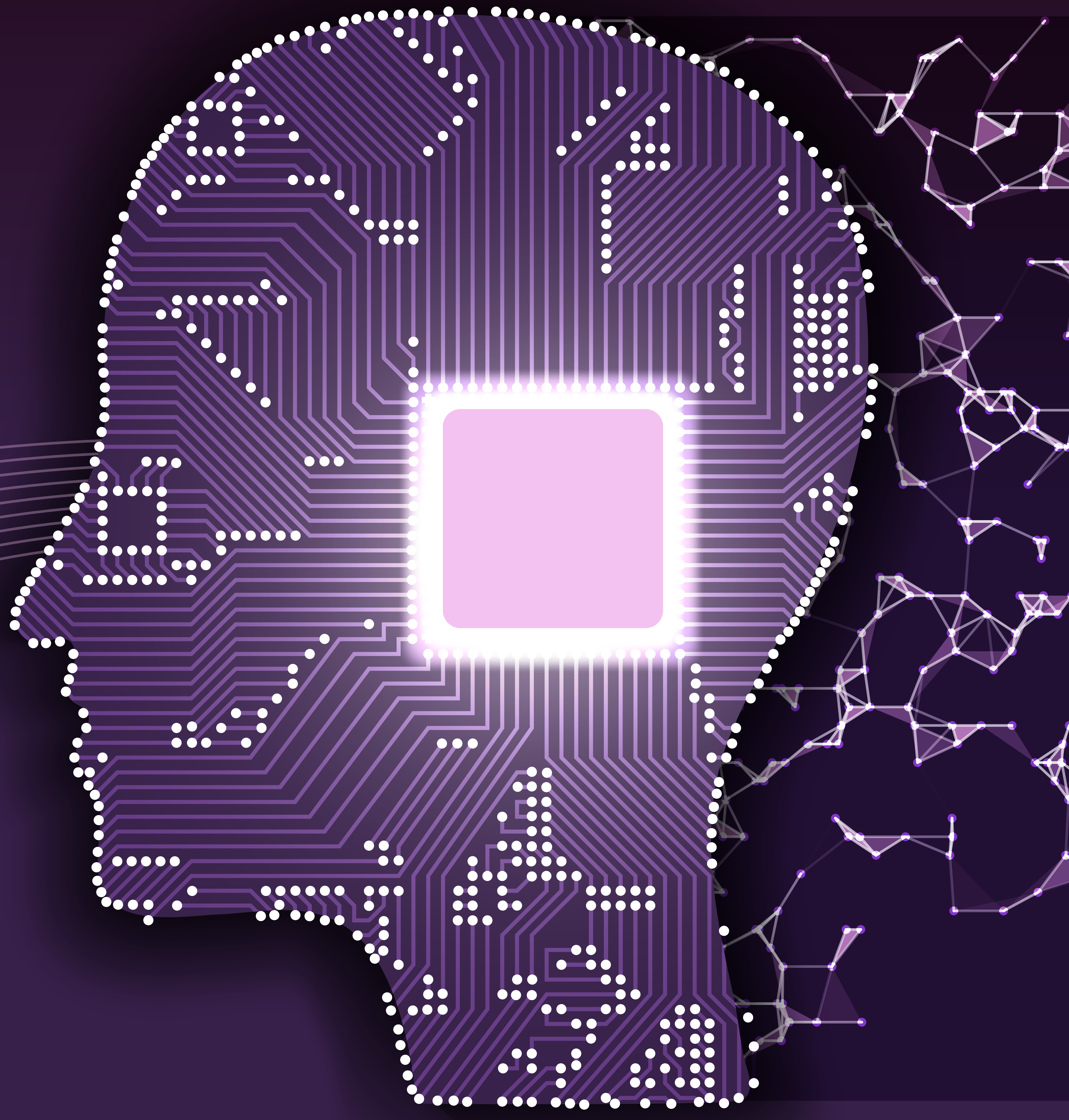
Human Interface Device (HID) Support in FreeBSD 13

BY VLADIMIR KONDRATYEV



The HID class consists primarily of devices that are used by humans to control the operationof computer systems. Typical examples of HID class devices include keyboards, pointing devices like standard mouse devices, trackballs, and joysticks. It was adopted primarily to simplify the installation of such devices.

