# Hệ điều hành là gì? Cho 5 ví dụ về hệ thống không có HDH và 5 ví dụ về các hệ thống có HDH?

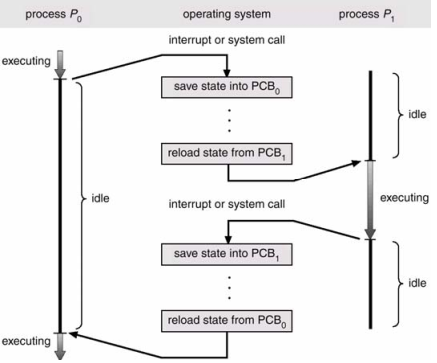
1. **HDH:**

* HDH là một bộ chương trình (mã nguồn và dữ liệu – các tham số hệ thống) đã được cài đặt sẵn có chức năng: điều phối, phân phối công việc cho các thành phần chức năng trong máy tính; thỏa mãn tối đa yêu cầu người dùng.
* Là chương trình trung gian giữa phần cứng máy tính và người sử dụng.
* Mục tiêu:
  + Giúp người dùng dễ dàng sử dụng hệ thống
  + Quản lý và cấp phát tài nguyên hệ thống một cách hiệu quả.

1. **Ví dụ:**

* 5 hệ thống không có HDH: hệ thống luận văn, hệ thống xí nghiệp, hệ thống cơ quan hành chính quốc gia, hệ thống thông tin, hệ thống làm mát.
* 5 hệ thống có HDH: hệ thống nhúng sử dụng hệ điều hành thời gian thực; Windows 95, 98, XP; MS Dos được sử dụng ở những năm 80 của thế kỷ XX; Ubuntu; Fedora.

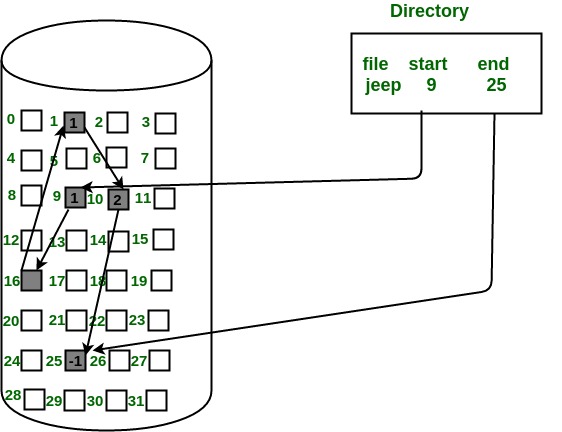
# Phân tích quá trình chuyển CPU giữa các tiến trình



* Tiến trình P1: vào hàng đợi Job-Queue ở trạng thái new, sẽ đợi 1 khoảng thời gian của quá trình điều phối chậm (Scheduler Long Term) của hệ điều hành (HĐH) để chọn tiến trình, sau khi được O.S chọn, P1 chuyển sang hàng đợi và ở trạng thái ready. Lúc này P1 chỉ đợi cấp CPU và running
* Sau một khỏang thời gian running, tiến trình P2 xuất hiện. Lúc này, hệ điều hành sẽ ghi lại thông tin của P1 vào thanh PCB1 bao gồm những thông tin: con trỏ, trạng thái của P1, số hiệu của tiến trình P1, Bộ đếm P1, nội dung của P1…Và chuyển P1 sang hàng đợi waiting và chuyển trạng thái Ready.
* Lúc này, P2 sẽ được cấp CPU và running. Và sau một khỏang thời gian running, P2 cũng sẽ chuyển sang hàng đợi waiting và chuyển trạng thái ready, lúc này HĐH cũng ghi lại thông tin vào thanh ghi PCB2 như đã làm ở P1. Sau đó, HĐH sẽ load lại thông tin của PCB1 và P1 sẽ tiếp tục running. Quá trình này cũng sẽ lập lại cho P2. Đển khi P1 và P2 kết thúc.

# Trình bày ý tưởng của kỹ thuật định vị file liên kết như sơ đồ bên dưới. Biết kích thước mỗi block bằng 512byte. Tính kích thước của Jeep.

* Kỹ thuật định vị file liên kết:
* Mỗi tập tin là một danh sách liên kết của các disk block: Các block có thể được phân tán rải rác) ở bất kỳ nơi nào trên đĩa
* Đơn giản - chỉ cần bắt đầu địa chỉ: địa chỉ bắt đầu là con trỏ đến đầu danh sách liên kết
* Hệ thống quản lý không gian trống - không lãng phí không gian
* Không có truy cập ngẫu nhiên
* Mapping:
  + Bảng phân bổ tệp (FAT) - dung lượng đĩa được phân chia bởi MSDOS và OS/2
  + Truy cập bảng FAT => danh sách các khối của file



* Tệp 'jeep' trong hình ảnh cho thấy cách các khối được phân phối ngẫu nhiên. Khối cuối cùng (25) chứa -1, biểu thị con trỏ null và không trỏ đến bất kỳ khối nào khác.
* Kích thước file jeep = 508 bytes

# Trình bày về các trạng thái và quá trình chuyển trạng thái của tiến trình. Cho ví dụ và phân tích.

1. **Các trạng thái của tiến trình:**

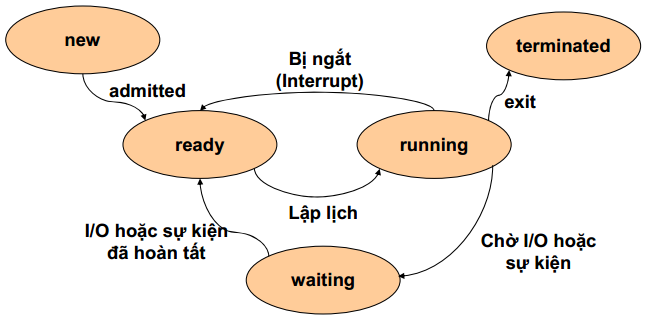
Trạng thái của tiến trình tại một thời điểm xác định bởi hoạt động của tiến trình tại thời điểm đó.

Trong quá trình sống, tiến trình có thể thay đổi trạng thái do các nguyên nhân:

* Phải dừng hoạt động do hết thời gian
* Đợi một thao tác I/O hoàn tất
* Phải chờ một sự kiện xảy ra

Tại 1 thời điểm, tiến trình có thể có một trong các trạng thái:

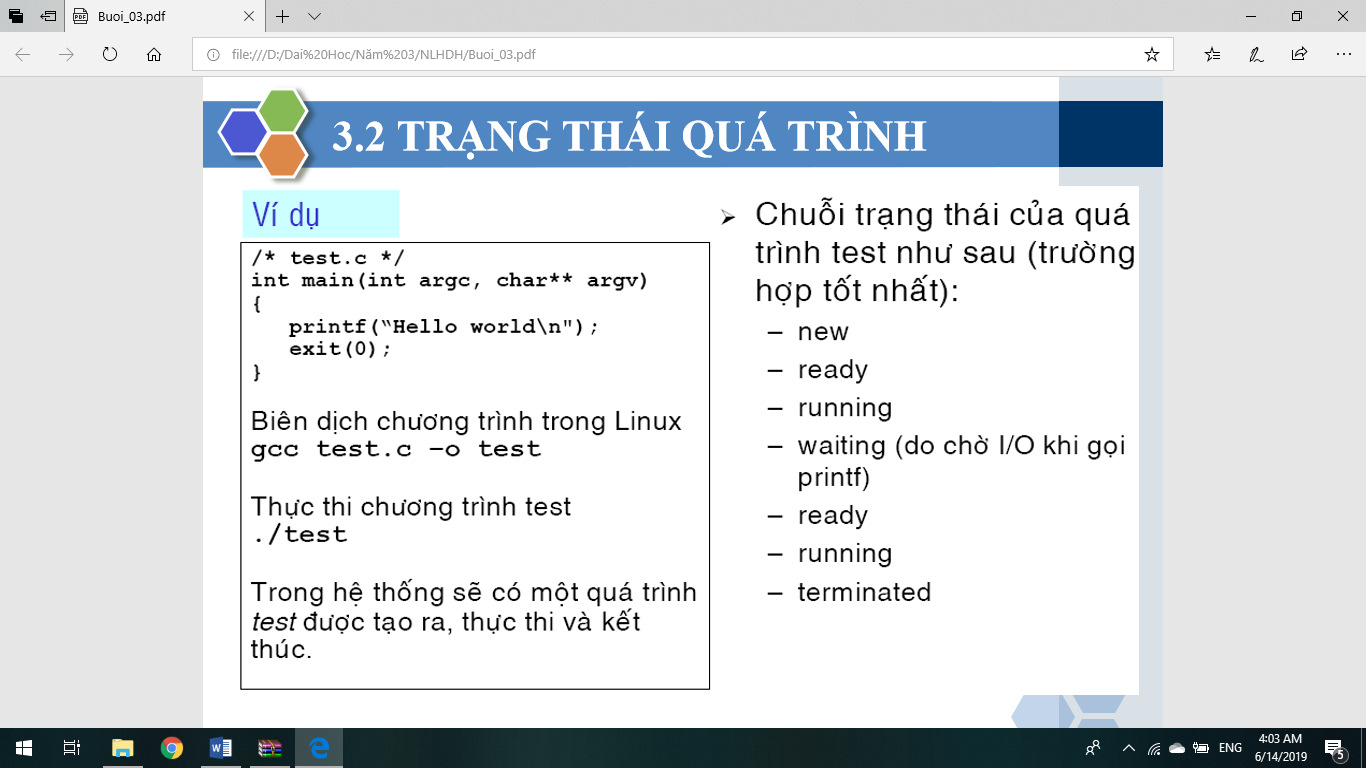
* new: tiến trình đang được tạo
* running: tiến trình đang chiếm hữu CPU và thực hiện các lệnh
* waiting: tiến trình đang chờ cũng được cấp tài nguyên hoặc chờ 1 sự kiện nào đó xuất hiện để chuyển sang trạng thái sẵn sàng
* ready: tiến trình ở trạng thái sẵn sàng, được phân phối đủ tài nguyên cần thiết, đang chờ đến lượt đc thực hiện theo cơ chế lập lịch của hệ điều hành
* terminated: tiến trình kết thúc. Nó không biến mất cho đến khi 1 tiến trình khác đọc đc trạng thái thoát của nó



