Easy SPURS Reference

Table of Contents

Namespace

cell::Util::Spurs

Namespace of entire Easy SPURS

Definition

```
#include <cellUtilSpursDefine.h>
namespace cell {
        namespace Util {
              namespace Spurs {
              }
        }
}
```

Description

Easy SPURS is in this name space.

Notes

The macro constants regarding this namespace have been defined as follows.

Macro	Definition

__CELL_UTIL_SPURS_BEGIN	namespace cell {\
	  namespace Util {\
	  namespace Spurs {\

__CELL_UTIL_SPURS_END	}\ }\ }
__CELL_UTIL_SPURS	::cell::Util::Spurs::

Spurs Class

Spurs Class

Class of SPURS

Definition

```
#include <cellUtilSpurs.h>
namespace cell {
        namespace Util {
              namespace Spurs {
                    class Spurs : public CellSpurs {
                    };
              }
        }
}
```

Conditions

This instance must be in an SPU-accessible area.

This instance must be kept until successful completion of `Spurs::finalize()`.

Conditions of Inheritance

Constructors must be empty.

Destructors must not be defined.

Copy constructor and copy operator must be declared as private to invalidate them.

Do not declare virtual functions.

Static Members

`ALIGN`	Align of this class
`DEFAULT_NUM_SPUS`	Default number of SPUs
`DEFAULT_PPU_THREAD_PRIORITY`	Default PPU thread priority
`DEFAULT_SPU_THREAD_GROUP_PRIORITY`	Default SPU thread group priority

Static Methods

`initialize`	Initialize this instance

Methods

`finalize`	Finalize this instance
`enableExceptionEventHandler`	Enable exception event handler
`disableExceptionEventHandler`	Disable exception event handler
`getSpuThreadGroupId`	Get ID of SPU thread group
`getNumSpuThread`	Get a number of SPU threads
`getSpuThreadId`	Get identifiers of SPU threads
`getInfo`	Get SPURS internal information
`setMaxContention`	Set max contention to a workload
`setPriorities`	Set priorities to a workload

Description

After initialization this instance is to be a SPURS instance. `Spurs::initialize()` must be called to use this instance and `Spurs::finalize()` must be called to finish using this instance.

A pointer to this class can be used as a pointer to `CellSpurs` for an argument of libspurs API.

Examples

```
#include <cstdlib>
#include <cassert>
#include <spurs_util.h>
using namespace cell::Util::Spurs;

Spurs* spurs = (Spurs*)std::memalign(Spurs::ALIGN, sizeof(Spurs));
int ret;

ret = Spurs::initialize(spurs, "example");
assert(ret == CELL_OK);

// use the SPURS instance

ret = spurs->finalize();
assert(ret == CELL_OK);
std::free(spurs);
```

Notes

Proper lifetime of this instance is similar to the main PPU thread, because an initialization of SPURS instance includes creating SPU thread group, PPU threads, and system synchronization objects, and initialized SPURS instance is used with adding and removing task sets or job chains.

See Also

`CellSpurs` of "libspurs Core Reference"

Spurs::initialize

Initialize SPURS instance

Definition

```
#include <cellUtilSpurs.h>

static Spurs::initialize (
        Spurs* spurs,
        const char* prefix,
        uint32_t nSpus = DEFAULT_NUM_SPUS,
        int spuPriority = DEFAULT_SPU_THREAD_GROUP_PRIORITY,
        int ppuPriority = DEFAULT_PPU_THREAD_PRIORITY,
        Type type = TYPE_EXCLUSIVE_NON_CONTEXT
);

static Spurs::initialize (
        Spurs* spurs,
        const char* prefix,
        uint32_t nSpus,
        int spuPriority,
        int ppuPriority,
        bool exitIfNoWork,
        sys_memory_container_t container
);
```

Arguments

spurs	Pointer to an instance of class Spurs
prefix	String of the name prefix for the SPURS instance (maximum 15 characters)
nSpus	Number of SPU threads
spuPriority	Priority of SPU thread group
ppuPriority	Priority of PPU thread for SPURS handler
type	Type of SPURS instance
exitIfNoWork	Specifies whether or not to free the SPU when idle
container	ID of the memory container for creating SPU thread group

Return Values

CELL_OK(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
EAGAIN	0x80010001	Necessary resources could not be obtained due to ID shortage in Cell OS Lv-2
EAGAIN	0x80010001	Necessary resources could not be obtained due to memory shortage in the Cell OS Lv-2 kernel
EINVAL	0x80010002	*nSpus*, *spuPriority*, or *ppuPriority* is an invalid value
ENOMEM	0x80010004	Stack memory for the SPURS handlers could not be allocated
ENOMEM	0x80010004	Memory for the SPU thread group on which the SPURS kernels run could not be allocated

Macro	Value	Description
ESRCH	0x80010005	Memory container ID for SPU thread group specified in the SPURS attribute is invalid
EBUSY	0x8001000a	Exceeds the number of SPUs that can be occupied
CELL_SPURS_CORE_ERROR_INVAL	0x80410702	The length of *prefix* is more than 15 characters
CELL_SPURS_CORE_ERROR_PERM	0x80410709	Attribute of specified memory area for SPURS instance is invalid
CELL_SPURS_CORE_ERROR_ALIGN	0x80410710	*spurs* is not aligned to a 128-byte boundary
CELL_SPURS_CORE_ERROR_NULL_POINTER	0x80410711	*spurs* is a null pointer

Description

This method initializes a new SPURS instance and writes its management information to this instance. An instance of class Spurs must be initialized by this method.

prefix is used for a name prefix of an SPU thread group, SPU threads, and PPU threads created at initializing SPURS instance. *prefix* is maximum 15 character and NUL terminated string. The specified string can be discarded after returning from this method.

The number of SPUs is specified with *nSpus*. When *nSpus* is not specified, SPURS instance is initialized with 5 SPUs.

type is used to specify the behavior of SPU thread group running SPURS with a value below. When *type* is not specified, SPURS instance is initialized with TYPE_EXCLUSIVE_NON_CONTEXT and occupies specified number of SPUs.

Type	Description
TYPE_EXCLUSIVE_NON_CONTEXT	A SPURS instance occupies and runs on SPUs, and will not be preempted by any other SPU thread group.
TYPE_SHARED_WITH_ANY	SPUs can be shared with any priority SPU thread group. SPUs are released when SPURS instance has nothing to do, and it takes long time to start new work because of allocating SPUs for SPURS instance.
TYPE_SHARED_WITH_HIGHER	SPUs can be shared only with higher priority SPU thread group. SPUs wait for a new work when SPURS instance has nothing to do, and it takes short time to start new work.

See Also

cellSpursAttributeInitialize(), cellSpursAttributeSetNamePrefix(), cellSpursAttributeSetMemoryContainerForSpuThread(), and cellSpursInitializeWithAttribute() of "libspurs Core Reference"

Spurs::finalize

Release resources allocated for SPURS

Definition

```
#include <cellUtilSpurs.h>
int finalize();
```

Arguments

None

Return Values

CELL_OK(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
CELL_SPURS_CORE_ERROR_BUSY	0x8041070A	SPURS still has workloads
CELL_SPURS_CORE_ERROR_STAT	0x8041070F	SPURS is already terminated
CELL_SPURS_CORE_ERROR_STAT	0x8041070F	sys_spu_thread_group_destroy() returned an error code
CELL_SPURS_CORE_ERROR_ALIGN	0x80410710	An instance of class Spurs is not aligned to a 128-byte boundary
CELL_SPURS_CORE_ERROR_NULL_POINTER	0x80410711	A null pointer to class Spurs is used

Description

This method releases the resources allocated for SPURS.

An error will be returned if this method is called for a SPURS with workloads that have not yet been removed.

See Also

cellSpursFinalize() of "libspurs Core Reference"

Spurs::enableExceptionEventHandler

Enable an SPU exception event handler

Definition

```
#include <cellUtilSpurs.h>
inline int enableExceptionEventHandler()
{
        return cellSpursEnableExceptionEventHandler(this, true);
}
```

Arguments

None

Return Values

CELL_OK(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
CELL_SPURS_CORE_ERROR_ALIGN	0x80410710	An instance of class Spurs is not aligned to a 128-byte boundary
CELL_SPURS_CORE_ERROR_NULL_POINTER	0x80410711	A null pointer to Spurs class is used

Description

This method enables an SPU exception event handler.

Notes

An SPU exception event handler is enabled by default.

See Also

cellSpursEnableExceptionEventHandler() of "libspurs Core Reference"

Spurs::disableExceptionEventHandler

Disable an SPU exception event handler

Definition

```
#include <cellUtilSpurs.h>
inline int disableExceptionEventHandler()
{
        return cellSpursEnableExceptionEventHandler(this, false);
}
```

Arguments

None

Return Values

CELL_OK(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
CELL_SPURS_CORE_ERROR_ALIGN	0x80410710	An instance of class Spurs is not aligned to a 128-byte boundary
CELL_SPURS_CORE_ERROR_NULL_POINTER	0x80410711	A null pointer to Spurs class is used

Description

This method disables an S`PU exception event handler.

Notes

An SPU exception event handler is enabled by default.

See Also

cellSpursEnableExceptionEventHandler() of "libspurs Core Reference"

Spurs::getSpuThreadGroupId

Get ID of SPU thread group

Definition

```
#include <cellUtilSpurs.h>
inline int getSpuThreadGroupId (
        sys_spu_thread_group_t* group
)
{
        return cellSpursGetSpuThreadGroupId(this, group);
}
```

Arguments

group Location to store ID of SPU thread group

Return Values

CELL_OK(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
CELL_SPURS_CORE_ERROR_ALIGN	0x80410710	An instance of class Spurs is not aligned to a 128-byte boundary
CELL_SPURS_CORE_ERROR_NULL_POINTER	0x80410711	A null pointer to Spurs class is used
CELL_SPURS_CORE_ERROR_NULL_POINTER	0x80410711	*group* is a null pointer

Description

This method returns ID of SPURS instance.

See Also

cellSpursGetSpuThreadGroupId() of "libspurs Core Reference"

Spurs::getNumSpuThread

Get a number of SPU threads

Definition

```
#include <cellUtilSpurs.h>
inline int getNumSpuThread (
        unsigned* nThreads
)
{
        return cellSpursGetNumSpuThread(this, nThreads);
}
```

Arguments

nThreads	Location to store a number of SPU threads

Return Values

CELL_OK(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
CELL_SPURS_CORE_ERROR_ALIGN	0x80410710	An instance of class Spurs is not aligned to a 128-byte boundary
CELL_SPURS_CORE_ERROR_NULL_POINTER	0x80410711	A null pointer to Spurs class is used
CELL_SPURS_CORE_ERROR_NULL_POINTER	0x80410711	*nThreads* is a null pointer

Description

This method returns a number of SPU threads.

