**CÂU HỎI ÔN**

**MÔN HỌC: HỆ ĐIỀU HÀNH.**

# 

Chọn phương án sai khi nói về chức năng của Hệ điều hành.

1. Quản lý bộ nhớ (Main-memory management).
2. Thực thi chương trình (Program execution)
3. Lập trình (Programming).
4. Quản lý thiết bị xuất nhập (I/O management).

# 

Thành phần nào sau đây không thuộc Hệ điều hành.

1. Dịch vụ Rom Bios (Rom Bios device drivers).
2. Quản lý tiến trình (Process management).
3. Hệ thống bảo vệ (Protection System).
4. Hệ thông dịch lệnh (Commander-Intpreter System)

… là Dịch vụ của Hệ điều hành.

1. Cài đặt chương trình (Program setting).
2. Xác định và xử lý lỗi (Error detection).
3. Sửa lỗi chương trình (Fix program errors).
4. Dịch chương trình thành mã thực thi (Translate a program into excecutable code)

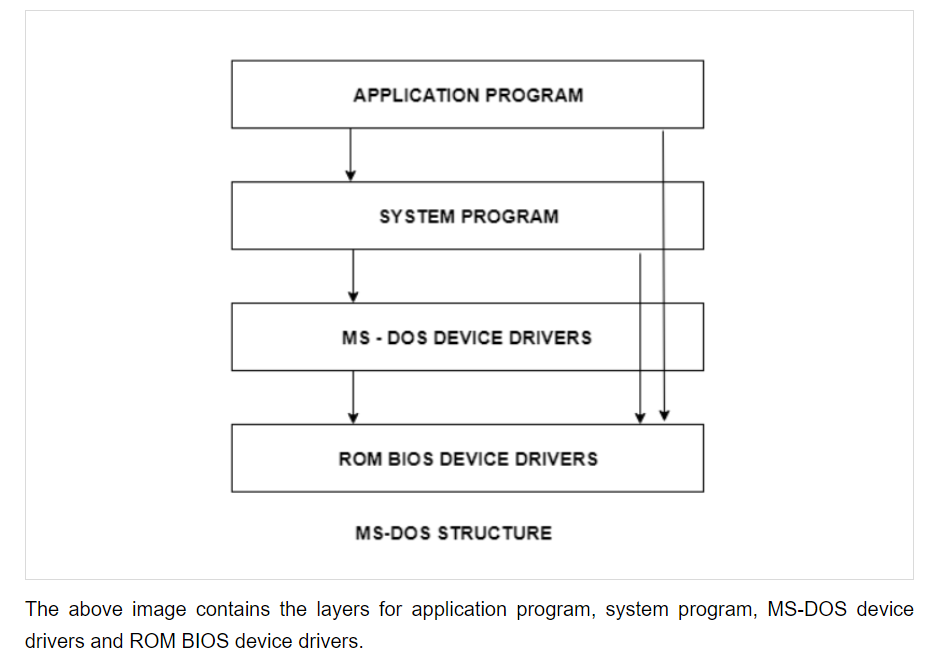
Thao tác của Command Line interface(CLI):

1. Nhập lệnh + Tham số (Commands + Command line parameters).
2. Double click trên Icon (Double click on Icon) .
3. Nhập lệnh + Tham số + ENTER (Commands+Command line parameters+ENTER).
4. Thao tác khác (Other operations).

# 

Thành phần nào sau đây thuộc hệ điều hành MS – DOS?

1. Resident system program.
2. Dịch vụ Rom Bios (Rom Bios device drivers).
3. [Command-line](https://en.wikipedia.org/wiki/Command-line_interface) Interpreters.
4. PowerShell.



# 

Các tiến trình (Process) hoạt động trong User mode sẽ có chế độ xử lý tiên trình là:

1. Đặc quyền (Nonpreemtive).
2. Không đặc quyền (Preemtive).

# 

Tiến trình (Process) có … trạng thái.

1. 3.
2. 4.
3. 5.
4. 6.

# 

# Khi tiến trình (Process) được cấp CPU và tài nguyên (Resources) sẽ chuyển từ trạng thái … sang trạng thái ...

1. Ready - Running.
2. Running – Terminated.
3. New - Running
4. Running - Waitting

# 

# Khi tiến trình (Process) đang ở trạng thái Running không thể chuyển sang trạng thái nào sau đây?

1. Ready.
2. Terminated.
3. New
4. Waitting

# 

# Khi tiến trình (Process) đang ở trạng thái Running và hết hạn thời gian được cấp (Time slice expired) sẽ chuyển sang trạng thái nào sau đây?

1. Ready.
2. Terminated.
3. New
4. Waitting

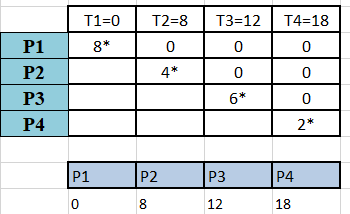
Diagram

Description automatically generated

# 

### Tính thời gian chờ (Waitting time) của các tiến trình (Process) P1, P2, P3, P4 khi sử dụng thuật toán lập lịch FCFS (**First-Come, First-Served Scheduling**).

