

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

5.6.2006 Liisa Marttinen 3-2

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
 - | Varmistettava että kaikki osat yhdessä toimivat oikein
- samaa mekanismia käytetään sekä poissulkemiseen että ehtosynkronointiin
 - | Kunkin P- ja V-operaation tarkoitus?

5.6.2006 Liisa Marttinen 3-3

• “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);
```

5.6.2006 Liisa Marttinen 3-4

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

5.6.2006 Liisa Marttinen 3-5

Hyötyjä:

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

5.6.2006 Liisa Marttinen 3-6

Toteutus

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

5.6.2006

Liisa Marttinen

3 - 7

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)
```

5.6.2006

Liisa Marttinen

3 - 8

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

5.6.2006

Liisa Marttinen

3 - 9

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 prosedureja
 - ohjelmoijan ei siis tarvitse huolehtia tästä
 - **ehtosynkronointi**
 - ohjelmoijan vastuulla; vain ohjelmoija tietää, millaista synkronointia prosessien välillä tarvitaan
 - täytyy ohjelmoida eksplisiittisesti **ehtomuuttujia** (condition variables) käyttäen

5.6.2006

Liisa Marttinen

3 - 10

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)**
 - palautaa true, jos jono on tyhjä

vrt. semafori!

5.6.2006

Liisa Marttinen

3 - 11

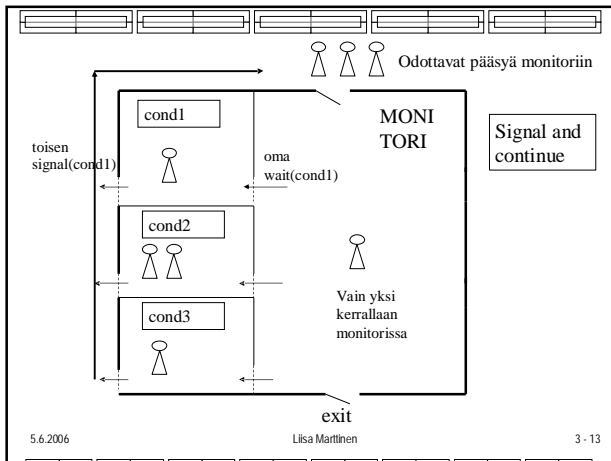
Monitorin käyttövuorot koodattava eksplisiittisesti

- 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?

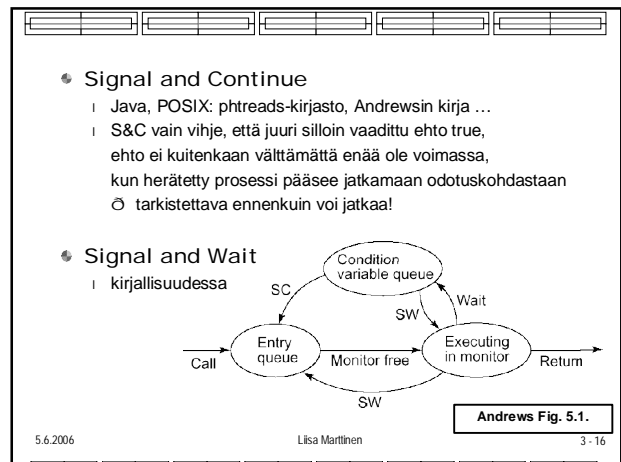
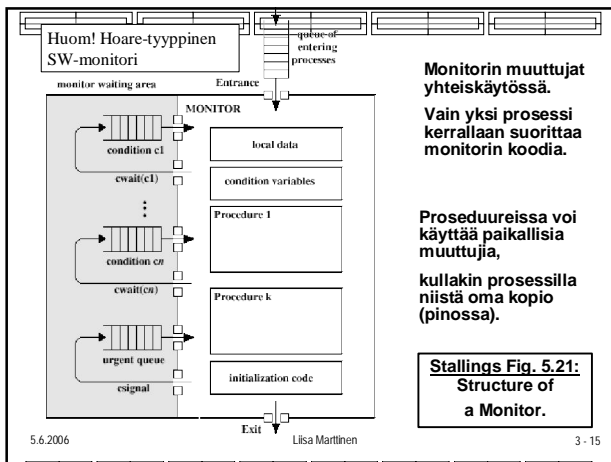
5.6.2006

Liisa Marttinen

3 - 12



- ### Signaloinnin vaihtoehdot
- Signal and Continue (nonpreemptive)
 - signaloija jatkaa**, herätetty prosessi suoritukseen myöhemmin, kun aikanaan saa monitorin haltuunsa
 - Signal and (urgent) Wait (preemptive)
 - signaloija 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
- 5.6.2006 Lisa Marttinen 3 - 14



- 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 (lehto) 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 (lempy(ehto_tosi) signal (ehto_tosi) else ehto=true;**
 - Muuten mikään tätä ehtoa testaava uusi tulija ei pääse monitoriin, vaan kaikki menevät odottamaan
- 5.6.2006 Lisa Marttinen 3 - 17

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);
    }
}
  
```

5.6.2006 Lisa Marttinen 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) {
    call Bounded_Buffer.fetch(data);
    kuluta data;
  }
}

```

Prosessit monitorin ulkopuolella!

