IF3230 – Sistem Terdistribusi
RPC dan Komunikasi

Achmad Imam Kistijantoro (imam@informatika.org)
Judhi Santoso (judhi@informatika.org)
Anggrahita Bayu Sasmita (bayu.anggrahita@informatika.org)

Low level layer

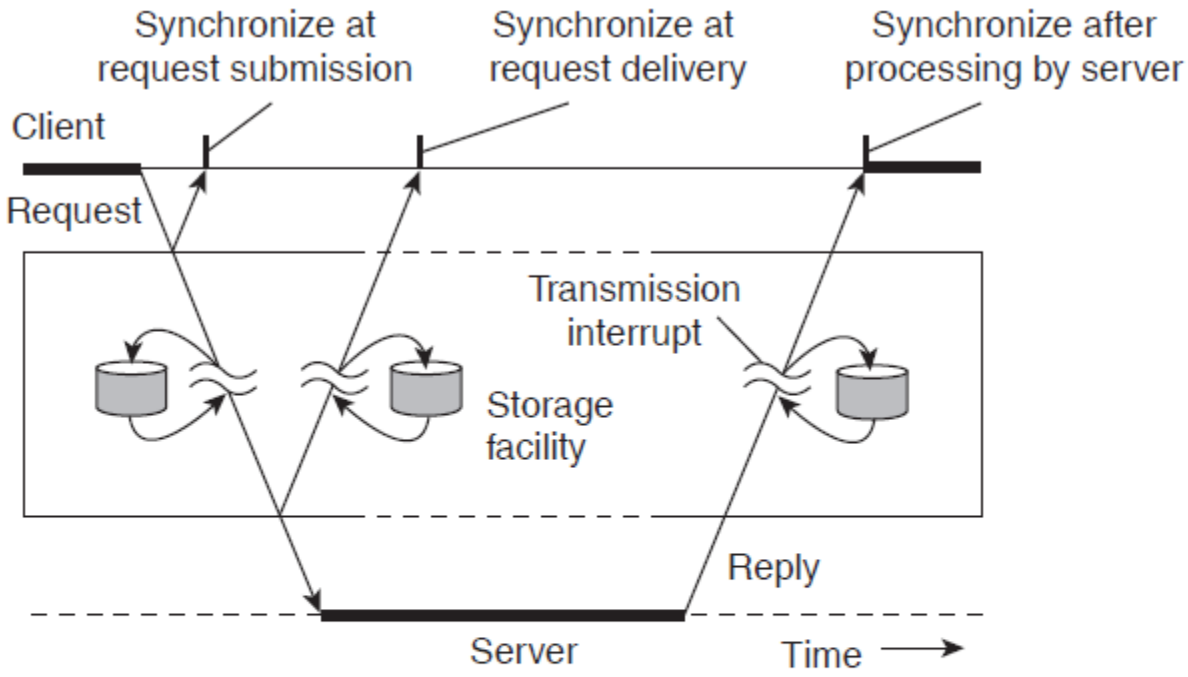
- Untuk sebagian besar sistem terdistribusi, antarmuka terbawah yang digunakan adalah layer network
- Transport layer: transport layer yang sesungguhnya menyediakan fasilitas komunikasi untuk sebagian besar sistem terdistribusi.
- Standard protocol: TCP, UDP

Middleware layer

- Middleware adalah layer yang dibuat untuk menyediakan layanan dan protokol umum yang dapat digunakan berbagai aplikasi.
  - Protokol komunikasi yang beragam
  - (un)marshaling data
  - Naming protocols => memudahkan sharing resources
  - Security protocols => untuk secure communication
  - Scaling mechanisms => seperti replikasi dan caching

