

③ Semaforit ja rinnakkaisuuden hallinta



Tuottajat ja kuluttajat
Resurssien hallinta, vuoron antaminen
Lukijat ja kirjoittajat

Andrews 4.2, 4.4-4.6

Tuottajat ja kuluttajat

Andrews: ss.158-160

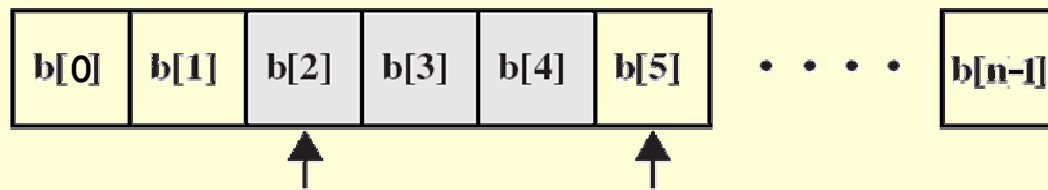
```

typeT buf;      /* a buffer of some type T */
sem empty = 1, full = 0;
process Producer[i = 1 to M] {
    while (true) {
        ...
        /* produce data, then deposit it in the buffer */
        P(empty);
        buf = data;
        V(full);
    }
}
process Consumer[j = 1 to N] {
    while (true) {
        /* fetch result, then consume it */
        P(full);
        result = buf;
        V(empty);
        ...
    }
}

```

Andrews Fig. 4.3:
Producers and consumers using semaphores.
 (split binary semaphores)

Toimiiko oikein? Mitä pitääkään tarkistaa?



```
typeT buf[n];  
int front = 0, rear = 0;  
sem empty = n, full = 0;
```

```
process Producer {  
  while (true) {  
    ...  
    produce message data  
    P(empty);  
    buf[rear] = data;  
    rear = (rear+1) % n;  
    V(full);  
  }  
}
```

```
process Consumer {  
  while (true) {  
    fetch and consume:  
    P(full);  
    result = buf[front];  
    front = (front+1) % n;  
    V(empty);  
    ...  
  }  
}
```

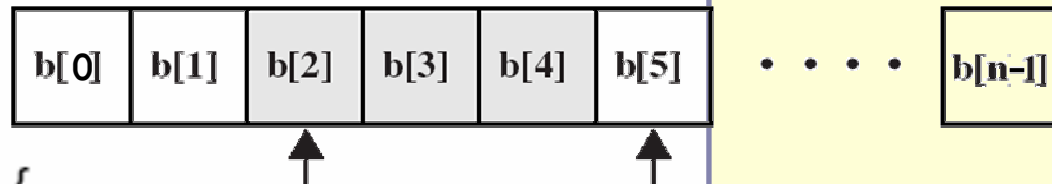
Andrews Fig. 4.4: Bounded buffer using semaphores.

```

typeT buf[n];          /* an array of some type T */
int front = 0, rear = 0;
sem empty = n, full = 0;    /* n-2 <= empty+full <= n */
sem mutexD = 1, mutexF = 1; /* for mutual exclusion */
process Producer[i = 1 to M] {
    while (true) {
        ...
        produce message data and deposit it in the buffer;
        P(empty);
        P(mutexD);
        buf[rear] = data; rear = (rear+1) % n;
        V(mutexD);
        V(full);
    }
}

process Consumer[j = 1 to N] {
    while (true) {
        fetch message result and consume it;
        P(full);
        P(mutexF);
        result = buf[front]; front = (front+1) % n;
        V(mutexF);
        V(empty);
        ...
    }
}

```



**Andrews Fig. 4.5:
Multiple producers
and consumers
using semaphores.**

Entä, jos vain yksi mutex?

