PPaDS MMXXI

Matúš Jókay, C-503, matus.jokay@stuba.sk

uim.fei.stuba.sk/predmet/i-ppds

konzultacie elektronicky

mail, discord

Základné synchronizačné vzory

- Mutex
- Multiplex
- Signalizácia
- Rendezvous
- Bariéra
- Znovupoužiteľná baréra
- Turniket

1. Mutex

- Iba jedno vlákno môže “zamknúť” mutex a pokračovať bez čakania daľej
- Istá forma “tokenu”: ten, kto drží “token”, môže niečo konať, ostatní (žiadajúci o token) musia čakať
- Príklady zo života?

- Semafor(1)

Mutex

shared vars:

arr ← [0, …., 0]

ind ← 0

Thread A

```
while True:
        arr[ind] += 1
        ind += 1
```

Thread B

```
while True:
        arr[ind] += 1
        ind += 1
```

Mutex

shared vars:

arr ← [0, …., 0]

ind ← 0

mutex ← Mutex()

Thread A

```
mutex.lock()
while True:
        arr[ind] += 1
        ind += 1
mutex.unlock()
```

Thread B

```
mutex.lock()
while True:
        arr[ind] += 1
        ind += 1
mutex.unlock()
```

Mutex

BAAAD

shared vars:

arr ← [0, …., 0]

ind ← 0

mutex ← Mutex()

Thread A

```
mutex.lock()
while True:
        arr[ind] += 1
        ind += 1
mutex.unlock()
```

Thread B

```
mutex.lock()
while True:
        arr[ind] += 1
        ind += 1
mutex.unlock()
```

Mutex

GOOOD?

shared vars:

arr ← [0, …., 0]

ind ← 0

mutex ← Mutex()

Thread A

```
while True:
    mutex.lock()
    arr[ind] += 1
    ind += 1
    mutex.unlock()
```

Thread B

```
while True:
    mutex.lock()
    arr[ind] += 1
    ind += 1
    mutex.unlock()
```

Mutex

shared vars:

arr ← [0, …., 0]

ind ← 0

mutex ← Mutex()

Thread A

```
while ind < len(arr):
    mutex.lock()
    arr[ind] += 1
    ind += 1
    mutex.unlock()
```

Thread B

```
while ind < len(arr):
    mutex.lock()
    arr[ind] += 1
    ind += 1
    mutex.unlock()
```

Mutex

BAAAD

shared vars:

arr ← [0, …., 0]

ind ← 0

mutex ← Mutex()

Thread A

```
while ind < len(arr):
    mutex.lock()
    arr[ind] += 1
    ind += 1
    mutex.unlock()
```

Thread B

```
while ind < len(arr):
    mutex.lock()
    arr[ind] += 1
    ind += 1
    mutex.unlock()
```

Mutex

shared vars:

arr ← [0, …., 0]

ind ← 0

mutex ← Mutex()

Thread XYZ

```
while True:
    mutex.lock()
    if ind >= len(arr):
        break
    arr[ind] += 1
    ind += 1
    mutex.unlock()
```

Thread XYZ

```
while ind < len(arr):
    mutex.lock()
    arr[ind] += 1
    ind += 1
    mutex.unlock()
```

Mutex

shared vars:

arr ← [0, …., 0]

ind ← 0

mutex ← Mutex()

Thread XYZ

```
while True:
    mutex.lock()
    if ind >= len(arr):
        break
    arr[ind] += 1
    ind += 1
    mutex.unlock()
```

BAAAD

Mutex

shared vars:

arr ← [0, …., 0]

ind ← 0

mutex ← Mutex()

Thread XYZ

```
while True:
    mutex.lock()
    if ind >= len(arr):
        mutex.unlock()
        break
    arr[ind] += 1
    ind += 1
    mutex.unlock()
```

GOOOD?

Mutex

shared vars:

arr ← [0, …., 0]

ind ← 0

mutex ← Mutex()

Thread XYZ

```
while True:
    mutex.lock()
    if ind >= len(arr):
        mutex.unlock()
        break
    arr[ind] += 1
    ind += 1
    mutex.unlock()
```

GOOOD!

Mutex

shared vars:

arr ← [0, …., 0]

ind ← 0

mutex ← Mutex()

Thread XYZ

```
while True:
    mutex.lock()
    arr[ind] += 1
    ind += 1
    if ind >= len(arr):
        mutex.unlock()
        break
    mutex.unlock()
```

GOOOD?

Mutex

shared vars:

arr ← [0, …., 0]

ind ← 0

mutex ← Mutex()

Thread XYZ

```
while True:
    mutex.lock()
    arr[ind] += 1
    ind += 1
    if ind >= len(arr):
        mutex.unlock()
        break
    mutex.unlock()
```

VEERY BAAAD!

Mutex

shared vars:

arr ← [0, …., 0]

ind ← 0

mutex ← Mutex()

Thread XYZ

```
while True:
    mutex.lock()
    arr[ind] += 1
    ind += 1
    if ind >= len(arr):
        mutex.unlock()
        break
    mutex.unlock()
```

Množstvo synchronizačného kódu oproti originálu…

Mutex

shared vars:

arr ← [0, …., 0]

ind ← 0

mutex ← Mutex()

Thread XYZ

```
while True:
    mutex.lock()
    if ind >= len(arr):
        mutex.unlock()
        break
    arr[ind] += 1
    ind += 1
    mutex.unlock()
```

GOOOD!

Dá sa to lepšie?

Mutex

Per thread var:

tmp

Thread XYZ

```
while True:
    mutex.lock()
    if ind >= len(arr):
        mutex.unlock()
        break
    arr[ind] += 1
    ind += 1
    mutex.unlock()
```

Thread XYZ

```
while True:
    mutex.lock()
    tmp = ind
    ind += 1
    mutex.unlock()

    if tmp >= len(arr):
        break
    arr[tmp] += 1
```

Mutex

V čom je to lepšie?

Per thread var:

tmp

Thread XYZ

```
while True:
    mutex.lock()
    tmp = ind
    ind += 1
    mutex.unlock()

    if tmp >= len(arr):
        break
    arr[tmp] += 1
```

Mutex

V čom je to lepšie?

Per thread var:

tmp

Thread XYZ

```
while True:
    mutex.lock()
    tmp = ind
    ind += 1
    mutex.unlock()

    if tmp >= len(arr):
        break
    arr[tmp] += 1
```

Mutex

V tomto naivnom príklade je čas trvania tejto operácie zanedbateľný

Per thread var:

tmp

Thread XYZ

```
while True:
    mutex.lock()
    tmp = ind
    ind += 1
    mutex.unlock()

    if tmp >= len(arr):
        break
    arr[tmp] += 1
```

Mutex

Koľko operácii treba vykonať?

Per thread var:

tmp

Thread XYZ

```
while True:
    mutex.lock()
    tmp = ind
    ind += 1
    mutex.unlock()

    if tmp >= len(arr):
        break
    arr[tmp] += 1
```

Mutex

Koľko operácii treba vykonať?

Toľko, koľko je prvkov poľa!

Per thread var:

tmp

Thread XYZ

```
while True:
    mutex.lock()
    tmp = ind
    ind += 1
    mutex.unlock()

    if tmp >= len(arr):
        break
    arr[tmp] += 1
```

Mutex

Táto časť aloritmu je paralelizovateľná.

Naopak, časť algoritmu uzavretá v KO je sériová.