Prior to the appearance of the HID, devices usually conformed to strictly defined protocols. All hardware design improvements resulted in either overloading the use of data in an existing protocol or the creation of custom device drivers and the propagation of a new protocol to developers. By contrast, all HID-defined devices deliver self-describing packages that may contain any number of data types and formats. The key idea is that information about a HID device is stored in segments of its ROM (read-only memory). These segments are called descriptors. A single HID driver on a computer parses descriptors and enables dynamic association of data with application functionality, which has enabled rapid innovation and development, and prolific diversification of new human-interface devices.

Among descriptors of different types, there is one which any HID device must have independently of physical transport. It is called a HID report descriptor. The report descriptor prescribes each piece of data that the device generates and what the data is measuring. The sample of a report descriptor for a 3-button mouse with wheel and tilt follows:

```
0x05, 0x01,         // Usage Page (Generic Desktop)
0x09, 0x02,         // Usage (Mouse)
0xa1, 0x01,         // Collection (Application)
0x09, 0x01,         //  Usage (Pointer)
0xa1, 0x00,         //  Collection (Physical)
0x05, 0x09,         //   Usage Page (Button)
0x19, 0x01,         //   Usage Minimum (1)
0x29, 0x03,         //   Usage Maximum (3)
0x15, 0x00,         //   Logical Minimum (0)
0x25, 0x01,         //   Logical Maximum (1)
0x95, 0x03,         //   Report Count (3)
0x75, 0x01,         //   Report Size (1)
0x81, 0x02,         //   Input (Data,Var,Abs)
0x95, 0x05,         //   Report Count (5)
0x81, 0x03,         //   Input (Const,Var,Abs)
0x05, 0x01,         //   Usage Page (Generic Desktop)
0x09, 0x30,         //   Usage (X)
0x09, 0x31,         //   Usage (Y)
0x09, 0x38,         //   Usage (Wheel)
0x15, 0x81,         //   Logical Minimum (-127)
0x25, 0x7f,         //   Logical Maximum (127)
0x75, 0x08,         //   Report Size (8)
0x95, 0x03,         //   Report Count (3)
0x81, 0x06,         //   Input (Data,Var,Rel)
0x05, 0x0c,         //   Usage Page (Consumer Devices)
0x0a, 0x38, 0x02,   //   Usage (AC Pan)
0x95, 0x01,         //   Report Count (1)
0x81, 0x06,         //   Input (Data,Var,Rel)
0xc0,               //  End Collection
0xc0,               // End Collection
```

Listing 1. Decoded HID report descriptor for a 3-button mouse with wheel and tilt axes

The initial import of HID support from NetBSD was done in 1998 along with the USB stack. It consisted of a report descriptor parser and 3 drivers based on it: ukbd(4), ums(4) and uhid(4). A userspace version of a report descriptor parser known as libusbhid(3) was imported later in 2000. It became a base of HID support in the bluetooth stack, bthidd(8), committed in 2004. While such a combination of drivers mostly suits desktop needs, it does not cope well with laptops and hand-held devices. Microsoft published HID-over-I2C specifications which were adopted by almost all (with Apple being the main exception) laptop producers. Touch devices got a large market share and a lot of complex devices that combine the functionality of two or more simpler devices appeared e.g., a keyboard with a mouse, or touch screen with a pen tablet and so on. All this necessitated revision of the HID subsystem.

