NGUYÊN LÝ HỆ ĐIỀU HÀNH CT107

- Mục tiêu của từng chương: Tại sao lại học chương này? Chương này giúp ta cái gì

Chương 1: Giới thiệu hệ điều hành

I. Hệ điều hành là gì

- Phần cứng phải có phần mềm để có thể điều khiển được chúng (firmware, HDH)

- HĐH là một **chương trình quản lý tài nguyên của máy tính**, đóng vai trò là một **lớp trung gian** giữa người sử dụng máy tính và phần cứng của máy tính (giữa máy tính và các ứng dụng)

- Mục tiêu của HĐH:

+ Là cung cấp **phương tiện giao tiếp giữa** người dùng và máy tính

+ **Nhận và thực thi** các yêu cầu của người dùng một cách hiệu quả, nhanh chóng và dễ dàng thông qua các chương trình ứng dụng

+ **Nó cũng quản lý và sử dụng tài nguyên** máy tính một cách hiệu quả

- Thành phần của một hệ thống máy tính:

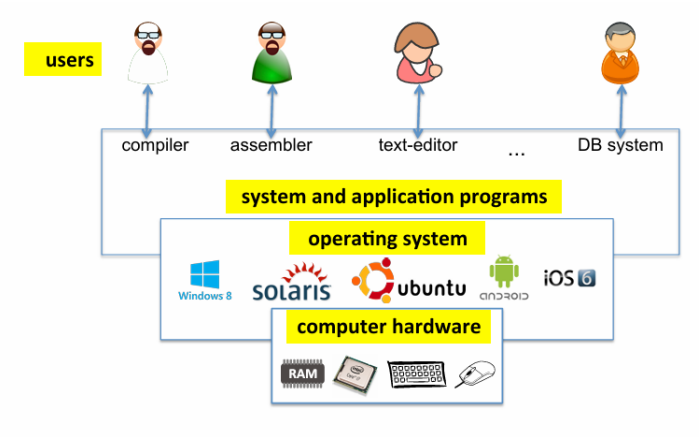
+ **Phần cứng (hardware):** Cung cấp các tài nguyên cơ bản cho việc tính toán (CPU, bộ nhớ, I/O).

+ **Hệ điều hành (OS):** Kiểm soát và điều phối việc sử dụng phần cứng của chương trình ứng dụng của người dùng.

+ **Các chương trình hệ thống và ứng dụng (system and application programs):** Sử dụng tài nguyên hệ thống để giải quyết các vấn đề tính toán của người dùng.

+ **Người dùng (user):** Con người hoặc các thiết bị có nhu cầu tính toán sử dụng các chương trình máy tính.

Dưới đây là bản tóm tắt ngắn gọn thành 4 ý chính:



1. **Người dùng (Users):** Sử dụng các chương trình như trình biên dịch, soạn thảo văn bản, hệ cơ sở dữ liệu...
2. **Chương trình hệ thống và ứng dụng:** Trung gian giúp người dùng thao tác với máy tính, sử dụng tài nguyên hệ thống.
3. **Hệ điều hành (Operating System):** Quản lý và điều phối việc sử dụng phần cứng, hỗ trợ các chương trình hoạt động.
4. **Phần cứng (Hardware):** Cung cấp tài nguyên vật lý (CPU, RAM, ổ đĩa, bàn phím...) để thực hiện tính toán và xử lý dữ liệu.

- Vai trò của HĐH:

+ Từ góc nhìn người dùng: HĐH là giao diện mà người dùng sử dụng. Trên **máy tính cá nhân (PC),** HĐH tập trung vào **sự tiện lợi**, **dễ sử dụng**, **hiệu năng cao** và ít quan tâm đến việc chia sẻ tài nguyên. Trong các hệ thống máy tính **chia sẻ** (mainframe, minicomputer), HĐH tập trung vào việc **tận dụng tài nguyên** và **chia sẻ công bằng**. Đối với thiết bị cầm tay, HĐH được thiết kế cân bằng **giữa hiệu năng và năng lượng.**

Dưới sự hỗ trợ của API, mà các nhà lập trình ứng dụng chỉ cần lập trình mà không cần tương tác với phần cứng.

+ Từ góc nhìn hệ thống: HĐH là

- **Bộ cấp phát tài nguyên** (quản lý và cung cấp các nguồn tài nguyên)
- **Chương trình điều khiển** (điều khiển các thiết bị nhập/xuất và sự thực thi của các chương trình người dùng)
- **Nhân (kernel)** của hệ thống máy tính, là chương trình duy nhất chạy thường trực toàn thời gian.

II. Tổ chức của một hệ thống máy tính

- Một hệ thống máy tính bao gồm một hoặc vài **bộ xử lý (CPU)**, **bộ nhớ chính (RAM)**, các **thiết bị I/O** (đĩa từ, bàn phím, chuột, màn hình, máy in), và các **bộ điều khiển thiết bị**.

- Các thiết bị I/O và CPU có thể **thực thi đồng thời**.

- Mỗi bộ điều khiển thiết bị điều khiển một loại thiết bị và có bộ nhớ đệm riêng.

- Mỗi hệ điều hành sẽ có một số driver phổ biến, để chạy các device phổ biến

- CPU, bộ nhớ chính và các bộ điều khiển thiết bị được nối kết thông qua một **bus chung**.

- Các bộ điều khiển thiết bị thông báo cho CPU sau khi thực hiện xong tác vụ bằng cách sử dụng các **ngắt (interrupt)**.

Quá trình khởi động của máy tính: Khi máy tính khởi động, nó cần một chương trình khởi tạo (initial program/bootstrap), được lưu trữ trong ROM (firmware). Chương trình này có chức năng định vị kernel của HĐH, nạp kernel vào bộ nhớ, khởi động kernel và sau đó nhường quyền điều khiển cho HĐH (kernel).

- Bootstrap thường được lưu trong ROM và EPROM vì những vùng nhớ này không bị mất khi tắt máy. Một cách gọi khác của phần này là firmware.

- Khi hệ điều hành được nạp và thực thi, hệ điều hành sẽ hoạt động và cung cấp các dịch vụ hệ thống thông qua lời gọi hệ thống cho các chương trình và người dùng

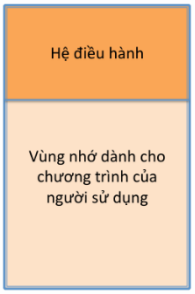
Định nghĩa và ký hiệu lưu trữ: Đơn vị lưu trữ cơ bản là **bit** (0 hoặc 1). Một **byte** là 8 bit, là đơn vị lưu trữ thuận tiện nhỏ nhất trên hầu hết các máy tính. Một **word** là đơn vị dữ liệu bản xứ của một kiến trúc máy tính, được tạo thành từ một hoặc nhiều byte (ví dụ: máy tính 64-bit có word là 64-bit hoặc 8-byte)8....

III. Các loại hệ thống máy tính

- Vai trò, chức năng và kiến trúc của HĐH phụ thuộc vào **kiến trúc** của hệ thống máy tính. (tức là tùy vào kiến trúc của hệ thống máy tính thì hệ điều hành sẽ có vai trò chức năng và kiến trúc phù hợp cho nó)

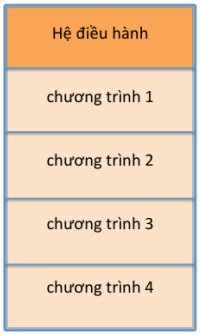
- Các hệ thống máy tính có thể chia làm 2 loại chính: **hệ thống đa dụng** và **hệ thống chuyên dụng**.

+ **Hệ thống đa dụng**:



Hệ thống Batch (Bó): Là HĐH sơ khai, người dùng không giao tiếp trực tiếp với máy tính. Công **việc tương tự được bó lại** để giảm thời gian thiết lập. CPU thường xuyên rảnh rỗi do tốc độ CPU nhanh hơn nhiều so với thiết bị I/O.

Hệ thống Đa Chương (1997) (Multiprogramming): Sự ra đời của công nghệ **đĩa** là **cơ sở**. Các công việc có thể lưu trữ và truy xuất một cách **không tuần tự**. Một số công việc được lưu trong bộ nhớ chính. CPU được điều phối thực hiện công việc khác nếu công việc hiện hành đang chờ I/O, giúp tận dụng thời gian rỗi của CPU. HĐH cần các cơ chế I/O, quản lý bộ nhớ (cấp phát bộ nhớ cho nhiều tiến trình), và **định thời CPU.** Chỉ nhường lại CPU khi hoàn thành hoặc chờ thực hiện thao tác I/O.



Hệ thống Chia Thời Gian (Time-Sharing/Multitasking): Mở rộng của hệ thống đa chương, cho phép nhiều người dùng chia sẻ máy tính bằng cách phân chia thời gian sử dụng tài nguyên. CPU được điều phối cho nhiều công việc trong **bộ nhớ và trên đĩa** (CPU chỉ thực hiện trong bộ nhớ chính). Công việc được hoán chuyển giữa bộ nhớ và đĩa. Nó phức tạp hơn HĐH đa chương, yêu cầu quản lý bộ nhớ phức tạp (quản lý cạnh tranh, bảo vệ bộ nhớ) (Khi CPU được điều phối cho nhiều công việc nằm trong bộ nhớ và trên đĩa, công việc có thể được hoán chuyển giữa bộ nhớ và đĩa, điều này đòi hỏi một cơ chế quản lý bộ nhớ phức tạp), bộ nhớ ảo (tăng số lượng chương trình nằm trong bộ nhớ), cơ chế định thời CPU tinh vi và hệ thống quản lý đĩa.

- Giao tiếp trực tuyến giữa hệ thống và người dùng là cơ chế cho phép **người dùng tương tác trực tiếp với hệ điều hành và các chương trình đang chạy**, thường thông qua các thiết bị đầu vào như bàn phím

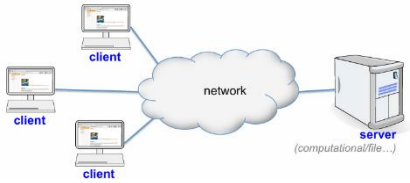
Hệ thống để bàn (Desktop): Máy tính cá nhân dành cho **một người dùng duy nhất**, ưu tiên sự tiện lợi và phản ứng nhanh. Có thể áp dụng các kỹ thuật từ hệ thống lớn hơn. Chạy nhiều họ hệ điều hành khác nhau (Win, Mac, Unix, Linux)

Hệ thống Đa Xử Lý (Multi-processor): Nhiều hơn một CPU được nối kết chặt chẽ (**hệ thống song song or ghép đôi chặt**). Các bộ xử lý chia sẻ bộ nhớ và xung đồng hồ. Lợi ích: tăng năng lực xử lý, kinh tế (chia sẻ ngoại vi), tăng tính tin cậy (chịu lỗi). Gồm:

- **Symmetric MultiProcessing (SMP)**: Mỗi CPU chạy một bản sao giống nhau của HĐH. Phổ biến trong HĐH hiện đại.
- **Asymmetric MultiProcessing (AMP)**: Mỗi CPU được giao một công việc cụ thể (CPU chủ (Master) lập lịch biểu và giao việc cho CPU tớ (Slave)).

Hệ thống Phân Tán (Distributed): Phân phối tính toán cho nhiều bộ xử lý vật lý (ở những máy tính khác nhau), mỗi bộ xử lý có bộ nhớ riêng (hệ thống ghép đôi lỏng lẻo). Giao tiếp qua mạng. Lợi ích: chia sẻ tài nguyên, tăng tốc độ tính toán (cân bằng tải), tin cậy. Hạn chế: Chậm hơn hệ thống đa xử lý

Gồm:



- Client-Server: Một số hệ thống tập trung hoạt động như máy phục vụ, thỏa mãn yêu cầu từ các hệ thống khách hàng.

- Peer-to-Peer: Các máy tính ngang hàng, không phân biệt client hay server, có thể tự đăng ký dịch vụ hoặc dùng giao thức khám phá. Trong một hệ thống client-server, máy chủ có thể trở thành điểm nghẽn cổ chai. Ngược lại, trong mô hình P2P, các dịch vụ có thể được cung cấp bởi nhiều nút phân tán khắp mạng, giúp tăng hiệu suất và khả năng mở rộng (Skype).

Hệ thống Cụm (Clustered): Hai hay nhiều máy tính được nhóm lại để hoạt động như một máy tính độc nhất. Mục đích: chia sẻ thiết bị lưu trữ, cân bằng tải, xử lý song song. Cung cấp khả năng sẵn dùng cao, chịu lỗi và độ tin cậy cao. Ciao tiếp qua **mạng nội bộ**. Hệ điều hành **phối hợp với nhau**. Gồm: ghép cụm bất đối xứng và đối xứng.