1. **Quá trình chuyển trạng thái của tiến trình:**

* Tại một thời điểm, chỉ có một tiến trình có thể nhận trạng thái running. Trong khi đó, nhiều tiến trình có thể ở trạng thái waiting hay ready.
* Tiến trình mới tạo được đưa vào hệ thống, được cung cấp đủ tài nguyên ở trạng thái **ready** (chờ được phân phối CPU để thực hiện)
* Khi tiến trình đang thực hiện (running), nó có thể chuyển sang trạng thái:
* Kết thúc(terminal) nếu thực hiện xong
* Chờ(waiting) tiến trình yêu cầu một tài nguyên nhưng chưa được đáp ứng vì tài nguyên chưa sẵn sàng để cấp phát tại thời điểm đó ; hoặc tiến trình phải chờ một sự kiện hay thao tác nhập/xuất
* Sẵn sàng(ready) khi xảy ra ngắt để chuyển CPU cho tiến trình có mức ưu tiên cao hơn Bộ điều phối cấp phát cho tiến trình một khoảng thời gian sử dụng CPU hoặc hết thời gian chiếm hữu CPU
* Bộ điều phối chọn một tiến trình khác có trạng thái ready cho xử lý.
* Tài nguyên mà tiến trình yêu cầu trở nên sẵn sàng để cấp phát hay sự kiện hoặc thao tác I/O tiến trình đang đợi(có trạng thái waiting) hoàn tất, tiến trình chuyển sang ready

1. **Ví dụ:**



# Phân biệt hệ song song và hệ phân tán. Cho ví dụ

|  |  |  |
| --- | --- | --- |
|  | **Hệ song song** | **Hệ phân tán** |
| **Đặc điểm** | * Phối hợp nhiều bộ xử lý nhằm giải quyết một vấn đề cho trước một cách nhanh nhất. * Cho hệ thống có nhiều CPU chia sẻ đường truyền, bộ nhớ, đồng hồ, thiết bị I/O,… * Các CPU làm việc phụ thuộc độc lập do phải chia sẻ tài nguyên. So với hệ thống 1 CPU thì nhanh hơn, an toàn hơn. * Thuận lợi: Xử lý nhiều công việc cùng lúc, tăng độ tin cậy. * Gồm hai loại: Đa xử lý đối xứng, đa xử lý không đối xứng | * Mỗi bộ xử lý nói chung có chương trình làm việc riêng bán độc lập, vì lợi ích chung nên cần phối hợp hành động với nhau * Mỗi bộ xử lý có vùng nhớ riêng, các bộ xử lý truyền thông với nhau qua hệ thống mạng tốc độc cao. * Thuận lợi: Chia sẻ tài nguyên, tăng tốc độ tính toán, đáng tin cậy, tính truyền thông cao. * Trong hệ thống yêu cầu cơ sở hạ tầng về mạng. LAN hoặc WAN, cũng có thể là client-server, peer to peer. |
| **Ví dụ** | Version Encore của UNIX cho máy tính Multimax. Hệ thống này có hàng tá bộ xử lý. Ưu điểm của nó là nhiều tiến trình có thể thực hiện cùng lúc. | Hệ thống world wide web hay hệ thống email, mạng xã hội, v.v.... |

# Khóa chết là gì?

* Là trạng thái 2 hay nhiều tiến trình cùng chờ đợi 1 số sự kiện nào đó và nếu không có tác động đặc biệt nào từ bên ngoài thì sự chờ đợi này xem như là vô hạn.
* Hiện tượng tắc nghẽn xảy ra khi và chỉ khi hệ thống tồn tại 4 điều kiện sau:

- Có thể sử dụng tài nguyên không thể chia sẻ: Mỗi thời điểm, một tài nguyên chỉ được cấp phát cho một tiến trình. Khi tiến trình sử dụng xong, hệ thông mới thu hồi và cấp phát tài nguyên này cho tiến trình khác.

- Có hiện tượng giữ và đợi tài nguyên: Các tiến trình chiếm giữ các tài nguyên đã cấp phát cho nó và chờ được cấp phát thêm tài nguyên mới

- Không có hệ thống phân phối lại tài nguyên: Tài nguyên không thể được được thu hồi từ tiến trình đang chiếm giữ chúng trước khi tiến trình này sử dụng chúng xong

- Có hiện tượng chờ đội vòng tròn: Có ít nhất 2 tiến trình chờ đợi lẫn nhau: tiến trình này chờ được cấp phát tài nguyên mà tiến trình kia đang chiếm giữ và ngược lại

# Phân biệt giữa địa chỉ logic và địa chỉ vật lý. Cho ví dụ. Một địa chỉ vật lý có thể có nhiều địa chỉ logic không? Cho ví dụ.

1. **Phân biệt địa chỉ logic và địa chỉ vật lý:**

- Địa chỉ logic (Logical address) – là một vị trí nhớ được diễn tả trong một chương trình. . Mọi tham chiếu bộ nhớ trong mã lệnh chương trình được compiler tạo ra đều là địa chỉ logic - sinh bởi CPU; còn gọi là địa chỉ ảo.

- Địa chỉ vật lý (Physical address): là một vị trí thực trong bộ nhớ chính. Địa chỉ vật lý được chuyển từ địa chỉ logic khi trình biên dịch thực hiện quá trình linking/loading- còn gọi là địa chỉ thật – sinh bởi đơn vị quản lý bộ nhớ.

- Địa chỉ thật và ảo giống nhau trong lược đồ ánh xạ địa chỉ “compile-time” và “load-time” và khác nhau trong “execution-time”.

- Ví dụ: Nếu thanh ghi tái định vị có giá trị là 4000, người dùng mong muốn địa chỉ là 45 thì MMU sinh ra địa chỉ 4025.

1. **Một địa chỉ vật lý có thể có nhiều địa chỉ logic.**

Ví dụ: địa chỉ vật lý 12345h ứng địa chỉ logic 1234h:0005h, 1230h:0045h,1200h:0345h...

Do các segment gối đầu nhau nên mỗi ô nhớ có thể thuộc một vài segment khác nhau. Vì vậy một địa chỉ vật lý có thể ứng vơi nhiều địa chỉ logic khác nhau.

# Trình bày ý tưởng của kỹ thuật định vị file liên tục. Kỹ thuật này có xảy ra phân mảnh trong/ngoài không? Cho ví dụ và giải thích.

1. **Định vị file liên tục:**

Mỗi file được cấp một khoảng không gian gồm các khối nằm liên tiếp trên đĩa. Vị trí file trên đĩa được xác định bởi vị trí khối đầu tiên và độ dài hoặc số khối mà file đó chiếm. Chẳng hạn, nếu file được cầp phát n khối bắt đầu từ khối thứ s, khi đó các khối của file sẽ là s, s+1, s+2,…s+n-1. Khoản mục của file trong thư mục sẽ chứa địa chỉ khối đầu tiên và số khối mà file chiếm.

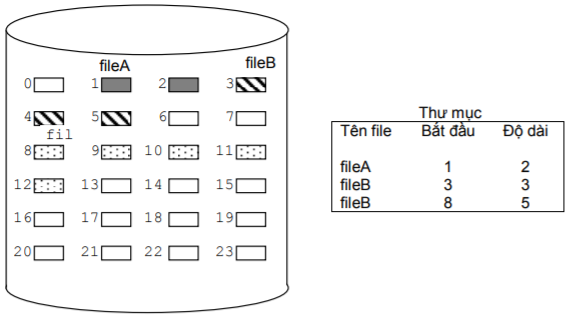
1. **Phân mảnh hay không?**

- Kỹ thuật này có xảy ra phân mảnh ngoài vì vấn đề trong cấp phát liên tục là tìm kiếm không gian trống dành cho file mới (ổ đĩa bị phân thành nhiều mảnh trống, không mảnh nào đủ lớn để lưu trữ dữ liệu).

