Writing s390 channel device drivers

Cornelia Huck

cornelia.huck@de.ibm.com

Writing s390 channel device drivers
by Cornelia Huck

Table of Contents

Chapter 1. Introduction

This document describes the interfaces available for device drivers that drive s390 based channel attached I/O devices. This includes interfaces for interaction with the hardware and interfaces for interacting with the common driver core. Those interfaces are provided by the s390 common I/O layer.

The document assumes a familarity with the technical terms associated with the s390 channel I/O architecture. For a description of this architecture, please refer to the "z/Architecture: Principles of Operation", IBM publication no. SA22-7832.

While most I/O devices on a s390 system are typically driven through the channel I/O mechanism described here, there are various other methods (like the diag interface). These are out of the scope of this document.

Some additional information can also be found in the kernel source under Documentation/s390/driver-model.txt.

Chapter 2. The ccw bus

The ccw bus typically contains the majority of devices available to a s390 system. Named after the channel command word (ccw), the basic command structure used to address its devices, the ccw bus contains so-called channel attached devices. They are addressed via I/O subchannels, visible on the css bus. A device driver for channel-attached devices, however, will never interact with the subchannel directly, but only via the I/O device on the ccw bus, the ccw device.

2.1. I/O functions for channel-attached devices

Some hardware structures have been translated into C structures for use by the common I/O layer and device drivers. For more information on the hardware structures represented here, please consult the Principles of Operation.

struct scsw

LINUX

Kernel Hackers ManualFebruary 2010

Name

`struct scsw` — subchannel status word

Synopsis

```
struct scsw {
  __u32 key:4;
  __u32 sctl:1;
  __u32 eswf:1;
  __u32 cc:2;
  __u32 fmt:1;
  __u32 pfch:1;
  __u32 isic:1;
  __u32 alcc:1;
  __u32 ssi:1;
  __u32 zcc:1;
```

```
  __u32 ectl:1;
  __u32 pno:1;
  __u32 res:1;
  __u32 fctl:3;
  __u32 actl:7;
  __u32 stctl:5;
  __u32 cpa;
  __u32 dstat:8;
  __u32 cstat:8;
  __u32 count:16;
};
```

Members

key

subchannel key

sctl

suspend control

eswf

esw format

cc

deferred condition code

fmt

format

pfch

prefetch

isic

initial-status interruption control

alcc

address-limit checking control

ssi

suppress-suspended interruption

zcc

zero condition code

ectl

extended control

pno

path not operational

res

reserved

fctl

function control

actl

activity control

stctl

status control

cpa

channel program address

dstat

device status

cstat

subchannel status

count

residual count

struct ccw1

LINUX

Name

`struct ccw1` — channel command word

Synopsis

```
struct ccw1 {
  __u8 cmd_code;
  __u8 flags;
  __u16 count;
  __u32 cda;
};
```

Members

cmd_code

command code

flags

flags, like IDA adressing, etc.

count

byte count

cda

data address

Description

The ccw is the basic structure to build channel programs that perform operations with the device or the control unit. Only Format-1 channel command words are supported.

struct erw

LINUX

Kernel Hackers ManualFebruary 2010

Name

`struct erw` — extended report word

Synopsis

```
struct erw {
  __u32 res0:3;
  __u32 auth:1;
  __u32 pvrf:1;
  __u32 cpt:1;
  __u32 fsavf:1;
  __u32 cons:1;
  __u32 scavf:1;
  __u32 fsaf:1;
  __u32 scnt:6;
  __u32 res16:16;
};
```

Members

res0

reserved

auth

authorization check

pvrf

path-verification-required flag

cpt

channel-path timeout

fsavf

failing storage address validity flag

cons

concurrent sense

scavf

secondary ccw address validity flag

fsaf

failing storage address format

scnt

sense count, if `cons` == `1`

res16

reserved

struct sublog

LINUX

Kernel Hackers ManualFebruary 2010

Name

`struct sublog` — subchannel logout area

Synopsis

```
struct sublog {
  __u32 res0:1;
  __u32 esf:7;
  __u32 lpum:8;
  __u32 arep:1;
  __u32 fvf:5;
  __u32 sacc:2;
  __u32 termc:2;
```

```
  __u32 devsc:1;
  __u32 serr:1;
  __u32 ioerr:1;
  __u32 seqc:3;
};
```

