**Ôn tập chương 1:**

**Câu 1 : Định nghĩa hệ điều hành?**

- Hệ điều hành là chương trình trung gian giữa phần cứng máy tính và người sử dụng, có chức năng điều khiển và phối hợp việc sử dụng phần cứng và cung cấp các dịch vụ cơ bản cho các ứng dụng.

- Trung gian giữa phần cứng máy tính và người sử dụng

- Mục tiêu:

+ Đối với Hardware: quản lý và phân phát tài nguyên cho hiệu quả nhất => Multitasking

+ Đối với người dùng: cung cấp giao diện thân thiện, cung cấp dịch vụ cho người sử dụng => Transparen (che phần cứng, người dùng chỉ hưởng dịch vụ)

- Máy tính tron-vo-nin-man: phải có CPU và MEM

**Câu 2 : Cấu trúc hệ thống máy tính gồm những phần nào?**

- Chia thành 4 tầng: hardware, OS, Applications, users

- Phần cứng (hardware): Bao gồm các tài nguyên phần cứng của máy tính như CPU, bộ nhớ, các thiết bị I/O. Ví dụ: Chuột, bàn phím, màn hình, …

- Hệ điều hành (operating system): Phân phối tài nguyên, điều khiển và phối hợp các hoạt động của các chương trình trong hệ thống. Ví dụ một số loại HĐH: MS – DOS, Linux, Window => HĐH xuất phát từ multi programing

- Chương trình ứng dụng (application programs): Sử dụng hệ thống tài nguyên để giải quyết một bài toán tính toán nào đó của người sử dụng. Ví dụ: game

- Users (people, machines, other computers)

**Câu 3 : Hệ điều hành có những chức năng gì?**

- Phân chia thời gian xử lý và định thời CPU.

- Phối hợp và đồng bộ hoạt động giữa các processes (coordination & synchronization).

- Quản lý tài nguyên hệ thống (thiết bị I/O, bộ nhớ, file chứa dữ liệu,…).

- Thực hiện và kiểm soát access control, protection.

- Duy trì sự nhất quán (integrity) của hệ thống, kiểm soát lỗi và phục hồi hệ thống khi có lỗi (error recovery).

- Cung cấp giao diện làm việc cho users.

- Multitasking: nhiều program tương tác với người dùng

**Câu bổ sung:**

- Chỉ viết được HĐH cho UMA

- Uniprocessor: 1 CPU + MEM => UMA (Unified Memory Acess)

+ MultiThread

+ MultiCore

- Multiprocessor: nhiều CPU, nhiều MEM => non-UMA: bộ nhớ có thể chia sẻ cho CPU khác => mỗi CPU phải có UMA, SMP, 1queue/CPU => mới có thể lập lịch được cho CPU

**Câu 4 : Dưới góc độ hình thức xử lý, hệ điều hành chia thành những loại nào? Trong mỗi loại có những yêu cầu gì với hệ điều hành?**

- Hệ thống xử lý theo lô

+ Hệ thống đơn chương (uniprograming OS):

* Tác vụ được thi hành tuần tự.
* Yêu cầu:
  + Bộ giám sát thường trực
  + CPU và các thao tác nhập xuất:
    - Xử lý offline
    - Đồng bộ hóa các thao tác bên ngoài – Spooling (Simultaneous Peripheral Operation On Line)

+ Hệ thống đa chương (multiprogramming OS):

* Nhiều công việc được nạp đồng thời vào bộ nhớ chính, tận dụng được thời gian rảnh của các tiến trình đang trong giai đoạn chờ thực thi.
* Yêu cầu:
  + Định thời công việc (job scheduling): chọn job trong job pool trên đĩa và nạp nó vào bộ nhớ để thực thi.
  + Quản lý bộ nhớ (memory management).
  + Định thời CPU (CPU scheduling).
  + Cấp phát tài nguyên (đĩa, máy in,…).
  + Bảo vệ.

+ Hệ thống chia sẻ thời gian:

* Là hệ thống đa nhiệm, lập lịch cho các tiến trình thực thi trên CPU.
* Yêu cầu:
  + Định thời công việc (job scheduling).
  + Quản lý bộ nhớ (memory management).
    - Virtual memory
  + Quản lý các quá trình (process management)
    - Định thời CPU
    - Đồng bộ các quá trình (synchronization)
    - Giao tiếp giữa các quá trình (process communication)
    - Tránh deadlock
  + Quản lý hệ thống file, hệ thống lưu trữ.
  + Cấp phát hợp lý các tài nguyên.
  + Bảo vệ (protection).

+ Hệ thống song song (Parallel System)

* Nhiều CPU, chia sẻ computer bus, clock
* Ưu điểm:
  + Năng suất: càng nhiều CPU thì càng xử lý công việc nhanh.
  + Multiprocessor system ít tốn kém hơn multiple single-processor system.
  + Độ tin cậy: khi một processor hỏng thì công việc của nó được chia sẻ giữa các processor còn lại.
* Phân loại: Đa xử lý đối xứng và đa xử lý bất đối xứng.

+ Hệ thống phân tán: SAN (lưu trữ), Database

* Mỗi processor có bộ nhớ riêng, giao tiếp với nhau qua các kênh nối như mạng, bus tốc độ cao nhưng người dùng chỉ thấy một hệ thống đơn nhất.
* Ưu điểm:
  + Chia sẻ tài nguyên (resource sharing)
  + Chia sẻ sức mạnh tính toán (computational sharing)
  + Độ tin cậy cao (high reliability)
  + Độ sẵn sàng cao (high availability): các dịch vụ của hệ thống được cung cấp liên tục cho dù một thành phần hardware trở nên hỏng.
* Các mô hình hệ thống song song: client-sever và peer-to-peer

+ Hệ thống xử lý thời gian thực:

* Sử dụng trong các thiết bị chuyên dụng như điều khiển các thử nghiệm khoa học, điều khiển trong y khoa, dây chuyền công nghiệp, thiết bị gia dụng, quân sự.
* Ràng buộc về thời gian: hard và soft real-time.
  + Hard real-time.
    - Hạn chế (hoặc không có) bộ nhớ phụ, tất cả dữ liệu nằm trong bộ nhớ chính (RAM hoặc ROM).
    - Yêu cầu về thời gian đáp ứng/xử lý rất nghiêm ngặt, thường sử dụng trong điều khiển công nghiệp, robotics,…
  + Soft real-time: Thường được dùng trong lĩnh vực multimedia, virtual reality với yêu cầu mềm dẻo hơn về thời gian đáp ứng.

