

# Monitorit

Monitori

Synkronointimenetelmiä

Esimerkkejä






# Tavoite

- Minimoi virhemahdollisuuksia
  - | poissulkeminen ohjelmoijan vastuulla
  - P():t ja V():t siellä, täällä ja tuolla - meniköhän oikein?
- Yksityiskohtia pois ohjelmoijalta kääntäjälle
  - | mitä yhteisiä muuttujia prosesseilla
  - | mikä semafori liittyy mihinkin kriittiseen alueeseen
  - | missä kohdassa kriittiset alueet sijaitsevat ohjelmakoodissa
- Kääntäjä voisi tuottaa koodia, jossa
  - | yhteiskäytön automaattinen kontrollointi
  - | yhteisiä muuttujia käytetään vain kriittisen alueen sisällä
  - | kriittiselle alueelle siirrytään ja sieltä poistutaan oikein
- mutta saattaa rajoittaa rinnakkaisuutta



# Semafori

- perusmekanismi synkronointiin
  - | voidaan systemaattisesti ratkaista synkronointiongelmia
- melko alhaisen tason mekanismi => helppo tehdä virheitä
  - | unohtaa jokin P- tai V-operaatio
  - | tehdä ylimääräisiä P- tai V-operaatioita
  - | käyttää väärää semaforia
  - | unohtaa suojata jokin kriittinen alue
- Globaaleja muuttujia
  - | Varmistuttava että kaikki osat yhdessä toimivat oikein
- samaa mekanismia käytetään sekä poissulkemiseen että ehtosynkronointiin
  - | Kunkin P- ja V-operaation tarkoitus?



● “Semaphores are like goto's and pointers: mistake prone, work okay but lack structure and ``discipline”.”

- | a disastrous typo: `V(S); criticalSection(); V(S)`
- | leads to deadlock: `P(S); criticalSection(); P(S)`
- | can lead to deadlock:

P1: `P(Q); P(S); ... V(S); V(Q);`

P2: `P(S); P(Q); ... V(Q); V(S);`

# Monitori

Hoare 1974

- Kapseloitu data + sitä käsittelevät operaatiot
  - | abstraktit objektit ja julkiset metodit
  - | Vain metodeilla voi käsitellä dataa
- Kaikki yhteiset, pysyvät muuttujat monitorin sisällä
- Tarjoaa automaattisesti poissulkemisen
  - | vain yksi monitorin aliohjelma kerrallaan aktiivinen
  - | muut prosessit voivat olla odottamassa - joko pääsyä monitoriin tai monitorin ehtomuuttujassa
  - | Kääntäjä huolehtii!
- Aktiivinen prosessi - passiivinen monitori
  - | Prosessi kutsuu monitorin proseduuria



# Hyötyjä:

- Kutsuvan prosessin ei tarvitse välittää siitä, kuinka monitorin aliohjelmat (proseduurit) on toteutettu.
- Monitorin toteuttajan ei tarvitse välittää siitä, missä ja miten sen proseduureja kutsutaan.
- => voidaan toteuttaa erikseen
- => rinnakkaisten ohjelmien tekeminen tulee helpommaksi, samoin niiden ymmärtäminen.



# Toteutus

- eri ohjelmointikielissä erilaiset määrittelyt
- Osa ohjelmointikieltä =>
  - kääntäjä osaa generoida monitorin toteutuksen koodin koodin
- poissulkemisen toteuttaminen:
  - ohjelmointikielessä, kirjaston avulla tai käyttöjärjestelmässä
    - | yhden prosessorin koneessa keskeytykset kieltämällä
    - | monen prosessorin koneessa lukoilla ja keskeytykset kieltämällä

## Esittely (Andrewsin kirjassa ja kurssilla käytetty)

```
monitor Mname {  
    pysyvien muuttujien määrittely  
    proseduurit  
    alustuslauseet  
}
```

- Monitori staattinen 'olio'
  - | Prosessi kutsuu monitorin prosedureja
  - | Monitorin muuttujien arvot pysyvät niin kauan kuin monitori on olemassa (permanent)

### Kutsu

call Mname.opname(arguments)

(Mname voidaan jättää pois, jos opname on yksikäsitteinen)





