SPE Runtime Management Library

Version 1.1

CBEA JSRE Series
Cell Broadband Engine Architecture
Joint Software Reference
Environment Series

June 15, 2006

Printed in the United States of America June 2006

IBM Systems and Technology Group
2070 Route 52, Bldg. 330
Hopewell Junction, NY 12533-6351

The IBM home page can be found at **ibm.com**

The IBM semiconductor solutions home page can be found at **ibm.com**/chips

June 15, 2006

Table of Contents

About This Document

This document describes SPE Runtime Management Library. This library provides applications access to Synergistic Processing Elements (SPEs) via a thread abstraction model in which SPE programs can be scheduled for execution on a SPE thread.

Audience

The document is intended for system and application programmers wishing to develop Cell Broadband Engine (CBE) applications that fully exploit the SPEs.

Version History

This section describes significant changes made to the SPE Runtime Management Library specification for each version of this document.

Version Number & Date	Changes
Version 1.0 October 31, 2005	Initial public release of the document.
Version 1.1 February 9, 2006	Changes include: • Replaced spe_get_ps function with spe_get_ps_area. • Added spe_mfc_get and spe_mfc_put functions • Added spe_mfc_read_tag_status functions
Version 1.1 June 15, 2006	No functional changes. Minor documentation corrections include: • RFC00001 – Correction Notice: MFC functions reference nonexistent *spe_ps_addr* parameter. • RFC00003 – Correction Notice: Correct event names within the spe_get_event Linux Notes. • RFC00004 - Add missing invalid DMA command event.

Related Documentation

The following table provides a list of reference and supporting materials for the SPE Runtime Management Library specification:

Document Title	Version	Date
Cell Broadband Engine Architecture	1.0	August 2005

Document Structure

This document contains the following major sections:

1. Overview
2. SPE Thread Management Facilities
3. MFC Problem State Facilities

Overview

The SPE Management Library consists of two sets of PPE functions:

- A set of PPE functions used to manage SPEs (Synergistic Processing Elements). These interfaces are similar to those used to manage PPE threads on a POSIX compliant operating system.
- Another set of functions used to access MFC (Memory Flow Control) problem state facilities.

The SPE Management library introduces the following terminology.

SPE Thread An **SPE thread** is a thread of control that can be executed independently of the calling task. SPE threads are created by calling **spe_create_thread**. SPE threads have a unique identifier, of type speid_t, which can be used to query or set SPE thread attributes.

SPE Group An **SPE group** represents a collection of SPE threads that share scheduling attributes. Each SPE thread belongs to exactly one SPE group. SPE groups are created by calling **spe_create_group**. SPE groups have a unique identifier, of type spe_gid_t, which can be used to query or set SPE group attributes.

Library Name(s)

libspe

Header File(s)

<libspe.h>

SPE Thread Management Facilities

spe_create_group

C Specification

```
#include <libspe.h>
#include <sched.h>
spe_gid_t spe_create_group (int policy, int priority, int spe_events)
```

Description

The **spe_create_group** function allocates a new SPE thread group. SPE thread groups define the scheduling policies and priorities for a set of SPE threads. Each SPE thread belongs to exactly one group.

As an application creates SPE threads, the new threads are added to the designated SPE group. However the total number of SPE threads in a group cannot exceed the group maximum, which is dependent upon scheduling policy, priority, and availability of system resources. The **spe_group_max** function returns the maximum allowable number of SPE threads for a group.

All runnable threads in an SPE group may be gang scheduled for execution. Gang scheduling permits low-latency interaction among SPE threads in shared-memory parallel applications.

Parameters

policy — Defines the scheduling class for SPE threads in a group. Accepted values are:

- **SCHED_RR** which indicates real-time round-robin scheduling.
- **SCHED_FIFO** which indicates real-time FIFO scheduling.
- **SCHED_OTHER** which is used for low priority tasks suitable for filling otherwise idle SPE cycles.

The real-time scheduling policies **SCHED_RR** and **SCHED_FIFO** are available only to processes with super-user privileges.

priority — Defines the SPE group's scheduling priority within the policy class. For the real-time policies **SCHED_RR** and **SCHED_FIFO**, priority is a value in the range of 1 to 99. For interactive scheduling (**SCHED_OTHER**) the priority is a value in the range 0 to 99. The priority for an SPE thread group can be modified with **spe_set_priority**, or queried with **spe_get_priority**.

spe_events — A non-zero value for this parameter allows the application to receive events for SPE threads in the group. SPE events are conceptually similar to Linux signals, but differ as follows: SPE events are queued, ensuring that if multiple events are generated, each is delivered; SPE events are delivered in the order received; SPE events have associated data, including the type of event and the SPE thread id. The spe_get_event function can be called to wait for SPE events.

Return Value

On success, a positive non-zero identifier for a new SPE group is returned. On error, zero is returned and errno is set to indicate the error.

Possible errors include:

- ENOMEM — The SPE group could not be allocated due to lack of system resources.
- ENOMEM — The total number of SPE groups in the system has reached the system maximum value.
- EINVAL — The requested scheduling policy or priority was invalid.
- EPERM — The process does not have sufficient privileges to create an SPE group with the requested scheduling policy or priority.
- ENOSYS — The SPE group could not be allocated due to lack of implementation support for the specified scheduling priority or policy.