**Câu 5 :** **Dưới góc độ loại máy tính, hệ điều hành chia thành những loại nào?**

- Hệ điều hành dành cho máy MainFrame.

- Hệ điều hành dành cho máy Server.

- Hệ điều hành dành cho máy nhiều CPU.

- Hệ điều hành dành cho máy tính cá nhân (PC).

- Hệ điều hành dành cho máy PDA (Embedded OS - hệ điều hành nhúng).

- Hệ điều hành dành cho máy chuyên biệt.

- Hệ điều hành dành cho thẻ chíp (SmartCard).

**Câu 6 :** **Nêu lịch sử phát triển của HĐH:**

- Thế hệ 1 (1945 - 1955)

* Thiết kế, xây dựng, lập trình, thao tác: do 1 nhóm người.
* Lưu trên phiếu đục lỗ.

- Thế hệ 2 (1955 - 1965)

* Xuất hiện sự phân công công việc
* Hệ thống sử lý theo lô ra đời, lưu trên băng từ.
* Hoạt động dưới sự điều khiển đặc biệt của 1 chương trình .
* Chưa xuất hiện hệ điều hành.

- Thế hệ 3 (1965 - 1980)

* Ra đời hệ điều hành, khái niệm đa chương.
* HĐH chia sẻ thời gian như CTSS của MIT.
* MULTICS, UNIX.

- Thế hệ 4 (1980)

* Ra đời máy tính cá nhân, IBM PC.
* HĐH MS-DOS, MacOS (Apple Macintosh), HĐH mạng,…

**Câu 7 : Những yêu cầu của hệ thống chia sẻ thời gian ?**

- Yêu cầu:

* Định thời công việc (job scheduling).
* Quản lý bộ nhớ (memory management).
  + Virtual memory.
* Quản lý các quá trình (process management).
* Định thời CPU.
* Đồng bộ các quá trình (synchronization).
* Giao tiếp giữa các quá trình (process communication).
  + Tránh deadlock.
* Quản lý hệ thống file, hệ thống lưu trữ
* Cấp phát hợp lý các tài nguyên
* Bảo vệ (protection).

**Câu 8 :** Đặc điểm của hệ thống đa chương?

- Nhiều công việc được nạp đồng thời vào bộ nhớ chính

- Khi một tiến trình thực hiện I/O, một tiến trình khác có thể được thực thi

- Tận dụng được thời gian rảnh, tăng hiệu suất sử dụng CPU (CPU utilization)

**Ôn tập chương 2 :**

**Câu 1 :** **Hệ điều hành bao gồm những thành phần nào? Cụ thể từng thành phần?**

- Quản lý tiến trình

- Quản lý bộ nhớ chính

- Quản lý file

- Quản lý hệ thống I/O

- Quản lý hệ thống lưu trữ thứ cấp

- Hệ thống bảo vệ

- Hệ thống thông dịch lệnh

- Dịch vụ local là tất cả phần cứng máy tính cung cấp

- Dịch vụ global là dịch vụ thông qua mạng

- Dịch vụ HĐH cung cấp:

+ Cấp phát tài nguyên (Resource Allocation)

+ Kế toán (Accounting)

+ Protection và Security

**Câu 2 : Các cơ chế trao đổi thông tin giữa các tiến trình?**

- Vùng nhờ chia sẻ

- Chuyển thông điệp

- Dùng tín hiệu \*

- Pipe \*

\* Hai cách này không có trong slide

**Câu 3 :** Cấu trúc hệ thống gồm những loại nào? Cho ví dụ từng loại (theo sách tham khảo)

- Cấu trúc đơn giản: MS-DOS

- Cấu trúc phức tạp hơn: UNIX

- Cấu trúc phân tầng: THE

- Cấu trúc vi nhân: QNX, WindowNT, POSIX, …

**Câu 4 :** **Chương trình hệ thống gồm những phần nào?**

- Quản lý hệ thống file: như create, delete, rename, list

- Thông tin trạng thái: như date, time, dung lượng bộ nhớ trống

- Soạn thảo file: như file editor

- Hỗ trợ ngôn ngữ lập trình: như compiler, assembler, interpreter

- Nạp, thực thi, giúp tìm lỗi chương trình: như loader, debugger

- Giao tiếp: như email, talk, web browser

**Câu 5 :** **Lời gọi hệ thống là gì và dùng để làm gì?**

- Lời gọi hệ thống là việc một chương trình máy tính yêu cầu một dịch vụ từ nhân của hệ điều hành mà nó được thực thi. => bản chất làm các hàm => tập trung thành thư viện, gọi là library => trên bộ nhớ phụ => API: 3 loại

- Tác dụng:

* Dùng để giao tiếp giữa User program và hệ điều hành (kernel)
* Cung cấp giao diện giữa User program và hệ điều hành (kernel)

- Truyền tham số cho dịch vụ: 3 phương pháp truyền

+ Qua thanh ghi

+ Qua vùng nhớ

+ Qua stack

- System application lưu ở bộ nhớ phụ: tại thư mục bin (Linux/Unix) và System32 của Windows

**Câu 6 :** **Hệ điều hành cung cấp những dịch vụ nào?**

- Thực thi chương trình

- Thực hiện các thao tác I/O theo yêu cầu của chương trình

- Các thao tác trên hệ thống file

- Trao đổi thông tin giữa các tiến trình qua hai cách:

* Chia sẻ bộ nhớ (Shared memory)
* Chuyển thông điệp (Message passing)

- Phát hiện lỗi

- Ngoài ra còn các dịch vụ giúp tăng hiệu suất của hệ thống:

+ Cấp phát tài nguyên (resource allocation):

* + CPU, bộ nhớ chính, ổ đĩa,…
  + OS có các routine tương ứng.