# Monitori on abstraktin datatyypin ilmentymä =>

- Vain proseduurien nimet (*opname*) näkyvät monitorin (*mname*) ulkopuolelle. Monitorin omat muuttujat käytössä vain monitorin sisällä
- Monitorissa ei voi viitata monitorin ulkopuolisiin muuttujiin, ainoastaan monitorissa määriteltyihin ja parametrina (*arguments*) saatuihin
- Monitorin pysyvät muuttujat alustetaan ennen kuin yhtäkään sen proseduuria kutsutaan eli heti monitorin luomisen yhteydessä suoritetaan alustuslauseet



# Prosessien synkronointi monitorilla

- poissulkeminen (mutual exclusion)

- | **implisiittistä**

- | vain yksi prosessi voi olla **aktiivisena** monitorissa eli suorittamassa jotain monitorin proseduuria
    - | ei useaa prosessia suorittamassa samaa proseduuria
    - | ei useaa prosessia suorittamassa eri proseduureja
  - | ohjelmoijan ei siis tarvitse huolehtia tästä

- ehtosynkronointi

- | ohjelmoijan vastuulla; vain ohjelmoija tietää, millaista synkronointia prosessien välillä tarvitaan
  - | täytyy ohjelmoida eksplisiittisesti **ehtomuuttujia** (condition variables) käyttäen

## Ehtomuuttujat ja operaatiot

### • **cond cv**

- | Ehtomuuttujalla ei arvoa - vain **jono** ehtoa cv odottaville Blocked-tilaan siirretyille prosesseille (**paikka** odotukselle, "odotushuone")

### • **wait(cv)**

- | laita prosessi jonoon **odottamaan** operaatiota *signal()*
- | prosessi joutuu **aina jonoon!**

### • **signal(cv)**

- | jos jono tyhjä, "no operation", ehtomuuttuja "ei muista"
- | jos jonossa prosesseja, herätä jonon ensimmäinen
- | "huuto odotushuoneeseen: Seuraava!"

### • **empty(cv)**

- | palauta true, jos jono on tyhjä

**vrt. semafori!**



- Monitorin käyttövuorot koodattava **eksplisiittisesti**

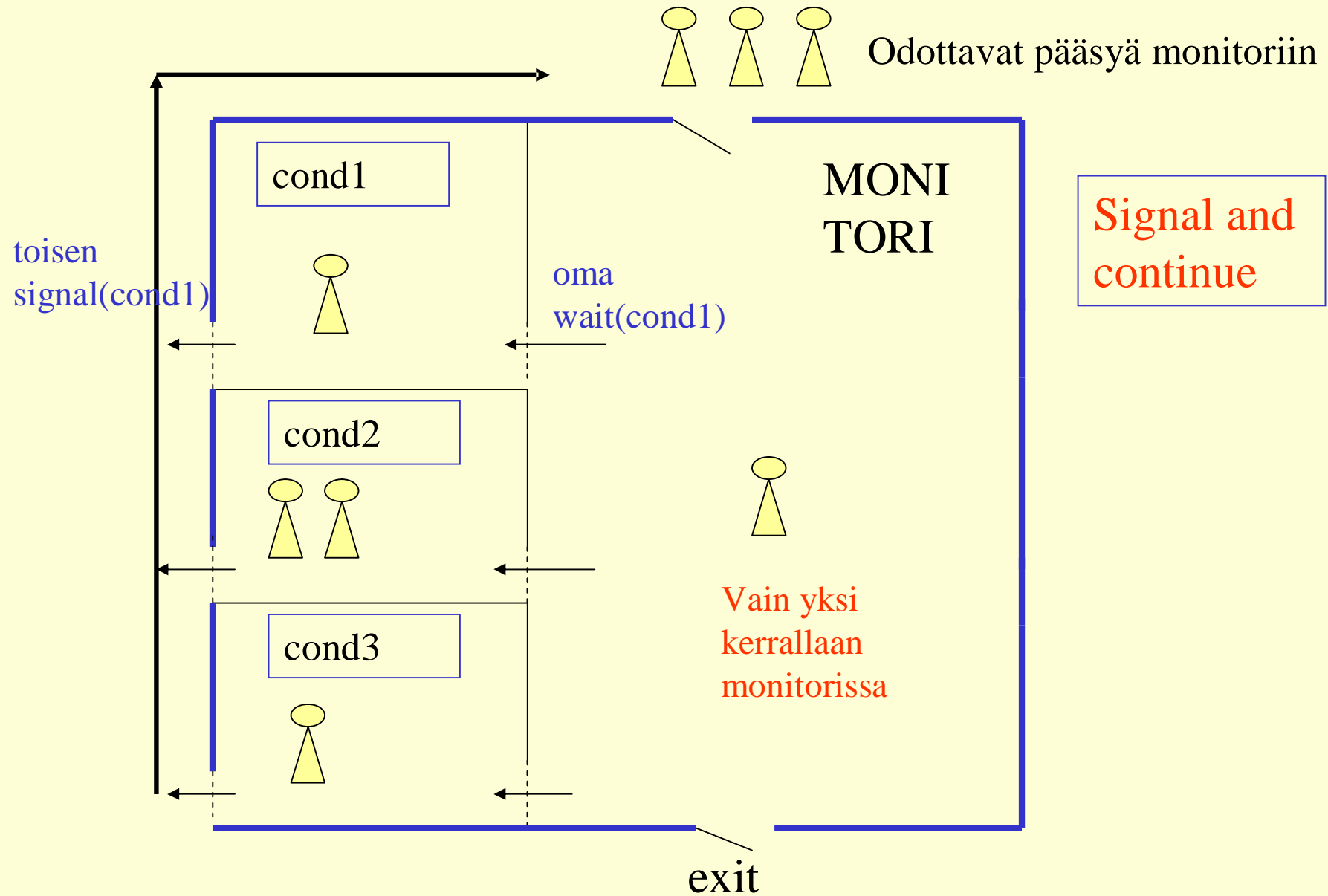
- synkronointi aina ohjelmoijan vastuulla

