Embedded Systems Programming

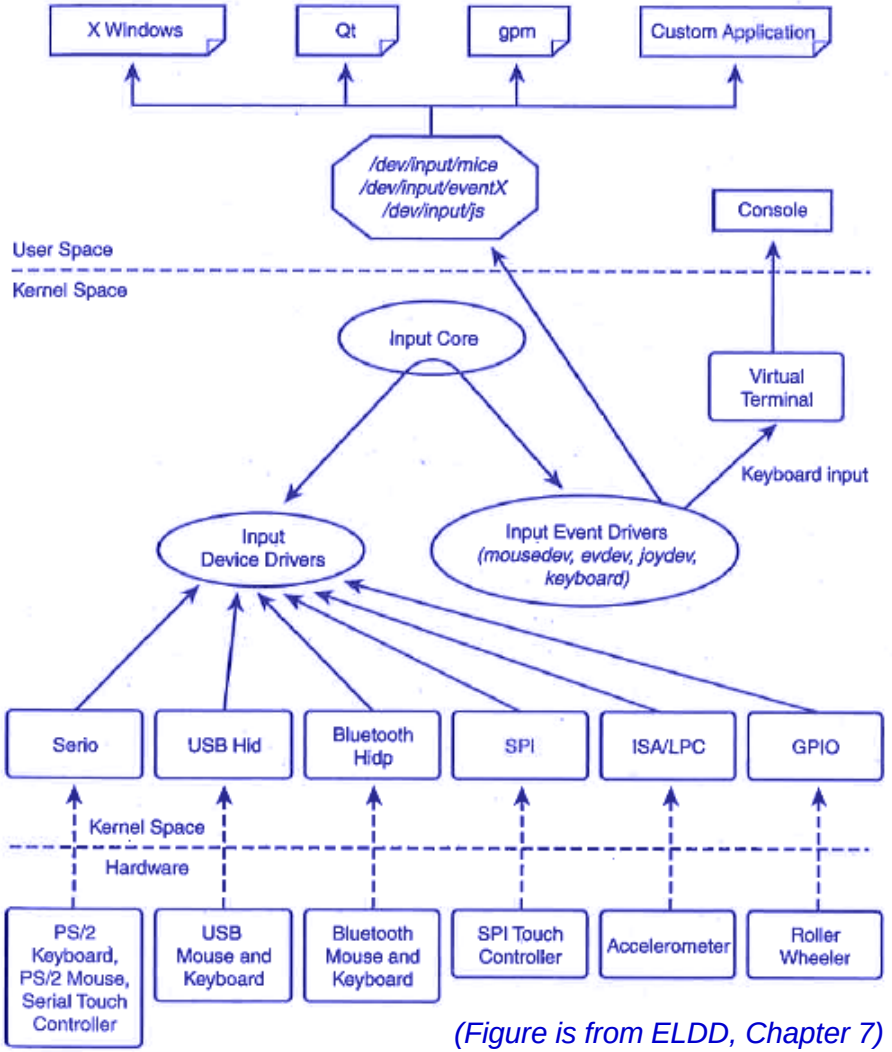
Input Processing in Linux (Module 17)

Yann-Hang Lee
Arizona State University
yhlee@asu.edu
(480) 727-7507

Summer 2014

Linux Input Systems

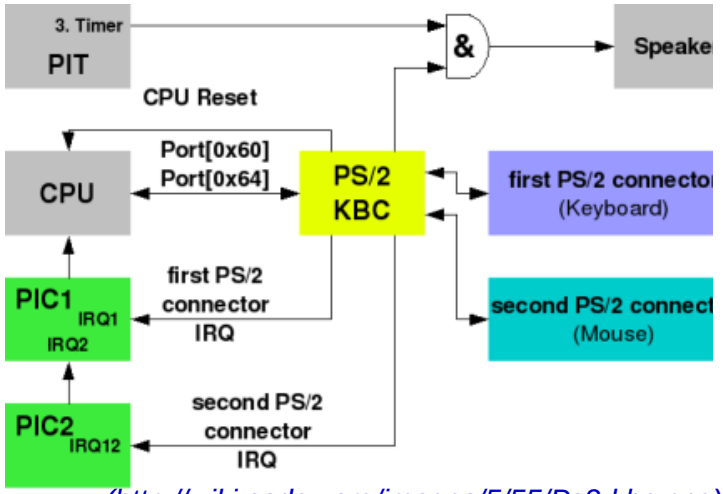
- **An option: each attached input device is handled by a driver with the details of input port and protocol the device used.**
- **The other one -- Layers**
  - adapter (controller) and port
  - device and driver
  - event interface



