Linux Pcmcia Documentation

The kernel development community

Jun 10, 2024

CONTENTS

CHAPTER

ONE

PCMCIA DRIVER

1.1 sysfs

New PCMCIA IDs may be added to a device driver pcmcia_device_id table at runtime as shown below:

```
echo "match_flags manf_id card_id func_id function device_no \
prod_id_hash[0] prod_id_hash[1] prod_id_hash[2] prod_id_hash[3]" > \
/sys/bus/pcmcia/drivers/{driver}/new_id
```

All fields are passed in as hexadecimal values (no leading 0x). The meaning is described in the PCMCIA specification, the match_flags is a bitwise or-ed combination from PCMCIA_DEV_ID_MATCH_* constants defined in include/linux/mod_devicetable.h.

Once added, the driver probe routine will be invoked for any unclaimed PCMCIA device listed in its (newly updated) pcmcia_device_id list.

A common use-case is to add a new device according to the manufacturer ID and the card ID (form the manf_id and card_id file in the device tree). For this, just use:

```
echo "0x3 manf_id card_id 0 0 0 0 0 0 0" > \
  /sys/bus/pcmcia/drivers/{driver}/new_id
```

after loading the driver.

CHAPTER

TWO

DEVICE TABLE

Matching of PCMCIA devices to drivers is done using one or more of the following criteria:

- manufactor ID
- card ID
- product ID strings _and_ hashes of these strings
- function ID
- device function (actual and pseudo)

You should use the helpers in include/pcmcia/device_id.h for generating the struct pcmcia_device_id[] entries which match devices to drivers.

If you want to match product ID strings, you also need to pass the crc32 hashes of the string to the macro, e.g. if you want to match the product ID string 1, you need to use

PCMCIA_DEVICE_PROD_ID1(“some_string”, 0x(hash_of_some_string)),

If the hash is incorrect, the kernel will inform you about this in “dmesg” upon module initialization, and tell you of the correct hash.

You can determine the hash of the product ID strings by catting the file “modalias” in the sysfs directory of the PCMCIA device. It generates a string in the following form: pcmcia:m0149cC1ABf06pfn00fn00pa725B842DpbF1EFEE84pc0877B627pd00000000

The hex value after “pa” is the hash of product ID string 1, after “pb” for string 2 and so on.

Alternatively, you can use crc32hash (see tools/pcmcia/crc32hash.c) to determine the crc32 hash. Simply pass the string you want to evaluate as argument to this program, e.g.: $ tools/pcmcia/crc32hash “Dual Speed”

CHAPTER

THREE

LOCKING

This file explains the locking and exclusion scheme used in the PCCARD and PCMCIA subsystems.

3.1 A) Overview, Locking Hierarchy:

pcmcia_socket_list_rwsem

  - protects only the list of sockets

- **skt_mutex**
  - serializes card insert / ejection
  - **ops_mutex**
    - serializes socket operation

3.2 B) Exclusion

The following functions and callbacks to struct pcmcia_socket must be called with "skt_mutex" held:

```
socket_detect_change()
send_event()
socket_reset()
socket_shutdown()
socket_setup()
socket_remove()
socket_insert()
socket_early_resume()
socket_late_resume()
socket_resume()
socket_suspend()

struct pcmcia_callback  *callback
```

The following functions and callbacks to struct pcmcia_socket must be called with "ops_mutex" held:

```
socket_reset()
socket_setup()

struct pccard_operations        *ops
struct pccard_resource_ops      *resource_ops;
```

Note that send_event() and *struct pcmcia_callback *callback* must not be called with "ops_mutex" held.

3.3 C) Protection

3.3.1 1. Global Data:

struct list_head pcmcia_socket_list;

protected by pcmcia_socket_list_rwsem;

3.3.2 2. Per-Socket Data:

The resource_ops and their data are protected by ops_mutex.

