OR wait Also, as an option it is possible to add one of the following specifications through logical OR: - `SCE_KERNEL_EVF_WAITMODE_CLEAR_ALL`: all bits are cleared after wait condition is satisfied - `SCE_KERNEL_EVF_WAITMODE_CLEAR_PAT`: the bits specified in *bitPattern* are cleared after wait condition is satisfied
pResultPat	Specify the pointer to the `SceInt32` type variable receiving the value of the event flag when the wait condition is satisfied. Event flag value will not be received if NULL is specified.
pTimeout	Specify the pointer to the `SceUInt32` type variable storing maximum waiting time (in microseconds). If NULL is specified waiting time will be infinite. When the wait condition is satisfied, the remaining time will return

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This system call waits for the event flag specified with *evfId* to be set, in accordance with the wait canceling condition specified with *waitMode*. If the event flag specified with *evfId* already satisfies the wait canceling condition specified with *waitMode*, the issuing thread will continue without transition to WAITING mode.

If SCE_KERNEL_EVF_WAITMODE_AND is specified in *waitMode*, the thread will wait until all bits specified in *bitPattern* become 1. If SCE_KERNEL_EVF_WAITMODE_OR is specified in *waitMode*, the thread will wait until one of the bits specified in *bitPattern* becomes 1. If SCE_KERNEL_EVF_WAITMODE_CLEAR_ALL is additionally specified in *waitMode*, all bits of the event flag will become 0 when the wait canceling condition is satisfied and the thread waiting is canceled. If SCE_KERNEL_EVF_WAITMODE_CLEAR_PAT is additionally specified in *waitMode*, the bits specified in *bitPattern* will become 0 when the wait canceling condition is satisfied and the thread waiting is canceled.

Event flag values will return to *pResultPat* in one of the following cases. It is possible to specify NULL if this value is not necessary.

- If the wait canceling condition is satisfied, (that is, if SCE_OK is returned), immediately after the condition is satisfied, the event flag value before clearing with SCE_KERNEL_EVF_WAITMODE_CLEAR_ALL or SCE_KERNEL_EVF_WAITMODE_CLEAR_PAT will be returned.
- After the thread enters wait state, and if exiting for causes other than satisfaction of the wait canceling condition (that is, if the errors SCE_KERNEL_ERROR_WAIT_CANCEL or SCE_KERNEL_ERROR_WAIT_DELETE or SCE_KERNEL_ERROR_WAIT_TIMEOUT are returned), the event flag value immediately before exiting wait state will be returned. If cancelling has been performed with sceKernelCancelEventFlag(), the event flag value set at the time of cancelling will be returned.

Specify *pTimeout* to set timeout operation (specified in microseconds). If *pTimeout* is NULL, timeout will not be performed. If *pTimeout* is specified, note that the value indicated by *pTimeout* will be updated when the system call terminates. If the condition is satisfied within the time limit, the value will be updated to the remaining time. If the condition is not satisfied, the value of *pTimeout* will be updated to 0, and the error SCE_KERNEL_ERROR_WAIT_TIMEOUT will return.

A thread cannot execute sceKernelWaitEventFlag() or sceKernelPollEventFlag() for an event flag with the attribute SCE_KERNEL_EVF_ATTR_SINGLE that another thread is already waiting for. In this case, an error will immediately return to threads that subsequently execute sceKernelWaitEventFlag() or sceKernelPollEventFlag().

If multiple threads enter wait state for an event flag with the attribute SCE_KERNEL_EVF_ATTR_MULTI, a thread waiting queue will be created. In this case, waiting state of multiple threads may be canceled by calling sceKernelSetEventFlag() once.

In case of an event flag with the SCE_KERNEL_EVF_ATTR_TH_FIFO attribute, threads will be placed in the wait queue in the order in which they have entered waiting state, with the thread that has entered waiting state first at the top. In the case of an event flag with the SCE_KERNEL_EVF_ATTR_TH_PRIO attribute, threads will be placed in the wait queue in accordance with their priority, with the thread with the highest priority at the top.

When SCE_KERNEL_EVF_WAITMODE_CLEAR_ALL or SCE_KERNEL_EVF_WAITMODE_CLEAR_PAT is specified for a thread during waiting, and the wait canceling condition for that thread is satisfied, the event flag will be cleared at the time of wait canceling. Waiting state of threads that were queued behind the thread for which SCE_KERNEL_EVF_WAITMODE_CLEAR_ALL or SCE_KERNEL_EVF_WAITMODE_CLEAR_PAT had been specified might not be canceled, since canceling waiting state of these threads depends on the event flag after clearing.

sceKernelWaitEventFlagCB() is a system call that adds a feature for checking for callback notifications to sceKernelWaitEventFlag(). Callback notifications will be checked immediately when this function is called, regardless of whether calling this function causes the thread to enter wait state. If the thread subsequently enters wait state, checks will continue while waiting also. If notification is performed for callback of the calling thread while waiting for time to elapse, the thread will momentarily exit wait state to execute the callback function before returning to wait state.

sceKernelPollEventFlag

Polling of event flag

Definition

```
#include <kernel.h>
SceInt32 sceKernelPollEventFlag(
        SceUID evfId,
        SceUInt32 bitPattern,
        SceUInt32 waitMode,
        SceUInt32 *pResultPat
);
```

Arguments

evfId	Specify the identifier of the event flag to be polled.
bitPattern	Specify the value to be compared with event flag.
waitMode	Specify the wait mode. Specify one of the following: - SCE_KERNEL_EVF_WAITMODE_AND: AND wait - SCE_KERNEL_EVF_WAITMODE_OR: OR wait Also, as an option it is possible to add one of the following specifications through logical OR: - SCE_KERNEL_EVF_WAITMODE_CLEAR_ALL: all bits are cleared after wait condition is satisfied - SCE_KERNEL_EVF_WAITMODE_CLEAR_PAT: the bits specified in *bitPattern* are cleared after wait condition is satisfied
pResultPat	Specify the pointer to the SceUInt32 type variable receiving the value of the event flag when the wait condition is satisfied.

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This performs polling of an event flag specified with *evfId*.

sceKernelPollEventFlag() is a system call without sceKernelWaitEventFlag()'s feature for entering wait state. Unlike sceKernelWaitEventFlag(), failure to satisfy the wait canceling condition will immediately result in an error (SCE_KERNEL_ERROR_EVF_COND). In this case, specification of SCE_KERNEL_EVF_WAITMODE_CLEAR_ALL / SCE_KERNEL_EVF_WAITMODE_CLEAR_PAT will be ignored.

sceKernelSetEventFlag

Set event flag

Definition

```
#include <kernel.h>
SceInt32 sceKernelSetEventFlag(
        SceUID evfId,
        SceUInt32 bitPattern
);
```

Arguments

evfId	Specify the identifier of the event flag to be set.
bitPattern	Specify the bit pattern to be set in the event flag value. In other words, the new event flag value will be the logical OR of the event flag value and *bitPattern*.

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This sets the bit of the event flag specified with *evfId*. The threads that had been waiting for the event flag will awaken due to satisfaction of event conditions by the new value of the event flag.

sceKernelClearEventFlag

Clear event flag

Definition

```
#include <kernel.h>
SceInt32 sceKernelClearEventFlag(
        SceUID evfId,
        SceUInt32 bitPattern
);
```

Arguments

evfId	Specify the identifier of the event flag to be cleared.
bitPattern	Clears the event flag value corresponding to the bits that are 0 in *bitPattern*. In other words, the new event flag value will be the logical AND of the event flag value and *bitPattern*.

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This clears the bit of the event flag specified with *evfId*. This system call will not cancel the waiting state of threads that been waiting for the event flag.

sceKernelCancelEventFlag

Cancel event flag

Definition

```
#include <kernel.h>
SceInt32 sceKernelCancelEventFlag(
        SceUID evfId,
        SceUInt32 setPattern,
        SceUInt32 *pNumWaitThreads
);
```

Arguments

evfId	Specify the identifier of the event flag to be canceled.
setPattern	Specify the event flag setting value.
pNumWaitThreads	Specify the pointer to the `SceUInt32` type variable receiving the number of threads of which wait state is canceled. The number of threads of which wait state is canceled will not be received if NULL is specified.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This function cancels waiting state of threads waiting for the event flag specified with *evfId*. Threads of which wait state is canceled will receive `SCE_KERNEL_ERROR_WAIT_CANCEL` as the return value of `sceKernelWaitEventFlag()`, and it is possible to determine that Waiting state of them has been canceled. Subsequently, the event flag value specified in *setPattern* will be set in the event flag.

sceKernelGetEventFlagInfo

Get event flag status

Definition

```
#include <kernel.h>
SceInt32 sceKernelGetEventFlagInfo(
        SceUID evfId,
        SceKernelEventFlagInfo *pInfo
);
```

Arguments

evfId Specify the identifier of the event flag whose status is to be obtained.

pInfo Specify the pointer to the structure valuable receiving event flag status. Always assign `sizeof(SceKernelEventFlagInfo)` to *pInfo*->*size* when calling.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This obtains the status of event flag specified with *evfId*.

This system call is provided to aid debugging. Information that can be obtained by this system call changes moment by moment. Programming that issues this system call frequently, then changes the control flow according to the information that was received is not recommended.

Semaphores

SceKernelSemaOptParam

Semaphore optional data

Definition

```
#include <kernel.h>
typedef struct _SceKernelSemaOptParam {
        SceSize size;
} SceKernelSemaOptParam;
```

Members

size Size of this structure. (Value of `sizeof(SceKernelSemaOptParam)`)

Description

This structure is used with `sceKernelCreateSema()` to store optional data that is provided when a semaphore is created.

It is provided for future expansion.

SceKernelSemaInfo

Semaphore status

Definition

```
#include <kernel.h>
typedef struct _SceKernelSemaInfo {
        SceSize size;
        SceUID semaId;
        char name[SCE_UID_NAMELEN+1];
        SceUInt32 attr;
        SceInt32 initCount;
        SceInt32 currentCount;
        SceInt32 maxCount;
        SceUInt32 numWaitThreads;
} SceKernelSemaInfo;
```

Members

size	Size of this structure. (Value of `sizeof(SceKernelSemaInfo)`)
semaId	Semaphore identifier
name	Semaphore name specified with `sceKernelCreateSema()`
attr	Semaphore attribute specified with `sceKernelCreateSema()`
initCount	Initial number of semaphore resources specified with `sceKernelCreateSema()`
currentCount	Current number of semaphore resources
maxCount	Maximum number of semaphore resources specified with `sceKernelCreateSema()`
numWaitThreads	Number of threads waiting for semaphore

Description

This structure is used to obtain the semaphore status with `sceKernelGetSemaInfo()`.

sceKernelCreateSema

Create semaphore

Definition

```
#include <kernel.h>
SceUID sceKernelCreateSema(
        const char *pName,
        SceUInt32 attr,
        SceInt32 initCount,
        SceInt32 maxCount,
        const SceKernelSemaOptParam *pOptParam
);
```

Argument

pName	Specify the name of the semaphore. Since the semaphore name is only used for the purpose of identification by the operator during debugging when inter-process communication is not performed, it need not be unique. The name's maximum length is 31 bytes. NULL cannot be specified.
attr	Specify the attribute of the semaphore. Specify one of the following as the order of the waiting queue: - `SCE_KERNEL_SEMA_ATTR_TH_FIFO`: waiting threads are queued on a FIFO basis - `SCE_KERNEL_SEMA_ATTR_TH_PRIO`: waiting threads are queued based on thread priority To create a semaphore that can be referenced with `sceKernelOpenSema()`, specify the following: - `SCE_KERNEL_ATTR_OPENABLE`
initCount	Specify the initial number of semaphore resources.
maxCount	Specify the maximum number of semaphore resources
pOptParam	Argument for future expansion. Specify NULL.

Return Values

Value	Description
Positive value	Semaphore identifier (UID)
Negative value	Error code

Description

This creates a semaphore and sets the initial number of semaphore resources. The identifier of the semaphore that is created will be returned as return value.

When specifying a name exceeding 31 bytes to *pName*, an error (`SCE_KERNEL_ERROR_UID_NAME_TOO_LONG`) will return when `SCE_KERNEL_ATTR_OPENABLE` is added to *attr*. An error will not occur if `SCE_KERNEL_ATTR_OPENABLE` is not added.

sceKernelDeleteSema

Delete semaphore

Definition

```
#include <kernel.h>
SceInt32 sceKernelDeleteSema(
        SceUID semaId
);
```

Argument

semaId Specify the identifier of the semaphore to be deleted.

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This deletes the semaphore specified with *semaId* and created with sceKernelCreateSema(). When sceKernelDeleteSema() is executed, *semaId* will be invalidated and become unusable. However, if other CPUs are in the midst of processing in relation to the target semaphore or references from other processes are being performed, actual object deletion will be delayed until referencing is over. When the semaphore is deleted, an error (SCE_KERNEL_ERROR_WAIT_DELETE) will return to the threads that were waiting for the target semaphore.

The identifier obtained with sceKernelOpenSema() cannot be specified to *semaId*.
When specified, SCE_KERNEL_ERROR_UNKNOWN_SEMA_ID will return.

sceKernelOpenSema

Reference semaphore

Definition

```
#include <kernel.h>
SceUID sceKernelOpenSema(
        const char *pName,
);
```

Arguments

pName Specify the name of the semaphore to be referenced.

Return Values

Value	Description
Positive value	Semaphore identifier
Negative value	Error code

Description

This references a semaphore specified with *pName*. The semaphore's new identifier is returned as a return value. This system call is provided for the purpose of performing inter-process communication.

In order to reference a semaphore, the SCE_KERNEL_ATTR_OPENABLE attribute must be specified as the *attr* argument of sceKernelCreateSema() when creating the semaphore. Note that it is necessary to define a unique name for a semaphore within the system if specifying the attribute.

sceKernelCloseSema

Close reference of semaphore

Definition

```
#include <kernel.h>
SceInt32 sceKernelCloseSema(
        SceUID semaId
);
```

Arguments

semaId Specify the identifier of the semaphore whose reference is to be terminated.

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This terminates reference of the semaphore specified with *semaId* and referenced with sceKernelOpenSema(). This system call is provided for the purpose of performing inter-process communication.

When sceKernelCloseSema() is executed, *semaId* is invalidated. Also, when there are no more references to the target semaphore, the semaphore will be deleted.

When the semaphore is deleted, an error (*SCE_KERNEL_ERROR_WAIT_DELETE*) will return to the threads that had been waiting the target semaphore.

The identifier obtained with sceKernelCreateSema() cannot be specified to *semaId*.

When specified, SCE_KERNEL_ERROR_UNKNOWN_SEMA_ID will return.

sceKernelWaitSema

Acquire semaphore resource

Definition

```
#include <kernel.h>
SceInt32 sceKernelWaitSema(
        SceUID semaId,
        SceInt32 needCount,
        SceUInt32 *pTimeout
);
```

Arguments

semaId	Specify the identifier of the semaphore whose resources are to be acquired.
needCount	Specify the number of resources to be acquired
pTimeout	Specify the pointer to the `SceUInt32` type variable storing maximum waiting time (in microseconds). If NULL is specified waiting time will be infinite. When the wait condition is satisfied, the remaining time will return.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This system call acquires the number of resources specified in *needCount* from the semaphore specified with *semaId*. If the number of requested resources specified with *needCount* is already present in the semaphore specified with *semaId*, the issuing thread will receive semaphore resources and continue execution without transition to WAITING state. If semaphore resources are below the requested number, the issuing thread will change to WAITING state and wait until the number of resources specified in *needCount* is returned to the semaphore.

Specify *pTimeout* to set timeout operation (specified in microseconds). If *pTimeout* is NULL, timeout will not be performed. If *pTimeout* is specified, note that the value indicated by *pTimeout* will be updated when the system call terminates. If the condition is satisfied within the time limit, the value will be updated to the remaining time. If the condition is not satisfied, the value of *pTimeout* will be updated to 0, and the error `SCE_KERNEL_ERROR_WAIT_TIMEOUT` will be returned.

If multiple threads enter wait state for a semaphore, a thread waiting queue will be created. In this case, waiting state of multiple threads may be canceled by calling `sceKernelSignalSema()` once.

In case of a semaphore with the `SCE_KERNEL_SEMA_ATTR_TH_FIFO` attribute, threads will be placed in the wait queue in the order in which they have entered waiting state, with the thread that has entered waiting state first at the top. In the case of a semaphore with the `SCE_KERNEL_SEMA_ATTR_TH_PRIO` attribute, threads will be placed in the wait queue in accordance with their priority, with the thread with the highest priority at the top.

When resources are returned with `sceKernelSignalSema()`, threads will acquire semaphore resources satisfying the requested number starting from the top of the waiting queue, and will awaken.

sceKernelWaitSemaCB

Acquire semaphore resource (with callback check feature)

Definition

```
#include <kernel.h>
SceInt32 sceKernelWaitSemaCB(
        SceUID semaId,
        SceInt32 needCount,
        SceUInt32 *pTimeout
);
```

Arguments

semaId	Specify the identifier of the semaphore whose resources are to be acquired.
needCount	Specify the number of resources to be acquired.
pTimeout	Specify the pointer to the `SceUInt32` type variable storing maximum waiting time (in microseconds). If NULL is specified waiting time will be infinite. When the wait condition is satisfied, the remaining time will return.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This system call acquires the number of resources specified in *needCount* from the semaphore specified with *semaId*. If the number of requested resources specified with *needCount* is already present in the semaphore specified with *semaId*, the issuing thread will receive semaphore resources and continue execution without transition to WAITING state. If semaphore resources are below the requested number, the issuing thread will change to WAITING state and wait until the number of resources specified in *needCount* is returned to the semaphore.

Specify *pTimeout* to set timeout operation (specified in microseconds). If *pTimeout* is NULL, timeout will not be performed. If *pTimeout* is specified, note that the value indicated by *pTimeout* will be updated when the system call terminates. If the condition is satisfied within the time limit, the value will be updated to the remaining time. If the condition is not satisfied, the value of *pTimeout* will be updated to 0, and the error `SCE_KERNEL_ERROR_WAIT_TIMEOUT` will be returned.

If multiple threads enter wait state for a semaphore, a thread waiting queue will be created. In this case, waiting state of multiple threads may be canceled by calling `sceKernelSignalSema()` once.

In case of a semaphore with the `SCE_KERNEL_SEMA_ATTR_TH_FIFO` attribute, threads will be placed in the wait queue in the order in which they have entered waiting state, with the thread that has entered waiting state first at the top. In the case of a semaphore with the `SCE_KERNEL_SEMA_ATTR_TH_PRIO` attribute, threads will be placed in the wait queue in accordance with their priority, with the thread with the highest priority at the top.

When resources are returned with `sceKernelSignalSema()`, threads will acquire semaphore resources satisfying the requested number starting from the top of the waiting queue, and will awaken.

Document serial number: 000004892117

Document serial number: 000004892117

`sceKernelWaitSemaCB()` is a system call that adds a feature for checking for callback notifications to `sceKernelWaitSema()`. Callback notifications will be checked immediately when this function is called, regardless of whether calling this function causes the thread to enter wait state. If the thread subsequently enters wait state, checks will continue while waiting also. If notification is performed for callback of the calling thread while waiting for time to elapse, the thread will momentarily exit wait state to execute the callback function before returning to wait state.

sceKernelPollSema

Polling of semaphore

Definition

```
#include <kernel.h>
SceInt32 sceKernelPollSema(
        SceUID semaId,
        SceInt32 needCount
);
```

Arguments

semaId	Specify the identifier of the semaphore to be polled
needCount	Specify the number of resources to be acquired

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This performs polling of a semaphore specified with *semaId*.

sceKernelPollSema() is a system call without sceKernelWaitSema()'s feature for entering wait state. Unlike sceKernelWaitSema(), failure to reach the number of resources specified in *needCount* for the target semaphore will immediately result in an error (SCE_KERNEL_ERROR_SEMA_ZERO).

sceKernelSignalSema

Free semaphore resource

Definition

```
#include <kernel.h>
SceInt32 sceKernelSignalSema(
        SceUID semaId,
        SceInt32 signalCount
);
```

Arguments

semaId	Specify the identifier of the semaphore to which resources are returned
signalCount	Specify the number of resources to be returned

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This function returns the number of resources specified in *signalCount* to the semaphore specified in *semaId*. However, if resources exceed the maximum value after returning, the number of resources will not change and an error will return (`SCE_KERNEL_ERROR_SEMA_OVF`).

Through the return of resources, threads will acquire semaphore resources satisfying the requested number starting from the top of the waiting queue, and will awaken.

The return of semaphore resources can also be performed from a context in which semaphore resources have not been acquired.

sceKernelCancelSema

Cancel semaphore

Definition

```
#include <kernel.h>
SceInt32 sceKernelCancelSema(
        SceUID semaId,
        SceInt32 setCount,
        SceUInt32 *pNumWaitThreads
);
```

Arguments

semaId	Specify the identifier of the semaphore to be canceled
setCount	Specify the number of semaphore resources after cancellation. If -1 is specified, the initial value at the time of semaphore creation will be used.
pNumWaitThreads	Specify the pointer to the `SceUInt32` type variable receiving the number of threads of which waiting state is canceled The number of threads of which wait state is canceled will not be received if NULL is specified.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This function cancels waiting state of the threads waiting for the semaphore specified with *semaId*. Threads of which waiting state is canceled will receive `SCE_KERNEL_ERROR_WAIT_CANCEL` as the return value of `sceKernelWaitSema()` and it will be possible to determine that waiting state of them have been canceled. Subsequently, the number of semaphore resources specified with *setCount* will be set for the semaphore.

Document serial number: 000004892117

sceKernelGetSemaInfo

Get semaphore status

Definition

```
#include <kernel.h>
SceInt32 sceKernelGetSemaInfo(
        SceUID semaId,
        SceKernelSemaInfo *pInfo
);
```

Arguments

semaId Specify the identifier of the semaphore whose status is to be obtained

pInfo Specify the pointer to the structured variable receiving semaphore status. Always assign `sizeof(SceKernelSemaInfo)` to *pInfo*->*size* when calling.

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This obtains the status of semaphore specified with *semaId*.

This system call is provided to aid debugging. Information that can be obtained by this system call changes moment by moment. Programming that issues this system call frequently, then changes the control flow according to the information that was received is not recommended.

Mutexes

SceKernelMutexOptParam

Mutex optional data

Definition

```
#include <kernel.h>
typedef struct _SceKernelMutexOptParam {
        SceSize size;
        SceInt32 ceilingPriority;
} SceKernelMutexOptParam;
```

Members

size	Size of this structure. (Value of `sizeof(SceKernelMutexOptParam)`)
ceilingPriority	Priority of priority ceiling

Description

This structure is used with `sceKernelCreateMutex()` to store optional data that is provided when a mutex is created.

It is provided for future expansion.

SceKernelMutexInfo

Mutex status

Definition

```
#include <kernel.h>
typedef struct _SceKernelMutexInfo {
        SceSize size;
        SceUID mutexId;
        char name[SCE_UID_NAMELEN+1];
        SceUInt32 attr;
        SceInt32 initCount;
        SceInt32 currentCount;
        SceUID currentOwnerId;
        SceUInt32 numWaitThreads;
        SceInt32 ceilingPriority;
} SceKernelMutexInfo;
```

Members

size	Size of this structure. (Value of `sizeof(SceKernelMutexInfo)`)
mutexId	Mutex identifier
name	Mutex name specified with `sceKernelCreateMutex()`
attr	Mutex attribute specified with `sceKernelCreateMutex()`
initCount	Initial lock count of mutex specified with `sceKernelCreateMutex()`
currentCount	Current lock count of mutex
currentOwnerId	Thread identifier currently owning mutex
numWaitThreads	Number of threads waiting for mutex
ceilingPriority	Priority of priority ceiling (0 if not using the priority ceiling feature)

Description

This structure is used to obtain the mutex status with `sceKernelGetMutexInfo()`.

sceKernelCreateMutex

Create mutex

Definition

```
#include <kernel.h>
SceUID sceKernelCreateMutex(
        const char *pName,
        SceUInt32 attr,
        SceInt32 initCount,
        const SceKernelMutexOptParam *pOptParam
);
```

Argument

pName	Specify the mutex name. Since the mutex name is only used for the purpose of identification by the operator during debugging when inter-process communication is not performed, it need not be unique. The name’s maximum length is 31 bytes. NULL cannot be specified.
attr	Specify the attribute of the mutex. Specify one of the following as the order of the waiting queue: - `SCE_KERNEL_MUTEX_ATTR_TH_FIFO`: waiting threads are queued on a FIFO basis - `SCE_KERNEL_MUTEX_ATTR_TH_PRIO`: waiting threads are queued based on priority As an option, it is possible to specify the following attributes through logical OR: - `SCE_KERNEL_MUTEX_ATTR_RECURSIVE`: allow recursive lock by threads that own the mutex - `SCE_KERNEL_MUTEX_ATTR_CEILING`: use the priority ceiling feature To create a mutex that can be referenced with `sceKernelOpenMutex()`, specify the following: - `SCE_KERNEL_ATTR_OPENABLE`
initCount	Specify initial mutex lock count. If `SCE_KERNEL_MUTEX_ATTR_RECURSIVE` is not specified for *attr*, the value that can be specified is 0 or 1.
pOptParam	Specify the pointer to the structure variable for specifying mutex options. Specify NULL if no option is to be specified. When specifying an option, always assign `sizeof(SceKernelMutexOptParam)` to *pOptParam->size* when calling. If using the priority ceiling feature, specify `SCE_KERNEL_MUTEX_ATTR_CEILING` as *attr* and, at the same time, specify ceiling priority in *pOptParam->ceilingPriority*.

Return Values

Value	Description
Positive value	Mutex identifier (UID)
Negative value	Error code

Description

This creates a mutex and sets the initial mutex lock count. The identifier of the mutex that has been created will be returned as a return value.

When specifying a name exceeding 31 bytes to *pName*, an error (SCE_KERNEL_ERROR_UID_NAME_TOO_LONG) will return when SCE_KERNEL_ATTR_OPENABLE is added to *attr*. An error will not occur if SCE_KERNEL_ATTR_OPENABLE is not added.

If *initCount* is set to 0, a created mutex will not belong to any thread. If *initCount* is set to a positive value, a created mutex will belong to a thread since its initial state and be locked as many times as specified in *initCount*.

If the attribute SCE_KERNEL_MUTEX_ATTR_CEILING is specified, the mutex will have a priority ceiling. If the priority of a thread with a priority ceiling mutex is lower than that of the priority ceiling, the thread's priority will be increased to that of the ceiling while the thread owns the mutex. However, it will not be possible to increase the priority of threads within the priority range of a common ready queue to the range of a individual ready queue.

sceKernelDeleteMutex

Delete mutex

Definition

```
#include <kernel.h>
SceInt32 sceKernelDeleteMutex(
        SceUID mutexId
);
```

Argument

mutexId Specify the identifier of the mutex to be deleted

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This deletes the mutex specified with *mutexId* and created with sceKernelCreateMutex().

When sceKernelDeleteMutex() is executed, *mutexId* will be invalidated and become unusable. However, if other CPUs are in the midst of processing in relation to the target mutex or references from other processes are being performed, actual object deletion will be delayed until referencing is over. When the mutex is deleted, an error (SCE_KERNEL_ERROR_WAIT_DELETE) will return to the threads that were waiting for the target mutex.

The identifier obtained with sceKernelOpenMutex() cannot be specified to *mutexId*.

When specified, SCE_KERNEL_ERROR_UNKNOWN_MUTEX_ID will return.

sceKernelOpenMutex

Reference mutex

Definition

```
#include <kernel.h>
SceUID sceKernelOpenMutex(
        const char *pName,
);
```

Arguments

pName Specify the name of the mutex to be referenced.

Return Values

Value	Description
Positive value	Mutex identifier
Negative value	Error code

Description

This references a mutex specified with *pName*. The new identifier of the mutex is returned as a return value. This system call is provided for the purpose of performing inter-process communication.

In order to reference a mutex, the SCE_KERNEL_ATTR_OPENABLE attribute must be specified as the *attr* argument of sceKernelCreateMutex() when creating the mutex. Note that it is necessary to define a unique name for a mutex within the system if specifying the attribute.

sceKernelCloseMutex

Close reference of mutex

Definition

```
#include <kernel.h>
SceInt32 sceKernelCloseMutex(
        SceUID mutexId
);
```

Arguments

`mutexId` Specify the identifier of the mutex whose reference is to be terminated.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This terminates reference of the mutex specified with *mutexId* and referenced with `sceKernelOpenMutex()`. This system call is provided for the purpose of performing inter-process communication.

When `sceKernelCloseMutex()` is executed, *mutexId* is invalidated. Also, when there are no more references to the target mutex, the mutex will be deleted.

When the mutex is deleted, an error (*SCE_KERNEL_ERROR_WAIT_DELETE*) will return to the threads that had been waiting for the target mutex.

The identifier obtained with `sceKernelCreateMutex()` cannot be specified to *mutexId*.

When specified, `SCE_KERNEL_ERROR_UNKNOWN_MUTEX_ID` will return.

sceKernelLockMutex

Own mutex

Definition

```
#include <kernel.h>
SceInt32 sceKernelLockMutex(
        SceUID mutexId,
        SceInt32 lockCount,
        SceUInt32 *pTimeout
);
```

Arguments

mutexId Specify the identifier of the mutex to be owned.

lockCount Specify the lock count after the mutex is owned.
In the case of a mutex with the attribute `SCE_KERNEL_MUTEX_ATTR_RECURSIVE`, specify a value of 1 or more. If the mutex does not have the attribute `SCE_KERNEL_MUTEX_ATTR_RECURSIVE`, specify 1.

pTimeout Specify the pointer to the `SceUInt32` type variable storing maximum waiting time (in microseconds).
If NULL is specified waiting time will be infinite. When the wait condition is satisfied, the remaining time will return.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This system call owns the mutex specified with *mutexId* and locks it for the number of times indicated in *lockCount*. If the mutex specified with *mutexId* does not belong to other threads, the issuing thread will owns the mutex and continue without transition to WAITING state.

If the mutex already belongs to another thread, the issuing thread will change to WAITING state and wait until it obtains the possession right.

Recursive lock (performed multiple times) by the owning thread is possible for mutexes with the `SCE_KERNEL_MUTEX_ATTR_RECURSIVE` attribute. Mutexes with recursive lock are released when `sceKernelUnlockMutex()` turns the lock count to 0.

When a mutex with the `SCE_KERNEL_MUTEX_ATTR_CEILING` attribute is owned, if the priority of the owning thread is lower than that of the priority ceiling specified in the mutex the thread's priority will be increased to that of the ceiling while the thread owns the mutex. Priority raised through the priority ceiling feature will return to normal when the mutex is released.

Specify *pTimeout* to set timeout operation (specified in microseconds). If *pTimeout* is NULL, timeout will not be performed. If *pTimeout* is specified, note that the value indicated by *pTimeout* will be updated when the system call terminates. If the condition is satisfied within the time limit, the value will be updated to the remaining time. If the condition is not satisfied, the value of *pTimeout* will be updated to 0, and the error `SCE_KERNEL_ERROR_WAIT_TIMEOUT` will return.

If multiple threads enter wait state for a mutex, a thread waiting queue will be created. In case of a mutex with the SCE_KERNEL_MUTEX_ATTR_TH_FIFO attribute, threads will be placed in the wait queue in the order in which they have entered waiting state, with the thread that has entered waiting state first at the top. In the case of a mutex with the SCE_KERNEL_MUTEX_ATTR_TH_PRIO attribute, threads will be placed in the wait queue in accordance with their priority, with the thread with the highest priority at the top.

When the mutex is released due to the execution of sceKernelUnlockMutex() by the thread that owns the mutex, or to the termination of that thread, the possession right will be passed to the thread at the top of the waiting queue, which will awaken.

sceKernelLockMutexCB

Own mutex (with callback check feature)

Definition

```
#include <kernel.h>
SceInt32 sceKernelLockMutexCB(
        SceUID mutexId,
        SceInt32 lockCount,
        SceUInt32 *pTimeout
);
```

Arguments

mutexId	Specify the identifier of the mutex to be owned.
lockCount	Specify the lock count after the mutex is owned. In the case of a mutex with the attribute `SCE_KERNEL_MUTEX_ATTR_RECURSIVE`, specify a value of 1 or more. If the mutex does not have the attribute `SCE_KERNEL_MUTEX_ATTR_RECURSIVE`, specify 1.
pTimeout	Specify the pointer to the `SceUInt32` type variable storing maximum waiting time (in microseconds). If NULL is specified waiting time will be infinite. When the wait condition is satisfied, the remaining time will return.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This system call owns the mutex specified with *mutexId* and locks it for the number of times indicated in *lockCount*. If the mutex specified with *mutexId* does not belong to other threads, the issuing thread will owns the mutex and continue without transition to WAITING state.

If the mutex already belongs to another thread, the issuing thread will change to WAITING state and wait until it obtains the possession right.

Recursive lock (performed multiple times) by the owning thread is possible for mutexes with the `SCE_KERNEL_MUTEX_ATTR_RECURSIVE` attribute. Mutexes with recursive lock are released when `sceKernelUnlockMutex()` turns the lock count to 0.

