# Hệ điều hành nâng cao

Mục lục

[Hệ điều hành nâng cao 1](#_Toc132951244)

[Chương 1: Tổng quan 6](#_Toc132951245)

[I.1 Hệ điều hành – Operating System 6](#_Toc132951246)

[I.2 Hệ điều hành làm gì 6](#_Toc132951247)

[I.3 Tổ chức hệ thống máy tính 6](#_Toc132951248)

[I.4 Khởi động máy tính 6](#_Toc132951249)

[I.5 Cấu trúc lưu trữ 6](#_Toc132951250)

[I.6 Cấu trúc I/O 7](#_Toc132951251)

[I.7 Kiến trúc hệ thống máy tính 7](#_Toc132951252)

[I.8 Hệ thống cụm 8](#_Toc132951253)

[I.9 Hệ thống chuyên dụng 8](#_Toc132951254)

[I.10 Hệ thống đa lập trình 8](#_Toc132951255)

[I.11 Hệ thống chia sẻ thời gian (đa nhiệm) 9](#_Toc132951256)

[I.12 Hoạt động của hệ điều hành 9](#_Toc132951257)

[I.13 Hoạt động ở chế độ kép 9](#_Toc132951258)

[I.14 Hẹn giờ 10](#_Toc132951259)

[I.15 Quản lý quy trình (quản lý quy trình) 10](#_Toc132951260)

[I.16 Quản lý bộ nhớ 11](#_Toc132951261)

[I.17 Quản lý lưu trữ 11](#_Toc132951262)

[I.18 Quản lý hệ thống tệp – quản lý tập tin (File) 11](#_Toc132951263)

[I.19 Quản lý Mass-Storage 12](#_Toc132951264)

[I.20 Bộ nhớ đệm (Caching) 12](#_Toc132951265)

[I.21 Bảo vệ và an ninh 13](#_Toc132951266)

[I.22 Giao diện người dùng 13](#_Toc132951267)

[I.23 Cuộc gọi hệ thống (System Calls) 13](#_Toc132951268)

[Chương 2: Quản lý bộ nhớ 14](#_Toc132951269)

[II.1 Bộ nhớ chính, tổng quan 14](#_Toc132951270)

[II.2 Ràng buộc địa chỉ 14](#_Toc132951271)

[II.3 Không gian địa chỉ logic và vật lý 14](#_Toc132951272)

[II.4 Hoán đổi 14](#_Toc132951273)

[II.5 Cấp phát bộ nhớ liền kề 15](#_Toc132951274)

[II.6 Cấp phát bộ nhớ 15](#_Toc132951275)

[II.7 Phân mảnh 16](#_Toc132951276)

[II.8 Phân trang – Paging 16](#_Toc132951277)

[II.9 Sự bảo vệ 17](#_Toc132951278)

[II.10 Trang được chia sẻ - chia sẻ trang 17](#_Toc132951279)

[II.11 Cấu trúc của bảng trang 17](#_Toc132951280)

[II.12 Phân đoạn 18](#_Toc132951281)

[Chương 3: Quy trình – process 19](#_Toc132951282)

[III.1 Khái niệm 19](#_Toc132951283)

[III.2 Trạng thái quy trình 19](#_Toc132951284)

[III.3 Khối điều khiển quá trình (PCB) 19](#_Toc132951285)

[III.4 Lập lịch quy trình 20](#_Toc132951286)

[III.5 Lập lịch hàng đợi 20](#_Toc132951287)

[III.6 Người lập lịch – Schedulers 21](#_Toc132951288)

[III.7 Tạo quy trình con 22](#_Toc132951289)

[III.8 Chấm dứt quy trình 22](#_Toc132951290)

[III.9 Truyền thông liên quy trình – chia sẻ quy trình 23](#_Toc132951291)

[III.10 Hệ thống bộ nhớ dùng chung – chia sẻ bộ nhớ giữa các quy trình 23](#_Toc132951292)

[III.11 Hệ thống truyền tin nhắn – Message-Passing 24](#_Toc132951293)

[III.12 Sockets 24](#_Toc132951294)

[III.13 Các cuộc gọi thủ tục từ xa - Remote Procedure Calls (RPC) 25](#_Toc132951295)

[Chương 4: Threads 25](#_Toc132951296)

[IV.1 Tổng quan 25](#_Toc132951297)

[IV.2 Lợi ích 26](#_Toc132951298)

[IV.3 Mô hình đa luồng 26](#_Toc132951299)

[IV.4 Mô hình Many – to – one 26](#_Toc132951300)

[IV.5 Mô hình One – to – one 26](#_Toc132951301)

[IV.6 Mô hình Many – to – many 27](#_Toc132951302)

[IV.7 Thư viện Thread 27](#_Toc132951303)

[IV.8 Posix Pthreads ((POSIX: Portable Operating System Interface [for Unix]) 28](#_Toc132951304)

[IV.9 Win32 Threads 28](#_Toc132951305)

[IV.10 Java Threads 28](#_Toc132951306)

[IV.11 Các vấn đề 28](#_Toc132951307)

[IV.12 Hủy bỏ luồng 29](#_Toc132951308)

[IV.13 Xử lý tín hiệu 29](#_Toc132951309)

[IV.14 Thread pools 30](#_Toc132951310)

[Chương 5: Lập lịch CPU 31](#_Toc132951311)

[V.1 Khái niệm 31](#_Toc132951312)

[V.2 Chu kỳ CPU-I/O Brust 31](#_Toc132951313)

[V.3 Bộ lập lịch CPU – CPU Scheduler 31](#_Toc132951314)

[V.4 Lập lịch không ưu tiên - Nonpreemptive scheduling 32](#_Toc132951315)

[V.5 Lập lịch ưu tiên - Preemptive scheduling 32](#_Toc132951316)

[V.6 Điều phối – Dispatcher 32](#_Toc132951317)

[V.7 Tiêu chí lập kế hoạch - Scheduling Criteria 32](#_Toc132951318)

[V.8 Thuật toán lập lịch đến trước phục vụ trước – FCFS 33](#_Toc132951319)

[V.9 Thuật toán lập lịch thực hiện công việc nhỏ nhất trước – SJF 33](#_Toc132951320)

[V.10 Lập lịch kế hoạch ưu tiên - Priority Scheduling 34](#_Toc132951321)

[V.11 Lập lịch kế hoạch quay vòng – RR 35](#_Toc132951322)

[V.12 Lập lịch hàng đợi đa cấp - Multilevel Queue Scheduling 35](#_Toc132951323)

[V.13 Lập lịch kế hoạch hàng đợi phản hồi đa cấp - Multilevel Feedback-Queue Scheduling 35](#_Toc132951324)

[V.14 Lập lịch đa bộ xử lý - Multiple-Processor Scheduling 36](#_Toc132951325)

[V.15 Đánh giá thuật toán 37](#_Toc132951326)

[Chương 6: Đồng bộ hóa quy trình - Process Synchronization 38](#_Toc132951327)

[VII.1 Tình huống cạnh tranh – race condition 38](#_Toc132951328)

[VII.2 Vấn đề phần quan trọng - The Critical-Section Problem 38](#_Toc132951329)

[VII.3 Giải pháp của Peterson 39](#_Toc132951330)

[VII.4 Giải pháp của phần cứng - Synchronization Hardware 39](#_Toc132951331)

[VII.5 Giải pháp Semaphores 40](#_Toc132951332)

[VII.6 Deadlocks và Starvation 40](#_Toc132951333)

[Chương 7: Deadlock 42](#_Toc132951334)

[VII.1 Khái niệm 42](#_Toc132951335)

[VII.2 Mô hình hệ thống 42](#_Toc132951336)

[VII.3 Đặc tính 43](#_Toc132951337)

[VII.4 Điều kiện xảy ra deadlock 43](#_Toc132951338)

[VII.5 Mô tả bằng biểu đồ có hướng 43](#_Toc132951339)

[VII.6 Phương pháp xử lý deadlock 44](#_Toc132951340)

[VII.7 Ngăn chặn deadlock 44](#_Toc132951341)

[VII.8 Tránh deadlock 45](#_Toc132951342)

[VII.9 Trạng thái an toàn - Safe State 45](#_Toc132951343)

[VII.10 Thuật toán đồ thị phân bổ tài nguyên 46](#_Toc132951344)

[VII.11 Thuật toán ngân hàng - Banker's Algorithm 46](#_Toc132951345)

[VII.12 Thuật toán yêu cầu thêm tài nguyên 47](#_Toc132951346)

[VII.13 Phát hiện Deadlock 47](#_Toc132951347)

[VII.14 Phục hồi từ deadlock 47](#_Toc132951348)

[VII.15 Ưu tiên tài nguyên 47](#_Toc132951349)

[Chương 8: Quản lý I/O 49](#_Toc132951350)

[VIII.1 Tổng quan 49](#_Toc132951351)

[VIII.2 Phần cứng I/O 49](#_Toc132951352)

[VIII.3 Bỏ phiếu hoặc bắt tay - Polling or Handshaking 50](#_Toc132951353)

[VIII.4 Ngắt – Interrupts 51](#_Toc132951354)

[VIII.5 Truy cập bộ nhớ trực tiếp - Direct Memory Access (DMA) 51](#_Toc132951355)

[VIII.6 Giao diện I/O ứng dụng - Application I/O interface 52](#_Toc132951356)

[VIII.7 Thiết bị khối và ký tự - Block and Character Devices 52](#_Toc132951357)

[VIII.8 Thiết bị mạng - Network Device 53](#_Toc132951358)

[VIII.9 Đồng hồ và hẹn giờ - Clocks and Timers 53](#_Toc132951359)

[VIII.10 Blocking and Nonblocking I/O 54](#_Toc132951360)

[VIII.11 Kernel I/O Subsystem 54](#_Toc132951361)

[VIII.12 Lập lịch I/O - I/O Scheduling 54](#_Toc132951362)

[VIII.13 Buffering – Bộ đệm 55](#_Toc132951363)

[VIII.14 Caching 55](#_Toc132951364)

[VIII.15 Spooling 55](#_Toc132951365)

[VIII.16 Xử lý lỗi - Error Handling 55](#_Toc132951366)

[Chương 9: Quản lý File 56](#_Toc132951367)

[IX.1 Khái niệm 56](#_Toc132951368)

[IX.2 Thuộc tính File 56](#_Toc132951369)

[IX.3 Hoạt động 56](#_Toc132951370)

[IX.4 Truy câp tuần tự - Sequential Access 57](#_Toc132951371)

[IX.5 Truy cập trực tiếp - Direct Access (or relative access) 57](#_Toc132951372)

[IX.6 Các cách truy cập khác 58](#_Toc132951373)

[IX.7 Cấu trúc thư mục 58](#_Toc132951374)

[IX.8 Cấu trúc lưu trữ 58](#_Toc132951375)

[IX.9 Tổng quan về thư mục 58](#_Toc132951376)

[IX.10 Chế độ nhiều người dùng 59](#_Toc132951377)

[IX.11 Hệ thống tệp từ xa - Remote File Systems 59](#_Toc132951378)

[Câu hỏi ôn tập 60](#_Toc132951379)

# Chương 1: Tổng quan

### Hệ điều hành – Operating System

Một chương trình quản lý phần cứng máy tính

Cung cấp cơ sở cho các chương trình ứng dụng

Hoạt động như một trung gian giữa người dùng máy tính và phần cứng.

### Hệ điều hành làm gì

Một hệ thống máy tính có thể chia thành bốn thành phần: phần cứng, hệ điều hành, các chương trình ứng dụng và người dùng.

Hệ điều hành là một chương trình điều khiển quản lý việc thực hiện các chương trình người dùng để ngăn ngừa lỗi và sử dụng không đúng cách của máy tính.

Nó đặc biệt quan tâm đến hoạt động và điều khiển các thiết bị I/O.

### Tổ chức hệ thống máy tính

Một hoặc nhiều CPU

Bộ điều khiển thiết bị được kết nối với bus chung cung cấp truy cập vào bộ nhớ chia sẻ.

CPU và bộ điều khiển thiết bị có thể thực thi song song.

Bộ điều khiển bộ nhớ cung cấp chức năng toàn bộ để đồng bộ hóa việc truy cập vào bộ nhớ.

### Khởi động máy tính

Có một chương trình cơ bản nhất hoặc chương trình bootstrap – trong Rom để chạy. Chương trình đó khởi chạy tất cả các khía cạnh của hệ thống, từ thanh ghi CPU đến thiết bị điều khiển. Chương trình đó phải tải nội dung vào nhân hệ thống. Thực hiện quy trình đầu tiên “init”

### Cấu trúc lưu trữ

Các chương trình máy tính phải được lưu trong bộ nhớ chính (RAM) để thực thi. RAM là khu vực lưu trữ duy nhất mà bộ xử lý có thể truy cập trực tiếp. RAM thường quá nhỏ để lưu trữ tất cả các chương trình và dữ liệu cần thiết một cách lâu dài. RAM là thiết bị lưu trữ tạm thời và mất nội dung của mình khi nguồn điện bị ngắt hoặc mất đi theo cách khác. Hầu hết các hệ thống máy tính cung cấp bộ nhớ lưu trữ thứ cấp như một phần mở rộng của bộ nhớ chính.

Thanh ghi nằm trực tiếp trên CPU

### Cấu trúc I/O

Hệ thống máy tính có nhiều bộ điều khiển thiết bị được kết nối thông qua một bus chung.

Mỗi bộ điều khiển thiết bị chịu trách nhiệm với một loại thiết bị cụ thể. Các bộ điều khiển thiết bị này được điều khiển bởi hệ điều hành để đảm bảo tính tương thích và hiệu quả của chúng.

Để bắt đầu một hoạt động I/O, trình điều khiển thiết bị tải các thanh ghi phù hợp trong bộ điều khiển thiết bị.

Bộ điều khiển thiết bị xem xét nội dung của các thanh ghi này để xác định hành động cần thực hiện (như "đọc một ký tự từ bàn phím").

Bộ điều khiển bắt đầu chuyển dữ liệu từ thiết bị đến bộ đệm lưu trữ cục bộ của nó, thông báo cho trình điều khiển thiết bị qua một ngắt rằng nó đã hoàn thành hoạt động của mình.

Sau đó, trình điều khiển thiết bị trả lại quyền điều khiển cho hệ điều hành.

DMA (Direct Memory Access) được sử dụng để chuyển dữ liệu số lượng lớn như I/O đĩa.

* Bộ điều khiển thiết bị chuyển toàn bộ khối dữ liệu trực tiếp vào hoặc ra khỏi bộ đệm lưu trữ riêng của nó đến bộ nhớ mà không có sự can thiệp của CPU.
* Chỉ có một ngắt được tạo ra cho mỗi khối, để thông báo cho trình điều khiển thiết bị rằng hoạt động đã hoàn thành, thay vì một ngắt cho mỗi byte được tạo ra cho các thiết bị tốc độ chậm.

### Kiến trúc hệ thống máy tính

Một máy tính có thể được tổ chức những các khác nhau:

Hệ thống đơn xử lý: Chỉ có một CPU tổng quát

Hệ thống đa xử lý – song song: Có 2 hoặc nhiều CPU. Lợi ích: Tăng thời lượng xử lý – thực hiện nhiều công việc trong thời gian ngắn; Tiết kiệm tài nguyên hệ thống – vì có thể chia sẻ các thiết bị ngoại vi, bộ nhớ và nguồn điện; Tăng độ tin cậy – nếu các chức năng được phân phối đúng đắn giữa các bộ xử lý, thì sự cố của một bộ xử lý sẽ không làm đình trệ hệ thống, chỉ làm nó chậm lại. Hệ thống multiprocessor có thể được chia thành hai loại chính:

* Asymmetric Multiprocessing (AMP): một CPU được chỉ định làm master processor để quản lý tất cả các hoạt động của hệ thống, trong khi các CPU khác được gọi là slave processors, chỉ được phân công thực hiện các tác vụ được chia sẻ bởi master processor.
* Symmetric Multiprocessing (SMP): tất cả các CPU trong hệ thống được coi là đồng bộ và có thể thực hiện bất kỳ tác vụ nào trong hệ thống. Hệ thống này có thể mở rộng dễ dàng bằng cách thêm các CPU mới và cân bằng tải công việc giữa các CPU.

### Hệ thống cụm

Một loại hệ thống nhiều CPU khác.

Giống như các hệ thống multiprocessing, các hệ thống gom nhiều CPU lại với nhau để thực hiện công việc tính toán.

Các hệ thống gom nhóm khác biệt so với các hệ thống multiprocessor là chúng được tạo thành từ hai hoặc nhiều hệ thống riêng lẻ được kết nối với nhau.

Định nghĩa được chấp nhận chung là các máy tính trong cụm chia sẻ bộ nhớ lưu trữ và được kết nối chặt chẽ thông qua một mạng nội bộ.

Clustering thường được sử dụng để đảm bảo tính khả dụng cao cho dịch vụ, tức là dịch vụ sẽ tiếp tục hoạt động ngay cả khi một hoặc nhiều hệ thống trong cụm bị lỗi.

Clustering có thể được cấu trúc không đối xứng hoặc đối xứng.

### Hệ thống chuyên dụng

Hệ điều hành thời gian thực - Là những hệ thống được đặc trưng bởi thời gian là tham số chính. Ví dụ, trong các hệ thống điều khiển quá trình công nghiệp.

Hệ điều hành nhúng - Tiếp tục với những hệ thống nhỏ hơn và nhỏ hơn, hệ điều hành đến với các máy tính cầm tay và các hệ thống nhúng. Các hệ thống nhúng chạy trên các máy tính điều khiển các thiết bị không phải là máy tính, chẳng hạn như TV, lò vi sóng và điện thoại di động. Chúng thường có một số đặc điểm của hệ thống thời gian thực nhưng cũng có các hạn chế về kích thước, bộ nhớ và năng lượng khiến chúng trở nên đặc biệt.

### Hệ thống đa lập trình

Một trong những khía cạnh quan trọng nhất của hệ điều hành là khả năng multiprogramming (đa chương trình). Đa chương trình tăng cường sử dụng CPU bằng cách tổ chức các tác vụ (dữ liệu và mã) sao cho CPU luôn có một tác vụ để thực thi.

Hệ điều hành chọn và bắt đầu thực thi một trong số các công việc trong bộ nhớ.

### Hệ thống chia sẻ thời gian (đa nhiệm)

Hệ thống time-sharing là một sự mở rộng logic của hệ thống multiprogramming. Trong hệ thống này, CPU thực thi nhiều công việc bằng cách chuyển đổi giữa chúng, nhưng các chuyển đổi xảy ra rất thường xuyên, cho phép người dùng tương tác với mỗi chương trình trong khi nó đang chạy.

