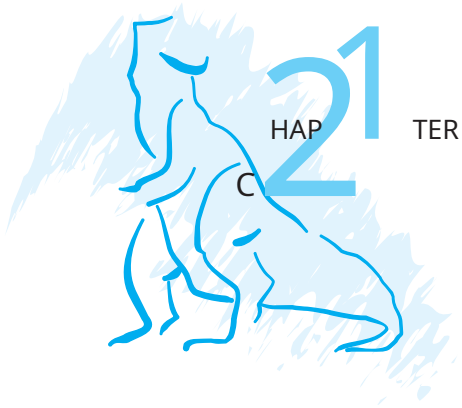

Hệ điều hành Linux

Hệ thống

Bài tập thực hành

21.1 Các mô-đun nhân có thể tải động mang lại sự linh hoạt khi trình điều khiển được thêm vào hệ thống, nhưng chúng cũng có nhược điểm? Trong những trường hợp nào thì một hạt nhân sẽ được biên dịch thành một tệp nhị phân duy nhất và khi nào thì tốt hơn nếu giữ nó được chia thành các mô-đun? Giải thich câu trả lơi của bạn.

Trả lời: Có hai hạn chế chính với việc sử dụng các mô-đun. Đầu tiên là kích thước: quản lý mô-đun sử dụng bộ nhớ nhân không thể gắn thẻ và một nhân cơ bản với một số mô-đun được tải sẽ tiêu tốn nhiều bộ nhớ hơn một nhân tương đương với các trình điều khiển được biên dịch thành chính hình ảnh nhân. Đây có thể là một vấn đề rất nghiêm trọng trên các máy có bộ nhớ vật lý hạn chế.

Hạn chế thứ hai là các mô-đun có thể làm tăng độ phức tạp của quá trình khởi động hạt nhân. Thật khó để tải một tập hợp các mô-đun từ đĩa nếu trình điều khiển cần thiết để truy cập vào đĩa đó một mô-đun cần được tải. Do đó, việc quản lý kernel bootstrap bằng các mô-đun có thể đòi hỏi người quản trị phải làm việc thêm: các mô-đun cần thiết để khởi động cần được đặt vào một ảnh đĩa ram được tải cùng với ảnh hạt nhân ban đầu khi hệ thống được khởi tạo.

Trong một số trường hợp nhất định, tốt hơn là sử dụng nhân mô-đun và trong các trường hợp khác, tốt hơn là sử dụng nhân có trình điều khiển thiết bị của nó được liên kết trước. Trong đó việc giảm thiểu kích thước của hạt nhân là quan trọng, sự lựa chọn sẽ phụ thuộc vào tần suất sử dụng các trình điều khiển thiết bị khác nhau. Nếu chúng được sử dụng liên tục, thì các mô-đun không phù hợp. Điều này đặc biệt đúng khi trình điều khiển cần thiết cho chính quá trình khởi động. Mặt khác, nếu một số trình điều khiển không phải lúc nào cũng cần thiết, thì cơ chế mô-đun sẽ

hạ thấp những trình điều khiển đó được tải và không tải theo yêu cầu, có khả năng cung cấp khả năng tiết kiệm ròng trong bộ nhớ vật lý.

Khi một hạt nhân được xây dựng phải có thể sử dụng được trên nhiều loại máy rất khác nhau, thì việc xây dựng nó bằng các mô-đun rõ ràng là tốt hơn so với việc sử dụng một hạt nhân duy nhất với hàng tá trình điều khiển không cần thiết tiêu tốn bộ nhớ. Điều này đặc biệt xảy ra đối với các hạt nhân được phân phối thương mại, nơi ưu tiên hỗ trợ nhiều loại phần cứng theo cách đơn giản nhất có thể.

Tuy nhiên, nếu một nhân đang được xây dựng cho một máy đơn lẻ có cấu hình được biết trước, thì việc biên dịch và sử dụng các mô-đun có thể đơn giản là một sự phức tạp không cần thiết. Trong những trường hợp như thế này, việc sử dụng các mô-đun có thể là một vấn đề quan trọng.