Per thread var:

tmp

Thread XYZ

```
while True:
    mutex.lock()
    tmp = ind
    ind += 1
    mutex.unlock()

    if tmp >= len(arr):
        break
    arr[tmp] += 1
```

Mutex

Per thread var:

tmp

Thread XYZ

```
while True:
    mutex.lock()
    if ind >= len(arr):
        mutex.unlock()
        break
    arr[ind] += 1
    ind += 1
    mutex.unlock()
```

Thread XYZ

```
while True:
    mutex.lock()
    tmp = ind
    ind += 1
    mutex.unlock()

    if tmp >= len(arr):
        break
    arr[tmp] += 1
```

Mutex

Per thread var:

tmp

Thread XYZ

```
while True:
    mutex.lock()
    if ind >= len(arr):
        mutex.unlock()
        break
    arr[ind] += 1
    ind += 1
    mutex.unlock()
```

Thread XYZ

```
while True:
    mutex.lock()
    tmp = ind
    ind += 1
    mutex.unlock()

    if tmp >= len(arr):
        break
    arr[tmp] += 1
```

Spracovanie prvkov poľa

- Nech zistenie prvku, ktorý sa má spracovať, trvá zanedbateľný čas
- Nech spracovanie prvku trvá netriviálny čas

- Príklad: regál s krabicami, ktoré treba odniesť

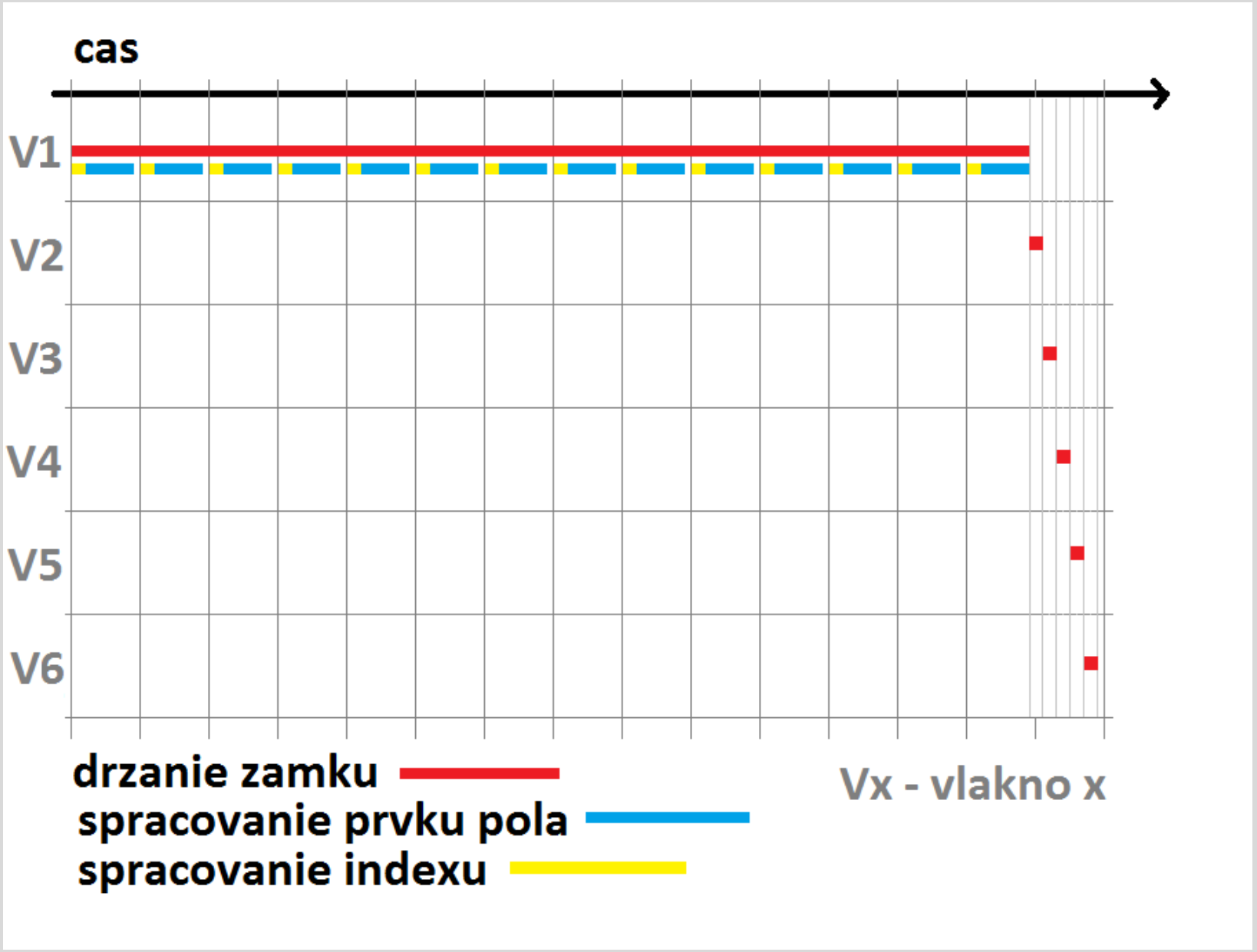
Mutex – Úroveň serializácie

1. Serializácia na úrovni vlákien (spracovania regálu)

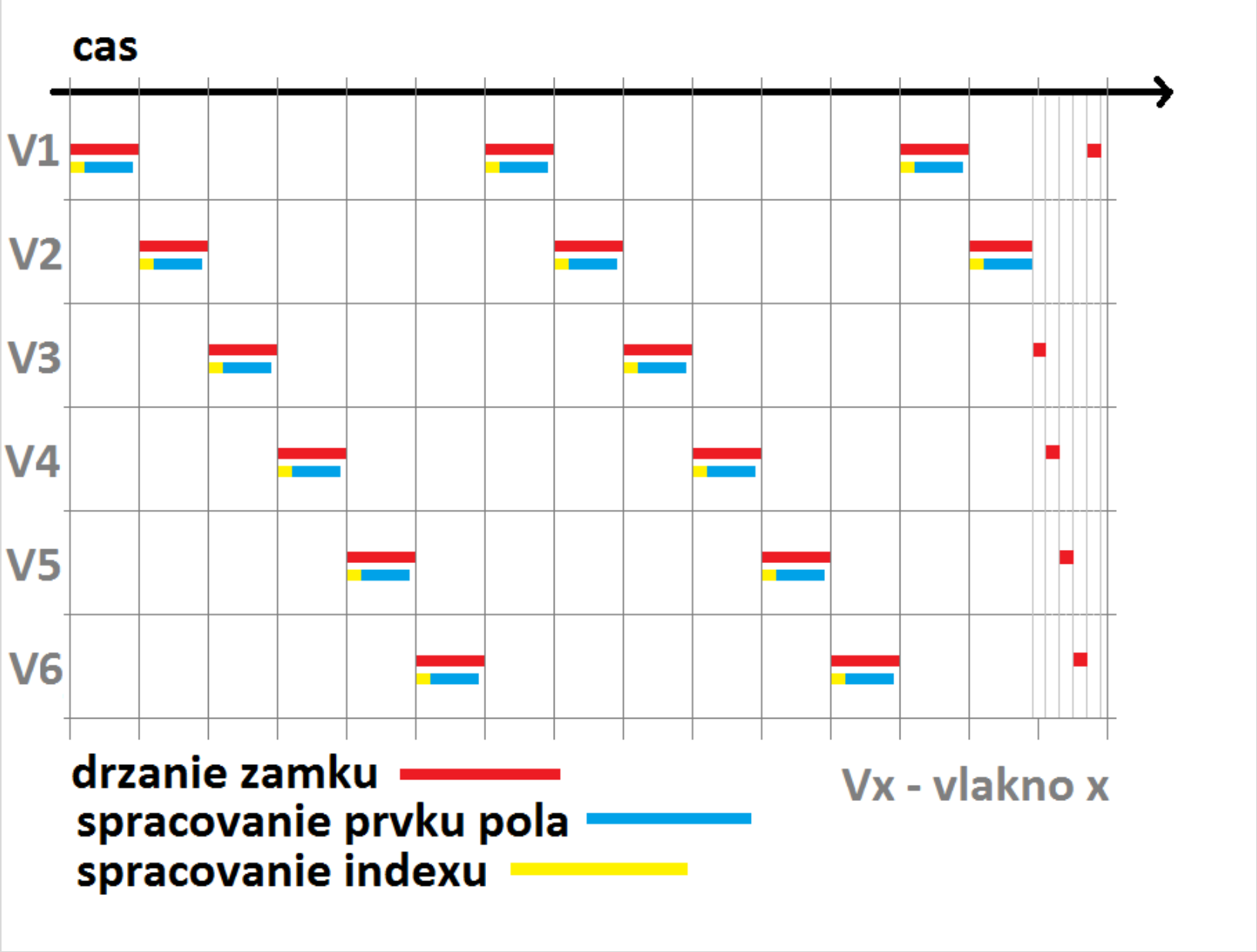
2. Serializácia na úrovni prístupu k prvku poľa (spracovania krabice)

