

OSA II:

Hajautettu ympäristö **Ei yhteistä muistia**

Sisältö, osa II

- ⑥ **Sanomanvälitys**
- ⑦ **Etäproseduurikutsu**
- ⑧ **Rendezvous**

⑥ Sanomanvälitys

Käsitteistöä

Kanavat

Asiakkaat ja Palvelijat

Kommunikointitapoja

Andrews 7.1-7.6, Stallings 5.6, 13.1-13.2

Käsitteistöä

Kommunikointi

Prosessi A

```
...  
X=f(..);  
send X to B  
...
```

X: 10

KJ:n send DC
ydin

Prosessi B

```
...  
receive X from A  
Y=f(X);  
...
```

X: 10

DC **receive** KJ:n
ydin

kanava

Sanomanvälitys

● Käyttö

- prosessien välinen tiedonvälitys
- synkronointi: vuorot / ajoitus

● Käyttäjätason operaatiot

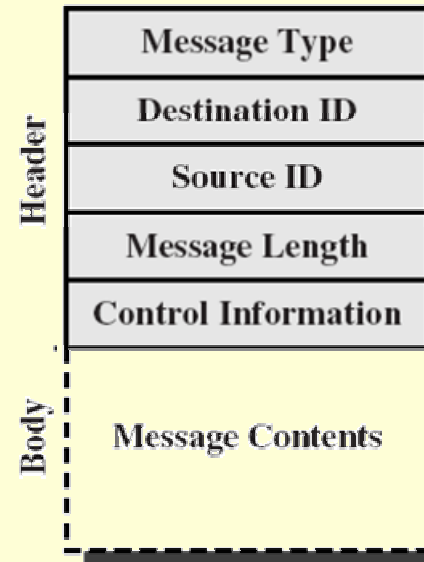
- lähetys **send (receiver-id, msg)**
- vastaanotto **receive([sender-id,] msg)**

● Matkassa myös KJ:n lisäämä tieto

- ainakin: sender-id, receiver-id, sanoma, ...

● Toteutus?

- pistokkeet: UDP/TCP, ...



Odotussementtiikka

● **Send**

- jatka heti, kun KJ kopioinut puskuriinsa **tai**
- odota kunnes vastaanottaja saanut sanoman

● **Receive**

- odota kunnes KJ kopioinut prosessin muuttujiin **tai**
- jos kanava tyhjä, jatka välittömästi
 - jos ei sanomaa ~“no operation”

● **Blocking, non-blocking**

Synkronointi

send

receive

	blocking	nonblocking
blocking	synkroninen	asynkroninen
nonblocking	asynkroninen	asynkroninen

Synkroninen

- lähettäjä tietää, että vastaanottaja on saanut sanoman
- toimintojen suoritusjärjestys selvä

Asynkroninen

- toimintojen tarkka suoritusjärjestys epäselvä

Andrews: asynkroninen tiedonvälitys

send non-blocking, receive blocking

Kanavat

Sitominen

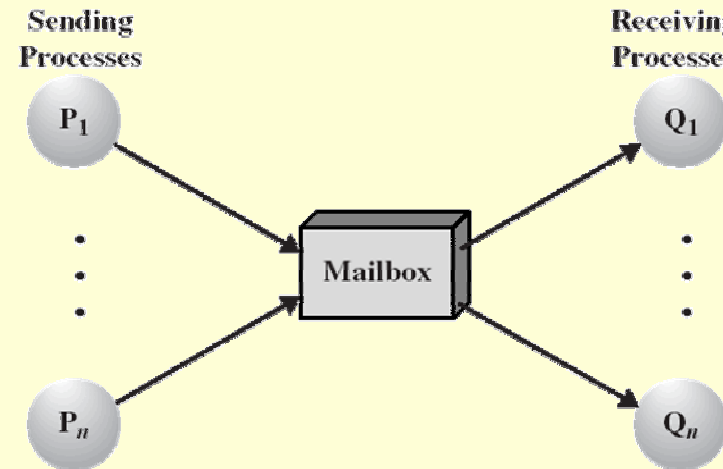
● **Lähettäjä** ↔ **Kanava** ↔ **Vastaanottaja**

