IBM Software Development Kit for Multicore Acceleration
Version 3.0

SPE Runtime Management Library Version 1 to Version 2 Migration Guide

SC33-8332-01

IBM Software Development Kit for Multicore Acceleration
Version 3.0

SPE Runtime Management Library Version 1 to Version 2 Migration Guide

SC33-8332-01

Note
Before using this information and the product it supports, read the information in "Notices" on page 51.

Edition notice

This edition applies to version 3, release 0, modification 0 of the IBM Software Development Kit for Multicore Acceleration (product number 5724-S84) and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition replaces SC33-8332-00.

Contents

About this publication

This document describes how to migrate code designed for the SPE Runtime Management Library (LIBSPE) version 1 to use version 2.

For information about the accessibility features of this product, for users who have a physical disability, see "Accessibility features," on page 49.

Who should use this book

This book is intended for use by software developers.

Related information

For a full list of documentation available for the SDK, see "Related documentation" on page 55.

How to send your comments

Your feedback is important in helping to provide the most accurate and highest quality information. If you have any comments about this publication, send your comments using Resource Link™ at http://www.ibm.com/servers/resourcelink. Click **Feedback** on the navigation pane. Be sure to include the name of the book, the form number of the book, and the specific location of the text you are commenting on (for example, a page number or table number).

Chapter 1. Introduction

This topic introduces the process of migrating code from the SPE Runtime Management Library (LIBSPE) version 1 to version 2.

The IBM® Software Development Kit for Multicore Acceleration (SDK) version 3.0 includes version 2.2 of LIBSPE, referred to as LIBSPE2. LIBSPE version 1, referred to as LIBSPE1, is deprecated. If your code depends on features available only in LIBSPE1, you will be affected when LIBSPE1 is eliminated. Therefore, you are encouraged to migrate your code to use LIBSPE2. We recommend that new code be written to use only LIBSPE2. When you migrate your code to the new library, keep in mind the following points:

- All code in the SDK uses LIBSPE2.
- Migration of the SDK code was done by re-coding, not by using a wrapper.
- An application cannot use the LIBSPE1 API and the LIBSPE2 API concurrently.
- Migration does not affect SPU code.
- Nearly all PPU code using LIBSPE1 is affected.

There are significant changes in LIBSPE2 with respect to LIBSPE1:

- Primarily in context and thread management.
- Secondarily in the removal of group capability.
- Finally in the renaming of all functions and some typedefs, and the creation of new typedefs.

The functions in this book appear in the same order as in the "SPE Runtime Management Library Version 1.2" document. The text explains for each group of functions whether a construct is replaced or removed in the new version, and shows how to migrate from the old to the new version if available. More code is required to perform an equivalent task in LIBSPE2, but it provides additional capabilities over LIBSPE1.

Why has LIBSPE changed?

LIBSPE is designed to be used as the low-level API to access SPE resources. The *SPE context* introduced in LIBSPE2 is a better low-level construct than the *SPE thread* construct defined in LIBSPE1, which suggests a particular programming model and view. This SPE thread model can be implemented using SPE contexts and the standard pthread library, if desired. By using SPE contexts, other programming models such as synchronous functions can more easily offload to SPEs without introducing the complexity and overhead that threading would include. LIBSPE2 has the ability to exchange code on an SPE but leave the data in place, thereby allowing for easy and efficient chaining of processing steps and PPE control. If you use the thread model, it relies on SPE programs using overlays. It is very easy to implement the LIBSPE1 thread model as a special case on top of LIBSPE2. IBM has successfully done this exercise internally.

Many people asked for a more complete SPE thread library similar to pthreads. This request has been satisfied by removing the special concept of an SPE thread as used in LIBSPE1. The programmer using LIBSPE2 relies on a thread package of choice, and simply uses SPEs in these threads. All aspects of an application specific to threads are standardized so you have full thread functionality available to you.

LIBSPE2 resolves many complaints about the event API in LIBSPE1, from usability to efficiency.

SPE groups in LIBSPE1 tied together orthogonal concepts such as scheduling and event handling. Therefore, this construct was discarded in the new library. LIBSPE2 introduces *SPE gang contexts* which will be leveraged by *gang scheduling*. Note that *gangs* are purely a scheduling construct and do not replace LIBSPE1 groups. LIBSPE2 introduces a new event mechanism that is based on SPE contexts and is not tied to scheduling in any way.

The proposed LIBSPE1 API to bind SPE threads to physical SPE resources was heavily debated and therefore never implemented. To provide an equivalent feature, LIBSPE2 introduces a new concept of *logical affinity* for SPE contexts. Using logical affinity, a programmer can request that two SPE contexts be placed on adjacent physical SPE resources. Affinity ensures low latency and high communication bandwidth between programs running on adjacent SPEs. The affinity API does not allow the programmer to directly select physical SPE resources, which are subject to change in new revisions of hardware. The operating system encapsulates the physical SPE topology, and uses this information to select adjacent processors. Therefore, an application can request logical conditions on relative context placement without the application having to manage physical details of Cell/B.E.™ topology information. *SPE affinity* was tied to the concept of SPE gangs, because placement constraints to improve communication efficiency only make sense if it can be assumed that the SPEs run concurrently.

Conventions

This document contains many examples that demonstrate how to migrate your code from LIBSPE1 to LIBSPE2. In order to use these examples, you must understand the following conventions:

Document Text	Meaning
LIBSPE1 PPU Example	The source code that follows is used with version 1 of the LIBSPE library.
LIBSPE2 PPU Example	The source code that follows is used with version 2 of the LIBSPE library.
...	The provided source code example is not a complete compilable program. The ellipsis (...) indicates where you can insert supporting code to complete the program.
<text>	For lines of example code other than those that begin with `#include`, you must choose the code to replace the text between opening (<) and closing (>) brackets. Identical names are used where possible in both LIBSPE1 and LIBSPE2 examples for continuity.

Here is a short example that illustrates the migration process:

LIBSPE1 PPU Example

```
int <name>;
```

LIBSPE2 PPU Example

```
long <name>;
```

Therefore, you would change your code from:

```
int abc;
```

to

```
long abc;
```

Chapter 2. SPE Thread Management Facilities

This section shows how to migrate the SPE thread management facilities.

Function: spe_count_physical_spes

The spe_count_physical_spes function has been replaced in LIBSPE2.

Introduction

The int spe_count_physical_spes() function is replaced in LIBSPE2 with spe_cpu_info_get, with specific arguments to request the count.

LIBSPE1 PPU Example

```
#include <libspe.h>
...
int <count>;
...
<count> = spe_count_physical_spes();
```

LIBSPE2 PPU Example

```
#include <libspe2.h>
...
int <count>;
...
<count> = spe_cpu_info_get(SPE_COUNT_PHYSICAL_SPES, -1);
```

Function: spe_create_group

The spe_create_group function is eliminated from LIBSPE2.

Introduction

The spe_gid_t spe_create_group(int **policy**, int **priority**, int **spe_event**) function has been eliminated. There is no replacement for groups in LIBSPE2. The setting of **policy** and **priority** parameters is done using pthread functions, and the **spe_event** parameter is set using the spe_context_create function.

- The **policy** parameter with values SCHED_RR, SCHED_FIFO, and SCHED_OTHER is set using the pthread_attr_setschedpolicy function and a previously initialized thread attribute object.
- The **priority** parameter is set using the pthread_attr_setschedparam function and a thread attribute object.
- The **spe_event** parameter is set when invoking the function spe_context_create and by providing the SPE_EVENTS_ENABLE value for the **flags** parameter when **spe_event** is non-zero.

