POK

Generated by Doxygen 1.8.1.2

Fri Jan 25 2013 11:46:26

Contents

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 __attribute__ Struct Reference

Public Attributes

- char **dst** [ETH_MAC_LEN]
- char **src** [ETH_MAC_LEN]
- unsigned short **ethertype**
- unsigned short **src**
- unsigned short **dst**
- unsigned short **len**
- unsigned short **chk**
- eth_hdr_t **eth**
- udp_hdr_t **udp**
- char **data** [NET_DATA_MAXLEN]
- uint32_t **len**
- uint32_t **off**
- unsigned char **status**
- unsigned char **next**
- unsigned short **size**

3.1.1 Detailed Description

Definition at line 34 of file rtl8029.h.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/drivers/rtl8029.h

3.2 ARINC_ATTRIBUTE Struct Reference

`#include <arincutils.h>`

Public Attributes

- PROCESS_NAME_TYPE **NAME**
- PRIORITY_TYPE **BASE_PRIORITY**
- STACK_SIZE_TYPE **STACK_SIZE**

3.2.1 Detailed Description

Struct for save data NAME => Not use by pok BASE_PRIORITY => This value, in pok, is modified. Here we save the base value

Definition at line 26 of file arincutils.h.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/include/arinc653/arincutils.h

3.3 bf_key_st Struct Reference

Public Attributes

- BF_LONG **P** [BF_ROUNDS+2]
- BF_LONG **S** [4 ∗256]

3.3.1 Detailed Description

Definition at line 100 of file blowfish.h.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/protocols/blowfish/blowfish.h

3.4 BLACKBOARD_STATUS_TYPE Struct Reference

Public Attributes

- EMPTY_INDICATOR_TYPE **EMPTY_INDICATOR**
- MESSAGE_SIZE_TYPE **MAX_MESSAGE_SIZE**
- WAITING_RANGE_TYPE **WAITING_PROCESSES**

3.4.1 Detailed Description

Definition at line 44 of file blackboard.h.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/include/arinc653/blackboard.h

3.5 BUFFER_STATUS_TYPE Struct Reference

Public Attributes

- MESSAGE_RANGE_TYPE **NB_MESSAGE**
- MESSAGE_RANGE_TYPE **MAX_NB_MESSAGE**
- MESSAGE_SIZE_TYPE **MAX_MESSAGE_SIZE**
- WAITING_RANGE_TYPE **WAITING_PROCESSES**

3.5.1 Detailed Description

Definition at line 43 of file buffer.h.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/include/arinc653/buffer.h

3.6 DES_ks Struct Reference

Public Attributes

- union {
  DES_cblock **cblock**
  DES_LONG **deslong** [2]
  } **ks** [16]

3.6.1 Detailed Description

Definition at line 97 of file des.h.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/protocols/des/des.h

3.7 ERROR_STATUS_TYPE Struct Reference

Public Attributes

- ERROR_CODE_TYPE **ERROR_CODE**
- MESSAGE_SIZE_TYPE **LENGTH**
- PROCESS_ID_TYPE **FAILED_PROCESS_ID**
- SYSTEM_ADDRESS_TYPE **FAILED_ADDRESS**
- ERROR_MESSAGE_TYPE **MESSAGE**

3.7.1 Detailed Description

Definition at line 51 of file error.h.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/include/arinc653/error.h

3.8 EVENT_STATUS_TYPE Struct Reference

Public Attributes

- EVENT_STATE_TYPE **EVENT_STATE**
- WAITING_RANGE_TYPE **WAITING_PROCESSES**

3.8.1 Detailed Description

Definition at line 64 of file event.h.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/include/arinc653/event.h

3.9 exception Struct Reference

Public Attributes

- int **type**
- char * **name**
- double **arg1**
- double **arg2**
- double **retval**

3.9.1 Detailed Description

Definition at line 26 of file libm.h.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/include/libm.h

3.10 ieee_double_shape_type Union Reference

Public Attributes

- double **value**
- struct {
  uint32_t **lsw**
  uint32_t **msw**
  } **parts**

3.10.1 Detailed Description

Definition at line 87 of file math_private.h.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/libm/math_private.h

