```
~~/*~~
~~ * cdc-acm.c~~
~~ *~~
~~ * Copyright (c) 1999 Armin Fuerst     <fuerst@in.tum.de>~~
~~ * Copyright (c) 1999 Pavel Machek     <pavel@ucw.cz>~~
~~ * Copyright (c) 1999 Johannes Erdfelt<johannes@erdfelt.com>~~
~~ * Copyright (c) 2000 Vojtech Pavlik   <vojtech@suse.cz>~~
~~ * Copyright (c) 2004 Oliver Neukum    <oliver@neukum.name>~~
~~ * Copyright (c) 2005 David Kubicek    <dave@awk.cz>~~
~~ * Copyright (c) 2011 Johan Hovold     <jhovold@gmail.com>~~
~~ *~~
~~ * USB Abstract Control Model driver for USB modems and ISDN~~
~~adapters~~
~~ *~~
~~ * Sponsored by SuSE~~
~~ *~~
~~ * This program is free software; you can redistribute it and/or~~
~~modify~~
~~ * it under the terms of the GNU General Public License as~~
~~published by~~
~~ * the Free Software Foundation; either version 2 of the License,~~
~~or~~
~~ * (at your option) any later version.~~
~~ *~~
~~ * This program is distributed in the hope that it will be~~
~~useful,~~
~~ * but WITHOUT ANY WARRANTY; without even the implied warranty of~~
~~ * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the~~
~~ * GNU General Public License for more details.~~
~~ *~~
~~ * You should have received a copy of the GNU General Public~~
~~License~~
~~ * along with this program; if not, write to the Free Software~~
~~ * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA~~
~~02111-1307 USA~~
~~ */~~

#undef DEBUG
#undef VERBOSE_DEBUG

#include <linux/kernel.h>       // su dung cac API cua kernel
#include <linux/sched/signal.h>
#include <linux/errno.h>
#include <linux/init.h>
#include <linux/slab.h>
#include <linux/tty.h>          Header file that contains the definition of struct tty_struct
#include <linux/serial.h>
#include <linux/tty_driver.h> contains the definition of struct tty_driver and declare the different flags used
#include <linux/tty_flip.h>contains some tty flip buffer inline functions that make it easier to manipulate the flip
#include <linux/module.h>  buffer structures.
#include <linux/mutex.h>
#include <linux/uaccess.h>
```

```
#include <linux/usb.h>
#include <linux/usb/cdc.h>
#include <asm/byteorder.h>
#include <asm/unaligned.h>
#include <linux/idr.h>
#include <linux/list.h>

#include "cdc-acm.h"
```

~~#define DRIVER_AUTHOR "Armin Fuerst, Pavel Machek, Johannes Erdfelt, Vojtech Pavlik, David Kubicek, Johan Hovold"~~
~~#define DRIVER_DESC "USB Abstract Control Model driver for USB modems and ISDN adapters"~~

```
static struct usb_driver acm_driver;
static struct tty_driver *acm_tty_driver;

static DEFINE_IDR(acm_minors);
static DEFINE_MUTEX(acm_minors_lock);

static void acm_tty_set_termios(struct tty_struct *tty,
                      struct ktermios *termios_old);

/*
 * acm_minors accessors
 */

/*
 * Look up an ACM structure by minor. If found and not
disconnected, increment
 * its refcount and return it with its mutex held.
 */
static struct acm *acm_get_by_minor(unsigned int minor)
{
	struct acm *acm;

	mutex_lock(&acm_minors_lock);
	acm = idr_find(&acm_minors, minor);
	if (acm) {
		mutex_lock(&acm->mutex);
		if (acm->disconnected) {
			mutex_unlock(&acm->mutex);
			acm = NULL;
		} else {
			tty_port_get(&acm->port);
			mutex_unlock(&acm->mutex);
		}
	}
	mutex_unlock(&acm_minors_lock);
	return acm;
}
```

```
/*
 * Try to find an available minor number and if found, associate
it with 'acm'.
 */
static int acm_alloc_minor(struct acm *acm)
{
	int minor;

	mutex_lock(&acm_minors_lock);
	minor = idr_alloc(&acm_minors, acm, 0, ACM_TTY_MINORS,
GFP_KERNEL);
	mutex_unlock(&acm_minors_lock);

	return minor;
}

/* Release the minor number associated with 'acm'.  */
static void acm_release_minor(struct acm *acm)
{
	mutex_lock(&acm_minors_lock);
	idr_remove(&acm_minors, acm->minor);
	mutex_unlock(&acm_minors_lock);
}

/*
 * Functions for ACM control messages.
 */

static int acm_ctrl_msg(struct acm *acm, int request, int value,
							void *buf, int len)
{
	int retval;

	retval = usb_autopm_get_interface(acm->control);
	if (retval)
		return retval;

	retval = usb_control_msg(acm->dev, usb_sndctrlpipe(acm->dev,
0),
		request, USB_RT_ACM, value,
		acm->control->altsetting[0].desc.bInterfaceNumber,
		buf, len, 5000);

	dev_dbg(&acm->control->dev,
		"%s - rq 0x%02x, val %#x, len %#x, result %d\n",
		__func__, request, value, len, retval);

	usb_autopm_put_interface(acm->control);

	return retval < 0 ? retval : 0;
}
```

```
~~/* devices aren't required to support these requests.~~
~~ * the cdc acm descriptor tells whether they do...~~
~~ */~~
~~static inline int acm_set_control(struct acm *acm, int control)~~
~~{~~
        ~~if (acm->quirks & QUIRK_CONTROL_LINE_STATE)~~
                ~~return -EOPNOTSUPP;~~

        ~~return acm_ctrl_msg(acm, USB_CDC_REQ_SET_CONTROL_LINE_STATE,~~
                        ~~control, NULL, 0);~~
~~}~~

~~#define acm_set_line(acm, line) \~~
        ~~acm_ctrl_msg(acm, USB_CDC_REQ_SET_LINE_CODING, 0, line,~~
~~sizeof *(line))~~
~~#define acm_send_break(acm, ms) \~~
        ~~acm_ctrl_msg(acm, USB_CDC_REQ_SEND_BREAK, ms, NULL, 0)~~

~~static void acm_kill_urbs(struct acm *acm)~~
~~{~~
        ~~int i;~~

        ~~usb_kill_urb(acm->ctrlurb);~~
        ~~for (i = 0; i < ACM_NW; i++)~~
                ~~usb_kill_urb(acm->wb[i].urb);~~
        ~~for (i = 0; i < acm->rx_buflimit; i++)~~
                ~~usb_kill_urb(acm->read_urbs[i]);~~
~~}~~

/*
 * Write buffer management.
 * All of these assume proper locks taken by the caller.
 */

static int acm_wb_alloc(struct acm *acm)
{
        int i, wbn;
        struct acm_wb *wb;

        wbn = 0;
        i = 0;
        for (;;) {
                wb = &acm->wb[wbn];
                if (!wb->use) {
                        wb->use = 1;
                        return wbn;
                }
                wbn = (wbn + 1) % ACM_NW;
                if (++i >= ACM_NW)
                        return -1;
        }
}

static int acm_wb_is_avail(struct acm *acm)
{
	int i, n;
	unsigned long flags;

	n = ACM_NW;
	spin_lock_irqsave(&acm->write_lock, flags);
	for (i = 0; i < ACM_NW; i++)
		n -= acm->wb[i].use;
	spin_unlock_irqrestore(&acm->write_lock, flags);
	return n;
}

/*
 * Finish write. Caller must hold acm->write_lock
 */
static void acm_write_done(struct acm *acm, struct acm_wb *wb)
{
	wb->use = 0;
	acm->transmitting--;
	usb_autopm_put_interface_async(acm->control);
}

/*
 * Poke write.
