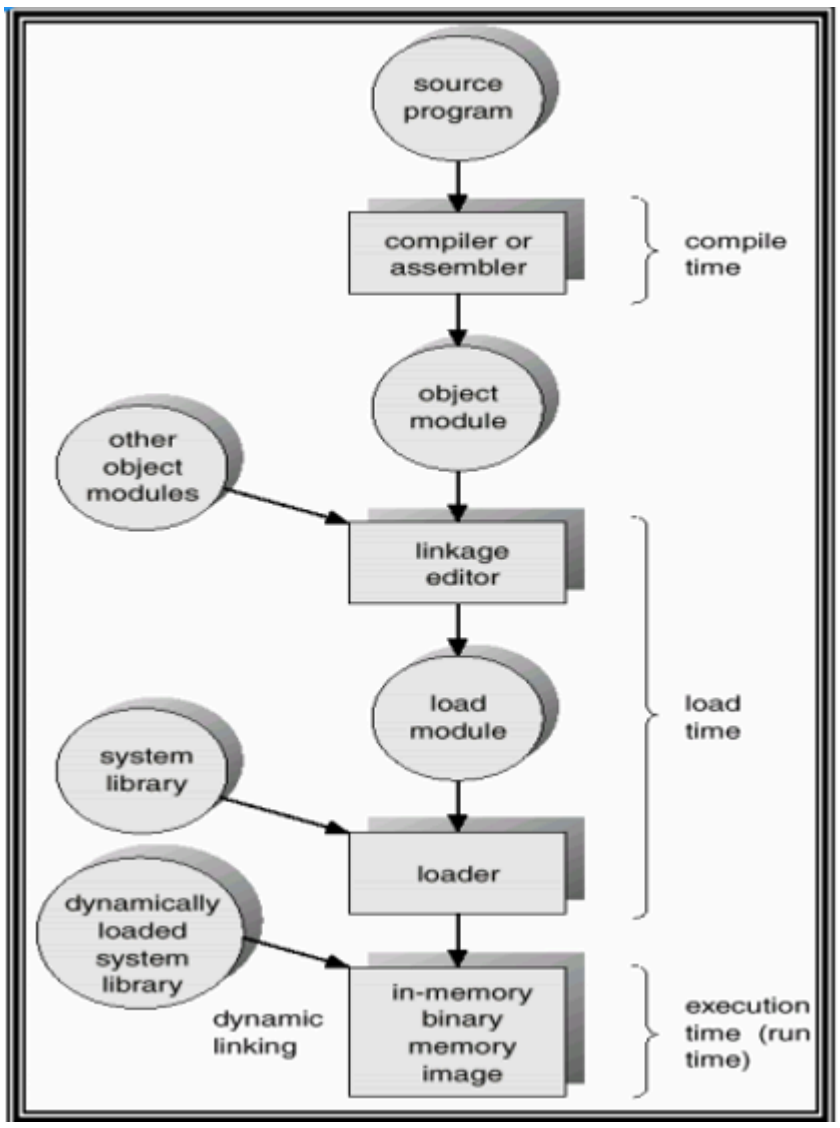
Cơ sở

- Chương trình muốn thực thi cần phải được tải vào bộ nhớ và đặt trong một tiến trình
- Hàng đợi vào (Input Queue)
+ Tập các tiến trình trên đĩa, đang đợi tải vào bộ nhớ để thực hiện
- Các chương trình người dùng muốn được thực thi cần phải qua một số bước trong đó có bước gán địa chỉ cho các câu lệnh/ dữ liệu.

Gán bộ nhớ cho các câu lệnh và dữ liệu

Việc gán địa chỉ cho các câu lệnh và dữ liệu được thực thi tại các thời điểm

- Biên dịch – nếu vị trí trong bộ nhớ đã được biết trước – sinh ra mã tuyệt đối (absolute code); cần phải được biên dịch lại nếu vị trí bắt đầu bị thay đổi
- Lúc tải (loading time) – phải sinh ra mã có thể định vị lại (relocatable code) – nếu vị trí trong bộ nhớ không được biết trước
  - Mã có thể định vị lại "14 bytes kể từ đầu module"
- Lúc thực thi–Gán địa chỉ được trì hoãn cho đến khi thực thi nếu tiến trình có thể thay đổi, từ đoạn bộ nhớ này đến đoạn bộ nhớ khác trong khi thực thi.
  - Yêu cầu phần cứng hỗ trợ cho các ánh xạ địa chỉ (thanh ghi cơ sở, thanh ghi giới hạn)



Không gian địa chỉ vật lý và không gian địa chỉ logic

- Khái niệm không gian địa chỉ logic gắn với không gian địa chỉ vật lý là trung tâm của các kĩ thuật quản lý bộ nhớ
+ Các địa chỉ logic –được sinh ra bởi CPU; còn được gọi là địa chỉ ảo
+ Địa chỉ vật lý – địa chỉ thật trong bộ nhớ, thấy được bởi đơn vị quản lý bộ nhớ
- Như nhau trong lược đồ gán địa chỉ lúc biên dịch, tải
- Khác nhau trong lược đồ gán địa chỉ lúc thực thi

