**Họ và tên: Lê Anh Thư**

**MSSV: 20521985**

# Lớp: IT007.M21.HTCL

**BÀI NỘP HỆ ĐIỀU HÀNH CHƯƠNG 7 & CHƯƠNG 8**

**CÂU HỎI VÀ BÀI TẬP CHƯƠNG 7**

**Câu 1: Quản lý bộ nhớ là gì? Các yêu cầu đối với quản lý bộ nhớ là gì?**

* Quản lý bộ nhớ là công việc của hệ điều hành với sự hỗ trợ của phần cứng nhằm phân phối, sắp xếp các process trong bộ nhớ sao cho hiệu quả.
* Mục tiêu cần đạt được là nạp càng nhiều process vào bộ nhớ càng tốt (gia tăng mức độ đa chương)

Trong hầu hết các hệ thống, kernel sẽ chiếm một phần cố định của bộ nhớ; phần còn lại phân phối cho các process

* Các yêu cầu đối với việc quản lý bộ nhớ
  + Cấp phát bộ nhớ cho các process
  + Tái định vị (relocation): khi swapping,…
  + Bảo vệ: phải kiểm tra truy xuất bộ nhớ có hợp lệ không
  + Chia sẻ: cho phép các process chia sẻ vùng nhớ chung
  + Kết gán địa chỉ nhớ luận lý của user vào địa chỉ thực

**Câu 2: Trình bày đặc điểm các loại địa chỉ bộ nhớ?**

Đặc điểm các loại địa chỉ bộ nhớ

* Địa chỉ vật lý (physical address) (địa chỉ thực) là một vị trí thực trong bộ nhớ chính
* Địa chỉ luận lý (logical address) là một vị trí nhớ được diễn tả trong một chương trình (còn gọi là địa chỉ ảo virtual address).
  + Các trình biên dịch (compiler) tạo ra mã lệnh chương trình mà trong đó mọi tham chiếu bộ nhớ đều là địa chỉ luận lý
  + Địa chỉ tương đối (relative address) (địa chỉ khả tái định vị, relocatable address) là một kiểu địa chỉ luận lý trong đó các địa chỉ được biểu diễn tương đối so với một vị trí xác định nào đó trong chương trình.
* Ví dụ: 12 byte so với vị trí bắt đầu chương trình,…
  + Địa chỉ tuyệt đối (absolute address): địa chỉ tương đương với địa chỉ thực.

**Câu 3: Địa chỉ lệnh và dữ liệu được chuyển đổi thành địa chỉ thực có thể xảy ra tại những thời điểm nào?**

Địa chỉ lệnh và dữ liệu được chuyển đổi thành địa chỉ thực có thể xảy ra tại ba thời điểm khác nhau :

* Compile time : Nếu biết trước địa chỉ bộ nhớ của chương trình thì có thể kết gắn địa chỉ tuyệt đối lúc biên dịch.
  + Ví dụ : Chương trình .COM của Microsoft.
  + Khuyết điểm : Phải biên dịch lại nếu thay đổi địa chỉ nạp chương trình.
* Load time : Vào thời điểm loading, loader phải chuyển đổi địa chỉ khả tái định vị thành địa chỉ thực dựa trên một địa chỉ nền.
  + Địa chỉ thực được tính toán vào thời điểm nạp chương trình -> Phải tiến hành reload nếu địa chỉ nền thay đổi.
* Execution time : Khi trong quá trình thực thi, process có thể được di chuyển từ segment này sang segment khác trong bộ nhớ thì quá trình chuyển đổi địa chỉ được trì hoãn đến thời điểm thực thi.

**Câu 4: Liên kết động (dynamic linking) là gì?**

Dynamic linking là quá trình link đến một module ngoài (external module) được thực hiện sau khi đã tạo xong load module.

Ví dụ trong Windows, module ngoài là các file .DLL. Trong UNIX, module ngoài là các file .so (shared library).

Load module chứa các stub tham chiếu (refer) đến routine của external module. Lúc thực thi, khi stub được thực thi lần đầu (do process gọi routine lần đầu), stub nạp routine vào bộ nhớ, tự thay thế bằng địa chỉ của routine và routine được thực thi.

Các lần gọi routine sau sẽ xảy ra bình thường.

Ưu điểm :

* Dùng các phiên bản khác nhau của external module mà không cần biên dịch lại.
* Chia sẻ mã : Một external module chỉ cần nạp vào bộ nhớ một lần và có thể dùng chung giữa các process.

**Câu 5: Phân mảnh ngoại và phân mảnh nội là gì?**

Phân mảnh ngoại tồn tại khi tổng lượng bộ nhớ còn trống đủ để phục vụ một yêu cầu mới từ process, tuy nhiên do các phần trống đó không liên tục nhau mà nằm ở nhiều vị trí cách xa nhau, vì vậy có bộ nhớ trống mà không thể cấp phát được.

Diagram

Description automatically generated

Phân mảnh ngoại.

Phân mảnh ngoại xảy ra do việc nạp và rút các process trong bộ nhớ. Như hình trên các bạn thấy, những ô xám từng là nơi những process nhỏ được nạp vào hệ thống trước đó. Những process nhỏ đó đã hoàn thành và rút ra khỏi hệ thống, để lại phần trống của bộ nhớ rời rạc như trên. Bây giờ, process 7 muốn vào nhưng không được, do không có vị trí nào đủ bộ nhớ để cung cấp cho nó.

Phương pháp giải quyết vấn đề phân mảnh ngoại là **compaction (gom cụm).** Compaction tức là việc gom các process đang trong bộ nhớ lại thành 1 cục bự, có thể nằm ở đầu hoặc nằm ở cuối bộ nhớ.

Một phương án giải quyết khác đó chính là sử dụng **phân đoạn (segmentation)** và **phân trang (paging)** để tận dụng tối đa các phần trống trong bộ nhớ, thậm chí cả những vùng không liên tục (non-contiguous).

Bộ nhớ cũng có thể bị phân mảnh nội. Phân mảnh nội là hiện tượng khi một process chiếm dung lượng bộ nhớ nhỏ hơn dung lượng mà nó được cấp phát (tức là phần bộ nhớ không được sử dụng nằm bên trong phân vùng đang được cấp phát cho process).