- **Ghép cụm bất đối xứng:** các sever chạy ứng dụng trong khi một sever khác ở trạng thái chờ (hot standby). Khi sever hoạt động bị lỗi thì sever chờ sẽ hoạt động
- **Ghép cụm đối xứng:** tất cả các host cùng chạy ứng dụng và kiểm soát lẫn nhau để thay thế công việc cho nhau.

+ **Hệ thống chuyên dụng**:

Hệ thống Thời Gian Thực (Real-Time): Thường dùng làm thiết bị điều khiển trong ứng dụng chuyên biệt, có ràng buộc về thời gian cố định. Gồm:

- Hệ thống Thời Gian Thực Cứng (Hard Real-Time): Đảm bảo các tác vụ tới hạn phải hoàn thành đúng giờ, hạn chế trì hoãn. Không dùng phần cứng thứ cấp (ổ đĩa), lưu trữ trong bộ nhớ ngắn kỳ, hoặc rom. Không hỗ trợ hệ điều hành đa năng

- Hệ thống Thời Gian Thực Mềm (Soft Real-Time): Tác vụ thời thực có độ ưu tiên cao hơn, có thể dùng trong HĐH đa năng nhưng ít được dùng trong điều khiển công nghiệp do dễ rủi ro. Hữu dụng trong các ứng dụng yêu cầu tính năng cao cấp của hệ điều hành (đa phương tiện, thực tế ảo).

Hệ thống Cầm Tay (Handheld): Bao gồm PDA, điện thoại di động. Các vấn đề: bộ nhớ giới hạn, bộ xử lý chậm, màn hình nhỏ

IV. Sự phát triển của hệ điều hành

Xu hướng chính:

- Từ hệ thống **không có phần mềm** đến hệ điều hành phức tạp và mạnh mẽ.
- Từ **đơn nhiệm, đơn người dùng** đến **đa nhiệm, đa người dùng, phân tán, mạng hóa, chịu lỗi**.
- **UNIX** đóng vai trò nền tảng quan trọng trong sự phát triển OS.

- Xu hướng mở rộng từ **mainframe → minicomputer → desktop → handheld**.

V. Các môi trường điện toán

- HĐH được sử dụng để thiết lập các môi trường tính toán khác nhau:

+ **Tính toán truyền thống (traditional computing)**: môi trường văn phòng, gia đình, thông qua mạng.

+ **Tính toán kiểu web (web-based computing)**: mở rộng môi trường tính toán thông qua nền tảng web, hỗ trợ nhiều thiết bị.

+ **Tính toán kiểu hệ thống nhúng (embedded computing)**: các máy tính chạy HĐH thời gian thực nhúng, phục vụ các tác vụ chuyên biệt.

Chương 2: Cấu trúc hệ điều hành

I. Các thành phần của Hệ điều hành:

Hệ điều hành là một hệ thống phức tạp, bao gồm nhiều thành phần với đầu vào, đầu ra và chức năng được định nghĩa rõ ràng. Các thành phần chính là:

1. Quản lý tiến trình (Process management):

- Tiến trình là **một chương trình đang thực thi**

- Cần tài nguyên như CPU, bộ nhớ, tập tin, thiết bị vào ra.

- Bộ quản lý tiến trình chịu trách nhiệm **tạo/hủy, ngừng/tiếp tục tiến trình**, và cung cấp các **cơ chế đồng bộ hóa**, giao tiếp giữa các tiến trình, và **chống tắc nghẽn** (deadlock).

2. Quản lý bộ nhớ chính (Main-memory management):

- Bộ nhớ là một mảng lớn các words hoặc bytes với địa chỉ riêng biệt, dùng để lưu trữ dữ liệu truy cập nhanh, được chia sẻ bởi CPU và các thiết bị I/O. Là thiết bị lưu trữ bay hơi, bị mất nội dung khi hệ thống gặp sự cố.

- Bộ quản lý bộ nhớ chính **theo dõi** việc sử dụng bộ nhớ, quyết định **tiến trình nào** được **nạp vào bộ nhớ**, và **cấp phát/thu hồi không gian nhớ**.

3. Quản lý hệ thống tập tin (File management):

Tập tin là tập hợp các thông tin liên quan, dùng để lưu chương trình hoặc dữ liệu trên các thiết bị lưu trữ. Bộ quản lý tập tin chịu trách nhiệm **tạo/xóa tập tin**, thư mục, hỗ trợ thao tác trên chúng**, ánh xạ** tập tin lên thiết bị lưu trữ thứ cấp và **sao lưu**.

4. Quản lý hệ thống nhập/xuất (I/O management):

- Bao gồm hệ thống **lưu trữ đệm** (buffering(tạm lưu dữ liệu giữa thiết bị IO và tiến trình), **caching** (lưu cache), **spooling** (dữ liệu chờ xử lý)), giao diện điều khiển thiết bị tổng quát (lớp trung gian giữa thiết bị và phần cứng), và trình điều khiển thiết bị cụ thể (giao tiếp trực tiếp với phần cứng).

- Thành phần này giao tiếp với các thành phần khác để **quản lý thiết bị**, chuyển tải dữ liệu và **phát hiện hoàn thành I/O**.

5. Quản lý hệ thống lưu trữ thứ cấp (Secondary storage management):

Do bộ nhớ chính dễ bay hơi và có **kích thước giới hạn**, các thiết bị lưu trữ thứ cấp (như đĩa từ) được dùng để lưu trữ dữ liệu và chương trình **lâu dài**. Bộ quản lý đĩa chịu trách nhiệm quản lý **không gian trống**, cấp phát không gian lưu trữ và **định thời** sử dụng đĩa.

6. Hệ thống kết nối mạng (Networking):

- Trong hệ thống phân tán, các bộ xử lý không **chia sẻ bộ nhớ hoặc xung đồng hồ**, giao tiếp qua mạng truyền thông.

- Một số giao tiếp như: FTP, NFS, HTTP

- Hệ thống này giúp **tăng tốc độ tính toán**, mức độ sẵn dùng của dữ liệu và độ tin cậy.

7. Hệ thống bảo vệ (Protection system):

Đề cập đến cơ chế điều khiển truy cập của chương trình, tiến trình hoặc người dùng đến tài nguyên hệ thống và người dùng. Cơ chế bảo vệ phải phân biệt được truy cập có thẩm quyền, xác định quyền kiểm soát có nguy cơ bị chiếm bất hợp pháp, và **cung cấp** phương tiện bảo vệ an ninh.

8. Giao diện người dùng (User interface):

Thông dịch lệnh (Command-line Interpreter - CLI):

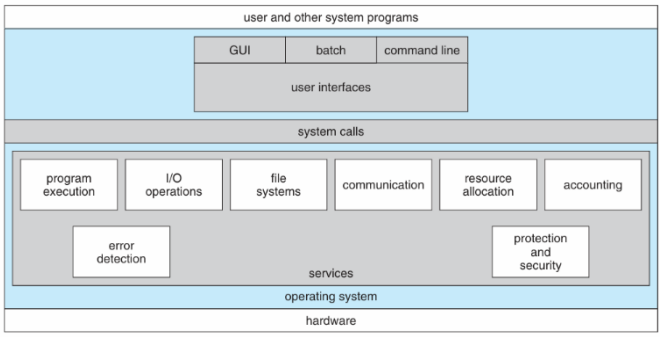
Nhận và thực hiện các câu lệnh điều khiển của người dùng để quản lý tiến trình, I/O, bộ nhớ, truy cập hệ thống tập tin. CLI có thể được cài đặt trong kernel (như DOS) hoặc thông qua các chương trình hệ thống (như Windows, Unix), còn gọi là shell. Các lệnh có thể là built-in của shell hoặc tên của một chương trình.

Môi trường nền (Desktop Environment - DE):

Giao diện người dùng đồ họa (GUI) như Windows DE, GNOME DE, KDE, cung cấp các biểu tượng, cửa sổ, thanh công cụ, thư mục, hình nền và khả năng kéo thả. DE bao gồm trình quản lý cửa sổ, trình quản lý tập tin, tập hợp các chủ đề, chương trình và thư viện quản lý desktop.

II. Các dịch vụ của hệ điều hành

- Các dịch vụ của Hệ điều hành: Hệ điều hành cung cấp các dịch vụ cho người dùng và các chương trình hệ thống khác.



- Dịch vụ cho Chương trình và Người dùng:

+ **Giao diện người dùng** (CLI, giao diện theo cụm, GUI).

+ **Thực thi chương trình** (nạp và thực thi chương trình).

+ **Thao tác I/O**: cung cấp phương tiện để thao tác I/O

+ **Thao tác hệ thống tập tin** (đọc, ghi, tạo, xóa tập tin).

+ **Giao tiếp** (chuyển thông tin giữa các tiến trình trên cùng máy hoặc qua mạng, dùng bộ nhớ chia sẻ hoặc truyền thông điệp).

+ **Phát hiện lỗi** (tại CPU, bộ nhớ, thiết bị I/O hoặc chương trình người dùng để đảm bảo tính chính sát).

- Dịch vụ cho Hệ thống (không dùng để hỗ trợ người dùng mà để **đảm bảo hoạt động** hiệu quả):

+ **Cấp phát tài nguyên** (cho nhiều người dùng hoặc nhiều công việc).

+ **Tính chi phí** (ghi lại việc sử dụng tài nguyên để tính tiền hoặc thống kê).

+ **Bảo vệ** (kiểm soát tất cả truy cập đến hệ thống).

III. Lời gọi hệ thống

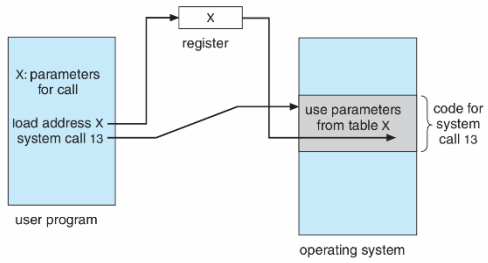
- Lời gọi hệ thống (System Calls): Là giao diện giữa tiến trình và hệ điều hành để gọi các dịch vụ của HĐH.

- Thường được hỗ trợ dưới dạng các chỉ thị **hợp ngữ** hoặc thông qua các giao diện lập trình ứng dụng (**API**) cấp cao hơn như **Windows API, POSIX API, Java API.**

- Một lời gọi thường đi kèm theo các **tham số**

- Tham số cho lời gọi hệ thống có thể được truyền qua thanh ghi (giới hạn vì tương đối ít), bộ nhớ (địa chỉ bảng/khối trong thanh ghi), hoặc stack(push tham số vào stack và lấy bằng cách pop).

- Ví dụ truyền qua bộ nhớ:



+ User program: Chạy ở User mode. Gồm có: vùng nhớ X chứa các tham số cho lời gọi hệ thống và lệnh như trong hình (nạp địa chỉ của vùng chứa tham số (X) vào thanh ghi, sau đó thực hiện lời gọi hệ thống số 13)

+ Register (thanh ghi): Nơi tạm thời lưu địa chỉ x để truyền vào hệ điều hành. Được CPU sử dụng để lấy tham số trong vùng bộ nhớ khi thực hiện lời gọi hệ thống.

+ Operating System: Đọc địa chỉ từ thanh ghi, truy cập đến vùng nhớ x, lấy tham số và thực hiện lời gọi hệ thống số 13

- Các kiểu lời gọi hệ thống bao gồm: **điều khiển tiến trình, quản lý file, quản lý thiết bị, duy trì thông tin trạng thái, giao tiếp, và các dịch vụ bảo vệ.**

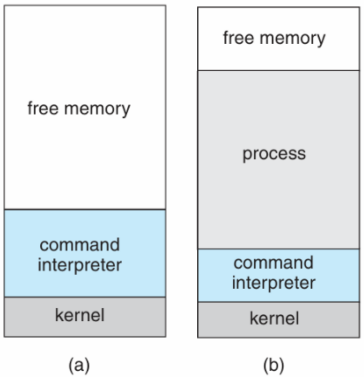
- Trong UNIX, trình thông dịch lệnh (shell) có thể tiếp tục chạy trong khi một chương trình khác được thực thi vì nó là một hệ thống đa nhiệm.

Thực thi chương trình trong MS-DOS

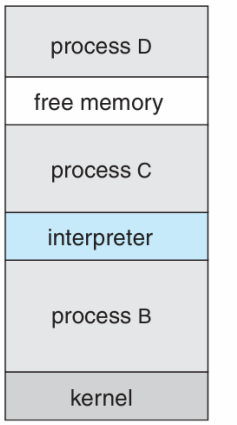
- Trong sơ đồ (b), bộ nhớ của tiến trình mới đã ghi đè lên vùng của shell — đây là lý do ta thấy "command interpreter" nhỏ lại hoặc biến mất.

