POK(kernelpart)

Generated by Doxygen 1.7.6.1

Wed May 9 2012 14:25:17

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 __attribute__ Struct Reference

```
#include <event.h>
```

Data Fields

- uint32_t offset_low:16
- uint32_t segsel:16
- uint32_t res0:8
- uint32_t type:3
- uint32_t d:1
- uint32_t res1:1
- uint32_t dpl:2
- uint32_t present:1
- uint32_t offset_high:16
- uint32_t limit_low:16
- uint32_t base_low:24 __attribute__ ((packed))
- uint32_t s:1
- uint32_t limit_high:4
- uint32_t available:2
- uint32_t op_size:1
- uint32_t granularity:1
- uint32_t base_high:8
- uint16_t limit
- uint32_t base
- uint16_t padding
- uint32_t back_link
- uint32_t esp0
- uint32_t ss0
- uint32_t esp1

- uint32_t ss1
- uint32_t esp2
- uint32_t ss2
- uint32_t cr3
- uint32_t eip
- uint32_t eflags
- uint32_t eax
- uint32_t ecx
- uint32_t edx
- uint32_t ebx
- uint32_t esp
- uint32_t ebp
- uint32_t esi
- uint32_t edi
- uint32_t es
- uint32_t cs
- uint32_t ss
- uint32_t ds
- uint32_t fs
- uint32_t gs
- uint32_t ldt
- uint16_t trace_trap
- uint16_t io_bit_map_offset

3.1.1 Detailed Description

Definition at line 33 of file event.h.

3.1.2 Field Documentation

3.1.2.1 uint32_t __attribute__::available

Definition at line 39 of file gdt.h.

3.1.2.2 uint32_t __attribute__::back_link

Definition at line 25 of file tss.h.

3.1.2.3 uint32_t __attribute__::base

Definition at line 24 of file sysdesc.h.

3.1.2.4 uint32_t __attribute__::base_high

Definition at line 42 of file gdt.h.

3.1.2.5 uint32_t __attribute__::base_low

Definition at line 33 of file gdt.h.

3.1.2.6 uint32_t __attribute__::cr3

Definition at line 32 of file tss.h.

3.1.2.7 uint32_t __attribute__::cs

Definition at line 44 of file tss.h.

3.1.2.8 uint32_t __attribute__::d

Definition at line 39 of file event.h.

3.1.2.9 uint32_t __attribute__::dpl

Definition at line 41 of file event.h.

3.1.2.10 uint32_t __attribute__::ds

Definition at line 46 of file tss.h.

3.1.2.11 uint32_t __attribute__::eax

Definition at line 35 of file tss.h.

3.1.2.12 uint32_t __attribute__::ebp

Definition at line 40 of file tss.h.

3.1.2.13 uint32_t __attribute__::ebx

Definition at line 38 of file tss.h.

3.1.2.14 uint32_t __attribute__::ecx

Definition at line 36 of file tss.h.

3.1.2.15 uint32_t __attribute__::edi

Definition at line 42 of file tss.h.

3.1.2.16 uint32_t __attribute__::edx

Definition at line 37 of file tss.h.

3.1.2.17 uint32_t __attribute__::eflags

Definition at line 34 of file tss.h.

3.1.2.18 uint32_t __attribute__::eip

Definition at line 33 of file tss.h.

3.1.2.19 uint32_t __attribute__::es

Definition at line 43 of file tss.h.

3.1.2.20 uint32_t __attribute__::esi

Definition at line 41 of file tss.h.

3.1.2.21 uint32_t __attribute__::esp

Definition at line 39 of file tss.h.

3.1.2.22 uint32_t __attribute__::esp0

Definition at line 26 of file tss.h.

3.1.2.23 uint32_t __attribute__::esp1

Definition at line 28 of file tss.h.

3.1.2.24 uint32_t __attribute__::esp2

Definition at line 30 of file tss.h.

3.1.2.25 uint32_t __attribute__::fs

Definition at line 47 of file tss.h.

3.1.2.26 uint32_t __attribute__::granularity

Definition at line 41 of file gdt.h.

3.1.2.27 uint32_t __attribute__::gs

Definition at line 48 of file tss.h.

3.1.2.28 uint16_t __attribute__::io_bit_map_offset

Definition at line 51 of file tss.h.

3.1.2.29 uint32_t __attribute__::ldt

Definition at line 49 of file tss.h.

3.1.2.30 uint16_t __attribute__::limit

Definition at line 23 of file sysdesc.h.

3.1.2.31 uint32_t __attribute__::limit_high

Definition at line 38 of file gdt.h.

3.1.2.32 uint32_t __attribute__::limit_low

Definition at line 32 of file gdt.h.

3.1.2.33 uint32_t __attribute__::offset_high

Definition at line 43 of file event.h.

3.1.2.34 uint32_t __attribute__::offset_low

Definition at line 35 of file event.h.

3.1.2.35 uint32_t __attribute__::op_size

Definition at line 40 of file gdt.h.

3.1.2.36 uint16_t __attribute__::padding

Definition at line 25 of file sysdesc.h.

3.1.2.37 uint32_t __attribute__::present

Definition at line 42 of file event.h.

3.1.2.38 uint32_t __attribute__::res0

Definition at line 37 of file event.h.

3.1.2.39 uint32_t __attribute__::res1

Definition at line 40 of file event.h.

3.1.2.40 uint32_t __attribute__::s

Definition at line 35 of file gdt.h.

3.1.2.41 uint32_t __attribute__::segsel

Definition at line 36 of file event.h.

3.1.2.42 uint32_t __attribute__::ss

Definition at line 45 of file tss.h.

3.1.2.43 uint32_t __attribute__::ss0

Definition at line 27 of file tss.h.

3.1.2.44 uint32_t __attribute__::ss1

Definition at line 29 of file tss.h.

3.1.2.45 uint32_t __attribute__::ss2

Definition at line 31 of file tss.h.

3.1.2.46 uint16_t __attribute__::trace_trap

Definition at line 50 of file tss.h.

3.1.2.47 uint32_t __attribute__::type

Definition at line 38 of file event.h.

The documentation for this struct was generated from the following files:

- /home/matias/projet/stage-pok/pok/trunk/kernel/arch/x86/event.h
- /home/matias/projet/stage-pok/pok/trunk/kernel/arch/x86/gdt.h
- /home/matias/projet/stage-pok/pok/trunk/kernel/arch/x86/sysdesc.h
- /home/matias/projet/stage-pok/pok/trunk/kernel/arch/x86/tss.h

3.2 context_t Struct Reference

```
#include <thread.h>
```

Data Fields

- uint32_t sp
- uint32_t unused_lr
- uint32_t cr
- uint32_t r2
- uint32_t r13
- uint32_t r14
- uint32_t r15
- uint32_t r16
- uint32_t r17
- uint32_t r18
- uint32_t r19
- uint32_t r20
- uint32_t r21
- uint32_t r22
- uint32_t r23
- uint32_t r24
- uint32_t r25
- uint32_t r26
- uint32_t r27

- uint32_t r28
- uint32_t r29
- uint32_t r30
- uint32_t r31
- uint32_t pad
- uint32_t back_chain
- uint32_t lr
- uint32_t edi
- uint32_t esi
- uint32_t ebp
- uint32_t __esp
- uint32_t ebx
- uint32_t edx
- uint32_t ecx
- uint32_t eax
- uint32_t eip
- uint32_t cs
- uint32_t eflags

3.2.1 Detailed Description

Definition at line 23 of file thread.h.

3.2.2 Field Documentation

3.2.2.1 uint32_t context_t::__esp

Definition at line 28 of file thread.h.

3.2.2.2 uint32_t context_t::back_chain

Definition at line 54 of file thread.h.

3.2.2.3 uint32_t context_t::cr

Definition at line 28 of file thread.h.

3.2.2.4 uint32_t context_t::cs

Definition at line 35 of file thread.h.

3.2.2.5 uint32_t context_t::eax

Definition at line 32 of file thread.h.

3.2.2.6 uint32_t context_t::ebp

Definition at line 27 of file thread.h.

3.2.2.7 uint32_t context_t::ebx

Definition at line 29 of file thread.h.

3.2.2.8 uint32_t context_t::ecx

Definition at line 31 of file thread.h.

3.2.2.9 uint32_t context_t::edi

Definition at line 25 of file thread.h.

3.2.2.10 uint32_t context_t::edx

Definition at line 30 of file thread.h.

3.2.2.11 uint32_t context_t::eflags

Definition at line 36 of file thread.h.

3.2.2.12 uint32_t context_t::eip

Definition at line 34 of file thread.h.

3.2.2.13 uint32_t context_t::esi

Definition at line 26 of file thread.h.

3.2.2.14 uint32_t context_t::lr

Definition at line 55 of file thread.h.

3.2.2.15 uint32_t context_t::pad

Definition at line 51 of file thread.h.

3.2.2.16 uint32_t context_t::r13

Definition at line 30 of file thread.h.

3.2.2.17 uint32_t context_t::r14

Definition at line 31 of file thread.h.

3.2.2.18 uint32_t context_t::r15

Definition at line 32 of file thread.h.

3.2.2.19 uint32_t context_t::r16

Definition at line 34 of file thread.h.

3.2.2.20 uint32_t context_t::r17

Definition at line 35 of file thread.h.

3.2.2.21 uint32_t context_t::r18

Definition at line 36 of file thread.h.

3.2.2.22 uint32_t context_t::r19

Definition at line 37 of file thread.h.

3.2.2.23 uint32_t context_t::r2

Definition at line 29 of file thread.h.

3.2.2.24 uint32_t context_t::r20

Definition at line 38 of file thread.h.

3.2.2.25 uint32_t context_t::r21

Definition at line 39 of file thread.h.

3.2.2.26 uint32_t context_t::r22

Definition at line 40 of file thread.h.

3.2.2.27 uint32_t context_t::r23

Definition at line 41 of file thread.h.

3.2.2.28 uint32_t context_t::r24

Definition at line 42 of file thread.h.

3.2.2.29 uint32_t context_t::r25

Definition at line 43 of file thread.h.

3.2.2.30 uint32_t context_t::r26

Definition at line 44 of file thread.h.

3.2.2.31 uint32_t context_t::r27

Definition at line 45 of file thread.h.

3.2.2.32 uint32_t context_t::r28

Definition at line 46 of file thread.h.

3.2.2.33 uint32_t context_t::r29

Definition at line 47 of file thread.h.

3.2.2.34 uint32_t context_t::r30

Definition at line 48 of file thread.h.

3.2.2.35 uint32_t context_t::r31

Definition at line 49 of file thread.h.

3.2.2.36 uint32_t context_t::sp

Definition at line 25 of file thread.h.

3.2.2.37 uint32_t context_t::unused_lr

Definition at line 26 of file thread.h.

The documentation for this struct was generated from the following files:

- /home/matias/projet/stage-pok/pok/trunk/kernel/arch/ppc/thread.h
- /home/matias/projet/stage-pok/pok/trunk/kernel/arch/x86/thread.h

3.3 cpio_bin_header Struct Reference

```
#include <cpio.h>
```

Data Fields

- unsigned short c_magic
- unsigned short c_dev
- unsigned short c_ino
- unsigned short c_mode
- unsigned short c_uid
- unsigned short c_gid
- unsigned short c_nlink
- unsigned short c_rdev
- unsigned short c_mtime [2]
- unsigned short c_namesize
- unsigned short c_filesize [2]

3.3.1 Detailed Description

Definition at line 33 of file cpio.h.

3.3.2 Field Documentation

3.3.2.1 unsigned short cpio_bin_header::c_dev

Definition at line 36 of file cpio.h.

3.3.2.2 unsigned short cpio_bin_header::c_filesize[2]

Definition at line 45 of file cpio.h.

3.3.2.3 unsigned short cpio_bin_header::c_gid

Definition at line 40 of file cpio.h.

3.3.2.4 unsigned short cpio_bin_header::c_ino

Definition at line 37 of file cpio.h.

3.3.2.5 unsigned short cpio_bin_header::c_magic

Definition at line 35 of file cpio.h.

3.3.2.6 unsigned short cpio_bin_header::c_mode

Definition at line 38 of file cpio.h.

3.3.2.7 unsigned short cpio_bin_header::c_mtime[2]

Definition at line 43 of file cpio.h.

3.3.2.8 unsigned short cpio_bin_header::c_namesize

Definition at line 44 of file cpio.h.

3.3.2.9 unsigned short cpio_bin_header::c_nlink

Definition at line 41 of file cpio.h.

3.3.2.10 unsigned short cpio_bin_header::c_rdev

Definition at line 42 of file cpio.h.

3.3.2.11 unsigned short cpio_bin_header::c_uid

Definition at line 39 of file cpio.h.

The documentation for this struct was generated from the following file:

- /home/matias/projet/stage-pok/pok/trunk/kernel/include/core/cpio.h

3.4 cpio_file Struct Reference

```
#include <cpio.h>
```

Data Fields

- int cpio_fmt
- void ∗ cpio_addr
- void ∗ curr_header
- void ∗ curr_fileaddr
- unsigned int curr_filesz
- char ∗ curr_filename
- unsigned int curr_filename_len
- int(∗ next_header)(struct cpio_file ∗cpio)

3.4.1 Detailed Description

Definition at line 48 of file cpio.h.

3.4.2 Field Documentation

3.4.2.1 void∗ **cpio_file::cpio_addr**

Definition at line 51 of file cpio.h.

3.4.2.2 int **cpio_file::cpio_fmt**

Definition at line 50 of file cpio.h.

3.4.2.3 void∗ **cpio_file::curr_fileaddr**

Definition at line 53 of file cpio.h.

3.4.2.4 char∗ **cpio_file::curr_filename**

Definition at line 55 of file cpio.h.

3.4.2.5 unsigned int **cpio_file::curr_filename_len**

Definition at line 56 of file cpio.h.

3.4.2.6 unsigned int **cpio_file::curr_filesz**

Definition at line 54 of file cpio.h.

3.4.2.7 void∗ cpio_file::curr_header

Definition at line 52 of file cpio.h.

3.4.2.8 int(∗ cpio_file::next_header)(struct cpio_file ∗cpio)

Definition at line 57 of file cpio.h.

The documentation for this struct was generated from the following file:

- /home/matias/projet/stage-pok/pok/trunk/kernel/include/core/cpio.h

3.5 Elf32_Ehdr Struct Reference

```
#include <elf.h>
```

Data Fields

- unsigned char e_ident [EI_NIDENT]
- Elf32_Half e_type
- Elf32_Half e_machine
- Elf32_Word e_version
- Elf32_Addr e_entry
- Elf32_Off e_phoff
- Elf32_Off e_shoff
- Elf32_Word e_flags
- Elf32_Half e_ehsize
- Elf32_Half e_phentsize
- Elf32_Half e_phnum
- Elf32_Half e_shentsize
- Elf32_Half e_shnum
- Elf32_Half e_shstrndx

3.5.1 Detailed Description

Definition at line 28 of file elf.h.

3.5.2 Field Documentation

3.5.2.1 Elf32_Half Elf32_Ehdr::e_ehsize

Definition at line 38 of file elf.h.

3.5.2.2 Elf32_Addr Elf32_Ehdr::e_entry

Definition at line 34 of file elf.h.

3.5.2.3 Elf32_Word Elf32_Ehdr::e_flags

Definition at line 37 of file elf.h.

3.5.2.4 unsigned char Elf32_Ehdr::e_ident[EI_NIDENT]

Definition at line 30 of file elf.h.

3.5.2.5 Elf32_Half Elf32_Ehdr::e_machine

Definition at line 32 of file elf.h.

3.5.2.6 Elf32_Half Elf32_Ehdr::e_phentsize

Definition at line 39 of file elf.h.

3.5.2.7 Elf32_Half Elf32_Ehdr::e_phnum

Definition at line 40 of file elf.h.

3.5.2.8 Elf32_Off Elf32_Ehdr::e_phoff

Definition at line 35 of file elf.h.

3.5.2.9 Elf32_Half Elf32_Ehdr::e_shentsize

Definition at line 41 of file elf.h.

3.5.2.10 Elf32_Half Elf32_Ehdr::e_shnum

Definition at line 42 of file elf.h.

3.5.2.11 Elf32_Off Elf32_Ehdr::e_shoff

Definition at line 36 of file elf.h.

3.5.2.12 Elf32_Half Elf32_Ehdr::e_shstrndx

Definition at line 43 of file elf.h.

3.5.2.13 Elf32_Half Elf32_Ehdr::e_type

Definition at line 31 of file elf.h.

3.5.2.14 Elf32_Word Elf32_Ehdr::e_version

Definition at line 33 of file elf.h.

The documentation for this struct was generated from the following file:

- /home/matias/projet/stage-pok/pok/trunk/kernel/include/elf.h

3.6 Elf32_Phdr Struct Reference

```
#include <elf.h>
```

Data Fields

- Elf32_Word p_type
- Elf32_Off p_offset
- Elf32_Addr p_vaddr
- Elf32_Addr p_paddr
- Elf32_Word p_filesz
- Elf32_Word p_memsz
- Elf32_Word p_flags
- Elf32_Word p_align

3.6.1 Detailed Description

Definition at line 48 of file elf.h.

3.6.2 Field Documentation

3.6.2.1 Elf32_Word Elf32_Phdr::p_align

Definition at line 57 of file elf.h.

3.6.2.2 Elf32_Word Elf32_Phdr::p_filesz

Definition at line 54 of file elf.h.

3.6.2.3 Elf32_Word Elf32_Phdr::p_flags

Definition at line 56 of file elf.h.

3.6.2.4 Elf32_Word Elf32_Phdr::p_memsz

Definition at line 55 of file elf.h.

3.6.2.5 Elf32_Off Elf32_Phdr::p_offset

Definition at line 51 of file elf.h.

3.6.2.6 Elf32_Addr Elf32_Phdr::p_paddr

Definition at line 53 of file elf.h.

3.6.2.7 Elf32_Word Elf32_Phdr::p_type

Definition at line 50 of file elf.h.

3.6.2.8 Elf32_Addr Elf32_Phdr::p_vaddr

Definition at line 52 of file elf.h.

The documentation for this struct was generated from the following file:

- /home/matias/projet/stage-pok/pok/trunk/kernel/include/elf.h

3.7 interrupt_frame Struct Reference

```
#include <interrupt.h>
```

Data Fields

- uint32_t es
- uint32_t ds
- uint32_t edi
- uint32_t esi

- uint32_t ebp
- uint32_t __esp
- uint32_t ebx
- uint32_t edx
- uint32_t ecx
- uint32_t eax
- uint32_t error
- uint32_t eip
- uint32_t cs
- uint32_t eflags
- uint32_t esp
- uint32_t ss

3.7.1 Detailed Description

Definition at line 24 of file interrupt.h.

3.7.2 Field Documentation

3.7.2.1 uint32_t interrupt_frame::__esp

Definition at line 31 of file interrupt.h.

3.7.2.2 uint32_t interrupt_frame::cs

Definition at line 40 of file interrupt.h.

3.7.2.3 uint32_t interrupt_frame::ds

Definition at line 27 of file interrupt.h.

3.7.2.4 uint32_t interrupt_frame::eax

Definition at line 35 of file interrupt.h.

3.7.2.5 uint32_t interrupt_frame::ebp

Definition at line 30 of file interrupt.h.

3.7.2.6 uint32_t interrupt_frame::ebx

Definition at line 32 of file interrupt.h.

3.7.2.7 uint32_t interrupt_frame::ecx

Definition at line 34 of file interrupt.h.

3.7.2.8 uint32_t interrupt_frame::edi

Definition at line 28 of file interrupt.h.

3.7.2.9 uint32_t interrupt_frame::edx

Definition at line 33 of file interrupt.h.

3.7.2.10 uint32_t interrupt_frame::eflags

Definition at line 41 of file interrupt.h.

3.7.2.11 uint32_t interrupt_frame::eip

Definition at line 39 of file interrupt.h.

3.7.2.12 uint32_t interrupt_frame::error

Definition at line 38 of file interrupt.h.

3.7.2.13 uint32_t interrupt_frame::es

Definition at line 26 of file interrupt.h.

3.7.2.14 uint32_t interrupt_frame::esi

Definition at line 29 of file interrupt.h.

3.7.2.15 uint32_t interrupt_frame::esp

Definition at line 45 of file interrupt.h.

3.7.2.16 uint32_t interrupt_frame::ss

Definition at line 46 of file interrupt.h.

The documentation for this struct was generated from the following file:

- /home/matias/projet/stage-pok/pok/trunk/kernel/include/arch/x86/interrupt.h

3.8 pok_aout_symbol_table_t Struct Reference

```
#include <multiboot.h>
```

Data Fields

- unsigned int tabsize
- unsigned int strsize
- unsigned int addr
- unsigned int reserved

3.8.1 Detailed Description

Definition at line 76 of file multiboot.h.

3.8.2 Field Documentation

3.8.2.1 unsigned int **pok_aout_symbol_table_t::addr**

Definition at line 80 of file multiboot.h.

3.8.2.2 unsigned int **pok_aout_symbol_table_t::reserved**

Definition at line 81 of file multiboot.h.

3.8.2.3 unsigned int **pok_aout_symbol_table_t::strsize**

Definition at line 79 of file multiboot.h.

3.8.2.4 unsigned int **pok_aout_symbol_table_t::tabsize**

Definition at line 78 of file multiboot.h.

The documentation for this struct was generated from the following file:

- /home/matias/projet/stage-pok/pok/trunk/kernel/include/arch/x86/multiboot.h

3.9 pok_elf_section_header_table_t Struct Reference

```
#include <multiboot.h>
```

Data Fields

- unsigned int num
- unsigned int size
- unsigned int addr
- unsigned int shndx

3.9.1 Detailed Description

Definition at line 84 of file multiboot.h.

3.9.2 Field Documentation

3.9.2.1 unsigned int **pok_elf_section_header_table_t::addr**

Definition at line 88 of file multiboot.h.

3.9.2.2 unsigned int **pok_elf_section_header_table_t::num**

Definition at line 86 of file multiboot.h.

3.9.2.3 unsigned int **pok_elf_section_header_table_t::shndx**

Definition at line 89 of file multiboot.h.

3.9.2.4 unsigned int **pok_elf_section_header_table_t::size**

Definition at line 87 of file multiboot.h.

The documentation for this struct was generated from the following file:

- /home/matias/projet/stage-pok/pok/trunk/kernel/include/arch/x86/multiboot.h

3.10 pok_lockobj_attr_t Struct Reference

```
#include <lockobj.h>
```

Data Fields

- pok_lockobj_kind_t kind
- pok_locking_policy_t locking_policy
- pok_queueing_discipline_t queueing_policy
- pok_sem_value_t initial_value
- pok_sem_value_t max_value

3.10.1 Detailed Description

Definition at line 63 of file lockobj.h.

3.10.2 Field Documentation

3.10.2.1 pok_sem_value_t pok_lockobj_attr_t::initial_value

Definition at line 68 of file lockobj.h.

3.10.2.2 pok_lockobj_kind_t pok_lockobj_attr_t::kind

Definition at line 65 of file lockobj.h.

3.10.2.3 pok_locking_policy_t pok_lockobj_attr_t::locking_policy

Definition at line 66 of file lockobj.h.

3.10.2.4 pok_sem_value_t pok_lockobj_attr_t::max_value

Definition at line 69 of file lockobj.h.

3.10.2.5 pok_queueing_discipline_t pok_lockobj_attr_t::queueing_policy

Definition at line 67 of file lockobj.h.

The documentation for this struct was generated from the following file:

- /home/matias/projet/stage-pok/pok/trunk/kernel/include/core/lockobj.h

3.11 pok_lockobj_lockattr_t Struct Reference

```
#include <lockobj.h>
```

Data Fields

- pok_lockobj_operation_t operation
- pok_lockobj_kind_t obj_kind
- pok_lockobj_lock_kind_t lock_kind
- uint64_t time

3.11.1 Detailed Description

Definition at line 115 of file lockobj.h.

3.11.2 Field Documentation

3.11.2.1 pok_lockobj_lock_kind_t pok_lockobj_lockattr_t::lock_kind

Definition at line 119 of file lockobj.h.

3.11.2.2 pok_lockobj_kind_t pok_lockobj_lockattr_t::obj_kind

Definition at line 118 of file lockobj.h.

3.11.2.3 pok_lockobj_operation_t pok_lockobj_lockattr_t::operation

Definition at line 117 of file lockobj.h.

3.11.2.4 uint64_t pok_lockobj_lockattr_t::time

Definition at line 120 of file lockobj.h.

The documentation for this struct was generated from the following file:

- /home/matias/projet/stage-pok/pok/trunk/kernel/include/core/lockobj.h

3.12 pok_lockobj_t Struct Reference

```
#include <lockobj.h>
```

Data Fields

- pok_spinlock_t spin
- pok_spinlock_t eventspin
- bool_t is_locked
- pok_mutex_state_t thread_state [POK_CONFIG_NB_THREADS+2]
- pok_locking_policy_t locking_policy
- pok_queueing_discipline_t queueing_policy
- pok_lockobj_kind_t kind
- bool_t initialized
- uint16_t current_value
- uint16_t max_value

3.12.1 Detailed Description

Definition at line 72 of file lockobj.h.

3.12.2 Field Documentation

3.12.2.1 uint16_t pok_lockobj_t::current_value

Definition at line 95 of file lockobj.h.

3.12.2.2 pok_spinlock_t pok_lockobj_t::eventspin

Definition at line 75 of file lockobj.h.

3.12.2.3 bool_t pok_lockobj_t::initialized

Definition at line 92 of file lockobj.h.

3.12.2.4 bool_t pok_lockobj_t::is_locked

Definition at line 78 of file lockobj.h.

3.12.2.5 pok_lockobj_kind_t pok_lockobj_t::kind

Definition at line 90 of file lockobj.h.

3.12.2.6 pok_locking_policy_t pok_lockobj_t::locking_policy

Definition at line 84 of file lockobj.h.

3.12.2.7 uint16_t pok_lockobj_t::max_value

Definition at line 96 of file lockobj.h.

3.12.2.8 pok_queueing_discipline_t pok_lockobj_t::queueing_policy

Definition at line 87 of file lockobj.h.

3.12.2.9 pok_spinlock_t pok_lockobj_t::spin

Definition at line 74 of file lockobj.h.

3.12.2.10 pok_mutex_state_t pok_lockobj_t::thread_state[POK_CONFIG_NB_THREADS+2]

Definition at line 81 of file lockobj.h.

The documentation for this struct was generated from the following file:

- /home/matias/projet/stage-pok/pok/trunk/kernel/include/core/lockobj.h

3.13 pok_memory_map_t Struct Reference

```
#include <multiboot.h>
```

Data Fields

- unsigned int size
- unsigned int base_addr_low
- unsigned int base_addr_high
- unsigned int length_low
- unsigned int length_high
- unsigned int type

3.13.1 Detailed Description

Definition at line 120 of file multiboot.h.

3.13.2 Field Documentation

3.13.2.1 unsigned int pok_memory_map_t::base_addr_high

Definition at line 124 of file multiboot.h.

3.13.2.2 unsigned int pok_memory_map_t::base_addr_low

Definition at line 123 of file multiboot.h.

3.13.2.3 unsigned int pok_memory_map_t::length_high

Definition at line 126 of file multiboot.h.

3.13.2.4 unsigned int pok_memory_map_t::length_low

Definition at line 125 of file multiboot.h.

3.13.2.5 unsigned int pok_memory_map_t::size

Definition at line 122 of file multiboot.h.

3.13.2.6 unsigned int pok_memory_map_t::type

Definition at line 127 of file multiboot.h.

The documentation for this struct was generated from the following file:

- /home/matias/projet/stage-pok/pok/trunk/kernel/include/arch/x86/multiboot.h

3.14 pok_module_t Struct Reference

```
#include <multiboot.h>
```

Data Fields

- unsigned int mod_start
- unsigned int mod_end
- unsigned int string
- unsigned int reserved

3.14.1 Detailed Description

Definition at line 112 of file multiboot.h.

3.14.2 Field Documentation

3.14.2.1 unsigned int pok_module_t::mod_end

Definition at line 115 of file multiboot.h.

3.14.2.2 unsigned int pok_module_t::mod_start

Definition at line 114 of file multiboot.h.

3.14.2.3 unsigned int pok_module_t::reserved

Definition at line 117 of file multiboot.h.

3.14.2.4 unsigned int pok_module_t::string

Definition at line 116 of file multiboot.h.

The documentation for this struct was generated from the following file:

- /home/matias/projet/stage-pok/pok/trunk/kernel/include/arch/x86/multiboot.h

3.15 pok_multiboot_header_t Struct Reference

```
#include <multiboot.h>
```

Data Fields

- unsigned int magic
- unsigned int flags
- unsigned int checksum
- unsigned int header_addr
- unsigned int load_addr
- unsigned int load_end_addr
- unsigned int bss_end_addr
- unsigned int entry_addr

3.15.1 Detailed Description

Definition at line 64 of file multiboot.h.

3.15.2 Field Documentation

3.15.2.1 unsigned int pok_multiboot_header_t::bss_end_addr

Definition at line 72 of file multiboot.h.

3.15.2.2 unsigned int pok_multiboot_header_t::checksum

Definition at line 68 of file multiboot.h.

3.15.2.3 unsigned int pok_multiboot_header_t::entry_addr

Definition at line 73 of file multiboot.h.

3.15.2.4 unsigned int pok_multiboot_header_t::flags

Definition at line 67 of file multiboot.h.

3.15.2.5 unsigned int pok_multiboot_header_t::header_addr

Definition at line 69 of file multiboot.h.

3.15.2.6 unsigned int pok_multiboot_header_t::load_addr

Definition at line 70 of file multiboot.h.

3.15.2.7 unsigned int pok_multiboot_header_t::load_end_addr

Definition at line 71 of file multiboot.h.

3.15.2.8 unsigned int pok_multiboot_header_t::magic

Definition at line 66 of file multiboot.h.

The documentation for this struct was generated from the following file:

- /home/matias/projet/stage-pok/pok/trunk/kernel/include/arch/x86/multiboot.h

3.16 pok_multiboot_info_t Struct Reference

```
#include <multiboot.h>
```

Data Fields

- unsigned int flags
- unsigned int mem_lower
- unsigned int mem_upper
- unsigned int boot_device
- unsigned int cmdline
- unsigned int mods_count
- unsigned int mods_addr
- union {
    pok_aout_symbol_table_t aout_sym
    pok_elf_section_header_table_t elf_sec
  } u
- unsigned int mmap_length
- unsigned int mmap_addr

3.16.1 Detailed Description

Definition at line 92 of file multiboot.h.

3.16.2 Field Documentation

3.16.2.1 pok_aout_symbol_table_t pok_multiboot_info_t::aout_sym

Definition at line 104 of file multiboot.h.

3.16.2.2 unsigned int pok_multiboot_info_t::boot_device

Definition at line 97 of file multiboot.h.

3.16.2.3 unsigned int pok_multiboot_info_t::cmdline

Definition at line 98 of file multiboot.h.

3.16.2.4 pok_elf_section_header_table_t pok_multiboot_info_t::elf_sec

Definition at line 105 of file multiboot.h.

3.16.2.5 unsigned int pok_multiboot_info_t::flags

Definition at line 94 of file multiboot.h.

3.16.2.6 unsigned int pok_multiboot_info_t::mem_lower

Definition at line 95 of file multiboot.h.

3.16.2.7 unsigned int pok_multiboot_info_t::mem_upper

Definition at line 96 of file multiboot.h.

3.16.2.8 unsigned int pok_multiboot_info_t::mmap_addr

Definition at line 109 of file multiboot.h.

3.16.2.9 unsigned int pok_multiboot_info_t::mmap_length

Definition at line 108 of file multiboot.h.

3.16.2.10 unsigned int pok_multiboot_info_t::mods_addr

Definition at line 100 of file multiboot.h.

3.16.2.11 unsigned int pok_multiboot_info_t::mods_count

Definition at line 99 of file multiboot.h.

3.16.2.12 union { ... } pok_multiboot_info_t::u

The documentation for this struct was generated from the following file:

- /home/matias/projet/stage-pok/pok/trunk/kernel/include/arch/x86/multiboot.h

3.17 pok_port_t Struct Reference

```
#include <port.h>
```

Data Fields

- pok_port_id_t identifier
- pok_partition_id_t partition
- pok_port_size_t index
- bool_t full
- pok_port_size_t size
- pok_port_size_t off_b
- pok_port_size_t off_e
- pok_port_direction_t direction
- pok_port_queueing_discipline_t discipline
- pok_bool_t ready
- bool_t empty
- uint8_t kind
- uint64_t refresh
- uint64_t last_receive
- pok_lockobj_t lock
- bool_t must_be_flushed

3.17.1 Detailed Description

Definition at line 57 of file port.h.

