PlayStation®2 EE Library Overview
Release 2.4

Sif Libraries

Summary Table of Contents

About This Manual

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

The purpose of this manual is to provide overview-level information about the PlayStation®2 EE sif libraries. For related descriptions of the PlayStation®2 EE sif library structures and functions, refer to the PlayStation®2 *EE Library Reference - Sif Libraries.*

Changes Since Last Release

Chapter 2: Sif System

- In the "Default Module Replacement" section of "IOP Module Replacement", the description of memory specification has been deleted from programming examples.

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
`courier`	Indicates literal program code.
italic	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
In North America:	*In North America:*
Attn: Developer Tools Coordinator Sony Computer Entertainment America 919 East Hillsdale Blvd. Foster City, CA 94404, U.S.A. Tel: (650) 655-8000	E-mail: PS2_Support@playstation.sony.com Web: http://www.devnet.scea.com/ Developer Support Hotline: (650) 655-5566 (Call Monday through Friday, 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
In Europe:	*In Europe:*
Attn: Production Coordinator Sony Computer Entertainment Europe 30 Golden Square London W1F 9LD, U.K. Tel: +44 (0) 20 7859-5000	E-mail: ps2_support@scee.net Web: https://www.ps2-pro.com/ Developer Support Hotline: +44 (0) 20 7859-5777 (Call Monday through Friday, 9 a.m. to 6 p.m., GMT)

Chapter 1: Standard IOP Services

Library Overview

Standard IOP services include functions used to load IOP modules from the EE side, manage the IOP heap, etc. These functions use the SIF RPC as an extension to the EE Kernel. They can be broadly divided into the two categories shown below.

Module Operations Management

These service functions load modules into IOP memory and load objects into EE memory. On the IOP side, the module named LoadModuleByEE performs these services.

Memory Heap Management

These service functions allocate/free IOP heap memory and load files into allocated areas. On the IOP side, the module named FILEIO_service performs these services.

Related Files

The following header file is needed to use the standard IOP services API.

Table 1-1

Category	Filename
Header file	sifdev.h

Note on RPC Re-entry

The standard IOP services functions use the SIF RPC internally. Thus, if multiple threads are used, RPC re-entry must be avoided. Refer to the discussion on RPC re-entry in the "SIF System" document.

RPC WAIT Function

In the following functions, sceSifBindRpc() / sceSifCallRpc() is executed in WAIT state. Besides being careful with RPC re-entry, make sure that interrupts are disabled and that these functions are not called from within an interrupt handler.

- sceSifLoadModule()
- sceSifLoadElf()
- sceSifLoadElfPart()
- sceSifInitIopHeap()
- sceSifAllocIopHeap()
- sceSifFreeIopHeap()
- sceSifLoadIopHeap()

Non-RPC Functions

The following function does not use sceSifBindRpc() / sceSifCallRpc(). It can be used without concern for RPC re-entry issues (this does not mean that the function itself is re-entrant).

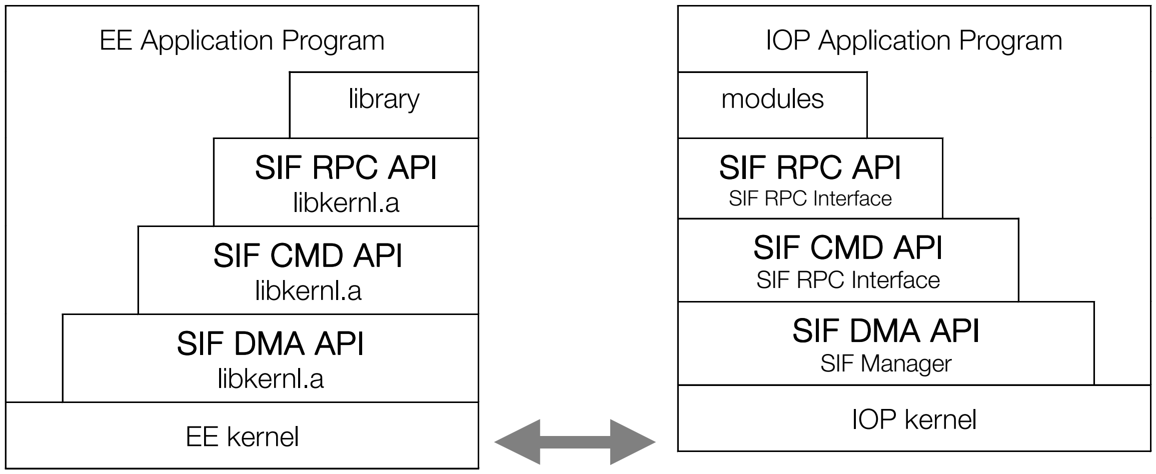
- sceSifLoadFileReset()