- | jos prosessi ei voi jatkaa monitorin sisällä, vapauta monitori muiden käyttöön: kutsu *wait(cv)*
  - | odotus tavallaan monitorin ulkopuolella, passiivisena!
- | kun odotukseen liittyvä ehto tulee todeksi, kutsu *signal(cv)*

- *signal()* herättää monitorin sisällä jo olleen toisen prosessin

- **Kumpi saa jatkaa proseduurissaan?**

- | Herättäjä?
- | Herätetty?



## Signaloinnin vaihtoehdot

- **Signal and Continue** (nonpreemptive)

- | signaloiija jatkaa, herätetty prosessi suoritukseen myöhemmin, kun aikanaan saa monitorin haltuunsa

- **Signal and (urgent) Wait** (preemptive)

- | signaloiija odottaa, herätetty saa jatkaa heti (= prosessin vaihto)
- | Signaloija saa jatkaa edenemistään ennen uusia monitoriin pyrkijöitä

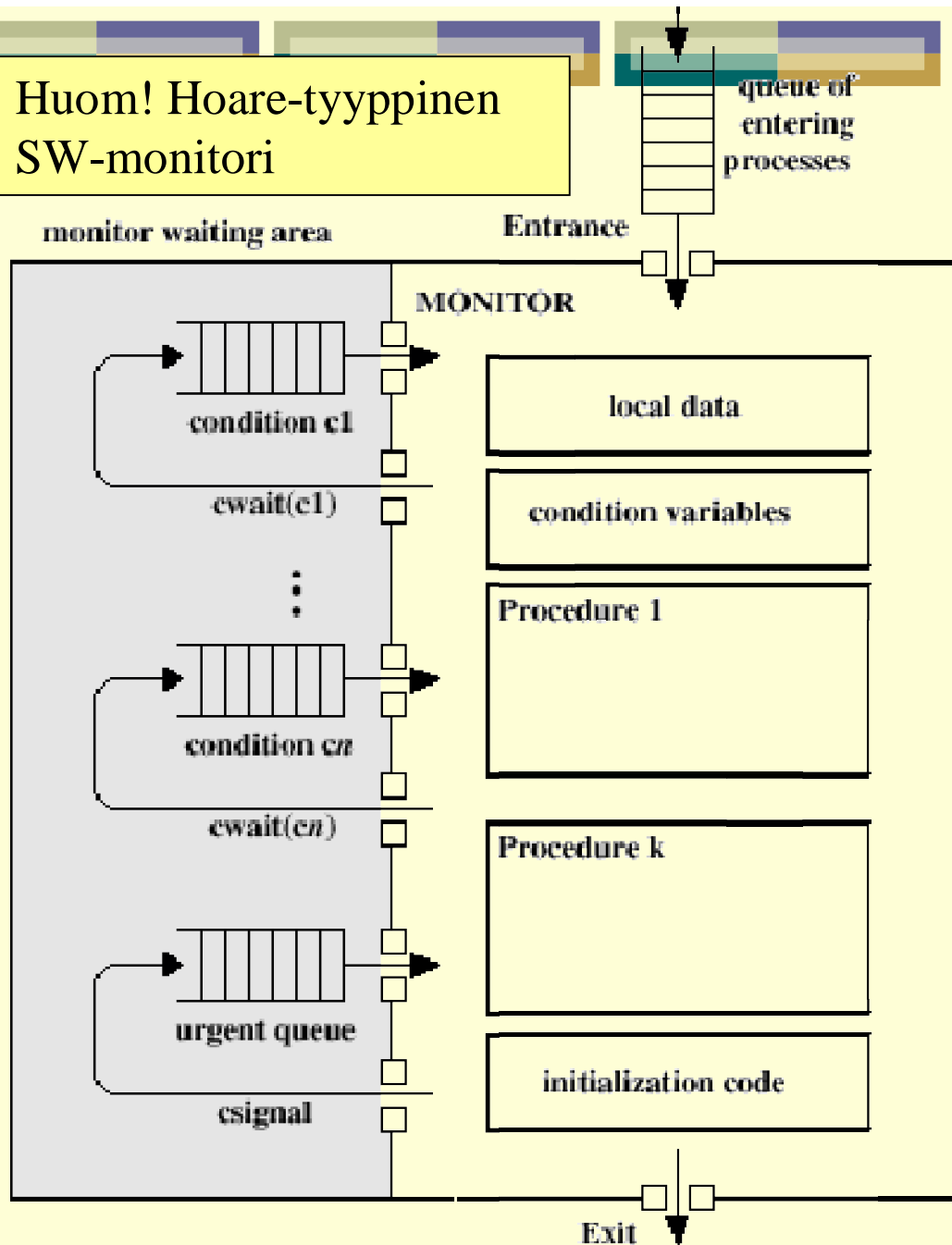
- **Odottavat prosessit ehtomuuttujan jonossa**

- | Odotus poissuljetun alueen ulkopuolella

- **Myös uudet prosessit kilpailemassa pääsystä monitorin sisälle**

- | Onko ehto enää true, kun herätetty pääsee jatkamaan?
- | Tarkastettava uudelleen! => while-silmukka

Huom! Hoare-tyyppinen SW-monitori



Monitorin muuttujat yhteiskäytössä.

Vain yksi prosessi kerrallaan suorittaa monitorin koodia.

Proseduureissa voi käyttää paikallisia muuttujia,

kullakin prosessilla niistä oma kopio (pinossa).

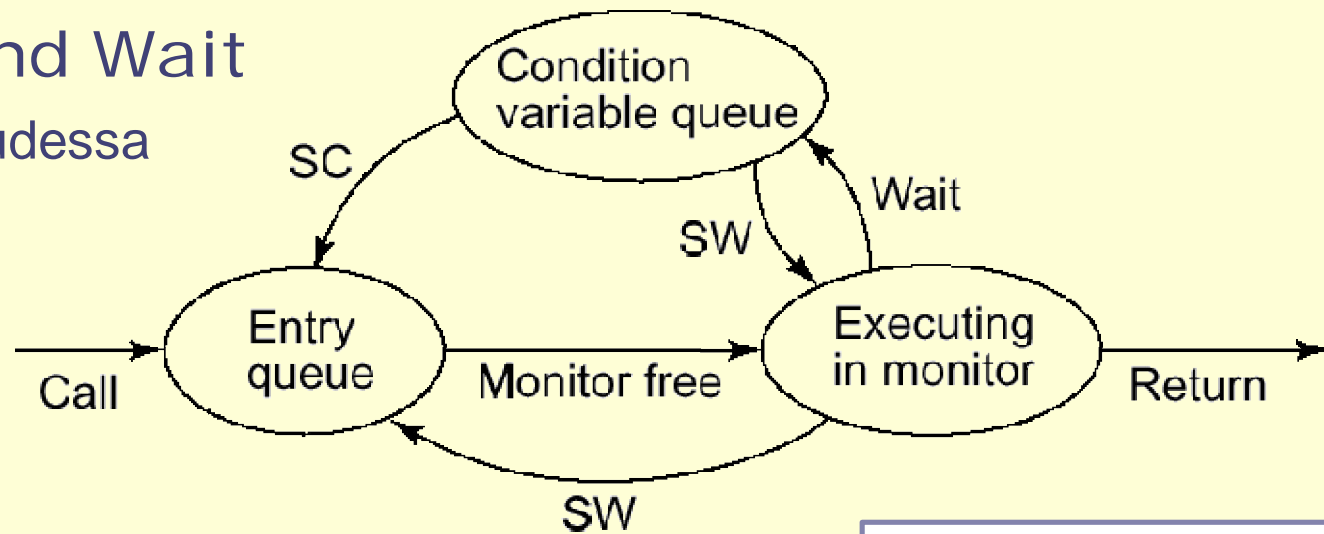
**Stallings Fig. 5.21:  
Structure of  
a Monitor.**

