Contents

[Introduction 2](#_Toc10615682)

[Model & Architecture 3](#_Toc10615683)

[Processes & Threads 11](#_Toc10615684)

[Communication 19](#_Toc10615685)

[Naming 35](#_Toc10615686)

# Introduction

Định nghĩa hệ phân tán: là hệ các máy tính độc lập được kết nối với nhau bằng một hạ tầng truyền thông cung cấp dịch vụ cho người sử dụng như một máy tính duy nhất

Hệ phân tán >< Hệ tính toán khắp nơi và di động (Ubiquitous computing)

Hệ phân tán xuất hiện nhiều máy tính, còn cái kia xuất hiện như không có máy tính nào cả.

4 tính chất của hệ phân tán:

* Chia sẻ tài nguyên
* Tính trong suốt
* Tính mở
* Tính co dãn
* Việc chia sẻ thông tin trên phần cứng thực tế là chia sẻ các tài nguyên phần cứng của máy tính
* Như vậy, chia sẻ tài nguyên là một đặc điểm đặc trưng của hệ thống phân tán
* Hệ thống phân tán phải chấp nhận các thiết bị tính toán của các nhà sản xuất khác nhau với các nền tảng khác nhau
* VD: trong hệ thống WWW, nếu có một Windows server bị lỗi ta có thể thay thế nó bằng một server chạy Mac hay Linux…=> Tính mở
* Tất cả các máy tính của hệ phân tán phải làm việc như một máy tính duy nhất đối với người sử dụng
* Tất cả các thay đổi trong hệ thống đều mang tính trong suốt đối với người sử dụng (các sự thay đổi đằng sau không làm ảnh hưởng tới trải nghiệm của người dung)
* Chia sẻ tài nguyên: các tài nguyên trong máy tính thường được quản lý bởi một chương trình quản lý tài nguyên. Chương trình này có nhiệm vụ nhận các yêu cầu tài nguyên do chương trình khác gửi đến, chuyển các yêu cầu truy cập này đến tài nguyên vật lý, rồi nhận phản hồi từ tài nguyên vật lý, và cung cấp ngược lại cho các chương trình.
  + Ưu điểm:
    - Tiết kiệm chi phí đầu tư: máy tính của người sử dụng có thể sử dụng các tài nguyên không nằm trong máy tính của mình, vd máy in, máy scan, webcam…do đó tiết kiệm chi phí đầu tư trên từng người sử dụng.
    - Tăng khả năng sẵn sàng của hệ thống: khi tài nguyên trên một máy tính không sẵn sàng, người dùng có thể sử dụng tài nguyên trên một máy tính khác.
  + Nhược điểm:
    - Nguy cơ về lỗ hổng bảo mật, lộ thông tin nhạy cảm
* Tính trong suốt:
  + Trong suốt truy cập (access): che giấu những sự khác biệt trong việc biểu diễn dữ liệu và các cách thức truy cập một tài nguyên là giống nhau trong mọi trường hợp.
  + Trong suốt vị trí (location): vị trí của tài nguyên không bị người sử dụng nhìn thấy
  + Trong suốt tái định cư (migration): trong quá trình hoạt động, nếu vì một lý do nào đó mà tài nguyên bị dịch chuyển, thì việc dịch chuyển đó cũng phải trong suốt đối với người sử dụng.
  + Trong suốt tái quy hoạch (relocation): mức cao hơn của migration, che giấu việc dịch chuyển tài nguyên đến một nơi khác TRONG KHI NÓ ĐANG ĐƯỢC SỬ DỤNG.
  + Trong suốt sao lưu (replication): che giấu việc lưu thông tin từ một máy ở xa vào bản sao cục bộ (local)
  + Trong suốt tính đồng thời (concurrency): che giấu việc tài nguyên đang được chia sẻ bởi nhiều người dùng
  + Trong suốt lỗi (failure): che giấu lỗi và quá trình phục hồi của một tài nguyên
* Tính trong suốt <> hiệu năng
* Đôi khi không cần trong suốt tuyệt đối
* Tính mở:
  + Hệ phân tán bao gồm các thành phân riêng biệt được kết nối với nhau
  + Hệ thống mở là hệ thống cho phép các thành phần từ các nhà sản xuất khác nhau có thể kết nối với nhau, đồng thời cũng cho phép các thành phần khác bổ sung vào hệ thống.
  + Các thành phần kết nối với nhau bằng các dịch vụ: cú pháp, thủ tục – hay còn gọi là giao thức. Những giao thức này sau đó được trừu tượng hoá thành những giao diện để cung cấp dịch vụ.
  + Điều kiện để nhà cung cấp dịch vụ và người sử dụng có thể tương tác với nhau là sử dụng cùng một giao diện.
  + Điều kiện để cài đặt một giao diện:
    - Đầy đủ (nếu không, nhà cung cấp sẽ cài đặt những phần thiếu sót sao cho người sử dụng không nhận ra)
    - Trung lập (không phụ thuộc vào bất kì một nhà cung cấp hay người dùng nào)
    - Sử dụng Interface definition language (IDL) để đặc tả giao diện
* Tính co dãn:
  + Co dãn về mặt quy mô: số lượng máy tính, số lượng người sử dụng tăng…
  + Co dãn về mặt địa lý: đảm bảo kết nối LAN giống như kết nối trên WAN
  + Co dãn về mặt tổ chức: số lượng và quan hệ giữa các đơn vị tổ chức (vd: tên miền)

# Model & Architecture

Một số từ khoá cơ bản:

* Kiến trúc phần mềm: là cách thức mà các thành phần phần mềm tổ chức và tương tác với nhau.
* Kiến trúc hệ thống: là bản thể cuối cùng của kiến trúc phần mềm.
* Kiểu kiến trúc: là cách mà các thành phần kết nối với nhau.
* Thành phần: là một đơn vị module với các giao diện đã được định nghĩa và cung ứng trước. Các thành phần có thể được thay thế trong môi trường của chúng.
* Kết nối: là cơ chế trung gian cho việc trao đổi thông tin, phối hợp, kết hợp giữa các thành phần.
* Server: là một tiến trình triển khai một dịch vụ đặc biệt (VD: dịch vụ hệ thống tệp, dịch vụ cơ sở dữ liệu, vv…).
* Client: là một tiến trình yêu cầu một dịch vụ từ server bằng cách gửi đi một yêu cầu và sau đó chờ câu trả lời từ server.

Phần 1: Giới thiệu về kiến trúc hệ thống và các mô hình

1. Kiểu kiến trúc

Định nghĩa:

* + Là kiến trúc dưới khía cạnh các thành phần riêng biệt và cách thức chúng kết nối với nhau trong hệ thống.
  + Hệ thống được chia ra thành các thành phần.
  + Thành phần là một đơn vị module được định nghĩa trước và cung ứng giao diện, có thể được thay thế trong môi trường của chúng.
  + Các thành phần được kết nối với nhau tạo thành một hệ thống.

Để phân biệt các kiểu kiến trúc liên quan đến 3 yếu tố:

* + Cách các thành phần kết nối với nhau
  + Cách dữ liệu được trao đổi giữa các thành phần
  + Các thành phần đó được cấu hình thế nào để tạo nên một hệ thống

Từ đó, ta có 4 kiểu kiến trúc chính:

* + Kiến trúc phân tầng
  + Kiến trúc hướng đối tượng
  + Kiến trúc hướng dữ liệu
  + Kiến trúc hướng sự kiện
  1. Kiến trúc phân tầng
  + Các chức năng trong hệ pt được phân rã thành các chức năng con.
  + Các chức năng con được thực hiện bởi các module phần mềm. Các thực thể phần mềm của các hệ thống khác nhau tương tác với nhau.
  + Các module phần mềm trong cùng một hệ thống phối hợp với nhau và giao tiếp với nhau để thực hiện một chức năng chung.
  + Để đơn giản hoá hệ thống, ta cần phải tối thiểu hoá các kết nối giữa các module.
* Từ đó sinh ra mô hình kiến trúc phân tầng.

Ý tưởng cơ bản của KT phân tầng:

Các thành phần được tổ chức theo phong cách phân tầng. Một thành phần ở tầng L(i) được cho phép gọi các thành phần ở tầng dưới L(i-1). Mô hình này được áp dụng rất rộng trong cộng đồng mạng.

Các lợi ích của kiến trúc phân tầng:

* Đối với các hệ thống phức tạp, người ta áp dụng nguyên lý chia để trị.
* Kiến trúc này cho phép xác định rõ nhiệm vụ của mỗi bộ phận và quan hệ giữa chúng.
* Cho phép dễ dàng bảo trì và nâng cấp hệ thống. Thay đổi bên trong một bộ phận không làm ảnh hưởng tới các bộ phận khác. VD: chuyển đổi từ IPv4 => IPv6, do mạng internet được thiết kế theo kiến trúc phân tầng nên việc chuyển đổi này chỉ ảnh hưởng tới tầng mạng chứ không ảnh hưởng tới các tầng khác.

Các thực thể trong mạng cần phải thống nhất giao thức chuẩn tương ứng với mỗi tầng để có thể trao đổi thông tin với nhau.

Giao thức: tập hợp tất cả các quy tắc, quy ước để các máy tính trong mạng có thể giao tiếp với nhau. Như vậy, các máy trong mạng muốn kết nối với nhau thì phải có chung một giao thức.

Các loại giao thức:

* Giao thức hướng kết nối: trước khi tiến hành trao đổi thông tin, dữ liệu, hai thực thể phải thiết lập một kênh truyền. Việc làm này cho phép 2 thực thể có thể kiểm soát được đường truyền. VD: TCP, SPX
* Giao thức không kết nối: không cần thiết lập kênh truyền. Dữ liệu thường dưới dạng datagram. VD: UDP
* Giao thức tin cậy: dữ liệu được truyền đi tuần tự. Nếu nhận thành công thì nơi nhận phải gửi tín hiệu báo nhận (acknowledge). Ngược lại là giao thức không tin cậy.
  1. Kiến trúc hướng đối tượng
* Các thành phần là các đối tượng, các kết nối là các lời gọi thủ tục. Nói cách khác, các thành phần được kết nối với nhau qua cơ chế lời gọi thủ tục.
* Đối tượng thường có hai loại: đối tượng khách (object client) và đối tượng chủ (object server). Vì vậy người ta nói kiến trúc này có điểm tương đồng với kiến trúc client-server.
* Kết nối giữa các đối tượng là kết nối lỏng. Có nghĩa là nó cho phép các đối tượng không đồng nhất. Chúng có thể khác nhau về các thuộc tính, các phương thức, nhưng vẫn có khả năng trao đổi thống tin với nhau. VD: COBRA
  1. Kiến trúc hướng dữ liệu
* Các thành phần trao đổi thông tin với nhau thông qua các sự kiện.
* Các sự kiện chứa các thông tin cần trao đổi và có thể kích hoạt các thao tác trong tiến trình.
* Kiến trúc này có thể hoạt động theo mô hình điểm-điểm hoặc mô hình trục quảng bá sự kiện.
* VD: mô hình thuê bao xuất bản

Ý tưởng: sau khi tiến trình xuất bản một sự kiện, thì middleware đảm bảo chỉ những tiến trình đã đăng kí thuê bao gói sự kiện vừa xuất bản mới nhận được các sự kiện đó.

* VD về mh thuê bao xuất bản: cơ chế truyền instance trong hệ thống android:

Các component trước khi được chạy luôn phải đăng ký có khả năng đáp ứng các instance loại nào. Khi một component gửi đi một instance cho hệ thống, middleware sẽ đảm bảo chỉ những component có đăng ký trước kiểu instance đó mới nhận được.

* Lợi thế: kết nối lỏng, các thành phần trong hệ thống không cần phải đồng bộ. Chúng có thể khác nhau về nền tảng phần cứng/phần mềm.
  1. Kiến trúc hướng sự kiện
* Các tiến trình có thiết kế tách rời theo thời gian. Chúng trao đổi thông tin qua một kho dữ liệu dùng chung => 2 tiến trình không cần phải chạy đồng thời để trao đổi thông tin.
* Để gửi và nhận thông tin từ kho dữ liệu dùng chung, các tiến trình có thể sử dụng giao diện SQLite, hoặc các truy vấn csdl, thay vì dùng các tham chiếu, ví dụ như địa chỉ cụ thể của một vùng nhớ.

1. Mô hình trong hệ PT

* Là kết quả của việc trừu tượng hoá hệ thống PT.
* Chỉ tập trung vào một khía cạnh của hệ PT.
* Có 3 loại mô hình:
  + Mô hình tương tác: được dùng để đánh giá tính tương tác của hệ thống.
  + Mô hình lỗi: được sử dụng mỗi khi cần đánh giá tính chịu lỗi của hệ thống.
  + Mô hình bảo mật: dùng để đánh giá độ bảo mật của hệ thống.

Phần 2: Các kiến trúc hệ thống

1. Kiến trúc tập trung
   1. Kiến trúc Client-Server:

* Client: là một tiến trình yêu cầu dịch vụ từ phía server bằng cách gửi một yêu cầu, và chờ server trả lời, nhận kết quả và hiển thị cho ng dùng
* Server: là một tiến trình triển khai một dịch vụ cụ thể. Công việc của nó là lắng nghe yêu cầu, xử lý và trả lời.
* Tương tác giữa client và server có thể sử dụng giao thức hướng kết nối hoặc không hướng kết nối.
* Vấn đề:
  + Đăng ký server: cơ chế các server phải thực hiện để client biết được vị trí để gửi yêu cầu. VD: DNS hoặc dịch vụ thư mục.
  + Có thể lặp lại yêu cầu hay không: nảy sinh khi client không nhận được phản hồi từ server. Client sẽ ko biết là do server chưa nhận được yêu cầu, hay thông điệp báo nhận bị mất trên đường truyền. Lúc này, tuỳ thuộc vào tính chất hoặc yêu cầu của dịch vụ thì client có hoặc không gửi lại yêu cầu.
  + Có nên sử dụng bộ nhớ trạng thái hay không.

Phân tích vấn đề của kiến trúc Client-server:

Hướng kết nối hay không:

Với giao thức không kết nối, khi client thực hiện một dịch vụ sẽ tiến hành đóng gói thông điệp, rồi chỉ rõ dịch vụ mong muốn cùng với dữ liệu đầu vào cần thiết. Đối với server thì luôn chờ các yêu cầu. Khi nhận được yêu cầu thì xử lý chúng, đóng gói kết quả vào trong một thông điệp trả lời, sau đó gửi trả lại cho client (chú ý: giao thức này k có bước tạo kênh truyền)

* Ưu điểm: hiệu quả và nhanh
* Nhược điểm: Thiếu tính đảm bảo do không có một kênh truyền riêng

Giao thức kết nối giống với giao thức không kết nối, tuy nhiên trước khi truyền tin thì client và server phải tạo một kênh truyền riêng. Khi kết thúc việc trao đổi dữ liệu, kênh đó sẽ bị huỷ.

* Ưu điểm: đáng tin cậy
* Nhược điểm: hiệu năng không cao vì hao phí thời gian và tài nguyên hệ thống vào việc tạo và huỷ kênh truyền.

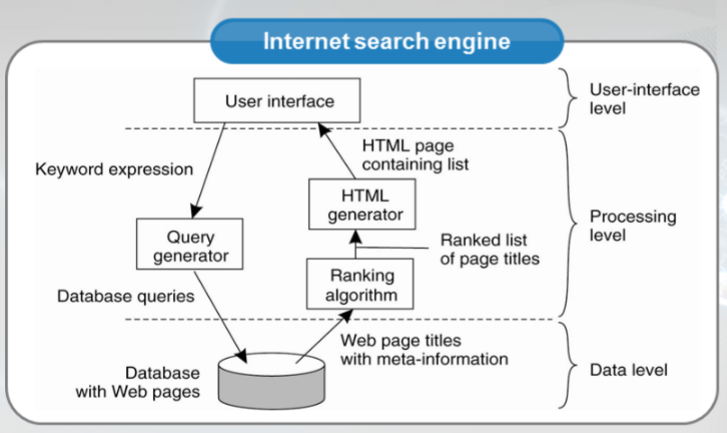
Có thể lập lại hay không: khi một thao tác thực thi có khả năng lặp đi lặp lại nhiều lần mà không làm ảnh hưởng đến hệ thống, nó được gọi là indempotent. Việc cho phép indempotent hay không phụ thuộc vào tính chất của dịch vụ đang sử dụng. VD: việc kiểm tra số dư tài khoản ngân hàng. Ngược lại: việc chuyển khoản.