3.11 ieee_float_shape_type Union Reference

Public Attributes

- float **value**
- uint32_t **word**

3.11.1 Detailed Description

Definition at line 159 of file math_private.h.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/libm/math_private.h

3.12 PARTITION_STATUS_TYPE Struct Reference

Public Attributes

- SYSTEM_TIME_TYPE **PERIOD**
- SYSTEM_TIME_TYPE **DURATION**
- PARTITION_ID_TYPE **IDENTIFIER**
- LOCK_LEVEL_TYPE **LOCK_LEVEL**
- OPERATING_MODE_TYPE **OPERATING_MODE**
- START_CONDITION_TYPE **START_CONDITION**

3.12.1 Detailed Description

Definition at line 43 of file partition.h.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/include/arinc653/partition.h

3.13 pok_allocator_space_t Struct Reference

Public Attributes

- size_t **start**
- size_t **size**
- bool_t **allocated**

3.13.1 Detailed Description

Definition at line 60 of file allocator.c.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/core/allocator.c

3.14 pok_arinc653_event_layer_t Struct Reference

Public Attributes

- pok_event_id_t **core_id**
- pok_bool_t **ready**

3.14.1 Detailed Description

Definition at line 43 of file event.c.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/arinc653/event.c

3.15 pok_arinc653_semaphore_layer_t Struct Reference

Public Attributes

- pok_bool_t **ready**
- pok_sem_id_t **core_id**

3.15.1 Detailed Description

Definition at line 42 of file semaphore.c.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/arinc653/semaphore.c

3.16 pok_blackboard_status_t Struct Reference

Public Attributes

- pok_port_size_t **msg_size**
- pok_bool_t **empty**
- pok_range_t **waiting_processes**

3.16.1 Detailed Description

Definition at line 37 of file blackboard.h.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/include/middleware/blackboard.h

3.17 pok_blackboard_t Struct Reference

Public Attributes

- pok_size_t **size**
- pok_bool_t **empty**
- pok_range_t **waiting_processes**
- pok_size_t **index**
- pok_bool_t **ready**
- pok_event_id_t **lock**

3.17.1 Detailed Description

Definition at line 27 of file blackboard.h.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/include/middleware/blackboard.h

3.18 pok_buffer_status_t Struct Reference

Public Attributes

- pok_range_t **nb_messages**
- pok_range_t **max_messages**
- pok_size_t **message_size**
- pok_range_t **waiting_processes**

3.18.1 Detailed Description

Definition at line 47 of file buffer.h.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/include/middleware/buffer.h

3.19 pok_buffer_t Struct Reference

Public Attributes

- pok_bool_t **ready**
- pok_bool_t **empty**
- pok_bool_t **full**
- pok_size_t **size**
- pok_size_t **index**
- pok_port_size_t **off_b**
- pok_port_size_t **off_e**
- pok_port_size_t **msgsize**
- pok_range_t **waiting_processes**
- pok_queueing_discipline_t **discipline**
- pok_event_id_t **lock**

3.19.1 Detailed Description

Definition at line 32 of file buffer.h.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/include/middleware/buffer.h

3.20 pok_error_report_t Struct Reference

Public Attributes

- uint32_t **thread**
- uint32_t **error**
- pok_time_t **when**

3.20.1 Detailed Description

Definition at line 37 of file error.h.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/include/core/error.h

3.21 pok_error_status_t Struct Reference

Public Attributes

- uint8_t **error_kind**
- uint32_t **failed_thread**
- uint32_t **failed_addr**
- char * **msg**
- uint32_t **msg_size**

3.21.1 Detailed Description

Definition at line 27 of file error.h.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/include/core/error.h

3.22 pok_lockobj_attr_t Struct Reference

Public Attributes

- 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**

3.22.1 Detailed Description

Definition at line 38 of file lockobj.h.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/include/core/lockobj.h

3.23 pok_lockobj_lockattr_t Struct Reference

Public Attributes

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

3.23.1 Detailed Description

Definition at line 62 of file lockobj.h.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/include/core/lockobj.h