  - Sehingga yang tersisa untuk dikembangkan adalah protokol spesifik aplikasi

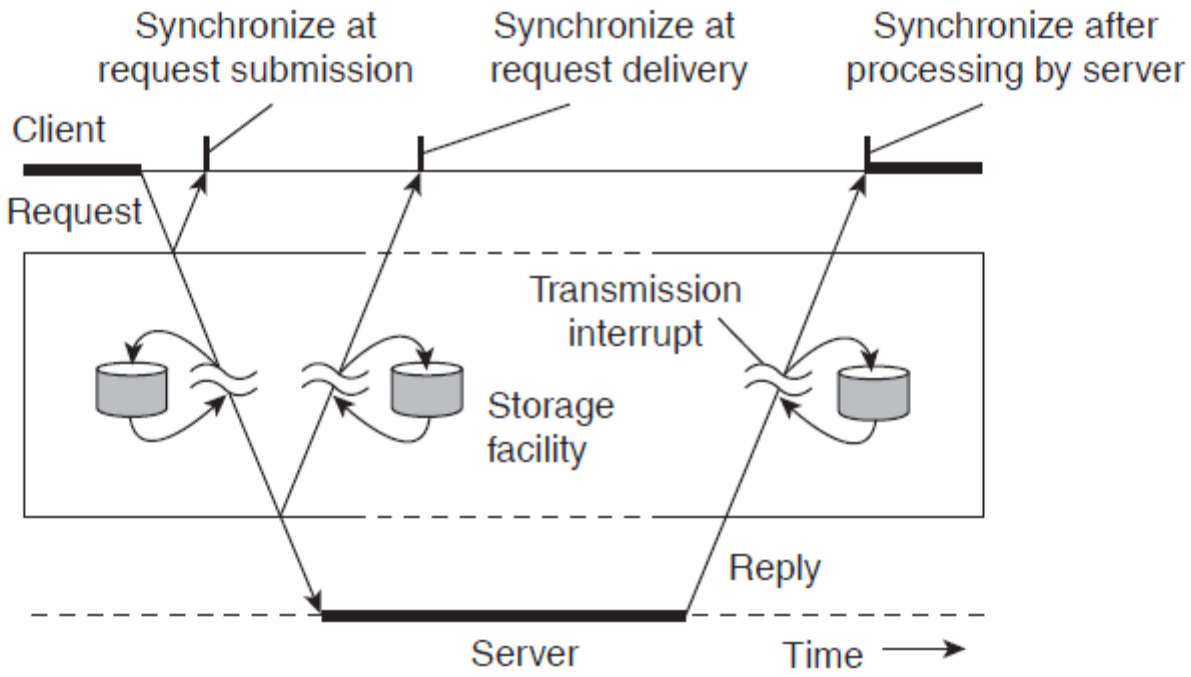
Types of communication



Distinguish

- Transient versus persistent communication
- Asynchrounous versus synchronous communication