Một hệ điều hành chia sẻ thời gian cho phép nhiều người dùng chia sẻ máy tính cùng một lúc

Khi hệ thống chuyển đổi nhanh chóng từ một người dùng sang người dùng khác, mỗi người dùng có cảm giác như toàn bộ hệ thống máy tính được dành riêng cho việc sử dụng của họ, mặc dù nó đang được chia sẻ giữa nhiều người dùng.

Hệ điều hành chia sẻ thời gian sử dụng CPU và đa chương trình để cung cấp cho mỗi người dùng một phần nhỏ của máy tính được chia sẻ thời gian. Mỗi người dùng có ít nhất một chương trình riêng biệt trong bộ nhớ. Một chương trình được tải vào bộ nhớ và thực thi được gọi là một quy trình.

### Hoạt động của hệ điều hành

Hệ điều hành hiện đại được điều khiển bởi các ngắt.

Nếu không có quá trình nào để thực thi, không có thiết bị I/O để phục vụ và không có người dùng để phản hồi, hệ điều hành sẽ ngồi im lặng, đợi điều gì đó xảy ra.

Sự kiện thường được báo hiệu bởi việc xảy ra một ngắt hoặc tràn số.

Một hệ điều hành được thiết kế đúng cách phải đảm bảo rằng một chương trình không chính xác (hoặc độc hại) không thể gây ra các chương trình khác thực thi không chính xác. Nếu không có bảo vệ chống lại những loại lỗi này, máy tính phải thực thi chỉ một quá trình tại một thời điểm hoặc tất cả đầu ra phải được xem là không đáng tin cậy.

### Hoạt động ở chế độ kép

Hai chế độ của hệ điều hành:

* Chế độ người dùng
* Chế độ hạt nhân (supervisor mode, chế độ hệ thống hoặc chế độ được đặc quyền)

Một bit, được gọi là bit mode, được thêm vào phần cứng của máy tính để chỉ ra chế độ hiện tại: chế độ kernel (0) hoặc chế độ người dùng (1).

Với bit mode, hệ điều hành có thể phân biệt được giữa một nhiệm vụ được thực thi thay mặt cho hệ điều hành và một nhiệm vụ được thực thi thay mặt cho người dùng.

Khi một ứng dụng người dùng yêu cầu một dịch vụ từ hệ điều hành (thông qua một cuộc gọi hệ thống), nó phải chuyển từ chế độ người dùng sang chế độ kernel để thực hiện yêu cầu đó.

Khi khởi động hệ thống, phần cứng bắt đầu ở chế độ kernel. Hệ điều hành sau đó được tải và khởi động các ứng dụng người dùng ở chế độ user.

Mỗi khi hệ điều hành giành quyền kiểm soát máy tính, nó sẽ ở chế độ kernel.

Các lệnh hệ thống (system calls) cung cấp phương tiện cho một chương trình người dùng yêu cầu hệ điều hành thực hiện các tác vụ được dành riêng cho hệ điều hành thay cho chương trình người dùng.

### Hẹn giờ

Để đảm bảo rằng hệ điều hành giữ được kiểm soát trên CPU và ngăn chặn một chương trình người dùng bị kẹt trong một vòng lặp vô hạn hoặc không gọi các dịch vụ hệ thống và không bao giờ trả lại kiểm soát cho hệ điều hành.

Hệ điều hành sử dụng bộ định thời để đạt được mục tiêu này.

Bộ định thời có thể được đặt để gián đoạn máy tính sau một khoảng thời gian cụ thể.

Khoảng thời gian có thể là cố định hoặc biến đổi.

### Quản lý quy trình (quản lý quy trình)

Một chương trình đang chạy là một quá trình (process).

Một quá trình cần một số tài nguyên - bao gồm thời gian CPU, bộ nhớ, tập tin và các thiết bị I/O để hoàn thành nhiệm vụ của nó. Những tài nguyên này được cung cấp cho quá trình khi nó được tạo ra hoặc được cấp phát cho nó khi nó đang chạy.

Ngoài các tài nguyên vật lý và logic khác nhau mà một quá trình nhận được khi được tạo ra, các dữ liệu khởi tạo khác (đầu vào) có thể được truyền đi. Khi quá trình kết thúc, hệ điều hành sẽ thu hồi bất kỳ tài nguyên có thể tái sử dụng nào.

Hệ điều hành có trách nhiệm:

* Tạo và xóa cả process của người dùng và hệ thống.
* Đình chỉ và tiếp tục các process.
* Cung cấp cơ chế đồng bộ hóa process.
* Cung cấp cơ chế giao tiếp process.
* Cung cấp cơ chế xử lý tình trạng khóa chết (deadlock).

### Quản lý bộ nhớ

Bộ nhớ chính là trung tâm của hệ thống máy tính hiện đại. Bộ nhớ chính là một mảng lớn các từ hoặc byte. Mỗi từ hoặc byte có địa chỉ riêng của nó.

Bộ nhớ chính là kho lưu trữ dữ liệu có thể truy cập nhanh được chia sẻ bởi CPU và các thiết bị I/O.

Bộ xử lý trung tâm đọc và ghi dữ liệu từ bộ nhớ chính.

Để một chương trình được thực thi, nó phải được ánh xạ vào các địa chỉ tuyệt đối và tải vào bộ nhớ.

Hệ điều hành có trách nhiệm:

* Theo dõi các phần của bộ nhớ hiện đang được sử dụng và được sử dụng bởi ai.
* Quyết định quá trình và dữ liệu nào sẽ được di chuyển vào và ra khỏi bộ nhớ.
* Phân bổ và giải phóng không gian bộ nhớ khi cần thiết.

### Quản lý lưu trữ

Để làm cho hệ thống máy tính tiện lợi cho người dùng, hệ điều hành cung cấp một cái nhìn thống nhất, logic về lưu trữ thông tin. Hệ điều hành trừu tượng hoá từ các thuộc tính vật lý của các thiết bị lưu trữ để định nghĩa một đơn vị lưu trữ logic - tập tin (file). Hệ điều hành ánh xạ các tập tin này vào phương tiện vật lý và truy cập các tập tin này thông qua các thiết bị lưu trữ.

### Quản lý hệ thống tệp – quản lý tập tin (File)

Máy tính có thể lưu trữ thông tin trên nhiều loại phương tiện vật lý khác nhau. Mỗi phương tiện này có đặc tính và tổ chức vật lý riêng của nó. Mỗi phương tiện được điều khiển bởi một thiết bị như ổ đĩa hoặc băng đĩa, mà cũng có đặc tính riêng của nó.

Hệ điều hành thực hiện khái niệm trừu tượng về một tập tin bằng cách quản lý các phương tiện lưu trữ đại trà, chẳng hạn như băng và đĩa cứng, và các thiết bị điều khiển của chúng. Ngoài ra, các tệp thông thường được tổ chức thành các thư mục để dễ sử dụng hơn.

Hệ điều hành có trách nhiệm:

* Tạo và xóa các tập tin.
* Tạo và xóa các thư mục để tổ chức các tập tin.
* Hỗ trợ các nguyên thủy (primitive) để thao tác các tập tin và thư mục.
* Ánh xạ các tập tin vào bộ nhớ lưu trữ thứ cấp.
* Sao lưu các tập tin trên các phương tiện lưu trữ ổn định (không bay hơi).

### Quản lý Mass-Storage

Hầu hết các hệ thống máy tính hiện đại sử dụng đĩa như một phương tiện lưu trữ trực tuyến chính cho cả chương trình và dữ liệu.

Hầu hết các chương trình được lưu trữ trên đĩa cho đến khi được tải vào bộ nhớ và sau đó sử dụng đĩa làm cả nguồn và đích đến của quá trình xử lý của chúng. Do đó, việc quản lý lưu trữ đĩa đúng cách là rất quan trọng đối với một hệ thống máy tính.

Hệ điều hành có trách nhiệm:

* Quản lý không gian trống trên đĩa
* Phân bổ lưu trữ
* Định thời đĩa (Disk schedule).

### Bộ nhớ đệm (Caching)

Caching (hay còn gọi là bộ nhớ cache) là một nguyên tắc quan trọng trong hệ thống máy tính. Thông tin thường được lưu trữ trong một hệ thống bộ nhớ như bộ nhớ chính. Khi thông tin được sử dụng, nó được sao chép vào một hệ thống lưu trữ nhanh hơn - bộ nhớ cache - trên cơ sở tạm thời.

Khi hệ điều hành cần một mẩu thông tin cụ thể, hệ điều hành đầu tiên kiểm tra xem nó có trong bộ nhớ cache hay không. Nếu có, hệ điều hành sử dụng thông tin trực tiếp từ bộ nhớ cache. Nếu không có, hệ điều hành sử dụng thông tin từ nguồn, đồng thời sao chép một bản sao vào bộ nhớ cache dưới giả định rằng hệ điều hành sẽ cần nó lại sớm.

Vì cache có kích thước giới hạn, quản lý cache là một vấn đề thiết kế quan trọng. Việc lựa chọn kích thước cache và chính sách thay thế phù hợp có thể dẫn đến cải thiện đáng kể hiệu suất.

### Bảo vệ và an ninh

Nếu một hệ thống máy tính có nhiều người dùng và cho phép thực thi nhiều quá trình, thì truy cập vào dữ liệu phải được quy định. Vì mục đích đó, các cơ chế đảm bảo rằng các tệp, phân đoạn bộ nhớ, CPU và các tài nguyên khác chỉ được thao tác bởi những quy trình đã có sự cho phép thích hợp từ hệ điều hành.

Bảo vệ là bất kỳ cơ chế nào để kiểm soát quyền truy cập của các quá trình hoặc người dùng vào các tài nguyên được định nghĩa bởi máy tính.

Bảo mật là để bảo vệ hệ thống khỏi các cuộc tấn công bên ngoài và bên trong. Các cuộc tấn công này trải rộng trên một phạm vi rất lớn và bao gồm các virus và sâu, từ chối dịch vụ, ăn cắp dịch vụ (sử dụng hệ thống một cách trái phép) ...

Bảo vệ và bảo mật yêu cầu hệ thống có thể phân biệt được giữa tất cả người dùng của nó. Hầu hết các hệ điều hành duy trì một danh sách tên người dùng và các định danh người dùng tương ứng (ID người dùng).

### Giao diện người dùng

Giao diện dòng lệnh (CLI) hoặc trình thông dịch lệnh cho phép nhập lệnh trực tiếp.

Giao diện đồ họa người dùng (GUI) thường sử dụng chuột, bàn phím và màn hình

### Cuộc gọi hệ thống (System Calls)

Dùng để người dùng yêu cầu dịch vụ từ hệ thống

Ba API phổ biến nhất là Win32 API cho Windows, POSIX API cho các hệ thống dựa trên POSIX (bao gồm hầu hết các phiên bản của UNIX, Linux và Mac OS X) và API Java cho máy ảo Java (JVM).

# Chương 2: Quản lý bộ nhớ

### Bộ nhớ chính, tổng quan

Bộ nhớ chính và các thanh ghi được tích hợp vào bộ xử lý là những vùng lưu trữ duy nhất mà CPU có thể truy cập trực tiếp.

Bộ nhớ nhanh giữa CPU và bộ nhớ chính - một bộ đệm bộ nhớ được sử dụng để chứa một sự chênh lệch về tốc độ, được gọi là bộ nhớ cache.

Mỗi quy trình có không gian bộ nhớ riêng và để đảm bảo rằng quy trình chỉ có thể truy cập vào các địa chỉ hợp lệ này.Sự bảo vệ này được thực hiện bằng cách sử dụng hai thanh ghi, thường là thanh ghi cơ sở (base) và thanh ghi giới hạn (limit).

### Ràng buộc địa chỉ

Các địa chỉ trong chương trình nguồn thường mang tính biểu tượng . Một trình biên dịch thường sẽ liên kết các địa chỉ biểu tượng này với các địa chỉ có thể di chuyển

### Không gian địa chỉ logic và vật lý

Địa chỉ logic - địa chỉ ảo: Một địa chỉ được tạo ra bởi CPU.

Tập hợp tất cả các địa chỉ logic được tạo ra bởi một chương trình là một không gian địa chỉ logic.

Địa chỉ vật lý: Một địa chỉ được nhìn thấy bởi đơn vị bộ nhớ.

Tập hợp tất cả các địa chỉ vật lý tương ứng với các địa chỉ logic này là một không gian địa chỉ vật lý.

Ánh xạ thời gian chạy từ địa chỉ ảo sang địa chỉ vật lý được thực hiện bởi một thiết bị phần cứng được gọi là đơn vị quản lý bộ nhớ (MMU).

### Hoán đổi

Một quy trình có thể bị gỡ tạm thời khỏi bộ nhớ đến một vùng lưu trữ phụ (backing store), sau đó được đưa trở lại bộ nhớ để tiếp tục thực thi.

Gỡ bỏ yêu cầu một vùng lưu trữ phụ. Vùng lưu trữ phụ thường là một ổ đĩa nhanh. Nó phải đủ lớn để chứa bản sao của tất cả các hình ảnh bộ nhớ cho tất cả các người dùng và phải cung cấp truy cập trực tiếp đến các hình ảnh bộ nhớ này.

Thông thường, một quy trình bị swap out sẽ được swap trở lại cùng không gian bộ nhớ mà nó đã chiếm trước đó. Ràng buộc này được quy định bởi phương pháp kết nối địa chỉ.

Nếu kết nối được thực hiện tại thời điểm lắp ráp hoặc tải lên, thì quá trình không thể dễ dàng di chuyển đến vị trí khác.

Tuy nhiên, nếu sử dụng kết nối thời gian thực thi, một quy trình có thể được swap vào một không gian bộ nhớ khác, vì các địa chỉ vật lý được tính toán trong thời gian thực thi.

### Cấp phát bộ nhớ liền kề

Bộ nhớ chính phải đáp ứng được cả hệ điều hành và các quy trình người dùng khác nhau. Do đó, hệ điều hành cần phải phân bổ các phần của bộ nhớ chính một cách hiệu quả nhất có thể.

### Cấp phát bộ nhớ

(MFT-Multiprogramming với số lượng Tác vụ Cố định):

* Một trong những phương pháp đơn giản nhất để phân bổ bộ nhớ là chia bộ nhớ thành một số phân vùng có kích thước cố định. Mỗi phân vùng có thể chứa chính xác một quy trình.
* Trong phương pháp phân vùng nhiều, khi một phân vùng trống, một quy trình được chọn từ hàng đợi đầu vào và được tải vào phân vùng trống. Khi quy trình kết thúc, phân vùng trở thành có sẵn cho một quy trình khác.

MVT (Multiprogramming với số lượng tác vụ Biến đổi hoặc chế độ phân vùng cố định):

* Hệ điều hành duy trì một bảng chỉ ra phần nào của bộ nhớ có sẵn và phần nào đang được sử dụng. Ban đầu, toàn bộ bộ nhớ đều có sẵn cho các quy trình người dùng và được coi là một khối lớn của bộ nhớ có sẵn, được gọi là "Hole".
* Khi một quy trình đến và cần bộ nhớ, hệ điều hành tìm kiếm một hole đủ lớn cho quy trình này. Nếu tìm thấy một hole đủ lớn, thì chỉ cấp phát bộ nhớ cần thiết, giữ lại phần còn lại để đáp ứng các yêu cầu trong tương lai.

Khi các quy trình nhập vào hệ thống, chúng được đưa vào một hàng đợi nhập.

Hệ điều hành xem xét yêu cầu về bộ nhớ của từng quy trình và lượng không gian bộ nhớ khả dụng để quyết định xem quy trình nào được cấp phát bộ nhớ.

Khi một quy trình được cấp phát không gian, nó được nạp vào bộ nhớ và sau đó có thể cạnh tranh cho CPU.

Khi một quy trình kết thúc, nó giải phóng bộ nhớ của nó, mà hệ điều hành có thể sử dụng để chứa một quy trình khác từ hàng đợi nhập.

Cấp phát sử dụng bộ nhớ:

* First fit: Cấp phát khoảng trống đầu tiên đủ lớn.
* Best fit: Cấp phát khoảng trống nhỏ nhất đủ lớn.
* Worst fit: Cấp phát khoảng trống lớn nhất.

Các mô phỏng đã cho thấy cả first fit và best fit đều tốt hơn worst fit

### Phân mảnh

Khi quá trình được tải lên và gỡ bỏ khỏi bộ nhớ, không gian nhớ trống được chia thành nhiều phần nhỏ.

Hiện tượng phân mảnh bên ngoài xảy ra khi tổng không gian nhớ đủ để đáp ứng yêu cầu, nhưng các không gian khả dụng không liên tục; bộ nhớ bị phân mảnh thành một số lỗ nhỏ.

External fragmentation – phân mảnh bên ngoài bộ nhớ có thể gây ra sự external fragmentation bên trong cũng như bên ngoài. Nếu bộ nhớ được phân bổ cho một quy trình lớn hơn so với bộ nhớ yêu cầu, thì sự khác biệt giữa hai số này là internal fragmentation, là bộ nhớ nằm trong một phân vùng nhưng không được sử dụng.

Một giải pháp cho vấn đề external fragmentation bên ngoài là nén. Mục tiêu là xáo trộn nội dung bộ nhớ để đặt tất cả các bộ nhớ trống cùng với nhau thành một khối lớn. Tuy nhiên, việc nén không phải lúc nào cũng khả thi

Một giải pháp khác cho vấn đề phân mảnh bên ngoài là cho phép không gian địa chỉ logic của các quy trình không liên tục, cho phép một quy trình được cấp phát bộ nhớ vật lý ở bất kỳ đâu nơi mà bộ nhớ vật lý có sẵn. Hai kỹ thuật bổ sung nhau để đạt được giải pháp này là phân trang và phân đoạn.

### Phân trang – Paging

Paging là một chương trình quản lý bộ nhớ cho phép không gian địa chỉ vật lý của một quy trình không liên tục.

Phương pháp cơ bản để thực hiện phân trang bao gồm việc chia bộ nhớ vật lý thành các khối cố định được gọi là khung (frames) và chia bộ nhớ logic thành các khối có cùng kích thước được gọi là trang (pages).

Khi một quy trình được thực thi, các trang của nó được tải vào bất kỳ khung bộ nhớ có sẵn nào từ bộ nhớ đệm. Bộ nhớ đệm được chia thành các khối có kích thước cố định giống như các khung bộ nhớ.

Khi sử dụng một phương pháp phân trang, chúng ta không có sự phân mảnh bên ngoài: Bất kỳ khung trống nào cũng có thể được phân bổ cho một quy trình cần nó. Tuy nhiên, chúng ta có thể gặp phải một số sự phân mảnh bên trong. Lưu ý rằng các khung được phân bổ dưới dạng các đơn vị.

### Sự bảo vệ

Bảo vệ bộ nhớ trong môi trường được phân trang được thực hiện thông qua các bit bảo vệ được liên kết với mỗi frame. Thông thường, các bit này được lưu trữ trong bảng phân trang.

