

Etäproseduurikutsu

RPC
Toteutus
Virhesemantiikka

Andrews 8.1, 10.3, Stallings 13.3

Etäproseduurikutsu, Remote Procedure Call (RPC)

- Palvelu etäkoneessa, ei yhteistä muistia
- Asiakkaat pyytävät palvelua proseduurikutsumeکانismilla
- Toteutuksen yksityiskohdat KJ:n palvelua
 - | taustalla sanomanvälitys
- RPC yhdistää monitorin ja synkronisen sanomanvälityksen piirteet
 - | kaksisuuntainen synkroninen kanava yhdellä kutsulla
 - | **asiakas odottaa**

Rio 2005 / Auvo Häkkinen, Liisa Marttilinen 7-2

Etäproseduurin moduuli

```

module mname
  [op] opname (formals) [returns result] julkisten operaatioiden esittely (export)
  body
    variable declarations;
    initialization code;
    [proc] opname (formal identifiers) returns result identifier
    declarations of local variables;
    statements
  end
  local procedures and processes;
end mname
  
```

Kutsu

```
call mname.opname (arguments)
```

Rio 2005 / Auvo Häkkinen, Liisa Marttilinen 7-3

Poissulkeminen moduulin sisällä?

- Monitorin tapaan implisiittinen poissulkeminen moduulin sisällä?
 - | vain yksi aktiivinen prosessi voi suorittaa
- Vai usea moduulin prosessi voi olla samaan aikaan suorituksessa => poissulkemisesta huolehdittava eksplisiittisesti?
 - | **kurssilla oletetaan, että tämä tapa on käytössä**
 - | On nykyisin yleisempi!
 - | poissulkeminen esim. semaforeilla tai paikallisilla monitoreilla tai rendezvousia käyttäen.

Rio 2005 / Auvo Häkkinen, Liisa Marttilinen 7-4

Aikapalvelin TimeServer

```

module TimeServer
  op get_time() returns int; # kysy aikaa
  op delay (int interval); # pyydä herätys
  body
    int tod = 0;
    sem m = 1; # mutex
    sem d[n] = ([n] 0); # omat odotussemaforit
    queue of (int waketime, int process_id) napQ;
  end
end TimeServer
  
```

Aikapalvelin, jolta voi

- kysyä kellonaikaa
- pyytää herätystä annetun ajan päästä

Rio 2005 / Auvo Häkkinen, Liisa Marttilinen 7-5

```

proc get_time () returns time {
  time = tod;
}

proc delay (int interval); {
  int waketime = tod + interval;
  P(m);
  insert (waketime, myid) oikeaan paikkaan napQ-jonoon;
  V(m);
  P(d[myid]; # jää odottamaan herätystä
}
  
```

Rio 2005 / Auvo Häkkinen, Liisa Marttilinen 7-6

```

process Clock { # sisäinen prosessi
  käynnistä ajastin;
  while (true) {
    odota keskeytystä ja käynnistä ajastin uudelleen;
    tod = tod + 1; # taas yksi tikitys lisää
    P(m);
    while (tod >= smallest waketime on napQ) {
      remove (waketime, id) from napQ;
      V(d[id]); # herätä prosessi
    }
    V(m);
  }
}
end TimeServer

```

Alkopalvelumuodull

```

module TimeServer
  op get_time() returns int; # retrieve time of day
  op delay(int interval); # delay interval ticks
  body
    int tod = 0; # the time of day
    sem m = 1; # mutual exclusion semaphore
    sem d[n] = ([n] 0); # private delay semaphores
    queue of (int waketime, int process_id) napQ;
    ## when m == 1, tod < waketime for delayed processes

    proc get_time() returns time {
      [time = tod;]
    }

    proc delay(interval) { # assume interval > 0
      int waketime = tod + interval;
      P(m);
      insert (waketime, myid) at appropriate place on napQ;
      V(m);
      P(d[myid]); # wait to be awakened
    }
  }
end

```

Andrews Fig. 8.1.

```

process Clock {
  start hardware timer;
  while (true) {
    wait for interrupt, then restart hardware timer;
    tod = tod+1;
    P(m);
    while (tod >= smallest waketime on napQ) {
      remove (waketime, id) from napQ;
      V(d[id]); # awaken process id
    }
    V(m);
  }
}
end TimeServer

```

Kutsu:

```

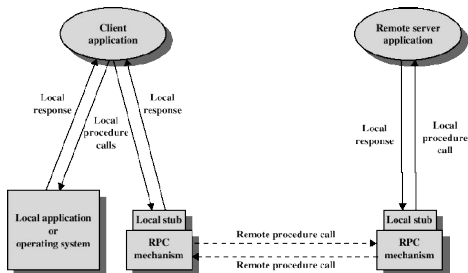
time = TimeServer.get_time();
call TimeServer.delay(10);

```

Andrews Fig. 8.1.

