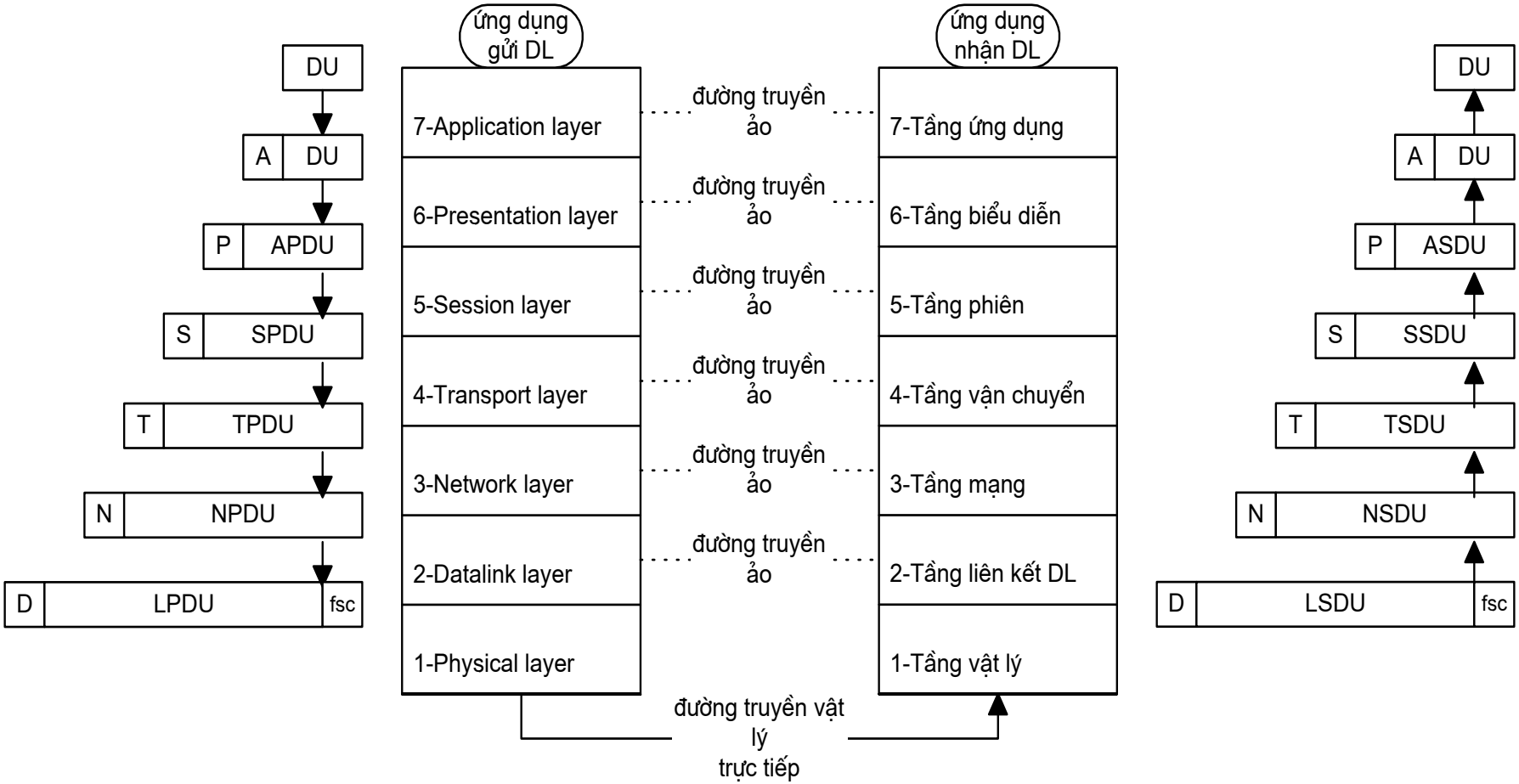
Liên lạc trong hệ thống phân tán

- Truyền thông liên mạng
- Gọi thủ tục từ xa
- Liên lạc theo nhóm

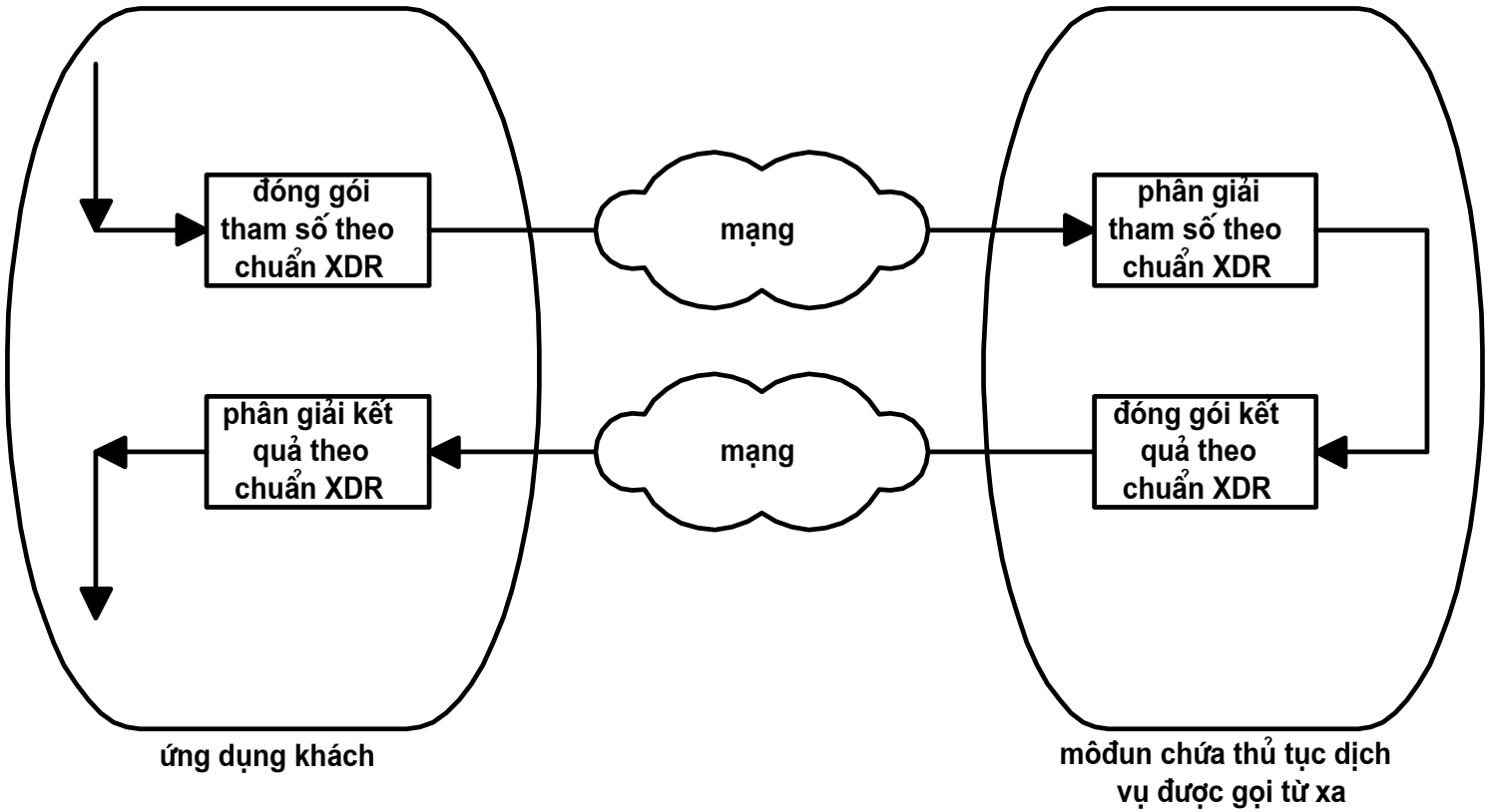
Truyền thông liên mạng



Lập trình socket theo mô hình client-server truyền thống

- Khái niệm giao diện socket
- Xem phần kèm theo về lập trình socket

Cài đặt khái niệm RPC



Các phương pháp trao đổi tham số

- call by value
- call by reference
- call by copy/restore

Cơ chế trao đổi tham số

- thủ tục client gọi client stub
- client stub tạo thông báo và bẫy HĐH
- HĐH gửi thông báo qua mạng
- HĐH bên kia nhận thông báo và trao cho server stub
- server stub giải mã tham số và gọi thủ tục server tại đó
- server chạy và trả lại kết quả cho server stub
- server stub đóng gói kết quả vào thông báo và bẫy HĐH
- HĐH bên server gửi thông báo lại cho HĐH bên client, qua đó đưa thông báo cho client stub
- client stub giải mã kết quả và trả lại cho thủ tục client

