

PlayStation®2 EE Library Reference

Release 2.4

Kernel Libraries

© 2001 Sony Computer Entertainment Inc.

Publication date: October 2001

Sony Computer Entertainment Inc.
1-1, Akasaka 7-chome, Minato-ku
Tokyo 107-0052, Japan

Sony Computer Entertainment America
919 E. Hillsdale Blvd.
Foster City, CA 94404, U.S.A.

Sony Computer Entertainment Europe
30 Golden Square
London W1F 9LD, U.K.


The *PlayStation®2 EE Library Reference - Kernel Libraries* manual is supplied pursuant to and subject to the terms of the Sony Computer Entertainment PlayStation® license agreements.

The *PlayStation®2 EE Library Reference - Kernel Libraries* manual is intended for distribution to and use by only Sony Computer Entertainment licensed Developers and Publishers in accordance with the PlayStation® license agreements.

Unauthorized reproduction, distribution, lending, rental or disclosure to any third party, in whole or in part, of this book is expressly prohibited by law and by the terms of the Sony Computer Entertainment PlayStation® license agreements.

Ownership of the physical property of the book is retained by and reserved by Sony Computer Entertainment. Alteration to or deletion, in whole or in part, of the book, its presentation, or its contents is prohibited.

The information in the *PlayStation®2 EE Library Reference - Kernel Libraries* manual is subject to change without notice. The content of this book is Confidential Information of Sony Computer Entertainment.

 and PlayStation are registered trademarks of Sony Computer Entertainment Inc. All other trademarks are property of their respective owners and/or their licensors.

Summary Table of Contents

About This Manual	v
Changes Since Last Release	v
Related Documentation	v
Typographic Conventions	v
Developer Support	v
Chapter 1: EE Kernel	1-1
Structures	1-3
Functions	1-5
Program Load/Exit Functions	1-56
Chapter 2: Standard I/O Service Functions (EE)	2-1
Structures	2-3
Functions	2-6

About This Manual

This is the Runtime Library Release 2.4 version of the *PlayStation®2 EE Library Reference - Kernel Libraries* manual.

The purpose of this manual is to define all available PlayStation®2 EE kernel library structures and functions. The companion *PlayStation®2 EE Library Overview - Kernel Libraries* describes the structure and purpose of the libraries.

Changes Since Last Release

Chapter 1: EE Kernel API

- In the "Description" section of `ExpandScratchPad()`, the description of `InitTLBFunctions()` has been deleted.
- A description that it is unnecessary for the user to call this function since it is called in `crt0.s` has been added to the "Description" section of `InitTLBFunctions()`.

Chapter 2: Standard I/O Service Functions (EE)

- Descriptions of the following structures have been added

`sce_dirent`
`sce_stat`

- Descriptions of the following functions have been added.

`sceChdir()`
`sceChstat()`
`sceDclose()`
`sceDopen()`
`sceDread()`
`sceFormat()`
`sceGetstat()`
`sceIoctl2()`
`sceLseek64()`
`sceMkdir()`
`sceMount()`
`sceReadlink()`
`sceRemove()`
`sceRename()`
`sceRmdir()`
`sceSymlink()`
`sceSync()`
`sceUmount()`

Related Documentation

Library specifications for the IOP can be found in the *PlayStation®2 IOP Library Reference* manuals and the *PlayStation®2 IOP Library Overview* manuals.

Note: the Developer Support Web site posts current developments regarding the Libraries and also provides notice of future documentation releases and upgrades.

Typographic Conventions

Certain Typographic Conventions are used throughout this manual to clarify the meaning of the text:

Convention	Meaning
<code>courier</code>	Indicates literal program code.
<i>italic</i>	Indicates names of arguments and structure members (in structure/function definitions only).
medium bold	Indicates data types and structure/function names (in structure/function definitions only).
blue	Indicates a hyperlink.

Developer Support

Sony Computer Entertainment America (SCEA)

SCEA developer support is available to licensees in North America only. You may obtain developer support or additional copies of this documentation by contacting the following addresses:

Order Information	Developer Support
<i>In North America:</i>	<i>In North America:</i>
Attn: Developer Tools Coordinator	E-mail: PS2_Support@playstation.sony.com
Sony Computer Entertainment America	Web: http://www.devnet.scea.com/
919 East Hillsdale Blvd.	Developer Support Hotline: (650) 655-5566
Foster City, CA 94404, U.S.A.	(Call Monday through Friday,
Tel: (650) 655-8000	8 a.m. to 5 p.m., PST/PDT)

Sony Computer Entertainment Europe (SCEE)

SCEE developer support is available to licensees in Europe only. You may obtain developer support or additional copies of this documentation by contacting the following addresses:

Order Information	Developer Support
<i>In Europe:</i>	<i>In Europe:</i>
Attn: Production Coordinator	E-mail: ps2_support@scee.net
Sony Computer Entertainment Europe	Web: https://www.ps2-pro.com/
30 Golden Square	Developer Support Hotline:
London W1F 9LD, U.K.	+44 (0) 20 7859-5777
Tel: +44 (0) 20 7859-5000	(Call Monday through Friday,
	9 a.m. to 6 p.m., GMT)

Chapter 1: EE Kernel

Table of Contents

Structures	1-3
SemaParam	1-3
ThreadParam	1-4
Functions	1-5
AddDmacHandler	1-5
AddDmacHandler2	1-7
AddIntcHandler	1-8
AddIntcHandler2	1-10
AddSbusIntcHandler	1-12
CancelWakeupThread / iCancelWakeupThread	1-13
ChangeThreadPriority / iChangeThreadPriority	1-14
CreateSema	1-15
CreateThread	1-16
DeleteSema	1-17
DeleteThread	1-18
DI	1-19
DisableDmac / iDisableDmac	1-20
DisableIntc / iDisableIntc	1-21
EI	1-22
EnableDmac / iEnableDmac	1-23
EnableIntc / iEnableIntc	1-24
ExitDeleteThread	1-25
ExitThread	1-26
ExpandScratchPad	1-27
FlushCache / iFlushCache	1-28
GetCop0 / iGetCop0	1-29
GetThreadId	1-30
InitThread	1-31
InitTLBFunctions	1-32
Interrupt2Iop	1-33
InvalidDCache / iInvalidDCache	1-34
ReferSemaStatus / iReferSemaStatus	1-35
ReferThreadStatus / iReferThreadStatus	1-36
ReleaseAlarm / iReleaseAlarm	1-37
ReleaseWaitThread / iReleaseWaitThread	1-38
RemoveDmacHandler	1-39
RemoveIntcHandler	1-40
RemoveSbusIntcHandler	1-41
ResumeThread / iResumeThread	1-42
RotateThreadReadyQueue / iRotateThreadReadyQueue	1-43
SetAlarm / iSetAlarm	1-44
SetDebugHandler	1-46
SignalSema / iSignalSema	1-48
SleepThread	1-49
StartThread	1-50
SuspendThread / iSuspendThread	1-51
SyncDCache / iSyncDCache	1-52

TerminateThread / iTerminateThread	1-53
WaitSema / PollSema / iPollSema	1-54
WakeupThread / iWakeupThread	1-55
Program Load/Exit Functions	1-56
Exit	1-56
LoadExecPS2	1-57

Structures

SemaParam

Semaphore

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	March 26, 2001

Structure

```
#include <eekernel.h>
```

```
struct SemaParam {
```

<code>int currentCount;</code>	Current semaphore count
<code>int maxCount;</code>	Maximum semaphore count
<code>int initCount;</code>	Initial semaphore count
<code>int numWaitThreads;</code>	Number of threads waiting on semaphore
<code>u_int attr;</code>	Semaphore attribute
<code>u_int option;}</code>	Optional user-defined data

Description

This structure is used for semaphores.

option is a member that is unaffected by the EE kernel and is freely available to user programs.

Notes

In the current version, *maxCount* is not correctly processed.

ThreadParam

Thread attributes

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	July 2, 2001

Structure

```
#include <eekernel.h>
```

```
struct ThreadParam {
```

<code>int status;</code>	Thread status
<code>void (*entry)(void *);</code>	Entry address for execution (*)
<code>void *stack;</code>	Stack address used by thread (*) (16-byte alignment)
<code>int stackSize;</code>	Stack size (in bytes: multiple of 16) (*)
<code>void *gpReg;</code>	Value of GP (Global Pointer) register (*)
<code>int initPriority;</code>	Initial priority (1 - 127) (*)
<code>int currentPriority;</code>	Current priority (1 – 127)
<code>u_int attr;</code>	System Reserved
<code>u_int option;</code>	Do not use
<code>int waitType;</code>	WAIT type
<code>int waitId;</code>	semaphore ID if WAIT type is "semaphore"
<code>int wakeupCount;</code>	Wakeup request count

(*) must be specified when creating the thread

Description

This structure holds thread attributes.

status indicates the state of the thread according to the following constants.

Table 1-1: Constants indicating state of thread

Constant	Value	State
THS_RUN	0x01	RUN state
THS_READY	0x02	READY state
THS_WAIT	0x04	WAIT state
THS_SUSPEND	0x08	SUSPEND state
THS_WAITSPEND	0x0c	WAIT-SUSPEND state
THS_DORMANT	0x10	DORMANT state

option is a member that is unaffected by the EE kernel and is freely available to user programs.

waitType indicates the WAIT type using the following values.