1. 8; 12; 18; 28
2. 0; 8; 14; 20
3. 18; 8; 12; 18
4. 0; 8; 12; 18

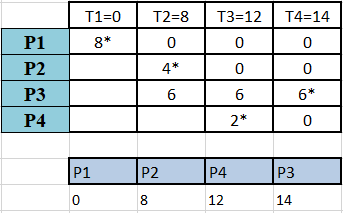


**Note: FCFS không quan tâm thời gian tới, chỉ quan tâm tiến trình nào tới trước thì sẽ xử lý trước. Xử lý xong tiến trình này sẽ tuần tự tới tiến trình tới kế đó. Cứ như vậy tới hết. Vì không quan tâm tới thgian tới, thười gian chờ của một tiến trình sẽ được tính bằng cách cộng dồn thời gian xử lý của các tiến trình được xử lý trước nó.**

# 

Tính thời gian chờ (Waitting time) của các tiến trình (Process) P1, P2, P3, P4 khi sử dụng thuật toán lập lịch SJF đặc quyền (Shortest-Job-First Scheduling nonpreemtive).

1. 0; 5; 9; 3
2. 5; 9; 3; 0
3. 0; 3; 5; 9
4. 0; 3; 9; 5



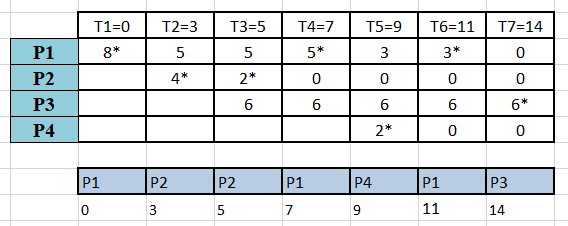
**Note: SJJF đặc quyền sẽ ưu tiên tiến trình có thời gian xử lý ngắn hơn. Khi một tiến trình đang xử lý, các tiến trình khác tới sau không được phép chèn vào, mà phải chờ hệ thống xử lý xong tiến trình hiện tại.**

# 

# 

### Tính thời gian chờ (Waitting time) của các tiến trình (Process) P1, P2, P3, P4 khi sử dụng thuật toán lập lịch SJF không đặc quyền (Shortest-Job-First Scheduling preemtive).

1. 0; 5; 9; 1
2. 6; 0; 9; 0
3. 4; 3; 6; 9
4. 0; 3; 5; 0



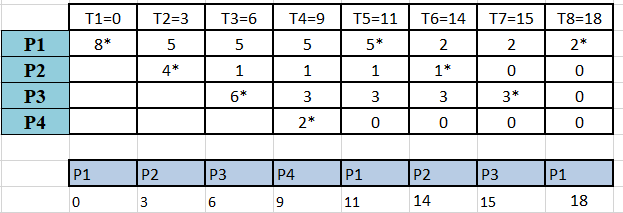
**Note: điểm khác biệt chính giữa SJF đặc quyền và không đặc quyền nằm ở việc ưu tiên xử lý các tiến trình. SJF đặc quyền cũng ưu tiên tiến trình có thười gian xử lý ít hơn. Điểm khác biệt ở đây là, SJF ko đặc quyền sẽ không chờ hệ thống xử lý xong một tiến trình rồi mới xét các tiến trình tiếp theo, tức là, nếu một tiến trình đang thực hiện mà có một tiến trình khác tới vời thời gian xử lý thấp hơn tiến trình hiện tại, thì nó sẽ ưu tiên tiến trình mới tới.**

# 

# 

### Tính thời gian chờ (Waitting time) của các tiến trình (Process) P1, P2, P3, P4 khi sử dụng thuật toán lập lịch RR (Round Robin Scheduling).

1. 12; 13; 14; 15
2. 9; 12; 12; 11
3. 12; 11; 12; 9
4. 12; 9; 11; 12

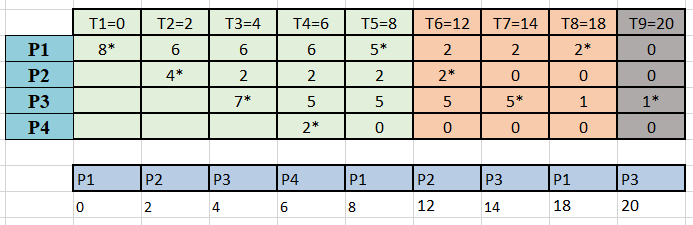


# 

# 

### Tính thời gian chờ (Waitting time) của các tiến trình (Process) P1, P2, P3, P4 khi sử dụng thuật toán lập lịch nhiều mức và có điều phối giữa các mưc (Multilevel Feedback Queue Scheduling).