Toteutus

Etäproseduurikutsu (RPC)



Stallings Fig. 13.13.

Toiminta:

- Asiakas
 - | "normaali proseduurikutsu"
 - | toteutus: kääntäjä muuttaa töpön kutsuksi
- Asiakkaan 'töpö' / 'tynkä' (stub)
 - | kirjastorutiini
 - | kokoaa parametrit sanomaan molempien koneiden ymmärtämässä muodossa (marshalling)
 - | kutsuu KJ:n palvelua
- Asiakkaan KJ
 - | vastaa töpöjen välisestä sanomanvälityksestä
 - | kanavan luonti, sanoman lähetykset
 - | ajastimet, järjestysnumerot, kuittaukset, uudelleenlähetykset

• Palvelijan KJ (etäkone)

- vastaa töpöjen välisestä *sanomanvälityksestä*
- sanoman vastaanotto, vastauksen lähetys
- ajastimet, järjestysnumerot, kuittaukset, uudelleenlähetykset
- käynnistää / luo prosessin palvelemaan
 - suorittaa töpön

• Palvelijan töpö (etäkone)

- purkaa parametrit kutsupinoon
- *tekee normaalin proseduurikutsun*
- kokoaa vastaussanoman molempien ymmärtämään muotoon
- kutsuu KJ:n sanomanvälityspalvelua

• Asiakkaan töpö

- purkaa vastauksen ja sijoittaa asiakkaan muuttujiin

• Etäpalvelun ohjelmoija määrittelee rajapinnat

- esittelee julkiset operatiot moduulin otsakeosassa

• Kääntäjä generoi tarvittavat töpöt

- kutsu ja kutsuttava rutiini käännetään toisistaan riippumatta
- voivat olla toteutettu eri kielillä

• Nimipalvelu (portmapper, rmdir)

- proseduurin nimi \leftrightarrow kone ja palvelurajapinta?
- palvelun tarjoaja rekisteröi (bind) palvelun nimipalveluun
 - export
- asiakas kysyy palvelun tiedot nimipalvelijalta (lookup)
 - import

Parametrien järjestely (marshalling)

• Parametrit koottava sanomaan

- ei yhteistä muistia
- kopioi kaikki parametrit, ei voi käyttää viiteparametreja

• Ongelmia

- perustietotyyppien esitysmuodot
- yhteiskäyttöiset muuttujat
- taulukot
- linkitetyt tietorakenteet
- verkot
- ohjelmoijan itse määrittelemät tyypit
 - kirjoita omat järjestelyrutiinit

Kuka suorittaa etäproseduurin (etäkoneessa)?

• Yksi odotteleva palvelijaprosessi

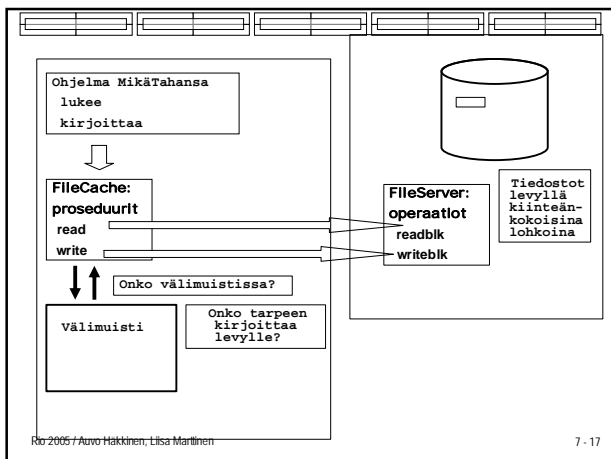
- silmukka, odota kutsua töpön receive()-operaatioissa
- kun kutsuttu, palvele ja lähetä vastaus send()-operaatiolla
- vain yksi kutsu kerrallaan, ei poissulkemistarvetta