Table 1-2: Values used to indicate WAIT type

Value	Type
0	Not in WAIT state
1	Waiting for WAKEUP request
2	Waiting for semaphore

Functions

AddDmacHandler

Add DMA interrupt handler

Library	Introduced	Documentation last modified
eekernel	1.1	March 26, 2001

Syntax

```
#include <eekernel.h>
int AddDmacHandler(
    int channel,
    int (*handler)(int ch),
    int next);
```

Channel number

Constant DMA channel

DMAC_VIF0	VIF0	DIR:to	GP:A
DMAC_VIF1	VIF1	DIR:both	GP:C
DMAC_GIF	GIF	DIR:to	GP:C
DMAC_FROM_IPU	from IPU	DIR:from	GP:C
DMAC_TO_IPU	to IPU	DIR:to	GP:C
DMAC_FROM_SPR	from SPR	DIR:from	GP:C
DMAC_TO_SPR	to SPR	DIR:to	GP:C

Handler function

Handler ID of a previously registered handler

0: register as first handler

-1: register as last handler

Calling conditions

Can be called from a thread

Multithread safe

Description

Adds a DMA interrupt handler by DMA channel. *handler* is the address of the handler that will be called at the time of the DMA interrupt. *channel* is the DMA channel number and will be passed to the handler in argument *ch*. *next* is the address of a previously registered DMA interrupt handler. If *next* is 0, handler will be registered as the first interrupt handler. If *next* is -1, handler will be registered as the last interrupt handler.

Notes

When a DMA interrupt is generated, the GPRs are saved before the interrupt handler is called and D_STAT.CIS, corresponding to the channel that generated the interrupt, is cleared. When the handler has finished executing, the GPRs are restored. However, note that there is no save/restore of FPU registers.

When writing interrupt handlers, note that interrupts are disabled and that different APIs can be used. Refer to \overview\eekernel for information on these issues.

If an interrupt handler returns -1, subsequently registered interrupt handlers will not be called.

Return value

Handler ID Normal termination

-1 Error

AddDmacHandler2

Add DMA interrupt handler

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.6	March 26, 2001

Syntax

```
#include <eekernel.h>
```

```
int AddDmacHandler2(
```

```
    int channel,
```

Channel number

Constant

DMA channel

DMAC_VIF0 VIF0 DIR:to GP:A

DMAC_VIF1 VIF1 DIR:both GP:C

DMAC_GIF GIF DIR:to GP:C

DMAC_FROM_IPU from IPU DIR:from GP:C

DMAC_TO_IPU to IPU DIR:to GP:C

DMAC_FROM_SPR from SPR DIR:from GP:C

DMAC_TO_SPR to SPR DIR:to GP:C

```
    int (*handler)(int ch, void *arg, void *addr)
```

Handler function

```
    int next);
```

Handler ID of a previously registered handler

0: register as first handler

-1: register as last handler

```
void *arg);
```

Argument passed to handler

Calling conditions

Can be called from a thread

Multithread safe

Description

Adds a DMA interrupt handler by DMA channel. The arguments passed to the interrupt handler are different to AddDmacHandler().

handler is the address of the handler that will be called at the time of the DMA interrupt. The handler *ch* argument is the DMA channel number, *arg* is the argument passed to the handler and *addr* is the value of the program counter when an interrupt is generated. If *next* is 0, handler will be registered as the first interrupt handler. If *next* is -1, handler will be registered as the last interrupt handler.

Notes

When a DMA interrupt is generated, the GPRs are saved before the interrupt handler is called and D_STAT.CIS, corresponding to the channel that generated the interrupt, is cleared. When the handler has finished executing, the GPRs are restored. However, note that there is no save/restore of FPU registers.

When writing interrupt handlers, note that interrupts are disabled and that different APIs can be used. Please refer to \overview\eekernel for information on these issues. If an interrupt handler returns -1, subsequently registered interrupt handlers will not be called.

Return value

Handler ID Normal termination

-1 Error

AddIntcHandler

Add INTC interrupt handler

Library	Introduced	Documentation last modified
eekernel	1.1	March 26, 2001

Syntax

#include <eekernel.h>

int AddIntcHandler(

int <i>cause</i> ,	Interrupt cause
	Constant Interrupt cause
	INTC_GS GS
	INTC_SBUS SBUS
	INTC_VBLANK_S V-blank start
	INTC_VBLANK_E V-blank end
	INTC_VIF0 VIF0
	INTC_VIF1 VIF1
	INTC_VU0 VU0
	INTC_VU1 VU1
	INTC_IPU IPU
	INTC_TIM0 Timer0
	INTC_TIM1 Timer1
int (* <i>handler</i>)(int <i>ca</i>),	Handler function
int <i>next</i>);	Handler ID of a previously registered handler
	0: register as first handler
	-1: register as last handler

Calling conditions

Can be called from a thread
Multithread safe

Description

Registers an INTC handler by interrupt cause.
cause is the interrupt cause. *handler* is the address of the handler that will be called when an interrupt is generated, and will be passed the interrupt cause in the argument *ca*. *next* is the handler ID of a previously registered handler. If *next* is 0, the handler will be registered as the first handler. If *next* is -1, the handler will be registered as the last handler.

Notes

When a DMA interrupt is generated, the GPRs are saved before the interrupt handler is called. When the handler has finished executing, the GPRs are restored. However, note that there is no save/restore of FPU registers.
When writing interrupt handlers, note that interrupts are disabled and that different APIs can be used. Refer to \overview\eekernel for information on these issues.
If an interrupt handler returns -1, subsequently registered interrupt handlers will not be called.

sceGsSyncV of the basic GS library and AddIntcHandler(INTC_VBLANK_S, ,) cannot be used together. In this case, use sceGSCallback instead of AddIntcHandler(INTC_VBLANK_S, ,).

Return value

Handler ID Normal termination

-1 Error return

AddIntcHandler2

Add INTC interrupt handler

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.6	March 26, 2001

Syntax

```
#include <eekernel.h>
```

```
int AddIntcHandler2(
```

```
    int cause,
```

Interrupt cause

Constant

Interrupt cause

INTC_GS

GS

INTC_SBUS

SBUS

INTC_VBLANK_S

V-blank start

INTC_VBLANK_E

V-blank end

INTC_VIF0

VIF0

INTC_VIF1

VIF1

INTC_VU0

VU0

INTC_VU1

VU1

INTC_IPU

IPU

INTC_TIM0

Timer0

INTC_TIM1

Timer1

```
    int (*handler)(int ca, void *arg, void *addr),
```

Handler function

```
    int next);
```

Handler ID of a previously registered handler

0: register as first handler

-1: register as last handler

```
void *arg);
```

Argument passed to handler

Calling conditions

Can be called from a thread

Multithread safe

Description

Registers an INTC handler by interrupt cause. The arguments passed to the handler are different to `AddIntcHandler`. *cause* is the interrupt cause. *handler* is the address of the handler that will be called when an interrupt is generated, the argument *ca* is the interrupt cause, *arg* is the argument passed to the handler and *addr* is the value of the program counter when an interrupt is generated. *next* is the handler ID of a previously registered handler. If *next* is 0, the handler will be registered as the first handler. If *next* is -1, the handler will be registered as the last handler.

Notes

When an external interrupt is generated, the GPRs are saved before the interrupt handler is called. When the handler has finished executing, the GPRs are restored. However, note that there is no save/restore of FPU registers.

When writing interrupt handlers, note that interrupts are disabled and that different APIs can be used. Refer to `\overview\eekernel` for information on these issues.

If an interrupt handler returns -1, subsequently registered interrupt handlers will not be called.

sceGsSyncV of the basic GS library and AddIntcHandler2(INTC_VBLANK_S, ,) cannot be used together. In this case, use sceGSCallback instead of AddIntcHandler2(INTC_VBLANK_S, ,).

Return value

Handler ID Normal termination

-1 Error

AddSbusIntcHandler

Add SBUS interrupt handler

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	March 26, 2001

Syntax

```
#include <eekernel.h>
int AddSbusIntcHandler(
    int cause,                Interrupt cause
    void (*handler)(int ca)); Handler function
```

Calling conditions

Can be called from a thread

Multithread safe

Description

Adds an SBUS interrupt handler for interrupts from the IOP. *cause* is the interrupt cause, and specifies a user-defined value from 0 to 15. Up to a maximum of 16 handlers can be added and when an interrupt is issued on the IOP, the handler from among these 16 which has the specified cause value will be called. When an interrupt from the IOP occurs, the handler is called which has a cause value that matches the one set by the interrupt.

For example, if the interrupt from the IOP sets a value of 0, the handler is called whose *cause* is 0. Other handlers are not called. When the interrupt occurs, the interrupt cause is set in the argument *ca* of the handler that is called.

Notes

When an external interrupt is generated, the GPRs are saved before the interrupt handler is called. When the handler has finished executing, the GPRs are restored. However, note that there is no save/restore of FPU registers.

When writing interrupt handlers, note that interrupts are disabled and that different APIs can be used. Refer to \overview\eekernel for information on these issues.

SBUS interrupts are always enabled as it is not possible to disable them.

