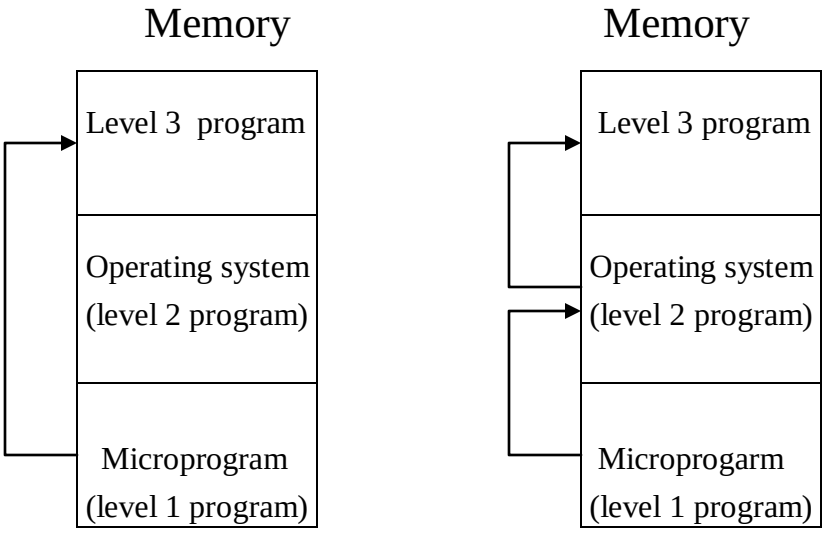
CHƯƠNG 5
CẤP MÁY HỆ ĐIỀU HÀNH

Cấp máy hệ điều hành được phát triển dần dần từ cấp máy qui ước. Hầu hết các chỉ thị của cấp máy hệ điều hành thuộc cấp máy qui ước. Chúng ta gọi những chỉ thị này là các chỉ thị cấp 3 tầm thường (ordinary), như là các chỉ thị thực hiện các phép toán số học, logic, dịch bit,... Chúng ta gọi các chỉ thị còn lại của cấp máy hệ điều hành (không có trong cấp máy qui ước) là các chỉ thị OSML (Operating System Machine Language).

Các chỉ thị cấp 3 tầm thường có thể được dịch trực tiếp bởi vi chương trình. Vi chương trình tìm nạp các chỉ thị từ chương trình cấp 3 và thực thi chúng. Khi gặp chỉ thị OSML, vi chương trình chuyển sang khởi động hệ điều hành. Hệ điều hành sau đó khảo sát chỉ thị OSML trong chương trình cấp 3 và dịch chỉ thị này. Sau đó chúng được dịch tiếp bởi vi chương trình.



Hình 5.1: Việc thực thi các chỉ thị máy cấp 3

Nghiên cứu sâu về hệ điều hành thuộc về môn học hệ điều hành, trong chương này chúng ta chỉ bước đầu làm quen với hệ điều hành. Chúng ta sẽ xem xét 3 vấn đề quan trong: bộ nhớ ảo, vào / ra các tập tin, và xử lí song song.

1. Bộ nhớ ảo

Bộ nhớ ảo (virtual memory) là một kỹ thuật được cung cấp bởi hệ điều hành làm cho máy tính có dung lương bộ nhớ làm việc lớn hơn dung lượng thực tế của RAM.

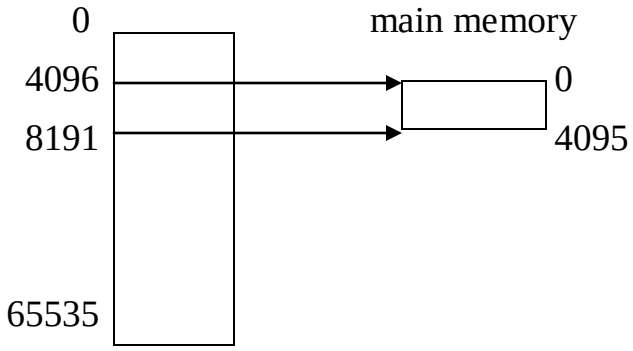
Trong những ngày đầu của lịch sử phát triển máy tính, bộ nhớ làm việc của máy tính rất nhỏ và rất đắt. Các nhà lập trình thời đó phải tốn khá nhiều công sức để rút gọn kích thước các chương trình sao cho đặt vừa chúng trong bộ nhớ.

Giải pháp cho vấn đề này là dùng bộ nhớ phụ (bộ nhớ ngoài). Người lập trình chia chương trình thành một số đơn vị nhỏ gọi là overlay (một phần của chương trình được lưu giữ trên đĩa và chỉ được gọi vào bộ nhớ khi có yêu cầu). Mỗi overlay có thể đặt vừa trong bộ nhớ, khi chạy chương trình, overlay đầu tiên được đưa vào bộ nhớ và được thực thi trong một thời gian. Khi hoàn tất, overlay kế tiếp được đưa vào và thực thi,...Người lập trình có nhiệm vụ phân chia chương trình thành các overlay, quyết định cất các overlay ở đâu trong bộ nhớ phụ, sắp xếp việc chuyển các overlay giữa bộ nhớ chính và bộ nhớ phụ, tức là quản lý toàn bộ quá trình overlay.