 *
 * the caller is responsible for locking
 */

static int acm_start_wb(struct acm *acm, struct acm_wb *wb)
{
	int rc;

	acm->transmitting++;

	wb->urb->transfer_buffer = wb->buf;
	wb->urb->transfer_dma = wb->dmah;
	wb->urb->transfer_buffer_length = wb->len;
	wb->urb->dev = acm->dev;

	rc = usb_submit_urb(wb->urb, GFP_ATOMIC);
	if (rc < 0) {
		dev_err(&acm->data->dev,
			"%s - usb_submit_urb(write bulk) failed: %d\n",
			__func__, rc);
		acm_write_done(acm, wb);
	}
	return rc;
}
```

Sysfs: do tính phức tạp trong cấu trúc vật lý của thiết bị USB, Sysfs được sử dụng như một trình quản lý dữ liệu của Linux giúp việc quản lý những thiết bị như này được đơn giản hơn. mô tả vật lý thiết bị USB cũng như interface của nó được mô tả như những thiết bị độc lập trong Sysfs

```
~~/*~~
 ~~* attributes exported through sysfs~~
 ~~*/~~
~~static ssize_t show_caps~~
~~(struct device *dev, struct device_attribute *attr, char *buf)~~
~~{~~
	~~struct usb_interface *intf = to_usb_interface(dev);~~
	~~struct acm *acm = usb_get_intfdata(intf);~~

	~~return sprintf(buf, "%d", acm->ctrl_caps);~~
~~}~~
~~static DEVICE_ATTR(bmCapabilities, S_IRUGO, show_caps, NULL);~~

~~static ssize_t show_country_codes~~
~~(struct device *dev, struct device_attribute *attr, char *buf)~~
~~{~~
	~~struct usb_interface *intf = to_usb_interface(dev);~~
	~~struct acm *acm = usb_get_intfdata(intf);~~

	~~memcpy(buf, acm->country_codes, acm->country_code_size);~~
	~~return acm->country_code_size;~~
~~}~~

~~static DEVICE_ATTR(wCountryCodes, S_IRUGO, show_country_codes,~~
~~NULL);~~

~~static ssize_t show_country_rel_date~~
~~(struct device *dev, struct device_attribute *attr, char *buf)~~
~~{~~
	~~struct usb_interface *intf = to_usb_interface(dev);~~
	~~struct acm *acm = usb_get_intfdata(intf);~~

	~~return sprintf(buf, "%d", acm->country_rel_date);~~
~~}~~

~~static DEVICE_ATTR(iCountryCodeRelDate, S_IRUGO,~~
~~show_country_rel_date, NULL);~~
~~/*~~
 ~~* Interrupt handlers for various ACM device responses~~
 ~~*/~~

~~/* control interface reports status changes with "interrupt"~~
~~transfers */~~
~~static void acm_ctrl_irq(struct urb *urb)~~
~~{~~
	~~struct acm *acm = urb->context;~~
	~~struct usb_cdc_notification *dr = urb->transfer_buffer;~~
	~~unsigned char *data;~~
	~~int newctrl;~~
	~~int difference;~~
	~~int retval;~~
	~~int status = urb->status;~~
```

```
        ~~switch (status) {~~
        ~~case 0:~~
                ~~/* success */~~
                ~~break;~~
        ~~case -ECONNRESET:~~
        ~~case -ENOENT:~~
        ~~case -ESHUTDOWN:~~
                ~~/* this urb is terminated, clean up */~~
                ~~dev_dbg(&acm->control->dev,~~
                        ~~"%s - urb shutting down with status: %d\n",~~
                        ~~__func__, status);~~
                ~~return;~~
        ~~default:~~
                ~~dev_dbg(&acm->control->dev,~~
                        ~~"%s - nonzero urb status received: %d\n",~~
                        ~~__func__, status);~~
                ~~goto exit;~~
        ~~}~~

        ~~usb_mark_last_busy(acm->dev);~~

        ~~data = (unsigned char *)(dr + 1);~~
        ~~switch (dr->bNotificationType) {~~
        ~~case USB_CDC_NOTIFY_NETWORK_CONNECTION:~~
                ~~dev_dbg(&acm->control->dev,~~
                        ~~"%s - network connection: %d\n", __func__, dr->~~
~~wValue);~~
                ~~break;~~

        ~~case USB_CDC_NOTIFY_SERIAL_STATE:~~
                ~~newctrl = get_unaligned_le16(data);~~

                ~~if (!acm->clocal && (acm->ctrlin & ~newctrl &~~
~~ACM_CTRL_DCD)) {~~
                        ~~dev_dbg(&acm->control->dev,~~
                                ~~"%s - calling hangup\n", __func__);~~
                        ~~tty_port_tty_hangup(&acm->port, false);~~
                ~~}~~

                ~~difference = acm->ctrlin ^ newctrl;~~
                ~~spin_lock(&acm->read_lock);~~
                ~~acm->ctrlin = newctrl;~~
                ~~acm->oldcount = acm->iocount;~~

                ~~if (difference & ACM_CTRL_DSR)~~
                        ~~acm->iocount.dsr++;~~
                ~~if (difference & ACM_CTRL_BRK)~~
                        ~~acm->iocount.brk++;~~
                ~~if (difference & ACM_CTRL_RI)~~
                        ~~acm->iocount.rng++;~~
                ~~if (difference & ACM_CTRL_DCD)~~
```

```
                        ~~acm->iocount.dcd++;~~
                ~~if (difference & ACM_CTRL_FRAMING)~~
                        ~~acm->iocount.frame++;~~
                ~~if (difference & ACM_CTRL_PARITY)~~
                        ~~acm->iocount.parity++;~~
                ~~if (difference & ACM_CTRL_OVERRUN)~~
                        ~~acm->iocount.overrun++;~~
                ~~spin_unlock(&acm->read_lock);~~

                ~~if (difference)~~
                        ~~wake_up_all(&acm->wioctl);~~

                ~~break;~~

        ~~default:~~
                ~~dev_dbg(&acm->control->dev,~~
                        ~~"%s - unknown notification %d received: index %d~~
~~"~~

                        ~~"len %d data0 %d data1 %d\n",~~
                        ~~__func__,~~
                        ~~dr->bNotificationType, dr->wIndex,~~
                        ~~dr->wLength, data[0], data[1]);~~
                ~~break;~~
        ~~}~~
~~exit:~~
        ~~retval = usb_submit_urb(urb, GFP_ATOMIC);~~
        ~~if (retval && retval != -EPERM)~~
                ~~dev_err(&acm->control->dev,~~
                        ~~"%s - usb_submit_urb failed: %d\n", __func__,~~
~~retval);~~
~~}~~

~~static int acm_submit_read_urb(struct acm *acm, int index, gfp_t~~
~~mem_flags)~~
~~{~~
        ~~int res;~~

        ~~if (!test_and_clear_bit(index, &acm->read_urbs_free))~~
                ~~return 0;~~

        ~~res = usb_submit_urb(acm->read_urbs[index], mem_flags);~~
        ~~if (res) {~~
                ~~if (res != -EPERM) {~~
                        ~~dev_err(&acm->data->dev,~~
                                ~~"urb %d failed submission with %d\n",~~
                                ~~index, res);~~
                ~~}~~
                ~~set_bit(index, &acm->read_urbs_free);~~
                ~~return res;~~
        ~~} else {~~
                ~~dev_vdbg(&acm->data->dev, "submitted urb %d\n",~~
~~index);~~
```

```
        ~~}~~

        ~~return 0;~~
~~}~~

~~static int acm_submit_read_urbs(struct acm *acm, gfp_t mem_flags)~~
~~{~~
        ~~int res;~~
        ~~int i;~~

        ~~for (i = 0; i < acm->rx_buflimit; ++i) {~~
                ~~res = acm_submit_read_urb(acm, i, mem_flags);~~
                ~~if (res)~~
                        ~~return res;~~
        ~~}~~

        ~~return 0;~~
~~}~~

~~static void acm_process_read_urb(struct acm *acm, struct urb~~
~~*urb)~~
~~{~~
        ~~if (!urb->actual_length)~~
                ~~return;~~

        ~~tty_insert_flip_string(&acm->port, urb->transfer_buffer,~~
                        ~~urb->actual_length);~~
        ~~tty_flip_buffer_push(&acm->port);~~
~~}~~

~~static void acm_read_bulk_callback(struct urb *urb)~~
~~{~~
        ~~struct acm_rb *rb = urb->context;~~
        ~~struct acm *acm = rb->instance;~~
        ~~unsigned long flags;~~
        ~~int status = urb->status;~~

        ~~dev_vdbg(&acm->data->dev, "got urb %d, len %d, status %d\n",~~
                ~~rb->index, urb->actual_length, status);~~

        ~~set_bit(rb->index, &acm->read_urbs_free);~~

        ~~if (!acm->dev) {~~
                ~~dev_dbg(&acm->data->dev, "%s - disconnected\n",~~
~~__func__);~~
                ~~return;~~
        ~~}~~

        ~~switch (status) {~~
        ~~case 0:~~
                ~~usb_mark_last_busy(acm->dev);~~
                ~~acm_process_read_urb(acm, urb);~~
```

```
            ~~break;~~
      ~~case -EPIPE:~~
            ~~set_bit(EVENT_RX_STALL, &acm->flags);~~
            ~~schedule_work(&acm->work);~~
            ~~return;~~
      ~~case -ENOENT:~~
      ~~case -ECONNRESET:~~
      ~~case -ESHUTDOWN:~~
            ~~dev_dbg(&acm->data->dev,~~
                  ~~"%s - urb shutting down with status: %d\n",~~
                  ~~__func__, status);~~
            ~~return;~~
      ~~default:~~
            ~~dev_dbg(&acm->data->dev,~~
                  ~~"%s - nonzero urb status received: %d\n",~~
                  ~~__func__, status);~~
            ~~break;~~
      ~~}~~

      ~~/*~~
       ~~* Unthrottle may run on another CPU which needs to see~~
~~events~~
       ~~* in the same order. Submission has an implict barrier~~
       ~~*/~~
      ~~smp_mb__before_atomic();~~

      ~~/* throttle device if requested by tty */~~
      ~~spin_lock_irqsave(&acm->read_lock, flags);~~
      ~~acm->throttled = acm->throttle_req;~~
      ~~if (!acm->throttled) {~~
            ~~spin_unlock_irqrestore(&acm->read_lock, flags);~~
            ~~acm_submit_read_urb(acm, rb->index, GFP_ATOMIC);~~
      ~~} else {~~
            ~~spin_unlock_irqrestore(&acm->read_lock, flags);~~
      ~~}~~
~~}~~

/* data interface wrote those outgoing bytes */
static void acm_write_bulk(struct urb *urb)
{
      struct acm_wb *wb = urb->context;
      struct acm *acm = wb->instance;
      unsigned long flags;
      int status = urb->status;

      if (status || (urb->actual_length != urb->
transfer_buffer_length))
            dev_vdbg(&acm->data->dev, "wrote len %d/%d, status %d
\n",
                  urb->actual_length,
                  urb->transfer_buffer_length,
                  status);
```

```
        spin_lock_irqsave(&acm->write_lock, flags);
        acm_write_done(acm, wb);
        spin_unlock_irqrestore(&acm->write_lock, flags);
        set_bit(EVENT_TTY_WAKEUP, &acm->flags);
        schedule_work(&acm->work);
}

~~static void acm_softint(struct work_struct *work)~~
~~{~~
        ~~int i;~~
        ~~struct acm *acm = container_of(work, struct acm, work);~~

        ~~if (test_bit(EVENT_RX_STALL, &acm->flags)) {~~
                ~~if (!(usb_autopm_get_interface(acm->data))) {~~
                        ~~for (i = 0; i < acm->rx_buflimit; i++)~~
                                ~~usb_kill_urb(acm->read_urbs[i]);~~
                        ~~usb_clear_halt(acm->dev, acm->in);~~
                        ~~acm_submit_read_urbs(acm, GFP_KERNEL);~~
                        ~~usb_autopm_put_interface(acm->data);~~
                ~~}~~
                ~~clear_bit(EVENT_RX_STALL, &acm->flags);~~
        ~~}~~

        ~~if (test_bit(EVENT_TTY_WAKEUP, &acm->flags)) {~~
                ~~tty_port_tty_wakeup(&acm->port);~~
                ~~clear_bit(EVENT_TTY_WAKEUP, &acm->flags);~~
        ~~}~~
~~}~~

/*
 * TTY handlers              important
 */

static int acm_tty_install(struct tty_driver *driver, struct
tty_struct *tty)
{
        struct acm *acm;
        int retval;

        acm = acm_get_by_minor(tty->index);
        if (!acm)
                return -ENODEV;      no device

        retval = tty_standard_install(driver, tty);
        if (retval)
                goto error_init_termios;

        tty->driver_data = acm;

        return 0;
```

```
error_init_termios:
	tty_port_put(&acm->port);
	return retval;
}

static int acm_tty_open(struct tty_struct *tty, struct file
*filp)
{
	struct acm *acm = tty->driver_data;

	return tty_port_open(&acm->port, tty, filp);
}

~~static void acm_port_dtr_rts(struct tty_port *port, int raise)~~
~~{~~
	~~struct acm *acm = container_of(port, struct acm, port);~~
	~~int val;~~
	~~int res;~~

	~~if (raise)~~
		~~val = ACM_CTRL_DTR | ACM_CTRL_RTS;~~
	~~else~~
		~~val = 0;~~

	~~/* FIXME: add missing ctrlout locking throughout driver */~~
	~~acm->ctrlout = val;~~

	~~res = acm_set_control(acm, val);~~
	~~if (res && (acm->ctrl_caps & USB_CDC_CAP_LINE))~~
		~~dev_err(&acm->control->dev, "failed to set dtr/rts~~
~~\n");~~
~~}~~

static int acm_port_activate(struct tty_port *port, struct
tty_struct *tty)
{
	struct acm *acm = container_of(port, struct acm, port);
	int retval = -ENODEV;
	int i;

	mutex_lock(&acm->mutex);
	if (acm->disconnected)
		goto disconnected;

	retval = usb_autopm_get_interface(acm->control);
	if (retval)
		goto error_get_interface;

	/*
	 * FIXME: Why do we need this? Allocating 64K of physically
contiguous
	 * memory is really nasty...