3. Serializácia na úrovni indexu do poľa (identifikácie krabice)

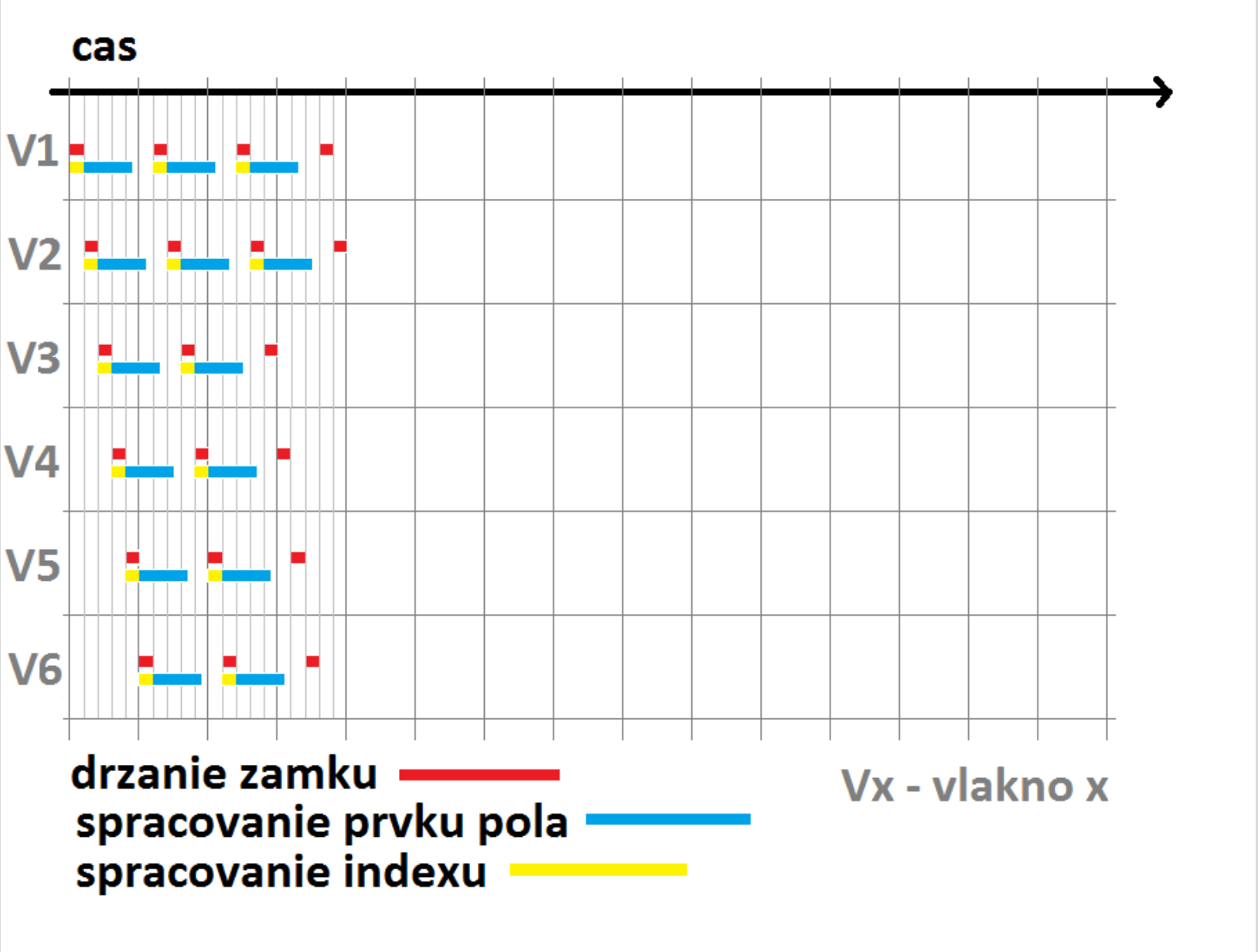
Mutex – Úroveň serializácie 1



Mutex – Úroveň serializácie 2



Mutex – Úroveň serializácie 3



Mutex

- Zachovanie integrity je nutnou podmienkou

- Dosiahnutie konkurencie je želaným stavom; nutná podmienka nestačí!

Mutex

- Zaručuje, že v jednom čase bude daný programový kód vykonávať iba jediný tok riadenia

- Dôsledok: SERIALIZÁCIA!
- Jednotlivé toky riadenia vykonávajú KO postupne jeden po druhom, za sebou
- Mutex úmyselne porušuje konkurentnosť!

2. Multiplex

- Zovšeobecnenie mutexu
- Nie 1, ale max. N vlákien môže vstúpiť do KO
- Napríklad: autobus, kino, sauna…
- Implementácia: multiplex = Semaphore(N)

Použitie: inicializácia

```
mplex := Semaphore(N)
```

Použitie: vo vlákne

```
mplex.wait()
    # kritická oblasť
mplex.signal()
```

3. Signalizácia

- Použitie **medzi 2 vláknami**
- Jedno signalizuje druhému nejakú udalosť
- Príklad:
  - T1 načíta riadok
  - T2 ho zobrazí

Použitie: inicializácia

```
event := Semaphore(0)
```

Použitie vlákno A

```
# ... nejaký kód
event.signal()
# ... nejaký kód
```

Použitie vlákno B

```
# ... nejaký kód
event.wait()
# ... nejaký kód
```

Signalizácia

Kto vyriešil domácu úlohu z prednášky?

4. Rendezvous

- **Obojstranná signalizácia**
- A čaká na B a B čaká na A
- Nemôžu pokračovať, kým sa nestretnú

Použitie: vlákno A	Použitie: vlákno B
1) # … nejaký kód a1	1) # … nejaký kód b1
2) # … nejaký kód a2	2) # … nejaký kód b2

- Požiadavky tejto úlohy:

$$a1 < b2 \quad \&\& \quad b1 < a2$$

Rendezvous

- Správne pomenovanie semaforov reprezentujúcich udalosti!!!
- Meno musí reprezentovať udalosť

Rendezvous

- Správne pomenovanie semaforov reprezentujúcich udalosti!!!
- Meno musí reprezentovať udalosť

Inicializácia

aArrived ← Semaphore(0)

bArrived ← Semaphore(0)

Rendezvous

Inicializácia

aArrived ← Semaphore(0)

bArrived ← Semaphore(0)

Použitie: vlákno A

1) # … nejaký kód a1
2)
3)
4) # … nejaký kód a2

Použitie: vlákno B

1) # … nejaký kód b1
2)
3)
4) # … nejaký kód b2

??? a1 < b2 && b1 < a2 ???

Rendezvous

Inicializácia

aArrived ← Semaphore(0)

bArrived ← Semaphore(0)

Použitie: vlákno A

1) # … nejaký kód a1
2) bArrived.wait()
3) aArrived.signal()
4) # … nejaký kód a2

Použitie: vlákno B

1) # … nejaký kód b1
2) aArrived.wait()
3) bArrived.signal()
4) # … nejaký kód b2

??? a1 < b2 && b1 < a2 ???

Rendezvous

Inicializácia

aArrived ← Semaphore(0)

bArrived ← Semaphore(0)

Použitie: vlákno A

1) # … nejaký kód a1
2) bArrived.wait()
3) aArrived.signal()
4) # … nejaký kód a2

Použitie: vlákno B

1) # … nejaký kód b1
2) aArrived.wait()
3) bArrived.signal()
4) # … nejaký kód b2

??? a1 < b2 && b1 < a2 ???

Rendezvous

Inicializácia

aArrived ← Semaphore(0)

bArrived ← Semaphore(0)

Použitie: vlákno A

1) # … nejaký kód a1
2) bArrived.wait()
3) aArrived.signal()
4) # … nejaký kód a2

Použitie: vlákno B

1) # … nejaký kód b1
2) aArrived.wait()
3) bArrived.signal()
4) # … nejaký kód b2

UVIAZNUTIE

??? a1 < b2 && b1 < a2 ???

Rendezvous

Inicializácia

aArrived ← Semaphore(0)

bArrived ← Semaphore(0)

Použitie: vlákno A

1) # … nejaký kód a1
2) bArrived.wait()
3) aArrived.signal()
4) # … nejaký kód a2

Použitie: vlákno B

1) # … nejaký kód b1
2) aArrived.wait()
3) bArrived.signal()
4) # … nejaký kód b2

??? a1 < b2 && b1 < a2 ???

Rendezvous

Inicializácia

aArrived ← Semaphore(0)

bArrived ← Semaphore(0)

Použitie: vlákno A

1) # … nejaký kód a1
2) bArrived.wait()
3) aArrived.signal()
4) # … nejaký kód a2

Použitie: vlákno B

1) # … nejaký kód b1
2) bArrived.signal()
3) aArrived.wait()
4) # … nejaký kód b2

??? a1 < b2 && b1 < a2 ???