3.17.2 Field Documentation

3.17.2.1 pok_port_direction_t pok_port_t::direction

Definition at line 66 of file port.h.

3.17.2.2 pok_port_queueing_discipline_t pok_port_t::discipline

Definition at line 67 of file port.h.

3.17.2.3 bool_t pok_port_t::empty

Definition at line 69 of file port.h.

3.17.2.4 bool_t pok_port_t::full

Definition at line 62 of file port.h.

3.17.2.5 pok_port_id_t pok_port_t::identifier

Definition at line 59 of file port.h.

3.17.2.6 pok_port_size_t pok_port_t::index

Definition at line 61 of file port.h.

3.17.2.7 uint8_t pok_port_t::kind

Definition at line 70 of file port.h.

3.17.2.8 uint64_t pok_port_t::last_receive

Definition at line 72 of file port.h.

3.17.2.9 pok_lockobj_t pok_port_t::lock

Definition at line 73 of file port.h.

3.17.2.10 bool_t pok_port_t::must_be_flushed

Definition at line 74 of file port.h.

3.17.2.11 pok_port_size_t pok_port_t::off_b

Definition at line 64 of file port.h.

3.17.2.12 pok_port_size_t pok_port_t::off_e

Definition at line 65 of file port.h.

3.17.2.13 pok_partition_id_t pok_port_t::partition

Definition at line 60 of file port.h.

3.17.2.14 pok_bool_t pok_port_t::ready

Definition at line 68 of file port.h.

3.17.2.15 uint64_t pok_port_t::refresh

Definition at line 71 of file port.h.

3.17.2.16 pok_port_size_t pok_port_t::size

Definition at line 63 of file port.h.

The documentation for this struct was generated from the following file:

- /home/matias/projet/stage-pok/pok/trunk/kernel/include/middleware/port.h

3.18 pok_space Struct Reference

Data Fields

- uint32_t phys_base
- uint32_t size

3.18.1 Detailed Description

Basic paritions informtation needed for memory managment.

Definition at line 34 of file space.c.

3.18.2 Field Documentation

3.18.2.1 uint32_t pok_space::phys_base

Pysical adress of the partition.

Definition at line 36 of file space.c.

3.18.2.2 uint32_t pok_space::size

Size of the partition.

Definition at line 37 of file space.c.

The documentation for this struct was generated from the following files:

- /home/matias/projet/stage-pok/pok/trunk/kernel/arch/ppc/space.c
- /home/matias/projet/stage-pok/pok/trunk/kernel/arch/sparc/space.c

3.19 pok_syscall_args_t Struct Reference

```
#include <syscall.h>
```

Data Fields

- uint32_t nargs
- uint32_t arg1
- uint32_t arg2
- uint32_t arg3
- uint32_t arg4
- uint32_t arg5

3.19.1 Detailed Description

Definition at line 86 of file syscall.h.

3.19.2 Field Documentation

3.19.2.1 uint32_t pok_syscall_args_t::arg1

Definition at line 89 of file syscall.h.

3.19.2.2 uint32_t pok_syscall_args_t::arg2

Definition at line 90 of file syscall.h.

3.19.2.3 uint32_t pok_syscall_args_t::arg3

Definition at line 91 of file syscall.h.

3.19.2.4 uint32_t pok_syscall_args_t::arg4

Definition at line 92 of file syscall.h.

3.19.2.5 uint32_t pok_syscall_args_t::arg5

Definition at line 93 of file syscall.h.

3.19.2.6 uint32_t pok_syscall_args_t::nargs

Definition at line 88 of file syscall.h.

The documentation for this struct was generated from the following file:

- /home/matias/projet/stage-pok/pok/trunk/kernel/include/core/syscall.h

3.20 pok_syscall_info_t Struct Reference

```
#include <syscall.h>
```

Data Fields

- pok_partition_id_t partition
- uint32_t thread
- uint32_t base_addr

3.20.1 Detailed Description

Definition at line 96 of file syscall.h.

3.20.2 Field Documentation

3.20.2.1 uint32_t pok_syscall_info_t::base_addr

Definition at line 100 of file syscall.h.

3.20.2.2 pok_partition_id_t pok_syscall_info_t::partition

Definition at line 98 of file syscall.h.

3.20.2.3 uint32_t pok_syscall_info_t::thread

Definition at line 99 of file syscall.h.

The documentation for this struct was generated from the following file:

- /home/matias/projet/stage-pok/pok/trunk/kernel/include/core/syscall.h

3.21 ppc_pte_t Struct Reference

Data Fields

- uint32_t vsid_api
- uint32_t rpn_flags

3.21.1 Detailed Description

Definition at line 109 of file space.c.

3.21.2 Field Documentation

3.21.2.1 uint32_t ppc_pte_t::rpn_flags

Definition at line 112 of file space.c.

3.21.2.2 uint32_t ppc_pte_t::vsid_api

Definition at line 111 of file space.c.

The documentation for this struct was generated from the following file:

- /home/matias/projet/stage-pok/pok/trunk/kernel/arch/ppc/space.c

3.22 space_context_t Struct Reference

```
#include <space.h>
```

Data Fields

- context_t ctx
- uint32_t fake_ret
- unsigned int partition_id
- uint32_t user_pc

- uint32_t user_sp
- uint32_t kernel_sp
- uint32_t arg1
- uint32_t arg2

3.22.1 Detailed Description

Definition at line 29 of file space.h.

3.22.2 Field Documentation

3.22.2.1 uint32_t space_context_t::arg1

Definition at line 37 of file space.h.

3.22.2.2 uint32_t space_context_t::arg2

Definition at line 38 of file space.h.

3.22.2.3 context_t space_context_t::ctx

Definition at line 31 of file space.h.

3.22.2.4 uint32_t space_context_t::fake_ret

Definition at line 32 of file space.h.

3.22.2.5 uint32_t space_context_t::kernel_sp

Definition at line 36 of file space.h.

3.22.2.6 unsigned int space_context_t::partition_id

Definition at line 33 of file space.h.

3.22.2.7 uint32_t space_context_t::user_pc

Definition at line 34 of file space.h.

3.22.2.8 uint32_t space_context_t::user_sp

Definition at line 35 of file space.h.

The documentation for this struct was generated from the following file:

- /home/matias/projet/stage-pok/pok/trunk/kernel/arch/x86/space.h

3.23 start_context_t Struct Reference

```
#include <thread.h>
```

Data Fields

- context_t ctx
- uint32_t fake_ret
- uint32_t entry
- uint32_t id

3.23.1 Detailed Description

Definition at line 40 of file thread.h.

3.23.2 Field Documentation

3.23.2.1 context_t start_context_t::ctx

Definition at line 42 of file thread.h.

3.23.2.2 uint32_t start_context_t::entry

Definition at line 44 of file thread.h.

3.23.2.3 uint32_t start_context_t::fake_ret

Definition at line 43 of file thread.h.

3.23.2.4 uint32_t start_context_t::id

Definition at line 45 of file thread.h.

The documentation for this struct was generated from the following file:

- /home/matias/projet/stage-pok/pok/trunk/kernel/arch/x86/thread.h

3.24 volatile_context_t Struct Reference

```
#include <thread.h>
```

Data Fields

- uint32_t sp
- uint32_t unused_lr
- uint32_t cr
- uint32_t r0
- uint32_t r2
- uint32_t r3
- uint32_t r4
- uint32_t r5
- uint32_t r6
- uint32_t r7
- uint32_t r8
- uint32_t r9
- uint32_t r10
- uint32_t r11
- uint32_t r12
- uint32_t r13
- uint32_t ctr
- uint32_t xer
- uint32_t srr0
- uint32_t srr1
- uint32_t back_chain
- uint32_t lr
- uint32_t pad0
- uint32_t pad1

3.24.1 Detailed Description

Definition at line 58 of file thread.h.

3.24.2 Field Documentation

3.24.2.1 uint32_t volatile_context_t::back_chain

Definition at line 83 of file thread.h.

3.24.2.2 uint32_t volatile_context_t::cr

Definition at line 63 of file thread.h.

3.24.2.3 uint32_t volatile_context_t::ctr

Definition at line 77 of file thread.h.

3.24.2.4 uint32_t volatile_context_t::lr

Definition at line 84 of file thread.h.

3.24.2.5 uint32_t volatile_context_t::pad0

Definition at line 87 of file thread.h.

3.24.2.6 uint32_t volatile_context_t::pad1

Definition at line 88 of file thread.h.

3.24.2.7 uint32_t volatile_context_t::r0

Definition at line 64 of file thread.h.

3.24.2.8 uint32_t volatile_context_t::r10

Definition at line 73 of file thread.h.

3.24.2.9 uint32_t volatile_context_t::r11

Definition at line 74 of file thread.h.

3.24.2.10 uint32_t volatile_context_t::r12

Definition at line 75 of file thread.h.

3.24.2.11 uint32_t volatile_context_t::r13

Definition at line 76 of file thread.h.

3.24.2.12 uint32_t volatile_context_t::r2

Definition at line 65 of file thread.h.

3.24.2.13 uint32_t volatile_context_t::r3

Definition at line 66 of file thread.h.

3.24.2.14 uint32_t volatile_context_t::r4

Definition at line 67 of file thread.h.

3.24.2.15 uint32_t volatile_context_t::r5

Definition at line 68 of file thread.h.

3.24.2.16 uint32_t volatile_context_t::r6

Definition at line 69 of file thread.h.

3.24.2.17 uint32_t volatile_context_t::r7

Definition at line 70 of file thread.h.

3.24.2.18 uint32_t volatile_context_t::r8

Definition at line 71 of file thread.h.

3.24.2.19 uint32_t volatile_context_t::r9

Definition at line 72 of file thread.h.

3.24.2.20 uint32_t volatile_context_t::sp

Definition at line 60 of file thread.h.

3.24.2.21 uint32_t volatile_context_t::srr0

Definition at line 79 of file thread.h.

3.24.2.22 uint32_t volatile_context_t::srr1

Definition at line 80 of file thread.h.

3.24.2.23 uint32_t volatile_context_t::unused_lr

Definition at line 61 of file thread.h.

3.24.2.24 uint32_t volatile_context_t::xer

Definition at line 78 of file thread.h.

The documentation for this struct was generated from the following file:

- /home/matias/projet/stage-pok/pok/trunk/kernel/arch/ppc/thread.h

Chapter 4

File Documentation

4.1 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/ppc/arch.c File Reference

Provide generic architecture access for PPC architecture.

`#include <types.h> #include <errno.h> #include <core/partition.- h> #include "msr.h"`

Functions

- void pok_arch_space_init (void)
- pok_ret_t pok_arch_init ()
- pok_ret_t pok_arch_preempt_disable ()
- pok_ret_t pok_arch_preempt_enable ()
- pok_ret_t pok_arch_idle ()
- pok_ret_t pok_arch_event_register (uint8_t vector, void(∗handler)(void))
- uint32_t pok_thread_stack_addr (const uint8_t partition_id, const uint32_t local- _thread_id)

4.1.1 Detailed Description

Provide generic architecture access for PPC architecture.

Author

Tristan Gingold

Date

2009

Definition in file arch.c.

4.1.2 Function Documentation

4.1.2.1 pok_ret_t pok_arch_event_register (uint8_t *vector*, void(∗)(void) *handler*)

Register an event (for example, an interruption)

Definition at line 83 of file arch.c.

```
{
  (void) vector;
  (void) handler;

  return (POK_ERRNO_OK);
}
```

4.1.2.2 pok_ret_t pok_arch_idle ( )

Function that do nothing. Useful for the idle task for example.

Definition at line 74 of file arch.c.

```
{
   while (1)
   {
   }

   return (POK_ERRNO_OK);
}
```

4.1.2.3 pok_ret_t pok_arch_init ( )

Function that initializes architecture concerns.

Definition at line 43 of file arch.c.

```
{
  set_msr (MSR_IP);
#if POK_NEEDS_PARTITIONS
  pok_arch_space_init();
#endif

  return (POK_ERRNO_OK);
}
```

4.1.2.4 pok_ret_t pok_arch_preempt_disable ( )

Disable interruptions

Definition at line 53 of file arch.c.

```
{
  unsigned int msr;

  msr = get_msr();
  msr &= ~MSR_EE;
  set_msr(msr);
  return (POK_ERRNO_OK);
}
```

4.1.2.5 pok_ret_t pok_arch_preempt_enable ( )

Enable interruptions

Definition at line 63 of file arch.c.

```
{
  unsigned int msr;

  msr = get_msr();
  msr |= MSR_EE;
  set_msr(msr);

  return (POK_ERRNO_OK);
}
```

4.1.2.6 void pok_arch_space_init (void)

Initilize MMU tables.

Definition at line 132 of file space.c.

```
{
  uint32_t sdr1;

  pt_base = 0;
  pt_mask = 0x3ff;

  sdr1 = pt_base | (pt_mask >> 10);
  asm volatile ("mtsdr1 %0" : : "r"(sdr1));
}
```

4.1.2.7 uint32_t pok_thread_stack_addr (const uint8_t *partition_id*, const uint32_t *local_thread_id*)

Returns the stack address for a the thread number N in a partition.

- partition_id indicates the partition that contains the thread.
- local_thread_id the thread-id of the thread inside the partition.

Returns

the stack address of the thread.

Definition at line 92 of file arch.c.

```
{
   return pok_partitions[partition_id].size - 16 - (local_thread_id *
      POK_USER_STACK_SIZE);
}
```

4.2 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/sparc/arch.c File Reference

`#include <types.h> #include <errno.h> #include <core/partition.-h> #include "traps.h" #include "space.h" #include "psr.h" #include "sparc_conf.h" #include "syscalls.h"`

Functions

- pok_ret_t pok_arch_init ()
- pok_ret_t pok_arch_preempt_disable ()
- pok_ret_t pok_arch_preempt_enable ()
- pok_ret_t pok_arch_idle ()
- pok_ret_t pok_arch_event_register (uint8_t vector, void(∗handler)(void))
- uint32_t pok_thread_stack_addr (const uint8_t partition_id, const uint32_t local-_thread_id)

4.2.1 Detailed Description

Author

Fabien Chouteau

Definition in file arch.c.

4.2.2 Function Documentation

4.2.2.1 pok_ret_t pok_arch_event_register (uint8_t *vector,* void(∗)(void) *handler*)

Attach the handler to the given trap number (vector).

See also

pok_sparc_isr

Definition at line 75 of file arch.c.

```
{
  if (pok_sparc_isr[vector] == NULL)
  {
    pok_sparc_isr[vector] = handler;
    return (POK_ERRNO_OK);
  }
  else
  {
    return (POK_ERRNO_UNAVAILABLE);
  }
}
```

4.2.2.2 pok_ret_t pok_arch_idle ( )

Function that do nothing. Useful for the idle task for example.

Definition at line 60 of file arch.c.

```
{
  while (1)
  {
    /* Leon3 Only ? */
    asm volatile ("wr %g0, %asr19");
  }

   return (POK_ERRNO_OK);
}
```

4.2.2.3 pok_ret_t pok_arch_init ( )

Initialize all SPARC managers (traps, syscalls, space).

Definition at line 34 of file arch.c.

```
{
  traps_init();
  psr_disable_interupt();
  psr_enable_traps();

  pok_arch_space_init();
  pok_syscalls_init();

  return (POK_ERRNO_OK);
}
```

4.2.2.4 pok_ret_t pok_arch_preempt_disable ( )

Disable interruptions

Definition at line 46 of file arch.c.

```
{
  psr_disable_interupt();

  return (POK_ERRNO_OK);
}
```

4.2.2.5 pok_ret_t pok_arch_preempt_enable ( )

Enable interruptions

Definition at line 53 of file arch.c.

```
{
  psr_enable_interupt();

  return (POK_ERRNO_OK);
}
```

4.2.2.6 uint32_t pok_thread_stack_addr (const uint8_t *partition_id*, const uint32_t *local_thread_id*)

Compute the stack adress for the given thread.

Definition at line 91 of file arch.c.

```
{
  return pok_partitions[partition_id].size - (local_thread_id *
      POK_USER_STACK_SIZE);
}
```

4.3 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/x86/arch.c File Reference

Provides generic architecture interface for x86 architecture.

`#include <errno.h>` `#include <core/partition.h>` `#include "event.h"` `#include "gdt.h"`

Functions

- pok_ret_t pok_arch_init ()
- pok_ret_t pok_arch_preempt_disable ()

- pok_ret_t pok_arch_preempt_enable ()
- pok_ret_t pok_arch_idle ()
- pok_ret_t pok_arch_event_register (uint8_t vector, void(∗handler)(void))
- uint32_t pok_thread_stack_addr (const uint8_t partition_id, const uint32_t local-_thread_id)

4.3.1 Detailed Description

Provides generic architecture interface for x86 architecture.

Author

Julian Pidancet
Julien Delange

Definition in file arch.c.

4.3.2 Function Documentation

4.3.2.1 pok_ret_t pok_arch_event_register (uint8_t *vector,* void(∗)(void) *handler*)

Register an event (for example, an interruption)

Attach the handler to the given trap number (vector).

See also

pok_sparc_isr

Definition at line 60 of file arch.c.

```
{
   pok_idt_set_gate (vector,
                     GDT_CORE_CODE_SEGMENT << 3,
                     (uint32_t)handler,
                     IDTE_TRAP,
                     3);

   return (POK_ERRNO_OK);
}
```

4.3.2.2 pok_ret_t pok_arch_idle ( )

Function that do nothing. Useful for the idle task for example.

Definition at line 50 of file arch.c.

```
{
   while (1)
   {
      asm ("hlt");
   }

   return (POK_ERRNO_OK);
}
```

4.3.2.3 pok_ret_t pok_arch_init ( )

Function that initializes architecture concerns.

Initialize all SPARC managers (traps, syscalls, space).

Definition at line 30 of file arch.c.

```
{
  pok_gdt_init ();
  pok_event_init ();

  return (POK_ERRNO_OK);
}
```

4.3.2.4 pok_ret_t pok_arch_preempt_disable ( )

Disable interruptions

Definition at line 38 of file arch.c.

```
{
  asm ("cli");
  return (POK_ERRNO_OK);
}
```

4.3.2.5 pok_ret_t pok_arch_preempt_enable ( )

Enable interruptions

Definition at line 44 of file arch.c.

```
{
  asm ("sti");
  return (POK_ERRNO_OK);
}
```

4.3.2.6 uint32_t pok_thread_stack_addr (const uint8_t *partition_id*, const uint32_t *local_thread_id*)

Returns the stack address for a the thread number N in a partition.

- partition_id indicates the partition that contains the thread.
- local_thread_id the thread-id of the thread inside the partition.

Returns

the stack address of the thread.

Compute the stack adress for the given thread.

Definition at line 72 of file arch.c.

```
{
   return pok_partitions[partition_id].size - 4 - (local_thread_id *
      POK_USER_STACK_SIZE);
}
```

4.4 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/ppc/msr.h File Reference

Defines

- #define MSR_DR (1 $<<$ 4)
- #define MSR_IR (1 $<<$ 5)
- #define MSR_IP (1 $<<$ 6)
- #define MSR_PR (1 $<<$ 14)
- #define MSR_EE (1 $<<$ 15)

4.4.1 Define Documentation

4.4.1.1 #define MSR_DR (1 $<<$ 4)

Definition at line 21 of file msr.h.

4.4.1.2 #define MSR_EE (1 $<<$ 15)

Definition at line 26 of file msr.h.

4.4.1.3 #define MSR_IP (1 $<<$ 6)

Definition at line 23 of file msr.h.

4.4.1.4 #define MSR_IR (1 $<<$ 5)

Definition at line 22 of file msr.h.

4.4.1.5 #define MSR_PR (1 << 14)

Definition at line 25 of file msr.h.

4.5 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/ppc/prep/bsp.c File Reference

`#include <errno.h> #include <arch.h> #include "cons.h"`

Functions

- int pok_bsp_init (void)
- void * pok_bsp_mem_alloc (size_t sz)

Variables

- char _end []

4.5.1 Function Documentation

4.5.1.1 int pok_bsp_init (void)

Definition at line 22 of file bsp.c.

```
{
   pok_cons_init ();

   return (POK_ERRNO_OK);
}
```

4.5.1.2 void* pok_bsp_mem_alloc (size_t *sz*)

Definition at line 34 of file bsp.c.

```
{
  char *res;

  res = (char *)(((unsigned int)heap_end + 4095) & ~4095);
  heap_end = res + sz;
  return res;
}
```

4.5.2 Variable Documentation

4.5.2.1 char _end[ ]

4.6 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/sparc/leon3/bsp.c File Reference

`#include <errno.h> #include <arch.h> #include <core/debug.h> #include "cons.h" #include "sparc_conf.h"`

Functions

- int pok_bsp_init (void)
- void ∗ pok_bsp_mem_alloc (size_t sz)

Variables

- char _end [ ]

4.6.1 Detailed Description

Author

Fabien Chouteau

Definition in file bsp.c.

4.6.2 Function Documentation

4.6.2.1 int pok_bsp_init (void)

Definition at line 32 of file bsp.c.

```
{
  pok_cons_init ();
  return (POK_ERRNO_OK);
}
```

4.6.2.2 void∗ pok_bsp_mem_alloc (size_t *sz*)

Used for partition allocation. For SPARC support, all partitions are aligned on page size and all partition sizes have to be less than page size.

See also

SPARC_PAGE_SIZE

Definition at line 44 of file bsp.c.

```
{
  char *res;

  /* Aligned on page size */
  res = (char *)(((uint32_t)heap_end + SPARC_PAGE_SIZE) & ~ (SPARC_PAGE_SIZE -
      1));
  heap_end = res + sz;
  return res;
}
```

4.6.3 Variable Documentation

4.6.3.1 char _end[ ]

4.7 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/x86/x86-qemu/bsp.c File Reference

`#include <errno.h>` `#include <arch.h>` `#include "cons.h"` ×
`#include "pm.h"` `#include "pit.h"` `#include "pic.h"`

Functions

- pok_ret_t pok_bsp_init (void)
- pok_ret_t pok_bsp_irq_acknowledge (uint8_t irq)
- pok_ret_t pok_bsp_irq_register (uint8_t irq, void(*handler)(void))
- void * pok_bsp_mem_alloc (size_t size)
- pok_ret_t pok_bsp_time_init ()

4.7.1 Function Documentation

4.7.1.1 pok_ret_t pok_bsp_init (void)

Definition at line 26 of file bsp.c.

```
{
   pok_cons_init ();
   pok_pm_init ();
   pok_pic_init ();

   return (POK_ERRNO_OK);
}
```

4.7.1.2 pok_ret_t pok_bsp_irq_acknowledge (uint8_t *irq*)

Definition at line 35 of file bsp.c.

```
{
   pok_pic_eoi (irq);

   return (POK_ERRNO_OK);
}
```

4.7.1.3 pok_ret_t pok_bsp_irq_register (uint8_t *irq*, void(∗)(void) *handler*)

Definition at line 42 of file bsp.c.

```
{
   pok_pic_unmask (irq);

   pok_arch_event_register (32 + irq, handler);

   return (POK_ERRNO_OK);
}
```

4.7.1.4 void∗ pok_bsp_mem_alloc (size_t *size*)

Allocate data. At this time, the pok_pm_sbrk function only increment size each time we allocate memory and was not designed to free previously allocated memory.

Definition at line 58 of file bsp.c.

```
{
   return ((void *)pok_pm_sbrk(size));
}
```

4.7.1.5 pok_ret_t pok_bsp_time_init ( )

Init time. *freq* is the frequency of the oscillator.

Definition at line 67 of file bsp.c.

```
{
   return (pok_x86_qemu_timer_init ());
}
```

4.8 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/ppc/prep/cons.c File Reference

`#include <errno.h> #include "ioports.h" #include <libc.h> #include <core/debug.h> #include <core/cons.h> #include "cons.h"`

Functions

- int pok_cons_init (void)

4.8.1 Function Documentation

4.8.1.1 int pok_cons_init (void)

Definition at line 68 of file cons.c.

```
{
   return 0;
}
```

4.9 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/sparc/leon3/cons.c File Reference

Leon3 UART driver.

`#include <errno.h>` `#include "ioports.h"` `#include <libc.h>` `#include <core/debug.h>` `#include <core/cons.h>` `#include "cons.h"`

Functions

- int pok_cons_init (void)

4.9.1 Detailed Description

Leon3 UART driver.

Author

Fabien Chouteau

Definition in file cons.c.

4.9.2 Function Documentation

4.9.2.1 int pok_cons_init (void)

Definition at line 62 of file cons.c.

```
{
  return 0;
}
```

4.10 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/x86/x86-qemu/cons.c File Reference

`#include <errno.h> #include <arch/x86/ioports.h> #include <libc.h> #include <core/debug.h> #include <core/cons.h>` ×
`#include "cons.h"`

Functions

- int pok_cons_init (void)

4.10.1 Function Documentation

4.10.1.1 int pok_cons_init (void)

Definition at line 235 of file cons.c.

```
{
   return 0;
}
```

4.11 /home/matias/projet/stage-pok/pok/trunk/kernel/core/cons.c File Reference

4.12 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/ppc/prep/cons.h File Reference

Functions

- int pok_cons_init (void)

4.12.1 Function Documentation

4.12.1.1 int pok_cons_init (void)

Definition at line 68 of file cons.c.

```
{
   return 0;
}
```

4.13 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/sparc/leon3/cons.h File Reference

Defines

- #define UART_STATUS_DR 0x00000001
- #define UART_STATUS_TSE 0x00000002
- #define UART_STATUS_THE 0x00000004
- #define UART_STATUS_BR 0x00000008
- #define UART_STATUS_OE 0x00000010
- #define UART_STATUS_PE 0x00000020
- #define UART_STATUS_FE 0x00000040
- #define UART_STATUS_ERR 0x00000078
- #define UART_CTRL_RE 0x00000001
- #define UART_CTRL_TE 0x00000002
- #define UART_CTRL_RI 0x00000004
- #define UART_CTRL_TI 0x00000008
- #define UART_CTRL_PS 0x00000010
- #define UART_CTRL_PE 0x00000020
- #define UART_CTRL_FL 0x00000040
- #define UART_CTRL_LB 0x00000080
- #define UART_DATA_OFFSET 0x0
- #define UART_STAT_OFFSET 0x4
- #define UART_CTRL_OFFSET 0x8
- #define UART_SCALER_OFFSET 0xc
- #define UART1 0x80000100

Functions

- int pok_cons_init (void)

4.13.1 Detailed Description

Author

Fabien Chouteau

Definition in file cons.h.

4.13.2 Define Documentation

4.13.2.1 #define UART1 0x80000100

First Leon3 UART IO adress

Definition at line 48 of file cons.h.

4.13.2.2 #define UART_CTRL_FL 0x00000040

Flow control enable

Definition at line 40 of file cons.h.

4.13.2.3 #define UART_CTRL_LB 0x00000080

Loop Back enable

Definition at line 41 of file cons.h.

4.13.2.4 #define UART_CTRL_OFFSET 0x8

Control register offset

Definition at line 45 of file cons.h.

4.13.2.5 #define UART_CTRL_PE 0x00000020

Parity enable

Definition at line 39 of file cons.h.

4.13.2.6 #define UART_CTRL_PS 0x00000010

Parity select

Definition at line 38 of file cons.h.

4.13.2.7 #define UART_CTRL_RE 0x00000001

Receiver enable

Definition at line 34 of file cons.h.

4.13.2.8 #define UART_CTRL_RI 0x00000004

Receiver interrupt enable

Definition at line 36 of file cons.h.

4.13.2.9 #define UART_CTRL_TE 0x00000002

Transmitter enable

Definition at line 35 of file cons.h.

4.13.2.10 #define UART_CTRL_TI 0x00000008

Transmitter interrupt enable

Definition at line 37 of file cons.h.

4.13.2.11 #define UART_DATA_OFFSET 0x0

Data register offset

Definition at line 43 of file cons.h.

4.13.2.12 #define UART_SCALER_OFFSET 0xc

Scaler register offset

Definition at line 46 of file cons.h.

4.13.2.13 #define UART_STAT_OFFSET 0x4

Stat register offset

Definition at line 44 of file cons.h.

4.13.2.14 #define UART_STATUS_BR 0x00000008

Break Error

Definition at line 28 of file cons.h.

4.13.2.15 #define UART_STATUS_DR 0x00000001

Data Ready

Definition at line 25 of file cons.h.

4.13.2.16 #define UART_STATUS_ERR 0x00000078

Error Mask

Definition at line 32 of file cons.h.

4.13.2.17 #define UART_STATUS_FE 0x00000040

RX Framing Error

Definition at line 31 of file cons.h.

4.13.2.18 #define UART_STATUS_OE 0x00000010

RX Overrun Error

Definition at line 29 of file cons.h.

4.13.2.19 #define UART_STATUS_PE 0x00000020

RX Parity Error

Definition at line 30 of file cons.h.

4.13.2.20 #define UART_STATUS_THE 0x00000004

TX Hold Register Empty

Definition at line 27 of file cons.h.

4.13.2.21 #define UART_STATUS_TSE 0x00000002

TX Send Register Empty

Definition at line 26 of file cons.h.

4.13.3 Function Documentation

4.13.3.1 int pok_cons_init (void)

Definition at line 68 of file cons.c.

```
{
   return 0;
}
```

4.14 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/x86/x86-qemu/cons.h File Reference

Functions

- int pok_cons_init (void)

4.14.1 Function Documentation

4.14.1.1 int pok_cons_init (void)

Definition at line 68 of file cons.c.

```
{
    return 0;
}
```

4.15 /home/matias/projet/stage-pok/pok/trunk/kernel/include/core/cons.h File Reference

4.16 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/ppc/prep/ioports.h File Reference

Defines

- #define POK_PREP_IOBASE 0x80000000
- #define outb(port, data) ∗((volatile unsigned char ∗)(POK_PREP_IOBASE + port)) = data
- #define inb(port) ∗((volatile unsigned char ∗)(POK_PREP_IOBASE + port))

4.16.1 Define Documentation

4.16.1.1 #define **inb**(*port*) ∗((volatile unsigned char ∗)(POK_PREP_IOBASE + port))

Definition at line 26 of file ioports.h.

4.16.1.2 #define **outb**(*port,* *data*) ∗((volatile unsigned char ∗)(POK_PREP_IOBASE + port)) = data

Definition at line 23 of file ioports.h.

4.16.1.3 #define **POK_PREP_IOBASE** 0x80000000

Definition at line 21 of file ioports.h.

4.17 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/sparc/leon3/ioports.h File Reference

SPARC "ioports". Use MMU bypass to access IO memory.

`#include <types.h>` `#include "sparc_conf.h"`

4.17.1 Detailed Description

SPARC "ioports". Use MMU bypass to access IO memory.

Author

Fabien Chouteau

Definition in file ioports.h.

4.18 /home/matias/projet/stage-pok/pok/trunk/kernel/include/arch/x86/ioports.h File Reference

```
#include <core/syscall.h>
```

Defines

- #define outb(port, data)
- #define inb(port)
- #define outl(port, data)
- #define inl(port)

4.18.1 Define Documentation

4.18.1.1 #define inb(*port*)

Value:

```
({                                                                  \
  unsigned char res;                                                \
  asm volatile ("inb %w1,%0"                                        \
                :"=a" (res)                                         \
                :"d" (port));                                       \
  res;                                                              \
})
```

Definition at line 28 of file ioports.h.

4.18.1.2 #define inl(*port*)

Value:

```
({                                                                  \
  unsigned int res;                                                 \
  asm volatile ("inl %w1,%0"                                        \
                :"=a" (res)                                         \
                :"d" (port));                                       \
  res;                                                              \
})
```

Definition at line 42 of file ioports.h.

4.18.1.3 #define outb(*port,* *data*)

Value:

```
asm volatile ("outb %b0,%w1"                                      \
                :                                                 \
                :"a" (data),"d" (port))
```

Definition at line 23 of file ioports.h.

4.18.1.4 #define outl(*port,* *data*)

Value:

```
asm volatile ("outl %0,%w1"                                        \
                :                                                  \
                :"a" (data),"d" (port))
```

Definition at line 37 of file ioports.h.

