Nguyễn Hoàng Tài Toàn (Slides 1 -> 15)

* Chào mọi người
* Giới thiệu
  + Nhóm
  + Nhiệm vụ của nhóm
  + Nhiệm vụ của bản thân
* Tổng quát qua về Tiến Trình ( giải thích sau về ý CPU chạy được nhiều tiến trình)
* Nói về ba mô hình của tiến trình với ba góc nhìn khác nhau và giải thích mô hình B ( CPU chạy được nhiều tiến trình là dưới góc nhìn của người dùng )
* Nói về bộ điều phối(một số nguồn dịch là bộ lập lịch) và đưa ra ví dụ trực quan (công an điều phối tắc đường )
* 5 trạng thái của tiến trình ( nói xơ qua và tập chung vào bắt đầu và kết thúc )
* Liên hệ thực tế rằng mọi quá trình 2 trạng thái cần thiết đó là Bắt đầu và Kết thúc
* Nêu các trường hợp để tạo nên một quá trình
  + Nói về phân cấp tiến trình
* Giải thích sơ đồ về các trạng thái của tiến trình
* Nêu các trường hợp mà tiến trình kết thúc
* Biểu thị thông tin của tiến trình thì nó sẽ có một khối điều khiển thiến trình ( giải thích các thành phần bên trong nó )
* Giải thích về sơ đồ biểu thị việc chuyển từ tiến trình này sang tiến trình khác ( láy ví dụ là (p0 và p1)

Hồ Phương Đồng (Slides 16 -> 31)

Single and multithreaded processes:

* Phần lớn các phần mềm trong máy tính hiện đại đều có dạng multithreaded, tức đa luồng. Các ứng dụng trong máy tính đa phần đều chạy một process nhất định cùng với đó là nhiều luồng chạy bên trong.
* Các ứng dụng cũng có thể được thiết kế để tận dụng khả năng xử lý trên các hệ thống multicore, giúp thực hiện nhiều CPU task song song.
* Minh hoạ dễ hiểu cho tính ưu việt của sự đa luồng là trình xử lý Word có thể vừa in tài liệu sử dụng một thread nền, vừa cùng lúc chạy một thread khác nhận dữ liệu vào từ người dùng để gõ một văn bản mới.
* multithreaded cũng được sử dụng rộng rãi trong nhân hệ hiều hành (operating system kernels). Có nhiều luồng hoạt động trong một kernel, và mỗi luồng đảm nhiệm một công việc riêng biệt, như quản lý thiết bị, quản lý bộ nhớ, quản lý ngắt…

Slide 18: Một quy trình:

- Không gian địa chỉ (phần văn bản, phần dữ liệu)

- Chuỗi thực thi đơn

•Bộ đếm chương trình

•Các thanh ghi

•Stack

- Tài nguyên khác (các tệp mở, quy trình con)

Slide 19: Đa luồng là Nhiều luồng thực thi trong cùng một môi trường của quá trình

- Không gian địa chỉ (phần văn bản, phần dữ liệu)

- Việc thực thi của nhiều luồng, mỗi luồng có bộ riêng

• Bộ đếm chương trình

• Các thanh ghi

• Stack

- Tài nguyên khác (các tệp mở, quy trình con)

Slide 20: Sự khác nhau giữa luồng đơn và đa luồng: Một luồng đề cập đến việc thực thi toàn bộ quy trình từ đầu đến cuối mà không bị gián đoạn bởi một luồng trong khi đa luồng đề cập đến việc cho phép nhiều luồng trong một quy trình để chúng được thực thi độc lập trong khi chia sẻ tài nguyên của chúng.

Slide 21: bên phải là mục riêng tư, ứng với các luồng, bên trái là mục được chia sẻ bởi các luồng trong 1 quá trình

Slide 23: Lợi ích mà luồng mang lại

* 1. Lợi ích của Thread:

- Multithread giúp các ứng dụng tương tác có thể hoạt động tốt hơn vì ngay cả khi một phần chương trình bị block hoặc cần một thời gian dài để hoạt động, chương trình nhìn chung vẫn có thể chạy. Bạn có thể hình dung: khi một chương trình đang chạy, và người dùng nhấn vào một nút lệnh mà cần rất nhiều thời gian để process, thì một hệ thống dạng single-threaded sẽ không kích hoạt bất kì hoạt động nào khác cho tới khi hoàn thành bước lệnh vừa rồi. Ngược lại, ứng dụng dạng multithread sẽ không làm gián đoạn quá nhiều quá trình của người dùng vì trong khi một thread được kích hoạt để thực hiện bước lệnh kia, một thread khác sẽ được kích hoạt để thực hiện bất kì bước lệnh ít tốn thời gian hơn mà người dùng yêu cầu

- Các tiến trình chỉ có thể chia sẻ dữ liệu thông qua các kĩ thuật như shared memory (vùng bộ nhớ chung) và message sharing (chia sẻ tin)

- Việc cung cấp tài nguyên và dữ liệu cho quá trình tạo process rất tốn kém. Như trong Solaris, tạo ra một process lâu hơn 30 lần so với tạo ra một thread trong process đó, và lâu hơn 5 lần so với tạo một context-switch.

- Multithread giúp các threads hoạt động song song trong các lõi xử lý khác nhau, trong khi đối với tiến trình dạng single-threaded, một thread chỉ có thể chạy trên một bộ xử lý, không quan trọng việc có bao nhiêu thread trong hệ thống hiện tại.

Slide 24 + 25 + 26: Các ví dụ về luồng:

Slide 24: 1 trình xử lí văn bản với 3 luồng: Nhiều bộ xử lý văn bản có tính năng tự động lưu toàn bộ tệp vào đĩa vài phút một lần để bảo vệ người dùng không bị mất công việc trong ngày trong trường hợp chương trình bị lỗi, hệ thống bị treo hoặc mất điện. Luồng đầu tiên chỉ tương tác với người dùng. Luồng thứ hai định dạng lại tài liệu khi được yêu cầu. Luồng thứ ba ghi nội dung của RAM vào đĩa theo định kỳ. (Many word processors have a feature of automatically saving the entire file to disk every few minutes to protect the user against losing a day's work in the event of a program crash, system crash, or power failure. The first thread just interacts with the user. The second thread reformats the document when told to. The third thread writes the contents of RAM to disk periodically.)

Slide 25: 1 máy chủ Web đa luồng: (This model allows the server to be written as a collection of sequential threads. The dispatcher's program consists of an infinite loop for getting a work request and handing it off to a worker. Each worker's code consists of an infinite loop consisting of accepting a request from the dispatcher and checking the Web cache to see if the page is present. If so, it is returned to the client and the worker blocks waiting for a new request. If not, it gets the page from the disk, returns it to the client, and blocks waiting for a new request.) Mô hình này cho phép máy chủ được viết như một tập hợp các luồng tuần tự. Chương trình của người điều phối bao gồm một vòng lặp vô hạn để nhận yêu cầu công việc và giao nó cho nhân viên. Mỗi mã của công nhân bao gồm một vòng lặp vô hạn bao gồm chấp nhận yêu cầu từ người điều phối và kiểm tra bộ nhớ cache của Web để xem trang đó có xuất hiện hay không. Nếu vậy, nó sẽ được trả lại cho máy khách và khối công nhân đang chờ một yêu cầu mới. Nếu không, nó sẽ lấy trang từ đĩa, trả lại cho máy khách và chặn để chờ một yêu cầu mới.

