IB109 Návrh a implementace paralelních systémů

Programování v prostředí se sdílenou pamětí

Jiří Barnat

Rizika spojená se sdílenou pamětí

Race condition

Pozorování

- Paralelní programy mohou při opakovaném spouštění zdánlivě náhodně vykazovat různá chování.
- Výsledek provedení programu může záviset na absolutním pořadí provedení instrukcí programu, tj. na proložení instrukcí zúčastněných procesů/vláken.

Race condition

- Nedokonalost paralelního programu, která se projevuje takovýmto nedeterministickým chováním se označuje jako **race condition**, (zkráceně race).

Race condition – příklad

Příklad

```
*myStructure p;

P0 {
  p = new myStructure;
  p -> data = 1;
  cout <<(p->data)<<endl;
}

P1 {
  p = new myStructure;
  p -> data = 2;
  cout <<(p->data)<<endl;
}
```

Atomicita operací

Pozorování

- Jednoduchý příkaz ve vyšším programovacím jazyce neodpovídá nutně jedné instrukci procesoru.
- V moderních operačních systémech je každé vlákno podrobeno plánovacímu procesu.
- Vykonání posloupnosti instrukcí procesoru odpovídající jednomu příkazu vyššího programovacího jazyka může být přerušeno a proloženo vykonáním instrukcí jiného vlákna.

Atomicita operací

Příklad

- Přičtení čísla do proměnné efektivně může znamenat načtení proměnné do registru, provedení aritmetické operace, a uložení výsledku do paměti.
- Při vhodném souběhu následujících procesů, se může efekt jednoho přiřazení do sdílené globální proměnné zcela vytratit

```
volatile int a=0;

P0 {                    P1 {
  a = a + 10;             a = a + 20;
}                       }
```

- Demonstrujte proložení instrukcí, které vyústí v jinou hodnotu, než 30.

Relativní rychlost výpočtu

Pozorování

- Nelze spoléhat na současný souběh vláken, potažmo relativní rychlost výpočtu jednotlivých vláken.

Příklad

```
volatile int a=0;

    P0 {                 P1 {
      usleep 200;          a = 1;
      a = 0;               usleep 200;
    }                    }
```

- Po skončení obou vláken (současně spuštěných) bude mít sdílená proměnná ve většině případů hodnotu 0. Není to však ničím garantováno, tj. může nastat situace, kdy bude mít hodnotu 1.

Uváznutí (Deadlock)

- Pokud mají vlákna inkrementální požadavky na unikátní sdílené zdroje, může dojít k tzv. uváznutí, tj. nemožnosti pokračování ve výpočtu.

Příklad

```
P0 {                    P1 {
  zamkni A;               zamkni B;
  zamkni B;               zamkni A;
...                     ...
}                       }
```

Hladovění, Stárnutí, Neprogrese (Livelock)

- Jev, kdy alespoň jedno vlákno není schopné vzhledem k paralelnímu souběhu s jiným vláknem pokročit ve výpočtu za danou hranici.

Příklad

- ```
  volatile int a=0;
  P0 {                        P1 {
    while (true) {;             while (a == 0) { };
      a++; a--;                 ...;
    }...
  }                           }
  ```

- Vlákno P1 může na vyznačeném řádku strávit mnoho času.
  ```

Thread-safe a re-entrantní procedury

Thread-safe procedura

- Označení procedury či programu, jejíž kód je bezpečné provádět (vzhledem k sémantice výstupu a stabilitě výpočtu) souběžné několika vlákny bez nutnosti vzájemné domluvy/synchronizace.
- Knihovní funkce nemusí být thread-safe!
  - rand() -> rand_r()

Re-entrantní procedura

- Procedura, jejíž provádění může být v rámci jednoho vlákna přerušeno, kód kompletně vykonán od začátku do konce v rámci téže úlohy, a poté obnoveno/dokončeno přerušené vykonávání kódu.
- Termím pochází z dob, kdy nebyly multitaskingové operační systémy.

Vztah thread-safe a re-entrantních procedur

Neporovnatelné

- Re-entrantní procedura nemusí být thread-safe.
  viz `http://en.wikipedia.org/wiki/Reentrancy_(computing)`
- Thread-safe procedura nemusí být re-entrantní.
  (Problémem je například používání globálních zámků.)

Příklad

- Thread-safe procedura, která není re-entrantní:

```
WC {
  je-li odemčeno, vejdi a zamkni, jinak čekej
  ...
  odemkni a opusť onu místnost
}
```

Thread-safe a re-entrantní procedury

Nebezpečné akce vzhledem k paralelnímu zpracování

- Nekontrolovaný přístup ke globálním proměnným a haldě.
- Uchovávání stavu procedury do globálních proměnných.
- Alokace, dealokace zdrojů globálního rozsahu (soubory, ...).
- Nepřímý přístup k datům skrze odkazy nebo ukazatele.
- Viditelný vedlejší efekt (modifikace nestálých proměnných).

Bezpečná strategie

- Přístup pouze k lokálním proměnným (zásobník).
- Kód je závislý pouze na argumentech dané funkce.
- Veškeré volané podprocedury a funkce jsou thread-safe.

Procedura, která není thread-safe

```
*myStructure p;

*myStructure function() {
  p=new myStructure;
  return p;
}

P0 {
  *myStructure x;
  x=function();
}
```

```
P1 {
  *myStructure x;
  x=function();
}
```

Přístup ke sdíleným proměnným

Pozorování

- Přístup ke sdíleným proměnným je „kořenem všeho zla“.
- Přístup a použití sdílených proměnných se musí provádět kontrolovaně.

Zamykání a kritické sekce

- Část kódu, jehož provedení je neproložitelné instrukcemi jiného vlákna.
- Problém realizace kritické sekce musí být řešen způsobem, který je odolný vůči plánování.

Jednoduché řešení

- Sdílená atomicky přistupovaná bitová proměnná, jejíž hodnota indikuje přítomnost procesu/vlákna v přidružené kritické sekci.
- Manipulována při vstupu a výstupu do/z kritické sekce.
- Vyžaduje podporu HW pro atomickou manipulaci.

Zamykání a související pojmy

Petersonův algoritmus (spinlock, user-space)

- Algoritmus pro vzájemné vyloučení.
- Nezpůsobuje stárnutí ani uváznutí.
- Pozor na implementaci a provádění instrukcí mimo pořadí.

Uspávání

- Procesy/vlákna se po neúspěchu vstoupit do kritické sekce sami vzdají procesorového kvanta (uspí se).
- Jsou buzeny buď po vypršení časového limitu nebo explicitně jiným běžícím vláknem.

Spinlock

- Vlákna opakovaně zkouší vstoupit do kritické sekce.
- Pro krátké čekací doby je efektivnější, než přepínání kontextů vláken, natož pak procesů.

Výkonostní rizika

Přístup ke sdíleným globálním proměnným

- Veškeré modifikace a neatomická čtení globálních proměnných musí být serializovány, tj. prováděny po získání odpovídajícího zámku na danou operaci.
- Získání zámku vynucuje vylití cache pamětí.
- Mnoho přístupů k zamykaným proměnným může být úzkým místem výkonu aplikace.

Lokální data vláken (Thread-private data)

- Vlákna mají své lokální proměnné.
- Data mohou uloženy v globální sdílené struktuře, pokud odpovídající část datové struktury je přistupována pouze daným vláknem.
- Typicky pole indexovaná unikátním identifikátorem vlákna.
- Riziko falešného sdílení.

POSIX Thread API

Historie a POSIX standard

Historie

- SMP systémy
- Vlákna implementována jednotlivými výrobci HW
- IEEE POSIX 1003.1c standard

IEEE POSIX 1003.1c

- Programátorský model semaforů a provádění operací v kritické sekci
- Rozhraní pro C
- POSIX threads, PThreads

Jiné normy

- Operační systémy: NT Threads (Win32), Solaris threads, ...
- Programovací jazyky: Java threads, C++11 threading, ...