Vào năm 1961, một nhóm các chuyên gia ở Manchester (Anh) đưa ra một phương pháp thực hiện tự động hoá quá trình overlay. Người lập trình được giải phóng khỏi nhiệm vụ khó khăn này, phương pháp này ngày nay gọi là thực hiện bộ nhớ ảo. Vào đầu những năm 1970, phương pháp bộ nhớ ảo được áp dụng trên hầu hết các máy tính.

1.1 Phân trang

Nhóm Manchester phân biệt 2 khái niệm: không gian địa chỉ các chỉ thị và bộ nhớ thực tế trong máy tính. Ví dụ, một máy tính có trường chỉ thị 16 bit và có 4096 byte nhớ thực tế, thì không gian địa chỉ các chỉ thị là 2^{16} = 65536 = 64 kb. Trước khi phát minh ra bộ nhớ ảo, người ta phân biệt các địa chỉ dưới 4096 và bằng hoặc trên 4096. Các địa chỉ trên hoặc bằng 4096 không được dùng vì chúng không tương ứng với các địa chỉ của bộ nhớ thực tế.



Hình 5.2: phân trang bộ nhớ ảo

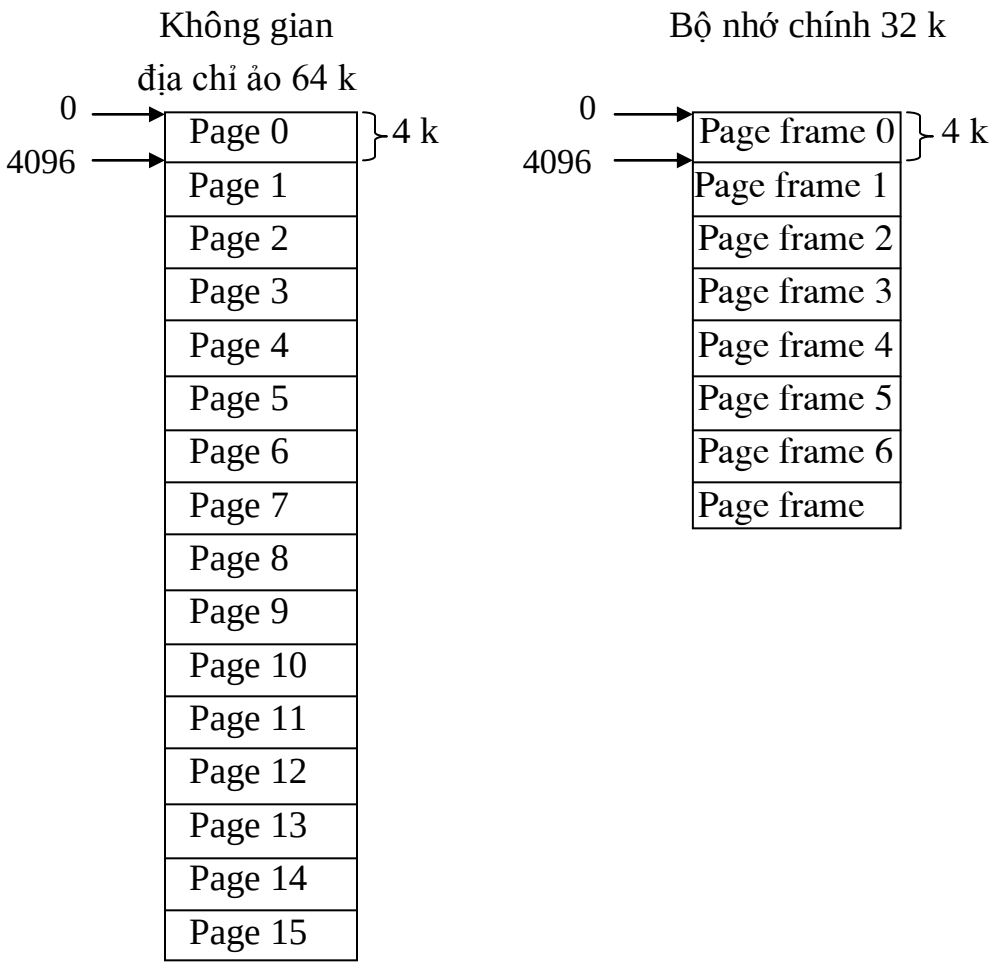
Ý tưởng của nhóm Manchester như sau: họ phân chia không gian địa chỉ chỉ thị thành các trang, mỗi trang có 4096 byte. Trang 1 từ 0 tới 4095, trang 2 từ 4096 đến 8191,... Khi chạy chương trình, các trang lần lượt được

đưa vào bộ nhớ chính một cách tự động.

1.2 Hiện thực phân trang

Điều kiện tiên quyết đối với phương pháp bộ nhớ ảo là phải có bộ nhớ phụ chứa được toàn bộ chương trình cần thực thi. Không gian địa chỉ ảo (phần không gian địa chỉ mà chương trình chiếm giữ trong bộ nhớ phụ) được phân ra thành nhiều trang có kích thước bằng nhau. Thường mỗi trang có kích thước từ 512 tới 4096 địa chỉ và phải là luỹ thừa của 2. Không gian địa chỉ bộ nhớ chính cũng được phân ra thành nhiều phần có kích thước bằng nhau sao cho mỗi phần chứa đúng một trang của bộ nhớ ảo. Mỗi phần của bộ nhớ chính được gọi là một khung trang. Trong hình 5.2, bộ nhớ chính chỉ có một khung trang, trong thực tế bộ nhớ chính chứa nhiều khung trang.