4.19 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/ppc/space.c File Reference

`#include <types.h> #include <errno.h> #include <libc.h> #include <bsp.h> #include <core/sched.h> #include <arch.h> #include "thread.h" #include "msr.h"`

Data Structures

- struct pok_space
- struct ppc_pte_t

Defines

- #define KERNEL_STACK_SIZE 8192
- #define PPC_SR_KP (1 << 29)
- #define PPC_SR_Ks (1 << 30)
- #define PPC_SR_T (1 << 31)
- #define PPC_PTE_V (1 << 31)
- #define POK_PAGE_SIZE (1 << 12)
- #define POK_PAGE_MASK (~(POK_PAGE_SIZE - 1))
- #define PPC_PTE_H (1 << 6)
- #define PPC_PTE_R (1 << 8)

- #define PPC_PTE_C (1 << 7)
- #define PPC_PTE_W (1 << 6)
- #define PPC_PTE_I (1 << 5)
- #define PPC_PTE_M (1 << 4)
- #define PPC_PTE_G (1 << 3)
- #define PPC_PTE_PP_NO 0
- #define PPC_PTE_PP_RO 1
- #define PPC_PTE_PP_RW 2

Functions

- pok_ret_t pok_create_space (uint8_t partition_id, uint32_t addr, uint32_t size)
- pok_ret_t pok_space_switch (uint8_t old_partition_id, uint8_t new_partition_id)
- uint32_t pok_space_base_vaddr (uint32_t addr)
- void pok_arch_rfi (void)
- uint32_t pok_space_context_create (uint8_t partition_id, uint32_t entry_rel, uint32_t stack_rel, uint32_t arg1, uint32_t arg2)
- void pok_arch_space_init (void)
- void pok_arch_isi_int (uint32_t pc, uint32_t msr)
- void pok_arch_dsi_int (uint32_t dar, uint32_t dsisr)

Variables

- struct pok_space spaces [POK_CONFIG_NB_PARTITIONS]

4.19.1 Define Documentation

4.19.1.1 #define **KERNEL_STACK_SIZE** 8192

Definition at line 28 of file space.c.

4.19.1.2 #define **POK_PAGE_MASK** (~(POK_PAGE_SIZE - 1))

Definition at line 120 of file space.c.

4.19.1.3 #define **POK_PAGE_SIZE** (1 << 12)

Definition at line 119 of file space.c.

4.19.1.4 #define **PPC_PTE_C** (1 << 7)

Definition at line 123 of file space.c.

4.19.1.5 #define PPC_PTE_G (1 $<<$ 3)

Definition at line 127 of file space.c.

4.19.1.6 #define PPC_PTE_H (1 $<<$ 6)

Definition at line 121 of file space.c.

4.19.1.7 #define PPC_PTE_I (1 $<<$ 5)

Definition at line 125 of file space.c.

4.19.1.8 #define PPC_PTE_M (1 $<<$ 4)

Definition at line 126 of file space.c.

4.19.1.9 #define PPC_PTE_PP_NO 0

Definition at line 128 of file space.c.

4.19.1.10 #define PPC_PTE_PP_RO 1

Definition at line 129 of file space.c.

4.19.1.11 #define PPC_PTE_PP_RW 2

Definition at line 130 of file space.c.

4.19.1.12 #define PPC_PTE_R (1 $<<$ 8)

Definition at line 122 of file space.c.

4.19.1.13 #define PPC_PTE_V (1 $<<$ 31)

Definition at line 118 of file space.c.

4.19.1.14 #define PPC_PTE_W (1 $<<$ 6)

Definition at line 124 of file space.c.

4.19.1.15 #define PPC_SR_KP (1 $<<$ 29)

Definition at line 30 of file space.c.

4.19.1.16 #define PPC_SR_Ks (1 $<<$ 30)

Definition at line 31 of file space.c.

4.19.1.17 #define PPC_SR_T (1 $<<$ 31)

Definition at line 32 of file space.c.

4.19.2 Function Documentation

4.19.2.1 void pok_arch_dsi_int (uint32_t *dar*, uint32_t *dsisr*)

Definition at line 203 of file space.c.

```
{
#ifdef POK_NEEDS_DEBUG
  printf("dsi_int: part=%d, dar=%x dsisr=%x\n",
         pok_current_partition, dar, dsisr);
#endif

  if (dsisr & (1 << 30))
    {
      /* Page fault  */
      if (dar < spaces[pok_current_partition].size)
        {
          uint32_t vaddr = dar & POK_PAGE_MASK;
          uint32_t v;
          v = (spaces[pok_current_partition].phys_base + vaddr) & POK_PAGE_MASK
      ;
          v |= PPC_PTE_R | PPC_PTE_C | PPC_PTE_PP_RW;
          pok_insert_pte (pok_current_partition, vaddr, v);
          return;
        }
    }
#ifdef POK_NEEDS_DEBUG
   printf("[DEBUG] Infinite loop in pok_arch_dsi_int\n");
#endif
  while (1)
    ;
}
```

4.19.2.2 void pok_arch_isi_int (uint32_t *pc*, uint32_t *msr*)

Definition at line 168 of file space.c.

```
{
```

```
#ifdef POK_NEEDS_DEBUG
  printf("isi_int: part=%d, pc=%x msr=%x\n",
         pok_current_partition, pc, msr);

  if (msr & ((1 << 28) | (1 << 27)))
  {
    printf (" Bad access\n");
  }
#endif

  if (msr & (1 << 30))
    {
      /* Page fault  */
      if (pc < spaces[pok_current_partition].size)
        {
          uint32_t vaddr = pc & POK_PAGE_MASK;
          uint32_t v;
          v = (spaces[pok_current_partition].phys_base + vaddr) & POK_PAGE_MASK
      ;
          v |= PPC_PTE_R | PPC_PTE_C | PPC_PTE_PP_RW;
          pok_insert_pte (pok_current_partition, vaddr, v);
          return;
        }
    }

#ifdef POK_NEEDS_DEBUG
   printf("[DEBUG] Infinite loop in pok_arch_isi_int\n");
#endif

  while (1)
    ;
}
```

4.19.2.3 void pok_arch_rfi (void)

4.19.2.4 void pok_arch_space_init (void)

Definition at line 132 of file space.c.

```
{
  uint32_t sdr1;

  pt_base = 0;
  pt_mask = 0x3ff;

  sdr1 = pt_base | (pt_mask >> 10);
  asm volatile ("mtsdr1 %0" : : "r"(sdr1));
}
```

4.19.2.5 pok_ret_t pok_create_space (uint8_t *partition_id*, uint32_t *addr*, uint32_t *size*)

Definition at line 42 of file space.c.

```
{
```

```
#ifdef POK_NEEDS_DEBUG
  printf ("pok_create_space: %d: %x %x\n", partition_id, addr, size);
#endif
  spaces[partition_id].phys_base = addr;
  spaces[partition_id].size = size;

  return (POK_ERRNO_OK);
}
```

4.19.2.6 uint32_t pok_space_base_vaddr (uint32_t *addr*)

Definition at line 64 of file space.c.

```
{
   (void) addr;
   return (0);
}
```

4.19.2.7 uint32_t pok_space_context_create (uint8_t *partition_id,* uint32_t *entry_rel,* uint32_t *stack_rel,* uint32_t *arg1,* uint32_t *arg2*)

Create a new context in the given space

Definition at line 72 of file space.c.

```
{
  context_t* ctx;
  volatile_context_t* vctx;
  char*      stack_addr;
  (void) partition_id;

  stack_addr = pok_bsp_mem_alloc (KERNEL_STACK_SIZE);

  vctx = (volatile_context_t *)
    (stack_addr + KERNEL_STACK_SIZE - sizeof (volatile_context_t));
  ctx = (context_t *)((char *)vctx - sizeof (context_t) + 8);

  memset (ctx, 0, sizeof (*ctx));
  memset (vctx, 0, sizeof (*vctx));

  vctx->r3     = arg1;
  vctx->r4     = arg2;
  vctx->sp     = stack_rel - 12;
  vctx->srr0   = entry_rel;
  vctx->srr1   = MSR_EE | MSR_IP | MSR_DR | MSR_IR | MSR_PR;
  ctx->lr      = (uint32_t) pok_arch_rfi;

  ctx->sp      = (uint32_t) &vctx->sp;

#ifdef POK_NEEDS_DEBUG
  printf ("space_context_create %d: entry=%x stack=%x arg1=%x arg2=%x ksp=%x\n"
      ,
          partition_id, entry_rel, stack_rel, arg1, arg2, &vctx->sp);
#endif

  return (uint32_t)ctx;
}
```

4.19.2.8 pok_ret_t pok_space_switch (uint8_t *old_partition_id,* uint8_t *new_partition_id*)

Switch from one space to another

Definition at line 55 of file space.c.

```
{
  (void) old_partition_id;
  /* printf ("space_switch %u -> %u\n", old_partition_id, new_partition_id); */
  asm volatile ("mtsr %0,%1" : : "r"(0), "r"(PPC_SR_KP | new_partition_id));
  return (POK_ERRNO_OK);
}
```

4.19.3 Variable Documentation

4.19.3.1 struct pok_space spaces[POK_CONFIG_NB_PARTITIONS]

Definition at line 40 of file space.c.

4.20 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/sparc/space.c File Reference

Memory management in SPARC.

`#include <types.h> #include <errno.h> #include <libc.h> x #include <bsp.h> #include <core/sched.h> #include <arch.- h> #include "thread.h" #include "space.h" #include "sparc_- conf.h" #include "context_offset.h" #include "ioports.h"`

Data Structures

- struct pok_space

Defines

- #define KERNEL_STACK_SIZE 8192

Functions

- ptd mmu_contexts_tab[POK_CONFIG_NB_PARTITIONS] __attribute__- ((aligned(POK_CONFIG_NB_PARTITIONS ∗sizeof(ptd))))
- ptd mmu_level1_tab[POK_CONFIG_NB_PARTITIONS][MM_LVL1_ENTRIES_- NBR] __attribute__ ((aligned(MM_LVL1_ENTRIES_NBR ∗sizeof(ptd))))
- pte mmu_level2_tab[POK_CONFIG_NB_PARTITIONS][MM_LVL2_ENTRIES_- NBR] __attribute__ ((aligned(MM_LVL2_ENTRIES_NBR ∗sizeof(pte))))

- pok_ret_t pok_create_space (uint8_t partition_id, uint32_t addr, uint32_t size)
- pok_ret_t pok_space_switch (uint8_t old_partition_id, uint8_t new_partition_id)
- uint32_t pok_space_base_vaddr (uint32_t addr)
- uint32_t pok_space_context_create (uint8_t id, uint32_t entry_rel, uint32_t stack-_rel, uint32_t arg1, uint32_t arg2)
- void pok_arch_space_init (void)

Variables

- struct pok_space spaces [POK_CONFIG_NB_PARTITIONS]

4.20.1 Detailed Description

Memory management in SPARC.

Author

Fabien Chouteau

Definition in file space.c.

4.20.2 Define Documentation

4.20.2.1 #define KERNEL_STACK_SIZE 8192

Definition at line 36 of file space.c.

4.20.3 Function Documentation

4.20.3.1 ptd mmu_contexts_tab [POK_CONFIG_NB_PARTITIONS] __attribute__ ((aligned(POK_CONFIG_NB_PARTITIONS *sizeof(ptd))))

MMU contexts table. (cf SPARC V8 Manual, page 243)

4.20.3.2 ptd mmu_level1_tab [POK_CONFIG_NB_PARTITIONS][MM_LVL1_ENTRIES-_NBR] __attribute__ ((aligned(MM_LVL1_ENTRIES_NBR *sizeof(ptd))))

MMU level 1 table. (cf SPARC V8 Manual, page 243)

4.20.3.3 pte mmu_level2_tab [POK_CONFIG_NB_PARTITIONS][MM_LVL2_ENTRIES-_NBR] __attribute__ ((aligned(MM_LVL2_ENTRIES_NBR *sizeof(pte))))

MMU level 2 table. (cf SPARC V8 Manual, page 243)

4.20.3.4 void pok_arch_space_init (void)

Initilize MMU tables.

Definition at line 159 of file space.c.

```
{
  int i = 0;
  int j = 0;

  for (i = 0; i < POK_CONFIG_NB_PARTITIONS; i++)
    mmu_contexts_tab[i] = MM_ET_INVALID;

  for (i = 0; i < POK_CONFIG_NB_PARTITIONS; i++)
  {
    mmu_contexts_tab[i] = (unsigned int)&(mmu_level1_tab[i]) >> 4 | MM_ET_PTD;

    for (j = 0; j < MM_LVL1_ENTRIES_NBR; j++)
    {
      mmu_level1_tab[i][j] = MM_ET_INVALID;
    }

    for (j = 0; j < MM_LVL2_ENTRIES_NBR; j++)
    {
      mmu_level2_tab[i][j] = MM_ET_INVALID;
    }
  }

  unsigned int kernel_pte = mm_index1(SPARC_RAM_ADDR);

  /* the kernel code is always mapped on a 16Mb page (including all partitions)
       */
  for (i = 0; i < POK_CONFIG_NB_PARTITIONS; i++)
  {
    mmu_level1_tab[i][kernel_pte] = (SPARC_RAM_ADDR >> 4) | MM_ACC_S_RWE |
      MM_ET_PTE | MM_CACHEABLE;
  }


  /* set context table */
  asm volatile ("sta %0, [%1] %2;\n"
                : /* no output */
                : "r" (((unsigned int) mmu_contexts_tab) >> 4), "r" (
      MMU_CTXTBL_PTR), "i" (ASI_M_MMUREGS)
                : "memory");

  /* set context number */
  pok_space_switch(0, 0);

  asm volatile ("flush\n"
                "sta %0, [%1] %2;\n"
                : /* no output */
                : "r" (0x1), "r" (MMU_CTRL_REG), "i" (ASI_M_MMUREGS)
                : "memory");


#ifdef POK_NEEDS_DEBUG
    printf ("pok_arch_space_init: ctx nbr=%u\n", POK_CONFIG_NB_PARTITIONS);
#endif
}
```

4.20.3.5 pok_ret_t pok_create_space (uint8_t *partition_id,* uint32_t *addr,* uint32_t *size*)

Set ptd and pte for the given partition.

Definition at line 70 of file space.c.

```
{
  if (size > SPARC_PARTITION_SIZE)
  {
#ifdef POK_NEEDS_DEBUG
    printf ("pok_create_space: %d: partition size too big 0x%x\n", partition_id
      , size);
#endif
    return (POK_ERRNO_SIZE);
  }

  if ((addr & (SPARC_PAGE_SIZE - 1)) != 0)
  {
#ifdef POK_NEEDS_DEBUG
    printf ("pok_create_space: %d: partition address not aligned 0x%x\n",
      partition_id, addr);
#endif
    return (POK_ERRNO_EFAULT);
  }
#ifdef POK_NEEDS_DEBUG
  printf ("pok_create_space: %d: %x %x\n", partition_id, addr, size);
#endif
  spaces[partition_id].phys_base = addr;
  spaces[partition_id].size = size;

  unsigned int as_ptd = mm_index1(SPARC_PARTITION_BASE_VADDR);
  unsigned int as_pte = mm_index2(SPARC_PARTITION_BASE_VADDR);

  mmu_level1_tab[partition_id][as_ptd] = ((unsigned int) &(mmu_level2_tab[
      partition_id]) >> 4) | MM_ET_PTD;
  /* partition as */
  mmu_level2_tab[partition_id][as_pte] = ((addr) >> 4) | MM_ACC_RWE | MM_ET_PTE
       | MM_CACHEABLE;

  return (POK_ERRNO_OK);
}
```

4.20.3.6 uint32_t pok_space_base_vaddr (uint32_t *addr*)

Returns

partition virtual base adress.

See also

SPARC_PARTITION_BASE_VADDR

Definition at line 125 of file space.c.

```
{
   (void) addr;
   return (SPARC_PARTITION_BASE_VADDR);
}
```

4.20.3.7 uint32_t pok_space_context_create (uint8_t *id*, uint32_t *entry_rel*, uint32_t *stack_rel*, uint32_t *arg1*, uint32_t *arg2*)

Initilize thread stack.

Definition at line 134 of file space.c.

```
{
  uint32_t ctx = spaces[id].phys_base + stack_rel - 0x40;

  outw(ctx - RESTORE_CNT_OFFSET, 1); /* Only 1 register window needed */
  outw(ctx - PC_OFFSET, entry_rel);
  outw(ctx - NPC_OFFSET, entry_rel + 4);
  outw(ctx - I0_OFFSET, arg1);
  outw(ctx - I1_OFFSET, arg2);

#ifdef POK_NEEDS_DEBUG
  printf ("space_context_create part_id=%d entry=%x stack=%x arg1=%x arg2=%x\n"
      ,
          id, entry_rel, stack_rel, arg1, arg2);
#endif

  return SPARC_PARTITION_BASE_VADDR + stack_rel - 0x40;
}
```

4.20.3.8 pok_ret_t pok_space_switch (uint8_t *old_partition_id*, uint8_t *new_partition_id*)

Switch adress space in MMU (context register).

Definition at line 108 of file space.c.

```
{
  (void) old_partition_id;

  asm volatile ("flush\n"
                "sta %0, [%1] %2;\n"
                : /* no output */
                : "r" (new_partition_id), "r" (MMU_CTX_REG), "i" (ASI_M_MMUREGS
      )
                : "memory");
  return (POK_ERRNO_OK);
}
```

4.20.4 Variable Documentation

4.20.4.1 struct pok_space spaces[POK_CONFIG_NB_PARTITIONS]

Definition at line 47 of file space.c.

4.21 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/x86/space.c File Reference

Handle address spaces.

```
#include <types.h> #include <errno.h> #include <libc.h>
#include <bsp.h> #include <arch.h> #include <arch/x86/interrupt.-
h> #include "gdt.h" #include "tss.h" #include "space.h"
```

Defines

- #define KERNEL_STACK_SIZE 8192

Functions

- pok_ret_t pok_create_space (uint8_t partition_id, uint32_t addr, uint32_t size)
- pok_ret_t pok_space_switch (uint8_t old_partition_id, uint8_t new_partition_id)
- uint32_t pok_space_base_vaddr (uint32_t addr)
- uint32_t pok_space_context_create (uint8_t partition_id, uint32_t entry_rel, uint32_t stack_rel, uint32_t arg1, uint32_t arg2)
- void pok_dispatch_space (uint8_t partition_id, uint32_t user_pc, uint32_t user_-sp, uint32_t kernel_sp, uint32_t arg1, uint32_t arg2)

4.21.1 Detailed Description

Handle address spaces.

Author

Julian Pidancet

Definition in file space.c.

4.21.2 Define Documentation

4.21.2.1 #define KERNEL_STACK_SIZE 8192

Definition at line 38 of file space.c.

4.21.3 Function Documentation

4.21.3.1 pok_ret_t pok_create_space (uint8_t *partition_id,* uint32_t *addr,* uint32_t *size*)

Set ptd and pte for the given partition.

Definition at line 40 of file space.c.

```
{
   gdt_set_segment (GDT_PARTITION_CODE_SEGMENT (partition_id),
         addr, size, GDTE_CODE, 3);

   gdt_set_segment (GDT_PARTITION_DATA_SEGMENT (partition_id),
         addr, size, GDTE_DATA, 3);

   return (POK_ERRNO_OK);
}
```

4.21.3.2 void pok_dispatch_space (uint8_t *partition_id*, uint32_t *user_pc*, uint32_t *user_sp*, uint32_t *kernel_sp*, uint32_t *arg1*, uint32_t *arg2*)

Definition at line 114 of file space.c.

```
{
   interrupt_frame   ctx;
   uint32_t          code_sel;
   uint32_t          data_sel;
   uint32_t          sp;

   code_sel = GDT_BUILD_SELECTOR (GDT_PARTITION_CODE_SEGMENT (partition_id), 0,
       3);
   data_sel = GDT_BUILD_SELECTOR (GDT_PARTITION_DATA_SEGMENT (partition_id), 0,
       3);

   sp = (uint32_t) &ctx;

   memset (&ctx, 0, sizeof (interrupt_frame));

   pok_arch_preempt_disable ();

   ctx.es = ctx.ds = ctx.ss = data_sel;

   ctx.__esp   = (uint32_t) (&ctx.error); /* for pusha */
   ctx.eip     = user_pc;
   ctx.eax     = arg1;
   ctx.ebx     = arg2;
   ctx.cs      = code_sel;
   ctx.eflags  = 1 << 9;
   ctx.esp     = user_sp;

   tss_set_esp0 (kernel_sp);

   asm ("mov %0, %%esp          \n"
        "pop %%es               \n"
        "pop %%ds               \n"
        "popa                   \n"
        "addl $4, %%esp         \n"
        "iret                   \n"
        :
        : "m" (sp)
       );
}
```

4.21.3.3 uint32_t pok_space_base_vaddr (uint32_t *addr*)

Returns

partition virtual base adress.

See also

SPARC_PARTITION_BASE_VADDR

Definition at line 64 of file space.c.

```
{
   (void) addr;
   return (0);
}
```

4.21.3.4 uint32_t pok_space_context_create (uint8_t *partition_id,* uint32_t *entry_rel,* uint32_t *stack_rel,* uint32_t *arg1,* uint32_t *arg2*)

Create a new context in the given space

Initilize thread stack.

Definition at line 70 of file space.c.

```
{
   char*              stack_addr;
   space_context_t*   sp;

   stack_addr = pok_bsp_mem_alloc (KERNEL_STACK_SIZE);

   sp = (space_context_t *)
      (stack_addr + KERNEL_STACK_SIZE - 4 - sizeof (space_context_t));

   memset (sp, 0, sizeof (space_context_t));

   sp->ctx.__esp  = (uint32_t)(&sp->ctx.eip); /* for pusha */
   sp->ctx.eip    = (uint32_t)pok_dispatch_space;
   sp->ctx.cs     = GDT_CORE_CODE_SEGMENT << 3;
   sp->ctx.eflags = 1 << 9;

   sp->arg1          = arg1;
   sp->arg2          = arg2;
   sp->kernel_sp     = (uint32_t)sp;
   sp->user_sp       = stack_rel;
   sp->user_pc       = entry_rel;
   sp->partition_id  = partition_id;

   return ((uint32_t) sp);
}
```

4.21.3.5 pok_ret_t pok_space_switch (uint8_t *old_partition_id*, uint8_t *new_partition_id*)

Switch from one space to another

Switch adress space in MMU (context register).

Definition at line 53 of file space.c.

```
{
   gdt_disable (GDT_PARTITION_CODE_SEGMENT(old_partition_id));
   gdt_disable (GDT_PARTITION_DATA_SEGMENT(old_partition_id));
   gdt_enable (GDT_PARTITION_CODE_SEGMENT(new_partition_id));
   gdt_enable (GDT_PARTITION_DATA_SEGMENT(new_partition_id));

   return (POK_ERRNO_OK);
}
```

4.22 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/ppc/syscalls.c File Reference

`#include <errno.h>` `#include <core/debug.h>` `#include <core/syscall.-h>` `#include <core/partition.h>` `#include <types.h>` `#include <libc.h>`

Functions

- void pok_arch_sc_int (uint32_t num, uint32_t arg1, uint32_t arg2, uint32_t arg3, uint32_t arg4, uint32_t arg5)

4.22.1 Function Documentation

4.22.1.1 void pok_arch_sc_int (uint32_t *num*, uint32_t *arg1*, uint32_t *arg2*, uint32_t *arg3*, uint32_t *arg4*, uint32_t *arg5*)

Definition at line 26 of file syscalls.c.

```
{
   uint8_t              part_id;

   pok_syscall_info_t   syscall_info;
   pok_syscall_args_t   syscall_args;
   pok_syscall_id_t     syscall_id;

   part_id  = pok_current_partition;

   /* prepare syscall_info */
   syscall_info.partition = part_id;
   syscall_info.base_addr = pok_partitions[part_id].base_addr;
   syscall_info.thread    = POK_SCHED_CURRENT_THREAD;
   /* prepare syscall_args */
   syscall_args.arg1 = arg1;
   syscall_args.arg2 = arg2;
   syscall_args.arg3 = arg3;
   syscall_args.arg4 = arg4;
   syscall_args.arg5 = arg5;

   syscall_args.nargs = 5;

   /* prepare syscall_id */
   syscall_id = (pok_syscall_id_t) num;

   if (POK_CHECK_PTR_IN_PARTITION(syscall_info.partition, &syscall_args) != 0)
   {
      /*
       * Perform the syscall baby !
       */
     pok_core_syscall (syscall_id, &syscall_args, &syscall_info);
   }
}
```

4.23 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/sparc/syscalls.c File Reference

Syscalls management in SPARC.

`#include <errno.h>` `#include <core/debug.h>` `#include <core/syscall.h>` `#include <core/partition.h>` `#include <types.h>` `#include <libc.h>` `#include "thread.h"` `#include "context_offset.h"` × `#include "traps.h"` `#include "arch.h"`

Functions

- void pok_arch_sc_int (void)
- void pok_syscalls_init (void)

4.23.1 Detailed Description

Syscalls management in SPARC.

Author

Fabien Chouteau

Definition in file syscalls.c.

4.23.2 Function Documentation

4.23.2.1 void pok_arch_sc_int (void)

Syscalls handler.

Definition at line 39 of file syscalls.c.

```
{
  uint8_t *ctx = (uint8_t *)pok_arch_sp;
  uint32_t num = *(uint32_t *)((char *)ctx - I0_OFFSET);
  uint8_t              part_id;
  pok_syscall_info_t   syscall_info;
  pok_ret_t            syscall_ret;
  pok_syscall_args_t   syscall_args;
  pok_syscall_id_t     syscall_id;


  part_id  = pok_current_partition;

  /* prepare syscall_info */
  syscall_info.partition = part_id;
  syscall_info.base_addr = pok_partitions[part_id].base_addr;
  syscall_info.thread    = POK_SCHED_CURRENT_THREAD;

  /* prepare syscall_args */
  syscall_args.arg1 = *(uint32_t *)(ctx - I1_OFFSET);
  syscall_args.arg2 = *(uint32_t *)(ctx - I2_OFFSET);
  syscall_args.arg3 = *(uint32_t *)(ctx - I3_OFFSET);
  syscall_args.arg4 = *(uint32_t *)(ctx - I4_OFFSET);
  syscall_args.arg5 = *(uint32_t *)(ctx - I5_OFFSET);

  syscall_args.nargs = 5;

  /* prepare syscall_id */
  syscall_id = (pok_syscall_id_t) num;

  /*
   * No pointer check needed, syscall_args is allocated in kernel stack.
   */
  syscall_ret = pok_core_syscall (syscall_id, &syscall_args, &syscall_info);

  *(uint32_t *)(ctx - I0_OFFSET) = syscall_ret;
  *(uint32_t *)(ctx - PC_OFFSET) += 4; // skip "ta" instruction
  *(uint32_t *)(ctx - NPC_OFFSET) += 4;
}
```

4.23.2.2 void pok_syscalls_init (void)

Syscalls initialization. Just register the syscall handler.

Definition at line 83 of file syscalls.c.

```
{
  pok_arch_event_register(SPARC_TRAP_SYSCALL_BASE + 0x2, pok_arch_sc_int);
}
```

4.24 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/x86/syscalls.c File Reference

This file implement system-calls for x86 platform.

`#include <errno.h>` `#include <core/debug.h>` `#include <core/partition.h>` `#include <core/syscall.h>` `#include "gdt.h"` `#include "event.h"`

Defines

- #define PARTITION_ID(cs) (((cs >> 3) - 4) / 2)

Functions

- INTERRUPT_HANDLER_syscall (syscall_gate)
- pok_ret_t pok_syscall_init ()

4.24.1 Detailed Description

This file implement system-calls for x86 platform.

Author

Julian Pidancet
Julien Delange
Laurent Lec

Definition in file syscalls.c.

4.24.2 Define Documentation

4.24.2.1 #define PARTITION_ID(*cs*) (((cs >> 3) - 4) / 2)

Definition at line 33 of file syscalls.c.

4.24.3 Function Documentation

4.24.3.1 INTERRUPT_HANDLER_syscall (syscall_gate)

Definition at line 35 of file syscalls.c.

```
{
   pok_syscall_info_t   syscall_info;
   pok_ret_t            syscall_ret;
   pok_syscall_args_t*  syscall_args;
   pok_syscall_id_t     syscall_id;

   /*
    * Give informations about syscalls: which partition, thread
    * initiates the syscall, the base addr of the partition and so on.
    */
   syscall_info.partition = PARTITION_ID (frame->cs);
   syscall_info.base_addr = pok_partitions[syscall_info.partition].base_addr;
   syscall_info.thread    = POK_SCHED_CURRENT_THREAD;

   syscall_args = (pok_syscall_args_t*) (frame->ebx + syscall_info.base_addr);

   /*
    * Get the syscall id in the eax register
    */
   syscall_id = (pok_syscall_id_t) frame->eax;

   /*
    * Check that pointer is inside the adress space
    */
   if (POK_CHECK_PTR_IN_PARTITION(syscall_info.partition, syscall_args) == 0)
   {
         syscall_ret = POK_ERRNO_EINVAL;
   }
   else
   {
      /*
       * Perform the syscall baby !
       */
      syscall_ret = pok_core_syscall (syscall_id, syscall_args, &syscall_info);
   }

   /*
    * And finally, put the return value in eax register
    */
   asm ("movl %0, %%eax  \n"
        :
        : "m" (syscall_ret));
}
```

4.24.3.2 pok_ret_t pok_syscall_init ( )

Init system calls

Definition at line 83 of file syscalls.c.

```
{
   pok_idt_set_gate (POK_SYSCALL_INT_NUMBER,
                     GDT_CORE_CODE_SEGMENT << 3,
                     (uint32_t) syscall_gate,
                     IDTE_INTERRUPT,
                     3);

   return (POK_ERRNO_OK);
}
```

4.25 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/ppc/thread.c File Reference

`#include <bsp.h> #include <libc.h> #include <errno.h>` ×
`#include <core/thread.h> #include "thread.h"`

4.26 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/sparc/thread.c File Reference

Thread management.

`#include <bsp.h> #include <libc.h> #include <errno.h>` ×
`#include <core/thread.h> #include "thread.h" #include "context_offset.h" #include "ioports.h"`

4.26.1 Detailed Description

Thread management.

Author

Fabien Chouteau

Definition in file thread.c.

4.27 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/x86/thread.c File Reference

`#include <bsp.h> #include <libc.h> #include <errno.-h> #include <core/thread.h> #include "gdt.h" #include "thread.h"`

4.28 /home/matias/projet/stage-pok/pok/trunk/kernel/core/thread.c File Reference

Thread management in kernel.

`#include <types.h> #include <arch.h> #include <core/debug.-h> #include <core/error.h> #include <core/thread.h>` ×
`#include <core/sched.h> #include <core/partition.h>` ×
`#include <core/time.h> #include <core/instrumentation.-h>`

4.28.1 Detailed Description

Thread management in kernel.

Author

Julien Delange

Date

2008-2009

Definition in file thread.c.

4.29 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/ppc/thread.h File Reference

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

Data Structures

- struct context_t
- struct volatile_context_t

Functions

- uint32_t pok_context_create (uint32_t id, uint32_t stack_size, uint32_t entry)
- void pok_context_switch (uint32_t *old_sp, uint32_t new_sp)

4.29.1 Function Documentation

4.29.1.1 uint32_t pok_context_create (uint32_t *id*, uint32_t *stack_size*, uint32_t *entry*)

4.29.1.2 void pok_context_switch (uint32_t * *old_sp*, uint32_t *new_sp*)

4.30 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/sparc/thread.h File Reference

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

Functions

- uint32_t pok_context_create (uint32_t id, uint32_t stack_size, uint32_t entry)
- void pok_context_switch (uint32_t ∗old_sp, uint32_t new_sp)

Variables

- uint32_t pok_arch_sp

4.30.1 Detailed Description

Author

Fabien Chouteau

Definition in file thread.h.

4.30.2 Function Documentation

4.30.2.1 uint32_t pok_context_create (uint32_t *id*, uint32_t *stack_size*, uint32_t *entry*)

4.30.2.2 void pok_context_switch (uint32_t ∗ *old_sp*, uint32_t *new_sp*)

4.30.3 Variable Documentation

4.30.3.1 uint32_t pok_arch_sp

4.31 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/x86/thread.h File Reference

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

Data Structures

- struct context_t
- struct start_context_t

Functions

- uint32_t pok_context_create (uint32_t id, uint32_t stack_size, uint32_t entry)
- void pok_context_switch (uint32_t ∗old_sp, uint32_t new_sp)

4.31.1 Function Documentation

4.31.1.1 uint32_t pok_context_create (uint32_t *id,* uint32_t *stack_size,* uint32_t *entry*)

4.31.1.2 void pok_context_switch (uint32_t ∗ *old_sp,* uint32_t *new_sp*)

4.32 /home/matias/projet/stage-pok/pok/trunk/kernel/include/core/thread.h File Reference

4.33 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/ppc/timer.c File Reference

```
#include <errno.h> #include <bsp.h> #include <core/time.-
h> #include <core/sched.h>
```

Defines

- #define BUS_FREQ (100 ∗ 1000000U)
- #define FREQ_DIV 40

Functions

- void pok_arch_decr_int (void)
- pok_ret_t pok_bsp_time_init ()

4.33.1 Define Documentation

4.33.1.1 #define BUS_FREQ (100 ∗ 1000000U)

Definition at line 24 of file timer.c.