Members

res0

reserved

esf

extended status flags

lpum

last path used mask

arep

ancillary report

fvf

field-validity flags

sacc

storage access code

termc

termination code

devsc

device-status check

serr

secondary error

ioerr

i/o-error alert

seqc

sequence code

struct esw0

LINUX

Kernel Hackers ManualFebruary 2010

Name

`struct esw0` — Format 0 Extended Status Word (ESW)

Synopsis

```
struct esw0 {
  struct sublog sublog;
  struct erw erw;
  __u32 faddr[2];
  __u32 saddr;
};
```

Members

sublog

subchannel logout

erw

extended report word

faddr[2]

failing storage address

saddr

secondary ccw address

struct esw1

LINUX

Kernel Hackers ManualFebruary 2010

Name

`struct esw1` — Format 1 Extended Status Word (ESW)

Synopsis

```
struct esw1 {
  __u8 zero0;
  __u8 lpum;
  __u16 zero16;
  struct erw erw;
  __u32 zeros[3];
};
```

Members

zero0

reserved zeros

lpum

last path used mask

zero16

reserved zeros

erw

extended report word

zeros[3]

three fullwords of zeros

struct esw2

LINUX

Kernel Hackers ManualFebruary 2010

Name

`struct esw2` — Format 2 Extended Status Word (ESW)

Synopsis

```
struct esw2 {
  __u8 zero0;
  __u8 lpum;
  __u16 dcti;
  struct erw erw;
  __u32 zeros[3];
};
```

Members

zero0

reserved zeros

lpum

last path used mask

dcti

device-connect-time interval

erw

extended report word

zeros[3]

three fullwords of zeros

struct esw3

LINUX

Kernel Hackers ManualFebruary 2010

Name

`struct esw3` — Format 3 Extended Status Word (ESW)

Synopsis

```
struct esw3 {
  __u8 zero0;
  __u8 lpum;
  __u16 res;
  struct erw erw;
  __u32 zeros[3];
};
```

Members

zero0

reserved zeros

lpum

last path used mask

res

reserved

erw

extended report word

zeros[3]

three fullwords of zeros

struct irb

LINUX

Kernel Hackers ManualFebruary 2010

Name

`struct irb` — interruption response block

Synopsis

```
struct irb {
  struct scsw scsw;
  union esw;
  __u8 ecw[32];
};
```

Members

scsw

subchannel status word

esw

extened status word, 4 formats

ecw[32]

extended control word

Description

The irb that is handed to the device driver when an interrupt occurs. For solicited interrupts, the common I/O layer already performs checks whether a field is valid; a field not being valid is always passed as `0`. If a unit check occured, *`ecw`* may contain sense data; this is retrieved by the common I/O layer itself if the device doesn't support concurrent sense (so that the device driver never needs to perform basic sene itself). For unsolicited interrupts, the irb is passed as-is (expect for sense data, if applicable).

struct ciw

LINUX

Kernel Hackers ManualFebruary 2010

Name

`struct ciw` — command information word (CIW) layout

Synopsis

```
struct ciw {
  __u32 et:2;
  __u32 reserved:2;
  __u32 ct:4;
  __u32 cmd:8;
  __u32 count:16;
};
```

Members

et

entry type

reserved

reserved bits

ct

command type

cmd

command code

count

command count

struct ccw_dev_id

LINUX

Kernel Hackers ManualFebruary 2010

Name

`struct ccw_dev_id` — unique identifier for ccw devices

Synopsis

```
struct ccw_dev_id {
  u8 ssid;
  u16 devno;
};
```

Members

ssid

subchannel set id

devno

device number

Description

This structure is not directly based on any hardware structure. The hardware identifies a device by its device number and its subchannel, which is in turn identified by its id. In order to get a unique identifier for ccw devices across subchannel sets, *struct* ccw_dev_id has been introduced.

ccw_dev_id_is_equal

LINUX

Kernel Hackers ManualFebruary 2010

Name

`ccw_dev_id_is_equal` — compare two ccw_dev_ids

Synopsis

```
int ccw_dev_id_is_equal (struct ccw_dev_id * dev_id1, struct
ccw_dev_id * dev_id2);
```

Arguments

dev_id1

a ccw_dev_id

dev_id2

another ccw_dev_id

Returns

`1` if the two structures are equal field-by-field, `0` if not.