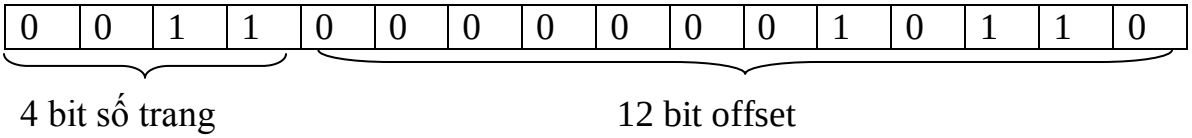
Hình 5.3 minh hoạ một sự phân trang của một không gian địa chỉ bộ nhớ ảo 64 k và sự phân chia khung trang của bộ nhớ chính 32 k.



Hình 5.3 phân trang không gian địa chỉ ảo và phân khung trang bộ nhớ chính

Địa chỉ ảo 16 bit được phân như sau : Số của trang ảo được xác định

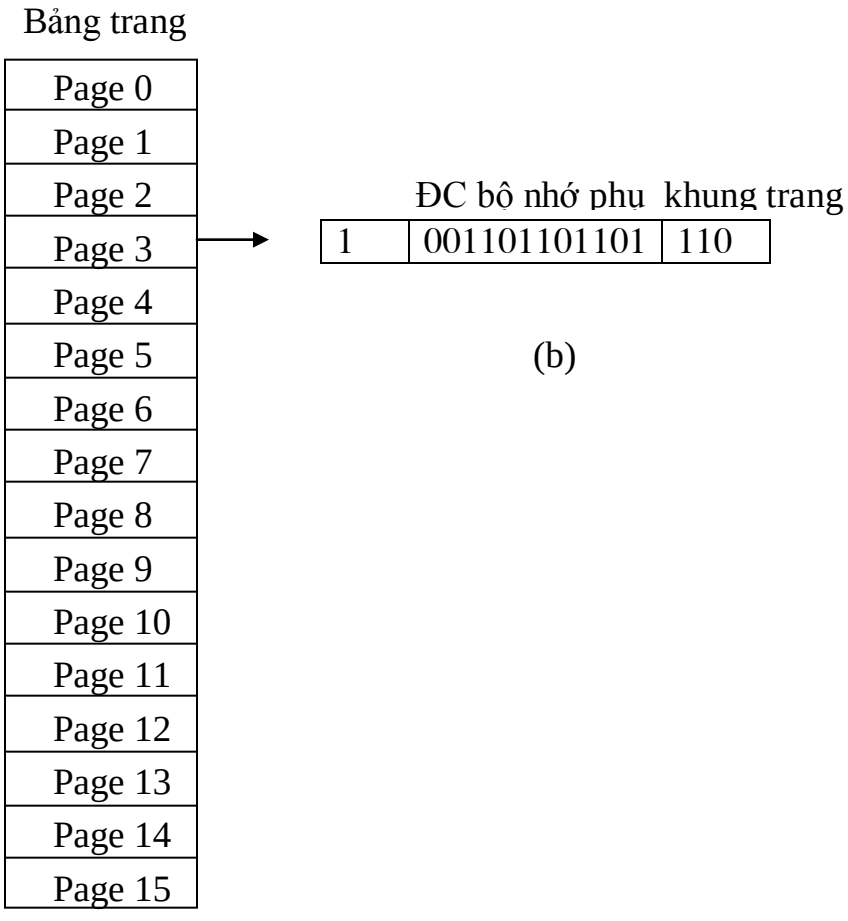
bởi 4 bit bên phải cùng và địa chỉ offset của trang được xác định bởi 12 bit còn lại của 16 bit địa chỉ ảo.



Hình 5.4: địa chỉ ảo 16 bit

Trong ví dụ trên, số trang ảo là 3, vị trí trong trang là 22. tương ứng với vị trí 12310 trong không gian địa chỉ 64 k.

Khi phát hiện cần trang ảo thứ 3, hệ điều hành có nhiệm vụ xác định vị trí hiện diện của trang 3. có 9 khả năng xẩy ra: trang 3 hiện diện một trong 8 khung trang của bộ nhớ chính, trang 3 nằm ở một nơi nào đó trên bộ nhớ phụ. Để làm được điều đó, hệ điều hành phải kiểm tra bảng trang.



(a)

(b)

