

Operacijski sistemi



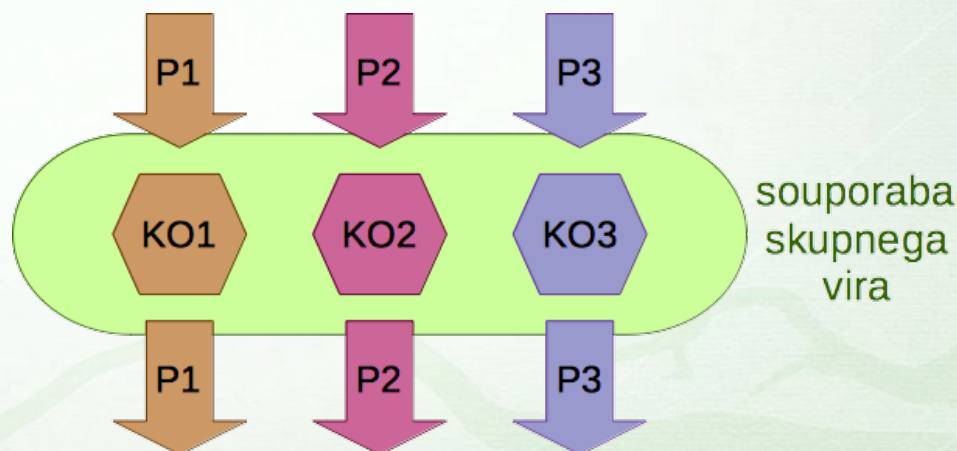
Vzajemno izključevanje

Vsebina

- Ključavnica
 - ključavnica, atomičnost
- Strojna izvedba
 - onemogočanje prekinitev, atomični strojni ukazi
- Programska izvedba
 - Dekkerjev algoritem, Petersonov algoritem
 - Lamportov pekarski algoritem
- Učinkovitost ključavnic

Ključavnica

- Ključavnica
 - osnovni mehanizem za zagotavljanje **vzajemnega izključevanja**
 - vzajemno izključevanje
 - v kritičnem odseku se sočasno nahaja le en proces
 - ključavnica ščiti odsek
 - vstop in izstop v odsek



Ključavnica

- Poskus izvedbe

```
fun lock_init(lock) is  
    lock = 0
```

```
fun lock_enter(lock) is  
    while lock == 1 do nothing  
    lock = 1
```

```
fun lock_exit(lock) is  
    lock = 0
```

spin:

```
compare lock, 1  
jump.equal spin  
store lock, 1
```

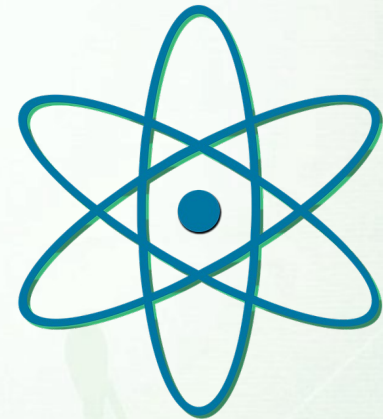
Kaj vse je
tu narobe?



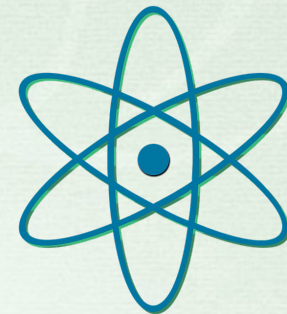
Ključavnica

- Kje je težava?
 - sočasnost preverjanja stanja ključavnice
 - ločenost nastavljanja stanja od preverjanja
 - rešitev
 - atomarnost celotne operacije

atomic fun lock_entr(lock) is
while lock == 1 **do** nothing
lock = 1

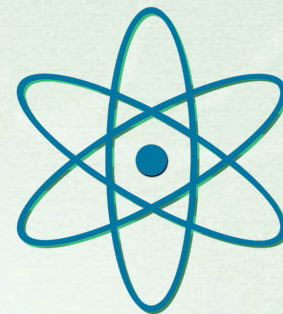
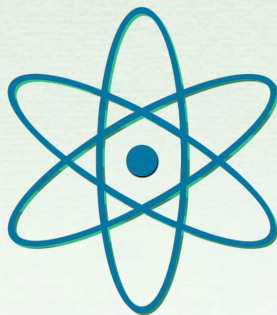