● **Kanava**

- tiedonsiirtoyhteys prosessien välillä
(point-to-point ↔ väylä)

● **Sanomat**

- send kanavaan, jonka tuntee yksi
 - unicast (yksittäislähetys)
- tai useampi prosessi
 - multicast (monilähetys)
 - broadcast (yleislähetys)



● Sitomistapoja

- one-to-one (kaksi prosessia, yksi kummankin tuntema kanava)
- many-to-one (asiakkaat-palvelija)
- one-to-many, many-to-many
(asiakas - palvelut, ryhmäkommunikointi)

● Andrews (tämä kurssi)

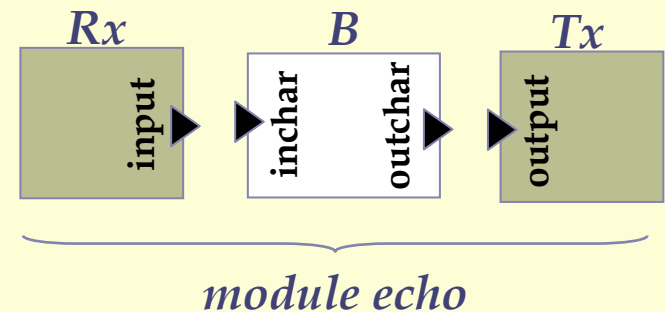
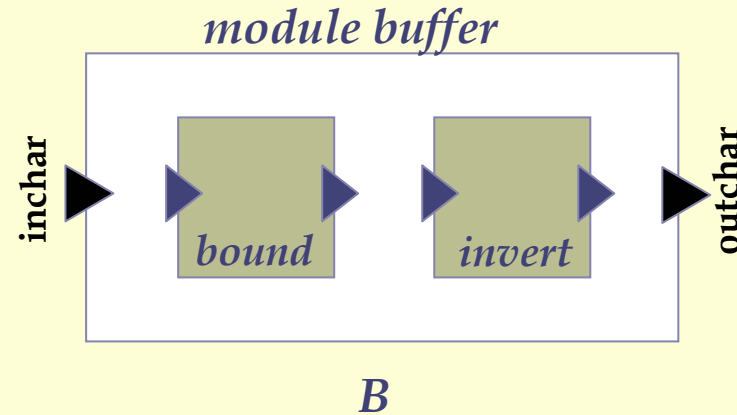
- one-to-one tai many-to-one
- luotettava, järjestyksen säilyttävä sanomanvälitys
- prosessi(parien) sitominen: ***nimetty kanava***
- ei monilähetystä
- ei yleislähetystä

- Huom: esimerkit eivät ota kantaa sitomiseen

Explisiittinen sitominen (Sloman, Kramer fig 3.9, 3.10)

```
GROUP MODULE buffer;
  ENTRYPORT inchar: char REPLY signaltype;
  EXITPORT  outchar: char REPLY signaltype;
  USE bound, invert;
  CREATE bound, invert;
  LINK  inchar TO bound.getchar;
        invert.getchar TO bound.putchar;
        invert.outchar TO outchar;
END.
```

```
GROUP MODULE echo;
  CONST status=177560#8;
        vector=100#8;
  USE serial-input, serial-output, buffer;
  CREATE Rx: serial-input(status,vector);
        Tx: serial-output(status+4, vector+4)
        B: buffer;
  LINK Rx.input TO B.inchar;
        B.outchar TO Tx.output;
END.
```



Kanavat (Andrews)

• Yhteinen 'postilaatikko'

- jono sanomia, FIFO
- kaikki kanavan sanomat rakenteeltaan samanlaisia

• **chan ch(type₁ id₁, ..., type_n id_n)**

- ch: kanavan nimi
- type_i id_i: sanoman osien tyypit, ja nimet (saavat puuttua)

