Datové struktury I

12. přednáška: Paralelní programování bez zámků

Jirka Fink

https://ktiml.mff.cuni.cz/~fink/

Katedra teoretické informatiky a matematické logiky
Matematicko-fyzikální fakulta
Univerzita Karlova v Praze

Zimní semestr 2024/25

Úvodní problém paralelního programování

Přičítání jedničky ke sdílenému čítači

Proč následující postup přičtení jedničky nefunguje, když k čítači přistupuje více vláken najednou?

1. local_copy ← shared_counter
2. local_copy ← local_copy + 1
3. shared_counter ← local_copy

Zámky (mutex) a vzájemné čekání (deadlock)

Zámek

Zabraňuje tomu, aby byly současně vykonávány dva (nebo více) kritické kódy nad stejným sdíleným prostředkem, jako například globální proměnné.

Granularita zámků v datových strukturách

- Jeden globální zámek
- Jeden zámek pro celou instanci datové struktury
- Zámek pro každou přihrádku (např. u hešování)
- Zámek pro každý prvek (např. ve stromu)

Vzájemné čekání (deadlock)

Úspěšné dokončení první akce je podmíněno předchozím dokončením druhé akce, přičemž druhá akce může být dokončena až po dokončení první akce. Příklad:

```
1  lock(A)
2  lock(B)
3  critical section
4  unlock(B)
5  unlock(A)
```

```
6  lock(B)
7  lock(A)
8  critical section
9  unlock(A)
10 unlock(B)
```

Řešení: Určit globální uspořádání zámků a vždy zamykat v tomto pořadí

Nevýhody zámku

1. Deadlock
2. Spravedlnost: Které vlákno získá uvolněný zámek?
3. Priorizace: Důležitější vlákno by nemělo čekat na pomalejší
4. Výkon: Zámky zpomalují výpočet, zejména při jejich větším využívání
5. Náchylnost na chyby: Při selhání jednoho vlákna může dojít k neuvolnění zámku

Atomické operace

Příklady

- Read and write: Prvek je možné celý najednou načíst a zapsat.
- Exchange: Prohození obsahu atomické a lokální proměnné.
- Test and set bit: Nastaví atomickou binární proměnnou na true a vrátí původní hodnotu.
- Fetch and add: Přičte k atomické proměnné danou hodnotu a vrátí původní hodnotu.
- Compare and swap (CAS): Pro atomický register R a hodnoty a a b nastaví R na b, pokud $R = a$, a vždy vrátí původní hodnotu.
- Load linked and store conditional (LL/SC): LL hodnotu registru R do lokální proměnné L a následná SC zapíše hodnotu z L do R, pokud mezitím nedošlo k jinému přístupu do R, jinak vrátí chybu.

Cvičení

Které z těchto atomických operací lze použít k implementaci zámků?

Zásobník bez zámků

Je tato implementace zásobníku správná?

```
1  Function push(node)
2      while true do
3          h ← head
4          node.next ← h
5          if CAS(head, h, node) == h then
6              return
```

```
7  Function pop()
8      while true do
9          h ← head
10         n ← h.next
11         if CAS(head, h, n) == h then
12             return h
```

Problém Livelock

Sice máme jistotu, že vždy alespoň jedno uspěje, ale teoreticky může jiné vlákno donekonečna cyklit.

Problém ABA

První vlákno je přerušeno mezi řádky 10 a 11, kdy druhé vlákno provede

```
1 A ← pop
2 B ← pop
3 push(A)
```

Zásobník bez zámků

Je tato implementace zásobníku správná?

```
1  Function push(node)
2      while true do
3          h ← head
4          node.next ← h
5          if CAS(head, h, node) == h then
6              return
```

```
7  Function pop()
8      while true do
9          h ← head
10         n ← h.next
11         if CAS(head, h, n) == h then
12             return h
```

Řešení problému ABA

- Použít LL/CS místo CAS
- Použít Wide CAS (Double CAS): pracuje s dvojicí sousedních buněk paměti
  V zásobníku máme za ukazatelem head uložení timestamp a testujeme

```
1 (h,t) ← (head,timestamp)
2 node.next ← h
3 if CAS((head,timestamp), (h,t), (node,t+1)) == (h,t) then
4     return
```

Zásobník bez zámků

Je tato implementace zásobníku správná?

```
1  Function push(node)
2      while true do
3          h ← head
4          node.next ← h
5          if CAS(head, h, node) == h then
6              return
```

```
7  Function pop()
8      while true do
9          h ← head
10         n ← h.next
11         if CAS(head, h, n) == h then
12             return h
```

Problém dealokace paměti

První vlákno je přerušeno mezi řádky 9 a 10, kdy druhé vlákno prvek odebere z fronty a dealokuje jej.

Dealokace paměti

Globální synchronizace

Ukládáme ukládáme uvolněné bloky paměti do seznamu a v bezpečném okamžiku paměť uvolníme.

Reference counting

Budeme počítat reference, ale může dojít dealokaci mezi získáním ukazatele na prvek a zvýšením čítače.

```
1  Function pop()
2      while true do
3          h ← head
4          Increment h.ref_count
5          if h ≠ head then
6              Decrement h.ref_cnt and retry the loop
7          n ← h.next
8          if CAS(head, h, n) == h then
9              Decrement h.ref_cnt and return h
10         Decrement h.ref_cnt
```

Předpokládáme, že uvolněná paměť bude v budoucnu využita ke stejnému typu objektu, protože na řádce 4 zvyšujeme čítač nějakého prvku.

Hazardní ukazatele

Každé vlákno má jeden/seznam hazardních ukazatelů.

```
1  Function pop()
2      while true do
3          h ← head
4          hp ← h
5          if h ≠ head then
6              Retry the loop
7          n ← h.next
8          if CAS(head, h, n) == h then
9              hp ← NULL
10             return h
```

Při uvolnění paměti si hazardní ukazatele uložíme do vyhledávací struktury a projdeme seznam bloků k uvolnění.

Problém producenta a spotřebitele

Fronta využívaná dvěma vlákny

- Chceme frontu implementovanou pomocí spojového seznamu
- Jedno vlákno producenta přidává prvku do fronty
- Jedno vlákno spotřebitele prvky vybírá
- Je následující postup správný?

Producent

```
1  node ← new node
2  node.data ← produced data
3  node.next ← NULL
4  last.next ← node
5  last ← node
```

Spotřebitel

```
6  if first.next ≠ NULL then
7      previous ← first
8      first ← first.next
9      delete previous
10     consume first.data
```

Jaké posloupnosti dvojic může pravé vlákno vypisovat

```
1  Atomic integers a = b = 0
2  while true do
3      increase(a)
4      increase(b)
```

```
5  while true do
6      print(a,b)
```

Garance paralelních datových struktur

1. Blocking: Operace může čekat neomezeně dlouho
2. Obstruction-free: Operace určitě doběhne, když jsou ostatní vlákna zastavena
3. Lock-free: Některé vlákno musí doběhnou, i když pro jiné může nastat live-lock.
4. Wait-free: Všechna vlákna doběhnou v konečném čase, tj. nemůže nastat live-lock.
5. Bounded wait-free: Všechna vlákna doběhnou v garantovaném čase.