NAME

spufs - the SPU file system

DESCRIPTION

The SPU file system is used on PowerPC machines that implement the Cell Broadband Engine Architecture in order to access Synergistic Processor Units (SPUs).

The file system provides a name space similar to posix shared memory or message queues. Users that have write permissions on the file system can use spu_create(2) to establish SPU contexts in the spufs root.

Every SPU context is represented by a directory containing a predefined set of files. These files can be used for manipulating the state of the logical SPU. Users can change permissions on those files, but not actually add or remove files.

MOUNT OPTIONS

uid=<uid>

set the user owning the mount point, the default is 0 (root).

gid=<gid>

set the group owning the mount point, the default is 0 (root).

FILES

The files in spufs mostly follow the standard behavior for regular system calls like read(2) or write(2), but often support only a subset of the operations supported on regular file systems. This list details the supported operations and the deviations from the behaviour in the respective man pages.

All files that support the read(2) operation also support readv(2) and all files that support the write(2) operation also support writev(2). All files support the access(2) and stat(2) family of operations, but only the st_mode, st_nlink, st_uid and st_gid fields of struct stat contain reliable information.

All files support the chmod(2)/fchmod(2) and chown(2)/fchown(2) operations, but will not be able to grant permissions that contradict the possible operations, e.g. read access on the wbox file.

The current set of files is:

/mem

the contents of the local storage memory of the SPU. This can be accessed like a regular shared memory file and contains both code and data in the address space of the SPU. The possible operations on an open mem file are:

read(2), pread(2), write(2), pwrite(2), lseek(2)
These operate as documented, with the exception that seek(2), write(2) and pwrite(2) are not supported beyond the end of the file. The file size is the size of the local storage of the SPU, which normally is 256 kilobytes.

mmap(2)
Mapping mem into the process address space gives access to the SPU local storage within the process address space. Only MAP_SHARED mappings are allowed.

/mbox
The first SPU to CPU communication mailbox. This file is read-only and can be read in units of 32 bits. The file can only be used in non-blocking mode and it even poll() will not block on it. The possible operations on an open mbox file are:

read(2)
If a count smaller than four is requested, read returns -1 and sets errno to EINVAL. If there is no data available in the mail box, the return value is set to -1 and errno becomes EAGAIN. When data has been read successfully, four bytes are placed in the data buffer and the value four is returned.

/ibox
The second SPU to CPU communication mailbox. This file is similar to the first mailbox file, but can be read in blocking I/O mode, and the poll family of system calls can be used to wait for it. The possible operations on an open ibox file are:

read(2)
If a count smaller than four is requested, read returns -1 and sets errno to EINVAL. If there is no data available in the mail box and the file descriptor has been opened with O_NONBLOCK, the return value is set to -1 and errno becomes EAGAIN.

If there is no data available in the mail box and the file descriptor has been opened without O_NONBLOCK, the call will block until the SPU writes to its interrupt mailbox channel. When data has been read successfully, four bytes are placed in the data buffer and the value four is returned.

poll(2)
Poll on the ibox file returns (POLLIN | POLLRDNORM) whenever data is available for reading.

/wbox
The CPU to SPU communation mailbox. It is write-only and can be written in units of 32 bits. If the mailbox is full, write() will block and poll can be used to wait for it becoming empty again. The possible operations on an open wbox file are: write(2) If a count smaller than four is requested, write returns -1 and sets errno to EINVAL. If there

is no space available in the mail box and the file descriptor has been opened with O_NONBLOCK, the return value is set to -1 and errno becomes EAGAIN.

If there is no space available in the mail box and the file descriptor has been opened without O_NONBLOCK, the call will block until the SPU reads from its PPE mailbox channel. When data has been read successfully, four bytes are placed in the data buffer and the value four is returned.

poll(2)
Poll on the ibox file returns (POLLOUT | POLLWRNORM) whenever space is available for writing.

/mbox_stat
/ibox_stat
/wbox_stat
Read-only files that contain the length of the current queue, i.e. how many words can be read from mbox or ibox or how many words can be written to wbox without blocking. The files can be read only in 4-byte units and return a big-endian binary integer number. The possible operations on an open *box_stat file are:

read(2)
If a count smaller than four is requested, read returns -1 and sets errno to EINVAL. Otherwise, a four byte value is placed in the data buffer, containing the number of elements that can be read from (for mbox_stat and ibox_stat) or written to (for wbox_stat) the respective mail box without blocking or resulting in EAGAIN.