```

```
         */
        set_bit(TTY_NO_WRITE_SPLIT, &tty->flags);
        acm->control->needs_remote_wakeup = 1;

        acm->ctrlurb->dev = acm->dev;
        retval = usb_submit_urb(acm->ctrlurb, GFP_KERNEL);
        if (retval) {
             dev_err(&acm->control->dev,
                  "%s - usb_submit_urb(ctrl irq) failed\n",
__func__);
             goto error_submit_urb;
        }

        acm_tty_set_termios(tty, NULL);

        ~~/*~~
         ~~* Unthrottle device in case the TTY was closed while~~
~~throttled.~~
         ~~*/~~
        ~~spin_lock_irq(&acm->read_lock);~~
        ~~acm->throttled = 0;~~
        ~~acm->throttle_req = 0;~~
        ~~spin_unlock_irq(&acm->read_lock);~~

        retval = acm_submit_read_urbs(acm, GFP_KERNEL);
        if (retval)
             goto error_submit_read_urbs;

        usb_autopm_put_interface(acm->control);

        mutex_unlock(&acm->mutex);

        return 0;

error_submit_read_urbs:
        for (i = 0; i < acm->rx_buflimit; i++)
             usb_kill_urb(acm->read_urbs[i]);
        usb_kill_urb(acm->ctrlurb);
error_submit_urb:
        usb_autopm_put_interface(acm->control);
error_get_interface:
disconnected:
        mutex_unlock(&acm->mutex);

        return usb_translate_errors(retval);
}

static void acm_port_destruct(struct tty_port *port)
{
        struct acm *acm = container_of(port, struct acm, port);

        acm_release_minor(acm);
```

```
	usb_put_intf(acm->control);
	kfree(acm->country_codes);
	kfree(acm);
}

static void acm_port_shutdown(struct tty_port *port)
{
	struct acm *acm = container_of(port, struct acm, port);
	struct urb *urb;
	struct acm_wb *wb;

	/*
	 * Need to grab write lock to prevent race with resume, but
no need to
	 * hold it due to the tty-port initialised flag.
	 */
	spin_lock_irq(&acm->write_lock);
	spin_unlock_irq(&acm->write_lock);

	usb_autopm_get_interface_no_resume(acm->control);
	acm->control->needs_remote_wakeup = 0;
	usb_autopm_put_interface(acm->control);

	for (;;) {
		urb = usb_get_from_anchor(&acm->delayed);
		if (!urb)
			break;
		wb = urb->context;
		wb->use = 0;
		usb_autopm_put_interface_async(acm->control);
	}

	acm_kill_urbs(acm);
}

static void acm_tty_cleanup(struct tty_struct *tty)
{
	struct acm *acm = tty->driver_data;

	tty_port_put(&acm->port);
}

static void acm_tty_hangup(struct tty_struct *tty)
{
	struct acm *acm = tty->driver_data;

	tty_port_hangup(&acm->port);
}

static void acm_tty_close(struct tty_struct *tty, struct file
*filp)
{
	struct acm *acm = tty->driver_data;

	tty_port_close(&acm->port, tty, filp);
}

static int acm_tty_write(struct tty_struct *tty,
					const unsigned char *buf, int count)
{
	struct acm *acm = tty->driver_data;
	int stat;
	unsigned long flags;
	int wbn;
	struct acm_wb *wb;

	if (!count)
		return 0;

	dev_vdbg(&acm->data->dev, "%d bytes from tty layer\n",
count);

	spin_lock_irqsave(&acm->write_lock, flags);
	wbn = acm_wb_alloc(acm);
	if (wbn < 0) {
		spin_unlock_irqrestore(&acm->write_lock, flags);
		return 0;
	}
	wb = &acm->wb[wbn];

	if (!acm->dev) {
		wb->use = 0;
		spin_unlock_irqrestore(&acm->write_lock, flags);
		return -ENODEV;
	}

	count = (count > acm->writesize) ? acm->writesize : count;
	dev_vdbg(&acm->data->dev, "writing %d bytes\n", count);
	memcpy(wb->buf, buf, count);
	wb->len = count;

	stat = usb_autopm_get_interface_async(acm->control);
	if (stat) {
		wb->use = 0;
		spin_unlock_irqrestore(&acm->write_lock, flags);
		return stat;
	}

	if (acm->susp_count) {
		if (acm->putbuffer) {
			/* now to preserve order */
			usb_anchor_urb(acm->putbuffer->urb, &acm->
delayed);
			acm->putbuffer = NULL;
		}
		usb_anchor_urb(wb->urb, &acm->delayed);
		spin_unlock_irqrestore(&acm->write_lock, flags);
		return count;
	} else {
		if (acm->putbuffer) {
			/* at this point there is no good way to handle
errors */
			acm_start_wb(acm, acm->putbuffer);
			acm->putbuffer = NULL;
		}
	}

	stat = acm_start_wb(acm, wb);
	spin_unlock_irqrestore(&acm->write_lock, flags);

	if (stat < 0)
		return stat;
	return count;
}

static void acm_tty_flush_chars(struct tty_struct *tty)
{
	struct acm *acm = tty->driver_data;
	struct acm_wb *cur = acm->putbuffer;
	int err;
	unsigned long flags;

	if (!cur) /* nothing to do */
		return;

	acm->putbuffer = NULL;
	err = usb_autopm_get_interface_async(acm->control);
	spin_lock_irqsave(&acm->write_lock, flags);
	if (err < 0) {
		cur->use = 0;
		acm->putbuffer = cur;
		goto out;
	}

	if (acm->susp_count)
		usb_anchor_urb(cur->urb, &acm->delayed);
	else
		acm_start_wb(acm, cur);
out:
	spin_unlock_irqrestore(&acm->write_lock, flags);
	return;
}

static int acm_tty_put_char(struct tty_struct *tty, unsigned char
ch)
{
      struct acm *acm = tty->driver_data;
      struct acm_wb *cur;
      int wbn;
      unsigned long flags;

overflow:
      cur = acm->putbuffer;
      if (!cur) {
            spin_lock_irqsave(&acm->write_lock, flags);
            wbn = acm_wb_alloc(acm);
            if (wbn >= 0) {
                  cur = &acm->wb[wbn];
                  acm->putbuffer = cur;
            }
            spin_unlock_irqrestore(&acm->write_lock, flags);
            if (!cur)
                  return 0;
      }

      if (cur->len == acm->writesize) {
            acm_tty_flush_chars(tty);
            goto overflow;
      }

      cur->buf[cur->len++] = ch;
      return 1;
}

static int acm_tty_write_room(struct tty_struct *tty)
{
      struct acm *acm = tty->driver_data;
      /*
       * Do not let the line discipline to know that we have a
reserve,
       * or it might get too enthusiastic.
       */
      return acm_wb_is_avail(acm) ? acm->writesize : 0;
}

static int acm_tty_chars_in_buffer(struct tty_struct *tty)
{
      struct acm *acm = tty->driver_data;
      /*
       * if the device was unplugged then any remaining characters
fell out
       * of the connector ;)
       */
      if (acm->disconnected)
            return 0;
      /*
       * This is inaccurate (overcounts), but it works.