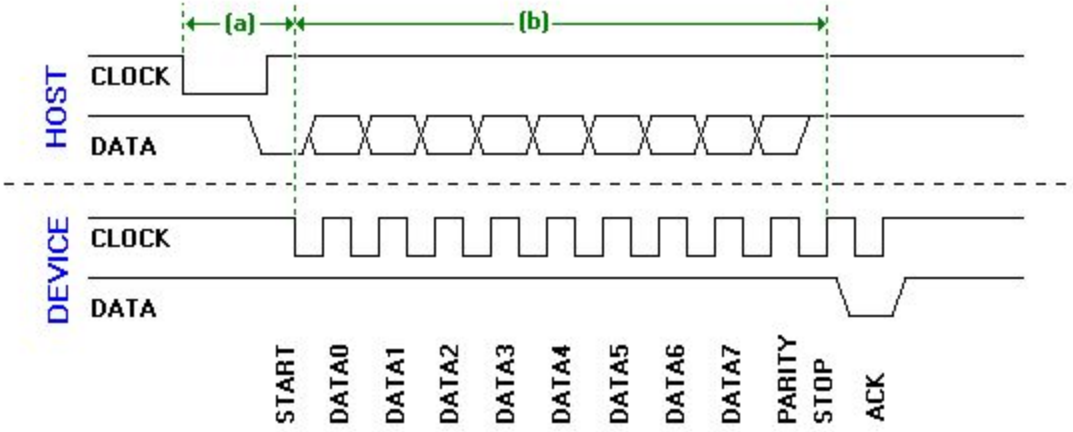
(Figure is from ELDD, Chapter 7)

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



(http://wiki.osdev.org/images/5/55/Ps2-kbc.png)



(http://codelake9.files.wordpress.com/2012/09/3.jpg)

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

```
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;
```

Request_threaded_irq

- **Threaded interrupt handlers**
  - isr acknowledges the interrupt to the hardware
  - wake the kernel interrupt handler thread

    int request_threaded_irq(unsigned int irq, irq_handler_t handler,
    irq_handler_t thread_fn,
    *unsigned long flags, const char *name, void *dev)*

  - *handler* is called in hard interrupt context and checks if the interrupt was from the device
    - if *thread_fn* is *NULL*, use the normal handler, no irq thread
  - *handle_IRQ_event – calls handler (check or normal)*

What is done in psmouse-base.c

- **The driver to handle ps2 mouse protocol**
- **When installed,**
  - probe serio bus, connect to serio device via the matching serio_id and *create a "psmouse" device*
- **psmouse registers itself as an input device to input core**
  - report events: EV_KEY and EV_REL
  - eventually, *input_pass_event* to handlers
- **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.
- **What else –**
  - mouse type and protocol – command and response with adapter, mouse state, and data decoding.

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*

Event Delivery to User Program

- **In evdev:**
  - once an event is detected, it invokes asynchronous notification

    kill_fasync(&client->fasync, SIGIO, POLL_IN);

    (with event data in client buffer)

  - User programs can
    - use blocking calls, poll() or select(), to wait for an event
    - set up a signal handler for SIGIO
  - Then, "read" input events from client buffer
    - in Linux/include/uapi/linux/input.h, type and code are defined, e.g.,

```
#define EV_REL          0x02        // event  type
#define REL_X           0x00        // event code
#define REL_Y           0x01
```

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