Rendezvous

Inicializácia

aArrived ← Semaphore(0)

bArrived ← Semaphore(0)

Použitie: vlákno A

1) # … nejaký kód a1
2) bArrived.wait()
3) aArrived.signal()
4) # … nejaký kód a2

Použitie: vlákno B

1) # … nejaký kód b1
2) bArrived.signal()
3) aArrived.wait()
4) # … nejaký kód b2

EFEKTIVITA

??? a1 < b2 && b1 < a2 ???

Rendezvous

Inicializácia

aArrived ← Semaphore(0)

bArrived ← Semaphore(0)

Použitie: vlákno A

1) # … nejaký kód a1
2) bArrived.wait()
3) aArrived.signal()
4) # … nejaký kód a2

Použitie: vlákno B

1) # … nejaký kód b1
2) bArrived.signal()
3) aArrived.wait()
4) # … nejaký kód b2

??? a1 < b2 && b1 < a2 ???

Rendezvous

Inicializácia

aArrived ← Semaphore(0)

bArrived ← Semaphore(0)

Použitie: vlákno A

1) # … nejaký kód a1
2) aArrived.signal()
3) bArrived.wait()
4) # … nejaký kód a2

Použitie: vlákno B

1) # … nejaký kód b1
2) bArrived.signal()
3) aArrived.wait()
4) # … nejaký kód b2

??? a1 < b2 && b1 < a2 ???

Rendezvous

Inicializácia

aArrived ← Semaphore(0)

bArrived ← Semaphore(0)

Použitie: vlákno A

1) # … nejaký kód a1
2) aArrived.signal()
3) bArrived.wait()
4) # … nejaký kód a2

Použitie: vlákno B

1) # … nejaký kód b1
2) bArrived.signal()
3) aArrived.wait()
4) # … nejaký kód b2

VERY GOOD

??? a1 < b2 && b1 < a2 ???

5. Bariéra

- Zovšeobecnenie stretnutia z 2 na N vlákien
- Žiadne z vlákien nesmie začať vykonávať KO, pokým všetky vlákna neskončili vykonávanie rendezvous
- Konštanta N je dostupná všetkým vláknam
- Prvých N-1 vlákien sa musí zablokovať, kým nepríde N-té vlákno; až potom môžu pokračovať

Vlákno (kód bariéry):

1) Rendezvous
2) KO

Bariéra

- N je počet vlákien (pre 2 je to klasická úloha Rendezvous)

- Doteraz sme si vystačili s Mutexom / Semaforom
- Pri bariére budeme potrebovať počítadlo
  - Prístup k počítadlu majú všetky vlákna
  - Preto jeho integritu musíme chrániť serializáciou pomocou mutexu

Bariéra

- N je počet vlákien (pre 2 je to klasická úloha Rendezvous)

- Čo budeme potrebovať:
  - Počítadlo vlákien, ktoré prichádzajú k bariére
  - Mutex, ktorým budeme chrániť integritu počítadla
  - Semafor, ktorý bude slúžiť ako bariéra (posledné vlákno, ktoré príde k bariére, tento semafor uvoľní, aby mohli vlákna pokračovať)

Bariéra

Inicializácia:

N – vopred dané a dostupné

count ← 0

mutex ← Semaphore(1)

barrier ← Semaphore(0)

Bariéra

Vlákno XYZ:

1) rendezvous

2) KO

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Semaphore(1)

barrier ← Semaphore(0)

Bariéra

Vlákno XYZ:

1) rendezvous

2) KO

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Semaphore(1)

barrier ← Semaphore(0)

- Pridajme blokovanie vlákien pred vstupom do ‘KO’

Bariéra

Vlákno XYZ:

1) rendezvous

2) barrier.wait()

3) KO

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Semaphore(1)

barrier ← Semaphore(0)

- Pridajme blokovanie vlákien pred vstupom do ‘KO’

Bariéra

Vlákno XYZ:

1) rendezvous

2) barrier.wait()

3) KO

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Semaphore(1)

barrier ← Semaphore(0)

- Veľmi sme si nepomohli, všetky vlákna ostávajú zablokované…

Bariéra

Vlákno XYZ:

1) rendezvous

2) barrier.wait()

3) KO

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Semaphore(1)

barrier ← Semaphore(0)

- Pridajme počítadlo vlákien, ktoré prechádzajú z ‘rendezvous’ do ‘KO’

Bariéra

Vlákno XYZ:

1) rendezvous

2) count += 1

3) barrier.wait()

4) KO

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Semaphore(1)

barrier ← Semaphore(0)

Bariéra

Vlákno XYZ:

1) rendezvous

2) count += 1

3) barrier.wait()

4) KO

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Semaphore(1)

barrier ← Semaphore(0)

- Počítadlo ale NIE je chránené proti súčasnému prístupu vlákien!!!

Bariéra

Vlákno XYZ:

1) rendezvous

2) count += 1

3) barrier.wait()

4) KO

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Semaphore(1)

barrier ← Semaphore(0)

- Pridajme teda mutex, ktorý zaručí atomicitu operácie

Bariéra

Vlákno XYZ:

1) rendezvous
2) mutex.wait()
3) count += 1
4) mutex.signal()

5) barrier.wait()
6) KO

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Semaphore(1)

barrier ← Semaphore(0)

- Počítadlo je chránené proti súčasnému prístupu vlákien
- Čo s tým? Stále máme zablokované vlákna...

Bariéra

Vlákno XYZ:

1) rendezvous
2) mutex.wait()
3) count += 1
4) mutex.signal()
5) barrier.wait()
6) KO

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Semaphore(1)

barrier ← Semaphore(0)

- Posledné (N-té) vlákno uvoľní bariéru!

Bariéra

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Semaphore(1)

barrier ← Semaphore(0)

Vlákno XYZ:

1) rendezvous
2) mutex.wait()
3) count += 1
4) mutex.signal()
5) if count == N: barrier.signal()
6) barrier.wait()
7) KO

Bariéra

Vlákno XYZ:

1) rendezvous
2) mutex.wait()
3) count += 1
4) mutex.signal()
5) if count == N: barrie
6) barrier.wait()
7) KO

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Semaphore(1)

barrier ← Semaphore(0)

- Testovanie premennej mimo uzamknutia
- Problém?

Bariéra

Vlákno XYZ:

1) rendezvous
2) mutex.wait()
3) count += 1
4) mutex.signal()
5) if count == N: barrier.signal()
6) barrier.wait()
7) KO

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Semaphore(1)

barrier ← Semaphore(0)

NIE, ALE...

Bariéra

Vlákno XYZ:

1) rendezvous
2) mutex.wait()
3) count += 1
4) mutex.signal()
5) if count == N: barrier.signal()
6) barrier.wait()
7) KO

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Semaphore(1)

barrier ← Semaphore(0)

PREČO???

Bariéra

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Semaphore(1)

barrier ← Semaphore(0)

Vlákno XYZ:

1) rendezvous
2) mutex.wait()
3) count += 1
4) mutex.signal()
5) if count == N: barrier.signal()
6) barrier.wait()
7) KO

- Uvoľní sa nám aspoň 1 vlákno, ale nie N!
- **Aspoň 1x** sa volá signal() nad bariérou

Bariéra

Vlákno XYZ:

1) rendezvous
2) mutex.wait()
3) count += 1
4) mutex.signal()
5) if count == N: barrier.signal()
6) barrier.wait()
7) KO

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Semaphore(1)

barrier ← Semaphore(0)

- Uvoľní sa nám aspoň 1 vlákno, ale nie N!
- **Aspoň 1x** sa volá signal() nad bariérou

Bariéra

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Semaphore(1)

barrier ← Semaphore(0)

Vlákno XYZ:

1) rendezvous
2) mutex.wait()
3) count += 1
4) mutex.signal()
5) if count == N: barrier.signal()
6) barrier.wait()
7) KO

- Nech každé vlákno po prechode bariérou uvoľní bariéru ďalšiemu vláknu

Bariéra

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Semaphore(1)

barrier ← Semaphore(0)

Vlákno XYZ:

1) rendezvous
2) mutex.wait()
3) count += 1
4) mutex.signal()
5) if count == N: barrier.signal()
6) barrier.wait()
7) barrier.signal()
8) KO

- Nech každé vlákno po prechode bariérou uvoľní bariéru ďalšiemu vláknu

Bariéra

Vlákno XYZ:

1) rendezvous
2) mutex.wait()
3) count += 1
4) mutex.signal()
5) if count == N: barrier.signal()
6) barrier.wait()
7) barrier.signal()
8) KO

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Semaphore(1)

barrier ← Semaphore(0)

- Nech každé vlákno po prechode bariérou uvoľní bariéru ďalšiemu vláknu

Bariéra

Vlákno XYZ:

1) rendezvous
2) mutex.wait()
3) count += 1
4) mutex.signal()
5) if count == N: barri
6) barrier.wait()
7) barrier.signal()
8) KO

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Semaphore(1)

barrier ← Semaphore(0)

- Synchronizačný vzor
- Turniket (turnstile)
- prejde 1 vlákno v 1 čase

Bariéra

Vlákno XYZ:

1) rendezvous
2) mutex.wait()
3) count += 1
4) mutex.signal()
5) if count == N: barrier.signal()
6) barrier.wait()
7) barrier.signal()
8) KO

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Semaphore(1)

barrier ← Semaphore(0)

- Bude nasledovať ukážka veľmi častej chyby

Bariéra

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Semaphore(1)

barrier ← Semaphore(0)

Vlákno XYZ:

1) rendezvous
2) mutex.wait()
3) count += 1
4) if count == N: barrier.signal()
5) barrier.wait()
6) barrier.signal()
7) mutex.signal()
8) KO

- O akú chybu sa jedná?

