**ASSIGNMENT: EXERCISES**

**Câu 1:**

- Nếu sử dụng thuật toán thay trang FIFO với bốn khung trang và tám trang, ta sẽ tính số lỗi trang (page faults) sẽ xảy ra với chuỗi tham chiếu 0172327103 khi bốn khung ban đầu rỗng.

+ Chuỗi tham chiếu: 0172327103

+ Số khung trang: 4

🡪 Ban đầu, tất cả các khung đều trống. Ta sẽ duyệt qua chuỗi tham chiếu và mô phỏng quá trình thay trang. Mỗi khi xảy ra lỗi trang, ta sẽ đếm nó.

* Bước 1: 0 (Lỗi trang, khung 0: [0, -, -, -])
* Bước 2: 1 (Lỗi trang, khung 1: [0, 1, -, -])
* Bước 3: 7 (Lỗi trang, khung 2: [0, 1, 7, -])
* Bước 4: 2 (Lỗi trang, khung 3: [0, 1, 7, 2])
* Bước 5: 3 (Lỗi trang, khung 3: [3, 1, 7, 2])
* Bước 6: 2 (Không lỗi trang, khung 3: [3, 1, 7, 2])
* Bước 7: 7 (Không lỗi trang, khung 3: [3, 1, 7, 2])
* Bước 8: 1 (Không lỗi trang, khung 3: [3, 1, 7, 2])
* Bước 9: 0 (Lỗi trang, khung 3: [3, 0, 7, 2])
* Bước 10: 1 (Lỗi trang, khung 3: [3, 0, 1, 2])
* Bước 10: 3 (Không lỗi trang, khung 3: [3, 0, 1, 2])

🡪 Có 7 lần lỗi trang

- Nếu sử dụng thuật toán thay trang LRU (Least Recently Used) với bốn khung trang và tám trang, ta sẽ tính số lỗi trang (page faults) sẽ xảy ra với chuỗi tham chiếu 0172327103 khi bốn khung ban đầu rỗng.

+ Chuỗi tham chiếu: 0172327103

+ Số khung trang: 4

🡪 Ban đầu, tất cả các khung đều trống. Ta sẽ duyệt qua chuỗi tham chiếu và mô phỏng quá trình thay trang. Mỗi khi xảy ra lỗi trang, ta sẽ đếm nó.

* Bước 1: 0 (Lỗi trang, khung 0: [0, -, -, -])
* Bước 2: 1 (Lỗi trang, khung 1: [0, 1, -, -])
* Bước 3: 7 (Lỗi trang, khung 2: [0, 1, 7, -])
* Bước 4: 2 (Lỗi trang, khung 3: [0, 1, 7, 2])
* Bước 5: 3 (Không lỗi trang, khung 3: [0, 1, 7, 3])
* Bước 6: 2 (Không lỗi trang, khung 3: [0, 1, 7, 2])
* Bước 7: 7 (Không lỗi trang, khung 3: [0, 1, 7, 2])
* Bước 8: 1 (Lỗi trang, khung 0: [1, 0, 7, 2])
* Bước 9: 0 (Không lỗi trang, khung 0: [1, 0, 7, 2])
* Bước 10: 3 (Lỗi trang, khung 2: [1, 0, 3, 2])

🡪 Tổng số lỗi trang sử dụng thuật toán thay trang LRU là 5.

**Câu 2:**

- Trong trường hợp này, ta sử dụng thuật toán aging với một bộ đếm 8-bit để quản lý các trang trên smart card. Ta được cung cấp thông tin về các giá trị của bit R (reference bit) trong các khung trang sau mỗi lần nhịp đồng hồ. Ta cần tính giá trị của bốn bộ đếm sau lần nhịp đồng hồ cuối cùng.

- Ban đầu, giá trị của các bộ đếm là 0 với 8-bit:

Bộ đếm 0: 00000000

Bộ đếm 1: 00000000

Bộ đếm 2: 00000000

Bộ đếm 3: 00000000

- Với mỗi lần nhịp đồng hồ, ta thực hiện việc cập nhật các bộ đếm theo giá trị của bit R và dịch trái bộ đếm. Nếu bit R là 1, ta đặt bit trái nhất của bộ đếm thành 1, ngược lại, ta giữ bit trái nhất của bộ đếm là 0.