See Also

spe_create_thread
spe_group_defaults
spe_group_max
spe_get_priority, spe_set_priority, spe_get_policy

spe_create_thread

C Specification

```
#include <libspe.h>
speid_t spe_create_thread(spe_gid_t gid, spe_program_handle_t *spe_program_handle, void *argp, void *envp,
                          unsigned long mask, int flags)
```

Description

spe_create_thread creates a new SPE thread of control that can be executed independently of the calling task. As an application creates SPE threads, the threads are added to the designated SPE group. The total number of SPE threads in a group cannot exceed the group maximum. The **spe_group_max** function returns the number of SPE threads allowed for the group.

Parameters

gid	Identifier of the SPE group that the new thread will belong to. SPE group identifiers are returned by **spe_create_group**. The new SPE thread inherits scheduling attributes from the designated SPE group. If **gid** is equal to SPE_DEF_GRP (0), then a new group is created with default scheduling attributes, as set by calling **spe_group_defaults**.
spe_program_handle	Indicates the program to be executed on the SPE. This is an opaque pointer to an SPE ELF image which has already been loaded and mapped into system memory. This pointer is normally provided as a symbol reference to an SPE ELF executable image which has been embedded into a PPE ELF object and linked with the calling PPE program. This pointer can also be established dynamically by loading a shared library containing an embedded SPE ELF executable, using **dlopen(2)** and **dlsym(2)**, or by using the **spe_open_image** function to load and map a raw SPE ELF executable.
argp	An (optional) pointer to application specific data, and is passed as the second parameter to the SPE program.
envp	An (optional) pointer to environment specific data, and is passed as the third parameter to the SPE program.
mask	The processor affinity mask for the new thread. Each bit in the mask enables (1) or disables (0) thread execution on a cpu. At least one bit in the affinity mask must be enabled. If equal to -1, the new thread can be scheduled for execution on any processor. The affinity mask for an SPE thread can be changed by calling **spe_set_affinity**, or queried by calling **spe_get_affinity**.
flags	A bit-wise OR of modifiers that are applied when the new thread is created. The following values are accepted:

0	No modifiers are applied
SPE_CFG_SIGNOTIFY1_OR	Configure the Signal Notification 1 Register to be in “logical OR” mode instead of the default “Overwrite” mode.
SPE_CFG_SIGNOTIFY2_OR	Configure the Signal Notification 1 Register to be in “logical OR” mode instead of the default “Overwrite” mode.
SPE_MAP_PS	Request permission for memory-mapped access to the SPE thread’s problem state area(s). Direct access to problem state is a real-time feature, and may only be available to programs running with privileged authority (or in Linux, to processes with access to CAP_RAW_IO; see capget(2) for details).
SPE_USER_REGS	Specifies that the SPE setup registers r3, r4, and r5 are initialized with the 48 bytes pointed to by argp.

Return Value

On success, a positive non-zero identifier of the newly created SPE thread is returned. On error, 0 is returned and errno is set to indicate the error.

Possible errors include:

ENOMEM	The SPE thread could not be allocated due to lack of system resources
EINVAL	The value passed for mask or flags was invalid.

EPERM	The process does not have permission to add threads to the designated SPE group, or to use the SPU_MAP_PS setting.
ESRCH	The SPE group could not be found.

See Also

spe_create_group
spe_get_group
spe_get_ls
spe_get_ps_area
spe_get_threads
spe_group_defaults
spe_group_max
spe_open_image, spe_close_image

spe_get_affinity, spe_set_affinity

C Specification

```
#include <libspe.h>
int spe_get_affinity(speid_t speid, unsigned long *mask)
int spe_set_affinity(speid_t speid, unsigned long mask)
```

Description

The **spe_get_affinity** function returns the processor affinity mask for an SPE thread.

The **spe_set_affinity** function sets the processor affinity mask for an SPE thread**.**

Parameters

speid	Identifier of a specific SPE thread.
mask	The affinity bitmap is represented by the value specified by **mask**. The least significant bit corresponds to the first cpu on the system, while the most significant bit corresponds to the last cpu on the system. A set bit corresponds to a legally schedulable processor while an unset bit corresponds to an illegally schedulable processor. In other words, a thread is bound to and will only run on a cpu whose corresponding bit is set. Usually, all bits in the mask are set.

Return Value

On success, **spe_get_affinity** and **spe_set_affinity** return 0. On failure, -1 is returned and errno is set appropriately. **spe_get_affinity** returns the affinity mask in the memory pointed to by the mask parameter.

Possible errors include:

EFAULT	The supplied memory address for mask was invalid.
EINVAL	The mask is invalid or cannot be applied.
ENOSYS	The affinity setting operation is not supported by the implementation or environment.
ESRCH	The specified SPE thread could not be found.

See Also

spe_create_thread
sched_setaffinity (2)

spe_get_context, spe_set_context

C Specification

#include <libspe.h>
int spe_get_context(speid_t speid, struct spe_ucontext *uc)
int spe_set_context(speid_t speid, struct spe_ucontext *uc)

Description

The **spe_get_context** call returns the SPE user context for an SPE thread.
The **spe_set_context** call sets the SPE user context for an SPE thread.

