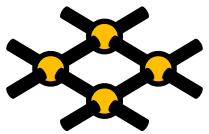
Vláknové programování
část IV

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



Laboratoř pokročilých síťových technologií

PV192
2011–03–31

Přehled přednášky

Architektura systému

- Vývoj
  - Single procesor
  - SMP systémy (symetrický multiprocesing – více rovnocenných procesorů)
    - Obvykle společná paměť
    - Všechny procesory mají do paměti „stejně daleko“
  - NUMA systémy (více procesorů, nejsou rovnocenné)
    - Procesory mají lokální paměť
    - Přístup do ne-lokální paměti přes ostatní CPU
  - SMT systémy (symetrický multithreading – procesory mají více jader)

Architektura systému

- Procesory mají lokální cache
- SMT systémy mívají lokální cache pro jádro a společnou cache pro více jader

Nastavení afinity

- Vlákno může být obecně plánováno na libovolný procesor
- Nemusí být vždy vhodné
- Migrace mezi procesory bývá poměrně drahá
- Pokud jde o výkon aplikace, můžeme chtít zabránit migraci procesů/vláken
- Úskalí ve statickém přiřazení procesů/vláken na procesor
- Afinita – nastavení množiny procesorů, na kterých má proces/vlákno běžet

Nastavení afinity

- Nastavení afinity procesů
  - `sched_setaffinity(pid_t pid, size_t cpusetsize, cpu_set_t *mask);`
  - `sched_getaffinity(pid_t pid, size_t cpusetsize, cpu_set_t *mask);`
  - Nelze použít pro vlákna
- Nastavení afinity vláknům
  - `pthread_setaffinity_np(pthread_t thread, size_t cpusetsize, const cpu_set_t *cpuset);`
  - `pthread_getaffinity_np(pthread_t thread, size_t cpusetsize, cpu_set_t *cpuset);`

CPU SET

- `cpu_set_t *CPU_ALLOC(int num_cpus);`
- `void CPU_ZERO_S(size_t setsize, cpu_set_t *set);`
- `void CPU_SET_S(int cpu, size_t setsize, cpu_set_t *set);`
- `void CPU_CLR_S(int cpu, size_t setsize, cpu_set_t *set);`
- `int CPU_ISSET_S(int cpu, size_t setsize, cpu_set_t *set);`
- `void CPU_COUNT_S(size_t setsize, cpu_set_t *set);`
- Logické operace mezi dvěma cpu_set_t: AND, OR, XOR, EQUAL

Příklad

```
1  #define _GNU_SOURCE
2  /* Musi byt jako prvni pred vsemi ostatnimi include */
3  #include <pthread.h>
4  #include <sched.h>
5  #define NUM_CPU 8
6
7  int
8  main()
9  {
10         cpu_set_t * set;
11
12         set = CPU_ALLOC(NUM_CPU);
13
14         CPU_ZERO_S(NUM_CPU, set);
15
16         CPU_SET(0, set);
17
18         pthread_setaffinity_np(pthread_self(), NUM_CPU, set);
19
20         CPU_FREE(set);
21
22         return 0;
23 }
```

Numactl, taskset

- Nastavení afinity u hotové aplikace
- **`numactl(8), taskset(1)`**
- Příklad

```
numactl -cpubind=0 aplikace
taskset 0x1 aplikace
```

  - Pustí aplikaci výhradně na CPU 0

Atributy

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.
- 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;
41         pthread_attr_setschedparam(&attr, &param);
42         pthread_create(&t3, &attr, foo, NULL);
43         sleep(2);
```

Příklad nastavení priorit plánovače

```
46          quit = 1;
47          pthread_join(t1, &res);
48          printf("Thread␣with␣prio␣10␣%ld\n", res);
49          pthread_join(t2, &res);
50          printf("Thread␣with␣prio␣20␣%ld\n", res);
51          pthread_join(t3, &res);
52          printf("Thread␣with␣prio␣30␣%ld\n", res);
53          return 0;
54  }
```

Výstup příkladu

- Thread with prio 10 39930000
- Thread with prio 20 65870000
- Thread with prio 30 103812132
- 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 }
```

Atributy mutexů - inverze priorit

- Mějme 2 vlákna s různou prioritou, A nechť má vysokou prioritu, B nechť má nízkou prioritu.
- Nechť B zamkne sdílený objekt. Pak A při přístupu ke sdílenému objektu je zdržováno nízkou prioritou vlákna B i přes vlastní vysokou prioritu.

