

Linux for S/390 and zSeries

Common Device Support (CDS) Device Driver I/O Support Routines

Authors:

- Ingo Adlung
- Cornelia Huck

Copyright, IBM Corp. 1999-2002

Introduction

This document describes the common device support routines for Linux/390. Different than other hardware architectures, ESA/390 has defined a unified I/O access method. This gives relief to the device drivers as they don't have to deal with different bus types, polling versus interrupt processing, shared versus non-shared interrupt processing, DMA versus port I/O (PIO), and other hardware features more. However, this implies that either every single device driver needs to implement the hardware I/O attachment functionality itself, or the operating system provides for a unified method to access the hardware, providing all the functionality that every single device driver would have to provide itself.

The document does not intend to explain the ESA/390 hardware architecture in every detail. This information can be obtained from the ESA/390 Principles of Operation manual (IBM Form. No. SA22-7201).

In order to build common device support for ESA/390 I/O interfaces, a functional layer was introduced that provides generic I/O access methods to the hardware.

The common device support layer comprises the I/O support routines defined below. Some of them implement common Linux device driver interfaces, while some of them are ESA/390 platform specific.

Note:

In order to write a driver for S/390, you also need to look into the interface described in Documentation/s390/driver-model.rst.

Note for porting drivers from 2.4:

The major changes are:

- The functions use a `ccw_device` instead of an `irq` (subchannel).
- All drivers must define a `ccw_driver` (see `driver-model.txt`) and the associated functions.
- `request_irq()` and `free_irq()` are no longer done by the driver.
- The `oper_handler` is (kindof) replaced by the `probe()` and `set_online()` functions of the `ccw_driver`.
- The `not_oper_handler` is (kindof) replaced by the `remove()` and `set_offline()` functions of the `ccw_driver`.
- The channel device layer is gone.
- The interrupt handlers must be adapted to use a `ccw_device` as argument. Moreover, they don't return a `devstat`, but an `irb`.
- Before initiating an io, the options must be set via `ccw_device_set_options()`.
- Instead of calling `read_dev_chars()/read_conf_data()`, the driver issues the channel program and handles the interrupt itself.

`ccw_device_get_ciw()`

get commands from extended sense data.

`ccw_device_start()`, `ccw_device_start_timeout()`, `ccw_device_start_key()`, `ccw_device_start_key_timeout()`

initiate an I/O request.

`ccw_device_resume()`

resume channel program execution.

`ccw_device_halt()`

terminate the current I/O request processed on the device.

`do_IRQ()`

generic interrupt routine. This function is called by the interrupt entry routine whenever an I/O interrupt is presented to the system. The `do_IRQ()` routine determines the interrupt status and calls the device specific interrupt handler according to the rules (flags) defined during I/O request initiation with `do_IO()`.

The next chapters describe the functions other than `do_IRQ()` in more details. The `do_IRQ()` interface is not described, as it is called from the Linux/390 first level interrupt handler only and does not comprise a device driver callable interface. Instead, the functional description of `do_IO()` also describes the input to the device specific interrupt handler.

Note:

All explanations apply also to the 64 bit architecture s390x.

Common Device Support (CDS) for Linux/390 Device Drivers

General Information

The following chapters describe the I/O related interface routines the Linux/390 common device support (CDS) provides to allow for device specific driver implementations on the IBM ESA/390 hardware platform. Those interfaces intend to provide the functionality required by every device driver implementation to allow to drive a specific hardware device on the ESA/390 platform. Some of the interface routines are specific to Linux/390 and some of them can be found on other Linux platforms implementations too. Miscellaneous function prototypes, data declarations, and macro definitions can be found in the architecture specific C header file `linux/arch/s390/include/asm/irq.h`.

Overview of CDS interface concepts

Different to other hardware platforms, the ESA/390 architecture doesn't define interrupt lines managed by a specific interrupt controller and bus systems that may or may not allow for shared interrupts, DMA processing, etc.. Instead, the ESA/390 architecture has implemented a so called channel subsystem, that provides a unified view of the devices physically attached to the systems. Though the ESA/390 hardware platform knows about a huge variety of different peripheral attachments like disk devices (aka. DASDs), tapes, communication controllers, etc. they can all be accessed by a well defined access method and they are presenting I/O completion a unified way : I/O interruptions. Every single device is uniquely identified to the system by a so called subchannel, where the ESA/390 architecture allows for 64k devices be attached.

