Device Driver Development Demystified

Thomas Cort

MINIXCon 2016

February 1, 2016

whoami(1)

Open Source Contributor and Former Google Summer of Code Student

- GSoC 2010 - UNIX domain sockets
- GSoC 2011 - software porting and pkgsrc improvements
- GSoC 2013 - i2c drivers for the BeagleBone Black
- Tested and Imported 90+ programs from NetBSD
- Other minor patches

Qualifications

I've developed several drivers:

- i2c bus driver for BeagleBone (Black and White) & BeagleBoard-xM
- Weather Cape: SHT21, BMP085, TSL2550
- Power Management: TPS65217, TPS65950
- RTC, CAT24C256, TDA19988 (basic), GPIO (basic)
- i2cscan(8), rebooting, poweroff (BeagleBone)

Goal

Inspire you to develop more device drivers for Minix

How we're going to get there?

Show how device drivers are built from concept to code

Developing device drivers on Minix rocks

- Infinite loops don't make the whole system freeze
- Bad pointers don't make the whole system crash
- No rebooting needed. Just start and stop your driver
- Most drivers are single threaded
- Most drivers only handle one request at a time
- Messages between drivers are well defined and documented

Questions to ask yourself when choosing a device

- is it already supported (partially or fully)?
- is a similar device already supported?
- is anyone else already working on it?
- is there documentation available?
- is there hardware available?
- is the device supported by other FOSS operating systems?
- is it possible to use the device on a system supported by Minix?

Where to begin

Gather documentation:

- Existing Minix drivers and the Minix wiki
- System Reference Manuals (SRM)
- Technical Reference Manuals (TRM)
- Data Sheets
- Example code from the device manufacturer (sdk)
- Code from other operating systems (Linux, BSD, Haiku, etc)

Eliminate hardware issues before you write a line of code

- Start with new or gently used hardware
- Test the device with another OS
- Test the computer with Minix

External Interface Design

Stand on the shoulders of giants!

- If it's a new class of device, follow the lead of NetBSD
- If it's an existing class of device, implement the established interface

Case Study

bmp085 temperature sensor

File Layout

```
/usr/src/minix/drivers/sensors/bmp085
          bmp085.c
          Makefile
          README.txt
```

../Makefile

```
.include <bsd.own.mk>

.if ${MACHINE_ARCH} == "earm"
SUBDIR+=        bmp085
SUBDIR+=        sht21
SUBDIR+=        tsl2550
.endif # ${MACHINE_ARCH} == "earm"

.include <bsd.subdir.mk>
```

./Makefile

```
# Makefile for the bmp085 pressure and temp sensor
PROG=   bmp085
SRCS=   bmp085.c

DPADD+= ${LIBI2CDRIVER} ${LIBCHARDRIVER} ${LIBSYS}\
        ${LIBTIMERS}
LDADD+= -li2cdriver -lchardriver -lsys -ltimers

CPPFLAGS+=      -I${NETBSDSRCDIR}

.include <minix.service.mk>
```

./README.txt

Provides answers to the following questions:

- what does the driver do?
- what is in each source file?
- how do I start/stop/test the driver?
- are there any limitations?
- where can I find more information about this device?

etc/system.conf

```
service bmp085
{
        ipc SYSTEM RS DS i2c;
};
```

distrib/sets/lists/minix-base/md.evbarm

```
./service/bmp085    minix-base
```

main()

```
int main(int argc, char *argv[]) {
        int r;

        env_setargs(argc, argv);

        r = i2cdriver_env_parse(&bus, &address,
                                              valid_addrs);
        if (r < 0) /* ... */
        else if (r > 0) /* ... */

        sef_local_startup();
        chardriver_task(&bmp085_tab);
        return 0;
}
```

sef_local_startup(void)

```
static void sef_local_startup(void) {
        /* Register init callbacks. */
        sef_setcb_init_fresh(sef_cb_init);
        sef_setcb_init_lu(sef_cb_init);
        sef_setcb_init_restart(sef_cb_init);

        /* Register live update callbacks. */
        sef_setcb_lu_state_save(
                                   sef_cb_lu_state_save);

        /* Let SEF perform startup. */
        sef_startup();
}
```