- Có xảy ra phân mảnh trong. Giả sử, kích thước file tăng dần trong một thời gian dài, thì file phải được cấp phát tối đa, kể cả khi phần lớn không gian này có thể không đc sử dụng trong một thời gian dài.

1. **Ví dụ:**

Trên hình sau, vùng trống nằm trước fileA có kích thước 1 khối và không thể cấp phát cho file kích thước lớn hơn. Những vùng trống như vậy do đó bị bỏ phí. Mặc dù hiện tượng này có thể khắc phục bằng cách chuyển các file lại gần nhau để tập trung tất cả vùng trống về cuối đĩa nhưng việc di chuyển file cần khá nhiều thời gian, đặc biệt khi kích thước đĩa lớn.

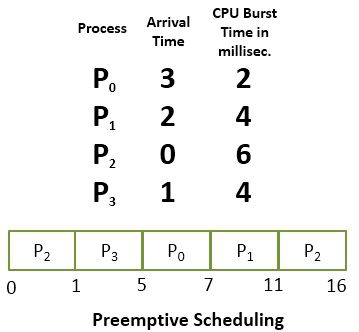


# Trình bày các loại lập lịch tác vụ. Với mỗi loại, cho ví dụ và phân tích

Lập lịch là quyết định tiến trình nào được sử dụng tài nguyên phần cứng khi nào, trong thời gian bao lâu. Bài toán lập được đặt ra với mọi dạng tài nguyên khác nhau, chẳng hạn thiết bị vào ra, CPU, bộ nhớ…, kể cả trong trường hợp có chia sẻ thời gian hay không.

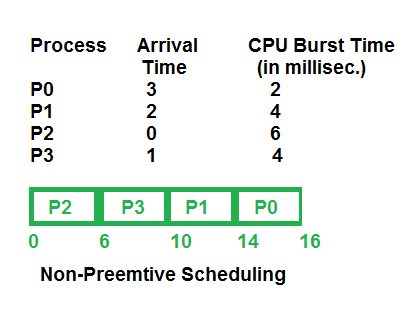
Tùy thuộc vào loại hình tác vụ, có 2 phương pháp lập lịch là không độc quyền (preemptive) và độc quyền (non-preemptive)

* **Lập lịch không độc quyền** là kiểu lập lịch trong đó hệ điều hành có thể sử dụng cơ chế ngắt để thu hồi CPU của một tiến trình đang trong trạng thái chạy. Với kiểu điều độ này, hệ điều hành có thể phân phối lại CPU một cách chủ động, không cần chờ cho tới khi tiến trình đang chạy kết thúc hoặc chuyển sang trạng thái chờ đợi. Windows 95, NT và các phiên bản sau sử dụng lập lịch độc quyền, cho phép thực hiện đa chương trình và chia sẻ thời gian đúng nghĩa và tin cậy hơn. Các thuật toán hoạt động theo lập lịch không độc quyền là Round Robin (phân phối luân phiên), SJF (Shortese Job First – công việc ngắn nhất trước), Priority Scheduling (sự ưu tiên).
* **Ví dụ và phân tích**



Có 4 tiến trình: P0, P1, P2, P3. Tại thời điểm vào 0 P2 đến, vì vậy CPU được cấp phát cho P2 vì không có tiến trình nào trong hàng đợi. Trong khi P2 thực thi, P3 đến vào thời điểm 1, thời gian còn lại cho tiến trình P2 (5 milisec) > thời gian P3 yêu cầu. Vì vậy, CPU được cấp phát cho P3. Thời điểm 2 ms, P3 đang thực thi thì P1 vào, thời gian còn lại cho tiến trình P3 (3 ms) < thời gian P1 yêu cầu (4 ms), nên P3 được tiếp tục. Khi P3 đang tiếp tục tiến trình, P0 đến vào thời điểm 3, thời gian còn lại cho tiến trình P3 (2 ms) = thời gian P0 yêu cầu (2 ms), P3 tiếp tục và sau khi P3 chấm dứt, CPU được cấp phát cho P0 vì burst time của nó ít nhất. Sau khi P0 kết thúc, CPU được cấp phát cho P1 sau đó đến P2.

* **Lập lịch độc quyền** là kiểu lập lịch trong đó tiến trình đang ở trạng thái chạy sẽ được sử dụng CPU cho đến khi xảy ra một trong các tình huống sau: tiến trình kết thúc, hoặc tiến trình phải chuyển sang trạng thái chờ đợi do thực hiện yêu cầu vào/ra, lời gọi hệ thống, hoặc chờ đợi tín hiệu đồng bộ từ tiến trình khác. Việc lập lịch chỉ có thể thực hiện khi tiến trình thể hiện thái độ hợp tác và nhường lại CPU do không cần dùng nữa. Trong trường hợp tiến trình chiếm CPU vô hạn, ví dụ khi sử dụng vòng lặp vô hạn không chứa lời gọi hệ thống, các tiến trình khác sẽ không bao giờ được cấp CPU. Các phiên bản đầu tiên của Windows là Windows 3.x sử dụng lập lịch không độc quyền. Các thuật toán hoạt động theo lập lịch độc quyền là FCFS, SRTF,….
* **Ví dụ và phân tích**



Tiến trình P2 đến thời điểm 0, do đó CPU được cấp phát cho P2, phải mất 6 ms để thực thi. Các tiến trình P0, P1, P3 đến hàng đợi sẵn sàng. Nhưng phải chờ cho đến khi P2 hoàn thành burst time CPU, sau đó P3 được cấp phát CPU, nó thực thi cho đến khi hoàn thành. Tương tự cho P1, P0

# Trình bày ý tưởng của kỹ thuật phân trang bộ nhớ. Giải thích tại sao kích thước của một trang luôn là lũy thừa của 2.

1. **Ý tưởng:**

- Phân bộ nhớ vật lý thành các khối (Block) có kích thước cố định và bằng nhau, gọi là khung trang. Không gian địa chỉ cũng được chia thành các khối có cùng kích thước với khung trang và được gọi la trang. Khi cần nạp một tiến trình dể xử lý các trang của tiến trình sẽ được nạp vào những khung trang còn trống. Một tiến trình có kích thước N trang sẽ yêu cầu N khung trang tự do

- Cơ chế MMU (Memory Management Unit) là cơ chế phần cứng hỗ trợ thực hiện chuyển đổi địa chỉ trong cơ chế phân trang gọi là bảng trang. Mỗi phần tử trong bảng trang cho biết địa chỉ bắt đầu của vị trí lưu trữ trang tương ứng trong bộ nhớ vât lý

- Mỗi địa chỉ phát sinh bởi CPU được chia làm 2 phần:

+ Số hiệu trang (p)

+ Địa chỉ tương ứng trong trang (d)

+ Địa chỉ vật lý của trang = d + địa chỉ bắt đầu của trang

1. **Giải thích:**

Việc chuyển một địa chỉ logic thành 2 phần riêng biệt là số hiệu trang (p) và độ dời của trang (d) một các dễ dàng. Bởi vì mỗi bit tượng trưng cho một số lũy thừa của 2, do đó khi tách địa chỉ thành 2 phần sẽ có kết quả là một lũy thừa của 2. Nếu không gian địa chỉ là 2m, kích thước trang nhớ là 2n thì sẽ có 2(m - n) mục trong bảng trang, với (m – n) bits cao cho số hiệu trang, n bits thấp cho độ dời trang.

# Trình bày ý tưởng thuật toán lập lịch ổ đĩa SSTF, FCFS.

1. **SSTF:**

* Di chuyển các đầu đọc đến các khối cần thiết theo vị trí lần lượt gần với vị trí hiện tại của đầu đọc nhất
* Luôn cho ra seektime nhỏ nhất.

1. **FCFS:**

* FCFS còn được gọi là first in first served
* Xử lý yêu cầu một cách tuần tự (yêu cầu nào đến trước thì thực hiện trước)
* Công bằng cho tất cả tiến trình
* Nếu nhiều tiến trình thì xác suất xem như là ngẫu nhiên
* Phương pháp này dễ lập trình nhưng không cung cấp dịch vụ tốt
* Ý tưởng: Di chuyển đầu đọc theo thứ tự yêu cầu

Không sắp xếp lại hàng đợi công việc

# Một tiến trình Găng có thể ở những trạng thái nào? Giải thích thông qua một ví dụ cụ thể

1. **Tài nguyên găng:**

* Tài nguyên Găng là các tài nguyên logic và vật lý phân bổ cho các tiến trình là song hành.
* Miền Găng là đoạn lệnh trong chương trình có khả năng phát sinh mâu thuẫn. Để không xảy ra mâu thuẫn truy xuất, cần đảm bảo tại một thời điểm chỉ có một tiến trình được vào miền Găng.

1. **Một tiến trình Găng có thể ở những trạng thái nào?**

* Một tiến trình đang ở trong đoạn Găng có thể có các trạng thái Running và Ready bởi vì đoạn Găng sẽ quyết định chỉ cho một tiến trình hợp lệ vào miền Găng để thực hiện các lệnh trong miền này (tức là đang ở trạng tháiRunning ), còn lại các tiến trình khác phải chờ khi tiến trình ở trong miền Găng thực hiện xong mới có thể thực hiện (tức là đang ở trạng thái Ready) và chúng không thể vào trong miền Găng gây xung đột trong quá trình xử lý.

