

LUENTO 18

## Hajautettu prosessointi

### Stallings, Luvut 13-14.3

1

## Sisältöä luento 18

- Hajautuksesta yleensä
- Tietoliikenne -pikakertaus
  - TCP/IP-pino
  - Pistokkeet
  - LINUX ja TCP/IP
- Asiakas / palvelin malli
- Etäproseduurikutsu - RPC

2

## Erilaisia arkkitehtuureja

Moniprosessori-kone

Hajautettu järjestelmä

Eroa?

1.6

Different basic organizations and memories in distributed computer systems

3

## Esimerkkejä: homogeenisia hajautettuja arkkitehtuureja

(a)

(b)

1.9 a) Grid      b) Hypercube

Uusi suunnitteluperiaate: Paikallisuus solmun tasolla

4

## Hajautettu järjestelmä yleisesti

- löyhästi kytkettyjä laitteita (loosely connected)
  - Itsenäiset, autonomiset solmut
  - Hidas ja "epäluotettava" tietoliikenne
 => Yhteistyö palvelujen (sovellusten) kautta
- Tyypilliset sovellustason arkkitehtuurit
  - multiprocessor systems: rinnakkaislaskenta (parallel computing)
  - multicomputer systems: hajautettu järjestelmä (distributed system)

*(Mikä eroa on termeillä parallel, concurrent ja distributed?)*

5

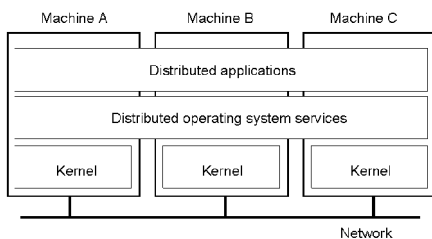
## Software Concepts

System	Description	Main Goal
DOS	Tightly-coupled <i>operating system</i> for multiprocessors and homogeneous multicomputers	Hide and manage hardware resources
NOS	Loosely-coupled <i>operating system</i> for heterogeneous multicomputers (LAN and WAN)	Offer local services to remote clients
Middle-ware	Additional <i>layer atop of NOS</i> implementing general-purpose services	Provide distribution transparency

DOS: Distributed OS; NOS: Network OS

6

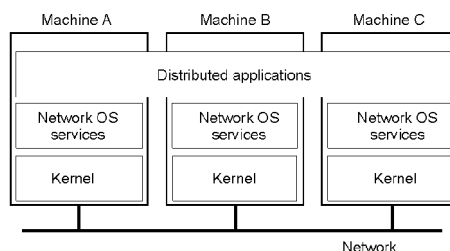
### Hajautettu käyttöjärjestelmä



1.14 General structure of a multicomputer operating system

- Yhteinen KJ, jota kaikki järjestelmän verkotetut koneet käyttävät

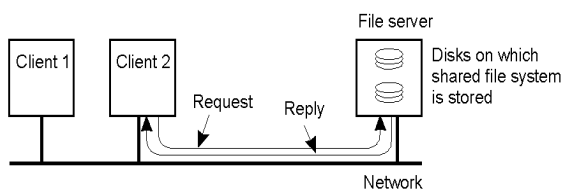
### Verkkokäyttöjärjestelmä?



1-19 General structure of a network operating system.

- Kullakin koneella oma käyttöjärjestelmä, joka voi tarjota paikallisten palvelujen lisäksi joitakin verkkopalveluja

### Verkkojärjestelmäesimerkki: tiedostopalvelin



1-20 Two clients and a server in a network operating system.

### Comparison between Systems

Item	Distributed OS		Network OS	Middleware-based OS
	Multiproc.	Multicomp.		
Degree of transparency	Very High	High	Low	High
Same OS on all nodes	Yes	Yes	No	No
Number of copies of OS	1	N	N	N
Basis for communication	Shared memory	Messages	Files	Model specific
Resource management	Global, central	Global, distributed	Per node	Per node
Scalability	No	Moderately	Yes	Varies
Openness	Closed	Closed	Open	Open

### Tietokoneiden yhdistäminen - tietoliikenne

- Tarvitaan sopimus tiedonsiirrosta eli protokolla.
- Protokollia on eri tasoilla ja niistä muodostuu hierarkia, jossa kullakin on oma tehtävänsä.