Lukijat ja kirjoittajat

Lukijat ja kirjoittajat

- **Yhteinen tietokanta DB**
 - tai joku muu objekti, esim. tdsto, lista, ...
- **Kaksi kilpailevaa käyttäjäluokkaa**
 - **Lukijat (readers)**
 - lukevat
 - useita lukijoita voi olla käsittelemässä yhtäaikaa
 - **Kirjoittajat (writers)**
 - lukevat ja muuttavat
 - vain yksi kerrallaan muuttamassa
- **Luokka aktiivinen vain,
jos toinen luokka passiivinen**

Ratkaisu 1: R/W poissulkemisongelmana

- **Yksinkertaistettu ongelma, yksink. ratkaisu**
 - Poissulkeminen kaikille muille ⇒ **semafori rw**
 - Andrews Fig 4.8

- **Salli lukijoiden toimia rinnakkain**
 - Luokan poissulkeminen ⇒ **semafori rw**
 - Eka lukija varaa DB:n lukijaluokalle, viimeinen lukija vapauttaa DB:n
 - Kuka on eka / viimeinen? ⇒ **laskuri nr**
 - “testaa, varaa / vapauta” ⇒ **atominen: < ... >**
 - Andrews Fig 4.9 and Fig 4.10

Andrews: ss.
167-169


```

sem rw = 1;
process Reader[i = 1 to M] {
    while (true) {
        ...
        P(rw);    # grab exclusive access lock
        read the database;
        V(rw);    # release the lock
    }
}
process Writer[j = 1 to N] {
    while (true) {
        ...
        P(rw);    # grab exclusive access lock
        write the database;
        V(rw);    # release the lock
    }
}

```

Andrews Fig. 4.8:
An overconstrained solution.

```

int nr = 0;          # number of active readers
sem rw = 1;         # lock for reader/writer exclusion
process Reader[i = 1 to M] {
  while (true) {
    ...
    < nr = nr+1;
      if (nr == 1) P(rw); # if first, get lock
    >
    read the database;
    < nr = nr-1;
      if (nr == 0) V(rw); # if last, release lock
    >
  }
}
process Writer[j = 1 to N] {
  while (true) {
    ...
    P(rw);
    write the database;
    V(rw);
  }
}

```

Andrews Fig. 4.9: Outline of readers and writers solution.

```

int nr = 0;           # number of active readers
sem rw = 1;          # lock for access to the database
sem mutexR = 1;      # lock for reader access to nr
process Reader[i = 1 to m] {
    while (true) {
        ...
        P(mutexR);
        nr = nr+1;
        if (nr == 1) P(rw); # if first, get lock
        V(mutexR);
        read the database;
        P(mutexR);
        nr = nr-1;
        if (nr == 0) V(rw); # if last, release lock
        V(mutexR);
    }
}
process Writer[j = 1 to n] {
    while (true) {
        ...
        P(rw);
        write the database;
        V(rw);
    }
}

```

Andrews Fig. 4.10:
Readers and writers exclusion
using semaphores.

Ratkaisu 2: R/W synkronointiongelmana

● **Odota, kunnes sopiva ehto tulee todeksi**

- Toiminnallisuus: $\langle \text{await (ehto) lauseet}; \rangle$
- Ehdon testaus ja lauseosa atomiseksi

● **Tila**

- **BAD:** $(nr > 0 \text{ and } nw > 0) \text{ or } nw > 1$
- **RW:** $(nr == 0 \text{ or } nw == 0) \text{ and } nw \leq 1$
 - RW muuttumaton, oltava voimassa aina
 - $nr =$ number of readers, $nw =$ number of writers

● **Ohjelmoi s.e. ehto RW on aina true**

- saa lukea $nw == 0$
- saa kirjoittaa $nr == 0 \text{ and } nw == 0$

```

int nr = 0, nw = 0;
## RW:  (nr == 0  $\vee$  nw == 0)  $\wedge$  nw <= 1
process Reader[i = 1 to m] {
    while (true) {
        ...
        <await (nw == 0) nr = nr+1;>
        read the database;
        <nr = nr-1;>
    }
}
process Writer[j = 1 to n] {
    while (true) {
        ...
        <await (nr == 0 and nw == 0) nw = nw+1;>
        write the database;
        <nw = nw-1;>
    }
}

```

Andrews Fig. 4.11:
**A coarse-grained
 readers/writers solution.**

Ratkaisu 2, laajennettu

● Ota huomioon käyttövuorot

- Lukijat ensin, tai ...
- Vuoro prioriteetin perusteella: Kirjoittajat ensin!
(jos muutokset tärkeitä; nälkiintymisvaara)

● Onko, joku odottamassa vuoroa?