21,2Đa luồng là một kỹ thuật lập trình thường được sử dụng. Mô tả ba cách khác nhau mà các luồng có thể được thực hiện. Giải thích cách những cách này so với Linux**dòng vô tính**cơ chế. Khi nào thì mỗi cơ chế thay thế có thể tốt hơn hoặc tệ hơn so với việc sử dụng các bản sao? **Trả lời:**Việc triển khai luồng có thể được phân loại rộng rãi thành hai nhóm: luồng dựa trên nhân và luồng chế độ người dùng. Các gói luồng chế độ người dùng dựa vào một số hỗ trợ hạt nhân — chẳng hạn, chúng có thể yêu cầu các phương tiện ngắt hẹn giờ — nhưng việc lập lịch giữa các luồng không được thực hiện bởi nhân mà bởi một số thư viện mã chế độ người dùng. Nhiều luồng trong quá trình triển khai như vậy xuất hiện với hệ điều hành như một ngữ cảnh thực thi duy nhất. Khi quy trình đa luồng đang chạy, nó tự quyết định luồng nào của nó sẽ thực thi, sử dụng các bước nhảy không cục bộ để chuyển đổi giữa các luồng theo quy tắc lập lịch ưu tiên hoặc không ưu tiên của riêng nó.

Ngoài ra, nhân hệ điều hành có thể cung cấp hỗ trợ cho chính các luồng. Trong trường hợp này, các luồng có thể được triển khai dưới dạng các quy trình riêng biệt chia sẻ một không gian địa chỉ chung hoàn chỉnh hoặc một phần, hoặc chúng có thể được triển khai dưới dạng các ngữ cảnh thực thi riêng biệt trong một quy trình duy nhất. Cho dù các luồng được tổ chức theo cách nào, chúng sẽ xuất hiện dưới dạng bối cảnh thực thi hoàn toàn độc lập với ứng dụng.

Việc triển khai kết hợp cũng có thể thực hiện được, trong đó một số lượng lớn các luồng được tạo sẵn cho ứng dụng bằng cách sử dụng một số lượng nhỏ hơn các luồng hạt nhân. Các luồng người dùng có thể chạy được chạy bởi luồng nhân có sẵn đầu tiên.

Trong Linux, các luồng được thực hiện bên trong hạt nhân bằng cơ chế sao chép tạo ra một quy trình mới trong cùng một không gian địa chỉ ảo với quy trình mẹ. Không giống như một số gói luồng dựa trên nhân, nhân Linux không tạo ra bất kỳ sự phân biệt nào giữa luồng và quy trình: luồng chỉ đơn giản là một quy trình không tạo ra một không gian địa chỉ ảo mới khi nó được khởi tạo.

Ưu điểm chính của việc triển khai các luồng trong nhân thay vì trong thư viện chế độ người dùng là:

- hệ thống luồng nhân có thể tận dụng nhiều bộ xử lý nếu chúng có sẵn; và
- nếu một luồng chặn trong quy trình dịch vụ hạt nhân (ví dụ, một lệnh gọi hệ thống hoặc lỗi trang), các luồng khác vẫn có thể chạy.

Một lợi thế ít hơn là khả năng gán các thuộc tính bảo mật khác nhau cho mỗi luồng.

Việc triển khai chế độ người dùng không có những ưu điểm này. Bởi vì các triển khai như vậy chạy hoàn toàn trong một ngữ cảnh thực thi hạt nhân duy nhất, chỉ một luồng có thể chạy cùng một lúc, ngay cả khi có nhiều CPU. Vì lý do tương tự, nếu một luồng tham gia lệnh gọi hệ thống, không luồng nào khác có thể chạy cho đến khi lệnh gọi hệ thống đó hoàn tất. Do đó, một luồng thực hiện đọc đĩa chặn sẽ giữ mọi luồng trong ứng dụng. Tuy nhiên, việc triển khai chế độ người dùng có những lợi thế riêng. Rõ ràng nhất là hiệu suất: việc gọi bộ lập lịch riêng của nhân để chuyển đổi giữa các luồng liên quan đến việc nhập miền bảo vệ mới khi CPU chuyển sang chế độ nhân, trong khi việc chuyển đổi giữa các luồng trong chế độ người dùng có thể đạt được đơn giản bằng cách lưu và khôi phục các thanh ghi CPU chính. Các luồng chế độ người dùng cũng có thể tiêu tốn ít bộ nhớ hệ thống hơn:

Phương pháp kết hợp, thực hiện nhiều luồng người dùng trên một số lượng nhỏ hơn các luồng nhân, cho phép đạt được sự cân bằng giữa các sự cân bằng này. Các luồng nhân sẽ cho phép nhiều luồng tham gia chặn các cuộc gọi nhân cùng một lúc và sẽ cho phép chạy trên nhiều CPU và chuyển đổi luồng chế độ người dùng có thể xảy ra trong mỗi luồng nhân để thực hiện phân luồng nhẹ mà không cần phải có quá nhiều luồng nhân. Nhược điểm của phương pháp này là sự phức tạp: việc trao quyền kiểm soát sự cân bằng sẽ làm phức tạp giao diện người dùng của thư viện luồng.

21.3 Nhân Linux không cho phép phân trang ngoài bộ nhớ nhân. Hạn chế này có ảnh hưởng gì đến thiết kế của nhân? Hai ưu điểm và hai nhược điểm của quyết định thiết kế này là gì?

Trả lời: Tác động chính của việc không cho phép phân trang bộ nhớ hạt nhân trong Linux là tính không ưu tiên của hạt nhân được bảo toàn. Bất kỳ quá trình nào xảy ra lỗi trang, dù là trong nhân hay ở chế độ người dùng, đều có nguy cơ bị lên lịch lại trong khi dữ liệu được yêu cầu được phân trang từ đĩa. Bởi vì hạt nhân có thể dựa vào việc không được lên lịch lại trong quá trình truy cập vào cấu trúc dữ liệu chính của nó, các yêu cầu khóa để bảo vệ tính toàn vẹn của các cấu trúc dữ liệu đó được đơn giản hóa rất nhiều. Mặc dù bản thân sự đơn giản trong thiết kế là một lợi ích, nhưng nó cũng mang lại một lợi thế hiệu suất quan trọng trên các máy đơn xử lý do thực tế là không cần thực hiện khóa bổ sung trên hầu hết các cấu trúc dữ liệu bên trong.

Tuy nhiên, có một số nhược điểm đối với việc thiếu bộ nhớ nhân có thể phân trang. Trước hết, nó áp đặt các ràng buộc về dung lượng bộ nhớ mà hạt nhân có thể sử dụng. Không hợp lý khi giữ các cấu trúc dữ liệu rất lớn trong bộ nhớ không thể phân trang, vì điều đó đại diện cho bộ nhớ vật lý hoàn toàn không thể được sử dụng cho bất kỳ việc gì khác. Điều này có hai tác động: trước hết, hạt nhân phải cắt bỏ nhiều cấu trúc dữ liệu bên trong của nó theo cách thủ công, thay vì có thể dựa vào một cơ chế bộ nhớ ảo duy nhất để kiểm soát việc sử dụng bộ nhớ vật lý. Thứ hai, nó làm cho việc triển khai các tính năng nhất định yêu cầu lượng lớn bộ nhớ ảo trong nhân, chẳng hạn như / tmp-

hệ thống tệp (một hệ thống tệp dựa trên bộ nhớ ảo nhanh được tìm thấy trên một số hệ thống UNIX).

Lưu ý rằng sự phức tạp của việc quản lý lỗi trang trong khi chạy mã nhân không phải là vấn đề ở đây. Mã nhân Linux đã có thể xử lý các lỗi trang: nó cần có khả năng xử lý các lệnh gọi hệ thống có các đối số tham chiếu bộ nhớ người dùng có thể được phân trang ra đĩa.

21.4 Ba ưu điểm của liên kết động (dùng chung) của các thư viện so với liên kết tĩnh là gì? Hai trường hợp mà liên kết tĩnh được ưu tiên là gì?

Trả lời: Ưu điểm chính của thư viện chia sẻ là chúng giảm bộ nhớ và không gian đĩa mà hệ thống sử dụng, đồng thời tăng cường khả năng bảo trì.