## Signal and Continue

- | Java, POSIX: pthreads-kirjasto, Andrews'n kirja ...
- | S&C vain vihje, että juuri silloin vaadittu ehto true, ehto ei kuitenkaan välttämättä enää ole voimassa, kun herätetty prosessi pääsee jatkamaan odotuskohdastaan  
○ tarkistettava ennenkuin voi jatkaa!

## Signal and Wait

- | kirjallisuudessa



Andrews Fig. 5.1.





- Miten saataisiin monitori toimimaan FIFO:na myös SC:tä käytettäessä?

- condition passing (vrt. paton passing)

- | Monitorissa oleva signaloiva prosessi herättää odottavan prosessin, mutta jättää ehdon muuttamatta

- | **if (!ehto) wait(ehto\_tosi) ....**

- | Uudet monitoriin pyrkijät menevät odottamaan, mutta monitorista vapautettu ei enää testaa ehtoa, vaan jatkaa eteenpäin

- | Ennen signalointia varmistettava, että kyseistä ehtomuuttajaa on joku odottamassa

- | **if (!empty(ehto\_tosi) signal (ehto\_tosi) else ehto=true;**

- | Muuten mikään tätä ehtoa testaava uusi tulija ei pääse monitoriin, vaan kaikki menevät odottamaan

## Synkronointi (Condition Synchronization)

```
monitor Bounded_Buffer {  
  
    typeT buf[n];      # an array of some type T  
    int front = 0,    # index of first full slot  
        rear = 0;     # index of first empty slot  
        count = 0;    # number of full slots  
    ## rear == (front + count) % n  
    cond not_full,    # signaled when count < n  
        not_empty;   # signaled when count > 0  
  
    procedure deposit(typeT data) {  
        while (count == n) wait(not_full);  
        buf[rear] = data; rear = (rear+1) % n; count++;  
        signal(not_empty);  
    }  
  
    procedure fetch(typeT &result) {  
        while (count == 0) wait(not_empty);  
        result = buf[front]; front = (front+1) % n; count--;  
        signal(not_full);  
    }  
}
```

```
process Producer[i=1 to N] {
    typeT data;
    while (true) {
        tuota data;
        call Bounded_Buffer.deposit(data);
    }
}
```

```
process Consumer[i=1 to M] {
    typeT data;
    while (true) {
        call Bounded_Buffer.fetch(data);
        kuluta data;
    }
}
```

## Prosessit monitorin ulkopuolella!

## Lisää monitorin operaatioita

- **wait (cv, rank)**

- | odota arvon mukaan kasvavassa järjestyksessä (priority wait)

- **minrank(cv)**

- | palauta jonon ensimmäisen prosessin arvo

- **signal\_all(cv)**

- | herätä kaikki ehtomuuttujassa cv odottavat prosessit
- | S&C: `while ( ! empty(cv)) signal(cv);`
- | S&W: ei kovin hyvin määritelty **miksei?**

