

Operacijski sistemi

Sinhronizacija procesov

Vsebina

- Ključavnica
- Problem proizvajalci-porabniki
- Pogojna spremenljivka
- Problem bralci pisalci
- Bralno-pisalna ključavnica
- Semafor
- Monitor

Ključavnica

mutual exclusion

- Ključavnica (lock, mutex)
 - mehanizem **zaklepanja**
 - dve stanji: **odklenjeno** in **zaklenjeno**
 - niti, ki zaklene ključavnico, si jo lasti (jo drži)
 - dvojno zaklepanje (ista nit) ni definirano (napaka, ignoriranje, smrtni objem)
 - le ena nit si lahko sočasno lasti ključavnico
 - nit, ki zaklene ključavnico, jo lahko tudi odklene
 - odklepanje ključavnice, ki jo drži druga nit, ni definirano
 - uporaba
 - vzajemno izključevanje na kritičnih odsekih

Ključavnica

- Uporaba
 - operacija: **zakleni** (lock, acquire)
 - če je ključavnica odklenjena, potem jo zaklene
 - sicer čaka na odklenjeno ključavnico
 - operacija: **odkleni** (unlock, release)
 - odklene zaklenjeno (ista nit) ključavnico

```
pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
```

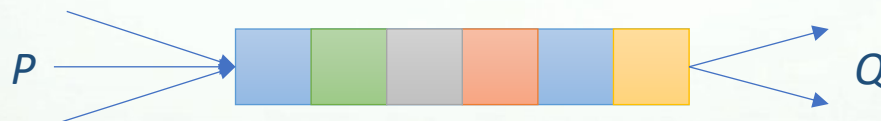
```
pthread_mutex_lock(&m);
```

```
// kritični odsek
```

```
pthread_mutex_unlock(&m);
```

Problem proizvajalci-porabniki

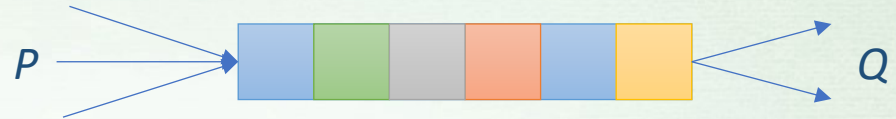
- Proizvajalci-porabniki
 - dana sta dva ali več sočasnih procesov
 - dve vrsti procesov
 - proizvajalec P proizvaja podatke
 - porabnik Q porablja podatke
 - komunikacija poteka preko *souporabe* vrste
 - pojav **tveganega stanja** pri sočasnem dostopu do vrste
 - omejena ali neomejena kapaciteta vrste



Kako razrešiti tvegano stanje?

Problem proizvajalci-porabniki

- Rešitev z zaklepanjem



```
pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;  
queue_t q = ...;
```

proizvajalec P

```
data = produce_data()  
pthread_mutex_lock(&m)  
q.enqueue(data)  
pthread_mutex_unlock(&m)
```

porabnik Q

```
while true do  
  if not queue.empty then  
    pthread_mutex_lock(&m)  
    q.dequeue(data)  
    pthread_mutex_unlock(&m)  
    consume_data(data)  
  else nothing  
done
```

Problem proizvajalci-porabniki

- Se da bolje?

proizvajalec P

```
data = produce_data()  
pthread_mutex_lock(&m)  
q.enqueue(data)  
pthread_mutex_unlock(&m)
```

porabnik Q

```
while true do  
  if not queue.empty then  
    pthread_mutex_lock(&m)  
    q.dequeue(data)  
    pthread_mutex_unlock(&m)  
    consume_data(data)  
  else nothing  
done
```

*Moti nas neprestano preverjanje
pogoja: ali je vrsta neprazna.*

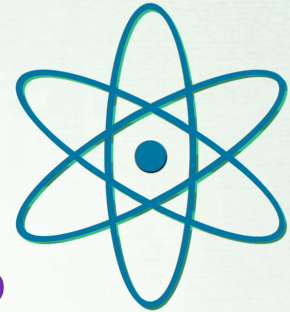
Pogojna spremenljivka

- Pogojna spremenljivka (condition variable)
 - mehanizem **obveščanja**
 - opazovani pogoj je potrebno eksplicitno preverjati
 - lahko gre za več pogojev
 - množica čakajočih niti
 - niti, ki čakajo na izpolnjenost pogoja
 - nit, ki obvesti čakajoče niti o izpolnjenosti pogoja
 - vsaj ena čakajoča nit se zbudi in nadaljuje izvajanje
 - se uporablja skupaj s ključavnico
 - uporaba
 - optimizacija uporabe procesorja