1. 14; 6; 14; 10
2. 6; 12; 14; 8
3. 14; 14; 12; 9
4. 6; 14; 14; 6
5. 12 10 14 6



### P1, P2 cùng sở hữu biến TaiKhoan, TienRut (không âm) và đoạn CT

…

If (TaiKhoan-TienRut>=0)

TaiKhoan=TaiKhoan-TienRut

Else

error();

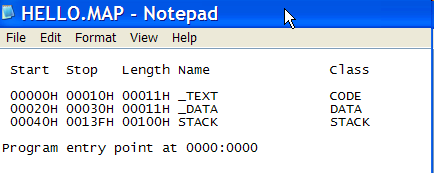
…

Khởi đầu : Taikhoan=1000; P1 TienRut=600; P2 TienRut=500

### Khi xảy ra tranh đoạt điều khiển (Race condition) giữa P1, P2 có khả năng TaiKhoan = …

1. 500
2. 600
3. -100
4. Giá trị không xác định (undefined value)

**Nếu P1 thực hiện trước thì TaiKhoan > TienRut nên điều kiện if thỏa. Race condition xảy ra, vì một lý do nào đó mà đã hết thười gian xử lý mà hệ thống vẫn chưa xử lý tiến trình P1. Lúc này P2 lại tới, với yêu cầu xử lý cũng trên các biến như P1. Hệ thống kiếm tra điều kiện if của P2 thấy thỏa thì thực hiện TaiKhoan=TaiKhoan-TienRut=1000-500=500. Sau khi xong P2 thì hệ thống mới quay lại P1 xử lý tiếp. Điều kiện if của P1 xét thỏa, lúc này hệ thống sẽ không xét lại nữa mà nhảy vô khối lệnh trong if luôn, tức là TaiKhoan = TaiKhoan (hiện tại) - TienRut = 500 - 600 = -100.**



Hình trên cho thấy địa chỉ các phân đoạn khi dịch một chương trình. Sự ràng buộc trên thuộc loại kết buộc địa chỉ … ?

1. Địa chỉ động (Relocatable address).
2. Địa chỉ tương đối (Relative address).
3. Địa chỉ tuyệt đối (Absolute address).
4. Địa chỉ không xác định (Undefined address).

Text

Description automatically generated

# 

### Mô phỏng nào sau đây của hàm Wait(s) { … } trong giải pháp Semaphore (Semaphore solution)?

1. While s<=0; s= s – 1;
2. While s<=0, s= s + 1;
3. While s<=0 s= s – 1;
4. While s<=0; s= s + 1;

**Letter

Description automatically generated**

**Hàm wait chạy khi s=1, hàm signal() chạy khi s=0.**

### Trong truy xuất độc quyền (Mutual exclusion). Cho 2 tiến trình P1, P2 cùng truy xuất Buffer; P1 đặt data vào Buffer; P2 lấy data từ Buffer. S Semaphore kiểm soát truy xuất Buffer (S=1). Cấu trúc lại miền Găng (Critical-section constuction) P1, P2 :

# 

### Khi tiến trình P1 truy xuất thành công Buffer, giá trị S = …

1. -1
2. 0
3. 1
4. Giá trị không xác định (undefined value).

**P1, P2 đang xung đột nên sẽ vô miền chờ. P1 thực hiện trước, sau khi gọi hàm wait(), vì s=1 nên hàm wait được thực hiện, tức là s=s-1=0. P1 thực hiện, sau đó gọi hàm signal(), vì s=0 nên thỏa điều kiện thực hiện hàm signal, tức là s=s+1=1**

**\*Note: miền găng là đoạn ct có khả năng xảy ra lỗi nếu truy xuất tài nguyên dùng chung**

### Trong hoạt động phối hợp (Synchronization). Tác vụ X = tác vụ X1+tác vụ X2 (tác vụ X1 thực hiện trước, tác vụ X2 thực hiện sau để kết thúc tác vụ X). S Semaphore kiểm soát truy xuất Buffer (S=1). Cấu trúc lại miền Găng (Critical-section construction) P1, P2 :

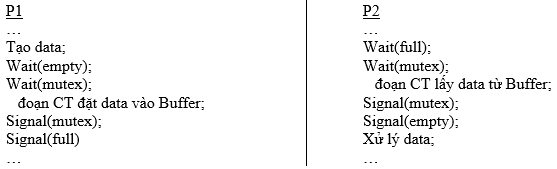
# 

### Khi tiến trình P2 kết thúc hoạt động, giá trị S = …

1. -1
2. 0
3. 1
4. Giá trị không xác định (undefined value).

**P1, P2 đang xung đột nên sẽ vô miền chờ. P1 thực hiện trước, sau khi gọi hàm wait(), s=0. P1 thực hiện, sau đó gọi hàm signal(), s=1.   
Tới lượt P2 thực hiện, gọi hàm wait(), s=0. Đoạn ct P2 thực hiện.**

# Cho 2 tiến trình P1, P2 cùng truy xuất Buffer hữu hạn (có n phần tử); P1 đặt data vào Buffer; P2 lấy data từ Buffer. Sử dụng 3 Semaphore: Mutex : KT truy xuất đồng thời P1, P2 (mutex=1); Full : số phần tử có data trong Buffer (full=0); Empty : số phần tử không có data trong Buffer (empty=n). Cấu trúc lại miền Găng (Critical-section construction) P1, P2 :