1. **Ví dụ:**
   * Để tránh sụp đổ, người ta chỉ có cho phép tối đa 3 xe lưu thông đồng thời qua một cây cầu rất cũ. Hãy xây dựng thủ tục **ArriveBridge(int direction)** và **ExitBridge()** kiểm soát giao thông trên cầu sao cho :
     + Tại mỗi thời điểm, chỉ cho phép tối đa 3 xe lưu thông trên cầu.
     + Tại mỗi thời điểm, chỉ cho phép tối đa 3 xe lưuthông cùng hướng   
       trên cầu.
   * Mỗi chiếc xe khi đến đầu cầu sẽ gọi **ArriveBridge(direction)** để kiểm tra điều kiện lên cầu, và khi đã qua cầu được sẽ gọi **ExitBridge()** để báo hiệu kết thúc.
   * Giả sử hoạt động của mỗi chiếc xe được mô tả bằng một tiến trình **Car()** sau đây:

**Car(int direction)**

**//direction** xác định hướng di chuyển của mỗi chiếc xe.  
{

RuntoBridge();ArriveBridge(direction);   
PassBridge();ExitBridge();  
RunfromBridge();

}

Đoạn găng: là đoạn đi trên cầu PassBridge()

* + Cầu là tài nguyên chung. Các tiến trình (xe) chia sẻ cầu để thực hiện công việc
  + Khi số tiến trình vượt qua 3 thì hệ thống xảy ra mâu thuẫn

# Hệ điều hành thời gian thực. Với mỗi loại, cho ví dụ và giải thích

Hệ điều hành thời gian thực

* Đảm bảo giải quyết bài toán (tiến trình) không muộn hơn một thời điểm xác định
* Mỗi tiến trình được gắn với một thời gian xác định phải hoàn thành gọi là DeadTime
* Hoàn thiện bài toán muộn hơn không có ý nghĩa
* Ví dụ: giải quyết bài toán trong các lò phản ứng hạt nhân; game 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
* Phân loại:
* Hard real-time:
  + Hạn chế bộ nhớ phụ, tất cả dữ liệu nằm trong bộ nhớ chính
  + 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.

# Trình bày các loại HDH. Mỗi loại cho 2 ví dụ.

* **Hệ điều hành đơn chương trình**
  + Toàn bộ hệ thống máy tính phục vụ 1 chương trình từ lúc bắt đầu khi ctr được đưa vào bộ nhớ đến khi kết thúc chương trình
  + Khi 1 chương trình được đưa vào bộ nhớ và thực hiện => nó chiếm giữ mọi tài nguyên hệ thống nên không thể đưa ctr khác vào bộ nhớ
  + Ví dụ: Hệ điều hành MS DOC
* **Hệ điều hành đa chương trình**
  + Tại một thời điểm có nhiều ctr có mặt đồng thời trong bộ nhớ
  + Các chương trình đều có nhu cầu được phân phối bộ nhớ và CPU
  + Gồm 2 loại:  Hệ điều hành hoạt động theo mẻ (batch - lo) và hệ điều hành chia sẻ thời gian.
  + Ví dụ: hệ điều hành windows 95; OS windows 2003 server
* **Hệ điều hành thời gian thực**
  + Đảm bảo giải quyết bài toán( tiến trình) không muộn hơn một thời điểm xác định
  + Mỗi tiến trình được gắn với một thời gian xác định phải hoàn thành gọi là **DeadTime**
  + Hoàn thiện bài toán muộn hơn không có ý nghĩa
  + Ví dụ: giải quyết bài toán trong các lò phản ứng hạt nhân; game thời gian thực
* **Hệ song song**
  + Cho các hệ thống có **nhiều CPU** chia xẻ đường truyền, bộ nhớ, đồng hồ, thiết bị I/O…
  + Các CPU làm việc phụ thuộc độc lập do phải chia xẻ tài nguyên
  + So với hệ thống 1 CPU:
    - Nhanh hơn
    - An toàn hơn: một CPU hỏng không ảnh hưởng tới toàn bộ hệ thống
  + Ví dụ: version Encore của UNIX cho máy tính Multimax
* **Hệ phân tán**
  + Tập hợp các thiết bị tính riêng rẽ có thể giao tiếp với nhau,
  + Các mô hình hệ thống phân tán : client-server, peer-to-peer (ví dụ : Gnutella)

# Trình bày phân mảnh bộ nhớ, phân mảnh trong, phân mảnh ngoài. Cho ví dụ và phân tích

1. **Phân mảnh bộ nhớ**

* Phân mảnh bộ nhớ: Khi tiến trình dc tải vào, sau đó giải phóng bộ nhớ, ko gian bộ nhớ trống bị phân thành nhiều mảnh nhỏ.

1. **Phân mảnh trong**

* Phân mảnh trong: bộ nhớ được phân phối có thể lớn hơn không đáng kể so với bộ nhớ được yêu cầu;
* Ví dụ: cấp một khoảng trống 18,464 bytes cho một process yêu cầu 18,462 bytes.
* Hiện tượng phân mảnh nội thường xảy ra khi bộ nhớ thực (physical memory) được chia thành các khối kích thước cố định (fixed-sized block) và các process được cấp phát theo đơn vị khối.
* Ví dụ: cơ chế phân trang (paging)

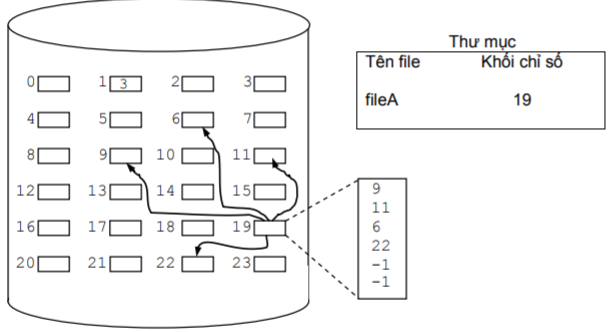
1. **Phân mảnh ngoài**

* Phân mảnh ngoài: là hiện tượng khi tổng lượng bộ nhớ trống đủ lớn để đáp ứng một yêu cầu nào đó, nhưng các khoảng trống không liên tục mà rải rác trên toàn bộ nhớ. Mức độ phân mảnh ngoài còn phụ thuộc vào tổng dung lượng bộ nhớ và kích thước trung bình của các tiến trình.
* Ví dụ: từ những phân tích thống kê về thuật toán first- fit, cứ trong N khối nhớ đc cấp phát thì N/2 khối nhớ sẽ không sử dụng được do hiện tượng phân mảnh. Vậy có tới ⅓ không gian bộ nhớ bị lãng phí.

# Trình bày ý tưởng kỹ thuật file theo chỉ số. Với kỹ thuật này, kích thước 1 khối bằng 512 byte, kích thước của mỗi phần tử trong khối chỉ số là 4 byte. Tính kích thước tối thiểu và tối đa của 1 file. Giải thích.

1. **Định vị file theo chỉ số:**

* Kỹ thuật định vị file theo chỉ số mang tất cả các con trỏ vào 1 vị trí: khối chỉ mục (index block).
* Khối chỉ mục của file là mảng chứa các khối dữ liệu nằm trên ổ đĩa. Thành phần thứ i trong mảng là địa chỉ khối thứ i của file. Thư mục chứa file ghi lại địa chỉ khối chỉ mục. Khi file mới được tạo ra, tất cả các con trỏ trong khối chỉ mục được thiết lập giá trị NIL( null). Trong lần đầu tiên ghi khối thứ i, bộ quản lý không gian trống tìm một khối tự do và địa chỉ khối này được đưa vào thành phần thứ i trong khối chỉ mục. Cấp phát theo chỉ mục hỗ trợ truy cập trục tiếp và không có hiện tượng phân mảnh trong, vì hệ thống có thể cấp phát bất kỳ khối trống nào trên ổ đĩa.