Hệ điều hành thiết lập bit này cho mỗi trang để cho phép hoặc cấm truy cập vào trang.

Một bit bổ sung được gắn vào mỗi mục trong bảng phân trang: một bit hợp lệ-không hợp lệ.

### Trang được chia sẻ - chia sẻ trang

Một lợi thế của phân trang là khả năng chia sẻ mã chung. Điều này rất quan trọng trong một môi trường chia sẻ thời gian.

Vd: Hãy xem xét một hệ thống hỗ trợ 40 người dùng, mỗi người dùng thực thi một trình soạn thảo văn bản. Nếu trình soạn thảo bao gồm 150 KB mã và 50 KB không gian dữ liệu, chúng ta cần 8.000 KB để hỗ trợ 40 người dùng.

Reentrant code là mã không tự chuyển đổi, nó không bao giờ thay đổi trong quá trình thực thi. Vì vậy, hai hoặc nhiều quá trình có thể thực thi cùng mã nguồn vào cùng một thời điểm. Mỗi quá trình có bản sao riêng của thanh ghi và bộ lưu trữ dữ liệu để lưu trữ dữ liệu cho việc thực thi của quá trình đó. Dữ liệu cho hai quá trình khác nhau sẽ, tất nhiên, là khác nhau.

Chỉ cần giữ một bản sao của trình soạn thảo trong bộ nhớ vật lý. Bảng trang của mỗi người dùng được ánh xạ vào cùng một bản sao vật lý của trình soạn thảo, nhưng các trang dữ liệu được ánh xạ vào các khung khác nhau. Do đó, để hỗ trợ 40 người dùng, chúng ta chỉ cần một bản sao của trình soạn thảo (150 KB), cộng với 40 bản sao của không gian dữ liệu 50 KB cho mỗi người dùng. Tổng không gian cần thiết hiện tại là 2.150 KB thay vì 8.000 KB.

### Cấu trúc của bảng trang

- Phân theo thứ tự bậc: Hầu hết các hệ thống máy tính hiện đại hỗ trợ không gian địa chỉ logic lớn (từ 232 đến 264). Trong môi trường như vậy, bảng trang (page table) trở nên quá lớn. Một giải pháp đơn giản cho vấn đề này là chia bảng trang thành các phần nhỏ hơn.

- Bảng trang băm: Phương pháp phổ biến để xử lý không gian địa chỉ lớn hơn 32 bit là sử dụng bảng trang bị băm (hashed page table), trong đó giá trị băm là số trang ảo (virtual page number).

Mỗi mục trong bảng băm chứa một danh sách liên kết các phần tử băm vào cùng một vị trí.

Mỗi phần tử bao gồm ba trường:

(1) số trang ảo

(2) giá trị khung trang được ánh xạ

(3) con trỏ đến phần tử tiếp theo trong danh sách liên kết.

- Bảng trang đảo ngược: Sử dụng bảng trang đảo để giải quyết vấn đề mỗi bảng trang có thể bao gồm hàng triệu mục và các bảng này có thể sử dụng nhiều bộ nhớ vật lý chỉ để theo dõi cách bộ nhớ vật lý khác đang được sử dụng.

Một bảng trang đảo có một mục cho mỗi trang thực (hoặc khung) của bộ nhớ. Mỗi mục bao gồm địa chỉ ảo của trang được lưu trữ trong vị trí bộ nhớ thực đó, kèm theo thông tin về quá trình sở hữu trang đó.

### Phân đoạn

Một khía cạnh quan trọng của quản lý bộ nhớ mà trở nên không thể tránh được với phân trang là phân tách giữa cách người dùng nhìn nhận bộ nhớ và bộ nhớ vật lý thực tế.

Cách nhìn nhận của người dùng về bộ nhớ không giống như bộ nhớ vật lý thực tế.

Cách nhìn nhận của người dùng được ánh xạ vào bộ nhớ vật lý.

Phép ánh xạ này cho phép phân biệt giữa bộ nhớ logic và bộ nhớ vật lý.

Người dùng thường muốn xem bộ nhớ như là một tập hợp các đoạn có kích thước khác nhau, mà không có sự sắp xếp cần thiết giữa các đoạn.

Segmentation là một kế hoạch quản lý bộ nhớ hỗ trợ cho cách nhìn của người dùng về bộ nhớ. Một không gian địa chỉ logic là một bộ sưu tập các đoạn.

Mỗi đoạn có một tên và một độ dài. Các địa chỉ chỉ định cả tên đoạn và độ lệch trong đoạn.

Do đó, người dùng chỉ định mỗi địa chỉ bằng hai lượng: tên đoạn và độ lệch.

# Chương 3: Quy trình – process

### Khái niệm

Quá trình là một chương trình đang thực thi; việc thực thi quá trình phải tiến triển theo trình tự tuần tự.

Một quá trình bao gồm:

* Hoạt động hiện tại, được đại diện bởi giá trị của bộ đếm chương trình và nội dung của các thanh ghi của bộ xử lý.
* Ngăn xếp, chứa dữ liệu tạm thời (như tham số của hàm, địa chỉ trở về và biến cục bộ)
* Một phần dữ liệu, chứa các biến toàn cục.
* Một quá trình cũng có thể bao gồm một heap, là bộ nhớ được cấp phát động trong thời gian chạy của quá trình.

### Trạng thái quy trình

Khi một quá trình thực thi, nó thay đổi trạng thái. Mỗi quá trình có thể ở một trong các trạng thái sau:

* New: Quá trình đang được tạo ra.
* Running: Các lệnh đang được thực thi.
* Waiting: Quá trình đang chờ đợi một sự kiện nào đó xảy ra (như hoàn thành I/O hoặc nhận tín hiệu).
* Ready: Quá trình đang chờ được phân bổ cho một bộ xử lý.
* Terminated: Quá trình đã hoàn thành thực thi.

### Khối điều khiển quá trình (PCB)

Mỗi quá trình được đại diện trong hệ điều hành bằng một khối điều khiển quá trình (PCB) - còn được gọi là khối điều khiển nhiệm vụ.

Trạng thái của quá trình. Trạng thái có thể là mới, sẵn sàng, đang chạy, đang chờ và đã dừng.

Bộ đếm chương trình. Bộ đếm chỉ ra địa chỉ của lệnh tiếp theo sẽ được thực thi cho quá trình này.

Các thanh ghi của CPU. Các thanh ghi khác nhau về số lượng và loại, tùy thuộc vào kiến trúc của máy tính. Chúng bao gồm bộ tích lũy, thanh ghi chỉ mục, trình chỉ ngăn xếp và các thanh ghi thông dụng khác, cùng với bất kỳ thông tin mã điều kiện nào.

Thông tin lập lịch CPU. Thông tin này bao gồm độ ưu tiên của quá trình, các con trỏ đến các hàng đợi lập lịch và bất kỳ thông số lập lịch nào khác.

Thông tin quản lý bộ nhớ. Thông tin này có thể bao gồm các thông tin như giá trị của các thanh ghi cơ sở và giới hạn, các bảng trang hoặc bảng phân đoạn, tùy thuộc vào hệ thống bộ nhớ được sử dụng bởi hệ điều hành.

Thông tin kế toán. Thông tin này bao gồm số lượng CPU và thời gian thực sử dụng, giới hạn thời gian, số tài khoản, số công việc hoặc quá trình và những thông tin tương tự.

Thông tin trạng thái I/O. Thông tin này bao gồm danh sách các thiết bị I/O được phân bổ cho quá trình, danh sách các tệp mở và các thông tin tương tự.

### Lập lịch quy trình

Mục tiêu của việc đa chương trình là để có ít nhất một quá trình đang chạy vào mọi thời điểm, để tối đa hóa sử dụng CPU.

Mục tiêu của việc chia sẻ thời gian là để chuyển đổi CPU giữa các quá trình rất thường xuyên để người dùng có thể tương tác với mỗi chương trình trong khi nó đang chạy.

Để đáp ứng các mục tiêu này, trình lập lịch quá trình chọn một quá trình khả dụng (có thể từ một tập hợp các quá trình khả dụng) để thực thi chương trình trên CPU.

### Lập lịch hàng đợi

Khi các quá trình nhập vào hệ thống, chúng được đưa vào một hàng đợi công việc, bao gồm tất cả các quá trình trong hệ thống.

Các quá trình đang lưu trú trong bộ nhớ chính và sẵn sàng và chờ để thực thi được giữ trên một danh sách gọi là hàng đợi sẵn sàng.

Danh sách các quá trình đang đợi cho một thiết bị I/O cụ thể được gọi là hàng đợi thiết bị. Mỗi thiết bị có hàng đợi thiết bị riêng của nó.

Các quá trình di chuyển giữa các hàng đợi khác nhau.

Một quá trình mới ban đầu được đưa vào hàng đợi sẵn sàng. Nó đợi ở đó cho đến khi được chọn để thực thi hoặc được giao nhiệm vụ. Khi quá trình được phân bổ CPU và đang thực thi, có thể xảy ra một trong vài sự kiện sau:

* Quá trình có thể yêu cầu I/O và sau đó được đưa vào hàng đợi I/O.
* Quá trình có thể tạo một quy trình phụ mới và đợi cho đến khi quy trình phụ kết thúc.
* Quá trình có thể bị gỡ bỏ đột ngột khỏi CPU do một ngắt (interrupt) và được đưa trở lại hàng đợi sẵn sàng.

### Người lập lịch – Schedulers

Quá trình lựa chọn được thực hiện bởi bộ lập lịch phù hợp.

Thường trong một hệ thống hàng loạt, có nhiều quá trình được gửi đến hệ thống hơn là có thể thực thi ngay lập tức. Những quá trình này được lưu trữ vào một thiết bị lưu trữ đại trà (thông thường là đĩa), nơi chúng được giữ để thực thi sau này.

Bộ định thời dài hạn - Lập lịch dài hạn – long-term scheduler hoặc lập lịch công việc chọn các quá trình từ vùng lưu trữ này và tải chúng vào bộ nhớ để thực thi.

Bộ định thời ngắn hạn - Lập lịch ngắn hạn – short-term scheduler hoặc lập lịch CPU chọn từ trong số các quá trình đã sẵn sàng để thực thi và cấp phát CPU cho một trong số chúng.

Sự phân biệt chính giữa hai bộ lập lịch này nằm ở tần suất thực thi.

Bộ lập lịch dài hạn kiểm soát mức độ đa chương trình (số quá trình trong bộ nhớ).

Hầu hết các quy trình có thể được mô tả là quy trình gắn liền với I/O hoặc quy trình gắn liền với CPU.

* Một quy trình gắn liền với I/O là quy trình chiếm nhiều thời gian làm I/O hơn là tính toán.
* Một quy trình gắn liền với CPU, ngược lại, tạo ra yêu cầu I/O ít, sử dụng nhiều thời gian để tính toán.

Trên một số hệ thống, bộ lập lịch dài hạn có thể không có hoặc rất ít.

Ví dụ, các hệ thống time-sharing như các hệ thống UNIX và Microsoft Windows thường không có bộ lập lịch dài hạn mà chỉ đơn giản đưa mọi quy trình mới vào bộ nhớ cho bộ lập lịch ngắn hạn.

Một số hệ điều hành, chẳng hạn như hệ thống chia sẻ thời gian, có thể giới thiệu một cấp độ lập lịch trung gian bổ sung. Lập lịch trung gian - medium-term scheduler này được gọi là lập lịch trung hạn.

Ý tưởng chính đằng sau lập lịch trung hạn là đôi khi có thể thuận lợi hơn để loại bỏ các quy trình khỏi bộ nhớ (và khỏi tranh chấp CPU hoạt động) và do đó giảm bậc đa chương trình.

Sau đó, quá trình có thể được đưa vào bộ nhớ lại và thực thi của nó có thể được tiếp tục từ nơi nó dừng lại. Hệ thống này được gọi là swapping. Quá trình được gỡ bỏ, và sau đó được đưa vào.

### Tạo quy trình con

Quá trình cha con là quá trình cha tạo ra các quy trình con, sau đó các quy trình con lại tạo ra các quy trình khác, tạo thành một cây quy trình.

Hầu hết các hệ điều hành (bao gồm UNIX và gia đình hệ điều hành Windows) xác định các quy trình theo một định danh quy trình duy nhất (hoặc pid), thường là một số nguyên.

Một quy trình cần các tài nguyên nhất định (thời gian CPU, bộ nhớ, tập tin, thiết bị I/O) để thực hiện nhiệm vụ của nó. Có thể xảy ra

* Cha và con chia sẻ tất cả các tài nguyên
* Con chia sẻ một tập hợp con của các tài nguyên của cha
* Cha và con không chia sẻ tài nguyên nào

Khi một quy trình tạo một quy trình mới, có hai khả năng về thực thi:

* Cha tiếp tục thực thi đồng thời với các quy trình con của nó.
* Cha đợi cho đến khi một vài hoặc tất cả các quy trình con của nó kết thúc.

Có hai khả năng về không gian địa chỉ của quy trình mới:

* Quy trình con là một bản sao của quy trình cha (nó có chương trình và dữ liệu giống với quy trình cha).
* Quy trình con có một chương trình mới được tải vào nó.

### Chấm dứt quy trình

Một quy trình kết thúc khi nó thực thi xong câu lệnh cuối cùng và yêu cầu hệ điều hành xóa nó bằng cách sử dụng hàm hệ thống exit ().

Một parent có thể kết thúc việc thực thi của một trong các children của nó vì nhiều lý do, chẳng hạn như:

* Child đã vượt quá tài nguyên được phân bổ.
* Công việc được giao cho child không còn cần thiết nữa.
* Parent đang thoát, và hệ điều hành không cho phép một child tiếp tục nếu parent của nó kết thúc.

Một số hệ thống, bao gồm VMS, không cho phép một child tồn tại nếu parent của nó đã kết thúc. Trong những hệ thống đó, nếu một quy trình kết thúc (bình thường hoặc bất thường), thì tất cả các children của nó cũng phải bị kết thúc - được gọi là kết thúc liên tục.

### Truyền thông liên quy trình – chia sẻ quy trình

Một hệ thống có thể chứa các quy trình độc lập hoặc hợp tác với nhau.

* Một quy trình độc lập nếu nó không thể ảnh hưởng hoặc bị ảnh hưởng bởi các quy trình khác đang chạy trong hệ thống. Bất kỳ quy trình nào không chia sẻ dữ liệu với bất kỳ quy trình nào khác đều là độc lập.
* Một quy trình hợp tác nếu nó có thể ảnh hưởng hoặc bị ảnh hưởng bởi các quy trình khác đang chạy trong hệ thống.

Có một số lý do để cung cấp một môi trường cho phép các quy trình hợp tác:

* Chia sẻ thông tin. Vì nhiều người dùng có thể quan tâm đến cùng một thông tin (một tập tin chia sẻ...)
* Tăng tốc tính toán. Nếu chúng ta muốn một nhiệm vụ cụ thể chạy nhanh hơn, chúng ta phải chia nó thành các tác vụ con, mỗi tác vụ sẽ được thực thi song song với các tác vụ khác.
* Tính mô-đun. Để xây dựng hệ thống theo kiểu mô-đun - chia các chức năng của hệ thống thành các quy trình hoặc luồng riêng biệt.
* Tiện lợi. Người dùng có thể làm việc trên nhiều nhiệm vụ cùng một lúc.

Các quy trình hợp tác yêu cầu một cơ chế giao tiếp giữa quy trình (IPC) cho phép chúng trao đổi dữ liệu và thông tin.

Có hai mô hình IPC cơ bản:

* Bộ nhớ chia sẻ: Cho phép tốc độ và tiện lợi tối đa của giao tiếp, vì nó có thể được thực hiện ở tốc độ bộ nhớ khi trong một máy tính.
* Truyền tin nhắn: Hữu ích để trao đổi số lượng dữ liệu nhỏ hơn, vì không cần tránh xung đột. Truyền tin nhắn cũng dễ dàng hơn để thực hiện hơn là bộ nhớ chia sẻ cho việc giao tiếp giữa các máy tính.

### Hệ thống bộ nhớ dùng chung – chia sẻ bộ nhớ giữa các quy trình

Truyền thông giữa các quy trình sử dụng bộ nhớ chung yêu cầu các quy trình giao tiếp thiết lập một khu vực bộ nhớ chung.

Các quy trình khác muốn giao tiếp bằng cách sử dụng phân đoạn bộ nhớ chung này phải đính kèm nó vào không gian địa chỉ của chúng.

Nhớ rằng, thường thì hệ điều hành sẽ cố gắng ngăn cản một quy trình truy cập vào bộ nhớ của một quy trình khác.

Bộ nhớ chia sẻ yêu cầu hai hoặc nhiều quy trình đồng ý với nhau.

Sau đó, các quy trình này có thể trao đổi thông tin bằng cách đọc và ghi dữ liệu trong các khu vực được chia sẻ. Dạng dữ liệu và vị trí được xác định bởi các quy trình này và không nằm trong sự kiểm soát của hệ điều hành.

Các quy trình cũng chịu trách nhiệm đảm bảo rằng họ không ghi vào cùng một vị trí đồng thời.

Vấn đề sản xuất và tiêu thụ (producer-consumer problem) là một mô hình phổ biến cho các quy trình hợp tác. Một quy trình sản xuất (producer process) tạo ra thông tin được tiêu thụ bởi một quy trình tiêu thụ (consumer process).

### Hệ thống truyền tin nhắn – Message-Passing

Message passing là cơ chế cung cấp cho phép các quy trình giao tiếp và đồng bộ hóa hành động của chúng mà không cần chia sẻ cùng một không gian địa chỉ.

Cơ chế này đặc biệt hữu ích trong môi trường phân tán, trong đó các quy trình giao tiếp có thể đặt trên các máy tính khác nhau được kết nối bằng một mạng.

Một cơ sở hạ tầng truyền tin có thể cung cấp ít nhất hai hoạt động:

* gửi (send) tin nhắn: kích thước của tin nhắn có thể cố định hoặc thay đổi.
* nhận (receive) tin nhắn.

Đặc tính của liên kết giao tiếp

* Liên kết được thiết lập tự động.
* Mỗi liên kết được liên kết với đúng một cặp quy trình đang giao tiếp.
* Giữa mỗi cặp, chỉ tồn tại duy nhất một liên kết.
* Liên kết có thể là một chiều, nhưng thường là hai chiều.

Indirect communication trong đó các tin nhắn được gửi đến và nhận từ các hộp thư hoặc cổng. Một hộp thư có thể được xem trừu tượng như một đối tượng mà các tin nhắn có thể được đặt vào bởi các quy trình và có thể được lấy ra từ đó.

### Sockets

Một socket được định nghĩa là một điểm cuối cho giao tiếp.

Một cặp quy trình giao tiếp trên mạng sử dụng một cặp socket - một cho mỗi quy trình.