4.33.1.2 #define FREQ_DIV 40

Definition at line 26 of file timer.c.

4.33.2 Function Documentation

4.33.2.1 void pok_arch_decr_int (void)

Definition at line 73 of file timer.c.

```
{
  int err;

  do
  {
    err = pok_arch_set_decr();

    pok_tick_counter += FREQ_DIV;
  } while (err != POK_ERRNO_OK);

  pok_sched ();
}
```

4.33.2.2 pok_ret_t pok_bsp_time_init ( )

Definition at line 87 of file timer.c.

```
{
  time_inter = (BUS_FREQ * FREQ_DIV) / POK_TIMER_FREQUENCY;
  time_last = get_ppc_tb ();
  pok_arch_set_decr();

  return (POK_ERRNO_OK);
}
```

4.34 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/sparc/leon3/timer.c File Reference

Leon3 timer management.

`#include <errno.h>` `#include <bsp.h>` `#include <core/time.h>` `#include <core/sched.h>` `#include <arch.h>` `#include "ioports.h"` `#include "sparc_conf.h"` `#include "timer.h"` × `#include "irq.h"` `#include "../traps.h"`

Functions

- void timer_isr (void)
- pok_ret_t pok_bsp_time_init ()

4.34.1 Detailed Description

Leon3 timer management.

Author

Fabien Chouteau

Definition in file timer.c.

4.34.2 Function Documentation

4.34.2.1 pok_ret_t pok_bsp_time_init ( )

Initialize the timer, register the ISR and unmask the interrupt.

See also

unmask_irq(irq_nbr)

Definition at line 50 of file timer.c.

```
{
  outw(TIMER1 + TIMER_SCALER_OFFSET, 1);
  outw(TIMER1 + TIMER_SCAL_RELOAD_OFFSET, 1);

  outw(TIMER1 + TIMER_CNT_VAL_OFFSET, 1);
  outw(TIMER1 + TIMER_RELOAD_OFFSET, SPARC_PROC_FREQ / POK_TIMER_FREQUENCY);
  outw(TIMER1 + TIMER_CTRL_OFFSET,
       TIMER_CTRL_EN | TIMER_CTRL_RS | TIMER_CTRL_LD | TIMER_CTRL_IE);

  pok_arch_event_register(SPARC_TRAP_IRQ_BASE + TIMER_IRQ, timer_isr);
  unmask_irq(TIMER_IRQ);
  return (POK_ERRNO_OK);
}
```

4.34.2.2 void timer_isr (void)

Timer interrupt subroutine.

See also

ack_irq(irq_nbr)

Definition at line 39 of file timer.c.

```
{
  ack_irq(TIMER_IRQ);
  CLOCK_HANDLER
  return;
}
```

4.35 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/sparc/context-_offset.h File Reference

Define registers offset in context stack.

Defines

- #define L0_OFFSET 0x00
- #define L1_OFFSET 0x04
- #define L2_OFFSET 0x08
- #define L3_OFFSET 0x0c
- #define L4_OFFSET 0x10
- #define L5_OFFSET 0x14
- #define L6_OFFSET 0x18
- #define L7_OFFSET 0x1c
- #define I0_OFFSET 0x20
- #define I1_OFFSET 0x24
- #define I2_OFFSET 0x28
- #define I3_OFFSET 0x2c
- #define I4_OFFSET 0x30
- #define I5_OFFSET 0x34
- #define I6_OFFSET 0x38
- #define I7_OFFSET 0x3c
- #define G7_OFFSET 0x04
- #define G6_OFFSET 0x08
- #define G5_OFFSET 0x0c
- #define G4_OFFSET 0x10
- #define G3_OFFSET 0x14
- #define G2_OFFSET 0x18
- #define G1_OFFSET 0x1c
- #define WIM_OFFSET 0x40
- #define PSR_OFFSET 0x44
- #define Y_OFFSET 0x48
- #define PC_OFFSET 0x4c
- #define NPC_OFFSET 0x50
- #define RESTORE_CNT_OFFSET 0x54

4.35.1 Detailed Description

Define registers offset in context stack.

Author

Fabien Chouteau

Definition in file context_offset.h.

4.35.2 Define Documentation

4.35.2.1 #define G1_OFFSET 0x1c

Definition at line 57 of file context_offset.h.

4.35.2.2 #define G2_OFFSET 0x18

Definition at line 56 of file context_offset.h.

4.35.2.3 #define G3_OFFSET 0x14

Definition at line 55 of file context_offset.h.

4.35.2.4 #define G4_OFFSET 0x10

Definition at line 54 of file context_offset.h.

4.35.2.5 #define G5_OFFSET 0x0c

Definition at line 53 of file context_offset.h.

4.35.2.6 #define G6_OFFSET 0x08

Definition at line 52 of file context_offset.h.

4.35.2.7 #define G7_OFFSET 0x04

Definition at line 51 of file context_offset.h.

4.35.2.8 #define I0_OFFSET 0x20

Definition at line 38 of file context_offset.h.

4.35.2.9 #define I1_OFFSET 0x24

Definition at line 39 of file context_offset.h.

4.35.2.10 #define I2_OFFSET 0x28

Definition at line 40 of file context_offset.h.

4.35.2.11 #define I3_OFFSET 0x2c

Definition at line 41 of file context_offset.h.

4.35.2.12 #define I4_OFFSET 0x30

Definition at line 42 of file context_offset.h.

4.35.2.13 #define I5_OFFSET 0x34

Definition at line 43 of file context_offset.h.

4.35.2.14 #define I6_OFFSET 0x38

Definition at line 44 of file context_offset.h.

4.35.2.15 #define I7_OFFSET 0x3c

Definition at line 45 of file context_offset.h.

4.35.2.16 #define L0_OFFSET 0x00

Definition at line 30 of file context_offset.h.

4.35.2.17 #define L1_OFFSET 0x04

Definition at line 31 of file context_offset.h.

4.35.2.18 #define L2_OFFSET 0x08

Definition at line 32 of file context_offset.h.

4.35.2.19 #define L3_OFFSET 0x0c

Definition at line 33 of file context_offset.h.

4.35.2.20 #define L4_OFFSET 0x10

Definition at line 34 of file context_offset.h.

4.35.2.21 #define L5_OFFSET 0x14

Definition at line 35 of file context_offset.h.

4.35.2.22 #define L6_OFFSET 0x18

Definition at line 36 of file context_offset.h.

4.35.2.23 #define L7_OFFSET 0x1c

Definition at line 37 of file context_offset.h.

4.35.2.24 #define NPC_OFFSET 0x50

Definition at line 66 of file context_offset.h.

4.35.2.25 #define PC_OFFSET 0x4c

Definition at line 65 of file context_offset.h.

4.35.2.26 #define PSR_OFFSET 0x44

Definition at line 63 of file context_offset.h.

4.35.2.27 #define RESTORE_CNT_OFFSET 0x54

Definition at line 67 of file context_offset.h.

4.35.2.28 #define WIM_OFFSET 0x40

Definition at line 62 of file context_offset.h.

4.35.2.29 #define Y_OFFSET 0x48

Definition at line 64 of file context_offset.h.

4.36 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/sparc/leon3/irq.h File Reference

Leon3 IRQ management.

```
#include "ioports.h"
```

Defines

- #define IRQMP_BASE 0x80000200
- #define IRQMP_CLEAR_OFFSET 0x10U
- #define IRQMP_MASK0_OFFSET 0x40U
- #define unmask_irq(irq_nbr)
- #define ack_irq(irq_nbr) outw(IRQMP_BASE + IRQMP_CLEAR_OFFSET, (1 << (irq_nbr)))

4.36.1 Detailed Description

Leon3 IRQ management.

Author

Fabien Chouteau

Definition in file irq.h.

4.36.2 Define Documentation

4.36.2.1 #define **ack_irq**(*irq_nbr*) outw(IRQMP_BASE + IRQMP_CLEAR_OFFSET, (1 << (irq_nbr)))

Acknowledge the given irq.

Definition at line 44 of file irq.h.

4.36.2.2 #define **IRQMP_BASE** 0x80000200

Leon3 IRQMP IO adress

Definition at line 28 of file irq.h.

4.36.2.3 #define **IRQMP_CLEAR_OFFSET** 0x10U

Clear register offset

Definition at line 30 of file irq.h.

4.36.2.4 #define **IRQMP_MASK0_OFFSET** 0x40U

Mask register offset

Definition at line 31 of file irq.h.

4.36.2.5 #define unmask_irq(*irq_nbr*)

Value:

```
outw(IRQMP_BASE + IRQMP_MASK0_OFFSET,         \
                                    inb(IRQMP_BASE + IRQMP_MASK0_OFFSET) | (1 << (
      irq_nbr)))
```

Unmask the given irq.

Definition at line 37 of file irq.h.

4.37 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/sparc/leon3/sparc_conf.h File Reference

Define all constant values for a SPARC bsp.

Defines

- #define SPARC_RAM_ADDR 0x40000000
- #define SPARC_PROC_FREQ 50000000U
- #define WINDOWS_NBR 8
- #define ASI_MMU_BYPASS 0x1c /∗ not sparc v8 compliant ∗/
- #define SPARC_PAGE_SIZE (256 ∗ 1024)
- #define SPARC_PARTITION_SIZE SPARC_PAGE_SIZE
- #define SPARC_PARTITION_BASE_VADDR 0x0

4.37.1 Detailed Description

Define all constant values for a SPARC bsp.

Author

Fabien Chouteau

Definition in file sparc_conf.h.

4.37.2 Define Documentation

4.37.2.1 #define ASI_MMU_BYPASS 0x1c /∗ not sparc v8 compliant ∗/

Definition at line 32 of file sparc_conf.h.

4.37.2.2 #define SPARC_PAGE_SIZE (256 ∗ 1024)

Page size (256 Kbytes)

Definition at line 34 of file sparc_conf.h.

4.37.2.3 #define SPARC_PARTITION_BASE_VADDR 0x0

Partition virtual base adress. Should always be 0x0

Definition at line 37 of file sparc_conf.h.

4.37.2.4 #define SPARC_PARTITION_SIZE SPARC_PAGE_SIZE

Maximum partition size

Definition at line 35 of file sparc_conf.h.

4.37.2.5 #define SPARC_PROC_FREQ 50000000U

Processor frequency (in Hz)

Definition at line 28 of file sparc_conf.h.

4.37.2.6 #define SPARC_RAM_ADDR 0x40000000

RAM base adress

Definition at line 26 of file sparc_conf.h.

4.37.2.7 #define WINDOWS_NBR 8

Number of register windows

Definition at line 30 of file sparc_conf.h.

4.38 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/sparc/leon3/timer.h File Reference

Defines

- #define TIMER_CTRL_EN (1 << 0)
- #define TIMER_CTRL_RS (1 << 1)
- #define TIMER_CTRL_LD (1 << 2)
- #define TIMER_CTRL_IE (1 << 3)
- #define TIMER_CTRL_IP (1 << 4)

- #define TIMER_CTRL_CH (1 << 5)
- #define TIMER_CTRL_DH (1 << 6)
- #define TIMER_SCALER_OFFSET 0x00
- #define TIMER_SCAL_RELOAD_OFFSET 0x04
- #define TIMER_CNT_VAL_OFFSET 0x10
- #define TIMER_RELOAD_OFFSET 0x14
- #define TIMER_CTRL_OFFSET 0x18
- #define TIMER_IRQ 0x8U
- #define TIMER1 0x80000300

4.38.1 Detailed Description

Author

Fabien Chouteau

Definition in file timer.h.

4.38.2 Define Documentation

4.38.2.1 #define TIMER1 0x80000300

first Leon3 TIMER IO adress

Definition at line 42 of file timer.h.

4.38.2.2 #define TIMER_CNT_VAL_OFFSET 0x10

Counter value register offset

Definition at line 36 of file timer.h.

4.38.2.3 #define TIMER_CTRL_CH (1 << 5)

Chain

Definition at line 30 of file timer.h.

4.38.2.4 #define TIMER_CTRL_DH (1 << 6)

Debug Halt

Definition at line 31 of file timer.h.

4.38.2.5 #define TIMER_CTRL_EN (1 $<<$ 0)

Enable

Definition at line 25 of file timer.h.

4.38.2.6 #define TIMER_CTRL_IE (1 $<<$ 3)

Interrupt enable

Definition at line 28 of file timer.h.

4.38.2.7 #define TIMER_CTRL_IP (1 $<<$ 4)

Interrupt Pending

Definition at line 29 of file timer.h.

4.38.2.8 #define TIMER_CTRL_LD (1 $<<$ 2)

Load

Definition at line 27 of file timer.h.

4.38.2.9 #define TIMER_CTRL_OFFSET 0x18

Control register offset

Definition at line 38 of file timer.h.

4.38.2.10 #define TIMER_CTRL_RS (1 $<<$ 1)

Restart

Definition at line 26 of file timer.h.

4.38.2.11 #define TIMER_IRQ 0x8U

Definition at line 40 of file timer.h.

4.38.2.12 #define TIMER_RELOAD_OFFSET 0x14

Counter reload register offset

Definition at line 37 of file timer.h.

4.38.2.13 #define TIMER_SCAL_RELOAD_OFFSET 0x04

Scaler reload register offset

Definition at line 34 of file timer.h.

4.38.2.14 #define TIMER_SCALER_OFFSET 0x00

Scaler value register offset

Definition at line 33 of file timer.h.

4.39 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/sparc/psr.h File Reference

Processor State Register utils.

Defines

- #define PSR_ET 0x20
- #define PSR_PS 0x40
- #define PSR_S 0x80
- #define PSR_CWP_MASK 0x1F
- #define PSR_PIL(pil) (((pil) & 0xF) $<<$ 8)

4.39.1 Detailed Description

Processor State Register utils.

Author

Fabien Chouteau

Definition in file psr.h.

4.39.2 Define Documentation

4.39.2.1 #define PSR_CWP_MASK 0x1F

Current Window Pointer Mask

Definition at line 29 of file psr.h.

4.39.2.2 #define PSR_ET 0x20

enable traps

Definition at line 26 of file psr.h.

4.39.2.3 #define PSR_PIL(*pil*) (((pil) & 0xF) << 8)

Proc Interrupt Level

Definition at line 30 of file psr.h.

4.39.2.4 #define PSR_PS 0x40

previous supervisor

Definition at line 27 of file psr.h.

4.39.2.5 #define PSR_S 0x80

supervisor

Definition at line 28 of file psr.h.

4.40 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/sparc/space.h File Reference

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

Defines

- #define LEON_CTX_NBR 256

PTD/PTE ET field

(cf SPARC V8 Manual, page 247)

- #define MM_ET_INVALID 0x0
- #define MM_ET_PTD 0x1
- #define MM_ET_PTE 0x2

PTE ACC field

Acces permisions. (cf SPARC V8 Manual, page 248)

- #define MM_ACC_R (0x0 << 2)
- #define MM_ACC_RW (0x1 << 2)
- #define MM_ACC_RE (0x2 << 2)

- #define MM_ACC_RWE (0x3 << 2)
- #define MM_ACC_E (0x4 << 2)
- #define MM_ACC_R_S_RW (0x5 << 2)
- #define MM_ACC_S_RE (0x6 << 2)
- #define MM_ACC_S_RWE (0x7 << 2)

PTE misc fields

(cf SPARC V8 Manual, page 248)

- #define MM_CACHEABLE (1 << 7)
- #define MM_MODIFIED (1 << 6)
- #define MM_REFERENCED (1 << 5)

MMU levels utils

- #define MM_LVL1_ENTRIES_NBR 256
- #define MM_LVL1_PAGE_SIZE (64 ∗ 64 ∗ 4 ∗ 1024)
- #define mm_index1(addr) (((addr) >> 24) & 0xFF)
- #define MM_LVL2_ENTRIES_NBR 64
- #define MM_LVL2_PAGE_SIZE (64 ∗ 4 ∗ 1024)
- #define mm_index2(addr) (((addr) >> 18) & 0x3F)
- #define MM_LVL3_ENTRIES_NBR 64
- #define MM_LVL3_PAGE_SIZE (4 ∗ 1024)
- #define mm_index3(addr) (((addr) >> 12) & 0x3F)

MMU ASI and registers

- #define ASI_M_MMUREGS 0x19 /∗ not sparc v8 compliant ∗/
- #define MMU_CTRL_REG 0x00000000
- #define MMU_CTXTBL_PTR 0x00000100
- #define MMU_CTX_REG 0x00000200
- #define MMU_FAULT_STATUS 0x00000300
- #define MMU_FAULT_ADDR 0x00000400

Typedefs

- typedef uint32_t pte
- typedef uint32_t ptd

Functions

- void pok_arch_space_init (void)

4.40.1 Detailed Description

Author

Fabien Chouteau

Definition in file space.h.

4.40.2 Define Documentation

4.40.2.1 #define ASI_M_MMUREGS 0x19 /∗ not sparc v8 compliant ∗/

Definition at line 97 of file space.h.

4.40.2.2 #define LEON_CTX_NBR 256

Maximum number of contexts

Definition at line 105 of file space.h.

4.40.2.3 #define MM_ACC_E (0x4 << 2)

All Execute only

Definition at line 46 of file space.h.

4.40.2.4 #define MM_ACC_R (0x0 << 2)

All Read only

Definition at line 42 of file space.h.

4.40.2.5 #define MM_ACC_R_S_RW (0x5 << 2)

User Read only, Supervisor Read Write

Definition at line 47 of file space.h.

4.40.2.6 #define MM_ACC_RE (0x2 << 2)

All Read Execute

Definition at line 44 of file space.h.

4.40.2.7 #define MM_ACC_RW (0x1 << 2)

All Read Write

Definition at line 43 of file space.h.

4.40.2.8 #define MM_ACC_RWE (0x3 << 2)

All Read Write Execute

Definition at line 45 of file space.h.

4.40.2.9 #define MM_ACC_S_RE (0x6 $<<$ 2)

Supervisor Read Write Execute

Definition at line 49 of file space.h.

4.40.2.10 #define MM_ACC_S_RWE (0x7 $<<$ 2)

Supervisor Read Execute

Definition at line 50 of file space.h.

4.40.2.11 #define MM_CACHEABLE (1 $<<$ 7)

Definition at line 58 of file space.h.

4.40.2.12 #define MM_ET_INVALID 0x0

Invalid

Definition at line 32 of file space.h.

4.40.2.13 #define MM_ET_PTD 0x1

Page Table Descriptor

Definition at line 33 of file space.h.

4.40.2.14 #define MM_ET_PTE 0x2

Page Table Entry

Definition at line 34 of file space.h.

4.40.2.15 #define mm_index1(*addr*) (((addr) $>>$ 24) & 0xFF)

Compute the index in 1st level table for the given adress.

Definition at line 73 of file space.h.

4.40.2.16 #define mm_index2(*addr*) (((addr) $>>$ 18) & 0x3F)

Compute the index in 2nd level table for the given adress.

Definition at line 81 of file space.h.

4.40.2.17 #define mm_index3(*addr*) (((addr) $>>$ 12) & 0x3F)

Compute the index in 3rd level table for the given adress.

Definition at line 89 of file space.h.

4.40.2.18 #define MM_LVL1_ENTRIES_NBR 256

Number of entries in 1st level table

Definition at line 67 of file space.h.

4.40.2.19 #define MM_LVL1_PAGE_SIZE (64 $*$ 64 $*$ 4 $*$ 1024)

16 MegaBytes

Definition at line 68 of file space.h.

4.40.2.20 #define MM_LVL2_ENTRIES_NBR 64

Number of entries in 2nd level table

Definition at line 75 of file space.h.

4.40.2.21 #define MM_LVL2_PAGE_SIZE (64 $*$ 4 $*$ 1024)

256 KiloBytes

Definition at line 76 of file space.h.

4.40.2.22 #define MM_LVL3_ENTRIES_NBR 64

Number of entries in 3rd level table

Definition at line 83 of file space.h.

4.40.2.23 #define MM_LVL3_PAGE_SIZE (4 $*$ 1024)

4 KiloBytes

Definition at line 84 of file space.h.

4.40.2.24 #define MM_MODIFIED (1 $<<$ 6)

Definition at line 59 of file space.h.

4.40.2.25 #define MM_REFERENCED (1 $<<$ 5)

Definition at line 60 of file space.h.

4.40.2.26 #define MMU_CTRL_REG 0x00000000

Definition at line 98 of file space.h.

4.40.2.27 #define MMU_CTX_REG 0x00000200

Definition at line 100 of file space.h.

4.40.2.28 #define MMU_CTXTBL_PTR 0x00000100

Definition at line 99 of file space.h.

4.40.2.29 #define MMU_FAULT_ADDR 0x00000400

Definition at line 102 of file space.h.

4.40.2.30 #define MMU_FAULT_STATUS 0x00000300

Definition at line 101 of file space.h.

4.40.3 Typedef Documentation

4.40.3.1 typedef uint32_t ptd

Definition at line 108 of file space.h.

4.40.3.2 typedef uint32_t pte

Definition at line 107 of file space.h.

4.40.4 Function Documentation

4.40.4.1 void pok_arch_space_init (void)

Initilize MMU tables.

Definition at line 132 of file space.c.

```
{
  uint32_t sdr1;

  pt_base = 0;
  pt_mask = 0x3ff;

  sdr1 = pt_base | (pt_mask >> 10);
  asm volatile ("mtsdr1 %0" : : "r"(sdr1));
}
```

4.41 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/x86/space.h File Reference

`#include <types.h>` `#include "thread.h"`

Data Structures

- struct space_context_t

4.41.1 Detailed Description

Author

Julian Pidancet

Date

2008-2009

Definition in file space.h.

4.42 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/sparc/syscalls.h File Reference

Functions

- void pok_syscalls_init (void)

4.42.1 Detailed Description

Author

Fabien Chouteau

Definition in file syscalls.h.

4.42.2 Function Documentation

4.42.2.1 void pok_syscalls_init (void)

Syscalls initialization. Just register the syscall handler.

Definition at line 83 of file syscalls.c.

```
{
  pok_arch_event_register(SPARC_TRAP_SYSCALL_BASE + 0x2, pok_arch_sc_int);
}
```

4.43 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/sparc/traps.c File Reference

Traps management.

`#include <types.h>` `#include <errno.h>` `#include <libc.h>` `#include <core/debug.h>` `#include "thread.h"` `#include "traps.h"`

Functions

- pok_ret_t traps_init (void)
- void trap_handler (unsigned int pc, unsigned int npc, unsigned int psr, unsigned int trap_nb, unsigned int restore_counter, unsigned int stack_pointer)

Variables

- sparc_traps_handler pok_sparc_isr [256]

4.43.1 Detailed Description

Traps management.

Author

Fabien Chouteau

Definition in file traps.c.

4.43.2 Function Documentation

4.43.2.1 void trap_handler (unsigned int *pc*, unsigned int *npc*, unsigned int *psr*, unsigned int *trap_nb*, unsigned int *restore_counter*, unsigned int *stack_pointer*)

Function called by interrupt pre-handler. Call the correct handler for the given trap number.

Parameters

trap_nb	The number of the current trap. (cf SPARC V8 Manual, page 76)
stack_-pointer	Adress of the interrupted stack.

See also

pok_arch_sp

Definition at line 53 of file traps.c.

```
{
  (void)restore_counter;

  pok_arch_sp = stack_pointer;

  if (pok_sparc_isr[trap_nb] != NULL)
  {
    pok_sparc_isr[trap_nb]();
  }
  else
  {
#ifdef POK_NEEDS_DEBUG
    printf ("[KERNEL] [ERROR] Unhandled trap: 0x%x %%PSR=%x %%PC=%x %%nPC=%x
       %%sp=0x%x\n", trap_nb, psr, pc, npc, stack_pointer);
    printf("%%psr : impl:0x%x ver:%x nzvc:%u%u%u%u EC:%u EF:%u PIL:0x%x S:%u
       PS:%u ET:%u CWP:%u\n\r",
            (psr >> 28) & 0xF, (psr >> 24) & 0xF,
            (psr >> 23) & 0x1, (psr >> 22) & 0x1c, (psr >> 21) & 0x1, (psr >> 20
      ) & 0x1,
            (psr >> 23) & 0x1, (psr >> 12) & 0x1, (psr >> 8) & 0xF, (psr >> 7) &
       0x1, (psr >> 6) & 0x1,
            (psr >> 5) & 0x1, psr & 0xF);
#else
    (void)psr;
    (void)npc;
    (void)pc;
#endif
    POK_FATAL ("Unhandled trap");
  }
  return;
}
```

4.43.2.2 pok_ret_t traps_init (void)

Initialize ISR table.

See also

pok_sparc_isr

Definition at line 40 of file traps.c.

```
{
  memset((unsigned char *)pok_sparc_isr, 0x0, sizeof (pok_sparc_isr));
  return POK_ERRNO_OK;
}
```

4.43.3 Variable Documentation

4.43.3.1 sparc_traps_handler pok_sparc_isr[256]

Interrupt subroutine table.

Definition at line 34 of file traps.c.

4.44 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/sparc/traps.h File Reference

`#include <types.h>` `#include <errno.h>`

Defines

- #define SPARC_TRAP_IRQ_BASE 0x10
- #define SPARC_TRAP_SYSCALL_BASE 0x80

Typedefs

- typedef void(∗ sparc_traps_handler)(void)

Functions

- pok_ret_t traps_init (void)

Variables

- sparc_traps_handler pok_sparc_isr [256]

4.44.1 Detailed Description

Author

Fabien Chouteau

Definition in file traps.h.

4.44.2 Define Documentation

4.44.2.1 #define SPARC_TRAP_IRQ_BASE 0x10

Definition at line 28 of file traps.h.

4.44.2.2 #define SPARC_TRAP_SYSCALL_BASE 0x80

Definition at line 29 of file traps.h.

4.44.3 Typedef Documentation

4.44.3.1 typedef void(∗ sparc_traps_handler)(void)

Definition at line 31 of file traps.h.

4.44.4 Function Documentation

4.44.4.1 pok_ret_t traps_init (void)

Initialize ISR table.

See also

pok_sparc_isr

Definition at line 40 of file traps.c.

```
{
  memset((unsigned char *)pok_sparc_isr, 0x0, sizeof (pok_sparc_isr));
  return POK_ERRNO_OK;
}
```

4.44.5 Variable Documentation

4.44.5.1 sparc_traps_handler pok_sparc_isr[256]

Interrupt subroutine table.

Definition at line 34 of file traps.c.

4.45 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/x86/event.c File Reference

`#include <libc.h> #include <types.h> #include <errno.- h> #include <core/syscall.h> #include "event.h" #include "sysdesc.h"`

Defines

- #define IDT_SIZE 256

Functions

- pok_ret_t pok_event_init ()
- pok_ret_t pok_idt_init ()
- void pok_idt_set_gate (uint16_t index, uint16_t segsel, uint32_t offset, e_idte_- type t, int dpl)

Variables

- idt_entry_t pok_idt [IDT_SIZE]

4.45.1 Define Documentation

4.45.1.1 #define IDT_SIZE 256

Definition at line 27 of file event.c.

4.45.2 Function Documentation

4.45.2.1 pok_ret_t pok_event_init ( )

Definition at line 31 of file event.c.

```
{
   pok_idt_init ();

#if defined (POK_NEEDS_DEBUG) || defined (POK_NEEDS_ERROR_HANDLING)
   pok_exception_init ();
#endif

   pok_syscall_init ();

   return (POK_ERRNO_OK);
}
```

4.45.2.2 pok_ret_t pok_idt_init ( )

Definition at line 44 of file event.c.

```
{
   sysdesc_t sysdesc;

   /* Clear table */
   memset(pok_idt, 0, sizeof (idt_entry_t) * IDT_SIZE);

   /* Load IDT */
   sysdesc.limit = sizeof (pok_idt);
   sysdesc.base = (uint32_t)pok_idt;

   asm ("lidt %0"
        :
        : "m" (sysdesc));

  return (POK_ERRNO_OK);
}
```

4.45.2.3 void pok_idt_set_gate (uint16_t *index,* uint16_t *segsel,* uint32_t *offset,* e_idte_type *t,* int *dpl*)

Definition at line 62 of file event.c.

```
{
   pok_idt[index].offset_low   = (offset) & 0xFFFF;
   pok_idt[index].offset_high  = (offset >> 16) & 0xFFFF;
   pok_idt[index].segsel       = segsel;
   pok_idt[index].dpl          = dpl;
   pok_idt[index].type         = t;
   pok_idt[index].d            = 1;
   pok_idt[index].res0         = 0; /* reserved */
   pok_idt[index].res1         = 0; /* reserved */
   pok_idt[index].present   = 1;
}
```

4.45.3 Variable Documentation

4.45.3.1 idt_entry_t pok_idt[IDT_SIZE]

Definition at line 29 of file event.c.

4.46 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/x86/event.h File Reference

```
#include <types.h>      #include <arch/x86/interrupt.h>×
#include "gdt.h"
```

Data Structures

- struct __attribute__

Defines

- #define EXCEPTION_DIVIDE_ERROR 0
- #define EXCEPTION_DEBUG 1
- #define EXCEPTION_NMI 2
- #define EXCEPTION_BREAKPOINT 3
- #define EXCEPTION_OVERFLOW 4
- #define EXCEPTION_BOUNDRANGE 5
- #define EXCEPTION_INVALIDOPCODE 6
- #define EXCEPTION_NOMATH_COPROC 7
- #define EXCEPTION_DOUBLEFAULT 8
- #define EXCEPTION_COPSEG_OVERRUN 9
- #define EXCEPTION_INVALID_TSS 10
- #define EXCEPTION_SEGMENT_NOT_PRESENT 11
- #define EXCEPTION_STACKSEG_FAULT 12
- #define EXCEPTION_GENERAL_PROTECTION 13
- #define EXCEPTION_PAGEFAULT 14
- #define EXCEPTION_RESERVED 15
- #define EXCEPTION_FPU_FAULT 16
- #define EXCEPTION_ALIGNEMENT_CHECK 17
- #define EXCEPTION_MACHINE_CHECK 18
- #define EXCEPTION_SIMD_FAULT 19

Typedefs

- typedef enum e_idte_type e_idte_type

Enumerations

- enum e_idte_type { IDTE_TASK = 5, IDTE_INTERRUPT = 6, IDTE_TRAP = 7 }

Functions

- void pok_idt_set_gate (uint16_t index, uint16_t segsel, uint32_t offset, e_idte_type t, int dpl)
- pok_ret_t pok_idt_init ()
- pok_ret_t pok_exception_init ()
- pok_ret_t pok_event_init ()
- pok_ret_t pok_syscall_init ()

4.46.1 Define Documentation

4.46.1.1 #define EXCEPTION_ALIGNEMENT_CHECK 17

Definition at line 63 of file event.h.

4.46.1.2 #define EXCEPTION_BOUNDRANGE 5

Definition at line 51 of file event.h.

4.46.1.3 #define EXCEPTION_BREAKPOINT 3

Definition at line 49 of file event.h.

4.46.1.4 #define EXCEPTION_COPSEG_OVERRUN 9

Definition at line 55 of file event.h.

4.46.1.5 #define EXCEPTION_DEBUG 1

Definition at line 47 of file event.h.

4.46.1.6 #define EXCEPTION_DIVIDE_ERROR 0

Definition at line 46 of file event.h.

4.46.1.7 #define EXCEPTION_DOUBLEFAULT 8

Definition at line 54 of file event.h.

4.46.1.8 #define EXCEPTION_FPU_FAULT 16

Definition at line 62 of file event.h.

4.46.1.9 #define EXCEPTION_GENERAL_PROTECTION 13

Definition at line 59 of file event.h.

4.46.1.10 #define EXCEPTION_INVALID_TSS 10

Definition at line 56 of file event.h.

4.46.1.11 #define EXCEPTION_INVALIDOPCODE 6

Definition at line 52 of file event.h.

4.46.1.12 #define EXCEPTION_MACHINE_CHECK 18

Definition at line 64 of file event.h.

4.46.1.13 #define EXCEPTION_NMI 2

Definition at line 48 of file event.h.