Chapter 2: SIF System

SIF System Configuration

SIF is a system that provides communications between the EE and IOP via the SubBus Interface (SIF). It is comprised of a hierarchical structure, described below.

Figure 2-1: SIF Hierarchical Structure



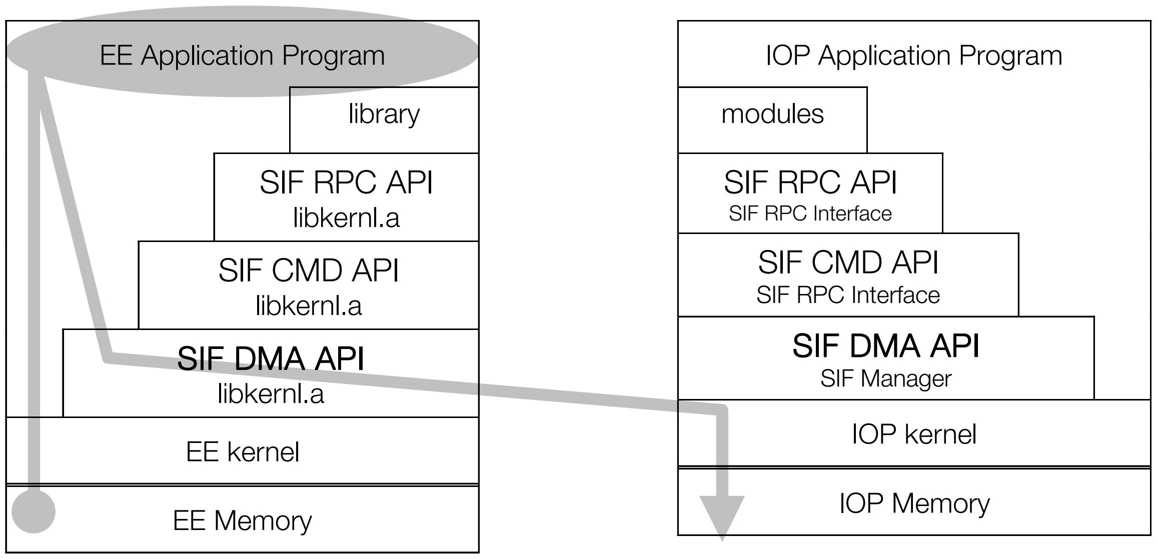
Both the EE and IOP sides are provided with almost identical APIs at three levels, namely SIF DMA, SIF CMD, and SIF RPC (Remote Procedure Call).

SIF DMA API

Overview

The SIF DMA API is the lowest level API of the SIF system. It provides functions for performing DMA transfers to the other side's memory, bidirectionally between the EE and IOP. It also provides functions for checking the transfer status, and functions for resetting DMA channels.

Figure 2-2



The SIF DMA receiving channels for both the EE and IOP are normally open, and DMA transfers to the other side's memory can be executed asynchronously (without being concerned with the status of the other side). However, when the EE is on the receiving end of a transfer, and a DMA termination interrupt is generated, the EE-side receiving channel will be closed. Similarly, when the IOP is on the receiving end of a transfer, and a DMA termination interrupt is generated by specifying SIF_DMA_ERT, the IOP-side receiving channel will be closed. In both cases, the receiving channel must be re-opened by resetting the DMA channel on the receiving side.

Related Files

The following files are required to use the SIF DMA API.