Khi các thư viện được chia sẻ đang được sử dụng bởi tất cả các chương trình đang chạy, chỉ có một phiên bản của mỗi quy trình thư viện hệ thống trên đĩa và nhiều nhất là một phiên bản trong bộ nhớ vật lý. Khi thư viện được đề cập là một trong những thư viện được sử dụng bởi nhiều ứng dụng và chương trình, thì khả năng tiết kiệm đĩa và bộ nhớ có thể khá lớn. Ngoài ra, thời gian khởi động để chạy các chương trình mới có thể được giảm bớt, vì nhiều chức năng phổ biến cần thiết của chương trình đó có thể đã được tải vào bộ nhớ vật lý.

Khả năng bảo trì cũng là một lợi thế chính của liên kết động so với liên kết tĩnh. Nếu tất cả các chương trình đang chạy sử dụng thư viện chia sẻ để truy cập các quy trình thư viện hệ thống của chúng, thì việc nâng cấp các quy trình đó, để thêm chức năng mới hoặc để sửa lỗi, có thể được thực hiện đơn giản bằng cách thay thế thư viện được chia sẻ đó. Không cần biên dịch lại hoặc liên kết lại bất kỳ ứng dụng nào; bất kỳ chương trình nào được tải sau khi nâng cấp hoàn tất sẽ tự động nhận các phiên bản mới của thư viện.

Có những lợi thế khác nữa. Một chương trình sử dụng thư viện chia sẻ thường có thể được điều chỉnh cho các mục đích cụ thể chỉ bằng cách thay thế một hoặc nhiều thư viện của nó, hoặc thậm chí (nếu hệ thống cho phép và hầu hết các UNIX bao gồm cả Linux đều có) thêm một thư viện mới tại thời điểm chạy. Ví dụ, một thư viện gỡ lỗi có thể được thay thế cho một thư viện thông thường để theo dõi sự cố trong ứng dụng. Thư viện được chia sẻ cũng cho phép các tệp nhị phân của chương trình được liên kết với mã thư viện thương mại, độc quyền mà không thực sự đưa bất kỳ mã nào trong số đó vào tệp thực thi cuối cùng của chương trình. Điều này rất quan trọng vì trên hầu hết các hệ thống UNIX, nhiều thư viện chia sẻ tiêu chuẩn là độc quyền và các vấn đề cấp phép có thể ngăn cản việc phân phối mã đó trong các tệp thực thi cho các bên thứ ba.

Tuy nhiên, ở một số nơi, liên kết tĩnh là thích hợp. Một ví dụ là trong môi trường cứu hộ dành cho quản trị viên hệ thống. Nếu quản trị viên hệ thống mắc lỗi trong khi cài đặt bất kỳ thư viện mới nào hoặc nếu phần cứng phát sinh sự cố, thì rất có thể các thư viện dùng chung hiện có bị hỏng. Do đó, thường một tập hợp các tiện ích cứu hộ cơ bản được liên kết tĩnh, để có cơ hội sửa lỗi mà không cần phải dựa vào các thư viện được chia sẻ hoạt động chính xác.

Ngoài ra còn có những lợi thế về hiệu suất mà đôi khi làm cho liên kết tĩnh được ưu tiên hơn trong những trường hợp đặc biệt. Để bắt đầu, liên kết động làm tăng thời gian khởi động cho một chương trình, vì liên kết bây giờ phải

được thực hiện tại thời điểm chạy hơn là tại thời điểm biên dịch. Liên kết động đôi khi cũng có thể làm tăng kích thước bộ làm việc tối đa của một chương trình (tổng số trang vật lý của bộ nhớ cần thiết để chạy chương trình). Trong thư viện được chia sẻ, người dùng không có quyền kiểm soát vị trí trong tệp nhị phân thư viện mà các hàm khác nhau nằm ở đâu. Vì hầu hết các hàm không điền chính xác toàn bộ trang hoặc các trang của thư viện, nên việc tải một hàm thường sẽ dẫn đến việc tải các phần của các hàm xung quanh. Với liên kết tĩnh, tuyệt đối không có chức năng nào không được ứng dụng tham chiếu (trực tiếp hoặc gián tiếp) cần được tải vào bộ nhớ.