+ Kế toán (accounting): Nhằm lưu vết user để tính phí hoặc đơn giản để thống kê.

+ Bảo vệ (protection)

* Hai tiến trình khác nhau không được ảnh hưởng nhau
* Kiểm soát được các truy xuất tài nguyên của hệ thống

+ An ninh (security): Chỉ các user được phép sử dụng hệ thống mới truy cập được tài nguyên của hệ thống (vd: thông qua username và password)

**Câu 7 :** **Các khái niệm liên quan đến máy ảo?**

- Máy ảo là phần mềm tạo ra môi trường giữa hệ nên máy tính và người dùng, người dùng có thể thực thi phần mềm trên máy ảo

- Ví dụ: Virtual Box, Parallels

**Ôn tập chương 3 :**

**Câu 1 : Một tiến trình chứa những thành phần gì ?**

- Một chương trình (program) sẽ trở thành tiến trình (process) chỉ khi nó được nạp lên bộ nhớ (memory) của máy tính (hay nói cách khác – được thực thi).

- Một tiến trình bao gồm :

Trong bộ nhớ (memory) :

* **Text Section (Program code) :** Chứa những đoạn mã chương trình đã được biên dịch bởi compiler.
* **Data Section (khu vực dữ liệu) :** Chứa các biến toàn cục (global variables) và các biến tĩnh (static variables) được khởi tạo trước khi hàm main được gọi.
* **Heap :** Dùng để lưu trữ các bộ nhớ được cấp phát động (như việc gọi new, delete, malloc, calloc, free,…).
* **Stack :** Dùng để lưu trữ các biến cục bộ (local variables). VD như lúc bạn khai báo trong C : int i = 0. Thì biến i sẽ được lưu trong bộ nhớ Stack của Process.

A diagram of a diagram

Description automatically generated

Một tiến trình trong bộ nhớ.

Ngoài ra, trong process còn có :

* Program Counter (PC).
* Process status word (PSW).
* Stack pointer (SP).
* Memory management registers.

**Câu 2 : Tiến trình có những trạng thái nào ? Cách tiến trình chuyển trạng thái ?**

Tiến trình có các trạng thái sau :

* **new :** Tiến trình vừa được tạo.
* **ready :**Tiến trình đã có đủ tài nguyên, đang chờ được cấp CPU để chạy.
* **running :**Các lệnh của tiến trình đang được thực thi.
* **waiting (hay blocked) :** Tiến trình đợi I/O hoàn tất.
* **terminated :** Tiến trình đã kết thúc (đã thực thi xong).

Cách tiến trình chuyển trạng thái :

Đầu tiên, khi vừa khởi tạo, tiến trình sẽ ở trạng thái là **new.**

Thông qua bộ định thời dài hạn **Long-term scheduling** (hay còn gọi là bộ định thời công việc – **Job Scheduler)**, tiến trình của chúng ta từ **new** sẽ được sắp xếp vị trí để “chui” vào trong hàng đợi **ready.**

Ở một số hệ điều hành có thêm bộ định thời **Medium-term Scheduler.** Thông qua bộ định thời này, các tiến trình sẽ được swap-out (chuyển tiến trình từ bộ nhớ chính sang bộ nhớ phụ) và swap-in (chuyển tiến trình từ bộ nhớ phụ vào bộ nhớ). VD : Paging trong HĐH Windows, hoặc Swap trong Linux.

Thông qua bộ định thời **Short-term Scheduling** (hay còn được gọi là **Dispatcher**), từ trạng thái **ready** tiến trình sẽ được sắp xếp để chuyển qua trạng thái **running**(là trạng thái chạy – hay trạng thái sử dụng CPU – của tiến trình).

Trong khi đang ở trạng thái **running**, có 3 trạng thái tiếp theo mà tiến trình có thể đạt được tiếp theo :

* **waiting :** Khi tiến trình đang chờ I/O (VD : Khi gọi hàm print(), scanf() trong C).
* **ready :**Khi tiến trình bị interrupt (bị ngắt, không cho chạy nữa) bởi **Short-term Scheduler.** Các lý do ngắt có thể là : Ngắt thời gian (Clock Interrupt), Ngắt ngoại vi (I/O Interrupt), Lời gọi hệ thống (Operating System Call), Signal.
* **terminated :** Khi ứng dụng thực thi xong : Khi gặp lệnh exit, khi thực thi lệnh cuối.

Trong khi ở trạng thái **waiting,** tiến trình sẽ chuyển sang trạng thái **ready**(vào hàng đợi **ready**) sau khi đã thực thi xong I/O.

A diagram of a running process

Description automatically generated

Chuyển các trạng thái của tiến trình.

A diagram of a schedule

Description automatically generated

Tiến trình chuyển trạng thái bằng các bộ định thời khác nhau.

(Nói thêm : Có thể có **>= 1** tiến trình ở trạng thái ready và waiting, tuy nhiên chỉ có **duy nhất 1** tiến trình ở trạng thái **running** tại một **thời điểm nhất định** mà ta đang xét).

**Câu 3 : Tại sao phải cộng tác giữa các tiến trình ?**

Trong tiến trình thực thi, các tiến trình có thể cộng tác (cooperate) để hoàn thành các công việc.

