Konkurentní datové struktury

14. března 2018

B4B36PDV – Paralelní a distribuované výpočty

Minulé cvičení:

“Paralelní programování v OpenMP...”

Minulé cvičení:

“Paralelní programování v OpenMP...”

Minulé cvičení:

“Paralelní programování v OpenMP...”

Minulé cvičení:

“Paralelní programování v OpenMP...”

Dnešní menu: Konkurentní datové struktury

Osnova

- Opakování z minulého cvičení
- Zámková architektura datových struktur
- Bezzámková architektura datových struktur

- Zadání třetí domácí úlohy

Opakování z minulého cvičení

http://goo.gl/a6BEMb

Tvorba konkurentních datových struktur

Co bychom si přáli?

Aby jednu strukturu používalo více vláken **současně**.

Co musíme změnit oproti frontě z prvního domácího úkolu?

Co bychom si přáli?

Aby jednu strukturu používalo více vláken **současně**.

Co musíme změnit oproti frontě z prvního domácího úkolu?

- Nesmíme zamykat **celou** datovou strukturu!
- Se zamykáním zámků musíme šetřit

⚠ Jinak se datová struktura stane brzdou výpočtu!

Možný přístup:

1. Vezmu kód existující jednovláknové datové struktury
2. Ve chvíli, kdy strukturu dělám nějaké zásahy, zamknu si část struktury pro sebe

Je to opravdu takto lehké?

Možný přístup:

1. Vezmu kód existující jednovláknové datové struktury
2. Ve chvíli, kdy strukturu dělám nějaké zásahy, zamknu si část struktury pro sebe

Je to opravdu takto lehké?

Typický vzor práce s jednovláknovými strukturami:

Příprava $\rightarrow$ "Poškození" dat $\xrightarrow{\text{↯}}$ Oprava $\rightarrow$ Hotovo!

Možný přístup:

1. Vezmu kód existující jednovláknové datové struktury
2. Ve chvíli, kdy strukturu dělám nějaké zásahy, zamknu si část struktury pro sebe

Je to opravdu takto lehké?

Typický vzor práce s jednovláknovými strukturami:

$$\text{Příprava} \rightarrow \text{“Poškození” dat} \xrightarrow{\text{↯}} \text{Oprava} \rightarrow \text{Hotovo!}$$

Musíme zabránit použití “rozbité” části = vyloučit i čtenáře
(a zamykat si části struktury, i když to není potřeba – např. při čtení)

Možný přístup:

1. Vezmu kód existující jednovláknové datové struktury
2. Ve chvíli, kdy strukturu dělám nějaké zásahy, zamknu si část struktury pro sebe

Je to opravdu takto lehké?

Typický vzor práce s jednovláknovými strukturami:

Příprava $\rightarrow$ "Poškození" dat $\overset{\text{⚡}}{\rightarrow}$ Oprava $\rightarrow$ Hotovo!

Musíme zabránit použití "rozbité" části = vyloučit i čtenáře
(a zamykat si části struktury, i když to není potřeba – např. při čtení)

To si ale moc nepomůžeme :-(

Jak to tedy udělat lépe?

1. Strčit hlavu do písku a (téměř) nezamykat
2. Když nastane problém, tak ho (nějak) vyřešit

To se snáz řekne, než udělá...

Jak to tedy udělat lépe?

1. Strčit hlavu do písku a (téměř) nezamykat
2. Když nastane problém, tak ho (nějak) vyřešit

To se snáz řekne, než udělá...

Některé z možných problémů:

- **Datová struktura se může nacházet v mezistavu:**
  Buď musí být použitelná, nebo si musíme být jistí, že problém detekujeme, než poškodíme data
- **Nesmíme uvolnit paměť, pokud s ní pracuje jiné vlákno:**
  Složitější datové struktury často využívají techniky podobné garbage-collectoru v Javě.

Jak to tedy udělat lépe?

O takových datových strukturách se těžko přemýšlí...

... a ještě hůř se v nich hledají chyby!