Pogojna spremenljivka

- Uporaba

- operacija: čakaj (wait)
 - sprosti pripadajočo ključavnico
 - trenutno nit se postavi v čakalno množico
 - blokira trenutno nit (preklop na drugo pripravljeno nit)
 - ko se nit spet zbudi, ponovno pridobi ključavnico
- operacija: obvesti (signal, notify)
 - zbudi eno nit iz čakalne množice
 - namen: obvestilo o izpolnjenem pogoju
- operacija: obvestiVse (broadcast, notifyAll)
 - zbudi vse niti iz čakalne množice
 - namen: kadar je izpolnjenih več pogojev



Pogojna spremenljivka

```
pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;  
pthread_cond_t c = PTHREAD_COND_INITIALIZER;
```

```
void* make(void *arg) {  
    // izpolni pogoj  
    ...  
    pthread_mutex_lock(&m);  
    update shared state  
    if (POGOJ)  
        pthread_cond_signal(&c)  
    pthread_mutex_unlock(&m);  
}
```

```
void* watch(void *arg) {  
    pthread_mutex_lock(&m);  
    while (not POGOJ)  
        pthread_cond_wait(&c, &m)  
    pthread_mutex_unlock(&m);  
    // obdelava po izpolnitvi pogoja  
    ...  
}
```

Problem proizvajalci-porabniki

- Rešitev z zaklepanjem in pogojno spremenljivko

```
pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;  
pthread_cond_t c = PTHREAD_COND_INITIALIZER;  
queue_t q = ...;
```

proizvajalec P

```
data = produce_data()  
pthread_mutex_lock(&m)  
q.enqueue(data)  
signal(c)  
pthread_mutex_unlock(&m)
```

porabnik Q

```
while true do  
    pthread_mutex_lock(&m)  
    while q.empty() do  
        wait(m, c)  
    q.dequeue(data)  
    pthread_mutex_unlock(&m)  
    consume_data(data)  
done
```

Problem proizvajalci-porabniki

- Omejena kapaciteta vrste
 - Kaj naj naredi proizvajalec, če je vrsta polna?
 - Kdo bo zbudil spečega proizvajalca?
 - Koliko pogojnih spremenljivk potrebujemo?
 - Katero obveščanje (signal, broadcast) uporabimo?

Problem bralci-pisalci

- Bralci-pisalci
 - več procesov sočasno dostopa do podatkov
 - bralci le berejo, pisalci podatke tudi spreminjajo

# bralcev	# pisalcev	možne težave	komentar
>0	0	ne	sočasno branje ni težava
0	1	ne	en pisalec ni težava
0	>1	da	sočasno pisanje je težava
>0	>0	da	sočasno branje in pisanje je težava

Problem bralci-pisalci

- Rešitev z osnovnim zaklepanjem
 - branje obdamo s ključavnico
 - pisanje obdamo s ključavnico
- Slabost
 - onemogočimo sočasno branje več bralcev

Se da bolje?

Problem bralci-pisalci

```
pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;  
pthread_cond_t r = PTHREAD_COND_INITIALIZER;  
pthread_cond_t w = PTHREAD_COND_INITIALIZER;  
volatile int counter = 0;
```

bralec

```
pthread_mutex_lock(&m)  
while counter == -1 do  
    wait(m, r)  
    counter++  
pthread_mutex_unlock(&m)  
// branje podatkov  
pthread_mutex_lock(&m)  
counter--  
if counter == 0 then  
    signal(w)  
pthread_mutex_unlock(&m)
```

pisalec

```
pthread_mutex_lock(&m)  
while counter != 0 do  
    wait(m, w)  
    counter = -1  
pthread_mutex_unlock(&m)  
// pisanje podatkov  
pthread_mutex_lock(&m)  
counter = 0  
broadcast(r)  
signal(w)  
pthread_mutex_unlock(&m)
```

Bralno-pisalna ključavnica



- **Izključujoče** zaklepanje (exclusive, write lock)
 - le ena nit jo lahko sočasno zaklene
 - poljubna dodatna zaklepanja so zavrnjena
 - onemogoča souporabo vira drugim nitim
 - uporaba: pisanje, ne da bi komu pokvarili branje
- **Deljeno** zaklepanje (shared, read lock)
 - več niti jo lahko sočasno “deljeno” zaklene
 - več deljenih ključavnic, izključujoče ključavnice so zavrnjene
 - onemogoča souporabo izključujoče ključavnice
 - uporaba: enovito/celovito branje