See Also

cellSpursGetNumSpuThread() of "libspurs Core Reference"

Spurs::getSpuThreadId

Get IDs of SPU threads

Definition

```
#include <cellUtilSpurs.h>
inline int getSpuThreadId (
        sys_spu_thread_t *thread,
        unsigned* nThreads
)
{
        return cellSpursGetSpuThreadId(this, thread, nThreads);
}
```

Arguments

thread	Location to store IDs of SPU threads
nThreads	Location to store the number of SPU threads

Return Values

CELL_OK(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
CELL_SPURS_CORE_ERROR_ALIGN	0x80410710	An instance of class Spurs is not aligned to a 128-byte boundary
CELL_SPURS_CORE_ERROR_NULL_POINTER	0x80410711	A null pointer to Spurs class is used
CELL_SPURS_CORE_ERROR_NULL_POINTER	0x80410711	*thread* is a null pointer
CELL_SPURS_CORE_ERROR_NULL_POINTER	0x80410711	*nThreads* is a null pointer

Description

This method writes the IDs of the SPU threads allocated for the SPURS to *thread*.

Provide an array and specify its starting address to *thread* and the number of elements to **nThreads*. If the *thread* array is large enough, IDs of all the SPU threads allocated to the SPURS will be written to the array. Otherwise, only as many IDs as the number specified with **nThreads* will be written to the *thread* array. In either case, the number of SPU threads actually allocated for the SPURS will be returned to **nThreads*.

See Also

cellSpursGetSpuThreadId() of "libspurs Core Reference"

Spurs::getInfo

Get SPURS internal information

Definition

```
#include <cellUtilSpurs.h>
inline int getInfo (
        CellSpursInfo* info
)
{
        return cellSpursGetInfo(this, info);
}
```

Arguments

info Pointer to `CellSpursInfo` to write SPURS internal information

Return Values

`CELL_OK`(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
`CELL_SPURS_CORE_ERROR_ALIGN`	0x80410710	An instance of class Spurs is not aligned to a 128-byte boundary
`CELL_SPURS_CORE_ERROR_NULL_POINTER`	0x80410711	A null pointer to Spurs class is used
`CELL_SPURS_CORE_ERROR_NULL_POINTER`	0x80410711	*info* is a null pointer

Description

This method is used to get internal information of the SPURS instance. For details on the internal information, please refer to the description of the `CellSpursInfo` structure.

See Also

`CellSpursInfo`, `cellSpursGetInfo()` of "libspurs Core Reference"

Spurs::setMaxContention

Set maximum contention

Definition

```
#include <cellUtilSpurs.h>
inline int setMaxContention (
        CellSpursWorkloadId workloadId,
        unsigned int maxContention
)
{
        return cellSpursSetMaxContention(this, workloadId, maxContention);
}
```

Arguments

workloadId	Workload ID
maxContention	Maximum contention

Return Values

CELL_OK(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
CELL_SPURS_CORE_ERROR_INVAL	0x80410702	Invalid workload ID is specified
CELL_SPURS_CORE_ERROR_SRCH	0x80410705	Workload corresponding to the specified ID cannot be found
CELL_SPURS_CORE_ERROR_STAT	0x8041070F	SPURS cannot continue due to SPU exception
CELL_SPURS_CORE_ERROR_ALIGN	0x80410710	An instance of class Spurs is not aligned to a 128-byte boundary
CELL_SPURS_CORE_ERROR_NULL_POINTER	0x80410711	A null pointer to Spurs class is used

Description

This method sets maximum contention of the workload specified by *workloadId* to *maxContention*.

See Also

cellSpursSetMaxContention() of "libspurs Core Reference"

Spurs::setPriorities

Set priorities

Definition

```
#include <cellUtilSpurs.h>
inline int setPriorities (
        CellSpursWorkloadId workloadId,
        const uint8_t priorities[CELL_SPURS_MAX_SPU]
)
{
        return cellSpursSetPriorities(this, workloadId, priorities);
}
```

Arguments

workloadId	Workload ID
priorities	Priorities for each SPU of the workload

Return Values

CELL_OK(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
CELL_SPURS_CORE_ERROR_INVAL	0x80410702	Invalid workload ID is specified
CELL_SPURS_CORE_ERROR_INVAL	0x80410702	Invalid value specified for *priorities*
CELL_SPURS_CORE_ERROR_SRCH	0x80410705	Workload corresponding to the specified ID cannot be found
CELL_SPURS_CORE_ERROR_STAT	0x8041070F	SPURS cannot continue due to SPU exception
CELL_SPURS_CORE_ERROR_ALIGN	0x80410710	An instance of class Spurs is not aligned to a 128-byte boundary
CELL_SPURS_CORE_ERROR_NULL_POINTER	0x80410711	A null pointer to Spurs class is used

Description

This method sets priorities for each SPU of the workload specified by *workloadId* to *priorities*. *priorities* is disposable after returning from this method.

See Also

cellSpursSetPriorities() of "libspurs Core Reference"

Taskset Class

Taskset Class

Class of SPURS task set

Definition

```
#include <cellUtilSpursTaskset.h>
namespace cell {
        namespace Util {
              namespace Spurs {
                    class Taskset : public CellSpursTaskset {
                    };
              }
        }
}
```

Conditions

This instance must be in an SPU-accessible area.

This instance must be kept until successful completion of `Taskset::join()`

Conditions of Inheritance

Constructors must be empty.

Destructors must not be defined.

Copy constructor and copy operator must be declared as private to invalidate them.

Do not declare virtual functions.

Static Members

`ALIGN`	Alignment of this class

Static Methods

`create`	Create a SPURS task set for a SPURS instance

Methods

`shutdown`	Shut down task set
`join`	Wait for completion of shutdown and remove task set
`getTasksetId`	Get workload ID of SPURS task set
`getSpursAddress`	Get a pointer to SPURS instance
`getInfo`	Get task set information

Description

This class is a SPURS task set that it is possible to wait for an exit of task. `Taskset::create()` must be called to use this instance, and `Taskset::join()` must be called to make sure the completion of shutdown requested by `Taskset::shutdown()` or `cellSpursShutdownTaskset()`.

A pointer to this class can be used as a pointer to `CellSpursTaskset` for an argument of libspurs API.

Examples

```
#include <cstdlib>
#include <cassert>
#include <spurs_util.h>
using namespace cell::Util::Spurs;

Taskset* taskset = (Taskset*)std::memalign(Taskset::ALIGN, sizeof(Taskset));
int ret;

ret = Taskset::create(taskset, "example", spurs, 0);
assert(ret == CELL_OK);

// use SPURS taskset

ret = taskset->shutdown();
assert(ret == CELL_OK);
ret = taskset->join();
assert(ret == CELL_OK);
std::free(taskset);
```

Notes

`Taskset::create()`, `Taskset::shutdown()`, and `cellSpursShutdownTaskset()` request all SPUs to run system service, and workload switch occurs on all SPUs. Therefore proper lifetime of SPURS task set is longer than a frame enough to avoid this overhead.

And also it is a big overhead and waste of resources to create a SPURS task set for each creation of SPURS task. Although only maximum 16 SPURS task set can be created in one SPURS instance with above overhead, maximum 128 SPURS task can be created in each SPURS task set without disturbing SPUs. Therefore continuous use of the same SPURS task set is preferable.

The size of `CellSpursTaskset` in this class is `CELL_SPURS_TASKSET_CLASS1_SIZE` to use extended features.

See Also

`CellSpursTaskset` of "libspurs Task Reference"

Taskset::create

Create a SPURS task set

Definition

```
#include <cellUtilSpursTaskset.h>

int create (
        Taskset* taskset,
        const char* name,
        Spurs* spurs,
        uint64_t argument,
        uint8_t priority = CELL_SPURS_MAX_PRIORITY - 1,
        uint8_t spuMask = 0xff
);

int create (
        Taskset* taskset,
        const char* name,
        Spurs* spurs,
        uint64_t argument,
        const uint8_t priority[CELL_SPURS_MAX_SPU],
        unsigned int maxContention
);
```

Arguments

taskset	Pointer to an instance of Taskset
name	Name of this SPURS task set
spurs	Pointer to initialized instance of class Spurs
argument	Common arguments within a SPURS task set
priority	Priority of a SPURS task set per SPU
spuMask	Mask of SPUs to apply priority (LSB=SPU0)
maxContention	Maximum contention

Return Values

CELL_OK(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
CELL_SPURS_TASK_ERROR_AGAIN	0x80410901	There are already 16 workloads and a task set cannot be created
CELL_SPURS_TASK_ERROR_INVAL	0x80410902	Value of *priority* is invalid
CELL_SPURS_TASK_ERROR_STAT	0x8041090F	The SPURS cannot continue due to an SPU exception
CELL_SPURS_TASK_ERROR_ALIGN	0x80410910	*spurs* or *taskset* is not aligned to a 128-byte boundary
CELL_SPURS_TASK_ERROR_NULL_POINTER	0x80410911	*spurs* or *taskset* is a null pointer

Description

This method creates a SPURS task set for a SPURS instance specified with *spurs*, and writes the management information of this newly created SPURS task set in the memory specified by *taskset*.

For *argument*, specify the common arguments that are to be passed to all SPURS tasks within this SPURS task set.

For *priority*, specify the priority of this SPURS task set. You can specify a value from 1 to 15. An SPU will prioritize execution of a SPURS task set with the smaller priority value. There is two way to specify the priority of this SPURS task set. The one is common *priority* for all SPU with *spuMask*, and the other is a different *priority* for each SPU with *maxContention*. If a priority of an SPU is 0, the task set will not be executed on that SPU.

For *spuMask*, specify the bit mask of SPUs that the LSB is SPU0 and a value 1 applies the common priority to the corresponding SPU.

For *maxContention*, specify this SPURS task set's maximum contention (the maximum number of SPUs on which this SPURS task set can be executed in parallel).

Notes

Because SPURS task set keeps just a pointer to a name strings, the name string pointed to by *name* must exist with the task set.

See Also

`cellSpursTasksetAttributeInitialize()`, `cellSpursTasksetAttributeSetName()`, `cellSpursTasksetAttributeSetTasksetSize()`, and `cellSpursCreateTasksetWithAttribute()` of "libspurs Task Reference"

Taskset::shutdown

Shut down task set

Definition

```
#include <cellUtilSpursTaskset.h>
inline int shutdown ()
{
        return cellSpursShutdownTaskset(this);
}
```

Arguments

None

Return Values

CELL_OK(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
CELL_SPURS_TASK_ERROR_INVAL	0x80410902	Specified task set does not exist
CELL_SPURS_TASK_ERROR_STAT	0x8041090F	The SPURS cannot continue due to the SPU exception
CELL_SPURS_TASK_ERROR_ALIGN	0x80410910	The instance of Taskset is not aligned to a 128-byte boundary
CELL_SPURS_TASK_ERROR_NULL_POINTER	0x80410911	A null pointer to class Taskset is used

Description

This method shuts down the SPURS task set, and prevents any SPURS task in this task set from being activated hereafter. Note that all tasks - being executed when this method is called - will be continued without being stopped.