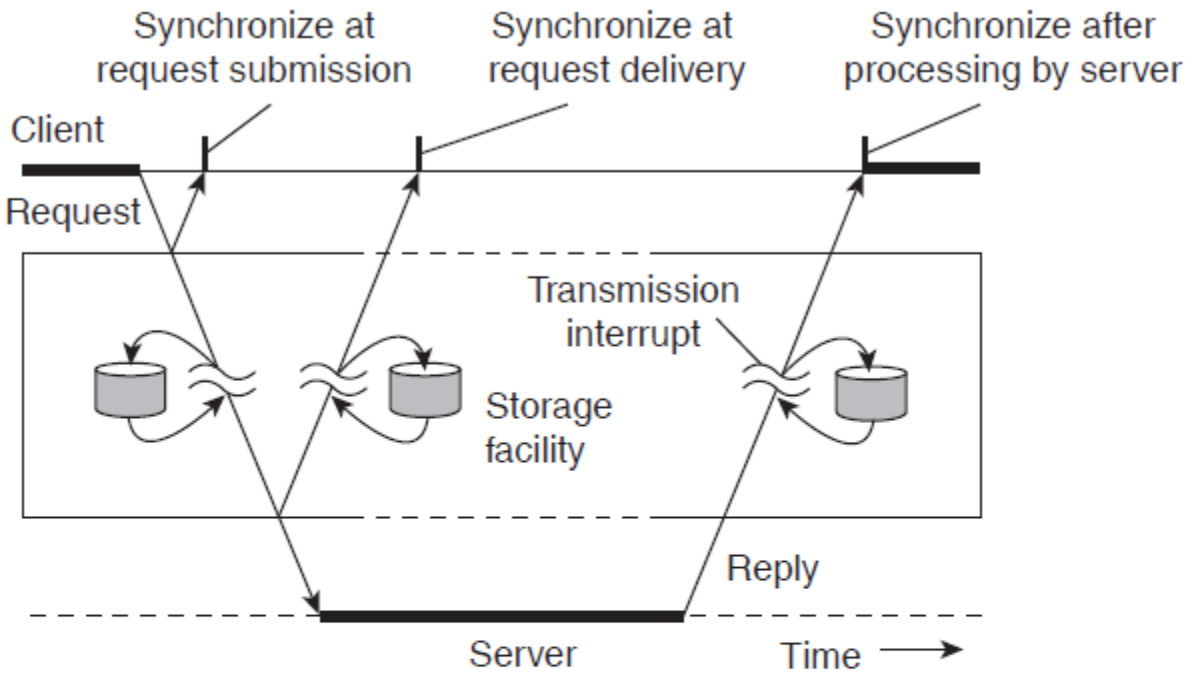
Types of communication



Transient versus persistent

- Transient communication: Comm. server discards message when it cannot be delivered at the next server, or at the receiver.
- Persistent communication: A message is stored at a communication server as long as it takes to deliver it.

Types of communication



Places for synchronization

- At request submission
- At request delivery
- After request processing

Client/Server

Some observations

Client/Server computing is generally based on a model of transient synchronous communication:

- Client and server have to be active at time of commun.
- Client issues request and blocks until it receives reply
- Server essentially waits only for incoming requests, and subsequently processes them

Drawbacks synchronous communication

- Client cannot do any other work while waiting for reply
- Failures have to be handled immediately: the client is waiting
- The model may simply not be appropriate (mail, news)

Messaging

Message-oriented middleware

Aims at high-level persistent asynchronous communication:

- Processes send each other messages, which are queued
- Sender need not wait for immediate reply, but can do other things
- Middleware often ensures fault tolerance

RPC

- Pemrograman client server menggunakan socket cukup sederhana
  - [connect]
  - Read/write
  - [disconnect]
- Namun tetap lebih mudah menggunakan abstraksi procedure/function call
- 1984: Birrell & Nelson mengajukan abstraksi Remote Procedure Call

Bagaimana cara kerja procedure call

- mesin menyediakan instr. call/ret
- Compiler memudahkan programmer dalam hal:
  - Parameter passing
  - Local variable
  - Return data

Procedure call

- Misal:
  - x = f(a, “text”, 5);