When a mutex with the `SCE_KERNEL_MUTEX_ATTR_CEILING` attribute is owned, if the priority of the owning thread is lower than that of the priority ceiling specified in the mutex the thread's priority will be increased to that of the ceiling while the thread owns the mutex. Priority raised through the priority ceiling feature will return to normal when the mutex is released.

Specify *pTimeout* to set timeout operation (specified in microseconds). If *pTimeout* is NULL, timeout will not be performed. If *pTimeout* is specified, note that the value indicated by *pTimeout* will be updated when the system call terminates. If the condition is satisfied within the time limit, the value will be updated to the remaining time. If the condition is not satisfied, the value of *pTimeout* will be updated to 0, and the error `SCE_KERNEL_ERROR_WAIT_TIMEOUT` will return.

If multiple threads enter wait state for a mutex, a thread waiting queue will be created. In case of a mutex with the `SCE_KERNEL_MUTEX_ATTR_TH_FIFO` attribute, threads will be placed in the wait queue in the order in which they have entered waiting state, with the thread that has entered waiting state first at the top. In the case of a mutex with the `SCE_KERNEL_MUTEX_ATTR_TH_PRIO` attribute, threads will be placed in the wait queue in accordance with their priority, with the thread with the highest priority at the top.

When the mutex is released due to the execution of `sceKernelUnlockMutex()` by the thread that owns the mutex, or to the termination of that thread, the possession right will be passed to the thread at the top of the waiting queue, which will awaken.

`sceKernelLockMutexCB()` is a system call that adds a feature for checking for callback notifications to `sceKernelLockMutex()`. Callback notifications will be checked immediately when this function is called, regardless of whether calling this function causes the thread to enter wait state. If the thread subsequently enters wait state, checks will continue while waiting also. If notification is performed for callback of the calling thread while waiting for time to elapse, the thread will momentarily exit wait state to execute the callback function before returning to wait state.

sceKernelTryLockMutex

Try to own mutex

Definition

```
#include <kernel.h>
SceInt32 sceKernelTryLockMutex(
        SceUID mutexId,
        SceInt32 lockCount
);
```

Arguments

mutexId	Specify the mutex whose possession is to be attempted.
lockCount	Specify the lock count after the mutex is owned. In the case of a mutex with the attribute SCE_KERNEL_MUTEX_ATTR_RECURSIVE, specify a value of 1 or more. If the mutex does not have the attribute SCE_KERNEL_MUTEX_ATTR_RECURSIVE, specify 1.

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This tries to possess the mutex specified with *mutexId*, and locks it for the number of times specified with *lockCount* if successful. sceKernelTryLockMutex() is a system call without sceKernelLockMutex()'s feature for entering wait state. Unlike sceKernelLockMutex(), if the target mutex already belongs to another thread an error (SCE_KERNEL_ERROR_MUTEX_FAILED_TO_OWN) will immediately return.

sceKernelUnlockMutex

Release owned mutex

Definition

```
#include <kernel.h>
SceInt32 sceKernelUnlockMutex(
        SceUID mutexId,
        SceInt32 unlockCount
);
```

Arguments

mutexId	Specify the identifier of the mutex to be released.
unlockCount	Specify the unlock count. In the case of a mutex with the attribute SCE_KERNEL_MUTEX_ATTR_RECURSIVE, specify a value of 1 or more. If the mutex does not have the attribute SCE_KERNEL_MUTEX_ATTR_RECURSIVE, specify 1.

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This function unlocks the mutex specified with *mutexId* for the number of times specified in *unlockCount*. When the lock count returns to 0, the mutex will be released. However, if the lock count becomes a negative value due to unlocking, the lock count will not change and an error (SCE_KERNEL_ERROR_MUTEX_UNLOCK_UDF) will return.

Through release, the possession right will be passed to the thread at the top of the waiting queue, which will awaken.

Only the thread that owns the target mutex can release the mutex.

sceKernelCancelMutex

Cancel mutex

Definition

```
#include <kernel.h>
SceInt32 sceKernelCancelMutex(
        SceUID mutexId,
        SceInt32 newCount,
        SceUInt32 *pNumWaitThreads
);
```

Arguments

mutexId	Specify the identifier of the mutex to be canceled.
newCount	Specify the number of times the mutex is locked after cancellation. The mutex will not be locked if 0 is specified. If -1 is specified, the initial lock count at the time of mutex creation will be used.
pNumWaitThreads	Specify the pointer to the `SceUInt32` type variable receiving the number of threads of which waiting state is canceled. The number of threads of which wait state is canceled will not be received if NULL is specified.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This function cancels threads waiting state waiting for the mutex specified with *mutexId*. Threads of which waiting state is canceled will receive `SCE_KERNEL_ERROR_WAIT_CANCEL` as the return value of `sceKernelLockMutex()`, and it is possible to determine that waiting state of them have been canceled.

After release, if *newCount* is 0 the target mutex will not be owned by any thread. If *newCount* is a value other than 0, the target mutex will be owned by the issuing thread and be locked for the number of times specified in *newCount*.

sceKernelGetMutexInfo

Get mutex status

Definition

```
#include <kernel.h>
SceInt32 sceKernelGetMutexInfo(
        SceUID mutexId,
        SceKernelMutexInfo *pInfo
);
```

Arguments

mutexId Specify the identifier of the mutex whose status is to be acquired.

pInfo Specify the pointer to the structure variable receiving mutex status. Always assign `sizeof(SceKernelMutexInfo)` to *pInfo*->*size* when calling.

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This obtains the status of mutex specified with *mutexId*.

This system call is provided to aid debugging. Information that can be obtained by this system call changes moment by moment. Programming that issues this system call frequently, then changes the control flow according to the information that was received is not recommended.

Lightweight Mutexes

SceKernelLwMutexWork

Lightweight mutex work area

Definition

```
#include <kernel.h>
typedef struct _SceKernelLwMutexWork {
        SceInt64 data[4];
} SceKernelLwMutexWork;
```

Members

data Work area of lightweight mutex

Description

This structure is used to store the work area used by the lightweight mutex. The work area of the lightweight mutex must be allocated in the user memory space. All operations on the lightweight mutex are performed through the `SceKernelLwMutexWork` structure.

Since the `SceKernelLwMutexWork` structure is used by the kernel as a work area, its contents must not be referenced/operated directly by user programs in the period from the creation of the lightweight mutex with `sceKernelCreateLwMutex()` to its deletion with `sceKernelDeleteLwMutex()`. Also, the address cannot be transferred.

The `SceKernelLwMutexWork` structure must be positioned with an 8-byte memory alignment.

SceKernelLwMutexOptParam

Lightweight mutex optional data

Definition

```
#include <kernel.h>
typedef struct _SceKernelLwMutexOptParam {
        SceSize size;
} SceKernelLwMutexOptParam;
```

Members

size Size of this structure. (Value of `sizeof(SceKernelLwMutexOptParam)`)

Description

This structure is used with `sceKernelCreateLwMutex()` to store optional data that is provided when a lightweight mutex is created.

It is provided for future expansion.

SceKernelLwMutexInfo

Lightweight mutex status

Definition

```
#include <kernel.h>
typedef struct _SceKernelLwMutexInfo {
        SceSize size;
        SceUID uid;
        char name[SCE_UID_NAMELEN+1];
        SceUInt32 attr;
        SceKernelLwMutexWork *pWork;
        SceInt32 initCount;
        SceInt32 currentCount;
        SceUID currentOwnerId;
        SceUInt32 numWaitThreads;
} SceKernelLwMutexInfo;
```

Members

size	Size of this structure. (Value of `sizeof(SceKernelLwMutexInfo)`)
uid	Identifier of lightweight mutex
name	Name of mutex specified with `sceKernelCreateLwMutex()`
attr	Attribute of mutex specified with `sceKernelCreateLwMutex()`
pWork	Work area of mutex specified with `sceKernelCreateLwMutex()`
initCount	Initial lock count of lightweight mutex specified with `sceKernelCreateLwMutex()`
currentCount	Current lock count of lightweight mutex
currentOwnerId	Thread identifier currently owning lightweight mutex
numWaitThreads	Number of threads waiting for lightweight mutex

Description

This structure is used to obtain the status of the lightweight mutex with `sceKernelGetLwMutexInfo()`.

sceKernelCreateLwMutex

Create lightweight mutex

Definition

```
#include <kernel.h>
SceInt32 sceKernelCreateLwMutex(
        SceKernelLwMutexWork *pWork,
        const char *pName,
        SceUInt32 attr,
        SceInt32 initCount,
        const SceKernelLwMutexOptParam *pOptParam
);
```

Arguments

pWork	Specify the address of the work area of the lightweight mutex to be created. *pWork* must be positioned with an 8-byte memory alignment.
pName	Specify the name of the lightweight mutex. Since the name of the lightweight mutex is only used for the purpose of identification by the operator during debugging, it need not be unique. The name's maximum length is 31 bytes. NULL cannot be specified.
attr	Specify the attribute of the lightweight mutex. Specify one of the following as the order of the waiting queue: - `SCE_KERNEL_LW_MUTEX_ATTR_TH_FIFO`: waiting threads are queued on a FIFO basis - `SCE_KERNEL_LW_MUTEX_ATTR_TH_PRIO`: waiting threads are queued based on thread priority As an option, it is possible to specify the following attribute through logical OR: - `SCE_KERNEL_LW_MUTEX_ATTR_RECURSIVE`: allow recursive lock by the thread that owns the lightweight mutex
initCount	Specify the initial lock count of the lightweight mutex. If `SCE_KERNEL_LW_MUTEX_ATTR_RECURSIVE` is not specified for *attr*, the value that can be specified is 0 or 1.
pOptParam	Argument for future expansion. Specify NULL.

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This creates a lightweight mutex and sets the initial lock count of the lightweight mutex.

If *initCount* is set to 0, the created lightweight mutex will not belong to any thread. If *initCount* is set to a positive value, a lightweight mutex will be created that belongs to a thread since its initial state and be locked as many times as specified in *initCount*.

sceKernelDeleteLwMutex

Delete lightweight mutex

Definition

```
#include <kernel.h>
SceInt32 sceKernelDeleteLwMutex(
        SceKernelLwMutexWork *pWork
);
```

Arguments

pWork Specify the address of the work area of the lightweight mutex to be deleted

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This deletes a lightweight mutex specified with *pWork*. When the lightweight mutex is deleted, an error (SCE_KERNEL_ERROR_WAIT_DELETE) will return to the threads that were waiting for the target lightweight mutex.

sceKernelLockLwMutex

Own lightweight mutex

Definition

```
#include <kernel.h>
SceInt32 sceKernelLockLwMutex(
        SceKernelLwMutexWork *pWork,
        SceInt32 lockCount,
        SceUInt32 *pTimeout
);
```

Arguments

pWork	Specify the address of the work area of the lightweight mutex to be owned.
lockCount	Specify the lock count after the lightweight mutex is owned. In the case of a mutex with the attribute `SCE_KERNEL_LW_MUTEX_ATTR_RECURSIVE`, specify a value of 1 or more. If the mutex does not have the attribute `SCE_KERNEL_LW_MUTEX_ATTR_RECURSIVE`, specify 1.
pTimeout	Specify the pointer to the `SceUInt32` type variable storing maximum waiting time (in microseconds). If NULL is specified waiting time will be infinite. When the wait condition is satisfied, the remaining time will return.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This system call owns the mutex lightweight mutex specified with *pWork* and locks it for the number of times indicated in *lockCount*. If the lightweight mutex specified with *pWork* does not belong to other threads, the issuing thread will own the lightweight mutex and continue without transition to WAITING state.

If the lightweight mutex already belongs to another thread, the issuing thread will change to WAITING state and wait until it obtains the possession right.

Recursive lock (performed multiple times) by the owning thread is possible for lightweight mutexes with the `SCE_KERNEL_MUTEX_ATTR_RECURSIVE` attribute. Lightweight mutexes with recursive lock are released when `sceKernelUnlockLwMutex()` turns the lock count to 0.

Specify *pTimeout* to set timeout operation (specified in microseconds). If *pTimeout* is NULL, timeout will not be performed. If *pTimeout* is specified, note that the value indicated by *pTimeout* will be updated when the system call terminates. If the condition is satisfied within the time limit, the value will be updated to the remaining time. If the condition is not satisfied, the value of *pTimeout* will be updated to 0, and the error `SCE_KERNEL_ERROR_WAIT_TIMEOUT` will return.

If multiple threads enter wait state for a lightweight mutex, a thread waiting queue will be created. In case of a lightweight mutex with the `SCE_KERNEL_LW_MUTEX_ATTR_TH_FIFO` attribute, threads will be placed in the wait queue in the order in which they have entered waiting state, with the thread that has entered waiting state first at the top. In the case of a lightweight mutex with the `SCE_KERNEL_LW_MUTEX_ATTR_TH_PRIO` attribute, threads will be placed in the wait queue in accordance with their priority, with the thread with the highest priority at the top.

When the lightweight mutex is released due to the execution of `sceKernelUnlockLwMutex()` by the thread that owns the lightweight mutex, or to the termination of that thread, the possession right will be passed to the thread at the top of the waiting queue, which will awaken.

Document serial number: 000004892117

Document serial number: 000004892117

sceKernelLockLwMutexCB

Own lightweight mutex (with callback check feature)

Definition

```
#include <kernel.h>
SceInt32 sceKernelLockLwMutexCB(
        SceKernelLwMutexWork *pWork,
        SceInt32 lockCount,
        SceUInt32 *pTimeout
);
```

Arguments

pWork Specify the address of the work area of the lightweight mutex to be owned.

lockCount Specify the lock count after the lightweight mutex is owned.
In the case of a mutex with the attribute SCE_KERNEL_LW_MUTEX_ATTR_RECURSIVE, specify a value of 1 or more. If the mutex does not have the attribute SCE_KERNEL_LW_MUTEX_ATTR_RECURSIVE, specify 1.

pTimeout Specify the pointer to the SceUInt32 type variable storing maximum waiting time (in microseconds).
If NULL is specified waiting time will be infinite. When the wait condition is satisfied, the remaining time will return.

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This system call owns the mutex lightweight mutex specified with *pWork* and locks it for the number of times indicated in *lockCount*. If the lightweight mutex specified with *pWork* does not belong to other threads, the issuing thread will own the lightweight mutex and continue without transition to WAITING state.

If the lightweight mutex already belongs to another thread, the issuing thread will change to WAITING state and wait until it obtains the possession right.

Recursive lock (performed multiple times) by the owning thread is possible for lightweight mutexes with the SCE_KERNEL_MUTEX_ATTR_RECURSIVE attribute. Lightweight mutexes with recursive lock are released when sceKernelUnlockLwMutex() turns the lock count to 0.

Specify *pTimeout* to set timeout operation (specified in microseconds). If *pTimeout* is NULL, timeout will not be performed. If *pTimeout* is specified, note that the value indicated by *pTimeout* will be updated when the system call terminates. If the condition is satisfied within the time limit, the value will be updated to the remaining time. If the condition is not satisfied, the value of *pTimeout* will be updated to 0, and the error SCE_KERNEL_ERROR_WAIT_TIMEOUT will return.

If multiple threads enter wait state for a lightweight mutex, a thread waiting queue will be created. In case of a lightweight mutex with the SCE_KERNEL_LW_MUTEX_ATTR_TH_FIFO attribute, threads will be placed in the wait queue in the order in which they have entered waiting state, with the thread that has entered waiting state first at the top. In the case of a lightweight mutex with the SCE_KERNEL_LW_MUTEX_ATTR_TH_PRIO attribute, threads will be placed in the wait queue in accordance with their priority, with the thread with the highest priority at the top.

Document serial number: 000004892117

Document serial number: 000004892117

When the lightweight mutex is released due to the execution of `sceKernelUnlockLwMutex()` by the thread that owns the lightweight mutex, or to the termination of that thread, the possession right will be passed to the thread at the top of the waiting queue, which will awaken.

`sceKernelLockLwMutexCB()` is a system call that adds a feature for checking for callback notifications to `sceKernelLockLwMutex()`. Callback notifications are checked during waiting state (they will not be checked for if lock is successful and the waiting state is not entered). If notification is performed for callback of the calling thread while waiting for time to elapse, the thread will momentarily exit wait state to execute the callback function before returning to wait state.

Document serial number: 000004892117

Document serial number: 000004892117

sceKernelTryLockLwMutex

Try to own lightweight mutex

Definition

```
#include <kernel.h>
SceInt32 sceKernelTryLockLwMutex(
        SceKernelLwMutexWork *pWork,
        SceInt32 lockCount
);
```

Arguments

pWork	Specify the address of the work area of the lightweight mutex whose possession is to be attempted.
lockCount	Specify the lock count after the lightweight mutex is owned. In the case of a lightweight mutex with the attribute `SCE_KERNEL_LW_MUTEX_ATTR_RECURSIVE`, specify a value of 1 or more. If the lightweight mutex does not have the attribute `SCE_KERNEL_LW_MUTEX_ATTR_RECURSIVE`, specify 1.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This tries to posses the lightweight mutex specified with *pWork*, and locks it for the number of times specified with *lockCount* if successful. `sceKernelTryLockLwMutex()` is a system call without `sceKernelLockLwMutex()`'s feature for entering wait state. Unlike `sceKernelLockLwMutex()`, if the target mutex already belongs to another thread an error (`SCE_KERNEL_ERROR_LW_MUTEX_FAILED_TO_OWN`) will immediately return.

sceKernelUnlockLwMutex

Release owned lightweight mutex

Definition

```
#include <kernel.h>
SceInt32 sceKernelUnlockLwMutex(
        SceKernelLwMutexWork *pWork,
        SceInt32 unlockCount
);
```

Arguments

pWork	Specify the address of the work area of the lightweight mutex to be released.
unlockCount	Specify the unlock count after the lightweight mutex is owned. In the case of a lightweight mutex with the attribute `SCE_KERNEL_LW_MUTEX_ATTR_RECURSIVE`, specify a value of 1 or more. If the lightweight mutex does not have the attribute `SCE_KERNEL_LW_MUTEX_ATTR_RECURSIVE`, specify 1.

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This function unlocks the lightweight mutex specified with *pWork* for the number of times specified in *unlockCount*. When the lock count returns to 0, the lightweight mutex will be released. However, if the lock count becomes a negative value due to unlocking, the lock count will not change and an error (`SCE_KERNEL_ERROR_LW_MUTEX_UNLOCK_UDF`) will return.

Through release, the possession right will be passed to the thread at the top of the waiting queue, which will awaken.

sceKernelGetLwMutexInfo

Get lightweight mutex status

Definition

```
#include <kernel.h>
SceInt32 sceKernelGetLwMutexInfo(
        SceKernelLwMutexWork *pWork,
        SceKernelLwMutexInfo *pInfo
);
```

Arguments

pWork Specify the address of the work area of the lightweight mutex whose status is to be acquired.
pInfo Specify the pointer to the structure variable receiving lightweight mutex status. Always assign `sizeof(SceKernelLwMutexInfo)` to *pInfo*->*size* when calling.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This obtains the status of a lightweight mutex specified with *pWork*.

This system call is provided to aid debugging. Information that can be obtained by this system call changes moment by moment. Programming that issues this system call frequently, then changes the control flow according to the information that was received is not recommended.

sceKernelGetLwMutexInfoById

Get lightweight mutex status by its identifier

Definition

```
#include <kernel.h>
SceInt32 sceKernelGetLwMutexInfoById(
        SceUID lwMutexId,
        SceKernelLwMutexInfo *pInfo
);
```

Arguments

lwMutexId	Specify the identifier of the lightweight mutex whose status is to be acquired.
pInfo	Specify the pointer to the structure variable receiving lightweight mutex status. Always assign `sizeof(SceKernelLwMutexInfo)` to *pInfo*->*size* when calling.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This obtains the status of a mutex specified with *lwMutexId*.

This system call is provided to aid debugging. Information that can be obtained by this system call changes moment by moment. Programming that issues this system call frequently, then changes the control flow according to the information that was received is not recommended.

Condition Variables

SceKernelCondOptParam

Condition variable optional data

Definition

```
#include <kernel.h>
typedef struct _SceKernelCondOptParam {
        SceSize size;
} SceKernelCondOptParam;
```

Members

`size` Size of this structure. (Value of `sizeof(SceKernelMutexOptParam)`)

Description

This structure is used to store optional data that is provided when a condition variable is created with `sceKernelCreateCond()`.

It is provided for future expansion.

SceKernelCondInfo

Condition variable status

Definition

```
#include <kernel.h>
typedef struct _SceKernelCondInfo {
        SceSize size;
        SceUID condId;
        char name[SCE_UID_NAMELEN+1];
        SceUInt32 attr;
        SceUID mutexId;
        SceUInt32 numWaitThreads;
} SceKernelCondInfo;
```

Members

size	Size of this structure. (Value of `sizeof(SceKernelCondInfo)`)
condId	Identifier of condition variable
name	Name of condition variable specified with `sceKernelCreateCond()`
attr	Attribute of condition variable specified with `sceKernelCreateCond()`
mutexId	Mutex associated with condition variable specified with `sceKernelCreateCond()`
numWaitThreads	Number of threads waiting at condition variable

Description

This structure is used to obtain the condition variable status with `sceKernelGetCondInfo()`.

sceKernelCreateCond

Create condition variable

Definition

```
#include <kernel.h>
SceUID sceKernelCreateCond(
        const char *pName,
        SceUInt32 attr,
        SceUID mutexId,
        const SceKernelCondOptParam *pOptParam
);
```

Arguments

pName	Specify the name of the condition variable. Since the name of the condition variable is only used for the purpose of identification by the operator during debugging when inter-process communication is not performed, it need not be unique. The name's maximum length is 31 bytes. NULL cannot be specified.
attr	Specify the attribute of the condition variable. Specify one of the following as the order of the waiting queue: - `SCE_KERNEL_COND_ATTR_TH_FIFO`: waiting threads are queued on a FIFO basis - `SCE_KERNEL_COND_ATTR_TH_PRIO`: waiting threads are queued based on thread priority
mutexId	Specify the identifier of the mutex to be related to the condition variable. The mutex related to the condition variable is released atomically with 1 unlock when the thread changes to WAITING state with the execution of `sceKernelWaitCond()` for this condition variable; and is owned with 1 lock when the thread is awakened from the WAITING state.
pOptParam	Argument for future expansion. Specify NULL.

Return Values

Value	Description
Positive value	Condition variable identifier (UID)
Negative value	Error code

Description

This creates a condition variable. The identifier of the condition variable that has been created is returned as a return value.

sceKernelDeleteCond

Delete condition variable

Definition

```
#include <kernel.h>
SceInt32 sceKernelDeleteCond(
        SceUID condId
);
```

Arguments

condId Specify the identifier of the condition variable to be deleted

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This deletes the condition variable specified with *condId* and created with sceKernelCreateCond().When sceKernelDeleteCond() is executed, *condId* will be invalidated and become unusable. However, if other CPUs are in the midst of processing in relation to the target condition variable or references from other processes are being performed, actual object deletion will be delayed until referencing is over.

When the condition variable is deleted, an error (undefined) will return to the threads that were waiting for the signal of the target condition variable. When the condition variable is deleted after receiving the signal of the condition variable, an error (undefined) will return to the threads that were waiting to own the related mutex.

Document serial number: 000004892117

sceKernelWaitCond

Wait for signal by condition variable

Definition

```
#include <kernel.h>
SceInt32 sceKernelWaitCond(
        SceUID condId,
        SceUInt32 *pTimeout
);
```

Arguments

condId Specify the identifier of the condition variable to wait for.

pTimeout Specify the pointer to the `SceUInt32` type variable storing maximum waiting time (in microseconds).
If NULL is specified waiting time will be infinite. When the wait condition is satisfied, the remaining time will return.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This system call waits for the signal of the condition variable specified with *condId*. The issuing thread changes to WAITING state. At the same time, the mutex related to the condition variable is released atomically with 1 unlock.

Specify *pTimeout* to set timeout operation (specified in microseconds). If *pTimeout* is NULL, timeout will not be performed. If *pTimeout* is specified, note that the value indicated by *pTimeout* will be updated when the system call terminates. If the condition is satisfied within the time limit, the value will be updated to the remaining time. If the condition is not satisfied, the value of *pTimeout* will be updated to 0, and the error `SCE_KERNEL_ERROR_WAIT_TIMEOUT` will return.

If multiple threads enter wait state for a condition variable, a thread waiting queue will be created. In case of a condition variable with the `SCE_KERNEL_COND_ATTR_TH_FIFO` attribute, threads will be placed in the wait queue in the order in which they have entered waiting state, with the thread that has entered waiting state first at the top. In the case of a condition variable with the `SCE_KERNEL_COND_ATTR_TH_PRIO` attribute, threads will be placed in the wait queue in accordance with their priority, with the thread with the highest priority at the top.

When a signal is reported to a waiting thread, the thread will awaken, and will own the mutex related to the condition variable with 1 lock. If the mutex cannot be owned immediately, the thread will wait to own the mutex.

If the mutex related to the condition variable has been canceled, the thread that had been waiting for the target condition variable will awaken and receive an error (undefined).

sceKernelSignalCond

Send signal to thread waiting for condition variable

Definition

```
#include <kernel.h>
SceInt32 sceKernelSignalCond(
        SceUID condId
);
```

Arguments

`condId` Specify the identifier of the condition variable for which the signal is to be sent.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This sends a signal to a thread waiting for a condition variable specified with *condId*. The thread at the top of the waiting queue for the target condition variable will receive the signal and awaken.

The signal sent will not be saved. In other words, if there are no threads waiting for the target condition variable when the signal is sent, this signal will be discarded. Note that threads entering wait state after the signal is sent cannot receive the signal.

sceKernelSignalCondAll

Send signal to all threads waiting for condition variable

Definition

```
#include <kernel.h>
SceInt32 sceKernelSignalCondAll(
        SceUID condId
);
```

Arguments

condId Specify the identifier of the condition variable for which the signal is to be sent.

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This sends a signal to all threads waiting for a condition variable specified with *condId*. All threads that had been waiting for the target condition variable when the signal is sent will receive the signal and awaken.

Note that threads entering wait state after the signal is sent cannot receive the signal.

sceKernelSignalCondTo

Send signal to specific thread waiting for condition variable

Definition

```
#include <kernel.h>
SceInt32 sceKernelSignalCondTo(
        SceUID condId,
        SceUID threadId
);
```

Arguments

condId Specify the identifier of the condition variable for which the signal is to be sent.
threadId Specify the identifier of the thread to which the signal is to be sent

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This sends a signal to a thread specified with *threadId* waiting for a condition variable specified with *condId*. If the target thread is waiting for the target condition variable, that thread will awaken.

If the target thread is not waiting for the target condition variable when the signal is sent, an error (undefined) will be returned. Note that if the thread enters wait state after the signal is sent it will not be able to receive the signal.

sceKernelGetCondInfo

Get condition variable status

Definition

```
#include <kernel.h>
SceInt32 sceKernelGetCondInfo(
        SceUID condId,
        SceKernelCondInfo *pInfo
);
```

Arguments

condId	Specify the identifier of the condition variable whose status is to be obtained
pInfo	Specify the pointer to the structure variable receiving condition variable status. Always assign `sizeof(SceKernelCondInfo)` to *pInfo*->*size* when calling.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This obtains the status of a condition variable specified with *condId*.

This system call is provided to aid debugging. Information that can be obtained by this system call changes moment by moment. Programming that issues this system call frequently, then changes the control flow according to the information that was received is not recommended.

Lightweight Condition Variables

SceKernelLwCondWork

Lightweight condition variable work area

Definition

```
#include <kernel.h>
typedef struct _SceKernelLwCondWork {
        SceInt32 data[4];
} SceKernelLwCondWork;
```

Members

`data` Work area of lightweight condition variables

Description

This structure is used to store the work area used by the lightweight condition variable. The work area of the lightweight condition variable must be allocated in the user memory space. All operations on the lightweight condition variable are performed through the `SceKernelLwCondWork` structure.

Since the `SceKernelLwCondWork` structure is used by the kernel as a work area, its contents must not be referenced/operated directly by user programs in the period from the creation of the lightweight condition variable with `sceKernelCreateLwCond()` to its deletion with `sceKernelDeleteLwCond()`. Also, the address cannot be transferred.

SceKernelLwCondOptParam

Lightweight condition variable optional data

Definition

```
#include <kernel.h>
typedef struct _SceKernelLwCondOptParam {
        SceSize size;
} SceKernelLwCondOptParam;
```

Members

size Size of this structure. (Value of `sizeof(SceKernelLwCondOptParam)`)

Description

This structure is used to store optional data that is provided when a condition variable is created with `sceKernelCreateCond()`.

It is provided for future expansion.

Document serial number: 000004892117

SceKernelLwCondInfo

Lightweight condition variable status

Definition

```
#include <kernel.h>
typedef struct _SceKernelLwCondInfo {
        SceSize size;
        SceUID uid;
        char name[SCE_UID_NAMELEN+1];
        SceUInt32 attr;
        SceKernelLwCondWork *pWork;
        SceKernelLwMutexWork *pLwMutex;
        SceUInt32 numWaitThreads;
} SceKernelLwCondInfo;
```

Members

size	Size of this structure. (Value of `sizeof(SceKernelLwCondInfo)`)
uid	Identifier of lightweight condition variable
name	Name of lightweight condition variable specified with `sceKernelCreateLwCond()`
attr	Attribute of lightweight condition variable specified with `sceKernelCreateLwCond()`
pWork	Work area of lightweight condition variable specified with `sceKernelCreateLwCond()`
pLwMutex	Work area of lightweight mutex associated with condition variable specified with `sceKernelCreateLwCond()`
numWaitThreads	Number of threads waiting at lightweight condition variable

Description

This structure is used to obtain the status of the lightweight condition variable with `sceKernelGetLwCondInfo()`.

sceKernelCreateLwCond

Create lightweight condition variable

Definition

```
#include <kernel.h>
SceInt32 sceKernelCreateLwCond(
        SceKernelLwCondWork *pWork,
        const char *pName,
        SceUInt32 attr,
        SceKernelLwMutexWork *pLwMutex,
        const SceKernelLwCondOptParam *pOptParam
);
```

Argument

pWork	Specify the address of the work area of the lightweight condition variable to be created.
pName	Specify the name of the lightweight condition variable. Since the name of the lightweight condition variable is only used for the purpose of identification by the operator during debugging, it need not be unique. The name's maximum length is 31 bytes. NULL cannot be specified.
attr	Specify the attribute of the lightweight condition variable. Specify for the order of the waiting queue: - `SCE_KERNEL_LW_COND_ATTR_TH_FIFO`: waiting threads are queued on a FIFO basis - `SCE_KERNEL_LW_COND_ATTR_TH_PRIO`: waiting threads are queued based on thread priority
pLwMutex	Specify the identifier of the lightweight mutex to be related to the lightweight condition variable. The lightweight mutex related to the lightweight condition variable is released atomically with 1 unlock when the thread changes to WAITING state with the execution of `sceKernelWaitLwCond()` for this lightweight condition variable and is owned with 1 lock when the thread is awakened from the WAITING state.
pOptParam	Argument for future expansion. Specify NULL.

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This creates a lightweight condition variable.

sceKernelDeleteLwCond

Delete lightweight condition variable

Definition

```
#include <kernel.h>
SceInt32 sceKernelDeleteLwCond(
        SceKernelLwCondWork *pWork
);
```

Argument

pWork Specify the address of the work area of the lightweight condition variable to be deleted.

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This deletes the lightweight condition variable specified with *pWork*.

When the lightweight condition variable is deleted, an error (undefined) will return to the threads that were waiting for the signal of the target condition variable. When the lightweight condition variable is deleted after receiving the signal of the lightweight condition variable, an error (undefined) will return to the threads that were waiting to own the related lightweight mutex.

sceKernelWaitLwCond

Wait for signal by lightweight condition variable

Definition

```
#include <kernel.h>
SceInt32 sceKernelWaitLwCond(
        SceKernelLwCondWork *pWork,
        SceUInt32 *pTimeout
);
```

Argument

pWork	Specify the work area of the lightweight condition variable to wait for.
pTimeout	Specify the pointer to the `SceUInt32` type variable storing maximum waiting time (in microseconds). If NULL is specified waiting time will be infinite. When the wait condition is satisfied, the remaining time will return.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This system call waits for the signal of the lightweight condition variable specified with *pWork*. The issuing thread changes to WAITING state. At the same time, the lightweight mutex related to the lightweight condition variable is released atomically with 1 unlock.

Specify *pTimeout* to set timeout operation (specified in microseconds). If *pTimeout* is NULL, timeout will not be performed. If *pTimeout* is specified, note that the value indicated by *pTimeout* will be updated when the system call terminates. If the condition is satisfied within the time limit, the value will be updated to the remaining time. If the condition is not satisfied, the value of *pTimeout* will be updated to 0, and the error `SCE_KERNEL_ERROR_WAIT_TIMEOUT` will return.

If multiple threads enter wait state for a lightweight condition variable, a thread waiting queue will be created. In case of a lightweight condition variable with the `SCE_KERNEL_LW_COND_ATTR_TH_FIFO` attribute, threads will be placed in the wait queue in the order in which they have entered waiting state, with the thread that has entered waiting state first at the top. In the case of a lightweight condition variable with the `SCE_KERNEL_LW_COND_ATTR_TH_PRIO` attribute, threads will be placed in the wait queue in accordance with their priority, with the thread with the highest priority at the top.