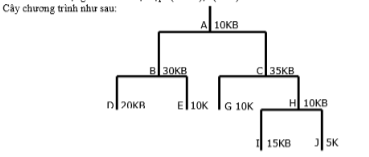


1. **Tính kích thước tối thiểu và tối đa của 1 file**

* Mỗi phần tử của khối chỉ số là 4 bytes = 2^2 bytes => có 2 bit
* Kích thước của mỗi block là 512 bytes = 2^9 bytes => có 9 bit
* Số bit tối thiểu là 9 – 2 = 7 bit => Kích thước tối thiểu của 1 file là 2^7 = 128 bytes
* Kích thước tối đa của 1 file là 2^9 x (2^7 + 1) = 66048 bytes

Cách 2: Kích thước tối đa = 512 \* 129 = 66048 bytes, kích thước tối thiểu = 128 bytes

# Trình bày ý tưởng của kỹ thuật Overlay. Chương trình gồm các module được tổ chức như hình vẽ. Tính kích thước bộ nhớ khi sử dụng kỹ thuật Overlay

****

* 1. **Ý tưởng:**
* Chỉ giữ trong bộ nhớ những lệnh và dữ liệu cần đến tại mọi thời điểm (thường là các module tải) 🡪 giảm không gian nhớ liên tục dành cho chương trình**.**
* Cơ chế Overlay
* Cho phép tổ chức ch.trình thành các đơn vị chương trình (module).
* Module luôn tồn tại trong quá trình thực hiện 🡪 module chương trình chính.
* Quan hệ độc lập/phụ thuộc chỉ sự có mặt của mộy nhóm module trong bộ nhớ đòi hỏi/không đòi hỏi sự có mặt của 1 nhóm module khác.
* Các module độc lập kg cần thiết phải có mặt đồng thời trong bộ nhớ
* Cần đến khi tiến trình có dung lượng lớn bộ nhứ đc cấp phát cho nó.
  1. **Tính kích thước**
* Modul A sd 2 modul độc lập B(30kb), C(35kb)
* Modul B sd 2 modul độc lập D(20kb), E(10kb)
* Modul C sd 2 modul độc lập G(30kb), H(20kb)
* Modul H sd 2 modul độc lập I(25kb), J(5kb)
* Kỹ thuật Overlay
* ABD = 10+30+20=60
* ABE = 10+30+10=50
* ACG = 10+35+10=55
* ACHI = 10+35+10+15=70
* ACHJ = 10+35+10+5=60

=>>>> Cả chương trình cần 70kb

# Phân biệt giữa luồng và tiến trình. Minh họa bằng một ví dụ với mã giả (C, Java,..)

* 1. **Phân biệt luồng và tiến trình**

|  |  |
| --- | --- |
| **Tiến trình** | **Luồng** |
| * Tiến trình có đoạn mã/dữ liệu/heap và các đoạn khác * Phải có ít nhất một luồng trong mỗi tiến trình * Các luồng trong phạm vi một tiến trình chia sẻ mã/dữ liệu/heap, và/ra nhưng có stack và tập thanh ghi riêng * Thao tác khởi tạo, luân chuyển tiến trình tốn kém. * Bảo vệ tốt do có không gian đĩa riêng * Khi tiến trình kết thúc, cài tài nguyên đươc đòi lại và các luồng phải kết thúc theo | * Luồng không có đoạn dữ liệu hay heap riêng * Luông không đứng riêng mà nằm trong một tiến trình * Có thể tồn tại nhiều luồng trong mỗi tiến trình. Luồng đầu là luồng chính và sở hữu không gian stack của tiến trình * Thao tác khởi và luôn chuyển luồng không tốn kém * Không gian địa chỉ chung, cần phải bảo vệ * Luồng kết thúc, stack của nó được thu hồi |

* 1. **Ví dụ:**

public class Hello extends Thread {

// luồng

public void run(){

System.out.println(“Hello Word”);

}

// tiến trình => hàm main thực hiện chương trình

Public static void main(String args[]){

(new Hello()).start();

// thực hiện luồng run() bằng phương thức start() có sẵn trong Thread

}

}

# Trình bày ý tưởng của các giải pháp Sleep-Weakup. Trong các giải pháp, khi một tiến trình muốn vào đoạn Găng nhưng chưa đủ điều kiện thì sau thời gian đã kiểm tra điều kiện vào đoạn Găng và trước thời gian được đánh thức tiến trình này có ảnh hưởng đến việc lập lịch cho các tiến trình khác không? Giải thích.

* 1. **Ý tưởng sử dụng SLEEP và WAKEUP:**
* Khi một tiến trình chưa đủ điều kiện vào miền găng, nó gọi SLEEP để tự khóa đến khi có một tiến trình khác gọi WAKEUP để giải phóng cho nó. Một tiến trình gọi WAKEUP khi ra khỏi miền găng để đánh thức một tiến trình đang chờ, tạo cơ hội cho tiến trình này vào miền găng :

*int busy; // 1 nếu miền găng đang bị chiếm, nếu không là 0*

*int blocked; // đếm số lượng tiến trình đang bị khóa*

*while (TRUE) {*

*if (busy){ blocked = blocked + 1; sleep();}*

*else busy = 1;*

*critical-section ();*

*busy = 0;*

*if(blocked){ wakeup(process); blocked = blocked - 1;}*

*Noncritical-section ();*

*}*

* 1. **Giải thích:**
* Tiến trình này có ảnh hưởng đến việc lập lịch cho các tiến trình khác :
* Giả sử tiến trình A vào miền găng, và trước khi nó rời khỏi miền găng thì tiến trình B được kích hoạt. Tiến trình B thử vào miền găng nhưng nó nhận thấy A đang ở trong đó, do vậy B tăng giá trị biến blocked và chuẩn bị gọi SLEEP để tự khoá. Tuy nhiên trước khi B có thể thực hiện SLEEP, tiến trình A lại được tái kích hoạt và ra khỏi miền găng. Khi ra khỏi miền găng A nhận thấy có một tiến trình đang chờ (blocked=1) nên gọi WAKEUP và giảm giá trị của blocked. Khi đó tín hiệu WAKEUP sẽ lạc mất do tiến trình B chưa thật sự « ngủ » để nhận tín hiệu đánh thức !Khi tiến trình B được tiếp tục xử lý, nó mới goi SLEEP và tự khóa vĩnh viễn !

# Trình bày các hoạt động của hệ điều hành trong quản lý tiến trình. Mỗi hoạt động cho ví dụ và phân tích.

* Các hoạt động: *Tạo tiến trình, kết thúc tiến trình* (hoạt động phụ: Tạm dừng tiến trình, Tái kích hoạt tiến trình, Thay đổi độ ưu tiên tiến trình)

1. **Tạo tiến trình:**

* Một tiến trình có thể tạo lập ra nhiều tiến trình mới bằng cách sử dụng lời gọi hệ thống tương ứng.
  + Tiến trình sử dụng lời gọi hệ thống để tạo lập là *Tiến trình cha*
  + Tiến trình được tạo gọi là tiến trình con
* Một tiến trình con có thể tạo ra các tiến trình mới…quá trình này tạo ra cây tiến trình
* Tạo tiến trình là một công việc "nặng nhọc" vì phải phân phối bộ nhớ và tài nguyên.
* Ví dụ: Nhập 100 số, in ra các số nguyên tố -> (1) Kiểm tra số nguyên tố -> (2) kiểm tra số dương -> (3) In ra số nguyên tố

1. **Kết thúc tiến trình:**

* Tiến trình kết thúc xử lý khi nó hoàn tất chỉ thị cuối cùng và sử dụng một lời gọi hệ thống để yêu cầu HĐH hủy bỏ
* Trong quá trình kết thúc:
  + Dữ liệu ra từ tiến trình con đến tiến trình cha (qua lệnh wait).
* Tiến trình cha có thể chấm dứt việc thực hiện tiến trình con (abort).
  + Tiến trình con dùng quá tài nguyên được phân phối.
  + Nhiệm vụ mà tiến trình con thực hiện không còn cần thiết.
* Khi tiến trình kết thúc, HĐH thực hiện các công việc:
  + Thu hồi các tài nguyên hệ thống đã cấp phát cho tiến trình
  + Hủy tiến trình khỏi tất cả các danh sách quản lý của hệ thống
  + Hủy bỏ PCB của tiến trình
  + Các tài nguyên được phân phối lại
* Hầu hết các hệ điều hành không cho phép các tiến trình con tiếp tục tồn tại nếu tiến trình cha đã kết thúc

# Thao tác nguyên tử là gì? Cho ví dụ. Tại sao trong giải pháp Test&Set, thao tác Test&Set(Lock) phải là thao tác nguyên tử.

* 1. **Thao tác nguyên tử là gì?**

Thao tác đọc và ghi bộ nhớ là thao tác nguyên tử (atomic) và không thể bị xen ngang giữa chừng.

* 1. **Ví dụ:**

TSL (boolean &target) {

TSL = target;

target = TRUE;

}

* 1. **Tại sao:**

Nguyên tắc chung của giải pháp Test&Set là hai thao tác kiểm tra giá trị và thay đổi giá trị cho một biến (một ô nhớ), hoặc các thao tác so sánh và hoán đổi giá trị hai biến, được thực hiện trong cùng một lệnh máy và do vậy sẽ đảm bảo được thực hiện cùng nhau mà không bị xen vào giữa. Đơn vị thực hiện không bị xen vào giữa như vậy được gọi là thao tác nguyên tử (atomic). Ta sẽ gọi lệnh như vậy là lệnh “kiểm tra và xác lập” Test\_and\_Set.

# Mô tả địa chỉ vật lý và địa chỉ logic của bộ nhớ ngoài. Thời gian truy cập ổ đĩa được tính thế nào? Tại sao với cùng một cấu hình phần cứng nhưng với các hệ điều hành khác nhau thì tốc độ ghi dữ liệu của đĩa lại khác nhau?

