

Monitorit

Monitori
Synkronointimenetelmiä
Esimerkkejä

Andrews 5.1-5.2, Stallings 5.5

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 siirytään ja sieltä poistutaan oikein
- **mutta saattaa rajoittaa rinnakkaisuutta**

Rio 2004 / Auvo Häkkinen

5 - 2

Monitori

Rio 2004 / Auvo Häkkinen

5 - 3

Monitori

Hoare 1974

- **Kapseloi datan + käsittelevät operaatiot**
 - abstraktit objektit ja julkiset metodit
- **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!
- **Aktiivinen prosessi - passiivinen monitori**

Rio 2004 / Auvo Häkkinen

5 - 4

Esittely

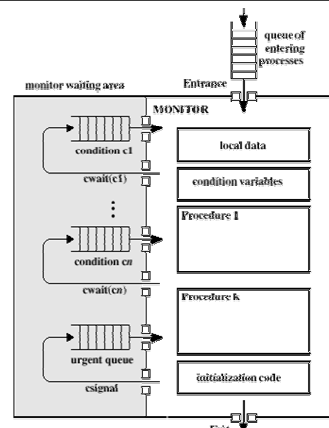
```
monitor Mname {  
    pysyvien muuttujien määrittely  
    proseduurit  
    alustuslauseet  
}
```

- **Monitori staattinen 'olio'**
 - Prosessi kutsuu monitorin prosedureja
 - Monitorin muuttujien arvot pysyvät niinkauan kuin monitori olemassa (permanent)

Kutsu
call Mname.opname(arguments)

Rio 2004 / Auvo Häkkinen

5 - 5



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.

Ehtomuuttujat ja operaatiot

- **cond cv**
 - ei arvoa - vain jono Blocked prosesseja (paikka odotukselle)
- **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
- **empty(cv)**
 - palautaa 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!
 - 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

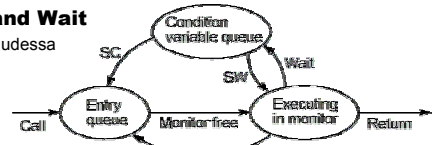
- **Signal and Continue** (nonpreemptive)
 - *signal*loija jatkaa, herätetty suoritukseen myöhemmin
- **Signal and Wait** (preemptive)
 - *signal*loija odottaa, herätetty saa jatkaa heti (= prosessin vaihto)
 - ks. Fig 5.21: signaler into urgent queue
- **Odottavat prosessit ehtomuuttujan jonossa**
 - Odotus poissuljetun alueen ulkopuolella
 - ks. Fig 5.21 monitor waiting area
- **Myös uudet prosessit kilpailemassa pääsystä monitorin sisälle**
 - Onko ehto enää true, kun herätetty pääsee jatkamaan?

Signal and Continue

- Java, POSIX: pthreads-kirjasto, Andrewsin kirja (tämä kurssi)...
- S&C vain vihje, että juuri silloin vaadittu ehto true, ehto ei kuitenkaan välttämättä enää voimassa, kun herätetty prosessi pääsee jatkamaan odotuskohdastaan
 - ⇒ kenties tarkistettava ennenkuin voi jatkaa!

Signal and Wait

- kirjallisuudessa



Andrews Fig. 5.1.

Esim: Semaforin toteutus monitorin avulla

- **Andrews Fig. 5.2**
 - Signal and Wait: FIFO semaphore
 - ehto varmasti voimassa, kun herätetty suoritukseen
 - Signal and Continue: toiminnallisuus ei täysin kiinnitetty
 - muuttujien arvot saattaneet muuttua uudelleen, ennenkuin herätetty pääsee suoritukseen
- **Andrews Fig 5.3:**
 - FIFO semaphore using passing the condition
 - Odotettu ehto voimassa, 'ojenna' se sellaisenaan herätettävälle
 - ehto varmasti voimassa, kun herätetty jatkaa
 - muille prosesseille ehto ei ole voimassa

```
monitor Semaphore {
  int s = 0; ## s >= 0
  cond pos; # signaled when s > 0
  procedure Psem() {
    while (s == 0) wait(pos);
    s = s-1;
  }
  procedure Vsem() {
    s = s+1;
    signal(pos);
  }
}
```

Andrews Fig. 5.2.

```
monitor FIFOsemaphore {
  int s = 0; ## s >= 0
  cond pos; # signaled when s > 0
  procedure Psem() {
    if (s == 0)
      wait(pos);
    else
      s = s-1;
  }
  procedure Vsem() {
    if (empty(pos))
      s = s+1;
    else
      signal(pos); condition passing
  }
}
```

Andrews Fig. 5.3.