When a signal is reported to a waiting thread, the thread will awaken, and will own the lightweight mutex related to the lightweight condition variable with 1 lock. If the lightweight mutex cannot be owned immediately, the thread will wait to own the lightweight mutex.

If the lightweight mutex related to the lightweight condition variable has been canceled, the thread that had been waiting for the target condition variable will awaken and receive an error (undefined).

sceKernelSignalLwCond

Send signal to thread waiting for lightweight condition variable

Definition

```
#include <kernel.h>
SceInt32 sceKernelSignalLwCond(
        SceKernelLwCondWork *pWork
);
```

Argument

pWork Specify the identifier of the lightweight condition variable for which the signal is to be sent

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This sends a signal to a thread waiting for a lightweight condition variable specified with *pWork*. The thread at the top of the waiting queue for the target lightweight condition variable will receive the signal and awaken.

The signal sent will not be saved. In other words, if there are no threads waiting for the target lightweight condition variable when the signal is sent, this signal will be discarded. Note that threads entering wait state after the signal is sent cannot receive the signal.

sceKernelSignalLwCondAll

Send signal to all threads waiting for lightweight condition variable

Definition

```
#include <kernel.h>
SceInt32 sceKernelSignalLwCondAll(
        SceKernelLwCondWork *pWork
);
```

Argument

pWork Specify the identifier of the lightweight condition variable for which the signal is to be sent.

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This sends a signal to all threads waiting for a lightweight condition variable specified with *pWork*. When the signal is sent, all threads waiting for the target lightweight condition variable will receive the signal and awaken. Note that threads entering wait state after the signal is sent cannot receive the signal.

sceKernelSignalLwCondTo

Send signal to specific thread waiting for lightweight condition variable

Definition

```
#include <kernel.h>
SceInt32 sceKernelSignalLwCondTo(
        SceKernelLwCondWork *pWork,
        SceUID threadId
);
```

Argument

pWork	Specify the identifier of the lightweight condition variable for which the signal is to be sent.
threadId	Specify the identifier of the thread to which the signal is to be sent

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This sends a signal to a thread specified with *threadId* waiting for a lightweight condition variable specified with *pWork*.

If the target thread is waiting for the target lightweight condition variable, that thread will awaken. If the target thread is not waiting for the target lightweight condition variable when the signal is sent, an error (undefined) will return. Note that threads entering wait state after the signal is sent cannot receive the signal.

sceKernelGetLwCondInfo

Get lightweight condition variable status

Definition

```
#include <kernel.h>
SceInt32 sceKernelGetLwCondInfo(
        SceKernelLwCondWork *pWork,
        SceKernelLwCondInfo *pInfo
);
```

Argument

pWork Specify the address of the work area of the lightweight condition variable whose status is to be obtained.

pInfo Specify the pointer to the structure variable receiving lightweight condition variable status. Always assign `sizeof(SceKernelLwCondInfo)` to *pInfo*->*size* when calling.

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This obtains the status of a lightweight condition variable specified with *pWork*.

This system call is provided to aid debugging. Information that can be obtained by this system call changes moment by moment. Programming that issues this system call frequently, then changes the control flow according to the information that was received is not recommended.

sceKernelGetLwCondInfoById

Get lightweight condition variable status by its identifier

Definition

```
#include <kernel.h>
SceInt32 sceKernelGetLwCondInfoById(
        SceUID lwCondId,
        SceKernelLwCondInfo *pInfo
);
```

Argument

lwCondId	Specify the identifier of the lightweight condition variable whose status is to be obtained
pInfo	Specify the pointer to the structure variable receiving lightweight condition variable status. Always assign `sizeof(SceKernelLwCondInfo)` to *pInfo*->*size* when calling.

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This obtains the status of lightweight condition variable specified with *lwCondId*.

This system call is provided to aid debugging. Information that can be obtained by this system call changes moment by moment. Programming that issues this system call frequently, then changes the control flow according to the information that was received is not recommended.

Timers

SceKernelTimerOptParam

Optional timer data

Definition

```
#include <kernel.h>
typedef struct _SceKernelTimerOptParam {
        SceSize size;
} SceKernelTimerOptParam;
```

Members

size Size of this structure. (Value of `sizeof(SceKernelTimerOptParam)`)

Description

This structure is used to store optional data provided when a timer is created with `sceKernelCreateTimer()`.

It is provided for future expansion.

SceKernelTimerInfo

Timer information

Definition

```
#include <kernel.h>
typedef struct _SceKernelTimerInfo {
        SceSize size;
        SceUID timerId;
        char name[SCE_UID_NAMELEN+1];
        SceUInt32 attr;
        SceInt32 fActive;
        SceKernelSysClock baseTime;
        SceKernelSysClock currentTime;
        SceKernelSysClock schedule;
        SceKernelSysClock interval;
        SceInt32 type;
        SceInt32 fRepeat;
        SceUInt32 numWaitThreads;
        SceInt32 reserved[1];
} SceKernelTimerInfo;
```

Members

size	Size of this structure. (Value of `sizeof(SceKernelTimerInfo)`)
timerId	Identifier of timer
name	Timer name specified with `sceKernelCreateTimer()`
attr	Attribute of timer specified with `sceKernelCreateTimer()`
fActive	Timer start flag (if 1, counter is running)
baseTime	Base process time of timer
currentTime	Current time of timer
schedule	Next scheduled event notification time of timer (when an event is set)
interval	Interval of timer event notification specified with `sceKernelSetTimerEvent()`
type	Event notification type specified with `sceKernelSetTimerEvent()`
fRepeat	Repeat flag of the timer specified with `sceKernelSetTimerEvent()` (if 1, periodic timer)
numWaitThreads	Number of threads waiting for timer
reserved	Reserved area

Description

This structure is for obtaining the timer information.

sceKernelCreateTimer

Create timer

Definition

```
#include <kernel.h>
SceUID sceKernelCreateTimer(
        const char *pName,
        SceUInt32 attr,
        const SceKernelTimerOptParam *pOptParam
);
```

Argument

pName Specify timer name.
Since the name of the timer is only used for the purpose of identification by the operator during debugging when inter-process communication is not performed, it need not be unique. The name's maximum length is 31 bytes.
NULL cannot be specified.

attr Specify the attribute of the timer.
Specify one of the following for the order of the waiting queue:
- `SCE_KERNEL_ATTR_TH_FIFO`: waiting threads are queued on a FIFO basis
- `SCE_KERNEL_ATTR_TH_PRIO`: waiting threads are queued based on thread priority

Specify one of the following as reset method of timer events:
- `SCE_KERNEL_EVENT_ATTR_MANUAL_RESET`: timer events remain in event notification state until manually switched to non-notification state
- `SCE_KERNEL_EVENT_ATTR_AUTO_RESET`: timer events return to event non-notification state automatically after waking up one waiting thread

Specify one of the following as the notification method for the callbacks registered to the timer:
- `SCE_KERNEL_ATTR_NOTIFY_CB_ALL`: when a timer event is reported, all the callbacks registered to the target timer will be reported
- `SCE_KERNEL_ATTR_NOTIFY_CB_WAKEUP_ONLY`: when a timer event is reported, only the callbacks of the threads that awake from waiting for the target timer will be reported among the callbacks registered to the target timer

To create a timer that can be referenced with `sceKernelOpenTimer()`, specify the following:
- `SCE_KERNEL_ATTR_OPENABLE`

pOptParam Argument for future expansion.
Specify NULL.

Return Values

Value	Description
Positive number	Timer identifier (UID)
Negative value	Error code

Description

This creates a timer. The identifier of the timer that has been created is returned as a return value. The timer is implemented as an event object.

When specifying a name exceeding 31 bytes to *pName*, an error (SCE_KERNEL_ERROR_UID_NAME_TOO_LONG) will return when SCE_KERNEL_ATTR_OPENABLE is added to *attr*. An error will not occur if SCE_KERNEL_ATTR_OPENABLE is not added.

SCE_KERNEL_EVENT_CREATE event will be reported to the created timer.

sceKernelDeleteTimer

Delete timer

Definition

```
#include <kernel.h>
SceInt32 sceKernelDeleteTimer(
        SceUID timerId
);
```

Argument

`timerId` Identifier of timer

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This deletes the timer specified with *timerId* and created with `sceKernelCreateTimer()`.

When `sceKernelDeleteTimer()` is executed, *timerId* will be invalidated and become unusable. However, if other CPUs are in the midst of processing in relation to the target timer or references from other processes are being performed, actual object deletion will be delayed until referencing is over. When the timer is deleted, an error (`SCE_KERNEL_ERROR_WAIT_DELETE`) will return to the threads that were waiting for the target timer.

The `SCE_KERNEL_EVENT_DELETE` event is notified to a deleted timer and if it is being referenced separately, this can be received.

The identifier obtained with `sceKernelOpenTimer()` cannot be specified to *timerId*.

When specified, `SCE_KERNEL_ERROR_UNKNOWN_TIMER_ID` will return.

sceKernelOpenTimer

Reference timer

Definition

```
#include <kernel.h>
SceUID sceKernelOpenTimer(
        const char *pName
);
```

Argument

pName Specify the name of the timer to be referenced

Return Values

Value	Description
Positive value	Timer identifier
Negative value	Error code

Description

This references the timer specified with *pName*. The new identifier of the timer is returned as a return value. This system call is provided for the purpose of performing inter-process communication.

SCE_KERNEL_EVENT_OPEN event will be reported to the referenced timer.

In order to reference a timer, the SCE_KERNEL_ATTR_OPENABLE attribute must be specified as the *attr* argument of sceKernelCreateTimer() when creating the timer. Note that it is necessary to define a unique name for a timer within the system if specifying the attribute.

sceKernelCloseTimer

Close reference of timer

Definition

```
#include <kernel.h>
SceInt32 sceKernelCloseTimer(
        SceUID timerId
);
```

Argument

`timerId` Specify the identifier of the timer whose reference is to be terminated.

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This terminates reference of the timer specified with *timerId* and referenced with `sceKernelOpenTimer()`. This system call is provided for the purposes of performing inter-process communication.

When `sceKernelCloseTimer()` is executed, *timerId* is invalidated. Also, when there are no more references to the target timer, the timer will be deleted.

When the timer is deleted, an error (`SCE_KERNEL_ERROR_WAIT_DELETE`) will return to the threads that had been waiting for the target timer.

The `SCE_KERNEL_EVENT_CLOSE` event is notified to the timer whose reference has been terminated.

The identifier obtained with `sceKernelCreateTimer()` cannot be specified to *timerId*.

When specified, `SCE_KERNEL_ERROR_UNKNOWN_TIMER_ID` will return.

sceKernelStartTimer

Start timer

Definition

```
#include <kernel.h>
SceInt32 sceKernelStartTimer(
        SceUID timerId
);
```

Argument

timerId Specify the identifier of the timer to be started

Return Values

Value	Description
0	Timer was started.
1	Timer was already started.
Negative value	Error code

Description

This starts the count of a timer specified with *timerId*.

sceKernelStopTimer

Stop timer

Definition

```
#include <kernel.h>
SceInt32 sceKernelStopTimer(
        SceUID timerId
);
```

Argument

`timerId` Specify the identifier of the timer to be stopped.

Return Values

Value	Description
0	Timer was already stopped.
1	Timer was stopped.
Negative value	Error code

Description

This stops the count of a timer specified with `timerId`.

sceKernelGetTimerBase

Get base process time of timer

Definition

```
#include <kernel.h>
SceInt32 sceKernelGetTimerBase(
        SceUID timerId,
        SceKernelSysClock *pBase
);
```

Argument

timerId	Specify the identifier of the timer whose base process time is to be obtained
pBase	Specify the pointer to the structure variable receiving base time.

Return Values

Value	Description
Negative value	Error code

Description

This obtains the base process time of the timer specified with `timerId` (in other words, the process time value corresponding to “0” time of that timer).

Since the count value is initialized to 0 immediately after the timer is created, the base process time of the timer normally corresponds to the time in which the timer is started with `sceKernelStopTimer()`. If the timer is momentarily stopped with `sceKernelStopTimer()`, or the time is modified with `sceKernelSetTimerTime()`, base process time will no longer correspond to the process time in which the timer is started.

sceKernelGetTimerBaseWide

Get base process time of timer in 64 bits wide

Definition

```
#include <kernel.h>
SceUInt64 sceKernelGetTimerBaseWide(
        SceUID timerId
);
```

Argument

timerId Specify the identifier of the timer whose base process time is to be obtained.

Return Values

Value	Description
Not -1	Base process time
-1	An invalid timer identifier has been specified

Description

This obtains the base process time of the timer specified with *timerId* (in other words, the process time value corresponding to "0" time of that timer).

It is different from `sceKernelGetTimerBase()` in that it returns the result directly as a `SceUInt64` value.

sceKernelGetTimerTime

Get current time of timer

Definition

```
#include <kernel.h>
SceInt32 sceKernelGetTimerTime(
        SceUID timerId,
        SceKernelSysClock *pClock
);
```

Argument

timerId	Specify the identifier of the timer whose current time is to be obtained.
pClock	Specify the pointer to the structure variable receiving current time

Return Values

Value	Description
Negative value	Error code

Description

This obtains the current time of the timer specified with *timerId* in microseconds.

sceKernelGetTimerTimeWide

Get current time of timer in 64 bits wide

Definition

```
#include <kernel.h>
SceUInt64 sceKernelGetTimerTimeWide(
        SceUID timerId
);
```

Argument

`timerId` Specify the identifier of the timer whose current time is to be obtained.

Return Values

Value	Description
Not -1	Current time
-1	An invalid timer identifier has been specified

Description

This obtains the current time of the timer specified with *timerId* in microseconds.

It is different from `sceKernelGetTimerTime()` in that it returns the result directly as a `SceUInt64` value.

sceKernelSetTimerTime

Set current time of timer

Definition

```
#include <kernel.h>
SceInt32 sceKernelSetTimerTime(
        SceUID timerId,
        SceKernelSysClock *pClock
);
```

Argument

timerId	Specify the identifier of the timer whose current time is to be set.
pClock	Specify the pointer to the structure variable for storing the current time to be set. If setting is successful, the current time before setting will be returned.

Return Values

Value	Description
Negative value	Error code

Description

This sets the current time of the timer specified with *timerId*. When the timer event has been set, and current time is set to a time past the scheduled time for timer event notification, the event will be reported immediately after setting.

sceKernelSetTimerTimeWide

Set current time of timer in 64 bits wide

Definition

```
#include <kernel.h>
SceUInt64 sceKernelSetTimerTimeWide(
        SceUID timerId,
        SceUInt64 clock
);
```

Argument

timerId	Specify the identifier of the timer whose current time is to be set.
clock	Specify the current time to be set

Return Values

Value	Description
Not -1	Current time of timer before being set
-1	An invalid timer identifier has been specified.

Description

This sets the current time of the timer specified with *timerId*. When the timer event has been set, and current time is set to a time past the scheduled time for timer event notification, the event will be reported immediately after setting.

It is different from `sceKernelSetTimerTime()` in that the setting value is specified as a `SceUInt64` value.

sceKernelSetTimerEvent

Set timer event

Definition

```
#include <kernel.h>
SceInt32 sceKernelSetTimerEvent(
        SceUID timerId,
        SceInt32 type,
        SceKernelSysClock *pInterval,
        SceInt32 fRepeat
);
```

Argument

timerId	Specify the identifier of the timer for which the timer event is to be set.
type	Specify the notification type of timer events to be reported. Specify one of the following. - `SCE_KERNEL_TIMER_TYPE_SET_EVENT`: set type notification of timer event - `SCE_KERNEL_TIMER_TYPE_PULSE_EVENT`: pulse type notification of timer event
pInterval	Specify the pointer to the structure variable storing the period of the timer event. The period is counted in microseconds.
fRepeat	Specify repetition settings of the timer event. If 0 is specified, the timer event will only occur once after the time specified in *pInterval* elapses. If 1 is specified, the timer event will be repeated, and will occur every time the time specified in *pInterval* elapses.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This sets the timer event to a timer specified with *timerId*.

Since the timer is implemented as an event object, it is possible to use all event features. When this system call is called, the timer enters the non-notification state, and after the time specified with *pInterval* has elapsed, the timer event (`SCE_KERNEL_EVENT_TIMER`) will be reported. User data attached to this event is 0. Event periods are counted in microseconds.

Notification of timer events can be waited for with `sceKernelWaitEvent()`/ `sceKernelWaitMultipleEvents()`.

If the timer is the `SCE_KERNEL_EVENT_ATTR_AUTO_RESET` attribute when a timer event is reported, only the thread at the top of the waiting queue will awaken.

If the timer is the `SCE_KERNEL_EVENT_ATTR_MANUAL_RESET` attribute, all threads in the waiting queue will awaken.

When a timer event is reported, the callback registered to the timer will also be reported at the same time. If the timer is the `SCE_KERNEL_ATTR_NOTIFY_CB_WAKEUP_ONLY` attribute, only the callback of the thread awakened by the relevant timer event is reported among the callbacks registered to the target timer. If the timer is the `SCE_KERNEL_ATTR_NOTIFY_CB_ALL` attribute, all callbacks registered to the target timer are reported.

Based on *type* specification, it is possible to choose set notification or pulse notification as the notification type of the event to be reported.

When *fRepeat* is specified as 1, the timer becomes a periodic timer and the timer event will be reported repeatedly every time the time specified in *pInterval* elapses.

Document serial number: 000004892117

Document serial number: 000004892117

sceKernelCancelTimer

Cancel timer

Definition

```
#include <kernel.h>
SceInt32 sceKernelCancelTimer(
        SceUID timerId,
        SceUInt32 *pNumWaitThreads
);
```

Argument

timerId	Specify the identifier of the timer to be canceled.
pNumWaitThreads	Specify the pointer to the `SceUInt32` type variable receiving the number of threads of which waiting state is canceled. The number of threads of which wait state is canceled will not be received if NULL is specified.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This cancels the thread waiting state that had been waiting for the timer specified with *timerId*. The threads of which waiting state is canceled receives `SCE_KERNEL_ERROR_WAIT_CANCEL` as the return value of `sceKernelWaitEvent()`, and it is possible to determine that waiting state of them have been canceled.

The timer event setting for the canceled timer will be canceled.

sceKernelGetTimerInfo

Get timer status

Definition

```
#include <kernel.h>
SceInt32 sceKernelGetTimerInfo(
        SceUID timerId,
        SceKernelTimerInfo *pInfo
);
```

Argument

timerId	Specify the identifier of the timer whose status is to be obtained
pInfo	Specify the pointer to the structure variable receiving timer status. Always assign `sizeof(SceKernelTimerInfo)` to *pInfo*->*size* when calling.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This obtains the status of a timer specified with *timerId*.

This system call is provided to aid debugging. Information that can be obtained by this system call changes moment by moment. Programming that issues this system call frequently, then changes the control flow according to the information that was received is not recommended.

Reader/Writer Locks

SceKernelRWLockOptParam

Reader/writer lock optional data

Definition

```
#include <kernel.h>
typedef struct _SceKernelRWLockOptParam {
        SceSize size;
} SceKernelRWLockOptParam;
```

Members

size Size of this structure. (Value of `sizeof(SceKernelRWLockOptParam)`)

Description

This structure is used to store optional data that is provided when a reader/writer lock is created with `sceKernelCreateRWLock()`.

This is provided for future expansion.

SceKernelRWLockInfo

Reader/writer lock status

Definition

```
#include <kernel.h>
typedef struct _SceKernelRWLockInfo {
        SceSize size;
        SceUID rwLockId;
        char name[SCE_UID_NAMELEN+1];
        SceUInt32 attr;
        SceInt32 lockCount;
        SceUID writeOwnerId;
        SceUInt32 numReadWaitThreads;
        SceUInt32 numWriteWaitThreads;
} SceKernelRWLockInfo;
```

Members

size	Size of this structure. (Value of `sizeof(SceKernelRWLockInfo)`)
rwLockId	Identifier of reader/writer lock
name	Name of reader/writer lock specified with `sceKernelCreateRWLock()`
attr	Attribute of reader/writer lock specified with `sceKernelCreateRWLock()`
lockCount	Current lock count of reader/writer lock
writeOwnerId	Identifier of the thread currently write-locking (owning) the reader/writer lock
numReadWaitThreads	Number of threads waiting for reader/writer lock (read lock)
numWriteWaitThreads	Number of threads waiting for reader/writer lock (write lock)

Description

This structure is used to obtain the reader/writer lock status with `sceKernelGetRWLockInfo()`.

sceKernelCreateRWLock

Create reader/writer lock

Definition

```
#include <kernel.h>
SceUID sceKernelCreateRWLock(
        const char *pName,
        SceUInt32 attr,
        const SceKernelRWLockOptParam *pOptParam
);
```

Arguments

pName	Specify reader/writer lock name. Since the name of the reader/writer lock is only used for the purpose of identification by the operator during debugging when inter-process communication is not performed, it need not be unique. The name's maximum length is 31 bytes. NULL cannot be specified.
attr	Specify the attribute of the reader/writer lock. Specify one of the following as the order of the waiting queue; - `SCE_KERNEL_RW_LOCK_ATTR_TH_FIFO`: waiting threads are queued on a FIFO basis - `SCE_KERNEL_RW_LOCK_ATTR_TH_PRIO`: waiting threads are queued based on priority Also, as an option it is possible to add one of the following specifications through logical OR: - `SCE_KERNEL_RW_LOCK_ATTR_RECURSIVE`: allow recursive lock by threads write-locking the reader/writer lock - `SCE_KERNEL_RW_LOCK_ATTR_CEILING`: use the priority ceiling feature when write-locking (not yet implemented) To create a reader/writer lock that can be referenced with `sceKernelOpenRWLock()`, specify the following: - `SCE_KERNEL_ATTR_OPENABLE`
pOptParam	Argument for future expansion. Specify NULL

Return Values

Value	Description
Positive value	Reader/writer lock identifier (UID)
Negative value	Error code

Description

This creates a reader/writer lock, and sets the initial write lock count of the reader/writer lock. The identifier of the reader/writer lock that is created is returned as a return value.

When the reader/writer lock is created, it is not write locked (owned) or read locked by any thread.

When specifying a name exceeding 31 bytes to *pName*, an error (SCE_KERNEL_ERROR_UID_NAME_TOO_LONG) will return when SCE_KERNEL_ATTR_OPENABLE is added to *attr*. An error will not occur if SCE_KERNEL_ATTR_OPENABLE is not added.

If the SCE_KERNEL_RW_LOCK_ATTR_CEILING attribute has been specified, the reader/writer lock will have a priority ceiling. If the priority of a thread write-locking a reader/writer lock with the priority ceiling is lower than that of the priority ceiling, the thread's priority will be increased to that of the ceiling while the thread write locks the reader/writer lock. However, it will not be possible to increase the priority of threads within the priority range of a common ready queue to the range of a Individual ready queue. The priority ceiling cannot be used in the case of read lock.

sceKernelDeleteRWLock

Delete reader/writer lock

Definition

```
#include <kernel.h>
SceInt32 sceKernelDeleteRWLock(
        SceUID rwLockId
);
```

Arguments

rwLockId Specify the identifier of reader/writer lock to be deleted.

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This deletes the reader/writer locks specified with *rwLockId* and created with sceKernelCreateRWLock().When sceKernelDeleteRWLock() is executed, *rwLockId* will be invalidated and become unusable. However, if other CPUs are in the midst of processing in relation to the target reader/writer lock, or references from other processes are being performed, actual object deletion will be delayed until referencing is over.

When the reader/writer lock is deleted, an error (SCE_KERNEL_ERROR_WAIT_DELETE) will return to the threads that were waiting for the target reader/writer lock.

The identifier obtained with sceKernelOpenRWLock() cannot be specified to *rwLockId*.

When specified, SCE_KERNEL_ERROR_UNKNOWN_RW_LOCK_ID will return.

sceKernelOpenRWLock

Reference of reader/writer lock

Definition

```
#include <kernel.h>
SceUID sceKernelOpenRWLock(
        const char *pName
);
```

Arguments

pName Specify the name of the reader/writer lock to be referenced.

Return Values

Value	Description
Positive value	Reader/writer lock identifier
Negative value	Error code

Description

This references the reader/writer lock specified with *pName*. The new identifier of the reader/writer lock will be returned as a return value. This system call is provided for the purpose of performing inter-process communication.

In order to reference a reader/writer lock, the `SCE_KERNEL_ATTR_OPENABLE` attribute must be specified as the *attr* argument of `sceKernelCreateRWLock()` when creating the reader/writer lock. Note that it is necessary to define a unique name for a reader/writer lock within the system if specifying the attribute.

sceKernelCloseRWLock

Close reference of reader/writer lock

Definition

```
#include <kernel.h>
SceInt32 sceKernelCloseRWLock(
        SceUID rwLockId
);
```

Arguments

rwLockId Specify the identifier of the reader/writer lock whose reference is to be terminated

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This terminates the reference of the reader/writer lock specified with *rwLockId* and referenced with sceKernelOpenRWLock(). This system call is provided for the purpose of performing inter-process communication.

When sceKernelCloseRWLock() is executed, *rwLockId* is invalidated. Also, when there are no more references to the target reader/writer lock, the reader/writer lock will be deleted.

When the reader/writer lock is deleted, an error (SCE_KERNEL_ERROR_WAIT_DELETE) will return to the threads that had been waiting for the target reader/writer lock.

The identifier obtained with sceKernelCreateRWLock() cannot be specified to *rwLockId*.

When specified, SCE_KERNEL_ERROR_UNKNOWN_RW_LOCK_ID will return.

Document serial number: 000004892117

Document serial number: 000004892117

sceKernelLockReadRWLock

Get read lock of reader/writer lock

Definition

```
#include <kernel.h>
SceInt32 sceKernelLockReadRWLock(
        SceUID rwLockId,
        SceUInt32 *pTimeout
);
```

Arguments

rwLockId	Specify the identifier of the reader/writer lock whose read lock is to be obtained.
pTimeout	Specify the pointer to the `SceUInt32` type variable storing maximum waiting time (in microseconds). If NULL is specified waiting time will be infinite. When the wait condition is satisfied, the remaining time will return.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This system call obtains the read lock of the reader/writer lock specified with *rwLockId*. If there are no threads write-locking (owning) the reader/writer lock, and there are no threads waiting for write lock in the waiting queue, the issuing thread will immediately be successful in read-locking, and continue without transition to WAITING state. Read lock of the reader/writer lock can be obtained by multiple threads simultaneously, or recursively by a single thread. When read lock is obtained successfully, the read lock count of the reader/writer lock will be increased by 1.

If the target reader/writer lock is write-locked by other threads, or there are threads waiting for write lock in the waiting queue, the issuing thread will change to WAITING state and wait until read lock can be obtained.

Specify *pTimeout* to set timeout operation (specified in microseconds). If *pTimeout* is NULL, timeout will not be performed. If *pTimeout* is specified, note that the value indicated by *pTimeout* will be updated when the system call terminates. If the condition is satisfied within the time limit, the value will be updated to the remaining time. If the condition is not satisfied, the value of *pTimeout* will be updated to 0, and the error `SCE_KERNEL_ERROR_WAIT_TIMEOUT` will return.

If multiple threads enter wait state for a reader/writer lock, a thread waiting queue will be created.

In case of a reader/writer lock with the `SCE_KERNEL_RW_LOCK_ATTR_TH_FIFO` attribute, threads will be placed in the wait queue in the order in which they have entered waiting state, with the thread that has entered waiting state first at the top. In the case of a reader/writer lock with the `SCE_KERNEL_RW_LOCK_ATTR_TH_PRIO` attribute, threads will be placed in the wait queue in accordance with their priority, with the thread with the highest priority at the top.

When all preceding write locks to the target reader/writer lock are unlocked, the thread waiting for read lock in the waiting queue will succeed in obtaining read lock and awaken.

sceKernelLockReadRWLockCB

Get read lock of reader/writer lock (with callback check feature)

Definition

```
#include <kernel.h>
SceInt32 sceKernelLockReadRWLockCB(
        SceUID rwLockId,
        SceUInt32 *pTimeout
);
```

Arguments

rwLockId	Specify the identifier of the reader/writer lock whose read lock is to be obtained.
pTimeout	Specify the pointer to the `SceUInt32` type variable storing maximum waiting time (in microseconds). If NULL is specified waiting time will be infinite. When the wait condition is satisfied, the remaining time will return.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This system call obtains the read lock of the reader/writer lock specified with *rwLockId*. If there are no threads write-locking (owning) the reader/writer lock, and there are no threads waiting for write lock in the waiting queue, the issuing thread will immediately be successful in read-locking, and continue without transition to WAITING state. Read lock of the reader/writer lock can be obtained by multiple threads simultaneously, or recursively by a single thread. When read lock is obtained successfully, the read lock count of the reader/writer lock will be increased by 1.

If the target reader/writer lock is write-locked by other threads, or there are threads waiting for write lock in the waiting queue, the issuing thread will change to WAITING state and wait until read lock can be obtained.

Specify *pTimeout* to set timeout operation (specified in microseconds). If *pTimeout* is NULL, timeout will not be performed. If *pTimeout* is specified, note that the value indicated by *pTimeout* will be updated when the system call terminates. If the condition is satisfied within the time limit, the value will be updated to the remaining time. If the condition is not satisfied, the value of *pTimeout* will be updated to 0, and the error `SCE_KERNEL_ERROR_WAIT_TIMEOUT` will return.

If multiple threads enter wait state for a reader/writer lock, a thread waiting queue will be created.

In case of a reader/writer lock with the `SCE_KERNEL_RW_LOCK_ATTR_TH_FIFO` attribute, threads will be placed in the wait queue in the order in which they have entered waiting state, with the thread that has entered waiting state first at the top. In the case of a reader/writer lock with the `SCE_KERNEL_RW_LOCK_ATTR_TH_PRIO` attribute, threads will be placed in the wait queue in accordance with their priority, with the thread with the highest priority at the top.

When all preceding write locks to the target reader/writer lock are unlocked, the thread waiting for read lock in the waiting queue will succeed in obtaining read lock and awaken.

`sceKernelLockReadRWLockCB()` is a system call that adds a feature for checking for callback notifications to `sceKernelLockReadRWLock()`. Callback notifications will be checked immediately when this function is called, regardless of whether calling this function causes the thread to enter wait state. If the thread subsequently enters wait state, checks will continue while waiting also. If notification is performed for callback of the calling thread while waiting for time to elapse, the thread will momentarily exit wait state to execute the callback function before returning to wait state.

sceKernelTryLockReadRWLock

Try to get read lock of reader/writer lock

Definition

```
#include <kernel.h>
SceInt32 sceKernelTryLockReadRWLock(
        SceUID rwLockId
);
```

Arguments

rwLockId Specify the identifier of the reader/writer lock for which to try to obtain read lock.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This tries to obtain the read lock of the reader/writer lock specified with *rwLockId*.

`sceKernelTryLockReadRWLock()` is a system call without `sceKernelLockReadRWLock()`'s feature for entering wait state. Unlike `sceKernelLockReadRWLock()`, if the target reader/writer lock is already write-locked by another thread, or if there is a thread waiting for write lock in the waiting queue, an error (`SCE_KERNEL_ERROR_RW_LOCK_FAILED_TO_LOCK`) will immediately be returned.

sceKernelUnlockReadRWLock

Unlock read lock of reader/writer lock

Definition

```
#include <kernel.h>
SceInt32 sceKernelUnlockReadRWLock(
        SceUID rwLockId
);
```

Arguments

rwLockId Specify the identifier of the reader/writer lock whose read lock is to be unlocked.

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This unlocks the read lock of the reader/writer lock specified with *rwLockId*. When the read lock is unlocked, the read lock count of the target reader/writer lock will decrease by 1. However, if the read lock count has become a negative value due to unlocking, the read lock count will not decrease and an error (SCE_KERNEL_ERROR_RW_LOCK_UNLOCK_UDF) will be returned.

When the read lock count reaches 0 through read lock unlocking, if there is a thread waiting for write lock in the waiting queue it will receive the possession right and be awakened.

sceKernelLockWriteRWLock

Get write lock of reader/writer lock

Definition

```
#include <kernel.h>
SceInt32 sceKernelLockWriteRWLock(
        SceUID rwLockId,
        SceUInt32 *pTimeout
);
```

Arguments

rwLockId Specify the identifier of the reader/writer lock whose write lock is to be obtained.

pTimeout Specify the pointer to the `SceUInt32` type variable storing maximum waiting time (in microseconds).
If NULL is specified waiting time will be infinite. When the wait condition is satisfied, the remaining time will return.

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This system call obtains the write lock of the reader/writer lock specified with *rwLockId*. If the reader/writer lock specified with *rwLockId* is not write-locked (owned) or read locked by any thread, the issuing thread will own the reader/writer lock, and continue without transition to WAITING state. When write lock is obtained successfully, the write lock count of the reader/writer lock will be increased by 1.

If the target reader/writer lock is write-locked or read-locked by other threads, the issuing thread will change to WAITING state and wait until write lock is possible.

Recursive lock (performed multiple times) by the owning thread is possible for reader/writer locks with the `SCE_KERNEL_RW_LOCK_ATTR_RECURSIVE` attribute. Reader/writer locks with recursive lock are released when `sceKernelUnlockWriteRWLock()` turns the lock count to 0.

