

⑧ Rendezvous

Kohtaaminen

Andrews 8.2, 10.3

Rendezvous

● Aktiivinen palvelija

- suorituksessa oleva prosessi antaa palvelun
- vrt: RPC: töpö luo/aktivoi prosessin, joka suorittaa palvelun

● Silloin tällöin

- aktiivinen palvelija ja asiakas *kohtaavat*
- palvelija suorittaa pyydetyn operaation (asiakas odottaa)
- ja palvelija jatkaa muita aktiviteetteja (jos on)

● Rendezvous

- synkronointi ja kommunikointi yhdistetty
- operaation suorittaminen (palvelija), yksi kerrallaan

Vrt.
etäproseduuri-
kutsu!

Rendezvous moduuli

```
module Mname
  op opname1(formals), opname2(formals);
body
  declarations of shared variables;
  local procedures and processes;
  process pname {
    declarations of local variables;
    while (true) {
      statements;
      in opname1 (formals) -> statements;
      [] opname2 (formals) -> statements;
      ni
      statements;
    }
  }
end mname
```

*julkisten operaatioiden
esittely (export)*

*kohtaamispaikat, jotka
toteuttavat operaatiot*

Syöttölause (palvelija)

~ Aika ja paikka kohtaamiselle

• **in opname(formal identifiers) -> S; ni**

- palvelija odottaa asiakasta, joka kutsuu (call Mname.opname())
- jonka jälkeen palvelija suorittaa lauseosan S

• **opname(...): vahti = kohtaamispaikka**

- Jos asiakas valmiina (“paikalla”), niin suorita lauseosa S
- Jos asiakas ei paikalla, tarkista muut in-lauseen kohtaamispaikat

• **S: vartioidut lauseet**

- asiakasprosessi saa jatkaa (“kohtaaminen ohi”) kun lauseosa S on suoritettu

• Yleinen muoto

in $op_1(\text{formals}_1)$ and B_1 by $e_1 \rightarrow S_1;$

[] ...

[] $op_n(\text{formals}_n)$ and B_n by $e_n \rightarrow S_n;$

ni

- in op(formals) operaation nimi ~ kohtaamispaikka
- and B synkronointilauseke (boolean lauseke)
- [] ... muut vartioidut kohtaamispaikat (FCFS)

Ja vuorottamislauseke (by lauseke)

- kohtaamiseen voi syntyä jonoa (synkronointilauseke ei true)
⇒ missä järjestyksessä odottavat palvellaan (~prioriteetti)
- oletus: palvele vanhin pyyntö ensin

module BoundedBuffer

op deposit (typeT), fetch (typeT);

body

process Buffer {

typeT buf [n];

int front = 0, rear = 0, count = 0;

while (true)

in deposit (item) and count < n ->

buf [rear] = item;

rear = (rear + 1) mod n;

count = count + 1;

[] fetch (item) and count > 0 ->

item = buf [front];

front = (front + 1) mod n;

count = count + 1;

ni

}

Bounded Buffer

```
module BoundedBuffer
  op deposit(typeT), fetch(result typeT);
body
  process Buffer {
    typeT buf[n];
    int front = 0, rear = 0, count = 0;
    while (true)
      in deposit(item) and count < n ->
        buf[rear] = item;
        rear = (rear+1) mod n; count = count+1;
      [] fetch(item) and count > 0 ->
        item = buf[front];
        front = (front+1) mod n; count = count-1;
    ni
  }
end BoundedBuffer
```

vrt. Andrews Fig. 5.4

Poissulkeminen? Synkronointi?

