The GNSS Subsystem

Johan Hovold

Hovold Consulting AB

Open Source Summit Europe, Edinburgh
October 24, 2018
Introduction

- Global Navigation Satellite System (GNSS)
  - GPS (US)
  - GLONASS (Russia)
  - BeiDou (China)
  - Galileo (EU)
- Satellite-based radio navigation
  - Position, velocity and time (PVT)
- GNSS receivers currently managed in user space
- Serial device bus (serdev) allows for a higher-level abstraction
  - Power management
  - Device detection
Outline

• Background and theory
• User interface
• Driver interface
• Currently supported devices
• Limitations
• Future work
GNSS history

- Ground-based radio navigation (1940s)
  - Gee, LORAN, Decca
- Satellite-based radio navigation
  - Transit (1960s)
  - GPS, GLONASS (1970s)
  - BeiDou (1990s)
  - Galileo (2000s)
- Politics
  - Military purposes
  - GPS Selective Availability (2000)
- Miniaturisation
  - First single-chip receiver (2004)
  - Smartphone with GPS (2007)
  - 5.8 billion GNSS devices in 2017 (forecasted to 8 billion in 2020)
GNSS theory

- **Satellites**
  - 24 + 6 satellites in three orbital planes (Galileo example)
  - Atomic clock

- **Radio signals**
  - L band (1–2 GHz)
  - Timing signal
  - Navigation data (ephemeris, status, ...)

- **Receivers**
  - Track satellites and estimate pseudo ranges
  - Position, velocity and time (PVT)
GNSS receivers

- Antenna, front-end, baseband signal processing, application processing
- Acquisition and tracking
- PVT solution (2D, 3D)
- Time to first fix (TTFF)
  - Cold, warm and hot start
- I/O interfaces (UART, ...)
  - Reports (out)
  - Control (in)
- Power supplies and enable signals
Receiver I/O interfaces

- UART
- I2C
- SPI
- Remote processor messaging (rpmsg)
- MMIO
- USB
- SDIO
- ...

...
Receiver protocols

• Periodic reports + control
  • Position, velocity and time
  • Satellites in view

• NMEA 0183
  • National Marine Electronics Association (1980s)
  • De-facto standard
  • Subset with vendor extensions
  • Proprietary
  • Much have been reverse-engineered

• Vendor protocols
  • Garmin, SiRF Binary, UBX, ...
  • Proprietary
  • NMEA and vendor mode (runtime configurable)
$$GPGGA,092750.000,5321.6802,N,00630.3372,W,1,8,1.03,\61.7,M,55.2,M,,*76$$

- Checksummed (printable) ASCII sentences
  - Time, position and fix-related data
  - Position
  - Velocity
  - Satellites in view
  - Time and date

- Incomplete PVT reports

- Underspecified report cycles

- No standard control commands (vendor extensions)
  - Port settings
  - Message rates
GNSS and Linux

- Handled in user space
  - gpsd
  - Android location services
- UART-interface only (TTY)
  - Custom drivers and hacks for non-UART
- Device description in user space
  - Device and protocol detection hacks
- Power management
  - Modem control signals (DTR)
  - GPIOs (gpiolib)
GTA04 GPS power management

- GTA04, OpenMoko
- Wi2Wi SiRFstar-based GPS receiver
  - on/off input, but no wakeup output signal
  - Monitor data channel to determine power state
- Various proposals over the years
  - Neil Brown, Nikolaus Schaller and others
- Serial device bus (serdev)
  - Finally possible to implement in kernel
  - Specific wi2wi serdev driver with custom TTY interface
- Need a GNSS-receiver framework
Problem statement

- I/O interface abstraction
- Device description and discovery (e.g. Device tree or ACPI)
- Power management
  - Regulators, GPIOs, clocks...
  - Data stream (GTA04)
- Vendor protocols...
Design decision

- Keep everything in user space?
  - User-space drivers
  - Some resources not available (e.g. regulators, clocks)
  - Device descriptions in user space
  - No I/O-interface abstraction
  - System-suspend coordination

- Handle everything in kernel?
  - Proprietary protocols
  - Legal issues
  - Non-reverse engineered
  - String parsing
  - Device-dependent features and quirks
  - Hard to generalise protocols
  - Would require new user-space services
  - Floating-point math?

- Keep protocol handling in user space
Design decision

- Keep everything in user space?
  - User-space drivers
  - Some resources not available (e.g. regulators, clocks)
  - Device descriptions in user space
  - No I/O-interface abstraction
  - System-suspend coordination

- Handle everything in kernel?
  - Proprietary protocols
    - Legal issues
    - Non-reverse engineered
  - String parsing
  - Device-dependent features and quirks
  - Hard to generalise protocols
  - Would require new user-space services
  - Floating-point math?
Design decision

- Keep everything in user space?
  - User-space drivers
  - Some resources not available (e.g. regulators, clocks)
  - Device descriptions in user space
  - No I/O-interface abstraction
  - System-suspend coordination

- Handle everything in kernel?
  - Proprietary protocols
    - Legal issues
    - Non-reverse engineered
  - String parsing
  - Device-dependent features and quirks
  - Hard to generalise protocols
  - Would require new user-space services
  - Floating-point math?

