Câu 1: So sánh DOS và NOS. Tại sao DOS khả thi để áp dụng trong các hệ thống thực?

|  |  |
| --- | --- |
| DOS | NOS |
| Quản lý nhiều tiến trình hoặc nhiều máy tính đồng nhất. | Kết nối nhiều máy tính độc lập với nhau và mỗi máy tính có thể có hệ điều hành khác nhau. |
| Mức độ trong suốt cao. | Mức độ trong suốt thấp. |
| Trên các nút chỉ có thể có cùng 1 hệ điều hành. | Trên các nút có thể có nhiều hệ điều hành khác nhau. |
| Không có tính mở. | Có tính mở. |
| Quản lý tài nguyên toàn cục. | Quản lý tài nguyên theo các nút. |

DOS không khả thi trong hệ thống thực vì:

* Multiprocessor OS: đòi hỏi phải sử dụng máy tính cực mạnh để đáp ứng nhiều người sử dụng.
* Multicomputer OS: đòi hỏi phải sử dụng nhiều máy tính giống hệt nhau.

Tại sao DOS không được ứng dụng trong những hệ thống thực tế?

Vì HĐH DOS không có tính co giãn, tính mở rộng và sử dụng 1 HĐH trên các nút. Các hệ thống phân tán hiện nay có rất nhiều máy tính khác nhau về phần cứng, phần mềm nên nếu DOS không có các tính chất nêu trên thì sẽ không thể ứng dụng vào các hệ thống thực tế.

Câu 2: Server Binding là một cơ chế để xác dịnh một tiến trình nguồn trong một máy chủ

A, mô tả cơ chế Server Binding sử dụng tiến trình Deamon. Hạn chế của cơ chế này?

B, mô tả cơ chế Server Binding sử dụng tiến trình  superserver. Tại sao superserver lại tiếp cận hiệu quả hơn Deamon? Ví dụ về superserver trong hệ thống UNIX.

a.

Mỗi máy server đều có 1 tiến trình daemon để theo dõi điểm cuối(end point) hiện tại của mỗi service được thực hiện bởi máy chủ đồng vị trí(co-located server).

Client đầu tiên sẽ request điểm cuối từ daemon rồi mới kết nối với server.

Hạn chế: Tiêu tốn tài nguyên.

b.

Superserver sẽ kết nối với nhiều điểm cuối của các service. Khi client gửi request tới superserver, superserver sẽ tạo ra 1 server chuyên dụng để xử lý.

Superserver hiệu quả hơn daemon vì trong 1 hệ thống có thể có rất nhiều tiến trình đang hoạt động chờ đợi request từ client. Việc theo dõi nhiều tiến trình này sẽ rất khó khăn.

Example: Inetd. Inetd sẽ nghe từ nhiều cổng cho dịch vụ internet. Khi có 1 request gửi tới, inetd sẽ fork 1 tiến trình mới để xử lý, tiến trình này sẽ tự thoát khi hoàn thành.

1, Business Logic

Thực hiện các nghiệp vụ chính của hệ thống, sử dụng các dịch vụ do lớp Data Access cung cấp, và cung cấp các dịch vụ cho lớp Presentation. Lớp này cũng có thể sử dụng các dịch vụ của các nhà cung cấp thứ 3 để thực hiện công việc của mình(ví dụ như sử dụng dịch vụ của các cổng thanh tóan trực tuyến như VeriSign, Paypal...).

3, Kiến trúc client-server 3 tầng

(Vẽ hình)

Đối với client liên tác với Middleware(Logic tích hợp và Logic ứng dụng) gọi chung là Application Logic

Kiến trúc client server phát triển và cài đặt các ứng dụng trong môi trường phân tán.

Các ứng dụng mức phân tán yêu cầu các dịch vụ mức hệ thông như:

+ Quản lý giao dịch

+ Bảo mật

+ Kết nối client

+ Truy xuất dữ liệu

*Ưu điểm*:

- Dễ dàng mở rộng, thay đổi quy mô của hệ thống: Khi cần tải lớn, người quản trị có thể dễ dàng thêm các máy chủ vào nhóm, hoặc lấy bớt ra trong trường hợp ngược lại.