Context

any

2.2. ccw devices

Devices that want to initiate channel I/O need to attach to the ccw bus. Interaction with the driver core is done via the common I/O layer, which provides the abstractions of ccw devices and ccw device drivers.

The functions that initiate or terminate channel I/O all act upon a ccw device structure. Device drivers must not bypass those functions or strange side effects may happen.

struct ccw_device

LINUX

Kernel Hackers ManualFebruary 2010

Name

`struct ccw_device` — channel attached device

Synopsis

```
struct ccw_device {
  spinlock_t * ccwlock;
  struct ccw_device_id id;
  struct ccw_driver * drv;
  struct device dev;
  int online;
  void (* handler) (struct ccw_device *, unsigned long, struct irb *);
};
```

Members

ccwlock

pointer to device lock

id

id of this device

drv

ccw driver for this device

dev

embedded device structure

online

online status of device

handler

interrupt handler

Description

`handler` is a member of the device rather than the driver since a driver can have different interrupt handlers for different ccw devices (multi-subchannel drivers).

struct ccw_driver

LINUX

Kernel Hackers ManualFebruary 2010

Name

`struct ccw_driver` — device driver for channel attached devices

Synopsis

```
struct ccw_driver {
  struct module * owner;
  struct ccw_device_id * ids;
  int (* probe) (struct ccw_device *);
  void (* remove) (struct ccw_device *);
  int (* set_online) (struct ccw_device *);
  int (* set_offline) (struct ccw_device *);
  int (* notify) (struct ccw_device *, int);
  void (* shutdown) (struct ccw_device *);
  struct device_driver driver;
  char * name;
};
```

Members

owner

owning module

ids

ids supported by this driver

probe

function called on probe

remove

function called on remove

set_online

called when setting device online

set_offline

called when setting device offline

notify

notify driver of device state changes

shutdown

called at device shutdown

driver

embedded device driver structure

name

device driver name

ccw_device_set_offline

LINUX

Kernel Hackers ManualFebruary 2010

Name

`ccw_device_set_offline` — disable a ccw device for I/O

Synopsis

```
int ccw_device_set_offline (struct ccw_device * cdev);
```

Arguments

cdev

target ccw device

Description

This function calls the driver's `set_offline` function for *cdev*, if given, and then disables *cdev*.

Returns

`0` on success and a negative error value on failure.

Context

enabled, ccw device lock not held

ccw_device_set_online

LINUX

Kernel Hackers ManualFebruary 2010

Name

`ccw_device_set_online` — enable a ccw device for I/O

Synopsis

```
int ccw_device_set_online (struct ccw_device * cdev);
```

Arguments

cdev

target ccw device

Description

This function first enables *cdev* and then calls the driver's `set_online` function for *cdev*, if given. If `set_online` returns an error, *cdev* is disabled again.

Returns

`0` on success and a negative error value on failure.

Context

enabled, ccw device lock not held

get_ccwdev_by_busid

LINUX

Kernel Hackers ManualFebruary 2010

Name

`get_ccwdev_by_busid` — obtain device from a bus id

Synopsis

```
struct ccw_device * get_ccwdev_by_busid (struct ccw_driver *
cdrv, const char * bus_id);
```

Arguments

cdrv

driver the device is owned by

bus_id

bus id of the device to be searched

Description

This function searches all devices owned by *cdrv* for a device with a bus id matching *bus_id*.

Returns

If a match is found, its reference count of the found device is increased and it is returned; else `NULL` is returned.

ccw_driver_register

LINUX

Kernel Hackers ManualFebruary 2010

Name

`ccw_driver_register` — register a ccw driver

Synopsis

```
int ccw_driver_register (struct ccw_driver * cdriver);
```

Arguments

cdriver

driver to be registered

Description

This function is mainly a wrapper around `driver_register`.

Returns

`0` on success and a negative error value on failure.

ccw_driver_unregister

LINUX

Kernel Hackers ManualFebruary 2010

Name

`ccw_driver_unregister` — deregister a ccw driver

Synopsis

```
void ccw_driver_unregister (struct ccw_driver * cdriver);
```

Arguments

`cdriver`

driver to be deregistered

Description

This function is mainly a wrapper around `driver_unregister`.

ccw_device_set_options_mask

LINUX

Kernel Hackers ManualFebruary 2010

Name

`ccw_device_set_options_mask` — set some options and unset the rest

Synopsis

```
int ccw_device_set_options_mask (struct ccw_device * cdev,
unsigned long flags);
```

Arguments

cdev

device for which the options are to be set

flags

options to be set

Description

All flags specified in *flags* are set, all flags not specified in *flags* are cleared.