Một socket được xác định bằng một địa chỉ IP ghép với một số cổng.

Máy chủ chờ các yêu cầu khách hàng đến bằng cách lắng nghe tại một cổng được chỉ định. Khi một yêu cầu được nhận, máy chủ chấp nhận một kết nối từ socket khách hàng để hoàn tất kết nối.

### Các cuộc gọi thủ tục từ xa - Remote Procedure Calls (RPC)

RPC (Remote Procedure Call) được thiết kế nhằm trừu tượng hóa cơ chế gọi thủ tục để sử dụng giữa các hệ thống kết nối mạng.

Các ý nghĩa của RPC cho phép một khách hàng gọi một thủ tục trên một máy chủ từ xa như là khách hàng gọi một thủ tục cục bộ.

Hệ thống RPC ẩn các chi tiết cho phép truyền thông diễn ra bằng cách cung cấp một đoạn mã stub trên phía khách hàng.

Stubs - là một proxy phía khách hàng cho thủ tục thực sự trên máy chủ.

Khi client gọi một thủ tục từ xa, hệ thống RPC gọi stub phù hợp, chuyền cho nó các tham số cung cấp cho thủ tục từ xa.

Stub này tìm đến cổng trên máy chủ và đóng gói các tham số thành dạng có thể truyền qua mạng. Đóng gói tham số gọi là marshalling tham số.

Một stub tương tự trên phía máy chủ nhận thông điệp này và gọi thủ tục trên máy chủ. Nếu cần, giá trị trả về được truyền trở lại cho client bằng cách sử dụng kỹ thuật tương tự

# Chương 4: Threads

### Tổng quan

Một luồng (thread) là đơn vị cơ bản của sự tận dụng CPU.

Mỗi luồng bao gồm:

* Một ID luồng
* Một bộ đếm chương trình (program counter)
* Một bộ thanh ghi (register set)
* Một ngăn xếp (stack)

Luồng chia sẻ với các luồng khác thuộc cùng một quy trình cùng một phần mã, phần dữ liệu và các tài nguyên hệ điều hành khác, chẳng hạn như các tập tin mở và các tín hiệu (signals).

Một quy trình truyền thống (hoặc nặng) có một luồng điều khiển đơn.

Nếu một quy trình có nhiều luồng điều khiển, nó có thể thực hiện nhiều tác vụ cùng một lúc.

### Lợi ích

Đáp ứng (Responsiveness). Đa luồng một ứng dụng tương tác có thể cho phép chương trình tiếp tục chạy ngay cả khi một phần của nó bị chặn hoặc đang thực hiện một hoạt động lâu dài, từ đó tăng tính đáp ứng đối với người dùng.

Chia sẻ tài nguyên. Lợi ích của việc chia sẻ mã và dữ liệu là cho phép ứng dụng có nhiều luồng hoạt động khác nhau trong cùng một không gian địa chỉ.

Tính kinh tế. Vì các luồng chia sẻ tài nguyên của quy trình mà chúng thuộc về, việc tạo và chuyển đổi ngữ cảnh giữa các luồng là hiệu quả hơn.

Tận dụng kiến trúc đa bộ vi xử lý. Trong một kiến trúc đa bộ vi xử lý, nơi các luồng có thể chạy song song trên các bộ xử lý khác nhau. Một quy trình đơn luồng chỉ có thể chạy trên một CPU. Đa luồng trên một máy đa CPU tăng tính đồng thời.

### Mô hình đa luồng

Có thể cung cấp các luồng ở cấp độ người dùng, cho các luồng người dùng, hoặc bởi nhân hệ điều hành, cho các luồng nhân hệ điều hành.

Phải tồn tại một mối quan hệ giữa các luồng người dùng và luồng nhân hệ điều hành.

### Mô hình Many – to – one

Ánh xạ nhiều luồng cấp người dùng vào một luồng nhân hệ điều hành.

Quản lý luồng được thực hiện bởi thư viện luồng ở không gian người dùng, vì vậy nó rất hiệu quả.

Tuy nhiên, toàn bộ quá trình sẽ bị chặn nếu một luồng thực hiện một cuộc gọi hệ thống chặn.

Vì chỉ có một luồng có thể truy cập vào nhân hệ điều hành tại một thời điểm, nhiều luồng không thể chạy song song trên các bộ xử lý đa lõi.

Ví dụ. Green thread - một thư viện luồng có sẵn cho Solaris - sử dụng mô hình này, cũng như GNU Portable Threads.

### Mô hình One – to – one

Ánh xạ mỗi luồng người dùng vào một luồng nhân hệ điều hành.

Cung cấp nhiều đồng thời hơn so với mô hình nhiều-đến-một bằng cách cho phép một luồng khác chạy khi một luồng thực hiện một cuộc gọi hệ thống chặn.

Cho phép nhiều luồng chạy song song trên các bộ xử lý đa lõi.

Hầu hết các phiên bản của mô hình này giới hạn số lượng luồng được hỗ trợ bởi hệ thống.

Ví dụ. Linux, cùng với họ các hệ điều hành Windows bao gồm Windows 95, 98, NT, 2000 và XP thực hiện mô hình một-đến-một.

### Mô hình Many – to – many

Mô hình nhiều-đến-nhiều đa luồng nhiều luồng cấp người dùng đến một số luồng nhân hệ điều hành nhỏ hơn hoặc bằng.

Số lượng luồng nhân có thể cụ thể cho một ứng dụng cụ thể hoặc một máy tính cụ thể.

Nhà phát triển có thể tạo ra bao nhiêu luồng người dùng cần thiết, và các luồng nhân tương ứng có thể chạy song song trên một bộ xử lý đa lõi.

Khi một luồng thực hiện một cuộc gọi hệ thống chặn, hệ điều hành có thể lên lịch cho một luồng khác để thực thi.

Ví dụ. Solaris trước phiên bản 9.

Một biến thể phổ biến trên mô hình nhiều-đến-nhiều vẫn cho phép đa luồng cấp người dùng đến một số nhỏ hoặc bằng số lượng luồng nhân cấp. Tuy nhiên, biến thể này còn cho phép một luồng cấp người dùng được ràng buộc với một luồng nhân cấp.

Biến thể này, thỉnh thoảng được gọi là mô hình hai cấp.

Các hệ điều hành IRIX, HP-UX, Tru64 UNIX, Solaris 8 và các phiên bản trước đó của Solaris đều sử dụng mô hình này.

### Thư viện Thread

Một thư viện thread cung cấp cho người lập trình một API (Application Programming Interface) để tạo và quản lý các thread.

Có hai cách chính để thực hiện một thư viện thread.

* Cung cấp một thư viện hoàn toàn ở user space mà không có hỗ trợ từ kernel. Tất cả các mã và cấu trúc dữ liệu cho thư viện tồn tại trong user space. Điều này có nghĩa là gọi một hàm trong thư viện dẫn đến một lời gọi hàm cục bộ trong user space và không phải là một system call.
* Cách thực hiện thư viện luồng cấp kernel được hỗ trợ trực tiếp bởi hệ điều hành. Mã và cấu trúc dữ liệu cho thư viện tồn tại trong không gian kernel. Gọi một hàm trong API của thư viện thường dẫn đến một lệnh gọi hệ thống đến kernel.

### Posix Pthreads ((POSIX: Portable Operating System Interface [for Unix])

Pthread có thể được cung cấp dưới dạng user hoặc kernal

POSIX Pthreads là một API thread chuẩn được định nghĩa trong tiêu chuẩn POSIX, và là một trong những thư viện luồng phổ biến nhất trên các hệ điều hành tương thích POSIX, chẳng hạn như Linux và macOS, Solaris, Tru64 Unix.

### Win32 Threads

Kỹ thuật tạo thread bằng thư viện thread Win32 tương tự như kỹ thuật của Pthreads ở một số khía cạnh.

Chú ý:

* Bao gồm tệp tiêu đề windows.h
* Sử dụng hàm CreateThread ()
* Kiểu dữ liệu DWORD là một số nguyên không dấu 32 bit.
* Win32 định nghĩa LPVOID là con trỏ trỏ tới void.

Win32 là một API thread được cung cấp bởi hệ điều hành Windows và hỗ trợ trong các ngôn ngữ lập trình như C và C ++.

### Java Threads

Các thread trong Java được quản lý bởi JVM (Java Virtual Machine).

Tất cả các chương trình Java đều bao gồm ít nhất một thread điều khiển.

Có hai kỹ thuật để tạo thread trong chương trình Java.

* Một cách tiếp cận là tạo một lớp mới được phát sinh từ lớp Thread và ghi đè phương thức run().
* Một cách tiếp cận khác và phổ biến hơn là định nghĩa một lớp thực hiện interface Runnable.

### Các vấn đề

Lệnh gọi hệ thống fork() và exec(): Một quy trình con mới được tạo bằng hàm hệ thống fork(), nếu một luồng gọi hệ thống exec(), chương trình được chỉ định trong tham số của exec() sẽ thay thế toàn bộ quá trình bao gồm tất cả các luồng.

### Hủy bỏ luồng

Hủy bỏ luồng (Thread cancellation) là nhiệm vụ chấm dứt một luồng trước khi nó hoàn thành.

Một luồng mà sẽ bị hủy bỏ thường được gọi là luồng mục tiêu (target thread). Hủy bỏ của một luồng mục tiêu có thể xảy ra trong hai kịch bản khác nhau:

* Hủy bỏ không đồng bộ (Asynchronous cancellation): Một luồng ngay lập tức chấm dứt luồng mục tiêu.
* Hủy bỏ trì hoãn (Deferred cancellation): Luồng mục tiêu định kỳ kiểm tra xem liệu nó có nên kết thúc hay không, cho phép nó có cơ hội kết thúc một cách trật tự.

Khó khăn xảy ra khi huỷ một luồng xảy ra trong các tình huống mà nguồn tài nguyên đã được cấp phát cho một luồng đã bị hủy hoặc khi một luồng bị hủy khi đang cập nhật dữ liệu nó đang chia sẻ với các luồng khác.

Thường, với việc huỷ bất đồng bộ, hệ điều hành sẽ thu hồi các tài nguyên hệ thống từ luồng bị hủy, nhưng sẽ không thu hồi tất cả các tài nguyên.

Do đó, việc huỷ một luồng bất đồng bộ có thể không giải phóng nguồn tài nguyên cần thiết cho toàn hệ thống.

### Xử lý tín hiệu

Một tín hiệu được sử dụng trong các hệ thống UNIX để thông báo cho một quy trình rằng một sự kiện cụ thể đã xảy ra.

Một tín hiệu có thể được nhận theo cách đồng bộ hoặc bất đồng bộ, phụ thuộc vào nguồn và lý do của sự kiện đang được tín hiệu.

* Tín hiệu đồng bộ được gửi đến chính quy trình thực hiện thao tác gây ra tín hiệu. Ví dụ: truy cập bộ nhớ không hợp lệ và phép chia cho 0.
* Một tín hiệu bất đồng bộ được tạo ra bởi một sự kiện bên ngoài một quy trình đang chạy, quy trình đó nhận tín hiệu bất đồng bộ và tín hiệu được gửi đến một quy trình khác.

Mỗi tín hiệu có một trình xử lý tín hiệu mặc định được chạy bởi kernel. Hành động mặc định này có thể được ghi đè bằng một trình xử lý tín hiệu được xác định bởi người dùng được gọi để xử lý tín hiệu.

Một tín hiệu có thể được gửi theo các tùy chọn sau:

* Gửi tín hiệu đến luồng mà tín hiệu áp dụng.
* Gửi tín hiệu đến mỗi luồng trong quá trình.
* Gửi tín hiệu đến một số luồng nhất định trong quá trình.
* Chỉ định một luồng cụ thể để nhận tất cả các tín hiệu cho quá trình.

Các tín hiệu có thể được xử lý theo nhiều cách khác nhau. Một số tín hiệu (như thay đổi kích thước cửa sổ) có thể bị bỏ qua đơn giản; các tín hiệu khác (như truy cập bộ nhớ bất hợp pháp) có thể được xử lý bằng cách kết thúc chương trình.

### Thread pools

Ý tưởng chung đằng sau một thread pool là tạo ra một số lượng các thread khi quá trình khởi động và đặt chúng vào một bể, nơi chúng đợi và chờ đợi công việc.

Các thread pool cung cấp những lợi ích sau:

* Phục vụ một yêu cầu bằng một thread hiện có thường nhanh hơn so với chờ đợi tạo một thread.
* Một thread pool giới hạn số lượng thread tồn tại tại bất kỳ thời điểm nào. Điều này đặc biệt quan trọng trên các hệ thống không thể hỗ trợ một số lượng lớn các thread đồng thời.

# Chương 5: Lập lịch CPU

### Khái niệm

Một quá trình được thực thi cho đến khi nó phải đợi, thường là đợi cho đến khi một yêu cầu I/O nào đó được hoàn thành. Sau đó, CPU chỉ đơn giản là rảnh rỗi. Thời gian chờ đợi này là lãng phí.

Mục tiêu của multiprogramming là có ít nhất một quá trình đang chạy vào mọi thời điểm, để tối đa hóa việc sử dụng CPU. Khi một quá trình phải đợi, hệ điều hành lấy CPU khỏi quá trình đó và chuyển CPU sang một quá trình khác.

Lập lịch kiểu này là một chức năng cơ bản của hệ điều hành.

### Chu kỳ CPU-I/O Brust

Thành công của lập lịch CPU phụ thuộc vào tính chất quan sát được của các quá trình:

* Thực thi quá trình bao gồm một chu trình của việc thực thi CPU và chờ đợi I/O.
* Các quá trình luân phiên giữa hai trạng thái này.
* Thực thi quá trình bắt đầu bằng một CPU burst. Sau đó là một I/O burst, tiếp theo là một CPU burst khác, rồi lại là một I/O burst và cứ như vậy.
* Cuối cùng, CPU burst cuối cùng kết thúc với yêu cầu hệ thống để kết thúc thực thi.

### Bộ lập lịch CPU – CPU Scheduler

Khi CPU trở thành trạng thái rảnh rỗi, hệ điều hành phải chọn một trong các quá trình trong hàng đợi sẵn sàng để thực thi.

Quá trình lựa chọn được thực hiện bởi lập lịch ngắn hạn – short-term schedule (hoặc lập lịch CPU).

Quyết định lập lịch CPU có thể xảy ra trong bốn trường hợp sau:

* Khi một quá trình chuyển từ trạng thái đang chạy sang trạng thái đợi (ví dụ như kết quả của yêu cầu I/O hoặc gọi wait để chờ kết thúc một trong các quá trình con).
* Khi một quá trình chuyển từ trạng thái đang chạy sang trạng thái sẵn sàng (ví dụ như khi xảy ra một ngắt).
* Khi một quá trình chuyển từ trạng thái đợi sang trạng thái sẵn sàng (ví dụ như khi hoàn thành I/O).
* Khi một quá trình kết thúc.

Khi lập lịch chỉ xảy ra trong các trường hợp 1 và 4, chúng ta nói rằng chế độ lập lịch là không ưu tiên (nonpreemptive) hoặc hợp tác (preemptive).

2 và 3 là ưu tiên (preemptive).

### Lập lịch không ưu tiên - Nonpreemptive scheduling

Sau khi CPU được cấp phát cho một quá trình, quá trình giữ CPU cho đến khi nó giải phóng CPU bằng cách kết thúc hoặc chuyển sang trạng thái đợi.

Phương pháp lập lịch này đã được sử dụng bởi Microsoft Windows 3.x và Apple Macintosh.

Phương pháp lập lịch này là phương pháp duy nhất có thể được sử dụng trên một số nền tảng phần cứng, vì nó không yêu cầu phần cứng đặc biệt.

### Lập lịch ưu tiên - Preemptive scheduling

Các process sẽ tranh nhau sử dụng CPU

Việc truy cập vào dữ liệu chia sẻ có thể tạo ra chi phí liên quan đến việc truy cập này, hệ điều hành cần cung cấp các cơ chế mới để phối hợp truy cập vào dữ liệu chia sẻ.

Phương pháp này cũng ảnh hưởng đến thiết kế của nhân hệ điều hành.

Phương pháp này đã được sử dụng bởi hầu hết các phiên bản của UNIX.

### Điều phối – Dispatcher

Bộ điều phối (dispatcher) là mô-đun cung cấp điều khiển của CPU cho quá trình được chọn bởi trình lập lịch ngắn hạn. Chức năng này bao gồm các hoạt động sau:

* Chuyển đổi ngữ cảnh
* Chuyển sang chế độ người dùng
* Nhảy đến vị trí phù hợp trong chương trình người dùng để khởi động lại chương trình đó

Bộ điều phối nên nhanh nhất có thể, vì nó được triệu hồi trong mỗi lần chuyển đổi quá trình. Thời gian mất cho bộ điều phối để dừng một quá trình và khởi chạy quá trình khác được gọi là độ trễ điều phối (dispatch latency).

### Tiêu chí lập kế hoạch - Scheduling Criteria

Các thuật toán lập lịch CPU khác nhau có các đặc tính khác nhau và việc lựa chọn một thuật toán cụ thể có thể ưu tiên một lớp quá trình hơn lớp khác.

Các tiêu chí bao gồm như sau:

* Sử dụng CPU. Chúng ta muốn giữ cho CPU bận rộn nhất có thể. Về mặt khái niệm, sử dụng CPU có thể dao động từ 0 đến 100 phần trăm. Trong một hệ thống thực tế, nó nên dao động từ 40 phần trăm (đối với một hệ thống tải nhẹ) đến 90 phần trăm (đối với một hệ thống được sử dụng nhiều).
* Thống lượng. Đây là đo lường công việc là số quá trình hoàn thành trong mỗi đơn vị thời gian.
* Thời gian hoàn thành. Khoảng thời gian từ thời điểm nộp quá trình cho đến thời điểm hoàn thành. Thời gian hoàn thành là tổng của các giai đoạn đã chờ để vào bộ nhớ, chờ trong hàng đợi sẵn sàng, thực thi trên CPU và thực hiện I / O.
* Waiting time. Thời gian (tổng của các giai đoạn) mà một quy trình phải chờ đợi trong hàng đợi sẵn sàng.
* Response time. Thời gian từ khi yêu cầu được gửi đi cho đến khi phản hồi đầu tiên được tạo ra (không phải thời gian để đưa ra phản hồi).

Trong hầu hết các trường hợp, chúng ta tối ưu hóa giá trị trung bình. Tuy nhiên, trong một số trường hợp, tối ưu hóa các giá trị tối thiểu hoặc tối đa là mong muốn hơn là tối ưu hóa giá trị trung bình.

### Thuật toán lập lịch đến trước phục vụ trước – FCFS

Thuật toán lập lịch là không chấp nhận gián đoạn (nonpreemptive)

Quá trình yêu cầu CPU đầu tiên được phân bổ CPU đầu tiên.

