BÀI LÝ THUYẾT SỐ 2

CÁC HỆ THỐNG PHÂN TÁN VÀ ỨNG DỤNG

CHƯƠNG 2: TIẾN TRÌNH VÀ TRAO ĐỔI THÔNG TIN TRONG HỆ PHÂN TÁN

Họ và tên: Nguyễn Duy Khánh Mã lớp: 157542

MSSV: 20225019 Mã học phần: IT4611

Câu hỏi 1: Có cần thiết phải giới hạn số lượng các luồng trong một tiến trình server?

Trong một tiến trình Server, giới hạn số lượng các luồng là điều cần thiết.

Vì khi tạo thêm luồng thì tiến trình phải khởi tạo thêm 1 stack, càng nhiều luồng thì càng dùng nhiều stack, tốn càng nhiều stack, có thể gây quá tải bộ nhớ.

Việc lập trình để cho các luồng không ảnh hưởng lẫn nhau là tốn kém, tốn chi phí lập trình, dễ bị xung đột lẫn nhau giữa các luồng, vì vậy giới hạn số lượng các luồng để cân bằng chi phí lập trình, đạt hiệu quả cao

Câu hỏi 2: Có nên chỉ gắn một luồng đơn duy nhất với một tiến trình nhẹ?

Không cần thiết phải gắn một luồng đơn duy nhất cho một tiến trình nhẹ.

Vì mỗi tiến trình nhẹ giữ một bảng luồng để tránh việc cùng sử dụng một luồng chính, còn việc tránh truy cập cùng lúc vào dữ liệu chia sẻ sẽ được đảm đương hoàn toàn ở mức người dùng. Lúc đó, nếu có một luồng gọi lời gọi hệ thống dừng thì nó chỉ block tiến trình nhẹ tương ứng, không block các tiến trình nhẹ khác, dẫn đến không ảnh hưởng các luồng khác.

Câu hỏi 3: Có nên chỉ có một tiến trình nhẹ đơn gắn với 1 tiến trình?

Việc chỉ có một tiến trình nhẹ đơn gắn với 1 tiến trình là không nên vì nó không giải quyết được vấn đề Block System Call. Vấn đề lời gọi hệ thống sẽ không được giải quyết khi một tiến trình, tiến trình bị ngắt sẽ ngắt luôn tiến trình nhẹ làm cho các luồng đang thực hiện cũng bị ngắt. Ngoài ra, nếu chỉ có một tiến trình nhẹ thì cũng không có ý nghĩa trong trường hợp đa luồng.

Câu hỏi 4: Bài toán này yêu cầu bạn so sánh thời gian đọc một tệp (file) của một máy chủ tập tin (file server) đơn luồng và một máy chủ đa luồng. Phải mất tổng cộng 15 ms để nhận 1 yêu cầu (request) và thực hiện quá trình xử lý, giả định rằng các dữ liệu cần thiết nằm ở bộ nhớ đệm trong bộ nhớ chính. Nếu cần thiết phải thực hiện một thao tác truy cập ổ đĩa thì cần thêm 75 ms, biết rằng việc phải thực hiện thao tác này có xắc suất là 1/3. Hỏi máy chủ có thể nhận bao nhiêu yêu cầu/giây trong 2 trường hợp: máy chủ là đơn luồng và máy chủ là đa luồng (ngoài luồng nhận và xử lý request, sẽ có thêm 1 luồng để truy cập ổ đĩa nếu cần thiết)? Giải thích.

Trường hợp Server đơn luồng: Server nhận request và xử lý tuần tự từng request

Thời gian trung bình cho việc xử lý 1 Request của Server:

$$\frac{1}{3} * (15 + 75) + \frac{2}{3} * 15 = 40(\text{ms})$$

Vậy số lượng request Server này có thể xử lý trong 1 giây là: $1000 : 40 = 25$ (request)

Trường hợp Server đa luồng: Việc nhận request và truy cập vào ổ đĩa có thể xử lý song song:

Máy chủ có thể nhận $1000 : 15 = 66{,}67$ yêu cầu/giây

Câu hỏi 5: Hệ thống X chỉ định máy của user chứa server, trong khi các ứng dụng lại được coi như client. Điều đó có vô lý không? Giải thích.