Đơn vị quản lý bộ nhớ (MMU)

- Đơn vị quản lý bộ nhớ (Memory Management Unit - MMU) là một thành phần phần cứng trong hệ thống máy tính quản lý quyền truy cập bộ nhớ giữa bộ vi xử lý trung tâm (CPU) và bộ nhớ (memory). Chức năng chính của nó là chuyển đổi địa chỉ bộ nhớ ảo được sử dụng bởi CPU thành các địa chỉ bộ nhớ vật lý được sử dụng bởi bộ nhớ.

- MMU chịu trách nhiệm thực hiện bảo vệ bộ nhớ, đảm bảo một quy trình không thể truy cập vào bộ nhớ mà nó không được ủy quyền truy cập. MMU cũng quản lý bộ nhớ ảo, đó là một kỹ thuật quản lý bộ nhớ cho phép một quy trình sử dụng nhiều bộ nhớ hơn so với bộ nhớ vật lý có sẵn bằng cách tạm thời chuyển trang của bộ nhớ giữa bộ nhớ vật lý và ổ đĩa lưu trữ.

- MMU thực hiện ánh xạ bộ nhớ, gán địa chỉ bộ nhớ vật lý cho địa chỉ bộ nhớ ảo. Điều này cho phép CPU truy cập bộ nhớ mà không biết vị trí vật lý của nó. MMU cũng thực hiện phân đoạn bộ nhớ, chia bộ nhớ thành các phân đoạn logic để quản lý và bảo vệ dễ dàng hơn.

- Trong các hệ thống máy tính hiện đại, MMU thường được tích hợp vào CPU, mặc dù nó có thể là một vi mạch riêng biệt trong các hệ thống cũ hơn.

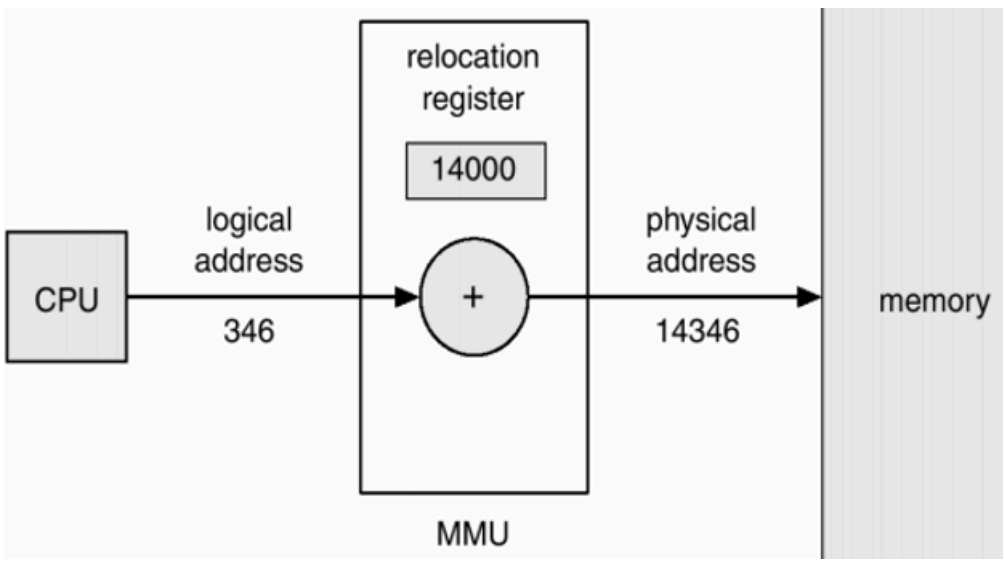
Cách hoạt động của MMU

- Giả sử 1 chương trình đang chạy trên máy tính có MMU. Chương trình cố gắng đọc dữ liệu từ địa chỉ bộ nhớ 0x12345678. CPU gửi địa chỉ bộ nhớ ảo này đến MMU.
- MMU kiểm tra bảng trang (page table), đó là 1 cấu trúc dữ liệu mà ánh xạ địa chỉ bộ nhớ ảo đến địa chỉ bộ nhớ vật lý. Nó tìm thấy rằng trang chứa 0x12345678 hiện được lưu trữ trên đĩa (disk) và không có trong bộ nhớ vật lý.

- MMU gửi tín hiệu lỗi trang (page fault) đến hệ điều hành, hệ điều hành phản hồi bằng cách tải trang yêu cầu từ đĩa vào bộ nhớ vật lý. MMU cập nhật bảng trang (page table) để phản ánh địa chỉ bộ nhớ vật lý mới của trang được yêu cầu.

