# BÀI TẬP TRÊN LỚP MÔN HỌC: HỆ PHÂN TÁN Chương 2: Tìm hiểu tiến trình trong HPT và quá trình Trao đổi thông tin trong HPT

# Sinh Viên : Vũ Văn Trọng Mã số sinh viên : 20184206

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

* Khi 1 tiến trình chạy 1 chương trình nó sẽ sinh ra 1 hay nhiều luồng.
* Sau khi được tiến trình sinh ra mỗi luồng có đoạn mã thực hiện riêng, độc lập với các luồng khác. -> Nếu càng tạo ra nhiều luồng thì càng tốn bộ nhớ chính
* Đồng thời, 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 để có thể cho phép các luồng chia sẻ CPU
* Cần giới hạn số lượng các luồng trong 1 tiến trình server

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

Không cần thiết vì:

* Mỗi tiến trình nhẹ giữ 1 bảng luồng để tránh việc cùng dùng 1 luồng. Việc tránh truy cập cùng lúc vào dữ liệu chia sẻ được đảm đương hoàn toàn bởi mức người dùng.
* Nếu 1 luồng bị dừng nó sẽ thực hiện lời gọi lập lịch. Khi 1 luồng khả chạy khác được tìm thấy nó sẽ chuyển ngữ cảnh sang cho luồng mới đó và tiến trình nhẹ đang chạy luồng đó hoàn toàn không biết việc chuyển ngữ cảnh

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

Viêc sử dụng tiến trình nhẹ là sự kết hợp của việc chạy đa luồng ở chế độ người dùng và nhân.

* Với cơ chế này khi chạy ngữ cảnh ở 1 tiến trình đơn và mỗi tiến trình đơn có nhiều tiến trình nhẹ.
* Khi 1 luồng thực hiện 1 lời gọi hệ thống dừng, việc thực thi được chuyển xuống cho kiểu nhân nhưng vẫn ở trong ngữ cảnh của tiến trình nhẹ hiện tại. HĐH sẽ chuyển ngữ cảnh sang cho tiến trình nhẹ khác, dẫn tới việc chuyển ngữ cảnh chuyển ngược lại về cho kiểu người dùng. => lời gọi hệ thống dùng không làm dừng cả hệ thống lại
  + Như vậy, khi 1 tiến trình có đủ các tiến trình nhẹ, 1 lời gọi hệ thống dừng không làm dùng cả hệ thống mà chỉ làm dừng 1 tiến trình nhẹ.

**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.

- Đơn luồng:

Thời gian để nhận 1 yêu cầu: 15 x 2/3 + (15+75) x 1/3 = 40 (ms)

=> Số yêu cầu/giây: 1000/40 = 250 (yêu cầu)

- Đa luồng: 1000/15 (yêu cầu)  
**Câu hỏi 5:** Hệ thống X chỉ định máy của user chưa server, trong khi các ứng dụng lại được coi  
như client. Điều đó có vô lý không? Giải thích

Điều đó không vô lý

**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 :

* Cải thiện giao thức X
* Nén thông điệp

**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.

* Tạo tiến trình mới -> Xử lý tách biệt tài nguyên -> Đơn giản hơn.

Nhưng việc chuyển CPU giữa các tiến trình rất tốn kém tài nguyên của hệ thống, đồng thời khi có quá nhiều tiến trình sẽ cho hệ điều hành phải kiểm soát chặt chẽ độ bảo mật và an toàn.  Dựa trên nguyên tắc: các thao tác lỗi của 1 tiến trình rất khó có thể ảnh hưởng đến 1 tiến trình khác không phụ thuộc

* Các ứng dụng đa luồng thường giúp chương trình có hiệu năng cao hơn
* Nếu muốn các collection trao đổi với nhau thì nên tạo ra luồng