       */
```

```
	return (ACM_NW - acm_wb_is_avail(acm)) * acm->writesize;
}

static void acm_tty_throttle(struct tty_struct *tty)
{
	struct acm *acm = tty->driver_data;

	spin_lock_irq(&acm->read_lock);
	acm->throttle_req = 1;
	spin_unlock_irq(&acm->read_lock);
}

static void acm_tty_unthrottle(struct tty_struct *tty)
{
	struct acm *acm = tty->driver_data;
	unsigned int was_throttled;

	spin_lock_irq(&acm->read_lock);
	was_throttled = acm->throttled;
	acm->throttled = 0;
	acm->throttle_req = 0;
	spin_unlock_irq(&acm->read_lock);

	if (was_throttled)
		acm_submit_read_urbs(acm, GFP_KERNEL);
}

static int acm_tty_break_ctl(struct tty_struct *tty, int state)
{
	struct acm *acm = tty->driver_data;
	int retval;

	retval = acm_send_break(acm, state ? 0xffff : 0);
	if (retval < 0)
		dev_dbg(&acm->control->dev,
			"%s - send break failed\n", __func__);
	return retval;
}
```

~~static int acm_tty_tiocmget(struct tty_struct *tty)~~
~~{~~
~~struct acm *acm = tty->driver_data;~~

~~return (acm->ctrlout & ACM_CTRL_DTR ? TIOCM_DTR : 0) |~~
~~(acm->ctrlout & ACM_CTRL_RTS ? TIOCM_RTS : 0) |~~
~~(acm->ctrlin & ACM_CTRL_DSR ? TIOCM_DSR : 0) |~~
~~(acm->ctrlin & ACM_CTRL_RI ? TIOCM_RI : 0) |~~
~~(acm->ctrlin & ACM_CTRL_DCD ? TIOCM_CD : 0) |~~
~~TIOCM_CTS;~~
~~}~~

~~static int acm_tty_tiocmset(struct tty_struct *tty,~~

```
                    ~~unsigned int set, unsigned int clear)~~
~~{~~
      ~~struct acm *acm = tty->driver_data;~~
      ~~unsigned int newctrl;~~

      ~~newctrl = acm->ctrlout;~~
      ~~set = (set & TIOCM_DTR ? ACM_CTRL_DTR : 0) |~~
                          ~~(set & TIOCM_RTS ? ACM_CTRL_RTS : 0);~~
      ~~clear = (clear & TIOCM_DTR ? ACM_CTRL_DTR : 0) |~~
                          ~~(clear & TIOCM_RTS ? ACM_CTRL_RTS :~~
~~0);~~

      ~~newctrl = (newctrl & ~clear) | set;~~

      ~~if (acm->ctrlout == newctrl)~~
            ~~return 0;~~
      ~~return acm_set_control(acm, acm->ctrlout = newctrl);~~
~~}~~

static int get_serial_info(struct acm *acm, struct serial_struct
__user *info)
{
      struct serial_struct tmp;

      memset(&tmp, 0, sizeof(tmp));
      tmp.xmit_fifo_size = acm->writesize;
      tmp.baud_base = le32_to_cpu(acm->line.dwDTERate);
      tmp.close_delay = acm->port.close_delay / 10;
      tmp.closing_wait = acm->port.closing_wait ==
ASYNC_CLOSING_WAIT_NONE ?
                        ASYNC_CLOSING_WAIT_NONE :
                        acm->port.closing_wait / 10;

      if (copy_to_user(info, &tmp, sizeof(tmp)))
            return -EFAULT;
      else
            return 0;
}

static int set_serial_info(struct acm *acm,
                        struct serial_struct __user *newinfo)
{
      struct serial_struct new_serial;
      unsigned int closing_wait, close_delay;
      int retval = 0;

      if (copy_from_user(&new_serial, newinfo,
sizeof(new_serial)))
            return -EFAULT;

      close_delay = new_serial.close_delay * 10;
      closing_wait = new_serial.closing_wait ==
ASYNC_CLOSING_WAIT_NONE ?
                ASYNC_CLOSING_WAIT_NONE : new_serial.closing_wait
* 10;

	mutex_lock(&acm->port.mutex);

	if (!capable(CAP_SYS_ADMIN)) {
		if ((close_delay != acm->port.close_delay) ||
		    (closing_wait != acm->port.closing_wait))
			retval = -EPERM;
		else
			retval = -EOPNOTSUPP;
	} else {
		acm->port.close_delay  = close_delay;
		acm->port.closing_wait = closing_wait;
	}

	mutex_unlock(&acm->port.mutex);
	return retval;
}

static int wait_serial_change(struct acm *acm, unsigned long arg)
{
	int rv = 0;
	DECLARE_WAITQUEUE(wait, current);
	struct async_icount old, new;

	do {
		spin_lock_irq(&acm->read_lock);
		old = acm->oldcount;
		new = acm->iocount;
		acm->oldcount = new;
		spin_unlock_irq(&acm->read_lock);

		if ((arg & TIOCM_DSR) &&
			old.dsr != new.dsr)
			break;
		if ((arg & TIOCM_CD)  &&
			old.dcd != new.dcd)
			break;
		if ((arg & TIOCM_RI) &&
			old.rng != new.rng)
			break;

		add_wait_queue(&acm->wioctl, &wait);
		set_current_state(TASK_INTERRUPTIBLE);
		schedule();
		remove_wait_queue(&acm->wioctl, &wait);
		if (acm->disconnected) {
			if (arg & TIOCM_CD)
				break;
			else
```

```
                        rv = -ENODEV;
                } else {
                        if (signal_pending(current))
                                rv = -ERESTARTSYS;
                }
        } while (!rv);



        return rv;
}

static int acm_tty_get_icount(struct tty_struct *tty,
                                        struct serial_icounter_struct *icount)
{
        struct acm *acm = tty->driver_data;

        icount->dsr = acm->iocount.dsr;
        icount->rng = acm->iocount.rng;
        icount->dcd = acm->iocount.dcd;
        icount->frame = acm->iocount.frame;
        icount->overrun = acm->iocount.overrun;
        icount->parity = acm->iocount.parity;
        icount->brk = acm->iocount.brk;

        return 0;
}

static int acm_tty_ioctl(struct tty_struct *tty,
                                        unsigned int cmd, unsigned long arg)
{
        struct acm *acm = tty->driver_data;
        int rv = -ENOIOCTLCMD;

        switch (cmd) {
        case TIOCGSERIAL: /* gets serial port data */
                rv = get_serial_info(acm, (struct serial_struct __user
*) arg);
                ~~break;~~
        ~~case TIOCSSERIAL~~:
                rv = set_serial_info(acm, (struct serial_struct __user
*) arg);
                break;
        case TIOCMIWAIT:
                rv = usb_autopm_get_interface(acm->control);
                if (rv < 0) {
                        rv = -EIO;
                        break;
                }
                rv = wait_serial_change(acm, arg);
                usb_autopm_put_interface(acm->control);
                break;
```

```
	}

	return rv;
}

static void acm_tty_set_termios(struct tty_struct *tty,
						struct ktermios *termios_old)
{
	struct acm *acm = tty->driver_data;
	struct ktermios *termios = &tty->termios;
	struct usb_cdc_line_coding newline;
	int newctrl = acm->ctrlout;

	newline.dwDTERate = cpu_to_le32(tty_get_baud_rate(tty));
	newline.bCharFormat = termios->c_cflag & CSTOPB ? 2 : 0;
	newline.bParityType = termios->c_cflag & PARENB ?
				(termios->c_cflag & PARODD ? 1 : 2) +
				(termios->c_cflag & CMSPAR ? 2 : 0) : 0;
	switch (termios->c_cflag & CSIZE) {
	case CS5:
		newline.bDataBits = 5;
		break;
	case CS6:
		newline.bDataBits = 6;
		break;
	case CS7:
		newline.bDataBits = 7;
		break;
	case CS8:
	default:
		newline.bDataBits = 8;
		break;
	}
	/* FIXME: Needs to clear unsupported bits in the termios */
	acm->clocal = ((termios->c_cflag & CLOCAL) != 0);

	if (C_BAUD(tty) == B0) {
		newline.dwDTERate = acm->line.dwDTERate;
		newctrl &= ~ACM_CTRL_DTR;
	} else if (termios_old && (termios_old->c_cflag & CBAUD) ==
B0) {
		newctrl |=  ACM_CTRL_DTR;
	}

	if (newctrl != acm->ctrlout)
		acm_set_control(acm, acm->ctrlout = newctrl);

	if (memcmp(&acm->line, &newline, sizeof newline)) {
		memcpy(&acm->line, &newline, sizeof newline);
		dev_dbg(&acm->control->dev, "%s - set line: %d %d %d %
d\n",
			__func__,
                le32_to_cpu(newline.dwDTERate),
                newline.bCharFormat, newline.bParityType,
                newline.bDataBits);
            acm_set_line(acm, &acm->line);
        }
}

static const struct tty_port_operations acm_port_ops = {
        .dtr_rts = acm_port_dtr_rts,
        .shutdown = acm_port_shutdown,
        .activate = acm_port_activate,
        .destruct = acm_port_destruct,
};

/*                                                        important
 * USB probe and disconnect routines.