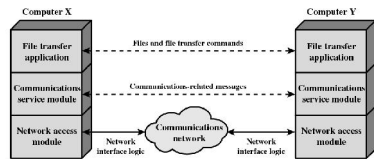


Figure 13.1 A Simplified Architecture for File Transfer

### TCP/IP yleisesti

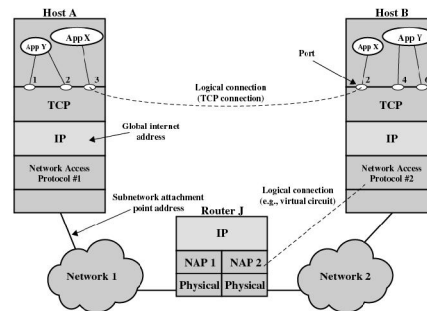
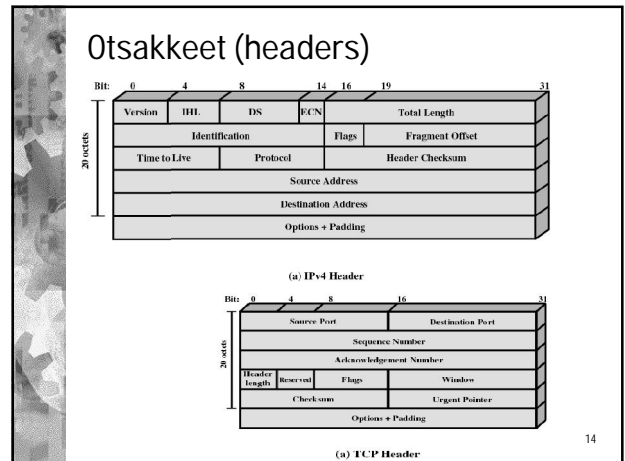
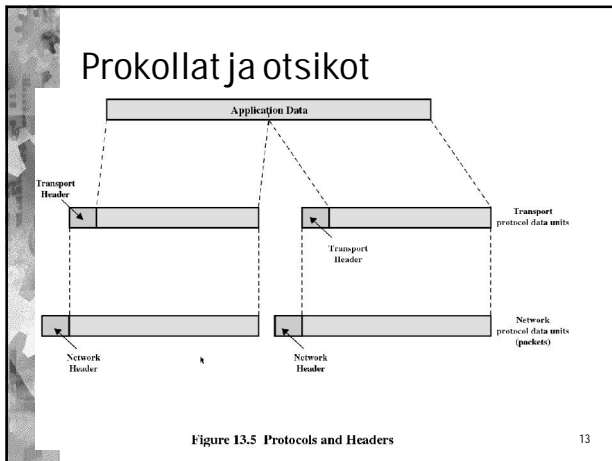
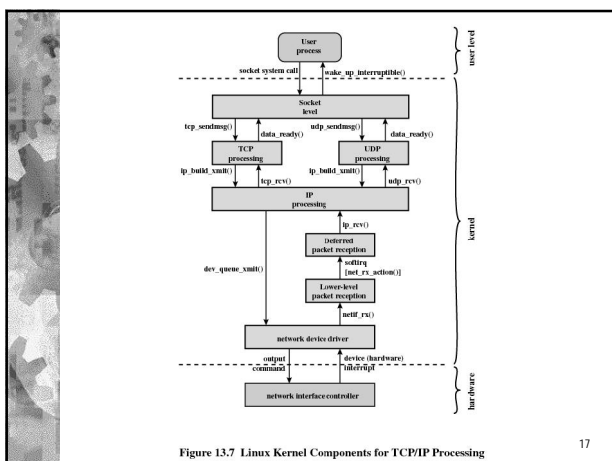
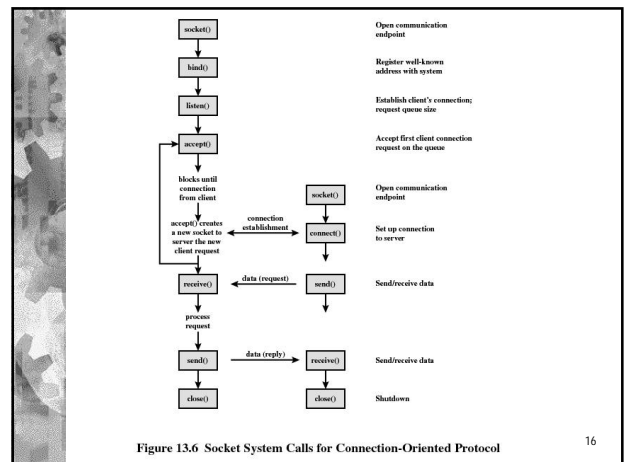