Ključavnica



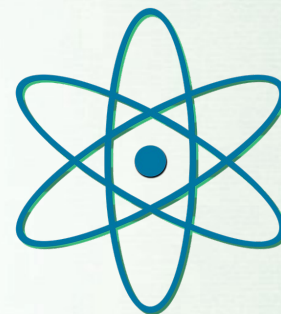
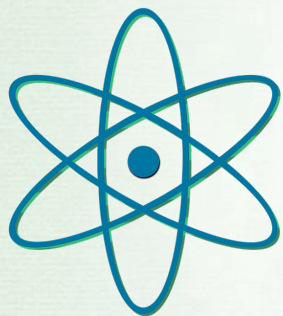
- Atomarna operacija
 - zaporedje enega ali več ukazov
 - s programskega vidika nedeljivo zaporedje
 - se izvede kot celota, ali pa se sploh ne izvede
 - izolacija od drugih sočasnih procesov
 - noben drug proces ne more prekiniti operacije ali imeti vpogleda v vmesno stanje operacije
 - vzajemno izključevanje
 - izvede se le ena naenkrat (več procesorjev)
 - obvoz predpomnilnika (več procesorjev)
 - zaradi usklajevanja vrednosti operanda
 - počasnejša operacija

Ključavnica

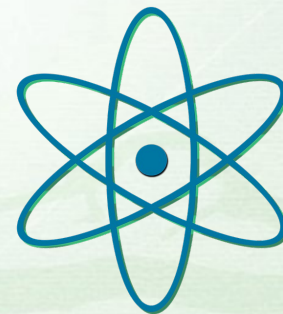
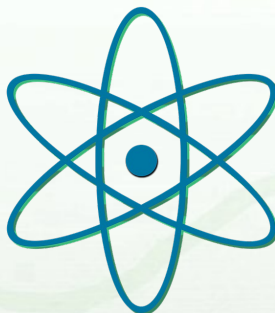
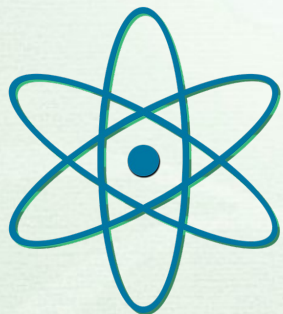


- Izziv

- strojnih ukazov `lock_enter` in `lock_exit` ni
- običajni strojni ukazi (navadno) niso atomični



Kako zagotoviti
atomičnost
ključnih operacij?



Strojna izvedba

- **Onemogočanje prekinitev**

- izvedba

- **vstop** v KO: onemogočimo prekinitve
 - **izstop** iz KO: omogočimo prekinitve

// uporaba

CLI vstop v KO

... kritični odsek

STI ... izstop v KO

- **slabosti**

- uporaba privilegiranih ukazov
 - npr. x86: STI (omogoči), CLI (onemogoči)
 - **onemogoči večopravilnost**
 - če se kritični odsek zacikla, se zacikla cel sistem
 - **ne deluje na večprocesorski arhitekturi**
 - na ostalih procesorji so prekinitve še vedno mogoče

Strojna izvedba

- Namenski strojni ukazi
 - **atomični** strojni ukazi
 - načelo ključavnice
 - vstop: zaklepanje
 - izstop: odklepanje
 - **vstop**
 - dokler je zaklenjeno čakaj
 - vrteče čakanje
(spin waiting, busy waiting)
 - **izstop**
 - enostavno prirejanje

```
// inicializacija ključavnice  
// skupna spremenljivka  
flag = 0
```

```
// vstop v KO: zaklepanje ključavnice  
fun enter() is  
    ... odvisno od ukaza
```

```
// izstop iz KO: odklepanje ključavnice  
fun leave() is  
    flag = 0
```