Việc thực hiện chính sách FCFS có thể dễ dàng quản lý với một hàng đợi FIFO. Khi một quy trình vào hàng đợi sẵn sàng, PCB của nó được liên kết vào đuôi của hàng đợi.

Khi CPU rảnh, nó được phân bổ cho quy trình đứng đầu trong hàng đợi. Quy trình đang chạy sau đó được loại bỏ khỏi hàng đợi.

### Thuật toán lập lịch thực hiện công việc nhỏ nhất trước – SJF

Thuật toán này liên kết với mỗi quy trình độ dài của chuỗi CPU burst kế tiếp của quy trình đó. Khi CPU có sẵn, nó được gán cho quy trình có chuỗi CPU burst tiếp theo nhỏ nhất.

Nếu chuỗi CPU burst tiếp theo của hai quy trình là giống nhau, FCFS được sử dụng để giải quyết xung đột.

Tuy nhiên, khó khăn thực sự của thuật toán SJF là không biết được độ dài của yêu cầu CPU tiếp theo.

Không có cách nào để biết độ dài yêu cầu CPU tiếp theo.

Một cách tiếp cận là cố gắng xấp xỉ lập lịch SJF. Chúng ta có thể không biết độ dài yêu cầu CPU tiếp theo, nhưng có thể dự đoán được giá trị của nó.

Thuật toán SJF có thể được thực hiện dưới dạng preemptive hoặc nonpreemptive. Sự lựa chọn phát sinh khi một quy trình mới đến hàng đợi ready trong khi một quy trình trước đó vẫn đang thực thi.

* Với thuật toán preemptive SJF: Nếu thời gian CPU burst tiếp theo của quy trình mới đến ngắn hơn so với thời gian CPU burst còn lại của quy trình đang thực thi thì quy trình mới sẽ gián đoạn quy trình đang thực thi.
* Với thuật toán nonpreemptive SJF: Sẽ cho phép quy trình đang chạy hoàn thành thời gian CPU burst của nó.

### Lập lịch kế hoạch ưu tiên - Priority Scheduling

Mỗi quá trình được gắn với một số ưu tiên (số nguyên).

CPU được cấp cho quá trình có độ ưu tiên cao nhất (số nguyên nhỏ nhất ≡ độ ưu tiên cao nhất)

Các quá trình có độ ưu tiên bằng nhau được lên lịch theo thứ tự FCFS.

Một thuật toán SJF đơn giản chỉ là một thuật toán ưu tiên trong đó ưu tiên (p) là nghịch đảo của chu kỳ CPU tiếp theo (được dự đoán). Càng lớn chu kỳ CPU, thì ưu tiên càng thấp và ngược lại.

Lập lịch ưu tiên có thể là ưu tiên có khả năng preemptive hoặc nonpreemptive. Khi một quy trình đến hàng đợi sẵn sàng, độ ưu tiên của nó được so sánh với độ ưu tiên của quy trình đang chạy hiện tại.

* Một thuật toán ưu tiên có khả năng gián đoạn sẽ gián đoạn CPU nếu độ ưu tiên của quy trình mới đến cao hơn độ ưu tiên của quy trình đang chạy hiện tại.
* Một thuật toán ưu tiên không gián đoạn đơn giản chỉ đặt quy trình mới đến vào đầu hàng đợi sẵn sàng.

Vấn đề chính của các thuật toán ưu tiên là đóng băng không xác định hoặc starvation ưu tiên: Một thuật toán lập lịch ưu tiên có thể để lại một số quá trình ưu tiên thấp đang chờ đợi mãi mãi, điều đó có nghĩa là các quá trình ưu tiên thấp có thể không bao giờ được thực thi.

Một giải pháp cho vấn đề đóng băng không xác định của các quá trình ưu tiên thấp là tuổi thọ: Một kỹ thuật tăng dần ưu tiên của các quá trình đợi trong hệ thống trong một khoảng thời gian dài.

Ví dụ, nếu các ưu tiên dao động từ 127 (thấp) đến 0 (cao), hệ điều hành có thể tăng độ ưu tiên của một quá trình đang chờ đợi lên 1 mỗi 15 phút.

### Lập lịch kế hoạch quay vòng – RR

Thuật toán lập lịch vòng tròn (Round-robin - RR) được thiết kế đặc biệt cho các hệ thống chia sẻ thời gian.

Mỗi quy trình được cấp một đơn vị thời gian CPU nhỏ (time quantum), thường là từ 10-100 mili giây. Sau khi thời gian này đã trôi qua, quy trình sẽ bị gián đoạn và được thêm vào cuối hàng đợi sẵn sàng.

Hàng đợi sẵn sàng được quản lý dưới dạng một hàng đợi FIFO của các quy trình.

Các quy trình mới được thêm vào cuối hàng đợi sẵn sàng. Bộ lập lịch CPU chọn quy trình đầu tiên từ hàng đợi sẵn sàng, đặt một bộ hẹn giờ để gián đoạn sau một đơn vị thời gian, và thực thi quy trình đó.

### Lập lịch hàng đợi đa cấp - Multilevel Queue Scheduling

Thuật toán lập lịch hàng đợi đa cấp phân chia hàng đợi sẵn sàng thành một số hàng đợi riêng biệt.

Các quy trình được gán một cách cố định cho một hàng đợi, thông thường dựa trên một thuộc tính của quy trình, chẳng hạn như kích thước bộ nhớ, ưu tiên quy trình hoặc loại quy trình.

Mỗi hàng đợi có thuật toán lập lịch riêng của nó.

Một phân chia thông thường được thực hiện giữa

* các quy trình foreground (tương tác)
* các quy trình background (đợt công việc).

### Lập lịch kế hoạch hàng đợi phản hồi đa cấp - Multilevel Feedback-Queue Scheduling

Thuật toán lập lịch hàng đợi phản hồi đa cấp (multilevel feedback-queue) cho phép một quy trình di chuyển giữa các hàng đợi.

Ý tưởng của thuật toán này là tách các quy trình theo đặc điểm của chuỗi thời gian CPU của chúng. Nếu một quy trình sử dụng quá nhiều thời gian CPU, nó sẽ được chuyển đến một hàng đợi ưu tiên thấp hơn.

Hệ thống này giữ các quy trình I/O-bound và tương tác trong các hàng đợi ưu tiên cao hơn.

Một quy trình đợi quá lâu trong một hàng đợi ưu tiên thấp có thể được chuyển đến một hàng đợi ưu tiên cao hơn. Hình thức lão hóa này ngăn chặn đói đến.

Một bộ lập lịch multilevel feedback-queue được định nghĩa bởi các thông số sau:

* Số lượng hàng đợi
* Thuật toán lập lịch cho mỗi hàng đợi
* Phương pháp sử dụng để xác định khi nào quy trình sẽ được nâng cấp lên hàng đợi ưu tiên cao hơn
* Phương pháp sử dụng để xác định khi nào quy trình sẽ bị giảm độ ưu tiên xuống hàng đợi ưu tiên thấp hơn
* Phương pháp sử dụng để xác định quy trình sẽ vào hàng đợi nào khi cần phục vụ.

### Lập lịch đa bộ xử lý - Multiple-Processor Scheduling

Đối với lập lịch đa bộ xử lý, nếu có nhiều CPU có sẵn, thì việc chia sẻ tải trọng trở nên khả thi; tuy nhiên, vấn đề lập lịch cũng trở nên phức tạp tương ứng.

Có nhiều phương pháp đã được thử; và như chúng ta đã thấy với lập lịch CPU đơn, không có một giải pháp tốt nhất.

Một số vấn đề cần quan tâm trong lập lịch đa bộ xử lý bao gồm:

* Các bộ xử lý trong một hệ thống đa bộ xử lý phải đồng nhất về chức năng của chúng.
* Asymmetric multiprocessing (AMP) là hệ thống có một bộ xử lý (CPU) truy cập các cấu trúc dữ liệu của hệ thống, giảm thiểu nhu cầu chia sẻ dữ liệu.
* Symmetric multiprocessing (SMP) - mỗi bộ xử lý đều tự lên lịch, tất cả các quy trình đều có trong hàng đợi chung, hoặc mỗi bộ xử lý có hàng đợi riêng của các quy trình đã sẵn sàng.
* Processor affinity - quy trình có liên kết với bộ xử lý mà nó đang chạy trên đó.
* Cân bằng tải cố gắng giữ cho tải công việc được phân bổ đồng đều trên tất cả các bộ xử lý trong hệ thống SMP.

### Đánh giá thuật toán

Một số tiêu chí có thể bao gồm:

* Sử dụng CPU (CPU utilization)
* Thời gian đáp ứng (Response time)
* Thông lượng (Throughput)

Sau khi các tiêu chí lựa chọn đã được định nghĩa, chúng ta muốn đánh giá các thuật toán đang được xem xét. Tiếp theo, chúng ta mô tả các phương pháp đánh giá khác nhau mà chúng ta có thể sử dụng.

Phương pháp đánh giá chính là phân tích.

Phân tích sử dụng giải thuật và khối lượng công việc hệ thống để tạo ra công thức hoặc số đánh giá hiệu suất của giải thuật đó cho khối lượng công việc đó.

Một loại phân tích xác định là mô hình xác định.

Phương pháp này lấy một khối lượng công việc xác định trước cụ thể và xác định hiệu suất của mỗi giải thuật cho khối lượng công việc đó.

# Chương 6: Đồng bộ hóa quy trình - Process Synchronization

### Tình huống cạnh tranh – race condition

Race condition là tình trạng xảy ra khi nhiều quy trình hoặc luồng thực thi trong chương trình cùng truy cập và thay đổi dữ liệu chung một cách đồng thời và không có sự điều khiển đúng đắn nào, dẫn đến kết quả của chương trình không đáng tin cậy và không thể dự đoán trước được. Các vấn đề liên quan đến race condition thường xảy ra trong các chương trình đa luồng hoặc trong các ứng dụng đồng thời.

### Vấn đề phần quan trọng - The Critical-Section Problem

Mỗi quy trình có một đoạn mã, gọi là phần giao critical section, trong đó quy trình có thể thay đổi các biến chung, cập nhật bảng, ghi tệp và nhiều hơn nữa.

Mỗi quy trình phải yêu cầu cho phép để vào critical section của nó. Phần mã thực hiện yêu cầu này được gọi là entry section.

Critical section có thể được theo sau bởi một phần exit section. Phần mã còn lại là phần remainder section.

Một giải pháp cho bài toán critical-section phải đáp ứng ba yêu cầu sau:

* Mutual exclusion (Đồng quyền): Nếu quá trình Pi đang thực thi trong critical section của nó, thì không có quá trình nào khác có thể thực thi trong critical section của chúng.
* Progress (Quy trình): Nếu không có quá trình nào đang thực thi trong critical section của nó và một số quá trình muốn vào critical section của chúng, thì chỉ có những quá trình đang không thực thi trong phần còn lại của chúng mới có thể tham gia vào quyết định về việc quá trình nào sẽ vào critical section của mình tiếp theo, và việc lựa chọn này không thể được trì hoãn vô thời hạn.
* Bounded waiting (Giới hạn thời gian chờ đợi): Tồn tại một giới hạn, hoặc giới hạn, về số lần mà các quá trình khác được phép vào critical section của họ sau khi một quá trình đã yêu cầu vào critical section của mình và trước khi yêu cầu đó được chấp nhận.

Có hai phương pháp chung được sử dụng để xử lý các phần tương đối nhạy cảm trong các hệ điều hành:

* preemptive kernels
* nonpreemptive kernels.

Một kernel preemptive

* Phù hợp hơn cho lập trình thời gian thực, vì nó sẽ cho phép một quy trình thời gian thực gián đoạn một quy trình đang chạy trong kernel.
* Có thể phản hồi nhanh hơn, vì có ít rủi ro hơn rằng một quy trình kernel-mode sẽ chạy trong một khoảng thời gian bất kỳ trước khi nhường bộ xử lý cho các quy trình đang chờ đợi.
* Kernel nonpreemptive: Windows 2000/XP và kernel UNIX truyền thống, trước Linux 2.6.
* Kernel preemptive: phiên bản 2.6 kernel được phát hành, Một số phiên bản thương mại của UNIX bao gồm Solaris và IRIX.

Nonpreemptive kernels không chặn về cơ bản không bị ảnh hưởng bởi các điều kiện cạnh tranh trên các cấu trúc dữ liệu nhân.

### Giải pháp của Peterson

Giải pháp của Peterson là một phương pháp đồng bộ hóa quy trình được sử dụng để giải quyết vấn đề khu vực g critical-section trong các hệ thống đa quy trình. Phương pháp này chỉ áp dụng được trong trường hợp hai quy trình thay phiên nhau thực thi giữa các khu vực g critical-section và phần còn lại của chúng.

Giải pháp của Peterson sử dụng hai biến shared (được chia sẻ) để điều khiển việc truy cập vào critical-section của mỗi quy trình. Mỗi quy trình có một biến flag để cho biết nó có ý định vào critical-section hay không. Nếu quy trình P0 muốn vào critical-section, nó đặt flag[0] = true, và set turn = 1, đồng thời kiểm tra xem quy trình P1 có muốn vào critical-section hay không và đang nằm trong phần còn lại của nó hay không. Nếu không, quy trình P0 được phép vào critical-section. Nếu có, quy trình P0 phải chờ cho đến khi quy trình P1 đã vào hoàn thành critical-section của nó. Tương tự với quy trình P1, nếu nó muốn vào critical-section, nó đặt flag[1] = true và set turn = 0, và sau đó kiểm tra xem quy trình P0 có muốn vào hay không.

Phương pháp của Peterson đảm bảo tính đối xứng, an toàn và deadlock-free. Tuy nhiên, giải pháp này có một số hạn chế, bao gồm chỉ áp dụng được trong trường hợp hai quy trình thay phiên nhau thực thi, không thể mở rộng để áp dụng cho nhiều quy trình hơn và độ trễ sẽ tăng khi thực hiện nhiều lần kiểm tra.

### Giải pháp của phần cứng - Synchronization Hardware

Chúng ta có thể khẳng định rằng bất kỳ giải pháp nào cho vấn đề critical section đều yêu cầu một công cụ đơn giản - một khóa.

Race conditions được ngăn chặn bằng cách yêu cầu các vùng critical được bảo vệ bằng khóa. Tức là, một quy trình phải lấy được khóa trước khi vào critical section; nó giải phóng khóa khi thoát khỏi vùng critical.

Thật không may, giải pháp này không khả thi trong môi trường đa bộ vi xử lý.

Các máy hiện đại cung cấp các chỉ thị phần cứng atomic đặc biệt

Atomic = không thể bị ngắt

Hoặc kiểm tra từ bộ nhớ và thiết lập giá trị

Hoặc trao đổi nội dung của hai từ bộ nhớ

### Giải pháp Semaphores

Semaphore là một công cụ đồng bộ hóa trong lập trình, nó được sử dụng để giải quyết các vấn đề đồng thời trong quản lý tài nguyên giữa các quy trình. Semaphore là một biến nguyên, được truy cập chỉ thông qua hai phép toán nguyên tử chuẩn: wait() và signal(), tương đương với acquire() và release(). Ban đầu, chúng được gọi là P() và V(). Semaphore có thể giúp đảm bảo rằng chỉ có một quy trình được phép truy cập vào tài nguyên chung tại một thời điểm, giải quyết vấn đề đối đầu (race condition).

Semaphore hỗ trợ hai phép toán chính:

* wait() (hay còn gọi là acquire()): giảm giá trị của S đi một đơn vị. Nếu giá trị mới của S là một số âm, quy trình đang thực thi sẽ bị block.
* signal() (hay còn gọi là release()): tăng giá trị của S lên một đơn vị. Nếu có quy trình khác đang bị block trên thao tác wait(), phép toán signal() sẽ thức tỉnh một quy trình trong số đó.

Các phép toán wait() và signal() được thực hiện theo nguyên tắc nguyên tố (atomic), nghĩa là nó sẽ không bị gián đoạn bởi các quy trình khác, đảm bảo tính đồng bộ và độ tin cậy của chúng.

Semaphore có thể được sử dụng để đồng bộ hóa việc truy cập vào tài nguyên chia sẻ, thực thi đồng thời các quy trình và tránh tình trạng đối đầu (deadlock) hoặc bế tắc (starvation) trong quá trình thực thi các quy trình.

### Deadlocks và Starvation

Khi triển khai semaphore với một hàng đợi chờ, có thể xảy ra trường hợp hai hoặc nhiều quy trình đang đợi vô thời hạn cho một sự kiện chỉ có thể được gây ra bởi một trong các quy trình đang đợi.

Sự kiện được nhắc đến ở đây là thực thi một hoạt động signal().

Khi đạt đến trạng thái này, các quy trình này được gọi là bị mắc kẹt (deadlock).

Vấn đề khác liên quan đến deadlock là chặn vô thời hạn hoặc starvation, một tình huống trong đó các quy trình đợi vô thời hạn trong semaphore.

Chặn vô thời hạn có thể xảy ra nếu chúng ta thêm và xóa các quy trình khỏi danh sách liên quan đến semaphore theo thứ tự LIFO (cuối vào trước ra).

# Chương 7: Deadlock

### Khái niệm

Một quy trình yêu cầu tài nguyên; và nếu tài nguyên không có sẵn vào thời điểm đó, quy trình đó sẽ vào trạng thái chờ đợi.

Đôi khi, một quy trình đang chờ đợi không bao giờ có thể thay đổi trạng thái nữa, bởi vì các tài nguyên mà nó đã yêu cầu đang được giữ bởi các quy trình đang chờ đợi khác. Tình huống này được gọi là một deadlock.

### Mô hình hệ thống

Một hệ thống bao gồm một số tài nguyên hữu hạn được phân phối cho một số quy trình cạnh tranh.

Các tài nguyên được chia thành một số loại, mỗi loại bao gồm một số lượng các trường hợp tương tự nhau.

Ví dụ, không gian bộ nhớ, chu kỳ CPU, tệp và các thiết bị I/O (như máy in và ổ đĩa DVD) là các loại tài nguyên.

Nếu hệ thống có hai CPU, thì loại tài nguyên CPU có hai thể hiện.

Nếu một quy trình yêu cầu một trường hợp của một loại tài nguyên, việc phân bổ bất kỳ trường hợp nào của loại đó sẽ đáp ứng yêu cầu.

Nếu không, thì các trường hợp không giống nhau và các lớp tài nguyên loại đã không được định nghĩa đúng cách.

Một quy trình phải yêu cầu một tài nguyên trước khi sử dụng nó và phải giải phóng tài nguyên sau khi sử dụng nó.

Một quy trình có thể yêu cầu bao nhiêu tài nguyên mà nó cần để thực hiện nhiệm vụ được chỉ định của mình. Tất nhiên, số lượng tài nguyên được yêu cầu không được vượt quá tổng số tài nguyên có sẵn trong hệ thống.

