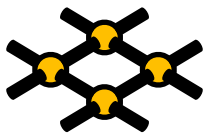
Vláknové programování
část VII

Lukáš Hejmánek, Petr Holub
{xhejtman,hopet}@ics.muni.cz



Laboratoř pokročilých síťových technologií

PV192
2014–04–08

Přehled přednášky

Ukončování

Ukončování

- Dvě varianty ukončení:
  - Samotným vláknem
    - **`pthread_exit()`**.
    - Návrat z hlavní funkce vlákna.
  - Jiným vláknem
    - **`pthread_kill()`**
    - **`pthread_cancel()`**

`pthread_cancel()`

- **`pthread_cancel()`** pošle danému vláknu notifikaci, aby se ukončilo.
- Vlákna mohou mít nastaveny dva různé typy kancelace:
  - **`PTHREAD_CANCEL_DEFERRED`** – vlákno je ukončeno pouze v tzv. kancelačních bodech (default).
  - **`PTHREAD_CANCEL_ASYNCHRONOUS`** – vlákno je ukončeno okamžitě.
- Dále vlákna mohou kancelaci odmítnout **`PTHREAD_CANCEL_DISABLE`**, opětovně přijmout kancelaci jde pomocí **`PTHREAD_CANCEL_ENABLE`**.
- Typy kancelace nastavíme pomocí **`pthread_setcanceltype()`**.
- Přijmout/odmítnout kancelaci lze pomocí **`pthread_setcancelstate()`**.

Kancelační body

- Kancelační bod je volání funkce, ve které může být vlákno ukončeno, je-li typu **`PTHREAD_CANCEL_DEFERRED`**.
- Základní kancelační body jsou:
  - **`pthread_testcancel()`** – pouze zjistí, zda nebylo signalizováno *cancel*
  - **`pthread_setcancelstate()`** – pokud měníme stav z **`PTHREAD_CANCEL_DISABLE`** na **`PTHREAD_CANCEL_ENABLE`**, je volání kancelačním bodem.
- Další kancelační body:
  - **`pthread_cond_wait()`**, **`pthread_cond_timedwait()`**, **`pthread_join()`**, **`sem_wait()`** (pouze z knihovny pthreads, pokud je poskytnuta knihovnou libc, není to kancelační bod!).
  - Většina funkcí **`libc`** (zejména I/O funkce), je vhodné konzultovat dokumentaci.

Příklad na kancelaci

```
1  #include <pthread.h>
2
3  void *
4  foo(void *arg)
5  {
6          int old;
7          pthread_setcanceltype(PTHREAD_CANCEL_DEFERRED, &old);
8          while(1) {
9                  pthread_testcancel();
10         }
11         return NULL;
12 }
13
14 int
15 main()
16 {
17         pthread_t t;
18
19         pthread_create(&t, NULL, foo, NULL);
20
21         pthread_cancel(t);
22
23         return 0;
24 }
```

Cleanup Push/pop

- Co dělat v případě, že vlákno, kterému posíláme cancel, zrovna drží nějaký zámek?
- **`pthread_testcancel()`** rovnou vlákno ukončí, nelze použít pro test a případně zámek odemknout.
- Push/pop
  - Vlákno má zásobník funkcí, které se mají provést v případě kancelace.
  - **`pthread_cleanup_push()`** přidá specifikovanou funkci na vrchol zásobníku.
  - **`pthread_cleanup_pop()`** odebere funkci z vrcholu zásobníku (lze říct, zda funkci rovnou provést).
  - Některé implementace pthreads hlídají párování push/pop pomocí maker a ke každému push v každé funkci musí být odpovídající pop!

Příklad na cleanup

```c
#include <pthread.h>

pthread_mutex_t lock;

void *
foo(void *arg)
{
        int old;
        pthread_setcanceltype(PTHREAD_CANCEL_DEFERRED, &old);
        pthread_cleanup_push(pthread_mutex_unlock, &lock);
        pthread_mutex_lock(&lock);
        while(1) {
                pthread_testcancel();
        }
        pthread_cleanup_pop(1); /*execute unlock*/
        return NULL;
}

int
main()
{
        pthread_t t;

        pthread_create(&t, NULL, foo, NULL);

        pthread_cancel(t);
        return 0;
}
```

Atributy funkcí pthread knihovny

Start vlákna

- **`pthread_create()`** funkci můžeme předávat atributy pro nově vytvářené vlákno.
- Atributy ovlivňují tři základní oblasti:
  - Osamostatnění vlákna
  - Nastavování priorit plánovače
  - Nastavení zásobníku
- Datový typ atributu **`pthread_attr_t`**.
- Inicializace **`pthread_attr_init()`**.
- Zrušení **`pthread_attr_destroy()`**.

Start vlákna – osamostatnění

- Osamostatněné vlákno uvolní všechny své zdroje jakmile skončí.
- Neosamostatněné vlákno je uvolní až při zavolání **`pthread_join()`**.
- Implicitně je každé vlákno neosamostatněné.
- **`pthread_attr_setdetachstate()`** nastaví vlákno osamostatněné (**`PTHREAD_CREATE_DETACHED`**) nebo neosamostatněné (**`PTHREAD_CREATE_JOINABLE`**).