To confirm completion, use Taskset::join().

See Also

cellSpursShutdownTaskset() of "libspurs Task Reference"

Taskset::join

Wait for completion of shutdown and remove task set

Definition

```
#include <cellUtilSpursTaskset.h>
inline int join ()
{
        return cellSpursJoinTaskset(this);
}
```

Arguments

None

Return Values

CELL_OK(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
CELL_SPURS_TASK_ERROR_INVAL	0x80410902	Specified task set does not exist
CELL_SPURS_TASK_ERROR_INVAL	0x80410902	Another PPU thread is already waiting for the end of this task set
CELL_SPURS_TASK_ERROR_STAT	0x8041090F	The SPURS cannot continue due to the SPU exception
CELL_SPURS_TASK_ERROR_ALIGN	0x80410910	The instance of Taskset is not aligned to a 128-byte boundary
CELL_SPURS_TASK_ERROR_NULL_POINTER	0x80410911	A null pointer of class Taskset is used

Description

This method waits for completion of shutdown and removes this task set. It blocks the caller execution until this task set is shutdowned and all SPURS tasks in this task set become not running.

Notes

In order to shut down SPURS task set, the system service must run on all SPUs in its SPURS instance. Therefore any SPU program without yielding its SPU can stop a caller of this method because of disturbing a shutdown operation.

See Also

cellSpursJoinTaskset() of "libspurs Task Reference"

Taskset::getId

Get workload ID of SPURS task set

Definition

```
#include <cellUtilSpursTaskset.h>
inline int getId (
        CellSpursWorkloadId *workloadId
)
{
        return cellSpursGetTasksetId(this, workloadId);
}
```

Arguments

workloadId	Location to store workload ID of SPURS task set

Return Values

CELL_OK(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
CELL_SPURS_TASK_ERROR_ALIGN	0x80410910	An instance of class Taskset is not aligned to a 128-byte boundary
CELL_SPURS_TASK_ERROR_NULL_POINTER	0x80410911	A null pointer of class Taskset is used

Description

This method returns the workload ID of this task set.

This method returns invalid workload ID after Taskset::join() is called.

See Also

cellSpursGetTasksetId() of "libspurs Task Reference"

Taskset::getSpursAddress

Get SPURS instance

Definition

```
#include <cellUtilSpursTaskset.h>
inline int getSpursAddress (
        CellSpurs **spurs
)
{
        return cellSpursGetSpursAddress(this, spurs);
}
```

Arguments

spurs	Location to store a pointer to a SPURS instance

Return Values

CELL_OK(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
CELL_SPURS_TASK_ERROR_INVAL	0x80410902	Specified task set does not exist
CELL_SPURS_TASK_ERROR_ALIGN	0x80410910	An instance of class Taskset is not aligned to a 128-byte boundary
CELL_SPURS_TASK_ERROR_NULL_POINTER	0x80410911	A null pointer of class Taskset is used
CELL_SPURS_TASK_ERROR_NULL_POINTER	0x80410911	*spurs* is a null pointer

Description

This method returns a pointer to a SPURS instance that this task set belongs to.

See Also

cellSpursGetSpursAddress() of "libspurs Task Reference"

Taskset::getInfo

Get task set information

Definition

```
#include <cellUtilSpursTaskset.h>
inline int getInfo (
        CellSpursTasksetInfo* info
)
{
        return cellSpursGetTasksetInfo(this, info);
}
```

Arguments

info	Location to store information of this task set

Return Values

CELL_OK(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
CELL_SPURS_TASK_ERROR_INVAL	0x80410902	Specified task set does not exist
CELL_SPURS_TASK_ERROR_ALIGN	0x80410910	An instance of class Taskset is not aligned to a 128-byte boundary
CELL_SPURS_TASK_ERROR_NULL_POINTER	0x80410911	A null pointer of class Taskset is used
CELL_SPURS_TASK_ERROR_NULL_POINTER	0x80410911	*info* is a null pointer

Description

This method returns information of this SPURS task set.

Notes

Do not perform synchronization with a task based only on the information obtained by using this method. For example, do not periodically repeat calling this method in order to detect some changes on the task to perform some processing.

This method does not guarantee to provide a consistent snapshot of the task set information, which may change dynamically.

See Also

CellSpursTasksetInfo and cellSpursGetTasksetInfo() of "libspurs Task Reference"

Task Class

Task Class

Class of SPURS task

Definition

```
#include <cellUtilSpursTask.h>
namespace cell {
        namespace Util {
              namespace Spurs {
                    class Task;
              }
        }
}
```

Conditions

This instance must be in an SPU-accessible area.

This instance must be kept until successful completion of `Task::join()`

Static Members

`ALIGN`	Alignment of this class

Static Methods

`create`	Create a SPURS task

Methods

`id`	Get SPURS task ID
`saveBuffer`	Get a pointer to a context save buffer
`join`	Wait for termination of a task
`tryJoin`	Test termination of a task

Description

This class creates SPURS task and waits for termination of the task. `Task::create()` must be called to use this instance, and `Task::join()` must be called to confirm termination of the task. After the confirmation of terminated task, it is safe to release task context save buffer and this instance.

Examples

Run a regular SPURS task

```
#include <cstdlib>
#include <cassert>
#include <spurs_util.h>
using namespace cell::Util::Spurs;

extern const char _binary_atsk_hello_elf_start[];

Task* task = (Task*)std::memalign(Task::ALIGN, sizeof(Task));
void* context = std::memalign(128, CELL_SPURS_TASK_CONTEXT_SIZE_ALL);
int ret;

ret = Task::create(task, taskset, _binary_task_hello_elf_start, 0, context);
```

```
	assert(ret == CELL_OK);

	// do another things

	ret = task->join();
	assert(ret == CELL_OK);
	std::free(task->saveBuffer());
	std::free(task);
```

Task::create

Create a SPURS task

Definition

```
#include <cellUtilSpursTask.h>

// for full context task
static int Task::create (
        Task* task,
        Taskset* taskset,
        const void* elf,
        const CellSpursTaskArgument& argument,
        const void* context
);

// for full context task
static int Task::create (
        Task* task,
        Taskset* taskset,
        const void* elf,
        const CellSpursTaskArgument* argument,
        const void* context
);

// for context free task
static int Task::create (
        Task* task,
        Taskset* taskset,
        const void* elf,
        const CellSpursTaskArgument& argument
);

// for context free task
static int Task::create (
        Task* task,
        Taskset* taskset,
        const void* elf,
        const CellSpursTaskArgument* argument
);

static int Task::create (
        Task* task,
        Taskset* taskset,
        const TaskElf& taskElf,
        const CellSpursTaskArgument& argument,
        const void* saveBuffer,
        unsigned saveBufferSize
);

static int Task::create (
        Task* task,
        Taskset* taskset,
        const TaskElf& taskElf,
        const CellSpursTaskArgument* argument,
        const void* saveBuffer,
        unsigned saveBufferSize
);

static int Task::create (
        Task* task,
```

```
            Taskset* taskset,
            const void* elf,
            const CellSpursTaskArgument* argument,
            const CellSpursTaskSaveConfig* config
);
```

Arguments

task	Pointer to an instance of class Task
taskset	Pointer to an instance of class Taskset initialized by `Taskset::create`
elf	Pointer to an ELF image of SPURS task (16-byte alignment)
argument	Argument to be passed to the SPURS task
context	Pointer to a context save storage of a SPURS task (128-byte aligned and `CELL_SPURS_TASK_CONTEXT_SIZE_ALL` bytes)
config	Pointer to a configuration of a SPURS task context saving
taskElf	An instance of class TaskElf
saveBuffer	Pointer to a context save storage of SPURS task (128-byte alignment)
saveBufferSize	Size of a context save storage area

Return Values

`CELL_OK`(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
`CELL_SPURS_TASK_ERROR_AGAIN`	0x80410901	Number of SPURS tasks reached limit
`CELL_SPURS_TASK_ERROR_INVAL`	0x80410902	Area indicated by the LS pattern is overlapped with SPURS management area
`CELL_SPURS_TASK_ERROR_INVAL`	0x80410902	*saveBufferSize* is smaller than the required size
`CELL_SPURS_TASK_ERROR_NOEXEC`	0x80410907	The ELF image is invalid
`CELL_SPURS_TASK_ERROR_STAT`	0x8041090F	The SPURS cannot continue due to the SPU exception
`CELL_SPURS_TASK_ERROR_ALIGN`	0x80410910	*context* or *elf* is not aligned to a 16-byte boundary
`CELL_SPURS_TASK_ERROR_NULL_POINTER`	0x80410911	*taskset* or *elf* is a null pointer

Description

This method creates a SPURS task in the task set specified by *taskset* and prepares to synchronize with completion of the created task. The created SPURS task will be automatically started.

Place an ELF image to be executed as the SPURS task on main memory and specify its effective address in *elf*.

For *argument*, specify an argument to be passed to the created SPURS task (not a common argument). Both a pointer and a reference are acceptable. Because an SPURS task receives this argument as a qword, any 16 bytes type is also acceptable. A null pointer can be specified as all 0 values.

For *context*, *savebuffer*, and *config*, specify a context save buffer of SPURS task. When either of them is not specified a run-complete type SPURS task will be created, that the task does not have its context save buffer and cannot use a blocking interface of synchronization libraries.

For context, specify a context save buffer of SPURS task that the size of the buffer is `CELL_SPURS_TASK_CONTEXT_SIZE_ALL`, and created task can save whole context.

For *taskElf*, specify the LS area to be saved as a context, and can create a task with partial LS context. Minimum required context save buffer by specifying *saveBuffer* and *saveBufferSize* can save memory budget.

For *config*, specify the LS area to be saved as a context, and can also create a task with partial LS context. Details are in "libspurs Overview".

Notes

An ELF image and a context save buffer of a SPURS task must be in an SPU-accessible are.

They must be kept until successful completion of `Task::join()`.

Although they must be aligned to a 16-byte boundary, 128-byte alignment is preferable for best performance DMA transfer.

See Also

Chapter 8 "Optimizing a SPURS Task Application" of "libspurs Overview"

`cellSpursCreateTaskWithAttribute()` of "libspurs Task Reference"

Task::id

Get ID of SPURS task

Definition

```
#include <cellUtilSpursTask.h>
inline CellSpursTaskId id () const;
```

Arguments

None

Description

This method gets ID of a SPURS task.