3.24 pok_mutex_attr_t Struct Reference

Public Attributes

- pok_mutex_policy_t **policy**

3.24.1 Detailed Description

Definition at line 31 of file mutex.h.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/include/core/mutex.h

3.25 pok_port_queueing_status_t Struct Reference

Public Attributes

- pok_port_size_t **size**
- pok_port_direction_t **direction**
- uint8_t **nb_messages**
- uint8_t **waiting_processes**

3.25.1 Detailed Description

Definition at line 60 of file port.h.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/include/middleware/port.h

3.26 pok_port_sampling_status_t Struct Reference

Public Attributes

- pok_port_size_t **size**
- pok_port_direction_t **direction**
- uint64_t **refresh**
- bool_t **validity**

3.26.1 Detailed Description

Definition at line 107 of file port.h.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/include/middleware/port.h

3.27 pok_queue_t Struct Reference

Public Attributes

- char * **data**
- uint8_t **size**
- uint8_t **available_size**

3.27.1 Detailed Description

Definition at line 25 of file queue.h.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/include/middleware/queue.h

3.28 pok_syscall_args_t Struct Reference

Public Attributes

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

3.28.1 Detailed Description

Definition at line 92 of file syscall.h.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/include/core/syscall.h

3.29 pok_thread_attr_t Struct Reference

Public Attributes

- uint8_t **priority**
- void * **entry**
- uint64_t **period**
- uint64_t **deadline**
- uint64_t **time_capacity**
- uint32_t **stack_size**
- uint32_t **state**

3.29.1 Detailed Description

Definition at line 32 of file thread.h.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/include/core/thread.h

3.30 PROCESS_ATTRIBUTE_TYPE Struct Reference

Public Attributes

- SYSTEM_TIME_TYPE **PERIOD**
- SYSTEM_TIME_TYPE **TIME_CAPACITY**
- SYSTEM_ADDRESS_TYPE **ENTRY_POINT**
- STACK_SIZE_TYPE **STACK_SIZE**
- PRIORITY_TYPE **BASE_PRIORITY**
- DEADLINE_TYPE **DEADLINE**
- PROCESS_NAME_TYPE **NAME**

3.30.1 Detailed Description

Definition at line 55 of file process.h.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/include/arinc653/process.h

3.31 PROCESS_STATUS_TYPE Struct Reference

Public Attributes

- SYSTEM_TIME_TYPE **DEADLINE_TIME**
- PRIORITY_TYPE **CURRENT_PRIORITY**
- PROCESS_STATE_TYPE **PROCESS_STATE**
- PROCESS_ATTRIBUTE_TYPE **ATTRIBUTES**

3.31.1 Detailed Description

Definition at line 65 of file process.h.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/include/arinc653/process.h

3.32 QUEUING_PORT_STATUS_TYPE Struct Reference

Public Attributes

- MESSAGE_RANGE_TYPE **NB_MESSAGE**
- MESSAGE_RANGE_TYPE **MAX_NB_MESSAGE**
- MESSAGE_SIZE_TYPE **MAX_MESSAGE_SIZE**
- PORT_DIRECTION_TYPE **PORT_DIRECTION**
- WAITING_RANGE_TYPE **WAITING_PROCESSES**

3.32.1 Detailed Description

Definition at line 32 of file queueing.h.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/include/arinc653/queueing.h

3.33 s_file Struct Reference

Public Attributes

- char **buff** [MY_BUF_SIZE]
- size_t **pos**

3.33.1 Detailed Description

Definition at line 38 of file printf.c.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/libc/stdio/printf.c

3.34 s_format Struct Reference

Public Attributes

- char **ch**
- t_fmtfun **fun**
- int **flags**

3.34.1 Detailed Description

Definition at line 55 of file printf.c.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/libc/stdio/printf.c

3.35 s_ne2000_dev Struct Reference

Public Attributes

- s_pci_device **pci**
- unsigned int **addr**
- char **mac** [6]
- pok_queue_t **recv_buf** [20]

3.35.1 Detailed Description

Definition at line 66 of file rtl8029.h.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/drivers/rtl8029.h