Parameters

speid — Specifies the SPE thread

uc — Points to the SPE user context structure, allocated by the application, of type:

```
  struct spe_ucontext {
    struct unsigned int gprs[128][4];  // 128 x 128-bit SPE GPRs
    unsigned int fpcr[4];              // Floating point cntl
    unsigned int decr;                 // SPE decrementing ctr
    unsigned int decr_status;          // SPE decrementer status
    unsigned int npc;                  // SPE next program counter
    unsigned int tag_mask;             // DMA tag query mask
    unsigned int event_mask;           // Event query mask
    unsigned int srr0;                 // Machine status register
    unsigned int _reserved[2];         // Unused
    void *ls;                          // SPE local storage area
};
```

Return Value

On success, both **spe_get_context** and **spe_set_context** return 0. On failure, -1 is returned and errno is set appropriately.

Possible error include:

EFAULT	The memory region pointed to by uc is invalid.
EINVAL	The execution status of the specified SPE thread is inappropriate.
ENOSYS	The operation is not supported by the implementation or environment.
EPERM	The caller does not have permission to query or set the user context for the specified SPE thread.
ESRCH	The specified SPE thread could not be found.

See Also

spe_kill
spe_create_thread
spe_wait
getcontext (2), setcontect (2)

spe_get_event

C Specification

#include <libspe.h>
int spe_get_event (struct spe_event *pevents, int nevents, int timeout)

Description

spe_get_event polls or waits for events that may be generated by threads in an SPE group.

Parameters

pevents This specifies an array of SPE event structures of type:

```
struct spe_event {
  spe_gid_t gid;             // input, SPE group id
  int events;                // input, requested event mask
  int revents;               // output, returned events
  speid_t speid;             // output, returned speid
  unsigned long data;        // output, returned data
};
```

gid This field is an input parameter, specifying the SPE group to query events for.

events This field is an input parameter, specifying a bit-mask of the SPE events the application is interested in.

revents This field is an output parameter, filled in by the operating system with the events that actually occurred, either of the type requested, or of one of the types **SPE_EVENT_ERR**, **SPE_EVENT_NVAL**, or **SPE_EVENT_THREAD_EXIT**.

The following possible bits in the **events** and **revents** masks are defined in <libspe.h>. (The **SPE_EVENT_ERR** and **SPE_EVENT_NVAL** bits are meaningless in the **events** field, and are set in the **revents** field whenever the corresponding condition is true).

Event	Description
SPE_EVENT_MAILBOX	Interrupting mailbox data
SPE_EVENT_STOP	SPE ‘stop-and-signal’ data
SPE_EVENT_TAG_GROUP	Tag group complete data
SPE_EVENT_DMA_ALIGNMENT	A DMA alignment error
SPE_EVENT_INVALID_DMA_CMD	An invalid or unsupported MFC command was executed on the SPE thread’s SPU or proxy command queue.
SPE_EVENT_SPE_ERROR	A SPE error including illegal channel instruction and invalid instructions.
SPE_EVENT_SPE_DATA_SEGMENT	A DMA segmentation error
SPE_EVENT_SPE_DATA_STORAGE	A DMA storage error
SPE_EVENT_SPE_TRAPPED	SPE ‘halt’ instruction was executed.

SPE_EVENT_THREAD_EXIT	A thread has exited
SPE_EVENT_ERR	An error occurred
SPE_EVENT_NVAL	Invalid request

speid — This field is an output parameter, filled in by the operating system to indicate the id of the SPE thread that generated the event.

data — This field is an output parameter, filled in by the operating system to indicate the SPE data associated with the event.

nevents — This specifies the number of spe_event structures in the **pevents** array.

timeout — This specifies the timeout value in milliseconds. A negative value means an infinite timeout. If none of the events requested (and no error) had occurred any of the SPE groups, the operating system waits for timeout milliseconds for one of these events to occur.

Return Value

On success, a positive number is returned, where the number returned indicates the number of structures which have non-zero **revents** fields (in other words, those with events or errors reported). A value of 0 indicates that the call timed out and no events have been selected. On error, -1 is returned and errno is set appropriately.

Possible errors include:

EFAULT — The array given as a parameter was not contained in the calling program's address space.

EINVAL — No SPE groups have yet been created.

EINTR — A signal occurred before any requested event.

EPERM — The current process does not have permission to get SPE events.

Linux Notes

If SPE-events are not enabled for an SPE group, then POSIX signals may be delivered to the application, as follows:

SPE-event	POSIX signal	Default Action
SPE_EVENT_MAILBOX	SIGSPE (SIGURG)	ignore
SPE_EVENT_STOP	SIGSPE	ignore
SPE_EVENT_TAG_GROUP	SIGSPE	ignore
SPE_EVENT_DMA_ALIGNMENT	SIGBUS	dump
SPE_EVENT_INVALID_DMA_CMD	SIGBUS	dump
SPE_EVENT_SPE_ERROR	SIGILL	dump
SPE_EVENT_SPE_DATA_SEGMENT	SIGSEGV	dump
SPE_EVENT_SPE_DATA_STORAGE	SIGSEGV	dump
SPE_EVENT_SPE_TRAPPED	SIGABRT	dump
SPE_EVENT_THREAD_EXIT	SIGCHLD	ignore

See Also

spe_create_group
poll (2)

spe_get_group

C Specification

```
#include <libspe.h>
spe_gid_t spe_get_group (speid_t speid)
```

Description

The **spe_get_group** function returns the SPE group identifier for the SPE thread, as indicated by **speid**.