*Nhược điểm*:

- Việc truyền dữ liệu giữa các tầng sẽ chậm hơn vì phải truyền giữa các tiến trình khác nhau (IPC), dữ liệu cần phải được đóng gói -> truyền đi -> mở gói trước khi có thể dùng được.

- Việc phát triển ứng dụng phức tạp hơn.

**Câu 3:** Một giải thuật bầu chọn (election algorithm) được mô tả như sau:

Các giả định hệ thống:

- Mỗi nút trong hệ thống được gán 1 số ID từ 1 đến N. Mục đích các giải thuật bầu chọn là luôn chọn nút có ID lớn nhất làm coordinator.

- Mỗi nút sẽ lưu trữ một bảng trạng thái lưu các thông tin của các nút khác như: còn chạy tốt hay hỏng, số ID. Giả định là lỗi có thể xảy ra chỉ là lỗi yên lặng (nút hỏng sẽ không trả lời bất cứ thông điệp nào).

- Hệ thống là kết nối đầy đủ (fully connected).

- Hệ thống xác định khoảng timeouts, nếu sau khoảng thời gian này không nhận được câu trả lời từ 1 nút thì coi như nút đó hỏng.

- Hệ thống áp dụng đồng hồ logic của Lamport.

Mô tả thuật toán:

Khi một nút Pi phát hiện coordinator hiện thời bị hỏng, nó sẽ tra trong bảng trạng thái và gửi ELECTION cho nút có ID ngay dưới nút coordinator vừa hỏng. Nếu nút nhận được ELECTION không bị hỏng thì sẽ gửi OK cho Pi và sau đó quảng bá cho cả hệ thống thông điệp COORDINATOR để báo rằng mình là coordinator mới. Nếu nút đó bị hỏng, Pi tiếp tục lặp lại bước trên với lần lượt các nút có ID lớn hơn mình (thông tin lấy trong bảng trạng thái) đến bao giờ nhận được OK, hoặc ID giảm đến chính ID của mình. Nếu ID giảm đến chính ID của mình thì biết rằng mình là coordinator và gửi quảng bá COORDINATOR cho cả hệ thống.

Câu hỏi 1: Số thông điệp cần để hoàn thành bầu chọn coordinator là bao nhiêu?

Câu hỏi 2: Thiết kế bổ sung cho thuật toán trên để xử lý vấn đề khi 1 nút đang từ trạng thái hỏng và được phục hồi về trạng thái chạy. Số thông điệp cần thiết cho việc phục hồi đó là bao nhiêu? (gợi ý: xét 2 trường hợp là ID nút đó nhỏ hơn và lớn hơn coordinator hiện tại).

Câu hỏi 1:

- Gọi số thông điệp cần để hoàn thành bầu chọn coordinator: x

ID nút lớn nhât: N

ID nút coordinator hiện tại: y

ID nút bên dưới gần coordinator nhất và không hỏng : z

=> x= (y-z)+1+(N-1) = N+y-z

Giải thích: y-z: số thông điệp mà nút Pi cân gửi cho đến khi tìm thấy coordinator mới

1: Số thông điệp mà coordinator mới gửi OK cho Pi

N-1: Số thông điệp mà coordinator mới cần phải gửi quảng bá đến N-1 nút còn lại

Câu hỏi 2:

Bổ xung thuật toán: Khi nút w được phục hồi, nó sẽ gửi thông điệp tới N-1 nút còn lại trong hệ thống. Nút Pi nếu phát hiện ra thông điệp của w sẽ xảy ra 2 trường hợp:

TH1: ID của w lớn hơn coordinator hiện tại:

Pi sẽ gửi thông điệp ELECTION cho w sau đó w gửi lại OK cho Pi và gửi thông điệp quảng bá tới N-1 nút còn lại

x = (N-1)+1+(N-1)=2N

TH2: ID của w nhỏ hơn coordinator hiện tại:

Thực hiện lần lượt các bước như thuật toán đề bài

x = (N-1)+N+y-z