Helsinki University of Technology
Networking Laboratory
S-38.138 Special Assignment on Networking Technology

Implementation of the GSM 07.10 Linux Device Driver

Author: Tuukka Karvonen
51272M
tuukka.karvonen@hut.fi

Instructor: Markus Peuhkuri

Returned: 05.01.2004
Updated: 21.04.2004

Abstract

The assignment was to implement a Linux device driver for the GSM 07.10 multiplexer protocol. The protocol operates between a mobile station and a terminal equipment and allows a number of simultaneous sessions over a normal serial asynchronous interface. This permits, for example, sending of SMS from the terminal equipment while a data connection is in progress.

The GSM 07.10 specification defines three operation modes: basic, advanced without error recovery and advanced with error recovery. Only a subset of basic mode operations were included in the implemented driver, because the driver was primarily made for the Siemens MC35 module which doesn't support the advanced modes and all the operations of the basic mode.

The driver creates a number of virtual serial ports and handles the communication between them and the actual serial line. There are at least three ways to implement virtual serial ports for Linux: pseudo terminals, own serial port driver in the kernel and an user space serial port (USSP) driver. The driver was implemented as an USSP driver. This allows the driver code to be run in the user-space, while the USSP kernel module implemented by Patrick van de Lageweg and Rogier Wolff transfers calls between the driver and the kernel. An user space program is more simple to implement and to debug, but is a little slower than a driver in the kernel space would be.

Contents

Abbreviations

8N1	8 data bits, no parity bit, 1 stop bit
CTS	Clear To Send
DLC	Data Link Connection
DTR	Data Terminal Ready
FC	Flow Control
FCS	Frame Check Sequence
FTP	File Transfer Protocol
GSM	Global System for Mobile Communications
GPRS	General Packet Radio Service
IP	Internet Protocol
MS	Mobile Station
pppd	Point to Point Protocol Daemon
SABM	Set Asynchronous Balanced Mode
SMS	Short Message Service
TE	Terminal Equipment
UI	Unnumbered Information
UIH	Unnumbered Information with Header check

1. Introduction

GSM 07.10 is a multiplexer protocol specified by ETSI. It operates between a MS and a TE and allows a number of simultaneous sessions over a normal serial asynchronous interface. This permits, for example, sending of SMS from the TE while a data connection is in progress.

The GSM 07.10 specification defines thee operation modes: basic, advanced without error recovery and advanced with error recovery [1]. At a moment only a few mobile stations implement the GSM 07.10 protocol. The driver was implemented primarily for Siemens TC3x/MC3x module which implements only a subset of the basic mode operations [2]. That's why the driver includes only basic mode operations and not even all of them.

Operating system is usually divided into a kernel and user-space. The kernel shares the resources of the computer for the programs that run in the user-space. Device drivers usually run in the kernel space and provide access to the devices for the programs. Kernel space programming is harder than user-space programming, because standard C-libraries are not in use, too much memory shouldn't be used, debugging is hard and bugs often crash the whole operating system. However, a kernel space code has also advantages such as it is fast to run and it has straight access to the interrupts, which is important for some device drivers. [3]

2. GSM 07.10 Multiplexer Protocol

A GSM 07.10 multiplexer implementation takes input data from multiple sources (e.g. virtual serial ports) and puts the data into frames before sending it over a serial link. It also reads data from the serial link, extracts the user data from the frames and sends it to a specific receiver (e.g. a virtual serial port). This way GSM 07.10 protocol forms virtual channels that are called Data Link Connections (DLC). Frame headers include a DLC identifier that distinguishes the channel in question.

Following frame types are defined:

- Set Asynchronous Balanced Mode (SABM) command opens a DLC
- Disconnect (DISC) command closes a DLC
- Unnumbered Acknowledgment (UA) response acknowledges DISC or SABM frame
- Disconnected Mode (DM) response informs that the DLC is not opened
- Unnumbered Information with header check (UIH) transfers control commands on the control channel and data on other DLCs. FCS is calculated only from the frame headers.
- Unnumbered Information (UI) transfers control commands on the control channel and data on other DLCs. FCS is calculated from the whole frame.

One DLC is specially for the multiplexer control information and it's called the control channel. Multiplexer session control information and modem signals can be transfered over the control channel in UI or UIH frames. Other DLCs are for transferring user data.

