Linux USB HID gadget driver

Introduction

The HID Gadget driver provides emulation of USB Human Interface Devices (HID). The basic HID handling is done in the kernel, and HID reports can be sent/received through I/O on the /dev/hidgX character devices.

For more details about HID, see the developer page on http://www.usb.org/developers/hidpage/

Configuration

g_hid is a platform driver, so to use it you need to add struct platform_device(s) to your platform code defining the HID function descriptors you want to use - E.G. something like:

```
#include <linux/platform_device.h>
#include <linux/usb/g_hid.h>

/* hid descriptor for a keyboard */
static struct hidg_func_descriptor my_hid_data = {
        .subclass               = 0, /* No subclass */
        .protocol               = 1, /* Keyboard */
        .report_length          = 8,
        .report_desc_length     = 63,
        .report_desc            = {
                0x05, 0x01,     /* USAGE_PAGE (Generic Desktop)          */
                0x09, 0x06,     /* USAGE (Keyboard)                      */
                0xa1, 0x01,     /* COLLECTION (Application)              */
                0x05, 0x07,     /*   USAGE_PAGE (Keyboard)               */
                0x19, 0xe0,     /*   USAGE_MINIMUM (Keyboard LeftControl) */
                0x29, 0xe7,     /*   USAGE_MAXIMUM (Keyboard Right GUI)  */
                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, 0x03,     /*   INPUT (Cnst,Var,Abs)                */
                0x95, 0x05,     /*   REPORT_COUNT (5)                    */
                0x75, 0x01,     /*   REPORT_SIZE (1)                     */
                0x05, 0x08,     /*   USAGE_PAGE (LEDs)                   */
                0x19, 0x01,     /*   USAGE_MINIMUM (Num Lock)            */
                0x29, 0x05,     /*   USAGE_MAXIMUM (Kana)                */
                0x91, 0x02,     /*   OUTPUT (Data,Var,Abs)               */
                0x95, 0x01,     /*   REPORT_COUNT (1)                    */
                0x75, 0x03,     /*   REPORT_SIZE (3)                     */
                0x91, 0x03,     /*   OUTPUT (Cnst,Var,Abs)               */
                0x95, 0x06,     /*   REPORT_COUNT (6)                    */
                0x75, 0x08,     /*   REPORT_SIZE (8)                     */
                0x15, 0x00,     /*   LOGICAL_MINIMUM (0)                 */
```

```
                0x25, 0x65,     /*   LOGICAL_MAXIMUM (101)                */
                0x05, 0x07,     /*   USAGE_PAGE (Keyboard)                */
                0x19, 0x00,     /*   USAGE_MINIMUM (Reserved)             */
                0x29, 0x65,     /*   USAGE_MAXIMUM (Keyboard Application) */
                0x81, 0x00,     /*   INPUT (Data,Ary,Abs)                 */
                0xc0            /* END_COLLECTION                         */
        }
};

static struct platform_device my_hid = {
        .name                   = "hidg",
        .id                     = 0,
        .num_resources          = 0,
        .resource               = 0,
        .dev.platform_data      = &my_hid_data,
};
```

You can add as many HID functions as you want, only limited by the amount of interrupt endpoints your gadget driver supports.

Send and receive HID reports

HID reports can be sent/received using read/write on the /dev/hidgX character devices. See below for an example program to do this.

hid_gadget_test is a small interactive program to test the HID gadget driver. To use, point it at a hidg device and set the device type (keyboard / mouse / joystick) - E.G.:

```
        # hid_gadget_test /dev/hidg0 keyboard
```

You are now in the prompt of hid_gadget_test. You can type any combination of options and values. Available options and values are listed at program start. In keyboard mode you can send up to six values.

For example type: g i s t r --left-shift

Hit return and the corresponding report will be sent by the HID gadget.

Another interesting example is the caps lock test. Type --caps-lock and hit return. A report is then sent by the gadget and you should receive the host answer, corresponding to the caps lock LED status.

```
        --caps-lock
        recv report:2
```

With this command:

```
        # hid_gadget_test /dev/hidg1 mouse
```

You can test the mouse emulation. Values are two signed numbers.