3.36 s_pci_device Struct Reference

Public Attributes

- uint16_t **bus**
- uint16_t **dev**
- uint16_t **fun**
- uint16_t **vendorid**
- uint16_t **deviceid**
- uint16_t **irq_line**
- uint16_t **io_range**
- uint32_t **bar** [6]
- uint32_t **addr**
- void * **irq_handler**

3.36.1 Detailed Description

Definition at line 28 of file pci.h.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/include/arch/x86/pci.h

3.37 SAMPLING_PORT_STATUS_TYPE Struct Reference

Public Attributes

- SYSTEM_TIME_TYPE **REFRESH_PERIOD**
- MESSAGE_SIZE_TYPE **MAX_MESSAGE_SIZE**
- PORT_DIRECTION_TYPE **PORT_DIRECTION**
- VALIDITY_TYPE **LAST_MSG_VALIDITY**

3.37.1 Detailed Description

Definition at line 39 of file sampling.h.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/include/arinc653/sampling.h

3.38 SEMAPHORE_STATUS_TYPE Struct Reference

Public Attributes

- SEMAPHORE_VALUE_TYPE **CURRENT_VALUE**
- SEMAPHORE_VALUE_TYPE **MAXIMUM_VALUE**
- WAITING_RANGE_TYPE **WAITING_PROCESSES**

3.38.1 Detailed Description

Definition at line 64 of file semaphore.h.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/include/arinc653/semaphore.h

3.39 u_arg Union Reference

Public Attributes

- uint32_t **value**
- uint32_t **uint**
- int **sint**
- double **vdouble**
- void * **ptr**

3.39.1 Detailed Description

Definition at line 44 of file printf.c.

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

- /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/libc/stdio/printf.c

Chapter 4

File Documentation

4.1 /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/arinc653/semaphore.c File Reference

Provides ARINC653 API functionnalities for semaphore management.

Classes

- struct pok_arinc653_semaphore_layer_t

Functions

- void **CREATE_SEMAPHORE** (SEMAPHORE_NAME_TYPE SEMAPHORE_NAME, SEMAPHORE_VALUE_TYPE CURRENT_VALUE, SEMAPHORE_VALUE_TYPE MAXIMUM_VALUE, QUEUING_DISCIPLINE_TYPE QUEUING_DISCIPLINE, SEMAPHORE_ID_TYPE *SEMAPHORE_ID, RETURN_CODE_TYPE *RETURN_CODE)
- void **WAIT_SEMAPHORE** (SEMAPHORE_ID_TYPE SEMAPHORE_ID, SYSTEM_TIME_TYPE TIME_OUT, RETURN_CODE_TYPE *RETURN_CODE)
- void **SIGNAL_SEMAPHORE** (SEMAPHORE_ID_TYPE SEMAPHORE_ID, RETURN_CODE_TYPE *RETURN_CODE)
- void **GET_SEMAPHORE_ID** (SEMAPHORE_NAME_TYPE SEMAPHORE_NAME, SEMAPHORE_ID_TYPE *SEMAPHORE_ID, RETURN_CODE_TYPE *RETURN_CODE)
- void **GET_SEMAPHORE_STATUS** (SEMAPHORE_ID_TYPE SEMAPHORE_ID, SEMAPHORE_STATUS_TYPE *SEMAPHORE_STATUS, RETURN_CODE_TYPE *RETURN_CODE)

Variables

- pok_bool_t **pok_arinc653_semaphores_initialized** = 0
- char * **pok_arinc653_semaphores_names** [POK_CONFIG_ARINC653_NB_SEMAPHORES]
- pok_arinc653_semaphore_layer_t **pok_arinc653_semaphores_layers** [POK_CONFIG_ARINC653_NB_SEMAPHORES]

4.1.1 Detailed Description

Provides ARINC653 API functionnalities for semaphore management.

Definition in file semaphore.c.

4.2 /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/drivers/rtl8029.c File Reference

RTL8029 driver.

Functions

- void rtl8029_read (pok_port_id_t port_id, void *data, uint32_t len)

  Reads data from the corresponding network stack.
- void rtl8029_write (pok_port_id_t port_id, const void *data, uint32_t len)

  Send data to the interface.
- void rtl8029_polling ()

  Polls rtl8029 device.
- void rtl8029_init ()

  Initializes rtl8029 device.