* 1. Kiến trúc phân tầng ứng dụng:

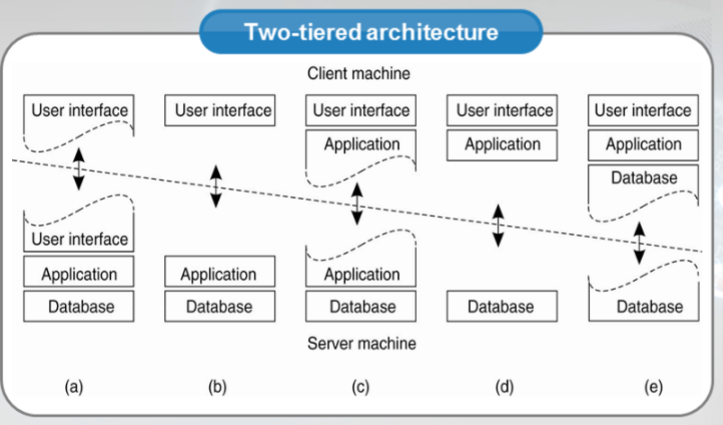
Ứng dụng client-server được chia làm 3 tầng chính:

* Tầng giao diện người dùng: quản lý trực tiếp giao diện với người dùng, VD: quản lý hiện thị màn hình. Tầng này cung cấp các dịch vụ cho người dùng cuối tương tác với ứng dụng và các chương trình khác nhau bởi sự phức tạp của nó. VD: giao diện terminal, giao diện đồ hoạ…
* Tầng nghiệp vụ: Chứa các chức năng xử lý chính của ứng dụng.
* Tầng dữ liệu: lưu trữ và duy trì dữ liệu để các ứng dụng có thể sử dụng. Đặc tính quan trọng phải giữ cho dữ liệu bền vững: cho dù dữ liệu không có ứng dụng sử dụng thì vẫn phải lưu trữ cho lần sử dụng kế tiếp.

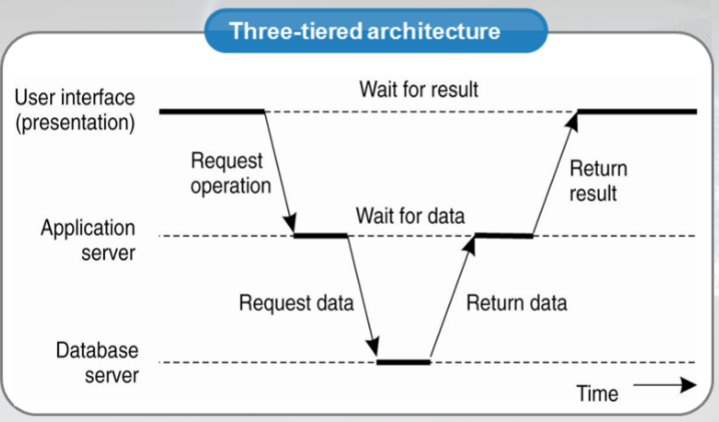
Mô hình phân tầng ứng dụng cho một dịch vụ tìm kiếm trên internet:



* 1. Kiến trúc đa tầng:

Kiến trúc hai tầng

Kiến trúc ba tầng:



Kiến trúc client-server 3 tầng bao gồm 3 tầng logic, với mỗi tầng được cài đặt ở các máy khác nhau. Tầng cao nhất bao gồm giao diện client người dùng, tầng giữa bao gồm ứng dụng, và tầng thấp nhất bao gồm dữ liệu được sử dụng.

1. Kiến trúc không tập trung

Kiến trúc client-server đa tầng là hệ quả trực tiếp của việc chia ứng dụng thành các tầng giao diện người dùng, tầng nghiệp vụ và tầng dữ liệu. Các thành phần khác nhau tương ứng với tổ chức logic của các ứng dụng. Trong rất nhiều môi trường làm việc, việc xử lý phân tán tương đương với việc tổ chức ứng dụng client-server như một kiến trúc đa tầng.

* Phân phối dọc: Các thành phần khác nhau về mặt logic được phân bố trên các máy khác nhau. Các chức năng được chia một cách logic và vật lý trên các máy khác nhau.

Tuy nhiên, trong các kiến trúc hiện đại, người ta thường xét đến phân phối các client và server.

* Phân phối ngang: client hoặc server sẽ được chia một cách vật lý các phần việc tương ứng. Mỗi phần sẽ được chia sẻ một phần việc riêng. Đây được gọi là cơ chế cân bằng tải.

Peer-to-peer: hỗ trợ phân phối ngang, bao gồm:

* 1. P2P có cấu trúc

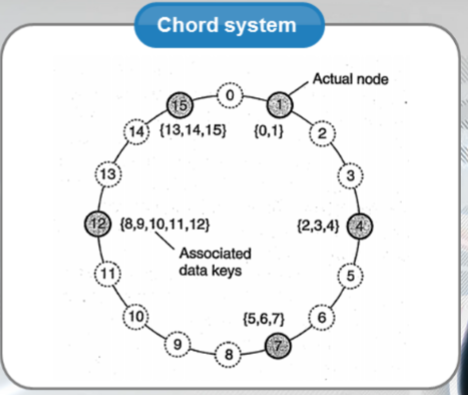
Mạng overlay: là một mạng mà các node được tạo bởi các tiến trình và các đường kết nối biểu diễn các kênh kết nối, như kết nối TCP. Mạng overlay được xây dựng bên trên hệ thống mạng vật lý thật.

Trong kiến trúc P2P có cấu trúc mạng overlay được xây dựng dựa trên một thủ tục xác định. Thủ tục thường được sử dụng nhất là sử dụng bảng băm phân tán (Distributed hash table - DHT). Hệ thống DHT là một lớp của hệ phân tán không tập trung. Nó cung ứng một dịch vụ tìm kiếm tương tự với bảng băm, sử dụng cặp khoá và giá trị.

Với việc sử dụng bảng băm phân tán, khi muốn lưu trữ dữ liệu, tên của tệp được băm bằng thuật toán SHA-1, tạo ra một khoá 160 bit gọi là K, sau đó dùng hàm push(K, data). Cặp (K, data) được truyền đi trong overlay để đến được node chịu trách nhiệm khoá K đó, và nó sẽ lưu trữ lại K và dữ liệu. Để lấy thông tin dữ liệu, chúng ta lại thực hiện băm tên tệp để có được khoá K và gọi lệnh get(K). Lúc đó thông điệp sẽ truyền trong mạng overlay để đến được với node phụ trách khoá K đó.

Để xây dựng mạng overlay, cần có thao tác phân vùng khoá. Thao tác này dùng để xác định các node chịu trách nhiệm các khoá. Đồng thời cần dùng khái niệm khoảng cách giữa các khoá. Các node khi được phân công chịu trách nhiệm một phân vùng khoá nghĩa là các khoá trong khoảng đó phải có khoảng cách < delta nào đó.

Ví dụ hệ thống Chord:



Các node được biểu diễn trong một vòng sao cho một mục dữ liệu với khoá K sẽ được ánh xạ tới node với định danh nhỏ nhất > k.

* 1. P2P không có cấu trúc

Với P2P không cấu trúc, người ta sử dụng các thuật toán ngẫu nhiên để xây dựng nên các mạng overlay.

Mỗi node có một danh sách các node hàng xóm.

Dữ liệu được đưa vào mạng một cách ngẫu nhiên. Vì vậy, hi tìm một dữ liệu nào đó, ta cần phải truy vấn toàn bộ mạng.

* 1. Kiến trúc hỗn hợp (hybrid architectures)
* Hệ thống máy chủ biên

Các server được triển khai ở biên các mạng. Ở đây khái niệm đường biên là ranh giới giữa mạng doanh nghiệp và mạng internet được cung ứng bởi ISP.

Người dùng muốn vào mạng phải kết nối thông qua server biên.

Ý tưởng:

Dữ liệu từ nhà cung cấp nội dung sẽ được phân tán đi các server biên. Tới đây, các server biên sẽ phân phối tiếp nội dung đó đến cho các khách hàng. Các server biên ở đây có thể là chính các nhà cung ứng dịch vụ mạng.

Mục tiêu:

Cung ứng nội dung cho các client mà các client không cần phải trực tiếp kết nối đến các nhà cung cấp nội dung.

Nguyên tắc hoạt động của BitTorrent:

Khi người dùng muốn tìm một file, người đó sẽ phải dùng mô hình tập trung để có thông tin tham chiếu đến tracker quản lý tệp đó, thông tường sẽ là một tệp định dạng .torrent. Từ đó, người dùng sẽ tải về các phần khác nhau của file đến khi đầy đủ. Các phần của file sẽ được cung ứng bởi các node khác đang cùng tải hoặc đã tải về xong.

* Hệ phân tán hợp tác

1. Mô hình cơ sở
   1. Mô hình tương tác:

Việc tính toán được xảy ra trong các tiến trình. Các tiến trình tương tác dựa trên việc truyền thông điệp, tạo ra quá trình truyền thông (dòng thông tin) và phối hợp (đồng bộ và sắp xếp các hoạt động) giữ các tiến trình.

Mô hình tương tác cần phải phản ánh quá trình các tiến trình truyền thông diễn ra với độ trễ và độ chính xác.

* 1. Mô hình sự cố:

Hoạt động sửa chữa được xảy ra ở bất cứ đâu, bất cứ máy tính nào đang gặp sự cố. Mô hình này phải định nghĩa và phân loại các sự cố. Nó tạo tiền đề cho việc phân tích các ảnh hưởng tiềm tàng cho việc thiết kế hệ thống và cho khả năng chịu lỗi của mỗi kiểu lỗi trong khi vẫn tiếp tục chạy.

* 1. Mô hình an ninh:

Hệ phân tán có kiểu module và có tính mở làm lộ ra khả năng bị tấn công từ các agent bên trong lẫn bên ngoài. Mô hình an toàn định nghĩa và phân loại các kiểu tấn công đó có thể tạo ra, cung ứng các kiểu kiến thức cơ sở để phân tích và xử lý, đồng thời thiết kể hệ thống có khả năng chống chịu các kiểu tấn công đó.

Hiệu năng của kênh truyền thông: là các đặc tính hệ thống quyết định hiệu năng của kênh truyền thông, vd delay, bandwidth, hay jitter.

Có 2 kiểu của mô hình tương tác:

* Hệ phân tán đồng bộ: chúng có các ràng buộc sau:
  + Thời gian để chạy một bước của 1 tiến trình, có ràng buộc trên và dưới biết trước.
  + Mỗi thông điệp gửi qua một kênh được nhận trong một khoảng thời gian biết trước.
  + Mỗi tiến trình có đồng hồ cục bộ mà drift rate từ thời gian thực của nó có một ràng buộc biết trước.
* Vấn đề đặt ra là làm sao xác định được các khoảng thời gian ràng buộc đó.
* Hệ phân tán không đồng bộ: là kiểu không có những giới hạn ràng buộc. Tốc độ chạy các tiến trình có thể nhanh hay chậm khác nhau. Về độ trễ của gói tin có thể hoàn toàn khác nhau giữa các lần truyền tin. Về drift rate là tuỳ ý và không ràng buộc.

VD: internet: mặc dù không có những ràng buộc về thời gian và độ trễ, nhưng vẫn phải có những cách xử lý và che giấu chúng. Ví dụ việc duyệt web, nếu dung lượng của trang web quá lớn, trình duyệt có thể bố trí cho người dùng làm việc gì đó trong khi chờ đợi, ví dụ tải trước các văn bản cho người dùng đọc trước trong khi trình duyệt tải các nd khác có dung lượng lớn hơn.

# Processes & Threads

Một số keyword:

Process: là một chương trình đang được chạy trên một trong những bộ xử lý ảo của OS.

Thread: luồng được chạy trên đoạn code riêng của mình, độc lập với các luồng khác, nhưng không có sự đồng thời (concurrency) và trong suốt (transperency) giữa chúng.

Process table: lưu các giá trị bản ghi thông tin vào thanh ghi CPU, bản đồ bộ nhớ, các tệp đang mở…

Dispatcher: là một luồng nhận các yêu cầu được gửi đến và phân phối chúng cho các luồng đang hoạt động.

1. Các tiến trình và luồng
   1. Giới thiệu về tiến trình và luồng

**Tiến trình:**

* Để chạy các chương trình, OS phải tạo ra các bộ xử lý ảo khác nhau (virtual processors). Mỗi VP chạy một chương trình. Để ghi lại thông tin của các VP này, OS cần có một bảng tiến trình ghi lại thông tin về: giá trị thanh ghi CPU, memory maps, các tệp đang mở, thông tin tài khoản, vv…
* Một tiến trình được định nghĩa như một chương trình đang chạy trên một bộ xử lý ảo. Vấn đề là OS làm cho hoạt động của các tiến trình đó là trong suốt với nhau. Chúng hoàn toàn độc lập và không ảnh hưởng lẫn nhau. OS cần sự hỗ trợ của phần cứng để thực hiện sự trong suốt này. Để thực hiện được điều đó, OS cần đặt ra các không gian địa chỉ hoàn toàn độc lập.
* Việc cấp phát vùng nhớ tương đương với việc:
  + khởi tạo vùng nhớ
  + xoá vùng dữ liệu
  + sao chép chương trình vào vùng nhớ đó
  + khởi tạo ngăn xếp cho dữ liệu tạm thời
* Việc chuyển CPU giữa các tiến trình cũng rất tốn kém tài nguyên của hệ thống. Nó bao gồm các việc lưu lại thông tin ngữ cảnh của CPU: giá trị thanh ghi, bộ đếm chương trình, con trỏ stack, vv…
* OS còn phải thay đổi thanh ghi cho đơn vị quản lý bộ nhớ (memory management unit)
* Ngoài ra, nếu số lượng tiến trình vượt quá khả năng lưu trữ của bộ nhớ chính, có thể phải dùng tạm bộ nhớ của ổ cứng để swap. Vấn đề cần thiết là phải kiểm soát chặt chẽ độ bảo mật và an toàn dựa trên nguyên tắc các thao tác lỗi của một tiến trình rất khó có thể ảnh hưởng tới các tiến trình khác không phụ thuộc. VD: dịch vụ quản lý màn hình trong linux, dịch vụ login trong windows…Đặc biệt, những dịch vụ login cần phải đảm bảo an toàn và độc lập với các tiến trình khác.

**Luồng:**

* Khi một tiến trình chạy, nó sẽ sinh ra một hay nhiều luồng.
* Sau khi được tiến trình sinh ra, mỗi luồng có một đoạn mã thực hiện riêng độc lập với các luồng khác. Tuy nhiên, khác với tiến trình, các luồng không cần phải hoạt động hoàn toàn độc lập và trong suốt với nhau. Vì vậy, mỗi luồng chỉ cần lưu trữ thông tin ít nhất có thể cho phép các luồng chia sẻ CPU.
* Ngữ cảnh của luồng chỉ là ngữ cảnh của CPU, giá trị thanh ghi, bộ đệm chương trình, con trỏ stack, vv...và một vài thông tin quản lý luồng khác. Những thông tin nào không cần thiết để quản lý đa luồng thì thường được bỏ qua. Chính vì vậy, việc bảo mật truy cập giữa các luồng thường được thực hiện bởi các lập trình viên.
* Các ứng dụng đa luồng giúp chương trình có hiệu năng cao hơn. Tuy nhiên, việc lập trình sao cho các luồng không ảnh hưởng tới nhau là khá khó khăn.
  1. Luồng trong hệ tập trung

Lợi ích khi sử dụng đa luồng:

* Nếu không sử dụng đa luồng, với các điều kiện dừng, toàn bộ đoạn mã sẽ bị dừng khiến chương trình bị treo. Vì vậy chúng ta cần đa luồng để giải quyết vấn đề đó.

VD chương trình spreadsheet: bao gồm bảng lưu các giá trị (vd: Excel). Người dùng muốn liên tục tương tác thay đổi giá trị của các ô, và các ô này có ràng buộc với nhau về giá trị. Vì vậy chương trình cần tạo ra 2 luồng: luồng tương tác nhận giá trị của người dùng, luồng còn lại đảm nhận cập nhật sự thay đổi giá trị của các ô.

* Trong khai thác đa vi xử lý ứng dụng trong chương trình tính toán song song: ở đây mỗi luồng nằm ở 1 cpu nhưng dữ liệu chia sẻ lại nằm ở bộ nhớ chính. Với mô hình này, chúng ta hoàn toàn có thể sử dụng các thuật toán tính toán song song để tăng hiệu năng hệ thống với kiến trúc đa vi xử lý.
* Trong các chương trình lớn: thông thường, chương trình lớn được cấu thành từ các chương trình nhỏ liên kết với nhau. Mỗi chương trình nhỏ đều có một luồng phân biệt. Hướng tiếp cận này được áp dụng trong cơ chế IPC (Inter-process Communication) của UNIX, vd như cơ chế đường ống (pipeline). Nhược điểm duy nhất của hướng tiếp cận này là các giao tiếp cần đến sự chuyển ngữ cảnh. Để khắc phục nhược điểm này người ta áp dụng cơ chế đa luồng, với việc mỗi chương trình con sẽ đảm nhận một luồng và không phải xử lý việc chuyển ngữ cảnh.
* Module hoá phần mềm: kĩ thuật lập trình đa luồng cho phép tạo ra các luồng đảm đương các chức năng khác nhau trong chương trình

***Cách thức cài đặt luồng:***

* Các luồng thường được cung cấp bởi các gói luồng. Các gói đó cung cấp các nhiệm vụ:
  + Tạo luồng
  + Huỷ luồng
  + Đồng bộ luồng
* Hai cách tiếp cận:

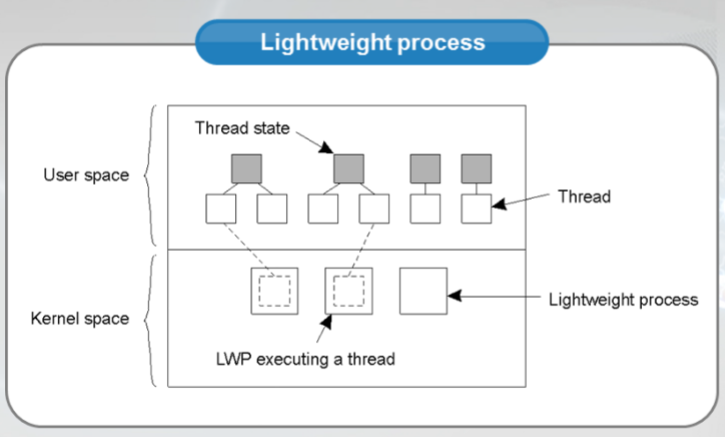
**User-mode:**

* + Ưu điểm:
    - Tiết kiệm tài nguyên hệ thống để tạo và huỷ luồng.
    - Việc chuyển ngữ cảnh được thực hiện nhanh.
  + Nhược điểm:
    - Lời triệu gọi của lời gọi hệ thống dừng sẽ làm ngưng toàn bộ tiến trình chứa luồng đó, và tất cả các luồng khác cùng tiến trình đó.