Hệ thống X chỉ định máy của User chứa Server, trong khi các ứng dụng khác lại được coi là Client. Điều này không vô lý theo quan điểm của Windows. Quan điểm này nêu ra rằng:

Các máy của User-terminal coi như X-Server

Các máy của application-server coi như X-client.

Lúc đó, user-terminal sẽ phục vụ cho các application-server như X-client

Câu hỏi 6: Giao thức thiết kế cho hệ thống X gặp phải vấn đề về tính mở rộng. Chỉ ra các giải pháp để giải quyết vấn đề đó?

Giải pháp giải quyết vấn đề:

- Cải thiện giao thức X-protocol
- Nén thông điệp lại

Câu hỏi 7: Với việc xây dựng một server đồng thời, hãy so sánh việc server này tạo một luồng mới và tạo một tiến trình mới khi nhận được yêu cầu từ phía client.

	Đa tiến trình	**Đa luồng**
Giống nhau	Xử lý song song nhiều công việc.	
Khác nhau	Chi phí lập trình thấp. Không tốn kém chi phí lập trình (sự quản lý tương tranh giữa các tiến trình là nhiệm vụ của hệ điều hành) Mỗi tiến trình có không gian bộ nhớ riêng, tốn nhiều tài nguyên hơn.	Chi phí lập trình cao. Các luồng trong cùng một tiến trình chia sẻ cùng không gian địa chỉ, giúp tiết kiệm bộ nhớ.

	Không cần quan tâm xung đột giữa các tiến trình. Mỗi tiến trình được tạo có tài nguyên riêng, có môi trường riêng. Hệ điều hành tự đảm bảo không để xảy ra xung đột. Chuyển đổi ngữ cảnh giữa các tiến trình và tốn kém tài nguyên hơn do hệ điều hành tách riêng tài nguyên. Lời gọi hệ thống dừng không xảy ra	Lập trình viên phải tự đảm bảo vấn đề lập trình xung đột giữa các luồng. Nhẹ nhàng chuyển ngữ cảnh. việc chuyển ngữ cảnh giữa các luồng gần như là không tốn kém do ngữ cảnh các luồng chính là ngữ cảnh của tiến trình sinh ra nó. Lời gọi hệ thống có xảy ra. Xảy ra khi lập trình đa luồng, có một luồng gọi hệ thống dừng ví dụ như lời gọi vào ra ⇒ làm treo tiến trình sinh ra nó và làm treo các luồng đang thực hiện song song

Câu hỏi 8: Nếu bây giờ một webserver tổ chức lưu lại thông tin về địa chỉ IP của client và trang web client đó vừa truy cập. Khi có 1 client kết nối với server đó, server sẽ tra xem trong bảng thông tin, nếu tìm thấy thì sẽ gửi nội dung trang web đó cho client. Server này là có trạng thái (stateful) hay không trạng thái (stateless)?

Theo định nghĩa **Stateful server** là server trạng thái bắt buộc phải lưu lại thông tin các phiên làm việc của client trong quá khứ để phục vụ cho công việc hoặc trả lời client trong tương lai. Do đó, Server này là có trạng thái (stateful) vì nó lưu lại dữ liệu của client.

Câu hỏi 9: So sánh Docker và Virtual Machine

Giống nhau: Docker và VM đều là hai công nghệ quan trọng trong việc triển khai ứng dụng và quản lý hệ thống.

