

Using uinput driver in Linux- 2.6.x to send user input

Mehul Patel



Using uinput driver in Linux-2.6.x to send user input

- **Introduction:**

The Linux 2.6.x provides a “uinput” driver, which helps users to inject data to the Linux kernel. This is very useful while writing applications to interface customized input devices like wireless joystick, keyboard etc.

The driver uses /dev/uinput device to send data to kernel space which in turn send data to X-windows OR active shell. This feature can be used to perform automated shell scripts which involve graphical user inputs.

Uinput is configured as a loadable module in most of the Linux kernels. You can load uinput driver by giving the following commands.

```
$ modprobe uinput  
$ lsmod
```

The “lsmod” command lists all the drivers loaded in the Linux system. You should see “uinput” driver in the list.

The next step is to develop a sample application. This application will send the user key sequence to kernel which is in turn sent to X-Windows or shell.

- **Opening an input device:**

```
// Open the input device  
uinp_fd = open("/dev/uinput", O_WRONLY | O_NDELAY);  
  
if (uinp_fd == NULL)  
{  
    printf("Unable to open /dev/uinput\n");  
    return -1;  
}
```

After opening device you have to configure the uinput device parameters (mouse, keyboard, etc) using the ioctl() function such as:

```
ioctl (out_fd, UI_SET_EVBIT, EV_KEY);  
ioctl (out_fd, UI_SET_EVBIT, EV_REP);
```

The EV_KEY and EV_REP inform the uinput driver as the event is a keyboard event and the key value contains the key repetition property.

- **Sending events to kernel:**

All the events coming from the user program will be carried to the kernel space through the structure "struct input_event" defined in kernels "/usr/include/linux/input.h".

A keyboard can be generated using following piece of code:

```
event.type = EV_KEY;  
event.code = KEY_ENTER;  
event.value = 1;  
write (uinp_fd, &event, sizeof(event));
```

The above example will send ENTER key to kernel. This key event in-turn will be sent to user space application. All the key definitions are defined in "/usr/include/linux/input.h" file.

You can use the following sample code to test Linux uinput interface.

```
// uinput.c  
  
#include <string.h>  
#include <stdio.h>  
#include <sys/types.h>  
#include <sys/stat.h>  
#include <fcntl.h>  
#include <linux/input.h>  
#include <linux/uinput.h>  
#include <stdio.h>  
#include <sys/time.h>  
#include <sys/types.h>  
#include <unistd.h>  
  
/* Globals */  
static int uinp_fd = -1;  
struct uinput_user_dev uinp; // uInput device structure  
struct input_event event; // Input device structure  
  
/* Setup the uinput device */  
  
int setup_uinput_device()  
{  
    // Temporary variable  
    int i=0;  
  
    // Open the input device  
    uinp_fd = open("/dev/uinput", O_WRONLY | O_NDELAY);  
  
    if (uinp_fd == NULL)  
    {
```

```

printf("Unable to open /dev/uinput\n");
return -1;
}

memset(&uinp,0,sizeof(uinp)); // Initialize the uInput device to NULL

strncpy(uinp.name, "PolyVision Touch Screen", UINPUT_MAX_NAME_SIZE);
uinp.id.version = 4;
uinp.id.bustype = BUS_USB;

// Setup the uinput device
ioctl(uinp_fd, UI_SET_EVBIT, EV_KEY);
ioctl(uinp_fd, UI_SET_EVBIT, EV_REL);
ioctl(uinp_fd, UI_SET_RELBIT, REL_X);
ioctl(uinp_fd, UI_SET_RELBIT, REL_Y);

for (i=0; i < 256; i++) {
    ioctl(uinp_fd, UI_SET_KEYBIT, i);
}

ioctl(uinp_fd, UI_SET_KEYBIT, BTN_MOUSE);
ioctl(uinp_fd, UI_SET_KEYBIT, BTN_TOUCH);

ioctl(uinp_fd, UI_SET_KEYBIT, BTN_MOUSE);
ioctl(uinp_fd, UI_SET_KEYBIT, BTN_LEFT);
ioctl(uinp_fd, UI_SET_KEYBIT, BTN_MIDDLE);
ioctl(uinp_fd, UI_SET_KEYBIT, BTN_RIGHT);
ioctl(uinp_fd, UI_SET_KEYBIT, BTN_FORWARD);
ioctl(uinp_fd, UI_SET_KEYBIT, BTN_BACK);

/* Create input device into input sub-system */
write(uinp_fd, &uinp, sizeof(uinp));

if (ioctl(uinp_fd, UI_DEV_CREATE))
{
    printf("Unable to create UINPUT device.");
    return -1;
}
return 1;
}

void send_click_events( )
{
    // Move pointer to (0,0) location

    memset(&event, 0, sizeof(event));
    gettimeofday(&event.time, NULL);

    event.type = EV_REL;
    event.code = REL_X;
}

```

```

event.value = 100;
write(uinp_fd, &event, sizeof(event));

event.type = EV_REL;
event.code = REL_Y;
event.value = 100;
write(uinp_fd, &event, sizeof(event));

event.type = EV_SYN;
event.code = SYN_REPORT;
event.value = 0;
write(uinp_fd, &event, sizeof(event));

// Report BUTTON CLICK - PRESS event

memset(&event, 0, sizeof(event));
gettimeofday(&event.time, NULL);
event.type = EV_KEY;
event.code = BTN_LEFT;
event.value = 1;
write(uinp_fd, &event, sizeof(event));

event.type = EV_SYN;
event.code = SYN_REPORT;
event.value = 0;
write(uinp_fd, &event, sizeof(event));

// Report BUTTON CLICK - RELEASE event

memset(&event, 0, sizeof(event));
gettimeofday(&event.time, NULL);
event.type = EV_KEY;
event.code = BTN_LEFT;
event.value = 0;
write(uinp_fd, &event, sizeof(event));

event.type = EV_SYN;
event.code = SYN_REPORT;
event.value = 0;
write(uinp_fd, &event, sizeof(event));
}

void send_a_button()
{
    // Report BUTTON CLICK - PRESS event

    memset(&event, 0, sizeof(event));
    gettimeofday(&event.time, NULL);
    event.type = EV_KEY;
    event.code = KEY_A;
}

```

```

event.value = 1;
write(uinp_fd, &event, sizeof(event));

event.type = EV_SYN;
event.code = SYN_REPORT;
event.value = 0;
write(uinp_fd, &event, sizeof(event));

// Report BUTTON CLICK - RELEASE event

memset(&event, 0, sizeof(event));
gettimeofday(&event.time, NULL);
event.type = EV_KEY;
event.code = KEY_A;
event.value = 0;
write(uinp_fd, &event, sizeof(event));

event.type = EV_SYN;
event.code = SYN_REPORT;
event.value = 0;
write(uinp_fd, &event, sizeof(event));
}

/* This function will open the uinput device. Please make
   sure that you have inserted the uinput.ko into kernel. */

int main()
{
    // Return an error if device not found.
    if (setup_uinput_device() < 0)
    {
        printf("Unable to find uinput device\n");
        return -1;
    }

    send_a_button();           // Send a "A" key

    send_click_events();      // Send mouse event

    /* Destroy the input device */
    ioctl(uinp_fd, UI_DEV_DESTROY);

    /* Close the UINPUT device */
    close(uinp_fd);
}

```