- Ei voi kysyä semafori-operaatioilla \Rightarrow oma laskuri

● Vuorojen toteutus rutiinissa *SIGNAL*

- Kriittinen alue vapautuu \Rightarrow joku muu saa jatkaa, kuka?
 - odottava lukija, odottava kirjoittaja, kokonaan uusi tulija

● Andrews Fig. 4.12

● **Jaettu binääriarvoinen semafori**

- Liitä kuhunkin vahtiin (ehtoon) semafori ja laskuri
- Vain yksi semafori kerrallaan 'auki' ($0 \leq (e+r+w) \leq 1$)

● **await (**nw==0**) nr = nr+1**

- semafori: $r = 0$ *waiting place for readers*
laskuri: dr *number of delayed readers*

● **await (nr==0 and nw==0) nw = nw+1**

- semafori: $w = 0$ *waiting place for writers*
laskuri: dw *number of delayed writers*

● **poissulkeminen < ... >**

- semafori: $e = 1$ *waiting place for entry*
laskurit: nr, nw *numbers of readers and writers*

• process Reader {

while (true) {

<await (nw == 0) nr = nr + 1;>

P(e); #varmistaa atomisuuden

if (nw > 0){ #joudutaan odottamaan

dr = dr + 1;

V(e);

P(r);}

nr=nr+1;

SIGNAL;

read the database;

<nr = nr -1;>

P(e); #varmistaa atomisuuden

nr=nr-1

SIGNAL;

}

}

process Reader {

while (true) {

P(e); #varmistaa atomisuuden

<await (nw == 0) nr = nr + 1;>

if (nr>0 or nw > 0){ #joudutaan odottamaan

dw = dw + 1;

V(e);

P(w);}

nw=nw+1;

SIGNAL;

read the database;

<nr = nr -1;>

P(e); #varmistaa atomisuuden

nw=nw-1

SIGNAL;

}

}

```

int nr = 0,    ## RW: (nr == 0 or nw == 0) and nw <= 1
    nw = 0;
sem e = 1,    # controls entry to critical sections
    r = 0,    # used to delay readers
    w = 0;    # used to delay writers
                # at all times 0 <= (e+r+w) <= 1
int dr = 0,    # number of delayed readers
    dw = 0;    # number of delayed writers

```

**Andrews Fig. 4.12:
Outline of readers
and writers with
passing the baton.**

```

process Reader[i = 1 to M] {
  while (true) {
    # <await (nw == 0) nr = nr+1;>
    P(e);
    if (nw > 0)
      { dr = dr+1; V(e); P(r); }
    nr = nr+1;
    SIGNAL;
    read the database;
    # <nr = nr-1;>
    P(e);
    nr = nr-1;
    SIGNAL;
  }
}

```

```

process Writer[j = 1 to N] {
  while (true) {
    # <await (nr== 0 and nw== 0) nw = nw+1;>
    P(e);
    if (nr > 0 or nw > 0)
      { dw = dw+1; V(e); P(w); }
    nw = nw+1;
    SIGNAL;
    write the database;
    # <nw = nw-1;>
    P(e);
    nw = nw-1;
    SIGNAL;
  }
}

```

● **SIGNAL - vuoron antaminen: Lukijat ensin!**

```
if (nw == 0 and dr > 0) {  
    dr = dr -1;  
    V(r);           # herätä odottava lukija, tai  
}  
else if (nr == 0 and nw == 0 and dw > 0) {  
    dw = dw -1;  
    V(w);           # herätä odottava kirjoittaja, tai  
}  
else  
    V(e);           # päästä joku uusi etenemään
```

● **Menetelmä: Viestikapulan välitys** (Baton passing)

Viestikapulan välitys (Baton passing)

● **Vain yksi etenee kerrallaan kriittisillä alueilla**

- pyydettävä etenemislupaa: $P(e)$
- se etenee, joka 'saa' semaforin e (\sim viestikapula)

● **Muiden odotettava**

- Kokonaan uusi lukija tai kirjoittaja: $P(e)$
- Jos etenijän pyyntöön ei voi suostua:
 - Lukijat: $V(e); P(r)$
 - Kirjoittajat: $V(e); P(w)$

• Etenijä aktivoi itse “seuraavan viestikapulan haltijan” (kohdassa *SIGNAL*)

- jos odottajia,
 - herätä odottaja semaforista: joko $V(r)$ tai $V(w)$
 - jätä sille poissulkemissemafori valmiiksi kiinni: älä tee $V(e)$
- jos ei odottajia,
 - jätä viestikapula vapaaksi uusille tulijoille: $V(e)$

• Herätetty välittää aikanaan kapulan seuraavalle ja se seuraavalle ja ...