Parameters

speid	The identifier of a specific SPE thread.

Return Value

The SPE group identifier for an SPE thread, or 0 on failure.

Possible errors include:

ESRCH	The specified SPE thread could not be found.

See Also

spe_create_group
spe_get_threads

spe_get_ls

C Specification

```
#include <libspe.h>
void *spe_get_ls (speid_t speid)
```

Description

The **spe_get_ls** function returns the address of the local storage for the SPE thread indicated by **speid**.

Parameters

speid	The identifier of a specific SPE thread.

Return Value

On success, a non-NULL pointer is returned. On failure, NULL is returned and errno is set appropriately.

Possible errors include:

ENOSYS	Access to the local store of an SPE thread is not supported by the operating system.
ESRCH	The specified SPE thread could not be found.

See Also

spe_create_group
spe_get_ps_area

spe_get_ps_area

C Specification

```
#include <libspe.h>
void *spe_get_ps_area (speid_t speid, enum ps_area)
```

Description

The **spe_get_ps_area** function returns a pointer to the problem state area specified by **ps_area** for the SPE thread indicated by **speid**. In order to obtain a problem state area pointer the specified SPE thread must have been created with the SPE_MAP_PS flag set with sufficient privileges.

The problem state pointer can be used to directly access problem state features without having to make library system calls. Problem state features include multi-source synchronization, proxy DMAs, mailboxes, and signal notifiers. In addition, these pointers, along with local store pointers (see **spe_get_ls**), can be used to perform SPE to SPE communications via mailboxes, DMA's and signal notification.

Parameters

speid — The identifier of a specific SPE thread.

ps_area — The problem state area pointer to be granted access and returned. Possible problem state areas include:

SPE_MSSYNC_AREA — Return a pointer to the specified SPE's MFC multisource synchronization register problem state area as defined by the following structure:

```
typedef struct spe_mssync_area {
  unsigned int MFC_MSSync;
} spe_mssync_area_t;
```

SPE_MFC_COMMAND_AREA — Return a pointer to the specified SPE's MFC command parameter and command queue control area as defined by the following structure:

```
typedef struct spe_mfc_command_area {
  unsigned char reserved_0_3[4];
  unsigned int MFC_LSA;
  unsigned int MFC_EAH;
  unsigned int MFC_EAL;
  unsigned int MFC_Size_Tag;
  union {
    unsigned int MFC_ClassID_CMD;
    unsigned int MFC_CMDStatus;
  };
  unsigned char reserved_18_103[236];
  unsigned int MFC_QStatus;
  unsigned char reserved_108_203[252];
  unsigned int Prxy_QueryType;
  unsigned char reserved_208_21B[20];
  unsigned int Prxy_QueryMask;
  unsigned char reserved_220_22B[12];
  unsigned int Prxy_TagStatus;
} spe_mfc_command_area_t;
```

Note: The MFC_EAH and MFC_EAL registers can be simultaneously written using a 64-bit store. Likewise, MFC_Size_Tag and MFC_ClassID_CMD registers can be simultaneously written using a 64-bit store.

SPE_CONTROL_AREA — Return a pointer to the specified SPE's SPU control area as defined by the following structure:

```
typedef struct spe_spu_control_area {
  unsigned char reserved_0_3[4];
  unsigned int SPU_Out_Mbox;
  unsigned char reserved_8_B[4];
  unsigned int SPU_In_Mbox;
  unsigned char reserved_10_13[4];
```

```
  unsigned int SPU_Mbox_Stat;
  unsigned char reserved_18_1B[4];
  unsigned int SPU_RunCntl;
  unsigned char reserved_20_23[4];
  unsigned int SPU_Status;
  unsigned char reserved_28_33[12];
  unsigned int SPU_NPC;
} spe_spu_control_area_t;
```

Note: Explicit programmer manipulation of the SPU run control is highly discouraged.

SPE_SIG_NOTIFY_1_AREA — Return a pointer to the specified SPE's signal notification area 1 as defined by the following structure:

```
typedef struct spe_sig_notify_1_area {
  unsigned char reserved_0_B[12];
  unsigned int SPU_Sig_Notify_1;
} spe_sig_notify_1_area_t;
```

SPE_SIG_NOTIFY_2_AREA — Return a pointer to the specified SPE's signal notification area 2 as defined by the following structure:

```
typedef struct spe_sig_notify_2_area {
  unsigned char reserved_0_B[12];
  unsigned int SPU_Sig_Notify_2;
} spe_sig_notify_2_area_t;
```

Return Value

On success, a non-NULL pointer to the requested problem state area is returned. On failure, NULL is returned and errno is set appropriately.