Slide 26: Phác thảo sơ bộ về mã cho slide trước :

1. Luồng gửi đi
2. Luồng làm việc

Vòng lặp chính của máy chủ Web nhận một yêu cầu, kiểm tra nó và thực hiện nó để hoàn thành trước khi nhận được yêu cầu tiếp theo. Trong khi chờ đĩa, máy chủ không hoạt động và không xử lý bất kỳ yêu cầu gửi đến nào khác. Nếu máy chủ Web đang chạy trên một máy chuyên dụng, như thường lệ, CPU chỉ đơn giản là không hoạt động trong khi máy chủ Web đang đợi đĩa. Kết quả thực là nhiều yêu cầu / giây ít hơn có thể được xử lý. Do đó, các luồng đạt được hiệu suất đáng kể, nhưng mỗi luồng được lập trình tuần tự, theo cách thông thường.(  The main loop of the Web server gets a request, examines it, and carries it out to completion before getting the next one. While waiting for the disk, the server is idle and does not process any other incoming requests. If the Web server is running on a dedicated machine, as is commonly the case, the CPU is simply idle while the Web server is waiting for the disk. The net result is that many fewer requests/sec can be processed. Thus threads gain considerable performance, but each thread is programmed sequentially, in the usual way.)

Slide 27:

Gói luồng mức người dùng: Loại luồng này được tải hoàn toàn trong không gian người dùng, kernel không biết gì về chúng. Khi luồng được quản lý trong không gian người dùng, mỗi tiến trình cần bảng luồng riêng của nó, bảng luồng bao gồm bộ đếm chương trình, con trỏ ngăn xếp, thanh ghi, trạng thái, v.v. Bảng luồng được quản lý bởi hệ thống thời gian chạy. Tạo một luồng mới, chuyển đổi giữa các luồng và đồng bộ hóa các luồng được thực hiện thông qua lệnh gọi thủ tục. tức là không có sự tham gia của nhân. Chủ đề Cấp độ người dùng nhanh hơn hàng trăm lần so với các chủ đề Cấp độ hạt nhân

Slide 28: Thư viện luồng, (hệ thống thời gian chạy) trong không gian người dùng

\* Luồng tạo

\* Luồng có sẵn

\* Luồng đệm

\* Vùng luồng (để tự nguyện từ bỏ CPU)  
Khối điều khiển luồng (TCB) (Bảng luồng) lưu trữ trạng thái của luồng người dùng (bộ đếm chương trình, thanh ghi, ngăn xếp)

* + - Kernel không biết sự hiện diện của luồng người dùng

Slide 29: Hệ điều hành truyền thống chỉ cung cấp một luồng nhân kernel mà được trình bày bởi PCB cho mỗi quy trình.

- Sự cố chặn: Nếu một luồng người dùng bị chặn -> luồng nhân bị chặn, -> tất cả các luồng khác trong quá trình đều bị chặn.

Slide 30: Trong luồng mức lõi, lõi thực hiện toàn bộ công việc chuyển động của luồng. Kernel có một bảng luồng theo dõi tất cả các luồng trong hệ thống. Khi một luồng muốn tạo một luồng mới hoặc phá hủy bất kỳ luồng nào hiện có, nó sẽ thực hiện một cuộc gọi Kernel, thực hiện hành động này.

Slide 31:

* Đa luồng được hệ điều hành hỗ trợ trực tiếp:

- Kernel quản lý các tiến trình và luồng

- Lập lịch CPU cho luồng được thực hiện trong kernel

Lợi thế của đa luồng trong kernel:

* Kernel có thể lên lịch đồng thời cho nhiều luồng từ cùng một quá trình trên nhiều quá trình.
* Nếu một luồng trong một tiến trình bị chặn, Kernel có thể lên lịch cho một luồng khác của cùng một tiến trình.
* Bản thân các quy trình hạt nhân có thể được đa luồng.

Nhược điểm của đa luồng trong kernel:

* Các luồng nhân thường chậm tạo và quản lý hơn các luồng người dùng.
* Việc chuyển quyền điều khiển từ luồng này sang luồng khác trong cùng một quy trình yêu cầu chuyển chế độ sang Kernel

Slide 32: Nhiều cách khác nhau đã được nghiên cứu để cố gắng kết hợp những ưu điểm của luồng cấp người dùng với luồng cấp nhân. Một cách là sử dụng các luồng cấp nhân và sau đó ghép các luồng cấp người dùng vào một số hoặc tất cả các luồng nhân

Trong thiết kế này, hạt nhân chỉ biết đến các luồng cấp hạt nhân và lên lịch cho các luồng đó. Một số luồng đó có thể có nhiều luồng cấp người dùng được ghép chồng lên trên chúng. Các luồng cấp người dùng này được tạo, hủy và lập lịch giống như các luồng cấp người dùng trong một quy trình chạy trên hệ điều hành mà không có khả năng đa luồng. Trong mô hình này, mỗi luồng cấp nhân có một số luồng cấp người dùng thay phiên nhau sử dụng nó.

Phạm Minh Giang (Slides 32 -> 55)

**Lập lịch trình**

*Giới thiệu về lập lịch trình*

* Sử dụng CPU tối đa có được với đa chương trình
* Chu kì bùng nổ I/O của CPU - Quá trình thực thi bao gồm chu kì thực thi CPU và chờ I/O
* Phân phối bùng nổ CPU
* Sự bùng nổ của việc sử dụng CPU xen kẽ với khoảng thời gian chờ đợi I / O
* **Quy trình ràng buộc CPU:** có nghĩa là tốc độ tiến trình của quá trình bị giới hạn bởi tốc độ của CPU.
* **Quy trình ràng buộc I/O:** nghĩa là tốc độ tiến trình của một quá trình bị giới hạn bởi tốc độ của hệ thống con I / O.