⇒ **Takaa ettei kukaan pääse ‘etuilemaan’!**

- Ehdot taatusti voimassa, kun jonottaja saa prosessorin
- *SIGNAL* aktivoi vain yhden prosessin, kun ehto tuli todeksi
- **FCFS**

- **SIGNAL - vuoron antaminen: Lukijat ensin!**

```
if (nw == 0 and dr > 0) {  
    dr = dr -1;  
    V(r);           # herätä odottava lukija, tai  
}  
else if (nr == 0 and nw == 0 and dw > 0) {  
    dw = dw -1;  
    V(w);           # herätä odottava kirjoittaja, tai  
}  
else  
    V(e);           # päästä joku uusi etenemään
```

- Osa ehdoista on jo tiedossa, kun ollaan **lukijassa** tai **kirjoittajassa** => voidaan jättää pois eikä tarvitse enää testata!

**Andrews Fig. 4.13:
A readers / writers
solution using
passing the baton.**

```
process Reader[i = 1 to M] {
  while (true) {
    # <await (nw == 0) nr = nr+1;>
    P(e);
    if (nw > 0) { dr = dr+1; V(e); P(r); }
    nr = nr+1;
    if (dr > 0) { dr = dr-1; V(r); }
    else V(e);
    read the database;
    # <nr = nr-1;>
    P(e);
    nr = nr-1;
    if (nr == 0 and dw > 0)
      { dw = dw-1; V(w); }
    else V(e);
  }
}
```

```
process Writer[j = 1 to N] {
  while (true) {
    # <await (nr==0 and nw==0) nw = nw+1;>
    P(e);
    if (nr > 0 or nw > 0)
      { dw = dw+1; V(e); P(w); }
    nw = nw+1;
    V(e);
    write the database;
    # <nw = nw-1;>
    P(e);
    nw = nw-1;
    if (dr > 0) { dr = dr-1; V(r); }
    elseif (dw > 0) { dw = dw-1; V(w); }
    else V(e);
  }
}
```

Lukijat ensin

Tarpeettomat osat annetusta
SIGNAL-koodista poistettu

- **SIGNAL - vuoron antaminen: Kirjoittajat ensin!**

- **Reader**: uusi lukija odottamaan, jos kirjoittaja odottamassa (rivi 5)

```
if (nw>0 or dw>0) # DELAY
    { dr=dr+1; V(e); P(r); }
```

- **Writer**: herätä lukija vain, jos kirjoittajia ei odottamassa (rivi 13)

```
if (dw>0) { # SIGNAL
    dw = dw-1; V(w); # herätä kirjoittaja
}
elseif (dr>0) { # herätä lukija
    dr = dr-1; V(r);
}
else V(e); # herätä uusi
```




Resurssien hallinta ja vuoronantaminen

Resurssien hallinta

• **Manageri**

- **Objektit:** varaus ja vapautus
- **Vuorot:** kuka odottajista saa jatkaa?

• **Sisäinen kirjanpito kustakin erillisestä resurssista** (private)

- ominaisuudet (esim. muistiosoite, koko, ...)
- vapaa / varattu
- käyttäjä (esim. prosessin ID)

• **Rajapinta, API** (public)

- **pyydä**(parametrit)
- **vapauta**(parametrit)