Diagram

Description automatically generated with low confidence

Phân mảnh nội.

VD : Như hình trên, process đầu tiên được cấp 100KB bộ nhớ nhưng chỉ dùng 80KB, để lại 20KB thừa. Process thứ 2 được cấp 100KB bộ nhớ nhưng chỉ dùng 90KB, để lại 10KB thừa. Phần thừa đó chính là phần phân mảnh nội.

Hiện tượng phân mảnh nội thường xảy ra khi bộ nhớ thực được chia thành các khối kích thước cố định (fixed-size block) và các process khi yêu cầu bộ nhớ, sẽ được cấp phát từng khối trong bộ nhớ.

Cơ chế phân trang (paging) là một ví dụ điển hình gây ra phân mảnh nội.

Hướng giải quyết có thể có là chia bộ nhớ ra thành các phần có kích thích tuỳ chỉnh và sử dụng chiến lược **Best-Fit** để phân bộ nhớ cho process. Hay nói cách khác, chúng ta sẽ tìm cách làm sao để cung cấp **chỉ vừa đủ dùng** bộ nhớ cho process.

**Câu 6: Như thế nào là phân vùng động, phân vùng cố định? Các chiến lược placement dùng để làm gì? Có những chiến lược nào?**

* Phân vùng cố định

Khi khởi động hệ thống, bộ nhớ chính được chia thành nhiều phần rời nhau gọi là các partition có kích thước bằng nhau hoặc khác nhau. Từng phần sẽ được cung cấp cho một và chỉ một process. Đó là Fixed-Partition.

Table

Description automatically generated

Fixed-Partitioning.

Process nào vừa với kích thước partition thì có thể được nạp vào partition đó (vừa tức là có kích thước nhỏ hơn hoặc bằng kích thước partition).

Nếu chương trình có kích thước lớn hơn partition thì phải dùng cơ chế overlay.

Fixed-Partitioning không hiệu quả do bị phân mảnh nội : Một chương trình dù lớn hay nhỏ đều được cấp phát trọn một partition.

Các chiến lược Placement (đưa process vào bộ nhớ) :

Partition có kích thước bằng nhau.

* Nếu còn partition trống => Process mới sẽ được nạp vào partition đó.
* Nếu không còn partition trống, nhưng trong đó có process đang bị blocked => Swap process đó ra bộ nhớ phụ, nhường chỗ cho process mới.

Nếu các partition không bằng nhau :

Giải pháp 1 :

* Gán mỗi process vào partition nhỏ nhất phù hợp với nó.
* Có hàng đợi cho mỗi partition.
* Giảm thiểu phân mảnh nội.
* Vấn đề ở đây là : Sẽ có một số hàng đợi trống không (vì không có process với kích thước tương ứng), trong khi đó lại có partition với hàng đợi dày đặc.

Diagram

Description automatically generated

Nhiều hàng đợi cho nhiều partition.

Giải pháp 2 :

* Dùng chung 1 hàng đợi cho tất cả các partition.
* Khi cần nạp một process vào bộ nhớ chính => Chọn partition nhỏ nhất còn trống.

Diagram

Description automatically generated

Một hàng đợi cho nhiều partition.

* Phân vùng động:

Dynamic Partitioning sẽ chia bộ nhớ thành nhiều phần như Fixed-Partition, tuy nhiên số lượng partition không cố định và partition có thể có kích thước khác nhau.

Mỗi process được cấp phát chính xác dung lượng bộ nhớ cần thiết.

Dynamic Partitioning không gây ra hiện tượng phân mảnh nội, nhưng lại gây ra hiện tượng phân mảnh ngoại.

Chiến lược placement :

* Mục tiêu : Giảm chi phí **compaction.** (chi phí vận chuyển các partition).
* Các chiến lượng placement :
  + Best-Fit : Chọn khối nhớ trống nhỏ nhất đủ cho process.
  + First-Fit : Chọn khối nhớ trống phù hợp đầu tiên kể từ đầu bộ nhớ.
  + Next-Fit : Chọn khối nhớ trống phù hợp đầu tiên kể từ vị trí cấp phát cuối cùng (nếu chưa có khối nhớ nào được cấp phát thì khối nhớ đầu tiên của bộ nhớ sẽ được chọn).
  + Worst-Fit : Chọn khối nhớ trống lớn nhất.

**Câu 7: Cơ chế phân trang là gì? Cơ chế chuyển đổi địa chỉ trong phân trang được thực hiện như thế nào?**

Cơ chế phân trang:

Không gian địa chỉ của 1 quá trình được chia thành những khối có cùng kích thước gọi là trang. Kỹ thuật phân trang (paging) cho  phép không gian địa chỉ vật lý (physical address space) của một process không cần liên tục.

* Bộ nhớ vật lý → khung trang (frame).
  + Kích thước của frame là lũy thừa của 2, từ khoảng 512 byte đến 16MB.
* Bộ nhớ luận lý (logical memory) hay không gian địa chỉ luận lý là tập mọi địa chỉ luận lý mà một chương trình bất kỳ có thể sinh ra → page.
  + Ví dụ
  + MOV REG,1000 //1000 là một địa chỉ luận lý
* Bảng phân trang (page table) để ánh xạ địa chỉ luận lý thành địa chỉ thực

Graphical user interface

Description automatically generated

Chart

Description automatically generated with low confidence

* Cơ chế chuyển đổi địa chỉ trong phân trang:

Địa chỉ luận lý gồm có:

* + Số hiệu trang (Page number) p
  + Địa chỉ tương đối trong trang (Page offset) d

Nếu kích thước của không gian địa chỉ ảo là 2m, và kích thước của trang là 2n (đơn vị là byte hay word tùy theo kiến trúc máy) thì