Returns

`0` on success, -`EINVAL` on an invalid flag combination.

ccw_device_set_options

LINUX

Kernel Hackers ManualFebruary 2010

Name

`ccw_device_set_options` — set some options

Synopsis

```
int ccw_device_set_options (struct ccw_device * cdev, unsigned
long flags);
```

Arguments

cdev

device for which the options are to be set

flags

options to be set

Description

All flags specified in *flags* are set, the remainder is left untouched.

Returns

`0` on success, -`EINVAL` if an invalid flag combination would ensue.

ccw_device_clear_options

LINUX

Kernel Hackers ManualFebruary 2010

Name

`ccw_device_clear_options` — clear some options

Synopsis

```
void ccw_device_clear_options (struct ccw_device * cdev,
unsigned long flags);
```

Arguments

cdev

device for which the options are to be cleared

flags

options to be cleared

Description

All flags specified in *flags* are cleared, the remainder is left untouched.

ccw_device_clear

LINUX

Name

`ccw_device_clear` — terminate I/O request processing

Synopsis

`int` **`ccw_device_clear`** `(struct ccw_device *` *`cdev`*`, unsigned long` *`intparm`*`);`

Arguments

`cdev`

target ccw device

`intparm`

interruption parameter; value is only used if no I/O is outstanding, otherwise the intparm associated with the I/O request is returned

Description

`ccw_device_clear` calls csch on *`cdev`*'s subchannel.

Returns

`0` on success, -`ENODEV` on device not operational, -`EINVAL` on invalid device state.

Context

Interrupts disabled, ccw device lock held

ccw_device_start_key

LINUX

Kernel Hackers ManualFebruary 2010

Name

`ccw_device_start_key` — start a s390 channel program with key

Synopsis

```
int ccw_device_start_key (struct ccw_device * cdev, struct
ccw1 * cpa, unsigned long intparm, __u8 lpm, __u8 key,
unsigned long flags);
```

Arguments

cdev

target ccw device

cpa

logical start address of channel program

intparm

user specific interruption parameter; will be presented back to *cdev*'s interrupt handler. Allows a device driver to associate the interrupt with a particular I/O request.

lpm

defines the channel path to be used for a specific I/O request. A value of 0 will make cio use the opm.

key

storage key to be used for the I/O

`flags`

additional flags; defines the action to be performed for I/O processing.

Description

Start a S/390 channel program. When the interrupt arrives, the IRQ handler is called, either immediately, delayed (dev-end missing, or sense required) or never (no IRQ handler registered).

Returns

`0`, if the operation was successful; -`EBUSY`, if the device is busy, or status pending; -`EACCES`, if no path specified in *`lpm`* is operational; -`ENODEV`, if the device is not operational.

Context

Interrupts disabled, ccw device lock held

ccw_device_start_timeout_key

LINUX

Kernel Hackers ManualFebruary 2010

Name

`ccw_device_start_timeout_key` — start a s390 channel program with timeout and key

Synopsis

```
int ccw_device_start_timeout_key (struct ccw_device * cdev,
struct ccw1 * cpa, unsigned long intparm, __u8 lpm, __u8 key,
```

```
unsigned long flags, int expires);
```

Arguments

`cdev`

target ccw device

`cpa`

logical start address of channel program

`intparm`

user specific interruption parameter; will be presented back to *`cdev`*'s interrupt handler. Allows a device driver to associate the interrupt with a particular I/O request.

`lpm`

defines the channel path to be used for a specific I/O request. A value of 0 will make cio use the opm.

`key`

storage key to be used for the I/O

`flags`

additional flags; defines the action to be performed for I/O processing.

`expires`

timeout value in jiffies

Description

Start a S/390 channel program. When the interrupt arrives, the IRQ handler is called, either immediately, delayed (dev-end missing, or sense required) or never (no IRQ handler registered). This function notifies the device driver if the channel program has not completed during the time specified by *`expires`*. If a timeout occurs, the channel program is terminated via xsch, hsch or csch, and the device's interrupt handler will be called with an irb containing ERR_PTR(-`ETIMEDOUT`).

Returns

`0`, if the operation was successful; -`EBUSY`, if the device is busy, or status pending; -`EACCES`, if no path specified in *lpm* is operational; -`ENODEV`, if the device is not operational.