**Kernel-mode:**

* Ưu điểm:
  + Giải quyết được nhược điểm khi xây dựng luồng ở user-mode.
* Nhược điểm:
  + Chi phí lớn. Việc tạo, huỷ, đồng bộ luồng sẽ được đảm nhiệm bởi nhân và cần một lời gọi hệ thống.

**Lightweight process:**

****

Là sự kết hợp của việc chạy đa luồng ở cả chế độ người dùng và nhân.

Khi chạy ở ngữ cảnh một tiến trình đơn, mỗi tiến trình đơn sẽ có nhiều tiến trình nhẹ. Ngoài việc có nhiều tiến trình nhẹ, hệ thống còn cung cấp gói luồng ở mức user, trợ giúp các ứng dụng trong việc tạo và huỷ luồng. Gói cũng cung cấp phương tiện để đồng bộ hoá các luồng, ví dụ như việc sử dụng biến mutex và biến điều kiện.

Chú ý: biến mutex để kiểm soát ở tại một thời điểm chỉ một luồng truy cập vào dữ liệu chia sẻ. Biến điều kiện để giúp một luồng chờ một điều kiện nào đó rồi mới tiếp tục thực thi. Vì việc sử dụng gói luồng hoàn toàn ở tầng user, nên các thao tác với luồng không cần sự can thiệp của tầng kernel, vì vậy giảm nhiều chi phí. Gói luồng có routine duy nhất là lập lịch cho luồng kế tiếp.

Khi một tiến trình nhẹ được tạo ra với một stack, nó sẽ tìm luồng phù hợp để thực thi. Mỗi tiến trình nhẹ giữ một bảng luồng để tránh việc cùng dùng một luồng. Việc tránh truy cập cùng một lúc vào dữ liệu chia sẻ được đảm đương hoàn toàn ở mức người dùng với các luồng sử dụng mutex.

Việc đồng bộ giữa các tiến trình nhẹ cũng không cần thực hiện ở mức kernel. Nếu một luồng bị dừng, nó sẽ thực hiện lời gọi lập lịch. Khi một luồng khả chạy khác được tìm thấy, nó sẽ chuyển ngữ cảnh sang cho luồng mới đó. Tiến trình nhẹ đang chạy luồng mới đó hoàn toàn không biết việc chuyển ngữ cảnh. Khi một luồng thực hiện một lời gọi hệ thống dừng, việc thực thi được chuyển xuống cho kiểu kernel, nhưng vẫn ở trong ngữ cảnh của tiến trình nhẹ hiện tại. HĐH sẽ chuyển ngữ cảnh sang cho tiến trình nhẹ khác, nó dẫn tới việc chuyển ngữ cảnh ngược về cho kiểu user. Vì vậy, lời gọi hệ thống dừng không làm dừng cả hệ thống lại.

Ưu điểm:

* + Các thao tác với luồng có chi phí thấp và hoàn toàn không có sự can thiệp của tầng kernel.
  + Khi một ứng dụng có đủ các tiến trình nhẹ, một lời gọi hệ thống dừng không làm dừng cả hệ thống, mà chỉ làm dừng một tiến trình nhẹ.
  + Ứng dụng không cần phải biết về các tiến trình nhẹ, nó chỉ nhìn thấy các luồng ở mức user.
  + Tiến trình nhẹ có thể được sử dụng trong môi trường đa vi xử lý, mỗi tiến trình nhẹ gán với 1 cpu.

Nhược điểm:

* + Việc tạo và huỷ các tiến trình nhẹ có chi phí cao vì được thực hiện ở tầng nhân. (Tuy nhiên, việc này diễn ra không thường xuyên)

Tổng kết:

Một tiến trình sinh ra nhiều tiến trình nhẹ, mỗi tiến trình nhẹ được gắn với nhiều luồng. Khi một luồng có thao tác vào-ra, tiến trình nhẹ tương ứng bị treo, tuy nhiên các tiến trình nhẹ khác vẫn tiếp tục được thực hiện.

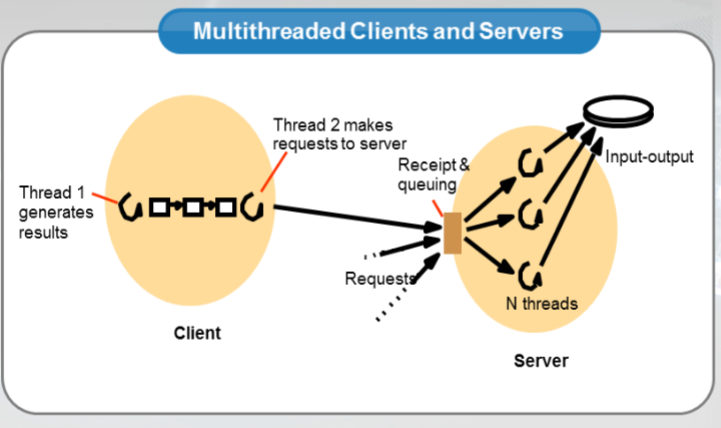
* 1. Luồng trong hệ thống phân tán:

**Server đơn luồng:** là tiến trình server chỉ sinh ra một luồng duy nhất để xử lý các việc và yêu cầu từ phía client.

Đặc điểm:

* Chỉ xử lý một request một lúc.
* Các request chỉ có thể được xử lý tuần tự, các yêu cầu chưa được xử lý được xếp vào hàng đợi.
* Các request có thể được xử lý bởi các tiến trình khác nhau. Do các tiến trình chỉ có một luồng nên các yêu cầu có thể được gửi đến các server khác nhau.
* Không đảm bảo tính trong suốt của hệ thống. Do chỉ có một luồng, nên khi các request được gửi lên chưa được xử lý kịp, và đang phải xếp trong hàng đợi, sẽ khiến user không cảm nhận được sự trong suốt của hệ thống.

**Server đa luồng:**

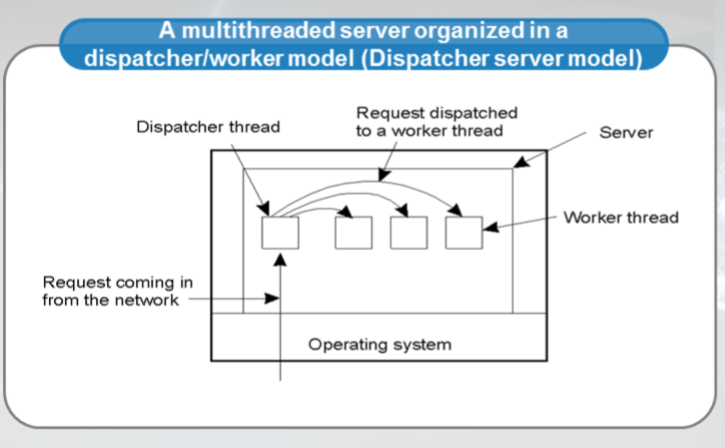


Các tiến trình client và server sẽ có nhiều luồng để xử lý các công việc.

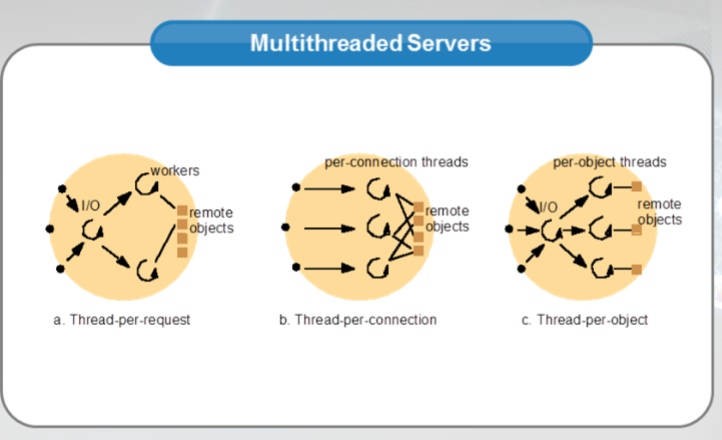
Ví dụ trong hình vẽ:

* Bên phía client sẽ có một luồng chuyên sinh ra các gói tin, luồng khác sẽ lo việc đóng gói gói tin và gửi cho server.
* Ở bên server, sẽ có một luồng chuyên lo thu nhận các gói tin rồi xếp chúng vào hàng đợi, sau đó, chuyển chúng đi các luồng khác chuyên xử lý thống điệp. Các luồng này sẽ làm việc trực tiếp với các module vào ra.
* Việc trao đổi giữa client-server có thể tiến hành song song => giúp làm tăng tốc độ và hiệu năng của hệ thống.

**Mô hình server có điều phối:**



* Trong mô hình này, ngoài các luồng để xử lý các công việc, server sẽ có một luồng chuyên điều phối công việc yêu cầu, gọi là dispatcher.
* Mỗi khi nhận được yêu cầu, luồng này sẽ phân tích yêu cầu đó và gửi đến cho luồng phù hợp để xử lý.
* Với mô hình đa luồng này, các luồng sẽ hoạt động hiệu quả hơn, với mỗi luồng sẽ xử lý các công việc riêng biệt do luồng điều phối phân loại và gửi đến.



**Một số kiến trúc tổ chức server đa luồng khác:**

*Luồng cho mỗi yêu cầu:* tiến trình vào-ra sinh ra một luồng worker mới cho mỗi yêu cầu nhận được. Mỗi luồng sẽ tự huỷ sau khi hoàn thành nhiệm vụ.

* + Ưu điểm:
    - Không cần có hàng đợi, vì mỗi yêu cầu được gửi đến đều có luồng riêng của nó.
    - Sử dụng tối đa băng thông, vì có khả năng sinh không giới hạn các luồng worker.
  + Nhược điểm:
    - Tạo và huỷ luồng cho mỗi request nhận được. Đặc biệt khi có quá nhiều luồng gửi đến, overhead cost của thao tác tạo và huỷ luồng sẽ rất cao, gây việc giảm hiệu năng và tăng chi phí cho các công việc khác của hệ thống.

*Luồng cho mỗi kết nối:* khi một client yêu cầu kết nối với server, server sẽ tạo ra một luồng cho kết nối đó, và huỷ luồng nếu kết nối đó ngắt.

*Luồng cho mỗi đối tượng:* tiến trình sẽ gắn một luồng với một đối tượng từ xa mà mình quản lý. Với kiến trúc này, luồng quản lý I/O phải quản lý thêm hàng đợi cho mỗi object

Ưu điểm của 2 hình thức trên:

* Không tốn nhiều overhead vì không cần tạo và huỷ nhiều luồng. Số lượng kết nối của các đối tượng từ xa là không thay đổi nhiều.

Nhược điểm:

* Các client có thể bị phục vụ với độ trễ cao, do trong khi một số luồng phục vụ quá nhiều yêu cầu, một số luồng khác không có công việc để làm.

Ví dụ: hoạt động của trình duyệt web, có các cách tổ chức khác nhau để lập trình đa luồng khác nhau.

Cách 1: mỗi một phần tử của tệp html tạo 1 connection, và mỗi connection đó do 1 luồng đảm trách.

Cách 2: mỗi một trang html mở 1 connection tương ứng với 1 luồng.

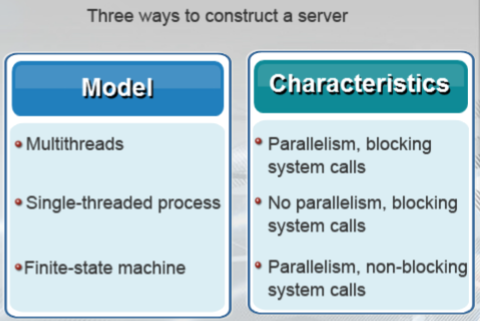
Cách 3: mỗi một trình duyệt tương ứng với một luồng.

**Mô hình máy trạng thái:**

Chỉ có một luồng nhận request từ các client. Khi nhận được thông điệp, nếu trong bộ đệm còn chỗ thì nhận và thực hiện luôn. Nếu không đủ chỗ thì không dừng tiến trình mà gửi vào đĩa cứng. Tiếp đó ghi thông tin trạng thái vào một bảng. Sau đó tiếp tục đi nhận thông điệp khác. Thông điệp nếu là công việc mới thì thực hiện luôn, còn nếu là câu trả lời của đĩa thì lấy thông tin phù hợp từ bảng và thực hiện gửi trả lời cho client.

Một số đặc điểm chính:

* Các yêu cầu từ client được xử lý và sẵn sàng.
* Tại một thời điểm server thực hiện thao tác trong hàng.
* Không cần đa luồng.
* Các lời gọi xử lý là các lời gọi không dừng.

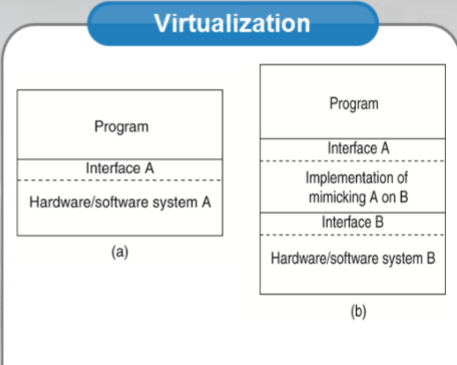


**Mô hình client đa luồng:**

* Tách biệt giao diện và xử lý. Ở giao diện ứng dụng người dùng luôn có 2 luồng, công việc chính đó là hiển thị UI và việc xử lý bên trong. VD: trình duyệt web hiển thị một trang web ra cho người dùng xem, công việc xử lý là việc kết nối với server và tải dữ liệu về. Việc tách biệt giao diện và xử lý giúp tăng tốc độ xử lý của hệ thống, nhờ đó che giấu được độ trễ của thời gian phục vụ. Cụ thể của cơ chế này là tạo ra các luồng khác nhau và phân cho chúng các công việc riêng.
* Để giải quyết vấn đề các thao tác chờ đợi lẫn nhau, các luồng dùng một cơ chế như sử dụng mutex để kiểm soát việc truy cập vào dữ liệu dùng chung, từ đó tránh được vấn đề chờ đợi lẫn nhau của các luồng.
* Tăng tốc độ khi làm việc với các server khác nhau: khi làm việc với các server khác nhau, ứng với mỗi server tiến trình client sẽ sinh ra một luồng để xử lý công việc với server đó, cụ thể là thiết lập kết nối và gửi nhận gói tin.
* Che giấu các chi tiết cài đặt: dành một luồng để hiện giao diện với người dùng, các luồng khác thực hiện các thao tác cài đặt cần thiết.
* VD: sử dụng trình duyệt để tải trang web: trình duyệt sẽ sinh ra một luồng để hiển thị trang web cho người sử dụng. Trình duyệt cũng sinh ra các luồng khác để tải dữ liệu về, vì vậy khi người dùng sử dụng trình duyệt, họ sẽ thấy các thông tin được hiển thị ra đầu tiên là các kiểu dữ liệu văn bản có kích cỡ nhỏ. Ở phía sau, các luồng khác vẫn đang thực hiện tải các dữ liệu khác có kích thước lớn hơn như đa phương tiện, video, audio…hoặc tiến trình client có thể kết nối đến các server khác nhau giúp tăng tốc cho việc tải dữ liệu.

1. Ảo hoá
   1. Vai trò của ảo hoá trong hệ phân tán

* Ở giai đoạn mà giá thành của phần cứng và phần mềm là cao, ảo hoá đóng vai trò mô phỏng phần cứng thay vì phát triển lại phần mềm.
* Khi giá thành phần cứng hạ, ảo hoá giúp cho việc sử dụng tối ưu hơn tài nguyên phần cứng.
* Khi tốc độ thay đổi phần cứng nhanh: nảy sinh vấn đề nhiều phần mềm không được hỗ trợ đầy đủ => cần ảo hoá
* Giúp cho người dùng dễ dàng quản trị các ứng dụng phần mềm.



Ý tưởng cơ bản của ảo hoá được thực hiện như 2 hình trên.

* 1. Một số kiến trúc máy ảo

# Communication

1. Tổng quan

Trong hpt, các tiến trình không thể không trao đổi thông tin với nhau. Mạng truyền thông cung cấp cơ chế bậc thấp để các chương trình có thể trao đổi thông tin. Liên quan đến trao đổi thông tin bậc cao và hướng ứng dụng giữa các tiến trình, chúng cần các cơ chế trao đổi thông tin bậc cao khác đảm bảo việc trao đổi thông tin giữa một số lượng lớn các tiến trình. Trong trường hợp tổng quát, trao đổi thông tin giữa các tiến trình được thực hiện bằng 2 cơ chế: truyền thông báo và chia sẻ bộ nhớ. Trong hpt, cơ chế thường được sử dụng là cơ chế truyền thông báo. Phương pháp này phức tạp hơn nhưng cho phép trao đổi thông tin một cách mềm dẻo hơn.