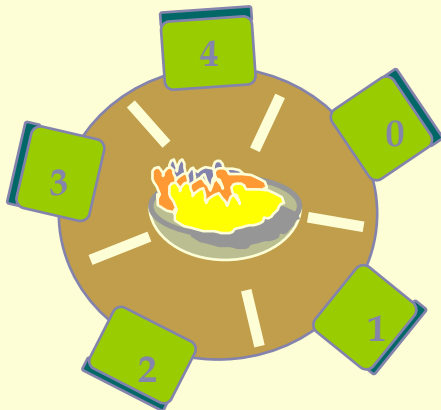
Andrews Fig. 8.5.

```
process Producer[i=1 to N]
{ typeT item;
  while (true) {
    ...
    item = add_this_and_that();
    call BoundedBuffer.deposit(item);
    ...
  }
}
```

```
process Consumer[i=1 to M]
{ typeT item;
  while (true) {
    ...
    BoundedBuffer.fetch(item);
    do_this_and_that(item);
    ...
  }
}
```


Aterioivat filosofit, keskitetty ratkaisu

```
module Table
  op getforks(int), relforks(int);
body
  process Waiter {
    bool eating[5] = ([5] false);
    while (true)
      in getforks(i) and not (eating[left(i)] and
        not eating[right(i)] -> eating[i] = true;
      [] relforks(i) ->
        eating[i] = false;
      ni
    }
end Table
```



```
process Philosopher[i = 0 to 4] {
  while (true) {
    call getforks(i);
    eat;
    call relforks(i);
    think;
  }
}
```

Aikapalvelija

```
module TimeServer
  op get_time() returns int;
  op delay(int);
  op tick();          # called by clock interrupt handler
body TimeServer
  process Timer {
    int tod = 0;    # time of day
    while (true)
      in get_time() returns time -> time = tod;
      [] delay(waketime) and waketime <= tod -> skip;
      [] tick() -> { tod = tod+1; restart timer; }
    ni
  }
end TimeServer
```

vrt. Andrews Fig. 8.1

Andrews Fig. 8.7.

Shortest Job-Next allokointi

```
module SJN_Allocator
  op request(int time), release();
body
  process SJN {
    bool free = true;
    while (true)
      in request(time) and free by time -> free = false;
      [] release() -> free = true;
    ni
  }
end SJN_Allocator
```

vrt. Andrews Fig 5.6

Andrews Fig. 8.8.

```
optype stream = (int); # type of data streams

module Merge[i = 1 to n]
  op in1 stream, in2 stream; # input streams
  op initialize(cap stream); # link to output stream
body
  process Filter {
    int v1, v2; # values from input streams
    cap stream out; # capability for output stream
    in initialize(c) -> out = c ni
    # get first values from input streams
    in in1(v) -> v1 = v; ni
    in in2(v) -> v2 = v; ni
    while (v1 != EOS and v2 != EOS)
      if (v1 <= v2)
        { call out(v1); in in1(v) -> v1 = v; ni }
      else # v2 < v1
        { call out(v2); in in2(v) -> v2 = v; ni }
    # consume the rest of the non-empty input stream
    while (v2 != EOS)
      { call out(v2); in in2(v) -> v2 = v; ni }
    while (v1 != EOS)
      { call out(v1); in in1(v) -> v1 = v; ni }
    call out(EOS);
  }
end Merge
```

Arvojen välittäminen

```
module Exchange [i = 1 to 2]
  op deposit(int);
body
  process Worker {
    int myvalue, othervalue;
    if (i == 1) {      # one process calls
      call Exchange[2].deposit(myvalue);
      in deposit(othervalue) -> skip; ni
    } else {          # the other process receives
      in deposit(othervalue) -> skip; ni
      call Exchange[1].deposit(myvalue);
    }
    ...
  }
end Exchange
```

Varo lukkiuraa!

Andrews Fig. 8.10.

Ada: Rendezvous ja kohtaamispaikat (s. 375)

```
select
  accept op1(formals1) do
    statements;
or
  ...
or
  accept opn(formalsn) do
    statements;
end select
```

Vastaavaa Select-or
-rakennetta saa käyttää
harjoituksissa ja kokeessa.

Käytä receive-operaatiota
acceptin paikalla.

Ks. myös (vaikka ei varsinaisesti kuulukaan kurssiin)

- Andrews Fig 8.17: Bounded buffer with Ada
- Andrews Fig 8.18, Fig 8.19: Dining philosophers with Ada

Kertauskysymyksiä?