- MS-DOS là một chương trình đơn nhiệm: tại 1 thời điểm chỉ có 1 chương trình được thực thi

Thành phần	Hình (a)	Hình (b)
Tiến trình người dùng	Chưa có tiến trình nào	Đã có tiến trình đang chạy
Bộ nhớ trống	Nhiều	Giảm đi
Trạng thái hệ thống	Hệ điều hành sẵn sàng chờ lệnh	Hệ điều hành đang thực thi chương trình



Thực thi chương trình trong UNIX

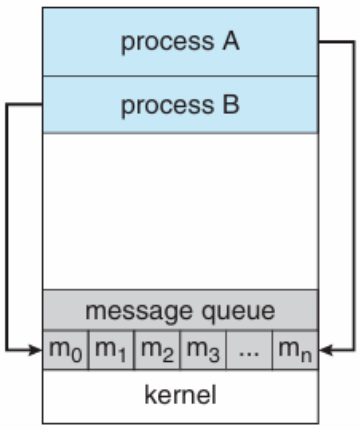


- UNIX là một chương trình đa nhiệm: tại 1 thời điểm có thể có nhiều chương trình được thực thi

- Cơ chế giao tiếp giữa các tiến trình: **Bộ nhớ chia sẻ** hay **chuyển thông điệp**

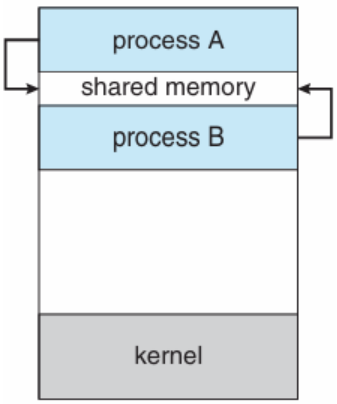
+ Chuyển thông điệp:

- **Mô tả:**
  - Các tiến trình (A và B) không chia sẻ vùng nhớ nào với nhau.
  - Để gửi dữ liệu, tiến trình A tạo một thông điệp (message) rồi gửi vào hàng đợi thông điệp (message queue) do kernel quản lý.
  - Tiến trình B đọc thông điệp đó từ hàng đợi để nhận dữ liệu.
- **Đặc điểm:**
  - Đơn giản, an toàn vì mỗi tiến trình có bộ nhớ riêng
  - Có kernel làm vai trò trung gian việc gửi/nhận
  - Phù hợp hệ thống phân tán bảo mật cao
  - Dễ kiểm soát, gỡ lỗi
  - Hiệu suất kém cho phải chuyển qua kernel



+ Bộ nhớ chia sẻ:

- **Mô tả:**
  - Cả tiến trình A, B cùng truy cập vào một vùng nhớ dùng chung



    - Dữ liệu chỉ cần được ghi một lần trong đó, tiến trình còn lại có thể đọc trực tiếp
- **Đặc điểm:**
    - Cần phối hợp để tránh xung đột dữ liệu
    - Vùng nhớ chia sẽ được tạo và cấu hình bởi hệ điều hành
    - Nhanh và hiệu quả hơn
    - Hiệu năng cao
    - Cẩn thận khi dùng, dễ xung đột, lỗi đồng bộ, phức tạp khi lập trình

IV. Các chương trình hệ thống

1. Vai trò của chương trình hệ thống:

- Cung cấp **môi trường thuận lợi** cho việc **phát triển và thực thi chương trình ứng dụng**.

- Đơn giản hóa việc tương tác với phần cứng thông qua các **lời gọi hệ thống** (system calls), từ đó giúp người dùng và lập trình viên không cần thao tác trực tiếp với phần cứng.

- Là cầu nối giữa **hệ điều hành (HĐH)** và các chương trình ứng dụng, đảm bảo hoạt động ổn định và hiệu quả cho toàn bộ hệ thống máy tính.

2. Các loại chương trình hệ thống:

- Quản lý tệp tin: Cho phép thao tác với tệp tin như tạo, đọc, ghi, xóa và chỉnh sửa tệp.

- Thông tin trạng thái: Hiển thị và ghi nhận trạng thái của hệ thống (CPU, bộ nhớ, dung lượng lưu trữ, tiến trình đang chạy,...).

- Hỗ trợ ngôn ngữ lập trình: Bao gồm các trình biên dịch, thông dịch, thư viện hệ thống và môi trường thực thi.

- Nạp và thực thi chương trình: Chịu trách nhiệm nạp mã chương trình vào bộ nhớ và khởi động tiến trình.

- Giao tiếp hệ thống: Đảm bảo khả năng **giao tiếp giữa các tiến trình, giữa người dùng với hệ thống**, và **giao tiếp mạng giữa các máy tính**.

3. Mối quan hệ giữa người dùng và chương trình hệ thống:

- Người dùng **hầu như chỉ tương tác với HĐH thông qua các chương trình hệ thống**, mà không cần làm việc trực tiếp với nhân hệ điều hành.

- Các chương trình hệ thống giúp trừu tượng hóa phần cứng và cung cấp giao diện dễ sử dụng hơn cho người dùng cuối.

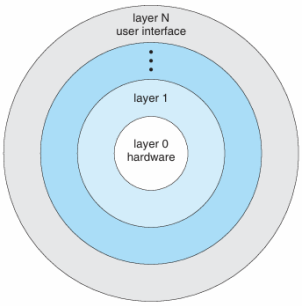
V. Kiến trúc hệ điều hành

- Là cách thức tổ chức các thành phần HĐH để xác định đặc quyền mà mỗi thành phần thực hiện.

- 3 loại kiến trúc:

+ **Kiến trúc nguyên khối (Monolithic)**: Tất cả các thành phần chứa trong nhân (kernel).

+ **Kiến trúc phân tầng (Layered)**: Hệ điều hành được chia thành các tầng, mỗi tầng được xây dựng trên nền tảng của tầng thấp hơn. Tầng thấp nhất là phần cứng vật lý, tầng cao nhất là giao diện người dùng.

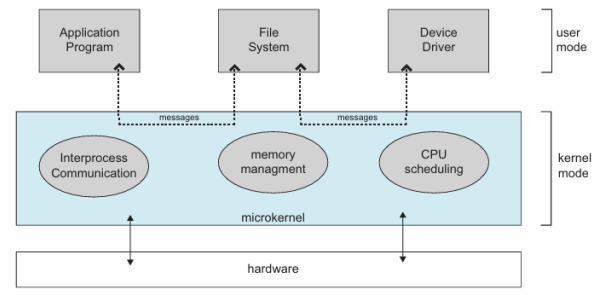


Ưu điểm: Tính module, đơn giản hóa thiết kế, cài đặt, gỡ rối và kiểm tra. Dễ sửa đổi, cải tiến tại từng tầng mà không ảnh hưởng đến các tầng khác.

Nhược điểm: Khó khăn trong việc định nghĩa chức năng tầng (vì mỗi tầng chỉ sử dụng các tầng dưới nó), khó khăn trong việc xác định một chức năng của hệ điều hành và tăng chi phí cho lời gọi hệ thống qua nhiều tầng.

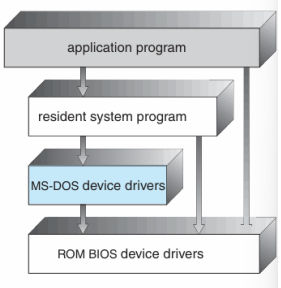
+ **Kiến trúc vi nhân (Microkernel)**: Di chuyển nhiều chức năng từ nhân lên mức người dùng. Việc giao tiếp giữa các module người dùng được thực hiện bằng cơ chế chuyển thông điệp.

- **Lợi ích**:
  - Dễ mở rộng, dễ chuyển đổi sang kiến trúc mới, tin cậy và an toàn hơn (ít mã lệnh chạy ở mức nhân hơn).
- **Nhược điểm**: Tăng chi phí giao tiếp giữa tiến trình người dùng và nhân.

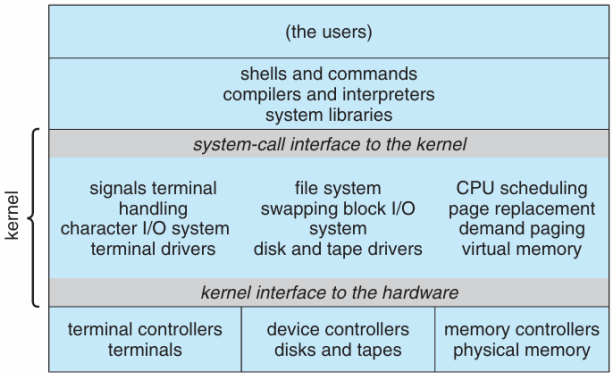


Kiến trúc một số hệ điều hành phổ biến:

+ **MS-DOS**: Không có kiến trúc rõ ràng, được viết để cung cấp nhiều chức năng nhất với dung lượng nhỏ nhất, không chia thành module, các lớp chức năng không được phân chia tốt.



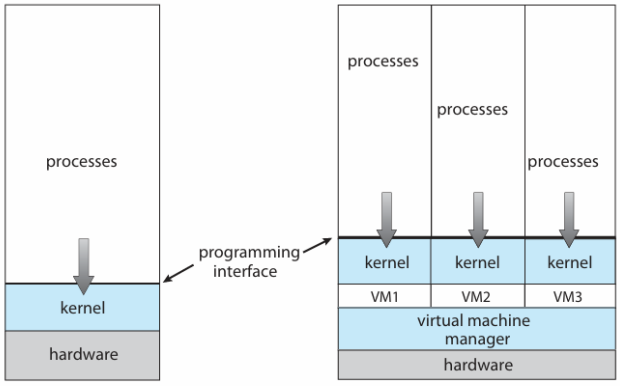
+ **UNIX Cổ điển**: Có kiến trúc giới hạn do phần cứng. Gồm hai phần tách biệt: các chương trình hệ thống và nhân (kernel). Nhân bao gồm quản lý hệ thống tập tin, định thời CPU, quản lý bộ nhớ và các chức năng khác.



+ **Máy ảo (Virtual machine)**: Sử dụng mô hình phân tầng, **xem** phần cứng và nhân HĐH đều là **phần cứng ảo (phần mềm).** Các tài nguyên thực của máy tính được chia sẻ để tạo ra các máy ảo, mang lại cảm giác nhiều máy khác nhau, mỗi máy sử dụng phần CPU và bộ nhớ ảo riêng giống như hệ thống truyền thống.

Ưu điểm: Cung cấp cơ chế bảo vệ tuyệt đối do mỗi máy ảo tách biệt, là công cụ hoàn hảo cho nghiên cứu và phát triển HĐH.

Nhược điểm: Khó cài đặt do yêu cầu cung cấp bản sao chính xác phần cứng máy thật, và sự tách biệt không cho phép chia sẻ tài nguyên trực tiếp.



Mô hình hệ thống không máy ảo — Mô hình hệ thống máy ảo

VI. Thiết kế, cài đặt hệ điều hành

- Thiết kế: Bắt đầu bằng việc xác định mục tiêu và đặc tả kỹ thuật (phần cứng, kiểu HĐH). Yêu cầu đối với người dùng là dễ dùng, dễ học, tin cậy, an toàn và nhanh. Yêu cầu đối với hệ thống là dễ thiết kế, cài đặt, bảo trì, linh hoạt, tin cậy, không lỗi và hiệu quả.

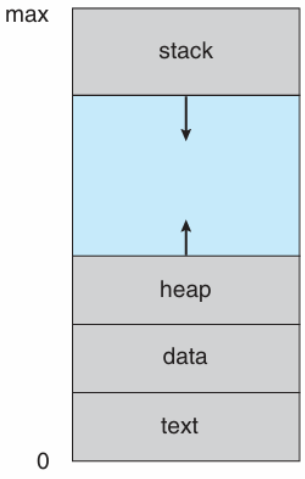
- Cơ chế (Mechanism) và Chính sách (Policy): Cơ chế xác định ***cách* thực hiện một việc**, còn chính sách quyết định *cái gì* được làm. Việc tách biệt hai khái niệm này giúp đạt được sự linh hoạt tối đa, cho phép thay đổi các quyết định chính sách sau này.