- MMU gửi địa chỉ bộ nhớ vật lý của dữ liệu được yêu cầu trở lại cho CPU, CPU đọc dữ liệu từ bộ nhớ và tiếp tục thực hiện chương trình.

- Trong suốt quá trình này, MMU đảm bảo rằng chương trình chỉ có quyền truy cập vào bộ nhớ mà nó được ủy quyền truy cập, và các chương trình khác không thể can thiệp vào bộ nhớ của nhau. Điều này giúp đảm bảo tính ổn định và an ninh của hệ thống máy tính.



Dynamically loading into memory

- Việc tải vào bộ nhớ động là quá trình tải 1 chương trình hoặc 1 thành phần phần mềm vào bộ nhớ trong thời gian chạy (run time), chứ không phải trong quá trình biên dịch (compiler time). Kỹ thuật này được sử dụng để giảm yêu cầu bộ nhớ của 1 ứng dụng và cải thiện hiệu suất của nó. Khi một chương trình được tải động vào bộ nhớ, chỉ các phần cần thiết của chương trình được tải, thay vì tải toàn bộ chương trình, điều này có thể tiết kiệm bộ nhớ.

- Kỹ thuật này thường được sử dụng trong các hệ điều hành và các khung phần mềm để cải thiện hiệu suất và tính linh hoạt. Bằng cách tải động các mô-đun hoặc thư viện vào bộ nhớ khi chúng cần thiết, hệ thống có thể giảm yêu cầu bộ nhớ và cải thiện khả năng phản hồi đối với các yêu cầu của người dùng.

- Tóm lại, tải vào bộ nhớ động là 1 kỹ thuật được sử dụng để tải các thành phần phần mềm vào bộ nhớ trong thời gian chạy, giúp cải thiện hiệu suất, giảm yêu cầu bộ nhớ và tăng tính linh hoạt.

Dynamic linking

- Là 1 kỹ thuật được dùng trong lập trình máy tính, trong đó 1 chương trình được liên kết với các thư viện hoặc các module bên ngoài tại thời điểm chạy chương trình thay vì tại thời điểm biên dịch. Chương trình không bao gồm (include) tất cả mã nguồn cần thiết khi được biên dịch, mà chỉ bao gồm các tham chiếu đến các thư viện bên ngoài.

- Lợi ích:
+ Cho phép chương trình sử dụng các thư viện và tài nguyên mà không cần phải liên kết tĩnh trước khi biên dịch → Giảm kích thước của chương trình và cho phép chia sẻ mã nguồn trên nhiều chương trình
+ Đơn giản hóa quá trình cập nhật các thư viện bên ngoài

- Nhược điểm:
+ Nguy cơ không tương thích giữa các phiên bản khác nhau của cùng 1 thư viện
+ Các rủi ro về an ninh liên quan đến việc tải mã nguồn bên ngoài vào bộ nhớ tại thời điểm chạy chương trình.

- Quá trình liên kết động được thực hiện theo các bước sau đây:
+ Tạo một file thực thi chứa mã máy và các thông tin khác như bảng liên kết động (dynamic linking table).
+ Khi chương trình được chạy, hệ điều hành đọc file thực thi vào bộ nhớ và bắt đầu thực thi chương trình.
+ Khi một hàm từ một thư viện được gọi, trình liên kết động sẽ tìm kiếm thư viện đó trong bộ nhớ hoặc tải thư viện từ ổ đĩa vào bộ nhớ nếu thư viện chưa được tải.
+ Các hàm và tài nguyên trong thư viện được liên kết đến chương trình và có thể được sử dụng.
+ Khi chương trình kết thúc, hệ điều hành giải phóng các tài nguyên đã được sử dụng và thu hồi bộ nhớ được sử dụng bởi chương trình.

Overlay

- Là 1 kỹ thuật trong lập trình máy tính cho phép 1 chương trình vượt qua giới hạn bộ nhớ vật lý của hệ thống và thực thi các chương trình lớn hơn bộ nhớ RAM. Kỹ thuật này giúp tối ưu hóa việc sử dụng bộ nhớ và giảm chi phí bộ nhớ của các chương trình lớn

- Cơ chế:
+ Chia chương trình thành những phần nhỏ hơn gọi là "overlay"
+ Chỉ tải lên bộ nhớ vật lý các phần overlay được sử dụng đến, thay thế cho việc tải toàn bộ chương trình lên bộ nhớ.
+ Khi cần sử dụng một phần overlay khác, các phần overlay trước đó sẽ được lưu trữ lại trên bộ nhớ thứ cấp (như ổ đĩa) và phần overlay mới sẽ được tải lên bộ nhớ vật lý.

