

PB173 – Ovladače jádra – Linux VIII.

Jiří Slabý

ITI, Fakulta Informatiky

15. 11. 2011

LDD3 kap. 7 a 10

- Práce s časem
 - Pozdržení vykonávání kódu (spánek)
 - Časovače
 - Odložené vykonání kódu
- Přerušeni HW
- Změny v kontextu přerušeni
 - Zámky
 - Alokace

Čekání na událost trvající pevně danou dobu

- 1 Spánek
 - Zapojení plánovače
 - Rozlišení: 1-10 ms (podle hodnoty HZ – `jiffies`)
- 2 Busy-waiting
 - Smyčka
 - Rozlišení: ns

API

- `linux/delay.h`
- ad 1. `ssleep`, `msleep`
- ad 2. `mdelay`, `udelay`, `ndelay`

V jádře lze čekat jen omezenou dobu

Část I

„Měkká” přerušení

- `Documentation/timers/*`
- Činnost v nastavených časech (i periodicky)
- Kontext přerušení (nelze spát)
- `/proc/timer_list` (jen nová jádra)

2 druhy

- 1 Obyčejné
 - Fungují na každé architektuře
 - Založené na `jiffies`
 - Rozlišení: 1-10 ms
- 2 High-res
 - Jen pro vybrané architektury
 - Ve valné většině kódu není třeba
 - Rozlišení: ns

API

- `linux/timer.h`, struct `timer_list`
- `DEFINE_TIMER`, `setup_timer`
- `mod_timer` (=del_timer+add_timer)
- `del_timer_sync`

```
static void my_fun(unsigned long data);
static DEFINE_TIMER(my_timer, my_fun, 0, 30);
static void my_fun(unsigned long data)
{
    my_timer.data *= 2;
    mod_timer(&my_timer, jiffies + msecs_to_jiffies(data));
}
...
mod_timer(&my_timer, jiffies + msecs_to_jiffies(100));
...
del_timer_sync(&my_timer);
```

Úkol: Každou vteřinu, kdy je modul v systému, vypsát `HZ` a `jiffies`

Při potřebě vykonat kód, který nelze vykonat teď

- Držím zámek, jsem v přerušení, . . .
- Nespecifikuje se pevně daný moment (na rozdíl od časovačů)

2 druhy

- 1 Workqueue
- 2 Tasklet

Rozdíly

- Rychlost zavolání
- Kontext zavolání
- Množství kódu (tj. rychlost vykonání)

- Speciální *proces* volající funkce ve frontě
 - Globální, společný pro všechny – většinou stačí
 - Vlastní – pro speciální případy
- Lze v něm spát
- Spustí se, až se plánovač rozhodne
 - Lze specifikovat minimální prodlevu
- **Documentation/workqueue.txt**

- **linux/workqueue.h**, struct `work_struct`
- Definice funkce (práce): `DECLARE_WORK_S`, `INIT_WORK_D`
- Globální proces:
 - Zařazení do fronty: `schedule_work`
 - Vyřazení z fronty (předčasně): `cancel_work_sync`
- Vlastní proces:
 - Vytvoření procesu: `create_workqueue`, `destroy_workqueue`
 - Zařazení do fronty: `queue_work`
 - Vyřazení z fronty: `cancel_work_sync` (stejně)
- Delayed (s min. prodlevou): `*_delayed_work`

Úkol: nahrát modul `ieee80211` (`request_module`). Nelze volat přímo v `module_init` a je nutný i vlastní proces

- Běží v kontextu přerušení (jako časovače)
- Nelze spát
- Musí být rychlý
- Spustí se *po* příštím přerušení

API

- `linux/interrupt.h`, `struct tasklet_struct`
- Definice funkce (taskletu): `DECLARE_TASKLET_S`, `tasklet_init_D`
- Zařazení do fronty: `tasklet_schedule`
- Vyřazení z fronty: `tasklet_kill`

Úkol: vytvořit tasklet, spustit a vypsát posloupnost volání pomocí `dump_stack`

Část II

„Tvrdá” přerušení

- HW informuje o změně stavu
 - Časovač tiknul, přišel paket, přečten blok z disku, . . .
- CPU přeruší chod programu/jádra a zavolá jádro
 - Cyklus? Priority přerušení!
- OS musí obsloužit přerušení
 - Zjistí zdroj a zavolá odpovídající ovladač (jeho obsluhu přerušení)
 - Přerušeni je identifikováno číslem (IRQ)
 - Vynuluje přerušeni na řadiči přerušeni

Nutná podpora HW (CPU, řadiče, sběrnice, zařízení)

Přerušení v Linuxu

- **linux/interrupt.h**
- request_irq, free_irq
- **Flags: hlavně IRQF_SHARED**
- **Návratová hodnota z obsluhy: IRQ_NONE, IRQ_HANDLED**
- /proc/interrupts

```
static irqreturn_t my_handler(int irq, void *data)
{ /* data == my_data */
  return my_device_raised_interrupt ? IRQ_HANDLED : IRQ_NONE;
}
static int my_probe(struct pci_dev *pdev, ...)
{
  /* enable device etc. */
  my_data = kmalloc(...);
  ret = request_irq(pdev->irq, my_handler, IRQF_SHARED, "my", my_data);
}
```

Úkol: vytvořte obsluhu pro Combo (zatím vračejte `IRQ_HANDLED`)

Změny

- Zámky
 - Třeba používat `*_irqflags` varianty
 - Jinak by mohlo dojít k deadlocku
- Alokace
 - `GFP_ATOMIC`
 - A tedy nevelké alokace
- Kód
 - Rychlý, krátký
 - (Víc kódu později v taskletu, ještě víc ve workqueue)

Práce s přerušením (součást domácího)

- 1 Vytvořit pojmenování (makra) pro registry (v tabulce)
- 2 Spustit 100 ms periodický časovač (pro každé Combo zařízení)
 - Vyvolat z Comba přerušení 0x1000 (registr 0x60)
- 3 Navázat se v probe na přerušení (`request_irq`)
- 4 Povolit v Combu přerušení 0x1000 (registr 0x44)
- 5 Implementovat obsluhu
 - Přečíst stav (registr 0x40), vypsát stav – limitovaně
 - Nenula: odsouhlasit přerušení (registr 0x64), vrátit `IRQ_HANDLED`
 - Jinak: vrátit `IRQ_NONE`

Offset	Len	R/W	Contents	Meaning
0x0040	4B	R	bitmap	Raised interrupts
0x0044	4B	R/W	bitmap	Enabled interrupts
0x0060	4B	W	bitmap	Raise interrupts
0x0064	4B	W	bitmap	Acknowledge interrupts

Tabulka: Specifikace baru 0 (pokračování z minula)