4.46.1.14 #define EXCEPTION_NOMATH_COPROC 7

Definition at line 53 of file event.h.

4.46.1.15 #define EXCEPTION_OVERFLOW 4

Definition at line 50 of file event.h.

4.46.1.16 #define EXCEPTION_PAGEFAULT 14

Definition at line 60 of file event.h.

4.46.1.17 #define EXCEPTION_RESERVED 15

Definition at line 61 of file event.h.

4.46.1.18 #define EXCEPTION_SEGMENT_NOT_PRESENT 11

Definition at line 57 of file event.h.

4.46.1.19 #define EXCEPTION_SIMD_FAULT 19

Definition at line 65 of file event.h.

4.46.1.20 #define EXCEPTION_STACKSEG_FAULT 12

Definition at line 58 of file event.h.

4.46.2 Typedef Documentation

4.46.2.1 typedef enum e_idte_type e_idte_type

4.46.3 Enumeration Type Documentation

4.46.3.1 enum e_idte_type

Enumerator:

IDTE_TASK

IDTE_INTERRUPT

IDTE_TRAP

Definition at line 26 of file event.h.

```
{
  IDTE_TASK = 5,
  IDTE_INTERRUPT = 6,
  IDTE_TRAP = 7
} e_idte_type;
```

4.46.4 Function Documentation

4.46.4.1 pok_ret_t pok_event_init ( )

Definition at line 31 of file event.c.

```
{
   pok_idt_init ();

#if defined (POK_NEEDS_DEBUG) || defined (POK_NEEDS_ERROR_HANDLING)
   pok_exception_init ();
#endif

   pok_syscall_init ();

   return (POK_ERRNO_OK);
}
```

4.46.4.2 pok_ret_t pok_exception_init ( )

4.46.4.3 pok_ret_t pok_idt_init ( )

Definition at line 44 of file event.c.

```
{
   sysdesc_t sysdesc;

   /* Clear table */
   memset(pok_idt, 0, sizeof (idt_entry_t) * IDT_SIZE);

   /* Load IDT */
   sysdesc.limit = sizeof (pok_idt);
   sysdesc.base = (uint32_t)pok_idt;

   asm ("lidt %0"
        :
        : "m" (sysdesc));

  return (POK_ERRNO_OK);
}
```

4.46.4.4 void pok_idt_set_gate (uint16_t *index,* uint16_t *segsel,* uint32_t *offset,* e_idte_type *t,* int *dpl*)

Definition at line 62 of file event.c.

```
{
   pok_idt[index].offset_low   = (offset) & 0xFFFF;
   pok_idt[index].offset_high  = (offset >> 16) & 0xFFFF;
   pok_idt[index].segsel       = segsel;
   pok_idt[index].dpl          = dpl;
   pok_idt[index].type         = t;
   pok_idt[index].d            = 1;
   pok_idt[index].res0         = 0; /* reserved */
   pok_idt[index].res1         = 0; /* reserved */
   pok_idt[index].present   = 1;
}
```

4.46.4.5 pok_ret_t pok_syscall_init ( )

Init system calls

Definition at line 83 of file syscalls.c.

```
{
   pok_idt_set_gate (POK_SYSCALL_INT_NUMBER,
                     GDT_CORE_CODE_SEGMENT << 3,
                     (uint32_t) syscall_gate,
                     IDTE_INTERRUPT,
                     3);

   return (POK_ERRNO_OK);
}
```

4.47 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/x86/exceptions.c File Reference

4.47.1 Detailed Description

Author

Julian Pidancet

Definition in file exceptions.c.

4.48 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/x86/gdt.c File Reference

`#include <libc.h> #include <types.h> #include <errno.h>`×
`#include "gdt.h" #include "sysdesc.h" #include "tss.h"`

Defines

- #define POK_CONFIG_NB_THREADS 0
- #define POK_CONFIG_NB_PARTITIONS 0
- #define GDT_SIZE 256

Functions

- pok_ret_t pok_gdt_init ()
- int pok_tss_init ()
- void tss_set_esp0 (uint32_t esp0)
- void gdt_set_segment (uint16_t index, uint32_t base_address, uint32_t limit, e_-gdte_type t, int dpl)
- void gdt_set_system (uint16_t index, uint32_t base_address, uint32_t limit, e_-gdte_type t, int dpl)
- void gdt_enable (uint16_t index)
- void gdt_disable (uint16_t index)

Variables

- gdt_entry_t pok_gdt [GDT_SIZE]
- tss_t pok_tss

4.48.1 Define Documentation

4.48.1.1 #define GDT_SIZE 256

Definition at line 35 of file gdt.c.

4.48.1.2 #define POK_CONFIG_NB_PARTITIONS 0

Definition at line 32 of file gdt.c.

4.48.1.3 #define POK_CONFIG_NB_THREADS 0

Definition at line 28 of file gdt.c.

4.48.2 Function Documentation

4.48.2.1 void gdt_disable (uint16_t *index*)

Definition at line 155 of file gdt.c.

```
{
   pok_gdt[index].present = 0;
}
```

4.48.2.2 void gdt_enable (uint16_t *index*)

Definition at line 150 of file gdt.c.

```
{
   pok_gdt[index].present = 1;
}
```

4.48.2.3 void gdt_set_segment (uint16_t *index*, uint32_t *base_address*, uint32_t *limit*, e_gdte_type *t*, int *dpl*)

Definition at line 99 of file gdt.c.

```
{
   if (limit > (1 << 20)) /* 4K granularity */
   {
      pok_gdt[index].limit_low = (limit >> 12) & 0xFFFF;
      pok_gdt[index].limit_high = (limit >> 28) & 0xF;
      pok_gdt[index].granularity = 1;
   }
   else /* 1B granularity */
   {
      pok_gdt[index].limit_low = limit & 0xFFFF;
      pok_gdt[index].limit_high = (limit >> 16) & 0xFF;
      pok_gdt[index].granularity = 0;
   }

   pok_gdt[index].base_low = base_address & 0xFFFFFF;
   pok_gdt[index].base_high = (base_address >> 24) & 0xFF;

   pok_gdt[index].type = t & 0xF;
   pok_gdt[index].dpl = dpl & 0x3;

   pok_gdt[index].s = 1;                      /* Segment is data/code type */
   pok_gdt[index].present = 1;
   pok_gdt[index].available = 0;
   pok_gdt[index].op_size = 1;        /* We work on 32 bits segments */
}
```

4.48.2.4 void gdt_set_system (uint16_t *index,* uint32_t *base_address,* uint32_t *limit,* e_gdte_type *t,* int *dpl*)

Definition at line 130 of file gdt.c.

```
{
   pok_gdt[index].limit_low = limit & 0xFFFF;
   pok_gdt[index].limit_high = (limit >> 16) & 0xFF;
   pok_gdt[index].base_low = base_address & 0xFFFFFF;
   pok_gdt[index].base_high = (base_address >> 24) & 0xFF;

   pok_gdt[index].type = t & 0xF;
   pok_gdt[index].dpl = dpl & 0x3;

   pok_gdt[index].s = 0;                        /* Segment is system type */
   pok_gdt[index].present = 1;
   pok_gdt[index].available = 0;
   pok_gdt[index].op_size = 0;
}
```

4.48.2.5 pok_ret_t pok_gdt_init ( )

Definition at line 41 of file gdt.c.

```
{
   sysdesc_t sysdesc;

   /* Set null descriptor and clear table */
   memset(pok_gdt, 0, sizeof (gdt_entry_t) * GDT_SIZE);

   /* Set kernel descriptors */
   gdt_set_segment(GDT_CORE_CODE_SEGMENT, 0, ~0UL, GDTE_CODE, 0);
   gdt_set_segment(GDT_CORE_DATA_SEGMENT, 0, ~0UL, GDTE_DATA, 0);

   /* Load GDT */
   sysdesc.limit = sizeof (pok_gdt);
   sysdesc.base = (uint32_t)pok_gdt;

   asm ("lgdt %0"
         :
         : "m" (sysdesc));

   /* Reload Segments */
   asm ("ljmp %0, $1f   \n"
         "1:            \n"
         "mov %1, %%ax  \n"
         "mov %%ax, %%ds        \n"
         "mov %%ax, %%es        \n"
         "mov %%ax, %%fs        \n"
         "mov %%ax, %%gs        \n"
         "mov %%ax, %%ss        \n"
         :
         : "i" (GDT_CORE_CODE_SEGMENT << 3),
         "i" (GDT_CORE_DATA_SEGMENT << 3)
         : "eax");

   pok_tss_init();
   return (POK_ERRNO_OK);
}
```

4.48.2.6 int pok_tss_init ( )

Definition at line 79 of file gdt.c.

```
{
   uint16_t sel = GDT_BUILD_SELECTOR(GDT_TSS_SEGMENT, 0, 0);

   memset(&pok_tss, 0, sizeof (tss_t));

   pok_tss.ss0 = GDT_BUILD_SELECTOR(GDT_CORE_DATA_SEGMENT, 0, 0);

   gdt_set_system(GDT_TSS_SEGMENT, (uint32_t)&pok_tss,
         sizeof (tss_t), GDTE_TSS, 0);

   asm ("ltr %0" : :"m"(sel));
   return (POK_ERRNO_OK);
}
```

4.48.2.7 void tss_set_esp0 (uint32_t *esp0*)

Definition at line 94 of file gdt.c.

```
{
   pok_tss.esp0 = esp0;
}
```

4.48.3 Variable Documentation

4.48.3.1 gdt_entry_t pok_gdt[GDT_SIZE]

Definition at line 37 of file gdt.c.

4.48.3.2 tss_t pok_tss

Definition at line 39 of file gdt.c.

4.49 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/x86/gdt.h File Reference

`#include <types.h>`

Data Structures

- struct __attribute__

Defines

- #define GDT_CORE_CODE_SEGMENT 1
- #define GDT_CORE_DATA_SEGMENT 2
- #define GDT_TSS_SEGMENT 3
- #define GDT_PARTITION_CODE_SEGMENT(partition_id) (4 + 2 ∗ partition_id)
- #define GDT_PARTITION_DATA_SEGMENT(partition_id) (4 + 2 ∗ partition_id + 1)
- #define GDT_BUILD_SELECTOR(seg, local, rpl) ((seg << 3) | ((local & 0x1) << 2) | (rpl & 0x3))

Typedefs

- typedef enum e_gdte_type e_gdte_type

Enumerations

- enum e_gdte_type { GDTE_CODE = 0xB, GDTE_DATA = 0x3, GDTE_TSS = 0x9 }

Functions

- pok_ret_t pok_gdt_init ()
- int pok_tss_init ()
- void tss_set_esp0 (uint32_t esp0)
- void gdt_set_segment (uint16_t index, uint32_t base_address, uint32_t limit, e_gdte_type t, int dpl)
- void gdt_set_system (uint16_t index, uint32_t base_address, uint32_t limit, e_gdte_type t, int dpl)
- void gdt_enable (uint16_t index)
- void gdt_disable (uint16_t index)

4.49.1 Define Documentation

4.49.1.1 #define GDT_BUILD_SELECTOR(*seg*, *local*, *rpl*) ((seg << 3) | ((local & 0x1) << 2) | (rpl & 0x3))

Definition at line 52 of file gdt.h.

4.49.1.2 #define GDT_CORE_CODE_SEGMENT 1

Definition at line 45 of file gdt.h.

4.49.1.3 #define GDT_CORE_DATA_SEGMENT 2

Definition at line 46 of file gdt.h.

4.49.1.4 #define GDT_PARTITION_CODE_SEGMENT(*partition_id*) (4 + 2 ∗ partition_id)

Definition at line 49 of file gdt.h.

4.49.1.5 #define GDT_PARTITION_DATA_SEGMENT(*partition_id*) (4 + 2 ∗ partition_id + 1)

Definition at line 50 of file gdt.h.

4.49.1.6 #define GDT_TSS_SEGMENT 3

Definition at line 47 of file gdt.h.

4.49.2 Typedef Documentation

4.49.2.1 typedef enum e_gdte_type e_gdte_type

4.49.3 Enumeration Type Documentation

4.49.3.1 enum e_gdte_type

Enumerator:

GDTE_CODE

GDTE_DATA

GDTE_TSS

Definition at line 23 of file gdt.h.

```
{
  GDTE_CODE = 0xB,
  GDTE_DATA = 0x3,
  GDTE_TSS = 0x9
} e_gdte_type;
```

4.49.4 Function Documentation

4.49.4.1 void gdt_disable (uint16_t *index*)

Definition at line 155 of file gdt.c.

```
{
   pok_gdt[index].present = 0;
}
```

4.49.4.2 void gdt_enable (uint16_t *index*)

Definition at line 150 of file gdt.c.

```
{
   pok_gdt[index].present = 1;
}
```

4.49.4.3 void gdt_set_segment (uint16_t *index,* uint32_t *base_address,* uint32_t *limit,* e_gdte_type *t,* int *dpl*)

Definition at line 99 of file gdt.c.

```
{
   if (limit > (1 << 20)) /* 4K granularity */
   {
      pok_gdt[index].limit_low = (limit >> 12) & 0xFFFF;
      pok_gdt[index].limit_high = (limit >> 28) & 0xF;
      pok_gdt[index].granularity = 1;
   }
   else /* 1B granularity */
   {
      pok_gdt[index].limit_low = limit & 0xFFFF;
      pok_gdt[index].limit_high = (limit >> 16) & 0xFF;
      pok_gdt[index].granularity = 0;
   }

   pok_gdt[index].base_low = base_address & 0xFFFFFF;
   pok_gdt[index].base_high = (base_address >> 24) & 0xFF;

   pok_gdt[index].type = t & 0xF;
   pok_gdt[index].dpl = dpl & 0x3;

   pok_gdt[index].s = 1;                      /* Segment is data/code type */
   pok_gdt[index].present = 1;
   pok_gdt[index].available = 0;
   pok_gdt[index].op_size = 1;        /* We work on 32 bits segments */
}
```

4.49.4.4 void gdt_set_system (uint16_t *index*, uint32_t *base_address*, uint32_t *limit*, e_gdte_type *t*, int *dpl*)

Definition at line 130 of file gdt.c.

```
{
   pok_gdt[index].limit_low = limit & 0xFFFF;
   pok_gdt[index].limit_high = (limit >> 16) & 0xFF;
   pok_gdt[index].base_low = base_address & 0xFFFFFF;
   pok_gdt[index].base_high = (base_address >> 24) & 0xFF;

   pok_gdt[index].type = t & 0xF;
   pok_gdt[index].dpl = dpl & 0x3;

   pok_gdt[index].s = 0;                       /* Segment is system type */
   pok_gdt[index].present = 1;
   pok_gdt[index].available = 0;
   pok_gdt[index].op_size = 0;
}
```

4.49.4.5 pok_ret_t pok_gdt_init ( )

Definition at line 41 of file gdt.c.

```
{
   sysdesc_t sysdesc;

   /* Set null descriptor and clear table */
   memset(pok_gdt, 0, sizeof (gdt_entry_t) * GDT_SIZE);

   /* Set kernel descriptors */
   gdt_set_segment(GDT_CORE_CODE_SEGMENT, 0, ~0UL, GDTE_CODE, 0);
   gdt_set_segment(GDT_CORE_DATA_SEGMENT, 0, ~0UL, GDTE_DATA, 0);

   /* Load GDT */
   sysdesc.limit = sizeof (pok_gdt);
   sysdesc.base = (uint32_t)pok_gdt;

   asm ("lgdt %0"
         :
         : "m" (sysdesc));

   /* Reload Segments */
   asm ("ljmp %0, $1f   \n"
         "1:            \n"
         "mov %1, %%ax  \n"
         "mov %%ax, %%ds        \n"
         "mov %%ax, %%es        \n"
         "mov %%ax, %%fs        \n"
         "mov %%ax, %%gs        \n"
         "mov %%ax, %%ss        \n"
         :
         : "i" (GDT_CORE_CODE_SEGMENT << 3),
         "i" (GDT_CORE_DATA_SEGMENT << 3)
         : "eax");

   pok_tss_init();
```

```
    return (POK_ERRNO_OK);
}
```

4.49.4.6 int pok_tss_init ( )

Definition at line 79 of file gdt.c.

```
{
    uint16_t sel = GDT_BUILD_SELECTOR(GDT_TSS_SEGMENT, 0, 0);

    memset(&pok_tss, 0, sizeof (tss_t));

    pok_tss.ss0 = GDT_BUILD_SELECTOR(GDT_CORE_DATA_SEGMENT, 0, 0);

    gdt_set_system(GDT_TSS_SEGMENT, (uint32_t)&pok_tss,
            sizeof (tss_t), GDTE_TSS, 0);

    asm ("ltr %0" : :"m"(sel));
    return (POK_ERRNO_OK);
}
```

4.49.4.7 void tss_set_esp0 (uint32_t *esp0*)

Definition at line 94 of file gdt.c.

```
{
    pok_tss.esp0 = esp0;
}
```

4.50 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/x86/interrupt.c File Reference

`#include <arch/x86/interrupt.h>`

Functions

- void update_tss (interrupt_frame ∗frame)

4.50.1 Function Documentation

4.50.1.1 void update_tss (interrupt_frame ∗ *frame*)

Definition at line 20 of file interrupt.c.

```
{
  uint32_t* esp0 = (&pok_tss) + 1;

  if ((frame->cs & 0xffff) != 0x8)
  {
    *esp0 = (uint32_t)frame + sizeof (interrupt_frame);
  }
}
```

4.51 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/x86/pci.c File Reference

4.52 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/x86/sysdesc.h File Reference

Data Structures

- struct __attribute__

4.53 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/x86/tss.h File Reference

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

Data Structures

- struct __attribute__

4.54 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/x86/types.h File Reference

Defines

- #define __POK_X86_TYPES_H__

Typedefs

- typedef unsigned short uint8_t
- typedef unsigned short uint16_t
- typedef unsigned int uint32_t
- typedef unsigned long long uint64_t

- typedef short int8_t
- typedef short int16_t
- typedef signed long long int64_t
- typedef unsigned int size_t
- typedef unsigned long int intptr_t

4.54.1 Define Documentation

4.54.1.1 #define __POK_X86_TYPES_H__

Definition at line 19 of file types.h.

4.54.2 Typedef Documentation

4.54.2.1 typedef short int16_t

Definition at line 27 of file types.h.

4.54.2.2 typedef signed long long int64_t

Definition at line 28 of file types.h.

4.54.2.3 typedef short int8_t

Definition at line 26 of file types.h.

4.54.2.4 typedef unsigned long int intptr_t

Definition at line 31 of file types.h.

4.54.2.5 typedef unsigned int size_t

Definition at line 30 of file types.h.

4.54.2.6 typedef unsigned short uint16_t

Definition at line 22 of file types.h.

4.54.2.7 typedef unsigned int uint32_t

Definition at line 23 of file types.h.

4.54.2.8 typedef unsigned long long uint64_t

Definition at line 24 of file types.h.

4.54.2.9 typedef unsigned short uint8_t

Definition at line 21 of file types.h.

4.55 /home/matias/projet/stage-pok/pok/trunk/kernel/include/arch/sparc/types.h File Reference

Typedefs

- typedef unsigned char uint8_t
- typedef unsigned short uint16_t
- typedef unsigned int uint32_t
- typedef unsigned long long uint64_t
- typedef char int8_t
- typedef short int16_t
- typedef signed long long int64_t
- typedef unsigned int size_t
- typedef unsigned long int intptr_t

4.55.1 Typedef Documentation

4.55.1.1 typedef short int16_t

Definition at line 27 of file types.h.

4.55.1.2 typedef signed long long int64_t

Definition at line 28 of file types.h.

4.55.1.3 typedef char int8_t

Definition at line 26 of file types.h.

4.55.1.4 typedef unsigned long int intptr_t

Definition at line 31 of file types.h.

4.55.1.5 typedef unsigned int **size_t**

Definition at line 30 of file types.h.

4.55.1.6 typedef unsigned short **uint16_t**

Definition at line 22 of file types.h.

4.55.1.7 typedef unsigned int **uint32_t**

Definition at line 23 of file types.h.

4.55.1.8 typedef unsigned long long **uint64_t**

Definition at line 24 of file types.h.

4.55.1.9 typedef unsigned char **uint8_t**

Definition at line 21 of file types.h.

4.56 /home/matias/projet/stage-pok/pok/trunk/kernel/include/arch/x86/types.h File Reference

Defines

- #define __POK_X86_TYPES_H__

Typedefs

- typedef unsigned short uint8_t
- typedef unsigned short uint16_t
- typedef unsigned int uint32_t
- typedef unsigned long long uint64_t
- typedef short int8_t
- typedef short int16_t
- typedef signed long long int64_t
- typedef unsigned int size_t
- typedef unsigned long int intptr_t

4.56.1 Define Documentation

4.56.1.1 #define __POK_X86_TYPES_H__

Definition at line 19 of file types.h.

4.56.2 Typedef Documentation

4.56.2.1 typedef short int16_t

Definition at line 27 of file types.h.

4.56.2.2 typedef signed long long int64_t

Definition at line 28 of file types.h.

4.56.2.3 typedef short int8_t

Definition at line 26 of file types.h.

4.56.2.4 typedef unsigned long int intptr_t

Definition at line 31 of file types.h.

4.56.2.5 typedef unsigned int size_t

Definition at line 30 of file types.h.

4.56.2.6 typedef unsigned short uint16_t

Definition at line 22 of file types.h.

4.56.2.7 typedef unsigned int uint32_t

Definition at line 23 of file types.h.

4.56.2.8 typedef unsigned long long uint64_t

Definition at line 24 of file types.h.

4.56.2.9 typedef unsigned short uint8_t

Definition at line 21 of file types.h.

4.57 /home/matias/projet/stage-pok/pok/trunk/kernel/include/types.h File Reference

```
#include <arch/x86/types.h>
```

Defines

- #define NULL 0
- #define FALSE 0
- #define TRUE 1
- #define bool_t int
- #define pok_bool_t int

Typedefs

- typedef uint32_t pok_port_size_t
- typedef uint8_t pok_port_direction_t
- typedef uint8_t pok_port_kind_t
- typedef uint8_t pok_queueing_discipline_t
- typedef uint8_t pok_port_id_t
- typedef uint8_t pok_size_t
- typedef uint8_t pok_range_t
- typedef uint8_t pok_buffer_id_t
- typedef uint8_t pok_blackboard_id_t
- typedef uint8_t pok_lockobj_id_t
- typedef uint8_t pok_sem_id_t
- typedef uint8_t pok_event_id_t
- typedef uint8_t pok_partition_id_t
- typedef uint16_t pok_sem_value_t

4.57.1 Define Documentation

4.57.1.1 #define bool_t int

Definition at line 30 of file types.h.

4.57.1.2 #define FALSE 0

Definition at line 28 of file types.h.

4.57.1.3 #define NULL 0

Definition at line 27 of file types.h.

4.57.1.4 #define **pok_bool_t** int

Definition at line 31 of file types.h.

4.57.1.5 #define **TRUE** 1

Definition at line 29 of file types.h.

4.57.2 Typedef Documentation

4.57.2.1 typedef **uint8_t pok_blackboard_id_t**

Definition at line 41 of file types.h.

4.57.2.2 typedef **uint8_t pok_buffer_id_t**

Definition at line 40 of file types.h.

4.57.2.3 typedef **uint8_t pok_event_id_t**

Definition at line 44 of file types.h.

4.57.2.4 typedef **uint8_t pok_lockobj_id_t**

Definition at line 42 of file types.h.

4.57.2.5 typedef **uint8_t pok_partition_id_t**

Definition at line 45 of file types.h.

4.57.2.6 typedef **uint8_t pok_port_direction_t**

Definition at line 34 of file types.h.

4.57.2.7 typedef **uint8_t pok_port_id_t**

Definition at line 37 of file types.h.

4.57.2.8 typedef **uint8_t pok_port_kind_t**

Definition at line 35 of file types.h.

4.57.2.9 typedef uint32_t pok_port_size_t

Definition at line 33 of file types.h.

4.57.2.10 typedef uint8_t pok_queueing_discipline_t

Definition at line 36 of file types.h.

4.57.2.11 typedef uint8_t pok_range_t

Definition at line 39 of file types.h.

4.57.2.12 typedef uint8_t pok_sem_id_t

Definition at line 43 of file types.h.

4.57.2.13 typedef uint16_t pok_sem_value_t

Definition at line 46 of file types.h.

4.57.2.14 typedef uint8_t pok_size_t

Definition at line 38 of file types.h.

4.58 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/x86/x86-qemu/debug.c File Reference

4.59 /home/matias/projet/stage-pok/pok/trunk/kernel/core/debug.c File Reference

4.60 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/x86/x86-qemu/pic.c File Reference

`#include <types.h> #include <errno.h> #include <arch/x86/ioports.-h> #include "pic.h"`

Functions

- int pok_pic_init ()
- int pok_pic_mask (uint8_t irq)

- int pok_pic_unmask (uint8_t irq)
- void pok_pic_eoi (uint8_t irq)

4.60.1 Function Documentation

4.60.1.1 void pok_pic_eoi (uint8_t *irq*)

Definition at line 90 of file pic.c.

```
{
   if (irq >= 8)
   {
      outb (PIC_SLAVE_BASE, 0x20);
   }

   outb (PIC_MASTER_BASE, 0x20);
}
```

4.60.1.2 int pok_pic_init ( )

Definition at line 25 of file pic.c.

```
{
   outb (PIC_MASTER_BASE, PIC_MASTER_ICW1);
   outb (PIC_SLAVE_BASE, PIC_SLAVE_ICW1);

   outb (PIC_MASTER_BASE + 1, PIC_MASTER_ICW2);
   outb (PIC_SLAVE_BASE + 1, PIC_SLAVE_ICW2);

   outb (PIC_MASTER_BASE + 1, PIC_MASTER_ICW3);
   outb (PIC_SLAVE_BASE + 1, PIC_SLAVE_ICW3);

   outb (PIC_MASTER_BASE + 1, PIC_MASTER_ICW4);
   outb (PIC_SLAVE_BASE + 1, PIC_SLAVE_ICW4);

   /* Mask everything */
   outb (PIC_MASTER_BASE + 1, 0xfb);
   outb (PIC_SLAVE_BASE + 1, 0xff);

   return (POK_ERRNO_OK);
}
```

4.60.1.3 int pok_pic_mask (uint8_t *irq*)

Definition at line 46 of file pic.c.

```
{
   uint8_t mask;

   if (irq > 15)
   {
      return (POK_ERRNO_EINVAL);
   }

   if (irq < 8)
   {
      mask = inb (PIC_MASTER_BASE + 1);
      outb (PIC_MASTER_BASE + 1, mask | (1 << irq));
   }
   else
   {
      mask = inb (PIC_SLAVE_BASE + 1);
      outb (PIC_SLAVE_BASE + 1, mask | (1 << (irq - 8)));
   }

   return (POK_ERRNO_OK);
}
```

4.60.1.4 int pok_pic_unmask (uint8_t *irq*)

Definition at line 69 of file pic.c.

```
{
   uint8_t mask;

   if (irq > 15)
      return (POK_ERRNO_EINVAL);

   if (irq < 8)
   {
      mask = inb(PIC_MASTER_BASE + 1);
      outb(PIC_MASTER_BASE + 1, mask & ~(1 << irq));
   }
   else
   {
      mask = inb(PIC_SLAVE_BASE + 1);
      outb(PIC_SLAVE_BASE + 1, mask & ~(1 << (irq - 8)));
   }

   return (POK_ERRNO_OK);
}
```

4.61 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/x86/x86-qemu/pic.h File Reference

Defines

- #define PIC_MASTER_BASE 0x20
- #define PIC_SLAVE_BASE 0xa0
- #define PIC_MASTER_ICW1 0x11
- #define PIC_MASTER_ICW2 0x20
- #define PIC_MASTER_ICW3 0x04
- #define PIC_MASTER_ICW4 0x01

- #define PIC_SLAVE_ICW1 0x11
- #define PIC_SLAVE_ICW2 0x28
- #define PIC_SLAVE_ICW3 0x02
- #define PIC_SLAVE_ICW4 0x01

Functions

- int pok_pic_init ()
- int pok_pic_mask (uint8_t irq)
- int pok_pic_unmask (uint8_t irq)
- void pok_pic_eoi (uint8_t irq)

4.61.1 Define Documentation

4.61.1.1 #define **PIC_MASTER_BASE** 0x20

Definition at line 21 of file pic.h.

4.61.1.2 #define **PIC_MASTER_ICW1** 0x11

Definition at line 24 of file pic.h.

4.61.1.3 #define **PIC_MASTER_ICW2** 0x20

Definition at line 25 of file pic.h.

4.61.1.4 #define **PIC_MASTER_ICW3** 0x04

Definition at line 26 of file pic.h.

4.61.1.5 #define **PIC_MASTER_ICW4** 0x01

Definition at line 27 of file pic.h.

4.61.1.6 #define **PIC_SLAVE_BASE** 0xa0

Definition at line 22 of file pic.h.

4.61.1.7 #define **PIC_SLAVE_ICW1** 0x11

Definition at line 29 of file pic.h.

4.61.1.8 #define PIC_SLAVE_ICW2 0x28

Definition at line 30 of file pic.h.

4.61.1.9 #define PIC_SLAVE_ICW3 0x02

Definition at line 31 of file pic.h.

4.61.1.10 #define PIC_SLAVE_ICW4 0x01

Definition at line 32 of file pic.h.

4.61.2 Function Documentation

4.61.2.1 void pok_pic_eoi (uint8_t *irq*)

Definition at line 90 of file pic.c.

```
{
   if (irq >= 8)
   {
      outb (PIC_SLAVE_BASE, 0x20);
   }

   outb (PIC_MASTER_BASE, 0x20);
}
```

4.61.2.2 int pok_pic_init ( )

Definition at line 25 of file pic.c.

```
{
   outb (PIC_MASTER_BASE, PIC_MASTER_ICW1);
   outb (PIC_SLAVE_BASE, PIC_SLAVE_ICW1);

   outb (PIC_MASTER_BASE + 1, PIC_MASTER_ICW2);
   outb (PIC_SLAVE_BASE + 1, PIC_SLAVE_ICW2);

   outb (PIC_MASTER_BASE + 1, PIC_MASTER_ICW3);
   outb (PIC_SLAVE_BASE + 1, PIC_SLAVE_ICW3);

   outb (PIC_MASTER_BASE + 1, PIC_MASTER_ICW4);
   outb (PIC_SLAVE_BASE + 1, PIC_SLAVE_ICW4);

   /* Mask everything */
   outb (PIC_MASTER_BASE + 1, 0xfb);
   outb (PIC_SLAVE_BASE + 1, 0xff);

   return (POK_ERRNO_OK);
}
```

4.61.2.3 int pok_pic_mask (uint8_t *irq*)

Definition at line 46 of file pic.c.

```
{
   uint8_t mask;

   if (irq > 15)
   {
      return (POK_ERRNO_EINVAL);
   }

   if (irq < 8)
   {
      mask = inb (PIC_MASTER_BASE + 1);
      outb (PIC_MASTER_BASE + 1, mask | (1 << irq));
   }
   else
   {
      mask = inb (PIC_SLAVE_BASE + 1);
      outb (PIC_SLAVE_BASE + 1, mask | (1 << (irq - 8)));
   }

   return (POK_ERRNO_OK);
}
```

4.61.2.4 int pok_pic_unmask (uint8_t *irq*)

Definition at line 69 of file pic.c.

```
{
   uint8_t mask;

   if (irq > 15)
      return (POK_ERRNO_EINVAL);

   if (irq < 8)
   {
      mask = inb(PIC_MASTER_BASE + 1);
      outb(PIC_MASTER_BASE + 1, mask & ~(1 << irq));
   }
   else
   {
      mask = inb(PIC_SLAVE_BASE + 1);
      outb(PIC_SLAVE_BASE + 1, mask & ~(1 << (irq - 8)));
   }

   return (POK_ERRNO_OK);
}
```

4.62 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/x86/x86-qemu/pit.c File Reference

`#include <errno.h>` `#include <bsp.h>` `#include <core/time.-h>` `#include <core/sched.h>` `#include <arch/x86/ioports.-`

`h> #include <arch/x86/interrupt.h> #include "pic.h"` ×
`#include "pit.h"`

Defines

- #define OSCILLATOR_RATE 1193180 /∗∗ The oscillation rate of x86 clock ∗/
- #define PIT_BASE 0x40
- #define PIT_IRQ 0

Functions

- INTERRUPT_HANDLER (pit_interrupt)
- pok_ret_t pok_x86_qemu_timer_init ()

4.62.1 Define Documentation

4.62.1.1 #define OSCILLATOR_RATE 1193180 /∗∗ The oscillation rate of x86 clock ∗/

Definition at line 29 of file pit.c.