Table 2-1

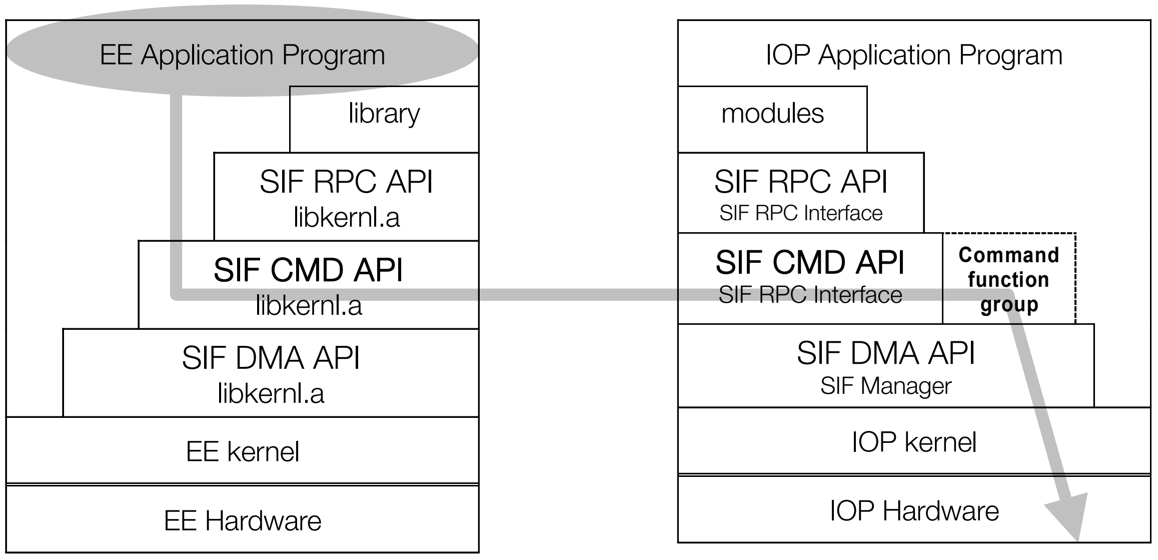
Category	Filename
Header file	sif.h
EE Library	libkernl.a
IOP module	iop.ilb

SIF Command API

Overview

The SIF CMD (Command) API, which is located at a level above the SIF DMA API, is used to send command packets that execute functions (command functions) that were previously defined and registered locally on either the EE and IOP, from the other side. For example, if a command function for changing a variable of an IOP application was registered on the IOP, the SIF CMD API would permit this operation to be performed from the EE. Likewise, by registering a command function on the EE, executing an instruction in an IOP application can cause an action to occur on the EE. The EE and IOP have nearly identical APIs.

Figure 2-3



The SIF CMD API can be used at the same time as the SIF DMA API.

Related Files

The following files are required to use the SIF CMD API.

Table 2-2

Category	filename
Header file	sifcmd.h
EE Library	Libkernl.a
IOP module	iop.ilb

Usage Procedure

First, sceSifInitCmd() is called by both the EE and IOP for initialization. The internal operations performed at this time are as follows.

1. Allocate a receive buffer (8 qwords) and convert its address.
2. Register the SIF CMD interrupt handler.
3. Initialize software registers.

Next, at the processor that will be the receiver, call sceSetCmdBuffer() to register a command function registration buffer (table). Also, call sceSifAddCmdHandler() to register a command function. A command function is executed as an interrupt function. To register several command functions, repeatedly call sceSifAddCmdHandler() the required number of times.

Then, when necessary, prepare an appropriate command packet and call sceSifSendCmd() to send it to the receiving processor. The internal operations performed at this time are as follows.

1. The command packet is transferred to the receive buffer on the other side and generates a transfer termination interrupt.
2. The SIF CMD interrupt handler searches the receive buffer on the receiving side.
3. If a command packet is found in the receive buffer, the packet is analyzed and the appropriate command function is executed in the interrupt area.

If no data is found in the receive buffer, processing returns directly.