• **Esim.**

- chan **input**(char);
- chan **disk_access** (int **cylinder**, int **block**, int **count**, char* **buffer**);
- chan **result[n]** (int); # *kanavien taulukko*

Operaatiot

- **send** kanava(lauseke₁, ... , lauseke_n)
 - lähetä sanoma kanavaan
- **receive** kanava(muuttuja₁, ... , muuttuja_n)
 - vastaanota sanoma kanavasta
- **empty(kanava)**
 - tarkista onko kanava tyhjä sanomista
- **Esim.**
 - send disk_access(cylinder+2, block, count, buf)
 - receive result[i](sum)
 - empty(input)

Ei ota kantaa minkä prosessin kanssa kommunikoi!

Suodin: Merkit riveiksi

```
chan input(char), output(char [MAXLINE]);
process Char_to_Line {
  char line[MAXLINE]; int i = 0;
  while (true) {
    receive input(line[i]);
    while (line[i] != CR and i < MAXLINE) {
      # line[0:i-1] contains the last i input characters
      i = i+1;
      receive input(line[i]);
    }
    line[i] = EOL;
    send output(line);
    i = 0;
  }
}
```

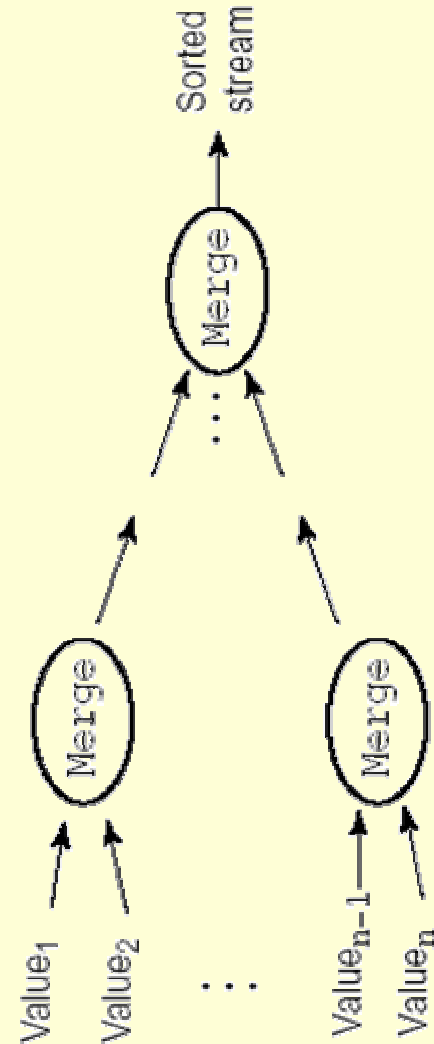
Yhteiskäyttöinen kanava (globaali)

esittely prosessien ulkopuolella (~ sitominen)

Andrews Fig. 7.1.

Suodin: Iomita kaksi syöttökanavaa

```
chan in1(int), in2(int), out(int);
process Merge {
  int v1, v2;
  receive in1(v1); # get first two input values
  receive in2(v2);
  # send smaller value to output channel and repeat
  while (v1 != EOS and v2 != EOS) {
    if (v1 <= v2)
      { send out(v1); receive in1(v1); }
    else # (v2 < v1)
      { send out(v2); receive in2(v2); }
  }
  # consume the rest of the non-empty input channel
if (v1 == EOS)
  while (v2 != EOS)
    { send out(v2); receive in2(v2); }
else # (v2 == EOS)
  while (v1 != EOS)
    { send out(v1); receive in1(v1); }
  # append a sentinel to the output channel
  send out(EOS);
}
```



Andrews Fig. 7.2.