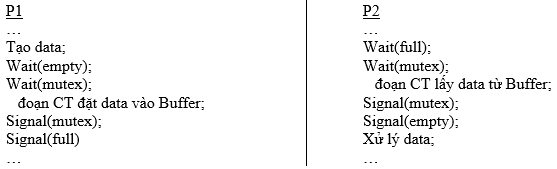
****

### Khi Buffer chứa đầy data (đầy), các giá trị mutex = …; full = …; empty = …

1. 1; 1; 1
2. 1; n; 0
3. 1; 0; n
4. 0; 0; 0

# 

# Cho 2 tiến trình P1, P2 cùng truy xuất Buffer hữu hạn (có n phần tử); P1 đặt data vào Buffer; P2 lấy data từ Buffer. Sử dụng 3 Semaphore: Mutex : KT truy xuất đồng thời P1, P2 (mutex=1); Full : số phần tử có data trong Buffer (full=0); Empty : số phần tử không có data trong Buffer (empty=n). Cấu trúc lại miền Găng (Critical-section reconstruction) P1, P2 :

****



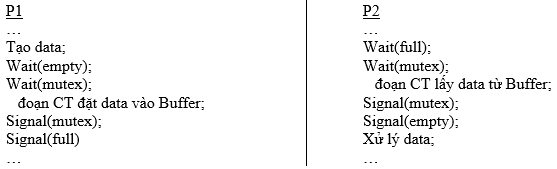
### Khi tiến trình P1 đang đặt data đầu tiên vào Buffer, các giá trị mutex = …; full = …; empty = … là.

1. 1; 1; 1
2. 1; n; 0
3. 0; 0; n-1
4. 1; 0; n

Qua hai wait trong P1 (wait: -, signal: +)

# 

# Cho 2 tiến trình P1, P2 cùng truy xuất Buffer hữu hạn (có n phần tử); P1 đặt data vào Buffer; P2 lấy data từ Buffer. Sử dụng 3 Semaphore: Mutex : KT truy xuất đồng thời P1, P2 (mutex=1); Full : số phần tử có data trong Buffer (full=0); Empty : số phần tử không có data trong Buffer (empty=n). Cấu trúc lại miền Găng (Critical-section reconstruction) P1, P2 :

****



### Khi tiến trình P2 lấy data cuối cùng từ Buffer, các giá trị mutex = …; full = …; empty = … là.

1. 1; 1; 1
2. 0; 0; n-1



1. 0; 1; n
2. 1; n; 0

# 

# Cấu trúc lại miền Găng (Critical-section reconstruction) P1, P2 sau đây có thể có Deadlock hay không ?

Semaphore A và B khởi đầu bằng 1.

P1 P2

Wait(A); Wait(B);

… …

Wait(B) wait(A)

… …

1. Có.
2. Không.

**Giải thích: Gỉa sử P1 thực hiện trước, sau khi P1 thực hiện xong:**

* **A=A-1=1-1=0**
* **B=B-1-1=0**

**Lúc này P2 sẽ không thực hiện được do luôn trong trạng thái chờ tài nguyên A và B từ P1.**

**Trường hợp tương tự cũng xảy ra nếu P2 thực hiện trước P1**

**\*Note: Deadlock xảy ra khi một tiến trình luôn trong trạng thái chờ vô tận tài nguyên từ một tiến trình khác.**

# 

# Đồ thị cấp phát tài nguyên (Resource allocation graph-RAG) sau đây có thể có Deadlock hay không ?

### °

1. Có.
2. Không.

\*\*Note: Nếu không có cycle thì chắc chắn không có deadlock, tuy nhiên, nếu có cycle thì có thể xảy ra deadlock hoặc KHÔNG.

# 

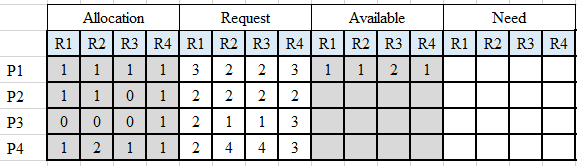
# Đồ thị cấp phát tài nguyên (Resource allocation graph-RAG) sau đây có thể có Deadlock hay không ?

### 

1. Có.
2. Không.

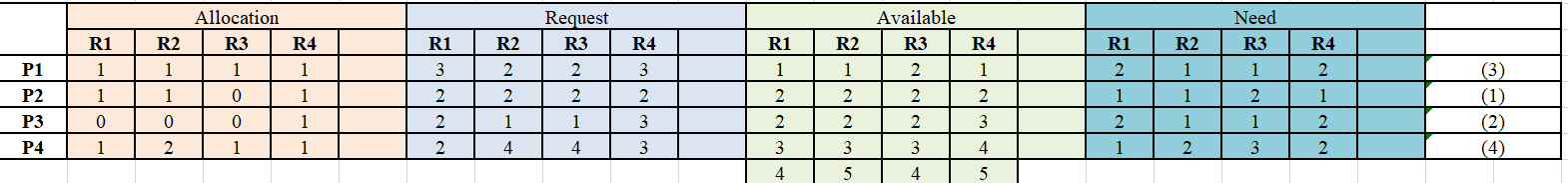
# 

# Cho bảng dữ liệu sau (giải thuật Banker-Banker’s algorithm).



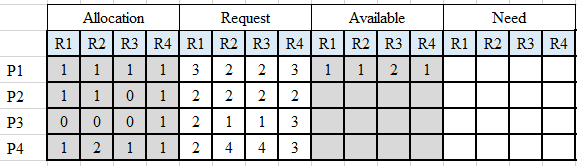
### Có chuỗi cấp phát tài nguyên cho các tiến trình là:

1. P1, P2, P3, P4
2. P4, P1, P2, P3
3. P2, P3, P1, P4
4. P2, P3, P1, \* (Unsafe allocation chain).



# 

# Cho bảng dữ liệu sau (giải thuật Banker-Banker’s algorithm).



### Tài nguyên ban đầu của hệ thống R1, R2, R3, R4 là:

1. 3, 4, 5, 6
2. 4, 5, 4, 5
3. 5, 5, 4, 4
4. 4, 3, 5, 7

# 

# Cho bảng dữ liệu sau:

|  |  |
| --- | --- |
| Tiến trình (Process) | Số đơn vị bộ nhớ yêu cầu  (Number of memory units required) |
| A | 3 |
| B | 5 |
| C | 2 |
| D | 2 |
| E | 3 |

Cấp phát bộ nhớ với kỹ thuật phân vùng động (Dynamic partitioning) với bộ nhớ có 15 đơn vị cấp phát. Tại thời điểm thu hồi vùng nhớ của tiến trình A có bao nhiêu vùng trống (Hole) được tạo ra. Sử dụng thuật toán cấp phát First-Fit. Chuỗi cấp phát như sau: A→B→C→thu hồi B→D→thu hồi A→E

1. 2.
2. 3.
3. 4.
4. 5.

Chart, bar chart

Description automatically generated

\*\*Note: các giải thuật cấp phát bộ nhớ:

**Text, letter

Description automatically generated**

**Text, letter

Description automatically generated**

# 

# Cho bảng dữ liệu sau:

|  |  |
| --- | --- |
| Tiến trình (Process) | Số đơn vị bộ nhớ yêu cầu  (Number of memory units required) |
| A | 3 |
| B | 5 |
| C | 2 |
| D | 2 |
| E | 3 |

Cấp phát bộ nhớ với kỹ thuật phân vùng động (Dynamic partitioning) với bộ nhớ có 15 đơn vị cấp phát. Danh sách vùng trống (Hole) tại thời điểm cấp phát vùng nhớ của tiến trình E?. Sử dụng thuật toán cấp phát First-Fit. Record tthay bằng H(x,y). Với chuỗi cấp phát như sau: A→B→C→thu hồi B→D→thu hồi A→E

|  |  |  |
| --- | --- | --- |
| H | x | y |

1. H(5,3); H(10,5).
2. H(5,5); H(5,5).
3. H(3,3); H(8,5)
4. H(4,3); H(9,5)

# 

Cấp phát bộ nhớ với kỹ thuật phân vùng động (Dynamic partitioning) với hiện trạng bộ nhớ như sau (mỗi ô là 1 đơn vị cấp phát):



Sử dụng giải thuật cấp phát Next-Fit. Cho biết record quản lý bộ nhớ của tiến trình E sau khi cấp phát cho D (3) và E(2). . Record tthay bằng D(x,y).

|  |  |  |
| --- | --- | --- |
| D | x | y |

1. D(3,2).
2. D(9,2).
3. E(15,2).
4. D(12,2).

Diagram

Description automatically generated

# 

Cấp phát bộ nhớ với kỹ thuật phân vùng động (Dynamic partitioning) với hiện trạng bộ nhớ như sau (mỗi ô là 1 đơn vị cấp phát):



Sử dụng giải thuật cấp phát Best-Fit. Cho biết record quản lý bộ nhớ của tiến trình D sau khi cấp phát cho D (3). . Record tthay bằng D(x,y).

|  |  |  |
| --- | --- | --- |
| D | x | y |

1. D(15,3).
2. D(9,3).
3. D(3,3).
4. D(12,3).

Diagram

Description automatically generated

# 

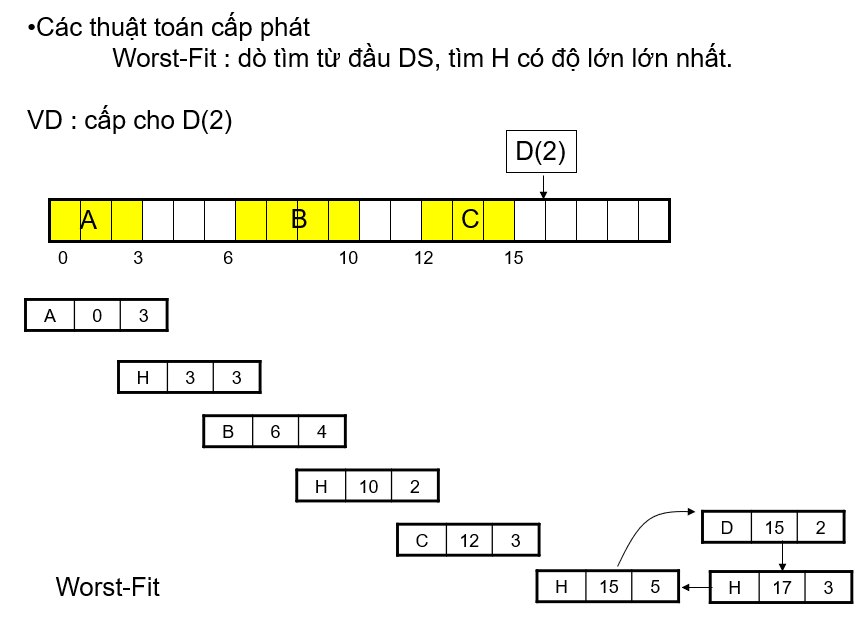
Cấp phát bộ nhớ với kỹ thuật phân vùng động (Dynamic partitioning) với hiện trạng bộ nhớ như sau (mỗi ô là 1 đơn vị cấp phát):



Sử dụng giải thuật cấp phát Worst-Fit. Cho biết record quản lý bộ nhớ của tiến trình D sau khi cấp phát cho D(2). Record thay bằng D(x,y).