Return value

cause	Normal termination
-1	Error

CancelWakeupThread / iCancelWakeupThread

Get wakeup request count and cancel

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	March 26, 2001

Syntax

```
#include <eekernel.h>
int CancelWakeupThread(
    int tid);                                Thread ID
int iCancelWakeupThread(int tid);
int tid);                                Thread ID
```

Calling conditions

CancelWakeupThread	Can be called from a thread
	Multithread safe
iCancelWakeupThread	Can be called from an interrupt

Description

Reads and clears the wakeup request count for the specified thread, canceling all wakeup requests.

Return value

Wakeup request count	Normal termination
-1	Error

ChangeThreadPriority / iChangeThreadPriority

Change thread priority

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	March 26, 2001

Syntax

```
#include <eekernel.h>
```

```
int ChangeThreadPriority(
```

```
    int tid,                      Thread ID
    int prio);                    Thread priority (1 – 127)
```

```
int iChangeThreadPriority(
```

```
    int tid,                      Thread ID
    int prio);                    Thread priority (1 – 127)
```

Calling conditions

ChangeTheadPriority	Can be called from a thread
	Multithread safe
iChangeTheadPriority	Can be called from an interrupt

Description

Changes the priority of the specified thread to *prio*.

The thread will be entered at the end of the ready queue at the corresponding priority. The new priority setting will be valid until the thread is terminated with Exit.

Return value

Priority before change	Normal termination
-1	Error

CreateSema

Create semaphore

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	July 2, 2001

Syntax

```
#include <eekernel.h>
```

```
int CreateSema(
```

```
struct SemaParam *sema);
```

sema->initCount

Initial semaphore value

sema->maxCount

Maximum semaphore value

sema->option

Additional semaphore information. This value can be accessed with ReferSemaStatus(). The multithreading manager cannot access this value so it is freely available.

Calling conditions

Can be called from a thread

Multithread safe

Description

This function creates a counting semaphore. The maximum number of semaphores that can be created is 256-3. The three semaphores that are subtracted are two semaphores that are created by the _InitSys function of crt0.s and the semaphore that is created by the InitThread function.

Return value

Semaphore ID Normal termination

-1 An attempt was made to create more semaphores than allowed (255), or an attempt was made to register a negative value (<0) for initCount

CreateThread

Create new thread

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	July 2, 2001

Syntax

```
#include <eekernel.h>
```

```
int CreateThread(
```

```
struct ThreadParam *param);
```

param->entry

Address of function to start executing

param->stack

Stack area used by the thread (aligned on 16-byte boundary)

param->stackSize

Stack size (in bytes, multiple of 16)

param->gpReg

Value of GP register(Global Pointer)

param->initPriority

Priority (1 - 127)

param->option

Do not set

Calling conditions

Can be called from a thread

Multithread safe

Description

This function creates a new thread (allocates and initializes the TCB) and returns its thread ID.

The thread created by this function is placed in DORMANT state and is not executed. The maximum number of threads that can be created is 256-3. The 3 threads that are subtracted are the main function thread, the thread created by the InitThread function, and the thread that operates during kernel idle time.

Notes

To align the stack on a 16-byte boundary, make the following variable declaration.

```
char stack[STACK_SIZE] __attribute__((aligned(16)));
```

Replace param->gpReg with the address of the global variable _gp.

```
param->gpReg = & gp;
```

Return value

Thread ID Normal termination

-1 Error

DeleteSema

Delete semaphore

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	March 26, 2001

Syntax

```
#include <eekernel.h>
```

```
int DeleteSema(
```

```
    int sid);
```

Semaphore ID

Calling conditions

Can be called from a thread

Multithread safe

Description

Deletes the specified semaphore. An error is returned to threads that have been registered in the semaphore's wait queue.

Return value

Semaphore ID Normal termination

-1 Error

DeleteThread

Delete thread

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	March 26, 2001

Syntax

```
#include <eekernel.h>
```

```
int DeleteThread(
```

```
    int tid );
```

ID of the thread to be deleted.

Calling conditions

Can be called from a thread

Multithread safe

Description

Deletes the specified thread and releases the TCB. The thread to be deleted must be in DORMANT state.

Notes

Use ExitDeleteThread to delete the calling thread.

Return value

Thread ID Normal termination

-1 Error

DI

Disable interrupts

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	March 26, 2001

Syntax

```
#include <eekernel.h>
```

```
int DI(void);
```

Calling conditions

Can be called from a thread

Multithread safe

Description

Sets the interrupt-enable bit of the CPU to 0.

The interrupt-enable bit is not part of the thread context. This function is provided for precise control of interrupts. For example, for controlling VU0 using macroinstructions. Execution rights should be controlled by adjusting thread priorities, and exclusive control should be performed using semaphores.

Notes

printf cannot be used during the interrupt-disabled state after DI() is called. Use sceprintf.

Starting with release 2.2, this API has been changed from a macro to a function. This is because as a macro, interrupts were getting disabled without the intention of the programmer, due to compiler optimization.

Return value

- 1 The state was interrupt enabled until the function was called.
- 0 The state was already interrupt disabled before the function was called.

DisableDmac / iDisableDmac

Disable DMA interrupts

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	March 26, 2001

Syntax

#include <eekernel.h>

int DisableDmac(

int *channel*);

Channel number

Constant

DMA channel

DMAC_VIF0

VIF0 DIR:to GP:A

DMAC_VIF1

VIF1 DIR:both GP:C

DMAC_GIF

GIF DIR:to GP:C

DMAC_FROM_IPU

from IPU DIR:from GP:C

DMAC_TO_IPU

to IPU DIR:to GP:C

DMAC_FROM_SPR

from SPR DIR:from GP:C

DMAC_TO_SPR

to SPR DIR:to GP:C

int iDisableDmac(

int *channel*

same as above

Calling conditions

DisableDMAC Can be called from a thread

Multithread safe

iDisableDMAC Can be called from an interrupt handler

Description

This system call disables DMA termination interrupts for the specified *channel*.

Return value

0 Already disabled

1 Changed to disabled

DisableIntc / iDisableIntc

Disable Intc interrupt

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	March 26, 2001

Syntax

```
#include <eekernel.h>
```

```
int DisableIntc(
```

```
    int cause);
```

Interrupt cause

Constant

Interrupt cause

INTC_GS

GS

INTC_SBUS

SBUS

INTC_VBLANK_S

V-blank start

INTC_VBLANK_E

V-blank end

INTC_VIF0

VIF0

INTC_VIF1

VIF1

INTC_VU0

VU0

INTC_VU1

VU1

INTC_IPU

IPU

INTC_TIM0

Timer0

INTC_TIM1

Timer1

```
int iDisableIntc(
```

```
    int cause);
```

same as above

Calling conditions

DisableIntc

Can be called from a thread

Multithread safe

iDisableIntc

Can be called from an interrupt handler

Description

This system call disables interrupts triggered by *cause*.

Return value

0 Already disabled

1 Changed to disabled

EI

Enable interrupts

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	March 26, 2001

Syntax

```
#include <eekernel.h>
```

```
int EI(void);
```

Calling conditions

Can be called from a thread

Multithread safe

Description

Sets the interrupt-enable bit of the CPU to 1.

The interrupt-enable bit is not part of the thread context. This function is provided for precise control of interrupts. For example, for controlling VU0 using macroinstructions. Execution rights should be controlled by adjusting thread priorities, and exclusive control should be performed using semaphores.

Starting with release 2.2, this API has been changed from a macro to a function. This is because as a macro, interrupts were getting enabled without the intention of the programmer, due to compiler optimization.

Return value

- 1 The state was already interrupt enabled before the function was called.
- 0 The state was interrupt disabled until the function was called.

EnableDmac / iEnableDmac

Enable DMA interrupt

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	March 26, 2001

Syntax

```
#include <eekernel.h>
```

```
int EnableDmac(
```

```
    int channel);
```

Channel number

Constant

DMA channel

DMAC_VIF0 VIF0 DIR:to GP:A

DMAC_VIF1 VIF1 DIR:both GP:C

DMAC_GIF GIF DIR:to GP:C

DMAC_FROM_IPU from IPU DIR:from GP:C

DMAC_TO_IPU to IPU DIR:to GP:C

DMAC_FROM_SPR from SPR DIR:from GP:C

DMAC_TO_SPR to SPR DIR:to GP:C

```
int iEnableDmac(
```

```
    int channel);
```

same as above

Calling conditions

EnableDmac Can be called from a thread

Multithread safe

iEnableDmac Can be called from an interrupt handler

Description

This system call enables DMA termination interrupts for the specified channel.

Return value

0 Already enabled

1 Changed to enabled

EnableIntc / iEnableIntc

Enable INTC interrupt

Library	Introduced	Documentation last modified
eekernel	1.1	March 26, 2001

Syntax

#include <eekernel.h>

int EnableIntc(

int <i>cause</i>);	Interrupt cause
	Constant Interrupt cause
	INTC_GS GS
	INTC_SBUS SBUS
	INTC_VBLANK_S V-blank start
	INTC_VBLANK_E V-blank end
	INTC_VIF0 VIF0
	INTC_VIF1 VIF1
	INTC_VU0 VU0
	INTC_VU1 VU1
	INTC_IPU IPU
	INTC_TIM0 Timer0
	INTC_TIM1 Timer1

int iEnableIntc(

int <i>cause</i>);	same as above
---------------------	---------------

Calling conditions

EnableIntc	Can be called from a thread
	Multithread safe
iEnableIntc	Can be called from an interrupt handler

Description

This system call enables interrupts triggered by *cause*.