4.62.1.2 #define PIT_BASE 0x40

Definition at line 30 of file pit.c.

4.62.1.3 #define PIT_IRQ 0

Definition at line 31 of file pit.c.

4.62.2 Function Documentation

4.62.2.1 INTERRUPT_HANDLER (pit_interrupt)

Definition at line 33 of file pit.c.

```
{
   (void) frame;
   pok_pic_eoi (PIT_IRQ);
   CLOCK_HANDLER
}
```

4.62.2.2 pok_ret_t pok_x86_qemu_timer_init ( )

Definition at line 40 of file pit.c.

```
{
   uint16_t pit_freq;

   pit_freq = POK_TIMER_FREQUENCY;

   outb (PIT_BASE + 3, 0x34); /* Channel0, rate generator, Set LSB then MSB */
   outb (PIT_BASE, (OSCILLATOR_RATE / pit_freq) & 0xff);
   outb (PIT_BASE, ((OSCILLATOR_RATE / pit_freq) >> 8) & 0xff);

   pok_bsp_irq_register (PIT_IRQ, pit_interrupt);

   return (POK_ERRNO_OK);
}
```

4.63 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/x86/x86-qemu/pit.h File Reference

Functions

- pok_ret_t pok_x86_qemu_timer_init ()

4.63.1 Function Documentation

4.63.1.1 pok_ret_t pok_x86_qemu_timer_init ( )

Definition at line 40 of file pit.c.

```
{
   uint16_t pit_freq;

   pit_freq = POK_TIMER_FREQUENCY;

   outb (PIT_BASE + 3, 0x34); /* Channel0, rate generator, Set LSB then MSB */
   outb (PIT_BASE, (OSCILLATOR_RATE / pit_freq) & 0xff);
   outb (PIT_BASE, ((OSCILLATOR_RATE / pit_freq) >> 8) & 0xff);

   pok_bsp_irq_register (PIT_IRQ, pit_interrupt);

   return (POK_ERRNO_OK);
}
```

4.64 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/x86/x86-qemu/pm.c File Reference

```
#include <errno.h>      #include <arch/x86/multiboot.h>×
#include <types.h> #include "pm.h"
```

Defines

- #define ALIGN_UP(boundary, val) (val + (boundary - 1)) & (∼(boundary - 1))

Functions

- int pok_pm_init ()
- uint32_t pok_pm_sbrk (uint32_t increment)

Variables

- void ∗ __pok_begin
- void ∗ __pok_end
- uint32_t pok_multiboot_magic
- uint32_t pok_multiboot_info
- uint32_t pok_x86_pm_heap_start
- uint32_t pok_x86_pm_brk
- uint32_t pok_x86_pm_heap_end

4.64.1 Detailed Description

Author

Julian Pidancet
Julien Delange

Date

2008-2009

Definition in file pm.c.

4.64.2 Define Documentation

4.64.2.1 #define ALIGN_UP(*boundary,* *val*) (val + (boundary - 1)) & (∼(boundary - 1))

Definition at line 30 of file pm.c.

4.64.3 Function Documentation

4.64.3.1 int pok_pm_init ( )

Definition at line 44 of file pm.c.

```
{
  pok_multiboot_info_t* mbi;
  uint32_t              free_mem;

  mbi = (pok_multiboot_info_t*) pok_multiboot_info;

#ifdef POK_NEEDS_DMA
  free_mem = MEM_16MB;
#else
  free_mem = ALIGN_UP (4096, (uint32_t)(&__pok_end));
#endif

  pok_x86_pm_heap_start = pok_x86_pm_brk = free_mem;

  pok_x86_pm_heap_end = (uint32_t)(mbi->mem_upper * 1024);

  return (POK_ERRNO_OK);
}
```

4.64.3.2 uint32_t pok_pm_sbrk (uint32_t *increment*)

Allocation function, very basic, just allocate new memory space each time

Definition at line 68 of file pm.c.

```
{
  uint32_t addr;

  addr = pok_x86_pm_brk;

  pok_x86_pm_brk += increment;

  return (addr);
}
```

4.64.4 Variable Documentation

4.64.4.1 void∗ __pok_begin

4.64.4.2 void∗ __pok_end

4.64.4.3 uint32_t pok_multiboot_info

4.64.4.4 uint32_t pok_multiboot_magic

4.64.4.5 uint32_t pok_x86_pm_brk

Definition at line 40 of file pm.c.

4.64.4.6 uint32_t pok_x86_pm_heap_end

Definition at line 41 of file pm.c.

4.64.4.7 uint32_t pok_x86_pm_heap_start

Definition at line 39 of file pm.c.

4.65 /home/matias/projet/stage-pok/pok/trunk/kernel/arch/x86/x86-qemu/pm.h File Reference

Defines

- #define MEM_16MB 0x1000000

Functions

- int pok_pm_init ()
- uint32_t pok_pm_sbrk (uint32_t increment)

4.65.1 Define Documentation

4.65.1.1 #define MEM_16MB 0x1000000

Definition at line 21 of file pm.h.

4.65.2 Function Documentation

4.65.2.1 int pok_pm_init ( )

Definition at line 44 of file pm.c.

```
{
  pok_multiboot_info_t* mbi;
  uint32_t              free_mem;

  mbi = (pok_multiboot_info_t*) pok_multiboot_info;

#ifdef POK_NEEDS_DMA
  free_mem = MEM_16MB;
#else
  free_mem = ALIGN_UP (4096, (uint32_t)(&__pok_end));
#endif

  pok_x86_pm_heap_start = pok_x86_pm_brk = free_mem;

  pok_x86_pm_heap_end = (uint32_t)(mbi->mem_upper * 1024);

  return (POK_ERRNO_OK);
}
```

4.65.2.2 uint32_t pok_pm_sbrk (uint32_t *increment*)

Allocation function, very basic, just allocate new memory space each time

Definition at line 68 of file pm.c.

```
{
  uint32_t addr;

  addr = pok_x86_pm_brk;

  pok_x86_pm_brk += increment;

  return (addr);
}
```

4.66 /home/matias/projet/stage-pok/pok/trunk/kernel/core/boot.c File Reference

Boot function to start the kernel.

`#include <arch.h> #include <bsp.h> #include <core/time.h> #include <core/thread.h> #include <core/sched.h>` ×
`#include <core/partition.h> #include <middleware/port.h> #include <middleware/queue.h> #include <core/boot.h> #include <core/instrumentation.h>`

Functions

- void pok_boot ()

  Boot function that launch everything.

4.66.1 Detailed Description

Boot function to start the kernel.

Author

Julien Delange

Date

2008-2009

Definition in file boot.c.

4.66.2 Function Documentation

4.66.2.1 void pok_boot ( )

Boot function that launch everything.

This function load every service according to system requirements (the POK_NEEDS_* maccro). If we don't use partitioning service, we execute a main function. In that case, POK is acting like an executive, not a real kernel

Definition at line 37 of file boot.c.

```
{
   pok_arch_init();
   pok_bsp_init();

#if defined (POK_NEEDS_TIME) || defined (POK_NEEDS_SCHED) || defined
       (POK_NEEDS_THREADS)
   pok_time_init();
#endif

#ifdef POK_NEEDS_PARTITIONS
   pok_partition_init ();
#endif

#ifdef POK_NEEDS_THREADS
   pok_thread_init ();
#endif

#if defined (POK_NEEDS_SCHED) || defined (POK_NEEDS_THREADS)
   pok_sched_init ();
#endif

#if (defined POK_NEEDS_LOCKOBJ) || defined (POK_NEEDS_PORTS_QUEUEING) ||
       defined (POK_NEEDS_PORTS_SAMPLING)
   pok_lockobj_init ();
#endif
#if defined (POK_NEEDS_PORTS_QUEUEING) || defined (POK_NEEDS_PORTS_SAMPLING)
   pok_port_init ();
   pok_queue_init ();
#endif

#if defined (POK_NEEDS_DEBUG) || defined (POK_NEEDS_CONSOLE)
  pok_cons_write ("POK kernel initialized\n", 23);
#endif

#ifdef POK_NEEDS_INSTRUMENTATION
  uint32_t tmp;
   printf ("[INSTRUMENTATION][CHEDDAR] <event_table>\n");
   printf ("[INSTRUMENTATION][CHEDDAR] <processor>\n");
   printf ("[INSTRUMENTATION][CHEDDAR] <name>pok_kernel</name>\n");

   for (tmp = 0 ; tmp < POK_CONFIG_NB_THREADS ; tmp++)
   {
      printf ("[INSTRUMENTATION][CHEDDAR] <task_activation>   0   task
       %d</task_activation>\n", tmp);
   }
#endif

  pok_arch_preempt_enable();
#ifndef POK_NEEDS_PARTITIONS

  main ();
#endif
}
```

4.67 /home/matias/projet/stage-pok/pok/trunk/kernel/core/error.c File Reference

4.68 /home/matias/projet/stage-pok/pok/trunk/kernel/core/instrumentation.c File Reference

4.69 /home/matias/projet/stage-pok/pok/trunk/kernel/core/kernel.c File Reference

4.70 /home/matias/projet/stage-pok/pok/trunk/kernel/core/loader.c File Reference

4.70.1 Detailed Description

Author

Julian Pidancet
Julien Delange

Date

2008-2009

Contains all needed stuff to load partitions (elf files). This needs the partitioning service (POK_NEEDS_PARTITIONS must be defined) to work.

Definition in file loader.c.

4.71 /home/matias/projet/stage-pok/pok/trunk/kernel/core/lockobj.c File Reference

Provides functionnalities for locking functions (mutexes, semaphores and so on)

4.71.1 Detailed Description

Provides functionnalities for locking functions (mutexes, semaphores and so on)

Author

Julien Delange

This file contains the implementation code for mutexes, conditions and semaphores. This is implemented in the same file since the functionnalities does not differ so much.

Definition in file lockobj.c.

4.72 /home/matias/projet/stage-pok/pok/trunk/kernel/core/partition.c File Reference

This file provides functions for partitioning services.

4.72.1 Detailed Description

This file provides functions for partitioning services.

Author

Julien Delange

The definition of useful structures can be found in partition.h header file. To enable partitioning services, you must set the POK_NEEDS_PARTITIONS maccro.

Definition in file partition.c.

4.73 /home/matias/projet/stage-pok/pok/trunk/kernel/core/sched.c File Reference

4.74 /home/matias/projet/stage-pok/pok/trunk/kernel/core/syscall.c File Reference

`#include <bsp.h> #include <types.h> #include <libc.h> × #include <arch/x86/ioports.h> #include <arch/x86/pci.-h> #include <errno.h> #include <core/debug.h> #include <core/syscall.h> #include <core/partition.h> #include <core/thread.h> #include <core/lockobj.h> #include <core/time.-h> #include <core/error.h> #include <middleware/port.h>`

Functions

- pok_ret_t pok_core_syscall (const pok_syscall_id_t syscall_id, const pok_-syscall_args_t ∗args, const pok_syscall_info_t ∗infos)

4.74.1 Function Documentation

4.74.1.1 pok_ret_t pok_core_syscall (const pok_syscall_id_t *syscall_id,* const pok_syscall_args_t ∗ *args,* const pok_syscall_info_t ∗ *infos*)

Function that performs the syscall. It is called by the architecture interruption handler.

Parameters

syscall_id	This param correspond to the syscal which should be performed. The list of available syscalls is available in the definition of the pok_syscall-_id_t type
args	Arguments of the syscall. It corresponds to data useful to perform the syscall.
infos	Informations about the syscall: which partition/thread initiates the syscall, etc ...

Returns

Returns an error code, which is defined in include/errno.h

Here is the default syscall handler. In this case, the syscall ID was not properly identified and thus, we should return an error. If error management is activated, we raise an error in kernel of partitions, calling the error handler.

Definition at line 40 of file syscall.c.

```
{
   switch (syscall_id)
   {
#if defined (POK_NEEDS_CONSOLE) || defined (POK_NEEDS_DEBUG)
      case POK_SYSCALL_CONSWRITE:
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg1 + infos->base_addr
      )
         if (pok_cons_write ((const char*)args->arg1 + infos->base_addr, args->
      arg2))
         {
            return POK_ERRNO_OK;
         }
         else
         {
            return POK_ERRNO_EINVAL;
         }
         break;
#endif

#ifdef POK_NEEDS_PORTS_VIRTUAL
      case POK_SYSCALL_MIDDLEWARE_VIRTUAL_CREATE:
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg1 + infos->base_addr
      )
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg2 + infos->
      base_addr)
         return pok_port_virtual_id ( (char*) (args->arg1 + infos->base_addr),
      (pok_port_id_t*) (args->arg2 + infos->base_addr));
         break;
      case POK_SYSCALL_MIDDLEWARE_VIRTUAL_NB_DESTINATIONS:
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg2 + infos->
      base_addr)
         return pok_port_virtual_nb_destinations ( (pok_port_id_t) (args->arg1)
      , (uint32_t*) (args->arg2 + infos->base_addr));
         break;

      case POK_SYSCALL_MIDDLEWARE_VIRTUAL_DESTINATION:
         POK_CHECK_PTR_OR_RETURN(infos->partition, ((void*) args->arg3)+infos->
      base_addr)
         return pok_port_virtual_destination ( (pok_port_id_t) (args->arg1), (
      uint32_t) (args->arg2), (uint32_t*) (args->arg3 + infos->base_addr));
         break;

      case POK_SYSCALL_MIDDLEWARE_VIRTUAL_GET_GLOBAL:
         POK_CHECK_PTR_OR_RETURN(infos->partition, (void*) (args->arg2 + infos-
      >base_addr))
         return pok_port_virtual_get_global ((pok_port_id_t) (args->arg1), (
      pok_port_id_t*) (args->arg2 + infos->base_addr));
         break;

#endif

#if defined POK_NEEDS_GETTICK
      case POK_SYSCALL_GETTICK:
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg1 + infos->base_addr
      )
         return pok_gettick_by_pointer ((uint64_t*) (args->arg1 + infos->
      base_addr));
         break;
#endif

      case POK_SYSCALL_THREAD_CREATE:
         return pok_partition_thread_create  ((uint32_t*)         (args->arg1 +
       infos->base_addr),
                                             (pok_thread_attr_t*) (args->arg2 +
       infos->base_addr),
                                             (uint8_t)            infos->
      partition);
         break;

#ifdef POK_NEEDS_THREAD_SLEEP
      case POK_SYSCALL_THREAD_SLEEP:
         return pok_thread_sleep (args->arg1);
         break;
#endif

#ifdef POK_NEEDS_THREAD_SLEEP_UNTIL
      case POK_SYSCALL_THREAD_SLEEP_UNTIL:
         return pok_thread_sleep_until (args->arg1);
         break;
#endif

      case POK_SYSCALL_THREAD_PERIOD:
         return pok_sched_end_period ();
         break;

#if defined (POK_NEEDS_THREAD_SUSPEND) || defined (POK_NEEDS_ERROR_HANDLING)
      case POK_SYSCALL_THREAD_SUSPEND:
         return pok_thread_suspend ();
         break;
#endif
```

```
#ifdef POK_NEEDS_THREAD_ID
      case POK_SYSCALL_THREAD_ID:
         return pok_sched_get_current ((uint32_t*) (args->arg1 + infos->
      base_addr));
         break;
#endif
      case POK_SYSCALL_THREAD_STATUS:
        return pok_thread_get_status (args->arg1, (pok_thread_attr_t*) (args->
      arg2 + infos->base_addr));
         break;

#ifdef POK_NEEDS_ERROR_HANDLING

      case POK_SYSCALL_THREAD_RESTART:
         return pok_partition_restart_thread (args->arg1);
         break;

      case POK_SYSCALL_THREAD_STOP:
         return pok_partition_stop_thread (args->arg1);
         break;

      case POK_SYSCALL_THREAD_STOPSELF:
         pok_sched_stop_self ();
         return POK_ERRNO_OK;
         break;

#endif

#ifdef POK_NEEDS_PARTITIONS
      case POK_SYSCALL_PARTITION_SET_MODE:
         return pok_partition_set_mode_current ((pok_partition_mode_t)args->arg1
      );
         break;
      case POK_SYSCALL_PARTITION_GET_ID:
        return pok_current_partition_get_id ((uint8_t*)(args->arg1 + infos->
      base_addr));
         break;
      case POK_SYSCALL_PARTITION_GET_PERIOD:
        return pok_current_partition_get_period ((uint64_t*)(args->arg1 + infos
      ->base_addr));
         break;
      case POK_SYSCALL_PARTITION_GET_DURATION:
        return pok_current_partition_get_duration ((uint64_t*)(args->arg1 +
      infos->base_addr));
         break;
      case POK_SYSCALL_PARTITION_GET_LOCK_LEVEL:
        return pok_current_partition_get_lock_level ((uint32_t*)(args->arg1 +
      infos->base_addr));
         break;
      case POK_SYSCALL_PARTITION_GET_OPERATING_MODE:
        return pok_current_partition_get_operating_mode ((pok_partition_mode_t*
      )(args->arg1 + infos->base_addr));
         break;
      case POK_SYSCALL_PARTITION_GET_START_CONDITION:
        return pok_current_partition_get_start_condition ((
      pok_start_condition_t*)(args->arg1 + infos->base_addr));
         break;
#endif

#ifdef POK_NEEDS_ERROR_HANDLING
      case POK_SYSCALL_ERROR_HANDLER_CREATE:
         return pok_error_thread_create (args->arg1 , (void*) (args->arg2));
         break;

      case POK_SYSCALL_ERROR_RAISE_APPLICATION_ERROR:
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg1 + infos->base_addr
      )
         pok_error_raise_application_error ((char*) (args->arg1 + infos->
      base_addr), args->arg2);
         return POK_ERRNO_OK;
         break;

      case POK_SYSCALL_ERROR_GET:
         return pok_error_get ((pok_error_status_t*) (args->arg1 + infos->
      base_addr));
         break;
#endif

         /* Middleware syscalls */
#ifdef POK_NEEDS_PORTS_SAMPLING
      case POK_SYSCALL_MIDDLEWARE_SAMPLING_CREATE:
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg5 + infos->base_addr
      )
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg1 + infos->
      base_addr)
         return pok_port_sampling_create  ((char*)(args->arg1 + infos->
      base_addr),
                                          (pok_port_size_t) args->arg2,
                                          (pok_port_direction_t) args->arg3,
                                          (uint64_t) args->arg4,
                                          (pok_port_id_t*) (args->arg5 + infos-
      >base_addr));
         break;

      case POK_SYSCALL_MIDDLEWARE_SAMPLING_WRITE:
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg2 + infos->
      base_addr)

         return pok_port_sampling_write   ((const pok_port_id_t)args->arg1,
                                          (const void*) ((void*)args->arg2 +
      infos->base_addr),
                                          (const uint8_t) args->arg3);
         break;

      case POK_SYSCALL_MIDDLEWARE_SAMPLING_READ:
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg2 + infos->
      base_addr)
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg4 + infos->
      base_addr)
         return pok_port_sampling_read ((const pok_port_id_t)args->arg1,
                                       (void*) args->arg2 + infos->base_addr,
                                       (pok_port_size_t*) (args->arg3 + infos->
      base_addr),
                                       (bool_t*) (args->arg4 + infos->base_addr
      ));
         break;

      case POK_SYSCALL_MIDDLEWARE_SAMPLING_ID:
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg1 + infos->
      base_addr)
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg2 + infos->
      base_addr)
         return pok_port_sampling_id   ((char*)(args->arg1 + infos->base_addr),
                                       (pok_port_id_t*)(args->arg2 + infos->
      base_addr));
         break;

#ifndef POK_GENERATED_CODE
      case POK_SYSCALL_MIDDLEWARE_SAMPLING_STATUS:
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg2+infos->base_addr)
         return pok_port_sampling_status ((const pok_port_id_t)args->arg1,
                                          (pok_port_sampling_status_t*) (args->
      arg2 + infos->base_addr));
         break;
#endif /* POK_GENERATED_CODE */
#endif /* POK_NEEDS_PORTS_SAMPLING */


#ifdef POK_NEEDS_PORTS_QUEUEING
      case POK_SYSCALL_MIDDLEWARE_QUEUEING_CREATE:
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg1 + infos->base_addr
      )
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg5 + infos->
      base_addr)
         return pok_port_queueing_create ((char*)                         (
      args->arg1 + infos->base_addr),
                                         (pok_port_size_t)                args
      ->arg2,
                                         (pok_port_direction_t)           args
      ->arg3,
                                         (pok_port_queueing_discipline_t)  args
      ->arg4,
                                         (pok_port_id_t*)                 (
      args->arg5 + infos->base_addr));
         break;

      case POK_SYSCALL_MIDDLEWARE_QUEUEING_SEND:
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg2 + infos->
      base_addr)
         return pok_port_queueing_send ((const pok_port_id_t)             args
      ->arg1,
                                       (const void*)                      ((
      void*)args->arg2 + infos->base_addr),
                                       (const uint8_t)                    (
      args->arg3),
                                       (const uint64_t)                   args
      ->arg4);
         break;

      case POK_SYSCALL_MIDDLEWARE_QUEUEING_RECEIVE:
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg4 + infos->
      base_addr)
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg5 + infos->
      base_addr)
         return pok_port_queueing_receive ((const pok_port_id_t)   args->arg1,
                                          (uint64_t)              args->arg2,
                                          (pok_port_size_t)       args->arg3,
                                          (void*)                 ((void*)args-
      >arg4 + infos->base_addr),
                                          (pok_port_size_t*)      (args->arg5 +
       infos->base_addr));
         break;

      case POK_SYSCALL_MIDDLEWARE_QUEUEING_ID:
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg1 + infos->
      base_addr)
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg2 + infos->
      base_addr)
         return pok_port_queueing_id ((char*)          (args->arg1 + infos->
      base_addr),
                                     (pok_port_id_t*)  (args->arg2 + infos->
      base_addr));
         break;

#ifndef POK_GENERATED_CODE
      case POK_SYSCALL_MIDDLEWARE_QUEUEING_STATUS:
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg2 + infos->base_addr
      )
         return pok_port_queueing_status     ((const pok_port_id_t)
      args->arg1,
                                              (pok_port_queueing_status_t*)    (
      args->arg2 + infos->base_addr));
         break;
#endif
#endif /* POK_NEEDS_PORTS_QUEUEING */

#ifdef POK_NEEDS_LOCKOBJECTS
      case POK_SYSCALL_LOCKOBJ_CREATE:
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg2+infos->base_addr)
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg1+infos->base_addr)
         return pok_lockobj_partition_create    ((pok_lockobj_id_t*)    (args->
      arg1 + infos->base_addr),
                                                 (pok_lockobj_attr_t*)   (args->
      arg2 + infos->base_addr));
         break;

      case POK_SYSCALL_LOCKOBJ_OPERATION:
         if (args->arg2 == NULL)
         {
            return pok_lockobj_partition_wrapper ((const uint8_t) args->arg1,
      NULL);
         }
         else
         {
            POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg2 + infos->
      base_addr)
            return pok_lockobj_partition_wrapper   ((const uint8_t) args->arg1,
                                                    (pok_lockobj_lockattr_t*) (
      args->arg2 + infos->base_addr));
         }
         break;
#endif /* POK_NEEDS_LOCKOBJECTS */

#ifdef POK_NEEDS_IO
      case POK_SYSCALL_INB:
         if ((args->arg1 < pok_partitions[infos->partition].io_min) ||
             (args->arg1 > pok_partitions[infos->partition].io_max))
         {
            return -POK_ERRNO_EPERM;
         }
         else
         {
            return inb((unsigned short) args->arg1);
         }
         break;

      case POK_SYSCALL_OUTB:
         if ((args->arg1 < pok_partitions[infos->partition].io_min) ||
             (args->arg1 > pok_partitions[infos->partition].io_max))
         {
            return -POK_ERRNO_EPERM;
         }
         else
         {
            outb((unsigned short) args->arg1, (unsigned char) args->arg2);
            return POK_ERRNO_OK;
         }
       break;
#endif /* POK_NEEDS_IO */

#ifdef POK_NEEDS_PCI
     case POK_SYSCALL_PCI_REGISTER:
       POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg1 + infos->base_addr)
       return pci_register((void*)args->arg1 + infos->base_addr, infos->
      partition);
       break;
#endif /* POK_NEEDS_PCI */


      default:
#ifdef POK_NEEDS_ERROR_HANDLING
         pok_error_declare (POK_ERROR_KIND_ILLEGAL_REQUEST);
         pok_sched_activate_error_thread ();
#else
         #ifdef POK_NEEDS_DEBUG
            printf ("Tried to use syscall %d\n", syscall_id);
         #endif
         POK_FATAL ("Unknown syscall");
#endif
         break;
   }

   return POK_ERRNO_EINVAL;
}
```

4.75 /home/matias/projet/stage-pok/pok/trunk/kernel/core/time.c File Reference

4.75.1 Detailed Description

Author

François Goudal
Julien Delange

Date

2008-2009

Definition in file time.c.

4.76 /home/matias/projet/stage-pok/pok/trunk/kernel/include/arch.h File Reference

Generic interface to handle architectures.

`#include <types.h> #include <errno.h>`

Functions

- pok_ret_t pok_arch_init ()
- pok_ret_t pok_arch_preempt_disable ()
- pok_ret_t pok_arch_preempt_enable ()
- pok_ret_t pok_arch_idle ()
- pok_ret_t pok_arch_event_register (uint8_t vector, void(∗handler)(void))
- uint32_t pok_context_create (uint32_t thread_id, uint32_t stack_size, uint32_t entry)
- void pok_context_switch (uint32_t ∗old_sp, uint32_t new_sp)
- pok_ret_t pok_create_space (uint8_t partition_id, uint32_t addr, uint32_t size)
- uint32_t pok_space_base_vaddr (uint32_t addr)
- void pok_dispatch_space (uint8_t partition_id, uint32_t user_pc, uint32_t user_-sp, uint32_t kernel_sp, uint32_t arg1, uint32_t arg2)
- uint32_t pok_space_context_create (uint8_t partition_id, uint32_t entry_rel, uint32_t stack_rel, uint32_t arg1, uint32_t arg2)
- void pok_space_context_restart (uint32_t sp, uint32_t entry, uint32_t user_stack)
- pok_ret_t pok_space_switch (uint8_t old_partition_id, uint8_t new_partition_id)
- uint32_t pok_thread_stack_addr (const uint8_t partition_id, const uint32_t local-_thread_id)

4.76.1 Detailed Description

Generic interface to handle architectures.

Author

Julian Pidancet
Julien Delange

Date

2008-2009

Definition in file arch.h.

4.76.2 Function Documentation

4.76.2.1 pok_ret_t pok_arch_event_register (uint8_t *vector,* void(∗)(void) *handler*)

Register an event (for example, an interruption)

Attach the handler to the given trap number (vector).

See also

pok_sparc_isr

Definition at line 83 of file arch.c.

```
{
  (void) vector;
  (void) handler;

  return (POK_ERRNO_OK);
}
```

4.76.2.2 pok_ret_t pok_arch_idle ( )

Function that do nothing. Useful for the idle task for example.

Definition at line 74 of file arch.c.

```
{
   while (1)
   {
   }

   return (POK_ERRNO_OK);
}
```

4.76.2.3 pok_ret_t pok_arch_init ( )

Function that initializes architecture concerns.

Initialize all SPARC managers (traps, syscalls, space).

Definition at line 43 of file arch.c.

```
{
  set_msr (MSR_IP);
#if POK_NEEDS_PARTITIONS
  pok_arch_space_init();
#endif

  return (POK_ERRNO_OK);
}
```

4.76.2.4 pok_ret_t pok_arch_preempt_disable ( )

Disable interruptions

Definition at line 53 of file arch.c.

```
{
  unsigned int msr;

  msr = get_msr();
  msr &= ~MSR_EE;
  set_msr(msr);
  return (POK_ERRNO_OK);
}
```

4.76.2.5 pok_ret_t pok_arch_preempt_enable ( )

Enable interruptions

Definition at line 63 of file arch.c.

```
{
  unsigned int msr;

  msr = get_msr();
  msr |= MSR_EE;
  set_msr(msr);

  return (POK_ERRNO_OK);
}
```

4.76.2.6 uint32_t pok_context_create (uint32_t *thread_id*, uint32_t *stack_size*, uint32_t *entry*)

4.76.2.7 void pok_context_switch (uint32_t ∗ *old_sp*, uint32_t *new_sp*)

4.76.2.8 pok_ret_t pok_create_space (uint8_t *partition_id*, uint32_t *addr*, uint32_t *size*)

Set ptd and pte for the given partition.

Definition at line 42 of file space.c.

```
{
#ifdef POK_NEEDS_DEBUG
  printf ("pok_create_space: %d: %x %x\n", partition_id, addr, size);
#endif
  spaces[partition_id].phys_base = addr;
  spaces[partition_id].size = size;

  return (POK_ERRNO_OK);
}
```

4.76.2.9 void pok_dispatch_space (uint8_t *partition_id*, uint32_t *user_pc*, uint32_t *user_sp*, uint32_t *kernel_sp*, uint32_t *arg1*, uint32_t *arg2*)

Definition at line 114 of file space.c.

```
{
   interrupt_frame   ctx;
   uint32_t          code_sel;
   uint32_t          data_sel;
   uint32_t          sp;

   code_sel = GDT_BUILD_SELECTOR (GDT_PARTITION_CODE_SEGMENT (partition_id), 0,
       3);
   data_sel = GDT_BUILD_SELECTOR (GDT_PARTITION_DATA_SEGMENT (partition_id), 0,
       3);

   sp = (uint32_t) &ctx;

   memset (&ctx, 0, sizeof (interrupt_frame));

   pok_arch_preempt_disable ();

   ctx.es = ctx.ds = ctx.ss = data_sel;

   ctx.__esp   = (uint32_t) (&ctx.error); /* for pusha */
   ctx.eip     = user_pc;
   ctx.eax     = arg1;
   ctx.ebx     = arg2;
   ctx.cs      = code_sel;
   ctx.eflags  = 1 << 9;
   ctx.esp     = user_sp;

   tss_set_esp0 (kernel_sp);

   asm ("mov %0, %%esp          \n"
        "pop %%es               \n"
        "pop %%ds               \n"
        "popa                   \n"
        "addl $4, %%esp         \n"
        "iret                   \n"
        :
        : "m" (sp)
       );
}
```

4.76.2.10 uint32_t pok_space_base_vaddr (uint32_t *addr*)

Returns

partition virtual base adress.

See also

SPARC_PARTITION_BASE_VADDR

Definition at line 64 of file space.c.

```
{
   (void) addr;
   return (0);
}
```

4.76.2.11 uint32_t pok_space_context_create (uint8_t *id,* uint32_t *entry_rel,* uint32_t *stack_rel,* uint32_t *arg1,* uint32_t *arg2*)

Create a new context in the given space

Initilize thread stack.

Definition at line 72 of file space.c.

```
{
  context_t* ctx;
  volatile_context_t* vctx;
  char*      stack_addr;
  (void) partition_id;

  stack_addr = pok_bsp_mem_alloc (KERNEL_STACK_SIZE);

  vctx = (volatile_context_t *)
    (stack_addr + KERNEL_STACK_SIZE - sizeof (volatile_context_t));
  ctx = (context_t *)((char *)vctx - sizeof (context_t) + 8);

  memset (ctx, 0, sizeof (*ctx));
  memset (vctx, 0, sizeof (*vctx));

  vctx->r3     = arg1;
  vctx->r4     = arg2;
  vctx->sp     = stack_rel - 12;
  vctx->srr0   = entry_rel;
  vctx->srr1   = MSR_EE | MSR_IP | MSR_DR | MSR_IR | MSR_PR;
  ctx->lr      = (uint32_t) pok_arch_rfi;

  ctx->sp      = (uint32_t) &vctx->sp;

#ifdef POK_NEEDS_DEBUG
  printf ("space_context_create %d: entry=%x stack=%x arg1=%x arg2=%x ksp=%x\n"
      ,
          partition_id, entry_rel, stack_rel, arg1, arg2, &vctx->sp);
#endif

  return (uint32_t)ctx;
}
```

4.76.2.12 void pok_space_context_restart (uint32_t *sp,* uint32_t *entry,* uint32_t *user_stack*)

4.76.2.13 pok_ret_t pok_space_switch (uint8_t *old_partition_id,* uint8_t *new_partition_id*)

Switch from one space to another

Switch adress space in MMU (context register).