• Luo uusi prosessi suorittamaan

- rinnakkaisuus
- yleisrasite
- poissulkeminen

• Yksi prosessi, jolla monta valmista säiettä

- "server pool", allokoit kutsulle olemassaoleva säie
- poissulkeminen



Hajautettu tiedostopalvelu: File cache (local)

```

module FileCache # located on each diskless workstation
op read(int count; result char buffer[*]);
op write(int count; char buffer[]);
body
cache of file blocks;
variables to record file descriptor information;
semaphores for synchronization of cache access (if needed);
proc read(count,buffer) {
if (needed data is not in cache) {
select cache block to use;
if (need to write out the cache block)
FileServer.writeblk(...);
FileServer.readblk(...);
}
buffer = appropriate count bytes from cache block;
}
proc write(count,buffer) {
if (appropriate block not in cache) {
select cache block to use;
if (need to write out the cache block)
FileServer.writeblk(...);
}
cache block = count bytes from buffer;
}
end FileCache
    
```

Huomaa:
 - Passiivisia rutiineita
 - joita etäasiakkaat käyttävät (RPC)
 - joita paikallinen toteutus kutsuu (stub)
 - joita suorittaa paikalliset prosessit

Andrews Fig. 8.2a.

Hajautettu tiedostopalvelu: File server (remote)

```

module FileServer # located on a file server
op readblk(int fileid, offset; result char blk[1024]);
op writeblk(int fileid, offset; char blk[1024]);
body
  cache of disk blocks;
  queue of pending disk access requests;
  semaphores to synchronize access to the cache and queue;
  # N.B. synchronization code not shown below
  proc readblk(fileid, offset, blk) {
    if (needed block not in the cache) {
      store read request in disk queue;
      wait for read operation to be processed;
    }
    blk = appropriate disk block;
  }
  proc writeblk(fileid, offset, blk) {
    select block from cache;
    if (need to write out the selected block) {
      store write request in disk queue;
      wait for block to be written to disk;
    }
    cache block = blk;
  }
end FileServer

```

Andrews Fig. 8.2b.

```

process DiskDriver {
  while (true) {
    wait for a disk access request;
    start a disk operation; wait for interrupt;
    awaken process waiting for this request to complete;
  }
}

```

Rio 2005 / Auvo Häkkinen, Liisa Marttinen

Merge-sort ja Capability

RPC sopii asiakas-palvelija sovelluksiin, muu käyttö hankalaa

- Prosessien välinen kommunikointi ohjelmoitava erikseen
- RPC ei tue jatkuvaa kommunikointia
 - ┆ kutsut aina erillisiä ja edellisistä riippumattomia
 - ┆ ohjelmoitava eksplisiittisesti
- RPC:n kommunikointikanava operaation nimi
 - ┆ Miten output-vuo kytketään seuraavan input-vuohon?
- Dynaaminen nimeäminen
 - ┆ capability ~ osoitin kommunikointioperaatioon

```

call Merge[i].initialize(Merge[j].in2)

```

Rio 2005 / Auvo Häkkinen, Liisa Marttinen

Lomituslajittelu

Rio 2005 / Auvo Häkkinen, Liisa Marttinen

```

otype stream = (int); # type of data stream operations
module Merge[i = 1 to n]
op in1 stream, in2 stream; # input streams
op initialize(cap stream); # link to output stream
body
  int v1, v2; # input values from streams 1 and 2
  cap stream out; # capability for output stream
  sem empty1 = 1, full1 = 0, empty2 = 1, full2 = 0;
  proc initialize(output) { # provide output stream
    out = output;
  }
  proc in1(value1) { # produce next value for stream 1
    P(empty1); v1 = value1; V(full1);
  }
  proc in2(value2) { # produce next value for stream 2
    P(empty2); v2 = value2; V(full2);
  }
end Merge

```

call Merge[i].initialize(Merge[j].in2)

Initialize kutsun jälkeen Merge[i].ssa: out = Merge[j].in2

Andrews Fig. 8.3.

Rio 2005 / Auvo Häkkinen, Liisa Marttinen

Lomituslajittelu

```

call Merge[i].initialize(Merge[j].in2)
call Merge[i+1].initialize(Merge[j].in1)

```

sidotaan toisiinsa eli i:n output tulee olemaan j:n in2 ja i+1:n output j:n in1.

Rio 2005 / Auvo Häkkinen, Liisa Marttinen

```

process M {
  P(full1); P(full2); # wait for two values
  while (v1 != EOS and v2 != EOS)
    if (v1 <= v2)
      { call out(v1); V(empty1); P(full1); }
    else # v2 < v1
      { call out(v2); V(empty2); P(full2); }
  # consume the rest of the non-empty stream
  while (v2 != EOS)
    { call out(v2); V(empty2); P(full2); }
  while (v1 != EOS)
    { call out(v1); V(empty1); P(full1); }
  call out(EOS); # append sentinel
}
end Merge

```

Vrt. Sanomanvälitys, Fig 7.2:
RPC => paljon enemmän kirjoitettavaa

Andrews Fig. 8.3.