Operaatioiden raaka runko

• **pyydä(parametrit)**

< **await** (pyyntöön voi suostua)
anna resurssi, merkitse varatuksi >

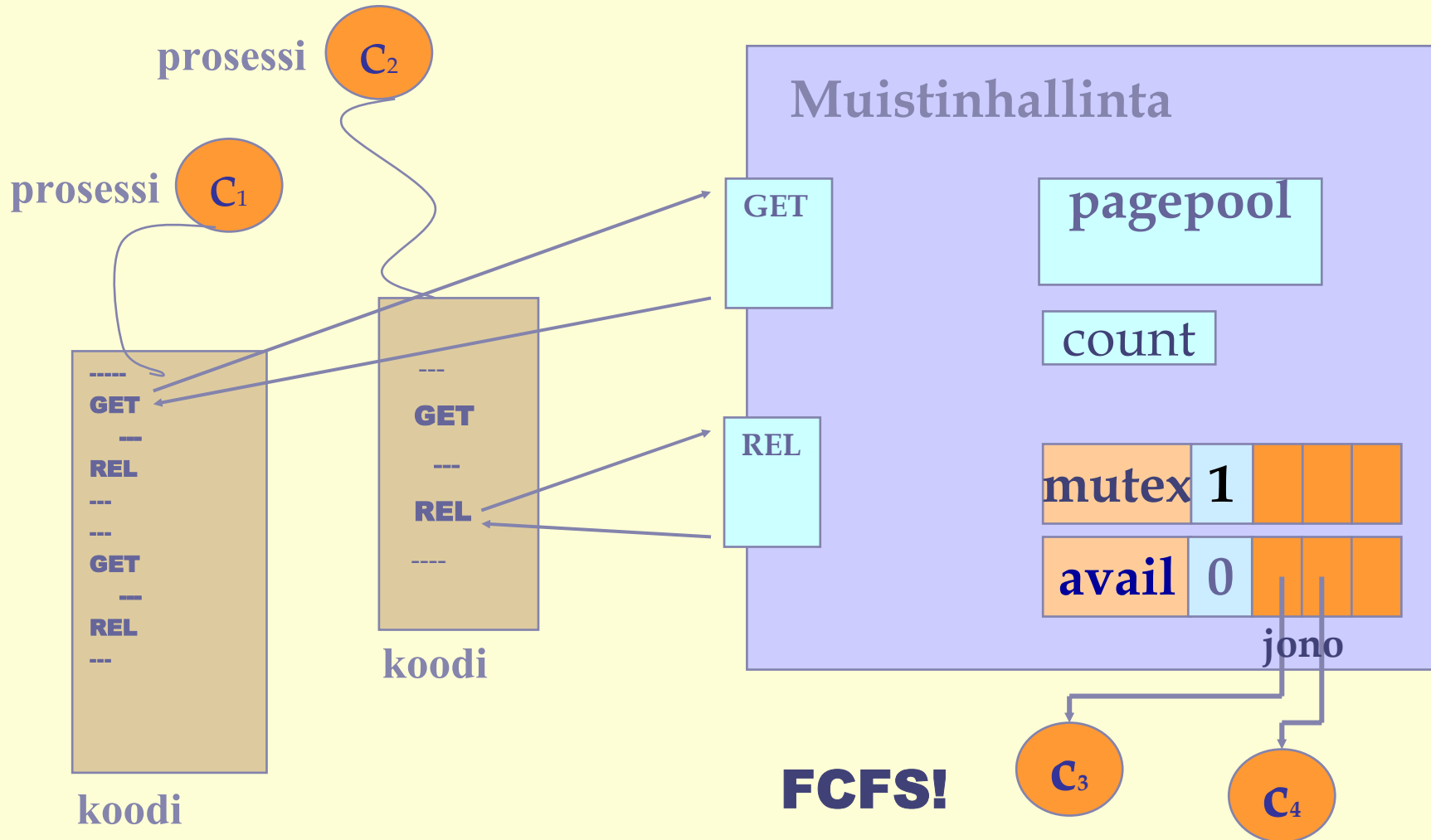
• **vapauta(parametrit)**

< palauta resurssi kirjanpitoon >

• **Vrt. entry protocol, exit protocol**

Muistinhallinta (3)