sef_cb_init(int type, sef_init_info_t *info)

```
static int
sef_cb_init(int type, sef_init_info_t *info) {

  if (type == SEF_INIT_LU) lu_state_restore();

  /* ... i2cdriver_reserve_address() ... */

  if (bmp085_init() != OK) return EXIT_FAILURE;

  if (type != SEF_INIT_LU) {
    if (i2cdriver_subscribe_bus_updates(bus) != OK)
      return EXIT_FAILURE;
    i2cdriver_announce(bus);
  }
  return OK;
}
```

sef_cb_lu_state_save(int result, int flags)

```
static int
sef_cb_lu_state_save(int result, int flags)
{
        ds_publish_u32("bus", bus, DSF_OVERWRITE);
        ds_publish_u32("address", address,
                                           DSF_OVERWRITE);
        return OK;
}
```

lu_state_restore(void)

```
static int
lu_state_restore(void)
{
        /* Restore the state. */
        u32_t value;

        ds_retrieve_u32("bus", &value);
        ds_delete_u32("bus");
        bus = (int) value;

        ds_retrieve_u32("address", &value);
        ds_delete_u32("address");
        address = (int) value;

        return OK;
}
```

bmp085_init(void)

```
static int
bmp085_init(void)
{
        if (version_check() != OK)
                return EXIT_FAILURE;

        if (read_cal_coef() != OK)
                return EXIT_FAILURE;

        return OK;
}
```

libchardriver callbacks

```
static ssize_t
bmp085_read(devminor_t minor, u64_t position,
    endpoint_t endpt, cp_grant_id_t grant,
    size_t size, int flags, cdev_id_t id);

static void
bmp085_other(message * m, int ipc_status);

static struct chardriver bmp085_tab = {
        .cdr_read       = bmp085_read,
        .cdr_other      = bmp085_other
};
```

bmp085_read(...)

```
if (measure(&temperature, &pressure) != OK)
    return EIO;

/* ... fill buffer with measurements ... */

dev_size = (u64_t)strlen(buffer);
if (position >= dev_size) return 0;
if (position + size > dev_size)
    size = (size_t)(dev_size - position);

r = sys_safecopyto(endpt, grant, 0,
   (vir_bytes)(buffer + (size_t)position),size);

return (r != OK) ? r : size;
```

measure(int32_t * temperature, int32_t * pressure)

```
/* ... */

/* trigger temperature reading */
if (i2creg_write8(bus_endpoint, address, CTRL_REG,
                  CMD_TRIG_T) != OK)
    return -1;

micro_delay(UDELAY_T); /* wait for sampling. */

/* read the uncompensated temperature */
if (i2creg_read16(bus_endpoint, address,
                  SENSOR_VAL_MSB_REG, &ut) != OK)
    return -1;

/* ... */

return OK;
```

Tips for Debugging and Minimizing Bugs

- Make many small changes
- Compile and test after each change
- Lots of logging via `minix/log.h`
- Hardware debugger when really stuck
- Avoid hardware simulators when possible

Pre-submission Checklist

- Does the code build without any compiler warnings?
- Does building Minix for the other platform still work?
- Is the driver stable, and does it word as advertised?
- Does the driver have a negligible impact on performance?
- Are the changes broken up into a series of small commits?
- Are there well written commit messages?
- Is the coding style consistent with the NetBSD coding style?
- Have you complied with all applicable licenses?
- Have you given proper credit to collaborators?

Submitting your changes

- Ask for feedback on `minix-dev` Google Group
- Submit a pull request on github
- Respond to feedback and requests for changes

Questions?

Questions?

Contact Info

Internet Relay Chat

tcort on irc.freenode.net

E-Mail

tcort@minix3.org

Twitter

@tomcort

Website

http://www.tomcort.com/