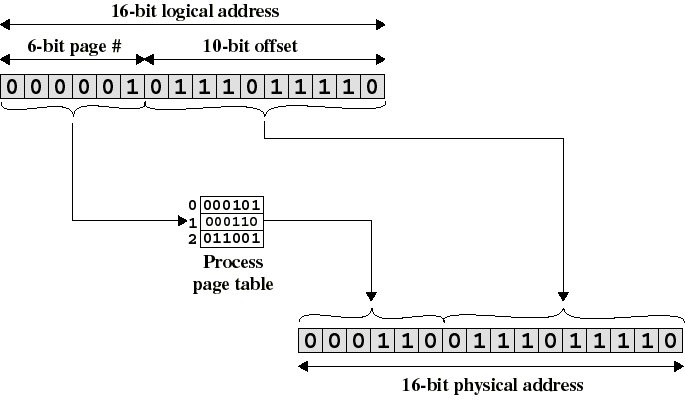
Table

Description automatically generated

Diagram

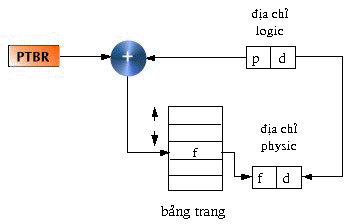
Description automatically generated

Ví du:

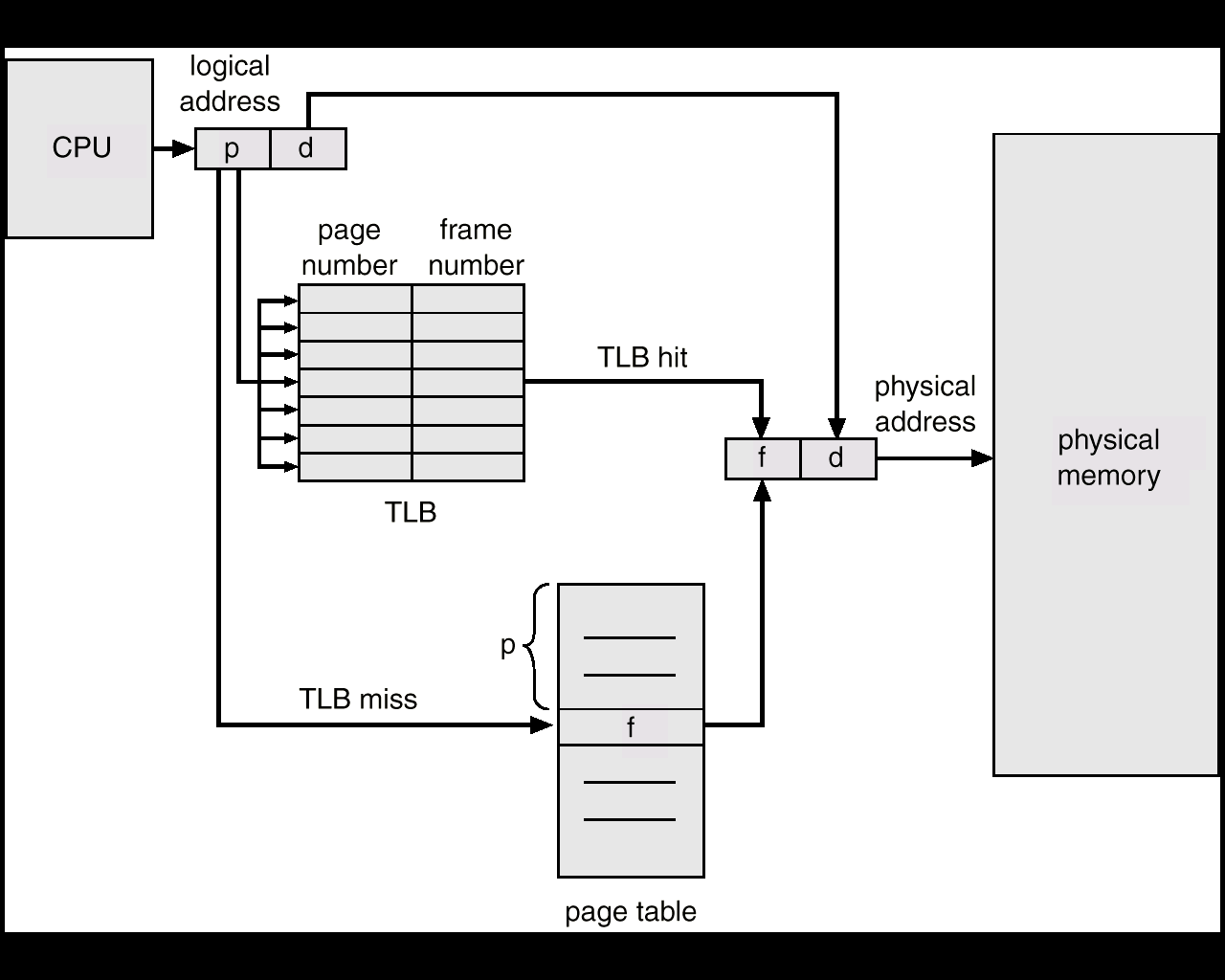


**Câu 8: Bảng trang được cài đặt và tổ chức như thế nào? Thời gian truy xuất hiệu dụng (effective access time) được xác định như thế nào?**

* Bảng trang được cài đặt và tổ chức
  + - Bảng trang được cài đặt:
* Bảng phân trang thường được lưu giữ trong bộ nhớ chính
  + Mỗi process được hệ điều hành cấp một bảng phân trang
  + Thanh ghi page-table base (PTBR) trỏ đến bảng phân trang
  + Thanh ghi page-table length (PTLR) biểu thị kích thước của bảng phân trang (có thể được dùng trong cơ chế bảo vệ bộ nhớ)
* Thường dùng một bộ phận cache phần cứng có tốc độ truy xuất và tìm kiếm cao, gọi là thanh ghi kết hợp (associative register) hoặc translation look-aside buffers (TLBs)
* Dùng thanh ghi Page-Table Base Register (PTBR)



* Dùng TLB



* + - Tổ chức bảng trang:
* Các hệ thống hiện đại đều hỗ trợ không gian địa chỉ ảo rất lớn (232 đến 264), ở đây giả sử là 232
  + Giả sử kích thước trang nhớ là 4KB (= 212) ⇒ bảng phân trang sẽ có 232/212 = 220 = 1M mục.
  + Giả sử mỗi mục gồm 4 byte thì mỗi process cần 4MB cho bảng phân trang
* Ví dụ: Phân trang 2 cấp