LIBSPE1 PPU Example

```
#include <libspe.h>
...
spe_gid_t <group>;
int <policy>;
int <priority>;
int <spe_event>;
...
<group> = spe_create_group(<policy>, <priority>, <spe_event>);
```

LIBSPE2 PPU Example

```
#include <libspe2.h>
#include <pthread.h>
...
int <policy>;
int <priority>;
int <spe_event>;
pthread_attr_t attr;
struct sched_param param;
spe_context_ptr_t <speid>;
...
pthread_attr_init(&attr);
pthread_attr_setschedpolicy(&attr, <policy>);
param.sched_priority = <priority>;
pthread_attr_setschedparam(&attr, &param);
...
<speid>=spe_context_create(<spe_event> != 0 ? SPE_EVENTS_ENABLE : 0, NULL);
```

Function: spe_create_thread

This spe_create_thread function is eliminated from LIBSPE2.

Introduction

The speid_t spe_create_thread(spe_gid_t **gid,** spe_program_handle_t ***spe_program,** void ***argp,** void ***envp,** unsigned long **mask,** int **flags**) function is eliminated in LIBSPE2. This function is replaced by a combination of spe_context_create, spe_program_load, pthread_create, and spe_context_run functions.

The following is a list of changes in LIBSPE2 that will help you understand how to create threads.

- The **gid** parameter is eliminated in LIBSPE2. There is no replacement for groups in LIBSPE2.
- The **spe_program** parameter is provided to the spe_program_load function.
- The **argp** parameter is passed to the spe_context_run function either directly or indirectly using an intermediate data structure.
- The **envp** parameter is passed to the spe_context_run function either directly or indirectly using an intermediate data structure.
- The **mask** parameter is eliminated in LIBSPE2. There is no replacement in LIBSPE2.
- The **flags** parameter with values of SPE_CFG_SIGNOTFY1_OR, SPE_CFG_SIGNOTFY2_OR, and SPE_MAP_PS are passed to the spe_context_create function. The flag with a value of SPE_USER_REGS is passed to the spe_context_run function.

The speid_t typedef is replaced by the combination of spe_context_ptr_t and pthread_t typedefs.

LIBSPE1 PPU Example

```
#include <libspe.h>
...
spe_gid_t <group>;
spe_program_handle_t <spe_program>;
void *<argp>;
void *<envp>;
unsigned long <mask>;
int <flags>;
speid_t <speid>;
...
<speid> = spe_create_thread(<group>, &<spe_program>, <argp>, <envp>,
                            <mask>, <flags>);
```

LIBSPE2 PPU Example

```
#include <libspe2.h>
#include <pthread.h>
...
typedef struct ppu_pthread_data {
   spe_context_ptr_t <speid>;
   pthread_t pthread;
   unsigned int entry;
   unsigned int <flags>;
   void *<argp>;
   void *<envp>;
   spe_stop_info_t stopinfo;
} ppu_pthread_data_t;
```

```
...
spe_program_handle_t <spe_program>;
void *<argp>;
void *<envp>;
int <flags>;
pthread_attr_t attr;
ppu_pthread_data_t ppdata;
...
void *ppu_pthread_function(void *arg) {
   ppu_pthread_data_t *datap = (ppu_pthread_data_t *)arg;
   int rc;
   do {
      rc = spe_context_run(datap-><speid>, &datap->entry, datap-><flags>,
                           datap-><argp>, datap-><envp>, &datap->stopinfo);
   } while (rc > 0); /* until exit or error, while stop & signal */
   pthread_exit(NULL);
}
...
ppdata.<speid> = spe_context_create(<flags>, NULL);
...
spe_program_load(ppdata.<speid>, &<spe_program>);
...
ppdata.entry = SPE_DEFAULT_ENTRY;
ppdata.flags = <flags>;
ppdata.argp = <argp>;
ppdata.envp = <envp>;
pthread_create(&ppdata.pthread, &attr, &ppu_pthread_function, &ppdata);
```

Function: spe_destroy_group

The `spe_destroy_group` function is eliminated from LIBSPE2.

Introduction

The `int spe_destroy_group(spe_gid_t gid)` function is eliminated in LIBSPE2. There is no replacement in LIBSPE2.

LIBSPE1 PPU Example

```
#include <libspe.h>
...
spe_gid_t <group>;
...
spe_destroy_group(<group>);
```

LIBSPE2 PPU Example

No replacement is possible.

Function: spe_get_affinity, spe_set_affinity

The spe_get_affinity and spe_set_affinity functions are eliminated from LIBSPE2.

Introduction

The int spe_get_affinity(speid_t **speid,** unsigned long ***mask**), and int spe_set_affinity(speid_t **speid,** unsigned long **mask**) functions have been eliminated. They are replaced by the spe_context_ptr_t spe_context_create_affinity(unsigned int **flags,** spe_context_ptr_t **affinity_neighbor,** spe_gang_context_ptr_t **gang**) function in LIBSPE2.

Program sequence for SPE-to-SPE affinity

From an application perspective, SPE-to-SPE affinity is specified in a three part sequence:

1. Create SPE Gang *X*.
2. Create *N* SPE Contexts with affinity in SPE Gang *X*.
3. Start *N* pthreads that run the *N* SPE contexts created in step 2.

Creating an SPE context with affinity

SPE-to-SPE affinity is specified in affinity pairs. The spe_context_create_affinity function allows an SPE context to be created and placed next to another previously created SPE context. The SPU file system (SPUFS) scheduler honors this relationship by scheduling the specified SPE contexts on physically adjacent SPUs. This function can be used to create a chain of SPE contexts that consumes all of the available SPE resources on a Cell/B.E., but not more. If you want to use additional SPE resources, you must create a separate gang or individual SPE contexts for that purpose. All SPE contexts in the gang must be created before you run any SPE contexts in the gang.

The LIBSPE2 *create with affinity* interface is the spe_context_ptr_t spe_context_create_affinity(unsigned int **flags,** spe_context_ptr_t **affinity_neighbor,** spe_gang_context_ptr_t **gang**) function. The **flags** parameter has the same semantics as it does when used with the spe_context_create function. The SPE_AFFINITY_MEMORY flag is available to specify SPE-to-memory affinity. If the flag is set, the newly created SPE context will be run on an SPU that is determined to be the closest to main memory storage. Only one SPE context in the group can be created with memory affinity. The **affinity_neighbor** parameter identifies a previously created SPE context in the named gang. A NULL value can be specified for the initial SPE context. Alternately, use the spe_context_create function to create the initial SPE context. The **gang** parameter identifies the previously created gang that the context will create. The **affinity_neighbor** parameter must be in the same gang.

For complete details of the spe_context_create_affinity function, see the SPE Runtime Management Library Version 2.1 Reference.