Bariéra

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Semaphore(1)

barrier ← Semaphore(0)

Vlákno XYZ:

1) rendezvous
2) mutex.wait()
3) count += 1
4) if count == N: barrier.signal()
5) barrier.wait()
6) barrier.signal()
7) mutex.signal()
8) KO

- Častá chyba!!!
- Blokovanie na semafore počas držania zámku!

Bariéra

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Semaphore(1)

barrier ← Semaphore(0)

Vlákno XYZ:

1) rendezvous
2) mutex.wait()
3) count += 1
4) mutex.signal()
5) if count == N: barrier.signal()
6) barrier.wait()
7) barrier.signal()
8) KO

- Takže zatiaľ finálna verzia jedenkrát použiteľnej bariéry

Bariéra

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Semaphore(1)

barrier ← Semaphore(0)

Vlákno XYZ:

1) rendezvous
2) mutex.wait()
3) count += 1
4) mutex.signal()
5) if count == N: barrier.signal()
6) barrier.wait()
7) barrier.signal()
8) KO

- Koľkokrát sa vyvola metóda signal()?
- **Minimálne** N-krát!

6. Znovupoužiteľná bariéra

- Každé vlákno vykonáva nejaký algoritmus v cykle
- Chceme, aby sa napríklad všetky vlákna počkali vždy PRED každou iteráciou algoritmu

Vlákno XYZ:

1) while True:
2) rendezvous
3) KO

Vlákno XYZ:

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)      mutex.unlock()
6)      if count == N: turnstile.signal()
7)      turnstile.wait()
8)      turnstile.signal()
9)      KO
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile ← Semaphore(0)

- Rozšírme teda naše riešenie bariéry
- Dajme ho do cyklu

Vlákno XYZ:

```
1) while True:
2)     rendezvous
3)     mutex.lock()
4)         count += 1
5)     mutex.unlock()
6)     if count == N: turnstile.signal()
7)     turnstile.wait()
8)     turnstile.signal()
9)     KO
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile ← Semaphore(0)

GOOOD?

Vlákno XYZ:

```
1) while True:
2)     rendezvous
3)     mutex.lock()
4)         count += 1
5)     mutex.unlock()
6)     if count == N: turnstile.signal()
7)     turnstile.wait()
8)     turnstile.signal()
9)     KO
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile ← Semaphore(0)

BAAAD!!!

Vlákno XYZ:

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)      mutex.unlock()
6)      if count == N: turnstile.signal()
7)      turnstile.wait()
8)      turnstile.signal()
9)      KO
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile ← Semaphore(0)

BAAAD!!!

Vlákno XYZ:

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)      mutex.unlock()
6)      if count == N: turnstile.signal()
7)      turnstile.wait()
8)      turnstile.signal()
9)      KO
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile ← Semaphore(0)

- Nevieme, aká je hodnota turnstile na konci cyklu
- Môže byť <1; N>
- Skúsme chybu opraviť

Vlákno XYZ:

```
1) while True:
2)     rendezvous
3)     mutex.lock()
4)         count += 1
5)         if count == N: turnstile.signal()
6)     mutex.unlock()
7)     turnstile.wait()
8)     turnstile.signal()
9)     KO
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile ← Semaphore(0)

- Teraz vieme **isto**, že hodnota turnstile bude po prechode všetkých vlákien na konci cyklu 1

Vlákno XYZ:

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N: turnstile.signal()
6)      mutex.unlock()
7)      turnstile.wait()
8)      turnstile.signal()
9)      KO
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile ← Semaphore(0)

GOOOD!

Vlákno XYZ:

```
1) while True:
2)     rendezvous
3)     mutex.lock()
4)         count += 1
5)         if count == N: turnstile.signal()
6)     mutex.unlock()
7)     turnstile.wait()
8)     turnstile.signal()
9)     KO
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile ← Semaphore(0)

- Ako sa správajú vlákna v cykle?

Vlákno XYZ:

```
1) while True:
2)     rendezvous
3)     mutex.lock()
4)         count += 1
5)         if count == N: turnstile.signal()
6)     mutex.unlock()
7)     turnstile.wait()
8)     turnstile.signal()
9)     KO
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile ← Semaphore(0)

- Máme N volaní wait()
- Máme N+1 volaní signal()

Vlákno XYZ:

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N: turnstile.signal()
6)      mutex.unlock()
7)      turnstile.wait()
8)      turnstile.signal()
9)      KO
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile ← Semaphore(0)

- Ako to opravíme?

Vlákno XYZ:

```
1) while True:
2)     rendezvous
3)     mutex.lock()
4)         count += 1
5)         if count == N: turnstile.signal()
6)     mutex.unlock()
7)     turnstile.wait()
8)     turnstile.signal()
9)     KO
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile ← Semaphore(0)

- Musí nám sedieť počet volaní signal() a wait()
- Jedno vlákno nech robí wait() navyše
- Podobne ako jedno robí signal() navyše

Vlákno XYZ:

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N: turnstile.signal()
6)      mutex.unlock()
7)      turnstile.wait()
8)      turnstile.signal()
9)      KO
10)     if count == N: turnstile.wait()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile ← Semaphore(0)

Vlákno XYZ:

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N: turnstile.signal()
6)      mutex.unlock()
7)      turnstile.wait()
8)      turnstile.signal()
9)      KO
10)     if count == N: turnstile.wait()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile ← Semaphore(0)

- Aspoň jedno vlákno robí wait()
- Ale potrebujeme práve jedno vlakno!

Vlákno XYZ:

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N: turnstile.signal()
6)      mutex.unlock()
7)      turnstile.wait()
8)      turnstile.signal()
9)      KO
10)     if count == N: turnstile.wait()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile ← Semaphore(0)

- Ako to opravíme?

Vlákno XYZ:

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N: turnstile.signal()
6)      mutex.unlock()
7)      turnstile.wait()
8)      turnstile.signal()
9)      KO
10)     mutex.lock()
11)         if count == N: turnstile.wait()
12)     mutex.unlock()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile ← Semaphore(0)

Vlákno XYZ:

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N: turnstile.signal()
6)      mutex.unlock()
7)      turnstile.wait()
8)      turnstile.signal()
9)      KO
10)     mutex.lock()
11)         if count == N: turnstile.wait()
12)     mutex.unlock()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile ← Semaphore(0)

PARÁDA!

Vlákno XYZ:

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N: turnstile.signal()
6)      mutex.unlock()
7)      turnstile.wait()
8)      turnstile.signal()
9)      KO
10)     mutex.lock()
11)         if count == N: turnstile.wait()
12)     mutex.unlock()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile ← Semaphore(0)

HMM....

Vlákno XYZ:

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N: turnstile.signal()
6)      mutex.unlock()
7)      turnstile.wait()
8)      turnstile.signal()
9)      KO
10)     mutex.lock()
11)         if count == N: turnstile.wait()
12)     mutex.unlock()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile ← Semaphore(0)

- Funguje nám bariéra?

Vlákno XYZ:

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N: turnstile.signal()
6)      mutex.unlock()
7)      turnstile.wait()
8)      turnstile.signal()
9)      KO
10)     mutex.lock()
11)         if count == N: turnstile.wait()
12)     mutex.unlock()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile ← Semaphore(0)

- Koľkokrát?