Return value

- 0 Already enabled
- 1 Changed to enabled

ExitDeleteThread

Exit and delete calling thread

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	March 26, 2001

Syntax

```
#include <eekernel.h>
```

```
void ExitDeleteThread(void);
```

Calling conditions

Can be called from a thread

Multithread safe

Description

Exits and deletes the calling thread. This system call does not return a value since it does not return to the caller.

Semaphores and other resources are not released when the thread is exited.

Return value

None

ExitThread

Exit calling thread

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	March 26, 2001

Syntax

```
#include <eekernel.h>
```

```
void ExitThread(void);
```

Calling conditions

Can be called from a thread

Multithread safe

Description

Exits the calling thread and places it in DORMANT state. This system call does not return a value since it does not return to the caller.

Semaphores and other resources are not released when the thread is exited.

Return value

None

ExpandScratchPad

Expand Scratch Pad Virtually

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	2.0	October 11, 2001

Syntax

```
#include <eekernel.h>
```

```
int ExpandScratchPad(
```

```
    u_int page);
```

Starting physical address of the 8K-byte contiguous memory area that is aligned on a 4K-byte boundary

Calling conditions

Can be called from a thread

Multithread safe

Description

The memory area specified by the *page* argument is mapped to the logical address following the scratchpad (0x70000000 - 0x70003fff). If the value of the parameter is 0, the mapped region will be freed.

Return value

TLB index Normal termination

-1 Error

See also

InitTLBFunctions()

FlushCache / iFlushCache

Flush cache

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	March 26, 2001

Syntax

```
#include <eekernel.h>
```

```
void FlushCache(
```

```
    int operation);
```

Operation to be performed on cache

0x0 WRITEBACK_DCACHE:

Write back contents of data cache and invalidate

0x1 INVALIDATE_DCACHE:

Invalidate contents of data cache

0x2 INVALIDATE_ICACHE:

Invalidate contents of instruction cache

0x3 INVALIDATE_CACHE:

Invalidate contents of instruction/data cache

```
void iFlushCache(
```

```
    int operation);
```

same as above

Calling conditions

FlushCache Can be called from a thread

Multithread safe

iFlushCache Can be called from an interrupt handler

Description

This system call flushes the cache. The contents of the cache can be written back to memory or discarded.

Return value

None

See also

SyncDCache(), iSyncDCache()

GetCop0 / iGetCop0

Get COP0 register

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	March 26, 2001

Syntax

```
#include <eekernel.h>
```

```
u_int GetCop0(
```

```
    int id);
```

Register number (0 - 31)

```
u_int iGetCop0(
```

```
    int id);
```

Register number (0 - 31)

Calling conditions

GetCop0 Can be called from a thread

Multithread safe

iGetCop0 Can be called from an interrupt handler

Description

Returns the value of the COP0 register having the specified *id*.

Return value

Register value

Normal termination

GetThreadId

Get ID of calling thread

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	March 26, 2001

Syntax

```
#include <eekernel.h>
```

```
int GetThreadId(void);
```

Calling conditions

Can be called from a thread

Multithread safe

Description

Gets the thread ID of the calling thread.

Return value

Thread ID Normal termination

InitThread

Initialize thread

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	2.0	March 26, 2001

Syntax

```
int InitThread (void);
```

Calling conditions

Can be called from a thread

Multithread safe

Description

A thread is created for scheduling and the thread ID is returned.

If the priority of the thread making this system call is 0, it will be set to 1. The priority of the scheduling thread will be set to 0.

This function is provided for dealing with problems in which the iWakeupThread(), iRotateThreadReadyQueue(), or iSuspendThread() function is not properly scheduled. It uses one semaphore.

To support the C++ constructor, this function was changed so that it is called by crt0.s. The user need not directly call it.

Return value

Thread ID Normal termination

-1 Initialization failed

InitTLBFunctions

ExpandScratchPad() initialization

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	2.1	October 11, 2001

Syntax

```
#include <eekernel.h>
```

```
void InitTLBFunctions (void);
```

Calling conditions

Can be called from a thread

Multithread safe

Description

This function performs ExpandScratchPad() initialization. This function can now be called in crt0.s, so there is no need for the user to call it directly.

Return value

None

See also

ExpandScratchPad()

Interrupt2lop

Interrupt the IOP

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	March 26, 2001

Syntax

```
#include <eekernel.h>
```

```
int Interrupt2lop(  
    int cause);
```

Interrupt cause

Calling conditions

Can be called from a thread

Multithread safe

Description

Sends an interrupt to the IOP with the specified cause. The value of *cause* is user-defined and can range from 0 to 15.

Return value

cause Normal termination

-1 Error

InvalidDCache / iInvalidDCache

Invalidate cache

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	July 2, 2001

Syntax

```
#include <eekernel.h>
```

```
void InvalidDCache(
```

```
    void *begin,           Beginning address
```

```
    void *end);           Ending address
```

```
void iInvalidDCache(
```

```
    void *begin,           Beginning address
```

```
    void *end);           Ending address
```

Calling conditions

InvalidDCache Can be called from a thread

 Multithread safe

iInvalidDCache Can be called from an interrupt handler

Description

If the contents of memory from the *begin* logical address up to and including the *end* logical address exist in the D-cache, this function invalidates that cache line.

Return value

None

ReferSemaStatus / iReferSemaStatus

Get Semaphore Status

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	March 26, 2001

Syntax

```
#include <eekernel.h>
```

```
int ReferSemaStatus(
```

```
    int sid,                               Semaphore ID
```

```
    SemaParam *sema);                     Pointer to a structure which will contain the semaphore status.
```

```
int iReferSemaStatus(
```

```
    int sid,                               Semaphore ID
```

```
    SemaParam *sema);                     Pointer to a structure which will contain the semaphore status.
```

Calling conditions

ReferSemaStatus Can be called from a thread

Multithread safe

iReferSemaStatus Can be called from an interrupt handler

Description

Gets the status of the specified semaphore.

Return value

semaphore ID	Normal termination
-1	Error
sema->currentCount	Semaphore current value.
sema->initCount	Semaphore initial value.
sema->maxCount	Semaphore maximum value.
sema->numWaitThreads	Number of semaphore wait threads.
sema->option	Additional information related to the semaphore. Specified using CreateSema.

ReferThreadStatus / iReferThreadStatus

Get thread status

Library	Introduced	Documentation last modified
eekernel	1.1	March 26, 2001

Syntax

```

#include <eekernel.h>
int ReferThreadStatus(
    int tid,                                Thread ID to be referenced
    ThreadParam *info);                    Pointer to a structure for holding thread status. Set to
                                          NULL if only the return value is needed.

int iReferThreadStatus(
    int tid,                                Thread ID to be referenced
    ThreadParam *info);                    Pointer to a structure for holding thread status. Set to
                                          NULL if only the return value is needed.

```

Calling conditions

ReferThreadStatus	Can be called from a thread
	Multithread safe
iReferThreadStatus	Can be called from an interrupt handler

Description

Gets the status of the thread specified by *tid*.

Notes

The ReferThreadStatus system call is provided for debugging purposes. Other uses of this function are not recommended.

Return value

Constants displaying thread status	Normal termination
info->currentPriority	Priority of current thread
info->status	Thread status is expressed with the following constants
	0x01 THS_RUN RUN state
	0x02 THS_READY READY state
	0x04 THS_WAIT WAIT state
	0x08 THS_SUSPEND SUSPEND state
	0x0c THS_WAITSPEND WAIT-SUSPEND state
	0x10 THS_DORMANT DORMANT state

ReleaseAlarm / iReleaseAlarm

Release alarm

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	July 2, 2001

Syntax

```
#include <eekernel.h>
```

```
int ReleaseAlarm(
```

```
    int id);           ID of alarm to be released
```

```
int iReleaseAlarm(
```

```
    int id);           ID of alarm to be released
```

Calling conditions

ReleaseAlarm	Can be called from a thread
	Multithread safe
iReleaseAlarm	Can be called from an interrupt handler

Description

This system call releases the specified alarm.

If an alarm which is already going to be called within the interrupt handler is released with iReleaseAlarm, the system call will fail.

Return value

Elapsed time after alarm is set (Hsync count)

-1 if release failed

ReleaseWaitThread / iReleaseWaitThread

Release thread in WAIT state

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	March 26, 2001

Syntax

```
#include <eekernel.h>
```

```
int ReleaseWaitThread(  
    int tid);
```

Thread ID

```
int iReleaseWaitThread(  
    int tid);
```

Thread ID

Calling conditions

ReleaseWaitThread	Can be called from a thread
	Multithread safe
iReleaseWaitThread	Can be called from an interrupt handler

Description

Releases the specified thread, which is currently in WAIT state.