Sample Programs

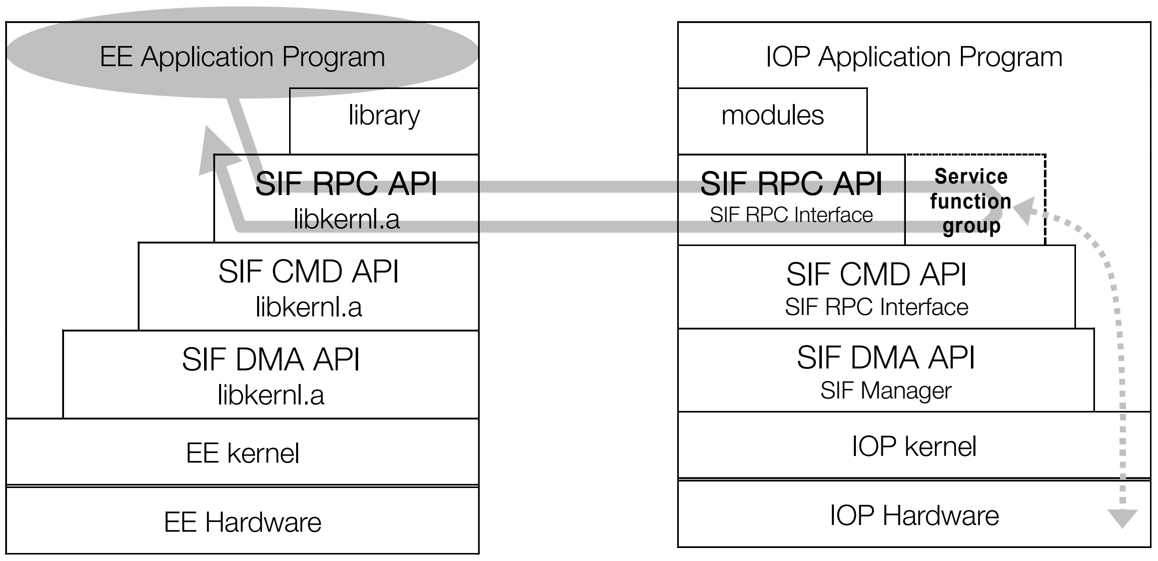
Sample programs that use the SIF CMD API can be found in /sce/ee/sample/sif/sifcmd and /sce/iop/sample/sif/sifcmd.

SIF RPC API

Overview

The SIF RPC (Remote Procedure Call) API is a mechanism for building a simple client/server model on top of the SIF CMD API. For example, a function that was registered on the IOP can be called from an EE application and the results can be returned to the EE application.

Figure 2-4



Since nearly identical APIs are provided for both the EE and IOP, the EE can also be the server and the IOP can be the client, which is the opposite to the situation shown in the above figure. Both sides can also play both roles at the same time.

In addition, the SIF RPC API can be used at the same time as the SIF DMA API and the SIF CMD API.

Related Files

The following files are required to use the SIF RPC API.