* **Lập lịch ba cấp**: chia quá trình [lập lịch](https://de.wikipedia.org/wiki/Scheduling) thành 3 quá trình phụ:
* Bộ lập lịch nhập học
* Bộ lập lịch bộ nhớ
* Bộ lập lịch CPU

Khi bắt đầu quá trình lập lịch trình này, có một [hàng đợi đầu vào](https://de.wikipedia.org/wiki/Warteschlange_(Datenstruktur)) . Đây chỉ đơn giản là một cấu trúc dữ liệu tương tự như một [danh sách](https://de.wikipedia.org/wiki/Datenstruktur#Grundlegende_Datenstrukturen) trong đó các “công việc” mới đến được lưu trữ tạm thời.  Bộ lập *lịch tuyển đầu vào* đã có hiệu lực. nó quyết định công việc nào đi vào hệ thống xa hơn (chính xác hơn là vào bộ nhớ). Các công việc sau đó được tải vào bộ nhớ [chính](https://de.wikipedia.org/wiki/Arbeitsspeicher) bởi bộ lập lịch nhập. Đây là nơi công việc của *bộ lập lịch bộ nhớ* bắt đầu. Vì mọi công việc đều yêu cầu dung lượng trong bộ nhớ và có thể xảy ra một số công việc trong bộ nhớ, bộ lập lịch bộ nhớ sẽ quyết định công việc nào còn lại trong bộ nhớ chính và công việc nào được hoán đổi trên đĩa cứng. Bộ lập *lịch CPU* bây giờ nhận một công việc từ bộ nhớ và xử lý nó

* Lựa chọn trong số các quy trình trong bộ nhớ đã sẵn sàng thực thi và phân bổ CPU cho một trong số chúng
* Các quyết định lập lịch trình CPU có thể diễn ra khi một quá trình:

1. Chuyển từ trạng thái đang chạy sang trạng thái chờ

2. Chuyển từ trạng thái đang chạy sang trạng thái sẵn sàng

3. Chuyển từ trạng thái chờ đợi hoặc quy trình mới được tạo để sẵn sàng

4. Chấm dứt

* Thuật toán lập lịch trình không ưu tiên: chọn quy trình và để nó chạy cho đến khi nó chặn hoặc cho đến khi nó tự nguyện giải phóng CPU
* Thuật toán lập lịch ưu tiên: chọn quy trình và để nó chạy trong thời gian khắc phục tối đa

**Trạng thái của Process**

Khi 1 process chạy, nó thay đổi trạng thái. Trạng thái của process được định nghĩa là 1 phần của hoạt động hiện tại. Process sẽ có 1 trong những trạng thái sau:

1. **New** - Process được khởi tạo
2. **Running** - các đoạn code được chạy
3. **Waiting**- Process đang đợi các sự kiện như I/O...
4. **Ready** - Process đang đợi để được gán vào processor
5. **Terminated** - Process kết thúc

Trạng thái của tiến trình tại một thời điểm được xác định bởi hoạt động hiện thời của tiến trình tại thời điếm đó.

Trong quá trình sống một tiến trình thay đổi trạng thái do nhiều nguyên nhân như: phải chờ một sự kiện nào đó xảy ra ,hay đợi một thao tác nhập xuất hoàn tất, buộc phải dừng hoạt động do hết thời gian xử lý……

*Tiêu chí lập lịch trình*

**Sử dụng CPU** - giữ cho CPU bận rộn nhất có thể

**Thông lượng** - # quy trình hoàn thành việc thực thi chúng trên mỗi đơn vị thời gian

**Thời gian quay vòng** - lượng thời gian để thực hiện một quy trình cụ thể

**Thời gian chờ** - khoảng thời gian mà một quá trình đã được chờ đợi trong hàng đợi sẵn sàng

**Thời gian phản hồi** - lượng thời gian cần từ khi gửi yêu cầu cho đến khi phản hồi đầu tiên được tạo, không phải đầu ra (đối với môi trường chia sẻ thời gian)

**Tiêu chí tối ưu hóa:**

Sử dụng CPU tối đa

Thông lượng tối đa

Thời gian quay vòng tối thiểu

Thời gian chờ tối thiểu

Thời gian phản hồi tối thiểu

*Mục tiêu của thuật toán lập lịch*

**Tất cả các hệ thống**

**Công bằng** - chia sẻ công bằng cho mỗi quá trình của CPU

**Thực thi chính sách** - thấy rằng chính sách đã nêu được thực hiện

**Cân bằng** - giữ cho tất cả các bộ phận của hệ thống bận rộn

**Hệ thống hàng loạt**

**Thông lượng** - tối đa hóa công việc mỗi giờ

**Thời gian quay vòng** - giảm thiểu thời gian từ khi gửi đến khi kết thúc

**Sử dụng CPU** - giữ cho CPU luôn bận rộn

**Hệ thống tương tác**

**Thời gian phản hồi** - phản hồi yêu cầu nhanh chóng

**Tương xứng** - đáp ứng mong đợi của người dùng

**Hệ thống thời gian thực**

**Thời hạn họp** - tránh mất dữ liệu

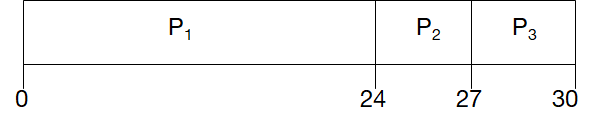
**Khả năng dự đoán** - tránh suy giảm chất lượng trong các hệ thống đa phương tiện

*Lập lịch trình trong hệ thống hàng loạt*

**Lập lịch cho người đến trước, người phục vụ trước (FCFS)**

Giả sử rằng các quá trình đến theo thứ tự: P1, P2, P3 Biểu đồ Gantt cho lịch trình là:





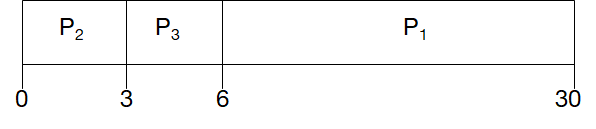
Thời gian chờ P1 = 0; P2 = 24; P3 = 27

Thời gian chờ trung bình: (0 + 24 + 27) / 3 = 17

Giả sử rằng các quy trình đến theo thứ tự

P2, P3, P1

Biểu đồ Gantt cho lịch trình là:



Thời gian chờ P1 = 6; P2 = 0; P3 = 3

Thời gian chờ trung bình: (6 + 0 + 3) / 3 = 3

Tốt hơn nhiều so với trường hợp trước

Hiệu ứng đoàn xe quá trình ngắn đằng sau quá trình dài

**Lập lịch trình công việc ngắn nhất (SJF)**

\* Liên kết với mỗi quá trình độ dài của lần bùng nổ CPU tiếp theo của nó. Sử dụng các độ dài này để lập  lịch cho quá trình với thời gian ngắn nhất

\* Hai phưong án:

- **Không ưu tiên** - một khi CPU được cung cấp cho quá trình, nó không thể được ưu tiên cho đến khi hoàn thành đợt bùng nổ CPU của nó

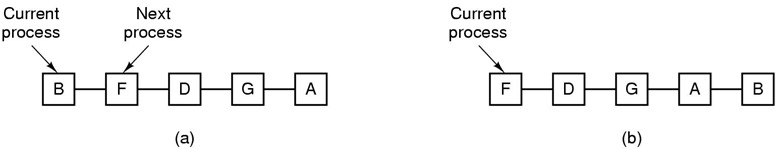
- **Ưu tiên** - nếu một quá trình mới đến với thời lượng bùng nổ CPU ít hơn thời gian còn lại của quy trình thực thi hiện tại. Lược đồ này gọi là thời gian đầu tiên còn lại ngắn nhất (SRTF)

\* SJF là tối ưu - cung cấp thời gian chờ trung bình tối thiểu cho một tập hợp các quy trình nhất định



Một ví dụ về lập lịch trình đầu tiên cho công việc ngắn nhất

*Lập lịch trong Hệ thống tương tác*



Lập lịch vòng quay

danh sách các quy trình có thể chạy được (a)

danh sách các quy trình có thể chạy sau khi B sử dụng hết lượng tử của nó (b)

**Vòng quay (RR)`**

Mỗi quá trình nhận được một đơn vị thời gian CPU nhỏ (lượng tử thời gian), thường là 10 - 100 mili giây. Sau khi thời gian này trôi qua, quy trình được ưu tiên và thêm vào cuối hàng đợi sẵn sàng.