Figure 13.4 TCP/IP Concepts



- ### Pistokkeet (sockets)
- Sovelluksen keino ottaa yhteyttä tietoliikenteen kautta toiseen ohjelmaan muussa koneessa.
  - Rajapintafunktio `socket()`, parametrit:
    - Protokollaperhe: PF\_INET kun TCP/IP
    - Tyyppi kertoo onko kyseessä yhteydellinen vai yhteydetön käyttötapa
    - Protokolla kertoo onko käytössä TCP vai UDP
  - IP osoite ja porttinumero identifioivat pistokkeen (`bind`)
  - Tavuvirtapistoke (Stream sockets)
    - Yhteydellinen, luotettava, järjestyksen säilyttävä, TCP
  - Tietosähkepistoke (Datagram sockets)
    - Ei yhteyttä, epäluotettava, sanomia voi kadota tai monistua, järjestystä ei taata, UDP
- 15



### Asiakas-palvelija -malli

18

<h3>Asiakas</h3> <ul style="list-style-type: none"> <li>• Aktiivinen, tekee aloitteen yhteistoiminnalle</li> <li>• Tyypillisesti lähettää palvelupyynnön ja jää odottamaan vastausta</li> <li>• Usein vain yksi pyyntö kerrallaan liikkeellä</li> </ul>	<h3>Palvelija</h3> <ul style="list-style-type: none"> <li>• Passiivinen, odottaa asiakkaan yhteydenottoa</li> <li>• Tyypillisesti odottaa yhteyttä miltä tahansa asiakkaalta, saatuaan pyynnön toteuttaa sen ja lähettää vastauksen</li> <li>• Asiakkaiden pyynnöt joko jonottavat tai palvelija voi palvelua useita samanaikaisesti</li> </ul>
---	---

19

### Figure 2.2

Clients invoke individual servers

CoDoKi, Fig. 2.2

- Miten asiakas löytää palvelijan?

20

```

/* Definitions needed by clients and servers. */
#define TRUE 1
#define MAX_PATH 255 /* maximum length of file name */
#define BUF_SIZE 1024 /* how much data to transfer at once */
#define FILE_SERVER 243 /* file server's network address */

/* Definitions of the allowed operations */
#define CREATE 1 /* create a new file */
#define READ 2 /* read data from a file and return it */
#define WRITE 3 /* write data to a file */
#define DELETE 4 /* delete an existing file */

/* Error codes. */
#define OK 0 /* operation performed correctly */
#define E_BAD_OPCODE -1 /* unknown operation requested */
#define E_BAD_PARAM -2 /* error in a parameter */
#define E_IO -3 /* disk error or other I/O error */

/* Definition of the message format. */
struct message {
    long source; /* sender's identity */
    long dest; /* receiver's identity */
    long opcode; /* requested operation */
    long count; /* number of bytes to transfer */
    long offset; /* position in file to start I/O */
    long result; /* result of the operation */
    char name[MAX_PATH]; /* name of file being operated on */
    char data[BUF_SIZE]; /* data to be read or written */
};
    
```

Esimerkki: header.h  
21

```

#include <header.h>
void main(void) {
    struct message m1, m2; /* incoming and outgoing messages */
    int r; /* result code */

    while(TRUE) { /* server runs forever */
        receive(FILE_SERVER, &m1); /* block waiting for a message */
        switch(m1.opcode) { /* dispatch on type of request */
            case CREATE: r = do_create(&m1, &m2); break;
            case READ: r = do_read(&m1, &m2); break;
            case WRITE: r = do_write(&m1, &m2); break;
            case DELETE: r = do_delete(&m1, &m2); break;
            default: r = E_BAD_OPCODE;
        }
        m2.result = r; /* return result to client */
        send(m1.source, &m2); /* send reply */
    }
}
    
```