**vrt. semafori!**

# Lukijat ja kirjoittajat monitorissa

## lukijat

usea lukija voi samanaikaisesti olla lukemassa

```
call RW.request_read;  
lue tietokantaa;  
call RW.release_read;
```

lukija-prosessit

## MONITORI RW

```
request_write  
release_write  
request_read  
release_read
```

tietokanta, johon kirjoitetaan ja josta luetaan

tietokanta on monitorin ulkopuolella, mutta sinne pääsee vain monitorin kautta

## kirjoittajat

kirjoittajalla yksinoikeus tietokantaan

```
call RW.request_write;  
kirjoita tietokantaan;  
call RW.release_write;
```

kirjoittaja -prosessit

```
monitor RW_Controller {
```

```
# paikallisten muuttujien määrittelyt
```

```
prosedure request_read( ) { .....}
```

```
prosedure release_read ( ) { ..... }
```

```
prosedure request_write ( ) { .....}
```

```
prosedure release_write ( ){.....}
```

```
}
```

```
process Reader [i=1 to M] {
```

```
....
```

```
while(true) {
```

```
call RW.request_read();
```

```
read_database;
```

```
call RW.release_read();
```

```
}}
```

```
process Writer [i=1 to N] {
```

```
....
```

```
while(true) {
```

```
call RM.request_write ( );
```

```
write_database;
```

```
call RM.release_write( )
```

```
}}
```

ehtomuuttujia:

**oktoread** kun ei kukaan kirjoittamassa,

**oktowitz** kun ei kukaan lukemassa tai kirjoittamassa

=> tarvitaan muuttujia:

nr = lukijoiden lukumäärä,

nw = kirjoittajien lukumäärä

```
procedure request_read {  
  while (nw >0) wait (oktoread);  
  nr = nr +1;  
}
```

Käytössä  
signal and  
continue

```
procedure request_write(){  
  while (nr>0 || nw>0) wait(oktowrite);  
  nw=nw+1;  
}
```

```
procedure release_read(){  
    nr = nr -1;  
    if (nr ==0) signal (oktowrite); # viimeinen lukija herättää  
}  
# yhden kirjoittajan
```

```
procedure release_write () {  
    nw = nw -1;  
# poistuva kirjoittaja herättää yhden kirjoittajan ja kaikki lukijat  
signal(oktowrite);  
signal_all (oktoread);  
}
```





# Lyhyin työ ensin


- herätys prioriteetin perusteella
  - | **wait(cv)** vie prosessin fifo-jonoon; herätys jonoonmenojärjestyksessä
  - | **wait(cv, rank)** vie prosessit rank-muuttujan mukaan nousevaan järjestykseen ja signal (cv) herättää pienimmällä rank-arvolla odottamaan menneen prosessin ensimmäiseksi
- procedure request (int time) {  
    if (free) free = false;  
    else **wait(turn, time);** }

## Prioriteetin mukaan jonotus (Priority Wait)

```
monitor Shortest_Job_Next {  
    bool free = true;  ## Invariant SJN: see text  
    cond turn;        # signaled when resource available  
  
    procedure request(int time) {  
        if (free)      jos while, niin kukaan ei pääse  
            free = false;  
        else           Odotus ajan (time) mukaisessa  
            wait(turn, time);      järjestyksessä  
    }  
  
    procedure release() {  
        if (empty(turn))  
            free = true  
        else  
            signal(turn);  
    }  
}
```

Condition passing:  
Pidä resurssi varattuna,  
anna varattuna seuraavalle prosessille!  
Ø Ei etuilla!

Andrews Fig. 5.6.




# Ajastinkello prosesseille

- kaksi operaatiota

- | **delay(interval) # viivyyttää kutsuvaa prosessia**  
**# interval määrän 'tikityksiä'**  
**# herätetään vasta kun haluttu aika**  
**#on kulunut**

**procedure delay(int interval){....}**

- | **kellon tikitys 'tick' # kasvattaa aikaa tietyin välein**  
**procedure tick() {....}**



# Eri tapoja toteuttaa

- aina kun kello tikittää, herätetään kaikki odottavat prosessit tarkastamaan itse, onko jo aika herätä  
= 'kattava herätys' (covering condition)
- herätetään vain ne, jotka todella ovat tarpeeksi 'nukkuneet' ja voivat jatkaa toimintaansa
  - | **prioriteettiodotus + tarkistus onko jo aika herätä**
- jokaisella odottavalla oma ehtomuuttuja
  - | **monimutkaisempi ratkaisu**

## "Kattava herätys" (Covering Condition)

```
monitor Timer {  
    int tod = 0;    ## invariant CLOCK -- see text  
    cond check;    # signaled when tod has increased  
  
    procedure delay(int interval) {  
        int wake_time;           Jokaiselle oma herätysaika!  
        wake_time = tod + interval;  
        while (wake_time > tod) wait(check);  
    }  
  
    procedure tick() {  
        tod = tod + 1;  
        signal all(check);  
    }  
}
```

Herätä kaikki odottajat - tarkistakoot itse, onko jatkamislupa edelleen voimassa!