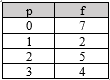
|  |  |  |
| --- | --- | --- |
| D | x | y |

1. D(15,2).
2. D(3,2).
3. D(9,2).
4. D(12,2).

****

# 

Trong cấp phát bộ nhớ với kỹ thuật phân trang (Paging). Cho kích thước trang (page) và kích thước khung trang (frame) là 100K, địa chỉ bắt đầu cấp phát trong bộ nhớ là 0K. Cho bảng trang (Page map table-PMT) của tiến trình P như sau :



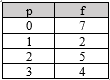
Địa chỉ vật lý (Physical Address) tương ứng với địa chỉ logic (Logical Address) là:

|  |  |
| --- | --- |
| 3 | 20K |

1. 520K.
2. 420K. (3 => f = 4 | Kq = 4\*100 + 20)
3. 220K.
4. 720K

# 

Trong cấp phát bộ nhớ với kỹ thuật phân trang (Paging). Cho kích thước trang (page) và kích thước khung trang (frame) là 100K, địa chỉ bắt đầu cấp phát trong bộ nhớ là 0K. Cho bảng trang (Page map table-PMT) của tiến trình P như sau :



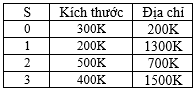
Với địa chỉ vật lý (Physical Address) 730K, địa chỉ logic (Logical Address) tương ứng là

|  |  |
| --- | --- |
| p | d |

1. p=0;d=30K.
2. p=1;d=30K.
3. p=2;d=30K.
4. p=3;d=30K.

# 

Trong cấp phát bộ nhớ với kỹ thuật phân đoạn (Segmentation). Địa chỉ bắt đầu cấp phát trong bộ nhớ là 0K. Cho bảng phân đoạn (Segmentation map table-SMT) của tiến trình P như sau :



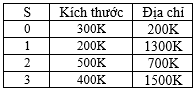
Địa chỉ vật lý (Physical Address) tương ứng với địa chỉ logic (Logical Address) là:

|  |  |
| --- | --- |
| 2 | 120K |

1. 320K.
2. 1420K.
3. 820K. (120 < 500 -> Hợp lệ | Kq = 700 + 120)
4. 1620K

# 

Trong cấp phát bộ nhớ với kỹ thuật phân đoạn (Segmentation). Địa chỉ bắt đầu cấp phát trong bộ nhớ là 0K. Cho bảng phân đoạn (Segmentation map table-SMT) của tiến trình P như sau :



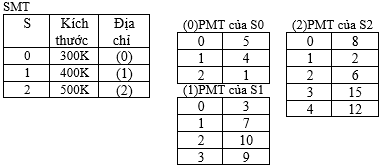
Với địa chỉ vật lý (Physical Address) 1600K, địa chỉ logic (Logical Address) tương ứng là

|  |  |
| --- | --- |
| s | d |

1. s=0;d=1400K.
2. s=1;d=300K.
3. s=2;d=900K.
4. s=3;d=100K.

# 

Trong cấp phát bộ nhớ với kỹ thuật phân đoạn kết hợp (Segmentation with paging). Địa chỉ bắt đầu cấp phát trong bộ nhớ là 0K. Cho bảng phân đoạn (Segmentation map table-SMT) và các bảng trang (Page map table-PMT) của tiến trình P như sau :



Địa chỉ vật lý (Physical Address) tương ứng với địa chỉ logic (Logical Address) là:

|  |  |
| --- | --- |
| 1 | 120K |

1. 320K.
2. 720K. (1 -> s1 -> (1) | 120 -> p =1 -> f = 7 | KQ = 7\*100 + 20)
3. 1020K.
4. 920K

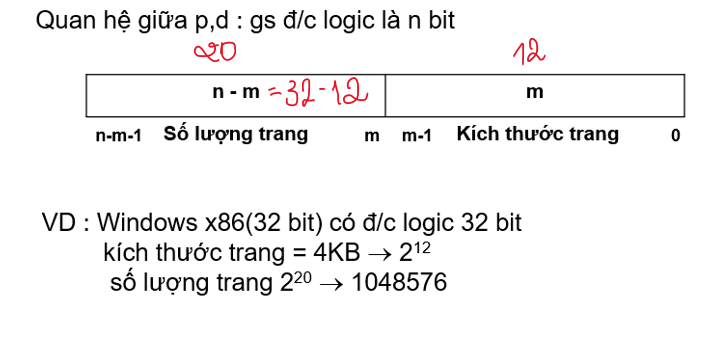
# 

Trong cấp phát bộ nhớ với kỹ thuật phân trang (Paging). Địa chỉ logic (Logical Address) n bít có dạng:



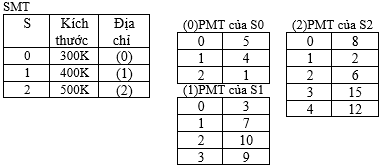
Nếu chọn m=12 sẽ có kích thước trang 4KB và số lượng trang là:

1. 1048576.
2. 524288.
3. 2097152.
4. 262144.