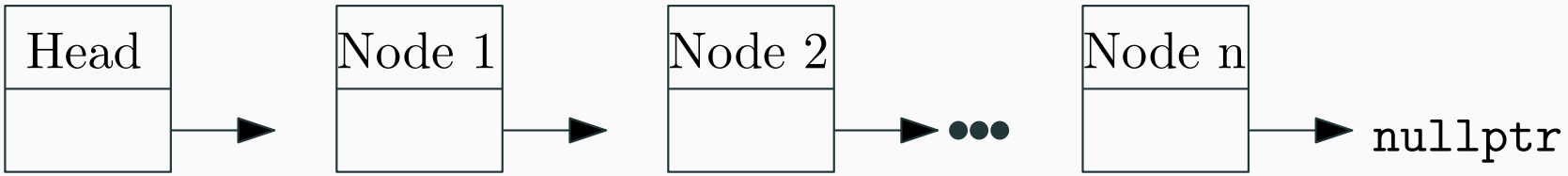
- `http://libcds.sourceforge.net`
- C++ Concurrency In Action: Practical Multithreading

Cvičení: konkuretní spojový seznam

Konkurentní spojový seznam

Reprezentace seznamu prvků

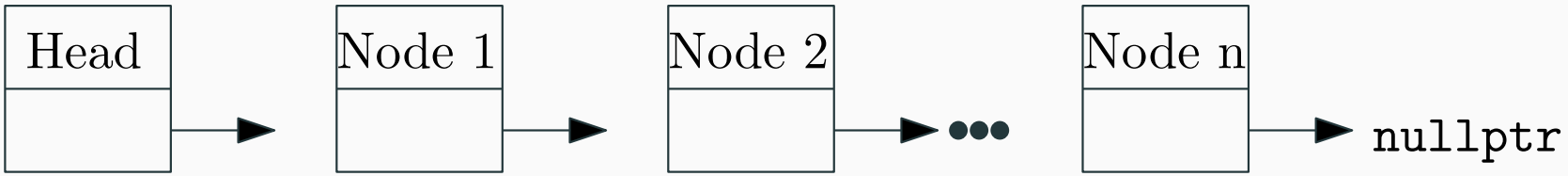
- My ho budeme chtít mít seřazený vzestupně...
- Vložení prvku = nalezení správné pozice + vložení nového uzlu



Konkurentní spojový seznam

Reprezentace seznamu prvků

- My ho budeme chtít mít seřazený vzestupně...
- Vložení prvku = nalezení správné pozice + vložení nového uzlu



Doimplementujte metodu `insert`

Doimplementujte tělo metody `insert` v souboru `lockBased.cpp`. Pro synchronizaci vláken použijte `spin_lock` (používá se stejně jako `std::mutex`), který umístíte ke každému uzlu seznamu. Snažte se zámky zamykat pouze na čas modifikace seznamu a pouze tam, kde jsou potřeba!

Data race

Nesynchronizované přístupy do paměti (s alespoň jedním zápisem) jsou ve standardu C++ vedené jako

⚠ *undefined behavior* ⚠

Může se nám stát spousta špatných věcí, například:
(ty navíc závisí na kompilátoru a platformě)

- Můžeme přečíst částečně zapsaná data (*dirty read*)
- Vlákno se nedozví o změně provedené jiným vláknem
- Vlákno se dozví pouze o části provedených změn

Data race

Nesynchronizované přístupy do paměti (s alespoň jedním zápisem) jsou ve standardu C++ vedené jako

⚠ *undefined behavior* ⚠

Může se nám stát spousta špatných věcí, například:
(ty navíc závisí na kompilátoru a platformě)

- Můžeme přečíst částečně zapsaná data (*dirty read*)
- Vlákno se nedozví o změně provedené jiným vláknem
- Vlákno se dozví pouze o části provedených změn

`std::atomic`

Synchronizace přístupů ke stejné proměnné zajištěna

Můžeme se zámků zbavit úplně?

Bezzámková architektura datových struktur

Navrhnout správně zamykání je náročné

- Špatné použití může vést k deadlocku
- Velké množství zámků snižuje potenciál opravdové konkurence
- Paměťový overhead (`std::mutex` na Linuxu má 40B!)

Navrhnout správně zamykání je náročné

- Špatné použití může vést k deadlocku
- Velké množství zámků snižuje potenciál opravdové konkurence
- Paměťový overhead (`std::mutex` na Linuxu má 40B!)

Jak na to?

Navrhnout správně zamykání je náročné

- Špatné použití může vést k deadlocku
- Velké množství zámků snižuje potenciál opravdové konkurence
- Paměťový overhead (`std::mutex` na Linuxu má 40B!)

Jak na to?

$\rightarrow$ Pomocí atomických operací s pamětí

Porovnej a prohoď

Klíčová operace pro *lock-free* datové struktury.