Chart

Description automatically generated with medium confidence

Diagram

Description automatically generated

Bảng trang nghịch đảo (IBM system/38, IBM RISC, IBM RT): sử dụng cho tất cả các Process

<IDP,p,d>



* Thời gian truy xuất hiệu dụng (effective access time) được xác định:
* Tính thời gian truy xuất hiệu dụng (effective access time, EAT)
* Thời gian tìm kiếm trong TLB (associative lookup): ε
* Thời gian một chu kỳ truy xuất bộ nhớ: x
* Hit ratio: tỉ số giữa số lần chỉ số trang được tìm thấy (hit) trong TLB và số lần truy xuất khởi nguồn từ CPU
  + Kí hiệu hit ratio: α
* Thời gian cần thiết để có được chỉ số frame
  + Khi chỉ số trang có trong TLB (hit) ε + x
  + Khi chỉ số trang không có trong TLB (miss) ε + x + x
* Thời gian truy xuất hiệu dụng

EAT = (ε + x)α + (ε + 2x)(1 – α)

= (2 – α)x + ε

* Ví dụ 1: đơn vị thời gian nano giây
  + Associative lookup = 20
  + Memory access = 100
  + Hit ratio = 0.8
  + EAT = (100 + 20) × 0.8 + (200 + 20) × 0.2  
     = 1.2 × 100 + 20

= 140

**Câu 9: Cơ chế hoán vị là gì?**

Một process có thể tạm thời bị swap ra khỏi bộ nhớ chính và lưu trên một hệ thống lưu trữ phụ. Sau đó, process có thể được nạp lại vào bộ nhớ để tiếp tục quá trình thực thi.

* RR (Round Robin) : Swap in-out theo quantum time.
* Roll out, roll in : Process có độ ưu tiên thấp sẽ bị swap out, nhường chỗ cho process có độ ưu tiên cao.

Diagram

Description automatically generated

Process Swap-In, Swap-Out.

**Câu 10: (Bài tập mẫu) Giả sử bộ nhớ chính được cấp phát thành các phân vùng có kích thước là 600K, 500K, 200K, 300K (theo thứ tự), sau khi thực thi xong, các tiến trình có kích thước 212K, 417K, 112K, 426K (theo thứ tự) sẽ được cấp phát bộ nhớ như thế nào, nếu sư dụng: Thuật toán First fit, Best fit, Next fit, Worst fit? Thuật toán nào cho phép sử dụng bộ nhớ hiệu quả nhất trong trường hợp trên?**

**Trả lời:**

a. Xét trường hợp bộ nhớ được phân vùng cố định:

\*First fit:

Tiến trình 212K sẽ được cấp phát vùng nhớ 600K.

Tiến trình 417K sẽ được cấp phát vùng nhớ 500K.

Tiến trình 112K sẽ được cấp phát vùng nhớ 200K.

Tiến trình 426K sẽ phải chờ vì không còn vùng nhớ trống thỏa yêu cầu.

\* Best fit:

Tiến trình 212K sẽ được cấp phát vùng nhớ 300K.

Tiến trình 417K sẽ được cấp phát vùng nhớ 500K.

Tiến trình 112K sẽ được cấp phát vùng nhớ 200K.

Tiến trình 426K sẽ được cấp phát vùng nhớ 600K.

\* Next fit:

Tiến trình 212K sẽ được cấp phát vùng nhớ 600K.

Tiến trình 417K sẽ được cấp phát vùng nhớ 500K.

Tiến trình 112K sẽ được cấp phát vùng nhớ 200K.

Tiến trình 426K sẽ sẽ phải chờ vì không còn vùng nhớ trống thỏa yêu cầu.

\* Worst fit:

Tiến trình 212K sẽ được cấp phát vùng nhớ 600K.

Tiến trình 417K sẽ được cấp phát vùng nhớ 500K.

Tiến trình 112K sẽ được cấp phát vùng nhớ 300K.

Tiến trình 426K sẽ sẽ phải chờ vì không còn vùng nhớ trống thỏa yêu cầu.

b. Xét trường hợp bộ nhớ được phân vùng động:

\*First fit:

Tiến trình 212K sẽ được cấp phát vùng nhớ 600K => Vùng nhớ trống 388K.

Tiến trình 417K sẽ được cấp phát vùng nhớ 500K.

Tiến trình 112K sẽ được cấp phát vùng nhớ 388K.

Tiến trình 426K sẽ phải chờ vì không còn vùng nhớ trống thỏa yêu cầu.

\* Best fit:

Tiến trình 212K sẽ được cấp phát vùng nhớ 300K.

Tiến trình 417K sẽ được cấp phát vùng nhớ 500K.

Tiến trình 112K sẽ được cấp phát vùng nhớ 200K.

Tiến trình 426K sẽ được cấp phát vùng nhớ 600K.

\* Next fit:

Tiến trình 212K sẽ được cấp phát vùng nhớ 600K.

Tiến trình 417K sẽ được cấp phát vùng nhớ 500K.

Tiến trình 112K sẽ được cấp phát vùng nhớ 200K.

Tiến trình 426K sẽ sẽ phải chờ vì không còn vùng nhớ trống thỏa yêu cầu.

\* Worst fit:

Tiến trình 212K sẽ được cấp phát vùng nhớ 600K. => Vùng nhớ trống 388K.

Tiến trình 417K sẽ được cấp phát vùng nhớ 500K.

Tiến trình 112K sẽ được cấp phát vùng nhớ 388K.

Tiến trình 426K sẽ sẽ phải chờ vì không còn vùng nhớ trống thỏa yêu cầu.

**Câu 11: (Bài tập mẫu) Xét một không gian địa chỉ có 12 trang, mỗi trang có kích thước 2K, ánh xạ vào bộ nhớ vật lý có 32 khung trang.**

a. Địa chỉ logic gồm bao nhiêu bit?

b. Địa chỉ physic gồm bao nhiêu bit?