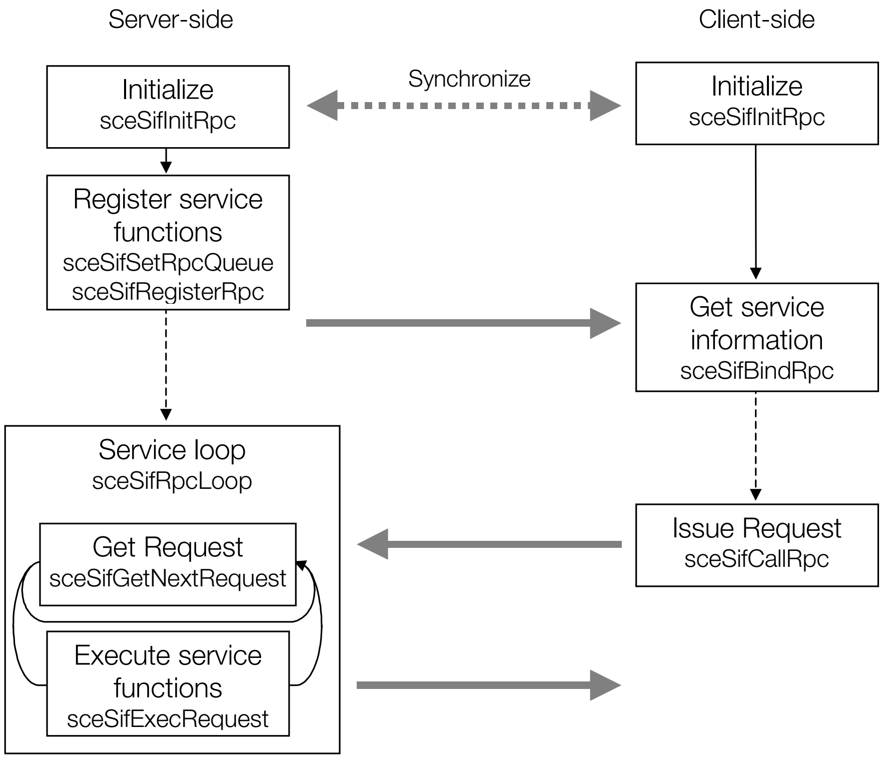
Table 2-3

Category	Filename
Header file	sifrpc.h
EE library	libkernl.a
IOP module	iop.ilb

Usage Procedure

The procedure to follow when using the SIF RPC API is summarized as follows.

Figure 2-5



When there is no request from the client while executing the server-side service loop, the thread calling the service loop will enter SLEEP state until a request is received.

With the client-side functions sceSifBindRpc() and sceSifCallRpc(), the thread calling these functions will wait for a semaphore or an event flag until there is a response from the server (this is known as synchronous calling or WAIT execution). This call can also be performed without a WAIT state. In this case, completion of the operation can be checked with sceSifCheckStatRpc() (this is known as asynchronous calling or NOWAIT execution).

For synchronous calling, completion of the service function is reported by a DMA interrupt from the server. Consequently, using synchronous calling with interrupts disabled or from within an interrupt handler can result in a hang.

Standard Service Functions

I/O services and standard IOP services are provided as service functions when the EE is the client. For more information, please refer to the libio/libiserv documents.

Sample Programs

Sample programs that use the SIF RPC API can be found in /sce/ee/sample/sif/sifrpc and /sce/ee/sample/sif/sifrpc.

Notes on RPC Re-entry

How RPC Re-entry Works

Service functions that call the SIF RPC API are not reentrant. Even if the functions themselves are coded to be reentrant, there can be no re-entry via RPC.

From the client side, if a service function is called using sceSifCallRpc(), additional calls cannot be made if the operation has not completed. If multiple calls are made, in many cases the proper data will not be obtained and a hang can occur in the internal function _sceRpcFreePacket() within the client-side library.

This kind of re-entry can occur when more than one thread uses the same service function. More specifically, these cases can be divided into two categories.

Cases where re-entry occurs from WAIT state

- A thread (A) runs sceSifCallRpc(&bd) in WAIT state.
- sceSifCallRpc(&bd) waits for a semaphore and execution rights are given to another thread (B).
- Thread (B) executes sceSifCallRpc(&bd). Re-entry takes place if the sceSifCallRpc(&bd) from (1) has not finished.

Cases where re-entry occurs from NOWAIT state

- A thread (A) runs sceSifCallRpc(&bd) in NOWAIT state.
- An interrupt occurs and control passes to an interrupt handler.
- The interrupt handler wakes up a thread (B), which has a higher priority than thread (A).
- Control returns from the interrupt handler and execution rights are given to thread (B).
- Thread (B) executes sceSifCallRpc(&bd). Re-entry takes place if the sceSifCallRpc(&bd) from (1) has not finished.

Preventing RPC Re-entry

The following methods can be used to prevent RPC re-entry.

- Use a single thread to group together calls for sceSifCallRpc() that use the same ClientData
- Use a semaphore, etc. when more than one thread calls sceSifCallRpc() and use the same ClientData (thread-safe structure)

Library Functions That Use RPC

The following functions from the libraries provided by SCE use sceSifCallRpc() internally. Basically, functions belonging to the same library use the same ClientData so make sure that RPC re-entry does not take place as described above.

- Some of the functions from libpad
- All of the functions from libmtap
- Almost all of the functions from libmc
- Almost all of the functions from libmcx
- Almost all of the functions from libcdvd
- Some of the functions from libsdr
- Almost all of the functions from standard IOP services
- Almost all of the functions from I/O services

For more information, refer to the Library Overview documents.