Asiakkaat ja Palvelija

Kommunikointitapoja

Osapuolet

- yksittäinen prosessi ⇔ ryhmä
- nimetty partneri ⇔ kuka tahansa

Tarpeita

- kahdenkeskinen, peer to peer: A ⇔ B
- monilähetys, yleislähetys
- asiakkaat ⇔ palvelija, kuka tahansa ⇔ FileServer
- asiakas ⇔ palvelu
 - mikä tahansa palvelijaprosessi
 - muut palvelijaprosessit palvelevat toisia prosesseja

Asiakkaat ja yhden palvelun palvelija

```
chan request(int clientID, types of input values);
chan reply[n] (types of results);

process Server {
  int clientID;
  declarations of other permanent variables;
  initialization code;
  while (true) {    ## loop invariant MI
    receive request(clientID, input values);
    code from body of operation op;
    send reply[clientID] (results);
  }

  process Client[i = 0 to n-1] {
    send request(i, value arguments);    # "call" op
    receive reply[i] (result arguments);  # wait for reply
  }
}
```

yhteinen pyyntökanava, yksityiset vastauskanavat

Andrews Fig. 7.4.

Asiakkaat ja monen palvelun palvelija

Pyyntökanava

- Asiakkaiden tuntema julkinen kanava
 - käytännössä: IP-osoite ja porttinumero
- Yksi pyyntökanava
 - *sanoma kanavaan* ⇒ *tulkitse tyyppi, valitse palvelu*
 - *tai* dedikoitu kanava kullekin palvelulle
 - *valitse palvelu* ⇒ *lähetä sopivaan kanavaan*

Vastauskanava

- Palvelijan tuntema (staattinen)
 - Jokaisella asiakkaalla oma yksityinen
 - kerro oma identiteetti pyyntösanomassa
- Asiakas kertoo pyynnössä (dynaaminen)
 - käytännössä: oma IP-osoite ja porttinumero

[Tällä kurssilla]

```

type op_kind = enum(op1, ..., opn);
type arg_type = union(arg1, ..., argn);
type result_type = union(res1, ..., resn);
chan request(int clientID, op_kind, arg_type);
chan reply[n](res_type);

process Server {
  int clientID; op_kind kind; arg_type args;
  res_type results; declarations of other variables;
  initialization code;
  while (true) {      ## loop invariant MI
    receive request(clientID, kind, args);
    if (kind == op1)
      { body of op1; }
    ...
    else if (kind == opn)
      { body of opn; }
    send reply[clientID](results);
  }
}

process Client[i = 0 to n-1] {
  arg_type myargs; result_type myresults;
  place value arguments in myargs;
  send request(i, opj, myargs);      # "call" opj
  receive reply[i](myresults);      # wait for reply
}

```

Asiakkaat ja monen palvelun palvelija

Aterioivat filosofit, yksi syöttökanava

```
type opType = enum(REQ, REL);  
chan phone(int, opType), reply[5]( );
```

```
process filosofer[i=0 to 4] {  
  while (true) {  
    think();  
    send phone(i, REQ);  
    receive reply[i]();  
    eat();  
    send phone(i, REL);  
  }  
}
```

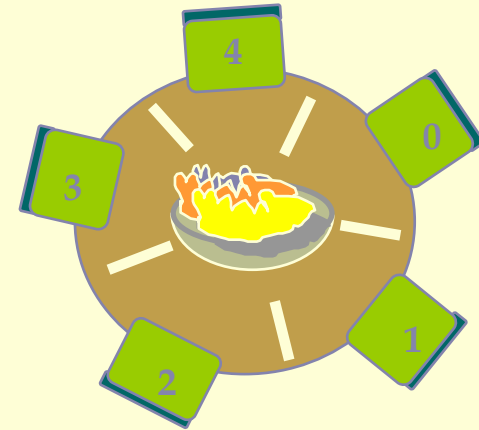
Keskitetty resurssien hallinta:

- "*public phone number*", *many-to-one*
- "*secret phone number*", *one-to-one*

```
process secretary {  
  while (true) {  
    receive phone(philo, what);  
    switch (what) {  
      case REQ: {  
        state[philo]=HUNGRY;  
        consider_allocation_to(philo);  
      }  
      case REL: {  
        state[philo]=THINKING;  
        consider_allocation_to(philo-1);  
        consider_allocation_to(philo+1);  
      }  
    }  
  }  
}
```

```
type activity = enum(THINKING, HUNGRY, EATING);
activity state[5] = ([4] THINKING);
```

```
procedure consider_allocation_to(int i) {
  if ( state[i] == HUNGRY)
    if (state[i-1] != EATING AND state[i+1] != EATING) {
      state[i] = EATING;
      send_reply[i]();
    }
}
```