- Lần lượt, ta áp dụng quy tắc này cho các giá trị bit R đã cung cấp:

- Lần nhịp đồng hồ thứ nhất (R = 0111):

Bộ đếm 0: 00111110

Bộ đếm 1: 00000000

Bộ đếm 2: 00000000

Bộ đếm 3: 00000000

- Lần nhịp đồng hồ thứ hai (R = 1011):

Bộ đếm 0: 10011111

Bộ đếm 1: 00000000

Bộ đếm 2: 00000000

Bộ đếm 3: 00000000

- Lần nhịp đồng hồ thứ ba (R = 1010):

Bộ đếm 0: 11001111

Bộ đếm 1: 00000000

Bộ đếm 2: 00000000

Bộ đếm 3: 00000000

- Lần nhịp đồng hồ thứ tư (R = 1101):

Bộ đếm 0: 11100111

Bộ đếm 1: 00000000

Bộ đếm 2: 00000000

Bộ đếm 3: 00000000

- Lần nhịp đồng hồ thứ năm (R = 0010):

Bộ đếm 0: 01110011

Bộ đếm 1: 00000000

Bộ đếm 2: 00000000

Bộ đếm 3: 00000000

- Lần nhịp đồng hồ thứ sáu (R = 1010):

Bộ đếm 0: 10111001

Bộ đếm 1: 00000000

Bộ đếm 2: 00000000

Bộ đếm 3: 00000000

- Lần nhịp đồng hồ thứ bảy (R = 1100):

Bộ đếm 0: 11011100

Bộ đếm 1: 00000000

Bộ đếm 2: 00000000

Bộ đếm 3: 00000000

- Lần nhịp đồng hồ cuối cùng (R = 0001):

Bộ đếm 0: 01101110

Bộ đếm 1: 00000000

Bộ đếm 2: 00000000

Bộ đếm 3: 00000000

- Cuối cùng, giá trị của bốn bộ đếm là:

Bộ đếm 0: 01101110

Bộ đếm 1: 00000000

Bộ đếm 2: 00000000

Bộ đếm 3: 00000000

**Câu 3:**

- Trong trường hợp này, chúng ta có một hệ thống với địa chỉ ảo 48-bit và địa chỉ vật lý 32-bit. Kích thước trang là 8 KB.

- Để tính số lượng mục cần thiết cho bảng trang mức đơn, chúng ta cần biết kích thước trang và kích thước một mục trong bảng trang.

- Kích thước trang: 8 KB = 8 \* 1024 bytes = 8192 bytes

- Kích thước một mục trong bảng trang: 32-bit (địa chỉ vật lý) = 4 bytes

- Để tính số lượng mục cần thiết, chúng ta sẽ chia kích thước tối đa của không gian ảo (48-bit) cho kích thước trang (8192 bytes), sau đó chia kết quả cho kích thước một mục (4 bytes):

- Số lượng mục = (2^48 bytes) / (8192 bytes) / (4 bytes)

Số lượng mục = (2^48) / (8192 \* 4)

**Câu 4:**

- Có một số lý do vì sao trong nhiều hệ thống, chúng ta vẫn giữ các hoạt động "copy", "delete" và "move" riêng biệt, thay vì thay thế "move" bằng "copy" và "delete". Dưới đây là một số lý do chính:

* Hiệu suất: Trong nhiều trường hợp, việc di chuyển một tệp tin hoặc dữ liệu từ một vị trí đến vị trí khác không yêu cầu sao chép toàn bộ nội dung. Thay vào đó, hệ thống chỉ cần cập nhật các liên kết hoặc siêu dữ liệu để chỉ định vị trí mới của tệp tin hoặc dữ liệu đó. Việc thực hiện một hoạt động "move" sẽ nhanh hơn và hiệu quả hơn so với việc thực hiện "copy" và sau đó "delete".
* Khả năng tự động: Trong một số trường hợp, hệ điều hành hoặc hệ thống tệp tin có thể tự động nhận biết khi một tệp tin hoặc dữ liệu được di chuyển trong cùng một ổ đĩa hoặc phân vùng. Trong trường hợp này, nếu chúng ta thực hiện một hoạt động "move", hệ thống có thể điều chỉnh các thông số nội bộ mà không cần sao chép dữ liệu.
* Nguyên tắc bảo toàn dữ liệu: Trong một số trường hợp, việc thực hiện một hoạt động "move" có thể dẫn đến mất mát dữ liệu nếu xảy ra lỗi trong quá trình di chuyển. Thay vì thực hiện một hoạt động "move" duy nhất, việc thực hiện "copy" và sau đó "delete" tạo ra một bản sao dữ liệu trước khi xóa bản gốc. Điều này đảm bảo tính toàn vẹn dữ liệu và giảm nguy cơ mất mát dữ liệu.

- Tuy nhiên, trong một số trường hợp đặc biệt, việc thay thế "move" bằng "copy" và "delete" có thể được sử dụng để đảm bảo tính toàn vẹn dữ liệu hoặc để đơn giản hóa quá trình thực hiện. Quyết định cuối cùng sẽ phụ thuộc vào thiết kế và yêu cầu cụ thể của hệ thống.