 */

/* Little helpers: write/read buffers free */
static void acm_write_buffers_free(struct acm *acm)
{
        int i;
        struct acm_wb *wb;

        for (wb = &acm->wb[0], i = 0; i < ACM_NW; i++, wb++)
            usb_free_coherent(acm->dev, acm->writesize, wb->buf,
wb->dmah);
}

static void acm_read_buffers_free(struct acm *acm)
{
        int i;

        for (i = 0; i < acm->rx_buflimit; i++)
            usb_free_coherent(acm->dev, acm->readsize,
                    acm->read_buffers[i].base, acm->
read_buffers[i].dma);
}

/* Little helper: write buffers allocate */
static int acm_write_buffers_alloc(struct acm *acm)
{
        int i;
        struct acm_wb *wb;

        for (wb = &acm->wb[0], i = 0; i < ACM_NW; i++, wb++) {
            wb->buf = usb_alloc_coherent(acm->dev, acm->writesize,
GFP_KERNEL,
                &wb->dmah);
            if (!wb->buf) {
                while (i != 0) {
                    --i;
```

```
                        --wb;
                        usb_free_coherent(acm->dev, acm->writesize,
                            wb->buf, wb->dmah);
                    }
                    return -ENOMEM;
                }
            }
            return 0;
}
// xác ??nh Endpoint, c?p phát b? nh?
static int acm_probe(struct usb_interface *intf,
                const struct usb_device_id *id)
{
        struct usb_cdc_union_desc *union_header = NULL;
        struct usb_cdc_call_mgmt_descriptor *cmgmd = NULL;
        unsigned char *buffer = intf->altsetting->extra;
        int buflen = intf->altsetting->extralen;
        struct usb_interface *control_interface;
        struct usb_interface *data_interface;
        struct usb_endpoint_descriptor *epctrl = NULL;
        struct usb_endpoint_descriptor *epread = NULL;
        struct usb_endpoint_descriptor *epwrite = NULL;
        struct usb_device *usb_dev = interface_to_usbdev(intf);
        struct usb_cdc_parsed_header h;
        struct acm *acm;
        int minor;
        int ctrlsize, readsize;
        u8 *buf;
        int call_intf_num = -1;
        int data_intf_num = -1;
        unsigned long quirks;
        int num_rx_buf;
        int i;
        int combined_interfaces = 0;
        struct device *tty_dev;
        int rv = -ENOMEM;

        /* normal quirks */
        quirks = (unsigned long)id->driver_info;

        if (quirks == IGNORE_DEVICE)
            return -ENODEV;

        memset(&h, 0x00, sizeof(struct usb_cdc_parsed_header));

        num_rx_buf = (quirks == SINGLE_RX_URB) ? 1 : ACM_NR;

        /* handle quirks deadly to normal probing*/
        if (quirks == NO_UNION_NORMAL) {
            data_interface = usb_ifnum_to_if(usb_dev, 1);
            control_interface = usb_ifnum_to_if(usb_dev, 0);
            /* we would crash */
            if (!data_interface || !control_interface)
                  return -ENODEV;
            goto skip_normal_probe;
      }

      /* normal probing*/
      if (!buffer) {
            dev_err(&intf->dev, "Weird descriptor references\n");
            return -EINVAL;
      }

      if (!intf->cur_altsetting)
            return -EINVAL;

      if (!buflen) {
            if (intf->cur_altsetting->endpoint &&
                        intf->cur_altsetting->endpoint->extralen &&
                        intf->cur_altsetting->endpoint->extra) {
                  dev_dbg(&intf->dev,
                        "Seeking extra descriptors on endpoint\n");
                  buflen = intf->cur_altsetting->endpoint->
extralen;
                  buffer = intf->cur_altsetting->endpoint->extra;
            } else {
                  dev_err(&intf->dev,
                        "Zero length descriptor references\n");
                  return -EINVAL;
            }
      }

      cdc_parse_cdc_header(&h, intf, buffer, buflen);
      union_header = h.usb_cdc_union_desc;
      cmgmd = h.usb_cdc_call_mgmt_descriptor;
      if (cmgmd)
            call_intf_num = cmgmd->bDataInterface;

      if (!union_header) {
            if (call_intf_num > 0) {
                  dev_dbg(&intf->dev, "No union descriptor, using
call management descriptor\n");
                  /* quirks for Droids MuIn LCD */
                  if (quirks & NO_DATA_INTERFACE) {
                        data_interface = usb_ifnum_to_if(usb_dev,
0);
                  } else {
                        data_intf_num = call_intf_num;
                        data_interface = usb_ifnum_to_if(usb_dev,
data_intf_num);
                  }
                  control_interface = intf;
            } else {
                  if (intf->cur_altsetting->desc.bNumEndpoints !=
3) {
                      dev_dbg(&intf->dev,"No union descriptor,
giving up\n");
                      return -ENODEV;
                } else {
                      dev_warn(&intf->dev,"No union descriptor,
testing for castrated device\n");
                      combined_interfaces = 1;
                      control_interface = data_interface = intf;
                      goto look_for_collapsed_interface;
                }
           }
      } else {
           data_intf_num = union_header->bSlaveInterface0;
           control_interface = usb_ifnum_to_if(usb_dev,
union_header->bMasterInterface0);
           data_interface = usb_ifnum_to_if(usb_dev,
data_intf_num);
      }

      if (!control_interface || !data_interface) {
           dev_dbg(&intf->dev, "no interfaces\n");
           return -ENODEV;
      }
      if (!data_interface->cur_altsetting || !control_interface->
cur_altsetting)
           return -ENODEV;

      if (data_intf_num != call_intf_num)
           dev_dbg(&intf->dev, "Separate call control interface.
That is not fully supported.\n");

      if (control_interface == data_interface) {
           /* some broken devices designed for windows work this
way */
           dev_warn(&intf->dev,"Control and data interfaces are
not separated!\n");
           combined_interfaces = 1;
           /* a popular other OS doesn't use it */
           quirks |= NO_CAP_LINE;
           if (data_interface->cur_altsetting->
desc.bNumEndpoints != 3) {
                dev_err(&intf->dev, "This needs exactly 3
endpoints\n");
                return -EINVAL;
           }
look_for_collapsed_interface:
           for (i = 0; i < 3; i++) {
                struct usb_endpoint_descriptor *ep;
                ep = &data_interface->cur_altsetting->
endpoint[i].desc;
```

```
                if (usb_endpoint_is_int_in(ep))
                     epctrl = ep;
                else if (usb_endpoint_is_bulk_out(ep))
                     epwrite = ep;
                else if (usb_endpoint_is_bulk_in(ep))
                     epread = ep;
                else
                     return -EINVAL;
           }
           if (!epctrl || !epread || !epwrite)
                return -ENODEV;
           else
                goto made_compressed_probe;
      }

skip_normal_probe:

      /*workaround for switched interfaces */
      if (data_interface->cur_altsetting->desc.bInterfaceClass
                                 != CDC_DATA_INTERFACE_TYPE) {
           if (control_interface->cur_altsetting->
desc.bInterfaceClass
                                 == CDC_DATA_INTERFACE_TYPE) {
                dev_dbg(&intf->dev,
                     "Your device has switched interfaces.\n");
                swap(control_interface, data_interface);
           } else {
                return -EINVAL;
           }
      }

      /* Accept probe requests only for the control interface */
      if (!combined_interfaces && intf != control_interface)
           return -ENODEV;

      if (!combined_interfaces &&
usb_interface_claimed(data_interface)) {
           /* valid in this context */
           dev_dbg(&intf->dev, "The data interface isn't
available\n");
           return -EBUSY;
      }


      if (data_interface->cur_altsetting->desc.bNumEndpoints < 2
||
          control_interface->cur_altsetting->desc.bNumEndpoints ==
0)
           return -EINVAL;

      epctrl = &control_interface->cur_altsetting->
endpoint[0].desc;
```

```
	epread = &data_interface->cur_altsetting->endpoint[0].desc;
	epwrite = &data_interface->cur_altsetting->endpoint[1].desc;


	/* workaround for switched endpoints */
	if (!usb_endpoint_dir_in(epread)) {
		/* descriptors are swapped */
		dev_dbg(&intf->dev,
			"The data interface has switched endpoints\n");
		swap(epread, epwrite);
	}
made_compressed_probe:
	dev_dbg(&intf->dev, "interfaces are valid\n");

	acm = kzalloc(sizeof(struct acm), GFP_KERNEL);
	if (acm == NULL)
		goto alloc_fail;

	minor = acm_alloc_minor(acm);
	if (minor < 0)
		goto alloc_fail1;

	ctrlsize = usb_endpoint_maxp(epctrl);
	readsize = usb_endpoint_maxp(epread) *
				(quirks == SINGLE_RX_URB ? 1 : 2);
	acm->combined_interfaces = combined_interfaces;
	acm->writesize = usb_endpoint_maxp(epwrite) * 20;
	acm->control = control_interface;
	acm->data = data_interface;
	acm->minor = minor;
	acm->dev = usb_dev;
	if (h.usb_cdc_acm_descriptor)
		acm->ctrl_caps = h.usb_cdc_acm_descriptor->
bmCapabilities;
	if (quirks & NO_CAP_LINE)
		acm->ctrl_caps &= ~USB_CDC_CAP_LINE;
	acm->ctrlsize = ctrlsize;
	acm->readsize = readsize;
	acm->rx_buflimit = num_rx_buf;
	INIT_WORK(&acm->work, acm_softint);
	init_waitqueue_head(&acm->wioctl);
