PB152cv/07 Cvičenie 10

(paralelizácia vláknami, súbežný prístup k pamäti)

Jozef Sabo

01. 12. 2025

Masarykova Univerzita, Brno

Cvičenie 10

Funkčné smerníky (funkčné ukazatele, function pointers) (cvičenie 00)

smerník, ktorý obsahuje adresu umiestnenia kódu funkcie

smerník získame bežným spôsobom: `&<meno_funkcie>`

vieme uložiť do premennej:

```
<návratový typ> (*<meno_premennej>) (<typ_argumentu1>, <typ_argumentu2>, ...)
```

využitie napríklad pri porovnávaní pomocou funkcie `qsort()` zo `<stdlib.h>`

príklad:

```
int funkcia( int a, int b )
{ ... }
int ( *smernik_na_funkcu )( int, int ) = &funkcia;
smernik_na_funkcu( 3, 5 );  // volanie funkcie, rovnako ako funkcia( 3, 5 )
```

Rozdeľovanie procesu do vláken

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

`pthread_t *thread` – smerník na typ `pthread_t` kam bude uložený identifikátor nového vlákna (Thread IDentifier)

`pthread_attr_t *attr` – smerník na štruktúru atribútov nového vlákna (zvyčajne `NULL`)

`void *(*start_routine)(void*)` – smerník funkciu, ktorá bude spustená vo vlákne

`void *arg` – počiatočná informácia pre spustené vlákno, najčastejšie štruktúra s dátami

parameter `start_routine` odkazuje na funkciu s jediným parametrom typu `void*` a návratovým typom `void*` – takto je možné odovzdávať informáciu po ukončení výpočtu vlákna (vraciame cez `return`)

Ukončenie vlákna

```
void pthread_exit( void *value_ptr );
```

`void *value_ptr` – infomácia odovzdávaná vláknu čakajúcemu na synchronizáciu
(alternatívne (a častejšie) sa volá `return value_ptr;`)

ukončenie vlastného vlákna (samého seba)

Čakanie na ukončenie vlákna

```
int pthread_join( pthread_t thread, void **value_ptr );
```

`pthread_t thread` – TID vlákna, na vlákno ktorého sa čaká

`void **value_ptr` – smerník na typ `void*` kam sa uloží návratová infomácia vlákna (odovzdaná volaním `pthread_exit()` alebo `return` vo vlákne)

uvoľňuje zdroje spojené s vláknom

vlákna nemajú medzi sebou hierarchiu, jedno vlákno môže ukončiť hociktoré susedné

každé vlákno môže ukončiť aj vlákno rodičovské, čím sa ale ukončí rodičovský proces a tým sa ukončia aj všetky vlákna tohto procesu

Zdieľanie pamäte vo vláknach

vlákna bežia v rámci procesu

na rozdiel od procesov, vlákna zdieľajú pamäť – tá je spoločná pre celý proces

z tohto dôvodu môžu dve rôzne vlákna pristupovať k pamäti súbežne

čítanie a zápis takejto pamäte nie je usporiadaný reláciou predchádzania (relácia predchádzania – skriptá PB152 z roku 2025, strana 3, odrážka B.3.1)

z tohto dôvodu vzniká hazard súbehu, na ktorý si je potrebné dávať pozor!

môže nastať uviaznutie (deadlock – 4 podmienky: vzájomné vylúčenie, čakajúci vlastník, neodnímateľnosť a čakanie do kruhu)

riešenie – atomické čítanie a zápis, synchronizačné primitíva

Atomické premenné (`-std=c11, stdatomic.h`)

zabezpečujú atomický zápis a čítanie (dáta sa uložia do pamäťovej bunky, nielen do registra, zapíšu sa a prečítajú sa celé, alebo vôbec)

najčastejšie nami používané typy:

`atomic_int`

`atomic_uint`

`atomic_long`

`atomic_ulong`

ďalšie typy je možné nájsť na cppreference.com (klik priamo na stránku)