Sample code

```
/* hid_gadget_test */

#include <pthread.h>
#include <string.h>
#include <stdio.h>
#include <ctype.h>
#include <fcntl.h>
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define BUF_LEN 512

struct options {
        const char    *opt;
        unsigned char val;
};

static struct options kmod[] = {
        {.opt = "--left-ctrl",          .val = 0x01},
        {.opt = "--right-ctrl",         .val = 0x10},
        {.opt = "--left-shift",         .val = 0x02},
        {.opt = "--right-shift",        .val = 0x20},
        {.opt = "--left-alt",           .val = 0x04},
        {.opt = "--right-alt",          .val = 0x40},
        {.opt = "--left-meta",          .val = 0x08},
        {.opt = "--right-meta",         .val = 0x80},
        {.opt = NULL}
};

static struct options kval[] = {
        {.opt = "--return",     .val = 0x28},
        {.opt = "--esc",        .val = 0x29},
        {.opt = "--bckspc",     .val = 0x2a},
        {.opt = "--tab",        .val = 0x2b},
        {.opt = "--spacebar",   .val = 0x2c},
        {.opt = "--caps-lock",  .val = 0x39},
        {.opt = "--f1",         .val = 0x3a},
        {.opt = "--f2",         .val = 0x3b},
        {.opt = "--f3",         .val = 0x3c},
        {.opt = "--f4",         .val = 0x3d},
        {.opt = "--f5",         .val = 0x3e},
        {.opt = "--f6",         .val = 0x3f},
        {.opt = "--f7",         .val = 0x40},
        {.opt = "--f8",         .val = 0x41},
        {.opt = "--f9",         .val = 0x42},
        {.opt = "--f10",        .val = 0x43},
        {.opt = "--f11",        .val = 0x44},
        {.opt = "--f12",        .val = 0x45},
        {.opt = "--insert",     .val = 0x49},
        {.opt = "--home",       .val = 0x4a},
        {.opt = "--pageup",     .val = 0x4b},
```

```
        {.opt = "--del",        .val = 0x4c},
        {.opt = "--end",        .val = 0x4d},
        {.opt = "--pagedown",   .val = 0x4e},
        {.opt = "--right",      .val = 0x4f},
        {.opt = "--left",       .val = 0x50},
        {.opt = "--down",       .val = 0x51},
        {.opt = "--kp-enter",   .val = 0x58},
        {.opt = "--up",         .val = 0x52},
        {.opt = "--num-lock",   .val = 0x53},
        {.opt = NULL}
};

int keyboard_fill_report(char report[8], char buf[BUF_LEN], int *hold)
{
        char *tok = strtok(buf, " ");
        int key = 0;
        int i = 0;

        for (; tok != NULL; tok = strtok(NULL, " ")) {

                if (strcmp(tok, "--quit") == 0)
                        return -1;

                if (strcmp(tok, "--hold") == 0) {
                        *hold = 1;
                        continue;
                }

                if (key < 6) {
                        for (i = 0; kval[i].opt != NULL; i++)
                                if (strcmp(tok, kval[i].opt) == 0) {
                                        report[2 + key++] = kval[i].val;
                                        break;
                                }
                        if (kval[i].opt != NULL)
                                continue;
                }

                if (key < 6)
                        if (islower(tok[0])) {
                                report[2 + key++] = (tok[0] - ('a' - 0x04));
                                continue;
                        }

                for (i = 0; kmod[i].opt != NULL; i++)
                        if (strcmp(tok, kmod[i].opt) == 0) {
                                report[0] = report[0] | kmod[i].val;
                                break;
                        }
                if (kmod[i].opt != NULL)
                        continue;

                if (key < 6)
                        fprintf(stderr, "unknown option: %s\n", tok);
        }
        return 8;
```

```
}

static struct options mmod[] = {
        {.opt = "--b1", .val = 0x01},
        {.opt = "--b2", .val = 0x02},
        {.opt = "--b3", .val = 0x04},
        {.opt = NULL}
};

int mouse_fill_report(char report[8], char buf[BUF_LEN], int *hold)
{
        char *tok = strtok(buf, " ");
        int mvt = 0;
        int i = 0;
        for (; tok != NULL; tok = strtok(NULL, " ")) {

                if (strcmp(tok, "--quit") == 0)
                        return -1;

                if (strcmp(tok, "--hold") == 0) {
                        *hold = 1;
                        continue;
                }

                for (i = 0; mmod[i].opt != NULL; i++)
                        if (strcmp(tok, mmod[i].opt) == 0) {
                                report[0] = report[0] | mmod[i].val;
                                break;
                        }
                if (mmod[i].opt != NULL)
                        continue;

                if (!(tok[0] == '-' && tok[1] == '-') && mvt < 2) {
                        errno = 0;
                        report[1 + mvt++] = (char)strtol(tok, NULL, 0);
                        if (errno != 0) {
                                fprintf(stderr, "Bad value:'%s'\n", tok);
                                report[1 + mvt--] = 0;
                        }
                        continue;
                }

                fprintf(stderr, "unknown option: %s\n", tok);
        }
        return 3;
}

static struct options jmod[] = {
        {.opt = "--b1",         .val = 0x10},
        {.opt = "--b2",         .val = 0x20},
        {.opt = "--b3",         .val = 0x40},
        {.opt = "--b4",         .val = 0x80},
        {.opt = "--hat1",       .val = 0x00},
        {.opt = "--hat2",       .val = 0x01},
        {.opt = "--hat3",       .val = 0x02},
        {.opt = "--hat4",       .val = 0x03},