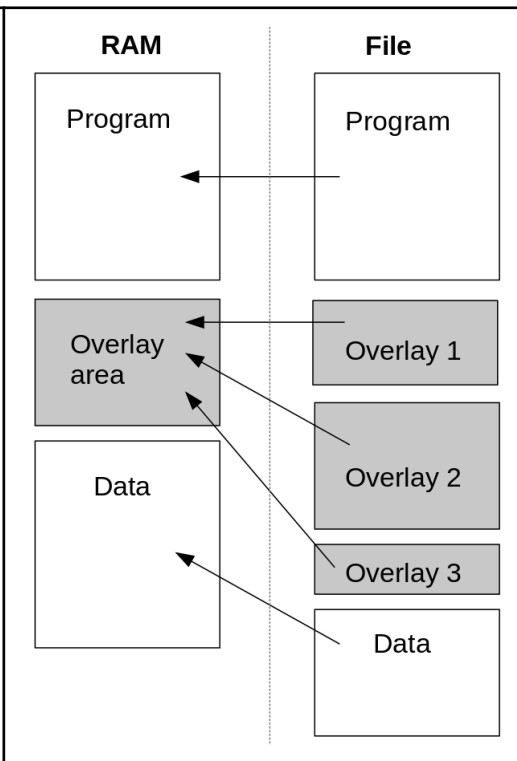
- Ưu điểm:
+ Tăng hiệu quả sử dụng và tiết kiệm bộ nhớ:

+ Giảm thời gian chuyển đổi giữa các phân đoạn: Khi sử dụng overlay, các phân đoạn được nạp lên và xóa khỏi bộ nhớ tùy thuộc vào nhu cầu sử dụng. Việc này giúp giảm thời gian chuyển đổi giữa các phân đoạn.

- Nhược điểm:
+ Tăng phức tạp cho việc phát triển phần mềm: Các phần mềm phải được thiết kế để hỗ trợ việc sử dụng overlay.

+ Độ trễ: Khi sử dụng overlay, các phân đoạn liên tục được tải lên bộ nhớ thứ cấp và giải phóng từ RAM, dẫn đến tăng độ trễ trong quá trình thực thi chương trình.

+ Hạn chế về dung lượng: Overlay giới hạn dung lượng của phần chương trình được tải lên và sử dụng tại 1 thời điểm. Việc này



có thể giới hạn khả năng mở rộng của chương trình

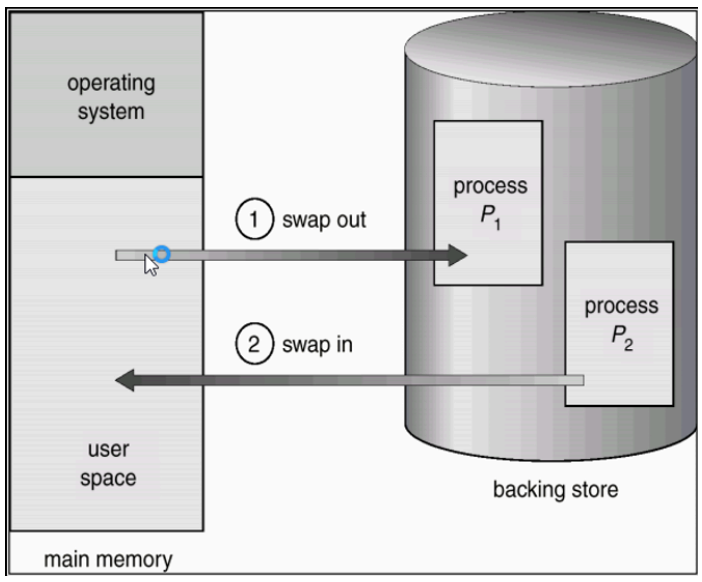
Swapping

.

- Là 1 kỹ thuật trong hệ thống máy tính cho phép dịch chuyển các phần tử từ bộ nhớ trung gian (RAM) sang bộ nhớ ngoài (ổ cứng hoặc thiết bị lưu trữ khác) và ngược lại, để tăng sức chứa của bộ nhớ trung gian. Khi không đủ bộ nhớ để chứa tất cả các chương trình và dữ liệu mà hệ thống đang sử dụng, swapping giúp di chuyển các phần tử không sử dụng ra bộ nhớ ngoài để giải phóng bộ nhớ trung gian. Khi cần sử dụng lại các phần tử đó, hệ thống sẽ di chuyển chúng trở lại bộ nhớ trung gian và giải phóng bộ nhớ không cần thiết khác nếu có.

- Kỹ thuật swapping thường được sử dụng trong hệ điều hành để quản lý bộ nhớ, đảm bảo tối ưu hóa sử dụng tài nguyên và đáp ứng được yêu cầu của các chương trình khi bộ nhớ trung gian không đủ

- Cơ chế:
+ Khi 1 chương trình cần sử dụng bộ nhớ lớn hơn bộ nhớ thực tế có sẵn, hệ điều hành sẽ chuyển các trang (pages) không sử dụng của chương trình đó từ bộ nhớ RAM sang ổ cứng, giải phóng bộ nhớ RAM. Khi chương trình yêu cầu truy cập đến các trang đã bị chuyển đi, hệ điều hành sẽ đọc lại các trang đó từ ổ cứng và đưa trở lại bộ nhớ RAM.



