PB173 – Ovladače jádra – Linux

XI. Komunikace mezi procesy

Jiri Slaby

Fakulta informatiky
Masarykova univerzita

10. 12. 2015

Komunikace mezi procesy

LDD3 část kap. 6 (zastaralá)

1. Čekání na událost
   - Completion
   - Wait event a ostatní
   - Scheduler a probouzení procesů
2. Vlákna

Sekce 1

Čekání na událost

Jednoduchá komunikace

- 2 procesy (producent-konzument)
  - A: čeká na nějakou hotovou práci
  - B: udělá nějakou práci a oznámí A dokončení
  - A: pokračuje

API

- `linux/completion.h`, `struct completion`
- `DECLARE_COMPLETION`$_S$, `init_completion`$_D$
- Čekání: `wait_for_completion`, `wait_for_completion_interruptible` (retval)
- Dokončení: `complete`, `complete_all`

Příklad

```
static DECLARE_COMPLETION(my_comp);
static char str[32];
```

A

```
wait_for_completion(&my_comp);
pr_info("%s\n", str);
```

B

```
strcpy(str, "Ahoj");
complete(&my_comp);
```

Úkol

Použití `completion`

1. Do `pb173/11/events` doplňte `completion`
2. Číst se bude, až někdo dokončí zápis
   - Čekání v `read` bude možné přerušit signálem (`wait_for_completion_interruptible`)
   - V případě signálu vraťte `-ERESTARTSYS`
3. Spusťte 2 instance `cat /dev/my_name`
4. Spusťte několikrát `echo XXX > /dev/my_name`
5. Pozorujte, co se děje, když použijete
   - `complete` a potom:
   - `complete_all`

Složitější komunikace

- Čekání na více místech na jinou událost
- 3 procesy
  - A: plní buffer
  - B: čeká na 100 B
  - C: čeká na 200 B

API

- `linux/wait.h`, `wait_queue_head_t`
- `DECLARE_WAIT_QUEUE_HEAD`$_S$, `init_waitqueue_head`$_D$
- Čekání: `wait_event`, `wait_event_interruptible`
- Dokončení: `wake_up`, `wake_up_all`

Pozn.: `completion` je jen `wait_queue` + počítadlo

Složitější komunikace

Příklad

```
static DECLARE_WAIT_QUEUE_HEAD(my_wait);
static atomic_t my_cnt = ATOMIC_INIT(0);
static char str[512];
...
A
  for (a = 0; a < sizeof(str); a++) {
    str[atomic_inc_return(&my_cnt) - 1] = 'A';
    wake_up_all(&my_wait);
    msleep(50);
  }

B
  wait_event(my_wait, atomic_read(&my_cnt) > 100);
  pr_info("%s\n", str);

C
  if (wait_event_interruptible(my_wait, atomic_read(&my_cnt) > 200)) {
    pr_info("interrupted\n");
    return;
  }
  pr_info("%s\n", str);
```

Úkol

Použití `wait_queue`

1. V `pb173/11/events` použijte namísto `completion` `wait_queue`
2. Číst se bude, až bude v bufferu zapsáno alespoň 5 znaků

Scheduler

- Celé je to instruování plánovače (`linux/sched.h`)
- Ve skutečnosti je `wait_event`:
  - Nastavení typu spícího stavu procesu: `set_current_state`
    - D stav: `TASK_UNINTERRUPTIBLE`
    - S stav: `TASK_INTERRUPTIBLE`
  - Uspání procesu bez/s timeoutem: `schedule`/`schedule_timeout`
- A `wake_up`:
  - Probuzení procesu: `wake_up_process`
  - Popř. s obsluhou signálů: `signal_pending`
- `completion` a `wait_queue` ulehčuje práci
  - Pamatováním si, koho vzbudit
  - Případnou kontrolou signálů
  - Případnou kontrolou vypršení timeoutu
  - `completion` má navíc čítač, kolik procesů vzbudit

Úkol

Explicitní čekání

1. Místo `wait_event`, použijte funkce z předchozího slidu
2. Jako vzor prostudujte kód ve funkci `afu_read`
3. Zkopírujte jej a použijte
   - Odstraňte zámek a ostatní irelevantní kód
4. Pozměňte čekací podmínku
   - Bude se čekat alespoň na 2 „`a`” a 1 „`b`” v poli

Sekce 2

Vlákna

kthread

- Vytvoření vlákna (procesu) pro výpočty
  - Běží v jádře
  - Chování jako uživatelský proces
- `linux/kthread.h`
- Vytvoření, zrušení: `kthread_run`, `kthread_stop`
- Test na skončení ve vlákně: `kthread_should_stop`

Příklad

```
static int my_fun(void *data)
{ /* data = my_data */
  while (1) {
    wait_event_interruptible (...) ;
    if (kthread_should_stop())
      break;
    do_some_work();
  }
  return 0;
}
```

```
struct task_struct *my_t;
static int my_init(void)
{
  my_t = kthread_run(my_fun, my_data, "my_fun");
  return IS_ERR(my_t) ? PTR_ERR(my_t) : 0;
}
static void my_exit(void)
{
  kthread_stop(my_t);
}
```

Úkol

Vytvoření vlákna

1. Vytvořte vlákno
2. Vlákno počká na `write` až někdo něco zapíše
3. Přepíše všechny znaky „`a`” na „`b`”
4. Vzbudí všechny v `read`, kteří čekají na data
5. Opakujte v cyklu dokud někdo neodebere modulu
6. Vyzkoušejte zápisem a čtením do/z `/dev/`