Bralno-pisalna ključavnica

```
pthread_rwlock_t lock = PTHREAD_RWLOCK_INITIALIZER;
```

```
// bralni odsek  
pthread_rwlock_rdlock(lock);  
    // enovitno branje  
pthread_rwlock_unlock(lock);
```

```
// pisalni odsek  
pthread_rwlock_wrlock(lock);  
    // pisanje  
pthread_rwlock_unlock(lock);
```

Bralno-pisalna ključavnica

- Vrste zaklepanja
 - **svetovalna ključavnica** (advisory lock)
 - neodvisnost operacije od ključavnice
 - dejansko branje/pisanje je se ne ozira na stanje ključavnic
 - programer mora sam poskrbeti za ustrezno uporabo ključavnic pri bralno/pisalnih odsekih programa
 - **obvezna ključavnica** (mandatory lock)
 - odvisnost operacije od ključavnice
 - OS poskrbi, da branje/pisanje upošteva stanje ključavic

Bralno-pisalna ključavnica

- Izvedba v POSIX ipd.
 - zaklepanje je vezano na inode
 - ni vezano na ime datoteke ali datotečni deskripto
 - vsak proces si lahko lasti eno ali več ključavnic
 - ključavnice se ne dedujejo: `fork()`
 - sistemski klic `int flock(fd, tip)`
 - `fd`: LOCK_SH (shared), LOCK_EX (exclusive), LOCK_UN (unlock), LOCK_NB (non-blocking)
 - ukaz: `flock -n fd`
 - ukaz: `lockfile`

Semafor

- Izzivi

- zahtevna uporaba ključavnic in pogojnih spremenljivk
 - vrstni red operacij, izpustitev operacije ipd.
- potrebujemo tako ključavnice kot pogojne spremenljivke

- Rešitev

- mehanizmi, ki združujejo delovanje
- semafor
- monitor

Semafor

- Semafor
 - združuje
 - števec, ključavnico in pogojno spremenljivko
 - **pozitivna** vrednost števca (vključno z 0)
 - predstavlja odprt semafor
 - prost prehod procesa v KO
 - **negativna** vrednost števca
 - predstavlja zaprt semafor
 - prehod v KO ni možen
 - proces čaka na odprtje semaforja

Semafor

- Uporaba semaforja
 - operacija **wait** (tudi *test, down, proberen, P*)
 - dekrementira števec
 - če je semafor zaprt, potem gre proces v čakalno vrsto semaforja
 - operacija **signal** (tudi *increment, up, verhogen, V*)
 - inkrementira števec
 - če je semafor odprt, potem zbudi morebitni čakajoči proces iz semaforjeve vrste
 - Kako zagotoviti atomičnost obeh operacij?
 - strojni ukazi, kratko vrteče čakanje
 - en procesor, onemogočanje prekinitev



Semafor

Izvedba in uporaba

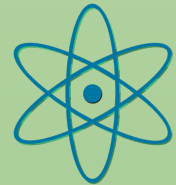
```
// priprava  
sem: Semaphore  
init(sem, 1)  
...
```

```
// zaklepanje  
wait(sem)  
// kritični odsek  
signal(sem)  
...
```

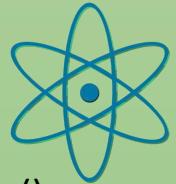
```
struct Semaphore is  
    count: Int  
    queue: Queue
```

```
fun init(sem, value) is  
    sem.value = value  
    sem.queue = []
```

```
atomic fun wait(sem) is  
    sem.count -= 1  
    if sem.count < 0 then  
        queue.enqueue(this process)  
        make_waiting(this process)
```



```
atomic fun signal(sem) is  
    sem.count += 1  
    if sem.count <= 0 then  
        process = queue.dequeue()  
        make_ready(this process)
```



Semafor

- Semafor – POSIX API

```
// tip semafor  
sem_t sem;
```

```
// inicializacija semaforja  
sem_init(sem_t *sem, int pshared, int count)
```

način
souporabe

začetna
vrednost

```
// operacija wait  
sem_wait(sem_t *sem);
```

```
// opearcija signal  
sem_post(sem_t *sem);
```