Esimerkki: palvelija  
22

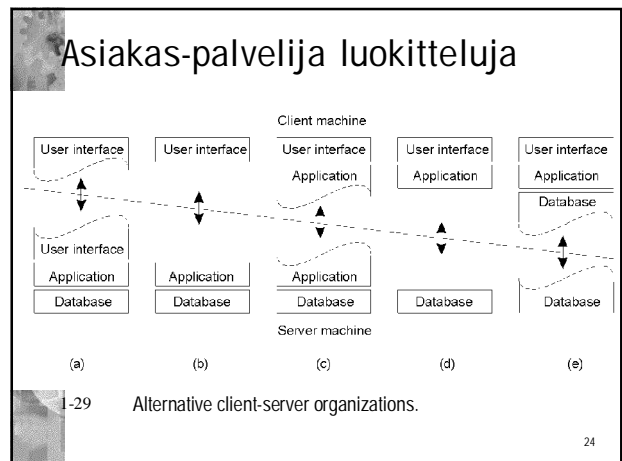
```

#include <header.h>
int copy(char *src, char *dst) {
    struct message m1; /* message buffer */
    long position; /* current file position */
    long client = 110; /* client's address */

    initialize(); /* prepare for execution */
    position = 0;
    do {
        m1.opcode = READ; /* operation is a read */
        m1.offset = position; /* current position in the file */
        m1.count = BUF_SIZE; /* how many bytes to read? */
        strcpy(&m1.name, src); /* copy name of file to be read to message */
        send(FILE_SERVER, &m1); /* send the message to the file server */
        receive(client, &m1); /* block waiting for the reply */

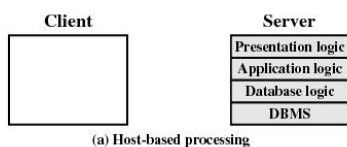
        /* Write the data just received to the destination file. */
        m1.opcode = WRITE; /* operation is a write */
        m1.offset = position; /* current position in the file */
        m1.count = m1.result; /* how many bytes to write */
        strcpy(&m1.name, dst); /* copy name of file to be written to buf */
        send(FILE_SERVER, &m1); /* send the message to the file server */
        receive(client, &m1); /* block waiting for the reply */
        position += m1.result; /* m1.result is number of bytes written */
    } while(m1.result > 0); /* iterate until done */
    return(m1.result >= 0 ? OK : m1.result); /* return OK or error code */
}
    
```

Esimerkki: asiakas  
23



## Asiakas-palvelija luokitteluja

- Kaikki palvelinkoneella (host-based)
  - Ei oikeastaan asiakas-palvelin
  - Tyypillinen keskuskonepalvelu, kun pääteyhteys

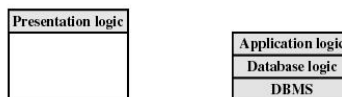


(a) Host-based processing

25

## Asiakas-palvelija luokitteluja

- Prosessointi palvelimella (Server-based)
  - Kaikki toimintalogiikka palvelimella
  - Asiakkaalla on graafinen käyttöliittymä
  - Esim: X-ohjelmat, osa www-sovelluksista



(b) Server-based processing

26

## Asiakas-palvelija luokitteluja

- Sovelluksen logiikkaa asiakkaalla
  - Sovellus toimii asiakkaan päässä
  - Tietokanta ja sen prosessointi on palvelimella

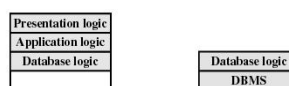


(c) Cooperative processing

27

## Asiakas-palvelija luokitteluja

- Yhteistoiminnallinen prosessointi (Cooperative)
  - Sovelluksen koodin sijoittelu asiakkaan ja palvelijan kesken on tehty optimaalisesti (minkä suhteen?)
  - Yleensä liian monimutkainen



(d) Client-based processing

28

## Kolmitasoinen malli (Three-Tier Client/Server)

- Sovellus (tai palvelu) jakautuu kolmeen osaan, vaikkapa kolmelle eri koneelle
  - Asiakas (User machine)
  - Välittäjä (Middle-tier server)
    - Gateway
    - Muuntaa protokollia
    - Yhdistelee tuloksia useammalta (tausta)palvelijalta
  - (Tausta)palvelija (Backend server)

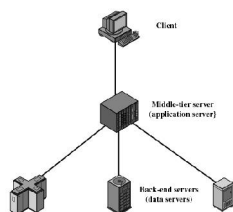
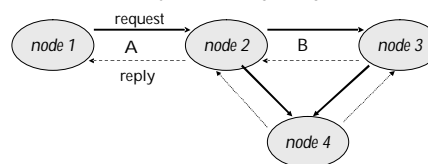