pyynnöt sivu kerrallaan

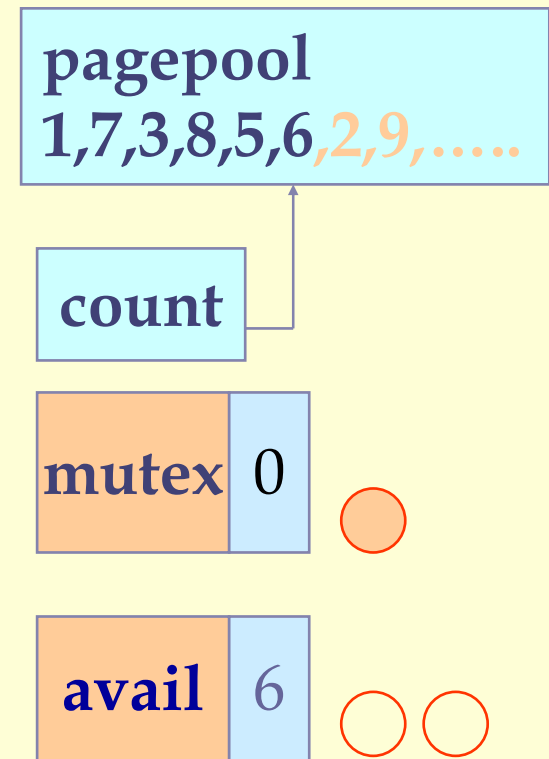


```

int count=MAX;
sem mutex=1, avail=MAX;
function GET(): returns addr {
    P(avail) ; // await
    P(mutex); // anna, merkitse
    get = pagepool[count];
    count = count-1;
    V(mutex);
}

procedure REL(addr freepage) {
    P(mutex); // palauta
    count = count+1;
    pagepool[count] = freepage;
    V(mutex);
    V(avail) ; // pyyntöön voi suostua
}

```



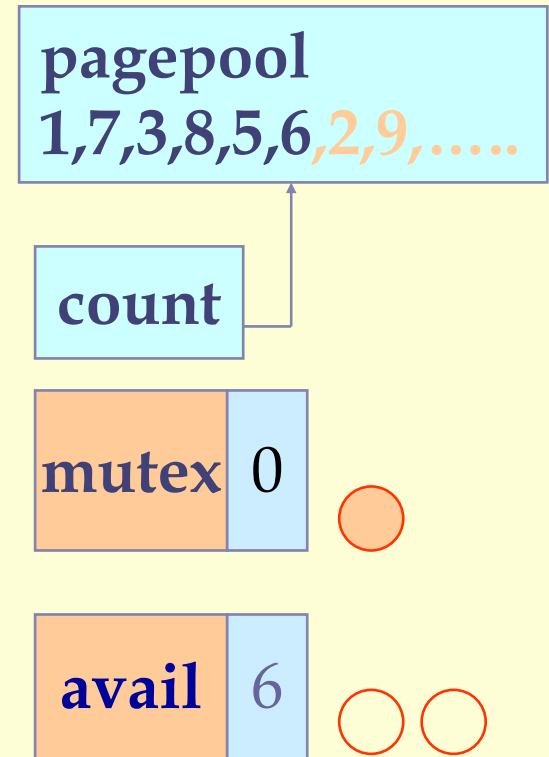
● **Toimivatko rutiinit oikein**

- **Poissulkeminen?**
- **Ei lukkiutumista (deadlock/livelock)?**
- **Ei tarpeettomia viipeitä?**
- **Lopulta onnistaa?**

VÄÄRIN!

```
int count=MAX;
sem mutex=1, avail=MAX;
function GET(): returns addr {
    P(mutex);
    P(avail) ;
    get = pagepool[count];
    count = count-1;
    V(mutex);
}
```

```
procedure REL(addr freepage) {
    P(mutex) ;
    count = count+1;
    pagepool[count] = freepage;
    V(avail) ;
    V(mutex);
}
```



Resurssien hallinta, Yleinen ratkaisu

• pyydä(parametrit)

P(mutex);

if (pyyntöön ei voi suostua) *DELAY;*

anna resurssi;

SIGNAL;

poissulkeminen

odota semaforissa

• vapauta(parametrit)

P(mutex);

palauta resurssi;

SIGNAL;

DELAY:

Älä jätä prosessia

Blocked-tilaan

tärkeä semafori kiinni!

• *DELAY* ~

- **V(mutex), P(odotussemafori)**

• *SIGNAL* ~

- **V(odotussemafori) else V(mutex)**

SIGNAL:

Herätä odottaja,

jätä kriittinen alue kiinni

(baton passing).

Palvelujärjestys

● Semaforin jonot aina FCFS

- Ongelma? Jäljellä 10 sivutilaa, eka haluaa 12, toka 5!

● Voiko semaforiin liittää prioriteetin?