Základní dělení funkcionality

Správa vláken

- Vytváření, oddělování a spojování vláken
- Funkce na nastavení a zjištění stavu vlákna

Vzájemná vyloučení (mutexes)

- Vytváření, ničení, zamykání a odemykání mutexů
- Funkce na nastavení a zjištění atributů spojených s mutexy

Podmínkové/podmíněné proměnné (conditional variable)

- Slouží pro komunikaci/synchronizaci vláken
- Funkce na vytváření, ničení, "čekání na" a "signalizování při" specifické hodnotě podmínkové proměnné
- Funkce na nastavení a zjištění atributů proměnných

POSIX standard

Přes 60 API funkcí

- `#include <pthread.h>`
- Překlad s volbou `-pthread`

Mnemotechnické předpony funkcí

- `pthread_`, `pthread_attr_`
- `pthread_mutex_`, `pthread_mutexattr_`
- `pthread_cond_`, `pthread_condattr_`
- `pthread_key_`

Pracuje se skrytými objekty (Opaque objects)

- Objekty v paměti, o jejichž podobě programátor nic neví
- Přistupovány výhradně pomocí odkazu (handle)
- Nedostupné objekty a neplatné (dangling) reference

Atributy objektů

Idea

- Vlastnosti všech vláken, mutexů i podmínkových proměnných nastavovány speciálními objekty
- Některé vlastnosti entity musí být specifikovány již v době vzniku entity

Typy atributových objektů

- Vlákna: `pthread_attr_t`
- Mutexy: `pthread_mutexattr_t`
- Podmínkové proměnné: `pthread_condattr_t`

Vznik a destrukce

- Funkce `_init` a `_destroy` s odpovídající předponou
- Parametr odkaz na odpovídající atributový objekt

Správa vláken

Vytváření vlákna

- Každý program má jedno hlavní vlákno
- Další vlákna musí být explicitně vytvořena programem
- Každé vlákno (i vytvořené) může dále vytvářet další vlákna
- Vlákno vytvářeno funkcí `pthread_create`
- Vytvářené vlákno je ihned připraveno k provádění
- Může být plánovačem spuštěno dříve, než se dokončí volání vytvářecí funkce
- Veškerá data potřebná při spuštění vlákna, musí být připravena před voláním vytvářecí funkce
- Maximální počet vláken je závislý na implementaci

Správa vláken

```
int pthread_create (
        pthread_t *thread_handle,
        const pthread_attr_t *attribute,
        void * (*thread_function)(void *),
        void *arg);
```

- `thread_handle` odkaz na vytvořené vlákno
- `attribute` odkaz na atributy vytvořeného vlákna (NULL pro přednastavené nastavení atributů)
- `thread_function` ukazatel na funkci nového vlákna
- `arg` ukazatel na parametry funkce `thread_function`
- Při úspěšném vytvoření vlákna vrací 0

Správa vláken

Ukončení vlákna nastává

- Voláním funkce `pthread_exit`
- Pokud skončí hlavní funkce rodičovského vlákna jinak než voláním `pthread_exit`
- Je-li zrušeno jiným vláknem pomocí `pthread_cancel`
- Rodičovský proces je ukončen (násilně nebo voláním `exit`)

```
void pthread_exit (void *value)
```

- Ukončuje běh vlákna
- Odkazy na prostředky procesu (soubory, IPC, mutexy, ...) otevřené v rámci vlákna se nezavírají
- Data patřící vláknu musí být uvolněna před ukončením vlákna (systém provede uvolnění prostředků až po skončení rodičovského procesu)
- Ukazatel `value` předán při spojení vláken

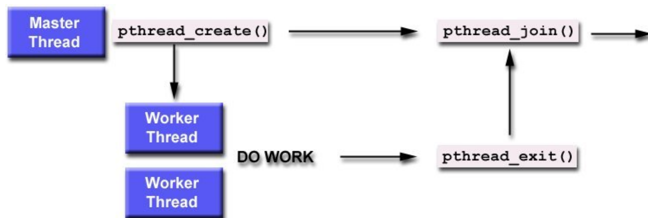
Správa vláken – příklad