Các vấn đề khác xung quanh liên kết tĩnh bao gồm tính dễ phân phối: việc phân phối tệp thực thi với liên kết tĩnh sẽ dễ dàng hơn so với liên kết động nếu nhà phân phối không chắc liệu người nhận có cài đặt trước các thư viện chính xác hay không. Cũng có thể có các hạn chế thương mại đối với việc phân phối lại một số tệp nhị phân dưới dạng thư viện được chia sẻ. Ví dụ: giấy phép cho môi trường đồ họa UNIX "Motif" cho phép các tệp nhị phân sử dụng Motif được phân phối tự do miễn là chúng được liên kết tĩnh, nhưng các thư viện được chia sẻ có thể không được sử dụng nếu không có giấy phép.

21,5 So sánh việc sử dụng các ổ cắm mạng với việc sử dụng bộ nhớ dùng chung làm cơ chế giao tiếp dữ liệu giữa các quá trình trên một máy tính. Ưu điểm của từng phương pháp là gì? Khi nào mỗi loại có thể được ưu tiên?

Trả lời: Sử dụng ổ cắm mạng thay vì bộ nhớ dùng chung để liên lạc cục bộ có một số lợi thế. Ưu điểm chính là giao diện lập trình socket có một tập hợp các tính năng đồng bộ hóa phong phú. Một quy trình có thể dễ dàng xác định thời điểm dữ liệu mới đến trên kết nối socket, lượng dữ liệu hiện có và ai đã gửi dữ liệu đó. Các quá trình có thể chặn cho đến khi dữ liệu mới đến trên ổ cắm hoặc chúng có thể yêu cầu gửi tín hiệu khi dữ liệu đến. Một ổ cắm cũng quản lý các kết nối riêng biệt. Một quy trình với một ổ cắm mở để nhận có thể chấp nhận nhiều kết nối đến ổ cắm đó và sẽ được thông báo khi các quy trình mới cố gắng kết nối hoặc khi các quy trình cũ ngắt kết nối của chúng.

Bộ nhớ dùng chung không cung cấp các tính năng này. Không có cách nào để một quá trình xác định liệu một quá trình khác đã phân phối hoặc thay đổi dữ liệu trong bộ nhớ dùng chung ngoài việc xem xét nội dung của bộ nhớ đó. Quá trình không thể chặn và yêu cầu đánh thức khi bộ nhớ dùng chung được phân phối và không có cơ chế tiêu chuẩn nào để các quá trình khác thiết lập liên kết bộ nhớ dùng chung với một quá trình hiện có.

Tuy nhiên, bộ nhớ chia sẻ có ưu điểm là nó nhanh hơn rất nhiều so với giao tiếp socket trong nhiều trường hợp. Khi dữ liệu được gửi qua một ổ cắm, nó thường được sao chép từ bộ nhớ sang bộ nhớ nhiều lần. Cập nhật bộ nhớ dùng chung không yêu cầu bản sao dữ liệu: nếu một quá trình cập nhật cấu trúc dữ liệu trong bộ nhớ dùng chung, thì bản cập nhật đó sẽ hiển thị ngay lập tức cho tất cả các quá trình khác chia sẻ bộ nhớ đó. Gửi hoặc nhận dữ liệu qua socket yêu cầu phải thực hiện lệnh gọi dịch vụ hệ thống hạt nhân để bắt đầu truyền, nhưng giao tiếp bộ nhớ dùng chung có thể được thực hiện hoàn toàn ở chế độ người dùng mà không cần chuyển quyền điều khiển.

Giao tiếp socket thường được ưu tiên khi quản lý kết nối là quan trọng hoặc khi có yêu cầu đồng bộ hóa người gửi và người nhận. Ví dụ, các quy trình máy chủ thường sẽ thiết lập một ổ cắm lắng nghe mà máy khách có thể kết nối khi họ muốn sử dụng dịch vụ đó. Sau khi ổ cắm được thiết lập, các yêu cầu riêng lẻ cũng được gửi bằng ổ cắm, do đó máy chủ có thể dễ dàng xác định khi nào một yêu cầu mới đến và nó đến từ ai.