Return value

Thread ID	Normal termination
-1	Error

RemoveDmacHandler

Remove DMA interrupt handler

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	March 26, 2001

Syntax

```
#include <eekernel.h>
```

```
int RemoveDmacHandler(
```

```
    int channel,
```

Channel number

Constant

DMA channel

DMAC_VIF0

VIF0

DIR:to

GP:A

DMAC_VIF1

VIF1

DIR:both

GP:C

DMAC_GIF

GIF

DIR:to

GP:C

DMAC_FROM_IPU

from IPU

DIR:from

GP:C

DMAC_TO_IPU

to IPU

DIR:to

GP:C

DMAC_FROM_SPR

from SPR

DIR:from

GP:C

DMAC_TO_SPR

to SPR

DIR:to

GP:C

```
    int hid);
```

Handler ID

Calling conditions

Can be called from a thread

Multithread safe

Description

Removes the specified *hid* handler from the DMA interrupt handlers for the specified *channel*. An error is returned if *the* hid handler is not registered.

Return value

Number of registered handlers

Normal termination

-1

Error

RemoveIntcHandler

Remove interrupt handler

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	March 26, 2001

Syntax

```
#include <eekernel.h>
int RemoveIntcHandler(
    int cause,                Interrupt cause
                               Constant          Interrupt cause
                               INTC_GS           GS
                               INTC_SBUS        SBUS
                               INTC_VBLANK_S    V-blank start
                               INTC_VBLANK_E    V-blank end
                               INTC_VIF0        VIF0
                               INTC_VIF1        VIF1
                               INTC_VU0         VU0
                               INTC_VU1         VU1
                               INTC_IPU         IPU
                               INTC_TIM0        Timer0
                               INTC_TIM1        Timer1
    int hid);                Handler ID
```

Calling conditions

Can be called from a thread
Multithread safe

Description

Removes the specified *hid* handler from the interrupt handlers for the specified *cause*. An error is returned if *hid* handler is not registered.

Return value

Number of registered handlers	Normal termination
-1	Error

RemoveSbusIntcHandler

Remove SBUS interrupt handler

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	March 26, 2001

Syntax

```
#include <eekernel.h>
int RemoveSbusIntcHandler(
    int cause);
```

Interrupt cause

Calling conditions

Can be called from a thread

Multithread safe

Description

Removes the SBUS interrupt handler having the specified *cause*. If the handler does not exist, an error is returned.

Return value

cause	Normal return
-1	Error

ResumeThread / iResumeThread

Move thread from SUSPEND state to READY state

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	March 26, 2001

Syntax

```
#include <eekernel.h>
```

```
int ResumeThread(  
    int tid);                Thread ID
```

```
int iResumeThread(  
    int tid);                Thread ID
```

Calling conditions

ResumeThread Can be called from a thread
 Multithread safe

iResumeThread Can be called from an interrupt handler

Description

Moves the specified thread from SUSPEND state to READY state.

Notes

The ResumeThread system call cannot specify the calling thread.

Return value

Thread ID Normal termination

-1 Error

RotateThreadReadyQueue / iRotateThreadReadyQueue

Rotate ready queue

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	March 26, 2001

Syntax

```
#include <eekernel.h>
```

```
int RotateThreadReadyQueue(
```

```
    int prio);
```

Priority level of the target queue (1 – 127)

```
int RotateThreadReadyQueue(
```

```
    int prio);
```

Priority level of the target queue (1 – 127)

Calling conditions

RotateThreadReadyQueue	Can be called from a thread
	Multithread safe
iRotateThreadReadyQueue	Can be called from an interrupt handler

Description

Rotates the ready queue at the specified priority level.

Return value

Priority	Normal termination
-1	Error

SetAlarm / iSetAlarm

Set alarm

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	July 2, 2001

Syntax

```
#include <eekernel.h>
```

```
int SetAlarm(
```

```
    u_short time,
```

Time (in Hsyncs)

```
    void *cbfunc(int, u_short, void *),
```

Address of callback function

```
        void cbfunc(int id, u_short time, void *arg)
```

```
        id : Alarm id
```

```
        time : Hsync count for timeout
```

```
        arg : Third argument for SetAlarm
```

```
void *arg);
```

Argument to be passed to cbfunc

```
int iSetAlarm(
```

```
    u_short time,
```

Time (in Hsyncs)

```
    void *cbfunc(int, u_short, void *),
```

Address of callback function

```
        void cbfunc(int id, u_short time, void *arg)
```

```
        id : Alarm id
```

```
        time : Hsync count for timeout
```

```
        arg : Third argument for SetAlarm
```

```
void *arg);
```

Argument to be passed to cbfunc

Calling conditions

SetAlarm Can be called from a thread

Multithread safe

iSetAlarm Can be called from an interrupt handler

Description

Specifies a time interval and conditions in which a callback function, *cbfunc*, will be called.

cbfunc will be called if ReleaseAlarm is not called within the time specified by time, after SetAlarm is called. (During this time the alarm will be released.)

A maximum of 64 alarms can be simultaneously set. The kernel reserves one HSync Timer (TIMER 3).

Notes

The alarm interrupt is generated in minimum units of 1Hsync. Hence, the handler process that is set by SetAlarm/iSetAlarm must be requested within 1Hsync.

Multiple alarms that have been set at the same time are called by one interrupt.

If there is an alarm set when processing time is long, that alarm is guaranteed to be called until such a time when the next alarm would be called.

Do not use scePrintf within an alarm except for the purposes of debugging. This is because scePrintf alone requires several Hsync.

Return value

ID of alarm that was set

-1 if alarm could not be set

SetDebugHandler

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	March 26, 2001

Syntax

```
#include <eekernel.h>
```

```
void *SetDebugHandler(
```

```
int code,
```

Exception Number

- 1 TLB change exception
- 2 TLB mismatch exception
(load or instruction fetch)
- 3 TLB mismatch exception (store)
- 4 Address error exception
(load or instruction fetch)
- 5 Address error exception (store)
- 6 Bus error exception (instruction fetch)
- 7 Bus error exception (data load or store)
- 8 System call exception
- 9 Breakpoint exception
- 10 Reserved instruction exception
- 11 Coprocessor unused exception
- 12 Calculation overflow exception
- 13 Trap exception

```
void (*handler)
```

Handler function

```
(u_int stat, u_int cause, u_int epc, u_int bva,  
u_int bpa, u_long128 *gpr)
```

Calling conditions

Can be called from a thread

Multithread safe

Description

Sets the exception handler that is called when a CPU exception occurs. Processing cannot return from the called handler.

When *code* is the exception number and *handler* is the address of the handler that is called when the exception occurs, the arguments of *handler* indicate the state when the exception occurred. The argument *stat* is the COP0 status register, the argument *cause* is the COP0 *cause* register, the argument *epc* is the program counter value when the exception occurred, the argument *bva* is the logical address value when a memory access or branch address is invalid, the argument *bpa* is the physical address on a bus error termination, and the argument *gpr* is the GPR value in an array of size 32.

Notes

This function is a debugging function. Use it to display the exception contents on the screen when an exception occurs.

Return Value

Previously registered handler address	Normal termination
-1	Error termination

SignalSema / iSignalSema

Release semaphore resource

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	March 26, 2001

Syntax

```
#include <eekernel.h>
```

```
int SignalSema(
    int sid);                Semaphore ID
int iSignalSema(int sid);
int sid);                Semaphore ID
```

Calling conditions

SignalSema	Can be called from a thread
	Multithread safe
iSignalSema	Can be called from an interrupt handler

Description

Releases the specified semaphore resource.

When the value of *sid* is 0 and there is no free space in the semaphore queue, the thread at the start of the semaphore's wait queue will be released and placed in READY state. In all other cases, the value of the semaphore is incremented.

Return value

Semaphore ID	Normal termination
-1	Error

SleepThread

Change to WAIT state

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	March 26, 2001

Syntax

```
#include <eekernel.h>
```

```
int SleepThread(void);
```

Calling conditions

Can be called from a thread

Multithread safe

Description

Places the calling thread in WAIT state. However, if the wake-up request count is greater than or equal to 1, this system call will only decrement the count and the thread state will remain unchanged.

Threads in WAIT state are released using WakeupThread or ReleaseWaitThread.

Return value

ID of calling thread Normal termination

StartThread

Start thread

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	March 26, 2001

Syntax

```
#include <eekernel.h>
```

```
int StartThread(
```

```
    int tid,                                ID of thread to be started
```

```
    void *arg);                             Argument
```

Calling conditions

Can be called from a thread

Multithread safe

Description

Starts the specified thread and initializes the stack. The thread to be started must be in DORMANT state.

Return value

Thread ID Normal termination

-1 Error

SuspendThread / iSuspendThread

Put thread in SUSPEND state

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	March 26, 2001

Syntax

```
#include <eekernel.h>
```

```
int SuspendThread(  
    int tid);                Thread ID
```

```
int iSuspendThread(  
    int tid);                Thread ID
```

Calling conditions

SuspendThread Can be called from a thread
 Multithread safe

iSuspendThread Can be called from an interrupt handler

Description

Puts the specified thread in SUSPEND state and suspends its execution. SUSPEND state can be released using the ResumeThread system call.

The state transitions resulting from SuspendThread and ResumeThread are shown below.

- If the thread is already in WAIT state, it will be placed in WAIT-SUSPEND state, which is a combination of WAIT and SUSPEND states.
- If the thread is in WAIT-SUSPEND state and the conditions for releasing WAIT state are met, the thread will be placed in SUSPEND state.
- If ResumeThread is issued on a thread in WAIT-SUSPEND state, the thread is put back in WAIT state.