Nếu có n quy trình trong hàng đợi sẵn sàng và lượng tử thời gian là q, thì mỗi quy trình sẽ nhận được 1 / n thời gian CPU với nhiều nhất là q đơn vị thời gian cùng một lúc. Không có quá trình nào chờ nhiều hơn (n-1) q đơn vị thời gian.

Hiệu suất

*q lớn ⇒ FIFO*

*q nhỏ ⇒ q phải lớn đối với chuyển đổi ngữ cảnh, nếu không thì chi phí quá cao*

Ví dụ về RR với Lượng tử thời gian = 20

Quá trình Thời gian bùng nổ

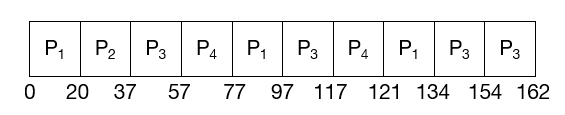
P1 53

P2 17

P3 68

P4 24

Biểu đồ Gantt là:



Thông thường, quay vòng trung bình cao hơn SJF, nhưng phản hồi tốt hơn

**Lập lịch ưu tiên :** Số ưu tiên (số nguyên) được liên kết với mỗi quy trình  -  CPU được phân bổ cho quá trình với mức ưu tiên cao nhất

Ưu tiên

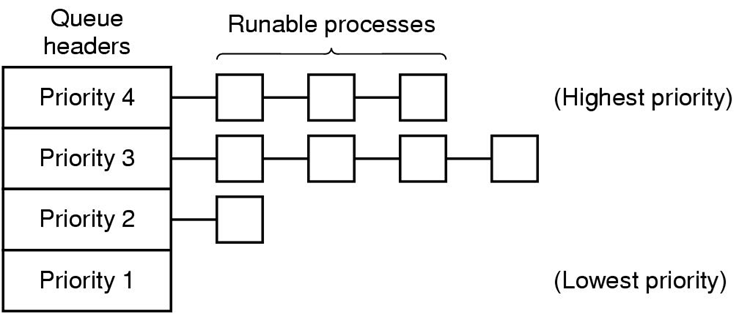
không ưu tiên

SJF là lập lịch ưu tiên trong đó ưu tiên là thời gian bùng nổ CPU tiếp theo được dự đoán

**Vấn đề Đói** - quá trình ưu tiên thấp có thể không bao giờ thực hiện

**Giải pháp tuổi** - theo thời gian tiến triển làm tăng mức độ ưu tiên của quy trình

Một thuật toán lập lịch với bốn lớp ưu tiên



**Lập lịch trong Hệ thống thời gian thực**

**Hệ thống thời gian thực cứng** - cần thiết để hoàn thành một nhiệm vụ quan trọng trong một khoảng thời gian được bảo đảm

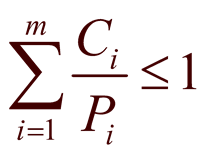
**Điện toán thời gian thực mềm** - yêu cầu các quy trình quan trọng nhận được ưu tiên hơn so với các quy trình kém may mắn hơn

**Hệ thống thời gian thực có thể lập lịch**

m sự kiện định kỳ

sự kiện i xảy ra trong khoảng thời gian Pi và yêu cầu Ci giây

Sau đó, tải chỉ có thể được xử lý nếu:



**Chính sách so với Cơ chế**

Phân tách những gì được phép thực hiện với cách nó được thực hiện

- Một quá trình biết chủ đề con nào là quan trọng và cần ưu tiên

Thuật toán lập lịch tham số

- Cơ chế trong kernel

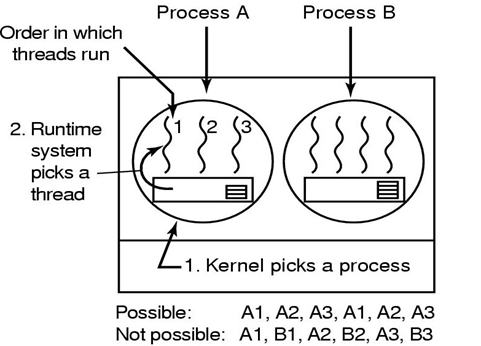
Các tham số được điền bởi các quy trình người dùng

- Chính sách được thiết lập bởi quy trình người dùng

*Lập lịch chuỗi*

**Lập lịch cục bộ** - cách thư viện chủ đề quyết định chủ đề nào sẽ được đưa vào

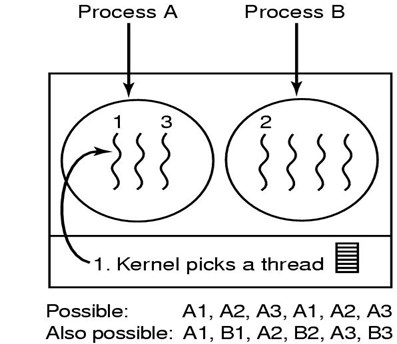
**Lập lịch toàn cục** - Cách kernel quyết định luồng kernel nào sẽ chạy tiếp theo



**Lập lịch có thể cho các chủ đề cấp người dùng**

Quy trình lượng tử 50 msec

luồng chạy 5 msec / CPU bùng nổ



**Lập lịch có thể của các luồng cấp hạt nhân**

\* Lượng tử quá trình 50-msec

\* Luồng chạy 5 msec / CPU bùng nổ

Nguyễn Đức Thịnh (Slides 56 -> 69)

Slide 57:

Một process (tiến trình) trong hệ điều hành có thể được tiến hành độc lập hoặc giao tiếp với nhau. Tiến trình độc lập là khi tiến trình không ảnh hưởng hoặc bị ảnh hưởng bởi các tiến trình khác trong hệ thống, và không chia sẻ data với bất kì tiến trình nào. Tiến trình hợp tác khi tiến trình đó có thể ảnh hưởng hoặc bị ảnh hưởng bởi các tiến trình khác trong hệ thống, và sự chia sẻ data có diễn ra.

Lợi ích của tiến trình hợp tác là:

+ Giúp chia sẻ thông tin giữa các users.

+ Giúp tang tốc độ tính toán

+ Giúp xây dựng modun.

+ Giúp thuận tiện trong chạy nhiều tác vụ cùng một lúc.

Slide 58:

Vấn đề về dữ liệu được chia sẻ:

+ Việc các tiến trình truy cập đồng thời vào dữ liệu được chia sẻ có thể dẫn đến sự không nhất quán của dữ liệu

+ Để duy trì tính nhất quán của dữ liệu đòi hỏi các cơ chế để đảm bảo thực hiện có trật tự các tiến trình hợp tác.