LIBSPE1 PPU Example

```
#include <libspe.h>
...
speid_t <speid>;
unsigned long <mask>;
```

```
...
spe_get_affinity(<speid>, &<mask>);
...
spe_set_affinity(<speid>, <mask>);
```

LIBSPE2 PPU Example

The following is a mostly complete LIBSPE2 program that creates a context with affinity:

```
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include "libspe2.h"

#define MAX_SPES_IN_BE    8

struct thread_args {
   struct spe_context *ctx;
   void *argp;
   void *envp;
};

void *spe_thread(void *arg);

__attribute__((noreturn)) void *spe_thread(void *arg) {
   int flags = 0;
   unsigned int entry = SPE_DEFAULT_ENTRY;
   int rc;
   spe_program_handle_t *program;
   struct thread_args *arg_ptr;

   arg_ptr = (struct thread_args *)arg;

   program = spe_image_open("hello");
   if (!program) {
      perror("spe_image_open");
      pthread_exit(NULL);
   }

   if (spe_program_load(arg_ptr->ctx, program)) {
      perror("spe_program_load");
      pthread_exit(NULL);
   }

   rc = spe_context_run(arg_ptr->ctx, &entry, flags, arg_ptr->argp,
                        arg_ptr->envp, NULL);
   if (rc < 0)
      perror("spe_context_run");

   pthread_exit(NULL);
}

int main() {
   int th_id;
   pthread_t pts[MAX_SPES_IN_BE];
   spe_context_ptr_t ctx[MAX_SPES_IN_BE], neighbor;
   struct thread_args t_args[MAX_SPES_IN_BE];
   spe_gang_context_ptr_t gang;
   int value = 1;
   int flags;
   int i;

   if ((gang = spe_gang_context_create(0)) == NULL) {
      perror("spe_gang_context_create");
      return -1;
```

```
    }

    /* First, create all of the contexts. */
    for (i = 0; i < MAX_SPES_IN_BE; i++) {
       if (i == 0) {
          /* Place the initial context near main storage. */
          flags = SPE_AFFINITY_MEMORY;
          neighbor = NULL;
       }
       else {
          /* Place the rest of them in order. */
          flags = 0;
          neighbor = ctx[i-1];
       }
       ctx[i] = spe_context_create_affinity(flags, neighbor, gang);
       if (ctx[i] == NULL) {
          perror("spe_context_create_affinity");
          return -2;
       }
       t_args[i].ctx = ctx[i];
       t_args[i].argp = &value;
    }

    /* Next, start them. */
    for (i = 0; i < MAX_SPES_IN_BE; i++) {
       th_id = pthread_create(&pts[i], NULL, &spe_thread, &t_args[i]);
    }

    /* Do stuff, process SPU events, and so on. */
    ...
    /* Wait for ctxs to terminate */
    for (i = 0; i < MAX_SPES_IN_BE; i++) {
       pthread_join(pts[i], NULL);
       spe_context_destroy(ctx[i]);
}

    spe_gang_context_destroy(gang);

    return 0;
}
```

Function: spe_get_context, spe_set_context

The spe_get_context and spe_set_context functions are eliminated from LIBSPE2.

Introduction

The int spe_get_context(speid_t **speid,** struct spe_ucontext ***uc**), and int spe_set_context(speid_t **speid,** struct ucontext ***uc**) functions have been eliminated. There are no replacements for these functions in LIBSPE2.

LIBSPE1 PPU Example

```
#include <libspe.h>
...
speid_t <speid>;
struct spe_ucontext <uc>;
...
spe_get_context(<speid>, &<uc>);
...
spe_set_context(<speid>, &<uc>);
```

LIBSPE2 PPU Example

No replacement is possible.

Function: spe_get_event

The spe_get_event function is replaced by a combination of other functions in LIBSPE2.

Introduction

The int spe_get_event(struct spe_event ***pevents**, int **nevents**, int **timeout**) function is replaced by a combination of functions detailed in the following table:

LIBSPE1	LIBSPE2
spe_get_event function	Replaced by a combination of:
	spe_event_handler_create function
	spe_event_handler_register function
	spe_event_wait function
	spe_event_handler_deregister function
	spe_event_handler_destroy function

The following list describes other details of migrating the spe_get_event function.

- The **pevents** parameter is replaced by the **spe_event_unit_t** parameter both as input when registering with the spe_event_handler_register function and as output after waiting with the spe_event_wait function.
  - The **pevents.gid** parameter is replaced by the **pevents.spe** parameter along with changing the type from spe_gid_t to spe_context_ptr_t.
  - The **pevents.events** parameter is replaced by the **pevents.events** parameter along with changing the bit-mask values.
  - The **pevents.revents** parameter is replaced by the **pevents.events** parameter along with changing the bit-mask values.
  - The **pevents.speid** parameter is replaced by the **pevents.spe** parameter.
  - The **pevents.data** parameter is replaced by the **stopinfo.stop_reason** parameter set by the spe_stop_info_read function.
- The **nevents** parameter is replaced by the **max_events** parameter in the spe_event_wait function.
- The **timeout** parameter is unchanged in the spe_event_wait function.

LIBSPE1 PPU Example

```
#include <libspe.h>
...
spe_gid_t <group>;
#define NUM_EVENTS <#>
struct spe_event <pevents>[NUM_EVENTS];
int <nevents> = NUM_EVENTS;
int <mask>;
int <timeout>;
int i;
...
for (i=0; i<NUM_EVENTS; i++) {
  <pevents>[i].gid = <group>;
  <pevents>[i].events = <mask>;
}
...
spe_get_event(<pevents>, <nevents>, <timeout>);
```

LIBSPE2 PPU Example

```
#include <libspe2.h>
...
spe_context_ptr_t <speid>;
spe_event_handler_ptr_t event_handler;
#define NUM_EVENTS <#>
spe_event_unit_t <pevents>[NUM_EVENTS];
int <nevents> = NUM_EVENTS;
int <mask>;
int <timeout>;
int i;
spe_stop_info_t stopinfo;
...
event_handler = spe_event_handler_create();
...
<speid>=spe_context_create(SPE_EVENTS_ENABLE, NULL);
...
<pevents>[0].events = <mask>;
<pevents>[0].spe = <speid>;
spe_event_handler_register(event_handler, &<pevents>[0]);
...
<nevents> = spe_event_wait(...);
...
for (i=0; i < <nevents>; i++) {
/* The spe_stop_info_read loop should check for SPE_EVENT_SPE_STOPPED
   event received in the events mask */
   if (<pevents>[i].events & SPE_EVENT_SPE_STOPPED) {spe_stop_info_read();}
   ...
}
...
spe_event_handler_deregister(event_handler, &<pevents>[0]);
...
spe_event_handler_destroy(event_handler);
```

Function: spe_get_group

The spe_get_group function is eliminated from LIBSPE2.

Introduction

The spe_gid_t spe_get_group(speid_t **speid**) function has been eliminated. There is no replacement in LIBSPE2.

LIBSPE1 PPU Example

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

LIBSPE2 PPU Example

No replacement is possible.

Function: spe_get_ls

The spe_get_ls function is replaced by the spe_ls_area_get function in LIBSPE2.

Introduction

The void *spe_get_ls(speid_t **speid**) function has been replaced by the void *spe_ls_area_get(spe_context_ptr_t **spe**) function.

The speid_t typedef is replaced by the spe_context_ptr_t typedef.

LIBSPE1 PPU Example

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

LIBSPE2 PPU Example

```
#include <libspe2.h>
...
spe_context_ptr_t <speid>;
void *<ls>;
...
<ls> = spe_ls_area_get(<speid>);
```

Function: spe_get_ps_area

The `spe_get_ps_area` function is replaced by the `spe_ps_area_get` function in LIBSPE2.

Introduction

The `void *spe_get_ps_area(speid_t` **`speid,`** `enum` **`ps_area`**`)` function is replaced by the `int spe_ps_area_get(spe_context_ptr_t` **`spe,`** `enum` **`pa_area`**`)` function.

The following table shows the changes for LIBSPE2.

LIBSPE1	**LIBSPE2**
`void *spe_get_ps_area(speid_t` **`speid,`** `enum` **`ps_area`**`)` function	`int spe_ps_area_get(spe_context_ptr_t` **`spe,`** `enum` **`pa_area`**`)` function
`speid_t` typedef	`spe_context_ptr_t` typedef
ps_area parameter	Unchanged including all existing enumeration values and secondary data structures.