* 1. **Mô tả địa chỉ vật lý và địa chỉ logic của bộ nhớ ngoài:**
* Địa chỉ vật lý là một vị trí thực trong bộ nhớ chính.
* Địa chỉ logic là một vị trí nhớ được diễn tả trong chương trình.
  + Các trình biên dịch tạo ra mã lệnh chương trình mà trong đó mọi tham chiếu bộ nhớ đều là địa chỉ luận lý
  + Địa chỉ tương đối là một kiểu địa chỉ logic mà trong đó các địa chỉ được biểu diễn tương đối so với 1 vị trí xác định nào đó trong chương trình.
  + Địa chỉ tuyệt đối là địa chỉ tương đương với địa chỉ thực
  1. **Thời gian truy cập ổ đĩa:**

Thời gian truy cập = Thời gian đợi + Thời gian trao đổi dữ liệu

                       = (thời gian tìm kiếm + thời gian trễ) + thời gian trao đổi dữ liệu

*Chú ý: Thời gian đợi (thời gian định vị + thời gian trễ) đôi khi cũng được gọi là thời gian truy cập.*

* 1. **Tại sao**
* Do hệ điều hành xử dụng tiến trình đơn luồng hay đa luồng:
* MS-DOS, UNIX: tiến trình có 1 luồng điều khiển (heavyweight process).
* Windows, Linux:
  + - Tiền trình có thể gồm nhiều luồng
    - Có thể thực hiện nhiều nhiệm vụ tại một thời điểm

# Khóa chết, RAG

**\* Khóa chết (Deadlock)**

* Bốn điều kiện cần (necessary condition) để xảy ra deadlock
* Mutual exclusion: ít nhất một tài nguyên được giữ theo nonsharable mode (ví dụ: printer; ví dụ sharable resource: read-only files).
* Hold and wait: một process đang giữ ít nhất một tài nguyên và đợi thêm tài nguyên do quá trình khác đang giữ.
* No preemption: (= no resource preemption) tài nguyên không thể bị lấy lại, mà chỉ có thể được trả lại từ process đang giữ tài nguyên đó khi khi nó muốn..
* Circular wait: tồn tại một tập {P0,…,Pn} các quá trình đang đợi sao cho :

P0 đợi một tài nguyên mà P1 đang giữ   
P1 đợi một tài nguyên mà P2 đang giữ   
…   
Pn đợi một tài nguyên mà P0 đang giữ

**\* Các phương pháp giải quyết deadlock**

Ba phương pháp:

* Sử dụng một phương thức để ngăn ngừa hoặc tránh xa, đảm bảo rằng hệ thống sẽ không bao giờ đi vào trạng thái deadlock
* Cho phép hệ thống đi vào trạng thái deadlock rồi khôi phục lại.
* Bỏ qua vấn đề này, xem như deadlock không bao giờ xuất hiện tròn hệ thống. Giải pháp này được sử dụng trong hầu hết các HĐH

**\* Mô hình RAG:**

- RAG là đồ thị có hướng, với tập đỉnh V và tập cạnh E

- Tập đỉnh V gồm 2 loại:

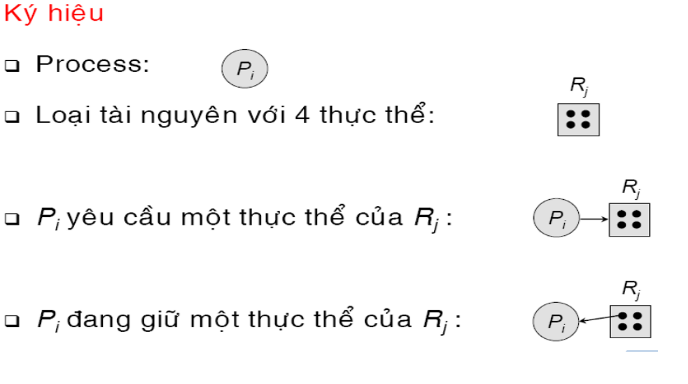
P = {P1, P2 …, Pn} //tất cả các Process trong hệ thống

R = {R1, R2…Rm}//tất cả các loại tài nguyên trong hệ thống

- Tập cạnh E gồm 2 loại:

Cạnh yêu cầu: Pi 🡪 Rj

Cạnh cấp phát: Rj 🡪 P



# Các giải pháp đồng bộ

**\* Giải pháp Busy waiting**

- Giải pháp phần mềm:

+ Sử dụng các biến cờ hiệu (simaphore)

+ Sử dụng việc kiểm tra luân phiên

+ Giải pháp của Peterson

- Giải pháp có sự hỗ trợ phần cứng;

+ Cắm ngắt

+ Test & Set

**\* Giải pháp Sleep and Wakeup**

* Khắc phục nhược điểm của các giải pháp Busy waiting bằng cách cho 1 tiến trình chưa đủ điều kiện vào đoạn găng sang trạng thái waiting
* Tạm khóa tiến trình kg cho sử dụng CPU ngay vì tiến trình chỉ sử dụng CPU khi ở trạng thái waiting
* Tiến trình chỉ có thể chuyển sang trạng thái running khi đang ở trạng thái ready (sẵn sàng)
* Giải pháp:
* HĐH sử dụng 2 thủ tục sleep và wakeup
* sleep là 1 lời gọi hệ thống có tác dụng tạm dừng hoạt động của tiến trình (chuyển sang trạng thái waiting) gọi nó và chờ đến khi đc 1 tiến trình khác <<Đánh thức>> lời gọi hệ thống wakeup nhận 1 tham số duy nhất: tiến trình sẽ đc tái kích hoạt (đặt về trạng thái ready)
* Ý tưởng
* Khi 1 tiến trình chưa đủ điều kiện vào đoạn găng, nó gọi sleep đẻ tự khóa đến khi có 1 tiến trình khác gọi wakeup để igiair phóng cho nó
* Một tiến trình gọi wakeup khi ra khỏi miền găng để đánh thức 1 tiến trình đg chờ, tạo cơ hội cho tiến trình này vào miền găng
* Các giải pháp phổ biến:

+ Simaphore (Dijktra đề xuất)

+ Monitors

+ Trao đổi thông điệp.

# Liên kết thư viện động

* **Dynamic linking (Liên kết động):**
* Quá trình link đến một module ngoài (external module) được thực hiện sau khi đã tạo xong load module (i.e. file có thể thực thi, executable)
  + - Ví dụ trong Windows: module ngoài là các file .DLL còn trong Unix, các 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
* Stub cần sự hỗ trợ của OS (như kiểm tra xem routine đã được nạp vào bộ nhớ chưa).
* **Ưu điểm của dynamic linking**:
* Thông thường, external module là một thư viện cung cấp các tiện ích của OS. Các chương trình thực thi có thể dùng các phiên bản khác nhau của external module mà không cần sửa đổi, biên dịch lại.
* Chia sẻ mã (code sharing): một external module chỉ cần nạp vào bộ nhớ một lần. Các process cần dùng external module này thì cùng chia sẻ đoạn mã của external module Þ tiết kiệm không gian nhớ và đĩa.
* Phương pháp dynamic linking cần sự hỗ trợ của OS trong việc kiểm tra xem một thủ tục nào đó có thể được chia sẻ giữa các process hay là phần mã của riêng một process (bởi vì chỉ có OS mới có quyền thực hiện việc kiểm tra này).

# Phân trang, phân đoạn

**\* Phân trang:**

* Không gian địa chỉ logic của 1 tiến trình có thể kg kề nhau; tiến trình đc phân phối bộ nhớ vật lý bất kỳ lúc nào khi bộ nhớ sẵn có.
* Chia bộ nhớ vật lý thành những khối có kích thước cố định là lũy thừa của 2 (512 bytes – 16Mb), đgl các frame (page vật lý)
* Chia bộ nhớ logic (dành cho các tiến trình) thành các khối cùng kích thước – các page, mỗi page có kích thước = 1 frame
* Luôn theo dõi all các frame còn trống
* Để chạy 1 ch.trình có kích thước n pages, cần phải tìm n frames còn trống và nạp chương trình
* Thiết lập 1 bảng phân trang (page table) để biên dịch (translate) các địa chỉ logic thành địa chỉ vật lý
* Nội dung mỗi phần tử trong page table co biết chỉ số frame (địa chỉ cơ sở) của bộ nhớ vật lý.

**\* Cách đánh địa chỉ theo trang**

- Địa chỉ đc đánh 1 cách phân cấp:

* Số hiệu trang (Page number - p) – Được sử dụng làm chỉ số đến phần tử trong bảng trang chứa địa chỉ cơ sở của các frame trong bộ nhớ vật lý
* Offset trang (Page offset - d) – Địa chỉ tương đối trong trang. Kết hợp vs địa chỉ cơ sở để xác định địa chỉ bộ nhớ vật lý đc gửi đến bộ nhớ.