Linux, however, was first built on the Intel PC architecture, with its two cascaded 8259 programmable interrupt controllers (PICs), that allow for a maximum of 15 different interrupt lines. All devices attached to such a system share those 15 interrupt levels. Devices attached to the ISA bus system must not share interrupt levels (aka. IRQs), as the ISA bus bases on edge triggered interrupts. MCA, EISA, PCI and other bus systems base on level triggered interrupts, and therewith allow for shared IRQs. However, if multiple devices present their hardware status by the same (shared) IRQ, the operating system has to call every single device driver registered on this IRQ in order to determine the device driver owning the device that raised the interrupt.

Up to kernel 2.4, Linux/390 used to provide interfaces via the IRQ (subchannel). For internal use of the common I/O layer, these are still there. However, device drivers should use the new calling interface via the `ccw_device` only.

During its startup the Linux/390 system checks for peripheral devices. Each of those devices is uniquely defined by a so called subchannel by the ESA/390 channel subsystem. While the subchannel numbers are system generated, each subchannel also takes a user defined attribute, the so called device number. Both subchannel number and device number cannot exceed 65535. During sysfs initialisation, the information about control unit type and device types that imply specific I/O commands (channel command words - CCWs) in order to operate the device are gathered. Device drivers can retrieve this set of hardware information during their initialization step to recognize the devices they support using the information saved in the struct `ccw_device` given to them. This methods implies that Linux/390 doesn't require to probe for free (not armed) interrupt request lines (IRQs) to drive its devices with. Where applicable, the device drivers can use issue the READ DEVICE CHARACTERISTICS ccw to retrieve device characteristics in its online routine.

In order to allow for easy I/O initiation the CDS layer provides a `ccw_device_start()` interface that takes a device specific channel program (one or more CCWs) as input sets up the required architecture specific control blocks and initiates an I/O request on behalf of the device driver. The `ccw_device_start()` routine allows to specify whether it expects the CDS layer to notify the device driver for every interrupt it observes, or with final status only. See `ccw_device_start()` for more details. A device driver must never issue ESA/390 I/O commands itself, but must use the Linux/390 CDS interfaces instead.

For long running I/O request to be canceled, the CDS layer provides the `ccw_device_halt()` function. Some devices require to initially issue a HALT SUBCHANNEL (HSCH) command without having pending I/O requests. This function is also covered by `ccw_device_halt()`.

`get_ciw()` - get command information word

This call enables a device driver to get information about supported commands from the extended SenseID data.

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

cdev	The ccw_device for which the command is to be retrieved.
cmd	The command type to be retrieved.

`ccw_device_get_ciw()` returns:

NULL	No extended data available, invalid device or command not found.
!NULL	The command requested.

`ccw_device_start()` - Initiate I/O Request

The `ccw_device_start()` routines is the I/O request front-end processor. All device driver I/O requests must be issued using this routine. A device driver must not issue ESA/390 I/O commands itself. Instead the `ccw_device_start()` routine provides all interfaces required to drive arbitrary devices.

This description also covers the status information passed to the device driver's interrupt handler as this is related to the rules (flags) defined with the associated I/O request when calling `ccw_device_start()`.

```
int ccw_device_start(struct ccw_device *cdev,
                    struct ccwl *cpa,
                    unsigned long intparm,
```

```

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

```

cdev	ccw_device the I/O is destined for
cpa	logical start address of channel program
user_intparm	user specific interrupt information; will be presented back to the device driver'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	the storage key to use for the I/O (useful for operating on a storage with a storage key != default key)
flag	defines the action to be performed for I/O processing
expires	timeout value in jiffies. The common I/O layer will terminate the running program after this and call the interrupt handler with ERR_PTR(-ETIMEDOUT) as irb.

