Paralelní DS

cíl: DS pro paralelní operace na asynchronních procesech při sdílené paměti

chtění: konzistence, spolehlivost, správa paměti, tolerance k chybám, nezávislé na OS, z předpokladů nějakého HW

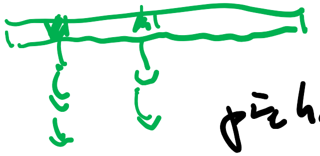
- blokující (zámky)
- obstruction-free: "když ostatní stojí, já uspěji"
- lock-free: "aspoň jeden uspěje v kon. čase"
- wait-free: "každý proces uspěje v konečném čase"
- bounded wait-free: navíc je známý odhad

Zámky (mutex)

použití: 1 na instanci

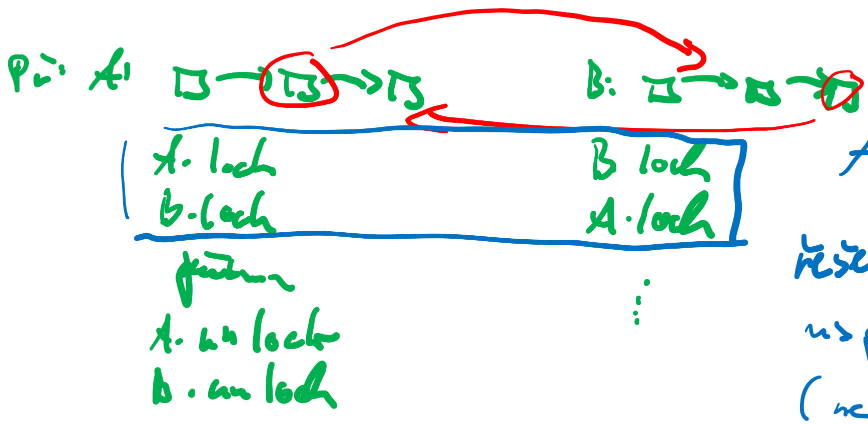
1 na prvek

1 po částech



přeházování!

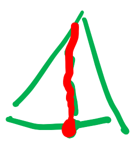
R/W mutex

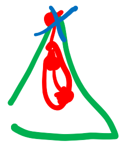


AB-BA deadlock

řešení: globální uspořádání zámků (ne vždy lze)

BVS s vyvažováním.





(a, 2a)-stromy ⇒ každý vrchol $a-1 \le$ # klíčů $\le 2a-1$ klíčů

INSERT: štěpení shora dolů: $2a-1$ klíčů → $a-1$ + $a-1$ + 1 (do rodiče)

invariant: aktuální vrchol má nejvýše $2a-2$ klíčů

zamykání: aktuální vrchol + rodič

DELETE: slučování shora dolů: $a-1 + a-1 + 1 \to 2a-1$

Problémy se zámky

- riziko deadlocku
- férovost
- inverze priorit
- výkon
- tolerance vůči chybám

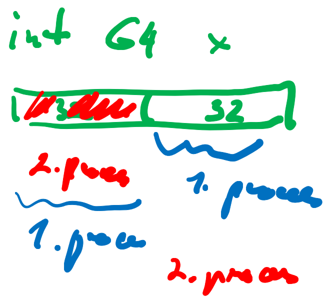
Lock-free struktury

předpoklad: atomické operace

vždy
- atomické registry – read, write (exchange)
- test&set bitu

1 z
- LL/SC – load lock / store conditional
- CAS – compare and swap

(k dispozici pouze pro 1 buňku)




```
CAS(adresa, x, y) : atomic {
    old ← [adresa]
    if old = x : [adresa] ← y
    return old
}
```

- …

Př.: Lock-free zásobník

head → [next] → [ ] → [ ] ⊥

```
Node:   atomic Node * next
        .... data ...

global: atomic NODE * head
```

```
Push(Node *n):
loop { h ← head
    n.next ← h
    head ← n        (crossed out)
    if CAS(&head, h, n) == h:
        return
}
```

⟹ lock-free

```
Pop:
loop { h ← head
    n ← h.next
    if CAS(&head, h, n) == h:
        return
}
```

head → B

Pop → A
Pop → B
Push(A)

head → A ⊥

Problém 1: ABA-problém

head → A → B ⊥

Řešení: 1) double CAS (CAS2)

```
if CAS2(<&head, &h.next>, <h,n>, <n,n>) == <h,n>:
    return
```

2) wide CAS (WCAS, DCAS) ... CAS2 na sousední buňky

verzování pointerů

```
if WCAS(<&head, &version>, <h,ver>, <n,ver++>) == <h,ver>:
    return
```

problém: přetečení (uvolnit nenadále)

3) omezení recyklace

Problém 2: Pop: $h \leftarrow head$

$n \leftarrow h.next$

jiný proces uvolnil head ⇒ memory acc. fault

správa paměti: uvolnit paměť pouze pokud se do ní nikdo nedívá

freelist odpojených uzlů, časem uvolnit

a) synchronizační body (kde jen někdy)

b) reference counting

c) hazard pointery

HP[PID]

- amortizace průchodu free listu

$\#$ procesů $\le P$, $\#$ referencovaných $\le R$

IDEA: projdi freelist $\ge 2PR$

⇒ odstraníte aspoň PR

```
Pop: loop { h ← head
    HP ← h
    if h ≠ head: continue
    n ← h.next
    if CAS(&head, h, n) == h:
        return
    HP ← ∅
}
```

Problém 3: přehledně

```
x = 1
⋮
a = x
```

Jak se bránit: `volatile int x` (jazyk C)

Problém 4: přerušitelné operace

```
a = 1
b = 2
```

Jak se bránit: bariéry
- univerzální
- čtení / zápisy
- acquire / release

```
m.lock()      ← acquire
... čtení ...
... zápisy ...
m.unlock()    ← release
```

C/C++ 11: paralelní programovací model
atomické operace, CAS
bariéry, zámky