Notes

ID from this method is valid only from success of `Task::create()` to termination of the created task. Note that ID is not valid until success of `Task::join()` because ID is possibly reused immediately after termination of the task. It is not recommended to use this ID without absolutely certain existence of the target task.

Task::saveBuffer

Get context save buffer of SPURS task

Definition

```
#include <cellUtilSpursTask.h>
inline void* saveBuffer() const;
```

Arguments

None

Description

This method returns the pointer to context save buffer of the SPURS task created by `Task::create()`. After `Task::join()`, it is safe to release the context save buffer of the address from this method.

When `Task::create()` has been failed, this method returns invalid value.

Task::join

Wait for termination of SPURS task

Definition

```
#include <cellUtilSpursTask.h>
inline int join (
        int* value = 0
);
```

Arguments

value	Location to store an exit code

Return Values

CELL_OK(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
CELL_SPURS_TASK_ERROR_AGAIN	0x80410901	Not enough resources are available to become a waiting state
CELL_SPURS_TASK_ERROR_BUSY	0x8041090A	Another PPU thread or SPURS task is already waiting for obtaining the specified SPURS task exit code
CELL_SPURS_TASK_ERROR_STAT	0x8041090F	Specified task exit code container is not attached to any SPURS task
CELL_SPURS_TASK_ERROR_ALIGN	0x80410910	An instance of class Task is not aligned to a 128-byte boundary
CELL_SPURS_TASK_ERROR_NULL_POINTER	0x80410911	A null pointer to class Task is used
CELL_SPURS_TASK_ERROR_SHUTDOWN	0x80410920	The SPURS task to which the task exit code container is attached, is shutdowned before it completes execution

Description

This method waits for termination of corresponding SPURS task and stores its exit code to *value*. When the SPURS task set is shutdowned before the SPURS task completes execution, this method returns with CELL_SPURS_TASK_ERROR_SHUTDOWN. When *value* is a null pointer, the exit code is discarded.

See Also

cellSpursTaskExitCodeGet() of "libspurs Task Reference"

Task::tryJoin

Make sure termination of SPURS task

Definition

```
#include <cellUtilSpursTask.h>
inline int tryJoin (
        int* value = 0
);
```

Arguments

value	Location to store an exit code

Return Values

CELL_OK(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
CELL_SPURS_TASK_ERROR_AGAIN	0x80410901	Not enough resources are available to become a waiting state
CELL_SPURS_TASK_ERROR_BUSY	0x8041090A	Another PPU thread or SPURS task is already waiting for obtaining the specified SPURS task exit code
CELL_SPURS_TASK_ERROR_STAT	0x8041090F	Specified task exit code container is not attached to any SPURS task
CELL_SPURS_TASK_ERROR_ALIGN	0x80410910	An instance of class Task is not aligned to a 128-byte boundary
CELL_SPURS_TASK_ERROR_NULL_POINTER	0x80410911	A null pointer to class Task is used
CELL_SPURS_TASK_ERROR_SHUTDOWN	0x80410920	The SPURS task to which the task exit code container is attached, is shut downed before it completes execution

Description

This method makes sure termination of corresponding SPURS task and stores its exit code to *value*. When the task has not completed its execution, this method returns CELL_SPURS_TASK_ERROR_AGAIN. When the SPURS task set is shut downed before the SPURS task completes execution, this method returns with CELL_SPURS_TASK_ERROR_SHUTDOWN. When *value* is a null pointer, the exit code is discarded.

See Also

cellSpursTaskExitCodeTryGet() of "libspurs Task Reference"

TaskElf Class

TaskElf Class

Class of SPURS task's ELF

Definition

```
#include <cellUtilSpursTaskElf.h>
namespace cell {
        namespace Util {
              namespace Spurs {
                    class TaskElf;
              }
        }
}
```

Conditions

An instance of TaskElf class can be discarded after calling `Task::create()` with this.

Constructors and Destructors

`TaskElf`	Constructor

Methods

`elf`	Get an address of an ELF image
`lsPattern`	Get LS pattern
`saveBufferSize`	Get required size of context save buffer
`saveBufferAlign`	Get alignment of context save buffer
`unsetAll`	Unset entire LS area from LS context
`unsetRange`	Unset specified LS area from LS context
`unsetReadOnlySegment`	Unset read only ELF segment from LS context
`setAll`	Set entire LS area as LS context
`setRange`	Set specified address range as LS context
`setWritableSegment`	Set writable ELF segment as LS context
`setStack`	Set specified address range as LS context

Description

This class generates LS pattern to specify LS context area based on ELF information of SPURS task. In order for SPURS task to wait on synchronization libraries, context save buffer is required to the task, but it costs about 244 Kbytes to save entire context. By limiting the LS area to be saved using an LS pattern, context save buffer can be reduced. This class uses ELF segment information of SPURS task that PPU can refer.

Examples

See "Easy SPURS Overview"

Notes

An instance of TaskElf class is referred only while `Task::create()` is called.

See Also

Chapter 8 "Optimizing a SPURS Task Application" of "libspurs Overview"

TaskElf::TaskElf

Constructor of TaskElf Class

Definition

```
#include <cellUtilSpursTaskElf.h>
inline TaskElf (const void* elf);
```

Arguments

elf Pointer to an ELF image of a SPURS task (16-byte align)

Return Values

None

Description

This constructor initializes the instance for a SPURS task specified by *elf* and clears its LS pattern.

TaskElf::lsPattern

Get LS pattern

Definition

```
#include <cellUtilSpursTaskElf.h>
inline const CellSpursTaskLsPattern* lsPattern () const;
```

Arguments

None

Description

This method returns a pointer to LS pattern. Because this pointer points to a member of TaskElf class, the valid time of this pointer is the same as the lifetime of this instance of TaskElf.

See Also

`CellSpursTaskLsPattern` of "libspurs Task Reference"

TaskElf::saveBufferSize

Get size of context save buffer

Definition

```
#include <cellUtilSpursTaskElf.h>
unsigned saveBufferSize () const;
```

Arguments

None

Description

This method returns a size of save buffer to save SPURS task. The size is calculated from current LS pattern.

Notes

The size is 2 Kbytes for each bit of LS pattern with `CELL_SPURS_TASK_EXECUTION_CONTEXT_SIZE`.

TaskElf::saveBufferAlign

Get alignment of context save buffer

Definition

```
#include <cellUtilSpursTaskElf.h>
static inline unsigned saveBufferAlign()
{
        return CELL_SPURS_TASK_CONTEXT_ALIGN;
}
```

Arguments

None

Description

This method returns the alignment of context save buffer for SPURS task.

TaskElf::unsetAll

Unset entire LS area from LS context

Definition

```
#include <cellUtilSpursTaskElf.h>
inline void unsetAll ();
```

Arguments

None

Return Values

None

Description

This method clears LS pattern. And the result is that entire LS area is out of LS context.

TaskElf::unsetRange

Unset specified LS area from LS context

Definition

```
#include <cellUtilSpursTaskElf.h>
void unsetRange (
        unsigned start,
        unsigned size
);
```

Arguments

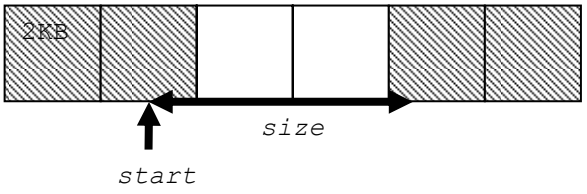
start	Starting address
size	Size of the area

Return Values

None

Description

This method unsets specified LS range not to be saved as LS context. The LS context is 2-Kbyte aligned and sized blocks within the specified range from *start* address with *size* bytes.



TaskElf::unsetReadOnlySegment

Unset read only ELF segment from LS context

Definition

```
#include <cellUtilSpursTaskElf.h>
void unsetReadOnlySegment ();
```

Arguments

None

Return Values

None

Description

This method unsets read only ELF segment of a SPURS task specified by `TaskElf::TaskElf()` not to be saved as LS context. The LS context is 2-Kbyte aligned and sized blocks within the read only ELF segment.

Examples

```
TaskElf elf(_binary_task_hello_elf_start);
elf.setAll();
elf.unsetReadOnlySegment();
```

Notes

When a SPURS task without entire LS context is restored to run, read only ELF segment of the SPURS task is transferred to LS automatically. Therefore it is possible to cut large size of context save buffer of ordinary task with large code size only by unsetting its read only ELF segment from LS context.

An instance of TaskElf class is initialized with no LS context. `TaskElf::setAll()` must be called before calling this method.

See Also

`TaskElf::setAll()`

TaskElf::setAll

Set entire LS area as LS context

Definition

```
#include <cellUtilSpursTaskElf.h>
inline void setAll ();
```

Arguments

None

Return Values

None

Description

This method sets entire LS area as LS context. The LS pattern pointed by `TaskElf::lsPattern()` is the same as `CELL_SPURS_TASK_LS_ALL` and `gCellSpursTaskLsAll`.

TaskElf::setRange

Set specified address range as LS context

Definition

```
#include <cellUtilSpursTaskElf.h>
void setRange (
        unsigned start,
        unsigned size
);
```

Arguments

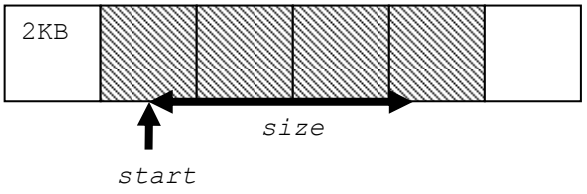
start	Starting address
size	Size of the area

Return Values

None

Description

This method sets specified LS range to be saved as LS context. The LS context is 2-Kbyte aligned and sized blocks including from *start* address with *size* bytes.



TaskElf::setWritableSegment

Set writable ELF segment as LS context

Definition

```
#include <cellUtilSpursTaskElf.h>
void setWritableSegment ();
```

Arguments

None

Return Values

None

Description

This method sets writable ELF segment of a SPURS task specified by `TaskElf::TaskElf()` to be saved as LS context. The LS context is 2-Kbyte aligned and sized blocks including the writable ELF segment.

Examples

```
TaskElf elf(_binary_task_hello_elf_start);
elf.unsetAll();
elf.setWritableSegment();
elf.setStack(2048);
```

Notes

An instance of TaskElf is initialized with no LS context. When writable ELF segment is set as LS context from no LS context, stack must be also added to LS context by calling `TaskElf::setStack()`.