	spin_lock_init(&acm->write_lock);
	spin_lock_init(&acm->read_lock);
	mutex_init(&acm->mutex);
	if (usb_endpoint_xfer_int(epread)) {
		acm->bInterval = epread->bInterval;
		acm->in = usb_rcvintpipe(usb_dev, epread->
bEndpointAddress);
	} else {
		acm->in = usb_rcvbulkpipe(usb_dev, epread->
bEndpointAddress);
	}
	if (usb_endpoint_xfer_int(epwrite))
		acm->out = usb_sndintpipe(usb_dev, epwrite->
bEndpointAddress);
	else
		acm->out = usb_sndbulkpipe(usb_dev, epwrite->
bEndpointAddress);
	tty_port_init(&acm->port);
	acm->port.ops = &acm_port_ops;
	init_usb_anchor(&acm->delayed);
	acm->quirks = quirks;

	buf = usb_alloc_coherent(usb_dev, ctrlsize, GFP_KERNEL,
&acm->ctrl_dma);
	if (!buf)
		goto alloc_fail2;
	acm->ctrl_buffer = buf;

	if (acm_write_buffers_alloc(acm) < 0)
		goto alloc_fail4;

	acm->ctrlurb = usb_alloc_urb(0, GFP_KERNEL);
	if (!acm->ctrlurb)
		goto alloc_fail5;

	for (i = 0; i < num_rx_buf; i++) {
		struct acm_rb *rb = &(acm->read_buffers[i]);
		struct urb *urb;

		rb->base = usb_alloc_coherent(acm->dev, readsize,
GFP_KERNEL,
								&rb->dma);
		if (!rb->base)
			goto alloc_fail6;
		rb->index = i;
		rb->instance = acm;

		urb = usb_alloc_urb(0, GFP_KERNEL);
		if (!urb)
			goto alloc_fail6;

		urb->transfer_flags |= URB_NO_TRANSFER_DMA_MAP;
		urb->transfer_dma = rb->dma;
		if (usb_endpoint_xfer_int(epread))
			usb_fill_int_urb(urb, acm->dev, acm->in, rb->
base,
					 acm->readsize,
					 acm_read_bulk_callback, rb,
					 acm->bInterval);
		else
			usb_fill_bulk_urb(urb, acm->dev, acm->in, rb->
base,
					  acm->readsize,
                                acm_read_bulk_callback, rb);

            acm->read_urbs[i] = urb;
            __set_bit(i, &acm->read_urbs_free);
      }
      for (i = 0; i < ACM_NW; i++) {
            struct acm_wb *snd = &(acm->wb[i]);

            snd->urb = usb_alloc_urb(0, GFP_KERNEL);
            if (snd->urb == NULL)
                  goto alloc_fail7;

            if (usb_endpoint_xfer_int(epwrite))
                  usb_fill_int_urb(snd->urb, usb_dev, acm->out,
                        NULL, acm->writesize, acm_write_bulk, snd,
epwrite->bInterval);
            else
                  usb_fill_bulk_urb(snd->urb, usb_dev, acm->out,
                        NULL, acm->writesize, acm_write_bulk, snd);
            snd->urb->transfer_flags |= URB_NO_TRANSFER_DMA_MAP;
            if (quirks & SEND_ZERO_PACKET)
                  snd->urb->transfer_flags |= URB_ZERO_PACKET;
            snd->instance = acm;
      }

      usb_set_intfdata(intf, acm);

      i = device_create_file(&intf->dev,
&dev_attr_bmCapabilities);
      if (i < 0)
            goto alloc_fail7;

      if (h.usb_cdc_country_functional_desc) { /* export the
country data */
            struct usb_cdc_country_functional_desc * cfd =
                              h.usb_cdc_country_functional_desc;

            acm->country_codes = kmalloc(cfd->bLength - 4,
GFP_KERNEL);
            if (!acm->country_codes)
                  goto skip_countries;
            acm->country_code_size = cfd->bLength - 4;
            memcpy(acm->country_codes, (u8 *)&cfd->wCountyCode0,
                                    cfd->bLength - 4);
            acm->country_rel_date = cfd->iCountryCodeRelDate;

            i = device_create_file(&intf->dev,
&dev_attr_wCountryCodes);
            if (i < 0) {
                  kfree(acm->country_codes);
                  acm->country_codes = NULL;
                  acm->country_code_size = 0;
```

```
                goto skip_countries;
            }

            i = device_create_file(&intf->dev,
                            &dev_attr_iCountryCodeRelDate);
            if (i < 0) {
                device_remove_file(&intf->dev,
&dev_attr_wCountryCodes);
                kfree(acm->country_codes);
                acm->country_codes = NULL;
                acm->country_code_size = 0;
                goto skip_countries;
            }
        }

skip_countries:
        usb_fill_int_urb(acm->ctrlurb, usb_dev,
                    usb_rcvintpipe(usb_dev, epctrl->
bEndpointAddress),
                    acm->ctrl_buffer, ctrlsize, acm_ctrl_irq, acm,
                    /* works around buggy devices */
                    epctrl->bInterval ? epctrl->bInterval : 16);
        acm->ctrlurb->transfer_flags |= URB_NO_TRANSFER_DMA_MAP;
        acm->ctrlurb->transfer_dma = acm->ctrl_dma;

        dev_info(&intf->dev, "ttyACM%d: USB ACM device\n", minor);

        acm->line.dwDTERate = cpu_to_le32(9600);
        acm->line.bDataBits = 8;
        acm_set_line(acm, &acm->line);

        usb_driver_claim_interface(&acm_driver, data_interface,
acm);
        usb_set_intfdata(data_interface, acm);

        usb_get_intf(control_interface);
        tty_dev = tty_port_register_device(&acm->port,
acm_tty_driver, minor,
                &control_interface->dev);
        if (IS_ERR(tty_dev)) {
            rv = PTR_ERR(tty_dev);
            goto alloc_fail8;
        }

        if (quirks & CLEAR_HALT_CONDITIONS) {
            usb_clear_halt(usb_dev, acm->in);
            usb_clear_halt(usb_dev, acm->out);
        }

        return 0;
alloc_fail8:
        if (acm->country_codes) {
```

```
			device_remove_file(&acm->control->dev,
					&dev_attr_wCountryCodes);
			device_remove_file(&acm->control->dev,
					&dev_attr_iCountryCodeRelDate);
			kfree(acm->country_codes);
		}
		device_remove_file(&acm->control->dev,
&dev_attr_bmCapabilities);
alloc_fail7:
		usb_set_intfdata(intf, NULL);
		for (i = 0; i < ACM_NW; i++)
			usb_free_urb(acm->wb[i].urb);
alloc_fail6:
		for (i = 0; i < num_rx_buf; i++)
			usb_free_urb(acm->read_urbs[i]);
		acm_read_buffers_free(acm);
		usb_free_urb(acm->ctrlurb);
alloc_fail5:
		acm_write_buffers_free(acm);
alloc_fail4:
		usb_free_coherent(usb_dev, ctrlsize, acm->ctrl_buffer, acm->
ctrl_dma);
alloc_fail2:
		acm_release_minor(acm);
alloc_fail1:
		kfree(acm);
alloc_fail:
		return rv;
}

static void acm_disconnect(struct usb_interface *intf)
{
		struct acm *acm = usb_get_intfdata(intf);
		struct tty_struct *tty;

		/* sibling interface is already cleaning up */
		if (!acm)
			return;

		mutex_lock(&acm->mutex);
		acm->disconnected = true;
		if (acm->country_codes) {
			device_remove_file(&acm->control->dev,
					&dev_attr_wCountryCodes);
			device_remove_file(&acm->control->dev,
					&dev_attr_iCountryCodeRelDate);
		}
		wake_up_all(&acm->wioctl);
		device_remove_file(&acm->control->dev,
&dev_attr_bmCapabilities);
		usb_set_intfdata(acm->control, NULL);
		usb_set_intfdata(acm->data, NULL);
```

```
	mutex_unlock(&acm->mutex);

	tty = tty_port_tty_get(&acm->port);
	if (tty) {
		tty_vhangup(tty);
		tty_kref_put(tty);
	}

	acm_kill_urbs(acm);
	cancel_work_sync(&acm->work);

	tty_unregister_device(acm_tty_driver, acm->minor);

	acm_write_buffers_free(acm);
	usb_free_coherent(acm->dev, acm->ctrlsize, acm->ctrl_buffer,
acm->ctrl_dma);
	acm_read_buffers_free(acm);

	if (!acm->combined_interfaces)
		usb_driver_release_interface(&acm_driver, intf ==
acm->control ?
					acm->data : acm->control);

	tty_port_put(&acm->port);
}

~~#ifdef CONFIG_PM~~
~~static int acm_suspend(struct usb_interface *intf, pm_message_t~~
~~message)~~
~~{~~
	~~struct acm *acm = usb_get_intfdata(intf);~~
	~~int cnt;~~

	~~spin_lock_irq(&acm->write_lock);~~
	~~if (PMSG_IS_AUTO(message)) {~~
		~~if (acm->transmitting) {~~
			~~spin_unlock_irq(&acm->write_lock);~~
			~~return -EBUSY;~~
		~~}~~
	~~}~~
	~~cnt = acm->susp_count++;~~
	~~spin_unlock_irq(&acm->write_lock);~~

	~~if (cnt)~~
		~~return 0;~~

	~~acm_kill_urbs(acm);~~
	~~cancel_work_sync(&acm->work);~~

	~~return 0;~~
~~}~~
```

```
~~static int acm_resume(struct usb_interface *intf)~~
~~{~~
	~~struct acm *acm = usb_get_intfdata(intf);~~
	~~struct urb *urb;~~
	~~int rv = 0;~~

	~~spin_lock_irq(&acm->write_lock);~~

	~~if (--acm->susp_count)~~
		~~goto out;~~

	~~if (tty_port_initialized(&acm->port)) {~~
		~~rv = usb_submit_urb(acm->ctrlurb, GFP_ATOMIC);~~

		~~for (;;) {~~
			~~urb = usb_get_from_anchor(&acm->delayed);~~
			~~if (!urb)~~
				~~break;~~

			~~acm_start_wb(acm, urb->context);~~
		~~}~~

		~~/*~~
		 ~~* delayed error checking because we must~~
		 ~~* do the write path at all cost~~
		 ~~*/~~
		~~if (rv < 0)~~
			~~goto out;~~

		~~rv = acm_submit_read_urbs(acm, GFP_ATOMIC);~~
	~~}~~
~~out:~~
	~~spin_unlock_irq(&acm->write_lock);~~

	~~return rv;~~
~~}~~

~~static int acm_reset_resume(struct usb_interface *intf)~~
~~{~~
	~~struct acm *acm = usb_get_intfdata(intf);~~

	~~if (tty_port_initialized(&acm->port))~~
		~~tty_port_tty_hangup(&acm->port, false);~~

	~~return acm_resume(intf);~~
~~}~~

~~#endif /* CONFIG_PM */~~

static int acm_pre_reset(struct usb_interface *intf)
{
	struct acm *acm = usb_get_intfdata(intf);
```

```
        clear_bit(EVENT_RX_STALL, &acm->flags);

        return 0;
}

~~#define NOKIA_PCSUITE_ACM_INFO(x) \~~
                ~~USB_DEVICE_AND_INTERFACE_INFO(0x0421, x, \~~
                ~~USB_CLASS_COMM, USB_CDC_SUBCLASS_ACM, \~~
                ~~USB_CDC_ACM_PROTO_VENDOR)~~

~~#define SAMSUNG_PCSUITE_ACM_INFO(x) \~~
                ~~USB_DEVICE_AND_INTERFACE_INFO(0x04e7, x, \~~
                ~~USB_CLASS_COMM, USB_CDC_SUBCLASS_ACM, \~~
                ~~USB_CDC_ACM_PROTO_VENDOR)~~

/*
 * USB driver structure.