Ưu điểm	Nhược điểm
+ Cho phép các tiến trình có thể lớn hơn dung lượng bộ nhớ vật lý có sẵn, giúp tăng hiệu suất hệ thống. + Các tiến trình được tự do di chuyển giữa bộ nhớ vật lý và bộ nhớ trên đĩa, giúp tối ưu hóa việc sử dụng bộ nhớ. + Cho phép chương trình được chia thành các phần nhỏ hơn, giúp tối ưu hóa việc quản lý bộ nhớ. + Cho phép chương trình được thiết kế một cách linh hoạt hơn, vì chúng không cần phải nằm trong bộ nhớ liên tục.	+ Chi phí tăng khi phải đọc/ghi dữ liệu giữa bộ nhớ vật lý và đĩa cứng. + Khi có nhiều tiến trình đang sử dụng bộ nhớ, hoạt động swapping có thể làm giảm hiệu suất hệ thống do đợi đến lượt di chuyển dữ liệu. + Trong trường hợp bộ nhớ không đủ lớn để chứa các tiến trình cần thiết, kỹ thuật swapping có thể gây ra tình trạng thrashing, khiến hệ thống hoạt động không ổn định.

Phân phối bộ nhớ liên tục (Contiguous memory allocation)

- Là 1 kỹ thuật quản lý bộ nhớ được dùng bởi HĐH để phân bổ và quản lý bộ nhớ trong 1 khối hoặc phạm vi liên tục của các địa chỉ bộ nhớ vật lý.
- Trong kỹ thuật này, 1 tiến trình được phân bổ 1 khối bộ nhớ liên tục và không bị gián đoạn. Kỹ thuật này được dùng bởi hầu hết các HĐH hiện đại để phân bổ bộ nhớ vật lý cho các tiến trình, vì nó cho phép quản lý bộ nhớ hiệu quả và truy cập nhanh hơn đến các vị trí bộ nhớ.

- Cơ chế:
+ Phân chia vùng nhớ vật lý thành các phân vùng liền nhau, mỗi phân vùng này sẽ chứa các trang bộ nhớ liên tiếp được cấp phát cho các tiến trình. Trong phân vùng, các trang bộ nhớ được gán cho các tiến trình theo cách liên tục và không bị gián đoạn. Khi tiến trình cần thêm bộ nhớ, hệ thống sẽ tìm kiếm một phân vùng trống phù hợp và cấp phát các trang bộ nhớ liên tục vào phân vùng đó.

- Các phân vùng này được quản lý bằng bảng phân vùng, mỗi phân vùng sẽ được đánh dấu là đang sử dụng hoặc trống. Khi một tiến trình kết thúc hoặc giải phóng bộ nhớ, các trang bộ nhớ của nó sẽ được trả lại cho hệ thống và đánh dấu phân vùng đó là trống để có thể sử dụng cho các tiến trình khác.

- Cơ chế này có thể được thực hiện thông qua việc sử dụng các thuật toán phân vùng như First-Fit (tìm vùng trống đầu tiên phù hợp), Best-Fit (tìm vùng trống tốt nhất) hoặc Worst-Fit (tìm vùng trống lớn nhất phù hợp - để lại nhiều ô trống không dùng đến) để tìm kiếm phân vùng phù hợp và cấp phát bộ nhớ cho tiến trình.

- Ưu điểm:
+ Tăng hiệu suất: Do bộ nhớ được phân phối liên tục và không bị gián đoạn, quá trình truy cập bộ nhớ của chương trình trở nên nhanh chóng và hiệu quả hơn, do không cần phải tìm kiếm và truy cập các đoạn bộ nhớ khác nhau.

+ Dễ dàng quản lý: Vì các đoạn bộ nhớ được phân phối liên tục, việc quản lý và theo dõi các vùng bộ nhớ dễ dàng hơn. Nó cũng giúp hệ điều hành dễ dàng quản lý các trang bộ nhớ và chia sẻ bộ nhớ giữa các tiến trình.

- Nhược điểm:
+ Giới hạn kích thước: Phương pháp này có giới hạn về kích thước bộ nhớ tối đa mà nó có thể quản lý. Vì vậy, nó không thể quản lý bộ nhớ lớn hơn kích thước của khối bộ nhớ liên tục lớn nhất có sẵn.

+ Fragmentation: Khi 1 tiến trình được xóa khỏi bộ nhớ, khoảng trống sẽ được tạo ra trong khối bộ nhớ liên tục. Những khoảng trống này có thể dẫn đến hiện tượng "fragmentation", làm cho bộ nhớ liên tục không thể được sử dụng hiệu quả.