- Địa chỉ ảo có m bit, sử dụng m-n bit cao làm số hiệu trang và n bít thấp làm Offset

- Kg có phân mảnh ngoài, có phân mảnh trong:

* giản cỡ trang --> giảm phân mảnh trong 🡪 giảm hiệu năng
* tăng cở trang 🡪 tăng hiệu suất 🡪 tăng phân mảnh trong

**\* Phân đoạn:**

- Phương thức quản lý bộ nhớ cho phép NSD “nhìn” bộ nhớ một cách dễ dàng dưới góc độ lập trình

- Một chương trình gồm nhiều phân đoạn, mỗi phân đoạn thể hiện dưới góc độ lập trình ở dạng:

main program (Chương trình chính)

function (Các hàm)

method (Các phương thức)

object (Các đối tượng, lớp)

local/global variables (Các biến)

common block (Các khối chung)

stack (Ngăn xếp)

symbol table, arrays (Bảng ký hiệu, mảng)

- Kiến trúc phân đoạn:

* Địa chỉ logic gồm 2 thành phần: <segment-number, offset>
* Segement table – ánh xạ địa chỉ vật lý 2 cấp, nội dung mỗi mục trong Sement table gồm có:

\_ Base: Chứa địa chỉ vật lý đầu tiên của đoạn trong bộ nhớ

\_ Limit: Xác định độ dài của đoạn

* Segment-table base register (STBR): trỏ tới vị trí của Segement table (bảng ohaan đoạn) trong bộ nhớ
* Segement-table length register (STLR): xác định số đoạn mà một chương trình sử dụng
* Segement number s là hợp lệ nếu s < STLR
* Phân đoạn: các đoạn có kicks thước khác nhau ( khác vs phân trang)
* Định vị: Động. Được thực hiện bởi bẳng phân đoạn
* Phân phối bộ nhớ: Giải quyết bài toán phân phối bộ nhớ động. First Fit / Best fit / Worst fit. Có sự phân mảnh ngoài.

# Kỹ thuật Overlay

* Ý tưởng: chỉ giữ trong bộ nhớ những lệnh và dữ liệu cần đến tại mọi thời điểm (thường là các module tải) 🡪 giảm kg gian nhớ liên tục dành cho ch.trình
* Cơ chế Overlay
* Cho phép tổ chức ch.trình thành các đơn vị ch.trình(module)
* Module luôn tồn tại trong quá trình thực hiện 🡪 module chương trình chính
* Quan hệ độc lập/phụ thuộc chỉ sự có mặt của 1 nhóm module trong bộ nhớ đòi hỏi/không đòi hỏi sự có mặt của 1 nhóm module khác
* Các module độc lập kg cần thiết phải có mặt đồng thời trong bộ nhớ
* Cần đến khi tiến trình có dung lượng lớn bộ nhứ đc cấp phát cho nó

**\* Ưu điêm – nhược điểm**

* Ưu điểm:
* Cấu trúc Overlay có tính chất định vị động 🡪 cho phép sử dụng bộ nhớ nhiều hơn phần bộ nhớ mà hệ thống dành cho ch.trình. Cấu trúc ch.trình mang tính chất tĩnh, kg thay đỏi trong tất cả các lần thực hiện chương trình
* So vs cấu trúc động, cấu trúc Overlay đòi hỏi cung cấp thông tin đơn giản, kg gắn cấu trúc vào chương trình nguồn
* Với sơ đồ Overlay tốt và các module độ dài kg quá lớn thì hiệu quả kg kém so vs cấu trúc động
* Nhược điểm:
* Hiệu quả tiết kieemk bộ nhớ phụ thuộc cách tổ chức, bố trí các module chương trình.

# Hacker vs Linux

Cần xây dựng hai thủ tục HackerArriver() và EmployeeArriver() được gọi tương ứng bởi bởi một hacker hoặc một nhân viên khi họ đến bờ sông để kiểm tra đieèu kiện có cho phép họ xuống thuyền không? Các thủ tục này sẽ sắp xếp những người thích hợp có thể lên thuyền. Những người đã lên thuyền khi thuyền chưa đầy sẽ phải chờ đến khi người thứ 4 lên thuyền thì mới có thể khởi hành sang sông (Không quan tâm đến số lượng thuyền hay thuyền qua sông rồi chở lại…Xem như luôn có thuyền để sắp xếp theo các yêu cầu hợp lệ.

Hacker(){

RuntoRiver(); //Đi đến bờ sông

HackerArrives(); //Kiểm tra điều kiện xuống thuyền

CrossRiver(); //Khởi hành qua sông

}

Employee(){

RuntoRiver(); //Đi đến bờ sông

EmployeeArrives(); //Kiểm tra điều kiện xuống thuyền

CrossRiver(); //Khởi hành qua sông

}

Đoạn găng và đồng bộ:

* Đoạn găng: CrossRiver(). Giải thích: Thuyền là tài nguyên tranh chấp. Khi có nhiều hơn số tiến trình (Hacker, Embloyee) được phép trong đoạn găng sẽ xảy ra mâu thuẫn (chìm thuyền).
* Đồng bộ: Đủ 4 người và 2 hacker, 2 nhân viên. Giải thích: yêu cầu đồng bộ xảy ra khi kiểm tra điều kiện xuống thuyền phải đúng 2 tiến trình nhân viên và 2 tiến trình hacker thì mới thực hiện CrossRiver().

# Với hệ thống đơn CPU, tại một thời điểm có N tiến trình ở hàng đợi Ready, M tiến trình hàng đợi Waiting. Hỏi có bao nhiêu cách lập lịch. Với N=3, giải thích rõ?

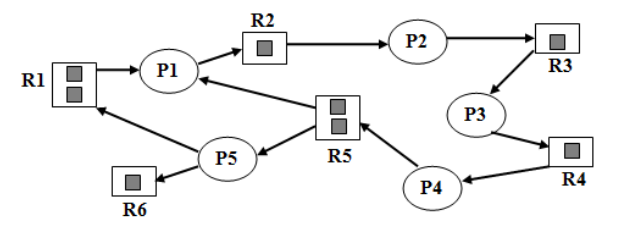
* Do hệ thống đơn CPU nên tại một thời điểm chỉ có một trạng thái Running
* Ở trạng thái W cần thiết bị ngoại vi hoặc được cung cấp tài nguyên mới có thể vào được trạng thái Running
* Nên không cần lập lịch cho trạng thái W. Còn tiến trình ở trạng thái Ready do không cần điều kiện để vào trạng thái Running nên ta cần lập lịch cho các tiến trình đang ở trạng thái Ready
* Theo đề bài có N tiến trình. Suy ra có N! Cách lập lịch
* Với N = 3. Giả sử có 3 tiến trình đang ở trạng thái Ready là a, b và c. Ta có 3! cách lập lịch như sau: a b c, a c b, b a c, b c a, c a b, c b a

# Trong kỹ thuật phân trang, giả sử bảng phân trang được lưu trong bộ nhớ và mỗi thao tác truy xuất bộ nhớ cân 200nS? Tính thời gian truy xuất dữ liệu trong bộ nhớ. Nếu sử dụng kỹ thuật thanh ghi kết hợp, với 79% tham chiếu bảng phân trang trong thanh ghi. Tính thời gian truy xuất dữ liệu trong bộ nhớ (Giả thiết thời gian truy xuất thanh ghi bằng 7%).

**Giải:**

* Thời gian truy xuất dữ liệu bộ nhớ:
  + Page Table+PhysicalMen = 200+200=400(nS)
* Thời gian truy xuất dữ liệu trong bộ nhớ:
* Tìm thấy thanh ghi: TLB + PhysicalMen
* Không thấy: TLB+PageTable+PhysicalMen
* Thời gian xuất trong bộ nhớ: 0,75(200+0)+0,25(0+200+200)=250nS

# Khóa chết là gì. Cho sơ đồ phân phối tài nguyên như trong hình vẽ. Hệ thống có khóa chết không. Giải thích.



* **Deadlock** (khoá chết - bế tắc):
  + Là hiện tượng một tiến trình chiếm hữu tài nguyên lâu dài làm cho các tiến trình có nhu cầu sử dụng tài nguyên này luôn ở trạng thái **waiting**
  + Một **tiến trình deadlock** là tiến trình đợi một sự kiện không bao giờ xảy ra
* **Deadlock có thể xảy ra nếu 4 điều kiện sau đồng thời tồn tại:**
* Ngăn chặn (loại trừ) lẫn nhau: tại một thời điểm, chỉ một tiến trình có thể sử dụng một tài nguyên(chỉ một tiến trình trong đoạn găng).
* Giữ và đợi: một tiến trình đang giữ ít nhất một tài nguyên và đợi để nhận được tài nguyên khác đang được giữ bởi tiến trình khác.
* Không có ưu tiên (độc quyền): một tài nguyên chỉ có thể được tiến trình (tự nguyện!) giải phóng khi nó đã hoàn thành công việc.
* Chờ đợi vòng tròn: tồn tại một tập các tiến trình chờ đợi {P0, P1, …, Pn, P0}

Ví dụ: Hai (hay nhiều hơn) ô tô đi ngược chiều trên một cây cầu hẹp chỉ đủ độ rộng cho một chiếc

* Giải thích (Hệ có khóa chết)
  + Mutual exclusion: ít nhất một tài nguyên đang ở chế độ không chia sẻ (R6).
  + Hold and wait: một process đang giữ ít nhất một tài nguyên và đợi thêm tài nguyên do quá trình khác đang giữ (P1, P2, P3, P4, P5)/
  + No preemption: tài nguyên không thể bị lấy lại, mà chỉ có thể được trả lại từ process đang giữ tài nguyên đó khi khi nó muốn.
  + Circular wait: tồn tại một tập {P0…, Pn} các quá trình đang đợi sao cho:
* P2 đợi một tài nguyên mà P1 đang giữ   
  P3 đợi một tài nguyên mà P2 đang giữ
* Chi tiết

Không. sơ đồ trên có chu trình nhưng không xảy ra deadlock vì P5 có thể kết thúc được,

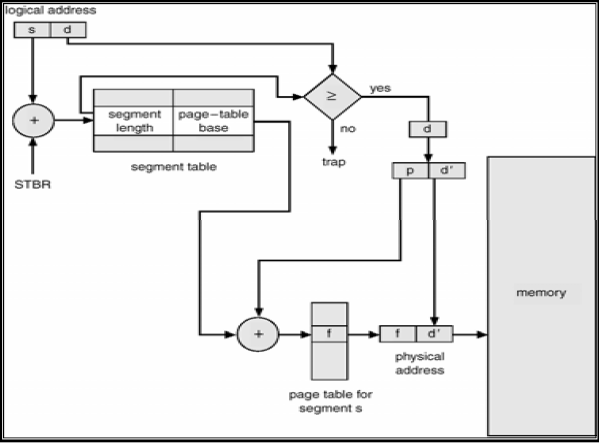
P5 giữ 1 thực thể tài nguyên R5 và đợi tài nguyên R6, R1, 2 tài nguyên này hoàn toàn có

khả năng đáp ứng. P5 kết thúc dẫn đến P4, P3, P2, P1 có thể kết thúc theo.