Các tiến trình cộng tác với nhau để :

* Chia sẻ dữ liệu (information sharing).
* Tăng tốc độ tính toán (computational speedup).
  + Các mạng lưới máy tính sẽ hợp với nhau để tạo thành các cluster.
  + Nếu hệ thống có nhiều CPU, chia công việc tính toán thành nhiều công việc tính toán nhỏ chạy song song.
* Thực hiện một công việc chung.
  + Xây dựng một phần mềm phức tạp bằng cách chia thành các module/process hợp tác nhau.

**Câu 4 : PCB là gì ? Chứa những thông tin gì ? Dùng để làm gì ?**

Mỗi tiến trình trong hệ thống đều được cấp phát một **Process Control Block** (PCB). Là một trong các cấu trúc dữ liệu quan trọng nhất của hệ điều hành. => cấu trúc danh sách liên kết

PCB chứa các thông tin liên quan đến process như :

* Trạng thái tiến trình (Process State) : new, ready, running,…
* Bộ đếm chương trình (Program Counter) : Chỉ đến địa chỉ của lệnh tiếp theo sẽ được thực thi cho tiến trình này.
* Các thanh ghi CPU (CPU Registers) : Phụ thuộc vào kiến trúc máy tính. Có thể kể đến vài loại như accumulators, index registers, stack pointers, general-purpose registers, condition-code information.
* Thông tin lập thời biểu CPU (CPU Scheduling Information) : Độ ưu tiên, con trỏ đến các hàng đợi, và các tham số của việc lập thời biểu.
* Thông tin quản lý bộ nhớ (Memory-Management Information) : Chứa page tables, segment tables, memory limits (giới hạn bộ nhớ).
* Thông tin trạng thái I/O (I/O status information) : Chứa danh sách các thiết bị I/O đã được cấp phát cho tiến trình, danh sách các file tiến trình đang mở,…
* Các thông tin quan trọng khác như : Lượng CPU, thời gian sử dụng,PID,…

A close-up of a blue and white rectangular object

Description automatically generated

Hình ảnh minh hoạ một Process Control Block.

Vậy, các PCB đơn giản là các kho dữ liệu, trong đó chứa các dữ liệu khác nhau của nhiều process khác nhau.

**Câu 5 : Tiểu trình là gì ?**

Tiểu trình : Là một đơn vị cơ bản sử dụng CPU, gồm :

* Thread ID.
* PC (Program Counter).
* Registers.
* Stack.
* **Chia sẻ chung code, data, resources (file).**
* **Dùng riêng thanh ghi, stack vì bản chất là luồng-PU, mà PU thì phải cần thanh ghi riêng, stack riêng**

- Các mô hình đa tiểu trình:

+ Nhiều – Một

Ưu: ít tốn tài nguyên

Nhược: lâu

+ Một – Một => bây giờ phần cứng phát triển, dùng cái này được

Ưu: nhanh, chạy đồng thời được

Nhược: tốn tài nguyên

+ Nhiều – Nhiều

A diagram of a software processing process

Description automatically generated with medium confidence

So sánh giữa đơn tiến trình và đa tiến trình.

**Câu 5: Các tác vụ đối với tiến trình**

- Tạo và Kết thúc

- Theo cấu trúc cây, là số nguyên, càng xa root thì càng lớn. Khi tạo mới, phải chọn 1 node để móc vào, gọi là parent node => vậy phải chọn 1 parent process để khi tạo mới. Có 2 giai đoạn: tạo process con từ process cho và đưa bộ nhớ vào không gian sử dụng cho process mới.

- Lệnh fork(): nếu không có template => duplicate của process cha, ngược lại thì nó theo template

+ Tài sản: Thừa hưởng và di truyền

+ Hoạt động: sau lệnh fork() thì thực hiện đồng thời, khi kết thúc thì con kết thúc trước cha => do theo cấu trúc cây

+ Kết thúc: 3 cách => nó tự kết thúc, bị cha kết thúc và bị hđh kết thúc

**Câu** **6: Sự cộng tác giữa các tiến trình**

- Một cộng tác luôn có sự Comunication (naming và send-receive) và Synchronize

- 3 trường hợp giữa 2 tiến trình:

+ chung bộ nhớ, chung OS => Protocal IPC: Shared MEM và Message Passing (trực tiếp và gián tiếp)

+ khác bộ nhớ, chung OS => Protocal RPC:

+ khác bộ nhớ, khác OS => lập trình Socket và Java Tool

**Ôn tập chương 4.**

**Câu 1 : Tại sao phải định thời ? Nêu các bộ định thời và mô tả về chúng ?**

Mục tiêu của việc lập trình đa luồng là hướng đến việc luôn luôn phải có tiến trình sử dụng CPU, hay nói cách khác, là tối đa hoá việc sử dụng CPU. Ngoài ra, mục tiêu của các hệ thống time sharing (chia sẻ thời gian – tức các hệ thống HĐH hiện nay) là việc mang đến cho người dùng cảm giác chiếc máy tính của mình có thể làm được nhiều công việc cùng một lúc. Việc đó chỉ có thể đạt được thông qua việc chuyển quyền sử dụng CPU thật nhanh qua lại giữa các tiến trình.

Và để đạt được các mục tiêu nêu trên, **trình định thời** (**Scheduler**) sẽ lựa chọn trong các tiến trình hiện có để thực thi trên CPU. Nguyên nhân là do, trong một thời điểm nhất định, chỉ duy nhất có một tiến trình được quyền ở trạng thái running mà thôi.