- Cài đặt: Thay vì viết bằng hợp ngữ truyền thống, ngày nay HĐH có thể được viết bằng ngôn ngữ lập trình cấp cao (như C, C++). Điều này giúp mã lệnh được viết **nhanh hơn, gọn hơn, dễ hiểu, sửa lỗi hơn** và dễ **chuyển đổi sang hệ thống phần cứng khác.**

Chương 3: Tiến trình (Process)

I. Khái niệm cơ bản về Tiến trình

- **Định nghĩa**: Tiến trình là một thể hiện (instance) của một chương trình máy tính đang được thực thi hoặc chờ thực thi trong bộ nhớ.
- **Định danh**: Mỗi tiến trình thường được gán một số định danh tiến trình (PID) duy nhất để nhận diện.
- **Thành phần của một tiến trình**: Một tiến trình bao gồm **mã lệnh chương trình** (program code), **bộ đếm chương trình** (program counter) và **các thanh ghi CPU**, **ngăn xếp** (stack), **phần dữ liệu** (data section), **và có thể có phần bộ nhớ cấp phát động** (heap).
    - **Program counter**: Bộ đếm chương trình, ghi nhận chỉ thị lệnh kế tiếp được thực thi



- **Chương trình và Tiến trình**: Chương trình là một thực thể bị động được lưu trữ trên đĩa, trong khi tiến trình là một thực thể chủ động lưu trú trên bộ nhớ chính. Khi một chương trình được kích hoạt, một thể hiện của nó sẽ được **nạp** vào bộ nhớ để tạo thành **một tiến trình**. Một chương trình có thể có **nhiều** tiến trình trong bộ nhớ.
- **Sơ đồ bố trí bộ nhớ**: gồm
    - **Text segment (vùng văn bản)**:
        - Chứa mã lệnh (code) của chương trình.
        - Là vùng chỉ đọc (read-only) để tránh bị sửa đổi trong khi chạy.
    - **Data segment (vùng dữ liệu)**:
        - Chứa các biến toàn cục (global) và tĩnh (static) đã được khởi tạo.
    - **Heap (vùng cấp phát động)**:
        - Dùng cho bộ nhớ cấp phát động (qua `malloc`, `new`,...).
        - Phát triển từ dưới lên (tăng địa chỉ).
    - **Vùng trống ở giữa (màu xanh dương)**:
        - Là **vùng mở rộng động giữa heap và stack**.
        - Đảm bảo không gian linh hoạt cho cả heap (phát triển lên) và stack (phát triển xuống).
    - **Stack (ngăn xếp)**:
        - Chứa dữ liệu tạm thời như biến cục bộ, địa chỉ trả về từ hàm,...
        - Phát triển từ trên xuống (giảm địa chỉ).

II. Trạng thái của Tiến trình (Process State)

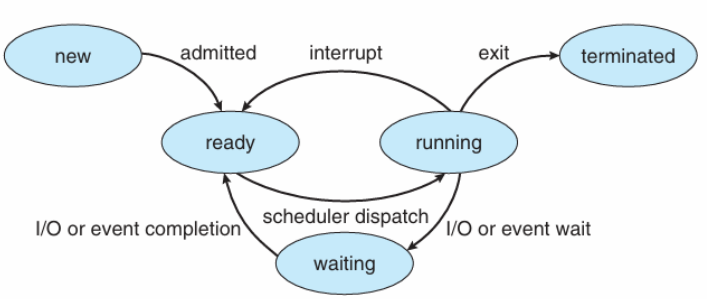
Một tiến trình có thể ở một trong các trạng thái sau:

- **New**: Tiến trình đang được khởi tạo.
- **Running**: Các chỉ thị của tiến trình đang được thực thi.
- **Waiting**: Tiến trình đang chờ một sự kiện nào đó xảy ra (ví dụ: hoàn thành I/O, tín hiệu từ tiến trình khác, …).
- **Ready**: Tiến trình sẵn sàng để thực thi (đang đợi được cấp CPU).
- **Terminated**: Tiến trình đã kết thúc.

- **Sơ đồ chuyển trạng thái**: Chương trình cũng cung cấp một sơ đồ minh họa sự chuyển đổi giữa các trạng thái này (ví dụ: từ new sang ready qua admitted, từ running sang waiting khi chờ I/O, v.v.)[13].

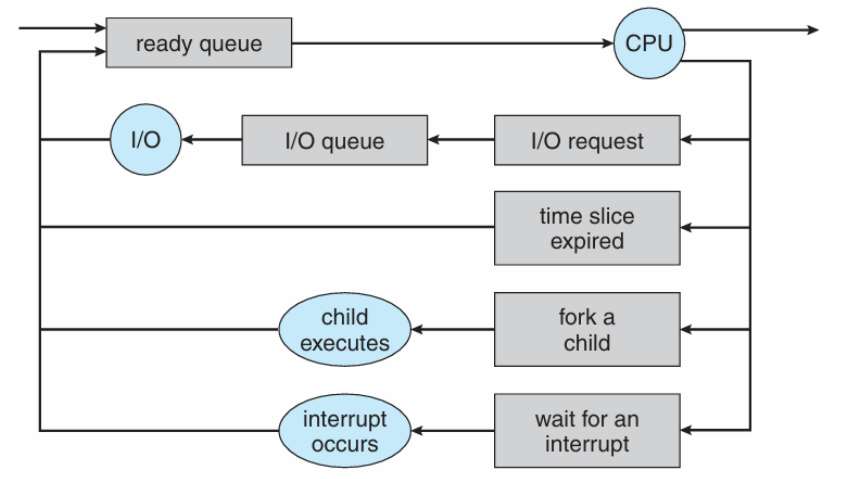
Lưu ý:

- Có thể có nhiều tiến trình ở trạng thái new, ready, waiting
- Chỉ có 1 tiến trình ở trạng thái running
- Ở trạng thái ready, tiến trình đã đầy đủ tài nguyên, chỉ còn thiếu mỗi CPU



- Chi phí chuyển đổi ngữ cảnh do phần cứng chịu

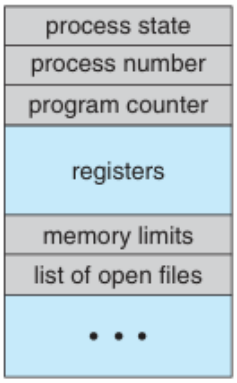
Sơ đồ định thời CPU:



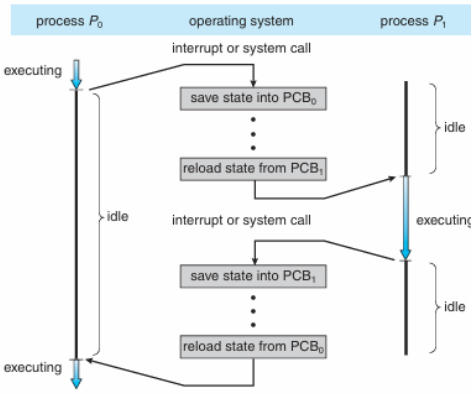
- Nếu tiến trình tạo tiến trình con (fork()) → hệ thống chuyển sang cho child process chạy. Sau đó, cả tiến trình cha và con đều quay lại các hàng đợi phù hợp.

III. Khối điều khiển Tiến trình (Process Control Block - PCB)

- **Chức năng**: PCB chứa tất cả các thông tin cần thiết về một tiến trình mà hệ điều hành quản lý[14].

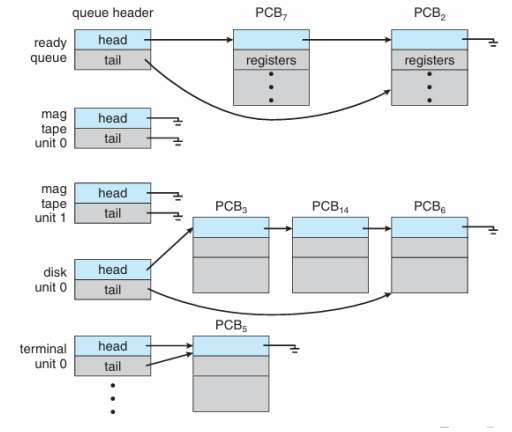


- **Thông tin lưu trữ**: Bao gồm
  - Trạng thái của quá trình (ready, running,...)
  - Bộ đếm chương trình (program counter)
  - Các thanh ghi
  - Thông tin định thời CPU: lập lịch CPU
  - Thông tin quản lý bộ nhớ: theo dõi vùng nhớ riêng của tiến trình
  - Thông tin chi phí (thời gian sử dụng CPU, PID,...)
  - Thông tin trạng thái nhập/xuất (thiết bị được cấp phát, danh sách tập tin đang mở,...).
- **Chuyển CPU giữa các tiến trình (Context Switch)**:
  - PCB là nơi lưu giữ trạng thái của tiến trình.
  - Khi có một ngắt xảy ra, trạng thái của tiến trình hiện tại phải được lưu vào PCB của nó, và trạng thái của tiến trình mới sẽ được nạp từ PCB của nó để tiếp tục thực thi[16].
  - Tác vụ này còn được gọi là chuyển ngữ cảnh (context switch)[17].



IV. Định thời cho Tiến trình (Process Scheduling)

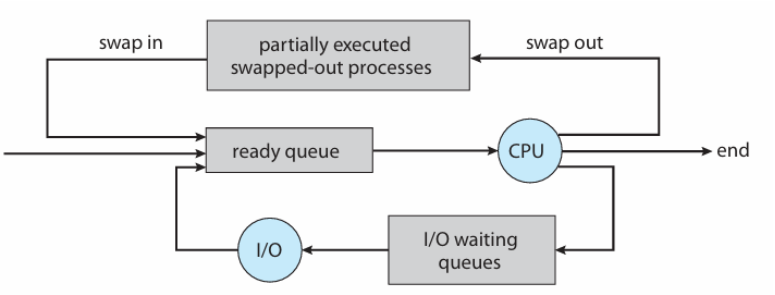
- **Mục đích**: Là tác vụ của hệ điều hành để lựa chọn tiến trình nào sẽ được cấp CPU và phân bổ thời gian sử dụng CPU cho tiến trình đó trong các hệ thống đa chương phân chia thời gian[18].
- **Bộ định thời tiến trình (Process Scheduler)**: Là thành phần chịu trách nhiệm lựa chọn và định thời các tiến trình, sử dụng một hệ thống các hàng đợi[19].
- **Các hàng đợi tiến trình (Process Queues)**:
  - **Hàng đợi công việc (job queue)**: Tập hợp tất cả các tiến trình trong hệ thống.
  - **Hàng đợi sẵn sàng (ready queue)**: Tập hợp các tiến trình đang ở trong bộ nhớ, sẵn sàng chờ để thực thi.
  - **Hàng đợi thiết bị (device queue)**: Tập hợp các tiến trình đang chờ sử dụng một thiết bị vào ra.



  - Các tiến trình có thể **di chuyển** giữa các hàng đợi khác nhau.
- **Các loại bộ định thời (Schedulers)**:
  - **Bộ định thời dài kỳ (long-term scheduler/job scheduler)**: Chọn tiến trình nào sẽ được nạp vào hàng đợi sẵn sàng (nạp vào bộ nhớ). Hoạt

động không thường xuyên (giây, phút) và kiểm soát cấp độ đa chương. Khi **click vào ứng dụng**

- **Bộ định thời ngắn kỳ (short-term scheduler/CPU scheduler)**: Chọn tiến trình sẽ được thực thi kế tiếp và cấp CPU cho nó. Hoạt động rất thường xuyên (mili giây) và yêu cầu tốc độ nhanh.
- **Bộ định thời trung kỳ (medium-term scheduler)**: Là mức **Trung gian** giữa bộ định thời ngắn và dài kỳ. Thực hiện hoán vị (**swapping**) các tiến trình ra/vào bộ nhớ/đĩa do cạnh tranh tài nguyên CPU, bộ nhớ. Thường dùng trong các hệ thống **phân chia thời gian**[26].



