uinput module

Introduction

uinput is a kernel module that makes it possible to emulate input devices from userspace. By writing to /dev/uinput (or /dev/input/uinput) device, a process can create a virtual input device with specific capabilities. Once this virtual device is created, the process can send events through it, that will be delivered to userspace and in-kernel consumers.

Interface

```
linux/uinput.h
```

The uinput header defines ioctls to create, set up, and destroy virtual devices.

libevdev

libevdev is a wrapper library for evdev devices that provides interfaces to create uinput devices and send events. libevdev is less error-prone than accessing uinput directly, and should be considered for new software.

For examples and more information about libevdev: https://www.freedesktop.org/software/libevdev/doc/latest/

Examples

Keyboard events

This first example shows how to create a new virtual device, and how to send a key event. All default imports and error handlers were removed for the sake of simplicity.

```c
#include <linux/uinput.h>

void emit(int fd, int type, int code, int val)
{
   struct input_event ie;

   ie.type = type;
   ie.code = code;
   ie.value = val;
   /* timestamp values below are ignored */
   ie.time.tv_sec = 0;
   ie.time.tv_usec = 0;

   write(fd, &ie, sizeof(ie));
}

int main(void)
{
   struct uinput_setup usetup;

   int fd = open("/dev/uinput", O_WRONLY | O_NONBLOCK);


   /*
    * The ioctls below will enable the device that is about to be
    * created, to pass key events, in this case the space key.
    */
   ioctl(fd, UI_SET_EVBIT, EV_KEY);
   ioctl(fd, UI_SET_KEYBIT, KEY_SPACE);

   memset(&usetup, 0, sizeof(usetup));
   usetup.id.bustype = BUS_USB;
   usetup.id.vendor = 0x1234; /* sample vendor */
   usetup.id.product = 0x5678; /* sample product */
   strcpy(usetup.name, "Example device");

   ioctl(fd, UI_DEV_SETUP, &usetup);
   ioctl(fd, UI_DEV_CREATE);

   /*
    * On UI_DEV_CREATE the kernel will create the device node for this
    * device. We are inserting a pause here so that userspace has time
    * to detect, initialize the new device, and can start listening to
    * the event, otherwise it will not notice the event we are about
    * to send. This pause is only needed in our example code!
```

```
    */
   sleep(1);

   /* Key press, report the event, send key release, and report again */
   emit(fd, EV_KEY, KEY_SPACE, 1);
   emit(fd, EV_SYN, SYN_REPORT, 0);
   emit(fd, EV_KEY, KEY_SPACE, 0);
   emit(fd, EV_SYN, SYN_REPORT, 0);

   /*
    * Give userspace some time to read the events before we destroy the
    * device with UI_DEV_DESTROY.
    */
   sleep(1);

   ioctl(fd, UI_DEV_DESTROY);
   close(fd);

   return 0;
}
```

Mouse movements

This example shows how to create a virtual device that behaves like a physical mouse.

```
#include <linux/uinput.h>

/* emit function is identical to of the first example */

int main(void)
{
   struct uinput_setup usetup;
   int i = 50;

   int fd = open("/dev/uinput", O_WRONLY | O_NONBLOCK);

   /* enable mouse button left and relative events */
   ioctl(fd, UI_SET_EVBIT, EV_KEY);
   ioctl(fd, UI_SET_KEYBIT, BTN_LEFT);

   ioctl(fd, UI_SET_EVBIT, EV_REL);
   ioctl(fd, UI_SET_RELBIT, REL_X);
   ioctl(fd, UI_SET_RELBIT, REL_Y);

   memset(&usetup, 0, sizeof(usetup));
   usetup.id.bustype = BUS_USB;
   usetup.id.vendor = 0x1234; /* sample vendor */
   usetup.id.product = 0x5678; /* sample product */
   strcpy(usetup.name, "Example device");

   ioctl(fd, UI_DEV_SETUP, &usetup);
   ioctl(fd, UI_DEV_CREATE);

   /*
    * On UI_DEV_CREATE the kernel will create the device node for this
    * device. We are inserting a pause here so that userspace has time
    * to detect, initialize the new device, and can start listening to
    * the event, otherwise it will not notice the event we are about
    * to send. This pause is only needed in our example code!
    */
   sleep(1);

   /* Move the mouse diagonally, 5 units per axis */
   while (i--) {
      emit(fd, EV_REL, REL_X, 5);
      emit(fd, EV_REL, REL_Y, 5);
      emit(fd, EV_SYN, SYN_REPORT, 0);
      usleep(15000);
   }

   /*
    * Give userspace some time to read the events before we destroy the
    * device with UI_DEV_DESTROY.
    */
   sleep(1);

   ioctl(fd, UI_DEV_DESTROY);
   close(fd);

   return 0;
}
```

uinput old interface

Before uinput version 5, there wasn't a dedicated ioctl to set up a virtual device. Programs supporting older versions of uinput interface need to fill a uinput_user_dev structure and write it to the uinput file descriptor to configure the new uinput device. New code should not use the old interface but interact with uinput via ioctl calls, or use libevdev.

```
#include <linux/uinput.h>

/* emit function is identical to of the first example */

int main(void)
{
   struct uinput_user_dev uud;
   int version, rc, fd;

   fd = open("/dev/uinput", O_WRONLY | O_NONBLOCK);
   rc = ioctl(fd, UI_GET_VERSION, &version);

   if (rc == 0 && version >= 5) {
      /* use UI_DEV_SETUP */
      return 0;
   }

   /*
    * The ioctls below will enable the device that is about to be
    * created, to pass key events, in this case the space key.
    */
   ioctl(fd, UI_SET_EVBIT, EV_KEY);
   ioctl(fd, UI_SET_KEYBIT, KEY_SPACE);

   memset(&uud, 0, sizeof(uud));
   snprintf(uud.name, UINPUT_MAX_NAME_SIZE, "uinput old interface");
   write(fd, &uud, sizeof(uud));

   ioctl(fd, UI_DEV_CREATE);

   /*
    * On UI_DEV_CREATE the kernel will create the device node for this
    * device. We are inserting a pause here so that userspace has time
    * to detect, initialize the new device, and can start listening to
    * the event, otherwise it will not notice the event we are about
    * to send. This pause is only needed in our example code!
    */
   sleep(1);

   /* Key press, report the event, send key release, and report again */
   emit(fd, EV_KEY, KEY_SPACE, 1);
   emit(fd, EV_SYN, SYN_REPORT, 0);
   emit(fd, EV_KEY, KEY_SPACE, 0);
   emit(fd, EV_SYN, SYN_REPORT, 0);

   /*
    * Give userspace some time to read the events before we destroy the
    * device with UI_DEV_DESTROY.
    */
   sleep(1);

   ioctl(fd, UI_DEV_DESTROY);

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