Specify *pTimeout* to set timeout operation (specified in microseconds). If *pTimeout* is NULL, timeout will not be performed. If *pTimeout* is specified, note that the value indicated by *pTimeout* will be updated when the system call terminates. If the condition is satisfied within the time limit, the value will be updated to the remaining time. If the condition is not satisfied, the value of *pTimeout* will be updated to 0, and the error `SCE_KERNEL_ERROR_WAIT_TIMEOUT` will return.

If multiple threads enter wait state for a reader/writer lock, a thread waiting queue will be created.

In case of a reader/writer lock with the `SCE_KERNEL_RW_LOCK_ATTR_TH_FIFO` attribute, threads will be placed in the wait queue in the order in which they have entered waiting state, with the thread that has entered waiting state first at the top. In the case of a reader/writer lock with the `SCE_KERNEL_RW_LOCK_ATTR_TH_PRIO` attribute, threads will be placed in the wait queue in accordance with their priority, with the thread with the highest priority at the top.

When all read locks/write locks of the reader/writer lock are unlocked, possession right will be passed to the thread waiting for write lock at the top of the waiting queue, which will awaken.

sceKernelLockWriteRWLockCB

Get write lock of reader/writer lock

Definition

```
#include <kernel.h>
SceInt32 sceKernelLockWriteRWLockCB(
        SceUID rwLockId,
        SceUInt32 *pTimeout
);
```

Arguments

rwLockId Specify the identifier of the reader/writer lock whose write lock is to be obtained.

pTimeout Specify the pointer to the SceUInt32 type variable storing maximum waiting time (in microseconds).
If NULL is specified waiting time will be infinite. When the wait condition is satisfied, the remaining time will return.

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This system call obtains the write lock of the reader/writer lock specified with *rwLockId*. If the reader/writer lock specified with *rwLockId* is not write-locked (owned) or read locked by any thread, the issuing thread will own the reader/writer lock, and continue without transition to WAITING state. When write lock is obtained successfully, the write lock count of the reader/writer lock will be increased by 1.

If the target reader/writer lock is write-locked or read-locked by other threads, the issuing thread will change to WAITING state and wait until write lock is possible.

Recursive lock (performed multiple times) by the owning thread is possible for reader/writer locks with the SCE_KERNEL_RW_LOCK_ATTR_RECURSIVE attribute. Reader/writer locks with recursive lock are released when sceKernelUnlockWriteRWLock() turns the lock count to 0.

Specify *pTimeout* to set timeout operation (specified in microseconds). If *pTimeout* is NULL, timeout will not be performed. If *pTimeout* is specified, note that the value indicated by *pTimeout* will be updated when the system call terminates. If the condition is satisfied within the time limit, the value will be updated to the remaining time. If the condition is not satisfied, the value of *pTimeout* will be updated to 0, and the error SCE_KERNEL_ERROR_WAIT_TIMEOUT will return.

If multiple threads enter wait state for a reader/writer lock, a thread waiting queue will be created.

In case of a reader/writer lock with the SCE_KERNEL_RW_LOCK_ATTR_TH_FIFO attribute, threads will be placed in the wait queue in the order in which they have entered waiting state, with the thread that has entered waiting state first at the top. In the case of a reader/writer lock with the SCE_KERNEL_RW_LOCK_ATTR_TH_PRIO attribute, threads will be placed in the wait queue in accordance with their priority, with the thread with the highest priority at the top.

Document serial number: 000004892117

Document serial number: 000004892117

When all read locks/write locks of the reader/writer lock are unlocked, possession right will be passed to the thread waiting for write lock at the top of the waiting queue, which will awaken.

`sceKernelLockWriteRWLockCB()` is a system call that adds a feature for checking for callback notifications to `sceKernelLockWriteRWLock()`. Callback notifications will be checked immediately when this function is called, regardless of whether calling this function causes the thread to enter wait state. If the thread subsequently enters wait state, checks will continue while waiting also. If notification is performed for callback of the calling thread while waiting for time to elapse, the thread will momentarily exit wait state to execute the callback function before returning to wait state.

sceKernelTryLockWriteRWLock

Try to get write lock of reader/writer lock

Definition

```
#include <kernel.h>
SceInt32 sceKernelTryLockWriteRWLock(
        SceUID rwLockId
);
```

Arguments

rwLockId Specify the identifier of the reader/writer lock for which to try to obtain write lock.

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This tries to obtain the write lock of the reader/writer lock specified with *rwLockId*.

sceKernelTryLockWriteRWLock() is a system call without sceKernelLockWriteRWLock()'s feature for entering wait state. Unlike sceKernelLockWriteRWLock(), if the target reader/writer lock is already write-locked or read-locked by other threads an error (SCE_KERNEL_ERROR_RW_LOCK_FAILED_TO_LOCK) will immediately be returned.

sceKernelUnlockWriteRWLock

Unlock write lock of reader/writer lock

Definition

```
#include <kernel.h>
SceInt32 sceKernelUnlockWriteRWLock(
        SceUID rwLockId
);
```

Arguments

rwLockId Specify the identifier of the reader/writer lock whose write lock is to be unlocked.

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This unlocks the write lock of the reader/writer lock specified with *rwLockId*. When the write lock is unlocked, the write lock count of the target reader/writer will decrease by 1 time.

When the write lock count reaches 0 through write lock unlocking, the thread at the top of the waiting queue will be awakened. If the thread at the top of the waiting queue has been waiting for read lock, the threads waiting after it will also obtain read lock and be awakened, provided that they have been waiting for read lock.

Only the thread that has write-locked the target reader/writer lock can unlock the write lock of the reader/writer lock.

sceKernelCancelRWLock

Cancel reader/writer lock

Definition

```
#include <kernel.h>
SceInt32 sceKernelCancelRWLock(
        SceUID rwLockId,
        SceUInt32 *pNumReadWaitThreads,
        SceUInt32 *pNumWriteWaitThreads,
        SceInt32 flag
);
```

Arguments

rwLockId	Specify the identifier of the reader/writer lock to be canceled
pNumReadWaitThreads	Pointer to the `SceUInt32` type variable receiving the number of threads waiting for read lock that have been canceled. The number of threads waiting for read lock of which waiting state is canceled will not be received if NULL is specified.
pNumWriteWaitThreads	Pointer to the `SceUInt32` type variable receiving the number of threads waiting for write lock that have been canceled. The number of threads waiting for write lock of which waiting state is canceled will not be received if NULL is specified.
flag	Specify the following flag with bit patterns. - `SCE_KERNEL_RW_LOCK_CANCEL_WITH_WRITE_LOCK`: after cancellation, the thread calling this function will be write-locked (if this flag is not specified, no thread will be write-locked).

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This cancels the thread waiting state waiting for the reader/writer lock specified with *rwLockId*. The thread of which waiting state is canceled receives `SCE_KERNEL_ERROR_WAIT_CANCEL` as the return value of `sceKernelLockReadRWLock()`/`sceKernelLockWriteRWLock()`, and it can be determined that waiting state of them have been canceled.

After waiting state canceling, if `SCE_KERNEL_RW_LOCK_CANCEL_WITH_WRITE_LOCK` is specified in *flag*, the issuing thread will be write-locked (1 lock). If `SCE_KERNEL_RW_LOCK_CANCEL_WITH_WRITE_LOCK` is not specified, no thread will be write-locked.

sceKernelGetRWLockInfo

Get reader/writer lock status

Definition

```
#include <kernel.h>
SceInt32 sceKernelGetRWLockInfo(
        SceUID rwLockId,
        SceKernelRWLockInfo *pInfo
);
```

Arguments

rwLockId	Specify the identifier of the reader/writer lock whose status is to be obtained.
pInfo	Specify the pointer to the structure variable receiving reader/writer lock status. Always assign `sizeof(SceKernelRWLockInfo)` to *pInfo*->*size* when calling.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This obtains the status of a reader/writer lock specified with *rwLockId*.

This system call is provided to aid debugging. Information that can be obtained by this system call changes moment by moment. Programming that issues this system call frequently, then changes the control flow according to the information that was received is not recommended.

Simple Events

SceKernelSimpleEventOptParam

Simple event optional data

Definition

```
#include <threadmgr.h>
typedef struct _SceKernelSimpleEventOptParam {
        SceSize size;
} SceKernelSimpleEventOptParam;
```

Members

size Size of this structure (value of `sizeof(SceKernelSimpleEventOptParam)`)

Description

This is the structure used to store the optional data to be given when generating a simple event with `sceKernelCreateSimpleEvent()`.

It is provided for future expansion.

sceKernelCreateSimpleEvent

Create simple event

Definition

```
#include <threadmgr.h>
SceUID sceKernelCreateSimpleEvent(
        const char *pName,
        SceUInt32 attr,
        SceUInt32 initPattern,
        const SceKernelSimpleEventOptParam *pOptParam
);
```

Arguments

pName	Specify the name of the simple event If inter-process communication is not performed, the name of the simple event is used only for identification purposes by the operator during debugging, so it need not be unique. However, if the `SCE_KERNEL_ATTR_OPENABLE` attribute is specified in *attr*, a unique name must be specified in the system. The maximum length of the name is 31 bytes. NULL cannot be specified.
attr	Specify the attribute of the simple event with logical OR Specify one of the following for the order of the waiting queue: - `SCE_KERNEL_ATTR_TH_FIFO`: waiting threads are queued on a FIFO basis - `SCE_KERNEL_ATTR_TH_PRIO`: waiting threads are queued based on thread priority Specify one of the following as reset method of simple events: - `SCE_KERNEL_EVENT_ATTR_MANUAL_RESET`: notified events remain in event notification state until manually switched to non-notification state - `SCE_KERNEL_EVENT_ATTR_AUTO_RESET`: notified events return to event non-notification state automatically after waking up one waiting thread Specify one of the following as the notification method for the callbacks registered to the simple event: - `SCE_KERNEL_ATTR_NOTIFY_CB_ALL`: when an event is reported, all the callbacks registered to the target simple event will be reported - `SCE_KERNEL_ATTR_NOTIFY_CB_WAKEUP_ONLY`: when an event is reported, only the callbacks of the threads that awake from waiting for the target simple event will be reported among the callbacks registered to the target simple event To create a simple event that can be referenced with `sceKernelOpenSimpleEvent()`, specify the following: - `SCE_KERNEL_ATTR_OPENABLE`
initPattern	Specify the initial event value of simple events The initial event values that can be specified from the user mode are only those in the user definition event (`SCE_KERNEL_EVENT_USER_DEFINED_MASK`) range.
pOptParam	Argument for future expansion. Specify NULL.

Return Values

Value	Description
Positive value	Simple event identifier (UID)
Negative value	Error code

Description

This generates simple events and sets the initial value of simple events. The identifier of the created simple event is returned as the return value. Simple events are implemented as event objects.

When specifying a name exceeding 31 bytes to *pName*, an error (SCE_KERNEL_ERROR_UID_NAME_TOO_LONG) will return when SCE_KERNEL_ATTR_OPENABLE is added to *attr*. An error will not occur if SCE_KERNEL_ATTR_OPENABLE is not added.

The SCE_KERNEL_EVENT_CREATE event is notified to a created simple event.

sceKernelDeleteSimpleEvent

Delete simple event

Definition

```
#include <threadmgr.h>
SceInt32 sceKernelDeleteSimpleEvent(
        SceUID simpleEventId
);
```

Arguments

simpleEventId Specify the identifier of the simple event to be deleted

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This deletes a simple event created with `sceKernelCreateSimpleEvent()` specified with *simpleEventId*.

Upon execution of `sceKernelDeleteSimpleEvent()`, *simpleEventId* becomes invalid and cannot be used, but if another CPU is currently performing processing related to the target simple event or the target simple event is being referenced from another process, the actual object deletion is delayed until all referencing ends.

If a simple event is deleted, an error (`SCE_KERNEL_ERROR_WAIT_DELETE`) is returned to the threads that had been waiting for that simple event.

The `SCE_KERNEL_EVENT_DELETE` event is notified to a deleted simple event and if it is being referenced separately, this can be received.

The identifier obtained with `sceKernelOpenSimpleEvent()` cannot be specified to *simpleEventId*.
When specified, `SCE_KERNEL_ERROR_UNKNOWN_SIMPLE_EVENT_ID` will return.

sceKernelOpenSimpleEvent

Reference simple event

Definition

```
#include <threadmgr.h>
SceUID sceKernelOpenSimpleEvent(
        const char *pName
);
```

Arguments

pName Specify the name of the simple event to be referenced

Return Values

Value	Description
Positive value	Simple event identifier
Negative value	Error code

Description

This references the simple event specified with *pName*. The new identifier of the simple event is returned as the return value. This is a system call provided for inter-process communication.

The SCE_KERNEL_EVENT_OPEN event is notified to the referenced simple event.

In order to reference a simple event, the SCE_KERNEL_ATTR_OPENABLE attribute must be specified as the *attr* argument of sceKernelCreateSimpleEvent() when creating the simple event. Note that it is necessary to define a unique name for a simple event within the system if specifying the attribute.

sceKernelCloseSimpleEvent

Close reference of simple event

Definition

```
#include <threadmgr.h>
SceInt32 sceKernelCloseSimpleEvent(
        SceUID simpleEventId
);
```

Arguments

simpleEventId Specify the identifier of the simple event whose referencing is to be closed

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This close reference of the simple event referenced with sceKernelOpenSimpleEvent() specified with *simpleEventId*. This system call is provided for the purposes of performing inter-process communication.

Upon execution of sceKernelCloseSimpleEvent(), *simpleEventId* becomes invalid. If all referencing of the target simple event is closed, the simple event is deleted.

If a simple event is deleted, an error (SCE_KERNEL_ERROR_WAIT_DELETE) is returned to the threads that had been waiting for that simple event.

The SCE_KERNEL_EVENT_CLOSE event is notified to simple events whose referencing has been closed.

The identifier obtained with sceKernelCreateSimpleEvent() cannot be specified to *simpleEventId*.
When specified, SCE_KERNEL_ERROR_UNKNOWN_SIMPLE_EVENT_ID will return.

Document serial number: 000004892117

Document serial number: 000004892117

Message Pipes

SceKernelMsgPipeOptParam

Message pipe optional data

Definition

```
#include <threadmgr.h>
typedef struct _SceKernelMsgPipeOptParam {
        SceSize size;
        SceUInt32 attr;
        SceInt32 reserved[2];
        SceUInt32 openLimit;
} SceKernelMsgPipeOptParam;
```

Members

size	Size of this structure (value of `sizeof(SceKernelMsgPipeOptParam)`)
attr	Bit pattern specifying the valid member of the structure
reserved	Reserved area
openLimit	The maximum number that can be referenced with `sceKernelOpenMsgPipe()` at one time This can be specified from 0 to `SCE_KERNEL_OPEN_LIMIT_MAX(127)`. If nothing is specified, infinite will be applied.

Description

This is a structure used to store the optional data given when creating a message pipe with `sceKernelCreateMsgPipe()`.

Specify the bit pattern indicating the valid member in *attr*.

The following macro can be specified with logical OR.

- `SCE_KERNEL_MSG_PIPE_OPT_ATTR_OPEN_LIMITATION`: Specifies the maximum number that can be referenced at one time with `sceKernelOpenMsgPipe()` (*openLimit*)

The maximum number of simultaneous referencing through `sceKernelOpenMsgPipe()` can be specified with *openLimit*.

The maximum number that can be specified is from 0 to `SCE_KERNEL_OPEN_LIMIT_MAX(127)`.

If nothing is specified, there are not restrictions on the number that can be referenced at one time.

Be sure to call this structure by assigning `sizeof(SceKernelMsgPipeOptParam)` to *size*.

Document serial number: 000004892117

SceKernelMsgPipeVector

Non-continuous data transmission/reception buffer information of message pipe

Definition

```
#include <threadmgr.h>
typedef struct _SceKernelMsgPipeVector {
        void *pBase;
        SceSize bufSize;
} SceKernelMsgPipeVector;
```

Members

pBase	Base address of buffer
bufSize	Size of buffer

Description

This is the structure used for specifying the non-continuous data to be transmitted/received with `sceKernelSendMsgPipeVector()`, `sceKernelSendMsgPipeVectorCB()`, `sceKernelTrySendMsgPipeVector()`, `sceKernelReceiveMsgPipeVector()`, `sceKernelReceiveMsgPipeVectorCB()`, or `sceKernelTryReceiveMsgPipeVector()`. One continuous memory area is specified with this structure.

Specify the base address in *pBase*, and specify the size (byte) in *bufSize*.

SceKernelMsgPipeInfo

State of message pipe

Definition

```
#include <threadmgr.h>
typedef struct _SceKernelMsgPipeInfo {
        SceSize size;
        SceUID msgPipeId;
        char name[SCE_UID_NAMELEN+1];
        SceUInt32 attr;
        SceSize bufferSize;
        SceSize freeSize;
        SceUInt32 numSendWaitThreads;
        SceUInt32 numReceiveWaitThreads;
} SceKernelMsgPipeInfo;
```

Members

size	Size of this structure (value of `sizeof(SceKernelMsgPipeInfo)`)
msgPipeId	Identifier of message pipe
name	Name of message pipe specified with `sceKernelCreateMsgPipe()`
attr	Attribute of message pipe specified with `sceKernelCreateMsgPipe()`
bufferSize	Total size of pipe buffer of message pipe (byte)
freeSize	Empty size of pipe buffer of message pipe (byte)
numSendWaitThreads	Number of threads waiting for transmission
numReceiveWaitThreads	Number of threads waiting for reception

Description

This is a structure used for getting the state of the message pipe with `sceKernelGetMsgPipeInfo()`.

sceKernelCreateMsgPipe

Create message pipe

Definition

```
#include <threadmgr.h>
SceUID sceKernelCreateMsgPipe(
        const char *pName,
        SceUInt32 type,
        SceUInt32 attr,
        SceSize bufSize,
        const SceKernelMsgPipeOptParam *pOptParam
);
```

Arguments

pName Specify message pipe name.
Since the name of the message pipe is only used for the purpose of identification by the operator during debugging when inter-process communication is not performed, it need not be unique. However, if the `SCE_KERNEL_ATTR_OPENABLE` attribute is specified in *attr*, a unique name must be specified in the system. The name's maximum length is 31 bytes.
NULL cannot be specified.

type Specify the type of the message pipe
Select from the following types.
- `SCE_KERNEL_MSG_PIPE_TYPE_USER_MAIN`: use the user main area memory as the pipe buffer
- `SCE_KERNEL_MSG_PIPE_TYPE_USER_CDRAM`: use the user CDRAM area memory as the pipe buffer (not implemented)

attr Specify the attribute of the message pipe with logical OR
Specify one of the following for the order of the waiting queue:
- `SCE_KERNEL_MSG_PIPE_ATTR_TH_FIFO`: waiting threads are queued on a FIFO basis
- `SCE_KERNEL_MSG_PIPE_ATTR_TH_PRIO`: waiting threads are queued based on thread priority

Specify one of the following as reset method of message pipe
However, regardless of what is specified here, `SCE_KERNEL_EVENT_DATA_EXIST` events will always be in notification state as long as the pipe buffer contains as much as 1 byte of data.
- `SCE_KERNEL_EVENT_ATTR_MANUAL_RESET`: events remain in event notification state until manually switched to non-notification state
- `SCE_KERNEL_EVENT_ATTR_AUTO_RESET`: notified events return to event non-notification state automatically after waking up one waiting thread

Specify one of the following as the notification method for the callbacks registered to the message pipe:
- `SCE_KERNEL_ATTR_NOTIFY_CB_ALL`: when an event is reported, all the callbacks registered to the target message pipe will be reported
- `SCE_KERNEL_ATTR_NOTIFY_CB_WAKEUP_ONLY`: when an event is reported, only the callbacks of the threads that awake from waiting for the target message pipe will be reported among the callbacks registered to the target message pipe

To create a message pipe that can be referenced with `sceKernelOpenMsgPipe()`, specify the following:
- `SCE_KERNEL_ATTR_OPENABLE`

bufSize	Specify the size of the pipe buffer The size of the pipe buffer must be a multiple of 4 KiB. Moreover, the total size of pipe buffers that can be allocated by a process is 32 MiB.
pOptParam	Specify the pointer to the structure variable specifying the message pipe option. Specify NULL if no option is to be specified. When specifying an option, always assign `sizeof(SceKernelMsgPipeOptParam)` to *pOptParam*->*size* when calling. To use the simultaneous reference number limitation feature, specify `SCE_KERNEL_MSG_PIPE_OPT_ATTR_OPEN_LIMITATION` in *pOptParam*->*attr* and at the same time specify the maximum simultaneous reference number in *pOptParam*->*openLimit*.

Return Values

Value	Description
Positive value	Message pipe identifier (UID)
Negative value	Error code

Description

This creates message pipes. The identifier of the created message pipe is returned as the return value. Message pipes are implemented as event objects.

When specifying a name exceeding 31 bytes to *pName*, an error (`SCE_KERNEL_ERROR_UID_NAME_TOO_LONG`) will return when `SCE_KERNEL_ATTR_OPENABLE` is added to *attr*. An error will not occur if `SCE_KERNEL_ATTR_OPENABLE` is not added.

The `SCE_KERNEL_EVENT_CREATE` event is notified to the created message pipes.

sceKernelDeleteMsgPipe

Delete message pipe

Definition

```
#include <threadmgr.h>
SceInt32 sceKernelDeleteMsgPipe(
        SceUID msgPipeId
);
```

Arguments

msgPipeId Specify the identifier of the message pipe to be deleted

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This deletes the message pipe created with `sceKernelCreateMsgPipe()` specified with *msgPipeId*.

Upon execution of `sceKernelDeleteMsgPipe()`, *msgPipeId* becomes invalid and cannot be used, but if another CPU is currently performing processing related to the target message pipe or the target message pipe is being referenced from another process, the actual object deletion is delayed until all referencing ends.

If a message pipe is deleted, an error (`SCE_KERNEL_ERROR_WAIT_DELETE`) is returned to the threads that had been waiting for that message pipe.

The `SCE_KERNEL_EVENT_DELETE` event is notified to a deleted message pipe and if it is being referenced separately, this can be received.

The identifier obtained with `sceKernelOpenMsgPipe()` cannot be specified to *msgPipeId*.
When specified, `SCE_KERNEL_ERROR_UNKNOWN_MSG_PIPE_ID` will return.

sceKernelOpenMsgPipe

Reference message pipe

Definition

```
#include <threadmgr.h>
SceUID sceKernelOpenMsgPipe(
        const char *pName
);
```

Arguments

pName Specify the name of the message pipe to be referenced

Return Values

Value	Description
Positive value	Message pipe identifier
Negative value	Error code

Description

This references the message pipe specified with *pName*. The new identifier of the message pipe is returned as the return value. This is a system call provided for inter-process communication.

The `SCE_KERNEL_EVENT_OPEN` event is notified to the referenced message pipe.

In order to reference a message pipe, the `SCE_KERNEL_ATTR_OPENABLE` attribute must be specified as the *attr* argument of `sceKernelCreateMsgPipe()` when creating the message pipe. Note that it is necessary to define a unique name for a message pipe within the system if specifying the attribute.

sceKernelCloseMsgPipe

Close reference of message pipe

Definition

```
#include <threadmgr.h>
SceInt32 sceKernelCloseMsgPipe(
        SceUID msgPipeId
);
```

Arguments

msgPipeId Specify the identifier of the message pipe whose referencing is to be closed

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This closes reference of the message pipe referenced with `sceKernelOpenMsgPipe()` specified with *msgPipeId*. This is a system call provided for inter-process communication.

Upon execution of `sceKernelCloseMsgPipe()`, *msgPipeId* becomes invalid. When all referencing of the target message pipe has closed, the message pipe is deleted.

If a message pipe is deleted, an error (`SCE_KERNEL_ERROR_WAIT_DELETE`) is returned to the threads that had been waiting for that message pipe.

The `SCE_KERNEL_EVENT_CLOSE` event is notified to the message pipe whose referencing has closed.

The identifier obtained with `sceKernelCreateMsgPipe()` cannot be specified to *msgPipeId*. When specified, `SCE_KERNEL_ERROR_UNKNOWN_MSG_PIPE_ID` will return.

sceKernelSendMsgPipe

Send data to message pipe

Definition

```
#include <threadmgr.h>
SceInt32 sceKernelSendMsgPipe(
        SceUID msgPipeId,
        const void *pSendBuf,
        SceSize sendSize,
        SceUInt32 waitMode,
        SceSize *pResult,
        SceUInt32 *pTimeout
);
```

Arguments

msgPipeId	Specify the identifier of the message pipe data is to be sent to
pSendBuf	Specify the start address of the send buffer
sendSize	Specify the send size (byte)
waitMode	Specify the wait mode with logical OR Specify one of the following: - SCE_KERNEL_MSG_PIPE_MODE_FULL: completion upon transmission of all the data of the specified size - SCE_KERNEL_MSG_PIPE_MODE_ASAP: completion upon transmission of even just 1 byte of data Specify one of the following: - SCE_KERNEL_MSG_PIPE_MODE_WAIT: perform send processing with blocking - SCE_KERNEL_MSG_PIPE_MODE_DONT_WAIT: perform send processing with non-blocking
pResult	Specify the pointer to the SceSize type variable for receiving the size that was actually sent The actually sent size is not received if NULL is specified.
pTimeout	Specify the pointer to the SceUInt32 type variable storing the maximum waiting time (in microseconds) If NULL is specified, the waiting time will be infinite. When the wait condition is satisfied, the remaining time will return.

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This sends the message data with *pSendBuf* as the start address to the message pipe specified with *msgPipeId*. The message data is copied indirectly via the pipe buffer. If *sendSize* is larger than the pipe buffer size, an error (SCE_KERNEL_ERROR_ILLEGAL_SIZE) is returned.

If SCE_KERNEL_MSG_PIPE_MODE_FULL is specified in *waitMode*, all the data is copied to the buffer if the available space in the message pipe buffer is equal to or greater than *sendSize*, immediately returning. If there is not sufficient available space in the buffer, the thread is transferred to the WAITING state until there is sufficient space and is connected to the message transmission wait queue of the message pipe.

If SCE_KERNEL_MSG_PIPE_MODE_ASAP is specified in *waitMode* and there is at least 1 byte of available space in the message pipe buffer, all the data that will fit inside the buffer is copied, immediately returning. If the buffer is full, the thread is transferred to the WAITING state until 1 byte or more of data can be sent, and it is connected to the message transmission wait queue of the message pipe.

The message length that actually can be sent is returned to the SceSize type variable specified by *pResult*. If SCE_KERNEL_MSG_PIPE_MODE_FULL was specified in *waitMode*, and when the send operation ends normally and sceKernelSendMsgPipe() returns SCE_OK, that length becomes equal to *sendSize*. However, if the transmission is interrupted by timeout or forcible cancellation of the waiting state before the send message has been received in its entirety by the receiving thread, or if SCE_KERNEL_MSG_PIPE_MODE_ASAP is specified in *waitMode*, a value smaller than *sendSize* may be returned.

If SCE_KERNEL_MSG_PIPE_MODE_WAIT is specified in *waitMode*, the send processing is done through blocking (synchronous) operation.

If SCE_KERNEL_MSG_PIPE_MODE_DONT_WAIT is specified in *waitMode*, the send processing is done through non-blocking (asynchronous) operation. In the case of non-blocking operation, return is immediate regardless of the state of the pipe buffer, and the thread is not transferred to the WAITING state. However, during internal processing, the operation is as described above when SCE_KERNEL_MSG_PIPE_MODE_FULL/SCE_KERNEL_MSG_PIPE_MODE_ASAP is specified.

Completion of the send processing can be detected through notification of the SCE_KERNEL_EVENT_IN event. When the SCE_KERNEL_EVENT_IN event is notified to the message pipe, the send result (corresponding to the blocking operation's return value) is saved to the lower 32 bits of the user data, and the send size (corresponding to *pResult*) is saved to the upper 32 bits.

When *pTimeout* is specified, the timeout operation specified in microseconds is set. If *pTimeout* is NULL, the timeout operation is not performed. Note that if *pTimeout* is specified, the value indicated by *pTimeout* is updated upon system call completion. If the conditions are met in time, update is done with the remaining time. If the conditions are not met in time, the value indicated by *pTimeout* is updated by 0 and in this case, the error SCE_KERNEL_ERROR_WAIT_TIMEOUT is returned.

sceKernelSendMsgPipeCB

Send data to message pipe (with callback check feature)

Definition

```
#include <threadmgr.h>
SceInt32 sceKernelSendMsgPipeCB(
        SceUID msgPipeId,
        const void *pSendBuf,
        SceSize sendSize,
        SceUInt32 waitMode,
        SceSize *pResult,
        SceUInt32 *pTimeout
);
```

Arguments

msgPipeId	Specify the identifier of the message pipe data is to be sent to
pSendBuf	Specify the start address of the send buffer
sendSize	Specify the send size (byte)
waitMode	Specify the wait mode using logical OR Specify one of the following: - `SCE_KERNEL_MSG_PIPE_MODE_FULL`: completion upon transmission of all the data of the specified size - `SCE_KERNEL_MSG_PIPE_MODE_ASAP`: completion upon transmission of even just 1 byte of data Specify one of the following: - `SCE_KERNEL_MSG_PIPE_MODE_WAIT` : perform send processing with blocking - `SCE_KERNEL_MSG_PIPE_MODE_DONT_WAIT`: perform send processing with non-blocking
pResult	Specify the pointer to the `SceSize` type variable for receiving the size that was actually sent The actually sent size is not received if NULL is specified.
pTimeout	Specify the pointer to the `SceUInt32` type variable storing the maximum waiting time (in microseconds) If NULL is specified, the waiting time will be infinite. When the wait condition is satisfied, the remaining time will return.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This sends the message data with *pSendBuf* as the start address to the message pipe specified with *msgPipeId*.

`sceKernelSendMsgPipeCB()` is a system call that adds to `sceKernelSendMsgPipe()` a feature for checking whether or not a callback notification exists. Callback notifications will be checked immediately when this function is called, regardless of whether calling this function causes the thread to enter wait state. If the thread subsequently enters wait state, checks will continue while waiting also. If a callback notification is received for the local thread while waiting for time to elapse, the thread temporarily exits from the waiting state, the callback function is executed, and then the thread enters the waiting state again.

sceKernelTrySendMsgPipe

Try sending data to message pipe

Definition

```
#include <threadmgr.h>
SceInt32 sceKernelTrySendMsgPipe(
        SceUID msgPipeId,
        const void *pSendBuf,
        SceSize sendSize,
        SceUInt32 waitMode,
        SceSize *pResult
);
```

Arguments

msgPipeId	Specify the identifier of the message pipe to which data transmission is to be attempted
pSendBuf	Specify the start address of the send buffer
sendSize	Specify the send size (byte)
waitMode	Specify the wait mode using logical OR Specify one of the following: - SCE_KERNEL_MSG_PIPE_MODE_FULL: completion upon transmission of all the data of the specified size - SCE_KERNEL_MSG_PIPE_MODE_ASAP: completion upon transmission of even just 1 byte of data
pResult	Specify the pointer to the SceSize type variable for receiving the size that was actually sent. The actually sent size is not received if NULL is specified.

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This attempts to send message data with *pSendBuf* as the start address to the message pipe specified with *msgPipeId*.

sceKernelTrySendMsgPipe() is a system call that removes from sceKernelSendMsgPipe() the feature for transferring a thread into the WAITING state.

An error (SCE_KERNEL_ERROR_MPP_FULL) is immediately returned when *waitMode* is SCE_KERNEL_MSG_PIPE_MODE_FULL and there is not sufficient available space in the message pipe buffer for receiving all the send messages or there is no thread waiting for receiving, or if the *waitMode* is SCE_KERNEL_MSG_PIPE_MODE_ASAP and not even 1 byte of message can be received.

Specification of SCE_KERNEL_MSG_PIPE_MODE_WAIT and SCE_KERNEL_MSG_PIPE_MODE_DONT_WAIT to *waitMode* is invalid.

Document serial number: 000004892117

sceKernelSendMsgPipeVector

Send data from non-continuous buffer to message pipe

Definition

```
#include <threadmgr.h>
SceInt32 sceKernelSendMsgPipeVector(
        SceUID msgPipeId,
        const SceKernelMsgPipeVector *pVector,
        SceUInt32 numVector,
        SceUInt32 waitMode,
        SceSize *pResult,
        SceUInt32 *pTimeout
);
```

Arguments

msgPipeId	Specify the identifier of the message pipe data is to be sent to
pVector	Specify the pointer to the non-continuous data transmission/reception buffer information
numVector	Specify the number of buffer information items
waitMode	Specify the wait mode using logical OR Specify one of the following: - `SCE_KERNEL_MSG_PIPE_MODE_FULL`: completion upon transmission of all the data of the specified size - `SCE_KERNEL_MSG_PIPE_MODE_ASAP`: completion upon transmission of even just 1 byte of data Specify one of the following: - `SCE_KERNEL_MSG_PIPE_MODE_WAIT`: perform send processing with blocking - `SCE_KERNEL_MSG_PIPE_MODE_DONT_WAIT`: perform send processing with non-blocking
pResult	Specify the pointer to the `SceSize` type variable for receiving the size that was actually sent. The actually sent size is not received if NULL is specified.
pTimeout	Specify the pointer to the `SceUInt32` type variable storing the maximum waiting time (in microseconds) If NULL is specified, the waiting time will be infinite. When the wait condition is satisfied, the remaining time will return.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This sends message data from the non-continuous memory area specified with *pVector* to the message pipe specified with *msgPipeId*.