+ Cần có cơ chế để các tiến trình giao tiếp với nhau và đồng bộ hóa các hành động của chúng

Slide 59:

Trong tiến trình hợp tác, việc chia sẻ dữ liệu cho nhau là rất dễ dàng, nhưng từ đó nó cũng phát sinh nhiều vấn đề liên quan. Một trong số vấn đề đó là điều kiện cuộc đua,

Khi 2 tiến trình hoặc nhiều tiến trình muốn truy cập đồng thời vào bộ nhớ chia sẻ và kết quả cuối cùng phụ thuộc vào tiến trình nào chạy chính xác, khi đó được goi là điều kiện cuộc đua.

Giải thích hình ảnh:

Ta có một thư mục bộ đệm với nhiều khe được đánh số, mỗi khe có khả năng chứa 1 tên tệp. Có 2 biến được chia sẻ là biến out và biến in trong đó biến in sẽ trỏ tới vị trí trống tiếp theo trong thư mục. Tại một thời điểm, các khe từ 0 -> 3 là các khe trống, các khe từ 4 - > 6 là các khe đã đầy ( Với tên các tệp xếp hang đợi để in ). Khi đó có 2 tiến trình A và B sẽ quyết đinh tệp nào sẽ được in.

Quy trình A đọc biến in và lưu trữ giá trị trong một biến cục bộ gọi là vị trí trống tiếp theo.Sau đó hiện tượng ngắt xảy ra, CPU quyết định tiến trình A sẽ dừng lại và chuyển sang tiến trình B. Tiến trình B cũng đọc biến in và cũng nhận được 7, nó cũng lưu trữ tại vị trí giống tiến trình A. Tại đây cả 2 tiến trình đều nghĩ khe khả dụng tiếp theo nằm ở vị trí số 7. Tiến trình B tiếp tục chạy và lưu tên tệp của nó trong khe số 7 và cập nhật biến in thành 8. Cuối cùng, tiến trình A chạy lại, bắt đầu từ vị trí mà nó đã dừng lại. Nó tìm vị khe số 7 và viết tên tệp của nó vào khe 7, xóa tên tệp mà tiến trình B vừa mới đặt ở đó. Như vậy tiến trình B không nhận được bất kì đầu ra nào.

Làm thế nào để ta có thể tránh được điều kiện cuộc đua? Đó là sử dụng giái pháp loại trừ lẫn nhau nghĩa là một số cách để ngăn cản nhiều tiến trình truy cập vào dữ liệu được chia sẻ cùng một lúc, đảm bảo rằng nếu một tiến trình đang sử dụng một biến hoặc tệp được chia sẻ, các tiến trình khác sẽ bị loại trừ khỏi việc thực hiện cùng một việc.

Slide 60:

Một phần thời gian, một tiến trình bận rộn thực hiện các tính toán nội bộ và những thứ khác không dẫn đến điều kiện cuộc đua. Tuy nhiên, đôi khi một tiến trình phải truy cập vào bộ nhớ hoặc tệp được chia sẻ hoặc thực hiện những việc quan trọng khác có thể dẫn đến điều kiện cuộc đua. Phần chương trình mà bộ nhớ dùng chung được truy cập được gọi là vùng trọng yếu hoặc vùng quan trọng. Nếu chúng ta có thể sắp xếp các vấn đề sao cho không có hai tiến trình nào ở trong các khu vực quan trọng của chúng cùng một lúc, chúng ta có thể tránh được các điều kiện cuộc đua.

Mặc dù yêu cầu này tránh được các điều kiện cuộc đua, nhưng nó không đủ để các quá trình song song hợp tác một cách chính xác và hiệu quả bằng cách sử dụng dữ liệu được chia sẻ. Chúng ta cần có bốn điều kiện để có một giải pháp tốt:

1. Không có hai tiến trình nào có thể đồng thời bên trong các vùng quan trọng của chúng.
2. Không có giả định nào được đưa ra về tốc độ hoặc số lượng CPU.
3. Không có tiến trình nào đang chạy bên ngoài vùng quan trọng của nó có thể chặn các tiến trình khác.
4. Không có quá trình nào phải đợi mãi để đi vào vùng quan trọng của nó.

Slide 61:

Ví dụ về loại trừ lẫn nhau bằng cách sử dụng các vùng quan trọng

Giải thích hình ảnh:

Tiến trình A đi vào vùng quan trọng của nó tại thời điểm T1. Một lúc sau, tại thời điểm T2, quá trình B cố gắng đi vào vùng quan trọng của nó nhưng không thành công vì một quá trình khác đã nằm trong vùng quan trọng của nó và do chỉ cho phép một quá trình tại một thời điểm. Do đó, B tạm thời sẽ đợi đến khi A đi ra khỏi vùng quan trọng của nó. Đến thời điểm T3, tiến trình A rời khỏi, khi đó B sẽ được phép đi vào tại lúc đó và cuối cùng B đi ra khỏi vùng quan trọng tại thời điểm T4.

Slide 62: Loại trừ lẫn nhau với chờ đợi bận rộn

Trong phần này, chúng ta sẽ xem xét các đề xuất khác nhau để đạt được loại trừ lẫn nhau, để trong khi một quy trình đang bận cập nhật bộ nhớ dùng chung trong vùng quan trọng của nó, thì không có quy trình nào khác sẽ xâm nhập vào vùng quan trọng của nó và gây ra sự cố.

Đầu tiên là các đề xuất liên quan đến phần mềm:

+ Khóa các biến

+ Thay thế nghiêm ngặt

+ Giai pháp của Peterson

Tiếp theo là các đề xuất liên quan đến phần cứng:

+ Vô hiệu hóa các ngắt

+ Mã lệnh TSL

Slide 63:

Trước tiên ta đi tìm hiểu các đề xuất phần mềm, thứ nhất là khóa các biến

Giải thích hình ảnh:

+ Ban đầu đặt lock = 0, Khi một quy trình muốn nhập vào vùng quan trọng của nó, trước tiên nó sẽ kiểm tra lock.

+ Nếu lock là 0, tiến trình đặt nó thành 1 và đi vào vùng quan trọng. Nếu lock đã là 1, tiến trình chỉ đợi cho đến khi lock trở thành 0.

+ Do đó, 0 có nghĩa là không có tiến trình nào nằm trong vùng quan trọng của nó và 1 có nghĩa là một số tiến trình nằm trong vùng quan trọng của nó.

Slide 64:

Tiếp theo là đề xuất phần mềm thứ 2 là thay thế nghiêm ngặt

Giải thích hình ảnh:

+ Có 1 biến số nguyên ban đầu Turn = 0, theo dõi xem tiến trình nào sẽ đến lượt đi vào vùng quan trọng

+ Ban đầu, tiến trình 0 kiểm tra biến Turn, thấy Turn = 0 và đi vào vùng quan trọng của nó, tiến trình 1 cũng kiểm tra biến Turn và nhận thấy Turn = 0, do đó tiến trình 1 sẽ nằm trong một vòng lặp chặt chẽ, liên tục kiểm tra Turn để xem khi nào nó trở thành 1. Việc kiểm tra liên tục một biến cho đến khi một số giá trị xuất hiện được gọi là chờ bận. Một khóa sử dụng chờ bận được gọi là khóa quay.