Context

Interrupts disabled, ccw device lock held

ccw_device_start

LINUX

Kernel Hackers ManualFebruary 2010

Name

`ccw_device_start` — start a s390 channel program

Synopsis

```
int ccw_device_start (struct ccw_device * cdev, struct ccw1 *
cpa, unsigned long intparm, __u8 lpm, unsigned long flags);
```

Arguments

cdev

target ccw device

cpa

logical start address of channel program *intparm*

user specific interruption parameter; will be presented back to *cdev*'s interrupt handler. Allows a device driver to associate the interrupt with a particular I/O request.

lpm

defines the channel path to be used for a specific I/O request. A value of 0 will make cio use the opm.

flags

additional flags; defines the action to be performed for I/O processing.

Description

Start a S/390 channel program. When the interrupt arrives, the IRQ handler is called, either immediately, delayed (dev-end missing, or sense required) or never (no IRQ handler registered).

Returns

0, if the operation was successful; -EBUSY, if the device is busy, or status pending; -EACCES, if no path specified in *lpm* is operational; -ENODEV, if the device is not operational.

Context

Interrupts disabled, ccw device lock held

ccw_device_start_timeout

LINUX

Name

`ccw_device_start_timeout` — start a s390 channel program with timeout

Synopsis

```
int ccw_device_start_timeout (struct ccw_device * cdev, struct
ccw1 * cpa, unsigned long intparm, __u8 lpm, unsigned long
flags, int expires);
```

Arguments

`cdev`

target ccw device

`cpa`

logical start address of channel program

`intparm`

user specific interruption parameter; will be presented back to *`cdev`*'s interrupt handler. Allows a device driver to associate the interrupt with a particular I/O request.

`lpm`

defines the channel path to be used for a specific I/O request. A value of 0 will make cio use the opm.

`flags`

additional flags; defines the action to be performed for I/O processing.

`expires`

timeout value in jiffies

Description

Start a S/390 channel program. When the interrupt arrives, the IRQ handler is called, either immediately, delayed (dev-end missing, or sense required) or never (no IRQ handler registered). This function notifies the device driver if the channel program has not completed during the time specified by *expires*. If a timeout occurs, the channel program is terminated via xsch, hsch or csch, and the device's interrupt handler will be called with an irb containing ERR_PTR(-ETIMEDOUT).

Returns

0, if the operation was successful; -EBUSY, if the device is busy, or status pending; -EACCES, if no path specified in *lpm* is operational; -ENODEV, if the device is not operational.

Context

Interrupts disabled, ccw device lock held

ccw_device_halt

LINUX

Kernel Hackers ManualFebruary 2010

Name

ccw_device_halt — halt I/O request processing

Synopsis

```
int ccw_device_halt (struct ccw_device * cdev, unsigned long
intparm);
```

Arguments

cdev

target ccw device

intparm

interruption parameter; value is only used if no I/O is outstanding, otherwise the intparm associated with the I/O request is returned

Description

`ccw_device_halt` calls hsch on *cdev*'s subchannel.

Returns

`0` on success, -`ENODEV` on device not operational, -`EINVAL` on invalid device state, -`EBUSY` on device busy or interrupt pending.

Context

Interrupts disabled, ccw device lock held

ccw_device_resume

LINUX

Kernel Hackers ManualFebruary 2010

Name

`ccw_device_resume` — resume channel program execution

Synopsis

```
int ccw_device_resume (struct ccw_device * cdev);
```

Arguments

cdev

target ccw device

Description

`ccw_device_resume` calls rsch on *cdev*'s subchannel.

Returns

`0` on success, -`ENODEV` on device not operational, -`EINVAL` on invalid device state, -`EBUSY` on device busy or interrupt pending.

Context

Interrupts disabled, ccw device lock held

ccw_device_get_ciw

LINUX

Kernel Hackers ManualFebruary 2010

Name

`ccw_device_get_ciw` — Search for CIW command in extended sense data.

Synopsis

```
struct ciw * ccw_device_get_ciw (struct ccw_device * cdev,
__u32 ct);
```

Arguments

cdev

ccw device to inspect

ct

command type to look for

Description

During SenseID, command information words (CIWs) describing special commands available to the device may have been stored in the extended sense data. This function searches for CIWs of a specified command type in the extended sense data.