LIBSPE1 PPU Example

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

LIBSPE2 PPU Example

```
#include <libspe2.h>
...
spe_context_ptr_t <speid>;
enum ps_area area;
void *<ps>;
...
<ps> = spe_ps_area_get(<speid>, area);
```

Function: spe_get_priority, spe_set_priority, spe_get_policy

The `spe_get_priority`, `spe_set_priority`, and `spe_get_policy` functions are eliminated from LIBSPE2.

Introduction

The `int spe_get_priority(spe_gid_t` **`gid`**`)`, `int spe_set_priority(spe_gid_t` **`gid`**`,` `int` **`priority`**`)`, and `int spe_get_policy(spe_gid_t` **`gid`**`)` functions have been eliminated. The following table shows their replacements:

LIBSPE1	LIBSPE2
`int spe_get_priority(spe_gid_t` **`gid`**`)` function	`pthread_attr_getschedparam` function and a previously initialized thread attribute object
`int spe_set_priority(spe_gid_t` **`gid`**`,` `int` **`priority`**`)` function	`pthread_attr_setschedparam` function and a previously initialized thread attribute object
`int spe_get_policy(spe_gid_t` **`gid`**`)` function	`pthread_attr_getschedpolicy` function and a previously initialized thread attribute object
`spe_gid_t` typedef	`pthread_attr_t` typedef
priority parameter	Unchanged

LIBSPE1 PPU Example

```
#include <libspe.h>
...
spe_gid_t <group>;
int <priority>;
int <policy>;
...
<priority> = spe_get_priority(<group>);
...
spe_set_priority(<group>, <priority>);
...
<policy> = spe_get_policy(<group>);
```

LIBSPE2 PPU Example

```
#include <pthread.h>
...
int <priority>;
int <policy>;
pthread_attr_t attr;
struct sched_param param;
...
pthread_attr_getschedparam(&attr, &param);
<priority> = param.sched_priority;
...
param.sched_priority = <priority>;
pthread_attr_setschedparam(&attr, &param);
...
pthread_attr_getschedpolicy(&attr, &<policy>);
```

Function: spe_get_threads

The spe_get_threads function is eliminated from LIBSPE2.

Introduction

The int spe_get_threads(spe_gid_t **gid,** speid_t ***spe_ids**) function has been eliminated. There is no replacement for this function in LIBSPE2.

LIBSPE1 PPU Example

```
#include <libspe.h>
...
speid_t <speids>[16];
spe_gid_t <group>;
...
spe_get_threads(<group>, <speids>);
```

LIBSPE2 PPU Example

No replacement is possible.

Function: spe_group_defaults

The `spe_group_defaults` function is eliminated from LIBSPE2.

Introduction

The `int spe_group_defaults(int policy, int priority, int spe_events)` function has been eliminated. There is no replacement for this function in LIBSPE2.

LIBSPE1 PPU Example

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

LIBSPE2 PPU Example

No replacement is possible.

Function: spe_group_max

The spe_group_max function is eliminated from LIBSPE2.

Introduction

The int spe_group_max(spe_gid_t **gid**) function has been eliminated. There is no replacement for this function in LIBSPE2. You can consider using the spe_cpu_info_get function.

LIBSPE1 PPU Example

```
#include <libspe.h>
...
spe_gid_t <group>;
int <count>;
...
<count> = spe_group_max(<group>);
```

LIBSPE2 PPU Example

No replacement is possible.

Function: spe_kill

The spe_kill function is eliminated from LIBSPE2.

Introduction

The int spe_kill(speid_t **speid,** int **signal**) function has been eliminated. It is replaced by the pthread_cancel(pthread_t **thread,** int **sig**) function.

The speid_t typedef is replaced by a combination of pthread_t and spe_context_ptr_t typedefs.

LIBSPE1 PPU Example

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

LIBSPE2 PPU Example

```
#include <libspe2.h>
#include <pthread.h>
...
spe_context_ptr_t <speid>;
pthread_t pthread;
...
pthread_cancel(pthread);
spe_context_destroy(<speid>);
```

Function: spe_open_image, spe_close_image

The `spe_open_image` and `spe_close_image` functions have been replaced in LIBSPE2.

Introduction

The `spe_open_image` and `spe_close_image` functions have been replaced. The following table shows the changes required to migrate your code to the new functions:

LIBSPE1	LIBSPE2
`spe_program_handle_t *spe_open_image(const char **filename**)` function	`spe_program_t *spe_image_open(const char **filename**)` function
`int spe_close_image(spe_program_handle_t **program**)` function	`int spe_image_close(spe_program_handle t **program**)` function
filename parameter	Unchanged
program parameter	Unchanged

LIBSPE1 PPU Example

```
#include <libspe.h>
...
spe_program_handle_t *<program_handle>;
...
<program_handle> = spe_open_image("<filename>");
...
spe_close_image(<program_handle>);
```

LIBSPE2 PPU Example

```
#include <libspe2.h>
...
spe_program_handle_t *<program_handle>;
...
<program_handle> = spe_image_open("<filename>");
...
spe_image_close(<program_handle>);
```

Function: spe_set_app_data, spe_get_app_data

The spe_set_app_data and spe_get_app_data functions have been replaced in LIBSPE2.

Introduction

The int spe_set_app_data(speid_t **speid,** void ***data**) and int spe_get_app_data(speid_t **speid,** void ****p_data**) functions are replaced by a combination of the spe_event_handler_create, spe_event_handler_register, spe_event_wait, spe_event_handler_deregister, and spe_event_handler_destroy functions.

- The speid_t typedef is replaced by the spe_context_ptr_t typedef.
- The **data** parameter is mapped to the **spe_event_data_t** parameter in the **spe_event_unit_t** parameter both as input when registering with the spe_event_handler_register function and as output after a wait using the spe_event_wait function.

LIBSPE1 PPU Example

```
#include <libspe.h>

speid_t <speid>;
spe_gid_t <group>;
spe_program_handle_t <program_handle>;
void *<argp>;
void *<envp>;
unsigned long <mask>;
int <flags>;
void *<data>;
...
<speid>=spe_create_thread(<group>, &<program_handle>, <argp>, <envp>,
                          <mask>, <flags>);
...
spe_set_app_data(<speid>, <data>);
...
spe_get_app_data(<speid>, &<data>);
```

LIBSPE2 PPU Example

```
#include <libspe2.h>

spe_context_ptr_t <speid>;
unsigned int <flags>;
int <mask>;
spe_event_handler_ptr_t event_handler;
#define NUM_EVENTS <#>
spe_event_unit_t <pevents>[NUM_EVENTS];
int <nevents> = NUM_EVENTS;
int <timeout>;
void *<data>;
...
<speid>=spe_context_create(<flags>, NULL);
...
<pevents>[0].events = <mask>;
<pevents>[0].spe = <speid>;
<pevents>[0].data.ptr = &<data>;
spe_event_handler_register(event_handler, &<pevents>[0]);
...
<nevents> = spe_event_wait(event_handler, <pevents>, <nevents>, <timeout>);
...
<data> = (int*)<pevents>[0].data.ptr;
```

```
...
spe_event_handler_deregister(event_handler, &<pevents>[0]);
...
spe_event_handler_destroy(event_handler);
```

Function: spe_wait

The spe_wait function is eliminated from LIBSPE2.

Introduction

The int spe_wait(speid_t **speid**, int ***status**, int **options**) function has been eliminated. The following table shows the details of its replacement.