Đặc điểm	Dài kỳ	Trung kỳ	Ngắn kỳ
Tên tiếng Anh	Long-term scheduler	Medium-term scheduler	Short-term scheduler
Vai trò	Chọn tiến trình vào RAM	Swap tiến trình ra/vào	Chọn tiến trình chạy CPU
Tần suất hoạt động	Ít	Trung bình	Rất thường xuyên
Tốc độ	Chậm	Trung bình	Rất nhanh
Mục tiêu	Điều phối tải hệ thống	Quản lý bộ nhớ	Tối ưu sử dụng CPU

V. Các thao tác trên Tiến trình

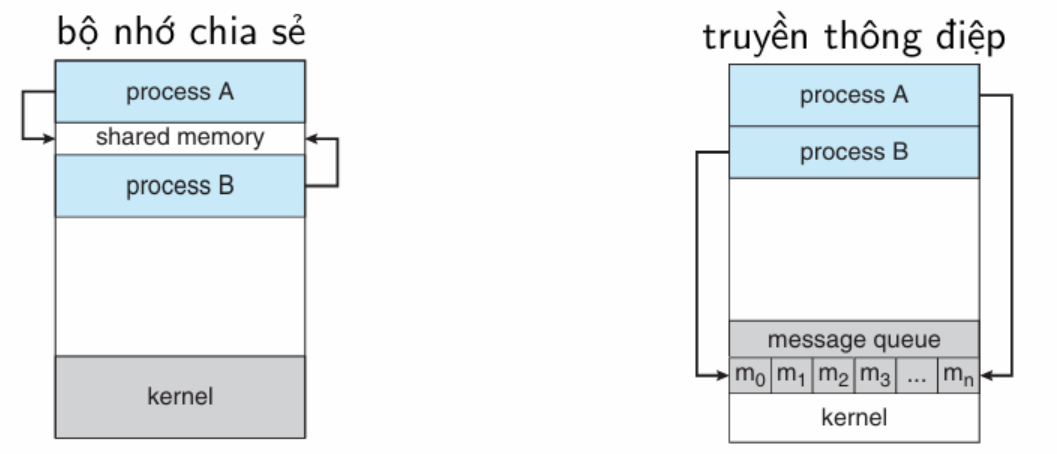
Có hai thao tác cơ bản:

- **Tạo tiến trình**: Một tiến trình cha có thể tạo ra các tiến trình con, hình thành cây tiến trình. Các vấn đề liên quan bao gồm:
  - **Chia sẻ tài nguyên:**
    - Tiến trình cha và con chia sẻ tất cả tài nguyên
    - Tiến trình cha chia sẻ một phần tài nguyên cho tiến trình con
    - Tiến trình cha và con không chia sẻ gì cả
  - **Dữ liệu khởi tạo:** được chuyển từ tiến trình cha sang con
  - **Thực thi song song** hoặc **tuần tự**
  - **Không gian địa chỉ:** tiến trình con **sao chép** từ tiến trình cha (cả code và dữ liệu) hoặc **tự nạp chương trình riêng**
- **Tạo tiến trình trên UNIX và Windows NT**:
  - **UNIX**: Sử dụng fork() để tạo tiến trình mới và execlp() để thay thế không gian địa chỉ của tiến trình gọi bằng một chương trình mới.

  - **Windows NT**: Sử dụng CreateProcess() để tạo tiến trình con và thay thế không gian địa chỉ bằng một tiến trình mới. Tiến trình mới được chỉ định trong đối số của lời gọi hệ thống
- **Kết thúc tiến trình**: Tiến trình tự kết thúc bằng lệnh exit()[33]. Hệ điều hành thu hồi tài nguyên đã cấp phát cho tiến trình[34].
  - **Zombie**: Tiến trình con kết thúc trước khi tiến trình cha gọi wait()[35].
  - **Orphan**: Tiến trình con còn thực thi khi tiến trình cha đã kết thúc[36].
  - Tiến trình cha có thể kết thúc tiến trình con bằng abort() vì các lý do như vượt quá tài nguyên, công việc không còn cần thiết, hoặc khi tiến trình cha thoát[37].
- Tiến trình cha có thể kết thúc tiến trình con (abort()), khi:
  - Tiến trình con đã có *vượt quá số tài nguyên* được cấp. Mỗi tiến trình được hệ điều hành cấp một lượng tài nguyên nhất định (bộ nhớ, CPU time, I/O,...). Nếu tiến trình con tiêu tốn quá mức giới hạn → tiến trình cha có thể **chủ động hủy** nó để bảo vệ hệ thống hoặc các tiến trình khác.
  - **Công việc giao cho tiến trình con làm nay** *không còn cần thiết nữa***.** Ví dụ: tiến trình cha tạo tiến trình con để xử lý dữ liệu. Nếu dữ liệu đầu vào bị lỗi hoặc kết quả không cần dùng → tiến trình cha **không cần giữ tiến trình con chạy nữa** → gọi `abort()` để giải phóng tài nguyên.
  - **Tiến trình cha đang thoát:** một vài HĐH không cho phép orphan

VI. Hợp tác Tiến trình (Cooperating Process)

- **Tiến trình độc lập**: Không ảnh hưởng hoặc không bị ảnh hưởng bởi quá trình thực thi của các tiến trình khác.
- **Hợp tác tiến trình**: Có thể ảnh hưởng hoặc bị ảnh hưởng bởi quá trình thực thi của các tiến trình khác.
  - **Lợi ích**: Chia sẻ thông tin, tăng tốc độ tính toán (trên máy đa CPU: thực hiện song song), module hóa, tiện dụng.
- **Giao tiếp liên tiến trình (Interprocess Communication - IPC)**: Các tiến trình trao đổi dữ liệu với nhau thông qua IPC, bao gồm:



  - **Bộ nhớ chia sẻ (Shared Memory)**:

- Các tiến trình giao tiếp thiết lập một vùng **bộ nhớ chia sẻ**. Vùng này thường nằm trong không gian địa chỉ của tiến trình tạo ra phân đoạn bộ nhớ chia sẻ. Các tiến trình khác muốn giao tiếp phải gắn vùng này vào không gian địa chỉ của chúng:
  - Kích thước **không giới hạn** (unbounded buffer): tiến trình đọc có thể chờ, tiến trình ghi không bao giờ chờ
  - Kích thước có **giới hạn** (bounded buffer): cả tiến trình đọc và ghi có thể chờ
- Hệ điều hành thường ngăn chặn một tiến trình truy cập bộ nhớ của tiến trình khác, nhưng bộ nhớ chia sẻ đòi hỏi các tiến trình phải đồng ý loại bỏ hạn chế này.
- Thông tin được trao đổi bằng cách đọc và ghi dữ liệu trong vùng chia sẻ, và hình thức dữ liệu do các tiến trình quyết định, không phải HĐH. Các tiến trình có trách nhiệm đảm bảo **không ghi cùng lúc vào cùng một vị trí**.
- Ưu điểm là **tốc độ nhanh** vì không cần sự can thiệp của kernel sau khi thiết lập. Tuy nhiên, có thể gặp vấn đề về **tính nhất quán** của bộ đệm (cache coherency issues) trên các hệ thống đa lõi.
- Ví dụ kinh điển: bài toán "Nhà sản xuất – Người tiêu thụ". Dữ liệu được chia sẻ qua một vùng đệm (buffer).

- **Truyền thông điệp (Message Passing)**:
  - Giao tiếp giữa các tiến trình mà không cần bộ nhớ chia sẻ, hữu ích trong môi trường **phân tán**.
  - Sử dụng hai thao tác cơ bản: send(msg) và receive(msg).
  - Để 2 tiến trình P, Q giao tiếp với nhau: Tạo nối kết giao tiếp → trao đổi thông điệp
    - **Giao tiếp trực tiếp (Direct Communication)**: Các quá trình phải được đặt tên rõ ràng (ví dụ: Send(P, msg)). Các nối kết **được thiết lập tự động**, mỗi nối kết kết hợp với **chính xác một cặp** quá trình và chỉ có **một nối kết** giữa mỗi cặp. Nối kết có thể 1 hướng, nhưng thường là 2 hướng. Giao tiếp bất đối xứng: Send(P, msg), Receive(id, msg)
    - **Giao tiếp gián tiếp (Indirect Communication)**: Thông điệp được gửi và nhận thông qua mailbox (hoặc port). Mỗi mailbox có một ID duy nhất, và các quá trình chỉ có thể giao tiếp nếu chúng dùng chung mailbox. Send/Receive(A, msg): gửi/nhận thông điệp tới/từ hộp thư A. Một nối kết **có thể kết hợp** với nhiều quá trình. Mỗi cặp quá trình **có thể dùng chung nhiều nối kết giao tiếp**. Nối kết có thể **1 hướng hay 2 hướng**. Một liên kết chỉ tương ứng với 2 quá trình, chỉ cho phép thao tác nhận tại một điểm, **có thông báo** cho tiến trình gửi biết người nhận
  - **Đồng bộ hóa (Synchronization)**:

        - **Blocking (đồng bộ)**: Blocking send (tiến trình gửi chờ đến khi thông điệp được nhận), Blocking receive (tiến trình nhận chờ đến khi thông điệp sẵn sàng).
        - **Non-blocking (bất đồng bộ)**: Non-blocking send (gửi thông điệp và tiếp tục công việc khác) , Non-blocking receive (nhận một thông điệp hoặc rỗng).
    - **Tạo vùng đệm (Buffering)**: Vùng đệm chứa các thông điệp của một nối kết, có thể có sức chứa là 0 (gửi bị chặn đến khi nhận), sức chứa giới hạn (gửi bị chặn khi vùng đệm đầy), hoặc sức chứa không giới hạn (gửi không bao giờ bị chặn).

VII. Giao tiếp trong hệ thống Client-Server

Có ba phương pháp cơ bản để giao tiếp trong mô hình client-server, ngoài bộ nhớ chia sẻ và truyền thông điệp:

- **Socket**: Là một điểm nút giao tiếp của hai tiến trình, được hình thành từ một địa chỉ IP và một số hiệu cổng (port). Mỗi cặp tiến trình giao tiếp dùng một cặp socket (client socket và server socket). Hai tiến trình trong cùng hệ thống phải chạy các port khác nhau.
- **Remote Procedure Calls (RPCs)**: (Không đi sâu vào chi tiết trong tài liệu này).
- **Pipe (ống dẫn)**: (Không đi sâu vào chi tiết trong tài liệu này).

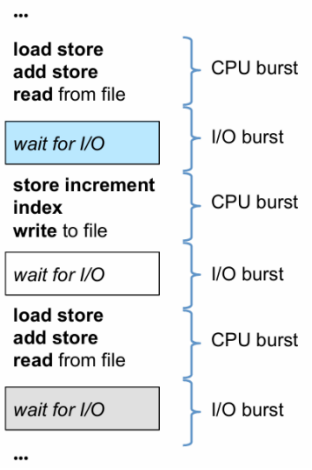
Chương 4: Định thời CPU

I. Các khái niệm cơ bản về Định thời CPU

- Định nghĩa: Định thời CPU là một chức năng cơ bản và quan trọng của các hệ điều hành **đa chương**

- Chức năng: chịu trách nhiệm **phân bổ thời gian** và **thời điểm** sử dụng CPU cho các tiến trình trong hệ thống.

- Mục tiêu: Tăng hiệu suất sử dụng CPU và **giảm thời gian đáp ứng** của hệ thống. Ý tưởng cơ bản là **phân bổ** thời gian rảnh rỗi của CPU (khi chờ đợi thao tác nhập/xuất của tiến trình đang chạy) cho các **tiến trình khác** trong hệ thống.



- Chu kỳ CPU–I/O (CPU–I/O Burst): Sự thực thi của tiến trình bao gồm nhiều chu kỳ CPU–I/O, mỗi chu kỳ gồm một **chu kỳ thực thi CPU (CPU burst)** và một **chu kỳ chờ đợi vào/ra (I/O burst)**.

+ **Phân bổ sử dụng CPU:** Chương trình **hướng nhập/xuất** (I/O-bound) thường có **nhiều chu kỳ CPU ngắn** (chủ yếu dành thời gian chờ đợi I/O), trong khi chương trình **hướng xử lý** (CPU-bound) thường có nhiều **chu kỳ CPU dài**.

- Bộ Định thời CPU (CPU Scheduler): Còn gọi là bộ định thời ngắn kỳ, có nhiệm vụ chọn một trong các tiến trình trong hàng đợi sẵn sàng và cấp phát CPU cho nó thực thi.

- Thời điểm quyết định định thời: Quyết định định thời xảy ra khi một tiến trình chuyển từ trạng thái đang chạy sang trạng thái chờ đợi, hoặc từ trạng thái đang chạy sang sẵn sàng, hoặc từ chờ đợi sang sẵn sàng, hoặc khi một tiến trình kết thúc.

- Định thời Trưng dụng và Không trưng dụng:

+ **Không trưng dụng (nonpreemptive scheduling):** Tiến trình được phân CPU có quyền sử dụng CPU đến khi nó sử dụng xong **(kết thúc hoặc chuyển sang trạng thái chờ).**

+ **Trưng dụng (preemptive scheduling):** Bộ định thời có thể thu hồi CPU của tiến trình **bất kỳ lúc nào** để phân cho tiến trình khác. Kiểu định thời này phức tạp hơn vì phải giải quyết:

- Sự cạnh tranh dữ liệu giữa các tiến trình

- Sự trưng dụng khi tiến trình đang thực thi trong chế độ kernel
- Dàn xếp giữa sự trưng dụng và xử lý các ngắt của hệ thống.