+ Khi tiến trình 0 rời khỏi vùng quan trọng, Turn sẽ chuyển thành 1, để cho phép tiến trình 1 đi vào vùng quan trọng của nó

+ Nếu tiến trình 0 cố gắng đi vào vùng quan trọng của nó một lần nữa, tiến trình đó sẽ bị chặn vì Turn không còn là 0 nữa.

Slide 65:

Tiếp theo là đề xuất phần mềm thứ 3 là Giải pháp của Peterson

Các tiến trình chia sẻ hai biến chung: turn và increase [2]

Nếu interesse[i] = TRUE có nghĩa là tiến trình Pi muốn vào vùng quan trọng. Khởi đầu, interesse[0]=interesse[1]=FALSE và giá trị của est được khởi động là 0 hay 1. Để có thể vào được vùng quan trọng, trước tiên tiến trình Pi đặt giá trị interesse[i]=TRUE ( xác định rằng tiến trình muốn vào vùng quan trọng), sau đó đặt turn=j (đề nghị thử tiến trình khác vào vùng quan trọng). Nếu tiến trình Pj không quan tâm đến việc vào vùng quan trọng (interesse[j]=FALSE), thì Pi có thể vào vùng quan trọng, nếu không, Pi phải chờ đến khi interesse[j]=FALSE. Khi tiến trình Pi rời khỏi vùng quan trọng, nó đặt lại giá trị cho interesse[i]= FALSE.

Slide 66:

Nhận xét về phương pháp của Peterson :

Thỏa mãn 3 điều kiện:

+ Loại trừ lẫn nhau

Pi có thể nhập vùng nhớ quan trọng khi interes[j] == F hoặc Turn==i

Nếu cả hai đều muốn quay lại, vì Turn chỉ có thể nhận giá trị 0 hoặc 1, vì vậy một quá trình nhập vùng nhớ quan trọng

+ Phát triển

Sử dụng 2 biến khác biệt interes[i] ==> không thể khóa đối lập

Chờ đợi có giới hạn: cả interes[i] và Turn thay đổi

+ Không mở rộng thành N quy trinh

Slice 67:

Bây giờ chúng ta sẽ đi tìm hiểu các đề xuất phần cứng, thứ nhất là Vô hiệu hóa các ngắt.

Có lẽ cách rõ ràng nhất để đạt được loại trừ lẫn nhau là cho phép một tiến trình vô hiệu hóa các ngắt trước khi nó đi vào vùng quan trọng và sau đó cho phép kích hoạt lại các ngắt sau khi nó rời khỏi vùng quan trọng. Bằng cách vô hiệu hóa ngắt, CPU sẽ không thể chuyển đổi các tiến trình. Điều này đảm bảo rằng tiến trình có thể sử dụng biến được chia sẻ mà không cần tiến trình khác truy cập vào nó.

Cách tiếp cận này nói chung là không hấp dẫn vì không khôn ngoan khi cung cấp cho các quy trình của người dùng quyền để tắt các ngắt. Giả sử rằng một trong số họ đã làm điều đó và không bao giờ bật chúng lên nữa? Đó có thể là sự kết thúc của hệ thống.Hơn nữa, nếu hệ thống là một bộ xử lý đa năng, có hai CPU trở lên, thì việc tắt ngắt chỉ ảnh hưởng đến CPU đã thực thi lệnh vô hiệu hóa. Những cái khác sẽ tiếp tục chạy và có thể truy cập bộ nhớ dùng chung.

Slide 68:

Giải pháp phần cứng tiếp theo là Chỉ thị TSL (Test-and-Set Lock)

CPU nguyên thủy hỗ trợ Kiểm tra và Đặt Khóa.

+ Trả về giá trị hiện tại của một biến, đặt biến thành giá trị true.

+ Việc kiểm tra, cập nhật nội dung một vùng nhớ trong một thao tác không thể phân chia

Để sử dụng lệnh TSL, chúng ta sẽ sử dụng một biến chia sẻ, lock, để điều phối quyền truy cập vào bộ nhớ được chia sẻ. Khi lock bằng 0, bất kỳ tiến trình nào cũng có thể đặt nó thành 1 bằng cách sử dụng lệnh TSL và sau đó đọc hoặc ghi bộ nhớ dùng chung. Khi nó được thực hiện, tiến trình đặt lock trở lại 0 bằng cách sử dụng một lệnh di chuyển thông thường.

Slide 69:

Nhận xét cho các giải pháp phần cứng trong Bận chờ

+ Hỗ trợ cơ chế phần cứng cần thiết

Không dễ dàng với hệ thống nhiều CPU

+ Dễ dàng mở rộng đến N quy trình

+ Sử dụng CPU không hiệu quả

Liên tục kiểm tra tình trạng khi chờ vùng quan trọng

+ Quá tải

Khóa các tiến trình không đủ điều kiện để đi kèm trong vùng quan trọng, nhượng bộ CPU cho tiến trình khác

Sử dụng Trình lập lịch biểu

Chờ và xem...

Phạm Thành Long (Slides 70 -> 86)

Slides 70:

Để loại bỏ các bất tiện của giải pháp « busy waiting », chúng ta có thể tiếp cận theo hướng cho một tiến trình chưa đủ điều kiện vào miền găng chuyển sang trạng thái blocked, từ bỏ quyền sử dụng CPU. Để thực hiện điều này, cần phải sử dụng các thủ tục do hệ điều hành cung cấp để thay đổi trạng thái tiến trình. Hai thủ tục cơ bản thường được sử dụng để phục vụ mục đích này là SLEEP và WAKEUP

Sleep là một lời gọi hệ thống có tác dụng tạm dừng hoạt động của tiến trình gọi nó và chờ đến khi được một tiến trình khác đánh thức.

Lời gọi hệ thống Wakeup nhận một tham số duy nhất: tiến trình sẽ được tái kích hoạt (đặt về trạng thái ready) .

Khi sử dụng Sleep và Wakeup, nếu không sử dụng cẩn thận thì sẽ xảy ra tình trạng mâu thuẫn truy xuất trong một vài tình huống đặc biệt

Để tránh những tình huống này, hệ điều hành cung cấp những cơ chế đồng bộ hóa dựa trên ý tưởng của chiến lược « SLEEP and WAKEUP » nhưng được xây dựng bao hàm cả phương tiện kiểm tra điều kiện vào miền găng giúp sử dụng an toàn. Đó là Semaphore (Truyền tin thị giác) và Message passing (Truyền thông điệp)

Slides 71:

Giờ ta sẽ đến với 3 giải pháp “SLEEP & WAKEUP”:

- Từ bỏ CPU khi chưa vào được miền găng

- Quá trình cần được đánh thức đến miền găng khi miền găng trống rỗng

- Và để đánh thức quá trình hay do quá trình thay đổi thì chúng sự hỗ trợ của hệ điều hành

Slides 72:

Còn về ý tưởng sử dụng SLEEP và WAKEUP như sau: Khi một tiến trình chưa đủ điều kiện vào miền găng, nó gọi SLEEP để tự khóa đến khi có một tiến trình khác gọi WAKEUP để giải phóng cho nó. Một tiến trình gọi WAKEUP khi ra khỏi miền găng để đánh thức một tiến trình đang chờ, tạo cơ hội cho tiến trình này vào miền găng.