 */

static const struct usb_device_id acm_ids[] = {
        /* quirky and broken devices */
        ~~{ USB_DEVICE(0x076d, 0x0006), /* Denso Cradle CU-321 */~~
        ~~.driver_info = NO_UNION_NORMAL, },/* has no union descriptor~~
~~*/~~
        ~~{ USB_DEVICE(0x17ef, 0x7000), /* Lenovo USB modem */~~
        ~~.driver_info = NO_UNION_NORMAL, },/* has no union descriptor~~
~~*/~~
        ~~{ USB_DEVICE(0x0870, 0x0001), /* Metricom GS Modem */~~
        ~~.driver_info = NO_UNION_NORMAL, /* has no union descriptor~~
~~*/~~
        ~~},~~
        ~~{ USB_DEVICE(0x0e8d, 0x0003), /* FIREFLY, MediaTek Inc;~~
~~andrey.arapov@gmail.com */~~
        ~~.driver_info = NO_UNION_NORMAL, /* has no union descriptor~~
~~*/~~
        ~~},~~
        ~~{ USB_DEVICE(0x0e8d, 0x3329), /* MediaTek Inc GPS */~~
        ~~.driver_info = NO_UNION_NORMAL, /* has no union descriptor~~
~~*/~~
        ~~},~~
        ~~{ USB_DEVICE(0x0482, 0x0203), /* KYOCERA AH-K3001V */~~
        ~~.driver_info = NO_UNION_NORMAL, /* has no union descriptor~~
~~*/~~
        ~~},~~
        ~~{ USB_DEVICE(0x079b, 0x000f), /* BT On-Air USB MODEM */~~
        ~~.driver_info = NO_UNION_NORMAL, /* has no union descriptor~~
~~*/~~
        ~~},~~
        ~~{ USB_DEVICE(0x0ace, 0x1602), /* ZyDAS 56K USB MODEM */~~
        ~~.driver_info = SINGLE_RX_URB,~~
        ~~},~~
        ~~{ USB_DEVICE(0x0ace, 0x1608), /* ZyDAS 56K USB MODEM */~~
        ~~.driver_info = SINGLE_RX_URB, /* firmware bug */~~
        ~~},~~
        ~~{ USB_DEVICE(0x0ace, 0x1611), /* ZyDAS 56K USB MODEM - new~~
~~version */~~
        ~~.driver_info = SINGLE_RX_URB, /* firmware bug */~~
        ~~},~~
        ~~{ USB_DEVICE(0x22b8, 0x7000), /* Motorola Q Phone */~~
        ~~.driver_info = NO_UNION_NORMAL, /* has no union descriptor~~
~~*/~~
        ~~},~~
        ~~{ USB_DEVICE(0x0803, 0x3095), /* Zoom Telephonics Model~~
~~3095F USB MODEM */~~
        ~~.driver_info = NO_UNION_NORMAL, /* has no union descriptor~~
~~*/~~
        ~~},~~
        ~~{ USB_DEVICE(0x0572, 0x1321), /* Conexant USB MODEM CX93010~~
~~*/~~
        ~~.driver_info = NO_UNION_NORMAL, /* has no union descriptor~~
~~*/~~
        ~~},~~
        ~~{ USB_DEVICE(0x0572, 0x1324), /* Conexant USB MODEM RD02-~~
~~D400 */~~
        ~~.driver_info = NO_UNION_NORMAL, /* has no union descriptor~~
~~*/~~
        ~~},~~
        ~~{ USB_DEVICE(0x0572, 0x1328), /* Shiro / Aztech USB MODEM~~
~~UM-3100 */~~
        ~~.driver_info = NO_UNION_NORMAL, /* has no union descriptor~~
~~*/~~
        ~~},~~
        ~~{ USB_DEVICE(0x20df, 0x0001), /* Simtec Electronics Entropy~~
~~Key */~~
        ~~.driver_info = QUIRK_CONTROL_LINE_STATE, },~~
        ~~{ USB_DEVICE(0x2184, 0x001c) },  /* GW Instek AFG-2225 */~~
        ~~{ USB_DEVICE(0x2184, 0x0036) },  /* GW Instek AFG-125 */~~
        ~~{ USB_DEVICE(0x22b8, 0x6425), /* Motorola MOTOMAGX phones */~~
        ~~},~~
        ~~/* Motorola H24 HSPA module: */~~
        ~~{ USB_DEVICE(0x22b8, 0x2d91) }, /* modem~~
~~*/~~
        ~~{ USB_DEVICE(0x22b8, 0x2d92),   /* modem           +~~
~~diagnostics        */~~
        ~~.driver_info = NO_UNION_NORMAL, /* handle only modem~~
~~interface          */~~
        ~~},~~
        ~~{ USB_DEVICE(0x22b8, 0x2d93),   /* modem + AT port~~
~~*/~~
        ~~.driver_info = NO_UNION_NORMAL, /* handle only modem~~
~~interface          */~~
        ~~},~~
        ~~{ USB_DEVICE(0x22b8, 0x2d95),   /* modem + AT port +~~
```

```
~~diagnostics            */~~
        ~~.driver_info = NO_UNION_NORMAL, /* handle only modem~~
~~interface              */~~
        ~~},~~
        ~~{ USB_DEVICE(0x22b8, 0x2d96),    /* modem~~
~~+ NMEA */~~
        ~~.driver_info = NO_UNION_NORMAL, /* handle only modem~~
~~interface              */~~
        ~~},~~
        ~~{ USB_DEVICE(0x22b8, 0x2d97),    /* modem            +~~
~~diagnostics + NMEA */~~
        ~~.driver_info = NO_UNION_NORMAL, /* handle only modem~~
~~interface              */~~
        ~~},~~
        ~~{ USB_DEVICE(0x22b8, 0x2d99),    /* modem + AT port~~
~~+ NMEA */~~
        ~~.driver_info = NO_UNION_NORMAL, /* handle only modem~~
~~interface              */~~
        ~~},~~
        ~~{ USB_DEVICE(0x22b8, 0x2d9a),    /* modem + AT port +~~
~~diagnostics + NMEA */~~
        ~~.driver_info = NO_UNION_NORMAL, /* handle only modem~~
~~interface              */~~
        ~~},~~

        ~~{ USB_DEVICE(0x0572, 0x1329), /* Hummingbird huc56s~~
~~(Conexant) */~~
        ~~.driver_info = NO_UNION_NORMAL, /* union descriptor~~
~~misplaced on~~
                                              ~~data interface instead of~~
                                              ~~communications interface.~~
                                              ~~Maybe we should define a new~~
                                              ~~quirk for this. */~~
        ~~},~~
        ~~{ USB_DEVICE(0x0572, 0x1340), /* Conexant CX93010-2x UCMxx~~
~~*/~~
        ~~.driver_info = NO_UNION_NORMAL,~~
        ~~},~~
        ~~{ USB_DEVICE(0x05f9, 0x4002), /* PSC Scanning, Magellan 800i~~
~~*/~~
        ~~.driver_info = NO_UNION_NORMAL,~~
        ~~},~~
        ~~{ USB_DEVICE(0x1bbb, 0x0003), /* Alcatel OT-I650 */~~
        ~~.driver_info = NO_UNION_NORMAL, /* reports zero length~~
~~descriptor */~~
        ~~},~~
        ~~{ USB_DEVICE(0x1576, 0x03b1), /* Maretron USB100 */~~
        ~~.driver_info = NO_UNION_NORMAL, /* reports zero length~~
~~descriptor */~~
        ~~},~~

        ~~{ USB_DEVICE(0x2912, 0x0001), /* ATOL FPrint */~~
        ~~.driver_info = CLEAR_HALT_CONDITIONS,~~
        ~~},~~

        ~~/* Nokia S60 phones expose two ACM channels. The first is~~
         ~~* a modem and is picked up by the standard AT-command~~
         ~~* information below. The second is 'vendor-specific' but~~
         ~~* is treated as a serial device at the S60 end, so we want~~
         ~~* to expose it on Linux too. */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x042D), }, /* Nokia 3250 */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x04D8), }, /* Nokia 5500 Sport */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x04C9), }, /* Nokia E50 */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x0419), }, /* Nokia E60 */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x044D), }, /* Nokia E61 */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x0001), }, /* Nokia E61i */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x0475), }, /* Nokia E62 */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x0508), }, /* Nokia E65 */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x0418), }, /* Nokia E70 */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x0425), }, /* Nokia N71 */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x0486), }, /* Nokia N73 */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x04DF), }, /* Nokia N75 */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x000e), }, /* Nokia N77 */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x0445), }, /* Nokia N80 */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x042F), }, /* Nokia N91 & N91 8GB~~
~~*/~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x048E), }, /* Nokia N92 */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x0420), }, /* Nokia N93 */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x04E6), }, /* Nokia N93i  */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x04B2), }, /* Nokia 5700~~
~~XpressMusic */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x0134), }, /* Nokia 6110 Navigator~~
~~(China) */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x046E), }, /* Nokia 6110 Navigator~~
~~*/~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x002f), }, /* Nokia 6120 classic &~~
~~*/~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x0088), }, /* Nokia 6121 classic~~