Returns

`NULL` if no extended sense data has been stored or if no CIW of the specified command type could be found, else a pointer to the CIW of the specified command type.

ccw_device_get_path_mask

LINUX

Kernel Hackers ManualFebruary 2010

Name

`ccw_device_get_path_mask` — get currently available paths

Synopsis

```
__u8 ccw_device_get_path_mask (struct ccw_device * cdev);
```

Arguments

`cdev`

ccw device to be queried

Returns

`0` if no subchannel for the device is available, else the mask of currently available paths for the ccw device's subchannel.

ccw_device_get_id

LINUX

Kernel Hackers ManualFebruary 2010

Name

`ccw_device_get_id` — obtain a ccw device id

Synopsis

```
void ccw_device_get_id (struct ccw_device * cdev, struct
ccw_dev_id * dev_id);
```

Arguments

cdev

device to obtain the id for

dev_id

where to fill in the values

2.3. The channel-measurement facility

The channel-measurement facility provides a means to collect measurement data which is made available by the channel subsystem for each channel attached device.

struct cmbdata

LINUX

Kernel Hackers ManualFebruary 2010

Name

`struct cmbdata` — channel measurement block data for user space

Synopsis

```
struct cmbdata {
  __u64 size;
```

```
    __u64 elapsed_time;
    __u64 ssch_rsch_count;
    __u64 sample_count;
    __u64 device_connect_time;
    __u64 function_pending_time;
    __u64 device_disconnect_time;
    __u64 control_unit_queuing_time;
    __u64 device_active_only_time;
    __u64 device_busy_time;
    __u64 initial_command_response_time;
};
```

Members

size

size of the stored data

elapsed_time

time since last sampling

ssch_rsch_count

number of ssch and rsch

sample_count

number of samples

device_connect_time

time of device connect

function_pending_time

time of function pending

device_disconnect_time

time of device disconnect

control_unit_queuing_time

time of control unit queuing

device_active_only_time

time of device active only

device_busy_time

time of device busy (ext. format)

initial_command_response_time

initial command response time (ext. format)

Description

All values are stored as 64 bit for simplicity, especially in 32 bit emulation mode. All time values are normalized to nanoseconds. Currently, two formats are known, which differ by the size of this structure, i.e. the last two members are only set when the extended channel measurement facility (first shipped in z990 machines) is activated. Potentially, more fields could be added, which would result in a new ioctl number.

enable_cmf

LINUX

Kernel Hackers ManualFebruary 2010

Name

`enable_cmf` — switch on the channel measurement for a specific device

Synopsis

```
int enable_cmf (struct ccw_device * cdev);
```

Arguments

cdev

The ccw device to be enabled

Description

Returns `0` for success or a negative error value.

Context

non-atomic

disable_cmf

LINUX

Kernel Hackers ManualFebruary 2010

Name

`disable_cmf` — switch off the channel measurement for a specific device

Synopsis

```
int disable_cmf (struct ccw_device * cdev);
```

Arguments

cdev

The ccw device to be disabled

Description

Returns `0` for success or a negative error value.

Context

non-atomic

cmf_read

LINUX

Kernel Hackers ManualFebruary 2010

Name

`cmf_read` — read one value from the current channel measurement block

Synopsis

```
u64 cmf_read (struct ccw_device * cdev, int index);
```

Arguments

cdev

the channel to be read

index

the index of the value to be read

Description

Returns the value read or `0` if the value cannot be read.

Context

any

cmf_readall

LINUX

Kernel Hackers ManualFebruary 2010

Name

`cmf_readall` — read the current channel measurement block

Synopsis

int **cmf_readall** (struct ccw_device * *cdev*, struct cmbdata * *data*);

Arguments

cdev

the channel to be read

data

a pointer to a data block that will be filled

Description

Returns `0` on success, a negative error value otherwise.

Context

any

Chapter 3. The ccwgroup bus

The ccwgroup bus only contains artificial devices, created by the user. Many networking devices (e.g. qeth) are in fact composed of several ccw devices (like read, write and data channel for qeth). The ccwgroup bus provides a mechanism to create a meta-device which contains those ccw devices as slave devices and can be associated with the netdevice.

