S/390 driver model interfaces

1. CCW devices

All devices which can be addressed by means of ccws are called 'CCW devices' - even if they aren't actually driven by ccws.

All ccw devices are accessed via a subchannel, this is reflected in the structures under devices/:

```
devices/
     - system/
     - css0/
           - 0.0.0000/0.0.0815/
           - 0.0.0001/0.0.4711/
           - 0.0.0002/
           - 0.1.0000/0.1.1234/
           ...
           - defunct/
```

In this example, device 0815 is accessed via subchannel 0 in subchannel set 0, device 4711 via subchannel 1 in subchannel set 0, and subchannel 2 is a non-I/O subchannel. Device 1234 is accessed via subchannel 0 in subchannel set 1.

The subchannel named 'defunct' does not represent any real subchannel on the system; it is a pseudo subchannel where disconnected ccw devices are moved to if they are displaced by another ccw device becoming operational on their former subchannel. The ccw devices will be moved again to a proper subchannel if they become operational again on that subchannel.

You should address a ccw device via its bus id (e.g. 0.0.4711); the device can be found under bus/ccw/devices/.

All ccw devices export some data via sysfs.

```
cutype:       The control unit type / model.

devtype:      The device type / model, if applicable.

availability: Can be 'good' or 'boxed'; 'no path' or 'no device' for
              disconnected devices.

online:       An interface to set the device online and offline.
              In the special case of the device being disconnected (see the
              notify function under 1.2), piping 0 to online will forcibly delete
              the device.
```

The device drivers can add entries to export per-device data and interfaces.

There is also some data exported on a per-subchannel basis (see under bus/css/devices/):

```
chpids:       Via which chpids the device is connected.
```

pimpampom: The path installed, path available and path operational masks.

There also might be additional data, for example for block devices.

1.1 Bringing up a ccw device

This is done in several steps.

a. Each driver can provide one or more parameter interfaces where parameters can be specified. These interfaces are also in the driver's responsibility.
b. After a. has been performed, if necessary, the device is finally brought up via the 'online' interface.

1.2 Writing a driver for ccw devices

The basic struct ccw_device and struct ccw_driver data structures can be found under include/asm/ccwdev.h.

```
struct ccw_device {
        spinlock_t *ccwlock;
        struct ccw_device_private *private;
        struct ccw_device_id id;

        struct ccw_driver *drv;
        struct device dev;
        int online;

        void (*handler) (struct ccw_device *dev, unsigned long intparm,
                         struct irb *irb);
};

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

The 'private' field contains data needed for internal i/o operation only, and is not available to the device driver.

Each driver should declare in a MODULE_DEVICE_TABLE into which CU types/models and/or device types/models it is interested. This information can later be found in the struct ccw_device_id fields:

```
struct ccw_device_id {
        __u16   match_flags;
        __u16   cu_type;
        __u16   dev_type;
        __u8    cu_model;
        __u8    dev_model;

        unsigned long driver_info;
};
```

```
The functions in ccw_driver should be used in the following way:
probe:   This function is called by the device layer for each device the driver
         is interested in. The driver should only allocate private structures
         to put in dev->driver_data and create attributes (if needed). Also,
         the interrupt handler (see below) should be set here.

int (*probe) (struct ccw_device *cdev);

Parameters:  cdev     - the device to be probed.


remove:  This function is called by the device layer upon removal of the driver,
         the device or the module. The driver should perform cleanups here.

int (*remove) (struct ccw_device *cdev);

Parameters:   cdev    - the device to be removed.


set_online: This function is called by the common I/O layer when the device is
            activated via the 'online' attribute. The driver should finally
            setup and activate the device here.

int (*set_online) (struct ccw_device *);

Parameters:   cdev      - the device to be activated. The common layer has
                          verified that the device is not already online.


set_offline: This function is called by the common I/O layer when the device is
             de-activated via the 'online' attribute. The driver should shut
             down the device, but not de-allocate its private data.

int (*set_offline) (struct ccw_device *);

Parameters:   cdev       - the device to be deactivated. The common layer has
                           verified that the device is online.