- Bộ Điều phối (Dispatcher): Có nhiệm vụ thực thi việc **trao quyền sử dụng CPU** cho tiến trình **được cấp phát** bởi bộ định thời. Bao gồm các **tác vụ** như **chuyển ngữ cảnh**, chuyển sang **chế độ người dùng,** và **nhảy** tới **vị trí** thích hợp trong chương trình.

- Độ trễ điều phối (Dispatcher latency): Là khoảng **thời gian** mà bộ điều phối **cần** để **ngưng** một tiến trình và **khởi động** một tiến trình khác.

- Tiêu chí cho việc định thời Các giải thuật định thời được đánh giá thông qua khả năng tối ưu hóa các tiêu chí sau:

+ Hiệu suất sử dụng CPU (CPU utilization): Tỷ lệ thời gian CPU được sử dụng trên tổng thời gian hoạt động của hệ thống. Mục tiêu: càng lớn càng tốt.

+ Thông lượng (Throughput): Số lượng tiến trình hoàn thành trên một đơn vị thời gian. Mục tiêu: càng lớn càng tốt.

+ Thời gian xoay vòng (Turnaround time): Tổng thời gian để thực thi một tiến trình, bao gồm thời gian thực thi, chờ I/O và chờ trong hàng đợi sẵn sàng. Mục tiêu: càng nhỏ càng tốt.

+ Thời gian chờ đợi (Waiting time): Tổng thời gian một tiến trình nằm trong hàng đợi sẵn sàng. Mục tiêu: càng nhỏ càng tốt.

+ Thời gian đáp ứng (Response time): Lượng thời gian từ lúc một yêu cầu được đệ trình cho đến khi bắt đầu được đáp ứng. Mục tiêu: càng nhỏ càng tốt.

II. Các giải thuật định thời CPU

1. First-Come, First-Served (FCFS):

- Là giải thuật đơn giản nhất, dựa trên nguyên tắc **đến trước, được phục vụ trước**

- Thường dùng hàng đợi **FIFO**.

- Ưu điểm: cài đặt dễ dàng, đơn giản, dễ hiểu

- Nhược điểm chính là thời gian chờ đợi trung bình thường dài và không thích hợp cho hệ thống phân chia thời gian do bản chất **không trưng dụng.**

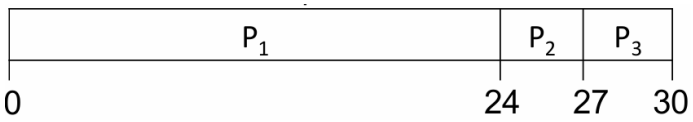
- Có thể xảy ra "hiệu ứng nối đuôi" **(convoy effect**) khi một tiến trình ngắn nằm sau một tiến trình dài.

Ví dụ:

Cho các tiến trình với thời gian thực thi và thứ tự xuất hiện như sau:

Process	TG sử dụng CPU	Thứ tự xuất hiện
P1	24	1
P2	3	2
P3	3	3

Biểu đồ Gantt cho lịch trình FCFS (First-Come, First-Served):



Tính thời gian chờ đợi cho từng tiến trình:

P1: 0, P2: 24, P3: 27

Thời gian chờ đợi trung bình:

(0+24+27)/3=17

2. Shortest-Job-First (SJF):

Cấp phát CPU cho tiến trình có thời gian thực thi CPU (CPU burst) kế tiếp nhỏ nhất. SJF được coi là tối ưu vì nó cho thời gian chờ đợi trung bình nhỏ nhất.

+ **Không trưng dụng:** Tiến trình chiếm giữ CPU đến khi thực thi xong CPU burst.

+ **Trưng dụng (Shortest-Remaining-Time-First – SRTF):** Nếu một tiến trình mới đến có CPU burst ngắn hơn thời gian thực thi còn lại của tiến trình đang chạy, CPU sẽ được thu hồi và cấp cho tiến trình mới.

- **Ước lượng thời gian CPU burst:** Thời gian CPU burst kế tiếp thường được dự đoán dựa trên lịch sử sử dụng CPU trước đó, sử dụng công thức trung bình mũ: $\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$, với α thường là 1/2.
  - $\alpha = 0 \Longrightarrow \tau_{n+1} = \tau_n = \ldots = \tau_0$ (nhất thời, không có ý nghĩa)
  - $\alpha = 1 \Longrightarrow \tau_{n+1} = t_n$ (thời gian sử dụng CPU thực tế gần đây nhất)
  - $\alpha = ½ \Longrightarrow$ thực tế và dự đoán có trọng số tương đương

3. Priority (Độ ưu tiên):

- Mỗi tiến trình được gán một chỉ số **ưu tiên**

- CPU sẽ được cấp phát cho tiến trình có chỉ số ưu tiên cao nhất.

- SJF là một trường hợp đặc biệt của giải thuật này.

- Có thể xảy ra tình trạng "chết đói" (**starvation**) đối với các tiến trình độ ưu tiên thấp, **giải pháp** là sử dụng cơ chế **"lão hóa"** (aging) **để tăng dần độ ưu tiên** của các tiến trình chờ đợi lâu.

4. Round-Robin (RR):

- Bộ điều phối cấp phát cho mỗi tiến trình trong hàng đợi sẵn sàng một đơn vị thời gian cố định gọi là **định mức thời gian (time-slice hoặc time quantum)**, thường từ 10 đến 100 miligiây.

- Sau khi hết định mức thời gian, CPU bị thu hồi và tiến trình chuyển vào cuối hàng đợi sẵn sàng. Nếu time quantum quá lớn, RR sẽ thoái hóa thành FCFS. Nếu quá nhỏ, số lần chuyển ngữ cảnh (context switch) sẽ rất lớn, gây hao phí. Một quy tắc kinh nghiệm là 80% các CPU burst nên ngắn hơn time quantum.

- Nếu hàng đợi sẵn sàng có n tiến trình, định mức thời gian là q:

+ Mỗi tiến trình sẽ nhận được **1/n** tổng thời gian CPU, trong đó thời gian mỗi lần sử dụng tối đa là **q**

+ Không có tiến trình nào đợi quá (n - 1) x q (**thời gian đáp ứng nhỏ nhất**)

5. Multilevel Queue (Hàng đợi đa cấp):

- Hàng đợi sẵn sàng được chia thành nhiều hàng đợi riêng biệt, ví dụ: **foreground** (tương tác) và **background** (bó).

- Mỗi hàng đợi có giải thuật định thời riêng (ví dụ: **foreground dùng RR**, **background dùng FCFS**). Có thể định thời giữa các hàng đợi bằng độ ưu tiên cố định hoặc phân chia thời gian.

Có 2 chiến lược định thời giữa các hàng đợi:

- Độ **ưu tiên cố định**:

+ Phục vụ tất cả các tiến trình trong hàng đợi ưu tiên cao, sau đó mới đến hàng đợi có độ ưu tiên thấp

+ Có khả năng bị "chết đói" (starvation)

- Định thời cho **phân chia thời gian (time-slice)**:

+ Mỗi hàng đợi sẽ nhận được một khoảng thời gian nào đó để CPU định thời cho từng hàng đợi cho các tiến trình trong đó

Vd : 80% cho foreground với RR, và 20% cho background với FCFS.

P1 P2 P4 P8	80%
P3 P5 P6 P7	20%

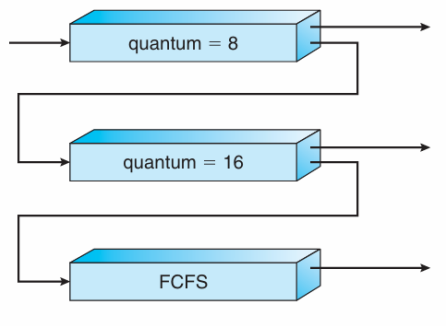
Q1	Q2	Q1	Q2

6. Multilevel Feedback Queue (Hàng đợi phản hồi đa cấp):

- Cho phép một tiến trình có thể di chuyển giữa các hàng đợi khác nhau.

- Nếu một tiến trình sử dụng quá nhiều thời gian CPU, nó có thể bị hạ cấp xuống hàng đợi có độ ưu tiên thấp hơn.

- Ngược lại, nếu một tiến trình đã chờ quá lâu trong hàng đợi ưu tiên thấp, nó sẽ được nâng cấp lên hàng đợi có độ ưu tiên cao hơn **(cơ chế "lão hóa").**



- Bộ định thời đa cấp có phản hồi có thể được định nghĩa bằng những **tham số** sau:

+ Số lượng hàng đợi
+ Giải thuật định thời cho từng hàng đợi
+ Phương thức dùng để quyết định khi nào thì **nâng cấp** một tiến trình.
+ Phương thức dùng để quyết định khi nào thì **hạ cấp** một tiến trình.
+ Phương thức dùng để quyết định là nên **đặt tiến trình vào hàng đợi nào** khi tiến trình này cần được phục vụ.

7. Chính sách, cơ chế và định thời

- Chính sách (policy) và cơ chế (mechanism) có liên hệ mật thiết đến sự lựa chọn giải thuật định thời.

+ Policy: Cần làm điều gì (what)?
+ Mechanism: Làm sao (how) để làm điều đó?

Ví dụ:

- Policy: Tất cả các người dùng cần được công bằng.

⇒ Mechanism: Sử dụng Robin-round.

- Policy: Các tiến trình người dùng trả tiền có độ ưu tiên cao hơn các tiến trình người dùng miễn phí.

⇒ Mechanism: Sử dụng các giải thuật trưng dụng (preemptive).

8. Định thời đa xử lý (Multi-Processor Scheduling)

- Cho phép cân bằng tải nhưng giải thuật định thời CPU sẽ phức tạp hơn

- Các CPU trong hệ thống đa xử lý thường đồng nhất

+ Hệ thống đa xử lý đối xứng (Symmetric Multi-Processing - SMP): Mỗi CPU tự định thời và có thể chia sẻ hàng đợi sẵn sàng. Đây là hệ thống phổ biến hiện nay, đòi hỏi quản lý sự cạnh tranh giữa các CPU khi chọn tiến trình và cập nhật cấu trúc dữ liệu.

+ Hệ thống đa xử lý bất đối xứng (Asymmetric Multi-Processing - AMP): Việc định thời được thực hiện bởi một CPU (master), quản lý cấu trúc dữ liệu dùng chung, giảm nhu cầu quản lý chia sẻ dữ liệu nhưng có thể dẫn đến tắc nghẽn (bottleneck) tại master CPU.

+ Chiến lược "bám CPU" (processor affinity): Một tiến trình sẽ được thực thi trên cùng một CPU trong suốt quá trình thực thi để tránh tổn phí di dời tiến trình sang CPU khác. Có soft-affinity và hard-affinity.

9. Định thời thời gian thực (Real-time Scheduling)

+ Hệ thống thời gian thực cứng (hard real-time systems): Đảm bảo các tác vụ quan trọng phải hoàn thành đúng thời hạn nghiêm ngặt. Chỉ thực hiện với phần mềm chuyên biệt trên nền phần cứng tận hiến và không được hỗ trợ bởi các hệ điều hành đa nhiệm.

+ Hệ thống thời gian thực mềm (soft real-time systems): Ưu tiên các tác vụ thời gian thực nhưng không đảm bảo thời hạn tuyệt đối. Hệ thống phải có định thời trưng dụng, với tiến trình thời gian thực có độ ưu tiên cao nhất.

10. Độ trễ điều phối (dispath latency)

- Là khoảng thời gian cần thiết để bộ định thời chuẩn bị cho sự phục vụ 1 tiến trình:

- Muốn giảm độ trễ điều phối: đưa điểm **trưng dụng** vào lời gọi hệ thống dài (để nhận biết để **thu hồi** CPU giữa chừng) và cho phép trưng dụng nhân (cho phép ngắt ngay cả tiến trình trong nhân (kernel mode), nhờ đó tăng tốc độ phản hồi)

11. Đánh giá giải thuật định thời

- Các phương pháp đánh giá và lựa chọn giải thuật định thời thích hợp cho hệ thống bao gồm:

+ Mô hình xác định (Deterministic modeling): Dựa vào tải công việc được xác định trước và tính toán hiệu năng của mỗi giải thuật (ví dụ: thời gian chờ trung bình). Đơn giản, nhanh nhưng khó có thông tin đầu vào chính xác trong thực tế.

+ Mô hình hàng đợi (Queueing model): Dựa vào sự phân bổ thời gian thực thi CPU và I/O, và thời gian đến. Có thể tính hiệu năng của mỗi giải thuật (ví dụ: hiệu năng CPU, thời gian chờ trung bình).