IOP Module Replacement

Default Module Replacement

Some modules are loaded from ROM into the IOP by default. To replace these modules, you must first reboot the IOP and restart the operating system. At that time, specify the modules to be replaced as image files. Image files are provided that match the version of the library.

Immediately after starting up the application from disk, the IOP should be rebooted so that the default modules are replaced.

Confirm that the message shown below is displayed by dsidb on the DTL-T10000 (IOP-side console) during a replacement.

1. Normal execution

   dsidb>

   Update reboot complete

2. Example of a failure (incorrect filename)

   dsidb>

   Update rebooting..

   open file name ********.***;1

   file 'cdrom0:********.***;1' can't open

   ***Resetted

A sample program is shown below.

```
#include <eekernel.h>
#include <eeregs.h>
#include <sifdev.h>
#include <stdio.h>
#include <libcdvd.h>

#define CDROM      /* Use CDROM */
```

```
        #define REPLACE    /* Module replacement */

        #ifdef CDROM
        #define IOPRP "cdrom0:\\MODULES\\"IOP_IMAGE_FILE";1"
#else
        #define IOPRP "host0:/usr/local/sce/iop/modules/"IOP_IMAGE_file
#endif

        int main()
        {
        #ifdef REPLACE
        #ifdef CDROM
          sceSifInitRpc(0);
          sceCdInit(SCECdINIT);
        #else
          sceSifInitRpc(0);
        #endif /* CDROM */
while (!sceSifRebootIop (IOPRP)); /* IOP reboot module
                                             substitution */
          while( !sceSifSyncIop() );        /* Wait for completion */
        #endif /* REPLACE */

          sceSifInitRpc(0);
          sceSifLoadFileReset();
          sceFsReset();
        #ifdef CDROM
          sceCdInit(SCECdINIT);
          sceCdMmode(SCECdCD);      /* Media: CD-ROM */
        #endif

        /* Start non-default modules */
          while (sceSifLoadModule(...) < 0);
          while (sceSifLoadModule(...) < 0);
                :
                :
        }
```

The IOP_IMAGE_FILE and IOP_IMAGE_file macros used in the program define the IOP replacement module from Release 1.6 in sifdev.h as shown below. This filename is updated every release.

<sifdev.h>

```
        #define IOP_IMAGE_FILE "IOPRPXX.IMG"
        #define IOP_IMAGE_file "ioprpXX.img"
```

This allows the specification method shown below to be used when replacing an IOP default module and makes modification unnecessary when only a library update is performed.

Example:

```
        #ifdef CDROM
          //  Starting up from a CD/DVD-ROM disk (access IOPreplace module
              on the CD/DVD-ROM)
          while (!sceSifRebootIop ("cdrom0:\\" IOP_IMAGE_FILE ";1"));
        #else
          // Access IOP replacement module on the local disk
          while (!sceSifRebootIop ("host0:/usr/local/sce/iop/modules/"
          IOP_IMAGE_file));
        #endif
```

No services can be used via the IOP, including printf(), from the time sceSifRebootIop() is issued until sceSifSyncIop() returns a non-zero value (however, the GS and EE coprocessor can be controlled).

Also, since all SIFCMD, SIFRPC, and DECI settings are reset, they should be set again. For sceOpen() or sceRead(), call sceFsReset() and for libcdvd, call sceCdInit().

Replacing Non-default Modules

The sample program shown above can be used to replace non-default modules. However, sceSifLoadModule() should be executed again after rebooting the IOP.

Specifying Thread Priorities for IOP Modules

When sceSifLoadModule() is used to start an IOP module, the thread priority used by the module can be specified. Use the third argument, as shown below.

For pad:

```
char* mes = "thpri=32,34"; <- Set to desired value
sceSifLoadModule( "host0:/usr/local/sce/iop/modules/padman.irx",
strlen(mes)+1, mes );
```

For libraries that have an IOP module (except for libpad), a function (sceXXXChangeThreadPriority()) is provided to change the thread priority of an active module.

For more information, refer to the library documents.