The "main" struct pcmcia_socket is protected as follows (read-only fields or single-use fields not mentioned):

- by pcmcia_socket_list_rwsem:

  ```
  struct list_head        socket_list;
  ```

- by thread_lock:

  ```
  unsigned int            thread_events;
  ```

- by skt_mutex:

  ```
  u_int                   suspended_state;
  void                    (*tune_bridge);
  struct pcmcia_callback  *callback;
  int                     resume_status;
  ```

- by ops_mutex:

  ```
  socket_state_t          socket;
  u_int                   state;
  u_short                 lock_count;
  pccard_mem_map          cis_mem;
  void __iomem            *cis_virt;
  struct { }              irq;
  io_window_t             io[];
  pccard_mem_map          win[];
  struct list_head        cis_cache;
  size_t                  fake_cis_len;
  ```

```
u8                      *fake_cis;
u_int                   irq_mask;
void                    (*zoom_video);
int                     (*power_hook);
u8                      resource...;
struct list_head        devices_list;
u8                      device_count;
struct                  pcmcia_state;
```

3.3.3 3. Per PCMCIA-device Data:

The “main” struct pcmcia_device is protected as follows (read-only fields or single-use fields not mentioned):

- by pcmcia_socket->ops_mutex:

```
struct list_head        socket_device_list;
struct config_t         *function_config;
u16                     _irq:1;
u16                     _io:1;
u16                     _win:4;
u16                     _locked:1;
u16                     allow_func_id_match:1;
u16                     suspended:1;
u16                     _removed:1;
```

- by the PCMCIA driver:

```
io_req_t                io;
irq_req_t               irq;
config_req_t            conf;
window_handle_t         win;
```

CHAPTER

FOUR

DRIVER CHANGES

This file details changes in 2.6 which affect PCMCIA card driver authors:

- **pcmcia_loop_config() and autoconfiguration (as of 2.6.36)**
  If *struct pcmcia_device *p_dev->config_flags* is set accordingly, pcmcia_loop_config() now sets up certain configuration values automatically, though the driver may still override the settings in the callback function. The following autoconfiguration options are provided at the moment:
  - CONF_AUTO_CHECK_VCC : check for matching Vcc
  - CONF_AUTO_SET_VPP : set Vpp
  - CONF_AUTO_AUDIO : auto-enable audio line, if required
  - CONF_AUTO_SET_IO : set ioport resources (->resource[0,1])
  - CONF_AUTO_SET_IOMEM : set first iomem resource (->resource[2])
- **pcmcia_request_configuration -> pcmcia_enable_device (as of 2.6.36)**
  pcmcia_request_configuration() got renamed to pcmcia_enable_device(), as it mirrors pcmcia_disable_device(). Configuration settings are now stored in struct pcmcia_device, e.g. in the fields config_flags, config_index, config_base, vpp.
- **pcmcia_request_window changes (as of 2.6.36)**
  Instead of win_req_t, drivers are now requested to fill out *struct pcmcia_device *p_dev->resource[2,3,4,5]* for up to four ioport ranges. After a call to pcmcia_request_window(), the regions found there are reserved and may be used immediately -- until pcmcia_release_window() is called.
- **pcmcia_request_io changes (as of 2.6.36)**
  Instead of io_req_t, drivers are now requested to fill out *struct pcmcia_device *p_dev->resource[0,1]* for up to two ioport ranges. After a call to pcmcia_request_io(), the ports found there are reserved, after calling pcmcia_request_configuration(), they may be used.
- **No dev_info_t, no cs_types.h (as of 2.6.36)**
  dev_info_t and a few other typedefs are removed. No longer use them in PCMCIA device drivers. Also, do not include pcmcia/cs_types.h, as this file is gone.
- **No dev_node_t (as of 2.6.35)**
  There is no more need to fill out a “dev_node_t” structure.
- **New IRQ request rules (as of 2.6.35)**
  Instead of the old pcmcia_request_irq() interface, drivers may now choose between:
  - calling request_irq/free_irq directly. Use the IRQ from **p_dev->irq*.

  - use pcmcia_request_irq(p_dev, handler_t); the PCMCIA core will clean up automatically on calls to pcmcia_disable_device() or device ejection.

- **no cs_error / CS_CHECK / CONFIG_PCMCIA_DEBUG (as of 2.6.33)**
  Instead of the cs_error() callback or the CS_CHECK() macro, please use Linux-style checking of return values, and -- if necessary -- debug messages using “dev_dbg()” or “pr_debug()”.