Vlákno XYZ:

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N: turn
6)      mutex.unlock()
7)      turnstile.w
8)      turns
9)
10)         tex.lock()
11)         if count == N: turnstile.wait()
12)     mutex.unlock()
```

Inicializácia:

N – dané a dostu

co

mute

turnstile e(0)

OPRAVTE

https://forms.gle/vvne7V9gCUsvaqSEA

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N: turnstile.signal()
6)      mutex.unlock()
7)      turnstile.wait()
8)      turnstile.signal()
9)      KO
10)     mutex.lock()
11)         if count == N: turnstile.wait()


12)     mutex.unlock()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile ← Semaphore(0)

- Opravme počítadlo

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N: turnstile.signal()
6)      mutex.unlock()
7)      turnstile.wait()
8)      turnstile.signal()
9)      KO
10)     mutex.lock()
11)         if count == N: turnstile.wait()
12)         count -= 1
13)     mutex.unlock()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile ← Semaphore(0)

- Opravme počítadlo

Inicializácia:
N – dané a dostupné
count ← 0
mutex ← Mutex()
turnstile ← Semaphore(0)

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N: turnstile.signal()
6)      mutex.unlock()
7)      turnstile.wait()
8)      turnstile.signal()
9)      KO
10)     mutex.lock()
11)         if count == N: turnstile.wait()
12)         count -= 1
13)     mutex.unlock()
```

STALE ZLE!

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N: turnstile.signal()
6)      mutex.unlock()
7)      turnstile.wait()
8)      turnstile.signal()
9)      KO
10)     mutex.lock()
11)         if count == N: turnstile.wait()
12)         count -= 1
13)     mutex.unlock()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile ← Semaphore(0)

- Kód beží v cykle!
- Po vykonaní posledného riadku každé vlákno ide znovu na rande!

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N: turnstile.signal()
6)      mutex.unlock()
7)      turnstile.wait()
8)      turnstile.signal()
9)      KO
10)     mutex.lock()
11)         if count == N: turnstile.wait()
12)         count -= 1
13)     mutex.unlock()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile ← Semaphore(0)

- Zatiaľ máme 1 turniket
  - Potrebujeme ale 2!
- Prvý máme medzi R a KO
- Druhý dáme **napríklad** medzi KO a R

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N: turnstile1.signal()
6)      mutex.unlock()
7)      turnstile1.wait()
8)      turnstile1.signal()
9)      KO
10)     mutex.lock()
11)         if count == N: turnstile1.wait()
12)         count -= 1
13)     mutex.unlock()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

- Premenujeme prvý turniket

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N: turnstile1.signal()
6)      mutex.unlock()
7)      turnstile1.wait()
8)      turnstile1.signal()
9)      KO
10)     mutex.lock()
11)         if count == N: turnstile1.wait()
12)         count -= 1
13)     mutex.unlock()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

turnstile2 ← Semaphore(0)

- Pridáme druhý turniket
  - turnstile2

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N: turnstile1.signal()
6)      mutex.unlock()
7)      turnstile1.wait()
8)      turnstile1.signal()
9)      KO
10)     mutex.lock()
11)         if count == N: turnstile1.wait()
12)         count -= 1
13)     mutex.unlock()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

turnstile2 ← Semaphore(0)

- Ako medzi R a KO

1. Všetky vlákna čakajú za KO
2. Posledné vlákno spúšťa turniket

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N: turnstile1.signal()
6)      mutex.unlock()
7)      turnstile1.wait()
8)      turnstile1.signal()
9)      KO
10)     mutex.lock()
11)         if count == N: turnstile1.wait()
12)         count -= 1
13)     mutex.unlock()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

turnstile2 ← Semaphore(0)

- Všetky vlákna čakajú za KO

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N: turnstile1.signal()
6)      mutex.unlock()
7)      turnstile1.wait()
8)      turnstile1.signal()
9)      KO
10)     mutex.lock()
11)         if count == N: turnstile1.wait()
12)         count -= 1
13)     mutex.unlock()
14)     turnstile2.wait()
15)     turnstile2.signal()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

turnstile2 ← Semaphore(0)

- Všetky vlákna čakajú za KO

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N: turnstile1.signal()
6)      mutex.unlock()
7)      turnstile1.wait()
8)      turnstile1.signal()
9)      KO
10)     mutex.lock()
11)         if count == N: turnstile1.wait()
12)         count -= 1

13)     mutex.unlock()
14)     turnstile2.wait()
15)     turnstile2.signal()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

turnstile2 ← Semaphore(0)

- Posledné vlákno ho otvára

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N: turnstile1.signal()
6)      mutex.unlock()
7)      turnstile1.wait()
8)      turnstile1.signal()
9)      KO
10)     mutex.lock()
11)         if count == N: turnstile1.wait()
12)         count -= 1
13)         if count == 0: turnstile2.signal()
14)     mutex.unlock()
15)     turnstile2.wait()
16)     turnstile2.signal()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

turnstile2 ← Semaphore(0)

- Posledné vlákno ho otvára

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N: turnstile1.signal()
6)      mutex.unlock()
7)      turnstile1.wait()
8)      turnstile1.signal()
9)      KO
10)     mutex.lock()
11)       if count == N: turnstile1.wait()
12)       count -= 1
13)       if count == 0: turnstile2.signal()
14)     mutex.unlock()
15)     turnstile2.wait()
16)     turnstile2.signal()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

turnstile2 ← Semaphore(0)

- Musíme mať dva rôzne testy?

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N: turnstile1.signal()
6)      mutex.unlock()
7)      turnstile1.wait()
8)      turnstile1.signal()
9)      KO
10)     mutex.lock()
11)         if count == N: turnstile1.wait()
12)         count -= 1
13)         if count == 0: turnstile2.signal()
14)     mutex.unlock()
15)     turnstile2.wait()
16)     turnstile2.signal()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

turnstile2 ← Semaphore(0)

- Teraz robí wait() prvé vlákno, ktoré sa sem dostane
- Môže ho vykonať aj posledné vlákno?

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N: turnstile1.signal()
6)      mutex.unlock()
7)      turnstile1.wait()
8)      turnstile1.signal()
9)      KO
10)     mutex.lock()
11)         count -= 1
12)         if count == 0:
13)             turnstile1.wait()
14)             turnstile2.signal()
15)     mutex.unlock()
16)     turnstile2.wait()
17)     turnstile2.signal()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

turnstile2 ← Semaphore(0)

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N: turnstile1.signal()
6)      mutex.unlock()
7)      turnstile1.wait()
8)      turnstile1.signal()
9)      KO
10)     mutex.lock()
11)         count -= 1
12)         if count == 0:
13)             turnstile1.wait()
14)             turnstile2.signal()
15)     mutex.unlock()
16)     turnstile2.wait()
17)     turnstile2.signal()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

turnstile2 ← Semaphore(0)

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N: turnstile1.signal()
6)      mutex.unlock()
7)      turnstile1.wait()
8)      turnstile1.signal()
9)      KO
10)     mutex.lock()
11)         count -= 1
12)         if count == 0:
13)             turnstile1.wait()
14)             turnstile2.signal()
15)     mutex.unlock()
16)     turnstile2.wait()
17)     turnstile2.signal()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

turnstile2 ← Semaphore(0)

- Záleží na poradí wait() a signal() rôznych turniketov?

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N: turnstile1.signal()
6)      mutex.unlock()
7)      turnstile1.wait()
8)      turnstile1.signal()
9)      KO
10)     mutex.lock()
11)         count -= 1
12)         if count == 0:
13)             turnstile2.signal()
14)         turnstile1.wait()
15)     mutex.unlock()
16)     turnstile2.wait()
17)     turnstile2.signal()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

turnstile2 ← Semaphore(0)

- Záleží na poradí wait() a signal() rôznych turniketov?
- Čo keby sme to urobili takto?

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N: turnstile1.signal()
6)      mutex.unlock()
7)      turnstile1.wait()
8)      turnstile1.signal()
9)      KO
10)     mutex.lock()
11)         count -= 1
12)         if count == 0:
13)             turnstile2.signal()
14)             turnstile1.wait()
15)     mutex.unlock()
16)     turnstile2.wait()
17)     turnstile2.signal()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

turnstile2 ← Semaphore(0)

- Čo keby sme nepoužívali iba jeden mutex, ale dva?
- Pre každý turniket zvlášť?