Figure 14.6 Three-tier Client/Server Architecture

29

## Monitasoinen arkkitehtuuri (Multitiered Architectures)

### Asiakas-palvelija -mallin yleistys

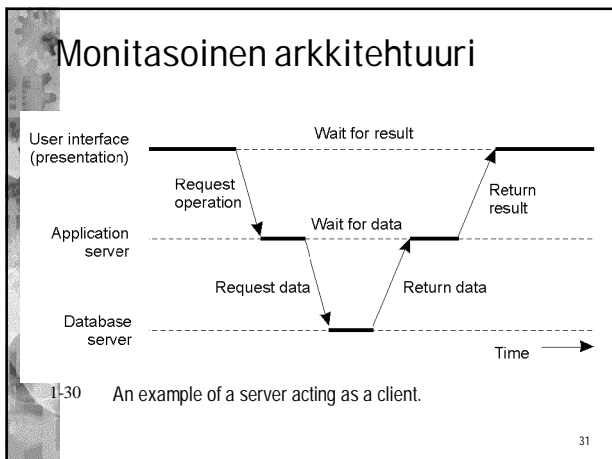


A: node 1 asiakas  
node 2 palvelija

B: node 2 asiakas  
node 3 palvelija

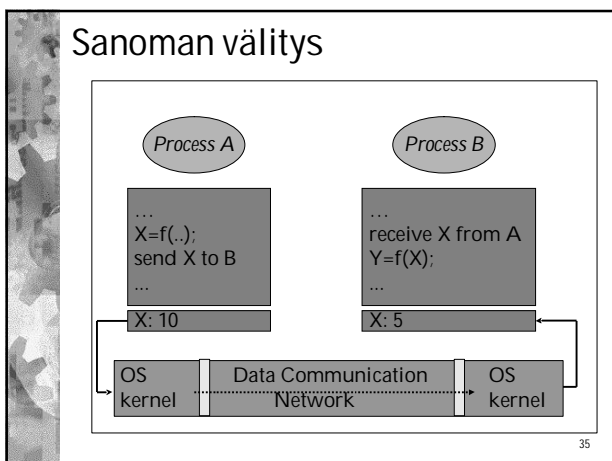
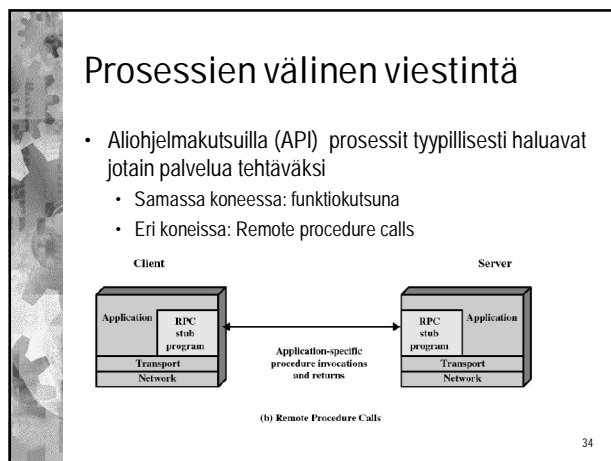
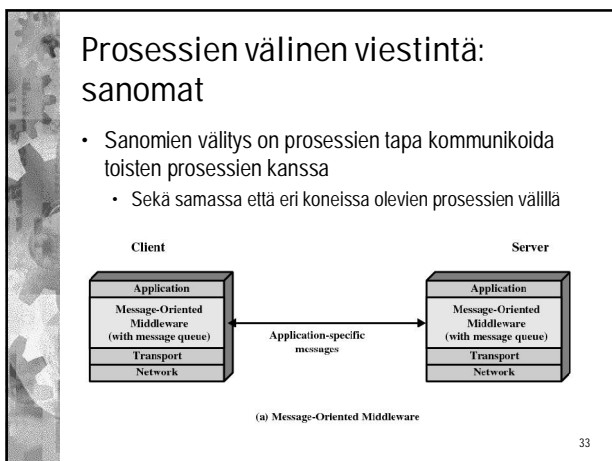
Kyse on siis kommunikoinnista, ei sinänsä solmuista

30



### Hajautettu viestintä

32



### Luotettava sanomanvälitys

- Luotettava välitysmekanismi takaa, että viesti menee perille, jos se ylipäätään on mahdollista
- Tarjoo
  - Virheen tarkistukset (mahdollisesti korjauksia)
  - Kuittaukset (jos lähettäjä haluaa)
  - Uudelleenlähetykset
  - Sanomien järjestyksen säilyttämisen
- Epäluotettava mekanismi on yksinkertainen toteuttaa, mutta ei takaa mitään viestin kulusta

36

## Synkroninen (Blocking) vai asynkroninen (nonblocking)

- Asynkroninen (Asynchronous communication)
  - Lähettäjä voi jatkaa heti lähetettyään, se ei jää odottamaan.
- Synkroninen (Synchronous communication)
  - Lähettäjä odottaa send-pyyntönsä kunnes
    - Sanoma on saapunut vastaanottavaan koneeseen (receipt-based synchrony)
    - Sanoma on saapunut vastaanottajalle (delivery based)
    - Vastausviesti on saapunut lähettäjälle (response based)
  - Vastaavasti receive-pyyntönsä odotellaan kunnes seuraava viesti on saapunut vastaanotettavaksi

