



**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**  
**ĐẠI HỌC KHOA HỌC TỰ NHIÊN**  
**KHOA CÔNG NGHỆ THÔNG TIN**

# **BÁO CÁO ĐỒ ÁN**

---

**MÔN HỌC: HỆ ĐIỀU HÀNH – CQ2019/4**

**GIAO VIÊN HƯỚNG DẪN: THÁI HÙNG VĂN – ĐẶNG TRẦN MINH HẬU**

**THỰC HIỆN:**

**NGUYỄN GIA VĨ – 18120260**

**LÊ ĐỨC THIÊN – 19120664**

**TRẦN VĂN TÌNH - 19120686**

## A. THÔNG TIN CHUNG:

### 1. Thông tin bài làm:

- Ngôn ngữ lập trình: Python, C++
- Môi trường lập trình: Visual Studio

### 2. Thành viên và Phân công:

Họ và Tên	MSSV	Phân công	Mức độ đóng góp
Nguyễn Gia Vĩ	18120260	Câu 1	100%
Lê Đức Thiện	19120664	Câu 2 (Parent)	100%
Trần Văn Tình	19120686	Câu 2 (Children)	100%

### 3. Bảng tự đánh giá chương trình:

#### **Đánh giá mức độ hoàn thành ở chương trình Children:**

Tính năng đã hoàn thành	Tự đánh giá	Hạn chế
Khởi động cùng windows	90%	Chỉ mới thích ứng trên máy lập trình chưa scale trên các máy khác
Phụ huynh đăng nhập	100%	
Lập trình song song (Kiểm tra 15s rồi tắt) và (nhập mật khẩu lại từ đầu)	100%	
Mật khẩu sai 3 lần thì thoát	100%	
Lập trình song song 3 tiến trình: Keyboard Listen, Đọc file Time, Check Remaining Time, Screen Capture	95%	KeyboardListen: Chỉ đọc được tiếng anh
Chặn rơi vào critical section	90%	Cài đặt còn đơn giản, chưa hay nhưng đã hoàn thành
Đã test chương trình thành công		

#### **Đánh giá mức độ hoàn thành ở chương trình Parent:**

Tính năng	Tự đánh giá	Hạn chế
Điều chỉnh khung giờ (time.txt)	95%	
Xem lịch sử (log.txt)	90%	Chỉ giám sát được phím trong thời điểm quá khứ -> hiện tại chưa xem được ở tương lai

#### 4. Tham khảo:

##### Câu 1:

- [1] [https://en.wikipedia.org/wiki/Virtual\\_memory](https://en.wikipedia.org/wiki/Virtual_memory)
- [2] <http://www.zun.vn/tai-lieu/he-dieu-hanh-ky-thuat-phan-trang-42393/>
- [3] <https://www.geeksforgeeks.org/paging-in-operating-system/>
- [4] <https://answers.microsoft.com/en-us/windows/forum/all/physical-and-virtual-memory-in-windows-10/e36fb5bc-9ac8-49af-951c-e7d39b979938>
- [5] <https://stackoverflow.com/questions/24358105/do-modern-oss-use-paging-and-segmentation>
- [6] [https://en.wikipedia.org/wiki/Memory\\_paging](https://en.wikipedia.org/wiki/Memory_paging)
- [7] [https://en.wikipedia.org/wiki/Page\\_%28computer\\_memory%29#Windows-based\\_operating\\_systems](https://en.wikipedia.org/wiki/Page_%28computer_memory%29#Windows-based_operating_systems)
- [8] <https://www.quora.com/How-can-we-find-the-maximum-number-of-pages-per-process>

##### Câu 2:

- [1] [https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/7\\_Deadlocks.html](https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/7_Deadlocks.html)
- [2] <https://www.tutorialspoint.com/critical-section-problem#:~:text=The%20critical%20section%20is%20a,execute%20in%20their%20critical%20sections.>
- [3] <https://www.geeksforgeeks.org/g-fact-70/>
- [4] <https://datatofish.com/screenshot-python/>
- [5] <https://support.microsoft.com/en-us/office/sync-onedrive-files-and-folders-3b8246e0-cc3c-4ae7-b4e1-4b4b37d27f68>

## B. CÂU 1:

### I. BỘ NHỚ ẢO

*Bộ nhớ ảo* là một kỹ thuật quản lý bộ nhớ với mục đích cung cấp tài nguyên lưu trữ của máy tính ở mức trừu tượng cao.

Hệ điều hành sẽ sử dụng cả phần cứng và phần mềm để liên kết địa chỉ sử dụng bởi chương trình (gọi là địa chỉ ảo) thành địa chỉ vật lý trong bộ nhớ máy tính. Đối với process, bộ nhớ chính sẽ là một không gian địa chỉ liên kề hoặc một tập các segment (đoạn) liên kề. Hệ điều hành sẽ quản lý không gian địa chỉ ảo và chỉ định bộ nhớ chính cho các bộ nhớ ảo này. Trong CPU có một phần cứng giúp chuyển đổi địa chỉ, thường gọi là đơn vị quản lý bộ nhớ (MMU - memory management unit), sẽ tự động thực hiện việc chuyển đổi địa chỉ ảo sang địa chỉ thực này.

Bộ nhớ ảo giúp việc lập trình dễ dàng hơn nhờ đặc tính sau:

- Che giấu bộ nhớ vật lý bên dưới, giao việc quản lý bộ nhớ cho nhân của hệ thống
- Loại bỏ yêu cầu phải tái xác định địa chỉ của chương trình hay truy cập bộ nhớ từ địa chỉ tương đối, khi mà mỗi process đều có một không gian địa chỉ riêng.

Lợi ích của bộ nhớ ảo:

- Giảm tải gánh nặng phải quản lý bộ nhớ vật lý, thường phức tạp do phải dùng chung giữa nhiều process, từ đó dễ dàng chia sẻ bộ nhớ, tăng cường bảo mật nhờ vào việc cô lập bộ nhớ
- Có thể sử dụng nhiều bộ nhớ hơn bộ nhớ thực tế của máy tính bằng việc tận dụng ổ đĩa lưu trữ và sử dụng các kỹ thuật như phân trang hay phân đoạn (kỹ thuật quản lý bộ nhớ không liên tục nói chung).

### II. KỸ THUẬT PHÂN TRANG

*Phân trang* (paging) là mô hình quản lý bộ nhớ. Kỹ thuật phân trang cho phép không gian địa chỉ vật lý (physical address space) của một process không cần liên tục.