- Jonotusjärjestys?

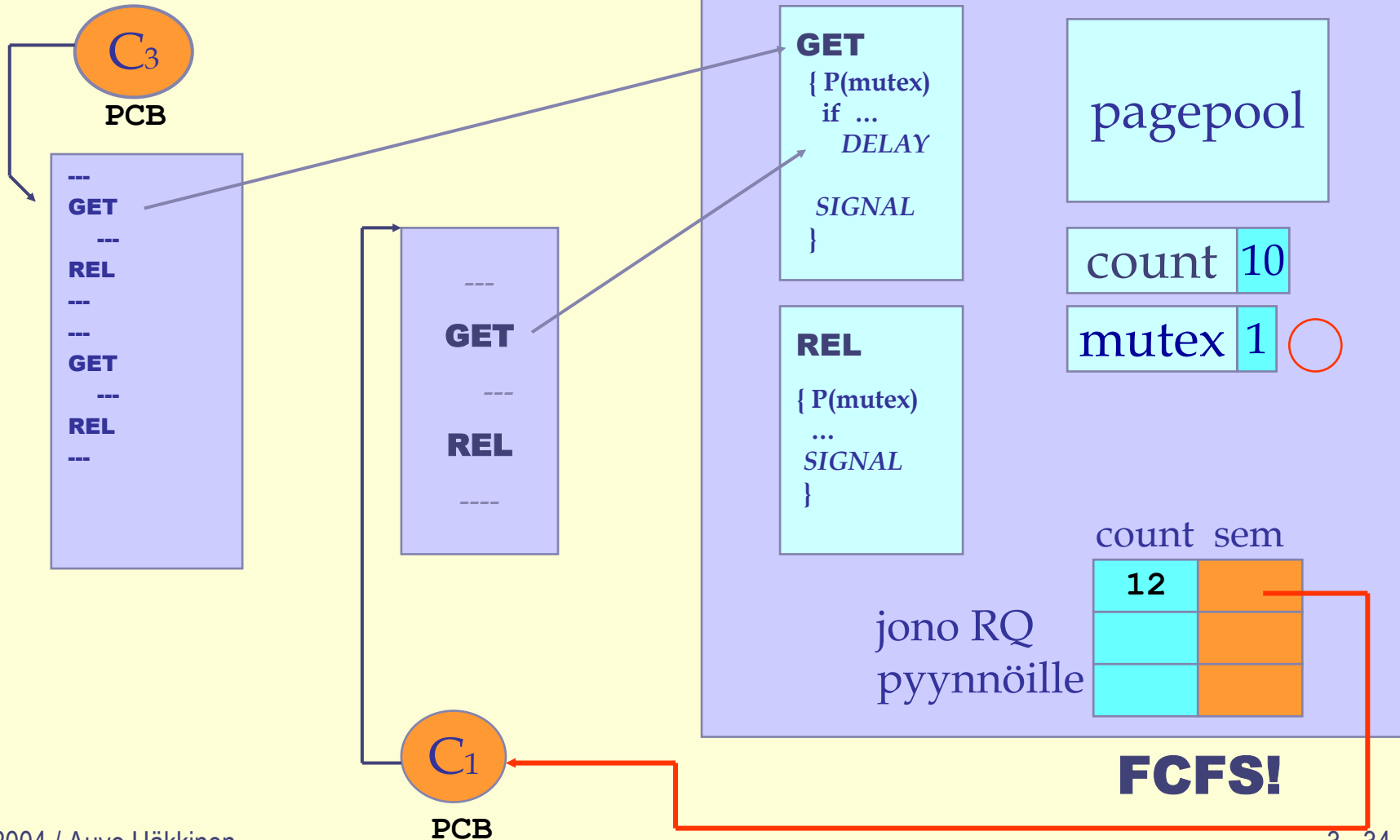
● Montako erilaista? Dynaaminen vuorottelu?