Possible errors include:

EACCES	Permission for direct access to the specified problem state area is denied or the SPE thread was not created with memory-mapped problem state access.
EINVAL	The specified problem state area is invalid.
ENOSYS	Access to the specified problem area for the specified SPE thread is not supported by the operating system.
ESRCH	The specified SPE thread could not be found.

See Also

spe_create_thread
spe_get_ls

spe_get_priority, spe_set_priority, spe_get_policy

C Specification

```
#include <libspe.h>
int spe_get_priority (spe_gid_t gid)

#include <libspe.h>
int spe_set_priority (spe_gid_t gid, int priority)

#include <libspe.h>
int spe_get_policy (spe_gid_t gid)
```

Description

The scheduling priority for the SPE thread group, as indicated by **gid**, is obtained by calling the **spe_get_priority** function, or is set by calling the **spe_set_priority** function.

For the real-time policies **SCHED_RR** and **SCHED_FIFO**, priority is a value in the range of 1 to 99. Only the super-user may modify real-time priorities. For the interactive policy **SCHED_OTHER**, priority is a value in the range 0 to 40. Only the super-user may raise interactive priorities.

The scheduling policy class for an SPE group is queried by calling the **spe_get_policy** function.

Parameters

gid	The identifier of a specific SPE group.
priority	Specified the SPE thread group's scheduling priority within the group's scheduling policy class.

Return Value

On success, **spe_get_priority** returns a priority value of 0 to 99. On failure, **spe_get_priority** returns -1 and sets errno appropriately.

On success, **spe_set_priority** returns zero. On failure, **spe_set_priority** returns -1 and sets errno appropriately.

On success, **spe_get_policy** returns a scheduling policy class value of **SCHED_RR**, **SCHED_FIFO**, or **SCHED_OTHER**. On failure, **spe_get_policy** returns -1 and sets errno appropriately.

Possible errors include:

EINVAL	The specified **priority** value is invalid.
EPERM	The current process does not have permission to set the specified SPE thread group priority.
ESRCH	The specified SPE thread group could not be found.

See Also

spe_create_group

spe_get_threads

C Specification

```
#include <libspe.h>
int spe_get_threads (spe_gid_t gid, speid_t *spe_ids)
```

Description

spe_get_threads returns a list of SPE threads in a group, as indicated by **gid**, to the array pointed to by **spe_ids**.

The storage for the **spe_ids** array must be allocated and managed by the application. Further, the **spe_ids** array must be large enough to accommodate the current number of SPE threads in the group. The number of SPE threads in a group can be obtained by setting the **spe_ids** parameter to NULL.

Parameters

gid	This is the identifier of the SPE group.
spe_ids	This is a pointer to an array of speid_t values that are filled in with the ids of the SPE threads in the group specified by **gid**.

Return Value

On success, the number of SPE threads in the group is returned. On failure, -1 is returned and errno is set appropriately.

Possible errors include:

EFAULT	The **spe_ids** array was contained within the calling program's address space.
EPERM	The current process does not have permission to query SPE threads for this group.
ESRCH	The specified SPE thread group could not be found.

See Also

spe_create_group
spe_create_thread

spe_group_defaults

C Specification

```
#include <libspe.h>
#include <sched.h>
int spe_group_defaults (int policy, int priority, int spe_events)
```

Description

spe_group_defaults changes the application defaults for SPE groups. When an application calls **spe_create_thread** and designates an SPE group id equal to **SPE_DEF_GRP** (0), then a new group is created and the thread is added to the new group. The group is created with default settings for memory access privileges and scheduling attributes. By calling **spe_group_defaults**, the application can override the settings for these attributes.

The initial attribute values for SPE group 0 are defined as follows: the **policy** is set to **SCHED_OTHER**; the **priority** is set to 0; and **spe_events** are disabled.

Parameters

policy — This defines the scheduling class. Accepted values are:

SCHED_RR	which indicates real-time round-robin scheduling.
SCHED_FIFO	which indicates real-time FIFO scheduling.
SCHED_OTHER	which is used for low priority tasks suitable for filling otherwise idle SPE cycles.

priority — This defines the default scheduling priority. For the real-time policies **SCHED_RR** and **SCHED_FIFO**, priority is a value in the range of 1 to 99. For interactive scheduling (**SCHED_OTHER**) the priority is a value in the range 0 to 99.

spe_events — A non-zero value for this parameter registers the application's interest in SPE events for the group.

Return Value

On success, 0 is returned. On failure, -1 is returned and errno is set appropriately.

Possible errors include:

EINVAL The specified **policy** or **priority** value is invalid.

See Also

spe_create_group
spe_create_thread

spe_group_max

C Specification

```
#include <libspe.h>
int spe_group_max (spe_gid_t gid)
```

Description

The **spe_group_max** function returns the maximum number of SPE threads that may be created for an SPE group, as indicated by **gid**.

The total number of SPE threads in a group cannot exceed the group maximum, which is dependent upon the group's scheduling policy, priority, and availability of system resources.

Parameters

gid This is the identifier of the SPE group.

Return Value

On success, the maximum number of SPE threads allowed for the SPE group is return. On error, -1 is returned and errno is set appropriately.

Possible errors include:

EPERM The calling process does not have privileges to query the SPE group.

ESRCH The specified SPE group could not be found.