Atributy mutexů

- Prioritní protokoly zámků.
- Lze specifikovat:
  - Žádný protokol (implicitní chování) **`PTHREAD_PRIO_NONE`**.
  - Protokol dědění priorit **`PTHREAD_PRIO_INHERIT`**.
    - Vlákno po zamčení objektu získá (zdědí) prioritu od vlákna s nejvyšší prioritou.
  - Protokol omezení priorit **`PTHREAD_PRIO_PROTECT`**.
    - Vlákno po zamčení objektu získá definovanou prioritu (nastavenou pomocí **`pthread_mutex_setprioceiling`**), je-li tato vyšší než jeho aktuální.
  - Změna pomocí volání **`pthread_mutexattr_setprotocol()`**.
  - Po odemčení objektu vlákno získá svou původní prioritu.
- Mechanismus protokolů a priorit je silně implementačně závislý, není všude podporován!

Atributy mutexů

- Datový typ **`pthread_mutexattr_t`**.
- Inicializace **`pthread_mutexattr_init()`**.
- Zrušení atributu **`pthread_mutexattr_destroy()`**.

Příklad prioritních mutexů

- Nutno kompilovat: **`gcc -g -o priomutex priomutex.c -pthread -D__REENTRANT -D_XOPEN_SOURCE=500`**

```
1  #include <pthread.h>
2
3  int
4  main()
5  {
6          pthread_mutex_t lock;
7          pthread_mutexattr_t attr;
8
9          pthread_mutexattr_init(&attr);
10         pthread_mutexattr_setprotocol(&attr, PTHREAD_PRIO_PROTECT);
11         pthread_mutexattr_setprioceiling(&attr, 10);
12
13         pthread_mutex_init(&lock, &attr);
14
15         pthread_mutex_destroy(&lock);
16         pthread_mutexattr_destroy(&attr);
17         return 0;
18 }
```

Atributy mutexů

- Definování chování zámků v případě násobného zamčení/odemčení stejným vláknem.
- Ve většině případů je pokus zamknout mutex vláknem, které již tento zámek drží, chybou končící deadlockem.
- Pthreads umožní nastavit chování v takových případech.
  - **`PTHREAD_MUTEX_NORMAL`** vlákno se při násobném zamčení zámku deadlockne. Výsledek odemčení nezamčeného zámku není definován.
  - **`PTHREAD_MUTEX_ERRORCHECK`** vláknu je vrácena chyba při pokusu o násobné zamčení zámku. *Je nutno kontrolovat návratový kód funkce zamčení zámku!* Pokus o odemčení nezamčeného zámku je signalizován chybovým návratovým kódem.
  - **`PTHREAD_MUTEX_RECURSIVE`** vlákno smí provést násobné zamčení zámku. Výsledek je stejný jako by byl zámek zamčen jen jednou. Při pokusu o odemčení nezamčeného zámku je vrácena chyba.
  - **`PTHREAD_MUTEX_DEFAULT`** násobné zamčení i odemčení nemají definované chování. Tato volba je implicitní.

Příklad násobných zámků

- Nutno kompilovat: **`gcc -g -o multilock multilock.c -pthread -D__REENTRANT -D_XOPEN_SOURCE=500`**

```
1  #include <stdio.h>
2  #include <pthread.h>
3
4  int
5  main()
6  {
7          pthread_mutexattr_t attr;
8          pthread_mutex_t lock;
9
10         pthread_mutexattr_init(&attr);
11         pthread_mutexattr_settype(&attr, PTHREAD_MUTEX_RECURSIVE);
12
13         pthread_mutex_init(&lock, &attr);
14
15         printf("Return code of the first lock %d\n",
16                         pthread_mutex_lock(&lock));
17         printf("Return code of the second lock %d\n",
18                         pthread_mutex_lock(&lock));
19
20         pthread_mutex_unlock(&lock);
21         pthread_mutex_destroy(&lock);
```

Příklad násobných zámků

```
21
22          pthread_mutexattr_settype(&attr, PTHREAD_MUTEX_ERRORCHECK);
23
24          pthread_mutex_init(&lock, &attr);
25
26          printf("Return␣code␣of␣the␣first␣lock␣%d\n",
27                          pthread_mutex_lock(&lock));
28          printf("Return␣code␣of␣the␣second␣lock␣%d\n",
29                          pthread_mutex_lock(&lock));
30
31          pthread_mutex_unlock(&lock);
32          pthread_mutex_destroy(&lock);
33
34          pthread_mutexattr_destroy(&attr);
35          return 0;
36  }
```

Výstup programu

- Rekurzivní zámek:
  Return code of the first lock 0
  Return code of the second lock 0
- Errorcheck:
  Return code of the first lock 0
  Return code of the second lock 35

Atributy podmínek

- Atribut podmínek umožňuje specifikovat typ času pro volání **`pthread_cond_timedwait()`**.
- Implicitně se timeout podmínky řídí dle hodin reálného času.
- Hodiny reálného času mohou skákat dopředu ale i dozadu, což může způsobit problémy.
- Proto lze pro podmínky specifikovat časový zdroj **`CLOCK_MONOTONIC`**, který se vždy pouze zvyšuje.