Příklad osamostatnění

```
1  #include <pthread.h>
2
3  void *
4  foo(void * arg)
5  {
6          return NULL;
7  }
8
9  int
10 main(void)
11 {
12         pthread_t t;
13         pthread_attr_t attr;
14
15         pthread_attr_init(&attr);
16         pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);
17
18         pthread_create(&t, &attr, foo, NULL);
19         pthread_attr_destroy(&attr);
20         return 0;
21 }
```

Start vlákna – nastavení priority plánovače

- Pro vlákna lze do jisté míry ovlivnit způsob plánování.
- Lze nastavit tři základní pravidla plánování:
  - **`SCHED_OTHER`** – vlákno je plánováno dle standardního jaderného plánovače.
  - **`SCHED_FIFO`** – vlákno je plánováno dokud samo neskončí, nezablokuje se nebo není zrušeno.
  - **`SCHED_RR`** – vlákno je plánováno dokud samo neskončí, nezablokuje se, není zrušeno nebo nevyprší přidělené časové kvantum.

Start vlákna – nastavení priority plánovače

- Pro pravidla lze dále nastavit prioritu.
- Abychom mohli prioritu plánování olivnit, je třeba nastavit explicitní plánování na **`PTHREAD_EXPLICIT_SCHED`**:
  - **`pthread_attr_setinheritsched()`**
- Nastavení priority blízké *realtime* prioritě (**`SCHED_FIFO`**, **`SCHED_RR`**) může udělat pouze proces s právy administrátora.
- Realtime procesy mají přednost před ostatními.

Příklad nastavení priorit plánovače

```
1  #include <pthread.h>
2  #include <stdio.h>
3  #include <unistd.h>
4  #include <string.h>
5
6  int quit = 0;
7
8  void *
9  foo(void *arg)
10 {
11         long i=0;
12
13         while(!quit) {
14                 i++;
15                 if((i % 10000) == 0)
16                         usleep(10);
17         }
18         return i;
19 }
```

Příklad nastavení priorit plánovače

```
19 int
20 main(void)
21 {
22         pthread_t t1, t2, t3;
23         pthread_attr_t attr;
24         struct sched_param param;
25         long res;
26
27         memset(&param, 0, sizeof(param));
28
29         pthread_attr_init(&attr);
30
31         pthread_attr_setinheritsched(&attr, PTHREAD_EXPLICIT_SCHED);
32         pthread_attr_setschedpolicy(&attr, SCHED_FIFO);
33         param.sched_priority = 10;
34         pthread_attr_setschedparam(&attr, &param);
35
36         pthread_create(&t1, &attr, foo, NULL);
37         param.sched_priority = 20;
38         pthread_attr_setschedparam(&attr, &param);
39         pthread_create(&t2, &attr, foo, NULL);
40         param.sched_priority = 30;
```

Příklad nastavení priorit plánovače

```
46          pthread_attr_setschedparam(&attr, &param);
47          pthread_create(&t3, &attr, foo, NULL);
48          sleep(2);
49          quit = 1;
50          pthread_join(t1, &res);
51          printf("Thread␣with␣prio␣10␣%ld\n", res);
52          pthread_join(t2, &res);
53          printf("Thread␣with␣prio␣20␣%ld\n", res);
54          pthread_join(t3, &res);
55          printf("Thread␣with␣prio␣30␣%ld\n", res);
56          return 0;
57  }
```

Výstup příkladu

- Thread with prio 10 39930000 iterations
- Thread with prio 20 65870000 iterations
- Thread with prio 30 103812132 iterations
- Poznámka:
  - Vynechání volání **`usleep()`** má za následek takřka zablokování systému.
  - Z tohoto důvodu je povoleno nastavit realtime priority pouze administrátorským procesům.
  - Při jejich programování je nutno brát ohled na preempci ostatních procesů.

Nastavení zásobníku

- Implicitní velikost zásobníku pro vlákno je v Linuxu 8 MB.
- Chceme-li vytvořit 1000 vláken, potřebovali bychom 8 GB paměti jen pro zásobníky vláken.
- Pthread knihovna umožňuje změnit velikost zásobníku pro vlákno.
- **`pthread_attr_setstacksize()`**.
- Je nutné nastavit velikost zásobníku tak, aby se na něj vešly lokání proměnné všech funkcí, které se po sobě mohou zavolat. V opačném případě obdržíme signál **`SIGSEGV`** při vstupu do funkce, jejíž proměnné se na zásobník už nevlezou. Chyba vypadá na první pohled dost záhadně!

Příklad nastavení zásobníku

```
1  #include <pthread.h>
2  #include <stdio.h>
3
4  void*
5  foo(void* arg)
6  {
7          return NULL;
8  }
9
10 int
11 main(void)
12 {
13         pthread_attr_t attr;
14         pthread_t t;
15
16         pthread_attr_init(&attr);
17
18         pthread_attr_setstacksize(&attr, 65536);
19
20         pthread_create(&t, &attr, foo, NULL);
21
22         pthread_join(t, NULL);
23         pthread_attr_destroy(&attr);
24
25         return 0;
26 }
```