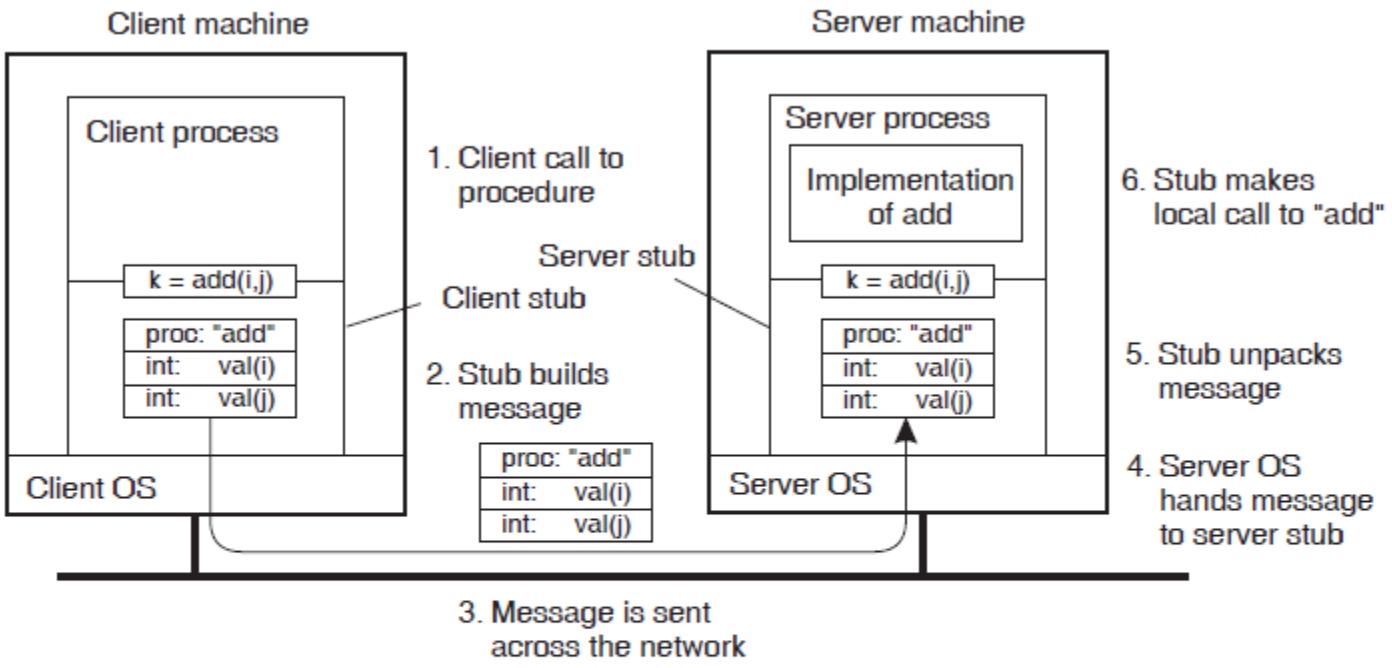
- Compiler akan mem-parse kode ini dan membangkitkan code untuk:
  - push nilai 5 ke stack
  - push address string “text” ke stack
  - Push isi variabel a ke stack
  - Memanggil/call fungsi f
  - Saat kembali dari stack, fungsi f akan menyimpan return value pada register

Remote Procedure Call

- Untuk menyediakan pemanggilan prosedur remote, digunakan mekanisme stub, sehingga pemanggilan prosedur remote dilakukan terhadap stub, dan stub yang menangani pengiriman pesan ke remote server

- Stub: prosedur/fungsi lokal yang memiliki interface sama dengan prosedur remote yang akan dipanggil

Basic RPC Operation



1. Client procedure calls client stub.
2. Stub builds message; calls local OS.
3. OS sends message to remote OS.
4. Remote OS gives message to stub.
5. Stub unpacks parameters and calls server.
6. Server returns result to stub.
7. Stub builds message; calls OS.
8. OS sends message to client's OS.
9. Client's OS gives message to stub.
10. Client stub unpacks result and returns to the client.

RPC

- RPC memberikan interface procedure call ke programmer
- Memudahkan pembuatan aplikasi
  - Kompleksitas kode jaringan disediakan oleh stub function
  - Programmer tidak perlu menangani detil seperti
    - Socket, port number, byte ordering
- RPC setara dengan presentation layer pada 7 layer OSI

Parameter passing

- Pass by value
  - Data dikopi ke jaringan
- Pass by reference
  - Tidak relevan jika tidak ada shared memory

Pass by reference

- Copy item yang di-reference ke message buffer
- Kirim via jaringan
- Unmarshal data di server
- Pass local pointer ke server stub
- Kirim nilai baru ke client

- Setiap struktur kompleks harus diubah menjadi representasi non pointer dan dikirim

Representasi data

- Pada pemanggilan lokal, tidak ada masalah tentang perbedaan format data

- Pada pemanggilan remote, mesin server dapat memiliki
  - Perbedaan byte ordering
  - Perbedaan ukuran integer dan tipe lainnya
  - perbedaan representasi floating point
  - Perbedaan character set
  - Perbedaan aturan alignment

Representasi data

- RPC membutuhkan standar encoding untuk komunikasi antar mesin
- Sun RPC menggunakan XDR (eXternal Data Representation)
- ISO menyediakan standar ASN.1 (Abstract Syntax Notation)

Representasi Data

- Saat dikirim, data dapat dikirim menggunakan
  - Implicit typing: hanya nilai yang dikirim, bukan tipe data atau parameter info, digunakan pada XDR
  - Explicit typing: type dikirim bersama nilai, digunakan pada ASN.1 dan XML