The operation of `sceKernelSendMsgPipeVector()` is equivalent to that of `sceKernelSendMsgPipe()`, and differs only in the method to specify the send source message data. The *numVector* number of buffer information items placed at the address specified with *pVector* is linked and they are regarded and treated as a concatenated message data.

sceKernelSendMsgPipeVectorCB

Send data from non-continuous buffer to message pipe (with callback check feature)

Definition

```
#include <threadmgr.h>
SceInt32 sceKernelSendMsgPipeVectorCB(
        SceUID msgPipeId,
        const SceKernelMsgPipeVector *pVector,
        SceUInt32 numVector,
        SceUInt32 waitMode,
        SceSize *pResult,
        SceUInt32 *pTimeout
);
```

Arguments

msgPipeId	Specify the identifier of the message pipe data is to be sent to
pVector	Specify the pointer to the non-continuous data transmission/reception buffer information
numVector	Specify the number of buffer information items
waitMode	Specify the wait mode using logical OR Specify one of the following: - `SCE_KERNEL_MSG_PIPE_MODE_FULL`: completion upon transmission of all the data of the specified size - `SCE_KERNEL_MSG_PIPE_MODE_ASAP`: completion upon transmission of even just 1 byte of data Specify one of the following: - `SCE_KERNEL_MSG_PIPE_MODE_WAIT`: perform send processing with blocking - `SCE_KERNEL_MSG_PIPE_MODE_DONT_WAIT`: perform send processing with non-blocking
pResult	Specify the pointer to the `SceSize` type variable for receiving the size that was actually sent. The actually sent size is not received if NULL is specified.
pTimeout	Specify the pointer to the `SceUInt32` type variable storing the maximum waiting time (in microseconds) If NULL is specified, the waiting time will be infinite. When the wait condition is satisfied, the remaining time will return.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This sends the message data from the non-continuous memory area specified with *pVector* to the message pipe specified with *msgPipeId*.

The operation of `sceKernelSendMsgPipeVectorCB()` is equivalent to that of `sceKernelSendMsgPipeCB()`, and differs only in the method to specify the send source message data. The *numVector* number of buffer information items placed at the address specified with *pVector* is linked and they are regarded and treated as a concatenated message data.

sceKernelTrySendMsgPipeVector

Try sending data from non-continuous buffer to message pipe

Definition

```
#include <threadmgr.h>
SceInt32 sceKernelTrySendMsgPipeVector(
        SceUID msgPipeId,
        const SceKernelMsgPipeVector *pVector,
        SceUInt32 numVector,
        SceUInt32 waitMode,
        SceSize *pResult
);
```

Arguments

msgPipeId	Specify the identifier of the message pipe to which data transmission is to be attempted
pVector	Specify the pointer to the non-continuous data transmission/reception buffer information
numVector	Specify the number of buffer information items
waitMode	Specify the wait mode using logical OR Specify one of the following: - `SCE_KERNEL_MSG_PIPE_MODE_FULL`: completion upon transmission of all the data of the specified size - `SCE_KERNEL_MSG_PIPE_MODE_ASAP`: completion upon transmission of even just 1 byte of data
pResult	Specify the pointer to the `SceSize` type variable for receiving the size that was actually sent The actually sent size is not received if NULL is specified.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This attempts to send message data from the non-continuous memory area specified with *pVector* to the message pipe specified with *msgPipeId*.

The operation of `sceKernelTrySendMsgPipeVector()` is equivalent to that of `sceKernelTrySendMsgPipe()`, and differs only in the method to specify the send source message data. The *numVector* number of buffer information items placed at the address specified with *pVector* is linked and they are regarded and treated as a concatenated message data.

sceKernelReceiveMsgPipe

Receive data from message pipe

Definition

```
#include <threadmgr.h>
SceInt32 sceKernelReceiveMsgPipe(
        SceUID msgPipeId,
        void *pRecvBuf,
        SceSize recvSize,
        SceUInt32 waitMode,
        SceSize *pResult,
        SceUInt32 *pTimeout
);
```

Arguments

msgPipeId	Specify the identifier of the message pipe from which data is to be received
pRecvBuf	Specify the start address of the receive buffer
recvSize	Specify the receive size (byte)
waitMode	Specify the wait mode using logical OR Specify one of the following: - `SCE_KERNEL_MSG_PIPE_MODE_FULL`: completion upon successful reception of all the data of the specified size - `SCE_KERNEL_MSG_PIPE_MODE_ASAP`: completion upon successful reception of even just 1 byte of data Specify one of the following: - `SCE_KERNEL_MSG_PIPE_MODE_WAIT`: perform receive processing with blocking - `SCE_KERNEL_MSG_PIPE_MODE_DONT_WAIT`: perform receive processing with non-blocking The following option can be additionally specified: - `SCE_KERNEL_MSG_PIPE_MODE_DONT_REMOVE`: don't remove the received data from the pipe buffer (peak processing)
pResult	Specify the pointer to the `SceSize` type variable for receiving the actually received size The actually received size is not received if NULL is specified.
pTimeout	Specify the pointer to the `SceUInt32` type variable storing the maximum waiting time (in microseconds) If NULL is specified, the waiting time will be infinite. When the wait condition is satisfied, the remaining time will return.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This receives the message data from the message pipe specified with *msgPipeId* and stores it to the receive buffer whose start address is *pRecvBuf*. The message data is copied indirectly via the pipe buffer. If *recvSize* is larger than the pipe buffer size, an error (`SCE_KERNEL_ERROR_ILLEGAL_SIZE`) is returned.

If SCE_KERNEL_MSG_PIPE_MODE_FULL is specified in *waitMode* and there are *recvSize* bytes of data in the message pipe buffer, the data is received from that buffer, immediately returning. If the size of the data in the buffer is less than *recvSize* bytes, the thread transitions to the WAITING state and is connected to the message receive wait queue of the message pipe until *recvSize* bytes or more of data enter the buffer.

If SCE_KERNEL_MSG_PIPE_MODE_ASAP is specified in *waitMode* and there is at least 1 byte of data in the message pipe buffer, up to *recvSize* bytes are received, immediately returning. If the buffer is empty, the thread is transferred to the WAITING state until transmission to the message pipe is done and is connected to the message reception queue of the message pipe.

The message length that can actually be received is returned to the SceSize type variable specified by *pResult*. If SCE_KERNEL_MSG_PIPE_MODE_FULL was specified in *waitMode*, and when the receive operation ends normally and sceKernelReceiveMsgPipe() returns SCE_OK, that length becomes equal to *recvSize*. However, if the reception is interrupted by timeout or forcible cancellation of the waiting state before the size of the received data is that of *recvSize*, or if SCE_KERNEL_MSG_PIPE_MODE_ASAP is specified in *waitMode*, a value smaller than *recvSize* may be returned.

If SCE_KERNEL_MSG_PIPE_MODE_WAIT is specified in *waitMode*, the receive processing is done through blocking (synchronous) operation.

If SCE_KERNEL_MSG_PIPE_MODE_DONT_WAIT is specified in *waitMode*, the receive processing is done through non-blocking (asynchronous) operation. In the case of non-blocking operation, return is immediate regardless of the state of the pipe buffer, and the thread is not transferred to the WAITING state. However, during internal processing, the operation is as described above when SCE_KERNEL_MSG_PIPE_MODE_FULL/SCE_KERNEL_MSG_PIPE_MODE_ASAP is specified. Completion of the receive processing can be detected through notification of the SCE_KERNEL_EVENT_OUT event.

When the SCE_KERNEL_EVENT_OUT event is notified to the message pipe, the receive result (corresponding to the blocking operation's return value) is saved to the lower 32 bits of the user data, and the receive size (corresponding to *pResult*) is saved to the upper 32 bits.

When SCE_KERNEL_MSG_PIPE_MODE_DONT_REMOVE is additionally specified in *waitMode*, the reception of the message data is the peak operation. In this case, the data on the pipe buffer is not deleted and remains as is even when the message data is received.

When *pTimeout* is specified, the specified timeout operation is set in microseconds. If *pTimeout* is NULL, the timeout operation is not performed. Note that if *pTimeout* is specified, the value indicated by *pTimeout* is updated upon system call completion. If the conditions are met in time, update is done with the remaining time. If the conditions are not met in time, the value indicated by *pTimeout* is updated by 0 and in this case, the error SCE_KERNEL_ERROR_WAIT_TIMEOUT is returned.

sceKernelReceiveMsgPipeCB

Receive data from message pipe (with callback check feature)

Definition

```
#include <threadmgr.h>
SceInt32 sceKernelReceiveMsgPipeCB(
        SceUID msgPipeId,
        void *pRecvBuf,
        SceSize recvSize,
        SceUInt32 waitMode,
        SceSize *pResult,
        SceUInt32 *pTimeout
);
```

Arguments

msgPipeId	Specify the identifier of the message pipe from which data is to be received
pRecvBuf	Specify the start address of the receive buffer
recvSize	Specify the receive size (byte)
waitMode	Specify the wait mode using logical OR Specify one of the following: - `SCE_KERNEL_MSG_PIPE_MODE_FULL`: completion upon successful reception of all the data of the specified size - `SCE_KERNEL_MSG_PIPE_MODE_ASAP`: completion upon successful reception of even just 1 byte of data Specify one of the following: - `SCE_KERNEL_MSG_PIPE_MODE_WAIT`: perform receive processing with blocking - `SCE_KERNEL_MSG_PIPE_MODE_DONT_WAIT`: perform receive processing with non-blocking The following option can be additionally specified: - `SCE_KERNEL_MSG_PIPE_MODE_DONT_REMOVE`: don't remove the received data from the pipe buffer (peak processing)
pResult	Specify the pointer to the `SceSize` type variable for receiving the actually received size The actually received size is not received if NULL is specified.
pTimeout	Specify the pointer to the `SceUInt32` type variable storing the maximum waiting time (in microseconds) If NULL is specified, the waiting time will be infinite. When the wait condition is satisfied, the remaining time will return.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This receives the message data from the message pipe specified with *msgPipeId* and stores it to the receive buffer whose start address is *pRecvBuf*.

`sceKernelReceiveMsgPipeCB()` is a system call that adds to `sceKernelReceiveMsgPipe()` a feature for checking whether or not a callback notification exists. Callback notifications will be checked immediately when this function is called, regardless of whether calling this function causes the thread to enter wait state. If the thread subsequently enters wait state, checks will continue while waiting also. If a callback notification is received for the local thread while waiting for time to elapse, the thread temporarily exits from the waiting state, the callback function is executed, and then the thread enters the waiting state again.

sceKernelTryReceiveMsgPipe

Try receiving data from message pipe

Definition

```
#include <threadmgr.h>
SceInt32 sceKernelTryReceiveMsgPipe(
        SceUID msgPipeId,
        void *pRecvBuf,
        SceSize recvSize,
        SceUInt32 waitMode,
        SceSize *pResult
);
```

Arguments

msgPipeId	Specify the identifier of the message pipe from which data reception is to be attempted
pRecvBuf	Specify the start address of the receive buffer
recvSize	Specify the receive size (byte)
waitMode	Specify the wait mode using logical OR Specify one of the following: - `SCE_KERNEL_MSG_PIPE_MODE_FULL`: completion upon successful reception of all the data of the specified size - `SCE_KERNEL_MSG_PIPE_MODE_ASAP`: completion upon successful reception of even just 1 byte of data The following option can be additionally specified: - `SCE_KERNEL_MSG_PIPE_MODE_DONT_REMOVE`: don't remove the received data from the pipe buffer (peak processing)
pResult	Specify the pointer to the `SceSize` type variable for receiving the actually received size The actually received size is not received if NULL is specified.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This attempts to receive message data from the message pipe specified with *msgPipeId* and stores it to the receive buffer with *pRecvBuf* as the start address.

`sceKernelTryReceiveMsgPipe()` is a system call without `sceKernelReceiveMsgPipe()`'s feature for transferring the thread to the WAITING state. If *waitMode* is `SCE_KERNEL_MSG_PIPE_MODE_FULL` and the combined data of the message pipe buffer and send wait thread is less than the *recvSize*, or if *waitMode* is `SCE_KERNEL_MSG_PIPE_MODE_ASAP`, the message pipe is empty, and there are no threads waiting for transmission to that message pipe, an error (`SCE_KERNEL_ERROR_MPP_EMPTY`) is immediately returned.

Specification of `SCE_KERNEL_MSG_PIPE_MODE_WAIT` and `SCE_KERNEL_MSG_PIPE_MODE_DONT_WAIT` to *waitMode* is invalid.

sceKernelReceiveMsgPipeVector

Receive data from message pipe to non-continuous buffer

Definition

```
#include <threadmgr.h>
SceInt32 sceKernelReceiveMsgPipeVector(
        SceUID msgPipeId,
        const SceKernelMsgPipeVector *pVector,
        SceUInt32 numVector,
        SceUInt32 waitMode,
        SceSize *pResult,
        SceUInt32 *pTimeout
);
```

Arguments

msgPipeId Specify the identifier of the message pipe from which data is to be received

pVector Specify the pointer to the non-continuous data transmission/reception buffer information

numVector Specify the number of buffer information items

waitMode Specify the wait mode using logical OR
Specify one of the following:
- `SCE_KERNEL_MSG_PIPE_MODE_FULL`: completion upon successful reception of all the data of the specified size
- `SCE_KERNEL_MSG_PIPE_MODE_ASAP`: completion upon successful reception of even just 1 byte of data

Specify one of the following:
- `SCE_KERNEL_MSG_PIPE_MODE_WAIT`: perform receive processing with blocking
- `SCE_KERNEL_MSG_PIPE_MODE_DONT_WAIT`: perform receive processing with non-blocking

The following option can be additionally specified:
- `SCE_KERNEL_MSG_PIPE_MODE_DONT_REMOVE`: don't remove the received data from the pipe buffer (peak processing)

pResult Specify the pointer to the `SceSize` type variable for receiving the actually received size
The actually received size is not received if NULL is specified.

pTimeout Specify the pointer to the `SceUInt32` type variable storing the maximum waiting time (in microseconds)
If NULL is specified, the waiting time will be infinite. When the wait condition is satisfied, the remaining time will return.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This receives message data to the non-continuous memory area specified with *pVector* from the message pipe specified with *msgPipeId*.

The operation of `sceKernelReceiveMsgPipeVector()` is equivalent to that of `sceKernelReceiveMsgPipe()`, and differs only in the method to specify the message data receive buffer. The *numVector* number of buffer information items placed at the address specified with *pVector* is linked and they are regarded and treated as a concatenated message data.

sceKernelReceiveMsgPipeVectorCB

Receive data from message pipe to non-continuous buffer (with callback check feature)

Definition

```
#include <threadmgr.h>
SceInt32 sceKernelReceiveMsgPipeVectorCB(
        SceUID msgPipeId,
        const SceKernelMsgPipeVector *pVector,
        SceUInt32 numVector,
        SceUInt32 waitMode,
        SceSize *pResult,
        SceUInt32 *pTimeout
);
```

Arguments

msgPipeId	Specify the identifier of the message pipe from which data is to be received
pVector	Specify the pointer to the non-continuous data transmission/reception buffer information
numVector	Specify the number of buffer information items
waitMode	Specify the wait mode using logical OR Specify one of the following: - `SCE_KERNEL_MSG_PIPE_MODE_FULL`: completion upon successful reception of all the data of the specified size - `SCE_KERNEL_MSG_PIPE_MODE_ASAP`: completion upon successful reception of even just 1 byte of data Specify one of the following: - `SCE_KERNEL_MSG_PIPE_MODE_WAIT`: perform receive processing with blocking - `SCE_KERNEL_MSG_PIPE_MODE_DONT_WAIT`: perform receive processing with non-blocking The following option can be additionally specified: - `SCE_KERNEL_MSG_PIPE_MODE_DONT_REMOVE`: don't remove the received data from the pipe buffer (peak processing)
pResult	Specify the pointer to the `SceSize` type variable for receiving the actually received size The actually received size is not received if NULL is specified.
pTimeout	Specify the pointer to the `SceUInt32` type variable storing the maximum waiting time (in microseconds) If NULL is specified, the waiting time will be infinite. When the wait condition is satisfied, the remaining time will return.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This receives the message data to the non-continuous memory area specified with *pVector* from the message pipe specified with *msgPipeId*.

The operation of `sceKernelReceiveMsgPipeVectorCB()` is equivalent to that of `sceKernelReceiveMsgPipeCB()`, and differs only in the method to specify the message data receive buffer. The *numVector* number of buffer information items placed at the address specified with *pVector* is linked and they are regarded and treated as a concatenated message data.

sceKernelTryReceiveMsgPipeVector

Try receiving data from message pipe to non-continuous buffer

Definition

```
#include <threadmgr.h>
SceInt32 sceKernelTryReceiveMsgPipeVector(
        SceUID msgPipeId,
        const SceKernelMsgPipeVector *pVector,
        SceUInt32 numVector,
        SceUInt32 waitMode,
        SceSize *pResult
);
```

Arguments

msgPipeId	Specify the identifier of the message pipe from which data reception is to be attempted
pVector	Specify the pointer to the non-continuous data transmission/reception buffer information
numVector	Specify the number of buffer information items
waitMode	Specify the wait mode using logical OR Specify one of the following: - `SCE_KERNEL_MSG_PIPE_MODE_FULL`: completion upon successful reception of all the data of the specified size - `SCE_KERNEL_MSG_PIPE_MODE_ASAP`: completion upon successful reception of even just 1 byte of data The following option can be additionally specified: - `SCE_KERNEL_MSG_PIPE_MODE_DONT_REMOVE`: don't remove the received data from the pipe buffer (peak processing)
pResult	Specify the pointer to the `SceSize` type variable for receiving the actually received size The actually received size is not received if NULL is specified.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This attempts to receive message data to the non-continuous memory area specified with *pVector* from the message pipe specified with *msgPipeId*.

The operation of `sceKernelTryReceiveMsgPipeVector()` is equivalent to that of `sceKernelTryReceiveMsgPipe()`, and differs only in the method to specify the message data receive buffer. The *numVector* number of buffer information items placed at the address specified with *pVector* is linked and they are regarded and treated as a concatenated message data.

sceKernelCancelMsgPipe

Cancel transmission/reception of message pipe

Definition

```
#include <threadmgr.h>
SceInt32 sceKernelCancelMsgPipe(
        SceUID msgPipeId,
        SceUInt32 *pNumSendWaitThreads,
        SceUInt32 *pNumReceiveWaitThreads
);
```

Arguments

msgPipeId	Specify the identifier of the message pipe for which wait is to be canceled
pNumSendWaitThreads	Specify the pointer to the SceUInt32 type variable for receiving the number of threads waiting for transmission of which waiting state is canceled. The number of threads waiting for transmission of which waiting state is canceled will not be received if NULL is specified.
pNumReceiveWaitThreads	Specify the pointer to the SceUInt32 type variable for receiving the number of threads waiting for receiving of which waiting state is canceled. The number of threads waiting for receiving of which waiting state is canceled will not be received if NULL is specified.

Return Values

Value	Description
SCE_OK	Success
Negative value	Error code

Description

This cancels the waiting state of the threads waiting the message pipe specified with *msgPipeId*. This also clears the data that remains in the pipe buffer of the message pipe.

Threads whose waiting state has been deleted receive the return values of sceKernelSendMsgPipe(), sceKernelReceiveMsgPipe() etc., or SCE_KERNEL_ERROR_WAIT_CANCEL as the user data of sceKernelWaitEvent() and can determine their cancellation of the waiting state.

sceKernelGetMsgPipeInfo

Get status of message pipe

Definition

```
#include <threadmgr.h>
SceInt32 sceKernelGetMsgPipeInfo(
        SceUID msgPipeId,
        SceKernelMsgPipeInfo *pInfo
);
```

Arguments

`msgPipeId`	Specify the identifier of the message pipe whose status is to be obtained
`pInfo`	Specify the pointer to the structure variable for receiving the status of the message pipe. Be sure to call this structure by assigning `sizeof(SceKernelMsgPipeInfo)` to *pInfo*->*size*.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This gets the status of the message pipe specified with *msgPipeId*.

This system call is provided to aid debugging. The information that can be obtained with this system call changes moment by moment. Programming that issues this system call frequency and changes the control flow according to the information that was received is not recommended.

IO/File Manager

Datatypes

SceIoDirent

Directory entry structure

Definition

```
#include <kernel.h>
typedef struct SceIoDirent {
        SceIoStat d_stat;
        char d_name[256];
        void *d_private;
        int dummy;
} SceIoDirent;
```

Members

d_stat	File status
d_name	File name (not file path)
d_private	Reserved (specify 0)
dummy	Padding

Description

This structure is used with `sceIoDread()`.

Notes

The operation is undefined when a value other than 0 is specified for `d_private`.

See Also

`sceIoDread()`

SceIoStat

Structure for obtaining status

Definition

```
#include <kernel.h>
typedef struct SceIoStat {
        SceIoMode st_mode;
        unsigned int st_attr;
        SceOff st_size;
        SceDateTime st_ctime;
        SceDateTime st_atime;
        SceDateTime st_mtime;
        unsigned int st_private[6];
} SceIoStat;
```

Members

st_mode	File type and mode
st_attr	Device-dependent attribute
st_size	File size
st_ctime	Creation time
st_atime	Last reference time
st_mtime	Last update time
st_private	Other

Description

This structure is used with `sceIoGetstat()`, `sceIoChstat()`, `sceIoGetstatByFd()` and `sceIoChstatByFd()`. The mode and file type (file/dir) specified with `sceIoOpen()` and `sceIoMkdir()` are entered into the *st_mode* field of `SceIoStat`. If time data does not exist in the device, all time data is NULL.

See Also

`sceIoChstat()`, `sceIoGetstat()`

SceIoMode

Type indicating file mode

Definition

```
#include <kernel.h>
typedef int SceIoMode;
```

Description

This is a type indicating the file mode.

Document serial number: 000004892117

Document serial number: 000004892117

SceIores

Iores type

Definition

```
#include <kernel.h>
typedef SceInt64 SceIores;
```

Description

This is an `Iores` type.

Functions

sceIoRemove

Delete file

Definition

```
#include <kernel.h>
int sceIoRemove(
        const char *filename
);
```

Arguments

filename Name of the file to be deleted

Return Values

Value	Description
0(SCE_OK)	Success
<0	Error SCE_ERROR_ERRNO_EMFILE SCE_ERROR_ERRNO_ENODEV

Description

This can be used to delete a file specified by an argument. If the attribute of the directory to which the file belongs is read-only, deletion will not be possible.

Notes

Refer to the list of error codes for other negative values.

sceIoMkdir

Make directory

Definition

```
#include <kernel.h>
int sceIoMkdir(
        const char *dirname,
        SceIoMode mode
);
```

Arguments

dirname	Name of directory to be created
mode	Mode of directory to be created

Return Values

Value	Description
0(SCE_OK)	Success
<0	Error SCE_ERROR_ERRNO_EMFILE SCE_ERROR_ERRNO_ENODEV

Description

This uses mode specified by an argument to create a directory specified by an argument. *mode* becomes the permission of the created directory. Specify SCE_STM_RWU or SCE_STM_RU for *mode*. *mode* can be obtained with sceIoGetstat() and is reflected in the *st_mode* field of the SceIoStat structure.

Notes

Refer to the list of error codes for other negative values.

See Also

sceIoRmdir()

sceIoRmdir

Delete directory

Definition

```
#include <kernel.h>
int sceIoRmdir(
        const char *dirname
);
```

Arguments

dirname Name of directory to be deleted

Return Values

Value	Description
0(SCE_OK)	Success
<0	Error SCE_ERROR_ERRNO_EMFILE SCE_ERROR_ERRNO_ENODEV

Description

This deletes the directory specified by an argument. If the specified directory contains files or directories and the directory is being opened, an error is returned without the directory being deleted.

Notes

Refer to the list of error codes for other negative values.

See Also

`sceIoMkdir()`

sceIoRename

Rename file

Definition

```
#include <kernel.h>
int sceIoRename(
        const char *oldname,
        const char *newname
);
```

Arguments

oldname Name of file to be renamed
newname Name of file after being renamed

Return Values

Value	Description
0(SCE_OK)	Success
<0	Error SCE_ERROR_ERRNO_EMFILE SCE_ERROR_ERRNO_ENODEV

Description

This renames the file. *oldname* and *newname* must have the same unit number for the same device. The device specification can be omitted for *newname*. Specifying only the filename to the *newname* argument enables the filename specified with *oldname* to be changed. This function can also be used to move files or directories. Specify a filename with a target path for *newname*.

Notes

Refer to the list of error codes for other negative values.

sceIoDevctl

Device control

Definition

```
#include <kernel.h>
int sceIoDevctl(
        const char *devname,
        int cmd,
        const void *arg,
        SceSize arglen,
        void *bufp,
        SceSize buflen
);
```

Arguments

devname	Specified device ("<devname><unit>:")
cmd	Command code
arg	Pointer to device-driver-dependent parameter block
arglen	Number of bytes in device-driver-dependent parameter block
bufp	Pointer to return data storage block
buflen	Size of return data storage block

Return Values

Value	Description
Non-negative (N>=0)	Success (driver-dependent)
<0	Error SCE_ERROR_ERRNO_EMFILE SCE_ERROR_ERRNO_ENODEV SCE_ERROR_ERRNO_EUNSUP

Description

This performs device-specific operations. The details of the operations are driver-dependent.

Notes

Refer to the list of error codes for other negative values.

sceIoSync

Synchronize device with memory

Definition

```
#include <kernel.h>
int sceIoSync(
        const char *devname,
        int flag
);
```

Arguments

devname	Device name
flag	Device-dependent flag

Return Values

Value	Description
0(SCE_OK)	Success
<0	Error SCE_ERROR_ERRNO_EMFILE SCE_ERROR_ERRNO_ENODEV SCE_ERROR_ERRNO_EUNSUP

Description

This synchronizes the memory state and device state. If the device name is specified, all files/directories under the target device name will be synchronized. If a particular file/directory is specified, the specified file/directory will be synchronized. The details of the operations are device driver-dependent.

Notes

Refer to the list of error codes for other negative values.

sceIoOpen

Open file

Definition

```
#include <kernel.h>
SceUID sceIoOpen(
        const char *filename,
        int flag,
        SceIoMode mode
);
```

Arguments

filename	Name of the file to be opened
flag	Mode of the file to be opened
mode	Mode of the file to be created (when SCE_O_CREAT was specified with *flag*)

Return Values

Value	Description
fd(*fd*>=0)	File descriptor
<0	Error SCE_ERROR_ERRNO_EMFILE SCE_ERROR_ERRNO_ENODEV SCE_ERROR_ERRNO_EUNSUP

Description

This opens a file specified by an argument. If the file is opened successfully, a small positive integer (the file descriptor) is returned. The filename specified by *filename* must have the following format.

device-name:[filename-within-device]

[Examples]

- "app0:/dir1/file.txt"

Specify a *flag* with SCE_O_RDONLY, SCE_O_WRONLY, or SCE_O_RDWR.

flag is handled as a bit pattern, not a signed integer.

Macro	Description
SCE_O_RDONLY	Opens a file with read-only access
SCE_O_WRONLY	Opens a file with write-only access
SCE_O_RDWR	Opens a file for both read and write

In addition, the following modes can be added to *flag* with a logical OR.

Macro	Description
SCE_O_APPEND	Append write mode. Writes are always done to the end of the file
SCE_O_CREAT	Creates a file if it does not already exist
SCE_O_TRUNC	When a file is opened for writing, the contents are discarded and the size is reset to 0
SCE_O_EXCL	When SCE_O_CREAT is specified, an error occurs if a file with the same name already exists
SCE_O_FDEXCL	An error occurs if a thread other than the thread that opened the file operates the file descriptor. (This flag is not supported.)

mode becomes the permission of the created file. The values that can be specified are:

- `SCE_STM_RWU`: Read and write
- `SCE_STM_RU`: Read-only

If no file is created, the specified value of *mode* does not affect the result of this API. *mode* is reflected in the *st_mode* field of the `SceIoStat` structure when the file status is obtained by `sceIoGetstat()`.

With this API, a file can be opened again even if it is already open. However, because this API has separate file registration tables, accessing the file, such as writing to the same file at the same time, will create a data mismatch. Therefore, do not write to the same file at the same time.

Notes

Refer to the list of error codes for other negative values.

See Also

`sceIoClose()`

sceIoClose

Close file

Definition

```
#include <kernel.h>
int sceIoClose(
        SceUID fd
);
```

Arguments

fd File descriptor obtained when the file was opened with `sceIoOpen()`

Return Values

Value	Description
0(`SCE_OK`)	Success

<0	Error
	`SCE_ERROR_ERRNO_EMFILE`
	`SCE_ERROR_ERRNO_ENODEV`
	`SCE_ERROR_ERRNO_EUNSUP`

Description

This closes an open file.

Notes

Refer to the list of error codes for other negative values.

See Also

`sceIoOpen()`

sceIoIoctl

File/device control (with return data)

Definition

```
#include <kernel.h>
int sceIoIoctl(
        SceUID fd,
        int cmd,
        const void *argp,
        SceSize arglen,
        void *bufp,
        SceSize buflen
);
```

Arguments

fd	File descriptor obtained when the file was opened with `sceIoOpen()`
cmd	Command code
argp	Pointer to device-driver-dependent parameter block
arglen	Number of bytes in device-driver-dependent parameter block
bufp	Pointer to return data storage block
buflen	Size of return data storage block

Return Values

Value	Description
Non-negative (N>=0)	Success (driver-dependent)
<0	Error SCE_ERROR_ERRNO_EMFILE SCE_ERROR_ERRNO_ENODEV SCE_ERROR_ERRNO_EUNSUP

Description

This performs device-dependent operations for open files.

Notes

The operation when this API is called is undefined.

Refer to the list of error codes for other negative values.

See Also

`sceIoOpen()`

sceIoLseek

Move file offset

Definition

```
#include <kernel.h>
SceOff sceIoLseek(
        SceUID fd,
        SceOff offset,
        int whence
);
```

Arguments

fd	File descriptor obtained when the file was opened with `sceIoOpen()`
offset	Move amount
whence	Move start point

Return Values

Value	Description
pos(*pos*>=0)	New file offset value
<0	Error `SCE_ERROR_ERRNO_EBADF` (Expanded to 64 bits with negative value) `SCE_ERROR_ERRNO_EINVAL` (Expanded to 64 bits with negative value) `SCE_ERROR_ERRNO_EUNSUP` (Expanded to 64 bits with negative value)

Description

This changes the position where the next read/write will be performed for an open file. Specify `SCE_SEEK_SET`, `SCE_SEEK_CUR`, or `SCE_SEEK_END` to *whence*.

offset is handled as the following:

Macro	Description
`SCE_SEEK_SET`	Distance from start of file
`SCE_SEEK_CUR`	Relative distance from current file offset
`SCE_SEEK_END`	Distance from end of file

The next file offset position is then determined.

Notes

Refer to the list of error codes for other negative values.

See Also

`sceIoOpen()`

sceIoLseek32

32-bit version file seek

Definition

```
#include <kernel.h>
long sceIoLseek32(
        SceUID fd,
        long offset,
        int whence
);
```

Arguments

fd	File descriptor obtained when the file was opened with `sceIoOpen()`
offset	Move amount
whence	Move start point

Return Values

Value	Description
pos(*pos*>=0)	New file offset value
<0	Error

Description

This is the 32-bit version file seek.

sceIoRead

Read file

Definition

```
#include <kernel.h>
SceSSize sceIoRead(
        SceUID fd,
        void *buf,
        SceSize nbyte
);
```

Arguments

fd File descriptor obtained when the file was opened with `sceIoOpen()`
buf Buffer address to be read
nbyte Size to be read

Return Values

Value	Description
bytes(bytes>=0)	Number of bytes actually read
<0	Error `SCE_ERROR_ERRNO_EBADF`

Description

This function reads files. Data is read from files opened with `sceIoOpen()`. For positive integers with a return value less than *nbyte*, data of that size only is read. If the value is negative, reading of the data has failed.

Notes

Refer to the list of error codes for other negative values.

See Also

`sceIoOpen(),sceIoWrite()`

sceIoWrite

Write file

Definition

```
#include <kernel.h>
SceSSize sceIoWrite(
        SceUID fd,
        const void *buf,
        SceSize nbyte
);
```

Arguments

fd File descriptor obtained when the file was opened with `sceIoOpen()`
buf Write data address
nbyte Write size

Return Values

Value	Description
bytes(bytes>=0)	Number of bytes actually written
<0	Error `SCE_ERROR_ERRNO_EBADF`

Description

This writes the file. This function writes data to files opened with `sceIoOpen()`. For positive integers with a return value less than *nbyte*, data of that size only is written.

Notes

Refer to the list of error codes for other negative values.