```

```
        {.opt = "--hatneutral", .val = 0x04},
        {.opt = NULL}
};

int joystick_fill_report(char report[8], char buf[BUF_LEN], int *hold)
{
        char *tok = strtok(buf, " ");
        int mvt = 0;
        int i = 0;

        *hold = 1;

        /* set default hat position: neutral */
        report[3] = 0x04;

        for (; tok != NULL; tok = strtok(NULL, " ")) {

                if (strcmp(tok, "--quit") == 0)
                        return -1;

                for (i = 0; jmod[i].opt != NULL; i++)
                        if (strcmp(tok, jmod[i].opt) == 0) {
                                report[3] = (report[3] & 0xF0) | jmod[i].val;
                                break;
                        }
                if (jmod[i].opt != NULL)
                        continue;

                if (!(tok[0] == '-' && tok[1] == '-') && mvt < 3) {
                        errno = 0;
                        report[mvt++] = (char)strtol(tok, NULL, 0);
                        if (errno != 0) {
                                fprintf(stderr, "Bad value:'%s'\n", tok);
                                report[mvt--] = 0;
                        }
                        continue;
                }

                fprintf(stderr, "unknown option: %s\n", tok);
        }
        return 4;
}

void print_options(char c)
{
        int i = 0;

        if (c == 'k') {
                printf("        keyboard options:\n"
                       "                        --hold\n");
                for (i = 0; kmod[i].opt != NULL; i++)
                        printf("\t\t%s\n", kmod[i].opt);
                printf("\n      keyboard values:\n"
                       "                        [a-z] or\n");
                for (i = 0; kval[i].opt != NULL; i++)
                        printf("\t\t%-8s%s", kval[i].opt, i % 2 ? "\n" : "");
```

```
                printf("\n");
        } else if (c == 'm') {
                printf("        mouse options:\n"
                       "                --hold\n");
                for (i = 0; mmod[i].opt != NULL; i++)
                        printf("\t\t%s\n", mmod[i].opt);
                printf("\n      mouse values:\n"
                       "                Two signed numbers\n"
                       "--quit to close\n");
        } else {
                printf("        joystick options:\n");
                for (i = 0; jmod[i].opt != NULL; i++)
                        printf("\t\t%s\n", jmod[i].opt);
                printf("\n      joystick values:\n"
                       "                three signed numbers\n"
                       "--quit to close\n");
        }
}

int main(int argc, const char *argv[])
{
        const char *filename = NULL;
        int fd = 0;
        char buf[BUF_LEN];
        int cmd_len;
        char report[8];
        int to_send = 8;
        int hold = 0;
        fd_set rfds;
        int retval, i;

        if (argc < 3) {
                fprintf(stderr, "Usage: %s devname mouse|keyboard|joystick\n",
                        argv[0]);
                return 1;
        }

        if (argv[2][0] != 'k' && argv[2][0] != 'm' && argv[2][0] != 'j')
          return 2;

        filename = argv[1];

        if ((fd = open(filename, O_RDWR, 0666)) == -1) {
                perror(filename);
                return 3;
        }

        print_options(argv[2][0]);

        while (42) {

                FD_ZERO(&rfds);
                FD_SET(STDIN_FILENO, &rfds);
                FD_SET(fd, &rfds);

                retval = select(fd + 1, &rfds, NULL, NULL, NULL);
```

```
                if (retval == -1 && errno == EINTR)
                        continue;
                if (retval < 0) {
                        perror("select()");
                        return 4;
                }

                if (FD_ISSET(fd, &rfds)) {
                        cmd_len = read(fd, buf, BUF_LEN - 1);
                        printf("recv report:");
                        for (i = 0; i < cmd_len; i++)
                                printf(" %02x", buf[i]);
                        printf("\n");
                }

                if (FD_ISSET(STDIN_FILENO, &rfds)) {
                        memset(report, 0x0, sizeof(report));
                        cmd_len = read(STDIN_FILENO, buf, BUF_LEN - 1);

                        if (cmd_len == 0)
                                break;

                        buf[cmd_len - 1] = '\0';
                        hold = 0;

                        memset(report, 0x0, sizeof(report));
                        if (argv[2][0] == 'k')
                                to_send = keyboard_fill_report(report, buf,
&hold);
                        else if (argv[2][0] == 'm')
                                to_send = mouse_fill_report(report, buf, &hold);
                        else
                                to_send = joystick_fill_report(report, buf,
&hold);

                        if (to_send == -1)
                                break;

                        if (write(fd, report, to_send) != to_send) {
                                perror(filename);
                                return 5;
                        }
                        if (!hold) {
                                memset(report, 0x0, sizeof(report));
                                if (write(fd, report, to_send) != to_send) {
                                        perror(filename);
                                        return 6;
                                }
                        }
                }
        }

        close(fd);
        return 0;
}
```