HID Subsystem Architecture

The new HID subsystem is designed as a bus. Any transport bus may provide HID devices and register them with the HID core. The HID bus then loads generic device drivers on top of it. The transport drivers are responsible for raw data transport and device setup/management. The HID core contains helper routines for report-parsing and is responsible for auto discovery of a child device for each Top Level Collection described by the report descriptor. Generic device drivers are responsible for report interpretation and the user-space API. Device specifics and quirks are handled by hidquirk and raw access is handled by the hidraw driver. hidmap is the HID item to the evdev event converter. Generic HID functionality is moved out of USB-HID into a new subsystem under a dev/hid directory and became a separate kernel module. That is something Open/NetBSD did 5 years ago.

```
Hardware-dependent    +-----------------------+
                      |     Transport bus     |
                      |  iicbus(4), usbus(4)  |
                      +-----------------------+
                                  |
                      +-----------------------+
                      | HID transport driver  |
                      | iichid(4), usbhid(4)  |
                      +-----------------------+
                                  |
Hardware-independent              | hid_if.m - HID transport abstraction
                                  |
           +----------------------------------------------+
           |                   HID Core                   |
           |  +------------------+  +------------------+  |
           |  |    hidbus(4)     |  |      hid(4)      |  |
           |  | TLC  management  |  |  report parser   |  |
           |  +------------------+  +------------------+  |
           +----------------------------------------------+
                                  |
+--------------------------------------------------------------------+
|                           HID Utilities                            |
|  +------------------+  +------------------+  +------------------+  |
|  |    hidraw(4)     |  |   hidquirk(4)    |  |    hidmap(4)     |  |
|  |   raw access     |  |                  |  | HID->evdev  1:1  |  |
|  +------------------+  +------------------+  +------------------+  |
+--------------------------------------------------------------------+
                                /     \
                ______________/        \____________
               /          /               \         \
   +------------+ +-------------+ +------------+ +---------------+
   |   hms(4)   | |   hkbd(4)   | |   hmt(4)   | | custom driver |
   +------------+ +-------------+ +------------+ +---------------+
```

HID subsystem architecture.

HID Transport Drivers

A transport bus normally provides hotplug detection or device enumeration KPIs to the transport drivers. Transport drivers use this to find any suitable HID device. They allocate HID device resources and attach hidbus. Hidbus is never aware of which transport drivers are available and is not interested in it. It is only interested in child devices.

Transport drivers implement an abstract HID transport interface that provides independent access to HID capabilities and functions through the device tree. A kobj-based HID interface can be found in sys/dev/hid/hid_if.m. Once hidbus child is attached, the bus methods are used by HID core to communicate with the device. Currently, the FreeBSD kernel includes support for USB, and I2C drivers.

hidbus

hidbus is a driver that provides support for multiple HID driver attachments to a single HID transport back-end. This ability existed in Net/OpenBSD from the day 0 (uhidev and ihidev drivers) but has never been ported to FreeBSD. Unlike Net/OpenBSD we do not use report number alone to a distinct report source, but we follow the MS way and use a Top Level Collection (TLC) usage to which the report belongs.

A TLC is a grouping of functionality that targets a particular software consumer (or type of consumer) of the functionality. An operating system uses the Usage associated with this collection to link the device to its controlling application or driver. Common examples are a keyboard or mouse. A keyboard with an integrated pointing device could contain two different application collections. The HID device describes the purpose of each TLC to allow the consumers of HID functionality to identify the TLC in which they might be interested. Hidbus generates a child device for each TLC described by the Report Descriptor and adds PnP strings to allow devd/devmatch to detect the proper driver. While running, hidbus broadcasts data generated by transport drivers to all the children.

```
0x05, 0x01,        // Usage Page (Generic Desktop Ctrls)
0x09, 0x06,        // Usage (Keyboard)
0xA1, 0x01,        // Collection (Application)
0x05, 0x07,        //   Usage Page (Kbrd/Keypad)
0x85, 0x01,        //   Report ID (1)
0x19, 0xE0,        //   Usage Minimum (0xE0)
0x29, 0xE7,        //   Usage Maximum (0xE7)
0x15, 0x00,        //   Logical Minimum (0)
0x25, 0x01,        //   Logical Maximum (1)
0x75, 0x01,        //   Report Size (1)
0x95, 0x08,        //   Report Count (8)
0x81, 0x02,        //   Input (Data,Var,Abs)
0x95, 0x01,        //   Report Count (1)
0x75, 0x08,        //   Report Size (8)
0x81, 0x01,        //   Input (Const,Array,Abs)
0x95, 0x06,        //   Report Count (6)
0x75, 0x08,        //   Report Size (8)
0x15, 0x00,        //   Logical Minimum (0)
0x26, 0xA4, 0x00,  //   Logical Maximum (164)
0x05, 0x07,        //   Usage Page (Kbrd/Keypad)
0x19, 0x00,        //   Usage Minimum (0x00)
0x29, 0xA4,        //   Usage Maximum (0xA4)
0x81, 0x00,        //   Input (Data,Array,Abs)
0xC0,              // End Collection
0x05, 0x01,        // Usage Page (Generic Desktop)
0x09, 0x02,        // Usage (Mouse)
0xa1, 0x01,        // Collection (Application)
0x09, 0x01,        //  Usage (Pointer)
0xa1, 0x00,        //  Collection (Physical)
0x85, 0x02,        //   Report ID (2)
0x05, 0x09,        //   Usage Page (Button)
0x19, 0x01,        //   Usage Minimum (1)
0x29, 0x03,        //   Usage Maximum (3)
0x15, 0x00,        //   Logical Minimum (0)
0x25, 0x01,        //   Logical Maximum (1)
```

```
    0x95, 0x03,         //   Report Count (3)
    0x75, 0x01,         //   Report Size (1)
    0x81, 0x02,         //   Input (Data,Var,Abs)
    0x95, 0x05,         //   Report Count (5)
    0x81, 0x03,         //   Input (Const,Var,Abs)
    0x05, 0x01,         //   Usage Page (Generic Desktop)
    0x09, 0x30,         //   Usage (X)
    0x09, 0x31,         //   Usage (Y)
    0x15, 0x81,         //   Logical Minimum (-127)
    0x25, 0x7f,         //   Logical Maximum (127)
    0x75, 0x08,         //   Report Size (8)
    0x95, 0x02,         //   Report Count (2)
    0x81, 0x06,         //   Input (Data,Var,Rel)
    0xc0,               //  End Collection
    0xc0,               // End Collection
```