Possible flag values are:

DOIO_ALLOW_SUSPEND	channel program may become suspended
DOIO_DENY_PREFETCH	don't allow for CCW prefetch; usually this implies the channel program might become modified
DOIO_SUPPRESS_INTER	don't call the handler on intermediate status

The cpa parameter points to the first format 1 CCW of a channel program:

```

struct ccw1 {
    __u8 cmd_code; /* command code */
    __u8 flags; /* flags, like IDA addressing, etc. */
    __u16 count; /* byte count */
    __u32 cda; /* data address */
} __attribute__((packed, aligned(8)));

```

with the following CCW flags values defined:

CCW_FLAG_DC	data chaining
CCW_FLAG_CC	command chaining
CCW_FLAG_SLI	suppress incorrect length
CCW_FLAG_SKIP	skip
CCW_FLAG_PCI	PCI
CCW_FLAG_IDA	indirect addressing
CCW_FLAG_SUSPEND	suspend

Via ccw_device_set_options(), the device driver may specify the following options for the device:

DOIO_EARLY_NOTIFICATION	allow for early interrupt notification
DOIO_REPORT_ALL	report all interrupt conditions

The ccw_device_start() function returns:

0	successful completion or request successfully initiated
-EBUSY	The device is currently processing a previous I/O request, or there is a status pending at the device.
-ENODEV	cdev is invalid, the device is not operational or the ccw_device is not online.

When the I/O request completes, the CDS first level interrupt handler will accumulate the status in a struct irb and then call the device interrupt handler. The intparm field will contain the value the device driver has associated with a particular I/O request. If a pending device status was recognized, intparm will be set to 0 (zero). This may happen during I/O initiation or delayed by an alert status notification. In any case this status is not related to the current (last) I/O request. In case of a delayed status notification no special interrupt will be presented to indicate I/O completion as the I/O request was never started, even though ccw_device_start() returned with successful completion.

The irb may contain an error value, and the device driver should check for this first:

-ETIMEDOUT	the common I/O layer terminated the request after the specified timeout value
-EIO	the common I/O layer terminated the request due to an error state

If the concurrent sense flag in the extended status word (esw) in the irb is set, the field `erw.scnt` in the esw describes the number of device specific sense bytes available in the extended control word `irb->scsw.ecw[]`. No device sensing by the device driver itself is required.

The device interrupt handler can use the following definitions to investigate the primary unit check source coded in sense byte 0 :

SNS0_CMD_REJECT	0x80
SNS0_INTERVENTION_REQ	0x40
SNS0_BUS_OUT_CHECK	0x20
SNS0_EQUIPMENT_CHECK	0x10
SNS0_DATA_CHECK	0x08
SNS0_OVERRUN	0x04
SNS0_INCOMPL_DOMAIN	0x01

Depending on the device status, multiple of those values may be set together. Please refer to the device specific documentation for details.

The `irb->scsw.cstat` field provides the (accumulated) subchannel status :

SCHN_STAT_PCI	program controlled interrupt
SCHN_STAT_INCORR_LEN	incorrect length
SCHN_STAT_PROG_CHECK	program check
SCHN_STAT_PROT_CHECK	protection check
SCHN_STAT_CHN_DATA_CHK	channel data check
SCHN_STAT_CHN_CTRL_CHK	channel control check
SCHN_STAT_INTF_CTRL_CHK	interface control check
SCHN_STAT_CHAIN_CHECK	chaining check

The `irb->scsw.dstat` field provides the (accumulated) device status :

DEV_STAT_ATTENTION	attention
DEV_STAT_STAT_MOD	status modifier
DEV_STAT_CU_END	control unit end
DEV_STAT_BUSY	busy
DEV_STAT_CHN_END	channel end
DEV_STAT_DEV_END	device end
DEV_STAT_UNIT_CHECK	unit check
DEV_STAT_UNIT_EXCEP	unit exception

Please see the ESA/390 Principles of Operation manual for details on the individual flag meanings.

Usage Notes:

`ccw_device_start()` must be called disabled and with the ccw device lock held.

The device driver is allowed to issue the next `ccw_device_start()` call from within its interrupt handler already. It is not required to schedule a bottom-half, unless a non deterministically long running error recovery procedure or similar needs to be scheduled. During I/O processing the Linux/390 generic I/O device driver support has already obtained the IRQ lock, i.e. the handler must not try to obtain it again when calling `ccw_device_start()` or we end in a deadlock situation!