In the figure 1 we can see the frame structure in the basic operation mode. Each frame starts and ends with a flag. The address field contains the DLC identifier and the control field specifies the frame type. The length indicator tells how much user information is included in the frame. The information field is only present in UI and UIH frames. Each frame has also a FCS that is used to recognize transmission errors. [1]

Flag	Address	Control	Length Indicator	Information	FCS	Flag
1 octet	1 octet	1 octet	1 or 2 octets	Unspecified length but integral number of octets	1 octet	1 octet

Figure 1

2.1. Example of the Communication

Figure 2 demonstrates, how the GSM 07.10 protocol can be used. First the MUX mode is turned on with AT command. Then the driver opens control channel (DLC 0) and 3 other DLCs by sending SABM frames. The MS acknowledges the openings of the DLCs by sending UA frames.

Then the user wants to send AT command to the first virtual serial port. The driver transmits the command to the MS in an UIH frame. The MS sends it's answer back in an UIH frame and the driver forwards the answer to the first virtual serial port, where the user can read it.

Finally the user turns the driver off. Driver sends multiplexer close down request over the control channel in an UIH frame and the MS acknowledges the command.

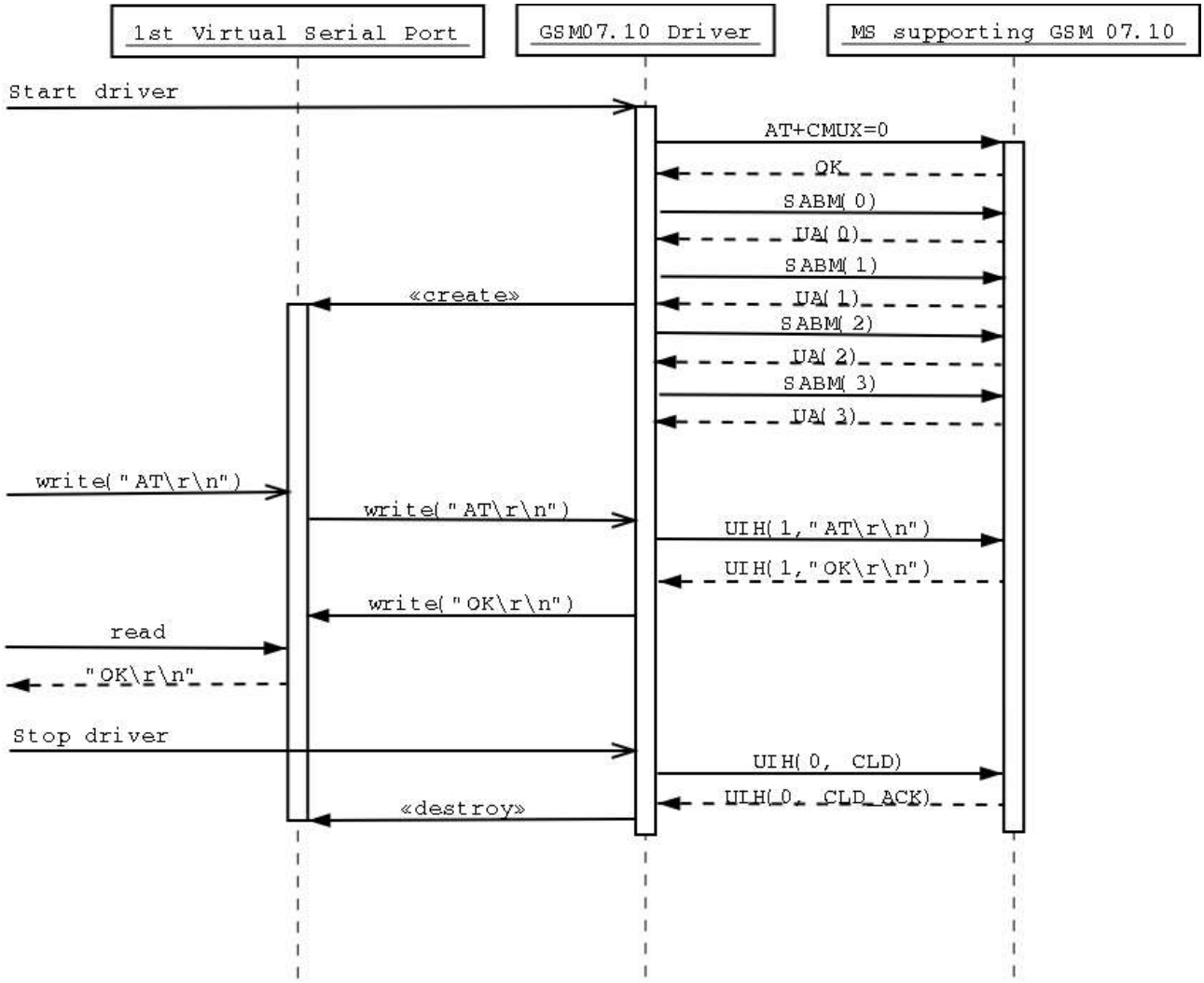

Figure 2

3. Different Approaches for Implementing Virtual Serial Ports

In Linux serial ports are device files. You can open them, write to them and read them just like normal files. There are also some special functions that can be used for example to control the baud rate. When a user accesses normal file, kernel invokes file system functions, but when a user accesses a device file, kernel needs to have device driver functions that it can call. At least three different methods exist to implement virtual serial ports in Linux.

One is pseudo terminals, which are used for example by telnetd. Pseudo terminals have a master

device, which the user sees and controls, and a slave device, which provides stdin, stderr, stdout for the server program that can run in user space [4]. The pseudo terminal driver in the kernel forwards the data between the master and slave devices. With pseudo terminals you can't control modem signals, so they are not so suitable for this driver.

Another way is to create own virtual serial port driver as a kernel module. A module is loaded into the kernel. It registers its' functions to the kernel so that the kernel knows to call them when the virtual serial port device file is accessed. This would have the best performance, but it would be hard to implement and there weren't many guides on how to write one. A best source that was found for Linux device writing was [3] and it didn't cover serial port drivers.