Hình 5.5: (a) Bảng trang có 16 điểm nhập
(b) Khuôn dạng mỗi điểm nhập

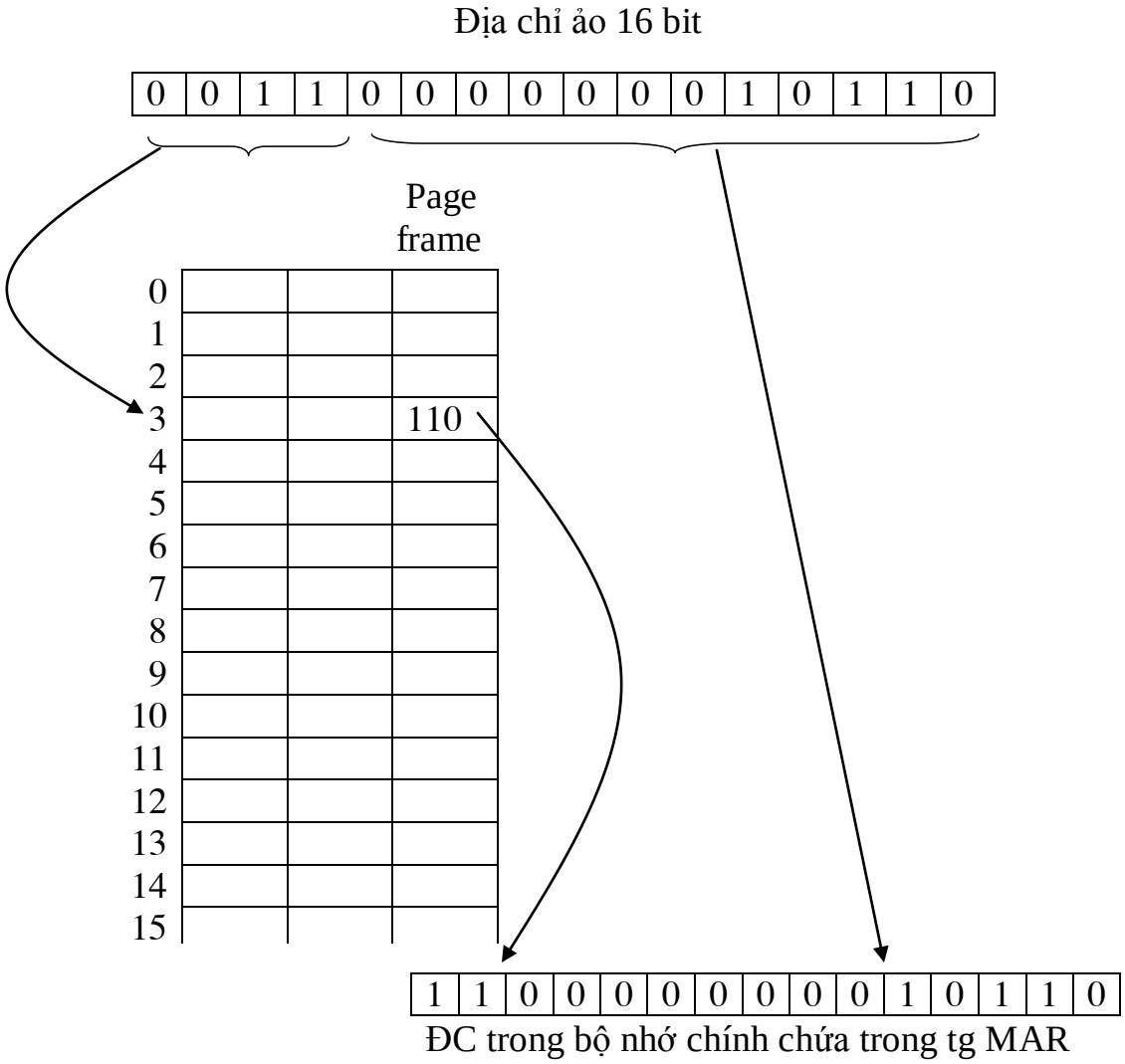
Hình 5.5 là ví dụ về bảng trang có 16 điểm nhập tương ứng với 16 trang ảo và khuôn dạng mỗi điểm nhập có 3 trường:

+ Trường đầu tiên có 1 bit. Bit là 1 nếu trang có mặt trong bộ nhớ chính, 0 nếu trang không có mặt trong bộ nhớ chính.

+ Trường thứ 2 có 12 bit cho biết vị trí của trang ở trong bộ nhớ phụ (như track, sector,... của đĩa) khi trang không ở trong bộ nhớ chính.

+ Trường thứ 3 có 3 bit cho biết khung trang trong bộ nhớ chính chứa trang cần truy nhập.

Giả sử, trang 3 hiện diện trong bộ nhớ chính tại vị trí khung trang 110 (6). Số của khung trang sẽ nạp vào 3 bit trái cùng của thanh ghi MAR, địa chỉ trong trang ảo (offset – 12 bit) được nạp vào 12 bit còn lại của thanh ghi MAR. (hình 5.6).



Hình 5.6: Hình thành địa chỉ bộ nhớ chính từ một địa chỉ ảo

1.3 Chiến lược thay thế trang

Trong trường hợp các khung trang trong bộ nhớ chính đã lấp đầy, muốn nạp một trang mới từ bộ nhớ phụ vào thì cần phải loại bỏ một trang nào đó trong bộ nhớ chính để tạo ra một khung trang trống. Vấn đề đặt ra là sẽ loại bỏ trang nào từ các khung trang của bộ nhớ chính?

1.3.1 Giải thuật LRU

Chiến lược thay thế trang sẽ phải đưa ra phương pháp loại bỏ trang nào để tối ưu nhất. Nếu chúng ta loại bỏ trang chứa chỉ thị sắp được thực thi, thì lỗi trang sẽ xuất hiện khi tìm nạp chỉ thị thị này. Đa số các hệ điều hành phải đoán trước trang nào trong bộ nhớ ít được dùng nhất theo nghĩa sự vắng mặt của chúng ít có ảnh hưởng nhất đến chương trình đang chạy. Tức là loại bỏ một trang mà trang này sẽ chưa được dùng đến trong thời gian dài nhất.