****

# 

Trong cấp phát bộ nhớ với kỹ thuật phân đoạn kết hợp (Segmentation with paging). Địa chỉ bắt đầu cấp phát trong bộ nhớ là 0K. Cho bảng phân đoạn (Segmentation map table-SMT) và các bảng trang (Page map table-PMT) của tiến trình P như sau :



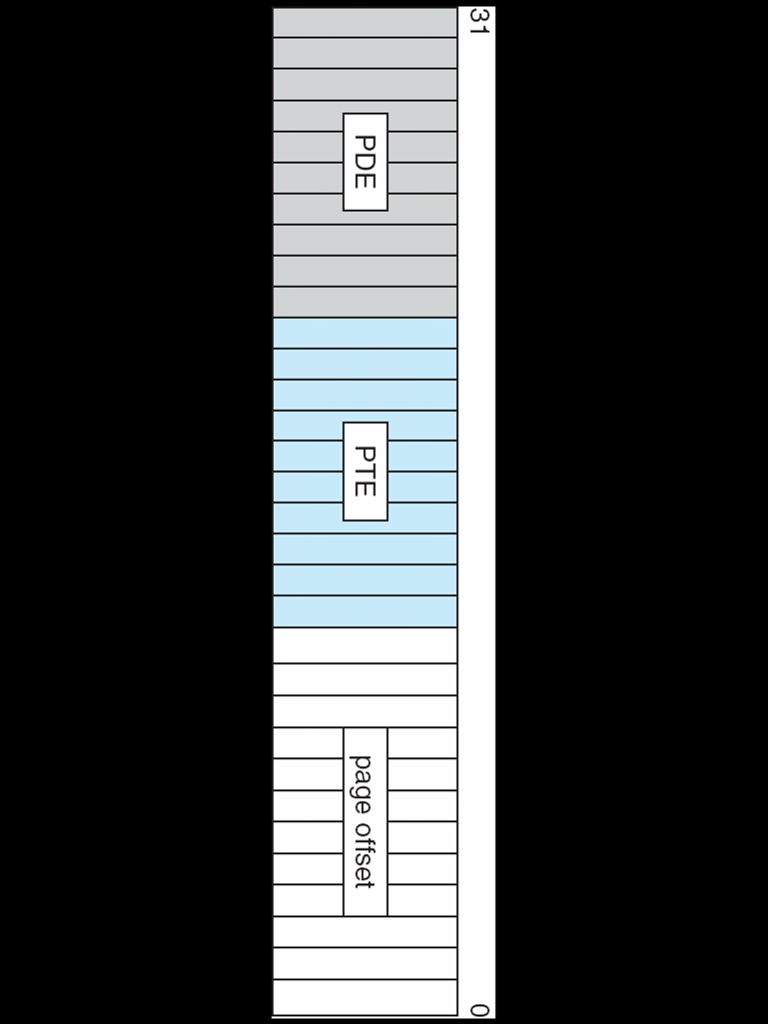
Với địa chỉ vật lý (Physical Address) 1520K, địa chỉ logic (Logical Address) tương ứng là

|  |  |
| --- | --- |
| s | d |

1. s=0;d=120K.
2. s=1;d=220K.
3. s=2;d=320K.
4. s=3;d=420K.

# (phân vân)

Trong Windows x86 tổ chức bảng trang 2 cấp có cấu trúc sau:



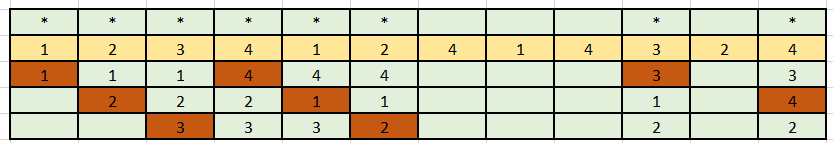
Số lần tìm kiếm tối đa để tìm thấy 1 trang bất ký là:

1. 1024.
2. 2048.
3. 4096.
4. 1000.

# 

Sử dụng thật toán thay thế trang FIFO (First In First Out-FIFO page replacement Algorithm) trên chuỗi trang (Reference series) 1, 2, 3, 4, 1, 2, 4, 1, 4, 3, 2, 4 với tổng số khung trang trống là 3. Số lỗi trang (Page faults) = …

1. 12.
2. 7.
3. 8.
4. 10.