Khác nhau:

	Docker	**Virtual Machine**
Thời gian khởi động	Khởi động trong vài giây	Mất vài phút để khởi động
Ảo hóa	Bộ chứa Docker chỉ ảo hóa lớp ứng dụng và chạy trên hệ điều hành máy chủ	VM khởi động hệ điều hành khách của chính nó
Khả năng tương thích	Docker tương thích với mọi bản phân phối Linux	VM tương thích với tất cả các hệ điều hành

Công cụ chạy	Docker sử dụng công cụ thực thi	Sử dụng bộ ảo hóa
Bộ nhớ	Không cần không gian để ảo hóa, do đó ít bộ nhớ hơn	Yêu cầu tải toàn bộ hệ điều hành trước khi khởi động bề mặt, do đó kém hiệu quả hơn
Kích thước	Hình ảnh Docker có trọng lượng nhẹ và thường ở mức kilobyte	Một phiên bản VM có thể lớn tới vài gigabyte hoặc thậm chí terabyte
Triển khai	Việc triển khai rất dễ dàng vì chỉ có một hình ảnh duy nhất, được đóng gói trong vùng chứa có thể được sử dụng trên tất cả các nền tảng	Quá trình triển khai tương đối dài vì các phiên bản riêng biệt chịu trách nhiệm thực thi
Kiến trúc	Chia sẻ tài nguyên với nhân máy chủ cơ bản	Chạy nhân và hệ điều hành riêng
Bảo mật	Nếu hệ điều hành máy chủ dễ gặp phải các lỗ hổng bảo mật thì các bộ chứa Docker cũng vậy	Máy ảo khởi động hệ điều hành của riêng chúng nên nó an toàn hơn
Cách sử dụng	Cơ chế sử dụng phức tạp bao gồm cả công cụ docker và bên thứ ba quản lý	Các công cụ rất dễ sử dụng và làm việc đơn giản hơn

Câu hỏi 10: Trong các giao thức phân tầng, mỗi tầng sẽ có một header riêng. Vậy có nên triển khai một hệ thống mà tất cả các header của các tầng đưa chung vào một phần (gọi là header chung), gắn vào đầu mỗi thông điệp để có thể xử lý chung? Giải thích.

Không nên triển khai một hệ thống sử dụng header chung cho tất cả các tầng trong mô hình phân tầng. Lý do chính là việc này sẽ phá vỡ nguyên tắc phân tầng và gây ra nhiều vấn đề trong thiết kế và hoạt động của giao thức mạng. Ngoài ra, nó còn phá vỡ tính trong suốt của hệ thống. Nếu tất cả các tầng chia sẻ một header chung, sẽ làm mất đi sự độc lập của từng tầng. Khi một tầng thay đổi, có thể cần sửa đổi header chung, dẫn đến ảnh hưởng đến tất cả các tầng còn lại, khó mở rộng và bảo trì hệ thống. Việc giữ nguyên cách mỗi tầng có header riêng giúp duy trì tính mô-đun, dễ bảo trì, mở rộng và tối ưu hiệu suất hệ thống mạng.

Câu hỏi 11: Xét 1 thủ tục incr với 2 tham số nguyên. Thủ tục làm nhiệm vụ là cộng 1 đơn vị vào mỗi tham số. Bây giờ xét trường hợp chúng ta gọi thủ tục đó với cùng một biến, ví dụ incr(i, i). Nếu biến i được khởi tạo giá trị 0, vậy giá trị của i sẽ là bao nhiêu sau khi gọi thủ tục này trong 2 trường hợp sau:

- Lời gọi tham chiếu

- Phương pháp sao chép-phục hồi được sử dụng.

Trong lời gọi tham chiếu, biến i được truyền vào trong thủ tục ở dạng tham chiếu, thủ tục incr sẽ thao tác trực tiếp đến biến i trong bộ nhớ.

Ban đầu i là 0, khi gọi incr(i, i):

1. Tham số đầu tiên (i): tăng lên 1 → i = 1
2. Tham số thứ hai (i) (vẫn là cùng một biến i): tiếp tục tăng lên 1 → i = 2

Phương pháp sao chép - phục hồi:

1. Sao chép giá trị của các tham số thực tế (i = 0) vào hai biến tạm riêng biệt (temp1, temp2).
2. Thực hiện thủ tục trên hai biến tạm đó:
    - temp1 = temp1 + 1 → temp1 = 1
    - temp2 = temp2 + 1 → temp2 = 1
3. Phục hồi kết quả về biến i từ các biến tạm.
    - Vì cả hai tham số đều là i, khi phục hồi temp1 và temp2 về i, giá trị cuối cùng của i sẽ là giá trị của tham số cuối cùng phục hồi, nên i cuối cùng là 1

Câu hỏi 12: Một kết nối socket cần 4 thông tin nào? Tại sao phải cần đủ 4 thông tin đó?