Một phương pháp thường dùng là loại bỏ trang được ít dùng gần đây nhất (least recently used - LRU). Giải thuật LRU về cơ bản là hợp lý nhưng có một số tình huống lại rất tồi.

Chẳng hạn chúng ta xét tình huống sau đây: Một vòng lặp lớn mở rộng đến 9 trang ảo, trong khi đó bộ nhớ chính chỉ có 8 khung trang. Sau khi 7 trang đầu của vòng lặp đã được đưa vào bộ nhớ chính, việc tìm nạp một chỉ thị thuộc trang ảo 8 sẽ dẫn đến việc cần phải thay thế một trang. Giải thuật LRU sẽ chọn trang 0 để loại bỏ và trang ảo 8 được đưa vào thay thế.

Sau khi thực thi một chỉ thị trên trang 8, chương trình nhẩy về đỉnh vòng lặp, tức lại cần tới trang 0, trang này lại phải đựa trở lại, giải pháp LRU chọn trang 1 để loại bỏ,... Như vậy trong tình huống này giải pháp LRU lại rất tồi.

Page 0	Page 8	Page 8
Page 1	Page 1	Page 0
Page 2	Page 2	Page 2
Page 3	Page 3	Page 3
Page 4	Page 4	Page 4
Page 5	Page 5	Page 5
Page 6	Page 6	Page 6
Page 7	Page 7	Page 7

Hình 5.7: Tình huống thất bại của giải thuật LRU

1.3.2 Giải thuật FIFO

Giải thuật FIFO (First In – First Out) – vào trước ra trước, loại bỏ trang được nạp vào trước nhất. Trong chiến lược này, đi kèm với mỗi khung trang là một bộ đếm. Ban đầu tất cả các bộ đếm đều bằng 0, sau mỗi lần xử lý mỗi trang bộ đếm của tất cả các trang có trong bộ nhớ được tăng lên 1, khi cần phải loại bỏ, trang nào có giá trị bộ đếm cao nhất sẽ được chọn.

1.4 Sự phân mảnh

Trong trường hợp chương trình và dữ liệu cần thực thi có kích thước không đúng bằng một số nguyên lần các trang thì trang cuối cùng sẽ có một phần không có dữ liệu hay chương trình. Ví dụ, nếu chương trình và dữ liệu cần 26000 byte, và mỗi trang có 4096 byte, như vậy trang thứ 7 chỉ chứa 1424 byte. Mỗi khi trang thứ 7 được nạp vào trong bộ nhớ thì bộ nhớ sẽ lãng phí 2672 byte. Vấn đề lãng phí bộ nhớ này gọi là sự phân mảnh.

Người ta tính toán được rằng nếu kích thước trang là n byte thì dung lượng bộ nhớ bị lãng phí ở trang cuối cùng do sự phân mảnh trung bình là n/2 byte. Nếu các trang có kích thước nhỏ, sự lãng phí sẽ được giảm thiểu, tuy nhiên, nếu kích thước trang nhỏ thì số trang sẽ lớn dẫn đến bảng phân trang lớn. Nếu bảng trang được lưu trữ trong các thanh ghi thì sẽ phải tốn thêm các thanh ghi. Hơn nữa thời gian truy xuất dữ liệu sẽ kéo dài hơn so với trường hợp kích thước trang lớn.

2. Các lệnh vào ra

Tập lệnh của cấp máy hệ điều hành chứ hầu hết các lệnh của cấp máy thông thường và có bổ sung thêm một số lệnh mới rất quan trong. Hoạt động vào ra là một trong những hoạt động quan trọng của cấp máy hệ điều hành. Điển hình của hoạt động vào ra lại là việc đọc ghi các tập tin (file).

2.1 Các tập tin đọc ghi tuần tự

Một phương pháp tổ chức vào ra thông thường là coi dữ liệu được đọc ghi như là một tập các bản ghi logic, trong đó một bản ghi là một đơn vị nào đó của thông tin có nghĩa đối với người lập trình. Một tập các bản ghi mang một ý nghĩa hay thực hiện một nhiệm vụ nào đó được gọi là một tập tin. Các bản ghi trong một tập tin có thể có chiều dài khác nhau.

Lệnh nhập cơ bản (read) đọc bản ghi kế tiếp của tập tin (đã được xác định) đặt vào các ô nhớ liên tiếp trong bộ nhớ chính ở một địa chỉ (đã được xác định). Để thực hiện thao tác đọc, lệnh phải xác định tối thiểu 2 thành phần thông tin sau:

1. Tập tin được đọc.

2. Địa chỉ bộ nhớ chính mà bản ghi được đặt vào.

Các lệnh READ tuần tự liên tiếp lấy các bản ghi logic liên tiếp từ tập tin đưa vào bộ nhớ chính.

Lệnh xuất cơ bản (write) ghi một bản ghi logic từ bộ nhớ lên một tập tin. Lệnh write tuần tự liên tiếp tạo ra các bản ghi liên tiếp trên tập tin.

Nhiều hệ điều hành yêu cầu một tập tin phải được mở trước khi đọc hoặc ghi, khi đó hệ điều hành sẽ cung cấp một lệnh OPEN. Khi kết thúc việc truy xuất tập tin, tập tin phải được đóng lại nhờ một lệnh (CLOSE) chẳng hạn.