- Keep protocol handling in user space
The GNSS subsystem

- Raw character-device interface
  - Protocols handled in user space
- I/O-interface abstraction
- Device detection and description
- Power management
- Compatible with current user space
- Can be extended with high-level interface later
- Merged in 4.19
User interface

- GNSS class device
  - /sys/class/gnss/gnss0
- type sysfs attribute and GNSS_TYPE uevent variable
  - "NMEA"
  - "SiRF"
  - "UBX"
- Character device
  - /dev/gnss0
  - Pollable read, 4k buffer
  - Synchronous write
Device-tree bindings

- Child node of I/O interface node
- Generic properties
  - compatible (required)
- Additional resources

```c
&uart1 {
  gnss {
    compatible = "wi2wi,w2sg0084i";
    vcc-supply = <&gnss_reg>;
    sirf, onoff-gpios = <&gpio0 16 GPIO_ACTIVE_HIGH>;
    sirf, wakeup-gpios = <&gpio0 17 GPIO_ACTIVE_HIGH>;
  }
};
```
Driver interface

- Allocation and registration
- Insertion of raw protocol data
- Callbacks for opening, closing and writing
Driver-interface functions

```c
struct gnss_device;

struct gnss_device *gnss_allocate_device(...);
void gnss_put_device(...);

int gnss_register_device(...);
void gnss_deregister_device(...);

void gnss_set_drvdata(...);
void *gnss_get_drvdata(...);

int gnss_insert_raw(...);
```

- `gnss_insert_raw()` serialised by caller, any context
Driver-interface callbacks

```c
struct gnss_operations {
    int (*open)(struct gnss_device *);
    void (*close)(struct gnss_device *);
    int (*write_raw)(struct gnss_device *,
                     const unsigned char *, size_t);
};
```

- **open()** called on first open
- **close()** called on final close (or disconnect)
- **write_raw()**
  - Synchronous, may sleep
Power management

- Handled on interface level (e.g. serdev device)
- Runtime power management
  - Open serial port and enable receiver using RPM on open()
  - Allows user space to set always-on (power/control)
- System suspend
  - Enable low-power mode or power off
Serial-library functions

```
struct gnss_serial;

struct gnss_serial *gnss_serial_allocate(...);
void gnss_serial_free(...);

int gnss_serial_register(...)
void gnss_serial_deregister(...);

void *gnss_serial_get_drvdata(...);
```

- Generic serial GNSS-driver implementation
- Callbacks for power management
Serial-library callbacks

```c
enum gnss_serial_pm_state {
    GNSS_SERIAL_OFF,
    GNSS_SERIAL_ACTIVE,
    GNSS_SERIAL_STANDBY,
};

struct gnss_serial_ops {
    int (*set_power)(struct gnss_serial *gserial,
                     enum gnss_serial_pm_state state);
};
```

- **ACTIVE** - open or runtime active
- **STANDBY** - closed or system suspended
- **OFF** - driver unbound
Merged drivers

- SiRFstar receivers (sirf)
  - Main supply
  - onoff input
  - wakeup output
  - Not using serial library (wakeup NC)
  - Not-connected wakeup not yet supported (e.g. GTA04)
- u-blox receivers (ubx)
  - Main and backup supplies
  - Serial library
Limitations

- Line-speed handling
  - Coordinate protocol and interface control
  - New GNSS ioctl()?
  - Handle in kernel?
Hotplugging

- USB-serial-connected receivers
- Unique idVendor and idProduct?
  - Kernel descriptions
  - User-space descriptions
- GNSS core supports hotplugging
- But serdev does not (yet)
Modems

- GNSS receiver integrated with modem
  - Assisted GPS (A-GPS)
  - Reduce time to first fix (e.g. almanac and time from network)
- Modems managed in user space
  - oFono telephony stack
- Kernel interfaces
  - TTY (cdev)
  - Phonet (socket)
  - CAIF (socket)
  - CDC WDM (cdev)
- Example
  - Control commands on one port (e.g. power management)
  - GNSS reports on another (e.g. NMEA 0183)
- User-space GNSS drivers
- Feed raw data to GNSS core
- Accessible through common interface
- Needed while modems are managed in user space
- Can also be used for testing
Future work

- Pulse per second (PPS)
- Low-noise amplifiers (LNA)
- ugnss
- Line-speed handling
- High-level interface?
Further reading

- **GNSS Market Report Issue 5**, European GNSS Agency

- **Navipedia**, European Space Agency
  - https://gssc.esa.int/navipedia/index.php/Main_Page

- **Towards A Better GPS Protocol**, Eric S. Raymond

- **Why GPSes suck, and what to do about it**, Eric S. Raymond
  - http://esr.ibiblio.org/?p=801
Thanks!

johan@hovoldconsulting.com
johan@kernel.org