Chapter 3: Memory management

Operationg System

@ptit.edu.vn

Posts and Telecommunications Institute of Technology
Faculty of Information Technology 1

IT1 Faculty
PTIT

August 15, 2023

Mục lục

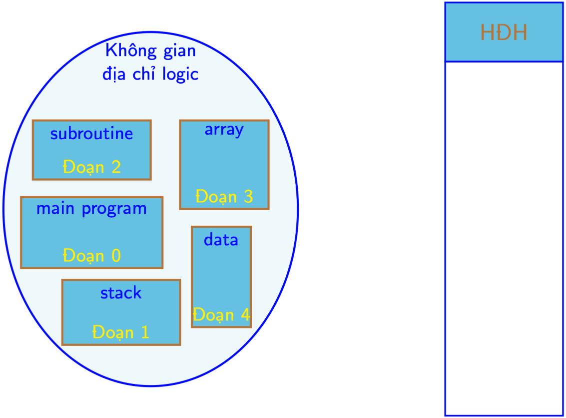
Contents

- The program is usually divided into several parts:
  - A main program
  - Set of subprograms
  - Variables, data structures
- Modules and objects in the program are identified by name:
  - sqrt() function, printf() procedure...
  - x, y, counter, Buffer...
- Elements in the module are determined by offset from the starting position:
  - The 5th statement of the sqrt()...
  - The 12th element of the Buffer array...

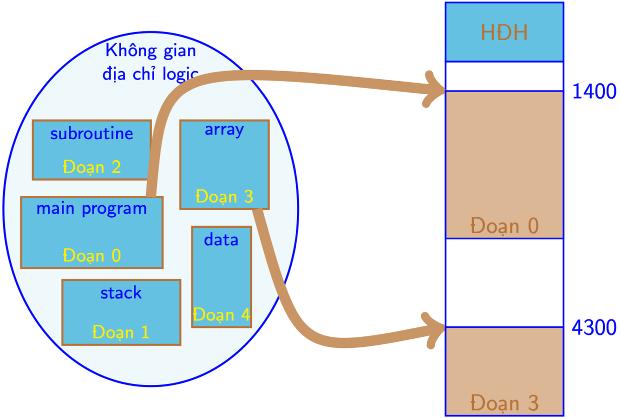
How is the program organized in memory?

- When paging, the program is divided into sections **page** are of equal size, regardless of the logical organization and meaning of each element.
- Another way of organization allows dividing the program into **segments** according to a logical structure
  - Program segment - Code segment: contains the entire program code, some functions or procedures of the program.
  - Data segment: contains global variables and arrays.
  - Stack segment: contains the process's stack
- Each segment occupies a continuous region
  - There is a starting position and a size
  - Can be located anywhere in memory
- Objects and elements in each paragraph are determined by their relative position compared to the beginning of the paragraph