Synkronointi

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);
    }
}
```

virt. 4.5

Andrews Fig. 5.4.

```
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) {
        Bounded_Buffer.fetch(data);
        kuluta data;
    }
}
```

Prosessit monitorin ulkopuolella! Miksi?

Lisää 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 määriteltä

miksei?

virt. semafori!

Kalkkien herätys (Broadcast Signal)

```
monitor RW_Controller {
    int nr = 0, nw = 0; ## (nr == 0 ∨ nw == 0) ∧ nw ≤ 1
    cond oktoread;    # signaled when nw == 0
    cond oktowitz;    # signaled when nr == 0 and nw == 0

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

    procedure release_read() {
        nr = nr - 1;
        if (nr == 0) signal(oktowitz); # awakes one writer
    }

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

    procedure release_write() {
        nw = nw - 1;
        signal(oktoread); # awakes one reader and
        signal_all(oktoread); # all readers
    }
}
```

Huom:

DB ei
monitorin
sisällä!
Miksei?

virt. 4.13

Andrews Fig. 5.5.

Prioriteetin mukaan jonotus (Priority Wait)

```
monitor Shortest_Job_Next {
    bool free = true; ## Invariant Δ/N: see text
    cond turn;        # signaled when resource available

    procedure request(int time) {
        if (free)
            free = false;
        else
            wait(turn, time);
    }

    procedure release() {
        if (empty(turn))
            free = true;
        else
            signal(turn);
    }
}
```

Condition passing:

Pidä resurssi varattuna,
anna varattuna seuraavalle prosessille!
⇒ Ei etuulia!

virt. 4.14

Andrews Fig. 5.6.

"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;
    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!

Andrews Fig. 5.7.

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);
  }
}

```

Herätä vain ne, jotka voivat jatkaa!

Andrews Fig. 5.8.

Synkronointi

1 Priority wait

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

2 Covering condition

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

3 Jos minrank ei riitä odotuksen/vuorojen järjestämiseen, talleta yhteiset odotusehdot pysyvästi muuttujiin ja jätä prosessit odottamaan yksittäisiin ehtomuuttujiin

- ohjelmoi itse jonon ylläpito (jonotusjärjestys)

Rendezvous: Nukkuva parturi

Useita aktiivisia prosesseja

Rendezvous: "kahden prosessin kohtaaminen"

- vrt. puomisyntronointi

Kutakin odotussyytä varten ehtomuuttuja ja laskuri

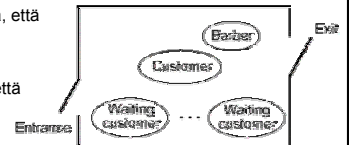
- rendezvous: uusi asiakas - vapaa parturi

- Asiakkaan odotettava, että

- 1 parturi vapaa
- 2 ovi auki

- Parturin odotettava, että

- 3 asiakas paikalla
- 4 asiakas poistuu



```

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

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

  procedure finished_cut() {
    open = open + 1; signal(door_open);
    4 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>

- ehtomuuttajat
- käyttö yhdessä mutexin kanssa ⇨ monitori
- pthread_cond_init(), *_wait(), *_signal(), *_broadcast(), *_timedwait(), *_destroy()

Java, synchronized methods

- automaattinen poissulkeminen ⇨ monitori
- ei ehtomuuttajia, yksi implisiittinen odotusjono / objekti
- operaatiot wait(), notify(), notifyAll()

⇨ Lue man- / opastussivut
⇨ Andrews ch 5.4, 5.5

Kertauskysymyksiä?