notify: This function is called by the common I/O layer for some state changes
        of the device.
        Signalled to the driver are:
        * In online state, device detached (CIO_GONE) or last path gone
          (CIO_NO_PATH). The driver must return !0 to keep the device; for
          return code 0, the device will be deleted as usual (also when no
          notify function is registered). If the driver wants to keep the
          device, it is moved into disconnected state.
        * In disconnected state, device operational again (CIO_OPER). The
          common I/O layer performs some sanity checks on device number and
          Device / CU to be reasonably sure if it is still the same device.
          If not, the old device is removed and a new one registered. By the
          return code of the notify function the device driver signals if it
          wants the device back: !0 for keeping, 0 to make the device being
          removed and re-registered.

int (*notify) (struct ccw_device *, int);

Parameters:   cdev    - the device whose state changed.
              event   - the event that happened. This can be one of CIO_GONE,
                        CIO_NO_PATH or CIO_OPER.
```

The handler field of the struct ccw_device is meant to be set to the interrupt
handler for the device. In order to accommodate drivers which use several
distinct handlers (e.g. multi subchannel devices), this is a member of
ccw_device
instead of ccw_driver.
The handler is registered with the common layer during set_online() processing
before the driver is called, and is deregistered during set_offline() after the
driver has been called. Also, after registering / before deregistering, path
grouping resp. disbanding of the path group (if applicable) are performed.

```
void (*handler) (struct ccw_device *dev, unsigned long intparm, struct irb
*irb);

Parameters:     dev     - the device the handler is called for
                intparm - the intparm which allows the device driver to identify
                          the i/o the interrupt is associated with, or to
recognize
                          the interrupt as unsolicited.
                irb     - interruption response block which contains the
accumulated
                          status.
```

The device driver is called from the common ccw_device layer and can retrieve
information about the interrupt from the irb parameter.

1.3 ccwgroup devices

The ccwgroup mechanism is designed to handle devices consisting of multiple ccw
devices, like lcs or ctc.

The ccw driver provides a 'group' attribute. Piping bus ids of ccw devices to
this attributes creates a ccwgroup device consisting of these ccw devices (if
possible). This ccwgroup device can be set online or offline just like a normal
ccw device.

Each ccwgroup device also provides an 'ungroup' attribute to destroy the device
again (only when offline). This is a generic ccwgroup mechanism (the driver does
not need to implement anything beyond normal removal routines).

A ccw device which is a member of a ccwgroup device carries a pointer to the ccwgroup device in the driver_data of its device struct. This field must not be touched by the driver - it should use the ccwgroup device's driver_data for its private data.

To implement a ccwgroup driver, please refer to include/asm/ccwgroup.h. Keep in mind that most drivers will need to implement both a ccwgroup and a ccw driver.

2. Channel paths

Channel paths show up, like subchannels, under the channel subsystem root (css0) and are called 'chp0.<chpid>'. They have no driver and do not belong to any bus. Please note, that unlike /proc/chpids in 2.4, the channel path objects reflect only the logical state and not the physical state, since we cannot track the latter consistently due to lacking machine support (we don't need to be aware of it anyway).

```
status - Can be 'online' or 'offline'.
         Piping 'on' or 'off' sets the chpid logically online/offline.
         Piping 'on' to an online chpid triggers path reprobing for all devices
         the chpid connects to. This can be used to force the kernel to re-use
         a channel path the user knows to be online, but the machine hasn't
         created a machine check for.
```

type - The physical type of the channel path.

shared - Whether the channel path is shared.

cmg - The channel measurement group.

3. System devices

3.1 xpram

xpram shows up under devices/system/ as 'xpram'.

3.2 cpus

For each cpu, a directory is created under devices/system/cpu/. Each cpu has an attribute 'online' which can be 0 or 1.

4. Other devices

4.1 Netiucv

The netiucv driver creates an attribute 'connection' under bus/iucv/drivers/netiucv. Piping to this attribute creates a new netiucv connection to the specified host.

Netiucv connections show up under devices/iucv/ as "netiucv<ifnum>". The interface
number is assigned sequentially to the connections defined via the 'connection' attribute.

```
user                          - shows the connection partner.

buffer                        - maximum buffer size.
                                Pipe to it to change buffer size.
```