See Also

`sceIoOpen()`, `sceIoRead()`

sceIoPread

Read file from specified offset

Definition

```
#include <kernel.h>
SceSSize sceIoPread(
        SceUID fd,
        void *buf,
        SceSize nbyte,
        SceOff offset
);
```

Arguments

fd	File descriptor obtained when the file was opened with `sceIoOpen()`
buf	Buffer address to be read
nbyte	Size to be read
offset	The start point to be read (distance from start of file)

Return Values

Value	Description
bytes(bytes>=0)	Number of bytes actually read
<0	Error `SCE_ERROR_ERRNO_EBADF`

Description

This reads a file from the specified offset.

Notes

Refer to the list of error codes for other negative values.

See Also

`sceIoOpen()`, `sceIoPwrite()`

sceIoPwrite

Write file from specified offset

Definition

```
#include <kernel.h>
SceSSize sceIoPwrite(
        SceUID fd,
        const void *buf,
        SceSize nbyte,
        SceOff offset
);
```

Arguments

fd	File descriptor obtained when the file was opened with `sceIoOpen()`
buf	Data address to be written
nbyte	Size to be written
offset	The start point to be written (distance from start of file)

Return Values

Value	Description
bytes(bytes>=0)	Number of bytes actually written
<0	Error `SCE_ERROR_ERRNO_EBADF`

Description

This writes data to a file from the specified offset.

Notes

Refer to the list of error codes for other negative values.

See Also

`sceIoOpen(),sceIoPread()`

sceIoDopen

Open directory

Definition

```
#include <kernel.h>
SceUID sceIoDopen(
        const char *dirname
);
```

Arguments

dirname Directory name

Return Values

Value	Description
fd(*fd*>=0)	File descriptor
<0	Error SCE_ERROR_ERRNO_EMFILE SCE_ERROR_ERRNO_ENODEV

Description

This opens a directory. Once a directory has been opened, information on files in the directory can be read by `sceIoDread()`.

Notes

Refer to the list of error codes for other negative values.

See Also

`sceIoDread(),sceIoDclose()`

sceIoDclose

Close directory

Definition

```
#include <kernel.h>
int sceIoDclose(
        SceUID fd
);
```

Arguments

fd File descriptor

Return Values

Value	Description
0(SCE_OK)	Success
<0	Error SCE_ERROR_ERRNO_EBADF

Description

This closes a directory. This closes the file descriptor opened with `sceIoDopen()`.

Notes

Refer to the list of error codes for other negative values.

See Also

`sceIoDopen()`

sceIoDread

Read directory

Definition

```
#include <kernel.h>
int sceIoDread(
        SceUID fd,
        SceIoDirent *buf
);
```

Arguments

fd File descriptor
buf Read address

Return Values

Value	Description
0	No entries could be read
Positive (N>0)	Directory entry was read.
<0	Error `SCE_ERROR_ERRNO_EBADF`

Description

This reads the directory. After calling `sceIoDopen()`, the directory entries are returned individually starting with the first entry. The file name is present in the *d_name* member of the `SceIoDirent` structure.

Notes

Refer to the list of error codes for other negative values.

See Also

`sceIoDopen()`

sceIoChstat

Change status of file or directory

Definition

```
#include <kernel.h>
int sceIoChstat(
        const char *name,
        const SceIoStat *buf,
        unsigned int cbit
);
```

Arguments

name	Name of file/directory to be changed
buf	New status
cbit	Bit specification of field to be changed

Return Values

Value	Description
0(SCE_OK)	Success
<0	Error SCE_ERROR_ERRNO_EMFILE SCE_ERROR_ERRNO_ENODEV

Description

This changes the status of the file or directory specified by arguments. The following are valid specification fields within *buf*.

Macro	Field
SCE_CST_MODE	*st_mode*
SCE_CST_SIZE	*st_size*
SCE_CST_CT	*st_ctime*
SCE_CST_AT	*st_atime*
SCE_CST_MT	*st_mtime*

If the flag of the field to be changed is set to argument *cbit*, the members of the SceIoStat structure are reflected in the target file/directory. If SCE_CST_MODE is specified, specify SCE_STM_RWU for R/W, or SCE_STM_RU for read-only, to the *st_mode* member.

cbit is handled as a bit pattern.

Notes

Refer to the list of error codes for other negative values.

Document serial number: 000004892117

sceIoGetstat

Get status of file or directory

Definition

```
#include <kernel.h>
int sceIoGetstat(
        const char *name,
        SceIoStat *buf
);
```

Arguments

name Name of file/directory to be obtained
buf Buffer for storing status

Return Values

Value	Description
0(SCE_OK)	Success
<0	Error SCE_ERROR_ERRNO_EMFILE SCE_ERROR_ERRNO_ENODEV

Description

This obtains the status of the file or directory specified by an argument.

Following are the valid fields of *buf*.

Macro	Field
SCE_CST_MODE	*st_mode*
SCE_CST_SIZE	*st_size*
SCE_CST_CT	*st_ctime*
SCE_CST_AT	*st_atime*
SCE_CST_MT	*st_mtime*

Notes

Refer to the list of error codes for other negative values.

sceIoChstatByFd

Change status of file or directory being opened

Definition

```
#include <kernel.h>
int sceIoChstatByFd(
        SceUID fd,
        const SceIoStat *buf,
        unsigned int cbit
);
```

Arguments

fd	File descriptor obtained with `sceIoOpen()` or `sceIoDopen()`
buf	New status
cbit	Bit specification of field to be changed

Return Values

Value	Description
0(`SCE_OK`)	Success

<0	Error
	`SCE_ERROR_ERRNO_EBADF`
	`SCE_ERROR_ERRNO_ENODEV`

Description

This changes the status of the file/directory corresponding to the file descriptor specified with the argument.

The following are valid specification fields within *buf*.

Macro	Field
`SCE_CST_MODE`	*st_mode*
`SCE_CST_SIZE`	*st_size*
`SCE_CST_CT`	*st_ctime*
`SCE_CST_AT`	*st_atime*
`SCE_CST_MT`	*st_mtime*

If the flag of the field to be changed is set to argument *cbit*, the members of the `SceIoStat` structure are reflected in the target file/directory. If `SCE_CST_MODE` is specified, specify `SCE_STM_RWU` for R/W, or `SCE_STM_RU` for read-only, to the *st_mode* member.

Notes

Refer to the list of error codes for other negative values.

sceIoGetstatByFd

Get status of file or directory being opened

Definition

```
#include <kernel.h>
int sceIoGetstatByFd(
        SceUID fd,
        SceIoStat *buf
);
```

Arguments

fd File descriptor obtained with `sceIoOpen()` or `sceIoDopen()`
buf Buffer for storing status

Return Values

Value	Description
0(`SCE_OK`)	Success

<0	Error
	`SCE_ERROR_ERRNO_EBADF`
	`SCE_ERROR_ERRNO_ENODEV`

Description

This obtains the status of the file/directory corresponding to the file descriptor specified with the argument.

The following are valid specification fields within *buf*.

Macro	Field
`SCE_CST_MODE`	*st_mode*
`SCE_CST_SIZE`	*st_size*
`SCE_CST_CT`	*st_ctime*
`SCE_CST_AT`	*st_atime*
`SCE_CST_MT`	*st_mtime*

Notes

Refer to the list of error codes for other negative values.

sceIoSyncByFd

Synchronize device state with state of file or directory being opened

Definition

```
#include <kernel.h>
int sceIoSyncByFd(
        SceUID fd,
        int flag
);
```

Arguments

fd File descriptor obtained with `sceIoOpen()`, `sceIoDopen()`
flag Device-dependent flag

Return Values

Value	Description
0(`SCE_OK`)	Success

<0	Error
	`SCE_ERROR_ERRNO_EBADF`
	`SCE_ERROR_ERRNO_ENODEV`

Description

This synchronizes the state of file or directory being opened and device state. The details of the operations are device driver-dependent.

Notes

Refer to the list of error codes for other negative values.

Module Manager

Datatypes

SceKernelLoadModuleOpt

Optional data for loading PRX

Definition

```
#include <kernel.h>
typedef struct SceKernelLoadModuleOpt {
        SceSize size;
} SceKernelLoadModuleOpt;
```

Members

size Size of this structure. (Value of `sizeof(SceKernelLoadModuleOpt)`)

Description

This structure is used for storing the optional data given when loading PRX with `sceKernelLoadStartModule()` and `sceKernelLoadModule()`.

It is provided for future expansion.

SceKernelStartModuleOpt

Optional data for startingPRX

Definition

```
#include <kernel.h>
typedef struct SceKernelStartModuleOpt {
        SceSize size;
        SceUInt32 flags;
        SceUInt32 prologue;
        SceUInt32 start;
} SceKernelStartModuleOpt;
```

Members

size	Size of this structure. (Value of `sizeof(SceKernelStartModuleOpt)`)
flags	Reserved (specify 0)
prologue	Reserved (specify 0)
start	Reserved (specify 0)

Description

This structure is used for storing the optional data given when starting PRX with `sceKernelStartModule()`.

It is provided for future expansion.

SceKernelStopModuleOpt

Optional data for stopping PRX

Definition

```
#include <kernel.h>
typedef struct SceKernelStopModuleOpt {
        SceSize size;
        SceUInt32 flags;
        SceUInt32 epilogue;
        SceUInt32 stop;
} SceKernelStopModuleOpt;
```

Members

size	Size of this structure. (Value of `sizeof(SceKernelStopModuleOpt)`)
flags	Reserved (specify 0)
epilogue	Reserved (specify 0)
stop	Reserved (specify 0)

Description

This structure is used for storing the optional data given when stopping PRX with `sceKernelStopModule()`.

It is provided for future expansion.

SceKernelUnloadModuleOpt

Optional data for unloading PRX

Definition

```
#include <kernel.h>
typedef struct SceKernelUnloadModuleOpt {
        SceSize size;
} SceKernelUnloadModuleOpt;
```

Members

size	Size of this structure. (Value of `sizeof(SceKernelUnloadModuleOpt)`)

Description

This structure is used for storing the optional data given when unloading PRX with `sceKernelStopUnloadModule()` and `sceKernelUnloadModule()`.

It is provided for future expansion.

Functions

sceKernelLoadStartModule

Load and start PRX

Definition

```
#include <kernel.h>
SceUID sceKernelLoadStartModule(
        const char *moduleFileName,
        SceSize args,
        const void *argp,
        SceUInt32 flags,
        const SceKernelLoadModuleOpt *pOpt,
        int *pRes
);
```

Arguments

moduleFileName	PRX file name
args	Argument block size
argp	Pointer to the argument block
flags	Unused (Always specify 0)
pOpt	Unused (Always specify `SCE_NULL`)
pRes	Pointer to the area storing the return value of `module_start()`

Return Values

Value	Description
(>0)	PRX identifier
(<0)	Error code

Description

This performs start processing after loading the PRX specified with the *moduleFileName* argument.

During start processing, `module_start()` is called by a thread initially set with `SCE_KERNEL_DEFAULT_PRIORITY_USER` as priority and `SCE_KERNEL_THREAD_STACK_SIZE_DEFAULT_USER_MAIN` as stack size. At this point, the argument block specified with the arguments *args* and *argp* will be copied onto the thread's stack and passed to `module_start()`.

If loading and start processing are successful, the return value of `module_start()` will be stored in the area pointed to by *pRes*.

If `SCE_KERNEL_START_NO_RESIDENT` returns as the return value of `module_start()`, the module will not reside in the memory and will be unloaded automatically after `module_start()` is executed. If `SCE_KERNEL_START_FAILED` is returned, the PRX will fail to be loaded. If a value other than `SCE_KERNEL_START_NO_RESIDENT` or `SCE_KERNEL_START_FAILED` (including `SCE_KERNEL_START_RESIDENT`) is returned, the PRX will become resident (will be loaded).

Document serial number: 000004892117

Document serial number: 000004892117

sceKernelStopUnloadModule

Stop and unload PRX

Definition

```
#include <kernel.h>
int sceKernelStopUnloadModule(
        SceUID uid,
        SceSize args,
        const void *argp,
        SceUInt32 flags,
        const SceKernelUnloadModuleOpt *pOpt,
        int *pRes
);
```

Arguments

uid	PRX identifier
args	Argument block size
argp	Pointer to the argument block
flags	Unused (Always specify 0)
pOpt	Unused (Always specify SCE_NULL)
pRes	Pointer to the area storing the return value of module_stop()

Return Values

Value	Description
SCE_OK	Success
(<0)	Error code

Description

This performs unloading after stop processing of the PRX specified with the *uid* argument.

During stop processing, module_stop() is called by a thread initially set with SCE_KERNEL_DEFAULT_PRIORITY_USER as priority and SCE_KERNEL_THREAD_STACK_SIZE_DEFAULT_USER_MAIN as stack size. At this point, the argument block specified with the argument *args* and *argp* will be copied onto the thread's stack and passed to module_stop().

If stop processing is successful, after the library exported from the PRX is deleted unloading will be performed, and the return value of module_stop() will be stored in the area pointed to by *pRes*.

PRX stop and unload processing will only fail when SCE_KERNEL_STOP_CANCEL is returned as the return value of module_stop(). Stop and unload processing will be successful if any other values are returned.

sceKernelLoadModule

Load PRX

Definition

```
#include <kernel.h>
int sceKernelLoadModule(
        const char *moduleFileName,
        SceUInt32 flags,
        const SceKernelLoadModuleOpt *pOpt,
);
```

Arguments

moduleFileName	PRX file name
flags	Unused (Always specify 0)
pOpt	Unused (Always specify SCE_NULL)

Return Values

Value	Description
id	Loaded PRX identifier
(<0)	Error code

Description

This performs loading of the PRX specified with the *moduleFileName* argument.

Unlike with sceKernelLoadStartModule(), this function only loads PRX and does not perform start processing. To start the PRX that has been loaded, it is necessary to perform start processing by specifying the PRX identifier obtained as a return value in sceKernelStartModule().

sceKernelStartModule

Start PRX

Definition

```
#include <kernel.h>
int sceKernelStartModule(
        SceUID uid,
        SceSize args,
        const void *argp,
        SceUInt32 flags,
        const SceKernelStartModuleOpt *pOpt,
        int *pRes
);
```

Arguments

uid	PRX identifier
args	Argument block size
argp	Pointer to the argument block
flags	Unused (Always specify 0)
pOpt	Unused (Always specify `SCE_NULL`)
pRes	Pointer to the area storing the return value of `module_start()`

Return Values

Value	Description
(>0)	PRX identifier
(<0)	Error code

Description

This function performs a PRX's start processing by specifying the PRX identifier obtained with `sceKernelLoadModule()`.

The contents of start processing are the same as with `sceKernelLoadStartModule()`.

It is not possible to perform start processing again with `sceKernelStartModule()` for a PRX stopped with `sceKernelStopModule()`.

sceKernelStopModule

Stop PRX

Definition

```
#include <kernel.h>
int sceKernelStopModule(
        SceUID uid,
        SceSize args,
        const void *argp,
        SceUInt32 flags,
        const SceKernelStopModuleOpt *pOpt,
        int *pRes
);
```

Arguments

uid	PRX identifier
args	Argument block size
argp	Pointer to the argument block
flags	Unused (Always specify 0)
pOpt	Unused (Always specify `SCE_NULL`)
pRes	Pointer to the area storing the return value of `module_stop()`

Return Values

Value	Description
`SCE_OK`	Success
(<0)	Error code

Description

This performs stopping processing of the PRX specified with the *uid* argument.

Unlike with `sceKernelStopUnloadModule()`, This function only stop PRX and does not unload it.

To unload the PRX that has been stopped, it is necessary to call `sceKernelUnloadModule()`.

The contents of stop processing are the same as with `sceKernelStopUnloadModule()`.

sceKernelUnloadModule

Unload PRX

Definition

```
#include <kernel.h>
int sceKernelUnloadModule(
        SceUID uid,
        SceUInt32 flags,
        const SceKernelUnloadModuleOpt *pOpt,
);
```

Arguments

uid	PRX identifier
flags	Unused (Always specify 0)
pOpt	Unused (Always specify SCE_NULL)

Return Values

Value	Description
SCE_OK	Success
(<0)	Error code

Description

This performs unloading of the PRX specified with the *uid* argument.

It is not possible to unload a PRX that has undergone start processing without stopping it with sceKernelStopModule().

Process Manager

Variables

sceUserMainThreadName

Specifying the Name of the User Main Thread

Definition

```
#include <kernel.h>
extern const char sceUserMainThreadName[] __attribute__ ((weak));
```

Description

Specify a string with the length in `SCE_UID_NAMELEN` as the maximum length for the name of the user main thread. If the name is not specified, the module name is used.

sceUserMainThreadPriority

Specifying the Scheduling Priority of the User Main Thread

Definition

```
#include <kernel.h>
extern int sceUserMainThreadPriority __attribute__ ((weak));
```

Description

Specify the scheduling priority of the user main thread. If the scheduling priority is not specified, `SCE_KERNEL_DEFAULT_PRIORITY_USER` is used.

sceUserMainThreadStackSize

Specifying the Stack Size of the User Main Thread

Definition

```
#include <kernel.h>
extern unsigned int sceUserMainThreadStackSize __attribute__ ((weak));
```

Description

Specify the stack size of the user main thread by the number of bytes. If the stack size is not specified, `SCE_KERNEL_THREAD_STACK_SIZE_DEFAULT_USER_MAIN` is used.

sceUserMainThreadAttribute

Attribute of user main thread

Definition

```
#include <kernel.h>
extern unsigned int  sceUserMainThreadAttribute __attribute__ ((weak));
```

Description

Specify the attribute of user main thread (in current specifications, there are no thread attributes to specify).

sceUserMainThreadCpuAffinityMask

CPU affinity mask of user main thread

Definition

```
#include <kernel.h>
extern unsigned int  sceUserMainThreadCpuAffinityMask __attribute__
((weak));
```

Description

Specify the CPU affinity mask of user main thread. If the CPU affinity mask is not specified, `SCE_KERNEL_THREAD_CPU_AFFINITY_MASK_DEFAULT` is used.

sceUserMainThreadOptParam

Additional data for user main thread

Definition

```
#include <kernel.h>
extern const SceKernelThreadOptParam sceUserMainThreadOptParam __attribute__
((weak));
```

Description

This specifies the additional data given for generating the user main thread. (Currently there is no additional data that needs to be specified in particular.)

Functions

sceKernelExitProcess

Exit current process

Definition

```
#include <kernel.h>
int sceKernelExitProcess(
        SceInt32 res
);
```

Arguments

res Process exit code

Return Values

Value	Description
SCE_OK	Success
<0	Error code

Description

This exits the current process.

sceKernelGetProcessTime

Get process time of current process

Definition

```
#include <kernel.h>
SceInt32 sceKernelGetProcessTime(
        SceKernelSysClock *pClock
);
```

Arguments

pClock Specify the pointer to the structure variable receiving the process time of the current process.

Return Values

Value	Description
`SCE_OK`	Success
Negative value	Error code

Description

This obtains the process time elapsed from the start of the process (process time). Process time stops during process suspension and system suspension periods. Process time is counted in microseconds.

sceKernelGetProcessTimeLow

Get lower 32 bit part of process time of current process

Definition

```
#include <kernel.h>
SceUInt32 sceKernelGetProcessTimeLow(
        void
);
```

Arguments

None

Return Values

Lower 32 bit of process time of the current process

Description

This obtains the lower 32 bits of the time elapsed from the start of the process (process time). Process time stops during process suspension and system suspension periods. Process time is counted in microseconds.

sceKernelGetProcessTimeWide

Get process time of current process in 64 bits wide

Definition

```
#include <kernel.h>
SceUInt64 sceKernelGetProcessTimeWide(
        void
);
```

Arguments

None

Return Values

Process time of the current process

Description

This obtains the process time elapsed from the start of the process (process time). Process time stops during process suspension and system suspension periods. Process time is counted in microseconds.

It is different from `sceKernelGetProcessTime()` in that it returns the result directly as a `SceUInt64` value.

Back Trace

Document serial number: 000004892117

Datatypes

SceKernelCallFrame

Structure representing one frame of call stack

Definition

```
#include <kernel.h>
typedef struct _SceKernelCallFrame {
        SceUIntVAddr sp;
        SceUIntVAddr pc;
} SceKernelCallFrame;
```

Members

sp stack pointer
pc program counter

Description

This structure represents one frame of the call stack.

Functions

sceKernelBacktrace

Get backtrace

Definition

```
#include <kernel.h>
SceInt32 sceKernelBacktrace(
        SceUID threadId,
        SceKernelCallFrame *pCallFrameBuffer,
        SceSize numBytesBuffer,
        SceUInt32 *pNumReturn,
        SceInt32 mode
);
```

Arguments

threadId	Specify the identifier of the thread for which back trace is to be obtained or following macro: - `SCE_KERNEL_BACKTRACE_CONTEXT_CURRENT`: get the backtrace of the current context
pCallFrameBuffer	Buffer for obtaining call frame
numBytesBuffer	Buffer size (byte)
pNumReturn	Pointer receiving the actual number of obtained frames
mode	Specify the following macro: - `SCE_KERNEL_BACKTRACE_MODE_USER`: get the backtrace of the user stack The following option can be specified with logical OR - `SCE_KERNEL_BACKTRACE_MODE_DONT_EXCEED`: don't get the depth of the call stack

Return Values

Value	Description
Positive value	Call stack depth, or `SCE_OK`
Negative value	Error code

Description

This function obtains the backtrace.

When this function is called with *pCallFrameBuffer* = NULL, *numBytesBuffer* = 0 and *pNumReturn* = NULL, the depth of the call stack can only be obtained as a return value.

If `SCE_KERNEL_BACKTRACE_MODE_DONT_EXCEED` is specified in *mode*, back tracing of a depth greater than can be accommodated by the given buffer size is not performed internally, so the processing may be faster. In this case, the return value when successful is `SCE_OK` and not the depth of the call stack.

The lr register values that indicate the return addresses for functions are stored in the obtained *pc* values in each call frame. In the ARM specifications, the least significant bit of the jump target address will indicate the CPU operation mode of the corresponding function, so the least significant bit of the *pc* value will be 1 when the function is in the thumb mode. Therefore, mask the least significant bit of the *pc* value of the obtained call frame when the operation mode information is not required.

This function is only for debugging programs, and therefore can be used only in **Development Mode** of the Development Kit.

If this function is used for another purpose, the `SCE_KERNEL_ERROR_ILLEGAL_PERMISSION` error will be returned.

Document serial number: 000004892117

Document serial number: 000004892117

sceKernelBacktraceSelf

Get backtrace of calling thread

Definition

```
#include <kernel.h>
SceInt32 sceKernelBacktraceSelf(
        SceKernelCallFrame *pCallFrameBuffer,
        SceSize numBytesBuffer,
        SceUInt32 *pNumReturn,
        SceInt32 mode
);
```

Arguments

pCallFrameBuffer	Buffer for obtaining call frames
numBytesBuffer	Buffer size (byte)
pNumReturn	Pointer for receiving the number of frames obtained
mode	Operation mode

Return Values

Value	Description
Positive value	Call stack depth, or `SCE_OK`
Negative value	Error code

Description

This obtains the backtrace of the thread that has called `sceKernelBacktraceSelf()`. The backtrace can be obtained with a shorter calling time than with `sceKernelBacktrace()`, but it is not possible to obtain backtraces for threads other than the calling thread

When this function is called with *pCallFrameBuffer* = NULL and *numBytesBuffer* = 0, the depth of the call stack can only be obtained as a return value.

The lr register values that indicate the return addresses for functions are stored in the obtained *pc* values in each call frame. In the ARM specifications, the least significant bit of the jump target address will indicate the CPU operation mode of the corresponding function, so the least significant bit of the *pc* value will be 1 when the function is in the thumb mode. Therefore, mask the least significant bit of the *pc* value of the obtained call frame when the operation mode information is not required.

This function is only for debugging programs, and therefore can be used only in **Development Mode** of the Development Kit.

If this function is used for another purpose, the `SCE_KERNEL_ERROR_ILLEGAL_PERMISSION` error will be returned.

sceKernelPrintBacktrace

Display backtrace

Definition

```
#include <kernel.h>
SceInt32 sceKernelPrintBacktrace(
        const SceKernelCallFrame *pCallFrame,
        SceUInt32 numFrames
);
```

Arguments

pCallFrame	Specify the stack frame data
numFrames	Specify the number of frames of data to be specified in *pCallFrame*

Return Values

Value	Description
SCE_OK	Success
<0	Error code

Description

This function displays the backtrace.

This function is only for debugging programs, and therefore can be used only in **Development Mode** of the Development Kit.

If this function is used for another purpose, the SCE_KERNEL_ERROR_ILLEGAL_PERMISSION error will be returned.

Kernel Service Library

Datatypes

SceKernelSysClock

System clock structure

Definition

```
#include <kernel.h>
typedef union _SceKernelSysClock {
        struct {
              SceUInt32 low;
              SceUInt32 hi;
        } u;
        SceUInt64 quad;
} SceKernelSysClock;
```

Members

low	Lower 32 bits of the system time
hi	Higher 32 bits of the system time
u	System time
quad	System time

Description

This structure stores the system time.

Functions

sceKernelGetThreadIdUserRW

Get user read/write-enabled thread-specific area

Definition

```
#include <kernel.h>
SceUInt32 sceKernelGetThreadIdUserRW(
        void
);
```

Arguments

None

Return Values

Value	Description
32-bit unsigned integer	Thread-specific area

Description

This gets the user read/write-enabled thread-specific area.

sceKernelSetThreadIdUserRW

Set user read/write-enabled thread-specific area

Definition

```
#include <kernel.h>
SceUInt32 sceKernelSetThreadIdUserRW(
        SceUInt32 v
);
```

Arguments

v Value

Return Values

Value	Description
32-bit unsigned integer	Set value

Description

This sets the user read/write-enabled thread-specific area.

sceKernelSetGPO

Output to LED

Definition

```
#include <kernel.h>
SceInt32 sceKernelSetGPO(
        SceUInt32 uiBits
);
```

Arguments

uiBits Bit pattern. only low-order 8 bits are valid.

Return Values

Value	Description
SCE_OK	Success

Description

This sets the content displayed on the LED.

sceKernelCheckDipsw

Check details of DIPSW

Definition

```
#include <kernel.h>
int sceKernelCheckDipsw(
        int no
);
```

Arguments

no DIPSW number

Return Values

Value	Description
0 or 1	DIPSW number

Description

This checks the details of DIPSW.

Notes

When developing a new program, use a GPI switch to which a value can be set from the development host computer.

See Also

`sceKernelGetGPI()`

sceKernelSetDipsw

Set DIPSW

Definition

```
#include <kernel.h>
void sceKernelSetDipsw(
        int no
);
```

Arguments

no DIPSW number

Return Values

None

Description

This sets DIPSW.

Notes

When developing a new program, use a GPI switch to which a value can be set from the development host computer.

See Also

`sceKernelGetGPI()`

sceKernelClearDipsw

Clear DIPSW

Definition

```
#include <kernel.h>
void sceKernelClearDipsw(
        int no
);
```

Arguments

no DIPSW number

Return Values

None

Description

This clears DIPSW

Notes

When developing a new program, use a GPI switch to which a value can be set from the development host computer.

See Also

`sceKernelGetGPI()`

sceKernelGetGPI

Get value of the GPI switch

Definition

```
#include <kernel.h>
SceUInt32 sceKernelGetGPI(
        void
);
```

Arguments

None

Return Values

Value	Description
32 bit	Value of GPI switch

Description

This gets the value of the GPI switch set from the development host computer. Note that the set value can be obtained only when **Release Check Mode** of the Development Kit is set to **Development Mode**. Otherwise, always 0 will return.

sceClibMemset

Embed the memory area with specific value

Definition

```
#include <kernel.h>
SCE_CDECL_BEGIN void *sceClibMemset(
        void *dst,
        int c,
        SceSize n
);
```

Arguments

dst	Start of memory area
c	Value
n	Size

Return Values

Returns `dst`.

Description

This embeds the memory area specified with *dst* and *n* in byte value *c*.

Constants

Definition List

Definition	Value	Description
SCE_ARM_L1_DCACHE_LINE_SIZE	32	Size of L1 data cache
SCE_ARM_L1_ICACHE_LINE_SIZE	32	Size of L1 command cache
SCE_ARM_L1_WAYS	4	Number of L1 cache ways
SCE_ARM_L2_CACHE_LINE_SIZE	32	Size of L2 cache
SCE_ARM_L2_CACHE_MAX_FILL_SIZE	64	Max fill size of L2 cache
SCE_CDECL_BEGIN	N/A	Start macro for referencing C header from C++
SCE_CDECL_END	N/A	End macro for referencing C header from C++
SCE_KERNEL_BACKTRACE_CONTEXT_CURRENT	(0x00000000)	Back trace the current context
SCE_KERNEL_BACKTRACE_MODE_USER	(0x00000000)	Backtrace for a user stack
SCE_KERNEL_BACKTRACE_MODE_DONT_EXCEED	(0x00000002)	Do not obtain a stack depth
SCE_KERNEL_PROCESS_ID_SELF	0	Process identifier currently being executed
SCE_KERNEL_4KiB	0x00001000	Value of 4 KiB
SCE_KERNEL_64KiB	0x00010000	Value of 64 KiB
SCE_KERNEL_256KiB	0x00040000	Value of 256 KiB
SCE_KERNEL_1MiB	0x00100000	Value of 1 MiB
SCE_KERNEL_16MiB	0x01000000	Value of 16 MiB
SCE_UID_NAMELEN	(31)	Maximum size of a name that can be given to a kernel resource
SCE_UID_ATTR_OPEN_FLAG	0x00080000U	Flag indicating that the identifier was obtained from a reference API
SCE_KERNEL_MAX_CPU	(4)	Number of CPUs supported by the kernel
SCE_KERNEL_CPU_MASK_SHIFT	(16)	Shift value for user processor number of CPU affinity mask
SCE_KERNEL_CPU_MASK_USER_0	(0x01 << SCE_KERNEL_CPU_MASK_SHIFT)	Processor number 0 for user game
SCE_KERNEL_CPU_MASK_USER_1	(0x02 << SCE_KERNEL_CPU_MASK_SHIFT)	Processor number 1 for user game
SCE_KERNEL_CPU_MASK_USER_2	(0x04 << SCE_KERNEL_CPU_MASK_SHIFT)	Processor number 2 for user game
SCE_KERNEL_CPU_MASK_USER_ALL	(SCE_KERNEL_CPU_MASK_USER_0 | SCE_KERNEL_CPU_MASK_USER_1 | SCE_KERNEL_CPU_MASK_USER_2)	Mask setting of CPU for user game
SCE_DIPSW_USER_MIN	0	Minimum value of DIPSW available to users
SCE_DIPSW_USER_MAX	63	Maximum value of DIPSW available to users
SCE_KERNEL_OK	SCE_OK	OK
SCE_KERNEL_START_SUCCESS	(0)	Module common macro Start success (resident)

Document serial number: 000004892117

Definition	Value	Description
SCE_KERNEL_START_RESIDENT	SCE_KERNEL_START_SUCCESS	Start success (resident)
SCE_KERNEL_START_NO_RESIDENT	(1)	Start success (no resident)
SCE_KERNEL_START_FAILED	(2)	Start failed
SCE_KERNEL_STOP_SUCCESS	(0)	Stop
SCE_KERNEL_STOP_FAIL	(1)	Do not stop
SCE_KERNEL_STOP_CANCEL	SCE_KERNEL_STOP_FAIL	Do not stop
SCE_MODULE_ATTR_NONE	(0x0000)	Attribute not specified
SCE_KERNEL_PROCESS_PRIORITY_SYSTEM_HIGH	32	System highest process priority
SCE_KERNEL_PROCESS_PRIORITY_USER_HIGH	64	User highest process priority
SCE_KERNEL_PROCESS_PRIORITY_USER_DEFAULT	96	User standard process priority
SCE_KERNEL_PROCESS_PRIORITY_SYSTEM_DEFAULT	96	System standard process priority
SCE_KERNEL_PROCESS_PRIORITY_USER_LOW	127	User lowest process priority
SCE_KERNEL_PROCESS_PRIORITY_SYSTEM_LOW	159	System lowest process priority
SCE_KERNEL_POWER_TICK_DEFAULT	0x00	Cancel idle state timer. Cancel idle state timer and send notification of no-idle state.
SCE_KERNEL_MEMBLOCK_TYPE_USER_RW	0x0c20d060U	Type for allocating readable/writable memory blocks in a user space
SCE_KERNEL_MEMBLOCK_TYPE_USER_RW_UNCACHE	0x0c208060U	Type for allocating memory blocks, which can be read/written and which do not use cache, on LPDDR2 in a user space
SCE_KERNEL_MEMBLOCK_TYPE_USER_MAIN_PHYCONT_RW	0x0c80d060U	Type for allocating readable/writable memory blocks from a physical continuous memory area in a user space
SCE_KERNEL_MEMBLOCK_TYPE_USER_MAIN_PHYCONT_NC_RW	0x0d808060U	Type for allocating memory blocks which are uncached and can be read/written from a physical continuous memory area in a user space
SCE_KERNEL_MEMBLOCK_TYPE_USER_CDRAM_RW	0x09408060U	Type for allocating readable/writable memory blocks on CDRAM in a user space
SCE_KERNEL_ALLOC_MEMBLOCK_ATTR_HAS_ALIGNMENT	0x00000004U	Alignment members are enabled.
SCE_KERNEL_MEMORY_ACCESS_R	0x04U	Readable memory
SCE_KERNEL_MEMORY_ACCESS_W	0x02U	Writable memory
SCE_KERNEL_MEMORY_TYPE_NORMAL_NC	0x80	Conventional uncached memory
SCE_KERNEL_MEMORY_TYPE_NORMAL	0xD0	Conventional cached memory
SCE_KERNEL_HIGHEST_PRIORITY_USER	(64)	Highest priority

