Další problémy

① překladač

```
x = 0
while (x == 0);
```

```
x = 1
...
a = x
...
b = x
```

```
if (a)
{
  x = b; ...
}
else
{
  y = b; ...
}
```

volatile int x;

② hardware

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

→ bariéry — univerzální / jen čtení / zápis / acquire / release

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

C / C++ 11 — paralelní paměťový model, atomické operace, bariéry, zámky

paralelní DS:

- blokující
- obstruction-free (pokud ostatní stojí, já určitě dokončím operaci)
- lock-free (alespoň 1 v konečném čase uspěje)
- wait-free (já v konečném čase uspěji)
- bounded wait-free ↳ + nějaký horní odhad

[Herlihy 1991]

Protokol pro konsensus

- procesy nabídnou hodnoty $x_1 \dots x_n$
- všichni dostanou stejnou odpověď
- odpověď je 1 z nabízených hodnot

k

Pro 2 procesy stačí test-and-set

$p \in \{0,1\}$

```
decide(x):  want[p] ← x
            if TAS(bit) == 0:
                return want[p]
            else:
                return want[p̄]
```

Pro lib. # procesů CAS:

reg ∅

```
decide(x):  y = CAS(reg, ∅, x)
            if y = ∅:
                return x
            else:
                return y
```

Věta: Pro 3 procesy TAS nestačí.

Def: Consensus number pro protokol X: max # procesů, pro které lze konsensus realizovat na X + atomické registry

---

Atomické operace

Cons. #

- atomické registry — 1
- exchange
- test and set bitu
- fetch & add
- paralelní přiřazení do n registrů — 2n−2
- LL/SC load locked, store conditional — 2
  - x ← [adresa]
  - [adresa] ← y
- CAS compare & swap — ∞

CAS(adresa, x, y)

```
atomic {
    old ← [adresa]
    if old == x: [adresa] ← y
    return old
}
```

Korektnost: linearizovatelnost

↳ na operacích s objektem

int64 x

x ← 123456789 ns

32	32

Univerzální bounded wait-free protokol [Herlihy]

$Op: Stav \times Arg \to Stav \times Res$

Stav struktury

Cell:
- seq (int) – pořadové číslo
- arg – argument operace, kterou vznikl akt. stav
- new – akt. stav (objekt pro konsensus)
- next – ukazatel na další buňku (obj. pro konsensus)
- prev – předch. buňka (atom. ukazatel)

globální:
- announce[P] ← právě přidaná buňka
- head[P] ← ukazatel na konec seznamu

na poč. kotva

"<" na buňkách porovnává seq

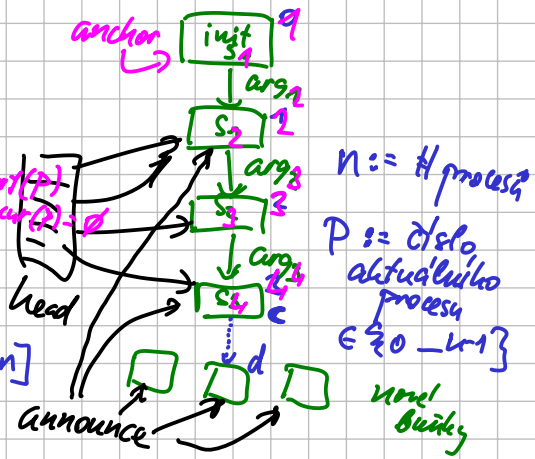
min(cell1, cell2)

max(head) := max c (c ∈ head)

Operate(arg):

1. mine ← Cell(seq=0, arg=arg, new=∅, next=∅, prev=∅)
2. announce[P] ← mine • start(P) • concur(P) = ∅
3. head[P] ← max(head)
4. while mine.seq = 0:
5. c ← Head[P]
6. help ← announce[c.seq % n]
7. if help.seq = 0: prefer ← help
   else: prefer ← mine
8. d ← c.next.decide(prefer)
9. d.new.decide(Op(c.new, d.arg))
10. d.prev ← c
11. d.seq ← c.seq + 1
12. head[P] ← d
13. head[P] ← mine
14. Return mine.new

přidává do všech concur(*)



start(P) ← max(head).seq v okamžiku announcování procesem P

concur(P) := množina buněk, které přibyly do hlav od okamžiku announce v P

---

max(head).seq = start(P) + |concur(P)|

L1 Pokud |concur(P)| > n, pak announce[P] ∈ head (během výpočtu procesu P)

Dk: ↳ concur(P) obsahuje buňky se

$seq \equiv p-1 \pmod n$ ← buňka q přidaná procesem Q

$seq \equiv p \pmod n$ ← buňka r přidaná procesem R

- q ∈ concur(P) ⇒ Q připojil buňku q poté, co P ohlédl
- R připojil r za q ⇒ v kroku 5 procesu R je head[R] nastaveno na q
- předtím Q nastavil head[Q] ← q ⇒ buňka q je připojena procesem Q

↳ v kroku 5 procesu R (čtení head[R]) je už nastaveno announce[P]

- pak je buď ann[P] už zapojené, nebo r = ann[P]

|concur(P)| := max(head).seq − start(P)

L2 $\max(head)_{.seq} \geq start(P)$

L4 po kroku 3: $head[P]_{.seq} \geq start(P)$

L5 $|concur(P)| \geq head(P).seq - start(P) \geq 0$ (L4)

↳ v každém průchodu cyklem while se dolní odhad na |concur(P)| zvýší aspoň o 1.

⇒ po nejvýše n+1 průchodech platí předp. od (L1)

⇒ cyklus skončí!