LIBSPE1	LIBSPE2
int spe_wait(speid_t **speid**, int ***status**, int **options**) function	Combination of the int spe_context_run(spe_context_ptr_t **spe**, unsigned int ***entry**, unsigned int **runflags**, void ***argp**, void ***envp**, spe_stop_info_t ***stopinfo**) function and the int pthread_join(pthread_t **thread**, void ****value_ptr**) function
speid typedef	Combination of spe_context_ptr_t and pthread_t typedefs
status parameter	**stopinfo.stop_reason** parameter along with **stopinfo.result.spe_exit_code** parameter or **stopinfo.result.spe_signal_code** parameter which is received from the spe_context_run function.
WNOHANG, WUNTRACED options	No replacement

LIBSPE1 PPU Example

```
#include <libspe.h>
...
speid_t <speid>;
int <status>;
int <options>;
...
spe_wait(<speid>, &<status>, <options>);
```

LIBSPE2 PPU Example

```
#include <libspe2.h>
#include <pthread.h>
...
typedef struct ppu_pthread_data {
   spe_context_ptr_t <speid>;
   pthread_t pthread;
   unsigned int entry;
   unsigned int flags;
   void *argp;
   void *envp;
   spe_stop_info_t stopinfo;
} ppu_pthread_data_t;
...
ppu_pthread_data_t ppdata;
void *value_ptr;
int <status>;
...
pthread_join(ppdata.pthread, &value_ptr);
<status> = ppdata.stopinfo.stop_reason;
...
spe_context_destroy(ppdata.<speid>);
```

Typedef: speid_t

The speid_t typedef is eliminated from LIBSPE2.

Introduction

The speid_t typedef is replaced by either the spe_context_ptr_t typedef or the pthread_t typedef as appropriate. In declarations, the type of the variable is typically changed from speid_t to spe_context_ptr_t and a new variable is declared as a pthread_t type.

LIBSPE1 PPU Example

```
#include <libspe.h>
...
speid_t <speid>;
```

LIBSPE2 PPU Example

```
#include <libspe2.h>
#include <pthread.h>
...
spe_context_ptr_t <speid>;
pthread_t pthread;
```

Typedef: spe_gid_t

The `spe_gid_t` typedef is eliminated from LIBSPE2.

Introduction

The `spe_gid_t` typedef has been eliminated. There is no replacement for groups in LIBSPE2.

LIBSPE1 PPU Example

```
#include <libspe.h>
...
spe_gid_t <group>;
```

LIBSPE2 PPU Example

No replacement is possible.

Typedef: spe_program_handle_t

The spe_program_handle_t typedef is unchanged in LIBSPE2.

Introduction

The spe_program_handle_t typedef is unchanged.

LIBSPE1 PPU Example

```
#include <libspe.h>
...
spe_program_handle_t <program_handle>;
```

LIBSPE2 PPU Example

```
#include <libspe2.h>
...
spe_program_handle_t <program_handle>;
```

Chapter 3. MFC Problem State Facilities

This section shows how to migrate the MFC Problem State Facilities functions.

Function: spe_mfc_get, spe_mfc_getb, spe_mfc_getf

The spe_mfc_get, spe_mfc_getb, and spe_mfc_getf functions have been replaced by other functions in LIBSPE2.

Introduction

The spe_mfc_get, spe_mfc_getb, and spe_mfc_getf functions have been replaced by other functions as shown in the following table:

LIBSPE1	LIBSPE2
int spe_mfc_get(speid_t **speid**, unsigned int **ls**, void ***ea**, unsigned int **size**, unsigned int **tag**, unsigned int **tid**, unsigned int **rid**) function	int spe_mfcio_get(spe_context_ptr_t **spe**, unsigned int **lsa**, void ***ea**, unsigned int **size**, unsigned int **tag**, unsigned int **tid**, unsigned int **rid**) function
int spe_mfc_getb(speid_t **speid**, unsigned int **ls**, void ***ea**, unsigned int **size**, unsigned int **tag**, unsigned int **tid**, unsigned int **rid**) function	int spe_mfcio_getb(spe_context_ptr_t **spe**, unsigned int **lsa**, void ***ea**, unsigned int **size**, unsigned int **tag**, unsigned int **tid**, unsigned int **rid**) function
int spe_mfc_getf(speid_t **speid**, unsigned int **ls**, void ***ea**, unsigned int **size**, unsigned int **tag**, unsigned int **tid**, unsigned int **rid**) function	int spe_mfcio_getf(spe_context_ptr_t **spe**, unsigned int **lsa**, void ***ea**, unsigned int **size**, unsigned int **tag**, unsigned int **tid**, unsigned int **rid**) function
speid_t typedef	spe_context_ptr_t typedef
All other arguments	Unchanged

LIBSPE1 PPU Example

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

LIBSPE2 PPU Example

```
#include <libspe2.h>
...
spe_context_ptr_t <speid>;
unsigned int <ls>;
void *<ea>;
unsigned int <size>;
unsigned int <tag>;
unsigned int <tid>;
unsigned int <rid>;
...
spe_mfcio_get(<speid>, <ls>, <ea>, <size>, <tag>, <tid>, <rid>);
...
```

```
spe_mfcio_getb(<speid>, <ls>, <ea>, <size>, <tag>, <tid>, <rid>);
...
spe_mfcio_getf(<speid>, <ls>, <ea>, <size>, <tag>, <tid>, <rid>);
```

Function: spe_mfc_put, spe_mfc_putb, spe_mfc_putf

The spe_mfc_put, spe_mfc_putb, and spe_mfc_putf functions have been replaced by other functions in LIBSPE2.

Introduction

The spe_mfc_put, spe_mfc_putb, and spe_mfc_putf functions have been replaced by other functions as shown in the following table:

LIBSPE1	LIBSPE2
int spe_mfc_put(speid_t **speid**, unsigned int **ls**, void ***ea**, unsigned int **size**, unsigned int **tag**, unsigned int **tid**, unsigned int **rid**) function	int spe_mfcio_put(spe_context_ptr_t **spe**, unsigned int **lsa**, void ***ea**, unsigned int **size**, unsigned int **tag**, unsigned int **tid**, unsigned int **rid**) function
int spe_mfc_putb(speid_t **speid**, unsigned int **ls**, void ***ea**, unsigned int **size**, unsigned int **tag**, unsigned int **tid**, unsigned int **rid**) function	int spe_mfcio_putb(spe_context_ptr_t **spe**, unsigned int **lsa**, void ***ea**, unsigned int **size**, unsigned int **tag**, unsigned int **tid**, unsigned int **rid**) function
int spe_mfc_putf(speid_t **speid**, unsigned int **ls**, void ***ea**, unsigned int **size**, unsigned int **tag**, unsigned int **tid**, unsigned int **rid**) function	int spe_mfcio_putf(spe_context_ptr_t **spe**, unsigned int **lsa**, void ***ea**, unsigned int **size**, unsigned int **tag**, unsigned int **tid**, unsigned int **rid**) function
speid_t typedef	spe_context_ptr_t typedef
All other arguments	Unchanged

LIBSPE1 PPU Example

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

LIBSPE2 PPU Example

```
#include <libspe2.h>
...
spe_context_ptr_t <speid>;
unsigned int <ls>;
void *<ea>;
unsigned int <size>;
unsigned int <tag>;
unsigned int <tid>;
unsigned int <rid>;
...
spe_mfcio_put(<speid>, <ls>, <ea>, <size>, <tag>, <tid>, <rid>);
...
```

```
spe_mfcio_putb(<speid>, <ls>, <ea>, <size>, <tag>, <tid>, <rid>);
...
spe_mfcio_putf(<speid>, <ls>, <ea>, <size>, <tag>, <tid>, <rid>);
```

Function: spe_mfc_read_tag_status_all, spe_mfc_read_tag_status_any, spe_mfc_read_tag_status_immediate

The spe_mfc_read_tag_status_all, spe_mfc_read_tag_status_any, and spe_mfc_read_tag_status_immediate functions have been replaced by other functions in LIBSPE2.