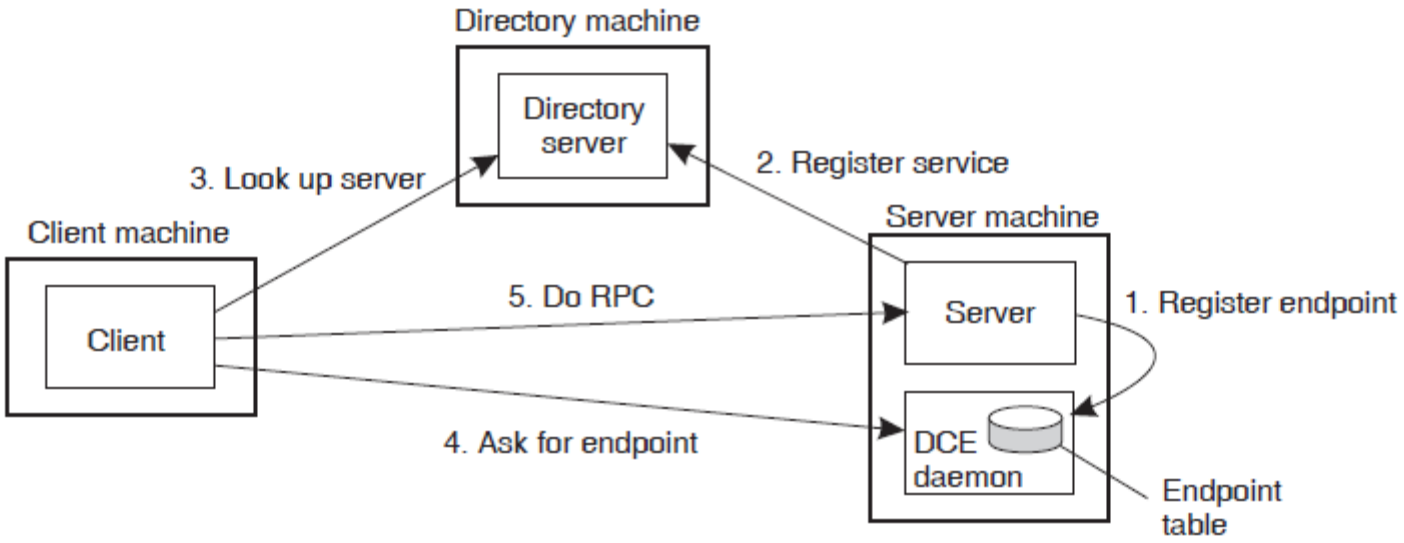
Binding

- Saat pemanggilan, client perlu menentukan host dan proses yang benar untuk sebuah RPC
- Birrell & Nelson (1984) mengajukan mekanisme database terpusat yang menyimpan informasi server dan proses mana yang menyediakan implementasi RPC tertentu
- Pada Sun RPC, database dikelola oleh masing2 host yang menyediakan RPC
  - Banyak host yang menyediakan layanan NFS, namun untuk file system yang berbeda2.

Client to server binding (DCE)

Issues

(1) Client must locate server machine, and (2) locate the server.



Transport protocol

- Beberapa implementasi RPC menggunakan hanya TCP
- Ada yang menyediakan beberapa transport protocol

Penanganan error

- Pemanggilan local call tidak gagal
  - Jika gagal, keseluruhan proses juga akan gagal
- Namun pada RPC, server dapat gagal independen terhadap client/pemanggil
- Aplikasi harus menangani kemungkinan2 kegagalan pemanggilan RPC

- Semantik RPC
  - Pada local call, semantiknya adalah exactly once
- Pada RPC, remote procedure call dapat dipanggil
  - 0 kali: jika server crash sebelum menjalankan kode server
  - 1 kali: jika semua berjalan baik
  - 1 or more: jika terjadi delay atau lost reply, sehingga client mengirim ulang pesan