~~*/~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x00fc), }, /* Nokia 6124 classic~~
~~*/~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x0042), }, /* Nokia E51 */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x00b0), }, /* Nokia E66 */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x00ab), }, /* Nokia E71 */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x0481), }, /* Nokia N76 */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x0007), }, /* Nokia N81 & N81 8GB~~
~~*/~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x0071), }, /* Nokia N82 */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x04F0), }, /* Nokia N95 & N95-3~~
~~NAM */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x0070), }, /* Nokia N95 8GB  */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x00e9), }, /* Nokia 5320~~
~~XpressMusic */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x0099), }, /* Nokia 6210~~
```

```
~~Navigator, RM-367 */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x0128), }, /* Nokia 6210~~
~~Navigator, RM-419 */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x008f), }, /* Nokia 6220 Classic~~
~~*/~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x00a0), }, /* Nokia 6650 */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x007b), }, /* Nokia N78 */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x0094), }, /* Nokia N85 */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x003a), }, /* Nokia N96 & N96-3~~
~~*/~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x00e9), }, /* Nokia 5320~~
~~XpressMusic */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x0108), }, /* Nokia 5320~~
~~XpressMusic 2G */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x01f5), }, /* Nokia N97, RM-505 */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x02e3), }, /* Nokia 5230, RM-588~~
~~*/~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x0178), }, /* Nokia E63 */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x010e), }, /* Nokia E75 */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x02d9), }, /* Nokia 6760 Slide */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x01d0), }, /* Nokia E52 */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x0223), }, /* Nokia E72 */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x0275), }, /* Nokia X6 */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x026c), }, /* Nokia N97 Mini */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x0154), }, /* Nokia 5800~~
~~XpressMusic */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x04ce), }, /* Nokia E90 */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x01d4), }, /* Nokia E55 */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x0302), }, /* Nokia N8 */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x0335), }, /* Nokia E7 */~~
        ~~{ NOKIA_PCSUITE_ACM_INFO(0x03cd), }, /* Nokia C7 */~~
        ~~{ SAMSUNG_PCSUITE_ACM_INFO(0x6651), }, /* Samsung GTi8510~~
~~(INNOV8) */~~

        ~~/* Support for Owen devices */~~
        ~~{ USB_DEVICE(0x03eb, 0x0030), }, /* Owen SI30 */~~

        ~~/* NOTE: non-Nokia COMM/ACM/0xff is likely MSFT RNDIS... NOT~~
~~a modem! */~~

        ~~/* Support for Droids MuIn LCD */~~
        ~~{ USB_DEVICE(0x04d8, 0x000b),~~
        ~~.driver_info = NO_DATA_INTERFACE,~~
        ~~},~~

~~#if IS_ENABLED(CONFIG_INPUT_IMS_PCU)~~
        ~~{ USB_DEVICE(0x04d8, 0x0082),     /* Application mode */~~
        ~~.driver_info = IGNORE_DEVICE,~~
        ~~},~~
        ~~{ USB_DEVICE(0x04d8, 0x0083),     /* Bootloader mode */~~
        ~~.driver_info = IGNORE_DEVICE,~~
        ~~},~~
```

```
~~#endif~~

      ~~/*Samsung phone in firmware update mode */~~
      ~~{ USB_DEVICE(0x04e8, 0x685d),~~
      ~~.driver_info = IGNORE_DEVICE,~~
      ~~},~~

      ~~/* Exclude Infineon Flash Loader utility */~~
      ~~{ USB_DEVICE(0x058b, 0x0041),~~
      ~~.driver_info = IGNORE_DEVICE,~~
      ~~},~~

      ~~/* control interfaces without any protocol set */~~
      ~~{ USB_INTERFACE_INFO(USB_CLASS_COMM, USB_CDC_SUBCLASS_ACM,~~
           ~~USB_CDC_PROTO_NONE) },~~

      ~~/* control interfaces with various AT command sets */~~
      ~~{ USB_INTERFACE_INFO(USB_CLASS_COMM, USB_CDC_SUBCLASS_ACM,~~
           ~~USB_CDC_ACM_PROTO_AT_V25TER) },~~
      ~~{ USB_INTERFACE_INFO(USB_CLASS_COMM, USB_CDC_SUBCLASS_ACM,~~
           ~~USB_CDC_ACM_PROTO_AT_PCCA101) },~~
      ~~{ USB_INTERFACE_INFO(USB_CLASS_COMM, USB_CDC_SUBCLASS_ACM,~~
           ~~USB_CDC_ACM_PROTO_AT_PCCA101_WAKE) },~~
      ~~{ USB_INTERFACE_INFO(USB_CLASS_COMM, USB_CDC_SUBCLASS_ACM,~~
           ~~USB_CDC_ACM_PROTO_AT_GSM) },~~
      ~~{ USB_INTERFACE_INFO(USB_CLASS_COMM, USB_CDC_SUBCLASS_ACM,~~
           ~~USB_CDC_ACM_PROTO_AT_3G) },~~
      ~~{ USB_INTERFACE_INFO(USB_CLASS_COMM, USB_CDC_SUBCLASS_ACM,~~
           ~~USB_CDC_ACM_PROTO_AT_CDMA) },~~

      ~~{ USB_DEVICE(0x1519, 0x0452), /* Intel 7260 modem */~~
      ~~.driver_info = SEND_ZERO_PACKET,~~
      ~~},~~

      ~~{ }~~
};

MODULE_DEVICE_TABLE(usb, acm_ids);

static struct usb_driver acm_driver = {
      .name =         "cdc_acm",
      .probe =   acm_probe,
      .disconnect =   acm_disconnect,
~~#ifdef CONFIG_PM~~
      ~~.suspend = acm_suspend,~~
      ~~.resume =  acm_resume,~~
      ~~.reset_resume = acm_reset_resume,~~
~~#endif~~
      .pre_reset =    acm_pre_reset,
      .id_table =     acm_ids,
~~#ifdef CONFIG_PM~~
      ~~.supports_autosuspend = 1,~~
```

```
~~#endif~~
	.disable_hub_initiated_lpm = 1,
};

/*
 * TTY driver structures.
 */

static const struct tty_operations acm_ops = {
	.install =		acm_tty_install,
	.open =			acm_tty_open,
	.close =		acm_tty_close,
	.cleanup =		acm_tty_cleanup,
	.hangup =		acm_tty_hangup,
	.write =		acm_tty_write,
	.put_char =		acm_tty_put_char,
	.flush_chars =		acm_tty_flush_chars,
	.write_room =		acm_tty_write_room,
	.ioctl =		acm_tty_ioctl,
	~~.throttle =		acm_tty_throttle,~~
	~~.unthrottle =		acm_tty_unthrottle,~~
	.chars_in_buffer =	acm_tty_chars_in_buffer,
	.break_ctl =		acm_tty_break_ctl,
	.set_termios =		acm_tty_set_termios,
	.tiocmget =		acm_tty_tiocmget,
	.tiocmset =		acm_tty_tiocmset,
	.get_icount =		acm_tty_get_icount,
};

/*
 * Init / exit.
 */

static int __init acm_init(void)
{
	int retval;
	acm_tty_driver = alloc_tty_driver(ACM_TTY_MINORS);
	if (!acm_tty_driver)
		return -ENOMEM;
	acm_tty_driver->driver_name = "acm",
	acm_tty_driver->name = "ttyACM",
	acm_tty_driver->major = ACM_TTY_MAJOR,
	acm_tty_driver->minor_start = 0,
	acm_tty_driver->type = TTY_DRIVER_TYPE_SERIAL,
	acm_tty_driver->subtype = SERIAL_TYPE_NORMAL,
	acm_tty_driver->flags = TTY_DRIVER_REAL_RAW |
TTY_DRIVER_DYNAMIC_DEV;
	acm_tty_driver->init_termios = tty_std_termios;
	acm_tty_driver->init_termios.c_cflag = B9600 | CS8 | CREAD |
								HUPCL | CLOCAL;
	tty_set_operations(acm_tty_driver, &acm_ops);
```

this structure will hold the functions to be called when a process does something to the device we created. since a pointer to this structure is kept in the devices table, it cant be local to init.

```
	retval = tty_register_driver(acm_tty_driver);
	if (retval) {
		put_tty_driver(acm_tty_driver);
		return retval;
	}

	retval = usb_register(&acm_driver);
	if (retval) {
		tty_unregister_driver(acm_tty_driver);
		put_tty_driver(acm_tty_driver);
		return retval;
	}

	printk(KERN_INFO KBUILD_MODNAME ": " DRIVER_DESC "\n");

	return 0;
}

static void __exit acm_exit(void)
{
	usb_deregister(&acm_driver);
	tty_unregister_driver(acm_tty_driver);
	put_tty_driver(acm_tty_driver);
	idr_destroy(&acm_minors);
}

module_init(acm_init);
module_exit(acm_exit);
```

~~MODULE_AUTHOR(DRIVER_AUTHOR);~~
~~MODULE_DESCRIPTION(DRIVER_DESC);~~
~~MODULE_LICENSE("GPL");~~

```
MODULE_ALIAS_CHARDEV_MAJOR(ACM_TTY_MAJOR);
```