Có 3 bộ định thời :

* Short-Term Scheduling (hay còn gọi là Dispatcher) : Dùng để định thời cho CPU.
  + Xác định process nào trong ready queue sẽ được chiếm CPU để thực thi kế tiếp.
  + Bộ định thời Short-Term sẽ được gọi mỗi khi có một trong các sự kiện/interrupt sau xảy ra :
    - Ngắt thời gian (clock interrupt).
    - Ngắt ngoại vi (I/O interrupt).
    - Lời gọi hệ thống (Operating System Call).
    - Signal.
* Medium-Term Scheduling : Dùng để định thời Swaping.
  + Process nào được đưa vào (swap-in), đưa ra khỏi (swap-out) bộ nhớ chính.
  + Được thực hiện bởi phần quản lý bộ nhớ và được thảo luận ở phần quản lý bộ nhớ.
* Long-Term Scheduling (hay còn gọi là Job Scheduler) :
  + Xác định chương trình nào được chấp nhận nạp vào hệ thống để thực thi.
  + Điều khiển mức độ multiprogramming của hệ thống.
  + Long-Term Scheduling thường cố gắng duy trì xen lẫn CPU-Bound và I/O Bound Process.

**Câu 2 : Các tiêu chuẩn định thời CPU ?**

Các thuật toán định thời CPU khác nhau có các tính chất khác nhau. Vì vậy, rất nhiều tiêu chí đã được đề ra để so sánh các thuật toán định thời CPU. Các tiêu chuẩn được liệt kê như sau :

* **Mức độ sử dụng CPU (CPU Utilization) :** Chúng ta muốn giữ làm sao cho CPU càng bận càng tốt. Theo lý thuyết thì CPU utilization có thể đạt từ 0 đến 100 phần trăm. Trong hệ thống thực tế, thông thường CPU Utilization sẽ đạt 40 phần trăm (cho hệ thống lúc rỗi), và có thể đạt mức lên đến 90 phần trăm (cho lúc hệ thống load nặng).
* **Băng thông (Throughput) :** Một trong những thước đo cho hiệu quả của quá trình làm việc đó chính là thông lượng. Thông lượng được tính bằng số lượng tiến trình đã được hoàn thành trong một đơn vị thời gian.
* **Thời gian hoàn thành (Turnaround time) :** Từ góc nhìn của một tiến trình nhất định, một yếu tố quan trọng cần xem xét đó chính là khoảng thời gian cần thiết để thực thi tiến trình đó. Khoảng thời gian từ lúc tiến trình được ghi nhận đến khi hoàn thành chính là **Turnaround Time.** Vì vậy, **Turnaround Time** là tổng của tất cả các khoảng thời gian : Tiến trình trong hàng đợi, thực thi trên CPU và thực thi lệnh I/O.
* **Thời gian đợi (Waiting time) :** Tổng thời gian tiến trình đã ở trong hàng đợi ready queue.
* **Thời gian đáp ứng (Response Time) :** Thời gian từ lúc tiến trình xuất hiện cho đến khi thực hiện tiến trình đó lần đầu tiên.

**Câu 3 : Có bao nhiêu giải thuật định thời ? Kể tên.**

* First-Come, First-Served (FCFS).
* Shortest-Job-First Scheduling (SJF).
* Preemptive SJF (hay Shortest-Remaining-Time First – SRTF).
* Priority Scheduling.
* Round-Robin (RR).
* Highest Response Ratio Next (HRRN).
* Multilevel Queue.
* Multilevel Feedback Queue.

**Câu 4 : Trong các hàng đợi định thời, định thời nào có thể preemptive ?**

* SJF có thể Preemptive -> SRTF.
* Priority Scheduling.
* RR.
* Mutilevel Queue (Tuỳ cách thiết lập).
* Multilevel Feedback Queue.

**Câu 5 : Trong các giải thuật định thời, giải thuật nào không xảy ra starvation ?**

Starvation xảy ra khi một tiến trình đã ở trong hàng đợi quá lâu, nhưng không được cung cấp CPU để sử dụng.

Các giải thuật sẽ không xảy ra starvation :

* FCFS.
* RR.
* HRRN.
* Mutilevel Queue (Tuỳ cách thiết lập).
* Multilevel Feedback Queue.

Phù, vậy là tạm ôn xong 4 chương rồi đấy. Dài quá các bạn nhỉ ?

Tụi mình vẫn sẽ tiếp tục cập nhập các câu hỏi đến các bạn trong thời gian sớm nhất nhé. Mình sẽ gửi comment mỗi khi có update những câu hỏi mới đến các bạn. Nhưng tạm thời hôm nay thì nhiêu đây thôi nhe ^^ !

Chúc các bạn ôn tập tốt và thi thật tốt !

**\*\*\*UPDATE\*\*\***

**Chương 3 :**

**Câu 6 : Trình tự thực thi của tiến trình cha và tiến trình con ?**

Có 2 khả năng có thể xảy ra :

* Tiến trình cha tiếp tục thực thi song song với các tiến trình con của nó.
* Tiến trình cha đợi đến khi một vài hoặc tất cả các tiến trình con của nó đã hoàn tất.

Trong khi giải bài tập thì chúng ta sẽ giải theo trường hợp 2 (Tiến trình cha đợi đến khi tiến trình con vừa gọi kết thúc mới thực thi tiếp).

**Câu 7 : Các tác vụ đối với tiến trình (Operations on Process) ?**

Có 2 tác vụ chính : Tạo tiến trình mới và kết thúc tiến trình.

* **Tạo tiến trình mới :** 
  + Một tiến trình có thể tạo nhiều tiến trình mới thông qua một lời gọi hệ thống create-process (VD : Hàm fork() trong Unix).
  + Tiến trình được tạo là tiến trình con của tiến trình tạo (tiến trình cha).
    - Quan hệ cha-con định nghĩa một cây tiến trình.

A diagram of a computer network

Description automatically generated

Hình ảnh minh hoạ của một cây tiến trình.

