

PB173 – Ovladače jádra – Linux

XI. Komunikace mezi procesy

Jiri Slaby

ITI, Fakulta informatiky

3. 12. 2013

LDD3 část kap. 6 (zastaralá)

- Čekání na událost
- Buzení procesů

- 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`

```
static DECLARE_COMPLETION(my_comp);  
static char *str;
```

```
...
```

```
A  
wait_for_completion(&my_comp);  
printk (KERN_DEBUG "%s\n", str);
```

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

Použití `completion`

- 1 Doplnit do `pb173/11/events completion`
- 2 Číst se bude, až někdo dokončí zápis
 - Čekání v `read` je možné přerušit signálem (`wait_for_completion_interruptible`)
- 3 Spustit 2 instance `cat /dev/my_name`
- 4 Pozorovat, co se děje
- 5 Spustit nekolikrát `echo XXX > /dev/my_name`

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

API

- `linux/wait.h`, `wait_queue_head_t`
- `DECLARE_WAIT_QUEUE_HEADS`, `init_waitqueue_head`
- Čekání: `wait_event`, `wait_event_interruptible`
- Dokončení: `wake_up`, `wake_up_all`
- `completion` je obalená `wait_queue`

Složitější komunikace – příklad

```
static DECLARE_WAIT_QUEUE_HEAD(my_wait);  
static atomic_t my_cnt = ATOMIC_INIT(0);  
static char *str;
```

...

```
A  
while (1) {  
    str[atomic_inc_return(&my_cnt) - 1] = 'A';  
    wake_up_all(&my_wait);  
    msleep(10);  
}
```

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

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

Použití `wait_queue`

- 1 Namísto `completion`, použít v `pb173/11/events` `wait_queue`
- 2 Číst se bude, až bude v bufferu více než 5 znaků

- Celé je to instruování plánovače
- Ve skutečnosti je `wait_event` a `wake_up`:
 - Nastavení stavu procesu: `set_current_state`
 - Uspání procesu: `schedule`
 - Probuzení procesu: `wake_up_process`
 - Popř. s obsluhou signálů: `signal_pending`
 - `linux/sched.h`
- `completion` a `wait_queue` ulehčuje práci
 - Pamatováním si, koho vzbudit
 - Případnou kontrolou signálů
 - Případnou kontrolou vypršení timeoutu
 - `completion` navíc řeší předání čítače (kolik procesů vzbudit)

Explicitní čekání

- 1 Místo `wait_event`, použijte funkce z předchozího slidu
 - Prostudujte (a použijte) tělo makra `__wait_event_interruptible`
- 2 Pozměňte čekací podmínku
 - Bude se čekat alespoň na 2 „a” a 1 „b” v poli

- Vytvoření vlákna (procesu) pro výpočty
 - Běží v jádře
 - Chování jako uživatelský proces
- `linux/kthread.h`
- `kthread_run`, `kthread_stop`
- `kthread_should_stop`

Příklad

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

```
static int my_init(void)
{
  struct task_struct *t;
  t = kthread_run(my_proc, my_data, "my_process");
  if (IS_ERR(t))
    return PTR_ERR(t);
  ...
  kthread_stop(t); /* some time later */
  ...
}
```

Vytvoření vlákna

- 1 Vytvořte vlákno
- 2 Vlákno počká, až někdo něco zapíše
- 3 Přepíše všechny znaky „a” na „b”
- 4 Vzbudí všechny, kteří čekají na data