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N: turnstile1.signal()
6)      mutex.unlock()
7)      turnstile1.wait()
8)      turnstile1.signal()
9)      KO
10)     mutex.lock()
11)         count -= 1
12)         if count == 0:
13)             turnstile1.wait()
14)             turnstile2.signal()
15)     mutex.unlock()
16)     turnstile2.wait()
17)     turnstile2.signal()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

turnstile2 ← Semaphore(0)

- A čo takto?
- Je to v pohode aj pri použití vlastného mutexu pre každý turniket?

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N: turnstile1.signal()
6)      mutex.unlock()
7)      turnstile1.wait()
8)      turnstile1.signal()
9)      KO
10)     mutex.lock()
11)         count -= 1
12)         if count == 0:
13)             turnstile1.wait()
14)             turnstile2.signal()
15)     mutex.unlock()
16)     turnstile2.wait()
17)     turnstile2.signal()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

turnstile2 ← Semaphore(0)

- A čo takto?
- Je to v pohode aj pri použití vlastného mutexu pre každý turniket?
- Čo počítadlo? Stále pohodička?

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N: turnstile1.signal()
6)      mutex.unlock()
7)      turnstile1.wait()
8)      turnstile1.signal()
9)      KO
10)     mutex.lock()
11)         count -= 1
12)         if count == 0:
13)             turnstile1.wait()
14)             turnstile2.signal()
15)     mutex.unlock()
16)     turnstile2.wait()
17)     turnstile2.signal()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

turnstile2 ← Semaphore(0)

- Toto riešenie vyžaduje symetriu!

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N: turnstile1.signal()
6)      mutex.unlock()
7)      turnstile1.wait()
8)      turnstile1.signal()
9)      KO
10)     mutex.lock()
11)         count -= 1
12)         if count == 0:
13)             turnstile1.wait()
14)             turnstile2.signal()
15)     mutex.unlock()
16)     turnstile2.wait()
17)     turnstile2.signal()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

turnstile2 ← Semaphore(0)

- Toto riešenie vyžaduje symetriu!
- Znovu máme problém, že #signal() a #wait() pri turnstile2 sa nezhodujú!

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N:

6)              turnstile1.signal()
7)      mutex.unlock()
8)      turnstile1.wait()
9)      turnstile1.signal()
10)     KO
11)     mutex.lock()
12)         count -= 1
13)         if count == 0:
14)             turnstile1.wait()
15)             turnstile2.signal()
16)     mutex.unlock()
17)     turnstile2.wait()
18)     turnstile2.signal()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

turnstile2 ← Semaphore(0)

- Toto riešenie vyžaduje symetriu!
- Znovu máme problém, že #signal() a #wait() pri turnstile2 sa nezhodujú!

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N:
6)              turnstile2.wait()
7)              turnstile1.signal()
8)      mutex.unlock()
9)      turnstile1.wait()
10)     turnstile1.signal()
11)     KO
12)     mutex.lock()
13)         count -= 1
14)         if count == 0:
15)             turnstile1.wait()
16)             turnstile2.signal()
17)     mutex.unlock()
18)     turnstile2.wait()
19)     turnstile2.signal()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

turnstile2 ← Semaphore(0)

- Pridaním wait() v jednom vlákne sa počet volaní signal() vyrovná

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N:
6)              turnstile2.wait()
7)              turnstile1.signal()
8)      mutex.unlock()
9)      turnstile1.wait()
10)     turnstile1.signal()
11)     KO
12)     mutex.lock()
13)         count -= 1
14)         if count == 0:
15)             turnstile1.wait()
16)             turnstile2.signal()
17)     mutex.unlock()
18)     turnstile2.wait()
19)     turnstile2.signal()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

turnstile2 ← Semaphore(0)

- Tešili sme sa, ze už to máme…
- Blížime sa ku cifre 2^7-1 obrázkov prezentácie…

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N:
6)              turnstile2.wait()
7)              turnstile1.signal()
8)      mutex.unlock()
9)      turnstile1.wait()
10)     turnstile1.signal()
11)     KO
12)     mutex.lock()
13)         count -= 1
14)         if count == 0:
15)             turnstile1.wait()
16)             turnstile2.signal()
17)     mutex.unlock()
18)     turnstile2.wait()
19)     turnstile2.signal()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

turnstile2 ← Semaphore(0)

- Tešili sme sa, ze už to máme…
- Blížime sa ku cifre 2^7-1 obrázkov prezentácie…

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N:
6)              turnstile2.wait()
7)              turnstile1.signal()
8)      mutex.unlock()
9)      turnstile1.wait()
10)     turnstile1.signal()
11)     KO
12)     mutex.lock()
13)         count -= 1
14)         if count == 0:
15)             turnstile1.wait()
16)             turnstile2.signal()
17)     mutex.unlock()
18)     turnstile2.wait()
19)     turnstile2.signal()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

turnstile2 ← Semaphore(0)

- Pridali sme 1 wait() do riešenia...

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N:
6)              turnstile2.wait()
7)              turnstile1.signal()
8)      mutex.unlock()
9)      turnstile1.wait()
10)     turnstile1.signal()
11)     KO
12)     mutex.lock()
13)         count -= 1
14)         if count == 0:
15)             turnstile1.wait()
16)             turnstile2.signal()
17)     mutex.unlock()
18)     turnstile2.wait()
19)     turnstile2.signal()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

turnstile2 ← Semaphore(0)

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N:
6)              turnstile2.wait()
7)              turnstile1.signal()
8)      mutex.unlock()
9)      turnstile1.wait()
10)     turnstile1.signal()
11)     KO
12)     mutex.lock()
13)         count -= 1
14)         if count == 0:
15)             turnstile1.wait()
16)             turnstile2.signal()
17)     mutex.unlock()
18)     turnstile2.wait()
19)     turnstile2.signal()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

turnstile2 ← Semaphore(0)

- Ako je inicializovaný turnstile2?

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N:
6)              turnstile2.wait()
7)              turnstile1.signal()
8)      mutex.unlock()
9)      turnstile1.wait()
10)     turnstile1.signal()
11)     KO
12)     mutex.lock()
13)         count -= 1
14)         if count == 0:
15)             turnstile1.wait()
16)             turnstile2.signal()
17)     mutex.unlock()
18)     turnstile2.wait()
19)     turnstile2.signal()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

turnstile2 ← Semaphore(0)

- Ako je inicializovaný turnstile2?

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N:
6)              turnstile2.wait()
7)              turnstile1.signal()
8)      mutex.unlock()
9)      turnstile1.wait()
10)     turnstile1.signal()
11)     KO
12)     mutex.lock()
13)         count -= 1
14)         if count == 0:
15)             turnstile1.wait()
16)             turnstile2.signal()
17)     mutex.unlock()
18)     turnstile2.wait()
19)     turnstile2.signal()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

turnstile2 ← Semaphore(1)

- Oprava...
- Môže to tak byť?

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N:
6)              turnstile2.wait()
7)              turnstile1.signal()
8)      mutex.unlock()
9)      turnstile1.wait()
10)     turnstile1.signal()
11)     KO
12)     mutex.lock()
13)         count -= 1
14)         if count == 0:
15)             turnstile1.wait()
16)             turnstile2.signal()
17)     mutex.unlock()
18)     turnstile2.wait()
19)     turnstile2.signal()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

turnstile2 ← Semaphore(1)

Dvojfázová bariéra

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N:
6)              turnstile2.wait()
7)              turnstile1.signal()
8)      mutex.unlock()
9)      turnstile1.wait()
10)     turnstile1.signal()
11)     KO
12)     mutex.lock()
13)         count -= 1
14)         if count == 0:
15)             turnstile1.wait()
16)             turnstile2.signal()
17)     mutex.unlock()
18)     turnstile2.wait()
19)     turnstile2.signal()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

turnstile2 ← Semaphore(1)

- Iba n-té vlákno môže zamykať/odomykať turnikety
- Pred odomknutím vlákno vždy zamkne druhý turniket, takže nejaké vlákno môže predbehnúť iné iba po ďalší turniket!