## Priority Wait

```
monitor Timer {
  int tod = 0;    ## invariant CLOCK -- see text
  cond check;    # signaled when minrank(check) <= tod
  procedure delay(int interval) {
    int wake_time;
    wake_time = tod + interval;
    if (wake_time > tod) wait(check, wake_time);
  }
  procedure tick() {
    tod = tod+1;
    while (!empty(check) && minrank(check) <= tod)
      signal(check);
  }
}
```

Herätysajan mukaan järjestykseen

Ensimmäisen jonossa olevan arvo  
(=herätysaika)

Herätä vain ne, jotka voivat jatkaa!



## Synkronointi

### ☒ Priority wait

- | helppo ohjelmoida, tehokas ratkaisu
- | voi käyttää, jos odotusehdoilla staattinen **järjestys**

### • Covering condition

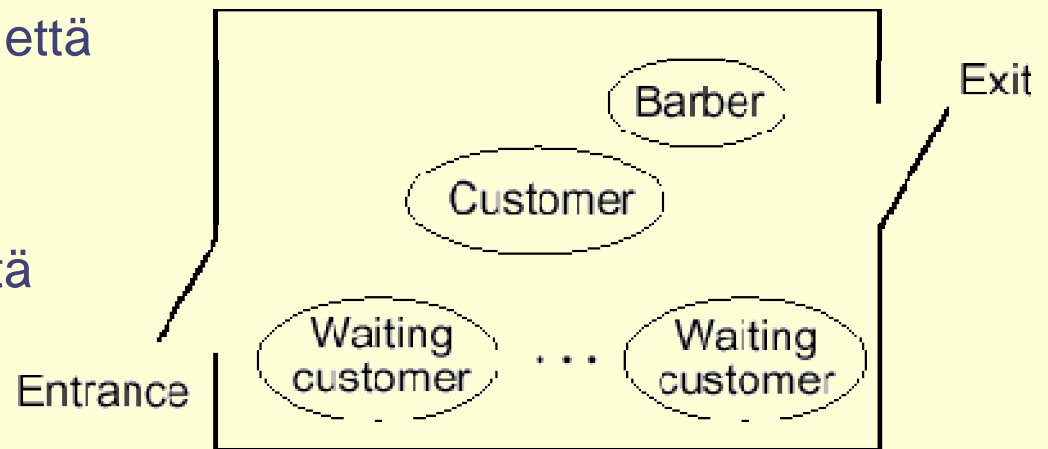
- | voi käyttää, jos herätetty prosessi voi **tarkistaa ehdon uudelleen**
- | ei voi käyttää, jos odotusehdot riippuvat myös muiden odottavien prosessien tiloista

Ž Jos minrank ei riitä odotuksen/vuorojen järjestämiseen, talleta yhteiset odotusehdot pysyviin muuttujiin ja jätä prosessit odottamaan yksityisiin ehtomuuttujiin

- | ohjelmoi itse jonon ylläpito (jonotusjärjestys)

## Rendezvous: Nukkuva parturi

- Useita aktiivisia prosesseja
- Rendezvous: “kahden prosessin kohtaaminen”
  - | vrt. puomisynkronointi
- Kutakin odotussyitä varten ehtomuuttuja ja laskuri
  - | rendezvous: uusi asiakas - vapaa parturi
  - | Asiakkaan odotettava, että
    - ⊕ parturi vapaa
      - ovi auki
  - | Parturin odotettava, että
    - ⊖ asiakas paikalla
      - asiakas poistuu





# Tapahtumien synkronointi

- Asiakkaan 'tärkeät' vaiheet (tapahtumat):
  - | **cinchair** eli istu tuoliin (asiakas on nyt kohtaauspaikalla)
  - | **cleave** eli poistu parturista
- Parturin vaiheet
  - | **bavail** eli vapaudu parranajoon
  - | **bbusy** eli aja partaa
  - | **bdone** eli asiakkaan parta ajettu

Ongelma:  
laskurit  
kasvavat  
koko ajan!