**Trả lời:**

a. Địa chỉ logic gồm 2 phần: chỉ số trang và độ dời (offset) trong trang. Cần 4 bit để biểu diễn chỉ số trang và 11 bit (211 = 2K = 2048) để biểu diễn độ dời trong trang. Vậy địa chỉ logic gồm 15 bit.

b. Thực hiện tương tự. Địa chỉ physic gồm 16 bit.

**Câu 12: (Bài tập mẫu) Xét một hệ thống sử dụng kỹ thuật phân trang, với bảng trang được lưu trữ trong bộ nhớ chính.**

a. Nếu thời gian cho một lần truy xuất bộ nhớ bình thường (memory reference) là 200ns thì mất bao nhiêu thời gian cho một thao tác truy xuất bộ nhớ (paged memory reference) trong hệ thống này?

b. Nếu sử dụng TLBs với hit-ratio là 75%, thời gian để tìm trong TLBs xem như bằng 0, tính thời gian truy xuất bộ nhớ (effective access time) trong hệ thống?

**Trả lời:**

1. Mỗi thao tác truy xuất bộ nhớ trong hệ thống này sẽ cần thực hiện 2 lần truy xuất bộ nhớ thông thường: truy xuất bảng trang (để xác định vị trí khung trang) và truy xuất vị trí bộ nhớ (xác định dựa trên sự kết hợp giá trị khung trang tìm được ở lần trước với độ dời trong trang). Do đó thời gian của một thao tác truy xuất bộ nhớ sẽ là 200\*2 = 400ns.
2. Thời gian truy xuất bộ nhớ (effective access time) trong hệ thống:   
   EAT = (2 – α)x + ε = 250 ns.

**Câu 13: Một máy tính 32-bit địa chỉ, sử dụng một bảng trang 2 cấp. Địa chỉ ảo được phân bổ như sau: 9 bit dành cho bảng trang cấp 1, 11 bit cho bảng trang cấp 2 và còn lại cho offset. Cho biết kích thước một trang trong hệ thống và địa chỉ ảo có bao nhiêu trang?**

Địa chỉ ảo = (p1, p2, d) = 32 bit

p1=9, p2=11=> d=12 bit => kích thước 1 trang = 212B=22x210B=4KB

Số trang trong địa chỉ ảo 29x210= 220 trang

**Câu 14: Giả sử địa chỉ ảo 32-bit được phân tách thành 4 trường a, b, c, d. 3 trường đầu tiên được dùng cho bảng trang 3 cấp, trường thứ 4 dành cho offset. Số lượng trang có phụ thuộc vào kích thước của cả 4 trường này không? Nếu không, những trường nào ảnh hưởng đến số lượng trang, những trường nào không ảnh hưởng?**

**Trả lời:**

Số lượng trang chỉ phụ thuộc vào kích thước trường d và bằng: 2(32 – d)

Các trường a, b, c có thể thay đổi kích thước nhưng vẫn không làm thay đổi số trang nếu trường d không thay đổi.

|  |  |
| --- | --- |
| 0 | 6 |
| 1 | 4 |
| 2 | 5 |
| 3 | 7 |
| 4 | 1 |
| 5 | 9 |

**Câu 15: (Bài tập mẫu) Cho bảng trang như hình bên.**

a. Địa chỉ vật lý 6568 sẽ được chuyển thành địa chỉ ảo bao nhiêu? Biết rằng kích thước mỗi frame là 1K bytes.

b. Địa chỉ ảo 3254 sẽ được chuyển thành địa chỉ vật lý bao nhiêu? Biết rằng kích thước mỗi frame là 2K bytes.

**Trả lời:**

a. Địa chỉ vật lý 6568 nằm ở khung trang 6 với độ dời 424. Trang 0 được nạp vào khung trang 6 => Địa chỉ ảo là 424.

b. Địa chỉ 3254 nằm ở trang 1 với độ dời 1206. Trang 1 được nạp vào khung trang 4 => Địa chỉ vật lý là 9398.

**Câu 16: Xét một hệ thống sử dụng kỹ thuật phân trang, với bảng trang được lưu trữ trong bộ nhớ chính.**

a. Nếu thời gian cho một lần truy xuất bộ nhớ bình thường là 124 nanoseconds, thì mất bao nhiêu thời gian cho một thao tác truy xuất bộ nhớ trong hệ thống này?

b. Nếu sử dụng TLBs với hit-ratio (tỉ lệ tìm thấy) là 95%, thời gian để tìm trong TLBs bằng 34ns, tính thời gian cho một thao tác truy xuất bộ nhớ trong hệ thống (effective memory reference time).

**Trả lời:**

a. Để tìm 1 dữ liệu khi biết địa chỉ luận lý của nó, cần 2 thao tác truy xuất bộ nhớ:

- Thao tác 1: Truy xuất vào bảng phân trang để tìm được địa chỉ frame vật lý.

- Thao tác 2: Khi có được frame, biết địa chỉ vật lý, truy xuất vào bộ nhớ vật lý lấy dữ liệu

Đề cho 1 lần truy xuất tốn 124ns => Phải mất 248ns cho 1 thao tác truy xuất bộ nhớ.

b.Thời gian truy xuất TLB = 34ns

Thời gian tìm thấy dữ liệu trong bộ nhớ khi TLB hit = 34ns + 124ns = 158ns

(thời gian truy xuất TLB + thời gian truy xuất vào bộ nhớ vật lý)

Thời gian tìm thấy dữ liệu trong bộ nhớ khi TLB miss = 34ns +124ns +124ns = 282ns

(thời gian truy xuất TLB + thời gian truy xuất vào bảng phân trang+thời gian truy xuất vào bộ nhớ vật lý)

* Tổng thời gian truy xuất bộ nhớ trong hệ thống = 158ns\*0.95 + 282ns\*0.05 = 164.2ns

**Câu 17: Xét một hệ thống sử dụng kỹ thuật phân trang, với bảng trang được lưu trữ trong bộ nhớ chính. Nếu sử dụng TLBs với hit-ratio (tỉ lệ tìm thấy) là 87%, thời gian để tìm trong TLBs là 24 ns. Thời gian truy xuất bộ nhớ trong hệ thống (effective memory reference time) là 175ns. Tính thời gian cho một lần truy xuất bộ nhớ bình thường?**