Listing 2. HID report descriptor for keyboard with integrated mouse consisting of 2 TLCs.

hidmap

hidmap is a generic HID item value to an evdev event conversion engine that makes it possible to write HID drivers in a mostly declarative way by defining translation tables. The motivation behind its creation was that existing USB-HID drivers are overgrown in size due to following factors:

- USB transfer handling
- Character device support code
- Protocol conversion routines e.g. HID to sysmouse or HID to AT kbd set 1
- Long hid_locate() and hid_get_data() chains in report parsers

p.1 is eliminated with help of the transport abstraction layer.

To solve p.2 support for legacy, the mouse interface was dropped. We use character device handlers built in evdev.

To reduce the amount of code required by p.3 and p.4, hidmap was created. It is based on the fact that HID and evdev are close relatives and we can directly map many HID usages to evdev events. Listing 3 illustrates a sample mapping of HID usages to evdev events for the mouse report from Listing 1.

```
                       HID Usage                        mapped evdev event
                       ---------                        ------------------
0x05, 0x09,        //  Usage Page (Button)
0x19, 0x01,        //  Usage Minimum (1)                BTN_LEFT   (BTN_MOUSE+0)
0x29, 0x08,        //  Usage Maximum (3)                BTN_RIGHT  (BTN_MOUSE+1)
0x95, 0x08,        //  Report Count (3)                 BTN_MIDDLE (BTN_MOUSE+2)
0x81, 0x02,        //  Input (Data,Var,Abs)
0x05, 0x01,        //  Usage Page (Generic Desktop)
0x09, 0x30,        //  Usage (X)                        REL_X
0x09, 0x31,        //  Usage (Y)                        REL_Y
0x09, 0x38,        //  Usage (Wheel)                    REL_WHEEL
0x95, 0x03,        //  Report Count (3)
0x81, 0x06,        //  Input (Data,Var,Rel)
0x05, 0x0c,        //  Usage Page (Consumer Devices)
0x0a, 0x38, 0x02,  //  Usage (AC Pan)                   REL_HWHEEL
```

```
0x95, 0x01,        //   Report Count (1)
0x81, 0x06,        //   Input (Data,Var,Rel)
```

Listing 3. Mapping of HID usages to evdev events for mouse report from Listing 1.