* 1. Các giao thức phân tầng

Do không sử dụng bộ nhớ chung nên trao đổi thông tin trong hpt thường thông qua trao đổi thông điệp. Việc trao đổi thông điệp phải đạt được sự thống nhất giữa bên gửi và bên nhận về những mặt sau:

* + - Cấu trúc của thông điệp: bên gửi và bên nhận phải thống nhất về cấu trúc thông điệp để phục vụ cho việc đóng gói và bóc tách thông điệp.
    - Cơ chế phát hiện thông điệp bị hỏng hoặc bị mất.
    - Kích cỡ dữ liệu.
    - Sự thống nhất giữa các bước khác nhau của thông điệp.

Mô hình OSI giúp các hệ thống mở giao tiếp bằng cách sử dụng một luật chuẩn về định dạng, nội dung và ý nghĩa của các thông điệp gửi & nhận, hay còn gọi là các giao thức. Quá trình gửi và nhận thông điệp, đóng gói hay bóc tách gói tin được thực hiện qua các tầng.

Các giao thức tầng thấp, như ở tầng vật lý đảm bảo việc truyền dẫn nhị phân, kiểm soát dây, đầu nối, điện áp, tốc độ truyền dữ liệu, các phương tiện truyền dẫn, chế độ truyền dẫn…

Ở tầng liên kết dữ liệu, đảm bảo việc điều khiển liên kết, truy xuất đường truyền, đóng frame, ghi địa chỉ vật lý, điều khiển luồng, kiểm soát lỗi, thông báo lỗi.

Ở tầng mạng: quản lý địa chỉ mạng và thiết lập đường đi tốt nhất, địa chỉ luận lý, topo mạng và việc định tuyến, tương tự với việc định đường đi cho gói tin.

Các giao thức giao vận được thực hiện khi kết nối end-to-end, vận chuyển giữa các host, vận chuyển tin cậy, thiết lập duy trì kết nối các mạch ảo, phát hiện lỗi, phục hồi thông tin và điều khiển luồng.

Với các giao thức tầng cao như là tầng phiên hay truyền thông liên host: thiết lập quản lý và kết thúc phiên giữa các ứng dụng.

Ở tầng trình bày: trình bày dữ liệu, định dạng dữ liệu, cấu trúc dữ liệu, mã hoá, nén dữ liệu.

Tầng ứng dụng: là các quá trình mạng của ứng dụng. Xác định giao diện giữa người sử dụng và môi trường OSI, cung cấp các dịch vụ mạng cho các ứng dụng như email, truyền file, vv…

**Trao đổi thông tin giữa các tiến trình** chủ yếu dựa vào 2 thao tác: gửi và nhận. Cơ chế hoạt động của hai thao tác này dẫn đến các kiểu trao đổi thông tin khác nhau.

Kiểu giao tiếp đồng bộ và bất đồng bộ: một hàng đợi được gắn với một đích của thông điệp. Tiến trình gửi đưa các thông điệp đó vào hàng đợi từ xa, tiến trình nhận rút dần các thông điệp trong hàng đợi để xử lý. Trao đổi thông điệp giữa các tiến trình có thể đồng bộ hoặc bất đồng bộ.

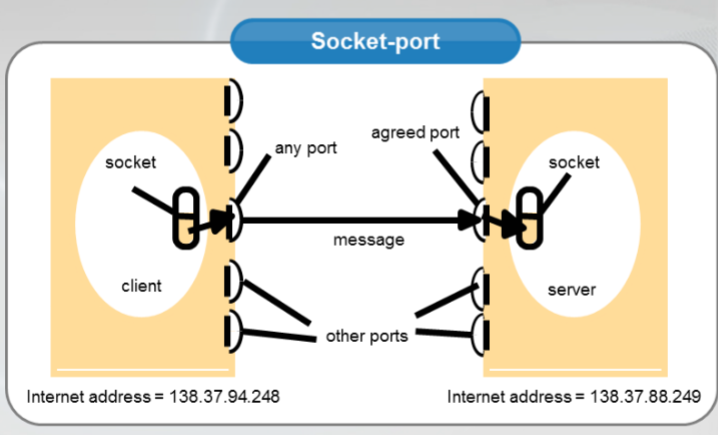
Với kiểu đồng bộ, tiến trình gửi và nhận đồng bộ từng thông điệp. Thao tác gửi và nhận đều là thao tác dừng. Bất cứ khi nào thao tác gửi được gọi, tiến trình gửi sẽ bị dừng đến khi nào thao tác nhận được diễn ra ở bên tiến trình nhận. Bất cứ khi nào thao tác nhận được gọi, tiến trình sẽ bị dừng cho đến khi thông điệp đến nơi.

Với kiểu bất đồng bộ, thao tác gửi là không bị dừng. Tiến trình gửi được cho phép thực thi đến khi nào thông điệp đã được sao chép vào một bộ đệm. Quá trình nhận diễn ra song song với tiến trình gửi. Tiến trình nhận chạy song song với thao tác nhận, nó cung ứng một bộ đệm cho phép chạy nền để nhận thông điệp, từ đó cần có cơ chế để thông báo khi nhận xong thông điệp. Trong môi trường hệ thống như Java, thì các thao tác nhận bị chặn không có bất lợi gì bởi vì nó cho phép đa luồng. Luồng nhận bị chặn thì các luồng khác vẫn hoạt động song song.

Đích đến của các thông điệp: thông tin đích đến của thông điệp thường là một cặp địa chỉ IP và cổng cục bộ. Một cổng thì có đích xác một tiến trình nhận, nhưng có thể có nhiều tiến trình gửi. Ví dụ các cổng cho các ứng dụng quen thuộc: HTTP là 80, FTP là 20 và 21, SSH là 22, Telnet là 23, SMTP là 25…Người sử dụng sử dụng IP để chọn một dịch vụ cụ thể, vì vậy dịch vụ đó nằm trên máy server nào thì phải đặt nguyên ở máy đó. Tuy nhiên, có giải pháp là cung cấp cơ chế phân giải địa chỉ IP để đưa ra địa chỉ cụ thể của máy đang cung cấp dịch vụ đó. Các dịch vụ có thể chuyển từ máy này sang máy khác, nhưng không cho phép việc di trú, tức là chuyển vị trí trong lúc hệ thống đang vận hành.

Về độ tin cậy, nó liên quan đến tính có hiệu lực và toàn vẹn của thông điệp.

Về thứ tự, một số ứng dụng yêu cầu các thông điệp nhận phải đúng thứ tự giống bên gửi, đặc biệt là các ứng đụng đa phương tiện thời gian thực, dữ liệu âm thanh và hình ảnh cần phải gửi và nhận cùng lúc.



Trao đổi thông tin qua socket:

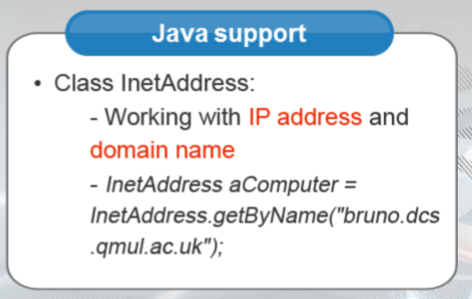
Quá trình trao đổi thông tin giữa các tiến trình là việc gửi thông điệp giữa một socket của một tiến trình và một socket của một tiến trình khác. Với một tiến trình, để nhận thông điệp, socket của nó phải gắn với một cổng và địa chỉ IP của máy tính mà nó đang chạy. 1 thông điệp được gửi đến một địa chỉ IP ở một cổng cục bộ thì sẽ được nhận bởi duy nhất một tiến trình mà socket của nó gắn với cổng và địa chỉ IP đó. Một tiến trình có thể nhận các thông điệp trên nhiều cổng khác nhau, nhưng không thể chia sẻ cổng với các tiến trình khác trên cùng một máy tính. Có một số vấn đề cần xem xét như sau:

Về phía server, phải xem server có sẵn sàng phục vụ hay không. Đó là vì khi server không nghe trên một cổng nào đó, thì client không thể gửi request đến cho server đó được. Về phía client, cần xem xét vấn đề sử dụng dịch vụ của server.

Với mô hình tương tác giữa client và server dẫn đến việc cần trao đổi thông tin hướng dòng. Với mô hình yêu cầu và trả lời thì cần quan tâm đến việc trao đổi thông tin hướng thông báo. Do trao đổi thông tin sử dụng dịch vụ của hạ tầng mạng, cần quan tâm đến dịch vụ tin cậy hay không tin cậy, dịch vụ hướng kết nối hay không hướng kết nối.

Liên quan đến vấn đề về địa chỉ server, việc định danh server có thể được giải quyết bằng cổng tĩnh hoặc bằng dịch vụ thư mục.

Trong bộ thư viện lập trình của Java có hỗ trợ một số lớp và phương thức để thực hiện trao đổi thông tin qua socket. Vd lớp InetAddress: đây là một class nằm trong package java.net. Lớp này có nhiệm vụ phân giải địa chỉ từ địa chỉ host sang địa chỉ IP bằng cách kết hợp các thông tin cấu hình cục bộ của máy và những dịch vụ mạng như DNS và NES.



* 1. Trao đổi thông tin với UDP

Đặc tính: Không hướng kết nối, không tin tưởng, không đồng bộ.

Tính không tin tưởng được thể hiện khi một datagram gửi bằng UDP thì không cần bên nhận báo nhận.

Tính không hướng kết nối được thể hiện khi client và server kết nối không cần phải tạo kênh trước khi trao đổi thông tin. Để gửi hoặc nhận thông điệp thì phải tạo một socket gắn với địa chỉ IP và cổng. Server thì gắn với cổng socket mà client viết đến để gửi đến. Client thì gắn với bất kì socket nào để gửi đi.

Tính không đồng bộ được thể hiện ở phương thức send không phải là một phương thức dừng. Khi nhận một yêu cầu kết nối, server sử dụng phương thức receive trả về địa chỉ IP và cổng của client để giúp server trả lời lại đúng client đó.

Một vài vấn đề của UDP:

* + - Kích cỡ thông điệp (message size): bên nhận dùng 1 bảng các byte có kích cỡ xác định 8kb để nhận thông điệp. Giao thức IP cho phép kích cỡ maximum của 1 packet lên đến 216 byte, vì vậy muốn gửi sang cho bên nhận cần có cơ chế chia nhỏ gói tin.
    - Vấn đề dừng (blocking): socket thường cung ứng các thao tác send không dừng và receive bị dừng. Thao tác send thường trả về khi đã gửi thông điệp xuống các tầng dưới, còn thao tác receive lấy các thông điệp trong hàng đợi ra để xử lý. Nếu không có tiến trình nào gắn với cổng socket nào đó thì các thông điệp gửi đến socket đó sẽ bị loại bỏ.
    - Timeout: bên nhận không thể cứ dừng thao tác receive mãi mãi để chờ nhận thông điệp của bên gửi, vì vậy cần có khoảng thời gian timeout. Chọn một khoảng timeout phù hợp là vấn đề cần giải quyết.
    - Receive from any: phương thức receive không chỉ được rõ là nhận thông điệp từ đâu. Giá trị trả về của nó cho biết thông tin của bên gửi, tuy nhiên khả năng là kết nối socket đó đến một địa chỉ cổng và IP từ xa, từ đó socket chỉ có thể gửi đi và nhận thông điệp từ một địa chỉ IP và cổng nhất định.

Trong trao đổi thông tin bằng giao thức UDP, tiến trình server và client làm những công việc cụ thể sau:

Server:

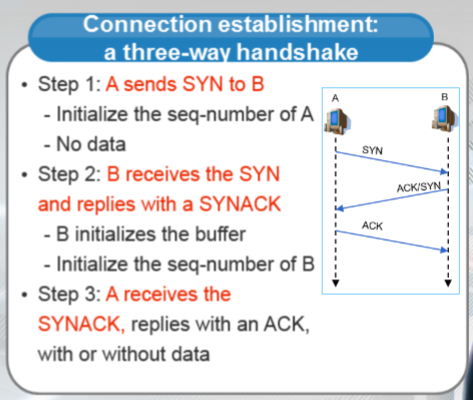
* + - Nhận yêu cầu từ client
    - Lấy IP, port của client
    - Xử lý thông điệp, chuẩn bị câu trả lời
    - Gửi trả lời về cho client

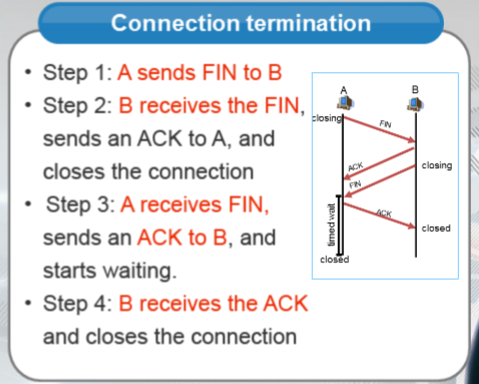
Client:

* + - Gửi yêu cầu, sử dụng địa chỉ IP & port của server
    - Chờ câu trả lời
* Phương thức primitive nào là dừng?
* aSocket.receive(reply) bên phía server và client
  1. Trao đổi thông tin với TCP

So với UDP thì TCP có một số ưu điểm như:

* + - Giao thức hướng kết nối và đáng tin cậy: thiết lập kênh truyền trước khi trao đổi thông tin. Mỗi khi nhận thông điệp thì bên nhận sẽ gửi một thông điệp báo nhận.





Các công việc của client và server trong phương thức TCP:

Server:

* + - Đăng kí và lắng nghe trên 1 cổng
    - Nhận 1 yêu cầu kết nối TCP
    - Trao đổi thông tin qua cổng đó
    - Kết thúc kết nối

Client:

* + - Thiết lập kết nối với server với các thông tin về địa chỉ IP và số port
    - Thực hiện truyền dữ liệu
    - Ngắt kết nối

Với giao thức TCP chúng ta có một số vấn đề cần xem xét sau:

* + Data type matching:

Hai tiến trình trao đổi thông tin cần thống nhất về một nội dung trao đổi trong một luồng dữ liệu. VD 1 tiến trình viết 1 phần tử dạng int sau đó là một phần tử dạng double, thì tiến trình đọc cũng phải đọc tương tự như vậy. Khi không phối hợp tốt thì sẽ dễ gây ra lỗi, hoặc bị dừng vì không đủ dữ liệu trong luồng.

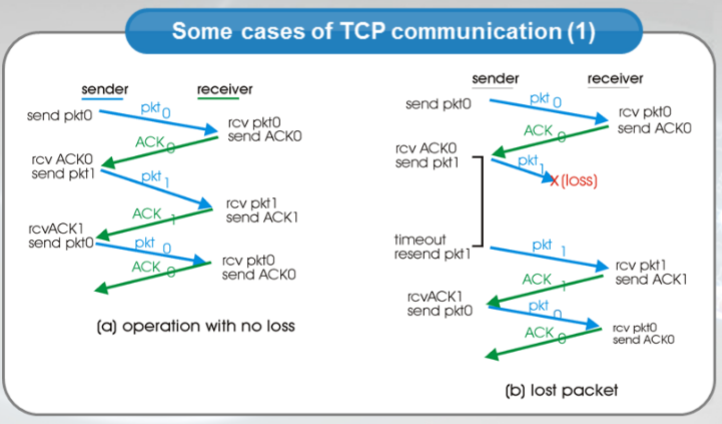
* + Blocking

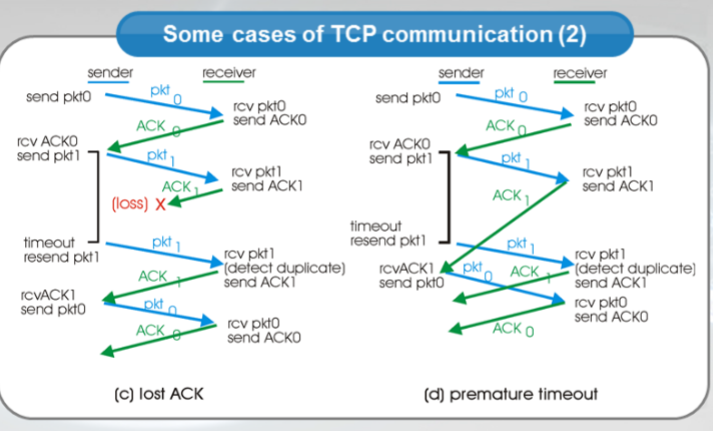
Dữ liệu viết vào luồng được giữ ở trong hàng đợi ở bên tiến trình nhận. Ở bên nhận, tiến trình nhận sẽ lần lượt lấy ra dữ liệu từ hàng đợi và xử lý, hoặc nó sẽ bị dừng đến bao giờ có dữ liệu. Bên tiến trình gửi cũng có dừng nếu tiến trình bên nhận bị đầy hàng đợi.