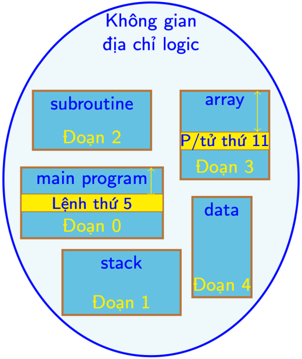
Example of memory segmentation

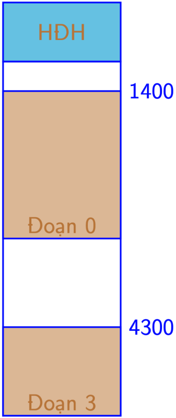


Example of memory segmentation

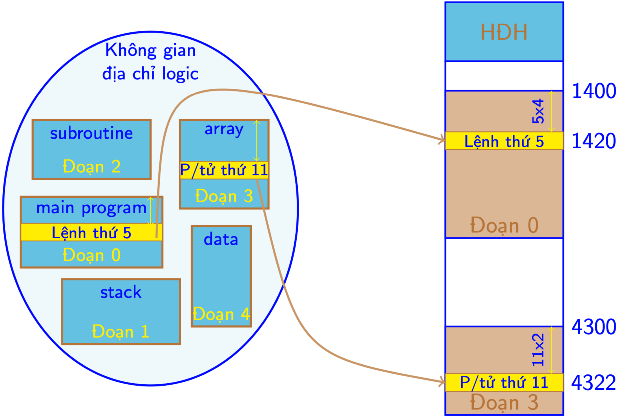


Example of memory segmentation

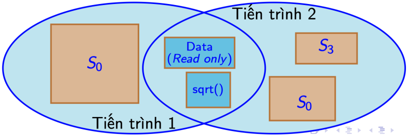




Example of memory segmentation



- Compare with dynamic chaptering:
  - **Similar**: memory is allocated in variable-sized regions
  - **Other**: programs can occupy more than 1 segment and do not need to be consecutive in MEM
- Advantage:
  - Avoid internal fragmentation, easily organize memory
  - Easily share fragments between different processes

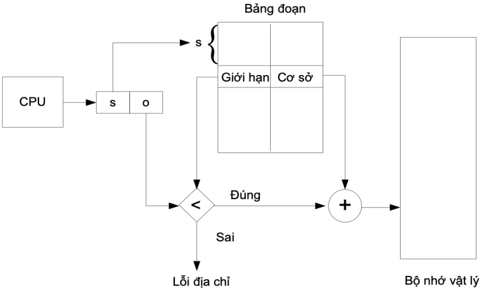


  - The size of each segment can be changed without affecting other segments
- disadvantages: There is external fragmentation

- Use **segment table** (**SCB**: *Segement Control Block*) for each process. Each element of the table corresponds to a paragraph, containing:

	Mark	Address	Length
0			
⋮	...	...	...
n	...	...	...

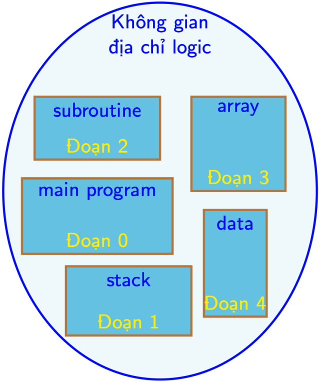
  - Mark (**Mark (0/1)**): The segment already exists in memory
  - Base address (**Address**): The starting location of the segment
  - Segment length (**Length**): Used to prevent unauthorized access outside the segment
- Logical address consists of 2 components (s, o):
  - s: serial number, paragraph name
  - o: translation in paragraph
- **Access address**: *<Name of the segment, offset in the segment>*



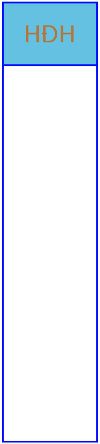
- From the segment index s, go to the segment table, find the starting physical address of the segment
- Compare offset o with segment length, if larger => wrong address - Access error
- The desired physical address is the sum of the segment start physical address and the offset address

Memory segmentation (cont.)
Address mapping

Example: Consider the segmentation strategy of the following system and calculate the following logical address:
$< 3, 345 >, < 1, 240 >, < 2, 120 >, < 2, 450 >$

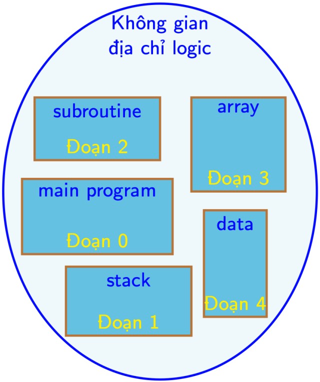


M	*A*	*L*
0	-	1000
0	-	400
0	-	400
0	-	1100
0	-	1000
SCB		

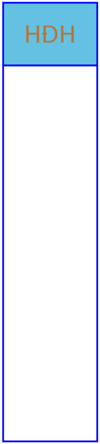


Example: Consider the segmentation strategy of the following system and calculate the following logical address:

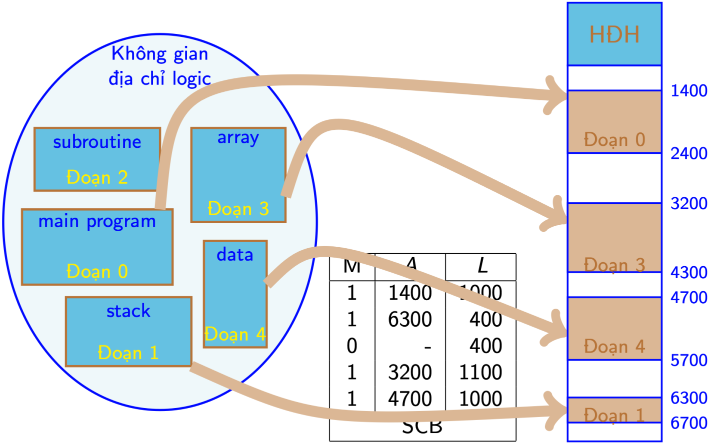
$< 3, 345 >, < 1, 240 >, < 2, 120 >, < 2, 450 >$



M	A	L
1	1400	1000
1	6300	400
0	-	400
1	3200	1100
1	4700	1000
SCB		



Example: Consider the segmentation strategy of the following system and calculate the following logical address:
$< 3, 345 >, < 1, 240 >, < 2, 120 >, < 2, 450 >$



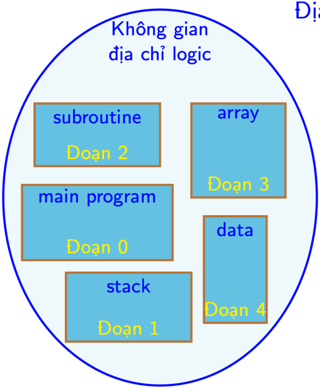
M	A	L
1	1400	1000
1	6300	400
0	-	400
1	3200	1100
1	4700	1000

SCB

Memory segmentation (cont.)

Address mapping

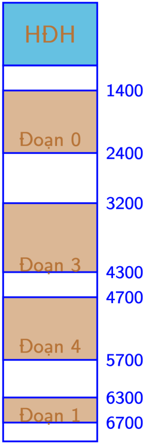
Example: Consider the segmentation strategy of the following system and calculate the following logical address:
$< 3, 345 >, < 1, 240 >, < 2, 120 >, < 2, 450 >$



Địa chỉ $<3,345> = ?$

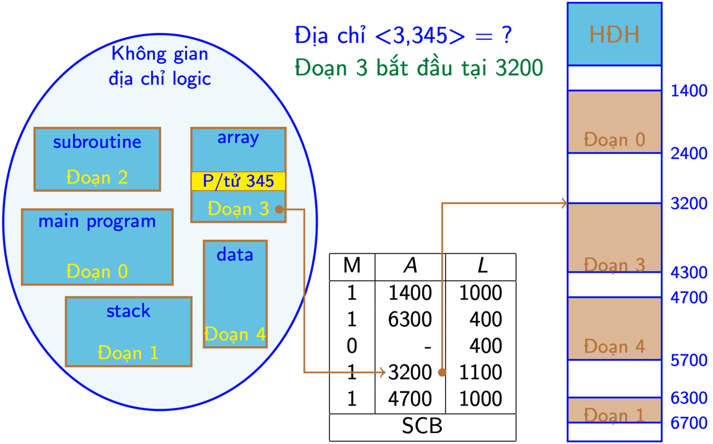
M	A	L
1	1400	1000
1	6300	400
0	-	400
1	3200	1100
1	4700	1000