With help of hidmap, the mouse driver for such a device fits in a couple dozen lines of code. See Listing 4.

```
/* my_mouse's HID usage to evdev event mappings */
static const struct hidmap_item my_mouse_map[] = {
  { HIDMAP_REL(HUP_GENERIC_DESKTOP, HUG_X,    REL_X) },
  { HIDMAP_REL(HUP_GENERIC_DESKTOP, HUG_Y,    REL_Y) },
  { HIDMAP_REL(HUP_GENERIC_DESKTOP, HUG_WHEEL,  REL_WHEEL) },
  { HIDMAP_REL(HUP_CONSUMER,        HUC_AC_PAN, REL_HWHEEL) },
  { HIDMAP_KEY_RANGE(HUP_BUTTON,    1, 3,     BTN_MOUSE) },
};

/* A match on these entries will load my_mouse */
static const struct hid_device_id my_mouse_devs[] = {
  { HID_TLC(HUP_GENERIC_DESKTOP, HUG_MOUSE) },
};

static int
my_mouse_probe(device_t dev)
{
  return (HIDMAP_PROBE(device_get_softc(dev), dev,
      my_mouse_devs, my_mouse_map, "My mouse"));
}

static int
my_mouse_attach(device_t dev)
{
  return (hidmap_attach(device_get_softc(dev)));
}

static int
my_mouse_detach(device_t dev)
{
  return (hidmap_detach(device_get_softc(dev)));
}
```

Listing 4. Sample of hidmap-based driver for mouse report from Listing 1.

For example, the real FreeBSD mouse driver has shrunk from ~1200LOC in traditional ums(4) to ~330 LOC in hms(4) based on new HID KPI. Also, it got I2C and absolute coordinate support missing in ums(4) and drift suppression code required to work around issues connected to missing GPIO interrupt support in FreeBSD on x86. This simplicity allowed the author and Greg V to create a bunch of hidmap-based drivers that are bundled with FreeBSD 13+, namely:

- hms - HID mouse driver.
- hcons - Consumer page AKA Multimedia keys driver.
- hsctrl - System Controls page (Power/Sleep keys) driver.
- hpen - Generic / MS Windows compatible HID pen tablet driver.
- hgame - Game controller and joystick driver.

- xb360gp - driver for Xbox360-compatible game controllers.
- ps4dshock - Sony DualShock 4 gamepad driver.

There are also some drivers which are not based on hidmap like hkbd (4)and hmt(4). They are ports of existing USB-HID drivers, namely ukbd(4) and wmt(4) to the new infrastructure. They add support for I2C keyboards and I2C MT touchpads/touchscreens.

Other Modules

The HID subsystem contains two other modules that can be loaded optionally:

hidraw(4) - driver for raw access to HID devices resembling uhid(4). Unlike uhid(4), it allows access to devices that were claimed by other drivers and supports both uhid and Linux hidraw interfaces.

Hidquirk(4) - quirk module which was mostly copied from existing USB-HID.

Conclusion and Further Work

The FreeBSD HID subsystem is still under development but is already used by many. Recent work added awaited support for widely used hardware like I2C touchpads and touchscreens, multimedia keys on a USB keyboard, game controllers, absolute mice found on many VMs and so on. This has led to improved user experience in areas where we lagged behind other OS-es including other BSDs. But there are lot of things still left in TODO list e.g:

- Implement usrhid, a user-space transport driver that can create kernel hid-devices for each device connected to the user-space controlled bus. Existing Linux uhid protocol can be used as a starting point. It defines an API for providing I/O events from the kernel to user-space and vice versa.
- Convert bthidd(8) to use usrhid to consolidate HID support between the kernel and user spaces.
- Complete evdev-awared WIP moused https://github.com/wulf7/moused and replace our in-base moused(8) with it. This is required as new hms and hmt drivers do not support our legacy sys/mouse.h interface.
- Enable usbhid(4) by default and start deprecation of ums(4), ukbd(4) and other legacy USB-HID drivers along with deprecation of all mouse(4)/sysmouse(4) stuff in the kernel and base system

VLADIMIR KONDRATYEV is an ex-FreeBSD system administrator and currently a core banking system specialist. He has been a FreeBSD committer since 2017 and a FreeBSD desktop user for the last ~20 years. In his spare time, he tries to improve the desktop experience, contributing mostly to input device drivers.