* + Threads

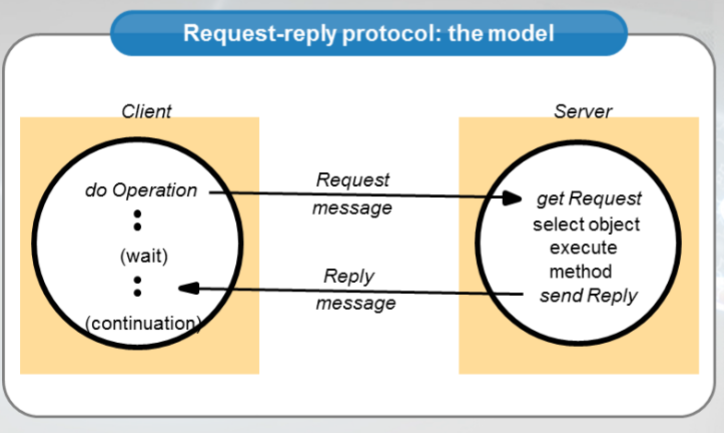
Khi server chấp nhận một kết nối nó sẽ tạo ra một luồng mới để kết nối với server đó. Ưu điểm của việc sử dụng các luồng phân biệt đó là server có thể tự dừng để chờ đợi các kết nối từ client mới mà không làm ảnh hưởng tới các client cũ.

* + Reliability





1. Lời gọi thủ tục từ xa
   1. Request/Reply protocol
      * Là cơ chế bậc cao hơn thông báo
      * Cho phép trao đổi thông tin giữa 2 tiến trình bằng 2 thông báo gửi và nhận liên tiếp.
      * Hỗ trợ các lời gọi từ xa
      * Đồng bộ: client sẽ tự dừng khi có câu trả lời của server
      * Tin cậy: có cơ chế tương tự với thông điệp báo nhận khi nhận được yêu cầu

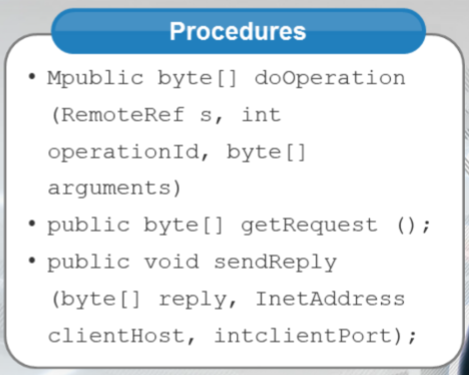


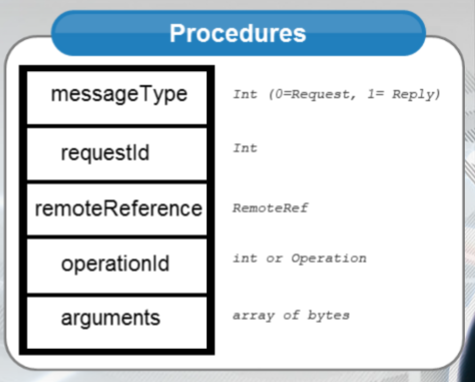
Giao thức yêu cầu-trả lời là một framework bao gồm 2 thông điệp trao đổi thông tin: thông điệp yêu cầu và thông điệp tra lời. Nó đảm bảo tính chất của phương thức trao đổi thông tin thông dụng, trong đó mô hình tương tác client-server bao gồm đồng bộ. Client bị block đến khi nhận được reply đáng tin cậy, tức là thông điệp trả lời từ server có vai trò tương đương thông điệp báo nhận.

Một vài đặc điểm của giao thức:

* + Không cần báo nhận khi đã có trả lời
  + Không cần kiểm soát luồng, chỉ việc gửi một vài tham số và nhận kết quả. Ẩn quá trình thiết lập kết nối
  + Phù hợp với datagram ở tầng giao vận

Thủ tục mà giao thức sử dụng là do Operation, nó thực hiện thao tác từ xa, với đầu vào là địa chỉ server và dữ liệu cần thiết để thao tác, đầu ra là kết quả trả về từ server, là một mảng byte. Client cần phải biến dữ liệu cục bộ thành mảng và ngược lại.

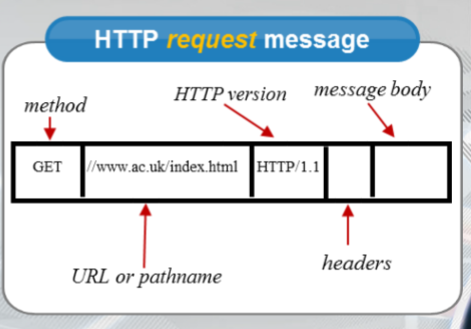


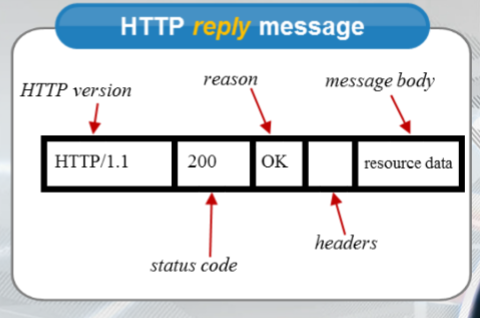


Hình: Đơn vị dữ liệu của giao thức yêu cầu trả lời

Các vấn đề khi thiết kể giao thức yêu cầu – trả lời:

* + Kích thước thông báo
  + Định danh thông báo: mỗi thông điệp cần có định danh duy nhất để tham chiếu. Định danh thông điệp gồm 2 phần:
    - requestId: là số nguyên tự tăng sau mỗi tiến trình gửi, thể hiện sự duy nhất đối với phiên gửi.
    - định danh của tiến trình gửi: bao gồm cổng và địa chỉ IP, ý nghĩa là thể hiện sự duy nhất trong hệ phân tán.
  + Mô hình lỗi:
    - Nếu là UDP thì sẽ có lỗi bỏ qua, có khả năng xảy ra sai thứ tự. Lỗi chủ yếu do đường truyền.
    - Nếu là TCP thì sẽ có lỗi dừng của tiến trình
* Hệ thống phát hiện các lỗi trên thông thường bằng timeouts. Vấn đề về timeout: sau một khoảng thời gian không nhận được phản hồi, cố gắng gửi lại request nhiều lần để chắc chắn lỗ mất gói tin, mà là do thiếu câu trả lời từ server.
  + Indempotent problem: khi cần gửi lại, có khả năng có thông báo lặp, nó phụ thuộc vào định danh của thông báo và tính chất của dịch vụ.



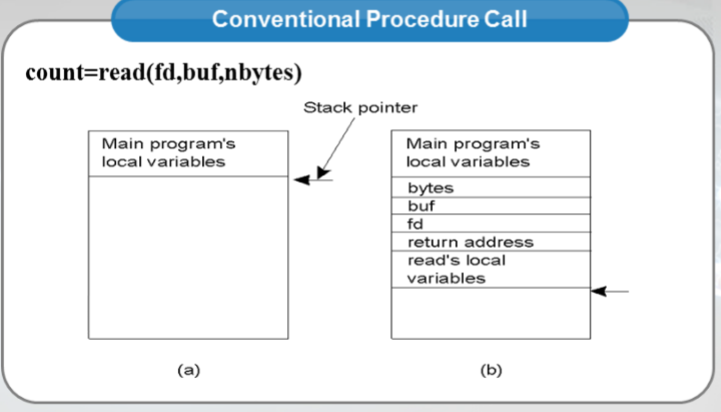


Các giao thức trao đổi thông tin:

* + R only: khi client gửi request lên và không quan tâm đến trả lời và ACK của server.
  + RR: gửi request và nhận reply không cần ACK, vì lúc này reply cũng đóng vai trò như ACK, và request tiếp theo của client cũng như một ACK đối với reply của server.
  + RRA: ACK kèm message gửi cùng requestId để cho biết báo nhận request đó rồi, và cũng có nghĩa là báo nhận tất cả các request trước đó có requestId nhỏ hơn requestId đó.
  1. Remote procedure call (RPC)

Thông thường, trao đổi thông điệp trong hpt sử dụng các thủ tục send & receive. Tuy nhiên, chúng không che giấu được việc trao đổi thông tin. Vì vậy người ta dùng cơ chế truy cập trong suốt với người dùng. Cơ chế đó chính là lời gọi thủ tục từ xa (RPC)

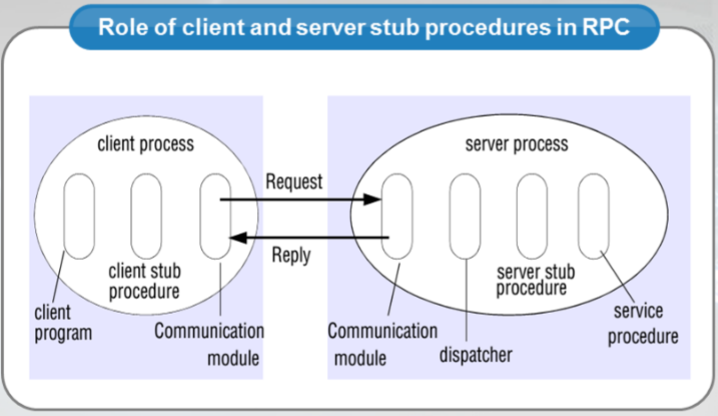
* + Là một bước tiến lớn trong tính toán phân tán
  + Mục đích: làm cho lập trình phân tán trở nên giống với phương pháp lập trình thông thường (mức trong suốt phân tán cao)
  + Che giấu các khía cạnh quan trọng trong phân tán:
    - Mã hoá và giải mã các tham số và kết quả
    - Việc truyền thông điệp
    - Giữ lại các ngữ nghĩa cần thiết cho việc gọi thủ tục



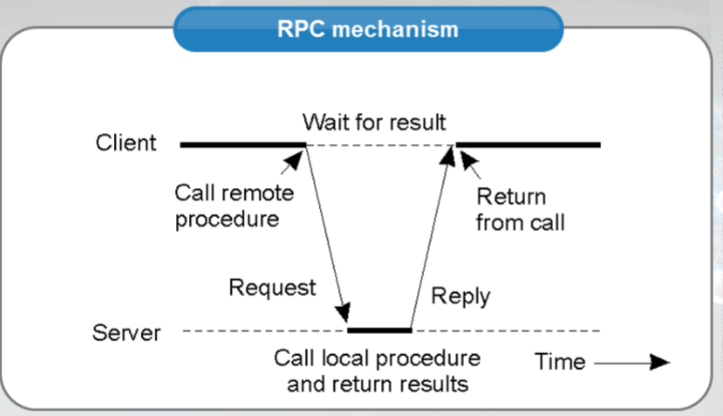
Cơ chế truyền tham số trong RPC:

Có 2 kiểu tham số:

* + - Tham số giá trị (tham biến – value parameter): là giá trị thật sự của biến đó, không thay đổi được trong thủ tục được gọi. Ở trong vd vừa rồi là fd và nbytes. Việc truyền tham số giá trị không gặp vấn đề trong RPC, vì giá trị đó được đóng gói và chuyển đi trong thông điệp gửi cho server.
    - Tham số tham chiếu (reference parameter): là con trỏ trỏ tới vùng nhớ, có thể thay đổi thật sự giá trị của vùng nhớ mà con trỏ trỏ tới. Ở vd vừa rồi là buf. Do server và client có hai nền tảng khác nhau về phần cứng và phần mềm, nên việc sử dụng y nguyên tham chiếu từ hệ thống này sang hệ thống khác là bất khả thi. Từ đó, người ta đưa ra giải pháp là sử dụng kiểu copy phục hồi. Copy dữ liệu vào stack, chương trình chạy sẽ thay đổi giá trị trong stack đó, và sau khi giá trị có thể bị thay đổi, copy ngược lại và ghi đè lên giá trị gốc ban đầu.



Ở bên tiến trình client và server đều có một client stub và server stub tương ứng. Khi client thực hiện gọi một thủ tục sẽ gửi xuống cho client stub. Client stub sẽ phân tích lời gọi và đóng gói thông điệp rồi gửi đi, bao gồm cả tham số, sau đó truyền cho module trao đổi thông tin thực hiện gửi cho server. Phía bên server, server stub thực hiện việc bóc tách gói tin, tạo ra một lời gọi đến một thủ tục với tham số đi kèm và thực hiện lời gọi thủ tục đó. Ở phía server có triển khai thủ tục được yêu cầu, sau khi thực hiện xong sẽ đưa ra kết quả. Lúc này server stub sẽ đóng gói thông điệp và gửi lại câu trả lời cho client.



Cơ chế RPC cho phép tiến trình gọi một thủ tục trên một hệ thống khác.

Cơ chế gọi thủ tục là trong suốt, làm nó giống như việc gọi một thủ tục cục bộ.

Cần một lời gọi tương tự như của một thủ tục cục bộ, nhưng được thực hiện theo cách khác, việc này do client stub đảm nhiệm.

Đoạn mã thực hiện thủ tục cũng cần thực hiện theo cách khác, điều này do server stub đảm nhiệm.

Các bước của RPC:

1. Chương trình khách (client) gọi client stub
2. Client stub đóng gói thông điệp và chuyển cho local OS
3. OS của client gửi thông điệp đến OS của server
4. OS của server chuyển thông điệp đến cho server stub
5. Server stub giải mã thông tin thông điệp và gọi chương trình con
6. Trả lại kết quả cho server stub
7. Thực hiện đóng gói kết quả chuyển cho local OS (server)
8. Gửi thông báo tới cho client OS
9. Truyền thông báo tới cho client stub
10. Giải mã thông báo và trả về cho client

Các chức năng của client stub:

* + Nhận các lời gọi từ chương trình client
  + Đóng gói dữ liệu với các tham số
  + Chuyển cho server
  + Nhận phản hồi
  + Giải mã phản hồi
  + Trả lại cho chương trình client

Các chức năng của server stub:

* + Nhận thông điệp từ client
  + Gọi chương trình server
  + Phản hồi lại cho client

Cơ chế truyền tham số bằng tham chiếu:

* + Chỉ hoạt động tốt khi hệ thống đầu-cuối là đồng nhất
  + Xuất hiện vấn đề khi:
    - Biểu diễn dữ liệu khác nhau
    - Kiểu dữ liệu khác nhau

VD: máy mainframe IBM sử dụng bảng mã EBCDIC còn máy IBM cá nhân sử dụng bảng mã ASCII. Vì vậy, không thể gửi tham số kí tự giữa hai máy đó cho nhau vì bên nhận sẽ xử lý nhầm.

Việc truyền tham số bằng tham chiếu:

Đây là vấn đề rất khó, con trỏ chỉ có ý nghĩa trong vùng địa chỉ của tiến trình mà trong đó nó đang được sử dụng. VD thủ tục read, tham số thứ 2 là buf, giả sử là 100 – chúng ta không thể truyền giá trị 100 này cho máy server.

Một số giải pháp được đưa ra:

* + - Cấm sử dụng các tham chiếu – tuy nhiên cách này rất không hay
    - Sao chép các giá trị của vùng nhớ đó đưa vào thông điệp và gửi đi. Ở bên phía server, server stub sẽ đưa mảng đó vào vùng nhớ, và tham số thứ 2 là địa chỉ trỏ tới mảng đó. Sau khi thực hiện xong, cả mảng đó được đóng gói và gửi trả về cho client. Bản chất của việc gọi call by reference được thay thể bởi kiểu copy-restore.

Giải pháp cải thiện: copy một lần: nếu mảng input cho server thì không cần copy ngược lại trả về. Ngược lại, nếu là output thì không cần copy thời điểm ban đầu gửi đi.

Tuy nhiên, những giải pháp trên không áp dụng được cho các dữ liệu có cấu trúc. Trong trường hợp đó, cần gửi pointer tới cho server stub và sinh ra các code đặc biệt trong thủ tục server để sử dụng các con trỏ đó.

Parameter specification:

Mục đích giấu đi RPC thì bên gửi và nhận cùng phải thống nhất về số lượng, kiểu dữ liệu, kích thước, cách trình bày các kiểu dữ liệu.

Cả 2 bên đều phải tuân thủ cùng một kiểu giao thức. Cần phải thống nhất các yếu tố sau:

* + - Định dạng thông điệp. VD xem xét thủ tục foobar(char x, float y, int z[5]){…}. Coi một từ có 4 bytes (tham số x) sẽ là byte cuối cùng trong 4 bytes của word. Client stub phải biết rằng phải sử dụng định dạng trong thông điệp như hình vẽ. Server stub cũng vậy, cần phải biết thông điệp nhận được sẽ có định dạng đó.
    - Cách biểu diễn các cấu trúc dữ liệu đơn giản: các kiểu ctdl đơn giản như integer, char, Boolean…vd integer biểu diễn theo kiểu mã bù 2, char biểu diễn 64 bit Unicode, float theo chuẩn IEE754, và tất cả theo kiểu little endian…
    - Kiểu trao đổi thông điệp: dùng dịch vụ vận chuyển hướng kết nối TCP/IP, hoặc sử dụng datagram không tin cậy, và cho phía client và server tự triển khai mô hình kiểm soát lỗi.
    - Triển khai client và server stub: các stub khác nhau gửi interface giao tiếp với ứng dụng. Một interface bao gồm các thủ tục mà nó sẽ được gọi ở client và được triển khai ở server. Interface thường được viết dựa trên nn lập trình của client và server, tuy nhiên thường được xây dựng bởi một IDL. Một interface được xây dựng bởi một IDL sẽ được dịch sang client stub và server stub.
    - Việc sử dụng IDL làm đơn giản nhiều ứng dụng client-server dựa trên RPC, vì rất dễ dàng để sinh ra các stub cho client và server. Tất cả các hệ thống middleware dựa trên RPC cung cấp một IDL để hỗ trợ phát triển một ứng dụng.