SCB



Memory segmentation (cont.)
Address mapping

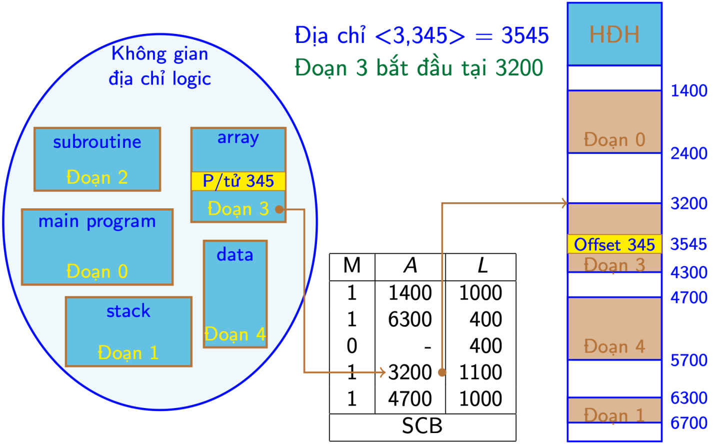
Example: Consider the segmentation strategy of the following system and calculate the following logical address:
$< 3, 345 >, < 1, 240 >, < 2, 120 >, < 2, 450 >$



M	A	L
1	1400	1000
1	6300	400
0	-	400
1	3200	1100
1	4700	1000

SCB

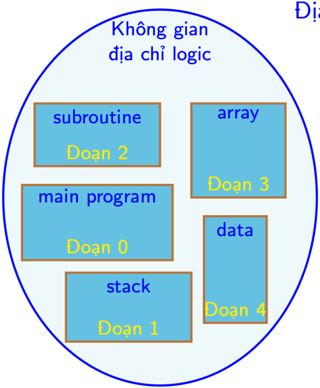
Example: Consider the segmentation strategy of the following system and calculate the following logical address:
$< 3, 345 >, < 1, 240 >, < 2, 120 >, < 2, 450 >$



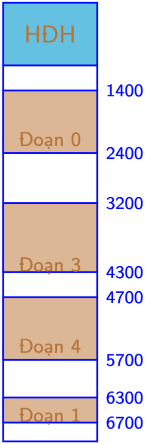
Memory segmentation (cont.)
Address mapping

Example: Consider the segmentation strategy of the following system and calculate the following logical address:
$< 3, 345 >, < 1, 240 >, < 2, 120 >, < 2, 450 >$

Địa chỉ <1,240> = ?



M	A	L
1	1400	1000
1	6300	400
0	-	400
1	3200	1100
1	4700	1000
SCB		

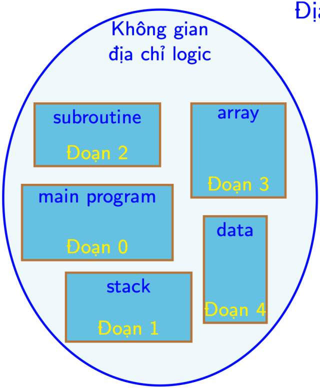


Memory segmentation (cont.)

Address mapping

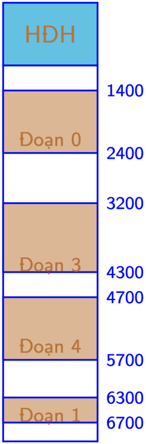
Example: Consider the segmentation strategy of the following system and calculate the following logical address:

$< 3, 345 >, < 1, 240 >, < 2, 120 >, < 2, 450 >$

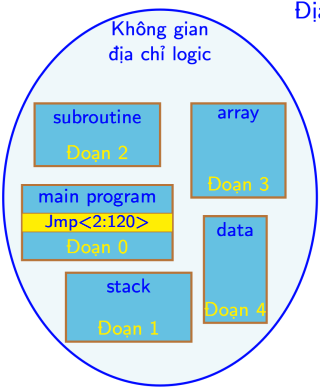


Địa chỉ <1,240> = 6540

M	A	L
1	1400	1000
1	6300	400
0	-	400
1	3200	1100
1	4700	1000
SCB		



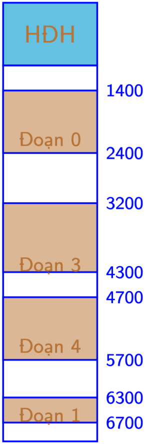
Example: Consider the segmentation strategy of the following system and calculate the following logical address:
$< 3, 345 >, < 1, 240 >, < 2, 120 >, < 2, 450 >$



Địa chỉ <2,120> = ?