Thời gian truy xuất hiệu dụng: EAT = (ε + x)α + (ε + 2x)(1 – α)

Trong đó:

* Thời gian tìm kiếm trong TLB (associative lookup): ε
* Thời gian một chu kỳ truy xuất bộ nhớ: x
* Hit ratio: tỉ số giữa số lần chỉ số trang được tìm thấy (hit) trong TLB và số lần truy xuất khởi nguồn từ CPU
  + Kí hiệu hit ratio: α
* 175 = 0.87(24 + x) + (1-0.87)(24 + 2x)
* x = 133.628 ns

Vậy thời gian cho một lần truy xuất bộ nhớ bình thường là 133.628 ns

**Câu 18: Biết thời gian truy xuất trong bộ nhớ thường không sử dụng TLBs là 250ns. Thời gian tìm kiếm trong bảng TLBs là 26ns. Hỏi xác suất tìm thấy trong TLBs bằng bao nhiêu nếu thời gian truy xuất trong bộ nhớ chính là 182ns?**

Thời gian truy xuất hiệu dụng: EAT = (ε + x)α + (ε + 2x)(1 – α)

Trong đó:

* Thời gian tìm kiếm trong TLB (associative lookup): ε
* Thời gian một chu kỳ truy xuất bộ nhớ: x
* Hit ratio: tỉ số giữa số lần chỉ số trang được tìm thấy (hit) trong TLB và số lần truy xuất khởi nguồn từ CPU
  + Kí hiệu hit ratio: α

(26+250)\*a% + (26+250\*2)\*(100- a)% = 182

* a = 75.2%

Vậy xác suất tìm thấy trong TLBs bằng bao nhiêu nếu thời gian truy xuất trong bộ nhớ chính là 182ns là 75.2%

**CÂU HỎI VÀ BÀI TẬP CHƯƠNG 8**

**Câu 1: Tại sao cần phải có bộ nhớ ảo?**

Hiện nay, như các bạn biết thì các chương trình ngày một phình to ra về kích thước. Vì vậy, việc load toàn bộ chương trình vào bộ nhớ chính để thực thi gần như là chuyện … bất khả thi.

Bộ nhớ ảo là một kỹ thuật cho phép chạy một process, mà process đó không cần thiết phải nằm hoàn toàn trong bộ nhớ.

Với bộ nhớ ảo, chúng ta sẽ có thể làm được những điều sau :

* Thực thi một chương trình có kích thước lớn hơn nhiều so với bộ nhớ chính.
* Lượng process trong bộ nhớ cũng nhiều hơn.
* Trừu tượng hoá bộ nhớ chính thành một mảng array lớn. Người dùng chỉ thấy được bộ nhớ thông qua bộ nhớ luận lý (logical memory). Từ đó giúp người lập trình viên đỡ cơ cực hơn.
* Bộ nhớ ảo cũng giúp cho các tiến trình chia sẻ dữ liệu với nhau dễ dàng, ngoài ra còn hỗ trợ trong việc cài đặt shared memory (bộ nhớ dùng chung).

Ví dụ thực tế : Phân vùng swap trong Linux, file pagefile.sys trong Windows.

**Câu 2: Có bao nhiêu kỹ thuật cài đặt bộ nhớ ảo? Mô tả sơ lược các kỹ thuật đó?**

Có hai kỹ thuật :

* Phân trang theo yêu cầu (Demand Paging) : Các trang dữ liệu sẽ không được sao chép vào bộ nhớ chính cho đến khi nó được cần tới.
* Phân đoạn theo yêu cầu (Segmentation Paging) : Các segment của chương trình sẽ không được chép vào bộ nhớ chính cho đến khi nó được cần tới.
* Phần cứng memory management phải hỗ trợ paging, segmentation.
* OS phải quản lý swap giữa bộ nhớ chính và bộ nhớ thứ cấp.

**Câu 3: Các bước thực hiện kỹ thuật phân trang theo yêu cầu?**

* Phân trang theo yêu cầu :
  + Các trang của quá trình chỉ được nạp vào bộ nhớ chính khi được yêu cầu.
  + Khi có một tham chiếu đến một trang mà không có trong bộ nhớ chính (valid bit) thì phần cứng sẽ gây ra một ngắt (gọi là pagefault – các bạn hay gặp từ này khi làm bài tập về các giải thuật phân trang), kích khởi pagefault service routine (PFSR) của hệ điều hành.
  + PFSR :
    - Chuyển process về trạng thái blocked.
    - Phát ra một yêu cầu đọc đĩa để nạp trang được tham chiếu vào một frame trống, trong khi đợi I/O, một process khác được cấp CPU để thực thi.
    - Sau khi I/O hoàn tất, đĩa gây ra một ngắt đến hệ điều hành, PFSR cập nhập page-table và chuyển process về trạng thái ready.

Diagram

Description automatically generated

Lỗi trang và các bước xử lý.

Bước 2 của PFSR : Giả sử phải thay trang vì không tìm được frame trống, PFSR được bổ sung như sau :

* Xác định vị trí trên đĩa của trang đang cần.
* Tìm một frame trống.
  + Nếu có frame trống thì dùng thôi chờ gì nữa.
  + Nếu không có frame trống thì dùng một giải thuật thay trang để chọn một trang hi sinh (victim page).
  + Ghi victim page lên đĩa, cập nhật page table và frame table tương ứng.
* Đọc trang đang cần vào frame trống (đã có được từ bước 2), cập nhật page table và frame table tương ứng.

Diagram

Description automatically generated

Hoạt động swap-in, swap-out.

Vậy phải xác định victim page như thế nào ? Đó chính là câu hỏi đã dẫn đến sự ra đời của các giải thuật thay trang như FIFO, LRU, OPT,…

**Câu 4: Mô tả các giải thuật thay thế trang FIFO, OPT, LRU?**

* **FIFO : First-In-First-Out.**

Như tên gọi của nó, trang nhớ nào được vào trước sẽ được “ưu tiên” swap ra ngoài khi có page-fault.