Semantik RPC

- Umumnya, implementasi RPC menyediakan semantik:
  - At least once
  - At most once
- Perlu memahami karakteristik aplikasi
  - Idempotent: fungsi dapat dipanggil berulangkali tanpa side effect
  - Non idempotent: fungsi tidak boleh dipanggil berulangkali

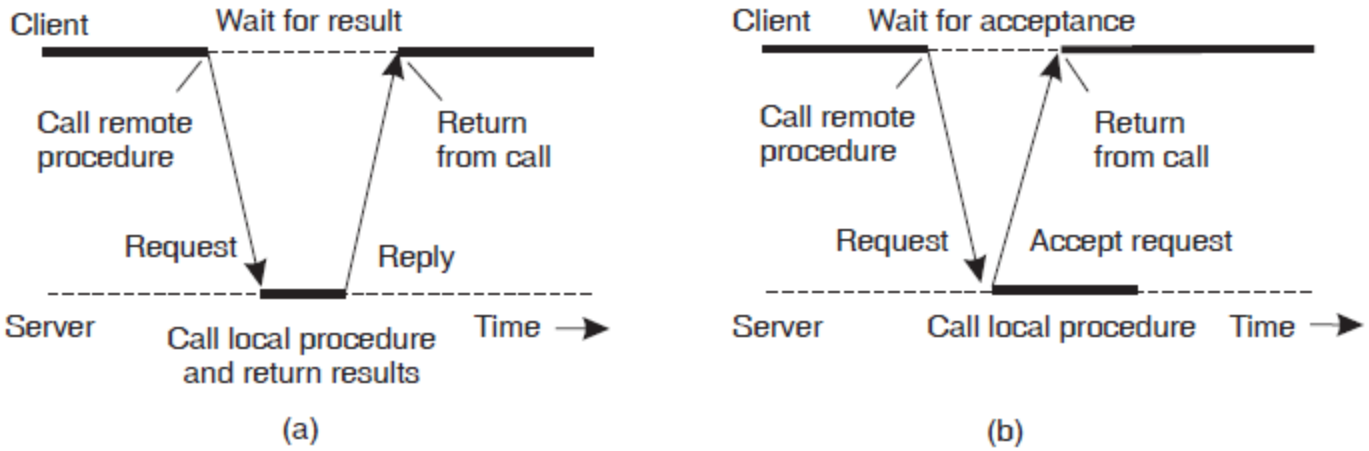
Isu lain

- Kinerja
  - RPC jauh lebih lambat dibandingkan local call
- Security
  - Message visible over network
  - Authenticate client
  - Authenticate server

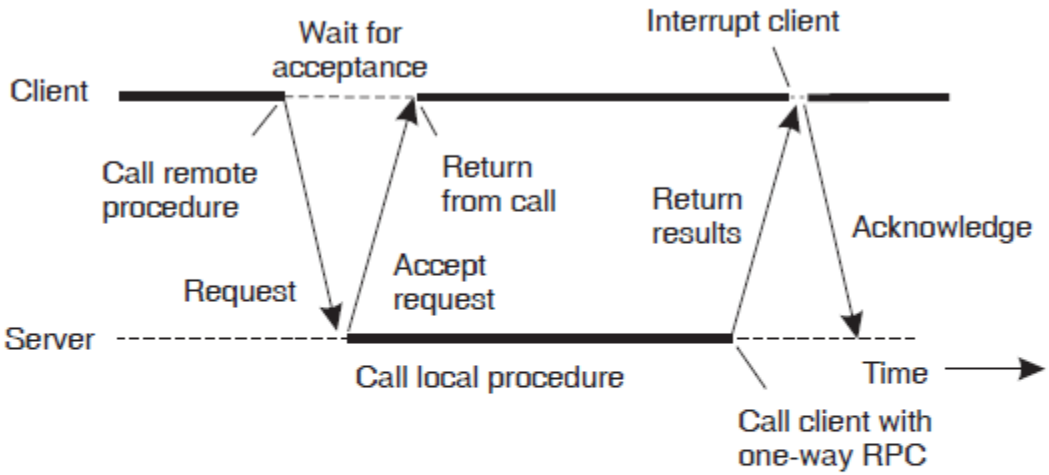
Asynchronous RPC

Essence

Try to get rid of the strict request-reply behavior, but let the client continue without waiting for an answer from the server.