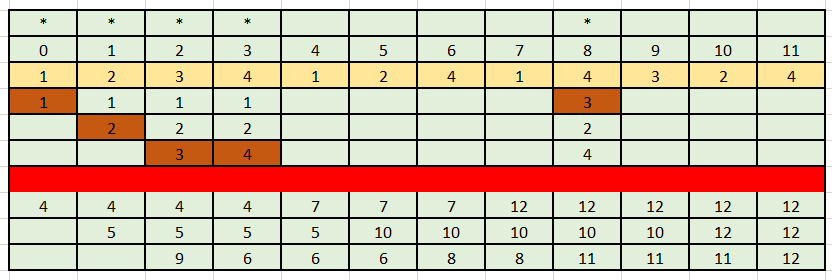


Thay trang theo thứ tự từ trên xuống (Vào trước ra trước)

# 

Sử dụng thật toán thay thế trang Tối ưu (Optimal-OPT page replacement Algorithm) trên chuỗi trang (Reference series) 1, 2, 3, 4, 1, 2, 4, 1, 4, 3, 2, 4 với tổng số khung trang trống là 3. Số lỗi trang (Page faults) = …

1. 10.
2. 7.
3. 6.
4. 5.

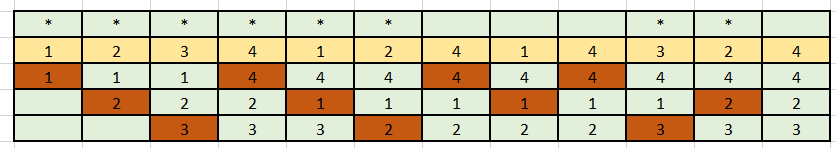


**Giải thích: Trang không nằm trong hàng đợi hiện tại sẽ thay thế vào trang mà có lần xuất hiện tiếp theo xa nhất (counter lớn nhất), lúc này lỗi trang sẽ xuất hiện (\*)**

# 

Sử dụng thật toán thay thế trang LRU (Least Recently Used-LRU page replacement Algorithm) trên chuỗi trang (Reference series) 1, 2, 3, 4, 1, 2, 4, 1, 4, 3, 2, 4 với tổng số khung trang trống là 3. Số lỗi trang (Page faults) = …

1. 8.
2. 11.
3. 9.
4. 7.



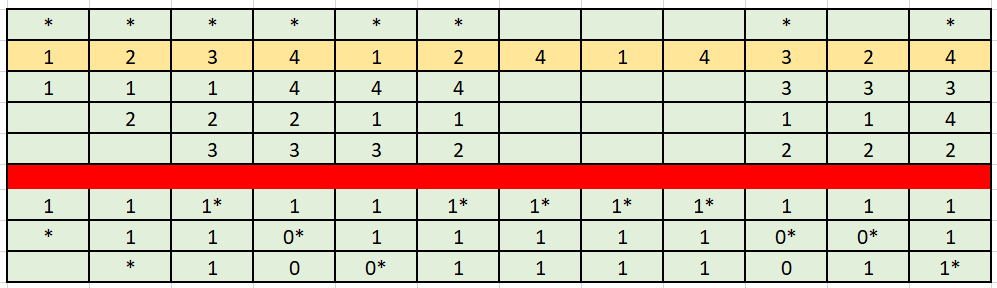
**Giải thích: thay cho thằng nào lâu rồi chưa thay**

# 

Sử dụng thật toán thay thế trang CLOCK (CLOCK page replacement Algorithm) trên chuỗi trang (Reference series) 1, 2, 3, 4, 1, 2, 4, 1, 4, 3, 2, 4 với tổng số khung trang trống là 3. Số lỗi trang (Page faults) = …

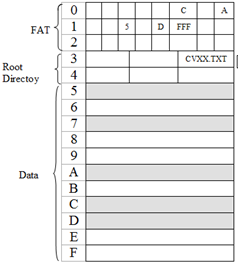
(Bit trạng thái =1 cho tất cả các trường hợp. Con trỏ không di chuyển khi truy xuất trang)

1. 10.
2. 9.
3. 8.
4. 7.



**Giải thích: khi gặp phần tử trùng thì trỏ không nhảy, nếu là 1 thì giữ nguyên, là 0 thì đổi thành 1. Nếu như thay trang là 1 thì tại đó đổi thành 0 và con trỏ di chuyển xuống, khi thay xong con trỏ nhảy tiếp (giống như nhảy 2 lần).**

Cho bảng dữ liệu FAT12 như sau:



Một tập tin có FAT bắt đầu là 7. Chuỗi FAT của tập tin là:

1. 7→A→C→D→5→8→9.
2. 7→A→5→C→D→8→9.
3. 7→ 5→C→D→8→9.→ A
4. 7→C→A→5→D.

**Giải thích: chữ A(10) ở ô mang giá trị trước nó (7). Tương tự cho các thằng sau.**

Cấu trúc đĩa cứng dạng MBR Partition có tối đa … Primary partition.

1. 1.
2. 2.
3. 3.
4. 4.

Cấu trúc đĩa cứng dạng GPT Partition có tối đa … Primary partition.

1. 1.
2. 4.
3. 64.
4. 128.

Kích thước tập tin lớn nhất lưu trữ trong FAT32 là:

1. 4 GB.
2. 3 GB.
3. 2 GB.
4. 1 GB.

LFN-Long File Name có trong định dạng:

1. FAT16.s
2. FAT32.
3. NTFS.
4. EXT2/3.