Trong kỹ thuật phân trang:

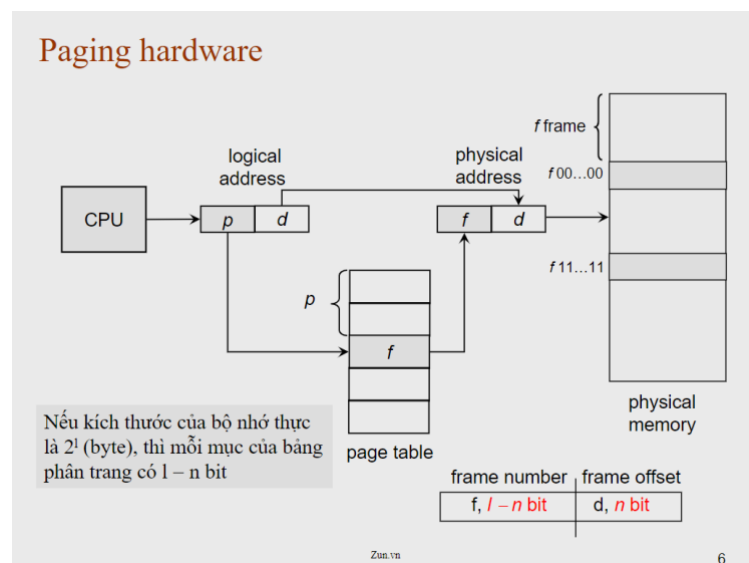
- Không gian địa chỉ của 1 process được chia thành những khối có cùng kích thước, gọi là trang (page).
- Không gian địa chỉ vật lý được chia thành các khối cố định và có kích thước bằng nhau gọi là frame (tương ứng với trang). Thông thường kích thước của frame là lũy thừa của 2.
- Frame và trang nhớ có kích thước bằng nhau.
- Hệ điều hành phải thiết lập một bảng phân trang (page table) để ánh xạ địa chỉ ảo thành địa chỉ thực. Mỗi process có một bảng phân trang, được quản lý qua một con trỏ lưu giữ trong PCB.

## 1. Chuyển đổi địa chỉ

Địa chỉ logic: <page number, offset>

Địa chỉ vật lý: <frame number, offset>

- Page number: được dùng làm chỉ mục vào bảng phân trang. Mỗi mục (entry) trong bảng phân trang chứa chỉ số frame (còn gọi là số frame cho gọn) của trang tương ứng trong bộ nhớ thực.
- Page offset: được kết hợp với địa chỉ nền (base address) của frame để định vị địa chỉ thực



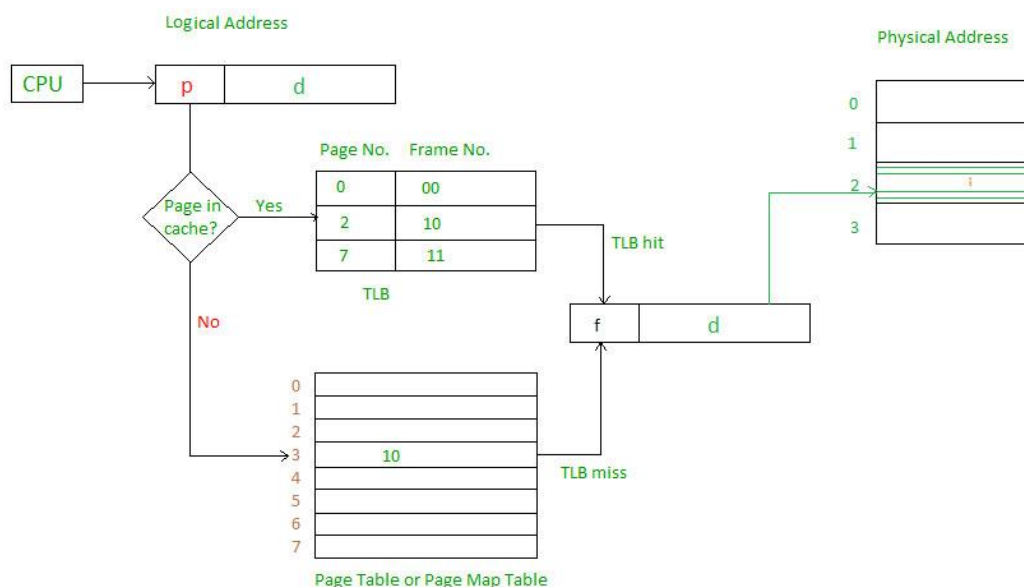
Hình 1: Minh họa cách chuyển đổi địa chỉ trong paging (nguồn: Zun.vn)

## 2. Lưu trữ bảng phân trang

- Bảng phân trang thường được lưu giữ trong bộ nhớ chính
  - Mỗi process được hệ điều hành cấp một bảng phân trang
  - Thanh ghi page-table base (PTBR) trỏ đến bảng phân trang
  - Thanh ghi page-table length (PTLR) biểu thị kích thước của bảng phân trang (có thể được dùng trong cơ chế bảo vệ bộ nhớ)
- Mỗi truy cập dữ liệu/lệnh cần hai thao tác truy xuất vùng nhớ
  - Dùng page number  $p$  làm index để truy cập mục trong bảng phân trang nhằm lấy số frame, và kế đến là dùng page offset  $d$  để truy xuất dữ liệu/lệnh trong frame
- Thường dùng một cache phần cứng có tốc độ truy xuất và tìm kiếm cao, gọi là thanh ghi kết hợp (associative register) hoặc translation look-aside buffers (TLBs)  
TLB, là thanh ghi hỗ trợ tìm kiếm truy xuất dữ liệu với tốc độ cực nhanh.

### 3. Ánh xạ trang với TLB

- Nếu page number nằm trong TLB (: hit, trúng)  $\Rightarrow$  lấy ngay được số frame  $\Rightarrow$  tiết kiệm được thời gian truy cập bộ nhớ chính để lấy số frame trong bảng phân trang.
- Ngược lại (: miss, trật), phải lấy số frame từ bảng phân trang như bình thường.