Aterioivat filosofit, dedikoidut kanavat

```
chan request(int), release(int),
    reply[5]();
sem mutex=1;
```

```
process filosofer[i=0 to 4] {
    while (true) {
        think();
        send request(i);
        receive reply[i]();
        eat();
        send release(I);
    }
}
```

*Miten odottaa yhtä aikaa
kahdesta eri kanavasta?*

```
process secretary {
    co while (true) {
        receive request(philos);
        P(mutex);
        state[philos]=HUNGRY;
        consider_allocation_to(philos);
        V(mutex);
    }
    // while (true) {
        receive release(philos);
        P(mutex);
        state[philos]=THINKING;
        consider_allocation_to(philos-1);
        consider_allocation_to(philos+1);
        V(mutex);
    }
}
oc
}
```

Huomaa rinnakkaisuus!

Monitori vs. Palvelija

• **proseduurikutsu vs. kanava & sanoma**

• **poissulkeminen**

- monitori: implisiittisesti, ei semaforeja!
- palvelija: palvele yksi asiakas kerrallaan,
uudet pyyntösanomat jäävät kanavaan

• **synkronointi**

- monitori: jos ei saa edetä, wait(ehtomuuttuja)
 - kutsunut prosessi nukahtaa
- palvelija: jos ei saa edetä, laita sisäiseen jonoon
 - palvelija ei voi nukahtaa!

Monitori vs. Palvelija

Monitor-Based Programs

permanent variables
procedure identifiers
procedure call
monitor entry
procedure return
wait statement
signal statement
procedure bodies

Message-Based Programs

local server variables
request channel and operation kinds
send request(); **receive reply**
receive request()
send reply()
save pending request
retrieve and process pending request
arms of case statement on operation kind

Andrews Table 7.1.

Resurssin varaus, Monitori

```
monitor Resource_Allocator {
    int avail = MAXUNITS;
    set units = initial values;
    cond free;    # signaled when a process wants a unit
    procedure acquire(int &id) {
        if (avail == 0)
            wait(free);
        else
            avail = avail-1;
        remove(units, id);
    }
    procedure release(int id) {
        insert(units, id);
        if (empty(free))
            avail = avail+1;
        else
            signal(free);
    }
}
```

Condition passing

```

type op_kind = enum(ACQUIRE, RELEASE);
chan request(int clientID, op_kind kind, int unitid);
chan reply[n](int unitID);

process Allocator {
  int avail = MAXUNITS; set units = initial values;
  queue pending; # initially empty
  int clientID, unitID; op_kind kind;
  declarations of other local variables;
  while (true) {
    receive request(clientID, kind, unitID);
    if (kind == ACQUIRE) {
      if (avail > 0) { # honor request now
        avail--; remove(units, unitID);
        send reply[clientID](unitID);
      } else # remember request
        insert(pending, clientID);
    } else { # kind == RELEASE
      if empty(pending) {
        avail++; insert(units, unitid);
      } else {
        remove(pending, clientID);
        send reply[clientID](unitID);
      }
    }
  }
}

```

Resurssin varaus, Palvelija

```

process Client[i = 0 to n-1] {
  int unitID;
  send request(i, ACQUIRE, 0)
  receive reply[i](unitID);
  # use resource, release
  send request(i, RELEASE, unitID);
  ...
}

```



Kommunikointitapoja

Jutustelun jatkuvuus / ylläpito

• Useita asiakkaita, useita palvelijoita

• Esim: Tiedostopalvelija

- yksi ulospäin näkyvä palvelu (tdstojen käyttö)
- yksi palvelijaprosessi kullekin asiakkaalle

• Sitominen

- asiakas ⇔ mikä tahansa prosessi

• Tarve

- prosessi haluaa tehdä sarjan peräkkäisiä tiedosto-operaatioita (open(...), read(), ...), sama palvelijaprosessi palvelee