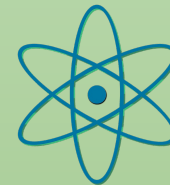
```
// uporaba  
enter() .... vstop v KO  
... kritični odsek ....  
leave() ... izstop v KO
```

Strojna izvedba

- **Ukaz: test & set**

- če je zaklenjeno, potem vrni `true`
- sicer zakleni in vrni `false`

```
atomic instruction test_and_set(ref val) is  
    if val then return true  
    val = true  
    return false
```



```
fun enter() is  
    while test_and_set(&flag) do nothing
```

Strojna izvedba

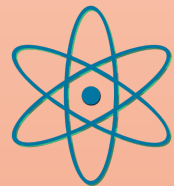
- **Ukaz: compare & swap**

- posplošitev `test & set`
- če je trenutna vrednost enaka testni vrednosti, potem jo zamenjaj z novo vrednostjo
- vedno vrne staro vrednost

testval = false
newval = true

atomic instruction `compare_and_swap(ref val, testval, newval)` is
 `oldval = val`
 if `val == testval` **then** `val = newval`
 return `oldval`

fun `enter()` **is**
 while `compare_and_swap(&flag, false, true)` **do** nothing



Strojna izvedba

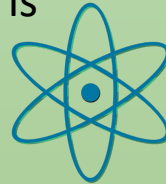
- **Ukaz: exchange**
 - zamenjava dveh vrednosti

atomic instruction exchange(ref a, ref b) is

t = a

a = b

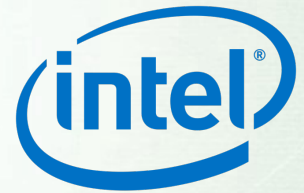
b = t



fun enter() is

key = true

do exchange(&flag, &key) **while** key



Intel: XCHG

ARM: SWP

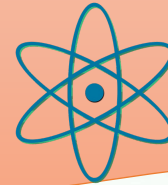


Strojna izvedba

- **Ukaz: fetch & add**

- poveča vrednost in
- vrne staro vrednost

atomic instruction `fetch_and_add(ref val)` is
oldval = val
val += 1
return oldval



fun `init()` **is**

ticket = turn = 0

fun `enter()` **is**

myturn = `fetch_and_add(&ticket)`

while lock.turn != myturn **do** nothing

fun `exit()` **is**

`fetch_and_add(turn)`

Ključavnica s
prepustnico



Strojna izvedba

- Namenski strojni ukazi

- prednosti

- rešitev deluje tudi na večprocesorskih sistemih
 - enostavnost izvedbe
 - podpora več kritičnim odsekom (več zastavic)

- slabosti

- vrteče čakanje zapravlja procesorski čas
 - možno stradanje
 - izbira vstopnega procesa je „naključna“ oz. odvisna od razvrščevalnika
 - možen smrtni objem
 - npr. P1 pridobi vir, prekine ga P2 z višjo prioriteto, ki gre v vrteče čakanje, ker vira ne dobi

Programska izvedba

Algoritem je objavil
znani kolega Dijkstra.

- **Dekkerjev algoritem, 1968**

- prvi algoritem za reševanje KO
- dva procesa si predajata prednost
- vrteče čakanje

```
me = 0
you = 1 - me
shared entering = [false, false]
shared turn = you

fun enter() is
    entering[me] = true
    while entering[you] do
        if turn == you then
            entering[me] = false
            while turn == you do nothing
            entering[me] = true
        endif
    endwhile

fun leave() is
    turn = you
    entering[me] = false
```