● Ratkaisu: **yksityiset semaforit** + oma jono

- Kullekin prosessille oma semafori, jossa odottaa yksin
- Vuoron antamiseen käytettävä tietorakenne (jono) erikseen
 - alkiossa semafori ja kenties muuta tietoa
- Vuorottaja valitsee sopivan prosessin vuorojonosta, ja päästää liikkeelle semaforissa odottavan asiakkaan

Muistinhallinta (4)

pyynnöt useita sivuja kerralla



```
procedure GET (nbr_of_units) {  
    P(mutex);  
    if (request can not be satisfied) { # DELAY  
        RQ[tail].count = nbr_of_units;  
        V(mutex);  
        P(RQ[tail].sem);  
    }  
    take nbr_of_units for this process;  
    if (! empty(RQ) and RQ[i].count < count)  
        V(RQ[i].sem);                # SIGNAL  
    else  
        V(mutex);  
}
```

```
procedure REL (list_of_units) {  
    P(mutex);  
    return units into pagepool;  
    if (! empty(RQ) and RQ[i].count < count)  
        V(RQ[i].sem);                # SIGNAL  
    else  
        V(mutex);  
}
```

SJN: Lyhyin työ seuraavaksi

- request(time,id):

P(e);

if (!free) DELAY;

free = false;

SIGNAL;

- release():

P(e);

free = true;

SIGNAL;

● DELAY:

- **Odottajan ID ja TIME** (suoritus aika) suoritusajan mukaan järjestettyyn jonoon (**PAIRS**) oikeaan kohtaan
- **V(e)** eli vapauta kriittinen alue
- jää odottamaan vuoroasi $P(b[ID])$
 - Tässä tarvitaan **kullekin oma semafori**, jotta pystytään 'herättämään' oikea prosessi: $b[n] = ([n] \ 0)$
 - **PAIRS-jono määrää järjestyksen**: herätetään aina jonon ensimmäinen prosessi

● SIGNAL:

Request-vaihe

- vapauta kriittinen alue **V(e)** eli päästä joku uusi Request-vaiheeseen

Release-vaiheen lopussa

- Jos jonossa odottajia, niin ota jonon ensimmäisen alkiopari (**time, ID**) ja herätä prosessi ID: **V(b[ID]);**
- muuten **V(e)**

```

bool free = true;
sem e = 1, b[n] = ([n] 0); # for entry and delay
typedef Pairs = set of (int, int);
Pairs pairs =  $\emptyset$ ;
## SJN: pairs is an ordered set  $\wedge$  free  $\Rightarrow$  (pairs ==  $\emptyset$ )
request(time, id):
    P(e);
    if (!free) {
        insert (time, id) in pairs;
        V(e); # release entry lock
        P(b[id]); # wait to be awakened
    }
    free = false;
    V(e); # optimized since free is false here
release():
    P(e);
    free = true;
    if (P !=  $\emptyset$ ) {
        remove first pair (time, id) from pairs;
        V(b[id]); # pass baton to process id
    }
    else V(e);

```

Andrews Fig. 4.14:
Shortest job next
allocation using
semaphores.

Entä, jos resurssia enemmän kuin 1 yksikkö?

- **amount** = montako yksikköä prosessi tarvitsee tai palauttaa
- **avail** = montako yksikköä on vapaana (~free)
- **request:**
 - testattava, onko vapaana tarvittu määrä yksiköitä **amount** \leq **avail**. Jos on, niin varataan, muuten talletetaan myös **amount**
 - myös tässä voidaan vapauttaa odottavia prosesseja, jos vapaita resursseja on tarpeeksi
- **release:**
 - vapautetaan jonosta ensimmäinen prosessi, jonka tarpeet pystytään tyydyttämään

POSIX-kirjasto, pthread

Kurssi: Verkkosovellusten toteuttaminen

include <pthread.h>

- pthread_mutex_init(), _lock(), _trylock(), _unlock(), _destroy() _mutexattr_*(), ...
- pthread_rwlock_init(), _rwlock_rdlock(),_rwlock_tryrdlock(), _rwlock_wrlock(), _rwlock_trywrlock(), _rwlock_unlock(), _rwlock_destroy(), _rwlockattr_*(), ...

include <semaphore.h>

- sem_init(), sem_wait(), sem_trywait(), sem_post(), sem_getvalue(), sem_destroy(), ...

Java

Kurssi: Ohjelmointitekniikka (Java)

⇒ Lue man- / opastussivut

⇒ Andrews ch 4.6, 5.5

• ei semaforeja

• synchronized objects ⇒ oma toteutus?

Kertauskysymyksiä?