Monitor

- Monitor
 - Hoare, 1974; Brinch Hansen, 1975
 - višje nivojski konstrukt
 - podpora s strani programskega jezika
 - npr. JVM: Java, ..., .NET: C#, ...
 - vstop in izstop v kritični odsek generira prevajalnik
 - vsebnik
 - eksplicitno specificira souporabljeni vir
 - vsebuje lokalne spremenljivke in funkcije
 - naenkrat se znotraj monitorja lahko izvaja le ena funkcija, četudi je več procesov
 - medsebojno izključevanje izvajanja funkcij

Monitor

- Primer

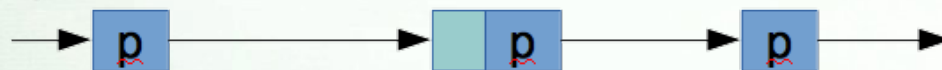
monitor BancniRacun **is**
stanje: Int

sync fun poloziDenar(polog) **is**
stanje += polog

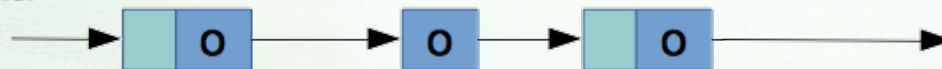
sync fun dvigniDenar(dvig) **is**
stanje -= dvig

sync fun obresti(p) **is**
stanje *= (1 + p / 100)

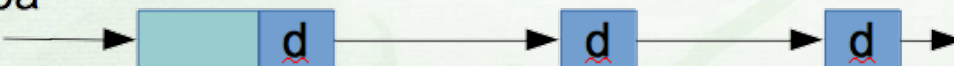
Podjetje



Banka



Oseba



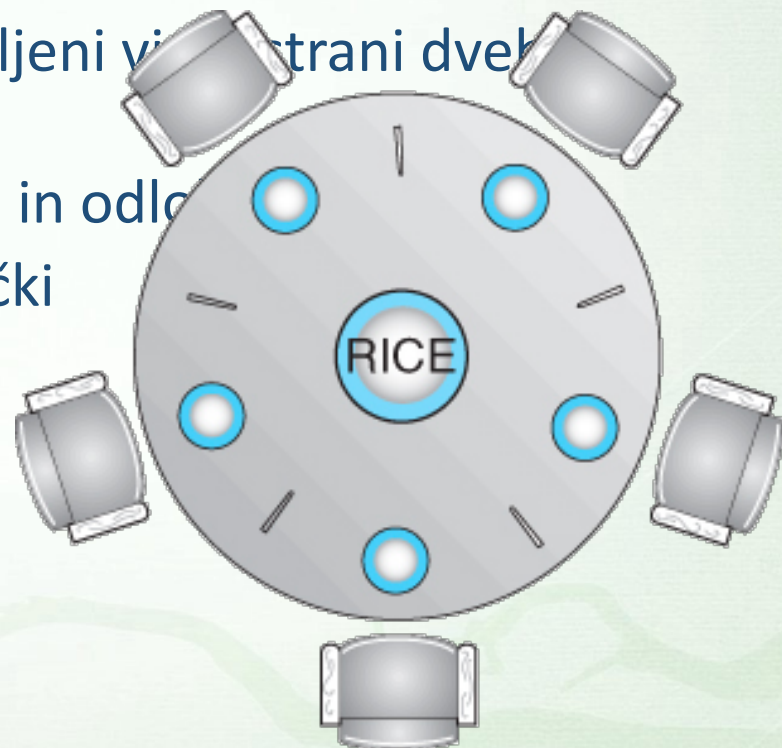
Ostali mehanizmi

- Ostali mehanizmi
 - pregrada (barrier)
 - ideja delovanja je obratna od semaforjev
 - semafor dopušča vstop N procesom, pregrada za vstop zahteva N procesov
 - serializers
 - definicija prioritet
 - path expressions
 - regularni izrazi za definicijo usklajenega obnašanja
 - wait-free sync
 - optimistično, read-copy-update ključavnica

Lačni filozofi

- Problem: **lačni filozofi**

- kitajski filozofi sedijo za okroglo mizo
 - vsak ima krožnik in eno palčko na vsaki strani krožnika
 - vsaka palčka predstavlja deljeni višinski strani dve (filozofov)
 - ko ni lačen, filozof razmišlja in odloči
 - filozof je, le če ima obe palčki
- izzivi
 - smrtni objem
 - stradanje



Lačni filozofi

- Rešitve: **lačni filozofi**

- centralna avtoriteta, ki odreja prehranjevanje
- filozof lahko vzame samo obe palčki hkrati
- asimetrična rešitev
 - lihi filozofi najprej vzamejo levo palčko, nato desno
 - sodi filozofi pa najprej desno, nato levo
- itd.