4.2.1 Detailed Description

RTL8029 driver.

Author

Laurent

Date

PFE GISTR 2010

Definition in file rtl8029.c.

4.2.2 Function Documentation

4.2.2.1 void rtl8029_init ( )

Initializes rtl8029 device.

Seeks and registers PCI interface, set configuration and fills the dev structure.

Definition at line 382 of file rtl8029.c.

```
{
  dev.pci.vendorid = 0x10ec;
  dev.pci.deviceid = 0x8029;
  dev.pci.io_range = 0x10;

  if (pci_register(&(dev.pci)) != 0)
  {
    printf("rtl8029: PCI init failed!\n");
    return;
  }

  dev.addr = dev.pci.bar[0] & (~0x1F);

  unsigned char i = 0;
  unsigned char buf[6 * 2]; // used for MAC address

  NE2000_SELECT_PAGE(&dev, 0);

  /* This bit is the STOP command. When it is set, no packets will be
     received or transmitted. POWER UP=1. */
  outb(NE2000_CR_STP, dev.addr + NE2000_CR);
```

```
  // Sets several options... Read the datasheet!
  outb(0x00, dev.addr + NE2000_TCR);
  outb(NE2000_RCR_AB, dev.addr + NE2000_RCR);
  outb(NE2000_DCR_LS | NE2000_DCR_FT1, dev.addr + NE2000_DCR);

  /* The Page Start register sets the start page address
     of the receive buffer ring. */
  outb(NE2000_RXBUF, dev.addr + NE2000_PSTART);
  /* The Page Stop register sets the stop page address
     of the receive buffer ring. */
  outb(NE2000_MEMSZ, dev.addr + NE2000_PSTOP);
  /* This register is used to prevent overwrite of the receive buffer ring.
     It is typically used as a pointer indicating the last receive buffer
     page the host has read. */
  outb(NE2000_RXBUF, dev.addr + NE2000_BNRY);

  /* These two registers set the data byte counts of remote DMA. */
  outb(0, dev.addr + NE2000_RBCR0);
  outb(0, dev.addr + NE2000_RBCR1);

  NE2000_SELECT_PAGE(&dev, 1);

  /* This register points to the page address of the first receive buffer
     page to be used for a packet reception. */
  outb(NE2000_RXBUF + 1, dev.addr + NE2000_CURR);

  // Init mac address
  /* Here's something I do not understand... Section 6.2.2 of the datasheet
     says bytes 00H-05H of the PROM corresponds to the Ethernet ID. But it
     looks like each byte of the MAC address is written twice...
     Therefore I read 2 * sizeof(mac) and select one of the two bytes
     corresponding to the MAC... Weird... Really... */
  ne2000_read(&dev, buf, 6 * 2, 0);
  for (i = 0; i < 6; i++)
    dev.mac[i] = buf[i * 2];

  /* These registers contain my Ethernet node address and are used to compare
    the destination address of incoming packets for acceptation or rejection.*/
  outb(dev.mac[0], dev.addr + NE2000_PAR0);
  outb(dev.mac[1], dev.addr + NE2000_PAR1);
  outb(dev.mac[2], dev.addr + NE2000_PAR2);
  outb(dev.mac[3], dev.addr + NE2000_PAR3);
  outb(dev.mac[4], dev.addr + NE2000_PAR4);
  outb(dev.mac[5], dev.addr + NE2000_PAR5);

  NE2000_SELECT_PAGE(&dev, 0);

  // Start command
  outb(NE2000_CR_STA, dev.addr + NE2000_CR);

  // Reactivating interrupts
  /* ISR register must be cleared after power up. */
  outb(0xFF, dev.addr + NE2000_ISR);
  /* All bits correspond to the bits in the ISR register. POWER UP=all 0s.
     Setting individual bits will enable the corresponding interrupts. */
  /* Since POK use polling, ALL interrupts are disabled */
  outb(0x00, dev.addr + NE2000_IMR);

  for (i = 0; i < 20; i++) /* TODO: random constant */
  {
    dev.recv_buf[i].len = 0;
    dev.recv_buf[i].off = 0;
  }

  return;
}
```

4.2.2.2 void rtl8029_polling ( )

Polls rtl8029 device.