Introduction

The spe_mfc_read_tag_status_all, spe_mfc_read_tag_status_any, and spe_mfc_read_tag_status_immediate functions have been replaced by other functions as shown in the following table:

LIBSPE1	LIBSPE2
int spe_mfc_read_tag_status_all(speid_t **speid**, unsigned int **mask**) function	int spe_mfcio_tag_status_read(spe_context_ptr_t **spe**, unsigned int **mask**, unsigned int **behavior**, unsigned int ***tag_status**) function with behavior set to SPE_TAG_ALL
int spe_mfc_read_tag_status_any(speid_t **speid**, unsigned int **mask**) function	int spe_mfcio_tag_status_read(spe_context_ptr_t **spe**, unsigned int **mask**, unsigned int **behavior**, unsigned int ***tag_status**) function with behavior set to SPE_TAG_ANY
int spe_mfc_read_tag_status_immediate(speid_t **speid**, unsigned int **mask**) function	int spe_mfcio_tag_status_read(spe_context_ptr_t **spe**, unsigned int **mask**, unsigned int **behavior**, unsigned int ***tag_status**) function with behavior set to SPE_TAG_IMMEDIATE
Function return values	Value set in **tag_status**
speid_t typedef	spe_context_ptr_t typedef
mask parameter	Unchanged

LIBSPE1 PPU Example

```
#include <libspe.h>
...
speid_t <speid>;
unsigned int <mask>;
int <tag_status>;
...
<tag_status> = spe_mfc_read_tag_status_all(<speid>, <mask>);
...
<tag_status> = spe_mfc_read_tag_status_any(<speid>, <mask>);
...
<tag_status> = spe_mfc_read_tag_status_immediate(<speid>, <mask>);
```

LIBSPE2 PPU Example

```
#include <libspe2.h>
...
spe_context_ptr_t <speid>;
unsigned int <mask>;
unsigned int <tag_status>;
...
spe_mfcio_tag_status_read(<speid>, <mask>, SPE_TAG_ALL, &<tag_status>);
...
spe_mfcio_tag_status_read(<speid>, <mask>, SPE_TAG_ANY, &<tag_status>);
...
spe_mfcio_tag_status_read(<speid>, <mask>, SPE_TAG_IMMEDIATE, &<tag_status>);
```

Function: spe_read_out_mbox

The spe_read_out_mbox function has been replaced in LIBSPE2.

Introduction

The following table shows the changes required to migrate code that uses the spe_read_out_mbox function.

LIBSPE1	LIBSPE2
unsigned int spe_read_out_mbox(speid_t **speid**) function	int spe_out_mbox_read(spe_context_ptr_t **spe**, unsigned int ***mbox_data**, int **count**) function
	Set the **mbox_data** parameter to an unsigned integer pointer
	Set the **count** parameter to 1
speid_t typedef	spe_context_ptr_t typedef

LIBSPE1 PPU Example

```
#include <libspe.h>
...
speid_t <speid>;
unsigned int <data>;
...
<data> = spe_read_out_mbox(<speid>);
```

LIBSPE2 PPU Example

```
#include <libspe2.h>
...
spe_context_ptr_t <speid>;
unsigned int <data>;
...
spe_out_mbox_read(<speid>, &<data>, 1);
```

Function: spe_stat_in_mbox, spe_stat_out_mbox, spe_stat_out_intr_mbox

The spe_stat_in_mbox, spe_stat_out_mbox, and spe_stat_out_intr_mbox functions have been replaced in LIBSPE2.

Introduction

The following table shows how to migrate code that uses the spe_stat_in_mbox, spe_stat_out_mbox, and spe_stat_out_intr_mbox functions:

LIBSPE1	LIBSPE2
int spe_stat_in_mbox(speid_t **speid**) function	int spe_in_mbox_status(spe_context_ptr_t **spe**) function
int spe_stat_out_mbox(speid_t **speid**) function	int spe_out_mbox_status(spe_context_ptr_t **spe**) function
int spe_stat_out_intr_mbox(speid_t **speid**) function	int spe_out_intr_mbox_status(spe_context_ptr_t **spe**) function
speid_t typedef	spe_context_ptr_t typedef

LIBSPE1 PPU Example

```
#include <libspe.h>
...
speid_t <speid>;
int <status>;
...
<status> = spe_stat_in_mbox(<speid>);
...
<status> = spe_stat_out_mbox(<speid>);
...
<status> = spe_stat_out_intr_mbox(<speid>);
```

LIBSPE2 PPU Example

```
#include <libspe2.h>
...
spe_context_ptr_t <speid>;
int <status>;
...
<status> = spe_in_mbox_status(<speid>);
...
<status> = spe_out_mbox_status(<speid>);
...
<status> = spe_out_intr_mbox_status(<speid>);
```

Function: spe_write_in_mbox

The spe_write_in_mbox function has been replaced in LIBSPE2.

Introduction

The spe_write_in_mbox function is replaced by the spe_in_mbox_write function. The following table shows the changes required to migrate your code to the new function:

LIBSPE1	LIBSPE2
int spe_write_in_mbox(speid_t **speid**, unsigned int **data**) function	int spe_in_mbox_write(spe_context_ptr_t **spe**, unsigned int ***mbox_data**, int **count**, unsigned int **behavior**) function
	Set the **mbox_data** parameter to point to an unsigned integer data
	Set the **behavior** parameter to SPE_MBOX_ANY_NONBLOCKING
speid_t typedef	spe_context_ptr_t typedef
data parameter	**mbox_data** parameter that contains the address of the **data** parameter

LIBSPE1 PPU Example

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

LIBSPE2 PPU Example

```
/* For passing an integer */
#include <libspe2.h>
#include <sync_utils.h>
...
spe_context_ptr_t <speid>;
unsigned int <data>;
...
spe_in_mbox_write(<speid>, &<data>, 1, SPE_MBOX_ANY_NONBLOCKING);
```

or,

```
/* For passing a 32-bit effective address low-order word */
#include <libspe2.h>
...
spe_context_ptr_t <speid>;
unsigned int <data>;
addr64 data_addr;
...
data_addr.ull = (unsigned long long)&<data>;
spe_in_mbox_write(<speid>, &data_addr.ui[1], 1, SPE_MBOX_ANY_NONBLOCKING);
```

Function: spe_write_signal

The `spe_write_signal` function is replaced by the `spe_signal_write` function in LIBSPE2.

Introduction

The following table shows how to migrate your code to the new `spe_signal_write` function:

LIBSPE1	LIBSPE2
`int spe_write_signal(speid_t` **`speid`**`, unsigned int` **`signal_reg`**`, unsigned int` **`data`**`)` function	`int spe_signal_write(spe_context_ptr_t` **`spe`**`, unsigned int` **`signal_reg`**`, unsigned int` **`data`**`)` function
`speid_t` typedef	`spe_context_ptr_t` typedef
All other arguments	Unchanged

LIBSPE1 PPU Example

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

LIBSPE2 PPU Example

```
#include <libspe2.h>
...
spe_context_ptr_t <speid>;
unsigned int <signal_reg>;
unsigned int <data>;
...
spe_signal_write(<speid>, <signal_reg>, <data>);
```

Chapter 4. Examples

The following sections give complete program examples showing the migration from LIBSPE1 to LIBSPE2.

Example: Non-threaded PPU/SPU application (non-embedded)

This is an example of a non-threaded PPU/SPU application.

