NAME
       spu_run − execute an SPU context

LIBRARY
       Standard C library (*libc*, *−lc*)

SYNOPSIS
       #include <sys/spu.h>   /* Definition of **SPU_*** constants */
       #include <sys/syscall.h>   /* Definition of **SYS_*** constants */
       #include <unistd.h>

       int syscall(SYS_spu_run, int *fd***, uint32_t ****npc***, uint32_t ****event***);**

       Note: glibc provides no wrapper for **spu_run**(), necessitating the use of **syscall**(2).

DESCRIPTION
       The **spu_run**() system call is used on PowerPC machines that implement the Cell Broadband Engine Ar-
       chitecture in order to access Synergistic Processor Units (SPUs). The *fd* argument is a file descriptor re-
       turned by **spu_create**(2) that refers to a specific SPU context. When the context gets scheduled to a physi-
       cal SPU, it starts execution at the instruction pointer passed in *npc*.

       Execution of SPU code happens synchronously, meaning that **spu_run**() blocks while the SPU is still run-
       ning. If there is a need to execute SPU code in parallel with other code on either the main CPU or other
       SPUs, a new thread of execution must be created first (e.g., using **pthread_create**(3)).

       When **spu_run**() returns, the current value of the SPU program counter is written to *npc*, so successive
       calls to **spu_run**() can use the same *npc* pointer.

       The *event* argument provides a buffer for an extended status code. If the SPU context was created with the
       SPU_CREATE_EVENTS_ENABLED flag, then this buffer is populated by the Linux kernel before
       spu_run() returns.

       The status code may be one (or more) of the following constants:

       SPE_EVENT_DMA_ALIGNMENT
              A DMA alignment error occurred.

       SPE_EVENT_INVALID_DMA
              An invalid MFC DMA command was attempted.

       SPE_EVENT_SPE_DATA_STORAGE
              A DMA storage error occurred.

       SPE_EVENT_SPE_ERROR
              An illegal instruction was executed.

       NULL is a valid value for the *event* argument. In this case, the events will not be reported to the calling
       process.

RETURN VALUE
       On success, **spu_run**() returns the value of the *spu_status* register. On failure, it returns −1 and sets *errno*
       is set to indicate the error.

       The *spu_status* register value is a bit mask of status codes and optionally a 14-bit code returned from the
       stop-and-signal instruction on the SPU. The bit masks for the status codes are:

       0x02   SPU was stopped by a **stop-and-signal** instruction.

       0x04   SPU was stopped by a **halt** instruction.

       0x08   SPU is waiting for a channel.

       0x10   SPU is in single-step mode.

       0x20   SPU has tried to execute an invalid instruction.

       0x40   SPU has tried to access an invalid channel.

0x3fff0000

>   The bits masked with this value contain the code returned from a **stop-and-signal** instruction.
>   These bits are valid only if the 0x02 bit is set.

If **spu_run**() has not returned an error, one or more bits among the lower eight ones are always set.

ERRORS

EBADF

>   *fd* is not a valid file descriptor.

EFAULT

>   *npc* is not a valid pointer, or *event* is non-NULL and an invalid pointer.

EINTR

>   A signal occurred while **spu_run**() was in progress; see **signal**(7). The *npc* value has been up-
>   dated to the new program counter value if necessary.

EINVAL

>   *fd* is not a valid file descriptor returned from **spu_create**(2).

ENOMEM

>   There was not enough memory available to handle a page fault resulting from a Memory Flow
>   Controller (MFC) direct memory access.

ENOSYS

>   The functionality is not provided by the current system, because either the hardware does not pro-
>   vide SPUs or the spufs module is not loaded.

VERSIONS

The **spu_run**() system call was added in Linux 2.6.16.

STANDARDS

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

NOTES

spu_run() 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 rec-
ommended libraries.

EXAMPLES

The following is an example of running a simple, one-instruction SPU program with the **spu_run**() system
call.

```
#include <err.h>
#include <fcntl.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

int main(void)
{
    int       context, fd, spu_status;
    uint32_t  instruction, npc;

    context = syscall(SYS_spu_create, "/spu/example-context", 0, 0755);
    if (context == -1)
        err(EXIT_FAILURE, "spu_create");

    /*
```

```
            * Write a 'stop 0x1234' instruction to the SPU's
            * local store memory.
            */
           instruction = 0x00001234;

           fd = open("/spu/example-context/mem", O_RDWR);
           if (fd == -1)
               err(EXIT_FAILURE, "open");
           write(fd, &instruction, sizeof(instruction));

           /*
            * set npc to the starting instruction address of the
            * SPU program. Since we wrote the instruction at the
            * start of the mem file, the entry point will be 0x0.
            */
           npc = 0;

           spu_status = syscall(SYS_spu_run, context, &npc, NULL);
           if (spu_status == -1)
               err(EXIT_FAILURE, "open");

           /*
            * We should see a status code of 0x12340002:
            *   0x00000002 (spu was stopped due to stop-and-signal)
            * | 0x12340000 (the stop-and-signal code)
            */
           printf("SPU Status: %#08x\n", spu_status);

           exit(EXIT_SUCCESS);
       }
```

SEE ALSO
       close(2), **spu_create**(2), **capabilities**(7), **spufs**(7)