Watches for events, typically for receiving queued packets.

Definition at line 279 of file rtl8029.c.

```
{
  unsigned char state; // ISR state

  NE2000_SELECT_PAGE(&dev, 0);

  while (1)
  {
```

```
// do we have an interrupt flag set?
if ((state = pok_inb(dev.addr + NE2000_ISR)) == 0)
  continue;

if (state & NE2000_ISR_PRX)
{
  if ((pok_inb(dev.addr + NE2000_RSR) & NE2000_RSR_PRX) == 0)
  {
    // error
  }

  printf("[*]\n");

  /* no errors */
  s_ne2000_header   ne2000_hdr;     // ne2000 packet header
  unsigned short    offset;         // dma offset
  unsigned char     start, end;     // pointers for the ring buffer
  pok_packet_t      recv_packet;

  while (1)
  {

    /* This register is used to prevent overwrite of the receive buffer
   ring.
       It is typically used as a pointer indicating the last receive buffer
       page the host has read.*/
    start = pok_inb(dev.addr + NE2000_BNRY) + 1;

    /* This register points to the page address of the first receive
       buffer page to be used for a packet reception. */
    NE2000_SELECT_PAGE(&dev, 1);
    end = pok_inb(dev.addr + NE2000_CURR);
    NE2000_SELECT_PAGE(&dev, 0);

    if ((end % NE2000_MEMSZ) == (start % NE2000_MEMSZ) + 1)
    {
      break;
    }

    /* et on decapsule! */
    offset = start << 8;
    // ne2000 header
    offset += ne2000_read(&dev, &ne2000_hdr, sizeof(s_ne2000_header),
                          offset);

    ne2000_read(&dev, &recv_packet,
                ne2000_hdr.size - sizeof(s_ne2000_header), offset);
    rtl8029_enqueue(&recv_packet);

    // update the BNRY register... almost forgot that
    outb(ne2000_hdr.next > NE2000_MEMSZ ?
         NE2000_RXBUF - 1 : ne2000_hdr.next - 1, dev.addr + NE2000_BNRY);

  }

  outb(NE2000_ISR_PRX, dev.addr + NE2000_ISR); // Clear PRX flag
}

if (state & NE2000_ISR_PTX)
{
  outb(NE2000_ISR_PTX, dev.addr + NE2000_ISR); // Clear PTX flag
}

if (state & NE2000_ISR_RXE)
{
  outb(NE2000_ISR_RXE, dev.addr + NE2000_ISR); // Clear RXE flag
}

if (state & NE2000_ISR_TXE)
{
  outb(NE2000_ISR_TXE, dev.addr + NE2000_ISR); // Clear TXE flag
}

if (state & NE2000_ISR_OVW)
{
  outb(NE2000_ISR_OVW, dev.addr + NE2000_ISR); // Clear OVW flag
}

if (state & NE2000_ISR_CNT)
{
  outb(NE2000_ISR_CNT, dev.addr + NE2000_ISR); // Clear CNT flag
}

if (state & NE2000_ISR_RST)
{
  outb(NE2000_ISR_RST, dev.addr + NE2000_ISR); // Clear RST bit
}
```

```
  }
}
```

4.2.2.3 void rtl8029_read (pok_port_id_t *port_id*, void * *data*, uint32_t *len*)

Reads data from the corresponding network stack.

Reads enqueued data in the stack partition.

Definition at line 146 of file rtl8029.c.

```
{
  pok_port_id_t global;
  pok_ret_t     ret;

  ret = pok_port_virtual_get_global (port_id, &global);

  if (ret == POK_ERRNO_OK)
  {
    char        *dest = data;
    pok_queue_t* queue = dev.recv_buf + global;
    uint32_t    size = len < queue->len ? len : queue->len;
    uint32_t    copied = 0;

    printf ("[RTL8029] READ DATA FROM LOCAL PORT %d "
            "GLOBAL_PORT=%d), size=%d\n", port_id, global, len);

    /* is there something to read ? */
    if (queue->len == 0)
    {
      printf("rtl8029_read: error: empty read ring buffer %d!\n", port_id);
      return;
    }

    /* copy from the queue to the buffer */
    for (copied = 0; copied < size; copied++)
    {
      dest[copied % RECV_BUF_SZ] = queue->data[queue->off];
      queue->off = (queue->off + 1) % RECV_BUF_SZ;
    }

    /* updating data length in this queue */
    queue->len -= size;
  }
}
```

4.2.2.4 void rtl8029_write (pok_port_id_t *port_id*, const void * *data*, uint32_t *len*)

Send data to the interface.