Ví dụ về FIFO :



Chuỗi yêu cầu.

A picture containing shoji

Description automatically generated

Kết quả chạy.

* **OPT : Optimal Page-Replacement Algorithm.**

Trang nhớ nào được tham chiếu trễ nhất trong tương lai sẽ được swap ra ngoài.

Ví dụ :

A picture containing shoji

Description automatically generated

Chuỗi yêu cầu.

A picture containing shoji

Description automatically generated

Kết quả chạy.

* **LRU : Least Recently Used Page-Replacement Algorithm.**

Mỗi trang được ghi nhận thời điểm được tham chiếu => Trang LRU là trang nhớ có thời điểm tham chiếu nhỏ nhất (OS tốn chi phí tìm kiếm trang LRU này mỗi khi có pagefault).

Do vậy, LRU cần sự hỗ trợ của phần cứng cho việc tìm kiếm.

(LRU giống OPT, nhưng tìm về quá khứ thay vì tương lai. Tức là, trang nhớ nào được tham chiếu ở thời điểm cách xa thời điểm hiện tại nhất sẽ được chọn để thay thế).

Ví dụ :



Chuỗi yêu cầu.

Table

Description automatically generated

Kết quả chạy.

**Câu 5: Giải pháp tập làm việc hoạt động như thế nào?**

* Được thiết kế dựa trên nguyên lý locality.
* Xác định xem process thực sự sử dụng bao nhiêu frame.
* Định nghĩa:
  + WS(t) - số lượng các tham chiếu trang nhớ của process gần đây nhất cần được quan sát.
  + 🛆 - khoảng thời gian tham chiếu
* Ví dụ:

2 4 5 6 9 1 3 2 6 3 9 2 1 4

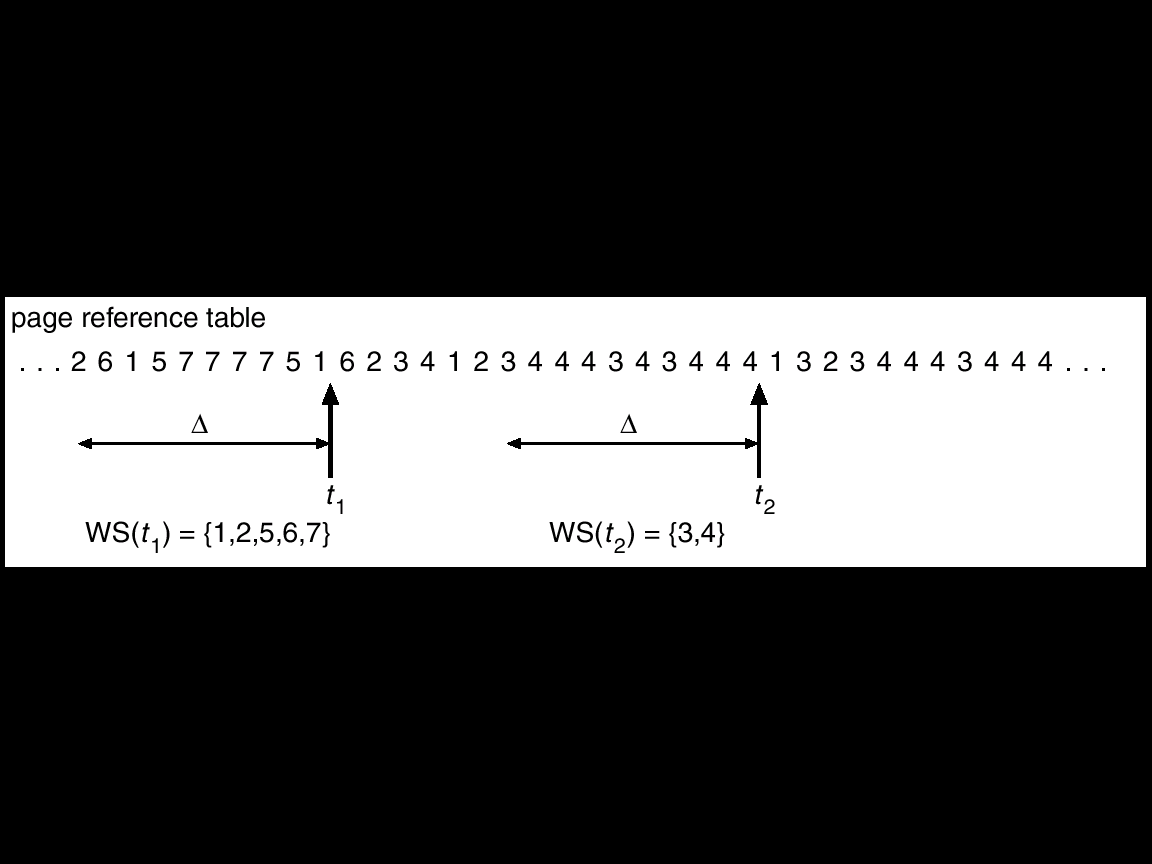
thời điểm t1

🛆 ⏺ ⏵

chuỗi tham khảo

trang nhớ

* Định nghĩa: Working set của process Pi , ký hiệu WSi, là tập gồm Δ các trang được sử dụng gần đây nhất.