Tính mở của RPC:

* + Client và server được cài đặt bởi các nsx khác nhau
  + Giao diện là thống nhất giữa client và server:
    - Không phụ thuộc vào ngôn ngữ lập trình
    - Mô tả đầy đủ và trung lập
    - Sử dụng IDL

RPC không đồng bộ

* + Client không đợi trả lời từ server
  + Sau khi sử dụng RPC, client tiếp tục làm việc mà không cần quan tâm đến phản hồi của server

Client – server binding:

Để một client kết nối tới 1 server, server đó phải được đăng kí một địa chỉ và chuẩn bị để nhận các lời gọi. Việc đăng kí địa chỉ của server giúp client có thể xác định được địa chỉ của server, từ đó kết nối và gửi yêu cầu đến. Việc đăng kí được tiến hành qua những bước sau:

* + Server đăng kí với DCE daemon để đăng kí port
  + Server đăng kí với dịch vụ thư mục để cung ứng địa chỉ mạng

Việc định vị server được thực hiện qua 2 bước:

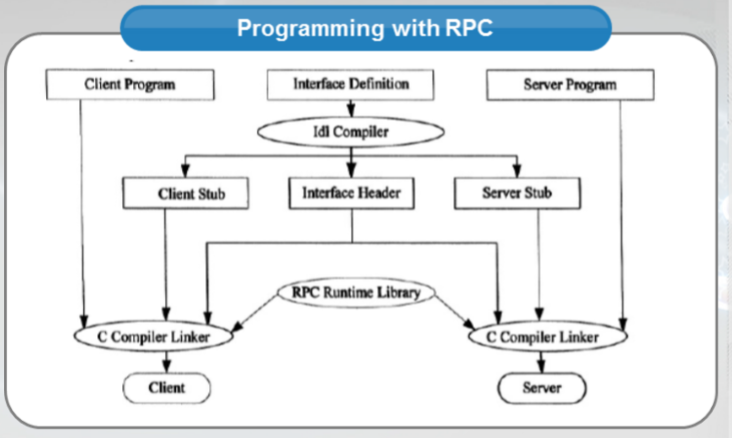
* + Xác định máy tính chứa server đó. Bước này tương đương với việc xác định địa chỉ mạng của máy. Cụ thể, client sẽ liên kết với máy chứa dịch vụ thư mục để yêu cầu cấp địa chỉ IP của server.
  + Xác định tiến trình server trong máy đó. Bước này tương đương với việc xác định port, cụ thể là client sẽ kết nối tới DCE daemon của máy đó, và yêu cầu được cấp port của tiến trình server.

2 loại liên kết với server:

* + Liên kết cục bộ: dịch vụ cục bộ trên server cho biết những thủ tục nào được liên kết
  + Liên kết toàn cục: dịch vụ thư mục cho biết địa chỉ máy tính và vị trí của dịch vụ

Một số vấn đề của client-server binding:

* + Client cần biết hàm nào gọi từ xa và trên máy nào, hàm nào liên quan đến tên hàm, máy nào liên quan tới địa chỉ của máy đó.
  + Các thông tin này được lưu trữ tại các bảng, và có thể được thêm, bớt các dòng trong bảng. Với các liên kết tĩnh ví dụ dịch vụ tên thư mục, còn với liên kết động thì là các dịch vụ kiểm soát server và client
  + Bảng có thể được xây dựng link hoặc thực hiện phụ thuộc vào RPC.



Để xây dựng chương trình bằng RPC, chúng ta cần trải qua các bước sau:

Đầu tiên, cần phải viết một chương trình mà định nghĩa giao diện bằng một ngôn ngữ định nghĩa giao diện tương ứng. Sau đó dùng trình dịch IDL để dịch ra 3 file đầu ra là client stub, server stub, và một file header của giao diện. Tiếp tới đưa 2 file client stub và server stub về client và server tương ứng, kết hợp file mã chương trình cho client và server, cùng với file header của giao diện, rồi dùng trình compiler linker để tạo ra chương trình client và server tương ứng.

* 1. Remote method invocation (RMI)

So sánh với RPC:

Tương đồng:

* + Cùng hỗ trợ lập trình với giao diện: các giao diện sẽ được công bố cho cả 2, client sẽ gọi những thủ tục và phương thức khai báo ở giao diện. Những thủ tục&phương thức này được triển khai ở phía server.
  + Dựa trên giao thức yêu cầu – trả lời. Trước chúng đều có yêu cầu gửi từ client sang server. Server thực hiện xử lý yêu cầu và gửi kết quả về cho client.
  + Mức độ trong suốt: đối với cả 2 cơ chế, người dùng chỉ nhìn thấy việc mình gọi thủ tục/phương thức đó từ máy client. Việc trao đổi thông tin ở máy client/server và việc thực thi ở máy server là trong suốt với người dùng.

Khác biệt:

* + Với RMI, dev có thể khai thác hết các điểm mạnh của OOP
  + Việc định danh của đối tượng: mỗi đối tượng trong hệ thống đều có định danh duy nhất, và tham chiếu đối tượng đó có thể truyền đi như một tham số

Đặc điểm nổi bật của RMI đó là dựa trên OOP. Cơ chế này sử dụng đối tượng từ xa ứng dụng phân tán hướng đối tượng.

Một vài vấn đề cần giải quyết:

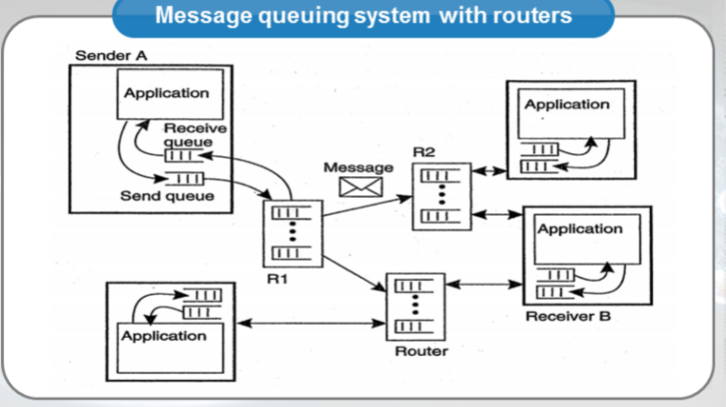
* + Định vị đối tượng từ xa
  + Trao đổi thông tin với đối tượng
  + Gọi các phương thức của đối tượng

Một vài ví dụ: RMI, T-RMI, DCOM, CORBA

Trong hệ thống hàng đợi thông điệp, cần có các đơn vị quản lý hàng đợi (queue managers). Các đơn vị quản lý hàng đợi tương tác trực tiếp với ứng dụng, gửi và nhận thông điệp. Tuy nhiên, có một vài đơn vị quản lý đặc biệt hoạt động như một router. Nó chuyển tiếp thông điệp nhận được cho các đơn vị quản lý hàng đơi khác. Việc sử dụng routers là rất hiệu quả trong mạng quy mô lớn, mỗi nút không thể lưu trữ được một bảng ánh xạ đầy đủ của hàng đợi tới địa chỉ (queue to location mapping). Ví dụ trong hình, máy A sẽ gửi thông điệp đến cho hàng đợi ở router 1, router 1 sẽ biết rằng, để đến được B thì sẽ phải chuyển tiếp cho router 2.

Một vài ưu điểm của việc sử dụng router:

* Giúp hàng đợi thông điệp có tính khả mở, tuy nhiên, hệ thống càng mở rộng thì việc cấu hình các đường đi cho từng router là càng bất khả thi. Chính vì vậy, hệ thống cần có cơ chế định tuyến tự thích nghi.
* Cho phép xử lý lần 2 các thông điệp. Các thông điệp cần được log lại để cho mục đích an ninh và chịu lỗi.
* Các router được sử dụng với mục đích multicasting



QoS (quality of service): các yêu cầu qos miêu tả cơ chế cần thiết của hệ phân tán cơ sở và mạng để mối quan hệ ràng buộc về thời gian cho một dòng dữ liệu được đảm bảo. QoS cho dòng dữ liệu liên tục chủ yếu liên quan đến việc gói tin đến kịp thời, dung lượng và độ tin tưởng. Các đặc tính ảnh hưởng tới QoS:

* Bit rate: tần số truyền bit
* Delay: độ trễ thời gian
* End 2 end delay: độ trễ thời gian đầu cuối
* Jitter: độ lệch thời gian giữa các gói tin bên nhận so với bên gửi
* Round-trip delay: khoảng thời gian cho gói tin đi và về

Các trao đổi thông tin hướng dòng dựa trên tầng IP: đơn giản, best-effort

# Naming

1. Các khái niệm cơ bản:

Access point: một loại thực thể đặc biệt trong hpt

Identifier: một tên chỉ đến một và chỉ một thực thể. Nó luôn luôn gắn với 1 thực thể đó

ARP: giao thức phân giải địa chỉ vật lý, là một giao thức để tìm địa chỉ kết nối dữ liệu của một máy khi biết địa chỉ IP của nó.

Home Location: lưu trữ địa chỉ hiện tại của thực thể.

Tên là rất quan trọng đối với bất kì hệ thống máy tính nào. Nó được dùng để chia sẻ tài nguyên, định danh duy nhất các thực thể, truy ra vị trí, vv… Vấn đề đối với tên là nó có thể được phân giải đến một thực thể. Phân giải tên cho phép một tiến trình có thể truy cập vào thực thể đã được định danh. Để phân giải tên, cần có dịch vụ định danh. Sự khác nhau giữa dịch vụ định danh trong hpt và tập trung là ở cách thức các dịch vụ định danh được triển khai. Trong hpt, sự triển khai của hệ thống định danh được phân bố trên nhiều máy. Sự phân bố các máy đó ntn đóng vai trò trong tính hiệu quả và khả mở của hệ thống định danh.

Mỗi thực thể cung cấp một tập các tác vụ. Muốn sử dụng các tác vụ phải truy cập đến thực thể đó. Mỗi thực thể được gắn 1 tên. Cơ chế truy cập từ tên đến thực thể được thực hiện bởi hệ thống định danh.

Tên: một chuỗi bit hoặc ký tự dùng để chỉ một thực thể. Một thực thể trong hpt có thể là bất cứ cái gì, vd host, máy in, ổ đĩa, tệp… Các thực thể thường được đặt tên là tiến trình, các người dùng, các hòm thư, các trang web, các thông điệp, kết nối mạng… Có thể nhìn dưới góc độ lý thuyết: tập hợp các thuộc tính dùng để phân biệt các thực thể. Tên phải đủ phân biệt các thực thể theo yêu cầu của ứng dụng một cách hiệu quả. Nó là quan niệm tường minh giữa thực thể và định danh. Một thực thể có thể sử dụng nhiều hệ thống tên. Trong một hệ thống định danh, một thực thể có thể có nhiều tên. Đối với bài toán thực tế, ví dụ 1 người sẽ có: số cmt, biển số xe, vv…Đối với bài toán tin học, chúng ta có những địa chỉ MAC, địa chỉ IP, tên người sử dụng, domain name, URL, vv… Một số tên dùng cho chức năng bậc cao, một số tên dùng để truy cập đến các thực thể, một số tên dùng vĩnh viễn cho các thực thể.

Xét 1 thực thể là kết nối mạng. Cần phải truy cập vào nó, và chúng ta cần 1 access point. Access point cũng là một thực thể, và tên của thực thể này chính là địa chỉ. Địa chỉ của 1 access point của 1 thực thể thì cũng là địa chỉ của tt đó.

Một thực thể có thể có nhiều địa chỉ. Địa chỉ của một dịch vụ là kết hợp của 2 thông tin: địa chỉ IP và port. Một thực thể có thể thay đổi địa chỉ, vd như một host có thể thay đổi địa điểm vị trí địa lý, và từ đó thay đổi AP. Địa chỉ là một kiểu đặc biệt của tên. Nó chỉ một access point của một thực thể. Vì 1 access point thường gắn với một thực thể nên dùng luôn địa chỉ của AP như là tên của thực thể tương ứng.

Tuy nhiên, một thực thể có thể thay đổi AP, và một AP sẽ được gán vào một thực thể khác. VD, khi một tiến trình server chuyển sang nằm ở một host khác, và host cũ thì triển khai server khác, tốt hơn hết là có một cơ chế định danh khác để tách biệt dịch vụ và server. Tương tự, nếu một thực thể có nhiều hơn 1 AP, vậy thì sẽ không rõ dùng địa chỉ nào để tham chiếu. Ví dụ, một dịch vụ web được triển khai phân tán trên nhiều server khác nhau.

Thiết kế tên độc lập với địa chỉ sẽ dễ dàng và mềm dẻo hơn khi sử dụng. Tên kiểu đó được gọi là độc lập với vị trí. Có một kiểu tên đặc biệt khác được dùng để chỉ đích xác 1 thực thể, được gọi là định danh.

Các đặc điểm của định danh:

* Một định danh chỉ đến nhiều nhất 1 thực thể
* Mỗi thực thể chỉ được xác định bởi 1 định danh.
* Một định danh mãi mãi chỉ trỏ đến một thực thể.

Với việc sử dụng định danh, dễ dàng để trỏ đến một thực thể. Ví dụ 2 tiến trình liên hệ với một thực thể bởi định danh. Để kiểm tra xem 2 tiến trình đó có cùng trỏ đến 1 thực thể, chỉ cần kiểm tra xem định danh của chúng có giống nhau hay không.

Như vậy, nếu địa chỉ có thể gán cho 1 thực thể khác thì nó không được gọi là định danh. Địa chỉ và định danh là 2 kiểu quan trọng của tên. Từ đó dẫn đến các vấn đề cần xem xét:

* Khi thực thể không còn tồn tại, liệu định danh của thực thể đó có còn được giữ lại không?
* Khi sử dụng lại các định danh, liệu có dẫn đến khả năng bị trùng?
* Không gian định danh không thể là vô hạn. Từ đó có thể sẽ dẫn tới trường hợp bị cạn kiệt định danh.

Các ví dụ của định danh:

* URI (Uniform resource identifier):
* URL (Uniform resource locator):
* URN (Uniform resource name):
* IP Address, TCP port, MAC,…

Về các dịch vụ tên và thư mục:

* Dùng để lưu trữ các thông tin về tập hợp các thực thể
* Thực hiện ánh xạ giữa tập các thuộc tính
* Phân giải tên: là quá trình biến đổi tên thành địa chỉ, vd đổi tên, đổi địa chỉ theo kiểu tĩnh và động, csdl có thể sử dụng csdl tập trung hoặc phân tán.
* Yêu cầu của dịch vụ tên:
  + Quy mô: phải vô hạn về tên và miền tên
  + Tính bền vững: chịu được các sự thay đổi
  + Tính sẵn sàng: chịu lỗi, chịu rủi ro bảo mật

VD về dịch vụ thư mục/tên:

File system (Linux), Domain name system (DNS), Directory services, LDAP…

1. Đặt tên phẳng

Vd về không gian tên phẳng: sđt, email…

Vì là chuỗi bit không có cấu trúc nên được gọi là không gian tên phẳng. Vì nó không có cấu trúc nên không có khả năng tìm được access point để xác định địa chỉ và vị trí.

Từ đó dẫn đến nhu cầu tìm cách để phân giải, xác định vị trí của các thực thể trong không gian tên phẳng này.

Một số giải pháp được đưa ra như sau:

* Các giải pháp đơn giản: các cơ chế về broadcast, chuyển tiếp con trỏ.
* Hướng tiếp cận home-based
* Sử dụng mảng băm phân tán
* Các hướng tiếp cận phân cấp

2.1. Hướng tiếp cận đơn giản

Quảng bá (broadcasting) và thông báo nhóm (multicasting)

Điều kiện thực hiện: hpt hỗ trợ việc trao đổi thông tin thông qua quảng bá.

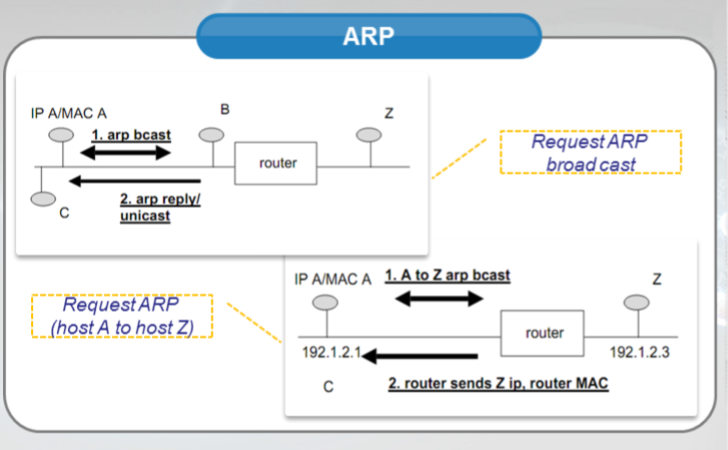
Một thông báo có chứa định danh cần được phân giải và quảng bá đến tất cả các thực thể trong hệ thống. Thực thể nào có đúng định danh trong thông báo nhận được sẽ quảng bá một thông báo chứa định danh và địa chỉ của thực thể. Tất cả các thực thể khác sẽ nhận được thông báo này, và có được ánh xạ giữa định danh và địa chỉ của thực thể nói trên.