(Đầu bài khác)

***Hệ thống không có Deadlock****. Vì tiến trình P5 đang đợi tài nguyên R6 không nằm trong chu trình để đáp ứng yêu cầu chờ vòng tròn của khóa chết nên hệ thống đã cho không có khóa chết.*

1. **Trình bày lược đồ biên dịch địa chỉ trong kỹ thuật phân trang. Phân trang bộ nhớ có xảy ra phân mảnh trong/ngoài không? Giải thích.**
2. **Lược đồ biên dịch địa chỉ:**



* Chuyển đổi địa chỉ: Mỗi địa chỉ phát sinh bởi CPU được chia thành hai phần:
* Số hiệu trang (p): sử dụng như chỉ mục đến phần tử tương ứng trong bảng trang.
* Địa chỉ tương đối trong trang (d): kết hợp với địa chỉ bắt đầu của trang để tạo ra địa chỉ vật lý mà trình quản lý bộ nhớ sử dụng.
* Kích thước của trang do phần cứng qui định. Để dễ phân tích địa chỉ ảo thành số hiệu trang và địa chỉ tương đối, kích thước của một trang thông thường là một lũy thừa của 2 (biến đổi trong phạm vi 512 bytes và 8192 bytes). Nếu kích thước của không gian địa chỉ là 2m và kích thước trang là 2 n, thì m-n bits cao của địa chỉ ảo sẽ biễu diễn số hiệu trang, và n bits thấp cho biết địa chỉ tương đối trong trang.

1. **Phân mảnh hay không?**

Kỹ thuật phân trang loại bỏ được hiện tượng phân mảnh ngoài: mỗi khung trang đều có thể được cấp phát cho một tiến trình nào đó có yêu cầu. Tuy nhiên hiện tượng phân mảnh trong vẫn có thể xảy ra khi kích thước của tiến trình không đúng bằng bội số của kích thước một trang, khi đó, trang cuối cùng sẽ không được sử dụng hết.

[**1. Hệ điều hành là gì? Cho 5 ví dụ về hệ thống không có HDH và 5 ví dụ về các hệ thống có HDH? 1**](#_Toc45848632)

[**2. Phân tích quá trình chuyển CPU giữa các tiến trình 1**](#_Toc45848633)

[**3. Trình bày ý tưởng của kỹ thuật định vị file liên kết như sơ đồ bên dưới. Biết kích thước mỗi block bằng 512byte. Tính kích thước của Jeep. 2**](#_Toc45848634)

[**4. Trình bày về các trạng thái và quá trình chuyển trạng thái của tiến trình. Cho ví dụ và phân tích. 3**](#_Toc45848635)

[**5. Phân biệt hệ song song và hệ phân tán. Cho ví dụ 5**](#_Toc45848636)

[**6. Khóa chết là gì? 5**](#_Toc45848637)

[**7. Phân biệt giữa địa chỉ logic và địa chỉ vật lý. Cho ví dụ. Một địa chỉ vật lý có thể có nhiều địa chỉ logic không? Cho ví dụ. 6**](#_Toc45848638)

[**8. Trình bày ý tưởng của kỹ thuật định vị file liên tục. Kỹ thuật này có xảy ra phân mảnh trong/ngoài không? Cho ví dụ và giải thích. 6**](#_Toc45848639)

[**9. Trình bày các loại lập lịch tác vụ. Với mỗi loại, cho ví dụ và phân tích 7**](#_Toc45848640)

[**10. Trình bày ý tưởng của kỹ thuật phân trang bộ nhớ. Giải thích tại sao kích thước của một trang luôn là lũy thừa của 2. 9**](#_Toc45848641)

[**11. Trình bày ý tưởng thuật toán lập lịch ổ đĩa SSTF, FCFS. 9**](#_Toc45848642)

[**12. Một tiến trình Găng có thể ở những trạng thái nào? Giải thích thông qua một ví dụ cụ thể 10**](#_Toc45848643)

[**13. Hệ điều hành thời gian thực. Với mỗi loại, cho ví dụ và giải thích 11**](#_Toc45848644)

[**14. Trình bày các loại HDH. Mỗi loại cho 2 ví dụ. 12**](#_Toc45848645)

[**15. Trình bày phân mảnh bộ nhớ, phân mảnh trong, phân mảnh ngoài. Cho ví dụ và phân tích 13**](#_Toc45848646)

[**16. Trình bày ý tưởng kỹ thuật file theo chỉ số. Với kỹ thuật này, kích thước 1 khối bằng 512 byte, kích thước của mỗi phần tử trong khối chỉ số là 4 byte. Tính kích thước tối thiểu và tối đa của 1 file. Giải thích. 14**](#_Toc45848647)

[**17. Trình bày ý tưởng của kỹ thuật Overlay. Chương trình gồm các module được tổ chức như hình vẽ. Tính kích thước bộ nhớ khi sử dụng kỹ thuật Overlay 15**](#_Toc45848648)

[**18. Phân biệt giữa luồng và tiến trình. Minh họa bằng một ví dụ với mã giả (C, Java,..) 16**](#_Toc45848649)

[**19. Trình bày ý tưởng của các giải pháp Sleep-Weakup. Trong các giải pháp, khi một tiến trình muốn vào đoạn Găng nhưng chưa đủ điều kiện thì sau thời gian đã kiểm tra điều kiện vào đoạn Găng và trước thời gian được đánh thức tiến trình này có ảnh hưởng đến việc lập lịch cho các tiến trình khác không? Giải thích. 17**](#_Toc45848650)

[**20. Trình bày các hoạt động của hệ điều hành trong quản lý tiến trình. Mỗi hoạt động cho ví dụ và phân tích. 18**](#_Toc45848651)

[**21. Thao tác nguyên tử là gì? Cho ví dụ. Tại sao trong giải pháp Test&Set, thao tác Test&Set(Lock) phải là thao tác nguyên tử. 19**](#_Toc45848652)

[**22. Mô tả địa chỉ vật lý và địa chỉ logic của bộ nhớ ngoài. Thời gian truy cập ổ đĩa được tính thế nào? Tại sao với cùng một cấu hình phần cứng nhưng với các hệ điều hành khác nhau thì tốc độ ghi dữ liệu của đĩa lại khác nhau? 20**](#_Toc45848653)

[**23. Khóa chết, RAG 21**](#_Toc45848654)

[**24. Các giải pháp đồng bộ 22**](#_Toc45848655)

[**25. Liên kết thư viện động 23**](#_Toc45848656)

[**26. Phân trang, phân đoạn 24**](#_Toc45848657)

[**27. Kỹ thuật Overlay 26**](#_Toc45848658)

[**28. Hacker vs Linux 27**](#_Toc45848659)

[**29. Với hệ thống đơn CPU, tại một thời điểm có N tiến trình ở hàng đợi Ready, M tiến trình hàng đợi Waiting. Hỏi có bao nhiêu cách lập lịch. Với N=3, giải thích rõ? 28**](#_Toc45848660)

[**30. Trong kỹ thuật phân trang, giả sử bảng phân trang được lưu trong bộ nhớ và mỗi thao tác truy xuất bộ nhớ cân 200nS? Tính thời gian truy xuất dữ liệu trong bộ nhớ. Nếu sử dụng kỹ thuật thanh ghi kết hợp, với 79% tham chiếu bảng phân trang trong thanh ghi. Tính thời gian truy xuất dữ liệu trong bộ nhớ (Giả thiết thời gian truy xuất thanh ghi bằng 7%). 28**](#_Toc45848661)

[**31. Khóa chết là gì. Cho sơ đồ phân phối tài nguyên như trong hình vẽ. Hệ thống có khóa chết không. Giải thích. 29**](#_Toc45848662)

[**32. Trình bày lược đồ biên dịch địa chỉ trong kỹ thuật phân trang. Phân trang bộ nhớ có xảy ra phân mảnh trong/ngoài không? Giải thích. 30**](#_Toc45848663)