Shared SPU Example

This is an SPU program. It is used by the LIBSPE2 example as the program named teslibspe2hello.

```
#include<stdio.h>

int main(long long speid, void *argp, void *envp) {
   printf("\t\tHello World! speid=0x%llx, argp=%p, envp=%p\n", speid,
          argp, envp);
   return 0;
}
```

LIBSPE1 PPU Example

In LIBSPE1, defining and running a non-threaded SPU application is not possible. All PPU applications must create a SPE thread using the `spe_create_thread` function to launch an SPU application (see the following example). Alternatively, you can launch a standalone SPU application from the PPU command line using the elfspe capability.

LIBSPE2 PPU Example

```
#include <stdio.h>
#include <libspe2.h>

int main(void) {
   spe_context_ptr_t context;
   unsigned int entry = SPE_DEFAULT_ENTRY;
   spe_program_handle_t *program;
   spe_stop_info_t stop_info;

   context = spe_context_create(0, NULL);
   program = spe_image_open("testlibspe2hello");
   spe_program_load(context, program);
   spe_context_run(context, &entry, 0, NULL, NULL, &stop_info);
   spe_context_destroy(context);

   return 0;
}
```

The following is the output from the example:

```
Hello World! speid=0x181f008, argp=(nil), envp=(nil)
```

Example: Single-threaded PPU/SPU application (non-embedded)

This is an example of a single-threaded PPU/SPU application.

Shared SPU Example

This is an SPU program. It is used by the LIBSPE1 example as the program named testlibspe1hello and it is used by the LIBSPE2 example as the program named testlibspe2hello.

```
#include<stdio.h>

int main(long long speid, void *argp, void *envp) {
   printf("\t\tHello World! speid=0x%llx, argp=%p, envp=%p\n", speid,
          argp, envp);
   return 0;
}
```

LIBSPE1 PPU Example

```
#include <stdio.h>
#include <libspe.h>

int main(void) {
   spe_program_handle_t *program;
   speid_t speid;
   int status;

   program = spe_open_image("testlibspe1hello");
   speid = spe_create_thread(SPE_DEF_GRP, program, NULL, NULL, -1, 0);
   spe_wait(speid, &status, 0);
   return 0;
}
```

LIBSPE2 PPU Example

A secondary function must be defined which is passed to the `pthread_create` function. The secondary function should run the SPU context.

```
#include <stdio.h>
#include <libspe2.h>
#include <pthread.h>

void *ppu_pthread_function(void *arg) {
   spe_context_ptr_t context = *(spe_context_ptr_t *) arg;
   unsigned int entry = SPE_DEFAULT_ENTRY;
   spe_stop_info_t stop_info;

   spe_context_run(context, &entry, 0, NULL, NULL, &stop_info);
   pthread_exit(NULL);
}

int main(void) {
   spe_program_handle_t *program;
   spe_context_ptr_t context;
   int flags = 0;
   pthread_t pthread;

   context = spe_context_create(flags, NULL);
   program = spe_image_open("testlibspe2hello");
   spe_program_load(context, program);
   pthread_create(&pthread, NULL, &ppu_pthread_function, &context);
   pthread_join(pthread, NULL);
   spe_context_destroy(context);
   return 0;
}
```

The following is the output from the example:

```
Hello World! speid=0x1812050, argp=(nil), envp=(nil)
```

Example: Mailbox PPU/SPU

This is an SPU program. It is used by the LIBSPE1 example as the program named testlibspe1mailbox and it is used by the LIBSPE2 example as the program named testlibspe2mailbox.

Shared SPU Example

This example is shared by both LIBSPE1 and LIBSPE2. It is an SPU program.

```
#include <stdio.h>
#include <spu_mfcio.h>

int main(long long speid, void *argp, void *envp) {
   unsigned int data;
   printf("\t\tMailbox! speid=0x%llx, argp=%p, envp=%p\n", speid,
          argp, envp);
   printf("\t\tRead mailbox, waiting...\n");
   data = spu_read_in_mbox();
   printf("\t\tRead mailbox, data=%x\n", data);
   data++;
   printf("\t\tWrite mailbox, data=%x\n", data);
   spu_write_out_mbox(data);
   printf("\t\tWrite mailbox, completed\n");
   return 0;
}
```

LIBSPE1 PPU Example

```
#include <stdio.h>
#include <libspe.h>

int main(void) {
   spe_program_handle_t *program;
   speid_t speid;
   int status;
   int data;

   program = spe_open_image("testlibspe1mailbox");
   speid = spe_create_thread(SPE_DEF_GRP, program, NULL, NULL, -1, 0);
   data = 1;
   printf("Write mailbox, data=%x\n", data);
   spe_write_in_mbox(speid, data);
   printf("Write mailbox, completed\n");
   printf("Read mailbox, waiting...\n");
   while (spe_stat_out_mbox(speid) < 1);
   data = spe_read_out_mbox(speid);
   printf("Read mailbox, data=%x\n", data);
   spe_wait(speid, &status, 0);
   return 0;
}
```

LIBSPE2 PPU Example

```
#include <stdio.h>
#include <libspe2.h>
#include <pthread.h>

void *ppu_pthread_function(void *arg) {
   spe_context_ptr_t context = *(spe_context_ptr_t *) arg;
   unsigned int entry = SPE_DEFAULT_ENTRY;
   spe_stop_info_t stop_info;

   spe_context_run(context, &entry, 0, NULL, NULL, &stop_info);
   pthread_exit(NULL);
}
```

```
int main(void) {
   spe_program_handle_t *program;
   spe_context_ptr_t context;
   int flags = 0;
   pthread_t pthread;
   unsigned int data;

   context = spe_context_create(flags, NULL);
   program = spe_image_open("testlibspe2mailbox");
   spe_program_load(context, program);
   pthread_create(&pthread, NULL, &ppu_pthread_function, &context);
   data = 1;
   printf("Write mailbox, data=%x\n", data);
   spe_in_mbox_write(context, &data, 1, SPE_MBOX_ANY_NONBLOCKING);
   printf("Write mailbox, completed\n");
   printf("Read mailbox, waiting...\n");
   while (spe_out_mbox_status(context) < 1);
   spe_out_mbox_read(context, &data, 1);
   printf("Read mailbox, data=%x\n", data);
   pthread_join(pthread, NULL);
   spe_context_destroy(context);
   return 0;
}
```

The following is the output from the example:

```
Write mailbox, data=1
Write mailbox, completed
Read mailbox, waiting...
  Mailbox! speid=0x1812050, argp=(nil), envp=(nil)
  Read mailbox, waiting...
  Read mailbox, data=1
  Write mailbox, data=2
  Write mailbox, completed
Read mailbox, data=2
```

Appendix. Accessibility features

Accessibility features help users who have a physical disability, such as restricted mobility or limited vision, to use information technology products successfully.

The following list includes the major accessibility features:

- Keyboard-only operation
- Interfaces that are commonly used by screen readers
- Keys that are tactilely discernible and do not activate just by touching them
- Industry-standard devices for ports and connectors
- The attachment of alternative input and output devices

IBM and accessibility

See the IBM Accessibility Center at http://www.ibm.com/able/ for more information about the commitment that IBM has to accessibility.

Notices

This information was developed for products and services offered in the U.S.A.

The manufacturer may not offer the products, services, or features discussed in this document in other countries. Consult the manufacturer's representative for information on the products and services currently available in your area. Any reference to the manufacturer's product, program, or service is not intended to state or imply that only that product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any intellectual property right of the manufacturer may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any product, program, or service.

The manufacturer may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the manufacturer.

For license inquiries regarding double-byte (DBCS) information, contact the Intellectual Property Department in your country or send inquiries, in writing, to the manufacturer.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: THIS INFORMATION IS PROVIDED "AS IS " WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. The manufacturer may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to Web sites not owned by the manufacturer are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this product and use of those Web sites is at your own risk.