* Tiến trình con nhận tài nguyên từ HĐH hoặc từ tiến trình cha.
* Chia sẻ tài nguyên của tiến trình cha :
  + Tiến trình cha và tiến trình con chia sẻ mọi tài nguyên.
  + Tiến trình con chia sẻ một phần tài nguyên của cha.
* Trình tự thực thi của tiến trình cha và tiến trình con : (Như câu trên).
* **Kết thúc tiến trình**, có 2 TH :
  + Tiến trình tự kết thúc : Tiến trình kết thúc khi thực hiện lệnh cuối cùng của nó và yêu cầu hệ điều hành xoá mình đi, bằng cách sử dụng lệnh gọi hệ thống (system call) exit().
  + Tiến trình kết thúc do tiến trình khác có quyền kết thúc nó (VD tiến trình cha kết thúc tiến trình con).
    - Gọi system routine abort với tham số là pid của tiến trình cần được kết thúc.
    - Tiến trình cha kết thúc và kéo theo tất cả tiến trình con của nó đều kết thúc.
  + Hệ điều hành thu hồi tất cả các tài nguyên của tiến trình kết thúc (vùng nhớ, I/O buffer,…).

**Câu 8 :** So sánh giữa hàm fork() và họ các hàm exec() trong việc tạo tiến trình con mới ?

**fork() :**

* fork() tạo ra tiến trình mới bằng cách nhân bản (duplicate) tiến trình gọi hàm này. Tiến trình được tạo ra từ hàm fork() được gọi là tiến trình con (child process), là một bản sao giống y hệt với tiến trình cha tạo ra nó (kể cả trạng thái thực thi), chỉ trừ một số điểm sau :
  + Tiến trình con có PID (Process Identifier) của riêng nó. PID này không trùng với bất kỳ PID nào hiện có.
  + Mỗi tiến trình con được tạo ra, đều có parent PID của nó trùng với PID của tiến trình gọi hàm fork() để tạo ra nó.
* fork() sẽ trả về :
  + Cho tiến trình gọi hàm (parent process) : PID của tiến trình con nếu tạo được tiến trình con. Ngược lại, nếu quá trình tạo tiến trình con bị lỗi, trả về -1.
  + Cho tiến trình con được tạo ra bởi hàm fork() : 0 (Số 0).
* Tiến trình con sau khi được tạo ra, sẽ thực thi tiếp tục tại vị trí của cha.
  + VD : Khi gặp lệnh **int pid = fork()**, tiến trình con sẽ được tạo ra.
  + Tiến trình con sẽ có pid = 0, tiến trình cha sẽ có pid = (pid của con). Hay nói cách khác, 1 tiến trình gọi hàm fork(), mà hàm fork() lại trả về 2 kết quả khác nhau về lại cho 2 tiến trình (tiến trình cha và tiến trình con).
  + Sau đó tiến trình con sẽ thực hiện lệnh tiếp theo của lệnh trên chứ không thực hiện lại từ đầu (do tiến trình con là **bản sao** của cha, vì vậy nó sẽ kế thừa trạng thái thực thi của cha, nên sẽ thực hiện tiếp từ hàm fork()).

**Họ các hàm exec() :**

* Họ các hàm exec() thay thế toàn bộ tiến trình hiện tại bằng một tiến trình mới và bắt đầu thực thi từ đầu của tiến trình mới.
* Điểm thú vị ở đây là tiến trình mới không nhất thiết phải là bản sao của tiến trình trước, nó có thể là một chương trình hoàn toàn khác.
* Sau khi gọi exec(), tiến trình hiện tại đang chạy sẽ bị xoá sạch và thay bằng tiến trình mới. Vì vậy, sau khi gọi lệnh exec(), bất kỳ lệnh nào sau nó ở tiến trình cũ đều không được thực thi. Chỉ duy nhất khi exec() bị lỗi, lúc đó nó mới trả về cho tiến trình cũ.

(Tham khảo thêm : [https://stackoverflow.com/a/4205020](https://l.facebook.com/l.php?u=https%3A%2F%2Fstackoverflow.com%2Fa%2F4205020&h=AT3AKAjK0She49HhngoTB1a5XpBIsBAxhQ7qkinee0aGOyV4bOUsEie7Vzf_LwkuRipPFax8h-jc6f12fwaOXt9Z1F0zOVl2UFyEz2ZZFWHdnq5er5IgkDfqQfVwiUW--CoHdJbB_LXgP_X5UVv-8BnsUrgTmiDTXVZ4yg)).

**Câu 9 :** Bộ nhớ của cha và con trong một quan hệ cha-con (parent-child process) được chia sẻ, cấp phát như thế nào ?

Thông thường, khi một tiến trình tạo ra một tiến trình con thì tiến trình con đó sẽ cần tài nguyên hệ thống (CPU time, memory, files, I/O devices) để thực thi công việc của mình. Tiến trình con có thể lấy được các tài nguyên cần thiết trực tiếp từ HĐH, hoặc từ tiến trình cha. Tiến trình cha có thể sẽ cần phân vùng lại các tài nguyên giữa các tiến trình con.

Khi tiến trình con được tạo ra từ lệnh fork(), tiến trình con sẽ là một bản sao y hệt với tiến trình cha tạo ra nó. Đồng thời, nó là một tiến trình độc lập, có một PCB riêng của mình, lưu trữ dữ liệu của riêng mình. Vì vậy, khi thay đổi giá trị 1 biến ở tiến trình con, tiến trình cha sẽ không bị ảnh hưởng. Và điều kiện ngược lại cũng đúng.

**Câu 10 (Đọc thêm) :** Các giải thuật định thời được dùng trong một số hệ điều hành thông dụng ?

* Windows-NT : Multilevel Feedback Queue.
* macOS : Multilevel Feedback Queue.
* Linux (từ 2.6.23 đến nay) : CFS (Completely Fair Scheduler).