Tuy nhiên, trong một số trường hợp, bộ nhớ dùng chung được ưu tiên hơn. Bộ nhớ dùng chung thường là giải pháp tốt hơn khi phải truyền một lượng lớn dữ liệu hoặc khi hai quy trình cần truy cập ngẫu nhiên vào một tập dữ liệu chung lớn. Tuy nhiên, trong trường hợp này, các quá trình giao tiếp vẫn có thể cần một cơ chế bổ sung ngoài bộ nhớ dùng chung để đạt được sự đồng bộ hóa giữa chúng. Hệ thống Cửa sổ X, một môi trường hiển thị đồ họa cho UNIX, là một ví dụ điển hình về điều này: hầu hết các yêu cầu đồ họa được gửi qua các socket, nhưng bộ nhớ dùng chung được cung cấp như một phương tiện truyền tải bổ sung trong các trường hợp đặc biệt khi các bitmap lớn được hiển thị trên màn hình. Trong trường hợp này, một yêu cầu hiển thị bitmap sẽ vẫn được gửi qua socket, nhưng dữ liệu lớn của bản thân bitmap sẽ được gửi qua bộ nhớ dùng chung.

21,6 Các hệ thống UNIX thường sử dụng tối ưu hóa bố cục đĩa dựa trên vị trí xoay của dữ liệu đĩa, nhưng các triển khai hiện đại, bao gồm cả Linux, chỉ cần tối ưu hóa để truy cập dữ liệu tuần tự. Tại sao họ làm như vậy? Truy cập tuần tự tận dụng những đặc điểm phần cứng nào? Tại sao tối ưu hóa xoay vòng không còn hữu ích nữa?

Trả lời: Các đặc tính hiệu suất của phần cứng đĩa có đã thay đổi đáng kể trong những năm gần đây. Đặc biệt, nhiều cải tiến đã được giới thiệu để tăng băng thông tối đa có thể đạt được trên đĩa. Trong một hệ thống hiện đại, có thể có một đường ống dài giữa hệ điều hành và đầu đọc-ghi của đĩa. Một yêu cầu I / O đĩa phải chuyển qua bộ điều khiển đĩa cục bộ của máy tính, qua logic bus đến chính ổ đĩa, sau đó vào bên trong đĩa, nơi có khả năng có một bộ điều khiển phức tạp có thể truy cập dữ liệu vào bộ đệm và có khả năng tối ưu hóa thứ tự của các yêu cầu I / O.

Do sự phức tạp này, thời gian cần thiết để một yêu cầu I / O được thừa nhận và yêu cầu tiếp theo được tạo và nhận bởi đĩa có thể vượt xa khoảng thời gian giữa một khu vực đĩa đi qua đầu đọc-ghi và đến tiêu đề khu vực tiếp theo. Để có thể đọc nhiều sector cùng một lúc một cách hiệu quả, các đĩa sẽ sử dụng một bộ nhớ đệm trên đầu đọc. Trong khi một sector đang được chuyển trở lại máy tính chủ, đĩa sẽ bận đọc các sector tiếp theo đề phòng yêu cầu đọc chúng. Nếu các yêu cầu đọc bắt đầu đến theo thứ tự phá vỡ đường dẫn đầu đọc này, hiệu suất sẽ giảm. Do đó, hiệu suất mang lại lợi ích đáng kể nếu hệ điều hành cố gắng giữ các yêu cầu I / O theo thứ tự tuần tự nghiêm ngặt.

Đặc điểm thứ hai của đĩa hiện đại là hình học của chúng có thể rất phức tạp. Số lượng cung trên mỗi hình trụ có thể thay đổi tùy theo vị trí của hình trụ: nhiều dữ liệu có thể được đưa vào các rãnh dài gần mép đĩa hơn là ở tâm đĩa. Đối với một hệ điều hành để tối ưu hóa vị trí luân chuyển của dữ liệu trên

đĩa, nó sẽ phải có hiểu biết đầy đủ về hình dạng này, cũng như các đặc điểm thời gian của đĩa và bộ điều khiển của nó. Nói chung, chỉ logic bên trong của đĩa mới có thể xác định lịch trình tối ưu của I / Os và hình dạng của đĩa có khả năng đánh bại bất kỳ nỗ lực nào của hệ điều hành để thực hiện tối ưu hóa quay.