Notes

A thread enters SUSPEND state when it becomes suspended by a system call issued from another thread. The SuspendThread system call cannot be used to suspend the calling thread. The SuspendThread system call is provided primarily for debugging purposes.

Return value

Thread ID Normal termination

-1 Error

SyncDCache / iSyncDCache

Flush cache

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	July 2, 2001

Syntax

```
#include <eekernel.h>
```

```
void SyncDCache(
```

```
    void *begin,           Starting address
```

```
    void *end);           Ending address
```

```
void iSyncDCache(
```

```
    void *begin,           Starting address
```

```
    void *end);           Ending address
```

Calling conditions

SyncDCache	Can be called from a thread
	Multithread safe
iSyncDCache	Can be called from an interrupt handler

Description

Writes back the contents of the D-cache, from logical memory address *begin* up to and including logical memory address *end*, if the data is resident in the cache. Invalidates that cache line.

Return value

None

See also

FlushCache(), iFlushCache()

TerminateThread / iTerminateThread

Terminate thread

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	March 26, 2001

Syntax

```
#include <eekernel.h>
```

```
int TerminateThread(  
    int tid);
```

ID of thread to terminate

```
int iTerminateThread(  
    int tid);
```

ID of thread to terminate

Calling conditions

TerminateThread Can be called from a thread

Multithread safe

iTerminateThread Can be called from an interrupt handler

Description

Terminates the specified thread and places it in DORMANT state.

The *tid* is returned on successful termination.

If the thread is in WAIT or SUSPEND state rather than being in READY state, the thread will be released from its current state and terminated. If the thread is already in DORMANT state, an error is returned.

Notes

The calling thread cannot be terminated.

Return value

Thread ID Normal termination

-1 Error

WaitSema / PollSema / iPollSema

Get semaphore resource

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	March 26, 2001

Syntax

```
#include <eekernel.h>
```

```
int WaitSema(
    int sid);                Semaphore ID

int PollSema(
    int sid);                Semaphore ID

int iPollSema(
    int sid);                Semaphore ID
```

Calling conditions

WaitSema / PollSema	Can be called from a thread
	Multithread safe
iPollSema	Can be called from an interrupt handler

Description

If the value of the semaphore specified by sid is greater than 1, the semaphore is decremented and the function immediately terminates with a normal return. If the value of the semaphore is zero, the thread is put in WAIT state and placed in the semaphore's wait queue until another thread increments the semaphore by issuing a SignalSema.

The PollSema system call is the same as WaitSema except that the thread does not enter WAIT state. Unlike WaitSema, an error will be returned if the value of the semaphore is 0.

Return value

Semaphore ID	Normal termination
-1	Error

WakeupThread / iWakeupThread

Wake up thread in WAIT state

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	March 26, 2001

Syntax

```
#include <eekernel.h>
```

```
int WakeupThread(  
    int tid);                Thread ID
```

```
int iWakeupThread(  
    int tid);                Thread ID
```

Calling conditions

WakeupThread Can be called from a thread
 Multithread safe

iWakeupThread Can be called from an interrupt handler

Description

Moves the specified thread from WAIT state to READY state. If the thread is not in WAIT state, the wakeup request count is incremented so that the thread will not go into WAIT state even if SleepThread is subsequently called for the thread.

Return value

Thread ID Normal termination

-1 Error

Program Load/Exit Functions

Exit

Exit program

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	March 26, 2001

Syntax

```
#include <eekernel.h>
```

```
void Exit(
```

```
int n);
```

Exit code to be passed to the execution environment of the program (currently the debugger)

Calling conditions

Can be called from a thread

Multithread safe

Description

This function causes the program to exit.

Notes

This function is for debugging use only. Using it in programs included on a master disc is absolutely prohibited.

Return value

None

LoadExecPS2

Load program

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
eekernel	1.1	March 26, 2001

Syntax

```
#include <eekernel.h>
```

```
void LoadExecPS2(
```

```
    const char *file,
```

Filename

```
    int n,
```

Number of arguments (n <= 15)

```
    char **args);
```

Argument array

Calling conditions

Can be called from a thread

Multithread safe

Description

Loads the program specified by *file*.

The filename is passed to the program's main function as an argument.

Program example:

```
char *args[] = { "arg1", "arg2" };
LoadExecPS2("cdrom0:\\program.elf", 2, args);
```

For the code shown above, the arguments of the program's main function are:

```
argc = 3
```

```
argv = { "cdrom0:\\program.elf", "arg1", "arg2" }
```

Notes

The total size of the character strings for the filename and all of the arguments must not exceed 256 bytes.

Return value

None

Chapter 2: Standard I/O Service Functions (for EE)

Table of Contents

Structures	2-3
sce_dirent	2-3
sce_stat	2-4
Functions	2-6
sceChdir	2-6
sceChstat	2-7
sceClose	2-8
sceDclose	2-9
sceDevctl	2-10
sceDopen	2-11
sceDread	2-12
sceFormat	2-13
sceFsReset	2-14
sceGetstat	2-15
scelectl	2-16
scelectl2	2-17
sceLseek	2-18
sceLseek64	2-19
sceMkdir	2-20
sceMount	2-21
sceOpen	2-22
scePowerOffHandler	2-24
scePrintf	2-25
sceRead	2-26
sceReadlink	2-27
sceRemove	2-28
sceRename	2-29
sceRmdir	2-30
sceSymlink	2-31
sceSync	2-32
sceUmount	2-33
sceWrite	2-34

Structures

sce_dirent

Directory entries

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
libio	2.4	October 11, 2001

Structure

```
struct sce_dirent {  
    struct sce_stat d_stat;           File status  
    char d_name[256];                Filename  
    void *d_private);                Reserved
```

Description

This structure is used to store directory entries.

See also

sceDread()

sce_stat

File status

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
libio	2.4	October 11, 2001

Structure**struct sce_stat {****unsigned int** *st_mode*;

File mode

bit 0 other execute permission

bit 1 other write permission

bit 2 other read permission

bit 3 group execute permission

bit 4 group write permission

bit 5 group read permission

bit 6 user execute permission

bit 7 user write permission

bit 8 user read permission

bit 9 Reserved

bit 10 Reserved

bit 11 Reserved

bits 12-15 File type

1 Directory

2 Normal file

4 Symbolic link

unsigned int *st_attr*;

Memory card mode compatibility flag

unsigned int *st_size*;

File size (64 bits)

unsigned char *st_ctime*[8];

Creation time

unsigned char *st_atime*[8];

Last access time; updated with same timing as last modification time.

unsigned char *st_mtime*[8];

Last modification time

byte 0 Reserved

byte 1 Second

byte 2 Minute

byte 3 Hour

byte 4 Day

byte 5 Month

byte 6-7 Year (4 digits)

unsigned int *st_hisize*;**unsigned int** *st_private*[6];

word 0 uid

word 1 gid

word 2 Number of zones used by file

Description

This structure is used to save file status.

See also

struct sce_dirent

sceGetstat()

Functions

sceChdir

Change current directory

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
libio	2.4	October 11, 2001

Syntax

```
#include <sifdev.h>
```

```
int sceChdir(  
    const char *name)           File pathname
```

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupted-enabled state)

Description

This function changes the current directory.

Return value

If processing is successful, zero is returned.

On error, the value in errno multiplied by -1 is returned.

sceChstat

Change file/directory status

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
libio	2.4	October 11, 2001

Syntax

```
#include <sifdev.h>
```

```
int sceChstat(
```

```
    const char *name,
```

File pathname (including device name + "：")

```
    struct sce_stat *buf,
```

Buffer for storing status

```
    unsigned int cbit)
```

Macro for specifying fields to be changed. Any of the following constants can be specified.

SCE_CST_MODE

SCE_CST_ATTR

SCE_CST_SIZE

SCE_CST_CT

SCE_CST_AT

SCE_CST_MT

SCE_CST_PRVT

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupted-enabled state)

Description

This function changes the status of the specified file/directory. The members of the `sce_stat` structure that can be changed are the file mode bits other than the file type bits, the various times, and the memory card compatibility flag bits other than the subdirectory bit and close completion flag.

Return value

If processing is successful, zero is returned.

On error, the value in `errno` multiplied by -1 is returned.

sceClose

Close file

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
libio	1.1	October 11, 2001

Syntax

```
#include <sifdev.h>
```

```
int sceClose(
```

```
int fd)
```

File descriptor of a previously opened file

Calling conditions

Can be called from a thread

Multithread safe (must be called in an interrupt-enabled state)

Description

This function closes a file that had been opened and frees its file descriptor.

Return value

Returns 0 if successful.

If an error occurred, -1 multiplied by errno is returned.

sceDclose

Close directory

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
libio	2.4	October 11, 2001

Syntax

#include <sifdev.h>

int sceDclose(

int *fd*) File descriptor**Calling conditions**

Can be called from a thread

Multithread safe (must be called in interrupted-enabled state)

Description

This function closes a directory that had been opened and frees its file descriptor.

Return value

If processing is successful, zero is returned.