See Also

`TaskElf::setStack()`

TaskElf::setStack

Set specified address range as LS context

Definition

```
#include <cellUtilSpursTaskElf.h>
inline void setStack (
        unsigned stackSize
)
{
        setRange(CELL_SPURS_TASK_BOTTOM - stackSize, stackSize);
}
```

Arguments

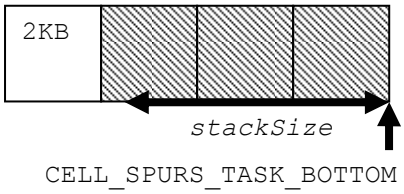
stackSize	A stack size of LS context save target

Return Values

None

Description

This method sets specified LS range to be saved as LS context. The LS context is 2-Kbyte aligned and sized blocks including *stackSize* from bottom LS address of SPURS task.



Event Flag Class

EventFlag Class

SPURS event flag class

Definition

```
#include <cellUtilSpursEventFlag.h>
namespace cell {
        namespace Util {
              namespace Spurs {
                    class EventFlag : public CellSpursEventFlag;
              }
        }
}
```

Conditions

This instance must be in an SPU-accessible area.

Static Members

`ALIGN`	Alignment of this class

Static Methods

`initialize`	Initialize

Methods

`set`	Set event flag
`wait`	Wait for event flag
`tryWait`	Wait for event flag
`attachLv2EventQueue`	Create and attach event queue
`detachLv2EventQueue`	Detach and delete event queue

Description

This is an abstracted class of the SPURS event flag. `EventFlag::initialize()` must be called to start the initialization process before using the instance.

See Also

"SPURS Event Flag" section in Chapter 6 "Task Synchronization Libraries" of "libspurs Overview"

EventFlag::initialize

Initialize SPURS event flag

Definition

```
#include <cellUtilSpursEventFlag.h>

static int EventFlag::initialize (
        EventFlag* eventflag,
        const Spurs* spurs
);
```

Arguments

eventflag	Pointer to the event flag instance
spurs	Pointer to the SPURS instance

Return Values

CELL_OK(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
CELL_SPURS_TASK_ERROR_INVAL	0x80410902	*eventflag* or *spurs* is invalid
CELL_SPURS_TASK_ERROR_ALIGN	0x80410910	*eventflag* or *spurs* is not aligned to a 128-byte boundary
CELL_SPURS_TASK_ERROR_NULL_POINTER	0x80410911	*eventflag* or *spurs* is a null pointer

Description

This method initializes the SPURS event flag specified by *eventflag* with the following conditions.

Condition	Setting
Communication target	Between all workloads
Communication direction	CELL_SPURS_EVENT_FLAG_ANY2ANY
Clear condition	CELL_SPURS_EVENT_FLAG_CLEAR_AUTO

See Also

cellSpursEventFlagInitializeIWL() of "libspurs Task Reference"

EventFlag::set

Set SPURS event flag

Definition

```
#include <cellUtilSpursEventFlag.h>
int set (
        uint16_t bits
)
{
        return cellSpursEventFlagSet(this, bits);
}
```

Argument

bits Bit pattern to set

Return Values

CELL_OK(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
CELL_SPURS_TASK_ERROR_ALIGN	0x80410910	this is not aligned to a 128-byte boundary
CELL_SPURS_TASK_ERROR_NULL_POINTER	0x80410911	this is a null pointer

Description

This method sets 1 to the bits of the SPURS event flag.

For *bits*, specify the bit pattern to set. The OR result of the event flag's current value and the value specified in *bits* will be the new value of the event flag.

When another SPURS task or PPU thread waits for the SPURS event flag, and the new value of the event flag satisfies the release condition, that SPURS task or PPU thread will be released from the waiting state.

See Also

cellSpursEventFlagSet() of "libspurs Task Reference"

EventFlag::wait

Wait for a SPURS event flag

Definition

```
#include <cellUtilSpursEventFlag.h>
inline int wait (
        uint16_t* mask,
        CellSpursEventFlagWaitMode mode
)
{
        return cellSpursEventFlagWait(this, mask, mode);
}
```

Arguments

mask	Pointer to the variable for specifying the bit pattern of the release condition and receiving the bit pattern upon release
mode	Wait mode

Return Values

CELL_OK(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
CELL_SPURS_TASK_ERROR_AGAIN	0x80410901	The specified release condition conflicts with the release condition that is already being waited for
CELL_SPURS_TASK_ERROR_INVAL	0x80410902	Value of *mode* is invalid
CELL_SPURS_TASK_ERROR_STAT	0x8041090F	Cell OS Lv-2 event queue is not attached
CELL_SPURS_TASK_ERROR_ALIGN	0x80410910	this is not aligned to a 128-byte boundary
CELL_SPURS_TASK_ERROR_NULL_POINTER	0x80410911	this or *mask* is a null pointer

Description

This method waits for a SPURS event flag to be set. The PPU thread that calls this method will enter the waiting state until the release condition specified by *mask* and *mode* is fulfilled.

For *mask*, specify a bit pattern to serve as the condition by which a release is to be made from the wait state.

For *mode*, specify one of the following macro definitions.

Macro	Description
CELL_SPURS_EVENT_FLAG_OR	OR mode: The release condition is satisfied when any of the bits, of the bit pattern specified by *mask*, is set.
CELL_SPURS_EVENT_FLAG_AND	AND mode: The release condition is satisfied when all of the bits, of the bit pattern specified by *mask*, is set.

This method returns when the value of the event flag satisfies the specified condition upon the cellSpursEventFlagSet() call of another SPURS task, or the EventFlag::set() call of a PPU thread. At this time, the value of the event flag that satisfied the condition will be stored in **mask*.

Notes

When a PPU thread calls this method, `EventFlag::attachLv2EventQueue()` must be called and the Cell OS Lv-2 event queue must be attached in advance. Only one PPU thread can wait for an event flag at one time.

EventFlag::tryWait

Wait for a SPURS event flag

Definition

```
#include <cellUtilSpursEventFlag.h>
inline int tryWait (
        uint16_t* mask,
        CellSpursEventFlagWaitMode mode
)
{
        return cellSpursEventFlagTryWait(this, mask, mode);
}
```

Arguments

mask	Pointer to the variable for specifying the bit pattern of the release condition and receiving the bit pattern upon release
mode	Wait mode

Return Values

CELL_OK(0) is returned for success, and the value of the event flag that satisfied the release condition will be stored in **mask*.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
CELL_SPURS_TASK_ERROR_INVAL	0x80410902	Value of *mode* is invalid
CELL_SPURS_TASK_ERROR_BUSY	0x8041090A	The release condition has not been satisfied, or another PPU thread is waiting for this event flag
CELL_SPURS_TASK_ERROR_ALIGN	0x80410910	this is not aligned to a 128-byte boundary
CELL_SPURS_TASK_ERROR_NULL_POINTER	0x80410911	this or *mask* is a null pointer

Description

This method waits for a SPURS event flag to be set.

For *mask*, specify a bit pattern to serve as the condition by which a release is to be made from the wait state.

For *mode*, specify one of the following macro definitions.

Macro	Description
CELL_SPURS_EVENT_FLAG_OR	OR mode: The release condition is satisfied when any of the bits, of the bit pattern specified by *mask*, is set
CELL_SPURS_EVENT_FLAG_AND	AND mode: The release condition is satisfied when all of the bits, of the bit pattern specified by *mask*, is set

EventFlag::attachLv2EventQueue

Attach SPURS event flag and event queue

Definition

```
#include <cellUtilSpursEventFlag.h>
inline int attachLv2EventQueue (
)
{
        return cellSpursEventFlagAttachLv2EventQueue(this);
}
```

Arguments

None

Return Values

`CELL_OK`(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
`CELL_SPURS_TASK_ERROR_AGAIN`	0x80410901	Currently unable to issue the event queue ID
`CELL_SPURS_TASK_ERROR_NOMEM`	0x80410904	Insufficient memory
`CELL_SPURS_TASK_ERROR_PERM`	0x80410909	Not initialized as SPU2PPU
`CELL_SPURS_TASK_ERROR_BUSY`	0x8041090A	A port number that can be allocated to the Cell OS Lv-2 event queue does not exist
`CELL_SPURS_TASK_ERROR_STAT`	0x8041090F	A Cell OS Lv-2 event queue is already attached
`CELL_SPURS_TASK_ERROR_ALIGN`	0x80410910	`this` is not aligned to a 128-byte boundary
`CELL_SPURS_TASK_ERROR_NULL_POINTER`	0x80410911	`this` is a null pointer

Description

This method creates the Cell OS Lv-2 event queue necessary for the PPU thread to carry out a blocking-wait, and attaches it to a SPURS event flag. This method must be called to perform the attach before the PPU thread calls `EventFlag::wait()`.

Notes

The port number of the created event queue is dynamically allocated by the library. The range of port numbers to be allocated has been defined as constants.

Do not call this method from multiple PPU threads for the same event flag at the same time. Otherwise, subsequent operation to the applicable event flag will be undefined.

EventFlag::detachLv2EventQueue

Detach event queue from SPURS event flag

Definition

```
#include <cellUtilSpursEventFlag.h>
inline int detachLv2EventQueue (
)
{
        return cellSpursEventFlagDetachLv2EventQueue(this);
}
```

Arguments

None

Return Values

`CELL_OK`(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
`CELL_SPURS_TASK_ERROR_BUSY`	0x8041090A	A PPU thread in the waiting state exists
`CELL_SPURS_TASK_ERROR_STAT`	0x8041090F	Cell OS Lv-2 event queue is not attached
`CELL_SPURS_TASK_ERROR_ALIGN`	0x80410910	`this` is not aligned to a 128-byte boundary
`CELL_SPURS_TASK_ERROR_NULL_POINTER`	0x80410911	`this` is a null pointer

Description

This method detaches the Cell OS Lv-2 event queue from the SPURS event flag and deletes it.

JobChain Class

JobChain Class

SPURS job chain class

Definition

```
#include <cellUtilSpursJobChain.h>
namespace cell {
        namespace Util {
              namespace Spurs {
                    class JobChain : public CellSpursJobChain {
                    };
              }
        }
}
```

Conditions

This instance must be in an SPU-accessible area.

This instance must be kept until successful completion of `JobChain::join()`

Conditions of Inheritance

Constructors must be empty.

Destructors must not be defined.

Copy constructor and copy operator must be declared as private to invalidate them.

Do not declare virtual functions.

Static Members

`ALIGN`	Alignment of this class

Static Methods

`create`	Initialize and register to a SPURS instance

Methods

`run`	Start execution of SPURS job chain
`shutdown`	Shut down SPURS job chain.
`join`	Wait for completion of shutdown and remove SPURS job chain
`getError`	Get cause of error for SPURS chain
`getId`	Get workload ID of SPURS job chain
`getSpursAddress`	Get SPURS instance
`getInfo`	Get information of job chain

Description

This is an abstracted class of the SPURS job chain. Initialize the instance with `JobChain::create()` to start its use. `JobChain::join()` must be called to make sure that the shutdown requested by `JobChain::shutdown()` or `cellSpursShutdownJobChain()` has been completed.