- Oikea järjestys laskurien avulla:

**bavail**  $\geq$  **cinchair**  $\geq$  **bbusy**  $\geq$  **bdone**  $\geq$  **cleave**

**barber** = **bavail** - **cinchair**

**open** = **bdone** - **cleave**

**chair** = **cinchair** - **bbusy**

Liisa Martinen

# Monitorin rakentelua

- `int barber = 0, int chair = 0, open = 0;`
- parturin ja asiakkaan toimintojen tahdistus ehtomuuttujilla:
  - | 4 eri tahdistuskohtaa
    - | asiakas odottaa parturin vapautuvan
    - | parturi odottaa asiakasta (= asiakas on tuolissa)
    - | asiakas odottaa poistumismerkkiä (= ovi auki)
    - | parturi odottaa asiakkaan poistumista
  - | kullekin oma ehtomuuttuja (**cond**)
    - | **barber\_available** # signalled when **barber > 0**
    - | **chair\_occupied** # signalled when **cinchair > 0**
    - | **door\_open** # signalled when **open > 0**
    - | **customer\_left** # signalled when **open == 0**

# asiakkaan käyttämä proseduuri

```
● procedure get_haircut() {
```

```
# odota tarvittaessa parturia
```

```
while (barber == 0) wait (barber_available);
```

```
# 'varaa parturi'
```

```
barber = barber - 1;
```

```
# istu tuoliin ja ilmoita olevasi paikalla
```

```
chair = chair + 1; signal(chair_occupied);
```

```
# odota parranajon valmistumista eli oven avautumista
```

```
while (open == 0) wait (door_open);
```

```
# sulje ovi ja ilmoita lähteneesi
```

```
open = open -1; signal (customer_left);
```



# Parturin käyttämät proseduurit

```
● procedure get_next_customer() {  
  # parturi vapautunut; ilmoittaa tästä mahdollisille odottajille  
  barber = barber + 1;  
  signal ( barber_available);  
  # odottaa asiakasta parturituoliin  
  while (chair == 0) wait (chair_occupied);  
  # tuoli merkataan vapaaksi (jo tässä!)  
  chair = chair - 1;  
}
```



● procedure finished\_cut() {

# avaa ovi ja ilmoita 'parranajon päättymisestä'

```
open = open +1;  
signal (door_open);
```

# odota asiakkaan poistumista

```
while (open > 0) wait (customer_left);
```

}

```

monitor Barber_Shop {
    int barber = 0, chair = 0, open = 0;
    cond barber_available;      # signaled when barber > 0
    cond chair_occupied;       # signaled when chair > 0
    cond door_open;            # signaled when open > 0
    cond customer_left;        # signaled when open == 0

    procedure get_haircut() {
        E while (barber == 0) wait(barber_available);
        barber = barber - 1;
        chair = chair + 1; signal(chair_occupied);
        • while (open == 0) wait(door_open);
        open = open - 1; signal(customer_left);
    }

    procedure get_next_customer() {
        barber = barber + 1; signal(barber_available);
        Z while (chair == 0) wait(chair_occupied);
        chair = chair - 1;
    }

    procedure finished_cut() {
        open = open + 1; signal(door_open);
        • while (open > 0) wait(customer_left);
    }
}

```

### Systemaatt. ratkaisu

resurssi: muuttuja  
 varaus: vähennä (-)  
 vapautus: lisää (+)  
 varo muita: while

Andrews Fig. 5.10.

```
process Barber {
    typeT data;
    while (true) {
        call Barber_Shop.get_next_customer();
        ... parturoi ...
        call Barber_Shop.finished_cut()
    }
}
```

```
process Customer[i=1 to M] {
    while (true) {
        .. tee sitä ja tätä ...
        call Barber_Shop.get_haircut();
    }
}
```



## POSIX-kirjasto, pthread

```
# include <pthread.h>
```

- | ehtomuuttujat
- | käyttö yhdessä mutexin kanssa  $\bar{O}$  monitori
- | pthread\_cond\_init(), \*\_wait(), \*\_signal(), \*\_broadcast(), \*\_timedwait(), \*\_destroy()

## Java, synchronized methods

- | automaattinen poissulkeminen  $\bar{O}$  monitori
- | ei ehtomuuttujia, yksi implisiittinen odotusjono / objekti
- | operaatiot wait(), notify(), notifyAll()

**+ paljon muita hieman toisistaan eroavia monitorikonstruktioita ja toteutuksia. Käytettäessä otettava huomioon kunkin toteutuksen erityispiirteet.**