On error, the value in errno multiplied by -1 is returned.

sceDevctl

Special operations on device

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
libio	2.3.4	August 31, 2001

Syntax

```
#include <sifdev.h>
```

```
int sceDevctl(
```

<code>const char *name,</code>	Filesystem device name
<code>int cmd,</code>	Operation command. See document for each device.
<code>void *arg,</code>	Argument assigned to command. cmd-dependent.
<code>unsigned int arglen,</code>	Size of arg
<code>void *bufp,</code>	Argument received from command. cmd-dependent.
<code>unsigned int buflen)</code>	Size of bufp

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

This function performs special operations on a device. For details of each cmd, refer to the documentation for each device.

Return value

If processing succeeds, a cmd-dependent value is returned. If an error occurs, the product of errno and -1 is returned.

sceDread

Read directory entries

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
libio	2.4	October 11, 2001

Syntax

```
#include <sifdev.h>
```

```
int sceDread(
```

```
    int fd,                                File descriptor
```

```
    struct sce_dirent *buf)                Address of buffer for storing data that was read
```

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupted-enabled state)

Description

This function copies the next entry in the directory entry stream indicated by fd to the sce_dirent structure buf. When the last entry is reached, this function returns zero.

Return value

When processing is successful, the filename length is returned. If the last entry is reached, zero is returned.

On error, the value in errno multiplied by -1 is returned.

sceFormat

Format filesystem

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
libio	2.4	October 11, 2001

Syntax

```
#include <sifdev.h>
```

```
int sceFormat(
```

<code>const char *devname,</code>	Device name of filesystem
<code>const char *blockdevname,</code>	Block device name of previously created partition
<code>void *arg,</code>	Pointer to zone size variable and fragment option
<code>int arglen)</code>	Size of arg

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupted-enabled state)

Description

This function builds a new filesystem.

For details, refer to the documentation for each device.

Return value

If processing is successful, zero is returned.

On error, the value in errno multiplied by -1 is returned.

sceFsReset

Invalidate file service bind information

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
libio	1.1	March 26, 2001

Syntax

```
#include <sifdev.h>
```

```
int sceFsReset(void )
```

Calling conditions

Can be called from a thread

Multithread safe (must be called in an interrupt-enabled state)

Description

This function should always be called after the IOP is reset since the file service RPC BIND information will be invalid.

Return value

Currently, 0 is always returned.

sceGetstat

Get file/directory status

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
libio	2.4	October 11, 2001

Syntax

```
#include <sifdev.h>
```

```
int sceGetstat(
```

```
    const char *name,                File pathname (includes device name + ":")
```

```
    struct sce_stat *buf)            Buffer for storing status
```

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupted-enabled state)

Description

This function copies file information to the sce_dirent structure buf. The file pathname is specified as device name + unit number + ":" + string.

Return value

If processing is successful, zero is returned.

On error, the value in errno multiplied by -1 is returned.

sceloctl

Special file operations

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
libio	1.1	October 11, 2001

Syntax

```
#include <sifdev.h>
```

```
int sceloctl(
```

```
    int fd,
```

File descriptor of target file

```
    int cmd,
```

Operation command. Refer to the documentation for each device.

```
    void *arg)
```

Argument to be assigned to command. cmd-dependent.

Calling conditions

Can be called from a thread

Multithread safe (must be called in an interrupt-enabled state)

Description

Various operations are performed on a file (device) that cannot be performed with sceRead/sceWrite. The operations are device-dependent. In the current implementation, 1024 bytes from the arg address is passed to the IOP and this is directly passed to ioctl() on the iop side.

Currently, only the following commands are implemented by sceloctl():

SCE_FS_EXECUTING

Argument

int *is_end

Example

```
int is_end;
sceloctl( fd, SCE_FS_EXECUTING, &is_end);
if (is_end) printf("Now Executing...\n");
else printf("Execute is end.\n");
```

Description

Checks to see if operations performed on a file opened with SCE_NOWAIT have completed. If the parameter value is 1, operations are still being performed. If 0, the operation has completed.

If SCE_FS_EXECUTING is used for polling in a multithreaded environment, the operation of other threads may be significantly hindered. As a result, the priorities of threads and the return of system resources during polling must be thoroughly considered.

Return value

Returns 0 if successful.

If an error occurred, -1 multiplied by errno is returned.

scelctl2

Special operations on a file

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
libio	2.4	October 11, 2001

Syntax

#include <sifdev.h>

int scelctl2(

int *fd*,

File descriptor of target file

int *cmd*,

Operation command. Refer to the documentation for each device.

void **arg*,

Argument to be assigned to command. cmd-dependent.

unsigned int *arglen*,

Size of arg.

void **bufp*,

Argument received from command. cmd-dependent.

unsigned int *buflen*)

Size of bufp.

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupted-enabled state)

Description

This command performs special operations on a file. For details of each cmd, refer to the documentation for each device.

Return value

If processing is successful, a cmd-dependent value is returned.

On error, the value in errno multiplied by -1 is returned.

sceLseek

Move file pointer

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
libio	2.4	October 11, 2001

Syntax**#include <sifdev.h>****int sceLseek(****int *fd*,**

File descriptor

int *offset*,

Distance pointer is to be moved

int *whence*)

Reference position for offset within partition's extended attribute area. Any of the following constants can be specified.

SCE_SEEK_SET Starting position

SCE_SEEK_CUR Current position

SCE_SEEK_END End

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

DescriptionThis function changes the offset of the file descriptor *fd* to the offset position according to *whence*.

The offset cannot be set to a location beyond the end of the file.

For a descriptor for which `sceOpen()` was executed with `SCE_NOWAIT`, the return value can only be used to learn whether or not the command was successful.**Return value**

If processing is successful, the value of the newly set file pointer is returned.

On error, the value in `errno` multiplied by -1 is returned.

sceLseek64

Move file pointer (64-bit support)

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
libio	2.4	October 11, 2001

Syntax

```
#include <sifdev.h>
```

```
int sceLseek64(
```

```
    int fd,
```

File descriptor

```
    long offset,
```

Distance pointer is to be moved

```
    int whence)
```

Reference position for offset within partition's extended attribute area. Any of the following constants can be specified.

SCE_SEEK_SET Starting position

SCE_SEEK_CUR Current position

SCE_SEEK_END End

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupted-enabled state)

Description

This function changes the offset of the file descriptor `fd` to the offset position according to `whence`. The offset cannot be set to a location beyond the end of the file. This function supports 64-bit file sizes. For a descriptor for which `sceOpen()` was executed with `SCE_NOWAIT`, the return value can only be used to learn whether or not the command was successful.

Notes

To use this function, the `-fno-strict-aliasing` option must be specified during compilation.

Return value

If processing is successful, the value of the newly set file pointer is returned.

On error, the value in `errno` multiplied by -1 is returned.

sceMkdir

Create directory

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
libio	2.4	October 11, 2001

Syntax

#include <sifdev.h>

int sceMkdir(

const char *name,

int mode)

Directory pathname (includes device name + ":")

File mode

bit 0 other execute permission

bit 1 other write permission

bit 2 other read permission

bit 3 group execute permission

bit 4 group write permission

bit 5 group read permission

bit 6 user execute permission

bit 7 user write permission

bit 8 user read permission

bit 9 Reserved

bit 10 Reserved

bit 11 Reserved

Although macros for each mode are provided in sifdev.h, the mode can also be easily specified in octal, e.g. 0777 or 0755.

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupted-enabled state)

Description

This function creates a directory. The pathname should be specified as device name + unit number + ":" + string.

Notes

For details, refer to the documentation for each device.

Return value

If processing is successful, zero is returned.

On error, the value in errno multiplied by -1 is returned.

sceMount

Mount device

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
libio	2.4	October 11, 2001

Syntax

```
#include <sifdev.h>
```

```
int sceMount(
```

```
    const char *fsname,
```

String for specifying mounted filesystem device name and unit number

```
    const char *devname,
```

Device identification string required to open block device to be mounted

```
    int flag,
```

Mount flag. Any of the following constants can be specified.

Multiple flags can be specified by taking the logical OR of the flags.

SCE_MT_RDWR Mount in read/write mode.

SCE_MT_RDONLY Mount in read-only mode.

SCE_MT_ROBUST Mount in robust mode.

SCE_MT_ERRCHECK Return an error if a problem is detected in the filesystem during mounting.

```
    void *arg,
```

Reserved

```
    int arglen)
```

Size of arg

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupted-enabled state)

Description

This function mounts the block device specified by devname on the file system logical device specified by fsname. For details, refer to the documentation for each device.

Return value

If processing is successful, zero is returned.

On error, the value in errno multiplied by -1 is returned.

sceOpen

Create and open file

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
libio	2.4	October 11, 2001

Syntax

```
#include <sifdev.h>
```

```
int sceOpen(
```

```
    const char *name,
```

```
    int flags,
```

```
    unsigned short mode)
```

File pathname (includes device name + ":")

Access mode. Any of the following constants can be specified.

Multiple flags can be specified by taking the logical OR of the flags.

SCE_RDONLY Open in read-only mode.

SCE_WRONLY Open in write-only mode.

SCE_RDWR Open in read/write mode.

SCE_APPEND Set append write mode. Data is always written at end of file.

SCE_CREAT Create new file if file does not exist.

SCE_TRUNC Discard existing file contents.

SCE_EXCL If file with same name exists for which O_CREAT was specified, an error will occur.

SCE_NOWAIT Open in no-wait mode. (This should not be used in a multithread environment.)