Fragmentation

- Fragmentation trong kỹ thuật phân phối bộ nhớ liên tục là tình trạng khi không gian bộ nhớ bị chia thành nhiều khối nhỏ hoặc lỗ trống, làm cho khó để tìm một khối bộ nhớ liên tục đủ lớn để đáp ứng yêu cầu bộ nhớ của 1 tiến trình. Có hai loại fragmentation: internal fragmentation và external fragmentation:

+ Internal fragmentation: 1 tiến trình được phân bổ 1 khối bộ nhớ lớn hơn bộ nhớ được yêu cầu. Bộ nhớ không sử dụng trong khối bị lãng phí, dẫn đến không hiệu quả trong sử dụng bộ nhớ.
+ External fragmentation: nhiều khối bộ nhớ nhỏ không sử dụng được rải rác trong không gian bộ nhớ, làm cho không thể phân bổ 1 khối bộ nhớ liên tục để đáp ứng yêu cầu bộ nhớ của 1 tiến trình, mặc dù tổng lượng bộ nhớ đang trống là đủ.

- Kỹ thuật phân phối bộ nhớ liên tục có thể gặp phải cả 2 loại fragmentation: internal và external. Internal fragmentation thường xảy ra khi sử dụng phân phối bộ nhớ cố định kích thước, trong khi external fragmentation thường xảy ra khi sử dụng phân phối bộ nhớ có kích thước biến. Để giảm thiểu fragmentation, đã được phát triển các kỹ thuật phân phối bộ nhớ khác như phân trang (paging) và phân đoạn (segmentation)

Thanh ghi relocation và thanh ghi limit

- Trong kỹ thuật phân phối bộ nhớ liên tục, thanh ghi relocation và thanh ghi limit được dùng để xác định vị trí bắt đầu và độ dài của khối bộ nhớ được phân bổ cho mỗi tiến trình:
+ Thanh ghi relocation chứa địa chỉ bắt đầu của khối bộ nhớ được phân bổ cho tiến trình. Để truy cập đến bất kỳ ô nhớ nào trong khối bộ nhớ đó, địa chỉ được sử dụng phải được cộng với giá trị của thanh ghi relocation.

+ Thanh ghi limit xác định độ dài của khối bộ nhớ được phân bổ. Trong quá trình thực thi, nếu một tiến trình cố gắng truy cập vào một ô nhớ nằm ngoài khối bộ nhớ của nó, nó sẽ bị kết thúc bởi hệ điều hành và tránh xảy ra lỗi không mong muốn.

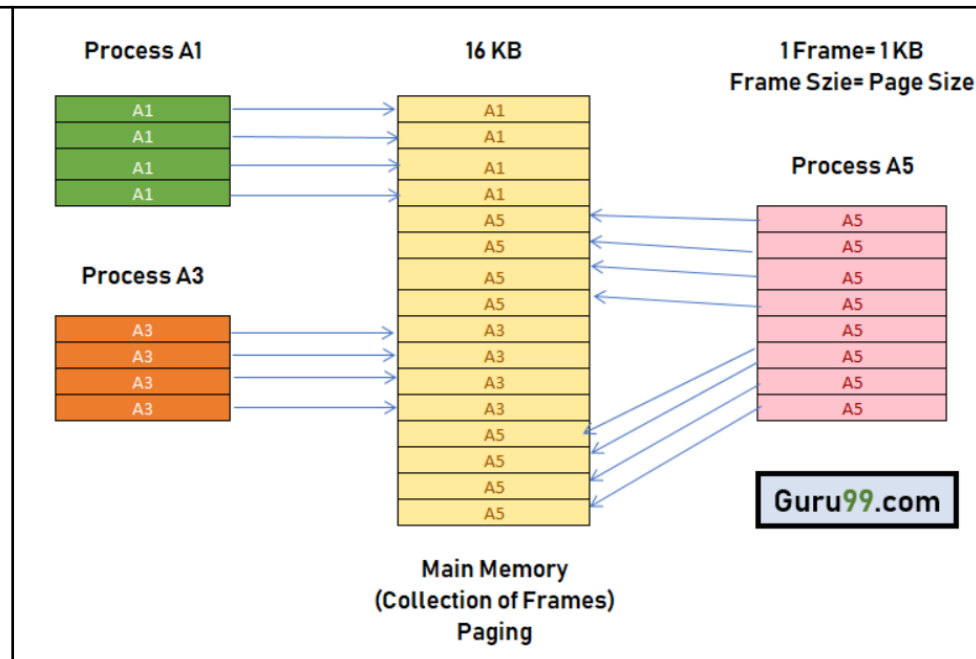
Việc sử dụng thanh ghi relocation và thanh ghi limit giúp đảm bảo rằng mỗi tiến trình chỉ truy cập vào vùng nhớ được cấp phép và không gian bộ nhớ được sử dụng hiệu quả hơn. Tuy nhiên, một nhược điểm của phương pháp này là việc quản lý không gian bộ nhớ rất khó khăn khi có nhiều tiến trình cùng chạy trong hệ thống.