Hình 2: Minh họa quy trình chuyển đổi địa chỉ khi có sử dụng TLB (nguồn: geeksforgeeks.org)

### 4. So sánh thời gian truy xuất

- Tính thời gian truy xuất hiệu dụng (EAT)
- Thời gian tìm kiếm trong TLB (associative lookup):  $\epsilon$
- Thời gian một chu kỳ truy xuất bộ nhớ:  $x$
- Hit ratio: tỉ số giữa số lần chỉ số trang được tìm thấy (hit) trong TLB và số lần truy xuất khởi nguồn từ CPU,
  - Kí hiệu hit ratio:  $\alpha$
- Thời gian cần thiết để có được địa chỉ thực
  - Khi chỉ số trang có trong TLB (hit) :  $\epsilon + x$
  - Khi chỉ số trang không có trong TLB (miss):  $\epsilon + x + x$
- Thời gian truy xuất hiệu dụng

$$\begin{aligned} \text{EAT} &= (\epsilon + x)\alpha + (\epsilon + 2x)(1 - \alpha) \\ &= (2 - \alpha)x + \epsilon \end{aligned}$$

- Giả sử (đơn vị thời gian: nano giây)
  - Associative lookup = 20
  - Memory access = 100

### ■ Ví dụ 1

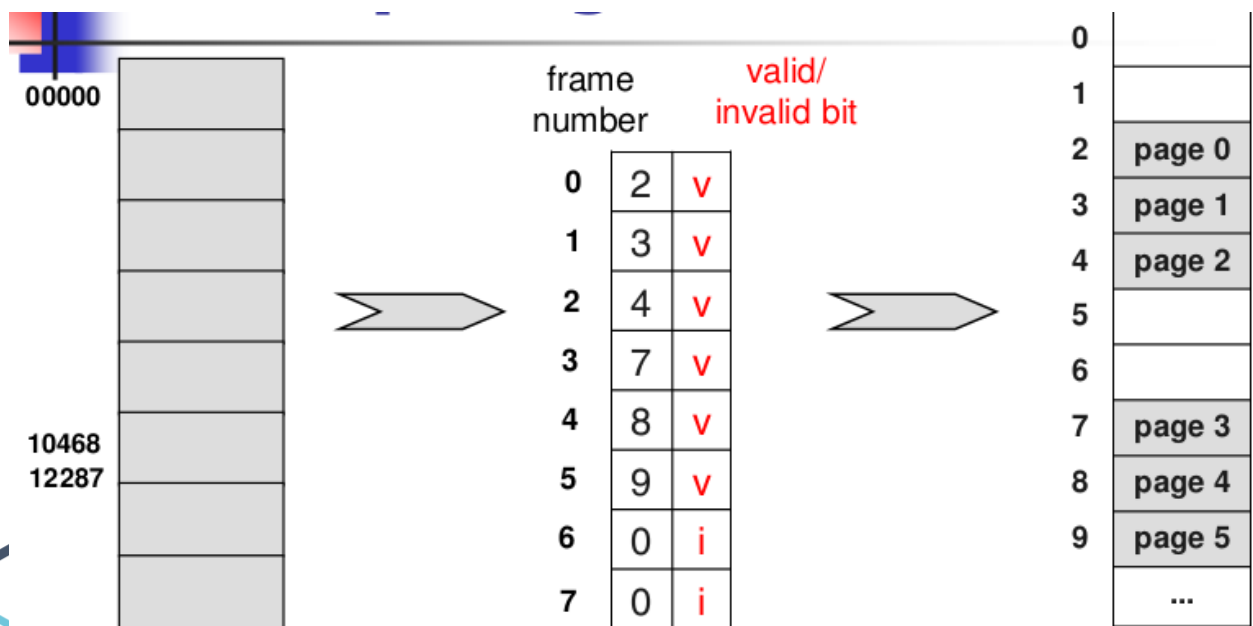
- Hit ratio = 0,8
- EAT =  $(100 + 20) \times 0,8$   
+  
 $(200 + 20) \times 0,2$   
=  $1,2 \times 100 + 20$   
= 140

### ■ Ví dụ 2

- Hit ratio = 0,98
- EAT =  $(100 + 20) \times 0,98$   
+  
 $(200 + 20) \times 0,02$   
=  $1,02 \times 100 + 20$   
= 122

## 5. Bảo vệ bộ nhớ

- Việc bảo vệ bộ nhớ được hiện thực bằng cách gắn với frame các bit bảo vệ (protection bit) được giữ trong bảng phân trang. Các bit này biểu thị các thuộc tính sau
  - read-only, read-write, execute-only
- Ngoài ra, còn có một valid/invalid bit gắn với mỗi mục trong bảng phân trang
  - "valid": cho biết là trang của process, do đó là một trang hợp lệ.
  - "invalid": cho biết là trang không của process, do đó là một trang bất hợp lệ.



Bảo vệ bằng valid/invalid bit

- Mỗi trang nhớ có kích thước  $2K = 2048$
- Process có kích thước 10.468  $\Rightarrow$  phân mảnh nội ở frame 9 (chứa page 5), các địa chỉ ảo  $> 12287$  là các địa chỉ invalid.
- Dùng PTLR để kiểm tra truy xuất đến bảng phân trang có nằm trong bảng hay không.

## 6. Bảng phân trang 2 mức

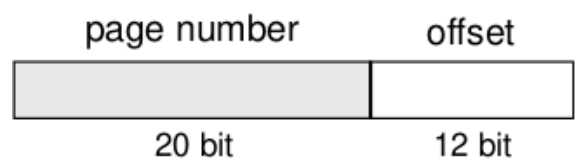
- Các hệ thống hiện đại đều hỗ trợ không gian địa chỉ ảo rất lớn ( $2^{32}$  đến  $2^{64}$ ), ở đây giả sử là  $2^{32}$ 
  - Giả sử kích thước trang nhớ là 4 KB ( $= 2^{12}$ )  $\Rightarrow$  bảng phân trang sẽ có  $2^{32} / 2^{12} = 2^{20} = 1$  M mục.
  - Giả sử mỗi mục gồm 4 byte thì mỗi process cần 4 MB cho bảng phân trang
- Thay vì dùng một bảng phân trang duy nhất cho mỗi process, "paging" bảng phân trang này, và chỉ giữ những bảng phân trang cần thiết trong bộ nhớ  $\Rightarrow$  bảng phân trang 2 mức (two-level page table).

### Ví dụ:

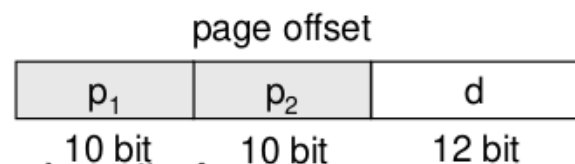
- Một địa chỉ luận lý trên hệ thống 32-bit với trang nhớ 4K được chia thành các phần sau:
  - Page number: 20 bit, Nếu mỗi mục dài 4 byte  $\Rightarrow$  Cần  $2^{20} \cdot 4$  byte = 4 MB cho mỗi page table
- Bảng phân trang cũng được chia nhỏ nên page number cũng được chia nhỏ thành 2 phần:

- 10-bit page number
- 10-bit page offset

Vì vậy, một địa chỉ luận lý sẽ như hình vẽ bên

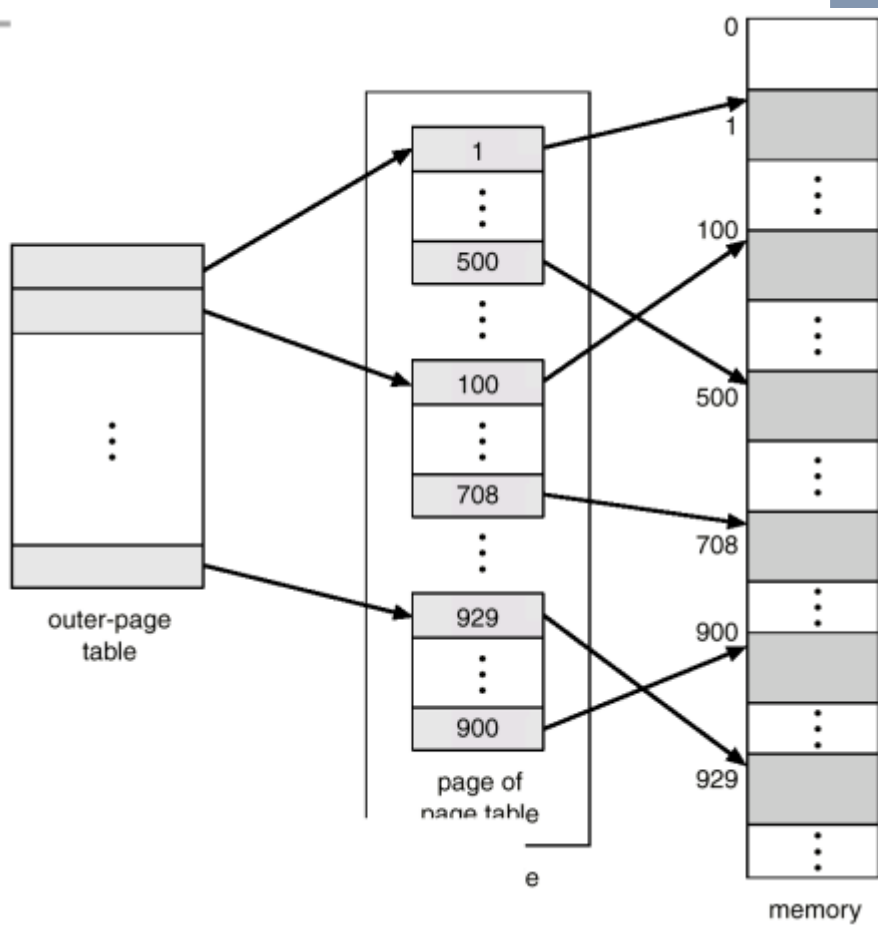


- $p_1$  : chỉ số của mục trong bảng phân trang mức 1 (outer-page table)
- $p_2$  : chỉ số của mục trong bảng phân trang mức 2

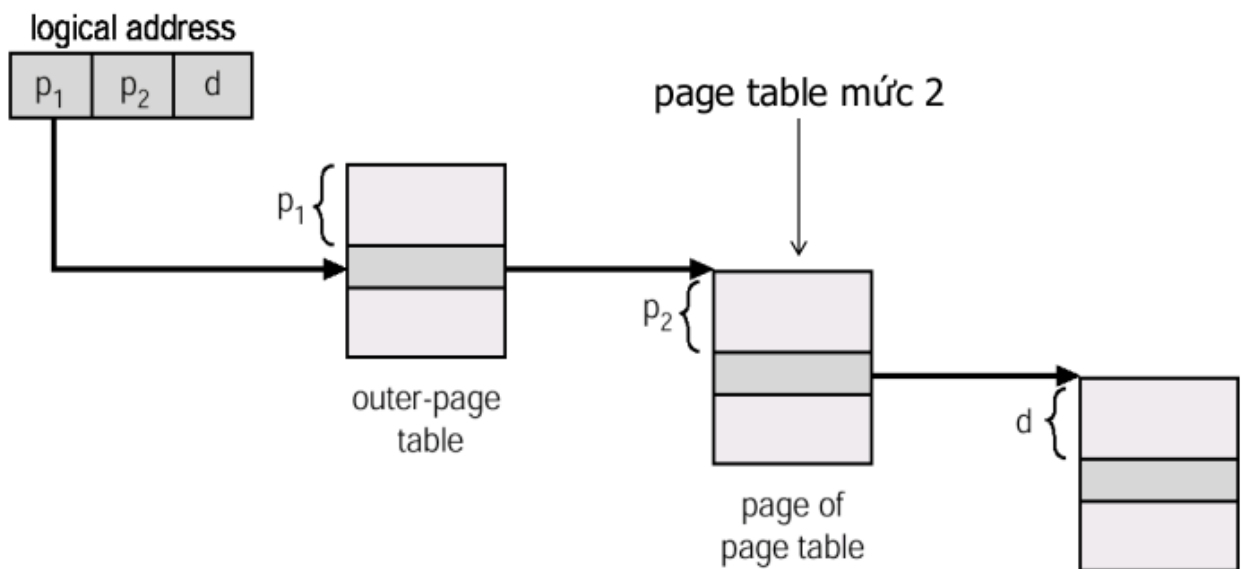


- Có  $2^{p_1}$  mục trong bảng phân trang mức 1
- Mỗi bảng phân trang mức 2 chứa  $2^{p_2}$  mục