Definition	Value	Description
SCE_KERNEL_LOWEST_PRIORITY_USER	(191)	Lowest priority
SCE_KERNEL_DEFAULT_PRIORITY	((SceInt32)0x10000100)	Default priority (entire system)
SCE_KERNEL_INDIVIDUAL_QUEUE_HIGHEST_PRIORITY	(64)	Highest priority of individual queue area
SCE_KERNEL_INDIVIDUAL_QUEUE_LOWEST_PRIORITY	(127)	Lowest priority of individual queue area
SCE_KERNEL_COMMON_QUEUE_HIGHEST_PRIORITY	(128)	Highest priority of common queue area
SCE_KERNEL_COMMON_QUEUE_LOWEST_PRIORITY	(191)	Lowest priority of common queue area
SCE_KERNEL_DEFAULT_PRIORITY_USER	SCE_KERNEL_DEFAULT_PRIORITY	Default priority (for users). The default priority for game applications is changed to 160 internally. The specification method "default value ± offset (SCE_KERNEL_DEFAULT_PRIORITY_USER - 10)" can also be used.
SCE_KERNEL_CURRENT_THREAD_PRIORITY	(0)	The priority of the thread currently being executed
SCE_KERNEL_THREAD_STACK_SIZE_MAX	SCE_KERNEL_32MiB	Maximum stack size
SCE_KERNEL_THREAD_STACK_SIZE_MIN	SCE_KERNEL_4KiB	Minimum stack size
SCE_KERNEL_THREAD_STACK_SIZE_DEFAULT	SCE_KERNEL_THREAD_STACK_SIZE_MIN	Default stack size
SCE_KERNEL_THREAD_STACK_SIZE_DEFAULT_USER_MAIN	SCE_KERNEL_256KiB	Default stack size of user main thread
SCE_KERNEL_THREAD_STATUS_RUNNING	(0x00000001U)	RUNNING state
SCE_KERNEL_THREAD_STATUS_READY	(0x00000002U)	READY state
SCE_KERNEL_THREAD_STATUS_STANDBY	(0x00000004U)	STANDBY state
SCE_KERNEL_THREAD_STATUS_WAITING	(0x00000008U)	WAITING state
SCE_KERNEL_THREAD_STATUS_DORMANT	(0x00000010U)	DORMANT state
SCE_KERNEL_THREAD_STATUS_DELETED	(0x00000020U)	DELETED state
SCE_KERNEL_THREAD_STATUS_DEAD	(0x00000040U)	DEAD state
SCE_KERNEL_THREAD_STATUS_STAGNANT	(0x00000080U)	STAGNANT state
SCE_KERNEL_THREAD_STATUS_SUSPENDED	(0x00000100U)	SUSPENDED state
SCE_KERNEL_THREAD_STATUS_MASK	(0x0000ffffU)	Valid thread state mask
SCE_KERNEL_WAITTYPE_DELAY	(0x00000001U)	Thread delay
SCE_KERNEL_WAITTYPE_WAITTHEND	(0x00000002U)	Thread termination
SCE_KERNEL_WAITTYPE_SIGNAL	(0x00000004U)	Signal
SCE_KERNEL_WAITTYPE_WAITTHSUSPEND	(0x00000008U)	Thread suspend
SCE_KERNEL_WAITTYPE_EVENTFLAG	(0x00000010U)	Event flag
SCE_KERNEL_WAITTYPE_SEMAPHORE	(0x00000020U)	Semaphore
SCE_KERNEL_WAITTYPE_MUTEX	(0x00000040U)	Mutex
SCE_KERNEL_WAITTYPE_RW_LOCK	(0x00000080U)	Reader/writer lock

Definition	Value	Description
SCE_KERNEL_WAITTYPE_COND_SIGNAL	(0x00000100U)	Condition variable (signal)
SCE_KERNEL_WAITTYPE_COND_MUTEX	(0x00000200U)	Condition variable (mutex)
SCE_KERNEL_WAITTYPE_FAST_MUTEX	(0x00001000U)	Fast mutex
SCE_KERNEL_WAITTYPE_FAST_MUTEX_SPIN	(0x00002000U)	Fast mutex (spin)
SCE_KERNEL_WAITTYPE_EVENT	(0x00010000U)	Single event
SCE_KERNEL_WAITTYPE_MP_EVENTS	(0x00020000U)	Multiple event
SCE_KERNEL_WAITTYPE_MSG_PIPE	(0x00040000U)	Message pipe
SCE_KERNEL_WAITTYPE_LW_MUTEX	(0x00100000U)	Lightweight mutex
SCE_KERNEL_WAITTYPE_LW_COND_SIGNAL	(0x00200000U)	Lightweight condition variable (signal)
SCE_KERNEL_WAITTYPE_LW_COND_LW_MUTEX	(0x00400000U)	Lightweight condition variable (Lightweight mutex)
SCE_KERNEL_WAITTYPE_DELAY_CB	(0x80000001U)	Thread delay with callback check
SCE_KERNEL_WAITTYPE_WAITTHEND_CB	(0x80000002U)	Thread end with callback check
SCE_KERNEL_WAITTYPE_SIGNAL_CB	(0x80000004U)	Signal with callback check
SCE_KERNEL_WAITTYPE_WAITTHSUSPEND_CB	(0x80000008U)	Thread suspend with callback check
SCE_KERNEL_WAITTYPE_EVENTFLAG_CB	(0x80000010U)	Event flag with callback check
SCE_KERNEL_WAITTYPE_SEMAPHORE_CB	(0x80000020U)	Semaphore with callback check
SCE_KERNEL_WAITTYPE_MUTEX_CB	(0x80000040U)	Mutex with callback check
SCE_KERNEL_WAITTYPE_RW_LOCK_CB	(0x80000080U)	Reader/writer lock with callback check
SCE_KERNEL_WAITTYPE_COND_SIGNAL_CB	(0x80000100U)	Condition variable (signal) with callback check
SCE_KERNEL_WAITTYPE_COND_MUTEX_CB	(0x80000200U)	Condition variable (mutex) with callback check
SCE_KERNEL_WAITTYPE_FAST_MUTEX_CB	(0x80001000U)	Fast mutex with callback check
SCE_KERNEL_WAITTYPE_FAST_MUTEX_SPIN_CB	(0x80002000U)	Fast mutex (spin) with callback check
SCE_KERNEL_WAITTYPE_EVENT_CB	(0x80010000U)	Single event with callback check
SCE_KERNEL_WAITTYPE_MP_EVENTS_CB	(0x80020000U)	Multiple event with callback check
SCE_KERNEL_WAITTYPE_MSG_PIPE_CB	(0x80040000U)	Message pipe with callback check
SCE_KERNEL_WAITTYPE_LW_MUTEX_CB	(0x80100000U)	Lightweight mutex with callback check
SCE_KERNEL_WAITTYPE_LW_COND_SIGNAL_CB	(0x80200000U)	Lightweight condition variable (signal) with callback check
SCE_KERNEL_WAITTYPE_LW_COND_LW_MUTEX_CB	(0x80400000U)	Lightweight condition variable (lightweight mutex) with callback check
SCE_KERNEL_THREAD_ID_SELF	(0)	UID of this thread
SCE_KERNEL_THREADMGR_UID_CLASS_THREAD	(1)	Thread class
SCE_KERNEL_THREADMGR_UID_CLASS_SEMA	(2)	Semaphore class

Definition	Value	Description
SCE_KERNEL_THREADMGR_UID_ CLASS_EVENT_FLAG	(3)	Event flag class
SCE_KERNEL_THREADMGR_UID_ CLASS_MUTEX	(4)	Mutex class
SCE_KERNEL_THREADMGR_UID_ CLASS_COND	(5)	Condition variable class
SCE_KERNEL_THREADMGR_UID_ CLASS_TIMER	(6)	Timer class
SCE_KERNEL_THREADMGR_UID_ CLASS_MSG_PIPE	(7)	Message pipe class
SCE_KERNEL_THREADMGR_UID_ CLASS_CALLBACK	(8)	Callback class
SCE_KERNEL_THREADMGR_UID_ CLASS_THREAD_EVENT	(9)	Thread event class
SCE_KERNEL_THREADMGR_UID_ CLASS_LW_MUTEX	(10)	Lightweight mutex class
SCE_KERNEL_THREADMGR_UID_ CLASS_LW_COND	(11)	Lightweight condition variable class
SCE_KERNEL_THREADMGR_UID_ CLASS_RW_LOCK	(12)	Reader/writer lock class
SCE_KERNEL_THREADMGR_UID_ CLASS_SIMPLE_EVENT	(13)	Simple event class
SCE_KERNEL_THREAD_CPU_ AFFINITY_MASK_DEFAULT	(0)	Default CPU affinity mask. When the default CPU affinity mask is specified to a thread, the CPU affinity mask of the process to which it belongs is used.
SCE_KERNEL_VFP_EXCEPTION_MASK_ QCE	(0x08000000U)	QCE flag interrupt enable
SCE_KERNEL_VFP_EXCEPTION_MASK_ IDE	(0x00000080U)	IDE flag interrupt enable
SCE_KERNEL_VFP_EXCEPTION_MASK_ IXE	(0x00000010U)	IXE flag interrupt enable
SCE_KERNEL_VFP_EXCEPTION_MASK_ UFE	(0x00000008U)	UFE flag interrupt enable
SCE_KERNEL_VFP_EXCEPTION_MASK_ OFE	(0x00000004U)	OFE flag interrupt enable
SCE_KERNEL_VFP_EXCEPTION_MASK_ DZE	(0x00000002U)	DZE flag interrupt enable
SCE_KERNEL_VFP_EXCEPTION_MASK_ IOE	(0x00000001U)	IOE flag interrupt enable
SCE_KERNEL_VFP_EXCEPTION_MASK_ ALL	(SCE_KERNEL_VFP_EXCEPTION_ MASK_QCE | SCE_KERNEL_VFP_EXCEPTION_ MASK_IDE | \ SCE_KERNEL_VFP_ EXCEPTION_MASK_IXE | SCE_KERNEL_VFP_EXCEPTION_ MASK_UFE | \ SCE_KERNEL_VFP_ EXCEPTION_MASK_OFE | SCE_KERNEL_VFP_EXCEPTION_ MASK_DZE | \ SCE_KERNEL_VFP_ EXCEPTION_MASK_IOE)	Valid VFP exception mask

Definition	Value	Description
SCE_KERNEL_ATTR_SINGLE	(0x00000000U)	Multiple threads cannot wait at the same time (Only event flags)
SCE_KERNEL_ATTR_MULTI	(0x00001000U)	Multiple threads cannot wait at the same time (Only event flags)
SCE_KERNEL_ATTR_TH_FIFO	(0x00000000U)	Queuing of waiting threads is in FIFO order
SCE_KERNEL_ATTR_TH_PRIO	(0x00002000U)	Queuing of waiting threads is in order of their thread priority.
SCE_KERNEL_ATTR_MS_FIFO	(0x00000000U)	Unused
SCE_KERNEL_ATTR_MS_PRIO	(0x00000000U)	Unused
SCE_KERNEL_THREAD_EVENT_TYPE_START	(0x04)	Thread startup event
SCE_KERNEL_THREAD_EVENT_TYPE_EXIT	(0x08)	Thread termination event
SCE_KERNEL_THREAD_EVENT_TYPE_ALL	(SCE_KERNEL_THREAD_EVENT_TYPE_START | SCE_KERNEL_THREAD_EVENT_TYPE_EXIT)	Valid thread event mask
SCE_KERNEL_THREAD_ID_USER	((SceUID)0xfffffff0)	Specified all user threads
SCE_KERNEL_EVENT_ATTR_MANUAL_RESET	(0x00000000U)	The event notification state continues until each bit is transitioned to the event non-notification state manually.
SCE_KERNEL_EVENT_ATTR_AUTO_RESET	(0x00000100U)	When each bit wakes up one waiting thread, the state returns to event non-notification state automatically.
SCE_KERNEL_ATTR_NOTIFY_CB_ALL	(0x00000000U)	All the callbacks registered in an event object at the time of event notification are notified
SCE_KERNEL_ATTR_NOTIFY_CB_WAKEUP_ONLY	(0x00000800U)	Of callbacks registered in an event object at the time of even notification, only the callbacks of threads waiting for the event to awaken are notified.
SCE_KERNEL_EVENT_IN	(0x00000001U)	Input event
SCE_KERNEL_EVENT_OUT	(0x00000002U)	Output event
SCE_KERNEL_EVENT_CREATE	(0x00000004U)	Create event
SCE_KERNEL_EVENT_DELETE	(0x00000008U)	Delete event
SCE_KERNEL_EVENT_OPEN	(0x00000100U)	Reference event
SCE_KERNEL_EVENT_CLOSE	(0x00000200U)	Terminate reference event
SCE_KERNEL_EVENT_TIMER	(0x00008000U)	Timer event
SCE_KERNEL_EVENT_ERROR	(0x00000010U)	Error event
SCE_KERNEL_EVENT_DATA_EXIST	(0x00010000U)	Event notified when the message pipe's buffer contains data (level-triggered)

Definition	Value	Description
SCE_KERNEL_EVENT_WAIT_MODE_OR	(0x00000001U)	Wait until one waiting event specified in the event list is completed
SCE_KERNEL_EVENT_WAIT_MODE_AND	(0x00000002U)	Wait until all waiting events specified in the event list are completed
SCE_KERNEL_EVF_ATTR_TH_FIFO	SCE_KERNEL_ATTR_TH_FIFO	Queuing of threads waiting for an event flag is in FIFO order
SCE_KERNEL_EVF_ATTR_TH_PRIO	SCE_KERNEL_ATTR_TH_PRIO	Queuing of threads waiting for an event flag is in order of their thread priority
SCE_KERNEL_EVF_ATTR_SINGLE	SCE_KERNEL_ATTR_SINGLE	Multiple threads cannot wait at the same time
SCE_KERNEL_EVF_ATTR_MULTI	SCE_KERNEL_ATTR_MULTI	Multiple threads can wait at the same time.
SCE_KERNEL_EVF_WAITMODE_AND	(0x00000000U)	AND wait
SCE_KERNEL_EVF_WAITMODE_OR	(0x00000001U)	OR wait
SCE_KERNEL_EVF_WAITMODE_CLEAR_ALL	(0x00000002U)	After waiting events are completed, all bits are cleared
SCE_KERNEL_EVF_WAITMODE_CLEAR_PAT	(0x00000004U)	After waiting events are completed, bits specified for *bitPattern* are cleared
SCE_KERNEL_SEMA_ATTR_TH_FIFO	SCE_KERNEL_ATTR_TH_FIFO	Queuing of threads waiting for a semaphore is in FIFO order
SCE_KERNEL_SEMA_ATTR_TH_PRIO	SCE_KERNEL_ATTR_TH_PRIO	Queuing of threads waiting for a semaphore is in order of their thread priority.
SCE_KERNEL_MUTEX_ATTR_TH_FIFO	SCE_KERNEL_ATTR_TH_FIFO	Queuing of threads waiting for a mutex is in FIFO order
SCE_KERNEL_MUTEX_ATTR_TH_PRIO	SCE_KERNEL_ATTR_TH_PRIO	Queuing of threads waiting for a mutex is in order of their thread priority
SCE_KERNEL_MUTEX_ATTR_RECURSIVE	(0x00000002U)	Mutex allows recursive locks
SCE_KERNEL_MUTEX_ATTR_CEILING	(0x00000004U)	The priority ceiling feature of mutex is used
SCE_KERNEL_LW_MUTEX_ATTR_TH_FIFO	SCE_KERNEL_ATTR_TH_FIFO	Queuing of threads waiting for a lightweight mutex is in FIFO order
SCE_KERNEL_LW_MUTEX_ATTR_TH_PRIO	SCE_KERNEL_ATTR_TH_PRIO	Queuing of threads waiting for a lightweight mutex is in order of their thread priority
SCE_KERNEL_LW_MUTEX_ATTR_RECURSIVE	(0x00000002U)	Lightweight mutex allows recursive locks
SCE_KERNEL_COND_ATTR_TH_FIFO	SCE_KERNEL_ATTR_TH_FIFO	Queuing of threads waiting for a condition variable is in FIFO order
SCE_KERNEL_COND_ATTR_TH_PRIO	SCE_KERNEL_ATTR_TH_PRIO	Queuing of threads waiting for a condition variable is in order of their thread priority

Definition	Value	Description
SCE_KERNEL_LW_COND_ATTR_TH_FIFO	SCE_KERNEL_ATTR_TH_FIFO	Queuing of threads waiting for a lightweight condition variable is in FIFO order.
SCE_KERNEL_LW_COND_ATTR_TH_PRIO	SCE_KERNEL_ATTR_TH_PRIO	Queuing of threads waiting for a lightweight condition variable is in order of their thread priority
SCE_KERNEL_TIMER_TYPE_SET_EVENT	(0)	Timer uses a set-type event notification
SCE_KERNEL_TIMER_TYPE_PULSE_EVENT	(1)	Timer uses a pulse-type event notification
SCE_KERNEL_RW_LOCK_ATTR_TH_FIFO	SCE_KERNEL_ATTR_TH_FIFO	Queuing of threads waiting for a reader/writer lock is in FIFO order
SCE_KERNEL_RW_LOCK_ATTR_TH_PRIO	SCE_KERNEL_ATTR_TH_PRIO	Queuing of threads waiting for a reader/writer lock is in order of their thread priority
SCE_KERNEL_RW_LOCK_ATTR_RECURSIVE	(0x00000002U)	Reader/writer lock allows recursive locks
SCE_KERNEL_RW_LOCK_ATTR_CEILING	(0x00000004U)	The priority ceiling feature is used at the time of the write lock of the reader/writer lock. (Unimplemented)
SCE_KERNEL_RW_LOCK_CANCEL_WITH_WRITE_LOCK	(1)	After canceling the reader/writer lock, obtain the write lock to atomic.
SCE_KERNEL_MSG_PIPE_ATTR_TH_PRIO_S	0x00000001U	Queuing of waiting threads on the message transmission side is in the FIFO order
SCE_KERNEL_MSG_PIPE_ATTR_TH_PRIO_R	0x00000002U	Queuing of waiting threads on the message receiving side is in the FIFO order
SCE_KERNEL_MSG_PIPE_ATTR_TH_PRIO	SCE_KERNEL_MSG_PIPE_ATTR_TH_PRIO_S | SCE_KERNEL_MSG_PIPE_ATTR_TH_PRIO_R	Queuing of waiting threads on the message transmission/reception side is in the FIFO order
SCE_KERNEL_MSG_PIPE_ATTR_TH_FIFO_S	0x00000004U	Queuing of waiting threads on the message transmission side is in the thread priority order
SCE_KERNEL_MSG_PIPE_ATTR_TH_FIFO_R	0x00000008U	Queuing of waiting threads on the message reception side is in the thread priority order
SCE_KERNEL_MSG_PIPE_ATTR_TH_FIFO	SCE_KERNEL_MSG_PIPE_ATTR_TH_FIFO_S | SCE_KERNEL_MSG_PIPE_ATTR_TH_FIFO_R	Queuing of waiting threads on the message transmission/reception side is in the thread priority order
SCE_KERNEL_MSG_PIPE_MODE_ASAP	0x00000000U	Completion upon successful transmission/reception of even just 1 byte of data
SCE_KERNEL_MSG_PIPE_MODE_FULL	0x00000001U	Completion upon successful transmission/reception of all the data of the specified size

Definition	Value	Description
SCE_KERNEL_MSG_PIPE_MODE_WAIT	0x00000000U	Block thread until the transmission/reception processing has been completed (synchronous mode)
SCE_KERNEL_MSG_PIPE_MODE_DONT_WAIT	0x00000010U	Issue transmission/reception request and return immediately (asynchronous mode)
SCE_KERNEL_MSG_PIPE_MODE_DONT_REMOVE	0x00000100U	Don't delete the received data from the pipe (peak processing)
SCE_KERNEL_MSG_PIPE_TYPE_USER_MAIN	64	User main area
SCE_KERNEL_MSG_PIPE_TYPE_USER_CDRAM	66	User CDRAM area (not implemented)
SCE_KERNEL_MSG_PIPE_OPT_ATTR_OPEN_LIMITATION	0x00000100U	Specify *openLimit*
SCE_IO_MAX_PATH_BUFFER_SIZE	(1024)	Maximum buffer size which can be specified as a path of file/directory
SCE_IO_MAX_PATH_LENGTH	(200)	Maximum number of characters which can be specified as a path of file/directory (including termination characters)

Error Codes

Definition List

Definition	Value	Description
SCE_KERNEL_ERROR_ERROR	0x80020001	General kernel error code. Cannot be used.
SCE_KERNEL_ERROR_NOT_IMPLEMENTED	0x80020002	Not implemented
SCE_KERNEL_ERROR_NOSYS	0x80020003	Not implemented
SCE_KERNEL_ERROR_UNSUP	0x80020004	Feature is not supported
SCE_KERNEL_ERROR_INVALID_ARGUMENT	0x80020005	Provided argument is invalid
SCE_KERNEL_ERROR_ILLEGAL_ADDR	0x80020006	Incorrect address specification
SCE_KERNEL_ERROR_ILLEGAL_ALIGNMENT	0x80020007	Incorrect alignment
SCE_KERNEL_ERROR_ILLEGAL_PERMISSION	0x80020008	Permission error
SCE_KERNEL_ERROR_INVALID_ARGUMENT_SIZE	0x80020009	Size of provided argument is invalid
SCE_KERNEL_ERROR_INVALID_FLAGS	0x8002000A	Flag setting of provided argument is invalid
SCE_KERNEL_ERROR_ILLEGAL_SIZE	0x8002000B	Specified size is invalid
SCE_KERNEL_ERROR_ILLEGAL_TYPE	0x8002000C	Invalid type
SCE_KERNEL_ERROR_ILLEGAL_PATTERN	0x8002000D	Invalid pattern
SCE_KERNEL_ERROR_ILLEGAL_ATTR	0x8002000E	Invalid attribute
SCE_KERNEL_ERROR_ILLEGAL_COUNT	0x8002000F	Invalid count
SCE_KERNEL_ERROR_ILLEGAL_MODE	0x80020010	Invalid mode
SCE_KERNEL_ERROR_ILLEGAL_OPEN_LIMIT	0x80020011	Maximum number of simultaneously referenceable items is invalid
SCE_KERNEL_ERROR_DEBUG_ERROR	0x80021000	Error occurred with debug service
SCE_KERNEL_ERROR_ILLEGAL_DIPSW_NUMBER	0x80021001	Invalid value of DIPSW number
SCE_KERNEL_ERROR_CPU_ERROR	0x80022000	Error occurred with CPU service
SCE_KERNEL_ERROR_MMU_ILLEGAL_L1_TYPE	0x80022001	Value of page table entry of MMU Leve1l is invalid
SCE_KERNEL_ERROR_MMU_L2_INDEX_OVERFLOW	0x80022002	MMU Level2 page index overflow
SCE_KERNEL_ERROR_MMU_L2_SIZE_OVERFLOW	0x80022003	MMU Level2 page index size overflow
SCE_KERNEL_ERROR_INVALID_CPU_AFFINITY	0x80022004	Setting of processor number is invalid
SCE_KERNEL_ERROR_INVALID_MEMORY_ACCESS	0x80022005	Memory access error
SCE_KERNEL_ERROR_INVALID_MEMORY_ACCESS_ PERMISSION	0x80022006	Memory access permission error
SCE_KERNEL_ERROR_VA2PA_FAULT	0x80022007	VA2PA conversion failed
SCE_KERNEL_ERROR_VA2PA_MAPPED	0x80022008	Conversion of VA2PA not intended for mapping was successful
SCE_KERNEL_ERROR_VALIDATION_CHECK_FAILED	0x80022009	Error found in validation check after MMU update
SCE_KERNEL_ERROR_SYSMEM_ERROR	0x80024000	Memory manager error
SCE_KERNEL_ERROR_INVALID_PROCESS_CONTEXT	0x80024001	Invalid process context
SCE_KERNEL_ERROR_UID_NAME_TOO_LONG	0x80024002	UID name exceeds size limit
SCE_KERNEL_ERROR_VARANGE_IS_NOT_PHYSICAL_ CONTINUOUS	0x80024003	Virtual address range is not physical continuous memory
SCE_KERNEL_ERROR_PHYADDR_ERROR	0x80024100	Physical address management subsystem error

Definition	Value	Description
SCE_KERNEL_ERROR_NO_PHYADDR	0x80024101	Specified physical address is invalid
SCE_KERNEL_ERROR_PHYADDR_USED	0x80024102	Specified physical address is already in use
SCE_KERNEL_ERROR_PHYADDR_NOT_USED	0x80024103	Tried to release unused physical address
SCE_KERNEL_ERROR_NO_IOADDR	0x80024104	Specified address is not I/O address
SCE_KERNEL_ERROR_PHYMEM_ERROR	0x80024300	Physical memory manager error
SCE_KERNEL_ERROR_ILLEGAL_PHYPAGE_STATUS	0x80024301	Physical page status is invalid
SCE_KERNEL_ERROR_NO_FREE_PHYSICAL_PAGE	0x80024302	Cannot allocate physical page
SCE_KERNEL_ERROR_NO_FREE_PHYSICAL_PAGE_UNIT	0x80024303	Cannot allocate physical page
SCE_KERNEL_ERROR_PHYMEMPART_NOT_EMPTY	0x80024304	Physical memory partition to be deleted is not empty
SCE_KERNEL_ERROR_NO_PHYMEMPART_LPDDR2	0x80024305	Corresponding physical memory partition does not exist on LPDDR2
SCE_KERNEL_ERROR_NO_PHYMEMPART_CDRAM	0x80024306	Corresponding physical memory partition does not exist on CDRAM
SCE_KERNEL_ERROR_PHYMEMPART_OUT_OF_INDEX	0x80024307	Specified physical page is not included in the target partition
SCE_KERNEL_ERROR_FIXEDHEAP_ERROR	0x80024400	Error occurred with fixed size heap
SCE_KERNEL_ERROR_FIXEDHEAP_ILLEGAL_SIZE	0x80024401	Incorrect fixed heap size
SCE_KERNEL_ERROR_FIXEDHEAP_ILLEGAL_INDEX	0x80024402	Incorrect fixed heap index
SCE_KERNEL_ERROR_FIXEDHEAP_INDEX_OVERFLOW	0x80024403	Fixed heap index overflow
SCE_KERNEL_ERROR_FIXEDHEAP_NO_CHUNK	0x80024404	Error occurred with fixed heap
SCE_KERNEL_ERROR_UID_ERROR	0x80024500	Resource identifier manager error
SCE_KERNEL_ERROR_INVALID_UID	0x80024501	Format of resource identifier is invalid
SCE_KERNEL_ERROR_SYSMEM_UID_INVALID_ARGUMENT	0x80024502	Argument to resource identifier is invalid
SCE_KERNEL_ERROR_SYSMEM_INVALID_UID_RANGE	0x80024503	Resource identifier value out of range
SCE_KERNEL_ERROR_SYSMEM_NO_VALID_UID	0x80024504	Invalid resource identifier value
SCE_KERNEL_ERROR_SYSMEM_CANNOT_ALLOCATE_ UIDENTRY	0x80024505	Cannot allocate resource identifier
SCE_KERNEL_ERROR_NOT_PROCESS_UID	0x80024506	Specified user mode identifier is invalid
SCE_KERNEL_ERROR_NOT_KERNEL_UID	0x80024507	Specified kernel mode identifier is invalid
SCE_KERNEL_ERROR_INVALID_UID_CLASS	0x80024508	Not a class with a specified class corresponding to UID
SCE_KERNEL_ERROR_INVALID_UID_SUBCLASS	0x80024509	Not a subclass with a specified class corresponding to UID
SCE_KERNEL_ERROR_UID_CANNOT_FIND_BY_NAME	0x8002450A	Obtaining UID by name is failed
SCE_KERNEL_ERROR_UID_NOT_VISIBLE	0x8002450B	Cannot access with the specified visibility level
SCE_KERNEL_ERROR_UID_MAX_OPEN	0x8002450C	The number of opened UIDs exceeds the maximum number
SCE_KERNEL_ERROR_UID_RL_OVERFLOW	0x8002450D	The number of resources exceeds the specified limitation
SCE_KERNEL_ERROR_VIRPAGE_ERROR	0x80024600	Virtual page manager error

Definition	Value	Description
SCE_KERNEL_ERROR_ILLEGAL_VIRPAGE_TYPE	0x80024601	Invalid virtual page type
SCE_KERNEL_ERROR_BLOCK_ERROR	0x80024700	Error occurred within memory block subsystem
SCE_KERNEL_ERROR_ILLEGAL_BLOCK_ID	0x80024701	Specified block identifier is invalid
SCE_KERNEL_ERROR_ILLEGAL_BLOCK_TYPE	0x80024702	Specified block type is invalid
SCE_KERNEL_ERROR_BLOCK_IN_USE	0x80024703	Cannot free memory blocks in use
SCE_KERNEL_ERROR_PARTITION_ERROR	0x80024800	Error occurred with virtual memory partition
SCE_KERNEL_ERROR_ILLEGAL_PARTITION_ID	0x80024801	Specified virtual memory partition identifier is invalid
SCE_KERNEL_ERROR_ILLEGAL_PARTITION_INDEX	0x80024802	Specified virtual partition index is invalid
SCE_KERNEL_ERROR_NO_L2PAGETABLE	0x80024803	L2 page table corresponds to target virtual address does not exist
SCE_KERNEL_ERROR_PARTITION_NO_VIRTUAL_ADDRES S	0x80024804	Out of virtual addresses for target virtual partition
SCE_KERNEL_ERROR_HEAPLIB_ERROR	0x80024900	Error occurred with kernel heap subsystem
SCE_KERNEL_ERROR_ILLEGAL_HEAP_ID	0x80024901	Specified heap identifier is invalid
SCE_KERNEL_ERROR_OUT_OF_RANG	0x80024902	Error found in heap structure
SCE_KERNEL_ERROR_HEAPLIB_NOMEM	0x80024903	Heap has no available memory
SCE_KERNEL_ERROR_SYSMEM_ADDRESS_SPACE_ERROR	0x80024A00	Address space manager error
SCE_KERNEL_ERROR_INVALID_ADDRESS_SPACE_ID	0x80024A01	Invalid address space identifier
SCE_KERNEL_ERROR_INVALID_PARTITION_INDEX	0x80024A02	Partition for specified index does not exist in address space
SCE_KERNEL_ERROR_ADDRESS_SPACE_CANNOT_FIND_ PARTITION_BY_ADDR	0x80024A03	Partition with specified virtual address does not exist
SCE_KERNEL_ERROR_SYSMEM_MEMBLOCK_ERROR	0x80024B00	Memory block manager error
SCE_KERNEL_ERROR_ILLEGAL_MEMBLOCK_TYPE	0x80024B01	Invalid memory block type
SCE_KERNEL_ERROR_ILLEGAL_MEMBLOCK_REMAP_TYPE	0x80024B02	Block type for remap is invalid
SCE_KERNEL_ERROR_NOT_PHY_CONT_MEMBLOCK	0x80024B03	Not physical continuous memory block
SCE_KERNEL_ERROR_ILLEGAL_MEMBLOCK_CODE	0x80024B04	Invalid memory block code
SCE_KERNEL_ERROR_ILLEGAL_MEMBLOCK_SIZE	0x80024B05	Requested memory block size is incorrect
SCE_KERNEL_ERROR_ILLEGAL_USERMAP_SIZE	0x80024B06	Requested memory mapping size is incorrect
SCE_KERNEL_ERROR_MEMBLOCK_TYPE_FOR_KERNEL_ PROCESS	0x80024B07	Memory block type dedicated for kernel process was used
SCE_KERNEL_ERROR_PROCESS_CANNOT_REMAP_ MEMBLOCK	0x80024B08	Process is not allowed to remap of memory block
SCE_KERNEL_ERROR_SYSMEM_PHYMEMLOW_ERROR	0x80024C00	Error within physical memory managed lower part
SCE_KERNEL_ERROR_CANNOT_ALLOC_PHYMEMLOW	0x80024C01	Physical memory allocation within physical memory managed lower part failed
SCE_KERNEL_ERROR_UNKNOWN_PHYMEMLOW_TYPE	0x80024C02	Specified a physical memory low level type that does not exist
SCE_KERNEL_ERROR_SYSMEM_BITHEAP_ERROR	0x80024D00	Error in BitHeap
SCE_KERNEL_ERROR_CANNOT_ALLOC_BITHEAP	0x80024D01	Allocation from BitHeap failed
SCE_KERNEL_ERROR_SYSMEM_NAMEHEAP_ERROR	0x80024E00	Error of the NameHeap part
SCE_KERNEL_ERROR_NO_SUCH_NAME	0x80024E01	Such name does not exist