Phân trang (paging)

- Paging là 1 kỹ thuật quản lý bộ nhớ được dùng bởi các HĐH để xử lý bộ nhớ ảo. Nó chia bộ nhớ của chương trình thành các đơn vị nhỏ có kích thước cố định được gọi là trang (page) và lưu trữ chúng trong các vị trí bộ nhớ không liên tục.
- Khi 1 tiến trình cần truy cập vào 1 trang cụ thể, HĐH lấy trang đó từ bộ nhớ và tải nó vào 1 khối bộ nhớ vật lý liên tục. Paging cho phép các tiến trình truy cập vào bộ nhớ lớn hơn so với bộ nhớ vật lý có sẵn, và nó cũng giúp giảm thiểu hiện tượng fragmentation bằng cách cho phép HĐH phân bổ bộ nhớ thành các mảnh nhỏ hơn và dễ quản lý hơn.

- Cơ chế:
+ Chia không gian địa chỉ logic của 1 tiến trình thành các trang (pages) cố định kích thước và phân mảnh chúng thành các khối nhỏ hơn gọi là khung trang (page frames) trong không gian địa chỉ vật lý. Mỗi trang được gán một số thứ tự riêng gọi là số trang, và mỗi khung trang được gán 1 địa chỉ vật lý riêng. Khi 1 tiến trình cần truy cập đến 1 trang nào đó, HĐH tìm kiếm khung trang trống và nạp trang đó vào khung trang đó. Khi không có khung trang trống, hệ điều hành thực hiện thuật toán thay thế trang (page replacement algorithm) để giải phóng 1 khung trang và nạp trang mới vào đó.

+ Quá trình này cho phép các tiến trình truy cập đến 1 không gian địa chỉ logic lớn hơn so với không gian địa chỉ vật lý có sẵn trên hệ thống, giúp tăng hiệu quả sử dụng bộ nhớ. Ngoài ra, paging cũng giúp giảm hiện tượng fragmentation bằng cách phân bố bộ nhớ theo các trang cố định, từ đó giúp quản lý bộ nhớ trở nên dễ dàng hơn.



- Ưu điểm:
+ Cho phép các quá trình truy cập đến bộ nhớ lớn hơn so với bộ nhớ vật lý có sẵn trên hệ thống.
+ Giảm sự phân mảnh bộ nhớ bằng cách phân bổ bộ nhớ theo từng trang nhỏ hơn, dễ quản lý hơn.
+ Cho phép hệ điều hành thực hiện các chức năng bảo vệ bộ nhớ, giảm nguy cơ bị lỗi truy cập bộ nhớ không hợp lệ.
+ Cho phép các trang được chia sẻ giữa nhiều tiến trình, tạo ra khả năng chia sẻ dữ liệu giữa các tiến trình.

- Nhược điểm:
+ Yêu cầu thêm thời gian và tài nguyên để chuyển đổi địa chỉ ảo sang địa chỉ vật lý, ảnh hưởng đến hiệu suất hệ thống.
+ Khi sử dụng trang nhỏ hơn, các trang sẽ trở nên nhiều hơn, dẫn đến sự lãng phí không gian bộ nhớ và mất hiệu quả khi tìm kiếm.
+ Yêu cầu quản lý bộ nhớ phức tạp hơn để theo dõi các trang được phân bổ và giải phóng các trang không cần thiết.

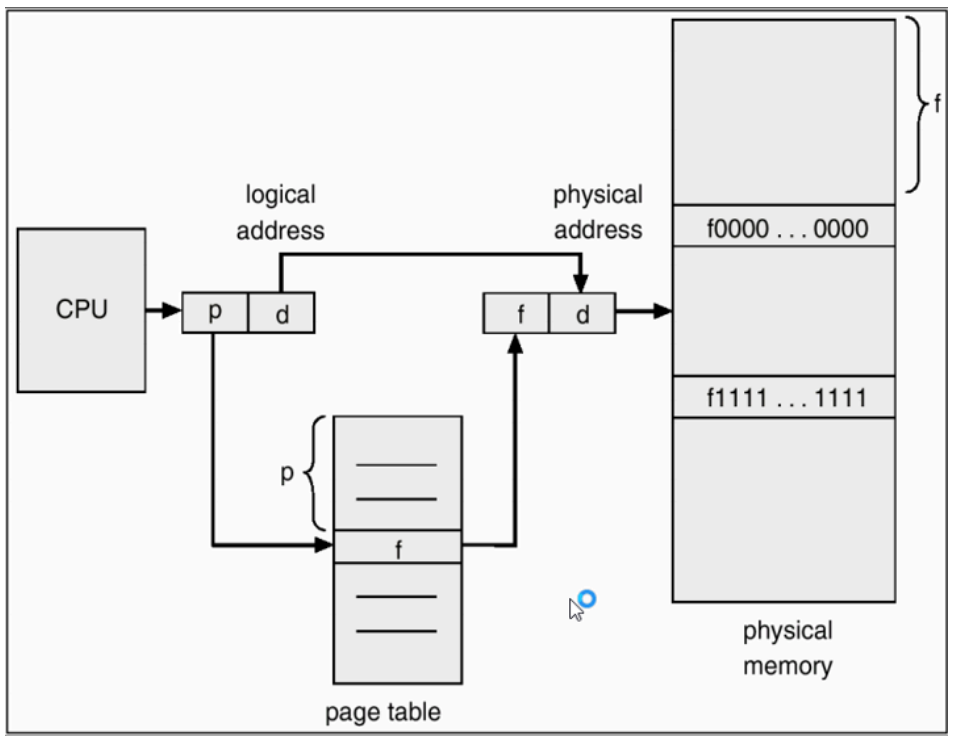
Lược đồ dịch địa chỉ (Address Translation)