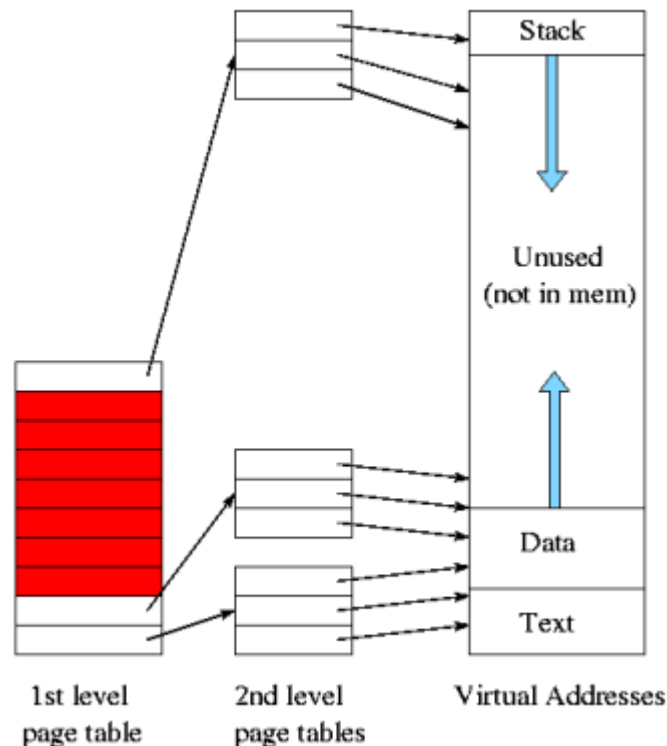


các bảng phân trang mức 2



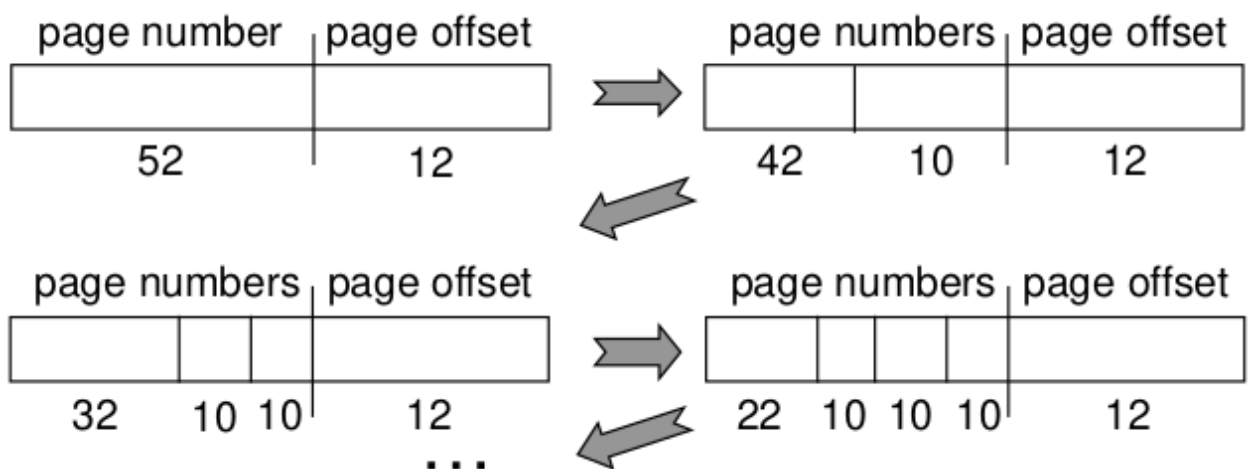
Sơ đồ chuyển đổi địa chỉ (address-translation scheme) cho kỹ thuật phân trang 2 mức, với 32-bit địa chỉ

- Bảng phân trang 2 mức giúp tiết kiệm bộ nhớ: Vùng màu đỏ tương ứng với phần chưa được sử dụng của không gian địa chỉ ảo. Các mục màu đỏ được đánh dấu là không có frame nên sẽ gây ra page fault nếu được tham chiếu đến; khi đó OS sẽ tạo thêm bảng phân trang mức 2 mới.



## 7. Bảng phân trang đa mức

- Ví dụ: Không gian địa chỉ luận lý 64-bit với trang nhớ 4K. Bảng phân trang 2-mức vẫn còn quá lớn! Tương tự bảng phân trang 2 mức, ta có bảng phân trang 3, 4,..., n mức



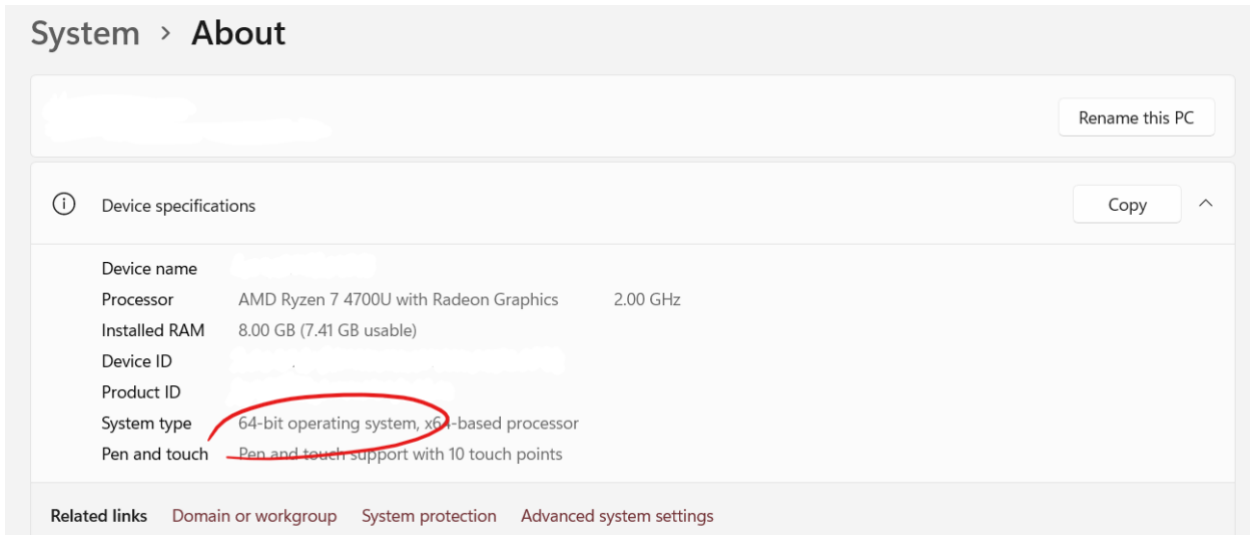
- Tiết kiệm chỗ trong bộ nhớ chính bằng cách chỉ giữ trong bộ nhớ chính các bảng phân trang mà process hiện đang cần.

### III. KHẢO SÁT HỆ THỐNG MÁY TÍNH THỰC TẾ

Đối tượng khảo sát: laptop của sinh viên sử dụng hệ điều hành Windows 11

#### 1. Tra cứu các thông tin

Truy cập About your PC



=> Hệ điều hành 64-bit -> Địa chỉ được lưu trữ bằng 64-bit

Tra cứu kiến trúc của CPU


<b>64 Bit</b>	64 Bit support
<b>Architecture</b>	x86 ←
<b>Announcement Date</b>	01/07/2020 = 733 days old
<b>Product Link (external)</b>	AMD Renoir (Ryzen 4000 APU) R7 4700U

Architecture	Smallest page size	Larger page sizes
32-bit x86 <sup>[17]</sup>	4 KiB	4 MiB in PSE mode, 2 MiB in PAE mode <sup>[18]</sup>
x86-64 <sup>[17]</sup>	4 KiB	2 MiB, 1 GiB (only when the CPU has PDPE1GB flag)
IA-64 (Itanium) <sup>[19]</sup>	4 KiB	8 KiB, 64 KiB, 256 KiB, 1 MiB, 4 MiB, 16 MiB, 256 MiB <sup>[18]</sup>

Hình 3: Nguồn wikipedia

=> Kích thước trang tối thiểu 4KB

System Information > System Summary



Installed Physical Memory (RAM)	8.00 GB
Total Physical Memory	7.41 GB
Available Physical Memory	863 MB
Total Virtual Memory	25.4 GB
Available Virtual Memory	12.1 GB
Page File Space	18.0 GB
Page File	C:\pagefile.sys
Kernel DMA Protection	Off
Virtualization-based security	Running
Virtualization-based security ...	
Virtualization-based security ...	Base Virtualization Support, Secure Boot, DMA Protection, UEFI Code Read...