37

## Sanomien pysyvyys (persistence)

- Pysyvä viestintä (Persistent communication)
 

Järjestelmä huolehtii, että sanoma säilyy kunnes se on saatu toimitettua vastaanottajalle  
(vastaanottaja käynnistyy joskus, lähettäjä on jo poistunut)
- Välitön viestintä (Transient communication)
 

sanomaa ei talleteta, vaan lähettäjän ja vastaanottajan on oltava aktiivisia sanoman kulkiessa  
(lähettäjän ja vastaanottajan on oltava käynnissä rinnakkain)

38

## Sidonta (binding)

- Miten asiakas ja palvelija löytävät toisensa?
- Tietoliikenneverkon rakenteen avulla
  - one-to-one (jaettu kanava, ei muita kuuntelemassa)
  - many-to-one (perinteinen asiakas- palvelija)
  - one-to-many, many-to-many (ryhmäviestintä)
- Viestin tyyppin avulla
  - Yksi vastaanottaja, monilähetys, yleislähetys

39

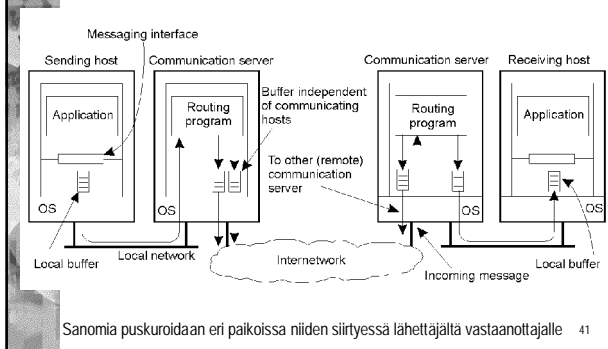
## Sidonta

Koska?

- Staattinen (ohjelmakoodiin suoraan tekovaiheessa)
- Dynaaminen (suoritusaikana)
  - Eksplisiittisesti tietyihin verkkorakenteisiin
  - Implisiittisesti nimipalvelun kautta

40

## Sanoman kulku



## Etäproseduurikutsu (RPC) – Remote Procedure Call

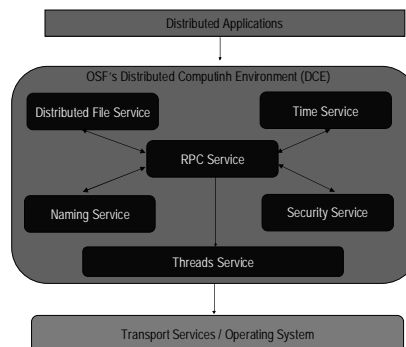
42

## RPC- Remote Procedure Calls

- Eri koneilla olevat ohjelmat (oliot) voivat kommunikoida yksinkertaisen aliohjelmakutsu/paluuarvo (procedure call/return) semantiikan avulla
- Yleisesti käytössä
- Standardoitu
  - Asiakas ja palvelija komponentteja voidaan siirtää koneelta toiselle ja jopa käyttöjärjestelmästä toiseen
- Useita toteutuksia. Yksi esimerkki DCE – Distributed Computing Environment
  - <http://www.opengroup.org/dce/>

43

## DCE:n arkkitehtuuri



44

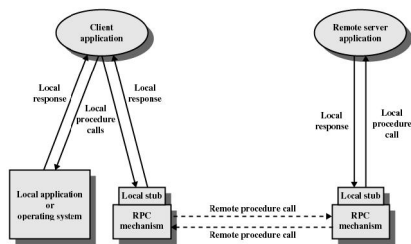


Figure 14.12 Remote Procedure Call Mechanism

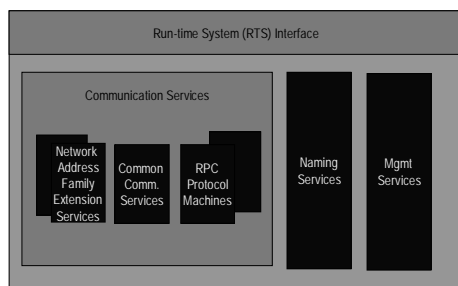
45

## RPC:n elementtejä

- RPC tyngät (*stubs*)
  - Palvelurajapinnan toteutus
  - Parametrien pakkaus (ja purku) siirron ajaksi
- Kuljetuspalvelu (Transportation service)
  - Siirtää sanomat solmusta toiseen
  - Voi olla osa käyttöjärjestelmää tai väliohjelmistoa
- Nimipalvelu (Name service): look up, binding
  - Proseduurien nimet, rajapintakuvaukset (interface definitions)