**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)?

* Stateless là thiết kế không lưu dữ liệu của client trên server. Có nghĩa là sau khi client gửi dữ liệu lên server, server thực thi xong, trả kết quả thì “quan hệ” giữa client và server bị “cắt đứt” – server không lưu bất cứ dữ liệu gì của client.
* Stateful là một thiết kế ngược lại, chúng ta cần server lưu dữ liệu của client, điều đó đồng nghĩa với việc ràng buộc giữa client và server vẫn được giữ sau mỗi request (yêu cầu) của client. Data được lưu lại phía server có thể làm input parameters cho lần kế tiếp.
* Server này có trạng thái stateful, lưu thông tin về địa chỉ IP và trang web client đó vừa truy cập. Thông tin này là tham số đầu vào cho lần request tiếp theo

**Câu hỏi 9:** So sánh Docker và Virtual Machine

|  |  |
| --- | --- |
| **Máy ảo** | **Docker container** |
| Kích thước (dung lượng) lớn. | Kích thước (dung lượng) nhỏ. |
| Hiệu suất hạn chế. | Hiệu suất gốc (native). |
| Mỗi máy ảo sẽ có một hệ điều hành riêng. | Container sẽ sử dụng hệ điều hành của host. |
| Ảo hóa về mặt phần cứng | Ảo hóa về mặt hệ điều hành |
| Thời gian khởi động tính theo phút | Thời gian khởi động tính theo mili giây |
| Phân bổ bộ nhớ theo nhu cầu cần thiết | Yêu cầu ít dung lượng bộ nhớ hơn |
| Hoàn toàn bị cô lập và an toàn hơn | Cô lập ở mức tiến trình, có thể kém an toàn 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 1 hệ thống như vậy vì:

+ khó bảo trì, nâng cấp hệ thống

+ phá vỡ tính độc lập tương đối với nhau thực hiện các chức năng riêng biệt

+ Thay đổi chức năng hoặc giao thức trong 1 tầng thì sẽ làm ảnh hưởng đến tầng khác

* + Mất đi tính trong suốt của hệ thống

+ tại một tầng, cách đóng gói dữ liệu khác nhau theo các giao thức khác nhau -> khó khăn trong việc đóng gói chung vào 1 header

+ khó khăn trong việc bóc tách, xử lý gói tin khi chuyển tiếp

**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.

* Trường hợp 1 :Lời gọi tham chiếu

Mỗi con trỏ đều trở tới i , nếu i tăng 1 đơn vị thì i tăng 2 lần -> I =2

* Trường hợp 2 : Phương pháp sao chép-phục hồi được sử dụng.

Sao lưu không làm tăng giá trị của i

Giá trị ghi đè 2 lần -> I =1

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

* Bản chất của việc trao đổi thông tin giữa các tiến trình là việc trao đổi thông tin giữa các socket của các tiến trình. Vì vậy mỗi tiến trình (cả tiến trình client và tiến trình server) đều phải tạo socket. Mỗi socket cần 2 thông tin: địa chỉ IP của máy, thông tin port gắn với socket đó. Do đó, mỗi kết nối socket cần 4 thông tin: địa chỉ IP của máy Client, port của tiến trình Client, địa chỉ IP máy server và port mà tiến trình Server đang chạy trên máy chủ
* Cần đủ 4 thông tin vì địa chỉ IP để xác định máy gửi và máy nhận trên môi tường mạng internet, cổng port để định danh ứng dụng mà dữ liệu sẽ được gửi tới

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

Giao thức request-reply được coi là đồng bộ và tin cậy vì:

Sau khi gửi request lên cho tiến trình Server thì tiến trình Client tự block mình để chờ reply từ server trả về (thể hiện tính đồng bộ). Tiến trình Server nhận được request, thực thi yêu cầu, trả về kết quả cho Client. Tuy không cần cơ chế báo nhận nhưng khi server gửi thông điệp Reply về cho Client thì gio thức này coi thông điệp Reply cũng đóng vai trò như là thông điệp báo nhận (tính tin cậy)

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

Hai vấn đề chính đối với giao thức RPC:

- Hệ thống không đồng nhất (trong việc truyền tham số)