## 2. Tổng kết:

Số bit quản lý địa chỉ ô nhớ: 64 bit

Kích thước trang (tối thiểu):  $4 \text{ KB} = 2^{12} \text{ Byte}$

⇒ Số bit tối thiểu để quản lý các offset trong trang: 12 bits

⇒ Số bit biểu diễn phân mục trang:  $64 - 12 = 52 \text{ bit}$

Kích thước phần bộ nhớ ảo: 25.4GB (mặc dù page file size là 18GB và RAM size là 8GB nhưng RAM đã có 1 phần dung lượng nhỏ dành ra cho hệ thống và process của mình không được sử dụng nên chỉ tính đúng con số là 25.4GB như đã tra cứu)

Số khung trang vật lý:  $25.4\text{GB} / 4\text{KB} = 6818260582$

Số khung trang logic tối đa trên không gian tiến trình: 6818260582 (mặc dù lúc lập trình thì không gian địa chỉ ảo không bị giới hạn bởi bộ nhớ vật lý nhưng khi sử dụng thì số khung sẽ bị hệ điều hành hạn chế để không vượt quá kích thước vật lý)

## C. CÂU 2:

### I. THIẾT KẾ CHÍNH ĐỂ TRÁNH CÁC PHẦN MỀM, TIẾN TRÌNH RƠI VÀO CRITICAL SECTION

**Ý tưởng:** Dùng một phương thức (protocol) cho các chương trình tham gia sao cho chúng không cùng lúc đi vào Critical Section

**Lý do:** Do các chương trình đều có chung nguồn tài nguyên file time.txt để thể hiện các khoảng thời gian được dùng máy tính

Cách thiết kế: Ta sẽ tạo ra một file chung Global.txt đây có thể là file ẩn mà các tiến trình sẽ phải read and write. Nội dung của file sẽ là

```
*Gloabl.txt - Notepad
File Edit Format View Help
C P1 P2 P3 P4
Using: C
TimeEnd: 07:30:15|
```

**Qui ước:** Tất cả các chương trình P(Parent) được sử dụng file time.txt 15s riêng chương trình C (Children) được sử dụng 30s

#### **Giải thích file:**

- + Dòng đầu tiên sẽ là các tiến trình đang tham gia theo thứ tự chương trình đứng đầu là chương trình vào trước.
- + Using: Là chương trình đang sử dụng
- + TimeEnd: Là thời gian mà tiến trình đó sẽ ngừng sử dụng file time.txt  
Thời gian end được tính bằng cách lấy thời gian vào cộng với khoảng thời gian sử dụng qui ước

#### **Phân tích cách hoạt động:**

Nếu các tiến trình có yêu cầu tài nguyên thì sẽ được thêm vào cuối dòng 1:

Vd: C P1 P2 P3

Nếu không còn yêu cầu tài nguyên sẽ tự động rời và xóa mình ra khỏi dòng 1:

Vd: C P1 P3

Khi một tiến trình đọc file Global.txt mà Using trống (none) thì có nghĩa là không có tiến trình nào đang dùng file time.txt ( tài nguyên chung). Sau đó tự thêm tên tiến trình mình vào và thời gian kết thúc

Vd: Using C      TimeEnd: 07:30:15

Tất cả các tiến trình đều đọc liên tục file Global.txt ;

Do Dòng 1 được sắp xếp theo thứ tự vào trước sau đồng thời ta cũng có dòng Using thể hiện tiến trình đang dùng sẽ giúp ta dễ dàng trong việc phân phối như một Stack

- Nếu tiến trình đang dùng là C thì sau đó sẽ suy ra tiến trình tiếp theo được dùng là P1
- Tương tự: C->P1->P2->C->P1->P2->C->....

Vậy chỉ cần nhìn vào Using ta sẽ ngẫm định được thứ tự tiến trình tiếp theo sẽ phải chạy và sử dụng file time.txt

Nếu các tiến trình đều hoạt động theo phương thức trên sẽ không xảy ra việc hai tiến trình cùng lúc vào **critical section** file time.txt

## II. THAY ĐỔI ĐỂ DỄ DÀNG CÀI ĐẶT (CODE)

**Nhận xét:** Với ý tưởng như trên ta cần một biến global để các chương trình tự điều phối thời gian và sử dụng file time.txt hợp lý. Tuy nhiên do khó khăn trong việc lập trình cài đặt nhóm em đã có giải pháp như sau

**Solution:** Thay vì phải tạo ra file Global.txt là cổng giao tiếp chung của các chương trình. Ta sẽ dùng một biến global chung đó chính là "thời gian hiện tại" điều này có nghĩa với các máy tính khác nhau đều có thể sử dụng một biến chung đó là "thời gian"

Vd: Máy tính 1: 07:30:00 (ICT)

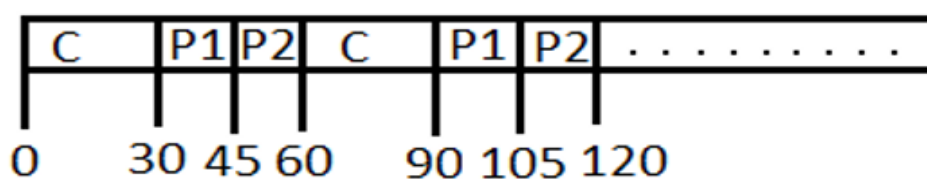
Máy tính 2: 07:30:00 (ICT)

Vì đây là thời gian nên các máy tính khi dùng sẽ có cùng một thời gian xấp xỉ nhau sai số rất ít. Đây sẽ là "đặc điểm" để cài đặt các tiến trình tránh đi vào critical section (file time.txt)

Thiết kế dưới đây để phù hợp cho việc code trong phạm vi nhỏ:

Tiến trình: C (Children), P1 (Parent1), P2 (Parent2)

Là 3 chương trình ta sẽ qui ước cho mỗi chương trình thời gian sử dụng tài nguyên chung (file time.txt) như sau:



Trong vòng 1 phút: 30s đầu cho chương trình C, 15s sau cho P1, 15s cuối cho P2

Vậy với một thời gian nhất định chỉ có duy nhất một tiến trình sử dụng được file

### Phân tích cách hoạt động:

+ Qui ước thời gian hoạt động bằng toán học

Giây  $[0, 30)$  tiến trình C

Giây  $[30, 45)$  tiến trình P1

Giây  $[45, 59.99)$  tiến trình P2

+ Mỗi chương trình sẽ đọc thời gian hiện tại liên tục nếu nằm trong thời gian được sử dụng sẽ sử dụng:

Vd: 06:02:41 là giây 41 sẽ là tiến trình P1 đang đọc file time.txt tất cả các file khác tạm dừng việc sử dụng file này

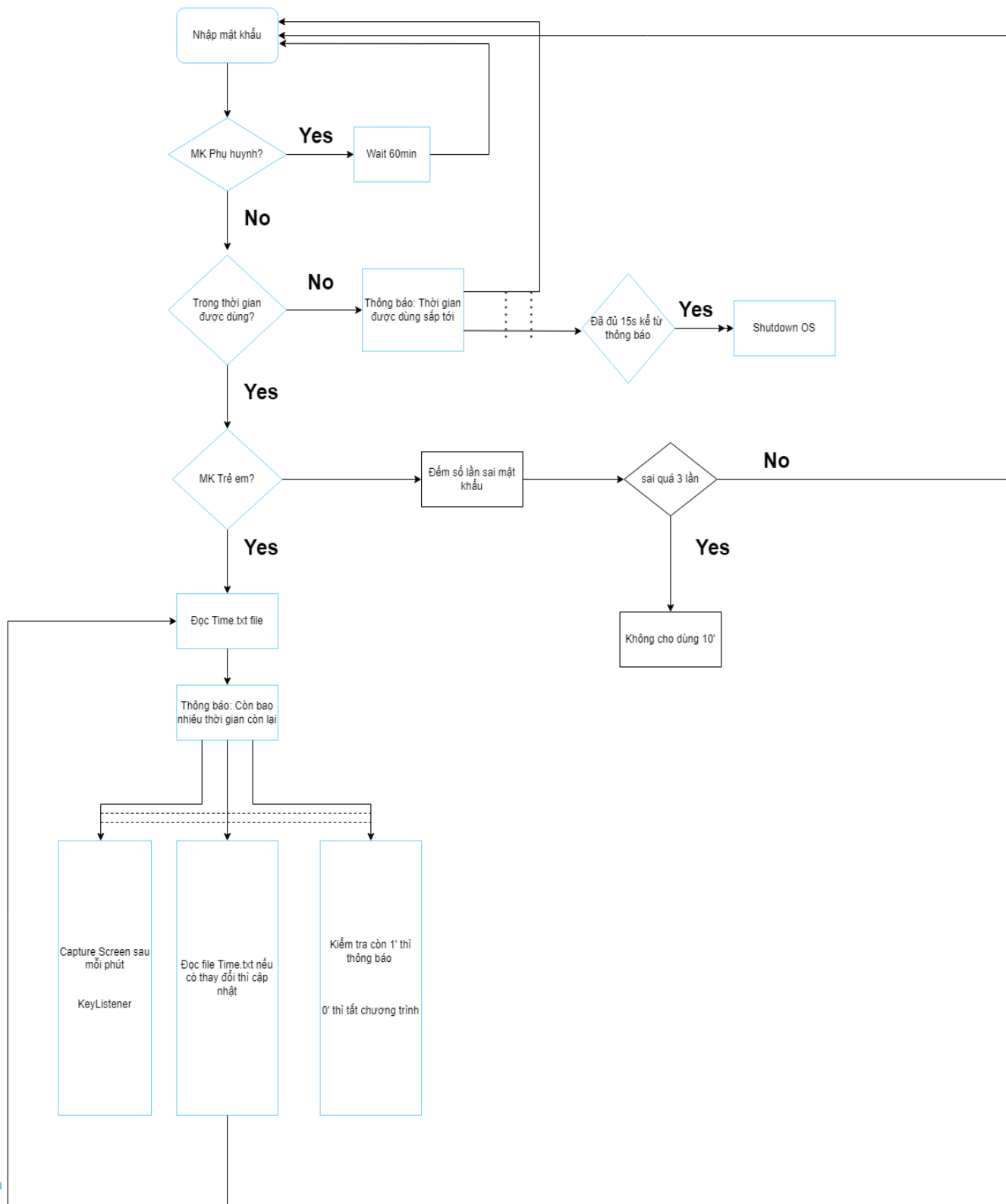
Vd: 02:07:58 là giây 58 duy nhất tiến trình P2 đọc file time.txt

### Kết quả:

Với cách qui ước thời gian sử dụng của mỗi tiến trình như trên các chương trình sẽ tránh rơi vào **critical section** nếu tất cả đều tuân thủ khung thời gian được sử dụng của mình

### III. CHƯƠNG TRÌNH CHILDREN:

#### Diagram





## Giải thích cách tổ chức:

```
autorun.py  
C.py  
Process.py  
Time.py
```

Gồm các file py: autorun.py dùng để viết script auto startup cùng windows  
Process và Time dùng để hỗ trợ cho hàm chính là C.py

## Hướng tiếp cận các vấn đề:

### 1. Khởi động cùng windows:

```
def startup(file_path=""):  
    if file_path == "":  
        file_path = os.path.dirname(os.path.realpath(__file__))  
    bat_path = f'C:/Users/{USER_NAME}/AppData/Roaming/Microsoft/Windows/Start Menu/Programs/Startup'  
    print(bat_path)  
    with open(bat_path + '/' + "open.bat", "w+") as bat_file:  
        bat_file.write(f'start "" {file_path}/C.py')  
    print(f'start "" {file_path}/C.py')  
startup()
```

**Solution:** Tạo script viết file bat ở thư mục Programs/Startup của Windows . Khi Windows khởi động sẽ mở file bat này với nội dung tương tự như sau: **start "" C:/Users/C.py**  
Drawback: Chưa thích ứng trên nhiều hệ điều hành khác nhau chỉ ở Windows

**Reference:** <https://stackoverflow.com/questions/4438020/how-to-start-a-python-file-while-windows-starts>

### 2. Phụ huynh đăng nhập:

```
if password == "Parent":  
    if check_process == True:  
        p1.terminate()  
    print("Hello Parent you have 60min!!")  
    time.sleep(3600) #Sleep 60 minute  
    continue
```

**Solution:** Khi người dùng nhập đúng mật khẩu sẽ cho chương trình dừng 60 phút rồi hỏi mật khẩu tiếp theo.

### 3. Lập trình song song Kiểm tra đủ 15s thì tắt máy và nhập mật khẩu lại từ đầu:

```
if Check_using_time_for_children(using_time) == False:
    print(f"Raise: The next time you can use computer: {Find_Next_Time(using_time)}")
    datetime_start = datetime.datetime.now()
    if check_process == False:
        p1 = multiprocessing.Process(target=Count_15_sec, args=(datetime_start, ))
        p1.start()
        check_process = True

def Count_15_sec(datetime_start):
    while True:
        if (datetime.datetime.now() - datetime_start).total_seconds() >= 15:
            os.system("shutdown /s /t 1")
            break
```

**Solution:** Song song với tiến trình chính nhập lại mật khẩu từ đầu ta tạo nên một process mới p1 có hàm là Count\_15\_sec hàm này sẽ kiểm tra song song và liên tục hệ thống đã vượt qua 15s kể từ lúc thông báo hay chưa nếu vượt qua rồi sẽ tắt hệ điều hành.