+ Mô phỏng (Simulation): Sử dụng các chương trình mô phỏng để thực thi các giải thuật với dữ liệu vết (trace data) hoặc dữ liệu ngẫu nhiên, sau đó đo lường hiệu năng. Độ chính xác cao hơn mô hình hàng đợi.

+ Cài đặt thực nghiệm (Implementation): Cài đặt giải thuật vào hệ điều hành thật và đo lường hiệu năng trên hệ thống thực. Phương pháp này cho độ chính xác cao nhất nhưng đòi hỏi chi phí cao và có thể ảnh hưởng đến người dùng.

12. Định thời trong một số hệ điều hành (Ví dụ)

+ Windows: Sử dụng giải thuật định thời dựa trên độ ưu tiên (32 mức) và định mức thời gian (Round-Robin), sử dụng chiến lược trưng dụng. Đơn vị cấp phát CPU

là luồng (thread). Windows sử dụng cơ chế nâng/hạ độ ưu tiên tương tự hàng đợi phản hồi đa cấp.

+ **Linux:** Từ kernel 2.6.23 trở đi, Linux sử dụng giải thuật **Completely Fair Scheduler (CFS)**, dựa trên sự ưu tiên có thay đổi, với mục tiêu nhanh, công bằng và hiệu năng tốt cho các tiến trình tương tác.

+ **Solaris:** Sử dụng định thời dựa trên độ ưu tiên (từ 0 đến 160) và có 6 hàng đợi (Time-sharing, Interactive, Real time, System, Fair share, Fixed priority), mỗi hàng đợi có độ ưu tiên và giải thuật định thời khác nhau. Solaris sử dụng một bảng điều phối để kiểm soát việc định thời, trong đó time quantum tỷ lệ nghịch với độ ưu tiên, và độ ưu tiên thay đổi tùy thuộc vào việc luồng đã sử dụng hết time quantum hay quay lại từ trạng thái chờ.

Chương 5: Đồng bộ hóa tiến trình

I. Giới thiệu (Background)

Cạnh tranh và sự nhất quán dữ liệu:

- Khi các tiến trình thực thi đồng thời và chia sẻ dữ liệu dùng chung, có thể dẫn đến tình trạng không nhất quán của dữ liệu.

- Nhất quán thể hiện bằng cách đúng đắn và chính xác, tùy thuộc vào ngữ cảnh, giao dịch.

- Có hai **lý do** chính để thực hiện đồng thời các tiến trình:

  + Tăng hiệu suất sử dụng tài nguyên hệ thống

  + Giảm thời gian đáp ứng trung bình của hệ thống.

- Để duy trì sự nhất quán của dữ liệu, cần một **cơ chế** đảm bảo sự thực thi có thứ tự của các tiến trình hợp tác.

- Ví dụ về giao dịch cạnh tranh:

Khi hai giao dịch (T1 và T2) cùng cập nhật một tài nguyên chia sẻ (ví dụ: số dư tài khoản P), nếu không có đồng bộ hóa, giá trị cuối cùng của tài nguyên có thể không chính xác, dẫn đến sự không nhất quán.

Tình trạng "tranh đua" (Race condition):

- Là tình trạng mà nhiều tiến trình cùng truy cập và thay đổi dữ liệu chia sẻ, dẫn đến giá trị cuối cùng của dữ liệu phụ thuộc vào quá trình hoàn thành sau cùng. Tình trạng này có thể gây ra sự không nhất quán dữ liệu và cần phải đồng bộ hóa để ngăn chặn.

II. Vấn đề miền tương trục (Critical-section problem - CSP)

Miền tương trục:

- Là một đoạn mã lệnh của các tiến trình chứa các hành động truy cập dữ liệu chia sẻ (ví dụ: thay đổi biến chung, cập nhật cơ sở dữ liệu, ghi tệp, …). Để tránh tình trạng tranh đua, khi một tiến trình đang ở trong miền tương trục, **không có tiến trình nào khác** được phép chạy trong **miền tương trục của nó.**

Cấu trúc, yêu cầu đối với các giải pháp cho miền tương trục (CSP): Một giải pháp bao gồm các **giao thức** phải thỏa mãn ba yêu cầu để tiến trình có thể **hợp tác/cạnh tranh** lẫn nhau:

1. Loại trừ hỗ tương (Mutual exclusion): Đảm bảo chỉ có tối đa một tiến trình được thực thi trong miền tương trục tại một thời điểm.

2. Tiến triển (Progress): Nếu không có tiến trình nào đang thực thi trong miền tương trục và có tiến trình đang chờ vào, thì việc lựa chọn tiến trình tiếp theo không được bị trì hoãn vô hạn.

3. Chờ đợi hữu hạn (Bounded wait): Mỗi tiến trình chỉ phải chờ một khoảng thời gian có hạn để được vào miền tương trục, tránh tình trạng "chết đói" (**starvation**).

```
do{
    entry section
        critial section
    exit section
        remainder section
} while (true);
```

III. Các giải pháp cho vấn đề miền tương trục

1. Giải pháp phần mềm:

- Với 2 tiến trình: Bao gồm các thuật toán

+ Chờ bận (Busy Wait) như GT1 và GT2

Chờ đợi bận 1:

```
int turn = 0;
do {
   while (turn != i) ;
   // critical section
   turn = j;
   // remainder section
} while (true);
```

Nhận xét:

Loại trừ hỗ tương: **Có**

Tiến triển: **Không**

Chờ đợi bận 2:

```
boolean flag[2];
khởi tạo: flag[0] = flag[1] = false
do {
flag[i] := true
while (flag[j]) ;
        critical section
flag[i] = false;
        remainder section
} while (true);
```

Nhận xét:

Loại trừ hỗ tương: **Có**

Tiến triển: **Không** (tình huống cạnh tranh cao)

Tuy nhiên mức độ cạnh tranh cao hơn

+ Giải thuật **Peterson**, là một giải pháp kết hợp biến khóa chia sẻ và biến khóa riêng, **đảm bảo** loại trừ **hỗ tương**, **tiến triển** và **chờ đợi hữu hạn.**

Khởi tạo:

```
int turn;
boolean flag[2];
```

```
do {
   flag[i] := true;
   turn := j;
   while (flag[j] && turn == j) ;
   // critical section
   flag[i] = false;
   // remainder section
} while (true);
```

+ Báo mình vào → flag[i] = true

+ Nhường đối thủ → turn = j

+ Chờ nếu đối thủ cũng muốn vào và mình nhường → while (flag[j] && turn == j);

+ Ra thì gỡ cờ → flag[i] = false;

- Với n ≥ 2 tiến trình: Giải thuật Bakery cho phép mỗi tiến trình nhận một số thứ tự trước khi vào miền tương trục, với số nhỏ nhất được ưu tiên cao nhất.

+ Dữ liệu chia sẻ:

```
boolean choosing[n];
int number[n];
```

+ Khởi tạo: choosing = false và number = 0;

```
do {
   choosing[i] = true; // tôi đang chọn số, đừng kiểm tra tôi lúc này
   number[i] = max(number[0], number[1], ..., number[n-1]) + 1;
// xếp số thứ tự
   choosing[i] = false;

   for (j = 0; j < n; j++) {
      while (choosing[j]) ; // chờ những người đang chọn số
      while ((number[j] != 0) && ((number[j], j) < (number[i], i))) ; //
chờ đến lượt mình theo số và thứ tự ưu tiên
   }
   // critical section
   number[i] = 0; //bỏ số
   // remainder section
} while (true);
```

Chú ý: (number #1, i) < (number #2, j) nếu:

(number #1 < number #2) hoặc (number #1 = number #2) AND (i < j)

2. Giải pháp phần cứng:

a) Vô hiệu hóa ngắt:

- Giải pháp đơn giản nhất là cho phép tiến trình vô hiệu hóa các ngắt khi vào miền tương trục. Điều này ngăn chặn các tiến trình khác thực thi đồng thời

- Chỉ khả thi cho **hệ thống không trưng dụng**, **hệ thống đơn xử lý** và **không hiệu quả** trên hệ thống **đa xử lý.**

b) Các lệnh nguyên tử (Atomic hardware instructions):

- Các lệnh như `**test_and_set**` và `**compare_and_swap**` cho phép đọc và sửa nội dung một cách nguyên tử, được hỗ trợ trong các hệ thống hiện đại với nhiều bộ xử lý để đảm bảo loại trừ hỗ tương.

+ **test_and_set:**

```
boolean test_and_set(boolean *target) {
   boolean rv = *target;
   *target = true;
```

```
        return rv;
    }
```

Dữ liệu chia sẻ: boolean lock = false;

Thực hiện:

```
    do {
        while (test_and_set(lock)) ; //do nothing
            critical section
        lock = false;
            remainder section
    } while (true);
```

Tính chất:

- Loại trừ hỗ tương và tiến triển: **Có**
- Chờ đợi hữu hạn: **Không.** Mỗi khi một tiến trình giải phóng lock (đặt lock = false), tất cả **tiến trình khác đua nhau** kiểm tra lại (test_and_set(&lock))**.** Một tiến trình có thể liên tục gọi test and set() nhưng luôn nhận true vì các tiến trình khác (có thể do tốc độ thực thi nhanh hơn hoặc lịch trình không may mắn) liên tục giành được và giải phóng khóa trước khi nó có cơ hội nhận **false**. Điều này có thể dẫn đến tình trạng "chết đói" (**starvation**).
- Set and test vẫn có cơ chế hỗ trợ test_and_test

+ **compare_and_swap:**

Dùng để swap hai biến

Chỉ thị swap:

```
void swap(boolean &oldValue, boolean expected, boolean newValue) {
   boolean temp = *oldValue;
   if (*oldValue == expected)
      *oldValue = newValue;
   return temp;
}
```

Dữ liệu chia sẻ: boolean lock = false;

Thực hiện:

```
do {
    while (compare_and_swap(&lock, false, true) != false) ;
        // critical section
    lock = false;
        // remainder section
} while (true);
```

Tính chất:

- Loại trừ hỗ tương: **Có**
- Chờ đợi hữu hạn: **Không.** Không có cơ chế nào đảm bảo tiến trình nào **đã chờ lâu hơn** thì được vào trước.

IV. Hiệu báo (Semaphores)

- Semaphore là một công cụ đồng bộ hóa tránh được chờ đợi bận, là một biến kiểu integer được truy cập thông qua hai thao tác nguyên tử: wait(S) (giảm giá trị của S, chặn nếu S <= 0) và signal(S) (tăng giá trị của S, đánh thức một tiến trình đang chờ nếu có).

+ wait(S):

```
while (S ≤ 0);
S--;
```

+ signal(S): S++;

- Các loại semaphore bao gồm

+ **semaphore đếm (counting semaphore)** (giá trị không giới hạn)

+ **semaphore nhị phân (binary semaphore):** hay còn gọi là mutex lock (giá trị 0 hoặc 1). Mutex lock thường được sử dụng để đảm bảo loại trừ hỗ tương khi truy cập miền tương trục.

- Ví dụ:

+ Có 2 tiến trình: P_1 với lệnh S_1 và P_2 với lệnh S_2

+ Yêu cầu: S_2 phải thực thi sau khi S_1 hoàn thành.

+ Cài đặt: Sử dụng semaphore flag, khởi tạo với giá trị 0.

Tiến trình P_1:

…

S1;

signal(flag);

Tiến trình P_2:

...

wait(flag);

S2;

- Tuy nhiên 2 thao tác này **vẫn chưa** tránh được chờ đợi bận (không có ai gọi thì ngủ hoài)

Cài đặt semaphore không chờ đợi bận: Sử dụng các hàm được cung cấp bởi HDH

+ `block()` để tạm thời ngưng tiến trình

+ `wakeup(P)` để khởi động lại tiến trình đang ngưng giúp tránh tình trạng chờ đợi bận liên tục.

- Semaphore được định nghĩa lại thành một struct với value và List *L

- Định nghĩa lại wait(S) và signal(S)

```
void wait(semaphore S) {
   S.value--;
   if (S.value < 0) {
      add this process into S.L;
      block();
   }
} //wait()
```

```
void signal(semaphore S) {
   S.value++;
   if (S.value <= 0) {
      remove a process P from S.L;
      wakeup(P);
   }
} //signal()
```

- Gía trị semaphore có thể âm (trong trường hợp chờ đợi bận thì không). Và số tiến trình đang chờ biểu thị số lượng tiến trình đang chờ đợi

- Yêu cầu: không thể có nhiều hơn một tiến trình thực thi wait() hoặc signal() đồng thời

- Khóa chết (Deadlock): Xảy ra khi hai hay nhiều tiến trình chờ đợi vô hạn một sự kiện mà sự kiện đó chỉ có thể được tạo ra bởi một trong các tiến trình đang chờ đợi khác.