Deferred Synchronous RPC



Variation

Client can also do a (non)blocking poll at the server to see whether results are available.

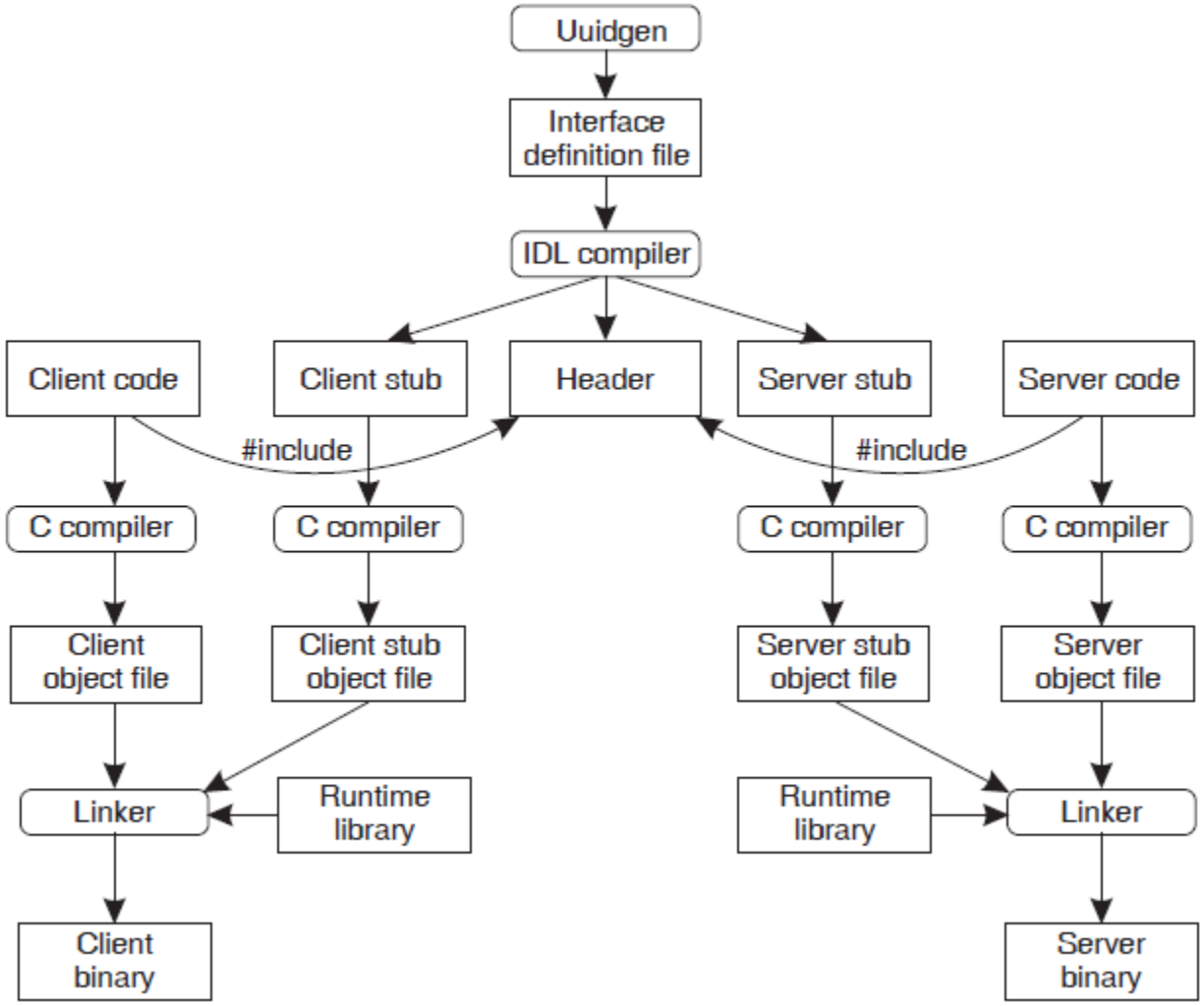
Pemrograman dengan RPC

- Dukungan bahasa
  - Kebanyakan bahasa pemrograman tidak menyediakan dukungan konsep RPC
  - Digunakan kompiler terpisah untuk membangkitkan kode stub