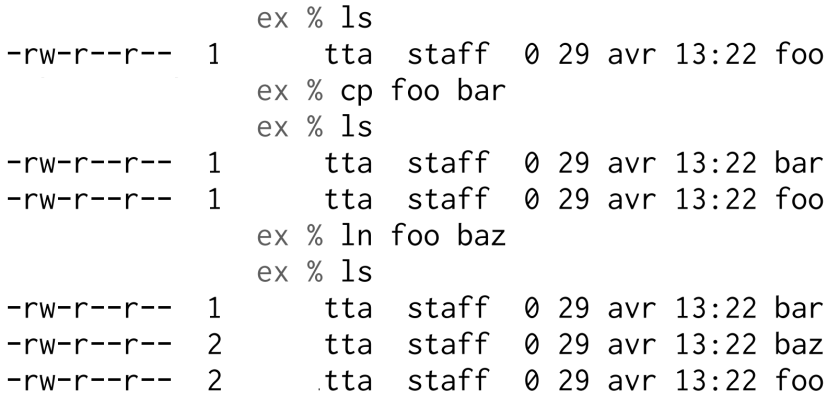
**Câu 5:**

- Khi cấu trúc thư mục được biểu diễn dưới dạng đồ thị (graph) thay vì cây (tree), có một số vấn đề có thể xảy ra. Dưới đây là một số vấn đề chính:

* Đường dẫn vô hạn: Trong một đồ thị, có thể tồn tại các chu trình (cycles) trong cấu trúc thư mục. Điều này có thể dẫn đến việc tồn tại đường dẫn vô hạn khi điều hướng trong cấu trúc thư mục. Một ví dụ đơn giản là khi một thư mục A chứa thư mục B và thư mục B chứa thư mục A. Khi đó, không có đường dẫn hợp lệ để xác định vị trí của một tệp tin hoặc thư mục.
* Trùng lặp và xung đột tên: Trong đồ thị, các thư mục có thể được liên kết với nhau thông qua nhiều đường dẫn khác nhau. Điều này có thể dẫn đến trùng lặp tên thư mục hoặc xung đột tên khi thực hiện các thao tác quản lý thư mục như tạo mới, đổi tên, hoặc xóa. Không có một định nghĩa rõ ràng và duy nhất cho mỗi tên thư mục, gây khó khăn trong việc định vị và xử lý các thư mục.
* Độ phức tạp của việc tìm kiếm: Khi cấu trúc thư mục là đồ thị, việc tìm kiếm một tệp tin hoặc thư mục có thể trở nên phức tạp hơn. Do không có sự cân nhắc theo thứ tự, việc tìm kiếm sẽ đòi hỏi quá trình duyệt qua các đường dẫn khác nhau trong đồ thị, có thể dẫn đến thời gian tìm kiếm tăng lên đáng kể.
* Độ phức tạp của thao tác quản lý: Khi cấu trúc thư mục là đồ thị, các thao tác quản lý như di chuyển, sao chép, hoặc xóa thư mục cũng trở nên phức tạp hơn. Do sự liên kết tự do trong đồ thị, việc xác định và thực hiện các thao tác này có thể mất nhiều thời gian và tài nguyên hơn so với trong cây thư mục.
* Đồng bộ hóa và xử lý xung đột: Khi có nhiều người dùng hoặc quy trình truy cập cùng một cấu trúc thư mục dạng đồ thị, việc đồng bộ hóa và xử lý xung đột trở nên phức tạp hơn. Các xung đột xảy ra khi có nhiều thao tác cùng xảy ra trên cùng một đối tượng trong đồ thị, gây ra khó khăn trong việc đảm bảo tính nhất quán và đúng đắn của cấu trúc thư mục.

- Trên thực tế, cấu trúc thư mục thường được tổ chức dưới dạng cây để tránh những vấn đề phức tạp và không mong muốn có thể xảy ra khi sử dụng đồ thị.

**Câu 6:**



- Dựa trên các lệnh và đầu ra đã được cung cấp, dưới đây là sự khác biệt giữa các tệp "bar" và "baz":

* "bar": Tệp này được tạo ra bằng cách sử dụng lệnh "cp", viết tắt của "copy" (sao chép). Nó là một bản sao chính xác của tệp gốc "foo". Bất kỳ thay đổi nào được thực hiện trên "foo" sau khi tạo "bar" sẽ không ảnh hưởng đến "bar", vì chúng là hai tệp độc lập.
* "baz": Tệp này được tạo ra bằng cách sử dụng lệnh "ln", viết tắt của "link" (liên kết) hoặc "symbolic link" (liên kết tượng trưng). "baz" là một liên kết tượng trưng đến tệp gốc "foo". Điều này có nghĩa là "baz" hoạt động như một con trỏ hoặc tham chiếu đến "foo". Bất kỳ thay đổi nào được thực hiện trên "foo" cũng sẽ được phản ánh trong "baz", vì chúng về cơ bản là cùng một tệp với các tên khác nhau.

- Nếu chúng ta ghi chuỗi "abcde" vào "foo", thì nội dung của tệp "foo" sẽ được thay đổi thành "abcde". Sự thay đổi này sẽ được phản ánh trong "baz" vì "baz" là một liên kết tượng trưng đến "foo". Tuy nhiên, "bar" vẫn giữ nguyên nội dung trước đó và không bị ảnh hưởng bởi sự thay đổi này.