Third method is to implement the driver as a user space serial port driver. In that the USSP kernel module implemented by Patrick van de Lageweg and Rogier Wolff handles kernel level serial port operations and communicates with the user-space driver through a dedicated device file. The driver only needs to open the dedicated device file and it can write commands to it for example to initialize the virtual serial ports. The driver can read from the dedicated file what has been written to the virtual serial ports. The user-space driver can also control the modem signals of the virtual serial ports. This method was chosen for this GSM 07.10 driver, because the code could be written in user-space and RFCOMM layer implementation by Marcel Holtmann [5] could be used as an example.

4.Implemented Features

The driver doesn't support the features of the advanced GSM 07.10 modes and only a subset of the basic mode features are implemented. The driver understands all the frame types, but it can't send UI frames. So only UIH frames are used for data transfer.

Command timeouts and retries are not implemented. The daemon sends in the beginning SABM frames to open all the DLCs needed. If a frame is lost, which is unusual, DLC remains closed and the daemon needs to be restarted to open the DLC.

The following control channel commands are implemented: multiplexer close down, test command, modem status command and non supported command response. The break signal octet in modem status command is not used, because a way to distinguish between +++ break string and similar normal data string was not found. To hang-up a call, DTR signal drop is transferred in the modem status command. DTR drop is recognized in two different ways by the driver:

- if an application changes the modem signal (`ioctl(fd, TIOCMSET, &status);`)
- or an application changes to the zero baud rate (`B0`) [6]

The USSP driver was modified so that it informs the user-space driver about the modem signal changes done with `TIOCMSET`.

For performance reasons, Siemens TC3x doesn't support all V.24 modem signals that can be transferred in the modem status command. It supports only flow control (FC) bit and ready to communicate (RTC) bit, which is used for DTR mapping. It sends the ring indication on the normal serial line [2]. The driver supports all the modem status command bits defined in the GSM 07.10 specification except that received FC bit it mapped to CTS signal, because the Siemens TC3x doesn't use the Ready to Receive bit, which is specified for CTS signal mapping. The ring indication from the normal serial line is not mapped to the virtual serial ports. So only indication of incoming call is RING result code text from the MS.

Version test message is sent in a test command once the control channel is open. This is needed to get the multiplexer version 2 features of the Siemens TC3x in use [2]. The version possibly returned by the MS is not taken into a account, because enough documentation on the version differences

couldn't be found. In other words, the driver works the same way despite of the multiplexer version of the MS.

5.Instructions for Use

The driver source code can be downloaded from the author's website (http://www.iki.fi/tkarvone/).