Dưới chế độ hoạt động bình thường, một quy trình chỉ có thể sử dụng một tài nguyên theo trình tự sau:

* Yêu cầu. Nếu yêu cầu không thể được cấp phát ngay lập tức (ví dụ, nếu tài nguyên đang được sử dụng bởi một quy trình khác), thì quy trình yêu cầu phải chờ đợi cho đến khi có thể sử dụng được tài nguyên.
* Sử dụng. Quy trình có thể thao tác trên tài nguyên (ví dụ, nếu tài nguyên là máy in, quy trình có thể in trên máy in).
* Giải phóng. Quy trình giải phóng tài nguyên.

Một tập hợp các quy trình đang ở trạng thái bế tắc khi mỗi quy trình trong tập hợp đang chờ đợi một sự kiện chỉ có thể được gây ra bởi một quy trình khác trong tập hợp.

Các sự kiện mà chúng ta quan tâm chủ yếu ở đây là việc yêu cầu và giải phóng tài nguyên.

Các tài nguyên có thể là tài nguyên vật lý (ví dụ như máy in, đầu đọc băng, không gian bộ nhớ và chu kỳ CPU) hoặc tài nguyên logic (ví dụ như tệp tin, semaphore và màn hình).

### Đặc tính

Trong trường hợp bế tắc, các quy trình không bao giờ kết thúc thực thi và các tài nguyên hệ thống bị liên kết, ngăn cản các công việc khác khởi động.

### Điều kiện xảy ra deadlock

Tình huống bế tắc có thể xảy ra nếu đồng thời tồn tại bốn điều kiện sau đây trong hệ thống:

* Sự đối xứng – loại trừ tương hổ: ít nhất một tài nguyên phải được giữ ở chế độ không chia sẻ; nghĩa là chỉ một quy trình tại một thời điểm có thể sử dụng tài nguyên. Nếu một quy trình khác yêu cầu tài nguyên đó, quy trình yêu cầu phải bị trì hoãn cho đến khi tài nguyên được giải phóng.
* Giữ và chờ: một quy trình phải đang giữ ít nhất một tài nguyên và đang chờ để giành được các tài nguyên bổ sung hiện đang được giữ bởi các quy trình khác.
* Không thể thu hồi: tài nguyên không thể bị thu hồi, tức là một tài nguyên chỉ có thể được giải phóng theo ý muốn của quy trình nắm giữ nó sau khi quy trình đó hoàn thành nhiệm vụ của mình.
* Vòng chờ: Tồn tại một tập hợp {P0, P1, ..., Pn} các quy trình đang chờ đợi sao cho P0 đang chờ đợi một tài nguyên được giữ bởi P1, P1 đang chờ đợi một tài nguyên được giữ bởi P2, ..., Pn-1 đang chờ đợi một tài nguyên được giữ bởi Pn và Pn đang chờ đợi một tài nguyên được giữ bởi P0.

### Mô tả bằng biểu đồ có hướng

Deadlock có thể được mô tả một cách chính xác hơn dưới dạng đồ thị có hướng được gọi là đồ thị phân bổ tài nguyên hệ thống.

Nếu tồn tại chu trình trong đồ thị thì sẽ xảy ra deadlock

### Phương pháp xử lý deadlock

Thông thường, chúng ta (HĐH) có thể giải quyết vấn đề deadlock bằng một trong ba cách sau:

* Chúng ta có thể sử dụng một giao thức để ngăn hoặc tránh deadlock, đảm bảo rằng hệ thống sẽ không bao giờ rơi vào trạng thái deadlock.
* Chúng ta có thể cho phép hệ thống vào trạng thái deadlock, phát hiện nó và khôi phục.
* Chúng ta có thể bỏ qua vấn đề hoàn toàn và giả vờ như không bao giờ có deadlock xảy ra trong hệ thống.

Giải pháp thứ ba là giải pháp được sử dụng bởi hầu hết các hệ điều hành, bao gồm UNIX và Windows.

### Ngăn chặn deadlock

Để xảy ra tình trạng deadlock, cần phải đảm bảo cả bốn điều kiện cần thiết. Bằng cách đảm bảo ít nhất một trong số các điều kiện này không thể xảy ra, chúng ta có thể ngăn chặn sự xảy ra của deadlock.

- Loại trừ lẫn nhau – loại trừ tương hổ:

Điều kiện đồng thời loại trừ phải được áp dụng đối với các tài nguyên không thể chia sẻ.

Trái lại, các tài nguyên có thể chia sẻ không yêu cầu quyền truy cập đồng thời loại trừ và do đó không thể bị mắc kẹt. Một quy trình không bao giờ cần phải đợi một tài nguyên có thể chia sẻ.

Tuy nhiên, nhìn chung, chúng ta không thể ngăn chặn kẹt khóa bằng cách từ chối điều kiện đồng thời loại trừ, vì một số tài nguyên không thể chia sẻ theo bản chất.

- Giữ và chờ:

Để đảm bảo rằng điều kiện giữ và chờ không bao giờ xảy ra trong hệ thống, chúng ta phải đảm bảo rằng, mỗi khi một quy trình yêu cầu một tài nguyên, nó không giữ bất kỳ tài nguyên nào khác.

Một giao thức có thể được sử dụng yêu cầu mỗi quy trình yêu cầu và được cấp phát tất cả các tài nguyên của nó trước khi nó bắt đầu thực thi.

Một giao thức thay thế cho phép một quy trình yêu cầu tài nguyên chỉ khi nó không có tài nguyên nào.

Cả hai giao thức này đều có hai nhược điểm chính.

Sử dụng tài nguyên có thể thấp.

Nguy cơ bị starvasion có thể xảy ra.

- No Preemptive:

Điều kiện cần thiết thứ ba cho kẹt khóa là không có việc giành lại tài nguyên đã được cấp phát trước đó.

Để đảm bảo điều kiện này không xảy ra, chúng ta có thể sử dụng giao thức sau đây.

Nếu một quy trình đang giữ một số tài nguyên và yêu cầu một tài nguyên khác mà không thể được cấp phát ngay lập tức cho nó (tức là quy trình phải chờ đợi), thì tất cả các tài nguyên đang được giữ hiện tại sẽ bị giành lại. Nói cách khác, những tài nguyên này được giải phóng một cách ngầm định.

Các tài nguyên bị giành lại được thêm vào danh sách các tài nguyên mà quy trình đang đợi.

Quy trình sẽ được khởi động lại chỉ khi nó có thể lấy lại các tài nguyên cũ của mình, cũng như những tài nguyên mới mà nó yêu cầu.

- Vòng chờ đợi:

Một cách để đảm bảo rằng điều kiện này không bao giờ xảy ra là áp đặt một thứ tự hoàn toàn của tất cả các loại tài nguyên và yêu cầu mỗi quy trình yêu cầu các tài nguyên theo thứ tự tăng dần của liệt kê.

### Tránh deadlock

Một phương pháp thay thế để tránh kẹt khóa là yêu cầu thông tin bổ sung về cách yêu cầu tài nguyên.

Mô hình đơn giản và hữu ích nhất yêu cầu mỗi quy trình khai báo số lượng tài nguyên tối đa của mỗi loại mà nó có thể cần.

Một thuật toán tránh kẹt khóa động đánh giá trạng thái phân bổ tài nguyên để đảm bảo rằng không bao giờ có điều kiện chặn vòng lặp.

Trạng thái phân bổ tài nguyên được xác định bởi số lượng tài nguyên khả dụng và đã phân bổ và các yêu cầu tối đa của các quy trình.

### Trạng thái an toàn - Safe State

Trạng thái an toàn là một trạng thái của hệ thống trong đó tất cả các quy trình có thể tiếp tục yêu cầu tài nguyên mà chúng cần mà không bị kẹt khóa. Điều này có nghĩa là hệ thống sẽ không bị kẹt khóa vì các quy trình có thể tiếp tục thực thi và hoàn thành công việc của chúng.

Khi hệ thống ở trạng thái an toàn, các quy trình có thể yêu cầu tài nguyên của mình và hệ thống sẽ đáp ứng yêu cầu đó mà không cần chờ đợi mãi mãi. Tuy nhiên, để đạt được trạng thái an toàn, hệ thống cần phải đảm bảo rằng các quy trình không yêu cầu tất cả các tài nguyên cùng một lúc và đảm bảo rằng tài nguyên không bị lãng phí.

Một trạng thái an toàn không phải là một trạng thái bị kẹt. Ngược lại, một trạng thái bị kẹt là một trạng thái không an toàn. Tuy nhiên, không phải tất cả các trạng thái không an toàn đều dẫn đến deadlock, chỉ một trạng thái không an toàn có thể dẫn đến deadlock.

### Thuật toán đồ thị phân bổ tài nguyên

Nếu chúng ta có một hệ thống phân bổ tài nguyên chỉ với một phiên bản của mỗi loại tài nguyên, một biến thể của đồ thị phân bổ tài nguyên có thể được sử dụng để tránh deadlock.

Ngoài các cạnh yêu cầu và phân bổ đã được mô tả, chúng ta giới thiệu một loại cạnh mới, được gọi là cạnh yêu cầu tài nguyên trong tương lai.

Một cạnh yêu cầu tài nguyên trong tương lai Pi -> Rj cho biết quá trình Pi có thể yêu cầu tài nguyên Rj vào một thời điểm nào đó trong tương lai. Cạnh này giống như một cạnh yêu cầu theo hướng nhưng được biểu diễn trong đồ thị bằng một đường kẻ đứt.

Đảm bảo rằng trong tương lai đồ thị sẽ không có chu trình thì như vậy sẽ ngăn chặn được deadlock xảy ra

### Thuật toán ngân hàng - Banker's Algorithm

Tên của giải thuật được chọn vì nó có thể được sử dụng trong hệ thống ngân hàng để đảm bảo rằng ngân hàng không bao giờ cấp phát số tiền khả dụng của mình một cách sao cho nó không còn đáp ứng được nhu cầu của tất cả khách hàng của nó.

Khi một quy trình mới nhập vào hệ thống, nó phải khai báo số lượng tối đa của mỗi loại tài nguyên mà nó có thể cần. Số lượng này không được vượt quá tổng số tài nguyên trong hệ thống.

Khi người dùng yêu cầu một tập các tài nguyên, hệ thống phải xác định liệu việc phân bổ các tài nguyên này có để lại trạng thái an toàn cho hệ thống hay không.

* Nếu có, tài nguyên sẽ được phân bổ.
* Ngược lại, quá trình sẽ phải chờ đợi cho đến khi một quá trình khác giải phóng đủ tài nguyên.

Khi một quá trình nhận được tất cả các tài nguyên của nó, nó phải trả lại chúng trong một khoảng thời gian hữu hạn.

### Thuật toán yêu cầu thêm tài nguyên

Sau khi đã có tài nguyên ps yêu cầu thêm một số tài nguyên nhất định và ta phải dùng các thuật toán để xử lý xem có thể cấp được không

Ý tưởng thuật toán cũng gần giống thuật toán ngân hàng

### Phát hiện Deadlock

Nếu một hệ thống không sử dụng thuật toán ngăn chặn hoặc tránh deadlock, thì tình huống deadlock có thể xảy ra. Trong môi trường này, hệ thống phải cung cấp:

* Một thuật toán kiểm tra trạng thái của hệ thống để xác định xem có deadlock xảy ra hay không.
* Một thuật toán khôi phục khỏi deadlock.

Nếu mỗi loại tài nguyên chỉ có một thể hiện, ta có thể loại bỏ nó khỏi đồ thì và từ đó xét chu trình trên có ps còn tồn tại

### Phục hồi từ deadlock

Có hai lựa chọn để phá vỡ một trạng thái mắc kẹt.

* Một cách đơn giản để phá vỡ vòng lặp chờ đợi là bỏ hủy một hoặc nhiều quy trình.
* Cách khác là thu hồi một số tài nguyên từ một hoặc nhiều quy trình mắc kẹt.

Hủy bỏ quy trình bị mắc kẹt:

* Huỷ bỏ tất cả các quy trình bị mắc kẹt. Phương pháp này rõ ràng sẽ phá vỡ chu trình mắc kẹt, nhưng chi phí lớn; các quy trình bị mắc kẹt có thể tính toán trong thời gian dài, và kết quả của các tính toán phần này phải được loại bỏ và có thể phải được tính toán lại sau đó.
* Huỷ bỏ từng quy trình một cho đến khi chu trình mắc kẹt được loại bỏ. Phương pháp này gây ra chi phí đáng kể, vì sau mỗi quy trình bị huỷ bỏ, một thuật toán phát hiện mắc kẹt phải được gọi để xác định liệu có quy trình nào vẫn bị mắc kẹt hay không.

### Ưu tiên tài nguyên

Nếu giải phóng tài nguyên là cần thiết để xử lý deadlock, thì cần giải quyết ba vấn đề sau:

* Lựa chọn nạn nhân - tối thiểu hóa chi phí
* Rollback - trở lại trạng thái an toàn nào đó, khởi động lại quy trình cho trạng thái đó
* Nạn đói - cùng một quy trình có thể luôn được chọn làm nạn nhân, bao gồm số lần rollback trong yếu tố chi phí.

# Chương 8: Quản lý I/O

### Tổng quan

Các yếu tố phần cứng I/O cơ bản, như cổng, bus và bộ điều khiển thiết bị

Để đóng gói các chi tiết và đặc tính kỳ quặc của các thiết bị khác nhau, nhân của một hệ điều hành được cấu trúc để sử dụng các mô-đun trình điều khiển thiết bị.

Các trình điều khiển thiết bị cung cấp một giao diện truy cập thiết bị đồng nhất cho hệ thống I/O, giống như các lệnh hệ thống cung cấp một giao diện chuẩn giữa ứng dụng và hệ điều hành.

### Phần cứng I/O

Máy tính vận hành rất nhiều loại thiết bị. Hầu hết thuộc vào các danh mục chung của thiết bị lưu trữ (đĩa, băng), thiết bị truyền tải (thẻ mạng, modem) và thiết bị giao tiếp với người dùng (màn hình, bàn phím, chuột).

Thiết bị giao tiếp với máy tính thông qua một điểm kết nối (hoặc cổng) - ví dụ: cổng nối tiếp.

Bus là một tập hợp các dây và giao thức được định nghĩa cứng rắn, chỉ định một tập hợp các thông điệp có thể được gửi trên các dây.

Bộ điều khiển là một bộ sưu tập của các linh kiện điện tử có thể vận hành một cổng, một bus hoặc một thiết bị.

Làm thế nào bộ xử lý có thể gửi lệnh và dữ liệu đến bộ điều khiển để thực hiện một truyền thông I/O? Câu trả lời ngắn là bộ điều khiển có một hoặc nhiều thanh ghi cho dữ liệu và tín hiệu điều khiển. Bộ xử lý liên lạc với bộ điều khiển bằng cách đọc và ghi các mẫu bit vào các thanh ghi này.

Một cách, truyền thông này có thể xảy ra thông qua việc sử dụng các lệnh I/O đặc biệt chỉ định truyền một byte hoặc từ đến địa chỉ cổng I/O. Lệnh I/O kích hoạt các đường bus để chọn thiết bị thích hợp và di chuyển các bit vào hoặc ra khỏi thanh ghi của thiết bị.

Hoặc, bộ điều khiển thiết bị có thể hỗ trợ I/O được ánh xạ vào bộ nhớ. Trong trường hợp này, các thanh ghi điều khiển thiết bị được ánh xạ vào không gian địa chỉ của bộ xử lý. CPU thực hiện các yêu cầu I/O bằng cách sử dụng các lệnh chuyển dữ liệu tiêu chuẩn để đọc và ghi các thanh ghi điều khiển thiết bị.

Một cổng I/O thường bao gồm bốn thanh ghi, được gọi là (1) thanh ghi trạng thái, (2) thanh ghi điều khiển, (3) thanh ghi dữ liệu vào, và (4) thanh ghi dữ liệu ra.

* Thanh ghi dữ liệu vào được đọc bởi máy chủ để lấy dữ liệu vào.
* Thanh ghi dữ liệu ra được ghi bởi máy chủ để gửi dữ liệu ra.
* Thanh ghi trạng thái chứa các bit có thể được đọc bởi máy chủ. Những bit này chỉ ra trạng thái của thiết bị.
* Thanh ghi điều khiển có thể được viết bởi máy chủ để bắt đầu một lệnh hoặc thay đổi chế độ của một thiết bị.

Các thanh ghi dữ liệu thường có kích thước từ 1 đến 4 byte.

### Bỏ phiếu hoặc bắt tay - Polling or Handshaking

Trong giao tiếp giữa máy chủ (CPU) và bộ điều khiển:

Bộ điều khiển thể hiện trạng thái của nó thông qua bit bận rộn (busy bit) trong thanh ghi trạng thái (status register) (đặt một bit có nghĩa là viết 1 vào bit đó và xóa một bit có nghĩa là viết 0 vào bit đó.)

Bộ điều khiển đặt bit bận rộn khi đang bận rộn và xóa bit bận rộn khi sẵn sàng tiếp nhận lệnh tiếp theo.

Máy chủ đưa ra ý muốn của mình thông qua bit sẵn sàng lệnh (command-ready bit) trong thanh ghi lệnh (command register). Máy chủ đặt bit sẵn sàng lệnh khi một lệnh sẵn sàng để bộ điều khiển thực hiện.

Máy chủ ghi đầu ra thông qua cổng:

* Host liên tục đọc bit bận trong thanh ghi trạng thái cho đến khi bit đó trở nên rõ ràng.
* Host thiết lập lệnh viết trong thanh ghi lệnh và ghi một byte vào thanh ghi dữ liệu ra.
* Host đặt bit sẵn sàng lệnh.
* Khi điều khiển nhận thấy rằng bit sẵn sàng lệnh đã được đặt, nó đặt bit bận.
* Bộ điều khiển đọc thanh ghi lệnh và thấy lệnh viết. Nó đọc thanh ghi dữ liệu ra để lấy byte và thực hiện I/O với thiết bị.
* Bộ điều khiển xóa bit sẵn sàng lệnh, xóa bit lỗi trong thanh ghi trạng thái để cho biết I/O của thiết bị đã thành công và xóa bit bận để cho biết nó đã hoàn thành.

### Ngắt – Interrupts

Phần cứng CPU có một dây được gọi là đường yêu cầu ngắt mà CPU cảm nhận sau khi thực thi mỗi lệnh.

Khi CPU phát hiện một bộ điều khiển đã kích hoạt tín hiệu trên đường yêu cầu ngắt, CPU thực hiện lưu trạng thái và nhảy đến chương trình xử lý ngắt tại một địa chỉ cố định trong bộ nhớ.