Xử lý tham trỏ

- Phương thức mô phỏng call by reference là call by copy/restore
- Nhược điểm:
  - Khó trao đổi dữ liệu có cấu trúc tham trỏ phức tạp (vd: đồ thị)
  - Giải pháp: thực hiện cơ chế gọi thủ tục từ xa lại phía khách để lấy giá trị

Kết nối động giữa client-server

- Tạo ra một dịch vụ cung cấp địa chỉ, cổng của server cho client (binder)
  - cho phép nhiều server tham gia vào dịch vụ
  - fault-tolerance
- server sẽ đăng ký các dịch vụ với binder
- client sẽ xin sử dụng các dịch vụ đã đăng ký tại binder

Ngữ nghĩa của RPC trong xử lý lỗi

- các khả năng gặp lỗi:
  - client không thể định vị server
  - thông báo yêu cầu từ client bị thất lạc
  - thông báo trả lời từ server bị thất lạc
  - server gục sau khi đã nhận thông báo
  - server gục sau khi đã trả lời thông báo

Lỗi định vị server

- Cách 1: trả lại mã lỗi như bình thường
  - không áp dụng được với các hàm tính giá trị
- Cách 2: tạo thủ tục bẫy lỗi (exception)
  - nhiều ngôn ngữ lập trình không có khái niệm này
  - xu hướng hiện nay: các ngôn ngữ hướng đối tượng đều hỗ trợ xử lý bẫy lỗi (xception)

Thất lạc thông báo yêu cầu

- áp dụng cơ chế truyền đảm bảo với xác nhận và với thời gian timeout

Thất lạc thông báo trả lời

- idempotent call:
  - không cần hành động gì
- non-idempotent call:
  - vd: rút tiền từ tài khoản ngân hàng
  - giải pháp: tạo ra mã số trình tự cho mỗi lời gọi thủ tục từ xa

Lỗi server gục

- Các tình huống:
  - gục sau khi đã chạy xong thủ tục
  - gục sau khi nhận được thông báo
- Các ngữ nghĩa xử lý:
  - at least once
  - at most once
  - không xác định
- Câu hỏi: tại sao không xảy ra đối với hệ thống tập trung?

Lỗi client gục

- Lỗi sinh ra orphan
- Giải pháp:
  - extermination: dùng log file và kill-off
  - reincarnation: dùng epoch number
  - gentle reincarnation: locate lại và epoch
  - expiration: dùng timeout

Giao thức hỗ trợ RPC

- dùng chồng giao thức có sẵn: ví dụ IP/UDP
  - dễ dàng và nhanh chóng xây dựng
  - có quá nhiều tiêu đề để đóng/bóc
- xây dựng chồng giao thức riêng
  - tăng hiệu suất hệ thống
  - không chia sẻ ngôn ngữ chung với internet

Nhược điểm của RPC

- Sử dụng biến tổng thể:
  - vd: biến errno dùng trả lỗi giữa thủ tục
- Xác định độ dài dữ liệu:
  - vd: các tham trỏ của C
- Nhược điểm của mô hình client-server
  - không áp dụng tổng quát được với mọi mô hình

Nhược điểm mô hình client-server

- Các ví dụ:
  - *sort < f1 > f2* hoặc *grep rat < f3 > f4*
  - *grep rat < f5 | sort > f6*
- Mô hình read-driven:
  - *p1 <f1 | p2 | p3 > f2*
  - p1 yêu cầu đọc f1 từ filer...
- Mô hình write-driven:
  - p3 yêu cầu ghi ra filer...

Liên lạc theo nhóm

- Các phương thức:
  - broadcast
  - multicast
  - unicast lần lượt theo danh sách

Thông báo atomic

- Hoặc là tất cả các thành viên (đang chạy) nhận được thông báo hoặc là không ai cả
- Ví dụ cài đặt:
  - sử dụng broadcast với timeout và truyền lại
  - mỗi máy lần đầu nhận thông báo sẽ broadcast tới tất cả các thành viên khác

Multicast liên mạng

- Multicast trên LAN có thể tạo vòng lặp thông qua các gateway
  - ví dụ: các máy bridges tạo thành chu trình