The manufacturer may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact the manufacturer.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, IBM License Agreement for Machine Code, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning products not produced by this manufacturer was obtained from the suppliers of those products, their published announcements or other publicly available sources. This manufacturer has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to products not produced by this manufacturer. Questions on the capabilities of products not produced by this manufacturer should be addressed to the suppliers of those products.

All statements regarding the manufacturer's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

The manufacturer's prices shown are the manufacturer's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to the manufacturer, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. The manufacturer, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

CODE LICENSE AND DISCLAIMER INFORMATION:

The manufacturer grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

SUBJECT TO ANY STATUTORY WARRANTIES WHICH CANNOT BE EXCLUDED, THE MANUFACTURER, ITS PROGRAM DEVELOPERS AND SUPPLIERS, MAKE NO WARRANTIES OR CONDITIONS EITHER EXPRESS OR

IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT, REGARDING THE PROGRAM OR TECHNICAL SUPPORT, IF ANY.

UNDER NO CIRCUMSTANCES IS THE MANUFACTURER, ITS PROGRAM DEVELOPERS OR SUPPLIERS LIABLE FOR ANY OF THE FOLLOWING, EVEN IF INFORMED OF THEIR POSSIBILITY:

1. LOSS OF, OR DAMAGE TO, DATA;
2. SPECIAL, INCIDENTAL, OR INDIRECT DAMAGES, OR FOR ANY ECONOMIC CONSEQUENTIAL DAMAGES; OR
3. LOST PROFITS, BUSINESS, REVENUE, GOODWILL, OR ANTICIPATED SAVINGS.

SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF DIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, SO SOME OR ALL OF THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO YOU.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information in softcopy, the photographs and color illustrations may not appear.

Edition notices

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

developerWorks
IBM
PowerPC
PowerPC Architecture
Resource Link

Cell Broadband Engine™ and Cell/B.E. are trademarks of Sony Computer Entertainment, Inc., in the United States, other countries, or both and is used under license therefrom.

Intel®, MMX, and Pentium® are trademarks of Intel Corporation in the United States, other countries, or both.

Java™ and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux® is a trademark of Linus Torvalds in the United States, other countries, or both.

Red Hat, the Red Hat "Shadow Man" logo, and all Red Hat-based trademarks and logos are trademarks or registered trademarks of Red Hat, Inc., in the United States and other countries.

UNIX® is a registered trademark of The Open Group in the United States and other countries.

Other company, product or service names may be trademarks or service marks of others.

Terms and conditions

Permissions for the use of these publications is granted subject to the following terms and conditions.

Personal Use: You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative works of these publications, or any portion thereof, without the express consent of the manufacturer.

Commercial Use: You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of the manufacturer.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any data, software or other intellectual property contained therein.

The manufacturer reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by the manufacturer, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

THE MANUFACTURER MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THESE PUBLICATIONS ARE PROVIDED ″AS-IS″ AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Related documentation

This topic helps you find related information.

Document location

Links to documentation for the SDK are provided on the developerWorks® Web site located at:

`http://www.ibm.com/developerworks/power/cell/`

Click on the **Docs** tab.

The following documents are available, organized by category:

Architecture

- *Cell Broadband Engine Architecture*
- *Cell Broadband Engine Registers*
- *SPU Instruction Set Architecture*

Standards

- *C/C++ Language Extensions for Cell Broadband Engine Architecture*
- *SPU Assembly Language Specification*
- *SPU Application Binary Interface Specification*
- *SIMD Math Library Specification for Cell Broadband Engine Architecture*
- *Cell Broadband Engine Linux Reference Implementation Application Binary Interface Specification*

Programming

- *Cell Broadband Engine Programming Handbook*
- *Programming Tutorial*
- *SDK for Multicore Acceleration Version 3.0 Programmer's Guide*

Library

- *SPE Runtime Management library*
- *SPE Runtime Management library Version 1 to Version 2 Migration Guide*
- *Accelerated Library Framework for Cell Programmer's Guide and API Reference*
- *Accelerated Library Framework for Hybrid-x86 Programmer's Guide and API Reference*
- *Data Communication and Synchronization for Cell Programmer's Guide and API Reference*
- *Data Communication and Synchronization for Hybrid-x86 Programmer's Guide and API Reference*
- *SIMD Math Library Specification*
- *Monte Carlo Library API Reference Manual (Prototype)*

Installation

- *SDK for Multicore Acceleration Version 3.0 Installation Guide*

IBM XL C/C++ Compiler and IBM XL Fortran Compiler

Detail about documentation for the compilers is available on the developerWorks Web site.

IBM Full-System Simulator and debugging documentation

Detail about documentation for the simulator and debugging tools is available on the developerWorks Web site.

PowerPC® Base

- *PowerPC Architecture™ Book, Version 2.02*
  - *Book I: PowerPC User Instruction Set Architecture*
  - *Book II: PowerPC Virtual Environment Architecture*
  - *Book III: PowerPC Operating Environment Architecture*
- *PowerPC Microprocessor Family: Vector/SIMD Multimedia Extension Technology Programming Environments Manual Version 2.07c*

Glossary

This glossary contains terms and abbreviations used in LIBSPE and Cell/B.E. systems.

ELF

Executable and Linking Format. The standard object format for many UNIX operating systems, including Linux. Compilers generate ELF files. Linkers link to files with ELF files in libraries. Systems run ELF files.

Gang context

The SPE gang context is one of the base data structures for the LIBSPE implementation. It holds all persistent information about a group of SPE contexts that should be treated as a gang, that is, be executed together with certain properties. This data structure should not be accessed directly; instead the application uses a pointer to an SPE gang context as an identifier for the SPE gang it is dealing with through LIBSPE API calls.

LS

Local Store. The 256-KB local store associated with each SPE. It holds both instructions and data.

Main thread

The application's main thread. In many cases, CBEA programs are multi-threaded using multiple SPEs running concurrently. A typical scenario is that the application consists of a main thread that creates as many SPE threads as needed and ″orchestrates″ them.

MFC

Memory Flow Controller. Part of an SPE which provides two main functions: it moves data via DMA between the SPE's local store (LS) and main storage, and it synchronizes the SPU with the rest of the processing units in the system.

PPE

PowerPC Processor Element. The general-purpose processor in the Cell/B.E. processor.

SPE

Synergistic Processor Element. It includes a SPU, a MFC, and a LS.

SPE context

The SPE context is one of the base data structures for the LIBSPE implementation. It holds all persistent information about a ″logical SPE″ used by the application. This data structure should not be accessed directly; instead the application uses a pointer to an SPE context as an identifier for the ″logical SPE″ it is dealing with through LIBSPE API calls.

SPE event

In a multi-threaded environment, it is often convenient to use an event mechanism for asynchronous notification. A common usage is that the main thread sets up an event handler to receive notification about certain events caused by the asynchronously running SPE threads. The current library supports events to indicate that an SPE has stopped execution, mailbox messages being written or read by an SPE, and PPE-initiated DMA operations have completed.

SPE thread

A thread scheduled and run on a SPE. A program has one or more SPE threads. Each such thread has its own SPU local store (LS), 128 x 128-bit register file, program counter, and MFC Command Queues, and it can communicate with other execution units (or with effective-address memory through the MFC channel interface). The API call `spe_context_run` is a synchronous, blocking call from the perspective of the thread using it, that is, while an SPE program is executed, the associated SPE thread blocks and is usually put to ″sleep″ by the operating system.

SPU

Synergistic Processor Unit. The part of an SPE that executes instructions from its local store (LS).

Index

Printed in USA

SC33-8332-01