Chương trình xử lý ngắt xác định nguyên nhân của ngắt, thực hiện xử lý cần thiết, thực hiện khôi phục trạng thái và trả lại CPU vào trạng thái thực thi trước khi có ngắt.

Trong một hệ điều hành hiện đại, chúng ta cần những tính năng xử lý ngắt phức tạp hơn.

* Chúng ta cần khả năng hoãn xử lý ngắt trong khi đang xử lý các nhiệm vụ quan trọng.
* Chúng ta cần một cách thức hiệu quả để phân phát cho trình xử lý ngắt thích hợp cho một thiết bị mà không cần phải thăm dò tất cả các thiết bị để xem thiết bị nào gây ra ngắt.
* Chúng ta cần ngắt đa cấp, để hệ điều hành có thể phân biệt giữa các ngắt ưu tiên cao và thấp và có thể phản ứng với mức độ cấp thiết phù hợp.

Trong phần cứng máy tính hiện đại, ba tính năng xử lý ngắt phức tạp này được cung cấp bởi CPU và phần cứng điều khiển ngắt.

Hầu hết các CPU đều có hai dây yêu cầu ngắt.

* Một dây là ngắt không thể khóa, được sử dụng để báo hiệu các điều kiện lỗi khác nhau.
* Dây ngắt thứ hai là có thể khóa: nó có thể bị tắt bởi CPU.

Ngắt có thể khóa được sử dụng bởi các bộ điều khiển thiết bị để yêu cầu dịch vụ.

Mục đích của cơ chế vectored interrupt là giảm sự cần thiết cho một trình xử lý ngắt duy nhất để tìm kiếm tất cả các nguồn ngắt có thể để xác định nguồn nào cần được phục vụ.

### Truy cập bộ nhớ trực tiếp - Direct Memory Access (DMA)

Với một thiết bị thực hiện các truyền dữ liệu lớn, chẳng hạn như đĩa cứng, có vẻ lãng phí khi sử dụng một bộ xử lý chính đắt tiền để giám sát các bit trạng thái và đưa dữ liệu vào một thanh ghi điều khiển của bộ điều khiển một byte một lần - quá trình này được gọi là I/O lập trình (PIO).

Để bắt đầu một truyền dữ liệu DMA, máy chủ ghi một khối lệnh DMA vào bộ nhớ. Khối lệnh này chứa một con trỏ đến nguồn của một truyền dữ liệu, một con trỏ đến đích của truyền dữ liệu và số lượng byte cần được truyền.

CPU ghi địa chỉ của khối lệnh này vào bộ điều khiển DMA, sau đó tiếp tục công việc khác. Bộ điều khiển DMA tiếp tục vận hành trực tiếp trên bus bộ nhớ, đặt địa chỉ trên bus để thực hiện truyền dữ liệu mà không cần sự trợ giúp của CPU chính.

### Giao diện I/O ứng dụng - Application I/O interface

Như những vấn đề khác của kỹ thuật phần mềm phức tạp, phương pháp ở đây liên quan đến trừu tượng hóa, đóng gói và lớp phần mềm. Cụ thể, chúng ta có thể trừu tượng hoá đi những sự khác biệt chi tiết trong các thiết bị I/O bằng cách xác định một số loại chung.

Mỗi loại chung được truy cập thông qua một tập hợp hàm được chuẩn hóa - một giao diện.

Chuẩn hóa các thiết bị chỉ quan tâm đến giao diện chuẩn và sử dụng driver tương ứng để tương tác

Mục đích của tầng điều khiển thiết bị (device-driver) là che giấu sự khác biệt giữa các bộ điều khiển thiết bị trong hệ thống I/O của kernel.

Việc làm cho hệ thống I/O độc lập với phần cứng giúp đơn giản hóa công việc của nhà phát triển hệ điều hành. Nó cũng có lợi cho các nhà sản xuất phần cứng.

* Họ có thể thiết kế các thiết bị mới để tương thích với giao diện điều khiển chủ của máy chủ hiện có.
* Viết các trình điều khiển thiết bị để giao tiếp giữa phần cứng mới và các hệ điều hành phổ biến.

Các thiết bị khác nhau khác nhau về nhiều chiều.

* Luồng ký tự hoặc khối
* Tuần tự hoặc truy cập ngẫu nhiên
* Đồng bộ hoặc không đồng bộ
* Có thể chia sẻ hoặc dành riêng
* Tốc độ hoạt động
* Đọc-ghi, chỉ đọc hoặc chỉ ghi

### Thiết bị khối và ký tự - Block and Character Devices

Driver ổ đĩa cứng

Các lệnh bao gồm đọc(), ghi(), tìm kiếm()

Hệ điều hành và các ứng dụng đặc biệt như hệ thống quản lý cơ sở dữ liệu có thể muốn truy cập vào một thiết bị khối như một mảng tuyến tính đơn giản của các khối. Phương pháp truy cập này đôi khi được gọi là I/O thô (raw I/O).

Việc truy cập tệp được ánh xạ vào bộ nhớ có thể được lắp ghép lên các driver thiết bị khối. Thay vì cung cấp các hoạt động đọc và ghi, một giao diện được ánh xạ bộ nhớ cung cấp truy cập đến lưu trữ đĩa thông qua một mảng các byte trong bộ nhớ chính.

Các thiết bị ký tự bao gồm bàn phím, chuột, cổng nối tiếp.

Các lệnh cơ bản trong giao diện này cho phép một ứng dụng get() hoặc put() một ký tự.

Trên cơ sở giao diện này, thư viện có thể được xây dựng cung cấp truy cập dòng theo từng dòng, với dịch vụ lưu trữ đệm và chỉnh sửa.

### Thiết bị mạng - Network Device

Giao diện có sẵn trong nhiều hệ điều hành là giao diện socket mạng.

Các cuộc gọi hệ thống trong giao diện socket cho phép ứng dụng tạo một socket, kết nối một socket cục bộ đến một địa chỉ từ xa, lắng nghe cho bất kỳ ứng dụng từ xa nào kết nối vào socket cục bộ và gửi và nhận các gói tin qua kết nối.

Một cuộc gọi đến select() trả về thông tin về các socket có gói tin đang chờ để nhận và các socket có chỗ trống để chấp nhận một gói tin để gửi -> loại bỏ việc đo sạch và đợi bận rộn mà sẽ cần thiết cho I/O mạng.

### Đồng hồ và hẹn giờ - Clocks and Timers

Hầu hết các máy tính đều có đồng hồ và bộ định thời phần cứng cung cấp ba chức năng cơ bản:

* Cung cấp thời gian hiện tại.
* Cung cấp thời gian đã trôi qua.
* Thiết lập một bộ định thời để kích hoạt hoạt động X vào thời điểm T.

Phần cứng để đo lường thời gian đã trôi qua và để kích hoạt các hoạt động được gọi là bộ định thời khoảng thời gian có thể lập trình. Nó có thể được thiết lập để chờ một khoảng thời gian nhất định và sau đó tạo ra một ngắt và nó có thể được thiết lập để thực hiện điều này một lần hoặc lặp lại quá trình để tạo ra các ngắt định kỳ.

### Blocking and Nonblocking I/O

Khi một ứng dụng gọi một lệnh hệ thống đồng bộ, thực thi của ứng dụng bị tạm dừng. Ứng dụng này được chuyển từ hàng đợi chạy của hệ thống điều hành sang một hàng đợi chờ. Sau khi lệnh hệ thống kết thúc, ứng dụng sẽ được chuyển trở lại hàng đợi chạy, nơi nó có thể tiếp tục thực thi và nhận các giá trị được trả về bởi lệnh hệ thống.

Hầu hết các hệ điều hành sử dụng các lệnh hệ thống đồng bộ cho giao diện ứng dụng, vì mã ứng dụng đồng bộ là dễ hiểu hơn mã ứng dụng không đồng bộ.

Cuộc gọi Nonblocking-I/O trả về nhiều dữ liệu nhất có thể có sẵn. Nó được sử dụng cho các giao diện người dùng và sao chép dữ liệu (I/O đệm): nhận đầu vào từ bàn phím và chuột trong khi xử lý và hiển thị dữ liệu trên màn hình. Nó được thực hiện thông qua đa luồng. Hàm trả về nhanh với số lượng byte được đọc hoặc viết.

Một ví dụ là ứng dụng video đọc các khung hình từ một tệp trên đĩa cứng trong khi đồng thời giải nén và hiển thị đầu ra trên màn hình.

Một lựa chọn khác cho hàm gọi không chặn là hàm gọi bất đồng bộ (asynchronous system call).

* Một hàm gọi bất đồng bộ trả về ngay lập tức mà không đợi I/O hoàn thành. Ứng dụng tiếp tục thực thi mã của nó.
* Việc hoàn thành I/O tại một thời điểm trong tương lai được truyền đạt cho ứng dụng, thông qua việc thiết lập một biến trong không gian địa chỉ của ứng dụng hoặc thông qua kích hoạt một tín hiệu (signal).

### Kernel I/O Subsystem

Kernel cung cấp nhiều dịch vụ liên quan đến I/O. Một số dịch vụ như lập lịch, đệm, cache, spooling và xử lý lỗi được cung cấp bởi hệ thống I/O của kernel và dựa trên cơ sở hạ tầng phần cứng và trình điều khiển thiết bị.

### Lập lịch I/O - I/O Scheduling

Việc lập lịch cho một tập các yêu cầu I/O có nghĩa là xác định một thứ tự tốt để thực hiện chúng.

Các nhà phát triển hệ điều hành thực hiện lập lịch bằng cách duy trì một hàng đợi chờ cho các yêu cầu cho mỗi thiết bị.

I/O scheduler là thành phần của hệ thống điều khiển dùng để tổ chức lại thứ tự của các yêu cầu I/O trong hàng đợi để cải thiện hiệu suất và thời gian phản hồi trung bình mà các ứng dụng trên hệ thống có được. Việc này được thực hiện bằng cách sắp xếp lại thứ tự các yêu cầu I/O trong hàng đợi, dựa trên các tiêu chí như ưu tiên của yêu cầu, độ ưu tiên của quy trình hoặc chương trình sử dụng yêu cầu I/O.

### Buffering – Bộ đệm

Bộ đệm (Buffer) là một khu vực bộ nhớ lưu trữ dữ liệu trong quá trình chuyển đổi giữa hai thiết bị hoặc giữa một thiết bị và một ứng dụng.

Việc đệm được thực hiện với ba mục đích:

* Để giải quyết sự khác biệt về tốc độ của thiết bị.
* Để giải quyết sự khác biệt về kích thước chuyển đổi của thiết bị.
* Để duy trì "các nguyên tắc sao chép": sao chép dữ liệu vào bộ đệm nhân hệ điều hành trước khi trả lại quyền điều khiển cho ứng dụng.

### Caching

Một cache là một khu vực bộ nhớ nhanh chứa các bản sao của dữ liệu. Truy cập vào bản sao được lưu trữ trong cache hiệu quả hơn so với truy cập vào bản gốc. Khi một yêu cầu truy cập dữ liệu được thực hiện, hệ thống sẽ trước tiên kiểm tra cache xem bản sao dữ liệu đó có nằm trong cache không. Nếu có, dữ liệu sẽ được lấy từ cache, giảm thiểu thời gian truy cập và tăng hiệu suất hệ thống. Các ứng dụng của cache bao gồm tăng tốc độ đọc và ghi dữ liệu từ đĩa cứng hoặc bộ nhớ chậm và tăng tốc độ truy xuất dữ liệu từ bộ nhớ trong.

### Spooling

Một spool là một buffer (bộ đệm) giữ đầu ra cho một thiết bị nếu thiết bị chỉ có thể phục vụ một yêu cầu tại một thời điểm. Ví dụ: máy in.

### Xử lý lỗi - Error Handling

Thiết bị và các truyền dữ liệu I/O có thể thất bại theo nhiều cách, có thể do các lý do tạm thời, ví dụ như một mạng quá tải, hoặc do các lý do "vĩnh viễn", ví dụ như khi bộ điều khiển đĩa bị lỗi.

Hệ điều hành thường có khả năng bù đắp tốt cho những lỗi tạm thời. Ví dụ, nếu một yêu cầu đọc từ đĩa gặp sự cố, nó sẽ thử đọc lại và nếu một lỗi gửi trên mạng xảy ra thì nó sẽ thử gửi lại.

Tuy nhiên, nếu một thành phần quan trọng gặp sự cố vĩnh viễn, hệ điều hành khó khăn để phục hồi.

Như một nguyên tắc chung, một lệnh hệ thống I/O sẽ trả về một bit thông tin về trạng thái của lệnh, cho biết thành công hoặc thất bại.

# Chương 9: Quản lý File

### Khái niệm

Một tệp là một bộ sưu tập các thông tin liên quan được ghi lại trên bộ nhớ thứ cấp với tên gọi.

Từ quan điểm của người dùng, tệp là đơn vị nhỏ nhất của bộ nhớ thứ cấp.

Thường thì, tệp đại diện cho các chương trình (cả nguồn lẫn đối tượng) và dữ liệu.

Nói chung, một tệp là một chuỗi các bit, byte, dòng hoặc bản ghi, ý nghĩa của nó được xác định bởi người tạo và người sử dụng tệp.

### Thuộc tính File

Một file có tên, cho tiện lợi của người sử dụng con người, và được gọi bằng tên.

Các thuộc tính của một file thay đổi từ hệ điều hành này sang hệ điều hành khác, nhưng thông thường bao gồm các thuộc tính sau:

* Tên - thông tin chỉ được giữ dưới dạng có thể đọc được cho con người.
* Định danh - thẻ duy nhất (số) xác định tệp trong hệ thống tệp.
* Loại - cần thiết cho các hệ thống hỗ trợ các loại khác nhau.
* Vị trí - con trỏ đến vị trí tệp trên thiết bị.
* Kích thước - kích thước tệp hiện tại.
* Bảo vệ - kiểm soát ai có thể đọc, ghi, thực thi tệp.
* Thời gian, ngày và nhận dạng người dùng - dữ liệu cho bảo vệ, bảo mật và giám sát sử dụng.

### Hoạt động

Các thao tác cơ bản với file:

* Tạo một file: Hai bước cần thiết. Đầu tiên, phải tìm được không gian trong hệ thống tệp cho file. Thứ hai, một mục cho file mới phải được tạo trong thư mục.
* Viết một file: Một lệnh gọi hệ thống chỉ định cả tên của file và thông tin để ghi vào file. Với tên của file, hệ thống tìm kiếm trong thư mục để tìm vị trí của file.
* Đọc một tập tin: Là một hệ thống gọi hàm chỉ định tên tập tin và vị trí (trong bộ nhớ) để đặt khối tiếp theo của tập tin.
* Định vị trong một tập tin: Hệ thống tìm kiếm thư mục để tìm thông tin cần thiết và con trỏ vị trí hiện tại trong tập tin được đặt lại vào một giá trị cụ thể.
* Xóa một tập tin: Hệ điều hành tìm kiếm thư mục cho tập tin được đặt tên và giải phóng tất cả không gian tập tin, để có thể được sử dụng lại bởi các tập tin khác, và xóa bỏ mục trong thư mục.
* Truncating a file là một thao tác để giảm kích thước của file, bằng cách đặt độ dài file về 0 và giải phóng không gian trên ổ đĩa mà file đó đã sử dụng trước đó. Các thuộc tính khác của file sẽ không bị thay đổi sau khi thực hiện thao tác này.
* Mở(Fi): tìm kiếm cấu trúc thư mục trên đĩa cho mục nhập Fi và di chuyển nội dung của mục đó vào bộ nhớ.
* Đóng(Fi): di chuyển nội dung của mục nhập Fi trong bộ nhớ vào cấu trúc thư mục trên đĩa.

Một số hệ điều hành cung cấp các tiện ích để khóa một tệp mở (hoặc một số phần của tệp). Khóa tệp cho phép một quy trình khóa tệp và ngăn các quy trình khác truy cập vào nó.

Khóa tệp rất hữu ích cho các tệp được chia sẻ bởi nhiều quy trình - ví dụ: một tệp nhật ký hệ thống có thể được truy cập và sửa đổi bởi một số quy trình trong hệ thống.

### Truy câp tuần tự - Sequential Access

Một phép đọc - "đọc tiếp theo" - đọc phần tiếp theo của tập tin và tự động di chuyển con trỏ tập tin, theo dõi vị trí I/O.

Tương tự, phép ghi - "ghi tiếp theo" - thêm vào cuối tập tin và di chuyển đến cuối phần vừa mới được ghi (làm tăng độ dài tập tin).

Tập tin có thể được đặt lại ở đầu, và trên một số hệ thống, chương trình có thể có khả năng bỏ qua n bản ghi về phía trước hoặc phía sau với n là một số nguyên nào đó.

Truy cập tuần tự dựa trên mô hình băng của tập tin.

### Truy cập trực tiếp - Direct Access (or relative access)

Một tập tin được tạo thành từ các bản ghi logic có độ dài cố định cho phép các chương trình đọc và ghi các bản ghi nhanh chóng mà không cần theo bất kỳ thứ tự cụ thể nào.

Phương pháp truy cập trực tiếp dựa trên mô hình đĩa của một tệp, vì đĩa cho phép truy cập ngẫu nhiên đến bất kỳ khối tệp nào.

Đối với phương pháp truy cập trực tiếp, các hoạt động tệp phải được sửa đổi để bao gồm số khối làm tham số.

Chúng ta có các hoạt động đọc n và ghi n, trong đó n là số khối.

### Các cách truy cập khác

Các phương pháp khác liên quan đến chỉ mục cho tập tin. Chỉ mục chứa các con trỏ tới các khối. HDH sẽ tìm kiếm bằng chỉ mục và sử dụng con trỏ để truy cập tệp

### Cấu trúc thư mục

Đó là một bộ sưu tập các nút chứa thông tin về tất cả các tệp tin.

Cả cấu trúc thư mục và các tệp tin đều nằm trên đĩa.

### Cấu trúc lưu trữ

Ổ đĩa có thể được phân chia thành các phân vùng hoặc lát cắt, hoặc (trong thế giới IBM) được gọi là các minidisk.

Phân vùng có thể được sử dụng dưới dạng không định dạng (swap space) - không có hệ thống tệp, hoặc được định dạng với hệ thống tệp.

Một ổ đĩa (hoặc bất kỳ thiết bị lưu trữ nào đủ lớn) có thể được sử dụng toàn bộ cho một hệ thống tệp hoặc đặt nhiều hệ thống tệp.

Thực thể chứa hệ thống tệp được gọi là một khối lượng (volume).

Mỗi khối lượng chứa hệ thống tệp cũng theo dõi thông tin của hệ thống tệp đó trong thư mục thiết bị hoặc bảng nội dung khối lượng (volume table of contents).

Thư mục thiết bị (thông thường được gọi là thư mục) lưu trữ thông tin - chẳng hạn như tên, vị trí, kích thước và loại - về tất cả các tệp trên ổ đĩa đó.

