spufs

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 actu- ally 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 sys- tem 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) opera- tions, 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 success- fully, 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 Regis- ter 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 regis- ter.

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 sig- nal2 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 notifi- cation 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 subse- quently 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, subse- quent reads from the same file descriptor will result in com- pleting 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)