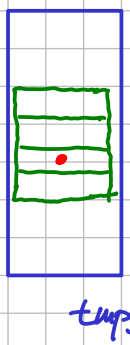
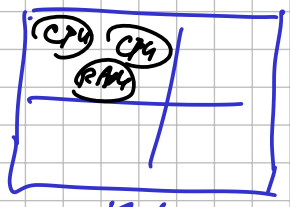
Sdílená paměť mezi procesy
- mmap – na Linuxu koherentní, jinde msync
- SysV IPC
- POSIX IPC shm_... na Linuxu mmap v /dev/shm/

zamykání
- Sloce
- POSIX IPC sem_...



NUMA
- automatika: alokace z aktuálního node
- explicitně: striped allocate
  pinning (via cgroups)
  topologie (via /sys/...)



1	2	3	4	1	2	3	4

Atomické operace

Repertoár: read / write
inc, dec
exchange
compare & exchange

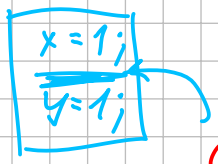
```
atomic { old = *p;
    if (*p == a)
        *p = b;
    return old;
}
```

(test & set)

Zrada: Uvedení uspořádání je lokální pro proměnnou
↳ a co celý stav programu?
↓
Sekvenční konzistence

příklad: atomic int x=0, y=0;

A: int a = load(y); store(x,a);
B: int b = load(x); store(y,42);



z pohledu x: store(x,A); load(x);
z pohledu y: store(y,42); load(y);

chceme bariéry

na konci a=b=42

příklad:

```
A: mutex.lock();
   x=1;
   y=2;
   mutex.unlock();

B: mutex.lock();
   a=x;
   b=y;
   mutex.unlock();
```

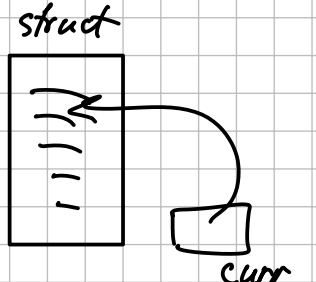
A: mutex.lock() – acquire
A: mutex.unlock() – release
B: mutex.lock() – acquire
B: mutex.unlock()

zápis nelze přesunout za release
čtení nelze přesunout před acquire

příklad:

```
writer: s = malloc(...);
        s->a = ...
        s->b = ...
release [ store(curr, s);

reader: acquire
        [ s = load(curr);
        s->a
        ⋮
```



C11 model: sémantika – seq. consistent, relaxed
operace – acquire / release / acq+rel, consume

C11 atomics

- kvalifikátor `_Atomic`
  - na lib. typ (kromě bit fields)
  - s proměnnou lze — load, store
    - += -= ++ --
    - spec. operace
  - sizeof, alignof se může lišit oproti neatom. verzi
  - atom. struct je nedělitelná

- funkce atomic_X()
  - load / load_explicit
    - sém. seq. consist.
    - sém. zadáme
  - store
  - exchange
  - add, sub
  - compare_exchange — strong / weak
  - thread_fence
  - signal_fence

- <stdatomic.h>
  - typy: atomic_int
  - makra: ATOMIC_INT_LOCK_FREE ...
  - atomic_flag
    - aspoň 1 bit
    - zaručeně lock-free
    - umí test-and-set

- inicializace: není atomická!
  - atomic_init(&var, value);
  - atomic_int x = ATOMIC_VAR_INIT(42);

- formalismus:
  - sequenced-before
  - v atom. var. má své uspoř. operací
  - synchronized-with
  - → happens-before

FUTEXy

- jádro ho vidí na fyz. adresu v paměti
- má hodnotu: int (32)
- má uvnitř jádra frontu čekajících procesů
- operace: futex_wait(&futex, oček. hodnota při volání)

  futex_wake(&futex, # procesů)

Implementace mutexu:

stav = { 0 volný; 1 zamčeno, ∄ další čekající; 2 zamčeno, možná ∃ další čekající }

```
Lock: if ((c = cmpxchg(&futex, 0, 1)) != 0) {
        do {
          if (c == 2 || cmpxchg(&f, 1, 2) != 0)
            futex_wait(&f, 2);
        } while (c = cmpxchg(&f, 0, 2));
      }

Unlock: if (atomic_load_dec(&f) != 1) {
          f = 0;
          futex_wake(&f, 1);
        }
```

Transakční paměť

- softwarová – žurnál + commit lock
- hardware
  - IBM POWER
  - Intel TSX
    - využívá MESI
    - hypotéza: spekulace v L1 a L2, v L3 původní data

Sémantika: START – čtení & zápisy do paměti – COMMIT (nebo ROLLBACK)

XSTART adresa → fail handler
- XABORT důvod
- registry i paměť v původním stavu
- v EAX důvod rollbacku

commit → XEND

HLE (Hardware Lock Elision)

prefixy XACQUIRE a XRELEASE

XACQ lock … XREL unlock