Atributy podmínek

- Datový typ **`pthread_condattr_t`**.
- Inicializace **`pthread_condattr_init()`**.
- Nastavení/zjištění zdroje času **`pthread_condattr_setclock()`**, **`pthread_condattr_getclock()`**.
- Zrušení atributu **`pthread_condattr_destroy()`**.

Příklad atributů podmínek

```
1  #include <time.h>
2  #include <pthread.h>
3
4  int
5  main()
6  {
7          pthread_condattr_t attr;
8          pthread_cond_t cond;
9
10         pthread_condattr_init(&attr);
11
12         pthread_condattr_setclock(&attr, CLOCK_MONOTONIC);
13
14         pthread_cond_init(&cond, &attr);
15
16         pthread_cond_destroy(&cond);
17         pthread_condattr_destroy(&attr);
18         return 0;
19 }
```

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

```
1  #include <pthread.h>
2
3  pthread_mutex_t lock;
4
5  void *
6  foo(void *arg)
7  {
8          int old;
9          pthread_setcanceltype(PTHREAD_CANCEL_DEFERRED, &old);
10         pthread_cleanup_push(pthread_mutex_unlock, &lock);
11         pthread_mutex_lock(&lock);
12         while(1) {
13                 pthread_testcancel();
14         }
15         pthread_cleanup_pop(1); /*execute unlock*/
16         return NULL;
17 }
18
19 int
20 main()
21 {
22         pthread_t t;
23
24         pthread_create(&t, NULL, foo, NULL);
25
26         pthread_cancel(t);
27         return 0;
28 }
```

Thread specific data

Thread-Specific Data

- Řada nástrojů pro paralelní běhy vláken umožňuje vytvořit privátní datovou oblast vlákna – TLS (Thread local storage).
- TLS je využito například knihovnou OpenGL (i když poněkud nešťastně) pro uchování kontextu.
- TLS je poskytnuto Javou, některými C/C++ variantami (GNU C, Intel C/C++, Visual C++, a další), C#, Python, Dephi.

TLS v Pthreads

- Princip použití:
  - Vytvoření klíče (s volitelným destruktorem).
  - Svázání klíče s nějakými daty.
  - Vyhledání dat podle klíče.
- Klíč je globální pro všechna vlákna daného procesu.
- Vazba dat na klíč je pro každé vlákno separátní.

TLS v Pthreads

- Datový typ klíče **`pthread_key_t`**.
- Vytvoření klíče **`pthread_key_create()`**.
  - Při vytváření klíče je možné specifikovat destruktor, který se zavolá v případě ukončení vlákna.
- Svázání dat a klíče **`pthread_setspecific()`**.
- Vyhledání dat dle klíče **`pthread_getspecific()`**.
- Zrušení klíče **`pthread_key_delete()`**.

Příklad na TLS

```
1  #include <pthread.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  pthread_key_t key;
6
7  void
8  msg(char *m)
9  {
10         char *buff = pthread_getspecific(key);
11         sprintf(buff, "%s\n", m);
12         printf(buff);
13 }
14
15 void *
16 runner(void *arg)
17 {
18         char *array;
19         int i;
20
21         array = malloc(20);
22         pthread_setspecific(key, array);
23         for(i = 0; i < 10; i++) {
24                 msg(arg);
25         }
26         return NULL;
27 }
```

Příklad na TLS

```
27
28 int
29 main(void)
30 {
31         pthread_t t1, t2;
32
33         pthread_key_create(&key, free);
34
35         pthread_create(&t1, NULL, runner, "Hello");
36         pthread_create(&t2, NULL, runner, "Hello world");
37
38         pthread_join(t1, NULL);
39         pthread_join(t2, NULL);
40
41         pthread_key_delete(key);
42         return 0;
43 }
```

Jednodušší použití

- Použití pomocí klíčů je trochu těžkopádné
- GCC nabízí (neportabilní) direktivu **`__thread`**
- Použití:
  - **`__thread proměnná`**
  - **`__thread int x`**
  - Má zde význam slovo **`volatile`**?

Příklad

```
1  #include <pthread.h>
2  #include <stdio.h>
3
4  __thread int x=0;
5
6
7  void *
8  worker(void *arg) {
9          for(;x<1000000;x++) {
10                 asm volatile(""::"m" (x));
11         }
12         printf("X␣val:␣%d,␣addr␣%p\n", x, &x);
13 }
14
15 int main()
16 {
17         pthread_t t[2];
18
19         pthread_create(&t[0], NULL, worker, NULL);
20         pthread_create(&t[1], NULL, worker, NULL);
21         pthread_join(t[0], NULL);
22         pthread_join(t[1], NULL);
23         printf("X␣val:␣%d,␣addr␣%p\n", x, &x);
24 }
```

- Příklad výstupu:
  X val: 1000000, addr 0x7ff3966d470c
  X val: 1000000, addr 0x7ff395ed370c
  X val: 0, addr 0x7ff396e706fc