```
1   #include <pthread.h>
2   #include <stdio.h>
3   #define NUM_THREADS 5
4
5   void *PrintHello(void *threadid)
6   {   printf("%d: Hello World!\n", threadid);
7       pthread_exit(NULL);
8   }
9
10  int main (int argc, char *argv[])
11  {   pthread_t threads[NUM_THREADS];
12      for(int t=0; t<NUM_THREADS; t++)
13        pthread_create(&threads[t], NULL,
14                       PrintHello, (void *)t);
15      pthread_exit(NULL);
16  }
```

Správa vláken

```
int pthread_join (pthread_t thread_handle,
                  void **ptr_value);
```

- Čeká na dokončení vlákna `thread_handle`
- Hodnota `ptr_value` je ukazatel na pointer specifikovaný vláknem `thread_handle` při volání `pthread_exit`
- Nutný například pokud main má vracet smysluplnou návratovou hodnotu



Vlastnosti (atributy) vláken

Nespojitelná vlákna (Detached threads)

- Nemohou být spojena voláním funkce `pthread_join`
- Šetří systémové prostředky
- Přednastavené nastavení typu vlákna není vždy zřejmé, proto je doporučeno typ vlákna explicitně nastavit

```
int pthread_detach (
      ptrhead_t *thread_handle)

int pthread_attr_setdetachstate(
      pthread_attr_t *attr,
      int detachstate)

int pthread_attr_getdetachstate(
      pthread_attr_t *attr,
      int *detachstate)
```

Vlastnosti (atributy) vláken

Velikost zásobníku

- Minimální velikost zásobníku není určena
- Při velkém počtu vláken se často stává, že vyhrazené místo pro zásobník je vyčerpáno
- POSIX umožňuje zjistit a nastavit pozici a velikost místa vyhrazeného pro zásobník jednoho vlákna

```
int pthread_attr_getstacksize (
      pthread_attr_t *attribute, size_t *stacksize)
int pthread_attr_setstacksize (
      pthread_attr_t *attribute, size_t stacksize)
int pthread_attr_getstackaddr (
      pthread_attr_t *attribute, void **stackadr)
int pthread_attr_setstackaddr (
      pthread_attr_t *attribute, void *stackadr)
```

Zrušení vlákna

```
int pthread_cancel (
      ptrhead_t *thread_handle)
```

- Žádost o zrušení vlákna `thread_handle`
- Adresované vlákno se může a nemusí ukončit
- Vlákno může ukončit samo sebe
- Při zrušení se provádí úklid dat spojených s rušeným vláknem
- Funkce skončí po odeslání žádosti (je neblokující)
- Návratový kód 0 značí, že adresované vlákno existuje, ne že bylo/bude zrušeno

Domácí úkol (5 bodů)

Zadání

- Program srovnávající rychlost vytváření procesů a vláken.
- Program, který realizuje rušení vláken (`pthread_cancel`).
- Výstup obou programů na konzolu je samovysvětlující.
- Při spuštění oba programy vypíší autorovo UČO.
- Spustitelné a přeložitelné na serveru `aisa`.

Odevzdání

- Termín do 16. 3. 2016, 23:59.
- Odevzdávárna v ISu, zabaleno programem TAR a komprimováno GZIPem: *IB109_01_učo.tar.gz*
- Archiv obsahuje sbalený adresář IB109_01_učo.
- Povinně obsahuje Makefile.
- Provedení `make` uvnitř adresáře přeloží a spustí aplikace.
- Nesprávné odevzdání jde na vrub studenta.