Writes data to be sent to network.

Definition at line 187 of file rtl8029.c.

```
{
  uint32_t        nbdest;
  uint32_t        tmp;
  uint32_t        dest;
  pok_ret_t       ret;
  char            node2[6] = { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF };
  pok_packet_t    packet;
  const char*     d;
  size_t          cpylen = 0;
  size_t          sndlen = 0;
  unsigned char state; // ISR state

  ret = pok_port_virtual_nb_destinations (port_id, &nbdest);
  if (ret != POK_ERRNO_OK)
  {
    return;
  }

  for (tmp = 0 ; tmp < nbdest ; tmp++)
  {
    ret = pok_port_virtual_destination (port_id, tmp, &dest);
    if (ret == POK_ERRNO_OK)
```

```
    {
      printf ("[RTL8029] SEND DATA THROUGH NETWORK FROM LOCAL PORT %d "
              "TO GLOBAL PORT %d, size=%d\n", port_id, dest, len);

      memcpy(packet.eth.src, dev.mac, ETH_MAC_LEN);
      memcpy(packet.eth.dst, node2, ETH_MAC_LEN);
      packet.eth.ethertype = 0x4242;
      packet.udp.src = port_id;
      packet.udp.dst = dest;

      for (d = data; len != 0; len -= cpylen, data += cpylen)
      {
        // too short; let's cut
        if (len <= NET_DATA_MINLEN)
        {
          cpylen = len;
          sndlen = ETH_DATA_MINLEN + sizeof(eth_hdr_t);
        }
        else
        {
          // too big; let's pad
          if (len >= NET_DATA_MAXLEN)
          {
            cpylen = NET_DATA_MAXLEN;
            sndlen = ETH_DATA_MAXLEN + sizeof(eth_hdr_t);
          }
          // normal
          else
          {
            cpylen = len;
            sndlen = sizeof(eth_hdr_t) + sizeof(udp_hdr_t) + cpylen;
          }
        }

        packet.udp.len = cpylen;
        memcpy(&(packet.data), data, cpylen);

        ne2000_write(&dev, &packet, sndlen, NE2000_TXBUF * 256);

        do
        {
          state = pok_inb(dev.addr + NE2000_ISR);
        }
        while ((state & NE2000_ISR_RDC) != NE2000_ISR_RDC);

        /* This register sets the start page address of
           the packet to the transmitted. */
        outb(NE2000_TXBUF, dev.addr + NE2000_TPSR); //?

        /* These two registers set the byte counts of
           the packet to be transmitted. */
        outb(sndlen, dev.addr + NE2000_TBCR0);
        outb(sndlen >> 8, dev.addr + NE2000_TBCR1);

        /* This bit must be set to transmit a packet. */
        outb(pok_inb(dev.addr + NE2000_CR) | NE2000_CR_TXP,
             dev.addr + NE2000_CR);

        outb(NE2000_ISR_RDC, dev.addr + NE2000_ISR); // Clear RDC bit
      }
    }
  }
}
```

4.3 /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/include/protocols/ceasar.h File Reference

Ceasar crypto protocol.

Functions

- void pok_protocols_ceasar_unmarshall (void *crypted_data, pok_size_t crypted_size, void *uncrypted_data, size_t *uncrypted_size)
- void pok_protocols_ceasar_marshall (void *uncrypted_data, pok_size_t uncrypted_size, void *crypted_data, size_t *crypted_size)

4.3.1 Detailed Description

Ceasar crypto protocol.