Mỗi Socket gắn với hai thông tin nên mỗi kết nối Socket giữa 2 đối tượng bao gồm 4 thông tin.

4 thông tin đó bao gồm:

- 2 địa chỉ IP của 2 máy phía Client và Server
- 2 số hiệu cổng (Port Number)

Cần phải đủ 4 thông tin trên bởi: Socket là giao diện lập trình ứng dụng mạng được dùng để truyền và nhận dữ liệu trên internet. Giữa hai chương trình chạy trên mạng cần có một liên kết giao tiếp hai chiều để kết nối 2 process trò chuyện với nhau. Để chúng có thể xác định, kết nối được nhau và thông tin được truyền đúng và tới nơi thì cần có cổng và địa chỉ IP.

Câu hỏi 13: Tại sao giao thức yêu cầu-trả lời (request-reply) lại được coi là đồng bộ và tin cậy?

Giao thức yêu cầu – trả lời (request - reply) được coi là đồng bộ và tin cậy vì:

- Không cần cơ chế kiểm soát luồng.
- Không cần cơ chế báo nhận.

- Client gửi request, nó sẽ tự block mình và đợi phản hồi từ server.
- Server tiếp nhận phản hồi từ client và gửi lại cho client.
- Sau khi client tiếp nhận trả lời từ server nó mới tiếp tục làm việc khác.

Câu hỏi 14: Hai vấn đề chính đối với giao thức RPC là gì?

Hai vấn đề chính mà giao thức RPC gặp phải là:

Hệ thống không đồng nhất: Gây ra vấn đề chủ yếu trong việc truyền tham số:

- Không gian nhớ khác nhau
- Cách thức biểu diễn thông tin, dữ liệu ở hai máy khác nhau

Khi có lỗi xảy ra, nếu 1 trong 2 máy bị lỗi thì sẽ không thể triển khai các thủ tục được. Ví dụ nếu máy 2 nhận được lời gọi thì bị treo thì sẽ gây ra hiện tượng chờ đợi vô hạn ở máy 1, hoặc nếu máy 2 đang thực thi yêu cầu thì máy 1 bị treo làm máy 2 khi thực thi xong không biết trả kết quả về đâu.

Câu hỏi 15: Vấn đề đối với truyền tham biến trong RPC là gì? Còn đối với truyền tham chiếu? Giải pháp đưa ra là gì?

Vấn đề với 2 loại tham số trong RPC là:

Vấn đề đối với truyền tham biến:

- Quy ước biểu diễn dữ liệu ở hai máy có sự khác nhau, các dữ liệu không thuộc cùng một kiểu, các kiểu dữ liệu khác nhau được biểu diễn khác nhau.
- Chỉ hoạt động tốt khi hệ thống đầu cuối đồng nhất

Vấn đề đối với truyền tham chiếu:

- Vì phía gọi và phía bị gọi nằm ở hai máy tính khác nhau, nên chúng sẽ được thực thi trên những không gian địa chỉ khác nhau
- Không tham chiếu được tới các dữ liệu có cấu trúc

Giải pháp:

- 2 bên gửi và nhận cùng phải thống nhất về đặc tả tham số (tuân thủ 1 kiểu giao thức)
- Cần thống nhất về định dạng thông điệp, cách biểu diễn cấu trúc dữ liệu cơ bản, kiểu trao đổi thông điệp, triển khai client-sub và server-sub

Câu hỏi 16: So sánh RMI và RPC. Nhược điểm của RMI so với RPC là gì?