Thứ tự thông báo

- phương thức đồng bộ chặt:
  - thời gian truyền bằng không
  - không thực tế
- đồng bộ lỏng:
  - có độ trễ thời gian truyền. Sự cố: thứ tự các thông báo có thể bị đảo lộn
  - giải pháp ABcast: tạo thứ tự thực sự
- khái niệm đồng bộ ảo:
  - dựa trên phụ thuộc “nhân-quả” giữa các thông báo
  - chỉ cần đảm bảo sự phụ thuộc logic giữa các lời gọi thông báo

ABcast - lỏng

- Mỗi máy lưu trữ timestamp lớn nhất đã biết.
- Máy gốc gửi tới các máy khác thông báo với timestamp mới
- Các máy trả lời và đề xuất timestamp lớn nhất của mình
- Máy gốc gửi lại thông báo quyết định số hiệu timestamp lớn nhất thu thập được

CBcast – ảo

- Sử dụng vector timestamp *V* chứa giá trị timestamp lớn nhất nhận được từ các máy tương ứng trong nhóm.
- Mỗi thông báo kèm vector của máy gửi: ký hiệu *L*
- Thông báo từ máy j được chấp nhận nếu:
  - *Vj = Lj + 1*
  - *Vi <= Li với i<>j*
- Các trường hợp còn lại: chờ, vì phụ thuộc logic vào 1 thông báo chưa nhận được

Ví dụ ABcast

Trạm 1: TS=14	Trạm 2: TS=15	Trạm 3: TS=16

M3	M1	M2	…
15.1	16.1	17.1	…
?	?	?	…

M2	M1	M3	…
16.2	17.2	18.2	…
?	?	?	…

M1	M3	M2	…
17.3	18.3	19.3	…
?	?	?	…

M3	M2	M1	…
15.1	17.1	17.3	…
?	?	!	…

M2	M1	M3	…
16.2	17.3	18.2	…
?	?	?	…

M1	M3	M2	…
17.3	18.3	19.3	…
?	?	?	…

M3	M1	M2	…
15.1	17.3	19.3	…
?	!	!	…

M1	M3	M2	…
17.3	18.2	19.3	…
!	?	!	…

M3	M2		…
18.3	19.3		…
?	!		…

Ví dụ CBcast

trạm 0 phát	trạm 1 ok	trạm 2 chờ	trạm 3 ok	trạm 4 chờ	trạm 5 ok
4	3	3	3	2	3
6	6	5	7	6	7
8	8	8	8	8	8
2	2	2	2	2	3
1	1	1	1	1	1
5	5	5	5	5	5

Ví dụ với Sun RPC

Xây dựng ƯD RPC đơn giản

- Server:
  - Đăng ký các thủ tục sẽ được gọi từ xa. Việc đăng ký cần chỉ ra:
    - số hiệu chương trình server
    - số hiệu phiên bản chương trình server
    - số hiệu thủ tục được đăng ký
    - thủ tục giải mã tham số và thủ tục mã kết quả
  - Gọi thủ tục chạy lắng nghe

... xây dựng ƯD RPC đơn giản

- Client:
  - Gọi thủ tục từ xa bên server theo
    - số hiệu chương trình
    - số hiệu phiên bản
    - số hiệu thủ tục
    - thủ tục mã hoá tham số và giải mã kết quả
    - danh sách tham số

Công cụ trợ giúp RPCGEN

- rpcgen: là chương trình dịch từ RPCL sang C
- Ví dụ:
  - chạy: *rpcgen rpcprog.x*
  - kết quả:
    - rpcprog_clnt.c - mã sinh cho client
    - rpcprog_svc.c - mã sinh cho server
    - rpcprog_xdr.c - mã sinh cho XDR filters(nếu cần)
    - rpcprog.h - header cho XDR filters

...công cụ trợ giúp RPCGEN

- Ví dụ một đoạn mã RPCL, tệp msg.x:

```
program MessageProg {
  version PrintMessageVers {
   int PrintMessage(string)=1;
  } = 1;
} = 0x20000001;
```

khai báo 3 số hiệu: chương trình, phiên bản và thủ tục. Được sử dụng để sinh mã cho cả server và client.

Các mức giao diện

- Các thủ tục hỗ trợ lập trình RPC được chia thành các mức từ dễ tới khó sử dụng và cũng đồng nghĩa với khả năng kiểm soát kết nối từ thấp tới cao:
  - Mức đơn giản (simplified)
  - Mức cao (top)
  - Mức trung (intermediate)
  - Mức chuyên sâu (expert)
  - Mức đáy (bottom)
- Phần giới thiệu dưới đây chỉ mô tả mức đơn giản.