Khi muốn xác định một thực thể nào thì sẽ gửi tin quảng bá kèm theo định danh. Các máy tính nhận được thông điệp sẽ kiểm tra xem mình có thực thể đó không. Chỉ máy có cung cấp điểm truy cập cho thực thể đó sẽ trả lời thông điệp có chứa địa chỉ của điểm truy cập đó.

VD: cơ chế giao thức ARP

Vấn đề: việc quảng bá cho tất cả các máy gây tốn tài nguyên mạng và ảnh hưởng tới các máy khác trong mạng.

Một giải pháp cải tiến là có thể chuyển sang sử dụng thông báo nhóm.



Phương thức hoạt động của giao thức ARP (Address resolution protocol):

Để các máy có thể liên lạc với nhau trong cùng một môi trường mạng, cần phải có một cơ chế diễn giải địa chỉ giữa IP và MAC. Giao thức ARP dùng để giải quyết vấn đề đó. Khi một thiết bị mạng cần biết địa chỉ MAC của một thiết bị mạng nào đó trong cùng một mạng, nó sẽ gửi một ARP request bao gồm địa chỉ MAC của nó và địa chỉ IP của thiết bị mà nó cần biết. Mỗi một thiết bị nhận được request này sẽ so sánh địa chỉ IP trong thông điệp request với địa chỉ ở tầng network của mình. Nếu trùng địa chỉ thì thiết bị đó phải gửi ngược lại cho thiết bị gửi một ARP request có chứa một gói tin, trong đó có chứa địa chỉ MAC của mình. Lúc này, A đã biết địa chỉ MAC của máy tính cần gửi và như thế, máy A bắt đầu truyền gói tin cho B.

Hoạt động của ARP trong môi trường liên mạng như sau:

* Máy A gửi một ARP request để tìm địa chỉ MAC của host X.
* Router trả lời, cung cấp cho máy A địa chỉ MAC của host X.
* Máy A truyền gói tin đến host X của router trong đó có địa chỉ thông tin của máy B
* Router nhận được gói tin từ máy A, chuyển gói tin ra port Y và router, dĩ nhiên trong gói tin có chứa địa chỉ IP của máy B.
* Router sẽ gửi ARP request để tìm địa chỉ MAC của máy B. Máy B sẽ trả lời cho router biết địa chỉ MAC của mình.
* Sau khi nhận được địa chỉ MAC của máy B, router sẽ gửi gói tin của A đến cho B.

Vấn đề của ARP: không đề cập đến vấn đề xác thực nào cả. Khi một host nhận được gói tin ARP reply, nó hoàn toàn tin tưởng và mặc nhiên sử dụng thông tin đó để sử dụng sau này mà không cần biết thông tin đó có được trả lời từ một host mà mình mong muốn hay không.

ARP không có cơ chế nào để kiểm tra việc đó, và trên thực tế, một host có thể chấp nhận gói ARP reply mà trước đó không cần phải gửi gói tin ARP request. Lợi dụng điều này, hacker có thể triển khai các phương thức tấn công như man-in-the-middle, DoS, MAC flooding, vv…

Với hình thức man in the middle, giả sử hacker muốn theo dõi host A gửi thông tin gì cho host B, đầu tiên, hacker sẽ gửi gói tin ARP reply đến host A với nội dung là địa chỉ MAC của hacker IP của host B. Tiếp theo, hacker sẽ gửi gói tin ARP reply tới host B với nội dung là địa chỉ MAC của hacker và IP của host A. Như vậy, cả 2 host A và host B đều tiếp nhận gói tin ARP reply đó, và lưu vào trong ARP table của mình. Đến lúc này, khi host A muốn gửi thông tin đến host B, nó liền tra vào ARP table và thấy có sẵn thông tin về địa chỉ MAC của host B nên sẽ lấy thông tin đó ra để sử dụng. Nhưng thực chất, địa chỉ MAC đó là của hacker. Đồng thời, máy tính của hacker sẽ mở chức năng IP forwarding, giúp chuyển tải nội dung mà host A gửi qua host B. Host A và host B giao tiếp bình thường và không có cảm giác bị qua máy trung gian là máy hacker.

Với hình thức tấn công DoS, cũng vận dụng kĩ thuật trên, hacker tiến hành tấn công bằng cách gửi gói tin ARP reply đến toàn bộ các host trong mạng với nội dung mang theo địa chỉ IP của gateway và địa chỉ MAC không hề tồn tại. Như vậy, các host trong mạng sẽ tưởng rằng mình đã biết được địa chỉ MAC của gateway, và khi gửi thông tin đến gateway, kết quả là gửi đến một nơi hoàn toàn không tồn tại. Toàn bộ các host trong mạng đều không thể truy cập được internet.

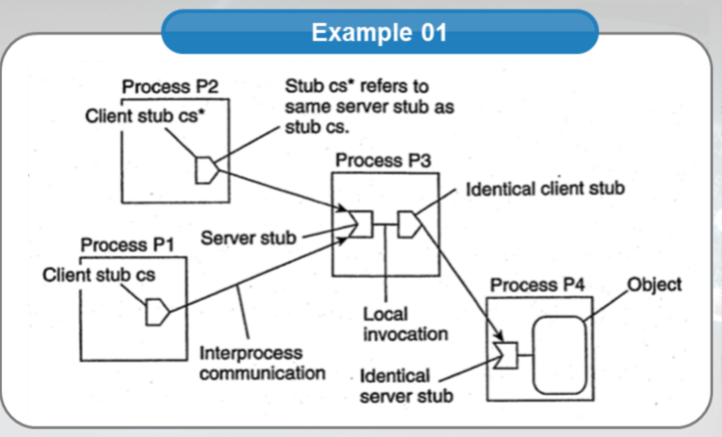
Với cơ chế tấn công MAC Flooding, đối tượng nhắm đến là các host. Hacker sẽ gửi một gói ARP reply giả tạo với số lượng khổng lồ, nhằm làm switch xử lý không kịp và trở nên quá tải. Khi đó switch sẽ không đủ sức để thực hiện bản chất layer 2 của mình nữa, mà broadcast gói tin ra toàn bộ các port của mình. Hacker dễ dàng bắt được toàn bộ thông tin trong mạng.

Cơ chế chuyển tiếp con trỏ (forwarding pointer)

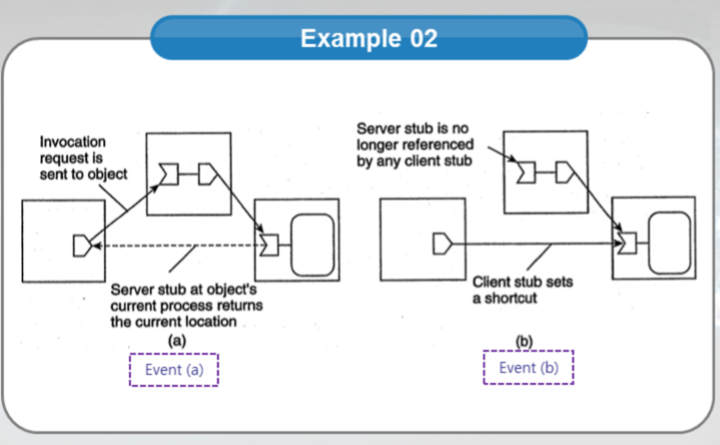
Cơ chế hoạt động: xuất phát điểm, tất cả các thực thể đều có thông tin ánh xạ đến tên và địa chỉ của các thực thể khác. Khi các thực thể thay đổi địa chỉ, sẽ để lại tham chiếu mới tại địa chỉ cũ cho các thực thể khác có thể truy cập. Sau quá trình vận hành, các thực thể sẽ tham chiếu đến nhau thông qua một chuỗi các con trỏ. Hệ thống sẽ bao gồm: bảng ánh xạ bạn đầu, tập hợp các con trỏ trên các địa chỉ cũ.

Cơ chế này dẫn đến một số nhược điểm sau:

* Chuỗi các tham chiếu cho một thực thể di động có thể rất dài, khiến cho quá trình tìm địa điểm trở nên tốn kém.
* Các nút trung gian phải lưu các con trỏ chuyển tiếp lâu nhất có thể.
* Khi một con trỏ tham chiếu bị hỏng, không thể tìm được thực thể đó.



Lấy ví dụ về việc truy cập đối tượng từ xa: mỗi con trỏ chuyển tiếp là một cặp client stub và server stub. Server stub chứa tham chiếu đến một đối tượng hiện tại bằng một tham chiếu đến client stub từ xa của đối tượng đó. Khi một đối tượng chuyển từ tiến trình A sang tiến trình B, nó sẽ để lại client stub ở A, sau đó tạo một server stub ở B. Điều hay ở quá trình di trú này là nó hoàn toàn trong suốt với người dùng. Tất cả những gì ng dùng nhìn thấy chỉ là client stub của họ.



Mô hình chuyển tiếp con trỏ sau một thời gian vận hành, cùng với sự dịch chuyển nhiều các đối tượng trong hệ thống sẽ nảy sinh một số vấn đề sau:

* Số lượng các con trỏ tăng lên
* Chiều dài chuỗi các con trỏ cũng tăng lên
* Ảnh hưởng hiệu năng của hệ thống, dẫn đến mất con trỏ.

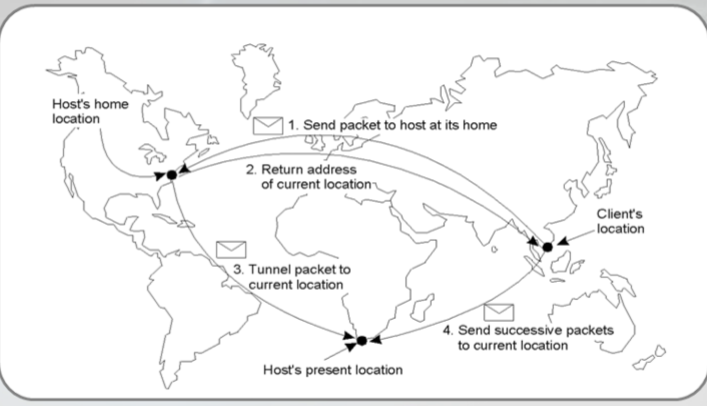
Để giải quyết vấn đề trên, cần có cơ chế loại bỏ các con trỏ:

Khi nhận được lời triệu gọi đối tượng, có định danh của client stub ở nơi khởi đầu lời triệu gọi. Sau đó, kết quả được gửi trả về theo đường đến. Khi nhận được kết quả với vị trí của đối tượng, client stub sẽ cập nhật lại và tạo một shortcut tới đối tượng đó. Khi một server stub không được trỏ đến bởi bất cứ client stub nào, thì nó có thể bị loại bỏ.

Các vấn đề với chuyển tiếp con trỏ:

* Chuỗi các con trỏ dài vô hạn có thể giải quyết bằng shortcut, tuy nhiên có trường hợp không tham chiếu được.
* Lưu trữ vô số các tham chiếu: vấn đề này được giải quyết bằng việc loại bỏ các tham chiếu, cụ thể là loại bỏ các server stub không còn tham chiếu nào trỏ tới nó nữa.
* Khi một tiến trình trong một chuỗi các client stub và server stub bị hỏng, sẽ không thể tham chiếu đến một đối tượng nào đó nữa.
* Giải pháp home-based: tại máy gốc, nơi mà đối tượng được khởi tạo ra, luôn luôn lưu trữ một tham chiếu trỏ đến vị trí của đối tượng đó. Khi chuỗi bị đứt hoặc hỏng, sẽ tiến hành hỏi máy gốc xem đối tượng đang ở đâu.

2.2. Phương thức home-based



Với các giải pháp trước như quảng bá, chuyển tiếp con trỏ, nảy sinh vấn đề về độ khả mở của hệ thống. Vd trong các mạng rất rộng lớn, việc quảng bá hay thông báo nhóm sẽ làm rất tốn tài nguyên hệ thống mạng, còn chuyển tiếp con trỏ làm ảnh hưởng tới hiệu năng hệ thống, và nhất là khi các chuỗi bị đứt.

Vì vậy, hướng tiếp cận đối với các thực thể di động trong một mạng quy mô lớn đó là dùng home location. Nó lưu trữ vị trí hiện tại của thực thể. Thực tế thì home location luôn được chọn là nơi mà thực thể được sinh ra. Để mô tả cơ chế, ví dụ trong mobile IP: mỗi một host di động có một IP cố định. Mỗi một kết nối đến IP đó đều phải liên hệ với home agent trước.

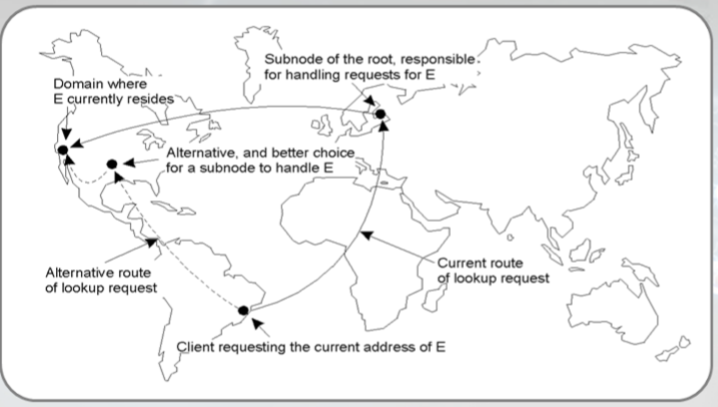
Nhược điểm:

* Mỗi khi muốn liên lạc với một thực thể, client phải liên lạc trước với home agent, dù nó ở rất xa. Từ đó dẫn đến độ trễ trao đổi thông tin. VD trên hình vẽ: khi khách hàng ở ĐNÁ muốn liên lạc với đối tượng ở Nam Phi, tuy nhiên vị trí home của đối tượng lại ở Bắc Mỹ, vì vậy khách hàng phải liên lạc tới tận Mỹ để có địa chỉ đối tượng ở Nam Phi, sau đó mới có thể liên lạc tới đối tượng đó.
* Vị trí của home agent là tĩnh, vì vậy phải đảm bảo home agent luôn tồn tại, nếu không thì sẽ không liên lạc được với thực thể.

Giải pháp:

Home agent di chuyển theo thực thể.

Home agent đăng kí dịch vụ định danh truyền thống, cho phép client đầu tiên tìm địa chỉ của home agent. Vì vị trí của home agent ít thay đổi nên cho phép sử dụng cơ chế bộ đệm để lưu trữ vị trí đó.



Giải pháp cho mô hình này đó là sử dụng home thay thế. Việc nằm gần hơn với đối tượng sẽ giải quyết vấn đề khoảng cách liên lạc, tăng độ trễ truyền tin. Trong ví dụ trên hình, khách hàng thay vì phải liên lạc đến tận châu Âu để tìm địa chỉ của một người ở Mỹ, thì họ chỉ cần liên lạc đến một home khác cũng ở Mỹ, vì vậy khoảng cách rõ ràng là đã được cải thiện. Vấn đề đặt ra là làm sao có thể phát hiện được địa chỉ của home agent gần nhất. Để giải quyết vấn đề này, chúng ta có thể sử dụng dịch vụ thư mục để tìm kiếm home agent gần nhất. Việc sử dụng home thay thế thì cần phải xem xét sự cân bằng giữa home agent thay đổi liên tục và ít thay đổi.

* 1. Bảng băm phân tán

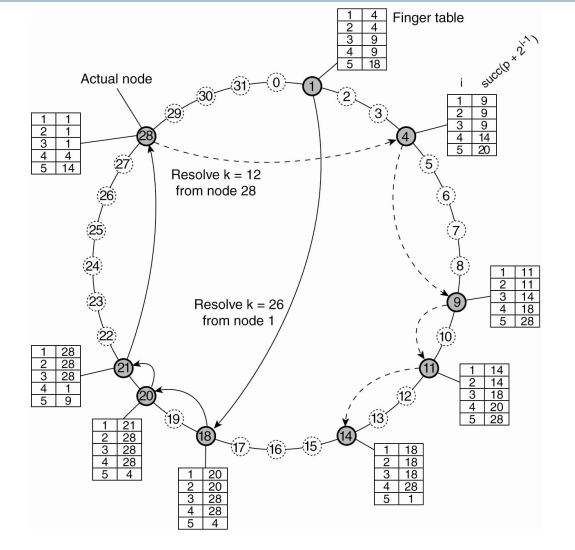
Hệ thống đặc trưng sử dụng bbpt: Chord system.

Hệ thống Chord sử dụng hệ không gian định danh m-bit để gán một cách lựa chọn ngẫu nhiên các định danh cho các nút cũng như các khoá cho các thực thể xác định. Các thực thể có thể là bất cứ gì: các tệp, các tiến trình, vv…

Một thực thể với khoá k sẽ nằm dưới sự quản lý của node với định danh ID nhỏ nhất mà lớn hơn k. Node đó được gọi là succ(k).