Giống nhau:

- Cùng hỗ trợ lập trình với các giao diện

- Dựa trên giao thức yêu cầu – trả lời
- Đều thực hiện phương thức gọi từ xa

Khác nhau:

Các thông số so sánh	RPC	RMI
Đặc điểm	RPC là một trang web cho các thư viện và hệ điều hành.	Nó là một diễn đàn cho java.
Đặc tính	RPC tạo điều kiện thuận lợi cho việc lập trình các thủ tục.	RMI hỗ trợ lập trình hướng đối tượng.
Sự bảo vệ	Không có bảo vệ cho RPC.	Nó cung cấp sự bảo vệ ở cấp độ khách hàng.
Thông số	Cấu trúc dữ liệu thông thường được chuyển đến các thủ tục từ xa	Các đối tượng được truyền cho các phương thức từ xa. Định danh duy nhất → truyền tham chiếu đối tượng
Các Ứng Dụng	Đối với các ứng dụng RPC cơ bản, cần có một số mã	Nhiều mã cho các ứng dụng RMI cơ bản là không cần thiết.

Nhược điểm của RMI so với RPC

RPC	RMI
Có thể triển khai dễ dàng hơn trên nhiều nền tảng do dùng chuẩn chung (HTTP, TCP/IP, XML, JSON...) nên Linh hoạt cao hơn trong triển khai đa nền tảng.	Thường phụ thuộc vào một nền tảng cụ thể (ví dụ: Java RMI cần JVM) nên ít linh hoạt khi triển khai ở môi trường khác nhau.
Thường dùng các giao thức độc lập ngôn ngữ (ví dụ: JSON-RPC, XML-RPC, gRPC...) Dễ dàng tích hợp nhiều ngôn ngữ khác nhau.	Chỉ hỗ trợ tốt các ứng dụng viết bằng cùng một ngôn ngữ lập trình (ví dụ Java RMI chỉ dùng tốt cho Java)
Đơn giản hơn, lập trình viên chỉ cần hiểu về gọi hàm và giao tiếp thông qua dữ liệu đơn giản.	Yêu cầu lập trình viên hiểu rõ các khái niệm liên quan đến đối tượng phân tán, remote reference, và serialization.
Thường gửi dữ liệu dưới dạng các kiểu cơ bản hoặc các cấu trúc đơn giản hơn (dễ dàng tối ưu hơn).	Sử dụng cơ chế serialization và deserialization của đối tượng, dẫn tới chi phí cao hơn trong việc mã hóa/giải mã đối tượng.

Câu hỏi 17: Hàm listen được sử dụng bởi TCP server có tham số là backlog. Giải thích ý nghĩa tham số đó

Tham số backlog xác định số lượng tối đa các kết nối TCP đồng thời mà server có thể giữ ở trạng thái "chờ xử lý" (pending) trước khi server gọi hàm accept() để lấy kết nối đó ra khỏi hàng đợi.

Câu hỏi 18: Trong trao đổi thông tin hướng dòng, những cơ chế thực thi QoS được thực hiện ở tầng nào? Giải thích. Trình bày một số cơ chế thực thi QoS để chứng minh điều đó.

Trong trao đổi thông tin hướng dòng, những cơ chế thực thi QoS được thực hiện ở tầng mạng, dựa trên tầng IP. Do IP là phương thức kết nối đơn giản, best-effort, tức là truyền dòng dữ liệu đi theo con đường ngắn nhất, từ đó giúp quản lý và sử dụng bang thông một cách hiệu quả.

Một số cơ chế thực thi QoS có thể chứng minh điều này, ví dụ như:

Differentiated services

- Phân loại và đặt độ ưu tiên cho các dịch vụ ⇒ Cải thiện chất lượng dịch vụ cho các dịch vụ ưu tiên mà không cần can thiệp vào tầng mạng IP.

Sử dụng bộ đệm để giảm jitter

- Dòng dữ liệu ở nút nhận sẽ không được chạy ngay cho client mà sẽ lưu trước vào bộ đệm rồi mới chạy ⇒ Đảm bảo khoảng thời gian giữa các dữ liệu liên tiếp nhau giống với bên nút gửi.

Forward error correction (FEC): Tỉ lệ mất mát gói tin

- Interleaved transmission
- Tỉ lệ này gây ra bởi tầng mạng IP nhưng không thể can thiệp được ⇒ Phân tán sự mất mát đó ra toàn bộ dòng dữ liệu để cải thiện chất lượng dịch vụ.