First you need to extract and compile the driver:

```
tar -xvzf gsm0710.tar.gz
cd gsm0710
make
```

The driver needs an USSP kernel module. The original USSP module was modified to inform the daemon about modem signal changes. If you don't have the USSP kernel module set up or you want to transfer modem signal changes to the MS, you need to compile and install the USSP driver included in the package. E.g.:

```
su
cd ussp
make
./mkdev.ussp
insmod ./ussp.o
exit
```

Next you should check that your MS is ready for the multiplexer mode. You can give required terminal settings and AT-commands e.g. with Minicom serial communication program. In the mux mode characters are transmitted in 8N1 mode [1]. In addition to that Siemens TC3x requires that `AT+IPR` is not in auto mode, so you should type `AT+IPR=57600`, and also use of RTS/CTS flow control is recommended (`AT\Q3`) [2].

Then you can start the daemon. First make sure that you have rw-permissions for the device `/dev/ussp_ctl` or `/dev/misc/ussp_ctl` in case of devfs and for the serial port that the MS is connected to. The driver command syntax is:

```
gsm0710 <dev> [channels] [first_ttyU] [max_frame_size]
```

where:

- dev is the device (e.g. /dev/ttyS0)
- channels is the number of logical channels (defaults to 3)
- first_ttyU is the number of first ttyU device (defaults to 0 i.e. /dev/ttyU0)
- max_frame_size is the maximum amount of information octets allowed in one frame (defaults to Siemens TC3x limit i.e. 97. The default value specified in the GSM 07.10 specification would be 64 octets.)

So if you have connected your Siemens TC3x/MC3x to the first serial port, you could start the driver with command:

```
./gsm0710 /dev/ttyS0
```

The driver can be stopped by pressing CTRL-C, which terminates the multiplexer session.

6.Test Results

The driver was tested on a 700 MHz PC with Siemens MC35 connected to a serial port. The Linux kernel version on the test platform was 2.4.20-18.7. The driver seemed to work fine even when all three virtual ports were in use. Hang-up could be signaled by dropping DTR and the driver reacted in the right manner to received frames.

On the first virtual port pppd was run first over a dial-up connection and then over a GPRS connection. SMS commands were given on the second and the third virtual ports with a help of Minicom serial communication program.

About 200 kbyte file was both uploaded and downloaded over FTP. Downloading couldn't be noticed on the other logical channels, but when uploading one could notice some short delays in echo when typing in SMS commands. This is due to the fact that the current driver can send several frames of data received from one virtual port before checking the other virtual ports. Anyway SMS commands could be given without bigger problems.

In the end of test the following data was collected from ifconfig, pppd and the implemented driver:

```
ifconfig:
ppp0      Link encap:Point-to-Point Protocol
          inet addr:10.64.0.229  P-t-P:192.168.254.254  Mask:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:2000  Metric:1
          RX packets:533 errors:15 dropped:0 overruns:0 frame:0
          TX packets:573 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:3
          RX bytes:340279 (332.3 Kb)  TX bytes:343076 (335.0 Kb)

pppd:
Connect time 11.0 minutes.
Sent 343446 bytes, received 342049 bytes.

gsm0710:
Received 6356 frames. Dropped 17 received frames
```

One can see that about 0,3 % of the received frames were dropped and this probably caused the errors in about 3 % of the received IP packets. The frames were dropped, because the FCSs didn't match. Most of these errors occurred during the file download and were probably caused by bit flips on the serial link.

7. Summary

The implemented GSM 07.10 device driver allows several applications to use a single mobile station simultaneously on a Linux machine. This can be useful for example when versatile mobile applications, which need both a data connection and SMSs, are developed.

The implemented driver doesn't support all the features specified in the GSM 07.10 specification. Missing features can be implemented, if needed, but for basic use the driver should be enough. If there is going to be high usage on all logical channels, some traffic control algorithm might need to be added, so that all the channels could get equal performance.

8. Bibliography

[1] Terminal Equiment to Mobile Station (TE-MS) multiplexer protocol (GSM 07.10 version 7.1.0) , ETSI 1998

[2] TC3x Multiplexer User's Guide V03.10, Siemens 2001

[3] Alessandro Rubini & Jonathan Corbet, Linux Device Drivers 2nd Edition, O'Reilly 2001

[4] Anonymous, The Linux Tutorial, 2003, http://www.linux-tutorial.info/cgi-bin/display.pl?325&0&0&0&3

[5] Marcel Holtmann, RFCOMM layer for the BlueZ stack, 2002, http://www.holtmann.org/linux/bluetooth/rfcomm.html

[6] Michael R. Sweet, Serial Programming Guide for POSIX Operating Systems, 1999-2003, http://www.easysw.com/~mike/serial/serial.html