Rio 2005 / Auvo Häkkinen, Liisa Marttinen

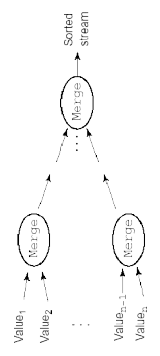
Suodin: lomita kaksi syöttökanavaa

```

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.



Arvojen välittäminen: kaksi prosessia vaihtaa arvot keskenään

```

module Exchange[i = 1 to 2]
  op deposit(int); Export-operaatio, jota toinen voi kutsua
  body
    int othervalue;
    sem ready = 0; # used for signaling

    proc deposit(other) { # called by other module
      othervalue = other; # save other's value
      V(ready); # let Worker pick it up
    }

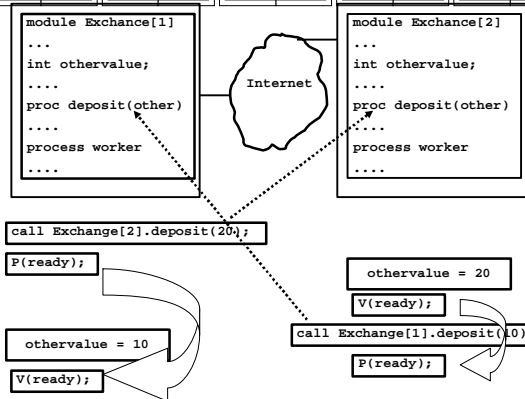
    process Worker { Oikean indeksin laskemiseen
      int myvalue;
      call Exchange[3-i].deposit(myvalue); # send to other
      P(ready); # wait to receive other's value
      ...
    }
  end Exchange

```

Tuloksena on vähän omituisia ohjelmia!

Vrt. Sanomanvälitys, Fig 7.11-7.13

Andrews Fig. 8.4.



Vertaa: Arvojen vaihto sanomavälityksellä:

```

chan ch1 (int), ch2 (int);
process P1 {
  int value = 2;
  send ch2(value);
  receive ch1(value);
}
process P2 {
  int value = 1;
  send ch1(value);
  receive ch2(value);
}

```

Virhesemantiikkaa

Virhesemantiikkaa

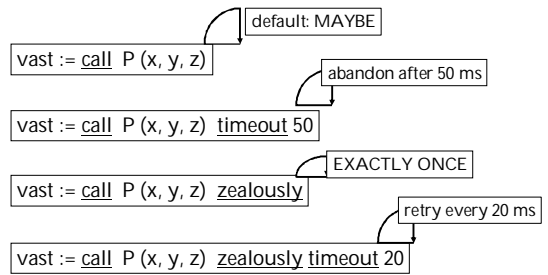
- Tiedonsiirto
 - | kadonneet sanomat, yms.
- Palvelijaprosessi (etä-)
 - | paikallinen virhetilanne (ei tarvetta raportoida)
 - | pitkä palveluaika (esim. jonotuksen aiheuttama viive)
- Asiakasprosessi
 - | ongelman havaitseminen: ajastin
 - | toipuminen?
- Erillinen suoritus Ø erilliset virheet
 - | Eivät riipu suoraan toisistaan

Suoritussemantiikka

- toipumispolitiikka töpöissä

- **exactly-once**
 - ┆ toteutus?
 - ┆ ajastimen laukeaminen
 - ┆ käytä sama sanomanumero uudestaan uudelleenkutsussa
- **at-least-once**
 - ┆ yritä uudelleen kunnes ok
- **at-most-once**
 - ┆ onnistui: tehty täsmälleen kerran
 - ┆ virhe: "no operation" (tieto asiakkaalle), ei uudelleen yritystä
- **maybe**
 - ┆ ei vastausta: suoritettu tai sitten ei

Esim: CCLU RPC



See Bacon97 s384

UNIX, rpc-kirjasto

- ks. RFC1050

man rpc
man rpcgen
man portmap

Java: Remote Method Invocation

- java.rmi, java.rmi.server, java.rmi.registry
- sovelluksen osat
 - ┆ interface (moduulin otsake)
 - ┆ server class
 - ┆ rekisteröi: Naming.bind()
 - ┆ client class
 - ┆ etsi palvelu rekisteristä: Naming.lookup()

lue opastussivut
C Andrews ch 8.5

Kertauskysymyksiä?