2.2 Các tập tin truy xuất ngẫu nhiên

Các tập tin truy xuất ngẫu nhiên cho phép đọc (READ) hoặc ghi lại một bản ghi tại một vị trí bất kỳ trong tập tin bằng cách cho biết số của bản ghi.

Hầu hết các hệ điều hành đều cung cấp lệnh đọc bản ghi logic thứ n của tập tin. Lệnh này phải cung cấp tối thiểu 3 thông tin:

1. Tên tập tin cần đọc.
2. Địa chỉ bộ nhớ chính mà bản ghi được đặt vào.
3. Vị trí của bản ghi logic trong tập tin.

Các lệnh ghi (WRITE) cũng phải cung cấp các thông tin tương tự.

2.3 Tính khả thi của các lệnh vào ra

Để hiểu được cách các lệnh vào ra có thể được thực hiện bằng một trình phiên dịch chạy trên máy cấp 2 (hệ điều hành), chúng ta cần phải xem xét cách các tập tin được tổ chức và lưu trữ. Ở đây chúng ta giả sử các tập tin được lưu trữ trên các đĩa từ.

Một đĩa từ có thể bao gồm nhiều cylinder, mỗi cylinder lại có 1 hay nhiều track, mỗi track lại bao gồm nhiều sector, mỗi sector chứa một số byte nào đó.

Việc chọn đơn vị cấp phát không gian đĩa cho các tập tin có thể là cylinder, track hay sector. Nếu đơn vị cấp phát là cylinder thì tập tin chỉ có 1 ký tự cũng chiếm giữ cả một cylinder, gây lãng phí không gian đĩa. Ngày nay đơn vị cấp phát thường là cluster, mỗi cluster chứa một hay nhiều sector tuỳ thuộc vào loại đĩa và dung lượng của đĩa. (có thể xem bằng cách: NU\System Information\Drive\Details\Logical Information). Các tập tin có thể được lưu trữ trong các đơn vị cấp phát liên tiếp hoặc không (ngày nay là không liên tiếp).

Nếu tập tin được cấp phát liên tiếp, hệ điều hành chỉ cần biết vị trí bắt đầu của tập tin, kích thước của bản ghi logic (phần mềm), kích thước của bản ghi vật lý (phần cứng) để tính toán vị trí của bản ghi lôgic.

Nếu cấp phát không liên tiếp, để định vị một bản ghi lôgic, hệ điều hành cần một bảng chỉ số tập tin cho biết địa chỉ trên đĩa của từng bản ghi.

2.4 Các lệnh quản lý thư mục

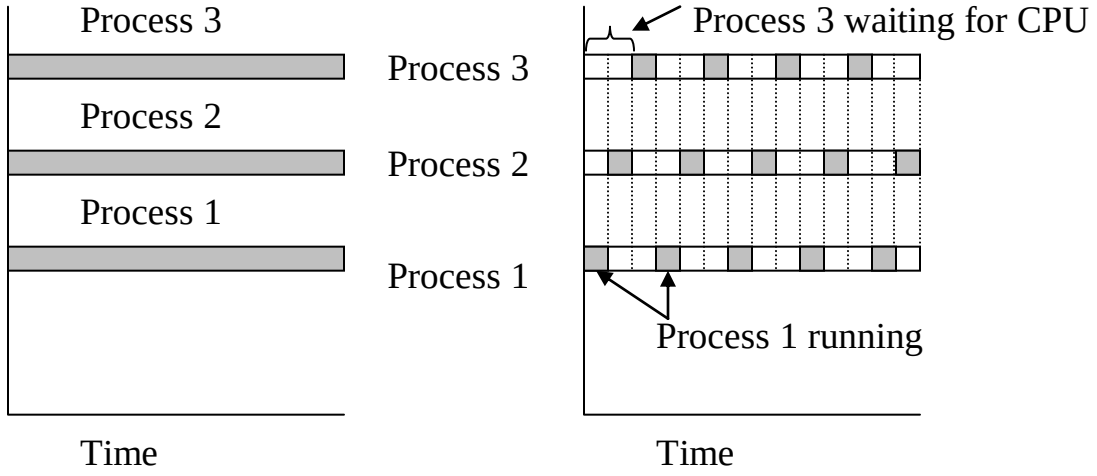
Số lương các tập tin trên bộ nhớ ngoài thường rất lớn, để tiện quản lý chúng, cần phải tổ chức chúng theo các nhóm có liên quan và đặt chúng vào trong các thư mục (directory). Tập lệnh máy cấp 3 cung cấp các chức năng tối thiểu sau đây liên quan tới thư mục và tập tin:

1. Tạo một tập tin và đưa vào một thư mục.
2. Xoá một tập tin khỏi thư mục.
3. Đổi tên tập tin.
4. Thay đổi trạng thái bảo vệ tập tin, thư mục.

3. Các lệnh dùng trong xử lý song song

3.1. Khái niệm về xử lí song song

Theo thuyết tương đối của Einstein, các tín hiệu điện không thể truyền với vận tốc vượt quá vận tốc của ánh sáng. Vì vậy, để rút ngắn thời gian truyền dữ liệu giữa các thành phần của hệ thống máy tính nhằm tăng tốc độ của máy tính thì kính thước của máy tính phải rất nhỏ. Một phương pháp khác để tăng tốc độ của máy tính là áp dụng phương pháp xử lí đồng thời nhiều tiến trình hay xử lí song song.