Tiedosto- palvelijat ja asiakkaat

```
type kind = enum(READ, WRITE, CLOSE);
chan open(string fname; int clientID);
chan access[n](int kind, types of other arguments);
chan open_reply[m](int serverID); # server id or error
chan access_reply[m](types of results); # data, error, ...

process File_Server [i = 0 to n-1] {
  string fname; int clientID;
  kind k; variables for other arguments;
  bool more = false;
  variables for local buffer, cache, etc.;
  while (true) {
    receive open(fname, clientID);
    open file fname; if successful then:
    send open_reply[clientID](i); more = true;
    while (more) {
      receive access[i](k, other arguments)
      if (k == READ)
        process read request;
      else if (k == WRITE)
        process write request;
      else # k == CLOSE
        { close the file; more = false; }
      send access_reply[clientID](results)
    }
  }
}
```

```
process Client[j = 0 to m-1] {
  int serverID;
  send open("foo", j);
  receive open_reply[j](serverID);
  # use file then close
  send access[serverID](access arguments)
  receive access_reply[j](results);
  ...
}
```

Vertaistoimijat (Interactive Peers)

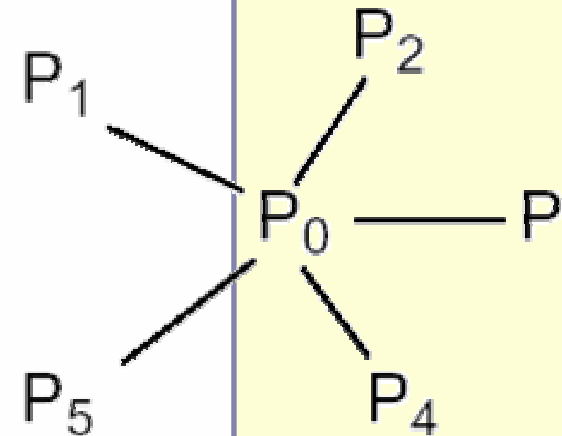
Esim. Arvojen välittäminen

- **n prosessia**
- **kullakin paikallinen arvo v**
- **etsi globaali $\min(v)$, $\max(v)$**

- **Tiedonvälitys**
 - Keskitetty? Symmetrinen? Rengas?
 - Rinnakkaisuus?
 - Tiedonvälityksen tehokkuus?

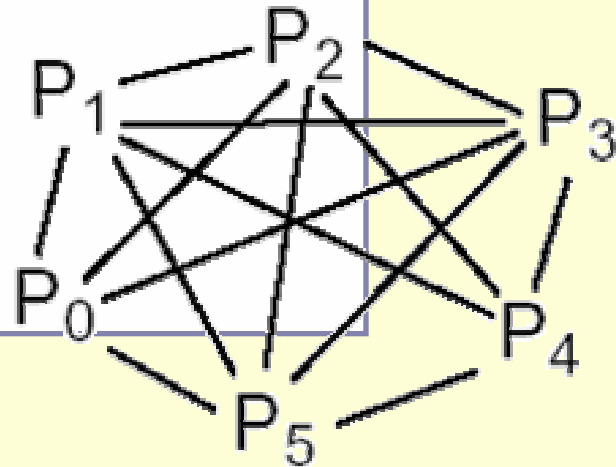
Keskitetty ratkaisu

```
chan values(int), results[n](int smallest, int largest);
process P[0] { # coordinator process
  int v; # assume v has been initialized
  int new, smallest = v, largest = v; # initial state
  # gather values and save the smallest and largest
  for [i = 1 to n-1] {
    receive values(new);
    if (new < smallest)
      smallest = new;
    if (new > largest)
      largest = new;
  }
  # send the results to the other processes
  for [i = 1 to n-1]
    send results[i](smallest, largest)
}
process P[i = 1 to n-1] {
  int v; # assume v has been initialized
  int smallest, largest;
  send values(v);
  receive results[i](smallest, largest);
}
```