Mức đơn giản

- Phía client sử dụng các hàm
  - rpc_call hoặc callrpc để gọi thủ tục từ xa
- Phía server sử dụng các hàm
  - rpc_reg hoặc registerrpc để đăng ký thủ tục sẽ được gọi từ xa
  - svc_run để chạy và chờ yêu cầu từ client

rpc_call

```
int rpc_call (
char *host              /* tên máy của server */,
u_long prognum          /* số hiệu chương trình */,
u_long versnum          /* phiên bản chương trình */,
u_long procnum          /* số hiệu thủ tục */,
xdrproc_t inproc        /* XDR filter mã hoá tham số */,
char *in                /* trỏ tới xâu tham số */,
xdr_proc_t outproc      /* XDR Filter giải mã tham số */,
char *out               /* trỏ tới xâu chứa kết quả */,
char *nettype           /* chọn dịch vụ truyền dữ liệu */
);
```

rpc_reg

```
int rpc_reg(
u_long prognum          /* số hiệu chương trình */,
u_long versnum          /* phiên bản chương trình */,
u_long procnum          /* số hiệu thủ tục */,
char *procname          /* tên thủ tục RPC */,
xdrproc_t inproc        /* XDR Filter giải mã tham số */,
xdrproc_t outproc       /* XDR Filter mã hoá kết quả */,
char *nettype     /* chọn dịch vụ truyền dữ liệu */
);
```

Trao đổi dữ liệu theo XDR

- Có các hàm chuyển đổi chuẩn:

  xdr_int(), xdr_u_long(), xdr_long(), xdr_bool(), xdr_short(), xdr_doulbe, xdr_u_short(), xdr_char(), xdr_u_char(), xdr_void(), xdr_string() ...

- Chỉ có 1 tham số được trao đổi. Nếu có nhiều tham số, phải gói vào một struct và mã hoá lại thành 1 xâu duy nhất

... trao đổi dữ liệu theo XDR

- Ví dụ trao đổi 1 struct:

struct simple { int a; short b; } simple;

XDR filter là hàm xdr_simple sẽ được đặt trong tệp rpcprog_xdr.c:

```
#include <rpc/rpc.h>
#include "simple.h"
bool_t xdr_simple(XDR *xdrsp, struct simple *simplep) {
    if (!xdr_int(xdrsp, &simplep->a)) return (FALSE);
    if (!xdr_short(xdrsp, &simplep->b)) return (FALSE);
    return (TRUE);
}
```

Ứng dụng: đọc thư mục từ xa

- Tệp remotedir.h

```
#define DIR_SIZE 8192
#define DIRPROG ((u_long) 0x20000001) /* số hiệu ch/trình */
#define DIRVERS ((u_long) 1)                    /* phiên bản */
#define READDIR ((u_long) 1)                    /* số hiệu thủ tục */
```

- Tệp remotedir_xdr.c

```
#include <rpc/rpc.h>
#include “remotedir.h"
bool_t xdr_dir(XDR *xdrs, char *objp) {
    return ( xdr_string(xdrs, &objp, DIR_SIZE) );
}
```

Đọc thư mục từ xa: phía server

- Tệp remotedir_svc.c

```
#include <rpc/rpc.h>
#include “remotedir.h"
void main() {
    extern bool_t xdr_dir();
    extern char * read_dir();
    registerrpc(DIRPROG, DIRVERS, READDIR,
                    read_dir, xdr_dir, xdr_dir);
    svc_run();
}
```

tất nhiên còn phải viết thủ tục read_dir() nữa!

Đọc thư mục từ xa: phía client

- Tệp remotedir_clnt.c

```
#include <stdio.h>
#include <strings.h>
#include <rpc/rpc.h>
#include “remotedir.h"
void main (argc, argv) int argc; char *argv[]; {
    char dir[DIR_SIZE];
    strcpy(dir, argv[2]);
    read_dir(argv[1], dir);            /* read_dir(host, directory) */
    printf("%s\n", dir);
    exit(0);
}
```

...phía client

```
read_dir(host, dir) char *dir, *host; {
    extern bool_t xdr_dir();
    enum clnt_stat clnt_stat;
    clnt_stat = callrpc ( host, DIRPROG, DIRVERS, READDIR,
                          xdr_dir, dir, xdr_dir, dir);
    if (clnt_stat != 0) clnt_perrno (clnt_stat);
}
```