+ Không gian nhớ khác nhau

+ Cách biểu diễn thông tin khác nhau

- Có lỗi xảy ra trong quá trình gọi thủ tục từ xa

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

- Vấn đề đối với truyền tham biến:

+ Vấn đề khi biểu diễn dữ liệu khác nhau gây hiểu sai về giá trị của các tham biến đó -> sai về logic hoạt động

+ Các dữ liệu không thuộc cùng một kiểu, các kiểu dữ liệu khác nhau được biểu diễn khác nhau

- Vấn đề đối với truyền tham chiếu: Bộ nhớ phân tán

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, và điều này gây ra sự phức tạp không nhỏ

* Giải pháp: mở rộng mô hình RPC ra làm hai loại chính: Synchronous RPC và Asynchronous RPC.

• Asynchronous RPC (RPC không đồng bộ)

- Client gửi tới server lời gọi thủ tục và chờ bản tin chấp nhận từ server.

- Phía server sẽ gửi một tín hiệu ACK về cho client thông báo đã nhận được yêu cầu và bắt đầu thực hiện yêu cầu RPC đó.

- Lúc này client sẽ tếp tục thực hiện công việc của mình mà không chờ kết quả từ server như ở RPC truyền thống.

• Synchronous RPC (RPC đồng bộ)

- Thực hiện hai lời gọi, một từ client và một từ server.

- Client gửi tới server lời gọi thủ tục và chờ bản tin chấp nhận từ server.

- Server gửi bản tin chấp nhận về cho client thông báo đã nhận được yêu cầu và bắt đầu thực hiện yêu cầu RPC đó.

- Lúc này client sẽ tếp tục thực hiện công việc của mình.

- Khi thực hiện thủ tục xong, server sẽ thực hiện lời gọi tới client báo nhận lấy kết quả.

- Client thực hiện ngắt, nhận kết quả và gửi lại cho server tín hiệu ACK đã nhận kết quả thành công.

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

RPC và RMI là các cơ chế cho phép máy khách gọi thủ tục hoặc phương thức từ máy chủ thông qua thiết lập giao tiếp giữa máy khách và máy chủ.

* **Giống nhau:**

**+** Cùng hỗ trợ lập trình với các giao diện

**+** Dựa trên giao thức yêu cầu/trả lời

**+** Mức độ trong suốt

* **Khác nhau:**

|  |  |  |
| --- | --- | --- |
| **Đặc điểm** | **RPC** | **RMI** |
| Hỗ trợ | Lập trình thủ tục | Lập trình hướng đối tượng |
| 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 |
| Hiệu quả | Thấp hơn RMI | Hơn RPC và được hỗ trợ bởi phương pháp lập trình hiện đại (ví dụ: mô hình hướng đối tượng) |
| Tham số | Bắt buộc | không bắt buộc |
| Lập trình | thấp hơn RMI | dễ dàng hơn RPC |
| Chi phí chung | hơn | Ít so sánh |

* **Nhược điểm của RMI so với RPC:**

+ Việc gọi phương thức của đối tượng từ xa luôn phức tạp hơn gọi phương thức cục bộ. Các đối tượng trên hai máy khác nhau hoạt động trên hai tiến trình khác nhau có hai không gian địa chỉ khác nhau nên:

* + Việc tham chiếu đến biến, địa chỉ của đối tượng là khác nhau ở các máy khác nhau
  + Các tham số truyền cho phương thức của đối tượng ở xa phải được đóng gói và chuyển qua mạng đến phương thức thực sự.
  + Lời gọi phương thức từ xa phải thông qua mạng và có thể bị ngắt ngang do mạng gặp sự cố

+ Phụ thuộc vào kết nối mạng

**Câu hỏi 17:** **Giải thích cơ chế trao đổi thông tin hướng thông điệp bất đồng bộ và bền vững.**

* Sử dụng MOM (Message-Oriented Middleware)
* Hệ thống hàng đợi thông điệp hỗ trợ trao đổi thông tin không đồng bộ bền vững
* Nút gửi và nút nhận không cần đồng bộ trong trong quá trình trao đổi thông điệp, nút gửi gửi thông điệp vào hàng đợi lưu trữ trung gian, sau đó nút nhận sẽ lấy yêu cầu từ hàng đợi trung gian đó ra -> phù hợp với dịch vụ chấp nhận độ trễ thời gian cao