A pointer to this class can be used instead of a pointer to `CellSpursJobChain` in an argument of a libspurs API.

Examples

Error handling has been omitted for simplification.

```
#include <cstdlib>
#include <spurs_util.h>
using namespace cell::Util::Spurs;

JobChain* jobChain
        =(JobChain*)std::memalign(JobChain::ALIGN,sizeof(JobChain));
JobChain::create(jobChain, "example", spurs, command_list, 128);

jobChain->run();
// do another things

jobChain->shutdown();
jobChain->join();
std::free(jobChain);
```

Notes

`JobChain::create()`, `JobChain::shutdown()`, and `cellSpursShutdownJobChain()` request all SPUs to run system service, and workload switch occurs on all SPUs. Therefore proper lifetime of SPURS task set is longer than a frame enough to avoid this overhead. For example, the CommandListDispatcher class provides synchronization in command list units.

See Also

CommandListDispatcher Class

`CellSpursJobChain` of "libspurs Job-streaming Reference"

JobChain::create

Create a SPURS job chain

Definition

```
#include <cellUtilSpursJobChain.h>

int create (
        JobChain* jobChain,
        const char* name,
        Spurs* spurs,
        const uint64_t* commandList,
        uint8_t priority = CELL_SPURS_MAX_PRIORITY - 1,
        uint8_t spuMask = 0xff
);

int create (
        JobChain* jobChain,
        const char* name,
        Spurs* spurs,
        const Job::Command* commandList,
        uint16_t sizeJobDescriptor,
        uint8_t priority = CELL_SPURS_MAX_PRIORITY - 1,
        uint8_t spuMask = 0xff
);

int create (
        JobChain* jobChain,
        Spurs* spurs,
        CellSpursJobChainAttribute* attr
);
```

Arguments

jobChain	Pointer to job chain instance
name	Name
spurs	Pointer to initialized SPURS instance
commandList	Address of command list (effective address: 8-byte aligned)
sizeJobDescriptor	Size of job descriptor (a multiple of 128, or 64)
priority	Priority of a SPURS task set per SPU
spuMask	Mask of SPUs to apply priority (LSB=SPU0)
attr	Job chain attribute

Return Values

CELL_OK(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
CELL_SPURS_JOB_ERROR_AGAIN	0x80410a01	Number of workloads reached the maximum (16)
CELL_SPURS_JOB_ERROR_INVAL	0x80410a02	Value of *sizeJobDescriptor* is invalid
CELL_SPURS_JOB_ERROR_INVAL	0x80410a02	Specified a value larger than 16, or 0 for *maxGrabbedJob*
CELL_SPURS_JOB_ERROR_INVAL	0x80410a02	*autoRequestSpuCount*=false and *initialRequestSpuCount* = 0 or greater then 255

Macro	Value	Description
CELL_SPURS_JOB_ERROR_INVAL	0x80410a02	Either *tag1* or *tag2* is greater than 31
CELL_SPURS_JOB_ERROR_INVAL	0x80410a02	*maxSizeJobDescriptor* is less than 256, greater than 1024, or not a multiple of 128
CELL_SPURS_JOB_ERROR_INVAL	0x80410a02	Job chain attributes specified by *attr* are inconsistent (This usually occurs when job chain attributes are not initialized)
CELL_SPURS_JOB_ERROR_STAT	0x80410a0f	SPURS cannot continue because an SPU exception occurred
CELL_SPURS_JOB_ERROR_ALIGN	0x80410a10	*spurs* or *jobChain* is not aligned to a 128-byte boundary. Or, *commandList* or *attr* is not aligned to an 8-byte boundary.
CELL_SPURS_JOB_ERROR_NULL_POINTER	0x80410a11	*spurs*, *jobChain*, *attr*, *commandList*, or *name* is a null pointer

Description

This method creates a SPURS job chain for a SPURS instance specified with *spurs*, and writes the management information of this newly created SPURS job chain in the memory specified by *jobChain*.

When the first or second method is used, specify start address of command list to *commandList*, and job descriptor size of jobs for CELL_SPURS_JOB_COMMAND_JOB to *sizeJobDescriptor*.

For *priority*, specify the priority of this SPURS job chain. You can specify a value from 1 to 15. An SPU will prioritize execution of a SPURS job chain with the smaller priority value. Common *priority* is a common priority for all SPU. For *spuMask*, specify the bit mask of SPUs that the LSB is SPU0 and a value 1 applies the common priority to the corresponding SPU.

Unspecified parameters are set to default values. Default value for each parameter is as follows.

Parameter	Default Value
maxGrabbedJob	4
maxContention	8
autoReadyCount	true
tag1	0
tag2	1
isFixedMemAlloc	false
maxSizeJobDescriptor	1024
readyCount	8

When the third method is used, specify initialized CellSpursJobChainAttribute instance to *attr*.

Notes

Because SPURS job chain keeps just a pointer to a name strings, the name string pointed to by *name* must exist with the task set.

See Also

cellSpursJobChainAttributeInitialize(), cellSpursJobChainAttributeSetName(), and cellSpursCreateTasksetWithAttribute() of "libspurs Job-streaming Reference"

JobChain::run

Start execution of SPURS job chain

Definition

```
#include <cellUtilSpursJobChain.h>
inline int run ()
{
        return cellSpursRunJobChain(this);
}
```

Arguments

None

Return Values

CELL_OK(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
CELL_SPURS_JOB_ERROR_INVAL	0x80410a02	Job chain does not exist
CELL_SPURS_JOB_ERROR_STAT	0x80410a0f	SPURS cannot continue because an SPU exception occurred
CELL_SPURS_JOB_ERROR_ALIGN	0x80410a10	The instance is not aligned to a 128-byte boundary
CELL_SPURS_JOB_ERROR_NULL_POINTER	0x80410a11	The instance is a null pointer

Description

This method requests starting the execution of the SPURS job chain. Before calling this method, use cellSpursCreateJobChainWithAttribute() to add this job chain to the SPURS instance.

See Also

cellSpursCreateJobChainWithAttribute() of "libspurs Job-streaming Reference"

JobChain::shutdown

Shut down SPURS job chain

Definition

```
#include <cellUtilSpursJobChain.h>
inline int shutdown ()
{
        return cellSpursShutdownJobChain(this);
}
```

Arguments

None

Return Values

`CELL_OK`(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
`CELL_SPURS_JOB_ERROR_INVAL`	0x80410a02	Job chain does not exist
`CELL_SPURS_JOB_ERROR_STAT`	0x80410a0f	SPURS cannot continue because an SPU exception occurred
`CELL_SPURS_JOB_ERROR_ALIGN`	0x80410a10	The instance is not aligned to a 128-byte boundary
`CELL_SPURS_JOB_ERROR_NULL_POINTER`	0x80410a11	The instance is a null pointer

Description

This method requests ending the execution of the job chain.

This method simply sends a request not to load subsequent jobs and returns without waiting for the execution of the job chain to end.

To confirm the end of execution, use `JobChain::join()`.

See Also

`cellSpursShutdownJobChain()` of "libspurs Job-streaming Reference"

JobChain::join

Wait for completion of shutdown and remove SPURS job chain

Definition

```
#include <cellUtilSpursJobChain.h>
inline int join ()
{
        return cellSpursJoinJobChain(this);
}
```

Arguments

None

Return Values

CELL_OK(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
CELL_SPURS_JOB_ERROR_INVAL	0x80410a02	Job chain does not exist
CELL_SPURS_JOB_ERROR_INVAL	0x80410a02	Another thread is already waiting for the job chain to end
CELL_SPURS_JOB_ERROR_PERM	0x80410a09	Invalid job entry function
CELL_SPURS_JOB_ERROR_JOB_DESCRIPTOR	0x80410a0b	Job descriptor reserved area is not initialized with zero
CELL_SPURS_JOB_ERROR_JOB_DESCRIPTOR_SIZE	0x80410a0c	Job descriptor size is invalid
CELL_SPURS_JOB_ERROR_STAT	0x80410a0f	SPURS cannot continue because an SPU exception occurred
CELL_SPURS_JOB_ERROR_ALIGN	0x80410a10	The instance is not aligned to a 128-byte boundary
CELL_SPURS_JOB_ERROR_NULL_POINTER	0x80410a11	The instance is a null pointer
CELL_SPURS_JOB_ERROR_MEMORY_SIZE	0x80410a17	Requested memory is too large
CELL_SPURS_JOB_ERROR_UNKNOWN_COMMAND	0x80410a18	Unknown command was specified
CELL_SPURS_JOB_ERROR_JOBLIST_ALIGNMENT	0x80410a19	Alignment of job list is invalid
CELL_SPURS_JOB_ERROR_JOB_ALIGNMENT	0x80410a1a	Alignment of job descriptor is invalid
CELL_SPURS_JOB_ERROR_CALL_OVERFLOW	0x80410a1b	Nesting of CALL commands exceeded 3 layers
CELL_SPURS_JOB_ERROR_ABORT	0x80410a1c	ABORT command was executed
CELL_SPURS_JOB_ERROR_DMALIST_ELEMENT	0x80410a1d	Invalid DMA list element
CELL_SPURS_JOB_ERROR_NUM_CACHE	0x80410a1e	More than 4 read-only buffers are specified
CELL_SPURS_JOB_ERROR_INVALID_BINARY	0x80410a1f	Job binary effective address is 0, or not aligned to a 16-byte boundary, or job binary size is 0

Description

This method waits until shutdown method is called and the execution of all the SPURS jobs ends, and then deletes the job chain from the SPURS instance.

Notes

In order to shut down SPURS job chain, the system service must run on all SPUs in its SPURS instance. Therefore any SPU program without yielding its SPU can stop a caller of this method because of disturbing a shutdown operation.

See Also

`cellSpursJoinJobChain()` of "libspurs Job-streaming Reference"

JobChain::getError

Get cause of error for a SPURS job chain

Definition

```
#include <cellUtilSpursJobChain.h>
inline int getError (
        void** cause
)
{
        return cellSpursJobChainGetError(this, cause);
}
```

Argument

cause Pointer to variable for storing the effective address of the error cause

Return Values

CELL_OK(0) is returned for success.

If an error occurs during the execution of a job chain, the method provides a return value for that error and sets a pointer to the cause of the error in *cause*.

Description

This method obtains the error that occurred during the execution of the job chain.

For details, refer to the section on cellSpursJobChainGetError() in "libspurs Job-streaming Reference".