**Chương 4 :**

**Câu 6 :** Hai yếu tố của giải thuật định thời là gì ?

* Hàm chọn lựa (selection function) : Dùng để chọn process nào trong ready queue được thực thi (tuỳ thuật toán định thời sẽ có cách chọn khác nhau).
* Chế độ quyết định (decision mode) : Chọn thời điểm thực thi hàm chọn lựa để định thời. Bao gồm 2 chế độ quyết định :
  + Non-Preemptive (Không trưng dụng) :
    - Khi ở trạng thái running, process sẽ thực thi cho đến khi kết thúc hoặc bị blocked do yêu cầu I/O.
  + Preemptive (Trưng dụng) :
    - Process đang thực thi (running) có thể bị ngắt và chuyển về trạng thái ready.
    - Chi phí cao hơn non-preemptive. Đổi lại, ta có được thời gian đáp ứng tốt hơn, vì không có trường hợp một process độc chiếm CPU quá lâu.

**Câu 7 :** Mô tả và nêu ưu điểm, nhược điểm của từng giải thuật định thời ? FCFS, SJF, SRTF, RR, Priority Scheduling, HRRN, MQ, MFQ. Chỉ có FCFS thì không có priority, còn lại thì đều có priority. Người ta sử dụng FC Queue nhiều nhất (tương đối)

**FCFS :**

* Mô tả :
  + Cơ chế thực thi :
    - Tiến trình nào yêu cầu CPU trước sẽ được cấp phát trước.
    - Tiến trình sẽ thực thi đến khi kết thúc hoặc bị blocked do I/O.
  + Chế độ quyết định : Non-Preemptive.
  + Hiện thực : Sử dụng hàng đợi FIFO.
    - Tiến trình đi vào được thêm vào cuối hàng đợi.
    - Tiến trình được lựa chọn để xử lý được lấy từ đầu của queue.
* Ưu điểm :
  + Sẽ không bị starvation.
  + Thuật toán này dễ cài đặt. Code đơn giản.
* Nhược điểm :
  + Thời gian chờ trung bình của FCFS thường khá dài (VD : Một process có burst-time rất dài đến trước, khi đó các process có burst-time nhỏ sẽ phải chờ 1 khoảng thời gian rất lâu mới đến lượt thực thi).
  + Lãng phí thời gian do thời gian phần cứng trống khá nhiều (convoy effect).
  + Non-preemptive. Sẽ không hoạt động tốt trong các hệ thống chia sẻ thời gian (time-sharing system) khi các user đều mong muốn được sử dụng CPU trong một khoảng thời gian và không muốn delay quá lâu.

**SJF :**

* Mô tả :
  + Cơ chế thực thi :
    - Định thời công việc ngắn nhất trước (Burst-time nhỏ nhất).
    - Khi CPU được tự do, nó sẽ cấp phát cho tiến trình nào yêu cầu ít thời gian nhất để kết thúc (burst-time nhỏ nhất).
    - Burst-time có được từ việc dự đoán, dựa vào các lần chạy trước của tiến trình.
    - Nếu có 2 tiến trình cùng Burst-time, tiến trình nào vào hàng đợi trước sẽ được chạy trước (không xét độ ưu tiên).
  + Chế độ quyết định : Non-Preemptive.
* Ưu điểm : Tối ưu. Cho thời gian chờ đợi trung bình tối thiểu với một tập tiến trình cho trước.
* Nhược điểm :
  + Cần phải ước lượng thời gian cần CPU tiếp theo của process (Burst time).
  + Có thể xảy ra starvation nếu số lượng process có burst time nhỏ cần được thực thi quá nhiều.

**SRTF :**

* Mô tả :
  + Cơ chế thực thi :
    - (Tương tự SJF).
    - Nếu một tiến trình mới được đưa vào danh sách với chiều dài sử dụng CPU cho lần tiếp theo **nhỏ hơn** (lưu ý, chỉ **nhỏ hơn,** nếu burst-time bằng thì không preempt) thời gian còn lại của tiến trình đang xử lý, nó sẽ dừng hoạt động tiến trình hiện hành (**preempt**).
  + Chế độ quyết định : Preemptive.
* Ưu điểm :
  + Preemptive. Thời gian đáp ứng nhanh cho các tác vụ nhỏ.
  + Tránh việc một tác vụ lớn độc chiếm CPU.
  + Thời gian chờ đợi trung bình thường sẽ nhỏ hơn SJF.
* Nhược điểm :
  + (Các nhược điểm của SJF).
  + Tăng thời gian hoàn thành trung bình.

**Priority Scheduling :**

* Mô tả :
  + Cơ chế hoạt động :
    - Mỗi tiến trình sẽ được gán 1 độ ưu tiên.
    - CPU sẽ được cấp cho tiến trình có độ ưu tiên cao nhất.
    - Định thời sử dụng độ ưu tiên có thể là :
      * Preemptive : Khi một tiến trình mới xuất hiện có độ ưu tiên cao hơn, nó sẽ preempt tiến trình đang chạy.
      * Non-Preemptive : Tiến trình đang chạy sẽ tiếp tục chạy.
    - Nếu có 2 tiến trình cùng độ ưu tiên, thì tiến trình nào đến trước sẽ được chạy trước. Burst-time không được áp dụng để so sánh ở đây.
  + Chế độ quyết định : Non-Preemptive hoặc Preemptive.
* Ưu điểm :
  + Các tác vụ quan trọng sẽ được thực thi trước.
* Nhược điểm :
  + Có thể xảy ra starvation : Các process có độ ưu tiên thấp có thể không bao giờ được thực thi (giải pháp : aging – Độ ưu tiên của process sẽ tăng theo thời gian).