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N:
6)              turnstile2.wait()
7)              turnstile1.signal()
8)      mutex.unlock()
9)      turnstile1.wait()
10)     turnstile1.signal()
11)     KO
12)     mutex.lock()
13)         count -= 1
14)         if count == 0:
15)             turnstile1.wait()
16)             turnstile2.signal()
17)     mutex.unlock()
18)     turnstile2.wait()
19)     turnstile2.signal()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

turnstile2 ← Semaphore(1)

- Kód bariéry sa dá zjednodušiť
- Ak máme možnosť zvýšiť hodnotu semaforu nie iba o 1, ale o N

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N:

6)              turnstile1.signal(N)
7)      mutex.unlock()
8)      turnstile1.wait()

9)      KO
10)     mutex.lock()
11)         count -= 1
12)         if count == 0:

13)             turnstile2.signal(N)
14)     mutex.unlock()
15)     turnstile2.wait()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

turnstile2 ← Semaphore(1)

- Kód bariéry sa dá zjednodušiť
- Ak máme možnosť zvýšiť hodnotu semaforu nie iba o 1, ale o N (a neviem, prečo by sme túto možnosť napríklad nemali)

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N:

6)              turnstile1.signal(N)
7)      mutex.unlock()
8)      turnstile1.wait()

9)      KO
10)     mutex.lock()
11)         count -= 1
12)         if count == 0:

13)             turnstile2.signal(N)
14)     mutex.unlock()
15)     turnstile2.wait()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

turnstile2 ← Semaphore(1)

- Všimnime si riadky 3-8 a 10-15
- Vzhľadom na turniket sú úplne nezávislé!
- Túto skutočnosť o chvíľku využijeme...

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N:
6)              turnstile1.signal(N)
7)      mutex.unlock()
8)      turnstile1.wait()
9)      KO
10)     mutex.lock()
11)         count -= 1
12)         if count == 0:
13)             turnstile2.signal(N)
14)     mutex.unlock()
15)     turnstile2.wait()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

turnstile2 ← Semaphore(1)

- Zvýšením hodnoty semaforu o N miesto o 1 ušetríme 4 riadky kódu

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N:
6)              turnstile1.signal(N)
7)      mutex.unlock()
8)      turnstile1.wait()
9)      KO
10)     mutex.lock()
11)         count -= 1
12)         if count == 0:
13)             turnstile2.signal(N)
14)     mutex.unlock()
15)     turnstile2.wait()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

turnstile2 ← Semaphore(1)

ŠETRÍME ZLE!

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N:
6)              turnstile1.signal(N)
7)      mutex.unlock()
8)      turnstile1.wait()
9)      KO
10)     mutex.lock()
11)         count -= 1
12)         if count == 0:
13)             turnstile2.signal(N)
14)     mutex.unlock()
15)     turnstile2.wait()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

turnstile2 ← Semaphore(1)

- Každá zmena kódu vyžaduje MAXIMÁLNE sústredenie a pozornosť!
- Po zmene kódu kontrola inicializácie a opačne!

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N:
6)              turnstile1.signal(N)
7)      mutex.unlock()
8)      turnstile1.wait()
9)      KO
10)     mutex.lock()
11)         count -= 1
12)         if count == 0:
13)             turnstile2.signal(N)
14)     mutex.unlock()
15)     turnstile2.wait()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

turnstile2 ← Semaphore(0)

- Každá zmena kódu vyžaduje MAXIMÁLNE sústredenie a pozornosť!
- Po zmene kódu kontrola inicializácie a opačne!

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N:
6)              turnstile1.signal(N)
7)      mutex.unlock()
8)      turnstile1.wait()
9)      KO
10)     mutex.lock()
11)         count -= 1
12)         if count == 0:
13)             turnstile2.signal(N)
14)     mutex.unlock()
15)     turnstile2.wait()
```

Inicializácia:

N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

turnstile2 ← Semaphore(0)

```
1)  while True:
2)      rendezvous
3)      mutex.lock()
4)          count += 1
5)          if count == N:
6)              turnstile1.signal(N)
7)      mutex.unlock()
8)      turnstile1.wait()
9)      KO
10)     mutex.lock()
11)         count -= 1
12)         if count == 0:
13)             turnstile2.signal(N)
14)     mutex.unlock()
15)     turnstile2.wait()
```

Inicializácia:

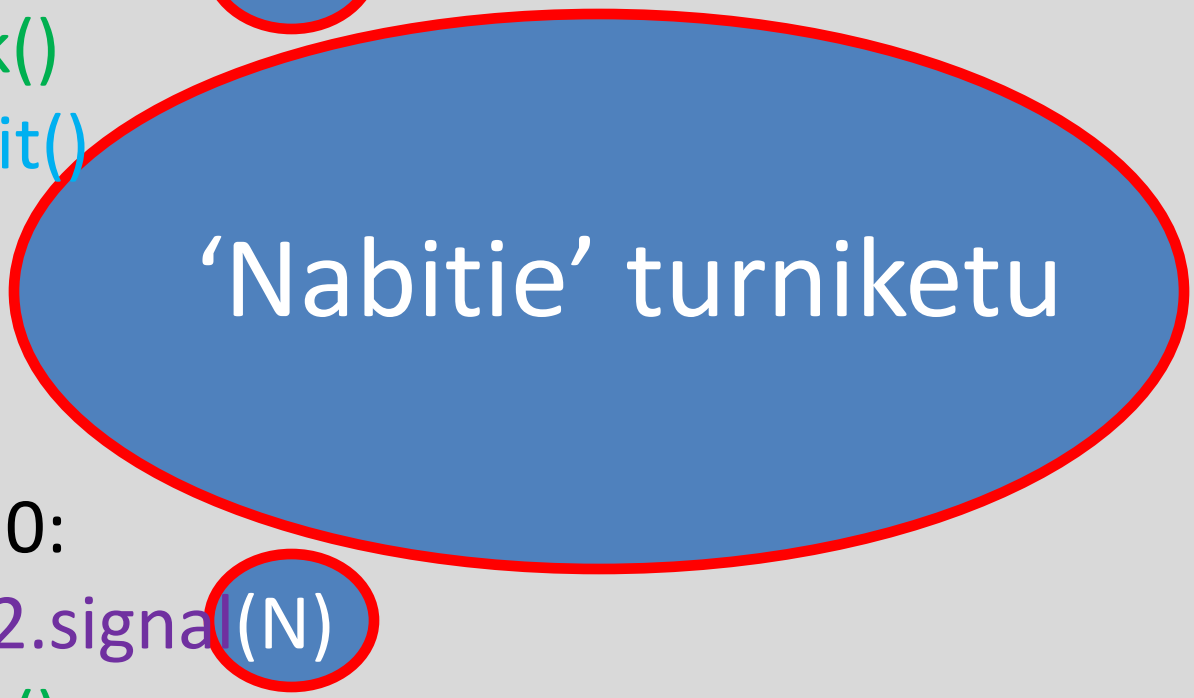
N – dané a dostupné

count ← 0

mutex ← Mutex()

turnstile1 ← Semaphore(0)

turnstile2 ← Semaphore(0)



7. ADT SimpleBarrier

- Na začiatku turniket zablokovaný
- N-té vlákno turniket odblokuje
- N vlákien môže prejsť turniketom

- Stav turniketu:
  - počet prechádzajúcich N, **dané pri vytváraní** ADT
  - mutex M, počítadlo C
  - Semafor turniketu T

Init(N):

C ← 0

M ← Mutex()

T ← Semaphore(0)

```
1)  M.lock()
2)      C += 1
3)      if C == N:
4)          C = 0
5)          T.signal(N)
6)  M.unlock()
7)  T.wait()
```

Init(N):

C ← 0

M ← Mutex()

T ← Semaphore(0)

```
1)  M.lock()
2)      C += 1
3)      if C == N:
4)          C = 0
5)          T.signal(N)
6)  M.unlock()
7)  T.wait()
```

Init(N):

C ← 0

M ← Mutex()

T ← Semaphore(0)

1) M.lock()
2) C += 1
3) if C == N:
4) C = 0
5) T.signal(N)
6) M.unlock()
7) T.wait()

- Ak by sme nevynulovali počítadlo, znovu upadneme do hroznej biedY!
- Ale iba v prípade, ak chceme znovu použiť tú istú inštanciu ADT

Znovu použiteľná bariéra s ADT SimpleBarrier

1) while True:
2) rendezvous
3) barrier1()
4) KO
5) barrier2()

Init(N):

barrier1 ← SimpleBarrier(N)

barrier2 ← SimpleBarrier(N)

Cvičenie

- Implementácia rôznych vecí z prednášky:
  - Jednoduchá bariéra
  - Znovu použiteľná bariéra
- Využitie nasledovných synchronizačných vzorov v príkladoch:
  - Signalizácia, Rendezvous, (opakovaná) Bariera

? Nejaké otázočkY ?

Ďakujem za pozornosť!