See Also

spe_create_group
spe_create_thread

spe_kill

C Specification

```
#include <libspe.h>
#include <signal.h>
int spe_kill (speid_t speid, int signal)
```

Description

The **spe_kill** can be used to send a control signal to an SPE thread.

Parameters

speid The signal is delivered to the SPE thread identified.

signal This indicates the type of control signal to be delivered. It may be one of the following values:

SIGKILL	Kill the specified SPE thread.
SIGSTOP	Stop execution of the specified SPE thread.
SIGCONT	Resume execution of the specified SPE thread.

Return Value

On success, 0 is returned. On error, -1 is returned and errno is set appropriately.

Possible errors include:

ENOSYS	The **spe_kill** operation is not supported by the implementation or environment.
EPERM	The calling process does not have permission to perform the kill action for the receiving SPE thread.
ESRCH	The SPE thread does not exist. Note that a existing SPE thread might be a zombie, an SPE thread which is already committed termination but yet had **spe_wait** called for it.

See Also

spe_create_thread
spe_wait
kill (2)

spe_open_image, spe_close_image

C Specification

```
#include <libspe.h>
spe_program_handle_t *spe_open_image (const char * filename)

#include <libspe.h>
int spe_close_image (spe_program_handle_t *spe_program_handle)
```

Description

spe_open_image maps an SPE ELF executable indicated by **filename** into system memory and returns the mapped address appropriate for use by the **spe_create_thread** API. It is often more convenient/appropriate to use the loading methodologies where SPE ELF objects are converted to PPE static or shared libraries with symbols which point to the SPE ELF objects after these special libraries are loaded. These libraries are then linked with the associated PPE code to provide a direct symbol reference to the SPE ELF object. The symbols in this scheme are equivalent to the address returned from the **spe_open_image** function.

SPE ELF objects loaded using this function are not shared with other processes, but SPE ELF objects loaded using the other scheme, mentioned above, can be shared if so desired.

spe_close_image unmaps an SPE ELF object that was previously mapped using **spe_open_image**.

Parameters

filename Specifies the filename of an SPE ELF executable to be loaded and mapped into system memory.

Return Values

On success, **spe_open_image** returns the address at which the specified SPE ELF object has been mapped. On failure, NULL is returned and errno is set appropriately.

On success, **spe_close_image** returns 0. On failure, -1 is returned and errno is set appropriately.

Possible errors include:

EACCES	The calling process does not have permission to access the specified file.
EFAULT	The **filename** parameter points to an address that was not contained in the calling process's address space.
EINVAL	From **spe_close_image**, this indicates that the file, specified by **filename**, was not previously mapped by a call to **spe_open_image**.

A number of other errno values could be returned by the **open(2)**, **fstat(2)**, **mmap(2)**, **munmap(2)**, or **close(2)** system calls which may be utilized by the **spe_open_image** or **spe_close_image** functions.

See Also

spe_create_thread

spe_wait

C Specification

```
#include <libspe.h>
#include <sys/wait.h>
int spe_wait (speid_t speid, int *status, int options)
```

Description

spe_wait suspends execution of the current process until the SPE thread specified by **speid** has exited. If the SPE thread has already exited by the time of the call (a so-called "zombie" SPE thread), then the function returns immediately. Any system resources used by the SPE thread are freed.

Parameters

speid — Wait for the SPE thread identified.

options — This parameter is an logical OR of zero or more of the following constants:

WNOHANG	Return immediately if the SPE thread has not exited.
WUNTRACED	Return if the SPE thread is stopped and its status has not been reported.

status — If this value is non-NULL, **spe_wait** stores the SPE thread's exit code at the address indicated by **status**. This status can be evaluated with the following macros. **Note:** these macros take the status buffer, an int, as a parameter - not a pointer to the buffer!

WIFEXITED(status)	Is non-zero if the SPE thread exited normally.
WEXITSTATUS(status)	Evaluates to the least significant eight bits of the return code of the SPE thread which terminated, which may have been set as the argument to a call to exit() or as the argument for a return statement in the main program. This macro can only be evaluated if WIFEXITED returned non-zero.
WIFSIGNALED(status)	Returns true if the SPE thread exited because of a signal which was not caught.
WTERMSIG(status)	Returns the number of the signal that caused the SPE thread to terminate. This macro can only be evaluated if WIFSIGNALED returned non-zero.
WIFSTOPPED(status)	Returns true if the SPE thread which caused the return is currently stopped; this is only possible if the call was done using WUNTRACED.
WSTOPSIG(status)	Returns the number of the signal which caused the SPE thread to stop. This macro can only be evaluated if WIFSTOPPED returned non-zero.

Return Values

On success, 0 is returned. A 0 value is returned if **WNOHANG** was used and the SPE thread was available. On failure, -1 is returned and errno is set appropriately.

Possible errors include:

ESRCH	The specified SPE thread could not be found.
EINVAL	The **options** parameter is invalid.
EFAULT	**status** points to an address that was not contained in the calling process's address space.
EPERM	The calling process does not have permission to wait on the specified SPE thread.
EAGAIN	The wait queue was active at the time **spe_wait** was called, prohibiting additional waits, so try again.