/npc
/decr
/decr_status
/spu_tag_mask
/event_mask
/srr0
Internal registers of the SPU. The representation is an ASCII string with the numeric value of the next instruction to be executed. These can be used in read/write mode for debugging, but normal operation of programs should not rely on them because access to any of them except npc requires an SPU context save and is therefore very inefficient.

The contents of these files are:

npc	Next Program Counter
decr	SPU Decrementer
decr_status	Decrementer Status
spu_tag_mask	MFC tag mask for SPU DMA
event_mask	Event mask for SPU interrupts

srr0 Interrupt Return address register

The possible operations on an open npc, decr, decr_status, spu_tag_mask, event_mask or srr0 file are:

read(2)
When the count supplied to the read call is shorter than the required length for the pointer value plus a newline character, subsequent reads from the same file descriptor will result in completing the string, regardless of changes to the register by a running SPU task. When a complete string has been read, all subsequent read operations will return zero bytes and a new file descriptor needs to be opened to read the value again.

write(2)
A write operation on the file results in setting the register to the value given in the string. The string is parsed from the beginning to the first non-numeric character or the end of the buffer. Subsequent writes to the same file descriptor overwrite the previous setting.

/fpcr
This file gives access to the Floating Point Status and Control Register as a four byte long file. The operations on the fpcr file are:

read(2)
If a count smaller than four is requested, read returns -1 and sets errno to EINVAL. Otherwise, a four byte value is placed in the data buffer, containing the current value of the fpcr register.

write(2)
If a count smaller than four is requested, write returns -1 and sets errno to EINVAL. Otherwise, a four byte value is copied from the data buffer, updating the value of the fpcr register.

/signal1
/signal2
The two signal notification channels of an SPU. These are read-write files that operate on a 32 bit word. Writing to one of these files triggers an interrupt on the SPU. The value written to the signal files can be read from the SPU through a channel read or from host user space through the file. After the value has been read by the SPU, it is reset to zero. The possible operations on an open signal1 or signal2 file are:

read(2)
If a count smaller than four is requested, read returns -1 and sets errno to EINVAL. Otherwise, a four byte value is placed in the data buffer, containing the current value of the specified signal notification register.

write(2)
If a count smaller than four is requested, write returns -1 and sets errno to EINVAL. Otherwise, a four byte value is copied from the data buffer, updating the value of the specified signal notification register. The signal notification register will either be replaced with the input data or will be updated to the bitwise OR or the old value and the input data, depending on the contents of the signal1_type, or signal2_type respectively, file.

/signal1_type
/signal2_type
These two files change the behavior of the signal1 and signal2 notification files. The contain a numerical ASCII string which is read as either "1" or "0". In mode 0 (overwrite), the hardware replaces the contents of the signal channel with the data that is written to it. in mode 1 (logical OR), the hardware accumulates the bits that are subsequently written to it. The possible operations on an open signal1_type or signal2_type file are:

read(2)
When the count supplied to the read call is shorter than the required length for the digit plus a newline character, subsequent reads from the same file descriptor will result in completing the string. When a complete string has been read, all subsequent read operations will return zero bytes and a new file descriptor needs to be opened to read the value again.

write(2)
A write operation on the file results in setting the register to the value given in the string. The string is parsed from the beginning to the first non-numeric character or the end of the buffer. Subsequent writes to the same file descriptor overwrite the previous setting.

EXAMPLES
/etc/fstab entry
none /spu spufs gid=spu 0 0

AUTHORS
Arnd Bergmann <arndb@de.ibm.com>, Mark Nutter <mnutter@us.ibm.com>, Ulrich Weigand <Ulrich.Weigand@de.ibm.com>

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

Linux 2005-09-28 SPUFS(2)

SPU_RUN(2) Linux Programmer's Manual SPU_RUN(2)

NAME

spu_run - execute an spu context

SYNOPSIS

#include <sys/spu.h>

int spu_run(int fd, unsigned int *npc, unsigned int *event);

DESCRIPTION

The spu_run system call is used on PowerPC machines that implement the Cell Broadband Engine Architecture in order to access Synergistic Processor Units (SPUs). It uses the fd that was returned from spu_create(2) to address a specific SPU context. When the context gets scheduled to a physical SPU, it starts execution at the instruction pointer passed in npc.