+ Mô hình a: nút sender gửi thông điệp vào hàng đợi, đồng thời nút receiver lấy thông điệp từ hàng đợi ra (cả sender và receiver đều đang chạy)

+ Mô hình b: cho phép nút receiver không chạy, tuy nhiên nút sender vẫn tiếp tục gửi thông điệp vào hàng đợi

+ Mô hình c: nút sender không chạy, nút sender tiếp tục lấy thông điệp từ hàng đợi ra

+ Mô hình d: cả nút sender và receiver đều không chạy, các thông điệp được lưu trữ bền vũng trong 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.**

Cơ chế thực thi QoS được thực thi ở 2 tầng là tầng liên kết dữ liệu (data-link layer) và tầng mạng (network layer)

- Ở tầng mạng:

• sử dụng 3 bit đầu tiên trong trường Service Type - ToS trong phần đầu của gói dữ liệu IP và 3 bits đầu tiên (P2 đến P0) dùng để quy định các giá trị đánh dấu độ ưu tiên của packet và các giá trị này được gọi là IP Precedence

• Giá trị IP precedence nằm trong khoảng từ 0 đến 7.

* 3 bits đầu tiên (P2 đến P0): IP Precedence. Do sử dụng 3 bits nên sẽ có 8 giá trị (000 đến 111) định ra độ ưu tiên của gói tin từ thấp đến cao. Giúp router xử lý các gói tin này theo chất lượng dịch vụ
* 3 bits tiếp theo (T2 đến T0): bit T2 (T2=1): Yêu cầu truyền gấp. bit T1 (T1=1): Yêu cầu truyền với đường truyền chất lượng cao. bit T0 (T0=1): Yêu cầu truyền đảm bảo.
* 2 bit cuối (CU1-CU2): Không dùng tới (Currently and Unused).

• Trong trường hợp cần thiết phải phân chia nhiều hơn 8 lớp lưu lượng, chúng ta có thể sử dụng 6 bit đầu tiên của trường ToS gọi là trường DSCP

- Ở tầng liên kết dữ liệu:

• Trong phần đầu của khung dữ liệu ở lớp liên kết dữ liệu không có trường nào phục vụ cho việc phân lớp lưu lượng. Tuy nhiên ta có thể phân lưu lượng dựa vào việc chèn thêm các thẻ định danh VLAN gọi là tag. Mỗi tag gồm 4 byte trong đó trường CoS gồm 3 bit được dùng để phân lớp lưu lượng. Như vậy tại mức liên kết dữ liệu chúng ta cũng có thể phân chia lưu lượng thành 8 lớp với các mức ưu tiên tăng dần tương tự như khi sử dụng IP Precedence tại lớp mạng của gói tin IP.

* 1 số cơ chế thực thi của QoS:

- Cơ chế Guaranteed Services: dữ liệu đi qua mạng được dành riêng 1 băng thông chắc chắn cho dữ liệu. Thực hiện thông qua cơ chế RSVP và CBWFQ của QoS.

- Cơ chế Differentiated Services: cơ chế này cho phép phân lớp, phân loại các dịch vụ khác nhau và đặt độ ưu tiên khác nhau cho các loại dịch vụ. Thực hiện thông qua các tool QoS là PQ, CQ, WFQ và WRED.

- Sử dụng bộ đệm để giảm jitter: dòng dữ liệu được chuyển đến ở nút nhận sẽ không được chạy ngay cho Client mà sẽ được lưu trước vào bộ đệm và sau đó mới chạy để đảm bảo khoảng thời gian giữa các đơn vị dữ liệu liên tiếp nhau là giống với bên nút gửi

- Cơ chế Forward error correction (FEC): phân tán sự mất mát gói tin ra toàn bộ dòng dữ liệu