Document serial number: 000004892117

Definition	Value	Description
SCE_KERNEL_ERROR_DUPLICATE_NAME	0x80024E02	Name to be registered already exists
SCE_KERNEL_ERROR_LOADCORE_ERROR	0x80025000	Loader lower layer error
SCE_KERNEL_ERROR_ILLEGAL_ELF_HEADER	0x80025001	ELF header not supported by kernel
SCE_KERNEL_ERROR_ILLEGAL_SELF_HEADER	0x80025002	SELF header not supported by kernel
SCE_KERNEL_ERROR_EXCPMGR_ERROR	0x80027000	Error occurred with exception handler manager
SCE_KERNEL_ERROR_ILLEGAL_EXCPCODE	0x80027001	Specified exception code is invalid
SCE_KERNEL_ERROR_ILLEGAL_EXCPHANDLER	0x80027002	Specified exception handler is invalid
SCE_KERNEL_ERROR_NOTFOUND_EXCPHANDLER	0x80027003	Exception handler not found
SCE_KERNEL_ERROR_CANNOT_RELEASE_EXCPHANDLER	0x80027004	Cannot free exception handler
SCE_KERNEL_ERROR_INTRMGR_ERROR	0x80027100	Interrupt handler manager internal error
SCE_KERNEL_ERROR_ILLEGAL_CONTEXT	0x80027101	Invalid context
SCE_KERNEL_ERROR_ILLEGAL_INTRCODE	0x80027102	Invalid interrupt code
SCE_KERNEL_ERROR_ILLEGAL_INTRPARAM	0x80027103	Invalid optional parameter
SCE_KERNEL_ERROR_ILLEGAL_INTRPRIORITY	0x80027104	Invalid interrupt priority value
SCE_KERNEL_ERROR_ILLEGAL_TARGET_CPU	0x80027105	Target CPU value is invalid
SCE_KERNEL_ERROR_ILLEGAL_INTRFILTER	0x80027106	Invalid filter value
SCE_KERNEL_ERROR_ILLEGAL_INTRTYPE	0x80027107	Invalid interrupt handler value
SCE_KERNEL_ERROR_ILLEGAL_HANDLER	0x80027108	Invalid interrupt handler value
SCE_KERNEL_ERROR_FOUND_HANDLER	0x80027109	Handler is already registered
SCE_KERNEL_ERROR_NOTFOUND_HANDLER	0x8002710A	Handler is not registered
SCE_KERNEL_ERROR_NO_MEMORY	0x8002710B	Insufficient memory
SCE_KERNEL_ERROR_DMACMGR_ERROR	0x80027200	Error occurred with DMA service
SCE_KERNEL_ERROR_ALREADY_QUEUED	0x80027201	Queuing
SCE_KERNEL_ERROR_NOT_QUEUED	0x80027202	Not queued
SCE_KERNEL_ERROR_NOT_SETUP	0x80027203	Not set up
SCE_KERNEL_ERROR_ON_TRANSFERRING	0x80027204	Transferring
SCE_KERNEL_ERROR_NOT_INITIALIZED	0x80027205	Not initialized
SCE_KERNEL_ERROR_TRANSFERRED	0x80027206	Have been transferred
SCE_KERNEL_ERROR_NOT_UNDER_CONTROL	0x80027207	Not under control
SCE_KERNEL_ERROR_SYSTIMER_ERROR	0x80027300	System timer manager error
SCE_KERNEL_ERROR_NO_FREE_TIMER	0x80027301	No free timer available
SCE_KERNEL_ERROR_TIMER_NOT_ALLOCATED	0x80027302	Not allocated
SCE_KERNEL_ERROR_TIMER_COUNTING	0x80027303	Counting
SCE_KERNEL_ERROR_TIMER_STOPPED	0x80027304	Timer is stopped
SCE_KERNEL_ERROR_THREADMGR_ERROR	0x80028000	General error of the thread manager
SCE_KERNEL_ERROR_UNKNOWN_UID	0x80028001	UID does not exist
SCE_KERNEL_ERROR_DIFFERENT_UID_CLASS	0x80028002	UID class is different
SCE_KERNEL_ERROR_ALREADY_REGISTERED	0x80028003	Already registered
SCE_KERNEL_ERROR_CAN_NOT_WAIT	0x80028004	Cannot enter wait state
SCE_KERNEL_ERROR_WAIT_TIMEOUT	0x80028005	Wait timeout occurred
SCE_KERNEL_ERROR_WAIT_DELETE	0x80028006	Deleted during waiting
SCE_KERNEL_ERROR_WAIT_CANCEL	0x80028007	Waiting was canceled
SCE_KERNEL_ERROR_THREAD_ERROR	0x80028020	Thread error
SCE_KERNEL_ERROR_UNKNOWN_THREAD_ID	0x80028021	Thread ID which does not exist
SCE_KERNEL_ERROR_ILLEGAL_THREAD_ID	0x80028022	Invalid thread ID

Document serial number: 000004892117

Definition	Value	Description
SCE_KERNEL_ERROR_ILLEGAL_PRIORITY	0x80028023	Invalid priority
SCE_KERNEL_ERROR_ILLEGAL_STACK_SIZE	0x80028024	Invalid stack size
SCE_KERNEL_ERROR_ILLEGAL_CPU_AFFINITY_MASK	0x80028025	Invalid CPU affinity mask
SCE_KERNEL_ERROR_DORMANT	0x80028027	Thread in DORMANT state
SCE_KERNEL_ERROR_NOT_DORMANT	0x80028028	Thread is not in DORMANT state
SCE_KERNEL_ERROR_RUNNING	0x80028029	Thread is in RUNNING state
SCE_KERNEL_ERROR_DELETED	0x8002802A	Thread has been deleted
SCE_KERNEL_ERROR_CAN_NOT_SUSPEND	0x8002802B	Cannot suspend thread
SCE_KERNEL_ERROR_THREAD_STOPPED	0x8002802C	Thread was forcibly stopped
SCE_KERNEL_ERROR_THREAD_SUSPENDED	0x8002802D	Thread was forcibly suspended
SCE_KERNEL_ERROR_NOT_SUSPENDED	0x8002802E	Thread is not suspended
SCE_KERNEL_ERROR_ALREADY_DEBUG_SUSPENDED	0x8002802F	Thread is already in debug suspended state
SCE_KERNEL_ERROR_NOT_DEBUG_SUSPENDED	0x80028030	Thread is not in debug suspended state
SCE_KERNEL_ERROR_CAN_NOT_USE_VFP	0x80028031	VFP not available
SCE_KERNEL_ERROR_THREAD_EVENT_ERROR	0x80028060	Thread event error
SCE_KERNEL_ERROR_UNKNOWN_THREAD_EVENT_ID	0x80028061	Thread event ID which does not exist
SCE_KERNEL_ERROR_KERNEL_TLS_ERROR	0x80028080	Kernel TLS error
SCE_KERNEL_ERROR_KERNEL_TLS_FULL	0x80028081	Kernel TLS is full
SCE_KERNEL_ERROR_ILLEGAL_KERNEL_TLS_INDEX	0x80028082	Kernel TLS index is invalid
SCE_KERNEL_ERROR_KERNEL_TLS_BUSY	0x80028083	There is a thread using Kernel TLS index
SCE_KERNEL_ERROR_CALLBACK_ERROR	0x800280A0	Callback error
SCE_KERNEL_ERROR_UNKNOWN_CALLBACK_ID	0x800280A1	Callback ID which does not exist
SCE_KERNEL_ERROR_NOTIFY_CALLBACK	0x800280A2	Callback has been reported
SCE_KERNEL_ERROR_CALLBACK_NOT_REGISTERED	0x800280A3	Callback has not been registered
SCE_KERNEL_ERROR_ALARM_ERROR	0x800280C0	Alarm error
SCE_KERNEL_ERROR_UNKNOWN_ALARM_ID	0x800280C1	Alarm ID which does not exist
SCE_KERNEL_ERROR_ALARM_CAN_NOT_CANCEL	0x800280C2	Cannot cancel alarm
SCE_KERNEL_ERROR_EVF_ERROR	0x800280E0	Event flag error
SCE_KERNEL_ERROR_UNKNOWN_EVF_ID	0x800280E1	Event flag ID which does not exist
SCE_KERNEL_ERROR_EVF_MULTI	0x800280E2	Event flag is already waited for by another thread
SCE_KERNEL_ERROR_EVF_COND	0x800280E3	Event flag wait condition was not satisfied
SCE_KERNEL_ERROR_SEMA_ERROR	0x80028100	Semaphore error
SCE_KERNEL_ERROR_UNKNOWN_SEMA_ID	0x80028101	Semaphore ID which does not exist
SCE_KERNEL_ERROR_SEMA_ZERO	0x80028102	Insufficient semaphore resources
SCE_KERNEL_ERROR_SEMA_OVF	0x80028103	Semaphore count overflow
SCE_KERNEL_ERROR_SIGNAL_ERROR	0x80028120	Signal error
SCE_KERNEL_ERROR_ALREADY_SENT	0x80028121	Signal is already sent
SCE_KERNEL_ERROR_MUTEX_ERROR	0x80028140	Mutex error
SCE_KERNEL_ERROR_UNKNOWN_MUTEX_ID	0x80028141	Mutex ID which does not exist
SCE_KERNEL_ERROR_MUTEX_RECURSIVE	0x80028142	Mutex does not allow recursive locks
SCE_KERNEL_ERROR_MUTEX_LOCK_OVF	0x80028143	Mutex lock count overflow
SCE_KERNEL_ERROR_MUTEX_UNLOCK_UDF	0x80028144	Mutex unlock count underflow
SCE_KERNEL_ERROR_MUTEX_FAILED_TO_OWN	0x80028145	Mutex acquisition failed
SCE_KERNEL_ERROR_MUTEX_NOT_OWNED	0x80028146	Mutex is not owned
SCE_KERNEL_ERROR_FAST_MUTEX_ERROR	0x80028160	Fast mutex error

Definition	Value	Description
SCE_KERNEL_ERROR_UNKNOWN_FAST_MUTEX_ID	0x80028161	Fast mutex ID which does not exist
SCE_KERNEL_ERROR_FAST_MUTEX_RECURSIVE	0x80028162	Fast mutex does not allow recursive locks
SCE_KERNEL_ERROR_FAST_MUTEX_LOCK_OVF	0x80028163	Fast mutex lock count overflow
SCE_KERNEL_ERROR_FAST_MUTEX_FAILED_TO_OWN	0x80028164	Fast mutex acquisition failed
SCE_KERNEL_ERROR_FAST_MUTEX_NOT_OWNED	0x80028165	Fast mutex is not owned
SCE_KERNEL_ERROR_FAST_MUTEX_OWNED	0x80028166	Fast mutex is owned
SCE_KERNEL_ERROR_FAST_MUTEX_ALREADY_ INITIALIZED	0x80028167	Fast mutex is already initialized
SCE_KERNEL_ERROR_FAST_MUTEX_NOT_INITIALIZED	0x80028168	Fast mutex is not initialized
SCE_KERNEL_ERROR_LW_MUTEX_ERROR	0x80028180	Lightweight mutex error
SCE_KERNEL_ERROR_UNKNOWN_LW_MUTEX_ID	0x80028181	Lightweight mutex ID which does not exist
SCE_KERNEL_ERROR_LW_MUTEX_RECURSIVE	0x80028182	Lightweight mutex does not allow recursive locks
SCE_KERNEL_ERROR_LW_MUTEX_LOCK_OVF	0x80028183	Lock count overflow of lightweight mutex
SCE_KERNEL_ERROR_LW_MUTEX_UNLOCK_UDF	0x80028184	Lock count underflow of lightweight mutex
SCE_KERNEL_ERROR_LW_MUTEX_FAILED_TO_OWN	0x80028185	Lightweight mutex cannot be owned
SCE_KERNEL_ERROR_LW_MUTEX_NOT_OWNED	0x80028186	Lightweight mutex is not owned
SCE_KERNEL_ERROR_COND_ERROR	0x800281A0	Condition variable error
SCE_KERNEL_ERROR_UNKNOWN_COND_ID	0x800281A1	Condition variable ID which does not exist
SCE_KERNEL_ERROR_WAIT_DELETE_MUTEX	0x800281A2	Mutex deleted during waiting
SCE_KERNEL_ERROR_WAIT_CANCEL_MUTEX	0x800281A3	Mutex waiting canceled
SCE_KERNEL_ERROR_WAIT_DELETE_COND	0x800281A4	Condition variable deleted during waiting
SCE_KERNEL_ERROR_WAIT_CANCEL_COND	0x800281A5	Condition variable waiting canceled
SCE_KERNEL_ERROR_LW_COND_ERROR	0x800281C0	Lightweight condition variable error
SCE_KERNEL_ERROR_UNKNOWN_LW_COND_ID	0x800281C1	Lightweight condition variable ID which does not exist
SCE_KERNEL_ERROR_WAIT_DELETE_LW_MUTEX	0x800281C2	Lightweight mutex deleted during waiting
SCE_KERNEL_ERROR_WAIT_DELETE_LW_COND	0x800281C3	Lightweight condition variable deleted during waiting
SCE_KERNEL_ERROR_RW_LOCK_ERROR	0x800281E0	Reader/writer lock error
SCE_KERNEL_ERROR_UNKNOWN_RW_LOCK_ID	0x800281E1	Reader/writer lock ID which does not exist
SCE_KERNEL_ERROR_RW_LOCK_RECURSIVE	0x800281E2	Reader/writer lock does not allow recursive locks
SCE_KERNEL_ERROR_RW_LOCK_LOCK_OVF	0x800281E3	Reader/writer lock count overflow
SCE_KERNEL_ERROR_RW_LOCK_NOT_OWNED	0x800281E4	Reader/writer lock is not owned
SCE_KERNEL_ERROR_RW_LOCK_UNLOCK_UDF	0x800281E5	Reader/writer unlock count underflow
SCE_KERNEL_ERROR_RW_LOCK_FAILED_TO_LOCK	0x800281E6	Reader/writer lock failed
SCE_KERNEL_ERROR_RW_LOCK_FAILED_TO_UNLOCK	0x800281E7	Reader/writer unlock failed
SCE_KERNEL_ERROR_EVENT_ERROR	0x80028200	Event error
SCE_KERNEL_ERROR_UNKNOWN_EVENT_ID	0x80028201	Event ID which does not exist

Definition	Value	Description
SCE_KERNEL_ERROR_EVENT_COND	0x80028202	Event polling failed
SCE_KERNEL_ERROR_MSG_PIPE_ERROR	0x80028220	Message pipe error
SCE_KERNEL_ERROR_UNKNOWN_MSG_PIPE_ID	0x80028221	Message pipe ID which does not exist
SCE_KERNEL_ERROR_MSG_PIPE_FULL	0x80028222	Too many message pipes
SCE_KERNEL_ERROR_MSG_PIPE_EMPTY	0x80028223	Message pipe is empty
SCE_KERNEL_ERROR_MSG_PIPE_DELETED	0x80028224	Buffer of the message pipe has been deleted
SCE_KERNEL_ERROR_TIMER_ERROR	0x80028240	Timer error
SCE_KERNEL_ERROR_UNKNOWN_TIMER_ID	0x80028241	Timer ID which does not exist
SCE_KERNEL_ERROR_EVENT_NOT_SET	0x80028242	Event has not been set
SCE_KERNEL_ERROR_SIMPLE_EVENT_ERROR	0x80028260	Simple event error
SCE_KERNEL_ERROR_UNKNOWN_SIMPLE_EVENT_ID	0x80028261	Simple event ID which does not exist
SCE_KERNEL_ERROR_PMON_ERROR	0x80028280	Performance monitor error
SCE_KERNEL_ERROR_PMON_NOT_THREAD_MODE	0x80028281	PMON counter is not in thread mode
SCE_KERNEL_ERROR_PMON_NOT_CPU_MODE	0x80028282	PMON counter is not in CPU mode
SCE_KERNEL_ERROR_PROCESSMGR_ERROR	0x80029000	Process manager error
SCE_KERNEL_ERROR_INVALID_PID	0x80029001	Provided process identifier is invalid
SCE_KERNEL_ERROR_INVALID_PROCESS_TYPE	0x80029002	Invalid process type
SCE_KERNEL_ERROR_PLS_FULL	0x80029003	Process-specific work area is full
SCE_KERNEL_ERROR_INVALID_PROCESS_STATUS	0x80029004	Invalid process status
SCE_KERNEL_ERROR_PROCESS_CALLBACK_NOTFOUND	0x80029005	Process termination callback not registered
SCE_KERNEL_ERROR_INVALID_BUDGET_ID	0x80029006	Invalid budget UID
SCE_KERNEL_ERROR_INVALID_BUDGET_SIZE	0x80029007	Size of budget is invalid
SCE_KERNEL_ERROR_CP14_DISABLED	0x80029008	CP14 is disabled
SCE_KERNEL_ERROR_EXCEEDED_MAX_PROCESSES	0x80029009	Maximum number of processes for budget exceeded
SCE_KERNEL_ERROR_PROCESS_REMAINING	0x8002900A	Process is remaining
SCE_KERNEL_ERROR_NO_PROCESS_DATA	0x8002900B	Process specific data area is full
SCE_KERNEL_ERROR_PROCESS_EVENT_INHIBITED	0x8002900C	Operation in relation to process is prohibited
SCE_KERNEL_ERROR_IOFILEMGR_ERROR	0x8002A000	I/O file manager error
SCE_KERNEL_ERROR_IO_NAME_TOO_LONG	0x8002A001	Path name or file name too long
SCE_KERNEL_ERROR_IO_REG_DEV	0x8002A002	Device is already registered
SCE_KERNEL_ERROR_IO_ALIAS_USED	0x8002A003	Alias already in use
SCE_KERNEL_ERROR_IO_DEL_DEV	0x8002A004	Driver has been deleted
SCE_KERNEL_ERROR_IO_WOULD_BLOCK	0x8002A005	File is already locked
SCE_KERNEL_ERROR_MODULEMGR_START_FAILED	0x8002D000	Module start failed
SCE_KERNEL_ERROR_MODULEMGR_STOP_FAIL	0x8002D001	Module stop failed
SCE_KERNEL_ERROR_MODULEMGR_IN_USE	0x8002D002	Module unlinking failed
SCE_KERNEL_ERROR_MODULEMGR_NO_LIB	0x8002D003	Library does not exist
SCE_KERNEL_ERROR_MODULEMGR_SYSCALL_REG	0x8002D004	System call registration failed
SCE_KERNEL_ERROR_MODULEMGR_NOMEM_LIB	0x8002D005	Insufficient memory for library management
SCE_KERNEL_ERROR_MODULEMGR_NOMEM_STUB	0x8002D006	Insufficient memory for stub management
SCE_KERNEL_ERROR_MODULEMGR_NOMEM_SELF	0x8002D007	Insufficient memory for SELF management

Definition	Value	Description
SCE_KERNEL_ERROR_MODULEMGR_NOMEM	0x8002D008	Insufficient memory for module management
SCE_KERNEL_ERROR_MODULEMGR_INVALID_LIB	0x8002D009	Invalid library
SCE_KERNEL_ERROR_MODULEMGR_INVALID_STUB	0x8002D00A	Invalid stub
SCE_KERNEL_ERROR_MODULEMGR_NO_FUNC_NID	0x8002D00B	No function NID in corresponding library
SCE_KERNEL_ERROR_MODULEMGR_NO_VAR_NID	0x8002D00C	No variable NID in corresponding library
SCE_KERNEL_ERROR_MODULEMGR_INVALID_TYPE	0x8002D00D	Invalid module type
SCE_KERNEL_ERROR_MODULEMGR_NO_MOD_ENTRY	0x8002D00E	Module entry does not exist
SCE_KERNEL_ERROR_MODULEMGR_INVALID_PROC_PARAM	0x8002D00F	Invalid process parameter
SCE_KERNEL_ERROR_MODULEMGR_NO_MODOBJ	0x8002D010	Module object does not exist
SCE_KERNEL_ERROR_MODULEMGR_NO_MOD	0x8002D011	Module does not exist
SCE_KERNEL_ERROR_MODULEMGR_NO_PROCESS	0x8002D012	Process does not exist
SCE_KERNEL_ERROR_MODULEMGR_OLD_LIB	0x8002D013	Old library registration
SCE_KERNEL_ERROR_MODULEMGR_STARTED	0x8002D014	Module already started
SCE_KERNEL_ERROR_MODULEMGR_NOT_STARTED	0x8002D015	Module not started
SCE_KERNEL_ERROR_MODULEMGR_NOT_STOPPED	0x8002D016	Module not stopped
SCE_KERNEL_ERROR_MODULEMGR_INVALID_PROCESS_UID	0x8002D017	Invalid process UID
SCE_KERNEL_ERROR_MODULEMGR_CANNOT_EXPORT_LIB_TO_SHARED	0x8002D018	Cannot export library to shared text module
SCE_KERNEL_ERROR_MODULEMGR_INVALID_REL_INFO	0x8002D019	Invalid relocation information
SCE_KERNEL_ERROR_MODULEMGR_INVALID_REF_INFO	0x8002D01A	Invalid reference table information
SCE_KERNEL_ERROR_MODULEMGR_ELINK	0x8002D01B	Old version of library. Cannot link.
SCE_KERNEL_ERROR_MODULEMGR_NOENT	0x8002D01C	Object does not exist
SCE_KERNEL_ERROR_MODULEMGR_BUSY	0x8002D01D	Object is busy
SCE_KERNEL_ERROR_MODULEMGR_NOEXEC	0x8002D01E	Invalid object format
SCE_KERNEL_ERROR_MODULEMGR_NAMETOOLONG	0x8002D01F	Name too long
SCE_KERNEL_ERROR_LIBRARYDB_NOENT	0x8002D080	Library database does not exist
SCE_KERNEL_ERROR_LIBRARYDB_NO_LIB	0x8002D081	Library does not exist
SCE_KERNEL_ERROR_LIBRARYDB_NO_MOD	0x8002D082	Module does not exist in library database
SCE_KERNEL_ERROR_AUTHFAIL	0x8002F000	SELF authentication failed
SCE_KERNEL_ERROR_NO_AUTH	0x8002F001	No privileges
SCE_ERROR_ERRNO_EPERM	0x80010001	No execute permission
SCE_ERROR_ERRNO_ENOENT	0x80010002	No such file
SCE_ERROR_ERRNO_ESRCH	0x80010003	No such process
SCE_ERROR_ERRNO_EINTR	0x80010004	Function call interrupted
SCE_ERROR_ERRNO_EIO	0x80010005	I/O error
SCE_ERROR_ERRNO_ENXIO	0x80010006	No such device or address
SCE_ERROR_ERRNO_E2BIG	0x80010007	Argument list too long
SCE_ERROR_ERRNO_ENOEXEC	0x80010008	Execution format error
SCE_ERROR_ERRNO_EBADF	0x80010009	Invalid descriptor
SCE_ERROR_ERRNO_ECHILD	0x8001000A	No child process
SCE_ERROR_ERRNO_EAGAIN	0x8001000B	Process does not exist
SCE_ERROR_ERRNO_ENOMEM	0x8001000C	Cannot allocate needed memory
SCE_ERROR_ERRNO_EACCES	0x8001000D	No access privileges
SCE_ERROR_ERRNO_EFAULT	0x8001000E	Invalid address
SCE_ERROR_ERRNO_ENOTBLK	0x8001000F	Not a block device

Definition	Value	Description
SCE_ERROR_ERRNO_EBUSY	0x80010010	Resource busy
SCE_ERROR_ERRNO_EEXIST	0x80010011	File exists
SCE_ERROR_ERRNO_EXDEV	0x80010012	Multiple devices specified
SCE_ERROR_ERRNO_ENODEV	0x80010013	Device does not exist
SCE_ERROR_ERRNO_ENOTDIR	0x80010014	Not a directory
SCE_ERROR_ERRNO_EISDIR	0x80010015	Is a directory
SCE_ERROR_ERRNO_EINVAL	0x80010016	Invalid argument
SCE_ERROR_ERRNO_ENFILE	0x80010017	Too many open files
SCE_ERROR_ERRNO_EMFILE	0x80010018	Too many open files
SCE_ERROR_ERRNO_ENOTTY	0x80010019	Not a TTY device
SCE_ERROR_ERRNO_ETXTBSY	0x8001001A	Text file busy
SCE_ERROR_ERRNO_EFBIG	0x8001001B	File too large
SCE_ERROR_ERRNO_ENOSPC	0x8001001C	No space left on device
SCE_ERROR_ERRNO_ESPIPE	0x8001001D	Invalid seek
SCE_ERROR_ERRNO_EROFS	0x8001001E	Read-only file system
SCE_ERROR_ERRNO_EMLINK	0x8001001F	Too many links
SCE_ERROR_ERRNO_EPIPE	0x80010020	Broken pipe
SCE_ERROR_ERRNO_EDOM	0x80010021	Numeric argument out of domain
SCE_ERROR_ERRNO_ERANGE	0x80010022	Numerical result out of range
SCE_ERROR_ERRNO_ENOMSG	0x80010023	No message of requested type
SCE_ERROR_ERRNO_EIDRM	0x80010024	Identifier removed
SCE_ERROR_ERRNO_ECHRNG	0x80010025	Channel number out of range
SCE_ERROR_ERRNO_EL2NSYNC	0x80010026	Level 2 not synchronized
SCE_ERROR_ERRNO_EL3HLT	0x80010027	Level 3 halted
SCE_ERROR_ERRNO_EL3RST	0x80010028	Level 3 reset
SCE_ERROR_ERRNO_ELNRNG	0x80010029	Link number out of range
SCE_ERROR_ERRNO_EUNATCH	0x8001002A	Protocol driver not attached
SCE_ERROR_ERRNO_ENOCSI	0x8001002B	No CSI structure available
SCE_ERROR_ERRNO_EL2HLT	0x8001002C	Level 2 halted
SCE_ERROR_ERRNO_EDEADLK	0x8001002D	Deadlock
SCE_ERROR_ERRNO_ENOLCK	0x8001002E	No locks available
SCE_ERROR_ERRNO_EFORMAT	0x8001002F	Invalid file format
SCE_ERROR_ERRNO_EUNSUP	0x80010030	Operation not supported by device
SCE_ERROR_ERRNO_EBADE	0x80010032	Invalid exchange
SCE_ERROR_ERRNO_EBADR	0x80010033	Invalid request descriptor
SCE_ERROR_ERRNO_EXFULL	0x80010034	Exchange full
SCE_ERROR_ERRNO_ENOANO	0x80010035	No anode
SCE_ERROR_ERRNO_EBADRQC	0x80010036	Invalid request code
SCE_ERROR_ERRNO_EBADSLT	0x80010037	Invalid slot
SCE_ERROR_ERRNO_EDEADLOCK	0x80010038	File lock deadlock error
SCE_ERROR_ERRNO_EBFONT	0x80010039	Invalid font file format
SCE_ERROR_ERRNO_ENOSTR	0x8001003C	Device not a stream
SCE_ERROR_ERRNO_ENODATA	0x8001003D	No data (for no delay io)
SCE_ERROR_ERRNO_ETIME	0x8001003E	Timer expired
SCE_ERROR_ERRNO_ENOSR	0x8001003F	Out of stream resources
SCE_ERROR_ERRNO_ENONET	0x80010040	Machine is not on the network
SCE_ERROR_ERRNO_ENOPKG	0x80010041	Package not installed
SCE_ERROR_ERRNO_EREMOTE	0x80010042	Object is remote
SCE_ERROR_ERRNO_ENOLINK	0x80010043	Link has been severed
SCE_ERROR_ERRNO_EADV	0x80010044	Error notification
SCE_ERROR_ERRNO_ESRMNT	0x80010045	srmount error
SCE_ERROR_ERRNO_ECOMM	0x80010046	Communication error on send

Document serial number: 000004892117

Definition	Value	Description
SCE_ERROR_ERRNO_EPROTO	0x80010047	Protocol error
SCE_ERROR_ERRNO_EMULTIHOP	0x8001004A	Multihop attempted
SCE_ERROR_ERRNO_ELBIN	0x8001004B	node is remote (not an error)
SCE_ERROR_ERRNO_EDOTDOT	0x8001004C	Cross mount point (not an error)
SCE_ERROR_ERRNO_EBADMSG	0x8001004D	Trying to read illegible message
SCE_ERROR_ERRNO_EFTYPE	0x8001004F	File type error
SCE_ERROR_ERRNO_ENOTUNIQ	0x80010050	Name on network not unique
SCE_ERROR_ERRNO_EBADFD	0x80010051	Invalid file descriptor for this process
SCE_ERROR_ERRNO_EREMCHG	0x80010052	Remote address changed
SCE_ERROR_ERRNO_ELIBACC	0x80010053	Cannot access a needed shared library
SCE_ERROR_ERRNO_ELIBBAD	0x80010054	Accessing a corrupted shared library
SCE_ERROR_ERRNO_ELIBSCN	0x80010055	.lib section in a.out corrupted
SCE_ERROR_ERRNO_ELIBMAX	0x80010056	Attempting to link in too many libraries
SCE_ERROR_ERRNO_ELIBEXEC	0x80010057	Attempting to execute a shared library
SCE_ERROR_ERRNO_ENOSYS	0x80010058	Function not implemented
SCE_ERROR_ERRNO_ENMFILE	0x80010059	No more files
SCE_ERROR_ERRNO_ENOTEMPTY	0x8001005A	Directory not empty
SCE_ERROR_ERRNO_ENAMETOOLONG	0x8001005B	File name or path name too long
SCE_ERROR_ERRNO_ELOOP	0x8001005C	Cannot follow symbolic links
SCE_ERROR_ERRNO_EOPNOTSUPP	0x8001005F	Operation not supported
SCE_ERROR_ERRNO_EPFNOSUPPORT	0x80010060	Protocol family not supported
SCE_ERROR_ERRNO_ECONNRESET	0x80010068	Connection reset
SCE_ERROR_ERRNO_ENOBUFS	0x80010069	No buffer space
SCE_ERROR_ERRNO_EAFNOSUPPORT	0x8001006A	Address not supported by protocol
SCE_ERROR_ERRNO_EPROTOTYPE	0x8001006B	Invalid socket protocol type
SCE_ERROR_ERRNO_ENOTSOCK	0x8001006C	Socket operation on non-socket
SCE_ERROR_ERRNO_ENOPROTOOPT	0x8001006D	Protocol not available
SCE_ERROR_ERRNO_ESHUTDOWN	0x8001006E	Cannot send after socket shutdown
SCE_ERROR_ERRNO_ECONNREFUSED	0x8001006F	Connection refused
SCE_ERROR_ERRNO_EADDRINUSE	0x80010070	Address in use
SCE_ERROR_ERRNO_ECONNABORTED	0x80010071	Connection aborted
SCE_ERROR_ERRNO_ENETUNREACH	0x80010072	Network is unreachable
SCE_ERROR_ERRNO_ENETDOWN	0x80010073	Network is down
SCE_ERROR_ERRNO_ETIMEDOUT	0x80010074	Timeout occurred
SCE_ERROR_ERRNO_EHOSTDOWN	0x80010075	Host is down
SCE_ERROR_ERRNO_EHOSTUNREACH	0x80010076	Host is unreachable
SCE_ERROR_ERRNO_EINPROGRESS	0x80010077	Operation now in progress
SCE_ERROR_ERRNO_EALREADY	0x80010078	Operation already in progress
SCE_ERROR_ERRNO_EDESTADDRREQ	0x80010079	Destination address requested
SCE_ERROR_ERRNO_EMSGSIZE	0x8001007A	Message too long
SCE_ERROR_ERRNO_EPROTONOSUPPORT	0x8001007B	Protocol not supported
SCE_ERROR_ERRNO_ESOCKTNOSUPPORT	0x8001007C	Socket type not supported
SCE_ERROR_ERRNO_EADDRNOTAVAIL	0x8001007D	Cannot assign requested address
SCE_ERROR_ERRNO_ENETRESET	0x8001007E	Network dropped connection because of reset
SCE_ERROR_ERRNO_EISCONN	0x8001007F	Socket is already connected

Definition	Value	Description
SCE_ERROR_ERRNO_ENOTCONN	0x80010080	Socket is not connected
SCE_ERROR_ERRNO_ETOOMANYREFS	0x80010081	Too many references: cannot connect
SCE_ERROR_ERRNO_EPROCLIM	0x80010082	Too many processes
SCE_ERROR_ERRNO_EUSERS	0x80010083	Too many users
SCE_ERROR_ERRNO_EDQUOT	0x80010084	Disk quota exceeded
SCE_ERROR_ERRNO_ESTALE	0x80010085	File handle is old
SCE_ERROR_ERRNO_ENOTSUP	0x80010086	Not supported
SCE_ERROR_ERRNO_ENOMEDIUM	0x80010087	No medium found
SCE_ERROR_ERRNO_ENOSHARE	0x80010088	No shared name found
SCE_ERROR_ERRNO_ECASECLASH	0x80010089	Clash
SCE_ERROR_ERRNO_EILSEQ	0x8001008A	ILSEQ
SCE_ERROR_ERRNO_EOVERFLOW	0x8001008B	Overflow
SCE_ERROR_ERRNO_ECANCELED	0x8001008C	Canceled
SCE_ERROR_ERRNO_ENOTRECOVERABLE	0x8001008D	Not recoverable
SCE_ERROR_ERRNO_EOWNERDEAD	0x8001008E	Owner does not exist
SCE_ERROR_ERRNO_EICV	0x8001008F	Falsification check error