Symmetrinen ratkaisu

```
chan values[n] (int);
process P[i = 0 to n-1] {
  int v;    # assume v has been initialized
  int new, smallest = v, largest = v; # initial state
  # send my value to the other processes
  for [j = 0 to n-1 st j != i]
    send values[j] (v);
  # gather values and save the smallest and largest
  for [j = 1 to n-1] {
    receive values[i] (new);
    if (new < smallest)
      smallest = new;
    if (new > largest)
      largest = new;
  }
}
```



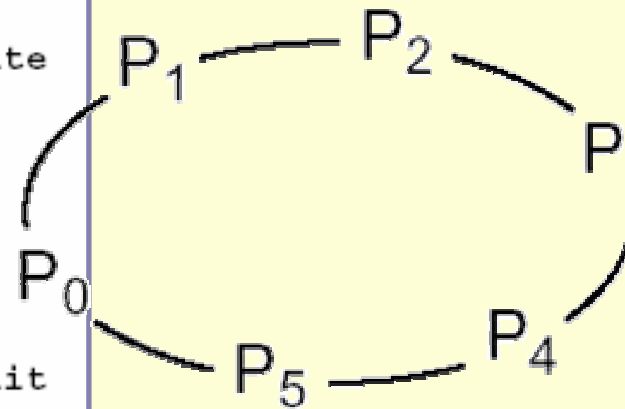
looginen vs. fyysinen rakenne

Andrews Fig. 7.12.

Rengasratkaisu

```
chan values[n](int smallest, int largest);
process P[0] { # initiates the exchanges
  int v; # assume v has been initialized
  int smallest = v, largest = v; # initial state
  # send v to next process, P[1]
  send values[1](smallest, largest);
  # get global smallest and largest from P[n-1] and
  # pass them on to P[1]
  receive values[0](smallest, largest);
  send values[1](smallest, largest);
}
process P[i = 1 to n-1] {
  int v; # assume v has been initialized
  int smallest, largest;
  # receive smallest and largest so far, then update
  # them by comparing their values to v
  receive values[i](smallest, largest)
  if (v < smallest)
    smallest = v;
  if (v > largest)
    largest = v;
  # send the result to the next processes, then wait
  # to get the global result
  send values[(i+1) mod n](smallest, largest);
  receive values[i](smallest, largest);
}
```

*Huom:
ratkaisussa virhe*



Andrews Fig. 7.13.

Mikä paras?

● **Keskitetty**

- $2 * (n-1)$ sanomaa (yleislähetyksenä n)

● **Symmetrinen**

- Yksi samanlainen ohjelmakoodi, eri datat
- $n*(n-1)$ sanomaa

● **Rengas**

- Huomaa koodissa oleva virhe!
- $2*n - 1$ sanomaa

● **Rinnakkaisuus?**

● **Tiedonvälityksen tehokkuus?**

Synkroninen sanomanvälitys

- **synch-send()**

- send() ja receive() “kohtaavat”

- **kommunikoivien prosessien synkronointi**

- naapuri tietää naapurinsa tilan

- **ei tarvitse välttämättä KJ:n apupuskureita**

- **rajoittaa rinnakkaisuutta**

- **Varo lukkiuraa**

- algoritmit, jotka toimivat asynkronisen kommunikoinnin yhteydessä eivät ehkä enää toimikkaan!
- toimivatko edelliset min/max esimerkit?

```

process Producer {
  int data[n];
  for [i = 0 to n-1] {
    do some computation;
    synch_send values(data[i]);
  }
}

process Consumer {
  int results[n];
  for [i = 0 to n-1] {
    receive values(results[i]);
    do some computation;
  }
}

```

Rinnakkaisuus?

```

channel in1(int), in2(int);
process P1 {
  int value1 = 1, value2;
  synch_send in2(value1);
  receive in1(value2);
}

process P2 {
  int value1, value2 = 2;
  synch_send in1(value2);
  receive in2(value1);
}

```

Ei OK!

Tarkistusksymyksiä?