See Also

spe_create_thread

MFC Problem State Facilities

In the event that direct problem state access is not available (see **spe_get_ps_area**), the functions described in this section provide indirect access to the set of problem state facilities. These functions are guaranteed to be thread safe.

spe_mfc_get, spe_mfc_getb, spe_mfc_getf

C Specification

```
#include <libspe.h>
int spe_mfc_get(speid_t speid, unsigned int ls, void *ea, unsigned int size, unsigned int tag, unsigned int tid, unsigned int rid)

#include <libspe.h>
int spe_mfc_getb(speid_t speid, unsigned int ls, void *ea, unsigned int size, unsigned int tag, unsigned int tid, unsigned int rid)

#include <libspe.h>
int spe_mfc_getf(speid_t speid, unsigned int ls, void *ea, unsigned int size, unsigned int tag, unsigned int tid, unsigned int rid)
```

Description

The **spe_mfc_get** function places a *get* DMA command on the proxy command queue of the SPE thread specified by **speid**. The *get* command transfers **size** bytes of data starting at the effective address specified by **ea** to the local store address specified by **ls**. The DMA is identified by the tag id specified by **tag** and performed according to the transfer class and replacement class specified by **tid** and **rid** respectively.

The **spe_mfc_getb** function is identical to **spe_mfc_get** except that it places a *getb* (get with barrier) DMA command on the proxy command queue. The barrier form ensures that this command and all sequence commands with the same tag identifier as this command are locally ordered with respect to all previously issued commands with the same tag group and command queue.

The **spe_mfc_getf** function is identical to **spe_mfc_get** except that it places a *getf* (get with fence) DMA command on the proxy command queue. The fence form ensure that this command is locally ordered with respect to all previously issued commands with the same tag group and command queue.

The caller of these functions must ensure that the address alignments and transfer size is in accordance with the limitation and restrictions of the Cell Broadband Engine Architecture.

Parameters

speid	Specifies the SPE thread whose proxy command queue the get command is to be placed into.
ls	Specifies the starting local store destination address.
ea	Specifies the starting effective address source address.
size	Specifies the size, in bytes, to be transferred.
tag	Specifies the tag id used to identify the DMA command.
tid	Specifies the transfer class identifier of the DMA command.
rid	Specifies the replacement class identifier of the DMA command.

Return Values

On success, **spe_mfc_get**, **spe_mfc_getb** and **spe_mfc_getf** return 0. On failure, -1 is returned.

See Also

spe_create_thread
spe_get_ps_area
spe_mfc_put, spe_mfc_putb, spu_mfc_putf
spe_mfc_read_tag_status

spe_mfc_put, spe_mfc_putb, spe_mfc_putf

C Specification

```
#include <libspe.h>
int spe_mfc_put(speid_t speid, unsigned int ls, void *ea, unsigned int size, unsigned int tag, unsigned int tid, unsigned int rid)

#include <libspe.h>
int spe_mfc_putb(speid_t speid, unsigned int ls, void *ea, unsigned int size, unsigned int tag, unsigned int tid, unsigned int rid)

#include <libspe.h>
int spe_mfc_putf(speid_t speid, unsigned int ls, void *ea, unsigned int size, unsigned int tag, unsigned int tid, unsigned int rid)
```

Description

The **spe_mfc_put** function places a *put* DMA command on the proxy command queue of the SPE thread specified by **speid**. The *put* command transfers **size** bytes of data starting at the local store address specified by **ls** to the effective address specified by **ea**. The DMA is identified by the tag id specified by **tag** and performed according transfer class and replacement class specified by **tid** and **rid** respectively.

The **spe_mfc_putb** function is identical to **spe_mfc_put** except that it places a *putb* (put with barrier) DMA command on the proxy command queue. The barrier form ensures that this command and all sequence commands with the same tag identifier as this command are locally ordered with respect to all previously issued commands with the same tag group and command queue.

The **spe_mfc_puttf** function is identical to **spe_mfc_put** except that it places a *putf* (put with fence) DMA command on the proxy command queue. The fence form ensures that this command is locally ordered with respect to all previously issued commands with the same tag group and command queue.

The caller of these functions must ensure that the address alignments and transfer size is in accordance with the limitation and restrictions of the Cell Broadband Engine Architecture.

Parameters

speid	Specifies the SPE thread whose proxy command queue the put command is to be placed into.
ls	Specifies the starting local store source address.
ea	Specifies the starting effective address destination address.
size	Specifies the size, in bytes, to be transferred.
tag	Specifies the tag id used to identify the DMA command.
tid	Specifies the transfer class identifier of the DMA command.
rid	Specifies the replacement class identifier of the DMA command.

Return Values

On success, **spe_mfc_put**, **spe_mfc_putb** and **spe_mfc_putf** return 0. On failure, -1 is returned.

See Also

spe_create_thread
spe_get_ps_area
spe_mfc_get, spe_mfc_getb, spu_mfc_getf
spe_mfc_read_tag_status

spe_mfc_read_tag_status

C Specification

```
#include <libspe.h>
int spe_mfc_read_tag_status_all(speid_t speid, unsigned int mask)

#include <libspe.h>
int spe_mfc_read_tag_status_any(speid_t speid, unsigned int mask)

#include <libspe.h>
int spe_mfc_read_tag_status_immediate(speid_t speid, unsigned int mask)
```

Description

The **spe_mfc_read_tag_status_all** function suspends execution until all DMA commands in the tag groups enabled by the **mask** parameter have no outstanding DMAs in the proxy command queue of the SPE thread specified by **speid**. The masked tag status is returned.