5.6.2006 Liisa Marttinen 3 - 19

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 määriteltä miksei?

vrt. semafori!

5.6.2006 Liisa Marttinen 3 - 20

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

5.6.2006 Liisa Marttinen 3 - 21

```

monitor RW_Controller {
  # paikallisten muuttujien määrittelyt

  procedure request_read() { ..... }
  procedure release_read () { ..... }
  procedure request_write () { ..... }
  procedure 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 RW.request_write ();
    write_database;
    call RW.release_write ();
  }
}

```

5.6.2006 Liisa Marttinen 3 - 22

ehtomuuttujia:

oktoread kun ei kukaan kirjoittamassa,
oktowrite 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;
}

```

5.6.2006 Liisa Marttinen 3 - 23

```

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);
}

```

5.6.2006 Liisa Marttinen 3 - 24

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

5.6.2006

Liisa Marttinen

3 - 25

Prioriteetin mukaan jonotus (Priority Wait)

```
monitor Shortest_Job_Next {
  bool free = true; ## Invariant S/N: see text
  cond turn;      # signaled when resource available
  procedure request(int time) {
    if (free)
      free = false; jos while, niin kukaan ei pääse
    else
      wait(turn, time); Odotus ajan(time) mukaisessa
                        järjestyksessä
  }
  procedure release() {
    if (empty(turn))
      free = true; Condition passing:
    else
      signal(turn); Pidä resurssi varattuna,
                    anna varattuna seuraavalle prosessille!
                    Ö Ei etuilla!
  }
}
```

Andrews Fig. 5.6.

5.6.2006

Liisa Marttinen

3 - 26

Ajastinkello prosesseille

- kaksi operaatiota
 - delay(interval)** # viivyttää 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() {...}

5.6.2006

Liisa Marttinen

3 - 27

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

5.6.2006

Liisa Marttinen

3 - 28

"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!

5.6.2006

Liisa Marttinen

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); Ensimmäisen jonossa olevan arvo
                    (=herätysaika)
  }
}
```

Herätä vain ne, jotka voivat jatkaa!

5.6.2006

Liisa Marttinen

Andrews Fig. 5.8.

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 yksittäisiin ehtomuuttujiin**

- ohjelmoi itse jonon ylläpito (jonotusjärjestys)

5.6.2006

Liisa Marttinen

3 - 31

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

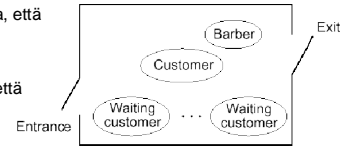
☒ parturi vapaa

- ovi auki

- Parturin odotettava, että

Ž asiakas paikalla

- asiakas poistuu



5.6.2006

Liisa Marttinen

3 - 32

Tapahtumien synkronointi

☒ Asiakkaan 'tärkeät' vaiheet (tapahtumat):

- cinchair** eli istu tuoliin (asiakas on nyt kohtauspaikalla)
- 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 >= cinchair >= bbusy >= bdone >= cleave

barber = bavail - cinchair

open = bdone - cleave

chair = cinchair - bbusy

5.6.2006

Liisa Marttinen

3 - 33

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

5.6.2006

Liisa Marttinen

3 - 34

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);
```

5.6.2006

Liisa Marttinen

3 - 35

Parturin käyttämät proseduurit

☒ procedure get_next_customer() {

parturi vapautunut; ilmoittaa tästä mahdollisille odottajille

```
barber = barber + 1;
```

```
signal (barber_available);
```

odottaa asiakasta parturituihin

```
while (chair == 0) wait (chair_occupied);
```

tuoli merkataan vapaaksi (jo tässä!)

```
chair = chair - 1;
```

```
}
```

5.6.2006

Liisa Marttinen

3 - 36

```

• procedure finished_cut() {

# avaa ovi ja ilmoita 'parranajon päättymisestä'

    open = open + 1;
    signal (door_open);

# odota asiakkaan poistumista

    while (open > 0) wait (customer_left);

}

```

5.6.2006 Liisa Marttinen 3 - 37

```

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() {
        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);
        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.

5.6.2006 Liisa Marttinen 3 - 37

```

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();
    }
}

```

5.6.2006 Liisa Marttinen 3 - 39

POSIX-kirjasto, pthread

```

# include <pthread.h>
    | ehtomuuttujat
    | käyttö yhdessä mutexin kanssa Ø monitori
    | pthread_cond_init(), *_wait(), *_signal(), *_broadcast(),
    | *_timedwait(), *_destroy()

```

Java, synchronized methods

```

    | automaattinen poissulkeminen Ø monitori
    | ei ehtomuuttujia, yksi implisiittinen odotusjono / objekti
    | operaatiot wait(), notify(), notifyAll()

```

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

5.6.2006 Liisa Marttinen 3 - 40