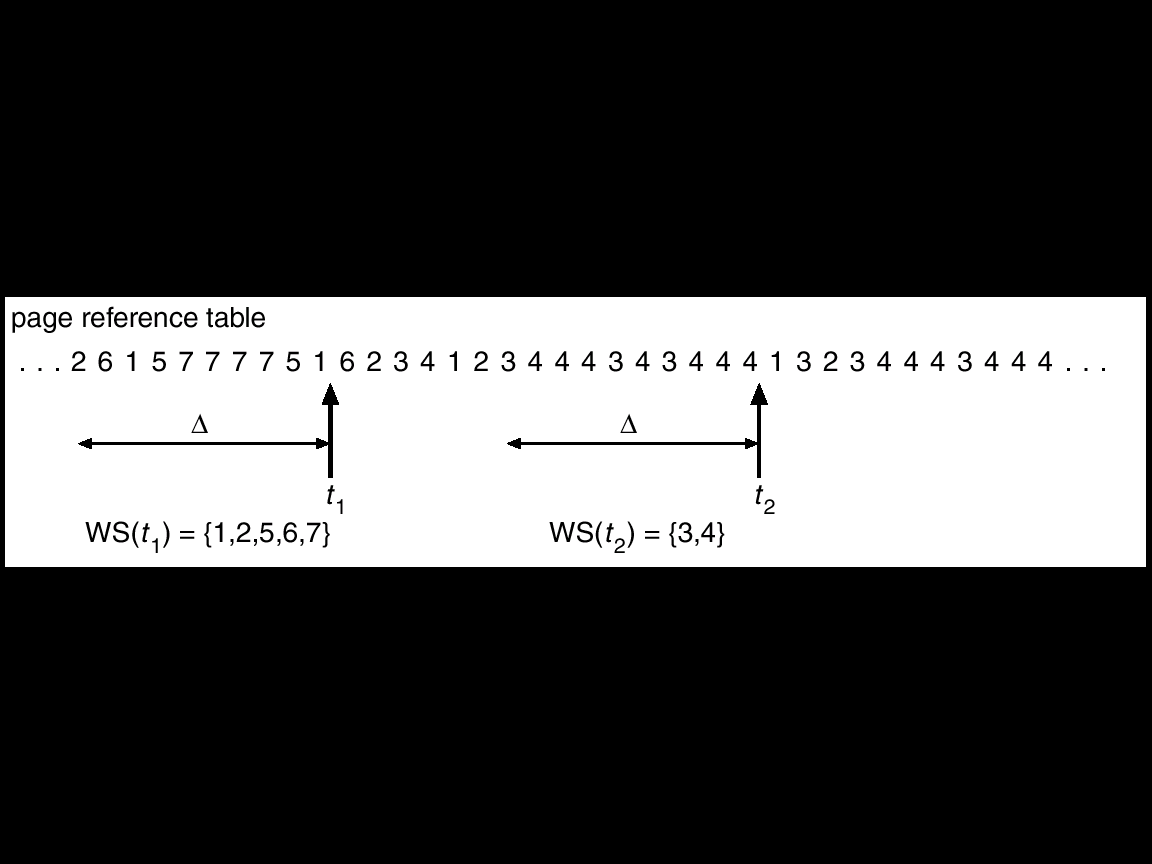
chuỗi tham khảo trang

Ví dụ: Δ = 10 và

* Nhận xét:
  + Δ quá nhỏ ⇒ không đủ bao phủ toàn bộ locality.
  + Δ quá lớn ⇒ bao phủ nhiều locality khác nhau.
  + Δ = ∞ ⇒ bao gồm tất cả các trang được sử dụng.
* Dùng working set của một process để xấp xỉ locality của nó.
* Định nghĩa: WSSi là kích thước của working set của Pi:
  + WSSi = số lượng các trang trong WSi

WSS(t2) = 2

WSS(t1) = 5



chuỗi tham khảo trang

Ví dụ: Δ = 10 và

* Đặt D = Σ WSSi = tổng các working-set size của mọi process trong hệ thống.
  + Nhận xét: Nếu D > m (số frame của hệ thống) ⇒ sẽ xảy ra thrashing.
* Giải pháp working set:
  + Khi khởi tạo một quá trình: cung cấp cho quá trình số lượng frame thỏa mản working-set size của nó.
  + Nếu D > m ⇒ tạm dừng một trong các process.
    - Các trang của quá trình được chuyển ra đĩa cứng và các frame của nó được thu hồi.
* WS loại trừ được tình trạng trì trệ mà vẫn đảm bảo mức độ đa chương

**Câu 6: (Bài tập mẫu) Xét chuỗi truy xuất bộ nhớ sau: 1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6 Có bao nhiêu lỗi trang xảy ra khi sử dụng các thuật toán thay thế sau đây, giả sử có 4 khung trang.**

a. LRU

b. FIFO

c. Chiến lược tối ưu (OPT)1

**Trả lời:**

a. LRU: Có 10 lỗi trang

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 2 | 1 | 5 | 6 | 2 | 1 | 2 | 3 | 7 | 6 | 3 | 2 | 1 | 2 | 3 | 6 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
|  | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
|  |  | 3 | 3 | 3 | 3 | 5 | 5 | 5 | 5 | 5 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
|  |  |  | 4 | 4 | 4 | 4 | 6 | 6 | 6 | 6 | 6 | 7 | 7 | 7 | 7 | 1 | 1 | 1 | 1 |
| \* | \* | \* | \* |  |  | \* | \* |  |  |  | \* | \* | \* |  |  | \* |  |  |  |

b. FIFO: Có 14 lỗi trang

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 2 | 1 | 5 | 6 | 2 | 1 | 2 | 3 | 7 | 6 | 3 | 2 | 1 | 2 | 3 | 6 |
| 1 | 1 | 1 | 1 | 1 | 1 | 5 | 5 | 5 | 5 | 5 | 3 | 3 | 3 | 3 | 3 | 1 | 1 | 1 | 1 |
|  | 2 | 2 | 2 | 2 | 2 | 2 | 6 | 6 | 6 | 6 | 6 | 7 | 7 | 7 | 7 | 7 | 7 | 3 | 3 |
|  |  | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
|  |  |  | 4 | 4 | 4 | 4 | 4 | 4 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 |
| \* | \* | \* | \* |  |  | \* | \* | \* | \* |  | \* | \* | \* |  | \* | \* |  | \* |  |

c. OPT: Có 8 lỗi trang

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 2 | 1 | 5 | 6 | 2 | 1 | 2 | 3 | 7 | 6 | 3 | 2 | 1 | 2 | 3 | 6 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 7 | 7 | 7 | 7 | 1 | 1 | 1 | 1 |
|  | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
|  |  | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
|  |  |  | 4 | 4 | 4 | 5 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| \* | \* | \* | \* |  |  | \* | \* |  |  |  |  | \* |  |  |  | \* |  |  |  |