Execution of SPU code happens synchronously, meaning that spu_run does not return while the SPU is still running. If there is a need to execute SPU code in parallel with other code on either the main CPU or other SPUs, you need to create a new thread of execution first, e.g. using the pthread_create(3) call.

When spu_run returns, the current value of the SPU instruction pointer is written back to npc, so you can call spu_run again without updating the pointers.

event can be a NULL pointer or point to an extended status code that gets filled when spu_run returns. It can be one of the following constants:

SPE_EVENT_DMA_ALIGNMENT
A DMA alignment error

SPE_EVENT_SPE_DATA_SEGMENT
A DMA segmentation error

SPE_EVENT_SPE_DATA_STORAGE
A DMA storage error

If NULL is passed as the event argument, these errors will result in a signal delivered to the calling process.

RETURN VALUE

spu_run returns the value of the spu_status register or -1 to indicate an error and set errno to one of the error codes listed below. The spu_status register value contains 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 stop-and-signal.

0x04 SPU was stopped by halt.

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 stop-and-signal.

There are always one or more of the lower eight bits set or an error code is returned from spu_run.

ERRORS

EAGAIN or EWOULDBLOCK
fd is in non-blocking mode and spu_run would block.

EBADF fd is not a valid file descriptor.

EFAULT npc is not a valid pointer or status is neither NULL nor a valid pointer.

EINTR A signal occurred while spu_run was in progress. The npc value has been updated to the new program counter value if necessary.

EINVAL fd is not a file descriptor returned from spu_create(2).

ENOMEM Insufficient memory was available to handle a page fault resulting from an MFC direct memory access.

ENOSYS the functionality is not provided by the current system, because either the hardware does not provide SPUs or the spufs module is not loaded.

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 recommended libraries.

CONFORMING TO

This call is Linux specific and only implemented by the ppc64 architecture. Programs using this system call are not portable.

BUGS

The code does not yet fully implement all features lined out here.

AUTHOR

Arnd Bergmann <arndb@de.ibm.com>

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

Linux 2005-09-28 SPU_RUN(2)

SPU_CREATE(2) Linux Programmer's Manual SPU_CREATE(2)

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);
```

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 handle to associated with it. pathname must point to a non-existing directory in the mount point of the SPU file system (spufs). When spu_create is successful, a directory gets created on pathname and it is populated with files.

The returned file handle can only be passed to spu_run(2) or closed, other operations are not defined on it. When it is closed, all associated directory entries in spufs are removed. When the last file handle pointing either inside of the context directory or to this file descriptor is closed, the logical SPU context is destroyed.

The parameter flags can be zero or any bitwise or'd combination of the following constants:

SPU_RAWIO
Allow mapping of some of the hardware registers of the SPU into user space. This flag requires the CAP_SYS_RAWIO capability, see capabilities(7).

The mode parameter specifies the permissions used for creating the new directory in spufs. mode is modified with the user's umask(2) value and then used for both the directory and the files contained in it. The file permissions mask out some more bits of mode because they typically support only read or write access. See stat(2) for a full list of the possible mode values.

RETURN VALUE

spu_create returns a new file descriptor. It may return -1 to indicate an error condition and set errno to one of the error codes listed below.

ERRORS

EACCESS
The current user does not have write access on the spufs mount point.

EEXIST An SPU context already exists at the given path name.

EFAULT pathname is not a valid string pointer in the current address space.

EINVAL pathname is not a directory in the spufs mount point.

ELOOP Too many symlinks were found while resolving pathname.

EMFILE The process has reached its maximum open file limit.

ENAMETOOLONG
pathname was too long.

ENFILE The system has reached the global open file limit.

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 SPUs or the spufs module is not loaded.

ENOTDIR
A part of pathname is not a directory.

NOTES

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.

FILES

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

CONFORMING TO

This call is Linux specific and only implemented by the ppc64 architecture. Programs using this system call are not portable.

BUGS

The code does not yet fully implement all features lined out here.

AUTHOR

Arnd Bergmann <arndb@de.ibm.com>

SEE ALSO

capabilities(7), close(2), spu_run(2), spufs(7)