Miền găng được hỗ trợ bởi 2 cái cơ bản:

- Sleep(): Cuộc gọi hệ thống nhận trạng thái bị chặn

- WakeUp(P): Quy trình P nhận trạng thái sẵn sàng

Slides 73:

Và đây là cấu trúc sử dụng SLEEP & WAKWUP.

Chúng ta sẽ có 2 biến là busy và blocked. Busy = 1 khi miền găng đang bận và bằng 0 khi miền găng rảnh. Còn block là biến đếm số lượng tiến trình đang bị khóa

Ta có quy trình làm việc như sau:

Ban đầu ta sẽ kiểm tra biến busy có bằng 1 hay không? Nếu busy = 0 thì lúc này miền găng đang rảnh thì nó sẽ không thực hiện đoạn block = block + 1.

Thì khi đó biến busy sẽ bị gán = 1 và bắt đầu bay vào miền găng để xử lý. Và khi những tiến trình khác chạy tới sẽ kiểm tra biến busy và thấy nó bằng 1 chứng tỏ nó đang bận, mà khi tiến trình đang bận thì nó sẽ tăng biến block lên 1 và tiến vào trạng thái SLEEP.

Sau khi xử lý xong miền găng thì nó sẽ thoát ra khỏi miền găng. Và xét lại biến busy = 0 để báo miền găng đang rảnh, sau đó nó sẽ kiểm tra coi xem có tiến trình nào đang bị khóa hay không? Nếu có sẽ tiến hành đánh thức tiến trình đó dậy và giảm biến block đi 1. Và nó sẽ nhảy ra để thực thi ngoài miền găng

Slides 74:

Giờ ta có 1 ví dụ:

*Giả sử tiến trình A vào miền găng và trước khi nó rời khỏi miền găng thì tiến trình B được kích hoạt. Tiến trình B thử vào miền găng nhưng nó nhận thấy A đang ở trong đó, do vậy B tăng giá trị biến blocked và chuẩn bị gọi SLEEP để tự khoá. Tuy nhiên trước khi B có thể thực hiện SLEEP, tiến trình A lại được tái kích hoạt và ra khỏi miền găng. Khi ra khỏi miền găng A nhận thấy có một tiến trình đang chờ (blocked = 1) nên gọi WAKEUP và giảm giá trị của blocked. Khi đó tín hiệu WAKEUP sẽ lạc mất do tiến trình B chưa thật sự « ngủ » để nhận tín hiệu đánh thức! Khi tiến trình B được tiếp tục xử lý, nó mới gọi SLEEP và tự khóa vĩnh viễn.*

Do đó lý do hay xảy ra vấn đề là do:

- Việc kiểm tra tư cách vào miền găng và việc gọi SLEEP hay WAKEUP là những hành động tách biệt, có thể bị ngắt nửa chừng trong quá trình xử lý, do đó có khi tín hiệu WAKEUP gửi đến một tiến trình chưa bị khóa sẽ lạc mất

- Do kiểm tra tình trạng và loại bỏ CPU có thể bị hỏng

- Do biến khóa không được bảo vệ.

Slides 75:

Sau đây chúng ta sẽ tìm hiểu về Semaphore:

- Được Dijkstra đề xuất vào 1965, một semaphore *s* là một biến có các thuộc tính sau:

+ Một giá trị nguyên dương e(s) hay được gọi là semaphore s

+ Một hàng đợi f(s) lưu danh sách các tiến trình đang bị khóa (chờ) trên semaphore s

+ Chỉ có hai thao tác được định nghĩa trên semaphore

\*Down

\* Up

+ Các thao tác trong hàm down và up cần thực hiện 1 cách không bị phân chia, không bị ngắt nửa chừng.

Slides 77:

Và đây là cấu trúc của hàm Down và hàm Up:

Đối với hàm Down: thì e(s) sẽ bị giảm đi 1 và sau đó kiểm tra, nếu e(s) < 0 thì P sẽ bị khóa và đưa vào danh sách f(s)

Còn đối với hàm Up: thì e(s) sẽ tăng lên 1 và sau đó kiểm tra, nếu e(s) <= 0 thì chứng tỏ vẫn có tiến trình đang trong hàng chờ. Thì khi đó sẽ đưa 1 tiến trình trong hàng chờ ra và đưa vào trạng thái ready để đi tiếp. Và sau đó lại đưa tiến trình đó vào danh sách ready-list.

Từ đó ta thấy được 2 hàm Down và Up là 2 hàm trái ngược nhau, 1 hàm để khóa 1 tiến trình, 1 hàm để giải phóng 1 tiến trình ra khỏi hàng chờ.

Và ngoài ra giá trị tuyệt của e(s) còn cho ta biết được số tiến trình đang trong hàng chờ semaphore. Nếu trị tuyệt đối của e(s) mà bằng 1 thì có 1 tiến trình đang trong hàng chờ, còn bằng 2 thì có 2 tiến trình đang trong hàng chờ.

Slides 78:

Khi sử dụng Semaphore có thể giải quyết các vấn đề như: vấn đề truy xuất độc quyền hay tổ chức phối hợp giữa các tiến trình

Để thực hiện quá trình đồng bộ hóa, thì tất cả các chương trình cần phải áp dụng cùng cấu trúc chương trình sau đây với e(s) = 1.

Ta sẽ có 2 tiến trình A và B.

Đầu tiên A sẽ nhảy vào hàm Down và e(s) lúc này sẽ bị giảm đi 1 và bằng 0 khi này điều kiện bị sai và A sẽ nhảy luôn vào miền găng.

Sau khi A đi xong thì B với e(s) = 0 đi vào hàm Down, sau khi vào hàm Down thì lúc này e(s) bị giảm đi và bằng -1, và làm cho B bị khóa và tiến vào f(s).

Sau khi B bị khóa thì A thoát khỏi miền gawgn tiếp tục tiế vào hàm Up với e(s) = -1, sau khi đi vào hàm Up e(s) được cộng thêm 1 và bằng 0 đúng với điều kiện e(s) <= 0. Lúc này A sẽ đánh thức B và đặt B vào trạng thái ready và đưa B vào hàm ready-list để đợt sự cấp phát của CPU để thực thi tiếp. Sau khi A chạy xong rồi thì A sẽ tiến vào vùng không tranh chấp.

Sau khi B được sự cấp phát của CPU thì B tiếp tục nhảy vào miền găng để thực thi, sau khi B thực thi xong rồi thì B sẽ nhảy vào Up và tăng e(s) lên 1 và làm điều kiện này sai. Sau đó B sẽ thực thi ngoài miền găng. Xong khi xong hết e(s) lại được khởi tạo lên 1 lại từ đầu.

Đấy là đối với vấn đề truy xuất độc quyền, còn đối với vấn đề tổ chức phối hợp giữa các tiến trình thì sao?