JobChain::getId

Get workload ID of SPURS job chain

Definition

```
#include <cellUtilSpursJobChain.h>
inline int getId (
        CellSpursWorkloadId *workloadId
)
{
        return cellSpursGetJobChainId(this, workloadId);
}
```

Arguments

workloadId	Location to store workload ID of SPURS job chain

Return Values

CELL_OK(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
CELL_SPURS_JOB_ERROR_ALIGN	0x80410a10	The instance is not aligned to a 128-byte boundary
CELL_SPURS_JOB_ERROR_NULL_POINTER	0x80410a11	The instance is a null pointer

Description

This method returns the workload ID of this SPURS job chain.

This method returns invalid workload ID after JobChain::join() is called.

See Also

cellSpursGetJobChainId() of "libspurs Job-streaming Reference"

JobChain::getSpursAddress

Get SPURS instance

Definition

```
#include <cellUtilSpursJobChain.h>
inline int getSpursAddress (
        CellSpurs **spurs
)
{
        return cellSpursJobChainGetSpursAddress(this, spurs);
}
```

Arguments

spurs	Location to store a pointer to a SPURS instance

Return Values

CELL_OK(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
CELL_SPURS_JOB_ERROR_INVAL	0x80410a02	Job chain does not exist
CELL_SPURS_JOB_ERROR_ALIGN	0x80410a10	The instance is not aligned to a 128-byte boundary
CELL_SPURS_JOB_ERROR_NULL_POINTER	0x80410a11	The instance or *spurs* is a null pointer

Description

This method returns a pointer to a SPURS instance that this SPURS job chain belongs to.

See Also

cellSpursJobChainGetSpursAddress() of "libspurs Job-streaming Reference"

JobChain::getInfo

Get information of job chain

Definition

```
#include <cellUtilSpursJobChain.h>
inline int getInfo (
        CellSpursJobChainInfo* info
)
{
        return cellSpursGetJobChainInfo(this, info);
}
```

Arguments

info	Location to store information of this job chain

Return Values

CELL_OK(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
CELL_SPURS_JOB_ERROR_INVAL	0x80410a02	The specified job chain does not exist
CELL_SPURS_JOB_ERROR_ALIGN	0x80410a10	The instance is not aligned to a 128-byte boundary
CELL_SPURS_JOB_ERROR_NULL_POINTER	0x80410a11	The instance or *info* is a null pointer

Description

This method returns information of this SPURS job chain.

Notes

This method may not return consistent information of a job chain. Therefore, do not use this method for synchronization.

See Also

CellSpursJobChainInfo and cellSpursGetJobChainInfo() of "libspurs Job-streaming Reference"

CommandListDispatcher Class

CommandListDispatcher class

Definition

```
#include <cellUtilSpursCommandListDispatcher.h>
namespace cell {
        namespace Util {
              namespace Spurs {
                    class CommandListDispatcher : public JobChain {
                    };
              }
        }
}
```

Conditions

This instance must be in an SPU-accessible area.

This instance must be kept until successful completion of `CommandListDispatcher::join()`

Conditions of Inheritance

Constructors must be empty.

Destructors must not be defined.

Copy constructor and copy operator must be declared as private to invalidate them.

Do not declare virtual functions.

Static Members

`ALIGN`	Alignment of this class

Static Methods

`create`	Initialize and register to a SPURS instance

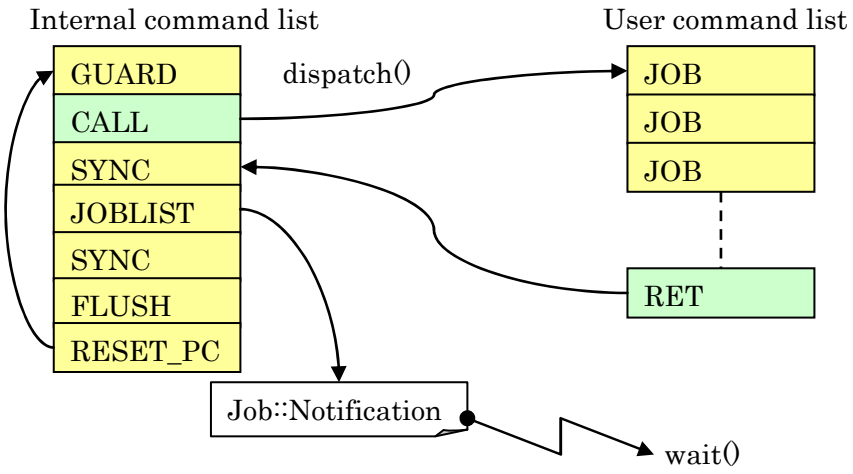
Methods

`dispatch`	Start execution of the command list
`wait`	Check completion of the command list

Description

This class is a derivative of the JobChain class. The command list specified by `CommandListDispatcher::dispatch()` can be repeatedly executed and its completion can be waited for by `CommandListDispatcher::wait()`. Because the command list is called by the `CALL` job command, it must end with the `RET` job command.

Always use `CommandListDispatcher::create()` to initialize and start using this class. Request execution termination with `CommandListDispatcher::shutdown()`, and then check that execution has been terminated using `CommandListDispatcher::join()`.



Example

Error handling has been omitted for simplification.

```
#include <cstdlib>
#include <spurs_util.h>
using namespace cell::Util::Spurs;

CommandListDispatcher* jobChain
=(JobChain*)std::memalign(CommandListDispatcher::ALIGN,sizeof(CommandListDis
patcher));
JobChain::create(jobChain, "example", spurs, 128);

jobChain->dispatch(command_list1);
jobChain->wait();
jobChain->dispatch(command_list2);
jobChain->wait();

jobChain->shutdown();
jobChain->join();
std::free(jobChain);
```

CommandListDispatcher::create

Create CommandListDispatcher

Definition

```
#include <cellUtilSpursCommandListDispatcher.h>

int create (
        JobChain* jobChain,
        const char* name,
        Spurs* spurs,
        uint16_t sizeJobDescriptor,
        uint8_t priority = CELL_SPURS_MAX_PRIORITY - 1,
        uint8_t spuMask = 0xff
);
```

Arguments

jobChain	Pointer to the CommandListDispatcher instance
name	Name
spurs	Pointer to the initialized SPURS instance
sizeJobDescriptor	Size of the job descriptor (a multiple of 128, or 64)
priority	Workload priority
spuMask	SPU mask for applying the workload priority (LSB=SPU0)

Return Value

CELL_OK(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
CELL_SPURS_JOB_ERROR_AGAIN	0x80410a01	Number of workloads reached the maximum (16)
CELL_SPURS_JOB_ERROR_INVAL	0x80410a02	Value of *sizeJobDescriptor* is invalid
CELL_SPURS_JOB_ERROR_STAT	0x80410a0f	SPURS cannot continue because an SPU exception occurred
CELL_SPURS_JOB_ERROR_ALIGN	0x80410a10	*spurs* or *jobChain* is not aligned to a 128-byte boundary
CELL_SPURS_JOB_ERROR_NULL_POINTER	0x80410a11	*spurs*, *jobChain*, or *name* is a null pointer

Description

This method creates a CommandListDispatcher for a SPURS instance specified by *spurs*, and writes its management information to the memory specified by *jobChain*.

For *sizeJobDescriptor*, specify the size of the job descriptor to use in CELL_SPURS_JOB_COMMAND_JOB.

For *priority*, specify the priority of the SPURS job chain. The range of values that can be specified for the priority is 1 to 15, where 1 is the highest priority. For *spuMask*, specify an SPU mask for applying a common *priority* for all SPUs. *spuMask* is a bit sequence where LSB corresponds to SPU0, and the priority will be applied if the value is 1.

For other parameters of the job chain, set values as follows.

Parameter	Default value
maxGrabbedJob	4
maxContention	8

Parameter	Default value
autoReadyCount	true
tag1	0
tag2	1
isFixedMemAlloc	false
maxSizeJobDescriptor	1024
readyCount	8

Notes

Because SPURS holds pointers to strings, the string specified by *name* must be kept while the job chain remains in existence.

See Also

JobChain Class

CommandListDispatcher::dispatch

Start execution of the command list

Definition

```
#include <cellUtilSpursCommandListDispatcher.h>
inline int dispatch (
        const Job::Command* command
)
```

Arguments

command Pointer to the target command list

Return Values

CELL_OK(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
CELL_SPURS_JOB_ERROR_INVAL	0x80410a02	Job chain does not exist
CELL_SPURS_JOB_ERROR_BUSY	0x80410a0a	CommandListDispatcher::wait() has not been executed on the previously started command list
CELL_SPURS_JOB_ERROR_STAT	0x80410a0f	SPURS cannot continue because an SPU exception occurred
CELL_SPURS_JOB_ERROR_ALIGN	0x80410a10	The instance is not aligned to a 128-byte boundary
CELL_SPURS_JOB_ERROR_NULL_POINTER	0x80410a11	The instance or *command* is a null pointer

Description

This method requests the start of a command list execution.

For *command*, specify a command list that ends with the RET job command. Because the CALL job command is used to call this command list, the CALL job command can be nested 2 more times within the command list.

Call this method in partner with CommandListDispatcher::wait() to call this method again without having to terminate the SPURS job chain.

CommandListDispatcher::wait

Wait for the completion of the command list

Definition

```
#include <cellUtilSpursCommandListDispatcher.h>
inline int wait (
)
```

Arguments

None

Return Values

CELL_OK(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
CELL_SPURS_TASK_ERROR_AGAIN	0x80410901	Currently unable to issue the event queue ID
CELL_SPURS_TASK_ERROR_NOMEM	0x80410904	Insufficient memory
CELL_SPURS_TASK_ERROR_BUSY	0x8041090A	A port number that can be allocated to the Cell OS Lv-2 event queue does not exist, or a PPU thread in the waiting state exists
CELL_SPURS_TASK_ERROR_STAT	0x8041090F	Failure to attach or detach a Cell OS Lv-2 event queue
CELL_SPURS_JOB_ERROR_STAT	0x80410a0f	The execution of the command list has not been started with CommandListDispatcher::dispatch()

Description

This method waits for the execution of the command list started by CommandListDispatcher::dispatch() to end.

Notes

If the execution of the command list is still going on when this method is called, the method internally creates a Cell OS Lv-2 event queue and blocks the PPU thread. Because of this, the error code includes the internally called cellSpursEventFlagWait(), cellSpursEventFlagAttachLv2EventQueue(), and cellSpursEventFlagDetachLv2EventQueue().

CommandListDispatcher::shutdown

Shut down SPURS job chain

Definition

```
#include <cellUtilSpursCommandListDispatcher.h>
int shutdown ();
```

Description

See "JobChain::shutdown".

CommandListDispatcher::join

Wait for completion of shutdown and remove SPURS job chain

Definition

```
#include <cellUtilSpursCommandListDispatcher.h>
int join ();
```

Description

See "JobChain::join".