If a device driver relies on an I/O request to be completed prior to start the next it can reduce I/O processing overhead by chaining a NoOp I/O command `CCW_CMD_NOOP` to the end of the submitted CCW chain. This will force Channel-End and Device-End status to be presented together, with a single interrupt. However, this should be used with care as it implies the channel will remain busy, not being able to process I/O requests for other devices on the same channel. Therefore e.g. read commands should never use this technique, as the result will be presented by a single interrupt anyway.

In order to minimize I/O overhead, a device driver should use the `DOIO_REPORT_ALL` only if the device can report intermediate interrupt information prior to device-end the device driver urgently relies on. In this case all I/O interruptions are presented to the device driver until final status is recognized.

If a device is able to recover from asynchronously presented I/O errors, it can perform overlapping I/O using the `DOIO_EARLY_NOTIFICATION` flag. While some devices always report channel-end and device-end together, with a single interrupt, others present primary status (channel-end) when the channel is ready for the next I/O request and secondary status (device-end) when the data transmission has been completed at the device.

Above flag allows to exploit this feature, e.g. for communication devices that can handle lost data on the network to allow for enhanced I/O processing.

Unless the channel subsystem at any time presents a secondary status interrupt, exploiting this feature will cause only primary status

interrupts to be presented to the device driver while overlapping I/O is performed. When a secondary status without error (alert status) is presented, this indicates successful completion for all overlapping ccw_device_start() requests that have been issued since the last secondary (final) status.

Channel programs that intend to set the suspend flag on a channel command word (CCW) must start the I/O operation with the DOIO_ALLOW_SUSPEND option or the suspend flag will cause a channel program check. At the time the channel program becomes suspended an intermediate interrupt will be generated by the channel subsystem.

ccw_device_resume() - Resume Channel Program Execution

If a device driver chooses to suspend the current channel program execution by setting the CCW suspend flag on a particular CCW, the channel program execution is suspended. In order to resume channel program execution the CIO layer provides the ccw_device_resume() routine.

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

cdev	ccw_device the resume operation is requested for
------	--

The ccw_device_resume() function returns:

0	suspended channel program is resumed
-EBUSY	status pending
-ENODEV	cdev invalid or not-operational subchannel
-EINVAL	resume function not applicable
-ENOTCONN	there is no I/O request pending for completion

Usage Notes:

Please have a look at the ccw_device_start() usage notes for more details on suspended channel programs.

ccw_device_halt() - Halt I/O Request Processing

Sometimes a device driver might need a possibility to stop the processing of a long-running channel program or the device might require to initially issue a halt subchannel (HSCH) I/O command. For those purposes the ccw_device_halt() command is provided.

ccw_device_halt() must be called disabled and with the ccw device lock held.

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

cdev	ccw_device the halt operation is requested for
intparm	interruption parameter; value is only used if no I/O is outstanding, otherwise the intparm associated with the I/O request is returned

The ccw_device_halt() function returns:

0	request successfully initiated
-EBUSY	the device is currently busy, or status pending.
-ENODEV	cdev invalid.
-EINVAL	The device is not operational or the ccw device is not online.

Usage Notes:

A device driver may write a never-ending channel program by writing a channel program that at its end loops back to its beginning by means of a transfer in channel (TIC) command (CCW_CMD_TIC). Usually this is performed by network device drivers by setting the PCI CCW flag (CCW_FLAG_PCI). Once this CCW is executed a program controlled interrupt (PCI) is generated. The device driver can then perform an appropriate action. Prior to interrupt of an outstanding read to a network device (with or without PCI flag) a ccw_device_halt() is required to end the pending operation.

```
ccw_device_clear() - Terminate I/O Request Processing
```

In order to terminate all I/O processing at the subchannel, the clear subchannel (CSCH) command is used. It can be issued via ccw_device_clear().

ccw_device_clear() must be called disabled and with the ccw device lock held.

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

cdev	ccw_device the clear operation is requested for
intparm	interruption parameter (see ccw_device_halt())

The ccw_device_clear() function returns:

0	request successfully initiated
-ENODEV	cdev invalid
-EINVAL	The device is not operational or the ccw device is not online.

Miscellaneous Support Routines

This chapter describes various routines to be used in a Linux/390 device driver programming environment.

`get_ccwdev_lock()`

Get the address of the device specific lock. This is then used in `spin_lock()` / `spin_unlock()` calls.

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

Get the mask of the path currently available for cdev.