SCE_NOWBDC Do not perform D-cache write back.

File mode

bit 0 other execute permission

bit 1 other write permission

bit 2 other read permission

bit 3 group execute permission

bit 4 group write permission

bit 5 group read permission

bit 6 user execute permission

bit 7 user write permission

bit 8 user read permission

bit 9 Reserved

bit 10 Reserved

bit 11 Reserved

Although macros for each mode are provided in sifdev.h, the mode can also be easily specified in octal, e.g. 0777 or 0755.

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupted-enabled state)

Description

This function creates and opens a file. A file descriptor is assigned to the file that was opened. The file pathname should be specified as device name + unit number + ":" + string.

If SCE_NOWAIT is specified for sceOpen, the function will exit without waiting for completion. The completion of the function can be verified with an sceIoctl SCE_FS_EXECUTING request. (However, SCE_NOWAIT should not be specified when sceOpen is executed in a multithread environment.)

For details, refer to the documentation for each device.

Return value

If processing completes normally, the file descriptor (0 or greater) is returned.

On error, the value in errno multiplied by -1 is returned.

For other errors, the following is returned.

SCE_EVERSIONMISS	Mismatch in version of IOP module
------------------	-----------------------------------

scePowerOffHandler

Define PlayStation 2 power off handler

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
libio	2.3.4	August 31, 2001

Syntax

```
int scePowerOffHandler (
void (*func)(void *)           Address of callback function
void *addr)                   Address of callback arguments
```

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupt-enabled state)

Description

A title that supports the hard disk drive must perform hard disk power-off processing.

This function sets the callback func that is to be called when the power-off operation is performed.

Once the callback is set, the function func will be called when the power-off operation is performed.

The function func is called by an interrupt handler.

If 0 is specified for func, no callback will be generated.

In terms of functionality, this function is the same as the sceCdPOffCallback() function in the libcdvd library. However, because the sceCdPOffCallback() function becomes unusable when cdvdfsv.irx has been unloaded, this function is used instead.

Notes

Sample power-off processing function calling sequence when hard disk drive is used

```
printf("power off request has come.\n");
/* close all files */
sceDevctl("pfs:", PDIOC_CLOSEALL, NULL, 0, NULL, 0);
/* dev9 power off, need to power off PS2 */
while(sceDevctl("hdd:", HDIOC_DEV9OFF, NULL, 0, NULL, 0) < 0);
/* PS2 power off */
while(sceDevctl("cdrom0:", CDIOC_POWEROFF, NULL, 0, NULL, 0) < 0);
while(1);
```

Return value

Address of callback function that was set previously. If no callback function was set, zero is returned.

scePrintf

Simple printf

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
libio	1.1	March 26, 2001

Syntax

```
#include <eekernel.h>
```

```
void scePrintf(
```

```
    const char *fmt,
```

Output format

```
    ...)
```

Variable arguments

Calling conditions

Can be called from an interrupt handler

Not multithread safe (must be called in an interrupt-disabled state)

Description

This is a simple output function that can be used when interrupt handlers and interrupts are disabled.

It should not be used when interrupt handlers and interrupts are enabled. This function does not use malloc.

Restrictions are given below.

When the user protocol driver of DECI2 is operating, timing may not allow the program to continue. When this happens, a message will be output and the program will terminate. Because the standard I/O driver is also implemented as the user protocol driver of DECI2, the above effect can occur when character input is performed from the debugger and console program.

Only the formats listed below are supported. Only zero padding is permitted between the '%' and the conversion characters.

Format

d	signed decimal
o	unsigned octal
u	unsigned decimal
x	unsigned hex
f	single-precision floating point (*)
s	string
c	character

(*) Because the design is such that memory efficiency takes priority, precision cannot be guaranteed.

Return value

None

sceRead

Read file

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
libio	1.1	October 11, 2001

Syntax

```
#include <sifdev.h>
```

```
int sceRead(
```

```
    int fd,
```

File descriptor of file to be read

```
    void *buf,
```

Address of buffer for storing data that was read

```
    int count)
```

Size of data to be read

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupted-enabled state)

Description

This function reads at most *count* bytes from a previously opened file into the buffer that begins at *buf*. Performance will decrease if the buffer is not aligned on a 64-byte boundary. Therefore, 64-byte alignment is recommended. For details, refer to the documentation for each device.

Notes

The SPR cannot be specified for *buf*.

Return value

If processing is successful, the number of bytes that were read is returned. The file position advances by this number.

On error, the value in *errno* multiplied by -1 is returned.

sceReadlink

Read value of symbolic link

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
libio	2.4	October 11, 2001

Syntax

```
#include <sifdev.h>
```

```
int sceReadlink(
```

<code>const char *path,</code>	File pathname
<code>char *buf,</code>	Buffer for writing contents
<code>unsigned int bufsiz)</code>	Size of buf (max value 1023 bytes)

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupted-enabled state)

Description

This function stores the contents of the symbolic link given by path in buf and uses bufsiz to set the size of buf. sceReadlink does not add null characters (NUL) to buf. If the buffer is too small to store all of the contents, the contents are truncated (to a size of bufsiz bytes).

Return value

If processing is successful, the number of characters that were stored in the buffer is returned.

On error, the value in errno multiplied by -1 is returned.

sceRemove

Delete file

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
libio	2.4	October 11, 2001

Syntax**#include** <sifdev.h>**int** sceRemove(
const char *name)
File pathname (includes device name + ":")
Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupted-enabled state)

Description

This function deletes the specified file. The file pathname should be specified as device name + unit number + ":" + string.

Return value

If processing is successful, zero is returned.

On error, the value in errno multiplied by -1 is returned.

sceRename

Rename file/directory

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
libio	2.4	October 11, 2001

Syntax

```
#include <sifdev.h>
```

```
int sceRename(
```

```
    const char *oldname,           Old file/directory pathname
```

```
    const char *newname)          New file/directory pathname
```

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupted-enabled state)

Description

This function changes the name of a file. When necessary, it moves a file between directories.

If newname exists, it will automatically be replaced if the following conditions are satisfied.

- Both oldname and newname are files.
- Both oldname and newname are directories.
- newname is an empty directory.
- newname is not open.

If newname existed and the operation failed for some reason, the newname entity will remain in its original state.

Return value

If processing is successful, zero is returned.

On error, the value in errno multiplied by -1 is returned.

sceRmdir

Delete directory

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
libio	2.4	October 11, 2001

Syntax

```
#include <sifdev.h>
```

```
int sceRmdir(
```

```
const char *name)
```

Directory pathname (includes device name + ":")

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupted-enabled state)

Description

This function deletes the specified directory. The directory to be deleted must be empty.

The directory pathname should be specified as device name + unit number + ":" + string.

Return value

If processing is successful, zero is returned.

On error, the value in `errno` multiplied by -1 is returned.

sceSymlink

Create symbolic link

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
libio	2.4	October 11, 2001

Syntax

```
#include <sifdev.h>
```

```
int sceSymlink(
```

```
    const char *oldname,           Original filename
```

```
    const char *newname)          New filename
```

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupted-enabled state)

Description

This function creates a symbolic link named newname, and links it to oldname. The symbolic link is interpreted during execution and the link is followed and replaced to obtain a file or directory. The symbolic link may be pointing to an existing file or a non-existent file. The .. notation may also be contained in the path. If newname already exists, it is not overwritten.

Return value

If processing is successful, zero is returned.

On error, the value in errno multiplied by -1 is returned.

sceSync

Synchronize disk with buffer cache

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
libio	2.4	October 11, 2001

Syntax**#include** <sifdev.h>**int** sceSync(**const char** *name,

Device name

int flag)

Flag (reserved; not used)

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupted-enabled state)

Description

This function synchronizes the state of memory with that of a device. For a disk, this function will write the contents of the buffer to the disk.

For details, refer to the documentation for each device.

Return value

If processing is successful, zero is returned.

On error, the value in errno multiplied by -1 is returned.

sceUmount

Unmount filesystem

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
libio	2.4	October 11, 2001

Syntax

```
#include <sifdev.h>
```

```
int sceUmount(
```

```
    const char *fsname)
```

String specifying mounted filesystem device name and unit number.

Calling conditions

Can be called from a thread

Multithread safe (must be called in interrupted-enabled state)

Description

This function unmounts a filesystem. The contents of the buffer cache in memory are flushed.

For details, refer to the document for each device.

Return value

If processing is successful, zero is returned.

On error, the value in errno multiplied by -1 is returned.

sceWrite

Write file

<i>Library</i>	<i>Introduced</i>	<i>Documentation last modified</i>
libio	1.1	October 11, 2001

Syntax**#include** <sifdev.h>**int** sceWrite(**int** *fd*,

File descriptor of file to write

const void **buf*,

Address at which write data is stored

int *count*)

Size of write data

Calling conditions

Can be called from a thread

Multithread safe (must be called in an interrupt-enabled state)

Description

This function writes to the file referenced by the file descriptor *fd* from the buffer indicated by *buf* to the maximum *count* bytes. Performance will decrease if the buffer is not aligned on a 64-byte boundary. Therefore, 64-byte alignment is recommended. For details, refer to the documentation for each device.

NotesThe SPR cannot be specified for *buf*.**Return value**

Size of data written (in bytes)

If an error occurred, -1 multiplied by *errno* is returned.