Definition at line 55 of file space.c.

```
{
  (void) old_partition_id;
  /* printf ("space_switch %u -> %u\n", old_partition_id, new_partition_id); */
  asm volatile ("mtsr %0,%1" : : "r"(0), "r"(PPC_SR_KP | new_partition_id));
  return (POK_ERRNO_OK);
}
```

4.76.2.14 uint32_t pok_thread_stack_addr (const uint8_t *partition_id*, const uint32_t *local_thread_id*)

Returns the stack address for a the thread number N in a partition.

- partition_id indicates the partition that contains the thread.
- local_thread_id the thread-id of the thread inside the partition.

Returns

the stack address of the thread.

Compute the stack adress for the given thread.

Definition at line 92 of file arch.c.

```
{
   return pok_partitions[partition_id].size - 16 - (local_thread_id *
      POK_USER_STACK_SIZE);
}
```

4.77 /home/matias/projet/stage-pok/pok/trunk/kernel/include/arch/ppc/spinlock.h File Reference

Defines

- #define SPIN_UNLOCK(_spin_) (_spin_) = 0
- #define SPIN_LOCK(_spin_)

Typedefs

- typedef unsigned int pok_spinlock_t

4.77.1 Define Documentation

4.77.1.1 #define SPIN_LOCK(*_spin_*)

Value:

```
do {                                                          \
    unsigned int val;                                          \
    asm volatile ("\n"                                         \
                  "1:\n\t"                                     \
                  "lwarx     %0,0,%1        \n\t"              \
                  "cmpwi     %0,0           \n\t"              \
                  "bne       1b             \n\t"              \
                  "stwcx.    %2,0,%1        \n\t"              \
                  "bne       1b             \n\t"              \
                  : "=&r"(val) : "r" (&_spin_), "r"(1));        \
  } while (0)
```

Definition at line 26 of file spinlock.h.

4.77.1.2 #define SPIN_UNLOCK(*_spin_*) (_spin_) = 0

Definition at line 23 of file spinlock.h.

4.77.2 Typedef Documentation

4.77.2.1 typedef unsigned int pok_spinlock_t

Definition at line 21 of file spinlock.h.

4.78 /home/matias/projet/stage-pok/pok/trunk/kernel/include/arch/sparc/spinlock.h File Reference

Defines

- #define SPIN_UNLOCK(_spin_) (_spin_) = 0
- #define SPIN_LOCK(_spin_)

Typedefs

- typedef unsigned int pok_spinlock_t

4.78.1 Define Documentation

4.78.1.1 #define SPIN_LOCK(*_spin_*)

Value:

```
do {                                             \
    asm volatile ("1:              \n"           \
                  "ldstub [%0], %1 \n"           \
                  "tst %1          \n"           \
                  "bnz 1b          \n"           \
                  : /* no output */              \
                  : "r" (&(_spin_)), "r"(1));    \
  } while (0)
```

Definition at line 30 of file spinlock.h.

4.78.1.2 #define SPIN_UNLOCK(*_spin_*) (_spin_) = 0

Definition at line 27 of file spinlock.h.

4.78.2 Typedef Documentation

4.78.2.1 typedef unsigned int pok_spinlock_t

Definition at line 25 of file spinlock.h.

4.79 /home/matias/projet/stage-pok/pok/trunk/kernel/include/arch/x86/spinlock.h File Reference

Defines

- #define SPIN_UNLOCK(_spin_)
- #define SPIN_LOCK(_spin_)

Typedefs

- typedef unsigned char pok_spinlock_t

4.79.1 Define Documentation

4.79.1.1 #define SPIN_LOCK(*_spin_*)

Value:

```
asm volatile ("mov $1, %%al           \n\t"                   \
                "1:                     \n\t"                   \
                "lock xchg %0, %%al     \n\t"                   \
                "test %%al, %%al        \n\t"                   \
                "jnz 1b                 \n\t"                   \
                :                                               \
                : "m" (_spin_)                                  \
                : "%al")
```

Definition at line 28 of file spinlock.h.

4.79.1.2 #define SPIN_UNLOCK(*_spin_*)

Value:

```
{                                                                              \
  (_spin_) = 0;                                                                \
}
```

Definition at line 23 of file spinlock.h.

4.79.2 Typedef Documentation

4.79.2.1 typedef unsigned char pok_spinlock_t

Definition at line 21 of file spinlock.h.

4.80 /home/matias/projet/stage-pok/pok/trunk/kernel/include/arch/x86/interrupt.h File Reference

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

Data Structures

- struct interrupt_frame

Defines

- #define INTERRUPT_HANDLER(name)
- #define INTERRUPT_HANDLER_errorcode(name)
- #define INTERRUPT_HANDLER_syscall(name)

Functions

- void update_tss (interrupt_frame ∗frame)

Variables

- uint32_t pok_tss

4.80.1 Define Documentation

4.80.1.1 #define INTERRUPT_HANDLER(*name*)

Value:

```
void name (void);                                                       \
void name##_handler(interrupt_frame* frame);                            \
  asm (                                                                 \
      ".global "#name "                 \n"                             \
      "\t.type "#name",@function        \n"                             \
      #name":                           \n"                             \
      "cli                      \n"                             \
      "subl $4, %esp                    \n"                             \
      "pusha                            \n"                             \
      "push %ds                         \n"                             \
      "push %es                         \n"                             \
      "push %esp                        \n"                             \
      "mov $0x10, %ax                   \n"                             \
      "mov %ax, %ds                     \n"                             \
      "mov %ax, %es                     \n"                             \
      "call " #name"_handler            \n"                             \
      "call update_tss                  \n"                             \
      "addl $4, %esp                    \n"                             \
      "pop %es                          \n"                             \
      "pop %ds                          \n"                             \
      "popa                             \n"                             \
      "addl $4, %esp                    \n"                             \
      "sti                      \n"                             \
      "iret                             \n"                             \
      );                                                                \
void name##_handler(interrupt_frame* frame)
```

Definition at line 53 of file interrupt.h.

4.80.1.2 #define INTERRUPT_HANDLER_errorcode(*name*)

Value:

```
void name (void);                                                       \
void name##_handler(interrupt_frame* frame);                    \
  asm (                                                                 \
      ".global "#name "                 \n"                             \
      "\t.type "#name",@function        \n"                             \
      #name":                           \n"                             \
      "cli                      \n"                             \
      "pusha                            \n"                             \
      "push %ds                         \n"                             \
      "push %es                         \n"                             \
      "push %esp                        \n"                             \
      "mov $0x10, %ax                   \n"                             \
      "mov %ax, %ds                     \n"                             \
      "mov %ax, %es                     \n"                             \
      "call " #name"_handler            \n"                             \
      "call update_tss                  \n"                             \
      "addl $4, %esp                    \n"                             \
      "pop %es                          \n"                             \
      "pop %ds                          \n"                             \
      "popa                             \n"                             \
      "addl $4, %esp                    \n"                             \
      "sti                      \n"                             \
      "iret                             \n"                             \
      );                                                                \
void name##_handler(interrupt_frame* frame)
```

Definition at line 81 of file interrupt.h.

4.80.1.3 #define INTERRUPT_HANDLER_syscall(*name*)

Value:

```
int name (void);                                                        \
void name##_handler(interrupt_frame* frame);                            \
  asm (                                                                 \
      ".global "#name "                 \n"                             \
      "\t.type "#name",@function        \n"                             \
      #name":                           \n"                             \
      "cli                      \n"                             \
      "subl $4, %esp                    \n"                             \
      "pusha                            \n"                             \
      "push %ds                         \n"                             \
      "push %es                         \n"                             \
      "push %esp                        \n"                             \
      "mov $0x10, %ax                   \n"                             \
      "mov %ax, %ds                     \n"                             \
      "mov %ax, %es                     \n"                             \
      "call " #name"_handler            \n"                             \
      "movl %eax, 40(%esp)         \n" /* return value */  \
      "call update_tss                  \n"                             \
      "addl $4, %esp                    \n"                             \
      "pop %es                          \n"                             \
      "pop %ds                          \n"                             \
      "popa                             \n"                             \
      "addl $4, %esp                    \n"                             \
      "sti                      \n"                             \
      "iret                             \n"                             \
      );                                                                \
void name##_handler(interrupt_frame* frame)
```

Definition at line 108 of file interrupt.h.

4.80.2 Function Documentation

4.80.2.1 void update_tss (interrupt_frame * *frame*)

Definition at line 20 of file interrupt.c.

```
{
  uint32_t* esp0 = (&pok_tss) + 1;

  if ((frame->cs & 0xffff) != 0x8)
  {
    *esp0 = (uint32_t)frame + sizeof (interrupt_frame);
  }
}
```

4.80.3 Variable Documentation

4.80.3.1 uint32_t pok_tss

Definition at line 39 of file gdt.c.

4.81 /home/matias/projet/stage-pok/pok/trunk/kernel/include/arch/x86/multiboot.h File Reference

Data Structures

- struct pok_multiboot_header_t
- struct pok_aout_symbol_table_t
- struct pok_elf_section_header_table_t
- struct pok_multiboot_info_t
- struct pok_module_t
- struct pok_memory_map_t

Defines

- #define MULTIBOOT_BOOTLOADER_MAGIC 0x2BADB002
- #define MULTIBOOT_HEADER_MAGIC 0x1BADB002
- #define MULTIBOOT_HEADER_FLAGS 0x00010003
- #define MULTIBOOT_BOOTLOADER_MAGIC 0x2BADB002
- #define MULTIBOOT_STACK_SIZE 0x4000
- #define MULTIBOOT_CMDLINE 4
- #define MULTIBOOT_MODS 8
- #define EXT_C(sym) sym

4.81.1 Detailed Description

Author

Julien Pidancet

Date

2008-2009

Definition in file multiboot.h.

4.81.2 Define Documentation

4.81.2.1 #define EXT_C(*sym*) sym

C symbol format. HAVE_ASM_USCORE is defined by configure.

Definition at line 57 of file multiboot.h.

4.81.2.2 #define MULTIBOOT_BOOTLOADER_MAGIC 0x2BADB002

The magic number passed by a Multiboot-compliant boot loader.

Definition at line 41 of file multiboot.h.

4.81.2.3 #define MULTIBOOT_BOOTLOADER_MAGIC 0x2BADB002

The magic number passed by a Multiboot-compliant boot loader.

Definition at line 41 of file multiboot.h.

4.81.2.4 #define MULTIBOOT_CMDLINE 4

Definition at line 48 of file multiboot.h.

4.81.2.5 #define MULTIBOOT_HEADER_FLAGS 0x00010003

The flags for the Multiboot header.

Definition at line 36 of file multiboot.h.

4.81.2.6 #define MULTIBOOT_HEADER_MAGIC 0x1BADB002

The magic number for the Multiboot header.

Definition at line 31 of file multiboot.h.

4.81.2.7 #define MULTIBOOT_MODS 8

Definition at line 49 of file multiboot.h.

4.81.2.8 #define MULTIBOOT_STACK_SIZE 0x4000

The size of our stack (16KB).

Definition at line 46 of file multiboot.h.

4.82 /home/matias/projet/stage-pok/pok/trunk/kernel/include/arch/x86/pci.h File Reference

4.83 /home/matias/projet/stage-pok/pok/trunk/kernel/include/bsp.h File Reference

Interfaces that BSP must provide.

```
#include <types.h> #include <errno.h>
```

Functions

- pok_ret_t pok_bsp_init ()
- pok_ret_t pok_bsp_irq_acknowledge (uint8_t irq)
- pok_ret_t pok_bsp_irq_register (uint8_t irq, void(∗handler)(void))
- void ∗ pok_bsp_mem_alloc (size_t size)
- pok_ret_t pok_bsp_time_init ()
- bool_t pok_cons_write (const char ∗s, size_t length)

4.83.1 Detailed Description

Interfaces that BSP must provide.

Author

Julian Pidancet

Date

2008-2009

Definition in file bsp.h.

4.83.2 Function Documentation

4.83.2.1 pok_ret_t pok_bsp_init ( )

Definition at line 22 of file bsp.c.

```
{
   pok_cons_init ();

   return (POK_ERRNO_OK);
}
```

4.83.2.2 pok_ret_t pok_bsp_irq_acknowledge (uint8_t *irq*)

Definition at line 35 of file bsp.c.

```
{
   pok_pic_eoi (irq);

   return (POK_ERRNO_OK);
}
```

4.83.2.3 pok_ret_t pok_bsp_irq_register (uint8_t *irq*, void(∗)(void) *handler*)

Definition at line 42 of file bsp.c.

```
{
   pok_pic_unmask (irq);

   pok_arch_event_register (32 + irq, handler);

   return (POK_ERRNO_OK);
}
```

4.83.2.4 void∗ pok_bsp_mem_alloc (size_t *size*)

Used for partition allocation. For SPARC support, all partitions are aligned on page size and all partition sizes have to be less than page size.

See also

SPARC_PAGE_SIZE

Allocate data. At this time, the pok_pm_sbrk function only increment size each time we allocate memory and was not designed to free previously allocated memory.

Definition at line 34 of file bsp.c.

```
{
  char *res;

  res = (char *)(((unsigned int)heap_end + 4095) & ~4095);
  heap_end = res + sz;
  return res;
}
```

4.83.2.5 pok_ret_t pok_bsp_time_init ( )

Initialize the timer, register the ISR and unmask the interrupt.

See also

unmask_irq(irq_nbr)

Init time. *freq* is the frequency of the oscillator.

Definition at line 87 of file timer.c.

```
{
  time_inter = (BUS_FREQ * FREQ_DIV) / POK_TIMER_FREQUENCY;
  time_last = get_ppc_tb ();
  pok_arch_set_decr();

  return (POK_ERRNO_OK);
}
```

4.83.2.6 bool_t pok_cons_write (const char ∗ *s,* size_t *length*)

4.84 /home/matias/projet/stage-pok/pok/trunk/kernel/include/core/boot.h File Reference

Functions

- void pok_boot ()

  Boot function that launch everything.

4.84.1 Detailed Description

Author

Julien Delange

Date

2008-2009

Definition in file boot.h.

4.84.2 Function Documentation

4.84.2.1 void pok_boot ( )

Boot function that launch everything.

This function load every service according to system requirements (the POK_NEEDS_∗ maccro). If we don't use partitioning service, we execute a main function. In that case, POK is acting like an executive, not a real kernel

Definition at line 37 of file boot.c.

```
{
   pok_arch_init();
   pok_bsp_init();

#if defined (POK_NEEDS_TIME) || defined (POK_NEEDS_SCHED) || defined
       (POK_NEEDS_THREADS)
   pok_time_init();
#endif

#ifdef POK_NEEDS_PARTITIONS
   pok_partition_init ();
#endif

#ifdef POK_NEEDS_THREADS
   pok_thread_init ();
#endif

#if defined (POK_NEEDS_SCHED) || defined (POK_NEEDS_THREADS)
   pok_sched_init ();
#endif

#if (defined POK_NEEDS_LOCKOBJ) || defined (POK_NEEDS_PORTS_QUEUEING) ||
       defined (POK_NEEDS_PORTS_SAMPLING)
   pok_lockobj_init ();
#endif
#if defined (POK_NEEDS_PORTS_QUEUEING) || defined (POK_NEEDS_PORTS_SAMPLING)
   pok_port_init ();
   pok_queue_init ();
#endif

#if defined (POK_NEEDS_DEBUG) || defined (POK_NEEDS_CONSOLE)
  pok_cons_write ("POK kernel initialized\n", 23);
#endif

#ifdef POK_NEEDS_INSTRUMENTATION
  uint32_t tmp;
   printf ("[INSTRUMENTATION][CHEDDAR] <event_table>\n");
   printf ("[INSTRUMENTATION][CHEDDAR] <processor>\n");
   printf ("[INSTRUMENTATION][CHEDDAR] <name>pok_kernel</name>\n");

   for (tmp = 0 ; tmp < POK_CONFIG_NB_THREADS ; tmp++)
   {
      printf ("[INSTRUMENTATION][CHEDDAR] <task_activation>   0   task
       %d</task_activation>\n", tmp);
   }
#endif

  pok_arch_preempt_enable();

#ifndef POK_NEEDS_PARTITIONS

  main ();
#endif
}
```

4.85 /home/matias/projet/stage-pok/pok/trunk/kernel/include/core/cpio.h File Reference

Data Structures

- struct cpio_bin_header
- struct cpio_file

Enumerations

- enum cpio_format { CPIO_BIN_FMT, CPIO_ODC_FMT, CPIO_NEWC_FMT, - CPIO_CRC_FMT, CPIO_TAR_FMT, CPIO_USTAR_FMT, CPIO_HPBIN_FMT, CPIO_HPODC_FMT }

Functions

- int cpio_open (struct cpio_file ∗cpio, void ∗file)
- char ∗ cpio_get_filename (struct cpio_file ∗cpio)
- int cpio_next_file (struct cpio_file ∗cpio)
- void ∗ cpio_get_fileaddr (struct cpio_file ∗cpio)

4.85.1 Enumeration Type Documentation

4.85.1.1 enum cpio_format

Enumerator:

CPIO_BIN_FMT

CPIO_ODC_FMT

CPIO_NEWC_FMT

CPIO_CRC_FMT

CPIO_TAR_FMT

CPIO_USTAR_FMT

CPIO_HPBIN_FMT

CPIO_HPODC_FMT

Definition at line 21 of file cpio.h.

```
{
   CPIO_BIN_FMT,
   CPIO_ODC_FMT,
   CPIO_NEWC_FMT,
   CPIO_CRC_FMT,
   CPIO_TAR_FMT,
   CPIO_USTAR_FMT,
   CPIO_HPBIN_FMT,
   CPIO_HPODC_FMT
};
```

4.85.2 Function Documentation

4.85.2.1 void∗ cpio_get_fileaddr (struct cpio_file ∗ *cpio*)

4.85.2.2 char∗ cpio_get_filename (struct cpio_file ∗ *cpio*)

4.85.2.3 int cpio_next_file (struct cpio_file ∗ *cpio*)

4.85.2.4 int cpio_open (struct cpio_file ∗ *cpio,* void ∗ *file*)

4.86 /home/matias/projet/stage-pok/pok/trunk/kernel/include/core/debug.h File Reference

Defines

- #define POK_DEBUG_PRINT_CURRENT_STATE
- #define POK_FATAL(arg)

4.86.1 Define Documentation

4.86.1.1 #define POK_DEBUG_PRINT_CURRENT_STATE

Definition at line 37 of file debug.h.

4.86.1.2 #define POK_FATAL(*arg*)

Definition at line 38 of file debug.h.

4.87 /home/matias/projet/stage-pok/pok/trunk/kernel/include/core/error.h File Reference

4.88 /home/matias/projet/stage-pok/pok/trunk/kernel/include/core/instrumentation.h File Reference

4.89 /home/matias/projet/stage-pok/pok/trunk/kernel/include/core/kernel.h File Reference

Functions

- void pok_kernel_restart (void)
- void pok_kernel_stop (void)

4.89.1 Function Documentation

4.89.1.1 void pok_kernel_restart (void)

4.89.1.2 void pok_kernel_stop (void)

4.90 /home/matias/projet/stage-pok/pok/trunk/kernel/include/core/loader.h File Reference

`#include <types.h>`

Functions

- void pok_loader_load_partition (const uint8_t part_id, uint32_t offset, uint32_t *entry)

  Load the program of the partition.

4.90.1 Function Documentation

4.90.1.1 void pok_loader_load_partition (const uint8_t *part_id,* uint32_t *offset,* uint32_t * *entry*)

Load the program of the partition.

It loads the program of the partition *part_id*. In fact, It will load the ELF file that corresponds to this partition.

4.91 /home/matias/projet/stage-pok/pok/trunk/kernel/include/core/lockobj.h File Reference

`#include <types.h> #include <arch.h>`

Data Structures

- struct pok_lockobj_attr_t
- struct pok_lockobj_t
- struct pok_lockobj_lockattr_t

Defines

- #define POK_CONFIG_NB_LOCKOBJECTS 0

Enumerations

- enum pok_lockobj_kind_t { POK_LOCKOBJ_KIND_MUTEX = 1, POK_LOCKOBJ_KIND_SEMAPHORE = 2, POK_LOCKOBJ_KIND_EVENT = 3 }
- enum pok_locking_policy_t { POK_LOCKOBJ_POLICY_STANDARD = 0, POK_LOCKOBJ_POLICY_PIP = 1, POK_LOCKOBJ_POLICY_PCP = 2 }
- enum pok_mutex_state_t { LOCKOBJ_STATE_LOCK = 0, LOCKOBJ_STATE_UNLOCK = 1, LOCKOBJ_STATE_WAITEVENT = 2 }
- enum pok_lockobj_lock_kind_t { LOCKOBK_LOCK_REGULAR = 1, LOCKOBJ_LOCK_TIMED = 2 }
- enum pok_lockobj_operation_t { LOCKOBJ_OPERATION_LOCK = 1, LOCKOBJ_OPERATION_UNLOCK = 2, LOCKOBJ_OPERATION_WAIT = 3, LOCKOBJ_OPERATION_SIGNAL = 4, LOCKOBJ_OPERATION_BROADCAST = 5 }

Functions

- pok_ret_t pok_lockobj_create (pok_lockobj_t ∗obj, const pok_lockobj_attr_t ∗attr)
- pok_ret_t pok_lockobj_init ()
- pok_ret_t pok_lockobj_partition_create (pok_lockobj_id_t ∗id, const pok_lockobj_attr_t ∗attr)
- pok_ret_t pok_lockobj_lock (pok_lockobj_t ∗obj, const pok_lockobj_lockattr_t ∗attr)
- pok_ret_t pok_lockobj_unlock (pok_lockobj_t ∗obj, const pok_lockobj_lockattr_t ∗attr)
- pok_ret_t pok_lockobj_eventwait (pok_lockobj_t ∗obj, const uint64_t timeout)
- pok_ret_t pok_lockobj_eventsignal (pok_lockobj_t ∗obj)
- pok_ret_t pok_lockobj_eventbroadcast (pok_lockobj_t ∗obj)
- pok_ret_t pok_lockobj_partition_wrapper (const pok_lockobj_id_t id, const pok_lockobj_lockattr_t ∗attr)

4.91.1 Define Documentation

4.91.1.1 #define POK_CONFIG_NB_LOCKOBJECTS 0

Definition at line 25 of file lockobj.h.

4.91.2 Enumeration Type Documentation

4.91.2.1 enum pok_locking_policy_t

Enumerator:

POK_LOCKOBJ_POLICY_STANDARD

POK_LOCKOBJ_POLICY_PIP

POK_LOCKOBJ_POLICY_PCP

Definition at line 47 of file lockobj.h.

```
{
   POK_LOCKOBJ_POLICY_STANDARD = 0,
   POK_LOCKOBJ_POLICY_PIP      = 1,
   POK_LOCKOBJ_POLICY_PCP      = 2
}pok_locking_policy_t;
```

4.91.2.2 enum pok_lockobj_kind_t

Enumerator:

POK_LOCKOBJ_KIND_MUTEX

POK_LOCKOBJ_KIND_SEMAPHORE

POK_LOCKOBJ_KIND_EVENT

Definition at line 38 of file lockobj.h.

```
{
   POK_LOCKOBJ_KIND_MUTEX = 1,
   POK_LOCKOBJ_KIND_SEMAPHORE = 2,
   POK_LOCKOBJ_KIND_EVENT = 3
}pok_lockobj_kind_t;
```

4.91.2.3 enum pok_lockobj_lock_kind_t

Enumerator:

LOCKOBK_LOCK_REGULAR

LOCKOBJ_LOCK_TIMED

Definition at line 100 of file lockobj.h.

```
{
   LOCKOBK_LOCK_REGULAR = 1,
   LOCKOBJ_LOCK_TIMED   = 2
}pok_lockobj_lock_kind_t;
```

4.91.2.4 enum pok_lockobj_operation_t

Enumerator:

LOCKOBJ_OPERATION_LOCK

LOCKOBJ_OPERATION_UNLOCK

LOCKOBJ_OPERATION_WAIT

LOCKOBJ_OPERATION_SIGNAL

LOCKOBJ_OPERATION_BROADCAST

Definition at line 106 of file lockobj.h.

```
{
   LOCKOBJ_OPERATION_LOCK = 1,
   LOCKOBJ_OPERATION_UNLOCK = 2,
   LOCKOBJ_OPERATION_WAIT = 3,
   LOCKOBJ_OPERATION_SIGNAL = 4,
   LOCKOBJ_OPERATION_BROADCAST = 5
}pok_lockobj_operation_t;
```

4.91.2.5 enum pok_mutex_state_t

Enumerator:

LOCKOBJ_STATE_LOCK

LOCKOBJ_STATE_UNLOCK

LOCKOBJ_STATE_WAITEVENT

Definition at line 55 of file lockobj.h.

```
{
   LOCKOBJ_STATE_LOCK = 0,
   LOCKOBJ_STATE_UNLOCK = 1,
   LOCKOBJ_STATE_WAITEVENT = 2
}pok_mutex_state_t;
```

4.91.3 Function Documentation

4.91.3.1 pok_ret_t pok_lockobj_create (pok_lockobj_t ∗ *obj*, const pok_lockobj_attr_t ∗ *attr*)

4.91.3.2 pok_ret_t pok_lockobj_eventbroadcast (pok_lockobj_t ∗ *obj*)

4.91.3.3 pok_ret_t pok_lockobj_eventsignal (pok_lockobj_t ∗ *obj*)

4.91.3.4 pok_ret_t pok_lockobj_eventwait (pok_lockobj_t ∗ *obj*, const uint64_t *timeout*)

4.91.3.5 pok_ret_t pok_lockobj_init ( )

4.91.3.6 pok_ret_t pok_lockobj_lock (pok_lockobj_t ∗ *obj*, const pok_lockobj_lockattr_t ∗ *attr*)

4.91.3.7 pok_ret_t pok_lockobj_partition_create (pok_lockobj_id_t ∗ *id*, const pok_lockobj_attr_t ∗ *attr*)

4.91.3.8 pok_ret_t pok_lockobj_partition_wrapper (const pok_lockobj_id_t *id*, const pok_lockobj_lockattr_t ∗ *attr*)

4.91.3.9 pok_ret_t pok_lockobj_unlock (pok_lockobj_t ∗ *obj*, const pok_lockobj_lockattr_t ∗ *attr*)

4.92 /home/matias/projet/stage-pok/pok/trunk/kernel/include/core/partition.h File Reference

Definition of structure for partitioning services.

4.92.1 Detailed Description

Definition of structure for partitioning services.

Author

Julien Delange

Definition in file partition.h.

4.93 /home/matias/projet/stage-pok/pok/trunk/kernel/include/core/sched.h File Reference

4.94 /home/matias/projet/stage-pok/pok/trunk/kernel/include/core/schedvalues.h File Reference

Enumerations

- enum pok_sched_t { POK_SCHED_FIFO = 0, POK_SCHED_RR = 1, POK_SCHED_GLOBAL_TIMESLICE = 2, POK_SCHED_RMS = 3, POK_SCHED_EDF = 4, POK_SCHED_LLF = 5, POK_SCHED_STATIC = 6 }

4.94.1 Enumeration Type Documentation

4.94.1.1 enum pok_sched_t

Enumerator:

POK_SCHED_FIFO

POK_SCHED_RR

POK_SCHED_GLOBAL_TIMESLICE

POK_SCHED_RMS

POK_SCHED_EDF

POK_SCHED_LLF

POK_SCHED_STATIC

Definition at line 21 of file schedvalues.h.

```
{
   POK_SCHED_FIFO              = 0,
   POK_SCHED_RR                = 1,
   POK_SCHED_GLOBAL_TIMESLICE  = 2,
   POK_SCHED_RMS               = 3,
   POK_SCHED_EDF               = 4,
   POK_SCHED_LLF               = 5,
   POK_SCHED_STATIC            = 6
} pok_sched_t;
```

4.95 /home/matias/projet/stage-pok/pok/trunk/kernel/include/core/syscall.h File Reference

`#include <types.h> #include <errno.h>`

Data Structures

- struct pok_syscall_args_t
- struct pok_syscall_info_t

Defines

- #define POK_CHECK_PTR_OR_RETURN(pid, ptr)

Enumerations

- enum pok_syscall_id_t { POK_SYSCALL_CONSWRITE = 10, POK_SYSCALL_GETTICK = 20, POK_SYSCALL_INT_NUMBER = 42, POK_SYSCALL_THREAD_CREATE = 50, POK_SYSCALL_THREAD_SLEEP_UNTIL = 51, POK_SYSCALL_THREAD_SLEEP = 52, POK_SYSCALL_THREAD_SUSPEND = 53, POK_SYSCALL_THREAD_RESTART = 54, POK_SYSCALL_THREAD_STOP = 55, POK_SYSCALL_THREAD_PERIOD = 56, POK_SYSCALL_THREAD_STOPSELF = 57, POK_SYSCALL_THREAD_ID = 58, POK_SYSCALL_THREAD_STATUS = 59 }

Functions

- pok_ret_t pok_core_syscall (const pok_syscall_id_t syscall_id, const pok_syscall_args_t ∗args, const pok_syscall_info_t ∗infos)
- pok_ret_t pok_syscall_init ()

4.95.1 Define Documentation

4.95.1.1 #define POK_CHECK_PTR_OR_RETURN(*pid*, *ptr*)

Value:

```
if (!POK_CHECK_PTR_IN_PARTITION(pid,ptr))       \
{                                               \
   return POK_ERRNO_EINVAL;                                        \
}
```

Definition at line 137 of file syscall.h.

4.95.2 Enumeration Type Documentation

4.95.2.1 enum pok_syscall_id_t

Enumerator:

POK_SYSCALL_CONSWRITE

POK_SYSCALL_GETTICK

POK_SYSCALL_INT_NUMBER

POK_SYSCALL_THREAD_CREATE

POK_SYSCALL_THREAD_SLEEP_UNTIL

POK_SYSCALL_THREAD_SLEEP

POK_SYSCALL_THREAD_SUSPEND

POK_SYSCALL_THREAD_RESTART

POK_SYSCALL_THREAD_STOP

POK_SYSCALL_THREAD_PERIOD

POK_SYSCALL_THREAD_STOPSELF

POK_SYSCALL_THREAD_ID

POK_SYSCALL_THREAD_STATUS

Definition at line 23 of file syscall.h.

```
{
   POK_SYSCALL_CONSWRITE                           =  10,
   POK_SYSCALL_GETTICK                             =  20,
   POK_SYSCALL_INT_NUMBER                          =  42,
   POK_SYSCALL_THREAD_CREATE                       =  50,
   POK_SYSCALL_THREAD_SLEEP_UNTIL                  =  51,
   POK_SYSCALL_THREAD_SLEEP                        =  52,
   POK_SYSCALL_THREAD_SUSPEND                      =  53,
   POK_SYSCALL_THREAD_RESTART                      =  54,
   POK_SYSCALL_THREAD_STOP                         =  55,
   POK_SYSCALL_THREAD_PERIOD                       =  56,
   POK_SYSCALL_THREAD_STOPSELF                     =  57,
   POK_SYSCALL_THREAD_ID                           =  58,
   POK_SYSCALL_THREAD_STATUS                       =  59,
#ifdef POK_NEEDS_PORTS_SAMPLING
   POK_SYSCALL_MIDDLEWARE_SAMPLING_ID              = 101,
   POK_SYSCALL_MIDDLEWARE_SAMPLING_READ            = 102,
   POK_SYSCALL_MIDDLEWARE_SAMPLING_STATUS          = 103,
   POK_SYSCALL_MIDDLEWARE_SAMPLING_WRITE           = 104,
   POK_SYSCALL_MIDDLEWARE_SAMPLING_CREATE          = 105,
#endif
#ifdef POK_NEEDS_PORTS_QUEUEING
   POK_SYSCALL_MIDDLEWARE_QUEUEING_CREATE          = 110,
   POK_SYSCALL_MIDDLEWARE_QUEUEING_SEND            = 111,
   POK_SYSCALL_MIDDLEWARE_QUEUEING_RECEIVE         = 112,
   POK_SYSCALL_MIDDLEWARE_QUEUEING_ID              = 113,
   POK_SYSCALL_MIDDLEWARE_QUEUEING_STATUS          = 114,
#endif
#ifdef POK_NEEDS_PORTS_VIRTUAL
   POK_SYSCALL_MIDDLEWARE_VIRTUAL_CREATE           = 150,
   POK_SYSCALL_MIDDLEWARE_VIRTUAL_NB_DESTINATIONS  = 151,
   POK_SYSCALL_MIDDLEWARE_VIRTUAL_DESTINATION      = 152,
   POK_SYSCALL_MIDDLEWARE_VIRTUAL_GET_GLOBAL       = 153,
#endif
#if defined (POK_NEEDS_LOCKOBJECTS) || defined (POK_NEEDS_MUTEXES) || defined
       (POK_NEEDS_SEMAPHORES) || defined (POK_NEEDS_EVENTS) || defined
       (POK_NEEDS_BUFFERS) || defined (POK_NEEDS_BLACKBOARDS)
   POK_SYSCALL_LOCKOBJ_CREATE                      = 201,
   POK_SYSCALL_LOCKOBJ_OPERATION                   = 202,
#endif
#ifdef POK_NEEDS_ERROR_HANDLING
   POK_SYSCALL_ERROR_HANDLER_CREATE                = 301,
   POK_SYSCALL_ERROR_HANDLER_SET_READY             = 302,
   POK_SYSCALL_ERROR_RAISE_APPLICATION_ERROR       = 303,
   POK_SYSCALL_ERROR_GET                           = 304,
#endif
#ifdef POK_NEEDS_PARTITIONS
   POK_SYSCALL_PARTITION_SET_MODE                  = 404,
   POK_SYSCALL_PARTITION_GET_ID                    = 405,
   POK_SYSCALL_PARTITION_GET_PERIOD                = 406,
   POK_SYSCALL_PARTITION_GET_DURATION              = 407,
   POK_SYSCALL_PARTITION_GET_LOCK_LEVEL            = 408,
   POK_SYSCALL_PARTITION_GET_OPERATING_MODE        = 409,
   POK_SYSCALL_PARTITION_GET_START_CONDITION       = 410,
#endif
#ifdef POK_NEEDS_IO
   POK_SYSCALL_INB                                 = 501,
   POK_SYSCALL_OUTB                                = 502,
#endif
#ifdef POK_NEEDS_PCI
   POK_SYSCALL_PCI_REGISTER                        = 601,
#endif
} pok_syscall_id_t;
```

4.95.3 Function Documentation

4.95.3.1 pok_ret_t pok_core_syscall (const pok_syscall_id_t *syscall_id*, const pok_syscall_args_t ∗ *args*, const pok_syscall_info_t ∗ *infos*)

Function that performs the syscall. It is called by the architecture interruption handler.