Vấn đề chính của hệ thống dựa trên bảng băm phân tán là phân giải một cách hiệu quả một khoá k đến địa chỉ của succ(k), thực chất là một hệ thống định tuyến để xác định vị trí các địa chỉ của khoá k.

Khi không có sự tham gia của các giá trị băm, một nút có các bảng định tuyến đến nút trước và sau trong vòng, và khi nhận được yêu cầu tìm kiếm khoá k, nó sẽ chuyển tiếp theo bảng định tuyến cho đến khi gặp nút có pred(p)<k<p. Ở đây có sự tham gia của các hàm băm, vì vậy bảng định tuyến có nhiều dòng hơn, cho nhiều trường hợp hơn, và cần có sự đơn giản để cập nhật bảng.



Xem xét hệ thống Chord

* Thiết lập vòng bằng các biến cục bộ prev(n) và succ(n)
* Sử dụng bảng băm để xác định địa chỉ succ(k) của tên k
* Với FTp là finger table của node p:



* Khi cần tìm khóa k, node p sẽ gửi cho node q:



* Cập nhật bảng băm khi có các nút được thêm vào.

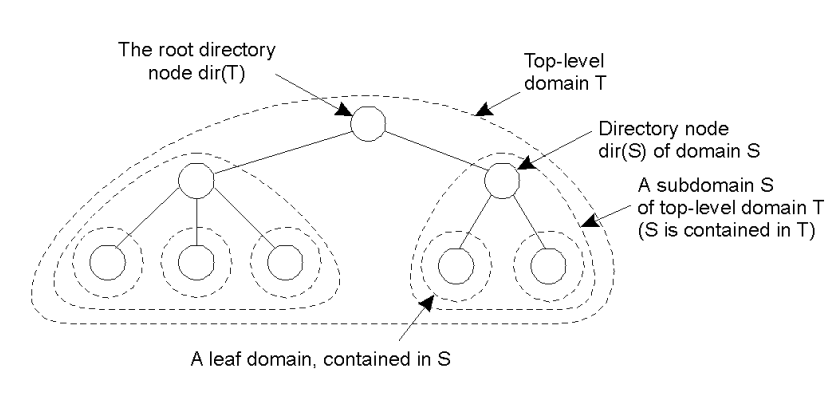
Ở hình trên, ta thấy node 1 cần xác định khoá k = 26. Đầu tiên, nó sẽ tra trong bảng. 26 lớn hơn mọi giá trị trong bảng, nên sẽ chon node số 18 để chuyển tiếp. Khi đến node số 18, tra trong bảng finger ta thấy 20 < 26 < 28, nên chuyển tiếp cho 20. Tương tự như vậy, yêu cầu dãy kết thúc ở node 28, chính là node quản lý k = 26. Việc phân giải khoá k = 12 từ node 28 cũng được thực hiện tương tự, và được kết thúc ở node 14.

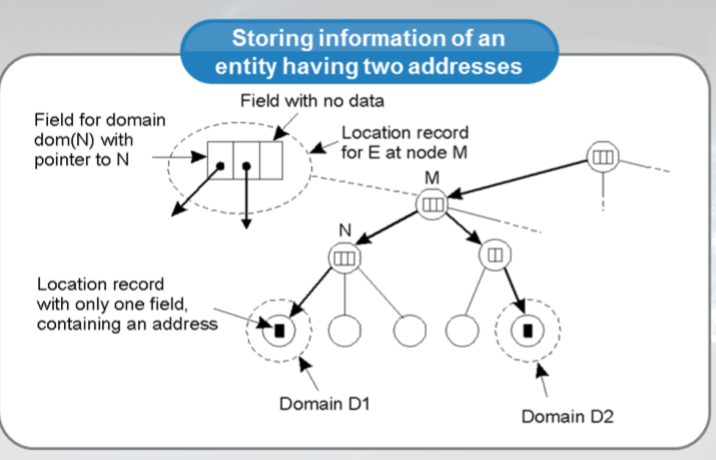
* 1. Giải pháp phân cấp

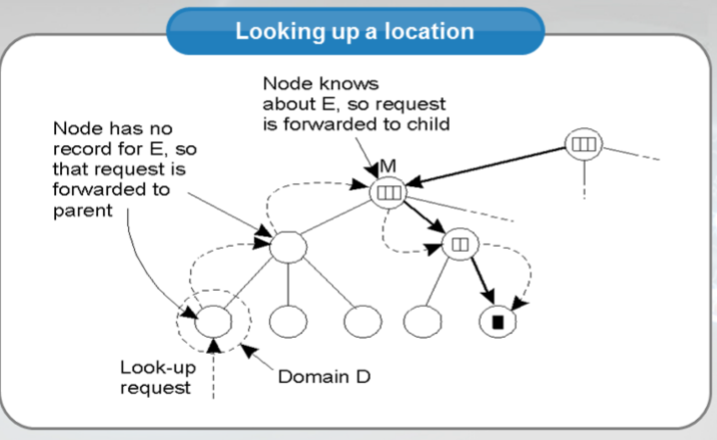
Mạng được chia ra làm tập hợp các domains. Domain đỉnh đơn (single-top domain): domain lớn nhất, kết nối với toàn bộ mạng. Mỗi domain có thể chia ra thành các domain con (sub-domain). Domain mức thấp nhất được gọi là domain mức lá, tương đương với một mạng LAN, hay một thiết bị di động trong mạng di động.

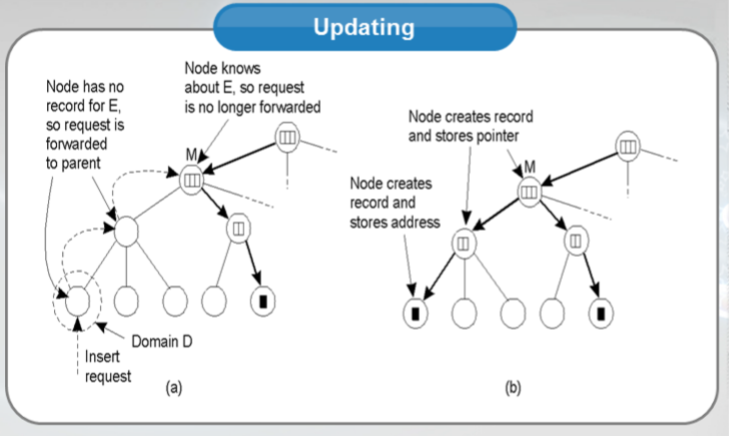
Mỗi domain có một directory node, nó theo dõi kiểm soát tất cả các node thành viên trong domain đó. Cấu trúc này dẫn đến một mô hình cây thư mục các node. Directory node của domain mức cao nhất được gọi là node root, biết thông tin về tất cả các thực thể.

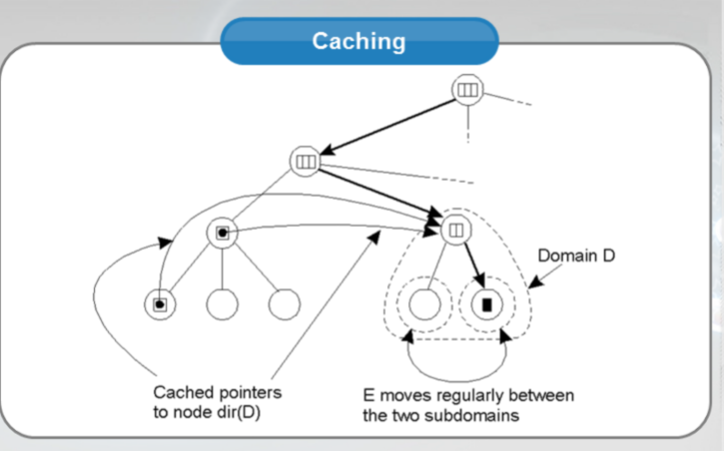
Một thực thể sẽ có trong tất các các nút thư mục từ nút lá đến nút gốc. Trong một nút thư mục, bản ghi có dạng thực thể, con trỏ đến nút thư mục có chứa thông tin về thực thể. Nếu có 2 địa chỉ, sẽ có 2 bản ghi trỏ vào 2 nút thư mục con chứa 2 địa chỉ riêng biệt.











Với không gian tên phẳng, các tên sẽ dần bị sử dụng hết. Điều này khiến chúng ta cần loại bỏ bớt các tên. Từ đó, các tên không được sử dụng sẽ phải bị loại bỏ. Vấn đề đặt ra là cần phải xác định tên nào là tên không được sử dụng. Có một số giải pháp như sử dụng bộ đếm tham chiếu và danh sách tham chiếu, xác định các đối tượng nào không kết nối được. Ngoài ra, có thể tham khảo giải pháp sử dụng giải thuật vét cạn: tìm tất cả các tiến trình trong hệ thống, lấy danh sách tất cả các tên đang sử dụng.

Danh sách tham chiếu:

Các tên trước khi đưa vào hoạt động sẽ phải được đăng kí.

Việc đăng kí sử dụng có thời hạn sẽ làm tăng hiệu năng của hệ thống khi kiểm soát không gian tên một cách chặt chẽ. Khi tên hết thời hạn, có thể sử dụng lại cho một thực thể khác.

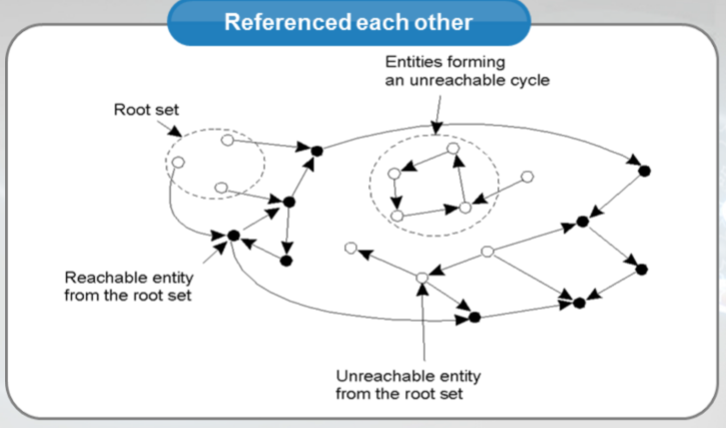
Để phát hiện được các thực thể không thể kết nối:

Với hệ tập trung: tiến hành theo dõi tham chiếu

Hệ phân tán: theo dõi tham chiếu theo nhóm

Đối với các đối tượng ko có tham chiếu nào thì cần bị loại bỏ.

Địa chỉ, tên của các đt không được tham chiếu bởi bất cứ một tiến trình, thực thể nào khác thì cần phải bị loại bỏ. Cần xác định liệu địa chỉ đó có còn được tham chiếu bởi một thực thể nào khác hay không. Từ đó dẫn đến nhu cầu quản lý mỗi khi thực thể được tham chiếu. Cần xác định liệu các tham chiếu có còn được sử dụng nữa hay không, và cần xác định liệu nhóm thực thể có còn được tham chiếu bởi các thực thể khác hay không, nhằm tránh trường hợp tự tham chiếu lẫn nhau, tạo ra các vùng rời, không sử dụng.



Con đếm tham chiếu:

Giải pháp được sử dụng trong hệ thống tệp UNIX, với cách thức sử dụng con đếm tham chiếu.

Mỗi khi có tham chiếu đến địa chỉ, nó sẽ tăng con đếm lên 1 đơn vị.

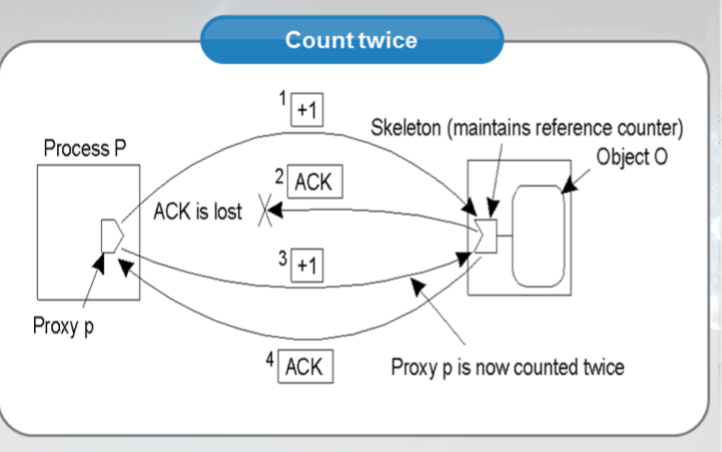
Ngược lại, mỗi khi loại bỏ tham chiếu nó sẽ giảm con đếm đi 1 đơn vị.

Khi số lượng tham chiếu bằng 0, có thể coi thực thể đó không có tham chiếu, ví dụ như sử dụng các lệnh ln và rm.

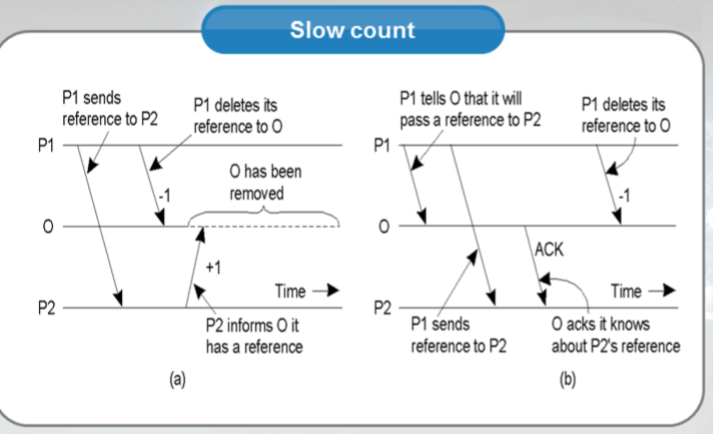
Với các hệ thống tập trung, khi dùng các lệnh ln và rm với nguồn và đích trên cùng một hệ thống sẽ không xuất hiện vấn đề. Nếu nguồn và đích nằm trên các hệ thống khác nhau, có thể sẽ có vấn đề xảy ra.

Một số vấn đề cần xem xét:

* Đếm 2 lần
* Đếm chậm

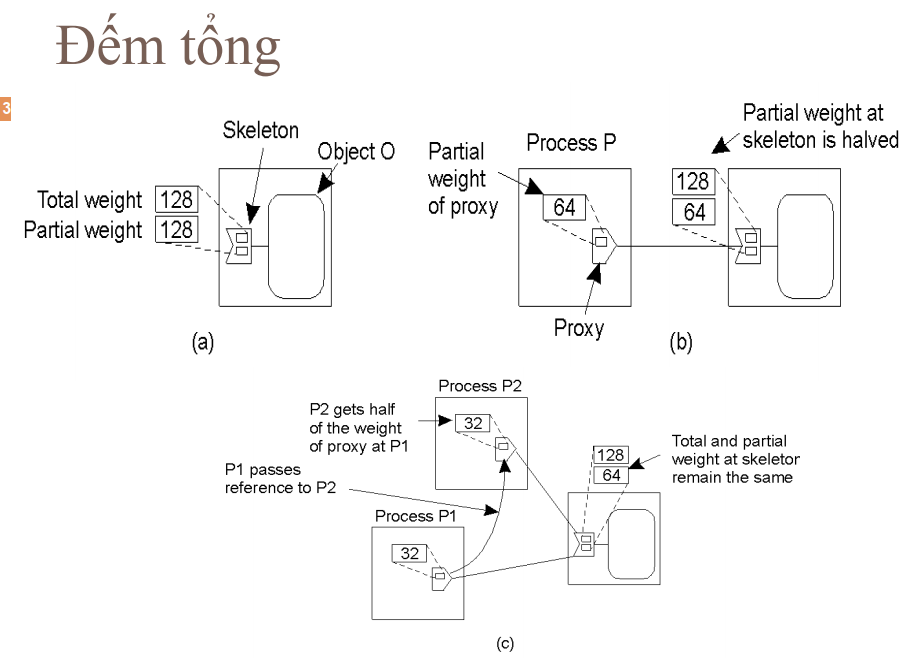


Khi thực hiện một tham chiếu từ hệ thống nguồn đến hệ thống đích, bộ đếm tham chiếu ở hệ thống đích sẽ tăng lên 1 đơn vị. Tuy nhiên quá trình gửi thông điệp ACK lại ko đến được với hệ thống nguồn. Điều này làm hệ thống nguồn tưởng việc thực thi không thành công và gửi lại yêu cầu 1 lần nữa. Lúc này hệ thống đích sẽ tăng bộ đếm tham chiếu lên 1 đơn vị nữa, và như vậy với cùng 1 tham chiếu đã bị đếm đến 2 lần. VD: khi thực hiện lệnh rm, tệp đích đã được loại bỏ khỏi thư mục đích, nhưng nó chưa kịp báo cho hệ thống nguồn. Counter của hệ thống nguồn chưa kịp cập nhật thì đã bị ngắt. Điều đó dẫn đến việc đếm thừa.



Ở hình a, tiến trình P1 đang có tham chiếu đến đối tượng O. P1 muốn sao chép remote reference của đối tượng O sang cho P2. P1 muốn xoá reference đến O, nên gửi -1. Lúc này, do P2 báo +1 cho skeleton của O quá chậm, O thấy số reference của mình = 0 nên đã tự huỷ.

Ở hình b mô tả giải pháp…



Các skeleton sẽ khởi tạo giá trị. Mỗi lần có tham chiếu mới skeleton tiến hành chia tổng trọng số cho các tham chiếu mới được tạo thành.