Có tình huống sau: Với semaphore có thể đồng bộ hóa hoạt động của 2 tiến trình trong tình huống một tiến trình phải đợi một tiến trình khác hoàn tất thao tác nào đó mới có thể bắt đầu hay tiếp tục xử lý. Tức là tiến trình A phải chạy xong rồi thì tiến trình B mới chạy được thì lúc này ta có thể sử dụng tổ chức đồng bộ để giải quyết vấn đề này. Để giải quyết vấn đề này chúng ta sẽ sử dụng đồng bộ hóa Semaphore như sau:

Tiến trình 1 sẽ chạy công việc job1, tiến trình 2 sẽ chạy công việc job2.

Và trong trường hợp này e(s) sẽ được gán bằng 0.

Giờ chúng ta sẽ chạy thử job2 trước. Khi tiến trình 2 chạy trước thì sẽ gặp phải hàm down, với e(s) khởi tạo bằng 0 thì khi này nó sẽ = -1 và thỏa mãn điều kiện < 0 và job2 sẽ bị khóa và tiến vào danh sách f(s). Thì lúc này CPU sẽ cấp lại cho tiến trình 1 chạy. Cuối cùng job1 vẫn là tiến trình thực thi trước. Khi tiến trình 1 chạy sẽ gọi hàm Up, lúc này e(s) sẽ tăng lên 1 và = 0 thỏa mãn điều kiện <= 0, và tiến hành giải phóng tiến trình 2 và đưa tiến trình 2 vào ready-list. Sau khi tiến trình 2 thoát ra thì sẽ thực thi được job2.

Với cấu trúc và cách sử dụng như thế này, ta có thể khống chế được việc job1 sẽ chắc chắn luôn luôn được thực thi trước job2

Slides 79:

Để có thể dễ viết đúng các chương trình đồng bộ hóa hơn, Hoare(1974) và Brinch & Hansen (1975) đã đề nghị một cơ chế cao hơn được cung cấp bởi ngôn ngữ lập trình , là monitor.

Monitor là 1 cấu trúc đặc biệt gồm các biến, thủ tục và cấu trúc dữ liệu

Cơ chế đồng bộ được cung cấp bởi ngôn ngữ lập trình

- Hỗ trợ với các chức năng, chẳng hạn như Semaphore

- Dễ dàng sử dụng và phát hiện hơn Semaphore

• Đảm bảo tự động loại trừ lẫn nhau

• Sử dụng biến điều kiện để thực hiện đồng bộ hóa

Slides 80:

Đây là hình cấu trúc cơ bản của Monitor gồm có: biến, thủ tục và cấu trúc dữ liệu

Trong đó hàng đợi monitor, nơi diễn ra các hoạt động chia sẻ dữ liệu của các mã khởi tạo.

Slides 81:

Và đây là tất cả các thao tác trên 1 nhóm tài nguyên gồm có:

Tên monitor; phần khai báo, chia sẻ các biến; mã khởi tạo

Slides 82:

Và tiếp theo đây chúng ta sẽ tìm hiểu về Message Passing (Trao đổi thông điệp)

Giải pháp này dựa trên cơ sở trao đổi thông điệp với hai primitive Send và Receive để thực hiện sự đồng bộ hóa.

• Các quy trình phải đặt tên cho nhau một cách rõ ràng:

– send (P, message) – gửi một thông điệp đến 1 tiến trình hay gửi vào hộp thư

– receive(Q, message) – nhận 1 thông điệp từ một tiến trình hay từ bất kỳ 1 tiến trình nào, tiến trình gọi sẽ chờ nếu không có thông điệp nào để nhận.

• Thuộc tính của liên kết liên lạc

– Các liên kết được thiết lập tự động

– Một liên kết được liên kết với chính xác một cặp quy trình giao tiếp

– Giữa mỗi cặp tồn tại đúng một liên kết

– Liên kết có thể là một chiều, nhưng thường là hai hướng

Slides 84:

Có 3 vấn đề cổ điến trong đồng bộ hóa đó là:

- Vấn đề bộ đệm bị giới hạn hay còn gọi là Vấn đề nhà sản xuất-người tiêu dùng

- Vấn đề của độc giả và nhà văn

- Vấn đề ẩm thực - triết gia

Slides 85: Vấn đề bộ đệm bị giới hạn hay còn gọi là Vấn đề nhà sản xuất-người tiêu dùng

- Vấn đề: Hai tiến trình cùng chia sẻ 1 bộ đệm có kích thước giới hạn. Một trong hai tiến trình đóng vai trò người sản xuất – tạo ra dữ liệu và đặt dữ liệu vào bộ đệm - và tiến trình kia đóng vai trò người tiêu thụ - lấy dữ liệu từ bộ đệm ra để xử lý.

- Giải pháp là Sử dụng ba semaphore : ***full***, đếm số chỗ đã có dữ liệu trong bộ đệm;***empty***, đếm số chỗ còn trống trong bộ đệm; và ***mutex***, kiểm tra việc Producer và Consumer không truy xuất đồng thời đến bộ đệm.

Slides 86: Vấn đề của độc giả và nhà văn

- Vấn đề: Nhiều tiến trình đồng thời sử dụng một cơ sở dữ liệu. Các tiến trình chỉ cần lấy nội dung của cơ sở dữ liệu được gọi là các tiến trình Reader, nhưng một số tiến trình khác lại có nhu cầu sửa đổi, cập nhật dữ liệu trong cơ sở dữ liệu chung này, chúng được gọi là các tiến trình Writer.

- Giải pháp là: Sử dụng 1 biến chung **rc** để ghi nhớ các tiến trình Reader muốn truy xuất cơ sở dữ liệu. Hai semaphore cũng được sử dụng: **mutex,** kiểm soát sự truy cập đến **rc,** và **db**, kiểm tra sự truy xuất độc quyền đến cơ sở dữ liệu

3. Vấn đề thứ 3 là: Vấn đề ẩm thực - triết gia

- Vấn đề: 5 nhà triết học cùng ngồi ăn tối với món mỳ Ý nổi tiếng. Mỗi nhà triết học cần dùng 2 cái nĩa để có thể ăn mỳ Ý . Nhưng trên bàn chỉ có tổng cộng 5 cái nĩa để xen kẽ với 5 cái đĩa. Mỗi nhà triết học sẽ suy ngẫm các triết lý của mình đến khi cảm thấy đói thì dự định lần lượt cầm 1 cái nĩa bên trái và 1 cái nĩa bên phải để ăn. Nếu cả 5 nhà triết học đều cầm cái nĩa bên trái cùng lúc, thì sẽ không có ai có được cái nĩa bên phải để có thể bắt đầu thưởng thức mỳ Ý . Đây chính là tình trạng tắc nghẽn.

- Giải pháp: Có 3 hướng tiếp cận để xử lý:

+ Sử dụng một nghi thức (protocol) để bảo đảm rằng hệ thống không bao giờ xảy ra tắc nghẽn.

+ Cho phép xảy ra tắc nghẽn và tìm cách sữa chữa tắc nghẽn.

+ Hoàn toàn bỏ qua việc xử lý tắc nghẽn, xem như hệ thống không bao giờ xảy ra tắc nghẽn