46

## DCE:n RPC arkkitehtuuri



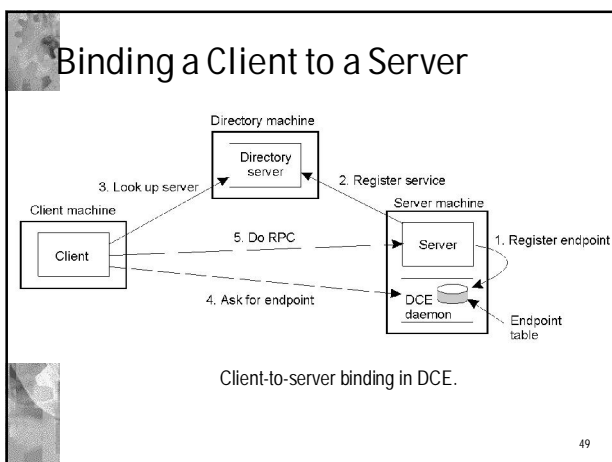
47

## Sidonta

- Nimipalvelun tehtävä
- Sidonnalla määrätään kutsuttavan aliohjelman ja kutsuvan ohjelman välinen suhde
- Sidonta vain kutsun ajaksi (*Nonpersistent binding*)
  - Looginen yhteys muodostetaan kutsussa ja vain kutsun ajaksi
- Pysyvä sidonta
  - Yhteys säilyy myös kutsun jälkeen.

48

## Binding a Client to a Server

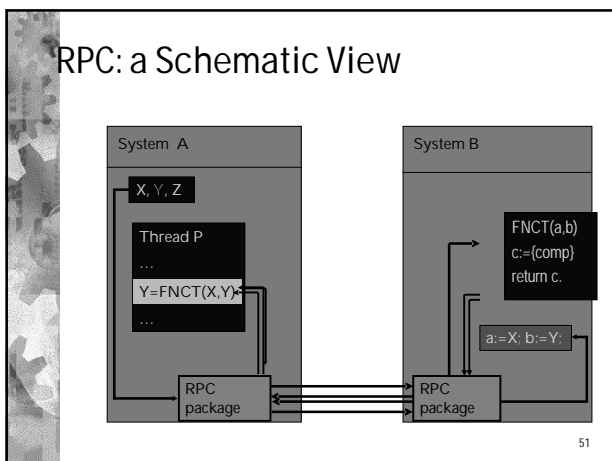


## Synkroninen vai asynkroninen

- Synkroninen RPC (synchronous RPC)
  - Kutsujaa odottaa ihan niin kuin tavallisessa aliohjelmakutsussa
- Asynkroninen RPC (Asynchronous RPC)
  - Kutsuja ei jää odottamaan
  - Kutsuva asiakas voi edetä rinnan palvelijan kanssa