### Tổng quan về thư mục

Danh sách các hoạt động mà có thể được thực hiện trên một thư mục:

* Tìm kiếm một file
* Tạo một file
* Xóa một file
* Liệt kê các file trong thư mục
* Đổi tên một file
* Điều hướng trong hệ thống file.

### Chế độ nhiều người dùng

Khi một hệ điều hành đáp ứng nhiều người dùng, các vấn đề chia sẻ tập tin, đặt tên tập tin và bảo vệ tập tin trở thành chính yếu.

ID người dùng xác định người dùng, cho phép quyền và bảo vệ được thiết lập cho từng người dùng.

ID nhóm cho phép người dùng thuộc nhóm, cho phép quyền truy cập của nhóm.

Chủ sở hữu là người dùng có thể thay đổi thuộc tính và cấp quyền truy cập và có quyền kiểm soát tập tin nhiều nhất.

### Hệ thống tệp từ xa - Remote File Systems

Mạng lưới cho phép chia sẻ các tài nguyên được phân bổ trên một khuôn viên trường hoặc thậm chí khắp thế giới. Một tài nguyên rõ ràng để chia sẻ là dữ liệu dưới dạng tập tin.

* Các chương trình như FTP
* Sử dụng các hệ thống tập tin phân tán (DFS)
* Thông qua trang web toàn cầu.

# Câu hỏi ôn tập

1. **Cho biết mục đích của system call**

Cho phép chương trình của người dùng yêu cầu dịch vụ của hệ điều hành

1. **Các hoạt động chính của hệ điều hành trong quản lý quy trình**

Tạo và xóa cả process của người dùng và hệ thống.

Đình chỉ và tiếp tục các process.

Cung cấp cơ chế đồng bộ hóa process.

Cung cấp cơ chế giao tiếp process.

Cung cấp cơ chế xử lý tình trạng khóa chết (deadlock).

1. **Các hoạt động chính của hệ điều hành trong quản lý bộ nhớ**

Theo dõi các phần của bộ nhớ hiện đang được sử dụng và được sử dụng bởi ai.

Quyết định quá trình và dữ liệu nào sẽ được di chuyển vào và ra khỏi bộ nhớ.

Phân bổ và giải phóng không gian bộ nhớ khi cần thiết.

1. **5 chức năng chính của hdh trong quản lý hệ thống tập tin – quản lý file – quản lý tập tin**

Tạo và xóa tập tin

Quản lý thư mục

Thao tác tập tin và thư mục

Phân quyền truy cập

Quản lý đĩa và không gian lưu trữ

1. **Các hoạt động chính của hdh trong quản lý thiết bị lưu trữ (mass - storage)**

Cấp phát không gian lưu trữ

Định thời đĩa (Disk scheduling)

1. **Địa chỉ logic hay địa chỉ ảo là địa chỉ**

Được tạo ra bởi tính toán của CPU

1. **Địa chỉ vật lý là địa chỉ**

Được tạo ra bởi đơn vị quản lý bộ nhớ (MMU)

1. **HDH sử dụng cặp con trỏ base-limit trong không gian bộ nhớ logic của một quy trình để**

Qui định vùng nhớ mà một quy trình có thể truy xuất

1. **Nguyên lý, đặc điểm cấp phát bộ nhớ kiểu MFT**

Bộ nhớ được chia thành nhiều phân vùng có kích thước cố định

Mỗi process sử dụng một phân vùng và sau khi kết thúc sẽ trả lại phân vùng đó cho hệ thống

1. **Cấp phát bộ nhớ theo kiểu MVT**

Phần bộ nhớ khả dụng gọi là hole được cấp cho một phần vừa đủ cho quy trình, phần còn lại tạo thành một hole khác

1. **Bộ định thời dài hạn có nhiệm vụ**

Chọn các process từ process pool để đưa vào bộ nhớ

1. **Bộ định thời ngắn hạn có nhiệm vụ**

Chọn từ trong số các quá trình đã sẵn sàng để thực thi và cấp phát CPU cho một trong số chúng.

1. **Trên phần lớn Window và Unix thì bộ định thời dài hạn**

Không có bộ định thời dài hạn

1. **Trên một số hdh chia sẽ thời gian người ta sử dụng bộ định thời trung gian làm gì**

Thực hiện swapping process

1. **Khi nào process cha kết thúc process con**

Vượt quá tài nguyên được cấp

Nhiệm vụ không còn cần thiết

Process cha đã thoát

1. **Khi process cha tạo ra một process con thì có thể**

Cha và con chia sẻ tất cả các tài nguyên

Con chia sẻ một tập hợp con của các tài nguyên của cha

Cha và con không chia sẻ tài nguyên nào

1. **Thư viện Win32 là**

Thư viện cung cấp trên hệ thống window ở mức kernel

1. **Thư viện Pthread là thread mở rộng của chuẩn POSIX là thư viện được cấp ở mức**

Cả kernel và user

1. **Quan hệ giữa kernel thread và user thread trên hdh linux là**

One-to-one

1. **Hai kỹ thuật tạo thread trong java**

Một cách tiếp cận là tạo một lớp mới được phát sinh từ lớp Thread và ghi đè phương thức run().

Một cách tiếp cận khác và phổ biến hơn là định nghĩa một lớp thực hiện interface Runnable.

1. **Chức năng thư viện thread**

Thư viện thread cung cấp các công cụ để tạo, quản lý và đồng bộ hóa các luồng thực thi độc lập trong một chương trình đa nhiệm.

1. **Cho biết 2 phương pháp để thực hiện 1 thư viện Thread**

Cung cấp một thư viện hoàn toàn ở user space mà không có hỗ trợ từ kernel. Tất cả các mã và cấu trúc dữ liệu cho thư viện tồn tại trong user space. Điều này có nghĩa là gọi một hàm trong thư viện dẫn đến một lời gọi hàm cục bộ trong user space và không phải là một system call.

Cách thực hiện thư viện luồng cấp kernel được hỗ trợ trực tiếp bởi hệ điều hành. Mã và cấu trúc dữ liệu cho thư viện tồn tại trong không gian kernel. Gọi một hàm trong API của thư viện thường dẫn đến một lệnh gọi hệ thống đến kernel

1. **Cho biết 3 chức năng chính của hệ điều hành**

Một chương trình quản lý phần cứng máy tính

Cung cấp cơ sở cho các chương trình ứng dụng

Hoạt động như một trung gian giữa người dùng máy tính và phần cứng.

1. **Chỉ thị nào sau đây có thể thực hiện ở user mode**

Access I/O device

1. **Thế nào là thiết bị lưu trữ nonvolatine**

Là một loại thiết bị lưu trữ dữ liệu có khả năng giữ lại dữ liệu mà không cần nguồn điện liên tục

Một số ví dụ về thiết bị lưu trữ nonvolatile bao gồm ổ đĩa cứng (HDD), ổ đĩa SSD, thẻ nhớ, đĩa CD/DVD và băng từ

1. **Cho biết các ưu điểm của MultiProcessor system**

Tăng thời lượng xử lý – thực hiện nhiều công việc trong thời gian ngắn; Tiết kiệm tài nguyên hệ thống – vì có thể chia sẻ các thiết bị ngoại vi, bộ nhớ và nguồn điện; Tăng độ tin cậy – nếu các chức năng được phân phối đúng đắn giữa các bộ xử lý, thì sự cố của một bộ xử lý sẽ không làm đình trệ hệ thống, chỉ làm nó chậm lại.

1. **Trong một hệ thống Asymmetric multiprocessing**

Một processor định thời và cấp phát công việc

1. **Cho biết các trạng thái có thể có của 1 proccess**

New: Quá trình đang được tạo ra.

Running: Các lệnh đang được thực thi.

Waiting: Quá trình đang chờ đợi một sự kiện nào đó xảy ra (như hoàn thành I/O hoặc nhận tín hiệu).

Ready: Quá trình đang chờ được phân bổ cho một bộ xử lý.

Terminated: Quá trình đã hoàn thành thực thi.

1. **Điều gì là vấn đề chính để làm nên sự khác biệt của bộ định thời dài hạn và bộ định thời ngắn hạn**

Sự phân biệt chính giữa hai bộ lập lịch này nằm ở tần suất thực thi.

1. **Một I/O bound proccess là gì và 1 CPU bound proccess là gì?**

Một I/O bound process là một tiến trình tốn nhiều thời gian để thực hiện các hoạt động đầu vào và đầu ra

CPU bound process là một tiến trình tốn nhiều thời gian để thực hiện các hoạt động tính toán trên CPU.

1. **Symmetric Multiprocessing là gì**

Tất cả process có quan hệ ngang hàng

1. **Cho biết phân mãnh ngoài (external fragmentation) khi cấp phát bộ nhớ là gì**

Hiện tượng phân mảnh bên ngoài xảy ra khi tổng không gian nhớ đủ để đáp ứng yêu cầu, nhưng các không gian khả dụng không liên tục; bộ nhớ bị phân mảnh thành một số lỗ nhỏ.

1. **Cho biết phân mãnh trong (internal fragmentation) khi cấp phát bộ nhớ là gì**

Nếu bộ nhớ được phân bổ cho một quy trình lớn hơn so với bộ nhớ yêu cầu, thì bộ giữa hai số này là internal fragmentation, là bộ nhớ nằm trong một phân vùng nhưng không được sử dụng.

1. **Để giải quyết phân mãnh ngoài (external fragmentation) của bộ nhớ, hệ điều hành sử dụng phương pháp Compaction. Cho biết ý tưởng chính của phương pháp này và điều kiện để thực hiện nó?**

Một giải pháp cho vấn đề external fragmentation bên ngoài là nén. Mục tiêu là xáo trộn nội dung bộ nhớ để đặt tất cả các bộ nhớ trống cùng với nhau thành một khối lớn. Tuy nhiên, việc nén không phải lúc nào cũng khả thi.

Điều kiện: phải có một trình quản lý bộ nhớ (memory manager) có khả năng tái sử dụng các vùng bộ nhớ đã được giải phóng và sắp xếp chúng liên tiếp nhau

1. **Hệ điều hành hỗ trợ cơ chế hợp tác (cooperation) giữa các process. Cho biết vài lý do cũng như sự hữu ích khi các proccess liên kết với nhau?**

Chia sẻ thông tin. Vì nhiều người dùng có thể quan tâm đến cùng một thông tin (một tập tin chia sẻ...)

Tăng tốc tính toán. Nếu chúng ta muốn một nhiệm vụ cụ thể chạy nhanh hơn, chúng ta phải chia nó thành các tác vụ con, mỗi tác vụ sẽ được thực thi song song với các tác vụ khác.

Tính mô-đun. Để xây dựng hệ thống theo kiểu mô-đun - chia các chức năng của hệ thống thành các quy trình hoặc luồng riêng biệt.

Tiện lợi. Người dùng có thể làm việc trên nhiều nhiệm vụ cùng một lúc.

1. **Cho biết các điều kiện cần thiết để các proccess có thể hợp tác (cooperation) với nhau theo cách chia sẻ bộ nhớ (share-memory)**

Bộ nhớ chia sẻ yêu cầu hai hoặc nhiều tiến trình đồng ý với nhau.

Các tiến trình cũng có trách nhiệm đảm bảo rằng chúng không ghi vào cùng một vị trí cùng một lúc.

1. **Cho biết RPC (Remote procedure call) được sử dụng khi nào và ý nghĩa của nó là gì**

RPC (Remote Procedure Call) được sử dụng khi cần thiết để gọi một hàm hoặc thủ tục trên một máy tính từ một máy tính khác trong mạng. RPC cho phép các ứng dụng chạy trên các máy tính khác nhau giao tiếp với nhau như thể chúng đang chạy trên cùng một máy tính.

Ý nghĩa của RPC là tạo ra một phương thức gọi hàm ở xa, giúp cho các ứng dụng có thể tương tác với nhau thông qua mạng máy tính một cách dễ dàng và hiệu quả. RPC đóng vai trò quan trọng trong các ứng dụng phân tán, nơi mà các thành phần khác nhau của ứng dụng chạy trên các máy tính khác nhau trong mạng.

1. **Hàng đợi sẵn sàng (ready queue) là hàng đợi**

Các quá trình đang lưu trú trong bộ nhớ chính và sẵn sàng và chờ để thực thi được giữ trên một danh sách gọi là hàng đợi sẵn sàng

1. **Vùng đệm nhập xuất (I/O bufering) là gì và mục đích của vùng này**

Bộ đệm (Buffer) là một khu vực bộ nhớ lưu trữ dữ liệu trong quá trình chuyển đổi giữa hai thiết bị hoặc giữa một thiết bị và một ứng dụng.

Việc đệm được thực hiện với ba mục đích:

* Để giải quyết sự khác biệt về tốc độ của thiết bị.
* Để giải quyết sự khác biệt về kích thước chuyển đổi của thiết bị.
* Để duy trì "các nguyên tắc sao chép": sao chép dữ liệu vào bộ đệm nhân hệ điều hành trước khi trả lại quyền điều khiển cho ứng dụng.

1. **Cho biết nội dung của DMA command block**

Địa chỉ bộ nhớ nguồn (Source Address): Đây là địa chỉ của bộ nhớ mà dữ liệu sẽ được sao chép từ đó bởi hoạt động DMA.

Địa chỉ bộ nhớ đích (Destination Address): Đây là địa chỉ của bộ nhớ mà dữ liệu sẽ được sao chép đến bởi hoạt động DMA.

Độ dài dữ liệu (Data Length): Đây là số lượng byte của dữ liệu sẽ được chuyển đổi bởi hoạt động DMA.

Thông số cấu hình (Configuration Parameters): Đây là các thông số cấu hình khác, bao gồm chế độ hoạt động DMA (single block hoặc multi-block), độ rộng bus, tốc độ truyền dữ liệu, và các thông số khác liên quan đến hoạt động DMA.

Địa chỉ bộ điều khiển DMA (DMA Controller Address): Đây là địa chỉ của bộ điều khiển DMA, nơi mà DMACB sẽ được truyền để bắt đầu hoạt động DMA.

1. **Hoạt động trong cơ chế DMA**

DMA command block được ghi vào bộ nhớ

CPU viết địa chỉ của khối lệnh DMA này vào bộ điều khiển DMA, sau đó tiếp tục với các công việc khác

1. **Khi controller đặt một tín hiệu interrupt-request line thì CPU sẽ thực hiện**

Lưu lại trạng thái hiện tại và chuyển điều khiển đến đoạn thủ tục

1. **Khi một process cha tạo một process con mới thì không gian địa chỉ của hai process này có thể**

Quy trình con là một bản sao của quy trình cha (nó có chương trình và dữ liệu giống với quy trình cha).

Quy trình con có một chương trình mới được tải vào nó

1. **Race condition là gì**

Race condition là tình trạng xảy ra khi nhiều quy trình hoặc luồng thực thi trong chương trình cùng truy cập và thay đổi dữ liệu chung một cách đồng thời và không có sự điều khiển đúng đắn nào, dẫn đến kết quả của chương trình không đáng tin cậy và không thể dự đoán trước được

1. **Trong một hệ thống có các process tương tác với nhau (cooperating process), hãy cho biết làm thế nào để ngăn chặn race conditon xảy ra**

Mutual exclusion (Đồng quyền): Nếu quá trình Pi đang thực thi trong critical section của nó, thì không có quá trình nào khác có thể thực thi trong critical section của chúng.

Progress (Quy trình): Nếu không có quá trình nào đang thực thi trong critical section của nó và một số quá trình muốn vào critical section của chúng, thì chỉ có những quá trình đang không thực thi trong phần còn lại của chúng mới có thể tham gia vào quyết định về việc quá trình nào sẽ vào critical section của mình tiếp theo, và việc lựa chọn này không thể được trì hoãn vô thời hạn.

Bounded waiting (Giới hạn thời gian chờ đợi): Tồn tại một giới hạn, hoặc giới hạn, về số lần mà các quá trình khác được phép vào critical section của họ sau khi một quá trình đã yêu cầu vào critical section của mình và trước khi yêu cầu đó được chấp nhận.

1. **Critical section của một process là vùng mà ở đó**

Process có thể thay đổi dữ liệu của nó

1. **Cho biết ý nghĩa các thành phần trong một đồ thị cấp phát tài nguyên (resource-allocation graph)**

Đỉnh của đồ thị: Gồm hai loại: tập gồm tất cả quy trình trong hệ thống (kí hiệu là P), tập gồm tất cả các loại tài nguyên (kí hiệu là R) trong hệ thống

Request Edge – Cạnh yêu cầu: Thể hiện quy trình Pi đang yêu cầu tài nguyên Rj

Assignment Edge – Cạnh gán: Thể hiện tài nguyên Rj đã được cấp cho quy trình Pi

1. **Nếu một process có nhiều thread thì các thread này**

Có cùng đoạn mã chương trình, dữ liệu và tài nguyên hệ thống

1. **Cho biết ý nghĩa vắn tắt của các mô hình quan hệ giữa user thread và kernel thread**

One – to – one: Ánh xạ mỗi luồng người dùng vào một luồng nhân hệ điều hành.

Many – to – one: Ánh xạ nhiều luồng cấp người dùng vào một luồng nhân hệ điều hành.

Many – to – many: Đa luồng nhiều luồng cấp người dùng đến một số luồng nhân hệ điều hành nhỏ hơn hoặc bằng.

1. **Preemptive kernel là phương pháp**

Cho phép một process đang chạy ở kernel mode ngừng và CPU phục vụ process khác.

1. **Nonpreemptive kernel là phương pháp**

Kernel–mode process sẽ thực hiện cho tới khi nó tự động giải phóng CPU

1. **Cho biết điều kiện để có loại trừ tương hỗ (multual exclusion) trong vấn đề deadlock là gì**

Một quy trình không bao giờ cần phải đợi một tài nguyên có thể chia sẻ.

1. **Cho biết lợi ích của các nhà phát triển hdh thiết kế các hệ thống con nhập xuất (I/O subsystem) độc lập vs phần cứng**

Họ có thể thiết kế các thiết bị mới để tương thích với giao diện điều khiển chủ của máy chủ hiện có.

Viết các trình điều khiển thiết bị để giao tiếp giữa phần cứng mới và các hệ điều hành phổ biến.

1. **Các phương pháp xử lý deadlock của hệ điều hành**

Sử dụng một giao thức để ngăn hoặc tránh deadlock, đảm bảo không bao giờ rơi vào deadlock

Cho phép hệ thống rơi vào deadlock phát hiện và khôi phục

Bỏ qua vấn đề deadlock

### Cho 2 ví dụ minh họa chương trình nhiều thread hoạt động tốt hơn so với chương trình một thread

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

Xử lý hình ảnh và âm thanh trong ứng dụng đa phương tiện trên máy tính