JobGuard Class

JobGuard Class

SPURS job guard class

Definition

```
#include <cellUtilSpursJobGuard.h>
namespace cell {
        namespace Util {
              namespace Spurs {
                     class JobGuard : public CellSpursJobGuard {
                     };
              }
        }
}
```

Conditions

This instance must be in an SPU-accessible area.

Conditions of Inheritance

Constructors must be empty.

Destructors must not be defined.

Copy constructor and copy operator must be declared as private to invalidate them.

Do not declare virtual functions

Static Members

`ALIGN`	Alignment of this class

Static Methods

`initialize`	Initialize

Methods

`notify`	Notify arrival at GUARD synchronization
`reset`	Reset GUARD synchronization

Description

This is an abstracted class of SPURS job guard. Make sure an instance of this class is initialized with `JobGuard::initialize()` before using.

A pointer to this class can be used as a pointer to `CellSpursJobGuard` for an argument of libspurs API.

Examples

```
#include <cstdlib>
#include <cassert>
#include <spurs_util.h>
using namespace cell::Util::Spurs;

JobGuard *guard
        = (JobGuard*)std::memalign(JobGuard::ALIGN, sizeof(JobGuard));
```

```
ret = JobGuard::initialize(guard, jobChain, 1);
assert(ret == CELL_OK);

ret = guard->notify();
assert(ret == CELL_OK);

std::free(guard);
```

See Also

`CellSpursJobGuard` of "libspurs Job-streaming Reference"

JobGuard::initialize

Create GUARD synchronization

Definition

```
#include <cellUtilSpursJobChain.h>
static int initialize (
        JobGuard* guard,
        const JobChain* jobChain,
        uint32_t notifyCount,
        uint8_t numReadyCount=8,
        uint8_t autoReset=1
)
{
        return cellSpursJobGuardInitialize(jobChain, guard, notifyCount,
                                           numReadyCount, autoReset);
}
```

Arguments

guard	Pointer to work area for GUARD synchronization
jobChain	Pointer to job chain
notifyCount	Initial value of counter
numReadyCount	The request SPU count to set when restarting execution
autoReset	Whether or not to re-initialize the counter

Return Values

CELL_OK(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
CELL_SPURS_JOB_ERROR_INVAL	0x80410a02	The specified job chain is invalid
CELL_SPURS_JOB_ERROR_ALIGN	0x80410a10	*guard* is not aligned to a 128-byte boundary
CELL_SPURS_JOB_ERROR_NULL_POINTER	0x80410a11	*guard* is a null pointer

Description

This method initializes SPURS job guard.

See Also

cellSpursJobGuardInitialize() of "libspurs Job-streaming Reference"

JobGuard::notify

Notify arrival at GUARD synchronization

Definition

```
#include <cellUtilSpursJobChain.h>
int notify ()
{
        return cellSpursJobGuardNotify();
}
```

Arguments

None

Return Values

CELL_OK(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
CELL_SPURS_JOB_ERROR_STAT	0x80410a0f	Counter is already 0
CELL_SPURS_JOB_ERROR_ALIGN	0x80410a10	The instance is not aligned to a 128-byte boundary
CELL_SPURS_JOB_ERROR_NULL_POINTER	0x80410a11	The instance is a null pointer

Description

This method decrements the counter of the GUARD synchronization.

When the value of the counter reaches 0, the guard will be released, and the execution of the stopped job chain at this guard will be restarted.

See Also

cellSpursJobGuardNotify() of "libspurs Job-streaming Reference"

JobGuard::reset

Reset GUARD synchronization

Definition

```
#include <cellUtilSpursJobChain.h>
int reset ()
{
        return cellSpursJobGuardReset();
}
```

Arguments

None

Return Values

CELL_OK(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
CELL_SPURS_JOB_ERROR_ALIGN	0x80410a10	The instance is not aligned to a 128-byte boundary
CELL_SPURS_JOB_ERROR_NULL_POINTER	0x80410a11	The instance is a null pointer

Description

This method resets the counter of the GUARD synchronization to the value specified at initialization.

See Also

cellSpursJobGuardReset() of "libspurs Job-streaming Reference"

Job::Command Class

Job::Command Class

Base class of job streaming commands

Definition

```
#include <cellUtilSpursJobCommand.h>
namespace cell {
        namespace Util {
              namespace Spurs {
                    namespace Job {
                          class Command {
                          public:
                                Command(uint64_t command);
                    }
              }
        }
}
```

Conditions

This instance must be in an SPU-accessible area.

Description

This is the base class for job streaming commands. It is equivalent to uint64_t.

Derivative Classes

Derivative classes have the macros of the job streaming commands replaced. For individual commands, refer to "libspurs Job-streaming Reference".

Job::Nop Class

```
Nop()
  : Command(CELL_SPURS_JOB_COMMAND_NOP) {}
```

Job::Job Class

```
Job(Descriptor& jd)
  : Command(CELL_SPURS_JOB_COMMAND_JOB(&jd)) {}
Job(Descriptor* jd)
  : Command(Cell_SPURS_JOB_COMMAND_JOB(jd)) {}
```

Job::List Class

```
JobList(CellSpursJobList& jl)
  : Command(CELL_SPURS_JOB_COMMAND_JOBLIST(&jl)) {}
JobList(CellSpursJobList* jd)
  : Command(Cell_SPURS_JOB_COMMAND_JOBLIST(jl)) {}
```

Job::Guard Class

```
Guard(JobGuard& guard)
  : Command(CELL_SPURS_JOB_COMMAND_GUARD(&guard)) {}
Guard(JobGuard* guard)
  : Command(Cell_SPURS_JOB_COMMAND_GUARD(guard)) {}
```

Job::Sync Class

```
Sync()
  : Command(CELL_SPURS_JOB_COMMAND_SYNC) {}
```

Job::LWSync Class

```
LWSync()
  : Command(CELL_SPURS_JOB_COMMAND_LWSYNC) {}
```

Job::SetLabel Class

```
SetLabel(unsigned label = 0)
  : Command(CELL_SPURS_JOB_COMMAND_LABEL(label)) {}
```

Job::SyncLabel Class

```
SyncLabel(unsigned label = 0)
  : Command(CELL_SPURS_JOB_COMMAND_SYNC_LABEL(label)) {}
```

Job::LWSyncLabel Class

```
LWSyncLabel(unsigned label = 0)
  : Command(CELL_SPURS_JOB_COMMAND_LWSYNC_LABEL(label)) {}
```

Job::ResetPC Class

```
ResetPC(const uint64_t* command)
  : Command(CELL_SPURS_JOB_COMMAND_RESET_PC(command)) {}
ResetPC(const Command* command)
  : Command(Cell_SPURS_JOB_COMMAND_RESET_PC(command)) {}
```

Job::Next Class

```
Next(const uint64_t* command)
  : Command(CELL_SPURS_JOB_COMMAND_NEXT(command)) {}
Next(const Command* command)
  : Command(Cell_SPURS_JOB_COMMAND_NEXT(command)) {}
```

Job::Call Class

```
Call(const uint64_t* command)
  : Command(CELL_SPURS_JOB_COMMAND_CALL(command)) {}
Call(const Command* command)
  : Command(Cell_SPURS_JOB_COMMAND_CALL(command)) {}
```

Job::Ret Class

```
Ret()
  : Command(CELL_SPURS_JOB_COMMAND_RET) {}
```

Job::Abort Class

```
Abort()
  : Command(CELL_SPURS_JOB_COMMAND_ABORT) {}
```

Job::End Class

```
End()
  : Command(CELL_SPURS_JOB_COMMAND_END) {}
```

Job::Flush Class

```
Flush()
  : Command(CELL_SPURS_JOB_COMMAND_FLUSH) {}
```

Job::Descriptor Class

Job::Descriptor Class

SPURS job descriptor class

Definition

```
#include <cellUtilSpursJobDescriptor.h>
namespace cell {
        namespace Util {
              namespace Spurs {
                     namespace Job {
                            class Descriptor {
                                   protected:
                                          CellSpursJobHeader mHeader;
                            };
                     };
              }
        }
}
```

Conditions

This instance must be in an SPU-accessible area.

Methods

`setBinary`	Set job binary

Description

This base class is an abstraction of the SPURS job descriptor. Inherit this class and define a job descriptor of the appropriate size.

Job::Descriptor::setBinary

Set job binary

Definition

```
#include <cellUtilSpursJobDescriptor.h>
void setBinary (
        const void* address,
        unsigned size
);

void setBinary (
        const void* address,
        const void* size
);
```

Arguments

address	Pointer to the job binary
size	Size of the job binary

Return Values

None

Description

This method sets a job binary to the job descriptor.

See Also

`CellSpursJobHeader` of "libspurs Job-streaming Reference"

Job::Notification Class

Notification job class

Definition

```
#include <cellUtilSpursJobNotification.h>
namespace cell {
        namespace Util {
              namespace Spurs {
                    namespace Job {
                          class Notification : public Descriptor {
                          };
                    };
              }
        }
}
```

Conditions

This instance must be in an SPU-accessible area.

Static Methods

`initialize`	Initialize

Methods

`wait`	Wait for notification
`tryWait`	Wait for notification

Description

This is a job class for making notifications. It requires 1 SPURS event flag. Make sure to initialize the instance by `Job::Notification::initialize()` before use.

Example

Refer to the following source codes.

`cellUtilSpursCommandListDispatcher.h`

`cellUtilSpursCommandListDispatcher.cc`

Job::Notification::initialize

Initialize the notification job

Definition

```
#include <cellUtilSpursJobNotification.h>
static int initialize (
        Notification* self,
        EventFlag* eventflag
);
```

Arguments

self	Notification job instance
eventflag	SPURS event flag to use for notification

Return Values

CELL_OK(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
CELL_SPURS_JOB_ERROR_NULL_POINTER	0x80410a11	The instance is a null pointer

Description

This method initializes the notification job. Make sure to specify an initialized SPURS event flag to *eventflag*.

Job::Notification::wait

Wait for notification

Definition

```
#include <cellUtilSpursJobNotification.h>
int wait (
);
```

Arguments

None

Return Values

CELL_OK(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
CELL_SPURS_JOB_ERROR_NULL_POINTER	0x80410a11	The instance is a null pointer

Description

This method waits for a notification job of the instance to be executed.

Job::Notification::tryWait

Wait for notification

Definition

```
#include <cellUtilSpursJobNotification.h>
int tryWait (
);
```

Arguments

None

Return Values

CELL_OK(0) is returned for success.

One of the following error codes (a negative value) is returned for an error.

Macro	Value	Description
CELL_SPURS_TASK_ERROR_BUSY	0x8041090A	A notification is not received yet
CELL_SPURS_JOB_ERROR_NULL_POINTER	0x80410a11	The instance is a null pointer

Description

This method waits for a notification job of the instance to be executed.