Parameters

syscall_id	This param correspond to the syscal which should be performed. The list of available syscalls is available in the definition of the pok_syscall-_id_t type
args	Arguments of the syscall. It corresponds to data useful to perform the syscall.
infos	Informations about the syscall: which partition/thread initiates the syscall, etc ...

Returns

Returns an error code, which is defined in include/errno.h

Here is the default syscall handler. In this case, the syscall ID was not properly identified and thus, we should return an error. If error management is activated, we raise an error in kernel of partitions, calling the error handler.

Definition at line 40 of file syscall.c.

```
{
   switch (syscall_id)
   {
#if defined (POK_NEEDS_CONSOLE) || defined (POK_NEEDS_DEBUG)
      case POK_SYSCALL_CONSWRITE:
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg1 + infos->base_addr
      )
         if (pok_cons_write ((const char*)args->arg1 + infos->base_addr, args->
      arg2))
         {
            return POK_ERRNO_OK;
         }
         else
         {
            return POK_ERRNO_EINVAL;
         }
         break;
#endif

#ifdef POK_NEEDS_PORTS_VIRTUAL
      case POK_SYSCALL_MIDDLEWARE_VIRTUAL_CREATE:
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg1 + infos->base_addr
      )
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg2 + infos->
      base_addr)
         return pok_port_virtual_id ( (char*) (args->arg1 + infos->base_addr),
      (pok_port_id_t*) (args->arg2 + infos->base_addr));
         break;

      case POK_SYSCALL_MIDDLEWARE_VIRTUAL_NB_DESTINATIONS:
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg2 + infos->
      base_addr)
         return pok_port_virtual_nb_destinations ( (pok_port_id_t) (args->arg1)
      , (uint32_t*) (args->arg2 + infos->base_addr));
         break;

      case POK_SYSCALL_MIDDLEWARE_VIRTUAL_DESTINATION:
         POK_CHECK_PTR_OR_RETURN(infos->partition, ((void*) args->arg3)+infos->
      base_addr)
```

```
         return pok_port_virtual_destination ( (pok_port_id_t) (args->arg1), (
      uint32_t) (args->arg2), (uint32_t*) (args->arg3 + infos->base_addr));
         break;

      case POK_SYSCALL_MIDDLEWARE_VIRTUAL_GET_GLOBAL:
         POK_CHECK_PTR_OR_RETURN(infos->partition, (void*) (args->arg2 + infos-
      >base_addr))
         return pok_port_virtual_get_global ((pok_port_id_t) (args->arg1), (
      pok_port_id_t*) (args->arg2 + infos->base_addr));
         break;

#endif

#if defined POK_NEEDS_GETTICK
      case POK_SYSCALL_GETTICK:
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg1 + infos->base_addr
      )
         return pok_gettick_by_pointer ((uint64_t*) (args->arg1 + infos->
      base_addr));
         break;
#endif

      case POK_SYSCALL_THREAD_CREATE:
         return pok_partition_thread_create  ((uint32_t*)         (args->arg1 +
       infos->base_addr),
                                             (pok_thread_attr_t*) (args->arg2 +
       infos->base_addr),
                                             (uint8_t)            infos->
      partition);
         break;

#ifdef POK_NEEDS_THREAD_SLEEP
      case POK_SYSCALL_THREAD_SLEEP:
         return pok_thread_sleep (args->arg1);
         break;
#endif

#ifdef POK_NEEDS_THREAD_SLEEP_UNTIL
      case POK_SYSCALL_THREAD_SLEEP_UNTIL:
         return pok_thread_sleep_until (args->arg1);
         break;
#endif

      case POK_SYSCALL_THREAD_PERIOD:
         return pok_sched_end_period ();
         break;

#if defined (POK_NEEDS_THREAD_SUSPEND) || defined (POK_NEEDS_ERROR_HANDLING)
      case POK_SYSCALL_THREAD_SUSPEND:
         return pok_thread_suspend ();
         break;
#endif

#ifdef POK_NEEDS_THREAD_ID
      case POK_SYSCALL_THREAD_ID:
         return pok_sched_get_current ((uint32_t*) (args->arg1 + infos->
      base_addr));
         break;
#endif
      case POK_SYSCALL_THREAD_STATUS:
        return pok_thread_get_status (args->arg1, (pok_thread_attr_t*) (args->
      arg2 + infos->base_addr));
```

```
         break;

#ifdef POK_NEEDS_ERROR_HANDLING

      case POK_SYSCALL_THREAD_RESTART:
         return pok_partition_restart_thread (args->arg1);
         break;

      case POK_SYSCALL_THREAD_STOP:
         return pok_partition_stop_thread (args->arg1);
         break;

      case POK_SYSCALL_THREAD_STOPSELF:
         pok_sched_stop_self ();
         return POK_ERRNO_OK;
         break;

#endif

#ifdef POK_NEEDS_PARTITIONS
      case POK_SYSCALL_PARTITION_SET_MODE:
         return pok_partition_set_mode_current ((pok_partition_mode_t)args->arg1
      );
         break;
      case POK_SYSCALL_PARTITION_GET_ID:
        return pok_current_partition_get_id ((uint8_t*)(args->arg1 + infos->
      base_addr));
         break;
      case POK_SYSCALL_PARTITION_GET_PERIOD:
        return pok_current_partition_get_period ((uint64_t*)(args->arg1 + infos
      ->base_addr));
         break;
      case POK_SYSCALL_PARTITION_GET_DURATION:
        return pok_current_partition_get_duration ((uint64_t*)(args->arg1 +
      infos->base_addr));
         break;
      case POK_SYSCALL_PARTITION_GET_LOCK_LEVEL:
        return pok_current_partition_get_lock_level ((uint32_t*)(args->arg1 +
      infos->base_addr));
         break;
      case POK_SYSCALL_PARTITION_GET_OPERATING_MODE:
        return pok_current_partition_get_operating_mode ((pok_partition_mode_t*
      )(args->arg1 + infos->base_addr));
         break;
      case POK_SYSCALL_PARTITION_GET_START_CONDITION:
        return pok_current_partition_get_start_condition ((
      pok_start_condition_t*)(args->arg1 + infos->base_addr));
         break;
#endif

#ifdef POK_NEEDS_ERROR_HANDLING
      case POK_SYSCALL_ERROR_HANDLER_CREATE:
         return pok_error_thread_create (args->arg1 , (void*) (args->arg2));
         break;

      case POK_SYSCALL_ERROR_RAISE_APPLICATION_ERROR:
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg1 + infos->base_addr
      )
         pok_error_raise_application_error ((char*) (args->arg1 + infos->
      base_addr), args->arg2);
         return POK_ERRNO_OK;
         break;
      case POK_SYSCALL_ERROR_GET:
         return pok_error_get ((pok_error_status_t*) (args->arg1 + infos->
      base_addr));
         break;
#endif

         /* Middleware syscalls */
#ifdef POK_NEEDS_PORTS_SAMPLING
      case POK_SYSCALL_MIDDLEWARE_SAMPLING_CREATE:
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg5 + infos->base_addr
      )
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg1 + infos->
      base_addr)
         return pok_port_sampling_create  ((char*)(args->arg1 + infos->
      base_addr),
                                           (pok_port_size_t) args->arg2,
                                           (pok_port_direction_t) args->arg3,
                                           (uint64_t) args->arg4,
                                           (pok_port_id_t*) (args->arg5 + infos-
      >base_addr));
         break;

      case POK_SYSCALL_MIDDLEWARE_SAMPLING_WRITE:
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg2 + infos->
      base_addr)

         return pok_port_sampling_write   ((const pok_port_id_t)args->arg1,
                                           (const void*) ((void*)args->arg2 +
      infos->base_addr),
                                           (const uint8_t) args->arg3);
         break;

      case POK_SYSCALL_MIDDLEWARE_SAMPLING_READ:
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg2 + infos->
      base_addr)
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg4 + infos->
      base_addr)
         return pok_port_sampling_read ((const pok_port_id_t)args->arg1,
                                        (void*) args->arg2 + infos->base_addr,
                                        (pok_port_size_t*) (args->arg3 + infos->
      base_addr),
                                        (bool_t*) (args->arg4 + infos->base_addr
      ));
         break;

      case POK_SYSCALL_MIDDLEWARE_SAMPLING_ID:
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg1 + infos->
      base_addr)
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg2 + infos->
      base_addr)
         return pok_port_sampling_id    ((char*)(args->arg1 + infos->base_addr),
                                        (pok_port_id_t*)(args->arg2 + infos->
      base_addr));
         break;

#ifndef POK_GENERATED_CODE
      case POK_SYSCALL_MIDDLEWARE_SAMPLING_STATUS:
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg2+infos->base_addr)
         return pok_port_sampling_status ((const pok_port_id_t)args->arg1,
                                           (pok_port_sampling_status_t*) (args->
      arg2 + infos->base_addr));
```

```
         break;
#endif /* POK_GENERATED_CODE */
#endif /* POK_NEEDS_PORTS_SAMPLING */


#ifdef POK_NEEDS_PORTS_QUEUEING
      case POK_SYSCALL_MIDDLEWARE_QUEUEING_CREATE:
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg1 + infos->base_addr
      )
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg5 + infos->
      base_addr)
         return pok_port_queueing_create ((char*)                         (
      args->arg1 + infos->base_addr),
                                   (pok_port_size_t)                 args
      ->arg2,
                                   (pok_port_direction_t)            args
      ->arg3,
                                   (pok_port_queueing_discipline_t)  args
      ->arg4,
                                   (pok_port_id_t*)                  (
      args->arg5 + infos->base_addr));
         break;

      case POK_SYSCALL_MIDDLEWARE_QUEUEING_SEND:
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg2 + infos->
      base_addr)
         return pok_port_queueing_send ((const pok_port_id_t)             args
      ->arg1,
                                 (const void*)                     ((
      void*)args->arg2 + infos->base_addr),
                                 (const uint8_t)                   (
      args->arg3),
                                 (const uint64_t)                  args
      ->arg4);
         break;

      case POK_SYSCALL_MIDDLEWARE_QUEUEING_RECEIVE:
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg4 + infos->
      base_addr)
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg5 + infos->
      base_addr)
         return pok_port_queueing_receive ((const pok_port_id_t)   args->arg1,
                                    (uint64_t)              args->arg2,
                                    (pok_port_size_t)       args->arg3,
                                    (void*)                 ((void*)args-
      >arg4 + infos->base_addr),
                                    (pok_port_size_t*)      (args->arg5 +
       infos->base_addr));
         break;

      case POK_SYSCALL_MIDDLEWARE_QUEUEING_ID:
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg1 + infos->
      base_addr)
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg2 + infos->
      base_addr)
         return pok_port_queueing_id ((char*)          (args->arg1 + infos->
      base_addr),
                              (pok_port_id_t*)  (args->arg2 + infos->
      base_addr));
         break;

#ifndef POK_GENERATED_CODE
```

```
      case POK_SYSCALL_MIDDLEWARE_QUEUEING_STATUS:
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg2 + infos->base_addr
      )
         return pok_port_queueing_status     ((const pok_port_id_t)
      args->arg1,
                                             (pok_port_queueing_status_t*)    (
      args->arg2 + infos->base_addr));
         break;
#endif
#endif /* POK_NEEDS_PORTS_QUEUEING */

#ifdef POK_NEEDS_LOCKOBJECTS
      case POK_SYSCALL_LOCKOBJ_CREATE:
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg2+infos->base_addr)
         POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg1+infos->base_addr)
         return pok_lockobj_partition_create    ((pok_lockobj_id_t*)    (args->
      arg1 + infos->base_addr),
                                                (pok_lockobj_attr_t*)   (args->
      arg2 + infos->base_addr));
         break;

      case POK_SYSCALL_LOCKOBJ_OPERATION:
         if (args->arg2 == NULL)
         {
            return pok_lockobj_partition_wrapper ((const uint8_t) args->arg1,
      NULL);
         }
         else
         {
            POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg2 + infos->
      base_addr)
            return pok_lockobj_partition_wrapper   ((const uint8_t) args->arg1,
                                                   (pok_lockobj_lockattr_t*) (
      args->arg2 + infos->base_addr));
         }
         break;
#endif /* POK_NEEDS_LOCKOBJECTS */

#ifdef POK_NEEDS_IO
      case POK_SYSCALL_INB:
         if ((args->arg1 < pok_partitions[infos->partition].io_min) ||
             (args->arg1 > pok_partitions[infos->partition].io_max))
         {
            return -POK_ERRNO_EPERM;
         }
         else
         {
            return inb((unsigned short) args->arg1);
         }
         break;

      case POK_SYSCALL_OUTB:
         if ((args->arg1 < pok_partitions[infos->partition].io_min) ||
             (args->arg1 > pok_partitions[infos->partition].io_max))
         {
            return -POK_ERRNO_EPERM;
         }
         else
         {
            outb((unsigned short) args->arg1, (unsigned char) args->arg2);
            return POK_ERRNO_OK;
         }
```

```
       break;
#endif /* POK_NEEDS_IO */

#ifdef POK_NEEDS_PCI
     case POK_SYSCALL_PCI_REGISTER:
       POK_CHECK_PTR_OR_RETURN(infos->partition, args->arg1 + infos->base_addr)
       return pci_register((void*)args->arg1 + infos->base_addr, infos->
      partition);
       break;
#endif /* POK_NEEDS_PCI */


      default:
#ifdef POK_NEEDS_ERROR_HANDLING
         pok_error_declare (POK_ERROR_KIND_ILLEGAL_REQUEST);
         pok_sched_activate_error_thread ();
#else
         #ifdef POK_NEEDS_DEBUG
            printf ("Tried to use syscall %d\n", syscall_id);
         #endif
         POK_FATAL ("Unknown syscall");
#endif
         break;
   }

   return POK_ERRNO_EINVAL;
}
```

4.95.3.2 pok_ret_t pok_syscall_init ( )

Init system calls

Definition at line 83 of file syscalls.c.

```
{
   pok_idt_set_gate (POK_SYSCALL_INT_NUMBER,
                     GDT_CORE_CODE_SEGMENT << 3,
                     (uint32_t) syscall_gate,
                     IDTE_INTERRUPT,
                     3);

   return (POK_ERRNO_OK);
}
```

4.96 /home/matias/projet/stage-pok/pok/trunk/kernel/include/core/time.h File Reference

4.97 /home/matias/projet/stage-pok/pok/trunk/kernel/include/dependencies.h File Reference

4.98 /home/matias/projet/stage-pok/pok/trunk/kernel/include/elf.h - File Reference

Data Structures

- struct Elf32_Ehdr
- struct Elf32_Phdr

Defines

- #define EI_NIDENT (16)

Typedefs

- typedef uint16_t Elf32_Half
- typedef uint32_t Elf32_Word
- typedef uint32_t Elf32_Off
- typedef uint32_t Elf32_Addr

4.98.1 Define Documentation

4.98.1.1 #define EI_NIDENT (16)

Definition at line 26 of file elf.h.

4.98.2 Typedef Documentation

4.98.2.1 typedef uint32_t Elf32_Addr

Definition at line 24 of file elf.h.

4.98.2.2 typedef uint16_t Elf32_Half

Definition at line 21 of file elf.h.

4.98.2.3 typedef uint32_t Elf32_Off

Definition at line 23 of file elf.h.

4.98.2.4 typedef uint32_t Elf32_Word

Definition at line 22 of file elf.h.

4.99 /home/matias/projet/stage-pok/pok/trunk/kernel/include/errno.h File Reference

Enumerations

- enum pok_ret_t { POK_ERRNO_OK = 0, POK_ERRNO_EINVAL = 1, POK_ERRNO_UNAVAILABLE = 2, POK_ERRNO_TOOMANY = 5, POK_ERRNO_EPERM = 6, POK_ERRNO_EXISTS = 7, POK_ERRNO_ERANGE = 8, POK_ERRNO_EDOM = 9, POK_ERRNO_HUGE_VAL = 10, POK_ERRNO_EFAULT = 11, POK_ERRNO_THREAD = 49, POK_ERRNO_THREADATTR = 50, POK_ERRNO_TIME = 100, POK_ERRNO_PARTITION_ATTR = 200, POK_ERRNO_PORT = 301, POK_ERRNO_NOTFOUND = 302, POK_ERRNO_DIRECTION = 303, POK_ERRNO_SIZE = 304, POK_ERRNO_DISCIPLINE = 305, POK_ERRNO_PORTPART = 307, POK_ERRNO_EMPTY = 308, POK_ERRNO_KIND = 309, POK_ERRNO_FULL = 311, POK_ERRNO_READY = 310, POK_ERRNO_TIMEOUT = 250, POK_ERRNO_MODE = 251, POK_ERRNO_LOCKOBJ_UNAVAILABLE = 500, POK_ERRNO_LOCKOBJ_NOTREADY = 501, POK_ERRNO_LOCKOBJ_KIND = 502, POK_ERRNO_LOCKOBJ_POLICY = 503, POK_ERRNO_PARTITION_MODE = 601, POK_ERRNO_PARTITION = 401 }

4.99.1 Enumeration Type Documentation

4.99.1.1 enum pok_ret_t

Enumerator:

POK_ERRNO_OK

POK_ERRNO_EINVAL

POK_ERRNO_UNAVAILABLE

POK_ERRNO_TOOMANY

POK_ERRNO_EPERM

POK_ERRNO_EXISTS

POK_ERRNO_ERANGE

POK_ERRNO_EDOM

POK_ERRNO_HUGE_VAL

POK_ERRNO_EFAULT

POK_ERRNO_THREAD

POK_ERRNO_THREADATTR

POK_ERRNO_TIME

POK_ERRNO_PARTITION_ATTR

POK_ERRNO_PORT

POK_ERRNO_NOTFOUND

POK_ERRNO_DIRECTION

POK_ERRNO_SIZE

POK_ERRNO_DISCIPLINE

POK_ERRNO_PORTPART

POK_ERRNO_EMPTY

POK_ERRNO_KIND

POK_ERRNO_FULL

POK_ERRNO_READY

POK_ERRNO_TIMEOUT

POK_ERRNO_MODE

POK_ERRNO_LOCKOBJ_UNAVAILABLE

POK_ERRNO_LOCKOBJ_NOTREADY

POK_ERRNO_LOCKOBJ_KIND

POK_ERRNO_LOCKOBJ_POLICY

POK_ERRNO_PARTITION_MODE

POK_ERRNO_PARTITION

Definition at line 21 of file errno.h.

```
{
    POK_ERRNO_OK                    =   0,
    POK_ERRNO_EINVAL                =   1,

    POK_ERRNO_UNAVAILABLE           =   2,
    POK_ERRNO_TOOMANY               =   5,
    POK_ERRNO_EPERM                 =   6,
    POK_ERRNO_EXISTS                =   7,

    POK_ERRNO_ERANGE                =   8,
    POK_ERRNO_EDOM                  =   9,
    POK_ERRNO_HUGE_VAL              =  10,

    POK_ERRNO_EFAULT                =  11,

    POK_ERRNO_THREAD                =  49,
    POK_ERRNO_THREADATTR            =  50,

    POK_ERRNO_TIME                 =  100,

    POK_ERRNO_PARTITION_ATTR        = 200,

    POK_ERRNO_PORT                 =  301,
    POK_ERRNO_NOTFOUND             =  302,
    POK_ERRNO_DIRECTION            =  303,
    POK_ERRNO_SIZE                 =  304,
    POK_ERRNO_DISCIPLINE           =  305,
    POK_ERRNO_PORTPART             =  307,
    POK_ERRNO_EMPTY                =  308,
    POK_ERRNO_KIND                 =  309,
    POK_ERRNO_FULL                 =  311,
    POK_ERRNO_READY                =  310,
    POK_ERRNO_TIMEOUT              =  250,
    POK_ERRNO_MODE                 =  251,
    POK_ERRNO_LOCKOBJ_UNAVAILABLE  =  500,
    POK_ERRNO_LOCKOBJ_NOTREADY     =  501,
    POK_ERRNO_LOCKOBJ_KIND         =  502,
    POK_ERRNO_LOCKOBJ_POLICY       =  503,

    POK_ERRNO_PARTITION_MODE       =  601,

    POK_ERRNO_PARTITION            =  401
} pok_ret_t;
```

4.100 /home/matias/projet/stage-pok/pok/trunk/kernel/include/libc.h File Reference

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

Functions

- void ∗ memcpy (void ∗to, const void ∗from, size_t n)
- void ∗ memset (void ∗dest, unsigned char val, size_t count)
- int strlen (const char ∗str)
- int strcmp (const char ∗s1, const char ∗s2)
- int strncmp (const char ∗s1, const char ∗s2, size_t size)

4.100.1 Function Documentation

4.100.1.1 void∗ memcpy (void ∗ *to,* const void ∗ *from,* size_t *n*)

Definition at line 20 of file memcpy.c.

```
{
#ifdef __i386__
  int d0;
  int d1;
  int d2;

  __asm__ __volatile__(
                     "rep ; movsl\n\t"
                     "testb $2,%b4\n\t"
                     "je 1f\n\t"
                     "movsw\n"
                     "1:\ttestb $1,%b4\n\t"
                     "je 2f\n\t"
                     "movsb\n"
                     "2:"
                     : "=&c" (d0), "=&D" (d1), "=&S" (d2)
                     :"0" (n/4), "q" (n),"1" ((long) to),"2" ((long) from)
                     : "memory");
#else
  char *cto = (char *)to;
  const char *cfrom = (const char *)from;
  for (; n > 0; n--)
  {
       *cto++ = *cfrom++;
  }
#endif
  return (to);
}
```

4.100.1.2 void∗ memset (void ∗ *dest*, unsigned char *val*, size_t *count*)

4.100.1.3 int strcmp (const char ∗ *s1*, const char ∗ *s2*)

4.100.1.4 int strlen (const char ∗ *str*)

4.100.1.5 int strncmp (const char ∗ *s1*, const char ∗ *s2*, size_t *size*)

4.101 /home/matias/projet/stage-pok/pok/trunk/kernel/include/middleware/port.h File Reference

Describe queueing and sampling ports structures.

`#include <types.h> #include <errno.h> #include <core/lockobj.-
h>`

Data Structures

- struct pok_port_t

Defines

- #define POK_PORT_MAX_SIZE 512

Typedefs

- typedef pok_queueing_discipline_t pok_port_queueing_discipline_t

Enumerations

- enum pok_port_queueing_disciplines_t { POK_PORT_QUEUEING_DISCIPLIN-E_FIFO = 1, POK_PORT_QUEUEING_DISCIPLINE_PRIORITY = 2 }
- enum pok_port_directions_t { POK_PORT_DIRECTION_IN = 1, POK_PORT_D-IRECTION_OUT = 2 }
- enum pok_port_kinds_t { POK_PORT_KIND_QUEUEING = 1, POK_PORT_KI-ND_SAMPLING = 2, POK_PORT_KIND_INVALID = 10 }

4.101.1 Detailed Description

Describe queueing and sampling ports structures.

Date

2008-2009

Author

Julien Delange

Definition in file port.h.

4.101.2 Define Documentation

4.101.2.1 #define **POK_PORT_MAX_SIZE** 512

Definition at line 31 of file port.h.

4.101.3 Typedef Documentation

4.101.3.1 typedef **pok_queueing_discipline_t pok_port_queueing_discipline_t**

Definition at line 45 of file port.h.

4.101.4 Enumeration Type Documentation

4.101.4.1 enum **pok_port_directions_t**

Enumerator:

POK_PORT_DIRECTION_IN

POK_PORT_DIRECTION_OUT

Definition at line 39 of file port.h.

```
{
   POK_PORT_DIRECTION_IN   = 1,
   POK_PORT_DIRECTION_OUT  = 2
} pok_port_directions_t;
```

4.101.4.2 enum **pok_port_kinds_t**

Enumerator:

POK_PORT_KIND_QUEUEING

POK_PORT_KIND_SAMPLING

POK_PORT_KIND_INVALID

Definition at line 47 of file port.h.

```
{
   POK_PORT_KIND_QUEUEING  = 1,
   POK_PORT_KIND_SAMPLING  = 2,
#ifdef POK_NEEDS_PORTS_VIRTUAL
   POK_PORT_KIND_VIRTUAL   = 2,
#endif
   POK_PORT_KIND_INVALID   = 10
} pok_port_kinds_t;
```

4.101.4.3 enum pok_port_queueing_disciplines_t

Enumerator:

POK_PORT_QUEUEING_DISCIPLINE_FIFO

POK_PORT_QUEUEING_DISCIPLINE_PRIORITY

Definition at line 33 of file port.h.

```
{
   POK_PORT_QUEUEING_DISCIPLINE_FIFO      = 1,
   POK_PORT_QUEUEING_DISCIPLINE_PRIORITY  = 2
} pok_port_queueing_disciplines_t;
```

4.102 /home/matias/projet/stage-pok/pok/trunk/kernel/include/middleware/queue.h File Reference

4.103 /home/matias/projet/stage-pok/pok/trunk/kernel/libc/__-udivdi3.c File Reference

Functions

- unsigned long long __udivdi3 (unsigned long long num, unsigned long long den)

4.103.1 Function Documentation

4.103.1.1 unsigned long long __udivdi3 (unsigned long long *num*, unsigned long long *den*)

Definition at line 19 of file __udivdi3.c.

```
{
#ifdef POK_NEEDS_DEBUG
   unsigned long long quot, qbit;

   quot = 0;
   qbit = 1;

   if (den == 0)
   {
      return 0;
   }

   while ((long long) den >= 0)
   {
      den <<= 1;
      qbit <<= 1;
   }

   while (qbit)
   {
      if (den <= num)
      {
         num -= den;
         quot += qbit;
      }
      den >>= 1;
      qbit >>= 1;
   }

   return quot;
#else
   (void) num;
   (void) den;
   return 0;
#endif
}
```

4.104 /home/matias/projet/stage-pok/pok/trunk/kernel/libc/memcpy.c File Reference

```
#include <libc.h>
```

Functions

- void ∗ memcpy (void ∗to, const void ∗from, size_t n)

4.104.1 Function Documentation

4.104.1.1 void∗ memcpy (void ∗ *to,* const void ∗ *from,* size_t *n*)

Definition at line 20 of file memcpy.c.

```
{
```

```
#ifdef __i386__
  int d0;
  int d1;
  int d2;

  __asm__ __volatile__(
                       "rep ; movsl\n\t"
                       "testb $2,%b4\n\t"
                       "je 1f\n\t"
                       "movsw\n"
                       "1:\ttestb $1,%b4\n\t"
                       "je 2f\n\t"
                       "movsb\n"
                       "2:"
                       : "=&c" (d0), "=&D" (d1), "=&S" (d2)
                       :"0" (n/4), "q" (n),"1" ((long) to),"2" ((long) from)
                       : "memory");
#else
  char *cto = (char *)to;
  const char *cfrom = (const char *)from;

  for (; n > 0; n--)
  {
       *cto++ = *cfrom++;
  }
#endif
  return (to);
}
```

4.105 /home/matias/projet/stage-pok/pok/trunk/kernel/libc/memset.c File Reference

`#include <libc.h>`

Functions

- __attribute__ ((weak))

4.105.1 Function Documentation

4.105.1.1 __attribute__ ((weak))

Definition at line 20 of file memset.c.

```
{
  unsigned char *d = (unsigned char *) dest;

  while (count--)
  {
    *d++ = val;
  }

  return dest;
}
```

4.106 /home/matias/projet/stage-pok/pok/trunk/kernel/libc/printf.c - File Reference

4.107 /home/matias/projet/stage-pok/pok/trunk/kernel/libc/strcmp.c File Reference

4.108 /home/matias/projet/stage-pok/pok/trunk/kernel/libc/strlen.c File Reference

4.109 /home/matias/projet/stage-pok/pok/trunk/kernel/middleware/portcreate.c File Reference

4.110 /home/matias/projet/stage-pok/pok/trunk/kernel/middleware/portflushall.c File Reference

Flush the ports and send the data of IN ports to OUT ports.

4.110.1 Detailed Description

Flush the ports and send the data of IN ports to OUT ports.

Date

2008-2009

Author

Julien Delange
Laurent Lec

Definition in file portflushall.c.

4.111 /home/matias/projet/stage-pok/pok/trunk/kernel/middleware/portinit.c File Reference

4.112 /home/matias/projet/stage-pok/pok/trunk/kernel/middleware/portqueueingcreate.c File Reference

4.113 /home/matias/projet/stage-pok/pok/trunk/kernel/middleware/portqueueingid.c File Reference

4.114 /home/matias/projet/stage-pok/pok/trunk/kernel/middleware/portqueueingreceive.c File Reference

4.115 /home/matias/projet/stage-pok/pok/trunk/kernel/middleware/portqueueingsend.c File Reference

4.116 /home/matias/projet/stage-pok/pok/trunk/kernel/middleware/portqueueingstatus.c File Reference

4.117 /home/matias/projet/stage-pok/pok/trunk/kernel/middleware/portsamplingcreate.c File Reference

4.118 /home/matias/projet/stage-pok/pok/trunk/kernel/middleware/portsamplingid.c File Reference

4.119 /home/matias/projet/stage-pok/pok/trunk/kernel/middleware/portsamplingread.c File Reference

4.120 /home/matias/projet/stage-pok/pok/trunk/kernel/middleware/portsamplingstatus.c File Reference

4.121 /home/matias/projet/stage-pok/pok/trunk/kernel/middleware/portsamplingwrite.c File Reference

Send data on a sampling port.

4.121.1 Detailed Description

Send data on a sampling port.

Author

Julien Delange

Date

2008-2009

Definition in file portsamplingwrite.c.

4.122 /home/matias/projet/stage-pok/pok/trunk/kernel/middleware/portutils.c File Reference

Various functions for ports management.

4.122.1 Detailed Description

Various functions for ports management.

Date

2008-2009

Author

Julien Delange

Definition in file portutils.c.

4.123 /home/matias/projet/stage-pok/pok/trunk/kernel/middleware/portvirtualdestinatio File Reference

4.124 /home/matias/projet/stage-pok/pok/trunk/kernel/middleware/portvirtualgetglobal. File Reference

4.125 /home/matias/projet/stage-pok/pok/trunk/kernel/middleware/portvirtualid.c File Reference

4.126 /home/matias/projet/stage-pok/pok/trunk/kernel/middleware/portvirtualnbdestina File Reference

4.127 /home/matias/projet/stage-pok/pok/trunk/kernel/middleware/queueinit.c File Reference

4.128 /home/matias/projet/stage-pok/pok/trunk/kernel/middleware/ressources.c File Reference