50

## RPC: a Schematic View

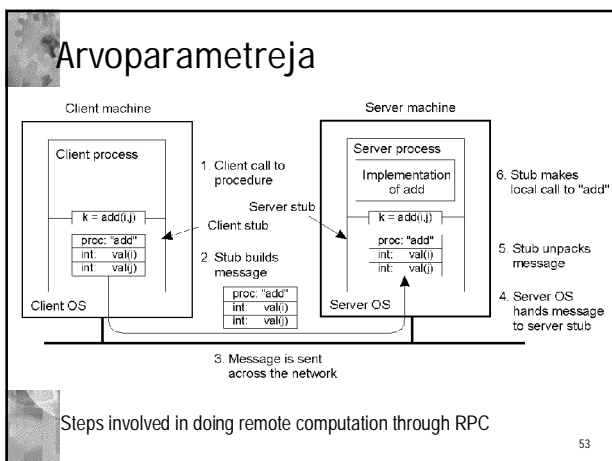


## Parametrien välitys

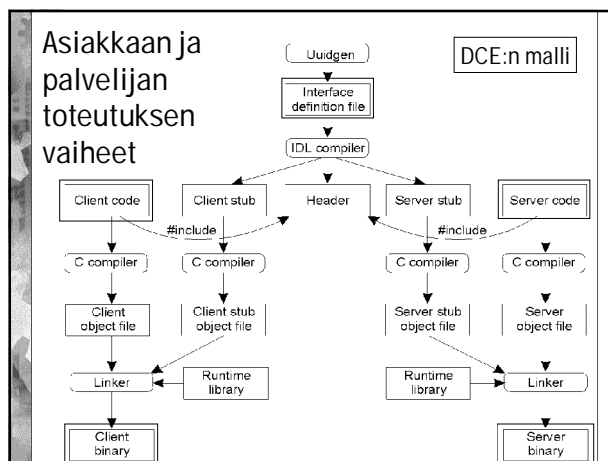
- Kutsujan ja kutsuttavan ymmärrettävä sanoman sisältö samoin
- Parametrit pakataan sovitettuun muotoon (marshaling of data), josta ne sitten voidaan purkaa (unmarshal).
- Esityskerroksen (presentation layer) tehtävä
- Arvoparametrit OK, mutta entä viiteparametrit ja osoittimet?

52

## Arvoparametreja



## Asiakkaan ja palvelijan toteutuksen vaiheet



## Toteutuksesta

Kuka/mikä suorittaa proseduurin palvelimella?

- Erillinen palvelinprosessi
  - Ikuinen silmukka, odottaa seuraavaa pyyntöä "receive"
  - Kutsu saapuu: prosessin suoritus voi jatkua
  - Palveliaan yksi kerrallaan, ei poissulkemisiongelmaa
- Uusi prosessi, jokaiselle kutsulle
  - Rinnakkaisuus on nyt mahdollista
  - Prosessin luonnin yleisrasite jokaisessa kutsussa
  - Poissulkemisiongelma täytyy nyt ratkaista
- Yksi palvelinprosessi, monta säiettä
  - Varataan yksi säie kullekin kutsulle

55

## Esimerkki: simple.idl

- IDL – Interface Description Language
- Määrittellään kutsuttavien etäproseduurien muoto
- IDL:stä voidaan generoida tyngät
  - edustaproseduuri ja etäkutsupalvelin
- Linuxissa *rpcgen* käyttää idl-kuvauksen sijaan RPC language kuvausta

```
/* SIMPLE.IDL */
[
  uuid(004C4B40-E7C5-1CB9-94E7-0000C07C3610),
  version(1.0)
]
interface simple
{
  void simple_operation(
    [in] long x,
    [out] long *y
  );
}
```

56

## Esimerkki: simple-client.c

```
/* SIMPLE_CLIENT.C */
#include <stdio.h>
#include "simple.h"

main(int argc, char *argv[])
{
  idl_long_int x;
  idl_long_int y;

  if (argc < 2) { x = 1; }
  else { x = atoi(argv[1]); }

  simple_operation(x, &y); /* This is the Remote Procedure Call */

  printf("The answer is: %ld.\n");
  return(1);
}
```

simple.h saadaan automaattisesti  
simple.idl:stä idl-kaantajalla

57

## Esimerkki: simple-server.c 1/3

```
/* SIMPLE_SERVER.C */

#include <stdio.h>
#include <dce/rpc.h>
#include "simple.h"

#define ERR_CHK(stat, msg) if(stat != rpc_s_ok) \
  { printf(stderr, "Error: %s in file: %s at line %d.\n", msg, __FILE__, __LINE__); \
  exit(1); }

/***** Server Control *****/

main()
{
  error_status_t status;
  rpc_binding_vector_t *bindings;
  unsigned_char_t *name = "/./applications/simple";
```

58

## Esimerkki: simple-server.c 2/3

```
rpc_server_register_if(simple_v1_0_s_ifspec, NULL, NULL, &status);
ERR_CHK(status, "Could not register interface");
rpc_server_use_all_protseqs(rpc_c_protseq_max_regs_default, &status);
ERR_CHK(status, "Could not use all protocols");
rpc_server_inq_bindings(&bindings, &status);
ERR_CHK(status, "Could not get binding vector");
rpc_ns_binding_export(rpc_c_ns_syntax_default, name, simple_v1_0_s_ifspec, bindings,
  NULL, &status);
ERR_CHK(status, "Could not export bindings");

rpc_ep_register(simple_v1_0_s_ifspec, bindings, NULL, NULL, &status);
ERR_CHK(status, "Could not register endpoint");

printf("Listening for requests\n");

rpc_server_listen(rpc_c_listen_max_calls_default, &status);
}
```

## Esimerkki: simple-server.c 3/3 - varsinainen toiminnallisuus

```
/***** Server Operation *****/

void simple_operation(idl_long_int x, idl_long_int *y)
{
  *y = ++x;
}
```

60