3.1. ccw group devices

struct ccwgroup_device

LINUX

Kernel Hackers ManualFebruary 2010

Name

`struct ccwgroup_device` — ccw group device

Synopsis

```
struct ccwgroup_device {
  unsigned long creator_id;
  enum state;
  unsigned int count;
  struct device dev;
  struct ccw_device * cdev[0];
};
```

Members

creator_id

unique number of the driver

state

online/offline state

count

number of attached slave devices

dev

embedded device structure

cdev[0]

variable number of slave devices, allocated as needed

struct ccwgroup_driver

LINUX

Kernel Hackers ManualFebruary 2010

Name

`struct ccwgroup_driver` — driver for ccw group devices

Synopsis

```
struct ccwgroup_driver {
  struct module * owner;
  char * name;
  int max_slaves;
  unsigned long driver_id;
  int (* probe) (struct ccwgroup_device *);
  void (* remove) (struct ccwgroup_device *);
  int (* set_online) (struct ccwgroup_device *);
  int (* set_offline) (struct ccwgroup_device *);
  void (* shutdown) (struct ccwgroup_device *);
  struct device_driver driver;
};
```

Members

owner

driver owner

name

driver name

max_slaves

maximum number of slave devices

driver_id

unique id

probe

function called on probe

remove

function called on remove

set_online

function called when device is set online

set_offline

function called when device is set offline

shutdown

function called when device is shut down

driver

embedded driver structure

ccwgroup_create

LINUX

Name

`ccwgroup_create` — create and register a ccw group device

Synopsis

```
int ccwgroup_create (struct device * root, unsigned int
creator_id, struct ccw_driver * cdrv, int argc, char *
argv[]);
```

Arguments

root

parent device for the new device

creator_id

identifier of creating driver

cdrv

ccw driver of slave devices

argc

number of slave devices

argv[]

bus ids of slave devices

Description

Create and register a new ccw group device as a child of *root*. Slave devices are obtained from the list of bus ids given in *argv*[] and must all belong to *cdrv*.

Returns

`0` on success and an error code on failure.

Context

non-atomic

ccwgroup_driver_register

LINUX

Kernel Hackers ManualFebruary 2010

Name

`ccwgroup_driver_register` — register a ccw group driver

Synopsis

```
int ccwgroup_driver_register (struct ccwgroup_driver *
cdriver);
```

Arguments

cdriver

driver to be registered

Description

This function is mainly a wrapper around `driver_register`.

ccwgroup_driver_unregister

LINUX

Kernel Hackers ManualFebruary 2010

Name

`ccwgroup_driver_unregister` — deregister a ccw group driver

Synopsis

```
void ccwgroup_driver_unregister (struct ccwgroup_driver *
cdriver);
```

Arguments

cdriver

driver to be deregistered

Description

This function is mainly a wrapper around `driver_unregister`.

ccwgroup_probe_ccwdev

LINUX

Name

`ccwgroup_probe_ccwdev` — probe function for slave devices

Synopsis

```
int ccwgroup_probe_ccwdev (struct ccw_device * cdev);
```

Arguments

cdev

ccw device to be probed

Description

This is a dummy probe function for ccw devices that are slave devices in a ccw group device.

Returns

always `0`

ccwgroup_remove_ccwdev

LINUX

Name

`ccwgroup_remove_ccwdev` — remove function for slave devices

Synopsis

```
void ccwgroup_remove_ccwdev (struct ccw_device * cdev);
```

Arguments

cdev

ccw device to be removed

Description

This is a remove function for ccw devices that are slave devices in a ccw group device. It sets the ccw device offline and also deregisters the embedding ccw group device.

Chapter 4. Generic interfaces

Some interfaces are available to other drivers that do not necessarily have anything to do with the busses described above, but still are indirectly using basic infrastructure in the common I/O layer. One example is the support for adapter interrupts.

s390_register_adapter_interrupt

LINUX

Kernel Hackers ManualFebruary 2010

Name

`s390_register_adapter_interrupt` — register adapter interrupt handler

Synopsis

```
void * s390_register_adapter_interrupt (adapter_int_handler_t
handler, void * drv_data);
```

Arguments

handler

adapter handler to be registered

drv_data

driver data passed with each call to the handler

Returns

Pointer to the indicator to be used on success `ERR_PTR` if registration failed

s390_unregister_adapter_interrupt

LINUX

Kernel Hackers ManualFebruary 2010

Name

`s390_unregister_adapter_interrupt` — unregister adapter interrupt handler

Synopsis

```
void s390_unregister_adapter_interrupt (void * ind);
```

Arguments

ind

indicator for which the handler is to be unregistered