### 4. Mật khẩu sai 3 lần thì thoát:

```
if password != "Children":
    count += 1
    if count > 3:
        print("Bạn sẽ bị cấm 10 phút")
        time.sleep(600)
        os.system("shutdown /s /t 1") #Shutting down OS
```

**Solution:** Nếu mật khẩu sai ba lần sẽ bị chặn 10 phút rồi tắt hệ điều hành

### 5. Lập trình song song 3 tiến trình: Keyboard Listen, Đọc file Time, Check Remaining Time, Screen Capture

```
if using_time != ReadFile(open("C:/Users/ASUS/OneDrive - VNU-HCMUS/Desktop/OneDrive - VNU-HCMUS/Desktop/ShareFolder/time.txt", 'r')):
    using_time = ReadFile(open("C:/Users/ASUS/OneDrive - VNU-HCMUS/Desktop/OneDrive - VNU-HCMUS/Desktop/ShareFolder/time.txt", 'r'))
    print(f"Time Left: {Find_How_Many_Minute_Left(using_time)}")
    time_end = Time_End(using_time)
    time_end = datetime.datetime.combine(datetime.date.today(), time_end)
    p2.terminate()
    p3.terminate()
    p2 = multiprocessing.Process(target=Check_1_min_left, args=(time_end,))
    p3 = multiprocessing.Process(target=CaptureAndKeyListener)
    p2.start()
    p3.start()

def CaptureAndKeyListener():
    t1 = threading.Thread(target=KeyListener)
    t2 = threading.Thread(target=ScreenCapture)
    t1.start()
    t2.start()

def Check_1_min_left(time_end):
    check = True
    while True:
        curtime = datetime.datetime.now()
        if (curtime-time_end).total_seconds() >= -10 and check:
            print("Time Remaining: 1 Min")
            check = False
        else:
            if (curtime-time_end).total_seconds() >= 0:
                print("Shutting down OS")
                os.system("shutdown /s /t 1")
                #Shutting down OS
                break
```

```
def listen(key):
    key = str(key)
    with open("C:/Users/ASUS/OneDrive - VNU-HCMUS/Desktop/OneDrive - VNU-HCMUS/Desktop/ShareFolder/Monitor/log.txt", "a") as file:
        file.write(key)
def KeyListener():
    with open("C:/Users/ASUS/OneDrive - VNU-HCMUS/Desktop/OneDrive - VNU-HCMUS/Desktop/ShareFolder/Monitor/log.txt", "a") as file:
        file.write('\n----- \n')
        file.write(f'Date: {datetime.datetime.now()}\n')
        file.write('-----\n')
    with Listener(on_press = listen) as listener:
        listener.join()

def ScreenCapture():
    while True:
        s = str(datetime.datetime.now())
        s = s.replace(':', '-')
        myScreenshot = pyautogui.screenshot()
        myScreenshot.save(f"C:/Users/ASUS/OneDrive - VNU-HCMUS/Desktop/OneDrive - VNU-HCMUS/Desktop/ShareFolder/Monitor/ScreenShot/{s}.png")
        time.sleep(60)]
```

**Solution:** Tạo nên hai tiến trình mới song song với tiến trình đọc file time.txt đó là tiến trình:

- CaptureAndKeyListener và tiến trình Check\_1\_min\_left  
Trong CaptureAndKeyListener sẽ tạo ra hai thread làm việc song song là một bên KeyListen
- Và một bên là chụp màn hình sau mỗi phút

**Reference:** <https://www.youtube.com/watch?v=Z0Jfcv29Cy8>

## 6. Chặn rơi vào critical section:

Như thiết kế đã nêu phía trên theo thời gian

```
def To_Second(time):
    return time.hour*3600+time.minute*60+time.second

def Check_In_Schedule():
    curtime = To_Second(datetime.datetime.now().time())
    k = int(curtime/60)
    if (curtime-28)/60 <= k and k < (curtime-2)/60:
        return True
    else:
        return False
```

Với hàm Check\_In\_Schedule() ta sẽ kiểm tra chương trình có nằm trong khoảng thời gian được sử dụng tài nguyên chung: file time.txt

Hàm sẽ tính thời gian hiện tại sang second kiểm tra xem có nằm trong khoảng cho phép đọc file nếu trong khoảng đó sẽ trả về True ngược lại không được sử dụng sẽ là False

## IV. CHƯƠNG TRÌNH PARENT

### 1. Điều chỉnh khung giờ (time.txt)

```
bool Check_In_Schedule() {
    time_t now = time(0);
    tm* date_time = localtime(&now);
    float currentTime = date_time->tm_hour * 3600 + date_time->tm_min * 60 + date_time->tm_sec;
    int k = int((currentTime - 31) / 60);
    if ((currentTime - 44) / 60 <= k && k < (currentTime - 31) / 60) {
        return true;
    }
    else {
        return false;
    }
}

while (true) {
    if (Check_In_Schedule() == true) {
        Change_Time("C:\\Users\\teoca\\OneDrive - VNU-HCMUS\\ShareFolder\\time.txt", hour_f, minute_f, hour_t, minute_t, n);
        ReadFile("C:\\Users\\teoca\\OneDrive - VNU-HCMUS\\ShareFolder\\time.txt");
        break;
    }
}
```

**Solution:** Theo cách cài đặt như trên File chỉ đọc được trong một khung giờ duy nhất với tiến trình duy nhất nên không có trường hợp 2 tiến trình cùng dùng file time.txt cùng lúc. Gồm hàm: Check\_In\_Schedule để kiểm tra thời điểm hiện tại được dùng file chưa? Nếu được dùng thì sẽ dùng hàm Change\_Time để viết vào file time.txt

### 2. Xem lịch sử:

```
if (selection == 1) {
    ReadFile("C:\\Users\\teoca\\OneDrive - VNU-HCMUS\\ShareFolder\\log.txt");
}
```

**Solution:** Đơn giản chỉ đọc file log.txt không lo xảy ra deadlock.

**Giải thích:** Tài nguyên chung cơ bản bao gồm hai file giám sát (log.txt) và file time.txt  
Do thiết kế mỗi một thời điểm chỉ có duy nhất một tiến trình "request và use" file time.txt nên sẽ không tạo ra circular nên không dẫn đến deadlock.

Trong khi chạy demo chưa xảy ra lỗi xung đột tài nguyên chung log.txt. Tuy nhiên tương tự file time.txt nên "chia lịch" cho mỗi tiến trình dùng file log.txt tránh rơi vào critical section.