The **spe_mfc_read_tag_status_any** function suspends execution until any DMA commands in the tag groups enabled by the **mask** parameter have no outstanding DMAs in the proxy command queue of the SPE thread specified by **speid**. The masked tag status is returned.

The **spe_mfc_read_tag_status_immediate** function returns the tag status for the tag groups specified by the **mask** parameter for the proxy command queue of the SPE thread specified by the **speid**.

Parameters

speid	Specifies the SPE thread whose proxy command queue status is to be read.
mask	Specifies the tag groups to be included in the query or wait operation.

Return Values

On success, **spe_mfc_read_tag_status_all**, **spe_mfc_read_tag_status_any**, **spe_mfc_read_tag_status_immediate** returns the current tag status. On failure, -1 is returned.

See Also

spe_mfc_get, spe_mfc_getb, spe_mfc_getf
spe_mfc_put, spu_mfc_putb, spu_mfc_putf

spe_read_out_mbox

C Specification

#include <libspe.h>
unsigned int spe_read_out_mbox(speid_t speid)

Description

The **spe_read_out_mbox** function returns the contents of the SPU outbound mailbox for the SPE thread specified by the **speid** parameter. This read is non-blocking and returns -1 if no mailbox data is available.

The **spe_stat_out_mbox** function can be called to ensure that data is available prior to reading the outbound mailbox.

Parameters

speid Specifies the SPE thread whose outbound mailbox is to be read.

Return Values

On success, **spe_read_out_mbox** returns the next 32-bit mailbox message. On failure, -1 is returned.

See Also

spe_stat_in_mbox, spe_stat_out_mbox, spe_stat_out_intr_mbox
spe_write_in_mbox
read (2)

spe_stat_in_mbox, spe_stat_out_mbox, spe_stat_out_intr_mbox

C Specification

```
#include <libspe.h>
int spe_stat_in_mbox(speid_t speid)

#include <libspe.h>
int spe_stat_out_mbox(speid_t speid)

#include <libspe.h>
int spe_stat_out_intr_mbox(speid_t speid)
```

Description

The **spe_stat_in_mbox** function fetches the status of the SPU inbound mailbox for the SPE thread specified by the **speid** parameter. A 0 value is return if the mailbox is full. A non-zero value specifies the number of available (32-bit) mailbox entries.

The **spe_stat_out_mbox** function fetches the status of the SPU outbound mailbox for the SPE thread specified by the **speid** parameter. A 0 value is return if the mailbox is empty. A non-zero value specifies the number of 32-bit unread mailbox entries.

The **spe_stat_out_intr_mbox** function fetches the status of the SPU outbound interrupt mailbox for the SPE thread specified by the **speid** parameter. A 0 value is return if the mailbox is empty. A non-zero value specifies the number of 32-bit unread mailbox entries.

Parameters

speid Specifies the SPE thread whose mailbox status is to be read.

Return Values

On success, **spe_stat_in_mbox**, **spe_stat_out_mbox**, and **spe_stat_out_intr_mbox** return the current status of the inbound, outbound and outbound interrupting mailbox, respectively. On failure, -1 is returned.

See Also

spe_read_out_mbox
spe_write_in_mbox
read (2)

spe_write_in_mbox

C Specification

```
#include <libspe.h>
int spe_write_in_mbox(speid_t speid, unsigned int data)
```

Description

The **spe_write_in_mbox** function places the 32-bit message specified by **data** into the SPU inbound mailbox for the SPE thread specified by the **speid** parameter.

If the mailbox is full, then **spe_write_in_mbox** can overwrite the last entry in the mailbox. The **spe_stat_in_mbox** function can be called to ensure that space is available prior to writing to the inbound mailbox.

Parameters

speid	Specifies the SPE thread whose outbound mailbox is to be read.
data	The 32-bit message to be written into the SPE's inbound mailbox.

Return Values

On success, **spe_write_in_mbox** returns 0. On failure, -1 is returned.

See Also

spe_read_out_mbox
spe_stat_in_mbox. Spe_stat_out_mbox, spe_stat_out_intr_mbox
write (2)

spe_write_signal

C Specification

```
#include <libspe.h>
int spe_write_signal(speid_t speid, unsigned int signal_reg, unsigned int data )
```

Description

The **spe_write_signal** function writes **data** to the signal notification register specified by **signal_reg** for the SPE thread specified by the **speid** parameter.

Parameters

speid	Specifies the SPE thread whose signal register is to be written to.
signal_reg	Specified the signal notification register to be written. Valid signal notification registers are:
	SPE_SIG_NOTIFY_REG_1 SPE signal notification register 1
	SPE_SIG_NOTIFY_REG_2 SPE signal notification register 2
data	The 32-bit data to be written to the specified signal notification register.

Return Values

On success, **spe_write_signal** returns 0. On failure, -1 is returned.

See Also

spe_get_ps_area
spe_write_in_mbox