Základní práce s vlákny v Linuxu 11

V unixových operačních systémech byl z historických důvodů základní jednotkou vykonávající program[1] proces. Jelikož se ale koncept vláken osvědčil, byla později vlákna do unixových operačních systémů doplněna a práce s nimi je velmi podobná tomu, co jsme viděli v případě operačního systému MS Windows.[2] V případě operačního systému Linux se s vlákny interně pracuje podobně jako s procesy, každé vlákno má svou sadu registrů, zásobník atd. a paměťový prostor a systémové prostředky jsou sdíleny v rámci jednoho procesu.

1 Vytvoření vlákna

K vytvoření vlákna slouží funkce:

```
int pthread_create(
        pthread_t *thread,
        pthread_attr_t *attr,
        void *(*start_routine)(void*),
        void *arg)
```

Tato funkce má právě čtyři argumenty, prvním je ukazatel na hodnotu typu `pthread_t`, která obsahuje identifikátor vytvořeného vlákna a umožňuje s tímto vláknem pracovat. Druhý argument představuje atributy, které má vytvořené vlákno mít. Pokud použijeme hodnotu `NULL`, použije se výchozí nastavení. Třetí argument udává funkci, která představuje kód, který se má vláknem vykonávat. Tato funkce vrací hodnotu typu `void *` a akceptuje jeden argument typu `void *`, který je vláknu předán při jeho vytvoření. K tomu slouží čtvrtý argument funkce `pthread_create`.

Následující kód demonstruje vytvoření vlákna:

```
1 #include <stdio.h>
2 #include <pthread.h>
3 #include <time.h>
4
5 #define COUNT   (20)
6
7 static void msleep(int ms)
```

[1] Instrukce programu.

[2] Tím, že vlákna byla doplněna až později, vznikají ostré hrany a koncepční problémy, např. jak by se měl OS zachovat, pokud jedno z vláken zavolá `fork()`?

```
8  {
9          struct timespec t;
10         t.tv_sec  = ms / 1000;
11         t.tv_nsec = (ms % 1000) * 1000000;
12         nanosleep(&t, NULL);
13 }
14
15 static void *thread_func(void *arg)
16 {
17         int id = *((int *) arg);
18         printf("Spusteno vlakno: %i\n", id);
19
20         for (int i = 0; i < COUNT; i++) {
21                 printf("thr #%i: %i\n", id, i);
22                 msleep(5);
23         }
24         return (void *) 42;
25 }
26
27 int main()
28 {
29         int id = 1;
30         long result;
31         pthread_t thread;
32         if (pthread_create(&thread, NULL, thread_func, &id)) {
33                 fprintf(stderr, "Vytvoreni vlakna selhalo\n");
34                 return 1;
35         }
36
37         for (int i = 0; i < COUNT; i++) {
38                 printf("thr #main: %i\n", i);
39                 msleep(5);
40         }
41         pthread_join(thread, (void **) &result);
42         printf("Result: %li\n", result);
43         return 0;
44 }
```

Při popisu tohoto kódu začneme spíše od konce, od řádků 31 až 35, kde dochází k vytvoření vlákna, které je dané funkcí `thread_func` a kterému je předán ukazatel `&id`.[3] Pokud vytvoření vlákna z nějakého

[3]Pokud předáváme odkaz na lokální proměnné, tj. data, která jsou alokována na zásobníku, je nutné zajistit, aby funkce, jež takto data předává, neskončila dřív než vlákno, které vytvořila. Jinak by společně s ukončenou funkcí zanikla i data umístěná zásobníku. Alternativně můžeme předávat dynamicky alokovaná data.

důvodu selže, je vrácena nenulová hodnota.

S vláknem můžeme pracovat pomocí hodnoty, která je určena prvním argumentem funkce `pthread_create`. Zejména bychom měli v některém z bodů programu počkat na dokončení daného vlákna. K tomu slouží funkce `pthread_join`, která jednak čeká na doběhnutí daného vlákna, a také umožňuje vyzvednout hodnotu vrácenou funkcí vláknem.[4]

Struktura programu se neliší od toho, co jsme viděli v předchozím cvičení. Narozdíl od předchozího cvičení ukázkový kód obsahuje pomocnou funkci `msleep`, která uspí aktuální vlákno na zadaný počet milisekund.[5]

Protože tento způsob práce s vlákny stojí mimo standardní knihovnu jazyka C, je potřeba při překladu buď použít přepínač `-pthread`[6] nebo připojit knihovnu `libpthread`, tj. použít přepínač `-lpthread`.[7]

1.1 Ukončení vlákna a uvolnění prostředků

Všimněme si, že při skončení práce s vláknem nikde explicitně neuvolňujeme prostředky s vláknem spojené. Jinými slovy chybí ekvivalent volání `CloseHandle`, jak jsme jej viděli ve Windows. Je to dáno tím, že o uvolnění těchto prostředků se stará funkce `pthread_join`.

Pokud bychom chtěli vlákno, které jen spustíme a necháme jej běžet s tím, že nás výsledek nezajímá, nebudeme mít v kódu vhodné místo pro volání `pthread_join`. V takovém případě musíme vlákno vytvořit s atributem `PTHREAD_CREATE_DETACHED`[8] nebo tento atribut nastavit za běhu funkcí `pthread_detach`. Pokud toto chceme nastavit u právě běžícího vlákna, můžeme použít funkci `pthread_self`, která vrací identifikátor aktuálně běžícího vlákna.

Úkoly:

1. Odstraňte z kódu volání `msleep`, případně změnte jeho argumenty, a sledujte, jak se změní průběh programu.
2. Rozšiřte ukázkový program, aby pracoval obecně s `N` vlákny, kde `N` je konstanta zadaná v kódu programu.
3. Naprogramujte funkci `int parfib(int)`, která spočítá Fibonacci číslo rekurzivním způsobem s využitím právě dvou vláken. Dvě počáteční větve výpočtu spusťte v samostatných vláknech.
4. Naprogramujte funkci `int pmin(int *numbers, unsigned int count, unsigned int threads)`, která použije `threads` vláken k tomu, aby v poli `numbers`, které obsahuje `count` hodnot, našla nejmenší hodnotu.

[4] V ukázkovém případě se využívá přetypování mezi číslem a typem ukazatel. Toto je relativně běžný postup, jak předávat do (resp. z) vlákna celočíselné hodnoty.

[5] Standardně je k dispozici funkce `sleep`, která pracuje s rozlišením na sekundy a `nanosleep`, která pracuje s rozlišením na nanosekundy.

[6] Novější verze GNU/Linux.

[7] Starší verze GNU/Linux.

[8] Viz funkce `pthread_attr_init`.