Porovnej a prohoď (neboli *compare-and-swap*) je atomická operace s pamětí na objektu `C`, definovaná v C++ jako

```
bool compare_exchange_strong( E& expected, D desired)
```

která ma funkcionalitu ekvivalentní

```
if ( C == E) C = D;
else E = C;
```

Porovnej a prohoď

Klíčová operace pro *lock-free* datové struktury.

Porovnej a prohoď (neboli *compare-and-swap*) je atomická operace s pamětí na objektu `C`, definovaná v C++ jako

```
bool compare_exchange_strong( E& expected, D desired)
```

která ma funkcionalitu ekvivalentní

```
if ( C == E) C = D;
else E = C;
```

Kontrolu a změnu datové struktury lze provést **atomicky**!

Doimplementujte metodu `insert`

Doimplementujte tělo metody `insert` v souboru `lockFree.cpp`. Namísto použití zámků nyní použijte atomickou operaci *compare-and-swap* pro úpravu pointerů ve spojovém seznamu.

Bonusové úlohy:

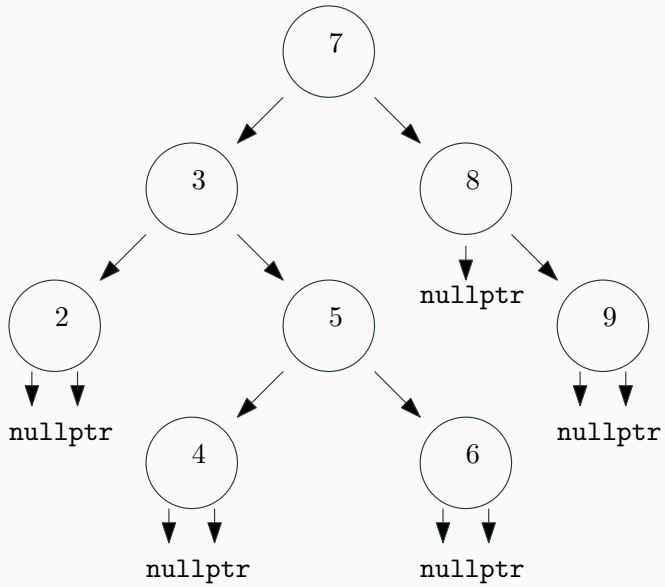
1. Zkuste se zamyslet, zda byste dokázali naimplementovat i dvousměrný spojový seznam
2. Zkuste naimplementovat spojový seznam, ve kterém může kromě přidávání docházet konkurentně i k mazání prvků

Zadání třetí domácí úlohy

Konkurentní binární vyhledávací strom

Struktura, v níž jsou jednotlivé prvky uspořádány tak, aby bylo možné rychle vyhledávat

- každý uzel tedy má nanejvýš dva potomky;
- každému uzlu je přiřazen určitý klíč;
- levý podstrom uzlu obsahuje klíče menší než je klíč uzlu; a
- pravý podstrom uzlu obsahuje klíče větší než je klíč uzlu.



Konkurentní binární vyhledávací strom

Naimplementujte metody v `binaryTree.cpp` a `binaryTree.h` a zajistěte, že

1. každý prvek je vložet právě jednou; a
2. žádný vložený prvek se neztratí.

Zpracování musí být **konkurentní**, nikoli serielní!

Konkurentní binární vyhledávací strom

Naimplementujte metody v `binaryTree.cpp` a `binaryTree.h` a zajistěte, že

1. každý prvek je vložet právě jednou; a
2. žádný vložený prvek se neztratí.

Zpracování musí být **konkurentní**, nikoli serielní!

Za spravné výsledky a vysoký stupeň konkurence dostanete až **2b**.

Konkurentní binární vyhledávací strom

Naimplementujte metody v `binaryTree.cpp` a `binaryTree.h` a zajistěte, že

1. každý prvek je vložet právě jednou; a
2. žádný vložený prvek se neztratí.

Zpracování musí být **konkurentní**, nikoli serielní!

Za spravné výsledky a vysoký stupeň konkurence dostanete až **2b**.

Termín odevzdání je **22.3. 23:59 CET** pro středeční cvičení a
23.3. 23:59 CET pro čtvrteční cvičení.

Soubory `binaryTree.cpp` a `binaryTree.h` nahrajte do systému BRUTE.

Díky za pozornost!

Budeme rádi za Vaši zpětnou vazbu! →

https://goo.gl/forms/EaPMqnYRCQZrUztX2