- Chết đói (Starvation): Xảy ra khi một tiến trình bị ngưng không hạn định và không bao giờ được xóa khỏi hàng đợi semaphore.

V. Các bài toán đồng bộ hóa

* **Bài toán Nhà sản xuất – Người tiêu dùng với vùng đệm giới hạn**: Giải quyết bằng cách sử dụng semaphore `mutex` (loại trừ hỗ tương), `empty` (đếm số khe

trống), và `full` (đếm số khe đầy) để kiểm soát việc truy cập và sử dụng vùng đệm chung.

* **Bài toán Bộ đọc – Bộ ghi (Readers–Writers Problem)**: Cho phép nhiều bộ đọc truy cập dữ liệu đồng thời, nhưng chỉ một bộ ghi được phép truy cập tại một thời điểm. Sử dụng semaphore `mutex` (cho `read_count`) và `rw_mutex` (cho các bộ ghi và bộ đọc đầu/cuối cùng).

* **Bài toán Các triết gia ăn tối (Dining-Philosophy)**: Minh họa vấn đề khóa chết khi nhiều tiến trình cạnh tranh tài nguyên hữu hạn (đũa).

VI. Monitor

* Khái niệm: Là một cấu trúc dữ liệu trừu tượng được cung cấp bởi các ngôn ngữ lập trình cấp cao để đơn giản hóa việc đồng bộ hóa.

* Cấu trúc và đặc tính: Một monitor bao gồm các thao tác (hàm), các biến dữ liệu cục bộ và mã khởi tạo. Các biến cục bộ chỉ được truy cập bởi các thủ tục của monitor, và chỉ có tối đa một tiến trình có thể vào monitor tại một thời điểm, đảm bảo loại trừ hỗ tương.

* Biến điều kiện (Condition Variable): Được sử dụng bên trong monitor để cho phép một tiến trình đợi (`x.wait()`) và được đánh thức bởi một tiến trình khác (`x.signal()`).

* Monitor giúp giải quyết các bài toán đồng bộ hóa phức tạp một cách dễ dàng và hiệu quả hơn, ví dụ như bài toán các triết gia ăn tối.

* Các ngôn ngữ có thể cài đặt (Java, C#)

Dựa trên các tài liệu đã cung cấp, dưới đây là mục tiêu của từng chương và những gì mỗi chương giúp người học đạt được:

- **Chương 1: Giới thiệu Hệ Điều Hành** (trong [CT107] Ch1 - Gioi thieu HDH.pdf)
    - **Mục tiêu:** Chương này giúp sinh viên **hiểu rõ Hệ Điều hành là gì và vai trò của Hệ Điều hành trong các hệ thống máy tính và các môi trường điện toán**.
    - **Chương này giúp ta cái gì:** Chương này cung cấp cái nhìn tổng quan về hệ điều hành, bao gồm định nghĩa, mục tiêu, vai trò của nó, các thành

phần cấu tạo nên một hệ thống máy tính, cách tổ chức của một hệ thống máy tính, quá trình khởi động của máy tính, các loại hệ thống máy tính khác nhau như hệ thống đa chương, chia thời gian, đa xử lý, phân tán, thời gian thực, và các môi trường điện toán hiện đại. Việc học chương này là nền tảng để nắm vững các khái niệm cơ bản và bối cảnh hoạt động của hệ điều hành.

- **Chương 2: Cấu Trúc Hệ Điều Hành** (trong [CT107] Ch2 - Cau truc HDH.pdf)
    - **Mục tiêu:** Chương này giới thiệu **các dịch vụ mà hệ điều hành cung cấp và các phương pháp thiết kế các kiến trúc và cài đặt hệ điều hành**.
    - **Chương này giúp ta cái gì:** Chương này trình bày chi tiết về các thành phần cốt lõi của hệ điều hành như quản lý tiến trình, bộ nhớ chính, hệ thống tập tin, nhập/xuất, lưu trữ thứ cấp, kết nối mạng, hệ thống bảo vệ và giao diện người dùng. Nó cũng giải thích các dịch vụ mà hệ điều hành cung cấp cho người dùng và các chương trình, cách thức gọi các dịch vụ này thông qua lời gọi hệ thống (system calls), và các loại chương trình hệ thống. Cuối cùng, chương này đi sâu vào các kiến trúc hệ điều hành khác nhau như nguyên khối, phân tầng, vi nhân và máy ảo, cùng với việc thiết kế và cài đặt hệ điều hành.
- **Chương 3: Tiến Trình (Process)** (trong [CT107] Ch3 - Tien trinh.pdf)
    - **Mục tiêu:** Chương này giới thiệu **các khái niệm về Tiến trình và những thao tác cơ bản trong quản lý Tiến trình như tạo, định thời và kết thúc tiến trình. Các phương thức giao tiếp liên tiến trình cũng sẽ được trình bày**.
    - **Chương này giúp ta cái gì:** Chương này làm rõ khái niệm tiến trình là một chương trình đang thực thi và các thành phần của nó. Nó mô tả các trạng thái của tiến trình (new, running, waiting, ready, terminated) và sự chuyển đổi giữa chúng, vai trò của Khối điều khiển Tiến trình (PCB) và cơ chế chuyển ngữ cảnh. Chương cũng giải thích cách định thời tiến trình bằng các hàng đợi (job, ready, device queues) và các loại bộ định thời (dài kỳ, ngắn kỳ, trung kỳ). Các thao tác cơ bản trên tiến trình như tạo (fork, CreateProcess) và kết thúc (exit, wait, abort, zombie, orphan) cũng được trình bày. Đặc biệt, chương này đi sâu vào giao tiếp liên tiến trình (IPC) thông qua mô hình bộ nhớ chia sẻ (shared memory) và truyền thông điệp (message passing) với các phương thức trực tiếp, gián tiếp, đồng bộ hóa và tạo vùng đệm. Cuối cùng, chương đề cập đến giao tiếp trong hệ thống Client-Server sử dụng Socket, Remote Procedure Calls (RPCs) và Pipe.

- **Chương 4: Định Thời CPU** (trong [CT107] Ch4 - Dinh thoi bieu CPU.pdf và ct1178-chuong4.pdf)
    - **Mục tiêu:** Chương này giới thiệu **về tác vụ định thời cho CPU (CPU scheduling) trong các hệ điều hành đa chương, bao gồm: các tiêu chí cho việc định thời CPU, các giải thuật định thời CPU, các tiêu chí để lựa chọn 1 giải thuật định thời cho 1 hệ thống**.
    - **Chương này giúp ta cái gì:** Chương này khám phá chi tiết về cách CPU được phân bổ cho các tiến trình để tối ưu hiệu suất. Nó giải thích chu kỳ CPU-I/O, các quyết định định thời, định thời trưng dụng và không trưng dụng, vai trò của bộ điều phối. Các tiêu chí quan trọng để đánh giá giải thuật định thời được làm rõ: hiệu suất sử dụng CPU, thời gian đáp ứng, thời gian chờ đợi, thời gian xoay vòng và thông lượng. Chương cũng trình bày các giải thuật định thời phổ biến như First-Come, First-Served (FCFS), Shortest-Job-First (SJF), Priority (có ưu tiên), Round-Robin (RR), Multilevel Queue và Multilevel Feedback Queue. Ngoài ra, chương còn đề cập đến định thời đa xử lý và định thời thời gian thực, các phương pháp đánh giá giải thuật và ví dụ thực tế trong Windows, Linux, Solaris.
- **Chương 5: Đồng Bộ Hóa Tiến Trình** (trong [CT107] Ch5 - Dong bo hoa tien trinh.pdf)
    - **Mục tiêu:** Chương này giới thiệu **vấn đề miền tương trục và các giải pháp để giải quyết vấn đề miền tương trục, nhằm đảm bảo sự nhất quán của dữ liệu được chia sẻ giữa các tiến trình cạnh tranh trong miền tương trục**.
    - **Chương này giúp ta cái gì:** Chương này giải thích lý do cần đồng bộ hóa là để đảm bảo tính nhất quán của dữ liệu chia sẻ giữa các tiến trình thực thi đồng thời, đặc biệt trong các tình huống tranh đua (race condition). Nó định nghĩa vấn đề miền tương trục (Critical-Section Problem - CSP) và ba yêu cầu cần thiết cho một giải pháp CSP: loại trừ hỗ tương, tiến triển và chờ đợi hữu hạn. Chương trình bày các giải pháp phần mềm (GT1, GT2, giải thuật Peterson, giải thuật Bakery) và giải pháp phần cứng (vô hiệu hóa ngắt, lệnh test_and_set, compare_and_swap). Các công cụ đồng bộ hóa cao cấp hơn như Semaphore (binary, counting semaphores, cài đặt không chờ bận) và vấn đề khóa chết (deadlock), chết đói (starvation) cũng được thảo luận. Cuối cùng, chương áp dụng các khái niệm này để giải quyết các bài toán đồng bộ hóa kinh điển như Nhà sản xuất – Người tiêu dùng, Bộ đọc – Bộ ghi và Các triết gia ăn tối. Các Monitor như một cấu trúc đồng bộ hóa tiện lợi cũng được giới thiệu.
- **Chương 6: Deadlock (Khóa Chết)** (trong [CT107] Ch6 - Khoa chet.pdf)

- **Mục tiêu:** Chương này mô tả **tình trạng deadlock của hệ thống – một trạng thái mà các tiến trình không thể tiến triển để hoàn thành các tác vụ của chúng. Trình bày các phương pháp để ngăn chặn hoặc tránh deadlock; và các biện pháp để phát hiện và phục hồi hệ thống một khi deadlock xảy ra**.
- **Chương này giúp ta cái gì:** Chương này định nghĩa deadlock là gì và các ví dụ minh họa, cùng với bốn điều kiện cần thiết để deadlock xảy ra: loại trừ hỗ tương, giữ và chờ, không trưng dụng tài nguyên, và chờ đợi vòng tròn. Nó giới thiệu đồ thị cấp phát tài nguyên (Resource Allocation Graph - RAG) như một công cụ để mô hình hóa hệ thống và phát hiện chu trình. Chương đi sâu vào các cách tiếp cận vấn đề deadlock: ngăn chặn (phủ định từng điều kiện), tránh (Banker's Algorithm, trạng thái an toàn, đồ thị cấp phát tài nguyên), và phát hiện cùng phục hồi (ngưng tiến trình, thu hồi tài nguyên).

- **Chương 7: Quản lý bộ nhớ** (trong [CT107] Ch7 - Quan ly bo nho.pdf)
  - **Mục tiêu:** Chương này mô tả **chi tiết các phương pháp tổ chức bộ nhớ. Giải thích các kỹ thuật quản lý bộ nhớ bao gồm phân trang và phân đoạn. Một số ví dụ thực tế về quản lý bộ nhớ: quản lý phân đoạn trong bộ xử lý Intel Pentium và quản lý địa chỉ bộ nhớ trong HĐH Linux**.
  - **Chương này giúp ta cái gì:** Chương này cung cấp tổng quan về bộ nhớ, giải thích sự khác biệt giữa thanh ghi, bộ nhớ chính và cache. Nó trình bày cách bảo vệ không gian nhớ bằng thanh ghi nền và thanh ghi giới hạn, cơ chế gắn kết địa chỉ (biên dịch, nạp, thực thi), sự khác biệt giữa địa chỉ vật lý và luận lý, và vai trò của Bộ Quản Lý Bộ Nhớ (MMU). Các kỹ thuật tối ưu hóa bộ nhớ như nạp động (dynamic loading) và liên kết động (dynamic linking) cũng được đề cập. Chương đi sâu vào kỹ thuật hoán vị (swapping), cấp phát bộ nhớ kề nhau (contiguous allocation) với các chiến lược cấp phát (first-fit, best-fit, worst-fit) và vấn đề phân mảnh (fragmentation). Đặc biệt, chương giới thiệu chi tiết kỹ thuật phân trang (paging), từ cơ chế hoạt động, định địa chỉ, bảng trang, TLB, thời gian truy xuất hiệu dụng (EAT), bảo vệ bộ nhớ và chia sẻ trang, đến các cấu trúc bảng trang phức tạp hơn như phân cấp, băm và đảo. Cuối cùng, chương giải thích kỹ thuật phân đoạn (segmentation) và sự kết hợp giữa phân trang và phân đoạn, cùng với các ví dụ thực tế trong bộ xử lý Intel Pentium và Linux.