

**NAME**

`spu_create` – create a new spu context

**SYNOPSIS**

```
#include <sys/types.h>
```

```
#include <sys/spu.h>
```

```
int spu_create(const char *pathname, int flags, mode_t mode);
```

```
int spu_create(const char *pathname, int flags, mode_t mode,  
               int neighbor_fd);
```

**DESCRIPTION**

The **spu\_create()** system call is used on PowerPC machines that implement the Cell Broadband Engine Architecture in order to access Synergistic Processor Units (SPUs). It creates a new logical context for an SPU in *pathname* and returns a file descriptor associated with it. *pathname* must refer to a non-existing directory in the mount point of the SPU file system (**spufs**). If **spu\_create()** is successful, a directory is created at *pathname* and it is populated with the files described in **spufs(7)**.

When a context is created, the returned file descriptor can only be passed to **spu\_run(2)**, used as the *dirfd* argument to the *\*at* family of system calls (e.g., **openat(2)**), or closed; other operations are not defined. A logical SPU context is destroyed (along with all files created within the context's *pathname* directory) once the last reference to the context has gone; this usually occurs when the file descriptor returned by **spu\_create()** is closed.

The *flags* argument can be zero or any bitwise OR-ed combination of the following constants:

**SPU\_CREATE\_EVENTS\_ENABLED**

Rather than using signals for reporting DMA errors, use the *event* argument to **spu\_run(2)**.

**SPU\_CREATE\_GANG**

Create an SPU gang instead of a context. (A gang is a group of SPU contexts that are functionally related to each other and which share common scheduling parameters — priority and policy. In the future, gang scheduling may be implemented causing the group to be switched in and out as a single unit.)

A new directory will be created at the location specified by the *pathname* argument. This gang may be used to hold other SPU contexts, by providing a *pathname* that is within the gang directory to further calls to **spu\_create()**.

**SPU\_CREATE\_NOSCHED**

Create a context that is not affected by the SPU scheduler. Once the context is run, it will not be scheduled out until it is destroyed by the creating process.

Because the context cannot be removed from the SPU, some functionality is disabled for **SPU\_CREATE\_NOSCHED** contexts. Only a subset of the files will be available in this context directory in **spufs**. Additionally, **SPU\_CREATE\_NOSCHED** contexts cannot dump a core file when crashing.

Creating **SPU\_CREATE\_NOSCHED** contexts requires the **CAP\_SYS\_NICE** capability.

**SPU\_CREATE\_ISOLATE**

Create an isolated SPU context. Isolated contexts are protected from some PPE (PowerPC Processing Element) operations, such as access to the SPU local store and the NPC register.

Creating **SPU\_CREATE\_ISOLATE** contexts also requires the **SPU\_CREATE\_NOSCHED** flag.

**SPU\_CREATE\_AFFINITY\_SPU**

Create a context with affinity to another SPU context. This affinity information is used within the SPU scheduling algorithm. Using this flag requires that a file descriptor referring to the other SPU

context be passed in the *neighbor\_fd* argument.

### **SPU\_CREATE\_AFFINITY\_MEM**

Create a context with affinity to system memory. This affinity information is used within the SPU scheduling algorithm.

The *mode* argument (minus any bits set in the process's **umask**(2)) specifies the permissions used for creating the new directory in **spufs**. See **stat**(2) for a full list of the possible *mode* values.

### **RETURN VALUE**

On success, **spu\_create**() returns a new file descriptor. On error, **-1** is returned, and *errno* is set to one of the error codes listed below.

### **ERRORS**

#### **EACCES**

The current user does not have write access to the **spufs**(7) mount point.

#### **EEXIST**

An SPU context already exists at the given path name.

#### **EFAULT**

*pathname* is not a valid string pointer in the calling process's address space.

#### **EINVAL**

*pathname* is not a directory in the **spufs**(7) mount point, or invalid flags have been provided.

#### **ELOOP**

Too many symbolic links were found while resolving *pathname*.

#### **EMFILE**

The process has reached its maximum open files limit.

#### **ENAMETOOLONG**

*pathname* is too long.

#### **ENFILE**

The system has reached the global open files limit.

#### **ENODEV**

An isolated context was requested, but the hardware does not support SPU isolation.

#### **ENOENT**

Part of *pathname* could not be resolved.

#### **ENOMEM**

The kernel could not allocate all resources required.

#### **ENOSPC**

There are not enough SPU resources available to create a new context or the user-specific limit for the number of SPU contexts has been reached.

#### **ENOSYS**

The functionality is not provided by the current system, because either the hardware does not provide SPU's or the **spufs** module is not loaded.

#### **ENOTDIR**

A part of *pathname* is not a directory.

#### **EPERM**

The **SPU\_CREATE\_NOSCHED** flag has been given, but the user does not have the **CAP\_SYS\_NICE** capability.

### **FILES**

*pathname* must point to a location beneath the mount point of **spufs**. By convention, it gets mounted in */spu*.

## VERSIONS

The **spu\_create()** system call was added to Linux in kernel 2.6.16.

## CONFORMING TO

This call is Linux-specific and only implemented on the PowerPC architecture. Programs using this system call are not portable.

## NOTES

Glibc does not provide a wrapper for this system call; call it using **syscall(2)**. Note however, that **spu\_create()** is meant to be used from libraries that implement a more abstract interface to SPUs, not to be used from regular applications. See <http://www.bsc.es/projects/deepcomputing/linuxoncell/> for the recommended libraries.

## EXAMPLE

See **spu\_run(2)** for an example of the use of **spu\_create()**

## SEE ALSO

**close(2)**, **spu\_run(2)**, **capabilities(7)**, **spufs(7)**

## COLOPHON

This page is part of release 3.22 of the Linux *man-pages* project. A description of the project, and information about reporting bugs, can be found at <http://www.kernel.org/doc/man-pages/>.