Author

Julien Delange

Date

2009 This is a very basic crypto protocol that just change the order of bytes in data. There is no public/private key, the algorithm is known by the attacker so that it's a very weak crypto protocol. Interested people can gather more information about this protocol on: http://en.wikipedia.org/wiki/Caesar_cipher

We don't provide an associated marshalling type for the Ceasar protocol since the crypted size is the same than the uncrypted size.

Definition in file ceasar.h.

4.3.2 Function Documentation

4.3.2.1 void pok_protocols_ceasar_marshall (void * *uncrypted_data*, pok_size_t *uncrypted_size*, void * *crypted_data*, size_t * *crypted_size*)

Function that encrypts data

\file libpok/protocols/ceasar.c \brief Function to crypt/uncrypt data using the Ceasar cipher. \author Julien Delange \brief Marshall data, the crypted size has the same size than uncrypted data.

Definition at line 34 of file ceasar.c.

```
{
   uint8_t* uncrypted;
   uint8_t* crypted;
   size_t   tmp;

   uncrypted   = (uint8_t*) uncrypted_data;
   crypted     = (uint8_t*) crypted_data;

   for (tmp = 0 ; tmp < uncrypted_size ; tmp++)
   {
         crypted[tmp] = (uncrypted[tmp] + 4) % 255;
   }

   *crypted_size = uncrypted_size;
}
```

4.3.2.2 void pok_protocols_ceasar_unmarshall (void * *crypted_data*, pok_size_t *crypted_size*, void * *uncrypted_data*, size_t * *uncrypted_size*)

Function that uncrypts data

\brief Unmarshall data, the crypted size has the same size than uncrypted data.

Definition at line 56 of file ceasar.c.

```
{
   uint8_t* uncrypted;
   uint8_t* crypted;
   size_t   tmp;

   uncrypted   = (uint8_t*) uncrypted_data;
   crypted     = (uint8_t*) crypted_data;

   for (tmp = 0 ; tmp < crypted_size ; tmp++)
   {
```

```
        uncrypted[tmp] = (crypted[tmp] - 4) % 255;
   }

   *uncrypted_size = crypted_size;
}
```

4.4 /home/rosen/projets/parsec/pok_orig/pok/trunk/libpok/protocols/des/des.c File Reference

DES crypto protocol.

Functions

- void **pok_protocols_des_init** ()
- void pok_protocols_des_marshall (void *uncrypted_data, pok_size_t uncrypted_size, void *crypted_data, size_t *crypted_size)
- void pok_protocols_des_unmarshall (void *crypted_data, pok_size_t crypted_size, void *uncrypted_data, size_t *uncrypted_size)

Variables

- unsigned char **initVector** [8] = POK_PROTOCOLS_DES_INIT
- int **pok_protocols_des_is_init** = 0

4.4.1 Detailed Description

DES crypto protocol.

Author

Julien Delange

Date

2009 This file is a wrapper that interfaces with OpenSSL functions. It sets the crypto key, initialisation string and calls OpenSSL function to crypts data.

Definition in file des.c.

4.4.2 Function Documentation

4.4.2.1 void pok_protocols_des_marshall (void * *uncrypted_data*, pok_size_t *uncrypted_size*, void * *crypted_data*, size_t * *crypted_size*)

Function that crypts data.

Definition at line 54 of file des.c.

```
{
   DES_cblock ivec;
   DES_key_schedule schedule;

   DES_set_key_checked (&cbc_key, &schedule);

   memcpy(ivec,initVector,sizeof(initVector));

   DES_ncbc_encrypt(uncrypted_data, crypted_data, uncrypted_size, &schedule, &
      ivec, DES_ENCRYPT);
   *crypted_size = 8;
}
```

4.4.2.2 void pok_protocols_des_unmarshall (void * *crypted_data*, pok_size_t *crypted_size*, void * *uncrypted_data*, size_t * *uncrypted_size*)

Function that uncrypts data.

Definition at line 68 of file des.c.

```
{
   DES_cblock ivec;
   DES_key_schedule schedule;

   DES_set_key_checked (&cbc_key, &schedule);

   memcpy(ivec,initVector,sizeof(initVector));
   DES_ncbc_encrypt(crypted_data, uncrypted_data, crypted_size, &schedule, &
      ivec, DES_DECRYPT);

   *uncrypted_size = 8;
}
```

Index