Linux Input Systems

Computer Science & Engineering Department
Arizona State University
Tempe, AZ 85287

Dr. Yann-Hang Lee
yhlee@asu.edu
(480) 727-7507
An option: each attached input device is handled by a driver with the details of input port and protocol the device used.

The other one -- Layers

- adapter (controller) and port
- device and driver
- event interface
Software Structure of Input Systems

- Device drivers, input core, and event handlers
- **Example:**
  - i8042 is the driver for 8042 adapter
  - psmouse is the driver for ps2 mouse
  - mousedev is the event handler for all mice

(http://www.linuxjournal.com/article/6396)
Example: PS2 Mouse Driver

- The adapter – 8042
  - from PC-AT, now a part of LPC IO
  - PS2 signals: clock, data 5V, and GND.
    - CLOCK and DATA are of "open collector" type
  - bidirectional serial protocol (start, data, parity, stop)
    - PC has always a priority and can stop the transmission any time by setting CLOCK low

What is done in i8042.c

- The driver for the adapter (controller)
- When installed –
  - create a platform_device
  - initialize kbd and aux controllers
  - create serio ports with ids (serio->id.type = SERIO_8042;)
  - request_IRQ and add interrupt handler, and register ports

```c
error = request_irq(I8042_KBD_IRQ, i8042_interrupt, IRQF_SHARED,
"i8042", i8042_platform_device);

if (likely(port->exists))
    serio_interrupt(port->serio, data, dfl);
```

- Important fields in `struct serio`

```
struct serio_device_id id;
struct serio_driver *drv;
struct device dev;
```
Threaded interrupt handlers

- isr acknowledges the interrupt to the hardware
- wake the kernel interrupt handler thread

\[
\text{int request_threaded_irq}(\text{unsigned int irq, irq_handler_t handler,}}
\]
\[
\text{irq_handler_t thread_fn,}
\]
\[
\text{unsigned long flags, const char *name, void *dev)}
\]

- \text{handler} is called in hard interrupt context and checks if the interrupt was from the device
  - if \text{thread_fn} is \text{NULL}, use the normal handler, no irq thread
- \text{handle_IRQ_event} – calls \text{handler (check or normal)}
What is done in psmouse-base.c (1)

- The driver to handle ps2 mouse protocol
- When installed,
  - create psmouse driver of type register it.
  - dispatch a serio_attach_driver event to “events_long” work queue
  - probe serio bus and connect to serio device via the matching serio_id
    - why work thread? in serio_bus
      ```c
      static struct bus_type serio_bus = {
        .name = "serio",
        .....,
        .match = serio_bus_match,
        .uevent = serio_uevent,
        .probe = serio_driver_probe,
        .....,
      };
      ```
    - serio_driver_probe → serio_connect_driver → psmouse_connect → create a “psmouse” device
    - psmouse_interrupt : called after finding the psmouse device from serio device
What is done in psmouse-base.c (2)

- **When psmouse_interrupt is called**
  - received mouse data and process the protocol
  - `psmouse_process_byte()` analyzes the PS/2 data stream and reports relevant events to the input module once full packet has arrived.

- **psmouse registers itself as an input device to input core**
  - report events: EV_KEY and EV_REL
  - eventually, `input_pass_event` to handlers

- **What else**
  - protocol – command and response with adapter, mouse state, and data decoding.
  - mouse types
  - use a work queue for mouse resync
Event Handlers

- **evdev:**
  - a generic input event interface to pass the events generated in the kernel straight to the program, with timestamps.
  - a char device to user space
    - when open, an evdev client is created with a buffer for events and is attached to file struct.
    - when read, fetch the events in the buffer and return to the user call.
  - register as a handler of an input device
    - when connected, evdev is created
    - handle is added to the input device
  - input device passes events to handler’s clients via
    - input core’s `input_pass_event`
    - handler’s `evdev_pass_event`

```c
struct input_event {
    struct timeval time;
    __u16 type;
    __u16 code;
    __s32 value;
};
```