Hình 5.8: Xử lí song song

Trên một máy tính có nhiều bộ xử lí, mỗi một tiến trình (trong nhiều tiến trình đồng thời được xử lí) được gán cho một bộ xử lí cho phép các tiến trình được thực thi đồng thời. Trường hợp này được gọi là xử lí song song thực sự.

Nếu một máy tính chỉ có một bộ xử lí, việc xử lí song song có thể được mô phỏng bằng cách cho bộ xử lí chạy lần lượt từng tiến trình trong các khoảng thời gian ngắn. Hình 5.8 minh hoạ hai trường hợp xử lí song thực sự và mô phỏng.

3.2 Tạo tiến trình

Khi một chương trình được thực thi, nó phải chạy như là một phần của tiến trình nào đó. Tiến trình này được đặc trưng bởi một trạng thái và một không gian địa chỉ qua đó chương trình và dữ liệu được truy xuất. Trạng thái bao gồm bộ đếm chương trình và có thể một từ trạng thái chương trình, con trỏ lệnh và các thanh ghi đa năng.

Một số hệ điều hành đơn giản thường hỗ trợ một số tiến trình cố định, các tiến trình được tạo ra khi khởi động máy tính và mất đi khi tắt máy tính. Một số hệ điều hành phức tạp hơn, cho phép các tiến trình được tạo ra và kết thúc mà không phải ngừng máy tính.

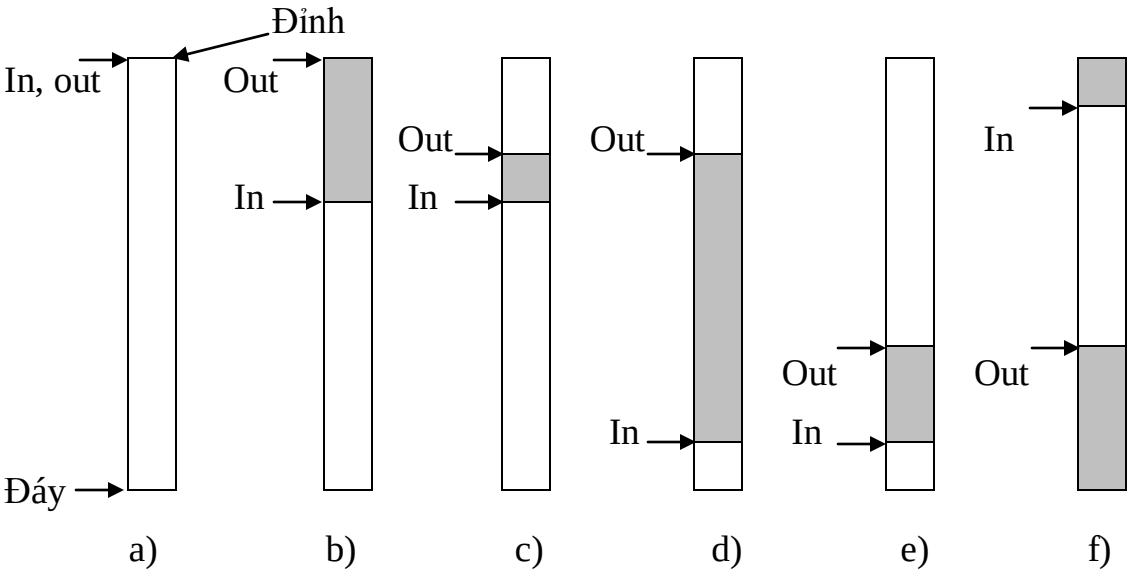
3.3 Điều kiện tranh đua

Chúng ta hãy khảo sát một tình huống có 2 tiến trình độc lập: tiến trình 1 (P1) và tiến trình 2 (P2). 2 tiến trình này truyền thông thông qua một vùng đệm dùng chung trong bộ nhớ chính. P1 là sản xuất (producer) còn P2 là tiêu thụ (consumer). P1 tính các số nguyên tố và đặt chúng vào vùng đệm từng số một. P2 lấy chúng ra khỏi vùng đệm từng số một và in chúng.

2 tiến trình này chạy song song ở các tốc độ khác nhau. Nếu P1 phát hiện ra vùng đệm đầy, nó sẽ tạm ngừng (sản xuất) để chờ tín hiệu từ P2. Sau đó nếu P2 đã lấy một số ra khỏi vùng đệm, nó gửi một tín hiệu để khởi động lại P1. Ngược lại, nếu P2 phát hiện ra vùng đệm rỗng, nó sẽ tạm ngừng (tiêu thụ) để chờ tín hiệu từ P1. Sau đó, nếu P1 nạp một số vào vùng đệm rỗng, nó gửi một tín hiệu để khởi động lại P2.