- Là quá trình chuyển đổi địa chỉ bộ nhớ ảo (Virtual Memory Address) sang địa chỉ bộ nhớ vật lý (Physical Memory Address) trong hệ thống máy tính. Quá trình này được thực hiện bởi bộ điều khiển bộ nhớ (Memory Management Unit) của CPU dựa trên các bản đồ dịch địa chỉ (Address Translation Map) được lưu trữ trong bộ nhớ và quản lý bởi hệ điều hành.

- Ưu điểm:
+ Cho phép sử dụng bộ nhớ ảo lớn hơn kích thước bộ nhớ vật lý có sẵn trên hệ thống
+ Tăng hiệu suất của hệ thống và cho phép nhiều tiến trình chạy đồng thời mà không xảy ra xung đột bộ nhớ.
+ Giúp giảm thiểu sự phân mảnh bộ nhớ (fragmentation) và cho phép tách bộ nhớ vật lý thành các khối nhỏ hơn để sử dụng hiệu quả hơn.

- Nhược điểm:
+ Làm tăng thời gian truy cập bộ nhớ
+ Làm giảm hiệu suất của hệ thống.
+ Việc sử dụng bộ nhớ ảo có thể dẫn đến lỗi khi không đủ bộ nhớ vật lý để đáp ứng nhu cầu của các tiến trình và gây ra tình trạng treo hoặc chết đột ngột của hệ thống.

Kiến trúc dịch địa chỉ



Thời gian truy cập hiệu quả (Effective Access Time - EAT)

- Thời gian trung bình để truy cập dữ liệu trong hệ thống máy tính, bao gồm thời gian truy cập bộ nhớ chính (RAM), bộ nhớ đệm (cache), ổ đĩa cứng và bất kỳ thiết bị nào khác được sử dụng để lưu trữ dữ liệu.

- Thời gian truy cập hiệu quả được tính bằng công thức: EAT = (m * Tm) + (c * Tc) + (d * Td)
- Trong đó, m, c và d lần lượt là tỷ lệ các truy cập đến bộ nhớ chính, bộ nhớ đệm và ổ đĩa cứng. Tm, Tc và Td là thời gian truy cập cho bộ nhớ chính, bộ nhớ đệm và ổ đĩa cứng tương ứng.

- Mục đích của việc tính toán EAT là để đánh giá hiệu suất của hệ thống máy tính và tối ưu hóa thiết kế bộ nhớ để giảm thiểu thời gian truy cập.

Phân đoạn(segmentation)

- Là 1 kỹ thuật quản lý bộ nhớ được sử dụng bởi HĐH để xử lý bộ nhớ ảo. Nó chia bộ nhớ của 1 tiến trình thành các đoạn logic có kích thước biến đổi, trong đó mỗi đoạn tương ứng với 1 loại dữ liệu hoặc mã nhất định (ví dụ: stack, heap, code segment, v.v.). Các đoạn này không liền kề và có thể được lưu trữ ở các vị trí bộ nhớ vật lý khác nhau.

- Khi 1 tiến trình yêu cầu bộ nhớ, hệ điều hành sẽ cấp phát một hoặc nhiều đoạn có kích thước yêu cầu cho tiến trình đó. Mỗi đoạn được gán một địa chỉ cơ sở, đại diện cho vị trí bắt đầu của đoạn đó trong bộ nhớ vật lý. Độ dài của đoạn cũng được ghi lại, được sử dụng để kiểm tra tính hợp lệ của các truy cập bộ nhớ bởi tiến trình.

- Segmentation cung cấp nhiều lợi ích, chẳng hạn như:
+ Nó cho phép sử dụng bộ nhớ hiệu quả hơn bằng cách chỉ cấp phát số lượng bộ nhớ cần thiết cho một tiến trình.
+ Nó cho phép chia sẻ mã và dữ liệu giữa các tiến trình, giảm yêu cầu bộ nhớ và cải thiện hiệu suất.
+ Nó đơn giản hóa quản lý bộ nhớ, vì mỗi đoạn có thể được quản lý độc lập.

Phân đoạn kết hợp với phân trang (Segmentation với Paging)