Programska izvedba

preprostejši kot
Dekkerjev algoritem

• Petersonov algoritem, 1981

me = 0 oz. 1

you = 1 – **me**

shared entering = [false, false]

shared **turn** = **you**

fun enter() **is**

entering[**me**] = true

turn = **you**

while **entering**[**you**] **and**
 turn == **you** **do** nothing

fun leave() **is**

entering[**me**] = false

... P0: **me** = 0, P1: **me** = 1

... P0: **you** = 1, P1: **you** = 0

... deljena spr., začetek: nobeden ne želi vstopiti

... deljena spr., na potezi je drugi proces

... želim vstopiti

... dam **ti** prednost

... dokler želiš **ti** vstopiti in

... je **tvoja** poteza, **jaz** čakam

... ne želim več vstopiti

Programska izvedba

- **Petersonov algoritem, 1981**
 - vzajemno izključevanje z vidka procesa P-me
 - če je P-me v KO, potem je `entering[me] = true`
 - za P-you velja ena od možnosti

- ne želi vstopiti: `entering[you] = false`
- želi vstopiti: `entering[you] = true` in je predal prednost: `turn = me`, zato čaka
- želi vstopiti: `entering[you] = true` in še ni predal prednosti: `turn = you`, zato še ni prišel do while zanke

```
me = 0 oz. 1
you = 1 - me
shared entering = [false, false]
shared turn = you

fun enter() is
    entering[me] = true
    turn = you
    while entering[you] and
        turn == you do nothing

fun leave() is
    entering[me] = false
```


Programska izvedba

- **Lamportov pekarski algoritem**
 - za N procesov, princip oštevilčenih listkov

```
shared taking = [false, false, ..., false]
```

```
shared number = [0, 0, ..., 0]
```

```
me = zaporedna št. procesa
```

```
fun enter() is
```

```
    taking[me] = true
```

... dodeljevanje številke

```
    number[me] = max(number) + 1
```

... največja dodeljena številka + 1

```
    taking[me] = false
```

```
for  $j = 0$  to  $N - 1$  do
```

```
    while taking[ $j$ ] do nothing
```

... počakaj, da nit j dobi številko

```
    while number[ $j$ ] != 0 and
```

... če j vstopa in če ima manjšo številko, potem čakaj

```
        (number[ $j$ ] < number[me] or number[ $j$ ] == number[me] and  $j < me$ )
```

```
        do nothing
```

```
fun leave() is
```

```
    number[me] = 0
```

Učinkovitost ključavnic

- Mere zmogljivosti
 - **zakasnitveni čas** (latency)
 - čas za pridobitev proste ključavnice
 - optimalno: takoj izvedemo atomično operacijo
 - **čakalni čas** (delay)
 - čas za pridobitev ravnokar sproščene ključavnice
 - optimalno: takoj pridobimo
 - **tekmovanje** (contention)
 - promet na (pomnilniškem) vodilu zaradi *atomične operacije* in *zagotavljanja koherentnosti predpomnilnika*
 - optimalno: nič

Učinkovitost ključavnic

- Izvedbe ključavnice
 - osnovna različica

```
fun enter() is
  while test_and_set(&flag) do nothing
```

- test + test and set

```
fun enter() is
  while flag or test_and_set(&flag) do nothing
```

flag

... testiranje v predpomnilniku, vrtenje

test_and_set

... atomična operacija, dostop do glavnega pomnilnika

- + zakasnitev

```
fun enter() is
  while flag or test_and_set(&flag) do
    while flag do nothing
    delay()
  end
```


Učinkovitost ključavnic

- Se komu vrti?
 - **vrteča ključavnica** (spinlock)
 - porablja procesorski čas za čakanje

```
fun enter() is  
    while test_and_set(&flag) do nothing
```

- uporabimo `yield()` (če ga imamo)

```
fun enter() is  
    while test_and_set(&flag) do yield()
```



Učinkovitost ključavnic

- Bi kdo raje spal?
 - namesto vrtečega čakanja
naj nit raje blokira
 - Kdaj se nit zbudi?



```
fun enter() is
    while test_and_set(&flag) do
        wait_for_flag_to_be_changed()

fun exit(ref lock) is
    lock = 0
    wake_up_some_waiting_thread()
```