- **New CIS tuple access (as of 2.6.33)**
  Instead of pcmcia_get_{first,next}_tuple(), pcmcia_get_tuple_data() and pcmcia_parse_tuple(), a driver shall use “pcmcia_get_tuple()” if it is only interested in one (raw) tuple, or “pcmcia_loop_tuple()” if it is interested in all tuples of one type. To decode the MAC from CISTPL_FUNCE, a new helper “pcmcia_get_mac_from_cis()” was added.

- **New configuration loop helper (as of 2.6.28)**
  By calling pcmcia_loop_config(), a driver can iterate over all available configuration options. During a driver’s probe() phase, one doesn’t need to use pcmcia_get_{first,next}_tuple, pcmcia_get_tuple_data and pcmcia_parse_tuple directly in most if not all cases.

- **New release helper (as of 2.6.17)**
  Instead of calling pcmcia_release_{configuration,io,irq,win}, all that’s necessary now is calling pcmcia_disable_device. As there is no valid reason left to call pcmcia_release_io and pcmcia_release_irq, the exports for them were removed.

- Unify detach and REMOVAL event code, as well as attach and INSERTION code (as of 2.6.16):

  ```
  void (*remove)          (struct pcmcia_device *dev);
  int (*probe)            (struct pcmcia_device *dev);
  ```

- Move suspend, resume and reset out of event handler (as of 2.6.16):

  ```
  int (*suspend)          (struct pcmcia_device *dev);
  int (*resume)           (struct pcmcia_device *dev);
  ```

  should be initialized in struct pcmcia_driver, and handle (SUSPEND == RESET_PHYSICAL) and (RESUME == CARD_RESET) events

- **event handler initialization in struct pcmcia_driver (as of 2.6.13)**
  The event handler is notified of all events, and must be initialized as the event() callback in the driver’s struct pcmcia_driver.

- **pcmcia/version.h should not be used (as of 2.6.13)**
  This file will be removed eventually.

- **in-kernel device<->driver matching (as of 2.6.13)**
  PCMCIA devices and their correct drivers can now be matched in kernelspace. See ‘*Device table*’ for details.

- **Device model integration (as of 2.6.11)**
  A struct pcmcia_device is registered with the device model core, and can be used (e.g. for SET_NETDEV_DEV) by using handle_to_dev(client_handle_t * handle).

- **Convert internal I/O port addresses to unsigned int (as of 2.6.11)**
  ioaddr_t should be replaced by unsigned int in PCMCIA card drivers.

- **irq_mask and irq_list parameters (as of 2.6.11)**
  The irq_mask and irq_list parameters should no longer be used in PCMCIA card drivers. Instead, it is the job of the PCMCIA core to determine which IRQ should be used. Therefore, link->irq.IRQInfo2 is ignored.

- **client->PendingEvents is gone (as of 2.6.11)**
  client->PendingEvents is no longer available.

- **client->Attributes are gone (as of 2.6.11)**
  client->Attributes is unused, therefore it is removed from all PCMCIA card drivers

- **core functions no longer available (as of 2.6.11)**
  The following functions have been removed from the kernel source because they are unused by all in-kernel drivers, and no external driver was reported to rely on them:

  ```
  pcmcia_get_first_region()
  pcmcia_get_next_region()
  pcmcia_modify_window()
  pcmcia_set_event_mask()
  pcmcia_get_first_window()
  pcmcia_get_next_window()
  ```

- **device list iteration upon module removal (as of 2.6.10)**
  It is no longer necessary to iterate on the driver's internal client list and call the ->detach() function upon module removal.

- **Resource management. (as of 2.6.8)**
  Although the PCMCIA subsystem will allocate resources for cards, it no longer marks these resources busy. This means that driver authors are now responsible for claiming your resources as per other drivers in Linux. You should use request_region() to mark your IO regions in-use, and request_mem_region() to mark your memory regions in-use. The name argument should be a pointer to your driver name. Eg, for pcnet_cs, name should point to the string "pcnet_cs".

- CardServices is gone CardServices() in 2.4 is just a big switch statement to call various services. In 2.6, all of those entry points are exported and called directly (except for pcmcia_report_error(), just use cs_error() instead).

- struct pcmcia_driver You need to use struct pcmcia_driver and pcmcia_{un,}register_driver instead of {un,}register_pccard_driver