Interface Definition Language

- Memungkinkan programmer untuk menspesifikasikan interface prosedur
- Pre-compiler menggunakan spesifikasi ini untuk membangkitkan kode stub client dan server
  - Marshaling code
  - Unmarshaling code
  - Network transport routines
  - Conformed to defined interface

RPC in practice



ONC (Sun) RPC

- ONC: Open Network Computing
- Dikembangkan oleh Sun (sekarang Oracle)
- RFC 1831 (1995), RFC 5531 (2009)
- Tetap digunakan karena dipakai pada NFS (Network File System)
- Interfaces didefinisikan dengan Interface Definition Language (IDL)
- • IDL compiler: *rpcgen*

RPC IDL

- name.x

```
program GETNAME {
  version GET_VERS {
    long GET_ID(string<50>) = 1;
    string GET_ADDR(long) = 2;
  } = 1; /* version */
} = 0x31223456;
```

Versioning

```
program GETNAME {
  version GET_VERS {
    long GET_ID(string<50>) = 1;
    string GET_ADDR(long) = 2;
  } = 1; /* version */
  version GET_VERS2 {
    long GET_ID(string<50>) = 1;
    string GET_ADDR(string<128>) = 2;
  } = 2; /* version */
} = 0x31223456;
```

```
struct intpair {
  int a;
  int b;
};

program ADD_PROG {
  version ADD_VERS {
    int ADD(intpair) = 1;
  } = 1;
} = 0x23451111;
```

rpcgen

rpcgen name.x => rpcgen -a -C name.x

menghasilkan:

– name.h header

– name_svc.c server stub (skeleton)

– name_clnt.c client stub

– [name_xdr.c] optional XDR conversion routines

• nama fungsi diturunkan dari nama fungsi pada IDL dan versinya

• Client menerima *pointer to result*

– dapat digunakan untuk mendeteksi pemanggilan yang gagal (null return)

– Reminder: C doesn't have exceptions!

Kode di sisi server

```
/*
 * This is sample code generated by rpcgen.
 * These are only templates and you can use them
 * as a guideline for developing your own functions.
 */

#include "add.h"

int *
add_1_svc(intpair *argp, struct svc_req *rqstp)
{
        static int  result;

        /*
         * insert server code here
         */

        return &result;
}
```

```
int *
add_1_svc(intpair *argp, struct svc_req *rqstp)
{
        static int  result;

        printf("add server called\n");
        result = argp->a + argp->b;
        return &result;
}
```

Sisi server

Saat server dijalankan:

– Server stub membuat socket dan binds ke any available local port

– memanggil fungsi pada RPC library:

• *svc_register to register program#, port #*

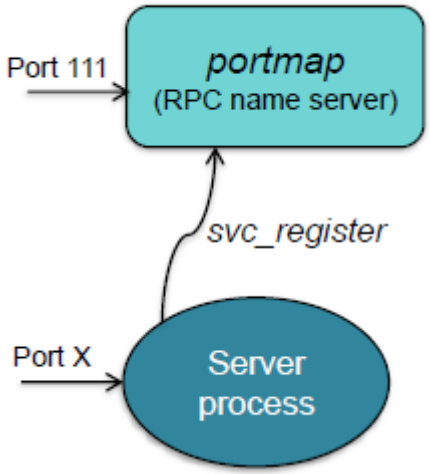
• mengontak portmap **(rpcbind on SVR4):**

– Name server

– Keeps track of

{program #, version #, protocol} => port # bindings

– Server then listens and waits to accept connections



Sisi client

Client calls clnt_create with:

– Name of server

– Program #

– Version #

– Protocol#

• *clnt_create contacts port mapper on that server to get*

the port for that interface

– early binding – done once, procedure call

• Communications

– Marshaling to XDR format (eXternal Data Representation)