**Round Robin :**

* Mô tả :
  + Cơ chế hoạt động :
    - Mỗi tiến trình nhận được một đơn vị nhỏ thời gian CPU (time-slice, quantum time), thông thường từ 10-100msec để thực thi.
    - CPU Schedulers sẽ chọn 1 tiến trình từ ready queue và “lên dây cót” một quantum cho tiến trình, sau đó cho tiến trình chạy. Lúc này, sẽ có 2 khả năng có thể xảy ra :
      * Thời gian chạy > Quantum : Khi đó, tiến trình sẽ bị interrupt và CPU Schedulers sẽ chọn tiếp tiến trình tiếp theo.
      * Thời gian chạy < Quantum : Tiến trình tiếp theo sẽ ngay lập tức được thực thi tiếp (không cần chờ hết quantum time của tiến trình trước), và tiến trình tiếp theo đó cũng được gán 1 quantum time.
    - Phụ thuộc nhiều vào quantum time :
      * Quantum time ngắn thì đáp ứng nhanh, tuy nhiên overhead lớn do chuyển ngữ cảnh nhiều. Quantum time phải > thời gian chuyển ngữ cảnh (context switch).
      * Quantum time dài thì đáp ứng chậm, tuy nhiên thông lượng (throughput) sẽ cao. Và khi quantum time quá lớn RR->FCFS (Quantum time lớn -> Không bao giờ bị ngắt -> Ai vào trước làm trước -> FCFS).
    - Khi cả tiến trình vừa thực thi xong và tiến trình mới cũng arrive vào cùng một thời điểm, thì tiến trình mới sẽ vào hàng đợi trước rồi mới đến tiến trình cũ.
    - Các tiến trình đều có độ ưu tiên giống nhau.
  + Chế độ quyết định : Preemptive.
* Ưu điểm :
  + Thời gian đáp ứng trung bình thường thấp -> Thích hợp cho các hệ thống time-sharing.
  + Không xảy ra tình trạng starvation.
* Nhược điểm :
  + Thời gian chờ đợi trung bình thường khá lớn.
  + Chuyển ngữ cảnh nhiều -> Hao phí cao.
  + Hiệu suất thuật toán phụ thuộc nhiều vào việc chọn quantum time.
  + Không thể sử dụng thuật toán nếu muốn các ứng dụng có độ ưu tiên khác nhau.

**Highest Response Ratio Next (HRRN) :**

* Mô tả :
  + Cơ chế hoạt động :
    - Chọn process tiếp có giá trị RR (Response Ratio) **lớn nhất**.
    - Các process ngắn được ưu tiên hơn vì service time (hay burst time) nhỏ.
  + Công thức :



Công thức tính RR (Response Ratio) của thuật toán HRRN.

* Ưu điểm :
  + Không xảy ra starvation.
  + Tự động cân bằng giữa việc ưu tiên một tiến trình có thời gian thực thi nhỏ và một tiến trình đã ở quá lâu trong hệ thống (aging).
* Nhược điểm :
  + Non-Preemptive.

**Multilevel Queue Scheduling :**

* Mô tả :
  + Cơ chế hoạt động :
    - Hàng đợi ready được chia thành nhiều hàng đợi riêng biệt theo một số tiêu chuẩn như :
      * Đặc điểm và yêu cầu định thời của process.
      * Foreground (interactive) và background process.
    - Process được gán cố định vào một hàng đợi, mỗi hàng đợi sẽ sử dụng một giải thuật riêng.
    - Có 2TH hệ điều hành định thời cho các hàng đợi :
      * Có một độ ưu tiên cố định cho từng hàng đợi (fixed priority scheduling).
        + Hàng đợi có độ ưu tiên cao hơn phải được chạy xong (empty) trước khi hàng đợi có độ ưu tiên thấp hơn được phép chạy.
        + Nếu có 1 tiến trình đi vào hàng đợi có độ ưu tiên cao hơn trong khi hàng đợi có độ ưu tiên thấp hơn đang được thực thi, hàng đợi có độ ưu tiên thấp hơn đó sẽ bị preempt.
      * Time-slice : Mỗi hàng đợi nhận được một khoảng thời gian chiếm CPU và phân phối cho các process trong hàng đợi khoảng thời gian đó.
  + Chế độ quyết định : Non-Preemptive hoặc Preemptive.

A diagram of process processes

Description automatically generated

Ví dụ về các hàng đợi được phân cấp trong thuật toán MQS.

* Ưu điểm :
  + Áp dụng nhiều giải thuật định thời cho nhiều loại tiến trình có độ ưu tiên khác nhau.
  + Cho phép các CPU-Bound process được ưu tiên hơn trong việc thực thi -> Thời gian hệ thống thực thi tác vụ được cải thiện.
  + Có thể hoạt động trong cả 2 chế độ : Preemptive và Non-Preemptive.
* Nhược điểm :
  + Các hàng đợi đa cấp này cần được giám sát -> Hao phí tài nguyên hệ thống.
  + Process không thể di chuyển từ hàng đợi này sang hàng đợi khác -> Không linh động.

**Multilevel Feedback Queue**

* Mô tả :
  + Cơ chế hoạt động :
    - (Tương tự Multilevel Feedback Queue).
    - Điểm khác biệt : Cho phép process nhảy từ queue này đến queue khác.
* Ưu điểm :
  + Thích nghi với các tiến trình. VD : Một tiến trình nếu sử dụng quá nhiều CPU time thì sẽ xếp nó vào queue có độ ưu tiên thấp hơn.
  + Aging. VD : Một process đã xuất hiện lâu mà không được thực thi, sẽ được đưa lên 1 queue có độ ưu tiên cao hơn.
  + Thuật toán chung nhất, có thể được thiết kế để phù hợp với các hệ thống khác biệt.
* Nhược điểm :
  + Tốn tài nguyên hệ thống để duy trì các queue -> Có thể không thích hợp đối với các hệ thống nhỏ.
  + Thiết kế phức tạp.