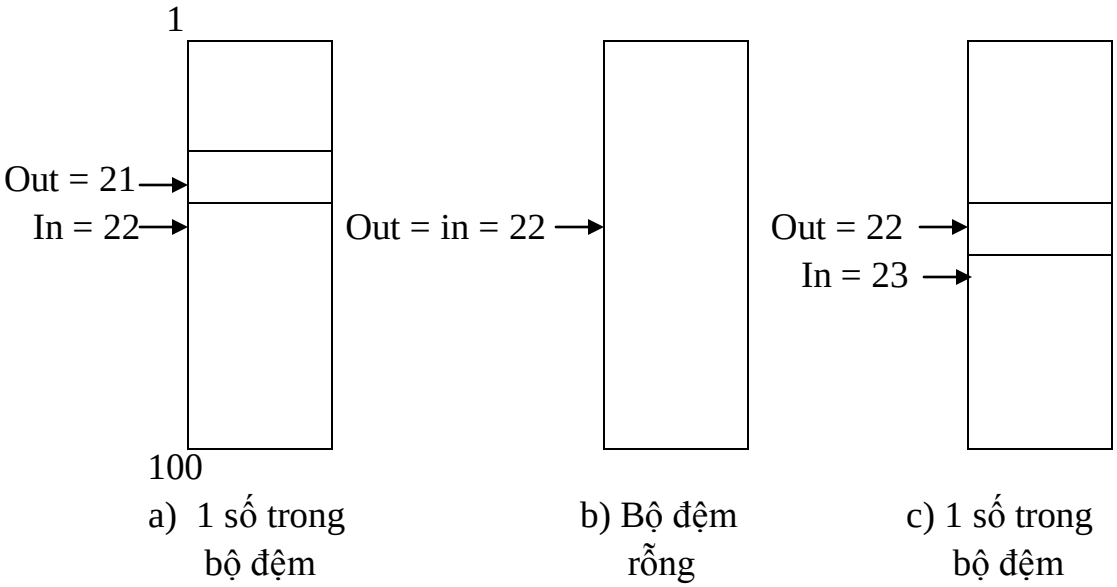
Trong thí dụ trên, người ta đã sử dụng một bộ đệm xoay vòng cho việc truyền thông giữa 2 tiến trình. Ở đây, sử dụng 2 con trỏ in và out : In trỏ tới từ trống kế tiếp (nơi P1 sẽ nạp số nguyên tố kế tiếp vào), out trỏ tới số kế tiếp sẽ được lấy ra bởi P2. Khi in = out, bộ đệm rỗng (hình 5.9 a). Sau khi P1 nạp vào bộ đệm một số số nguyên tố, bộ đệm ở trạng thái như hình 5.9 b. Sau khi P2 lấy một số số nguyên tố trong bộ đệm in ra, trạng thái bộ đệm như hình 5.9 c. Các hình 5.9 d – 5.9 f minh hoạ hoạt động quay vòng của bộ

đệm. Ở đây đỉnh của bộ đệm kề với đáy của bộ đệm theo nghĩa logic (chứ không phải theo nghĩa vật lý). Khi xẩy ra hiện tượng quay vòng, in đi theo out. Hiện tượng đầy xẩy ra khi in cách out một từ trống, từ này không thể được sử dụng, vì nếu sử dụng thì in = out, lúc đó sẽ không biết vùng đệm đầy hay rỗng.



Hình 5.9: Vùng đệm xoay vòng

Cơ chế producer – consumer có thể thất bại trong tình huống đặc biệt sau đây:



Hình 5.10: Điều kiện cạnh tranh

Chúng ta hãy xét trường hợp khi trong bộ đệm chỉ còn một số nguyên tố trong từ 21 và in = 22, out = 21 (hình 5.10 a). Lúc này P1 đang tìm kiếm một số nguyên tố và P2 đang in số ở vị trí 20. P2 kết thúc việc in số, lấy số cuối cùng ra khỏi vùng đệm, sau đó tăng out (in = out = 22). P2 in số và tìm nạp in, out để so sánh chúng.

Ngay lúc này sau khi P2 tìm nạp in, out nhưng chưa kịp so sánh chúng thì P1 tìm thấy một số nguyên tố mới, nó nạp số này vào vùng đệm và tăng in (in = 23, out = 22). P1 phát hiện ra in = next(out) và in lớn hơn out 1 từ, do vậy P1 kết luận P2 đang ngủ, nên gửi một tín hiện đánh thức (kết luận sai). Tuy nhiên, lúc này P2 đang thức nên tín hiệu dánh thức bị mất, P1 tìm kiếm một số nguyên tố mới.

P2 đã tìm nạp in, out trước khi P1 nạp một số cuối cùng vào vùng đệm, khi kiểm tra in = out nên P2 ngủ. Bây giờ P1 tìm thấy một số nguyên tố khác và nạp vào vùng đệm. P1 kiểm tra con trỏ và thấy in = 24, out = 22 và P1 kết luận có 2 số nguyên tố trong vùng đệm và P2 đang thức (kết luận sai). P1 tiếp tục tìm nạp cho tới khi đầy vùng đệm và bắt đầu ngủ. Như vậy cả 2 tiến trình đều ngủ mà không bị đánh thức.

Sự cố ở đây là tại thời điểm P2 chưa kịp so sánh in và out thì P1 nạp một số và phát hiện ra in = out + 1 nên kết luận P2 đang ngủ và gửi tín hiệu đánh thức nên bị mất. Sự cố này được gọi là điều kiện tranh đua (race condition).

Điều kiện tranh đua có thể được khắc phục bằng cách bổ sung cho mỗi tiến trình một bit chờ đánh thức. Mỗi khi có một tín hiệu đánh thức gửi tới mà tiến trình này đang chạy thì bit chờ được thiết lập giá trị 1. Mỗi khi một tiến trình đi ngủ nó kiểm tra bit chơ đánh thức nếu thấy là 1 nó sẽ lập tức khởi động lại và bit chờ được thiết lập về 0.