

Writing s390 channel device drivers

Cornelia Huck <cornelia.huck@de.ibm.com>

Table of Contents

1. Introduction	1
2. The ccw bus	2
I/O functions for channel-attached devices	2
ccw devices	16
The channel-measurement facility	49
3. The ccwgroup bus	55
ccw group devices	55
4. Generic interfaces	65
register_adapter_interrupt	66
unregister_adapter_interrupt	67
airq_iv_create	68
airq_iv_release	69
airq_iv_alloc	70
airq_iv_free	71
airq_iv_scan	72

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.

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.

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 addressing, 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.

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 <i>cons</i> == 1
res16	reserved

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

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

Name

struct irb — interruption response block

Synopsis

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

Members

scsw	subchannel status word
esw	extended status word
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 occurred, *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 sense itself). For unsolicited interrupts, the irb is passed as-is (expect for sense data, if applicable).

Name

`struct ccw_dev_id` — unique identifier for ccw devices

Synopsis

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

Members

<code>ssid</code>	subchannel set id
<code>devno</code>	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.

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

Name

`pathmask_to_pos` — find the position of the left-most bit in a pathmask

Synopsis

```
u8 pathmask_to_pos (u8 mask);
```

Arguments

mask pathmask with at least one bit set

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.

int_class interruption class to use for accounting interrupts

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.

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`.

Name

`ccw_device_siosl` — initiate logging

Synopsis

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

Arguments

cdev ccw device

Description

This function is used to invoke model-dependent logging within the channel subsystem.

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.

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.

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

Name

`ccw_device_get_mdc` — accumulate max data count

Synopsis

```
int ccw_device_get_mdc (struct ccw_device * cdev, u8 mask);
```

Arguments

cdev ccw device for which the max data count is accumulated

mask mask of paths to use

Description

Return the number of 64K-bytes blocks all paths at least support for a transport command. Return values ≤ 0 indicate failures.

Name

`ccw_device_tm_intrg` — perform interrogate function

Synopsis

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

Arguments

cdev ccw device on which to perform the interrogate function

Description

Perform an interrogate function on the given ccw device. Return zero on success, non-zero otherwise.

Name

`ccw_device_get_schid` — obtain a subchannel id

Synopsis

```
void    ccw_device_get_schid    (struct    ccw_device    *    cdev,    struct
subchannel_id * schid);
```

Arguments

cdev device to obtain the id for

schid where to fill in the values

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.

Name

arch/s390/include/asm/cmb.h — Document generation inconsistency

Oops

Warning

The template for this document tried to insert the structured comment from the file `arch/s390/include/asm/cmb.h` at this point, but none was found. This dummy section is inserted to allow generation to continue.

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

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

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.

ccw group devices

Name

ccwgroup_set_online — enable a ccwgroup device

Synopsis

```
int ccwgroup_set_online (struct ccwgroup_device * gdev);
```

Arguments

gdev target ccwgroup device

Description

This function attempts to put the ccwgroup device into the online state.

Returns

0 on success and a negative error value on failure.

Name

`ccwgroup_set_offline` — disable a ccwgroup device

Synopsis

```
int ccwgroup_set_offline (struct ccwgroup_device * gdev);
```

Arguments

gdev target ccwgroup device

Description

This function attempts to put the ccwgroup device into the offline state.

Returns

0 on success and a negative error value on failure.

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`.

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`.

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

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.

Name

`register_adapter_interrupt` — register adapter interrupt handler

Synopsis

```
int register_adapter_interrupt (struct airq_struct * airq);
```

Arguments

airq pointer to adapter interrupt descriptor

Description

Returns 0 on success, or -EINVAL.

Name

`unregister_adapter_interrupt` — unregister adapter interrupt handler

Synopsis

```
void unregister_adapter_interrupt (struct airq_struct * airq);
```

Arguments

airq pointer to adapter interrupt descriptor

Name

`airq_iv_create` — create an interrupt vector

Synopsis

```
struct airq_iv * airq_iv_create (unsigned long bits, unsigned long  
flags);
```

Arguments

bits number of bits in the interrupt vector

flags allocation flags

Description

Returns a pointer to an interrupt vector structure

Name

`airq_iv_release` — release an interrupt vector

Synopsis

```
void airq_iv_release (struct airq_iv * iv);
```

Arguments

iv pointer to interrupt vector structure

Name

`airq_iv_alloc` — allocate irq bits from an interrupt vector

Synopsis

```
unsigned long airq_iv_alloc (struct airq_iv * iv, unsigned long num);
```

Arguments

iv pointer to an interrupt vector structure

num number of consecutive irq bits to allocate

Description

Returns the bit number of the first irq in the allocated block of irqs, or -1UL if no bit is available or the `AIRQ_IV_ALLOC` flag has not been specified

Name

`airq_iv_free` — free irq bits of an interrupt vector

Synopsis

```
void airq_iv_free (struct airq_iv * iv, unsigned long bit, unsigned  
long num);
```

Arguments

iv pointer to interrupt vector structure

bit number of the first irq bit to free

num number of consecutive irq bits to free

Name

`airq_iv_scan` — scan interrupt vector for non-zero bits

Synopsis

```
unsigned long airq_iv_scan (struct airq_iv * iv, unsigned long start,  
unsigned long end);
```

Arguments

iv pointer to interrupt vector structure

start bit number to start the search

end bit number to end the search

Description

Returns the bit number of the next non-zero interrupt bit, or -1UL if the scan completed without finding any more any non-zero bits.