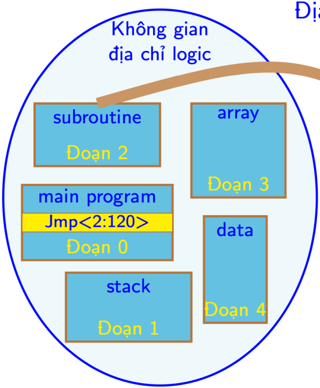
M	*A*	*L*
1	1400	1000
1	6300	400
0	-	400
1	3200	1100
1	4700	1000

SCB

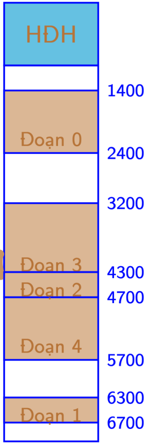


Example: Consider the segmentation strategy of the following system and calculate the following logical address:
$< 3, 345 >, < 1, 240 >, < 2, 120 >, < 2, 450 >$

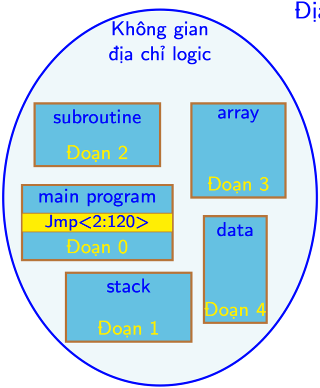
Địa chỉ <2,120> = ?



M	A	L
1	1400	1000
1	6300	400
0	-	400
1	3200	1100
1	4700	1000
SCB		



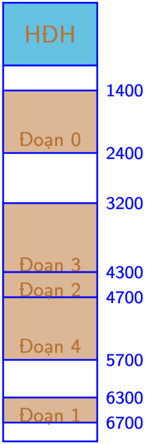
Example: Consider the segmentation strategy of the following system and calculate the following logical address:
$< 3, 345 >, < 1, 240 >, < 2, 120 >, < 2, 450 >$



Địa chỉ <2,120> = ?

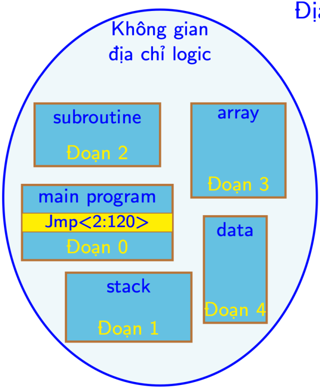
M	A	L
1	1400	1000
1	6300	400
1	4300	400
1	3200	1100
1	4700	1000

SCB

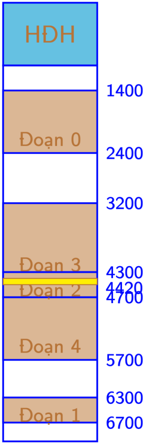


Example: Consider the segmentation strategy of the following system and calculate the following logical address:
$< 3, 345 >, < 1, 240 >, < 2, 120 >, < 2, 450 >$

Địa chỉ <2,120> = 4420



M	A	L
1	1400	1000
1	6300	400
1	4300	400
1	3200	1100
1	4700	1000
SCB		

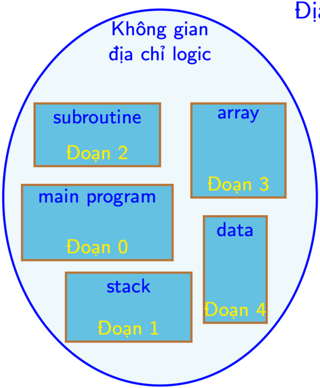


Memory segmentation (cont.)

Address mapping

Example: Consider the segmentation strategy of the following system and calculate the following logical address:

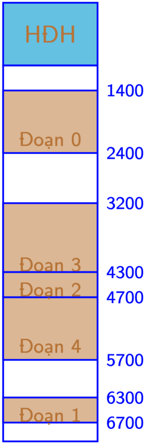
$< 3, 345 >, < 1, 240 >, < 2, 120 >, < 2, 450 >$



Địa chỉ <2,450> = ?

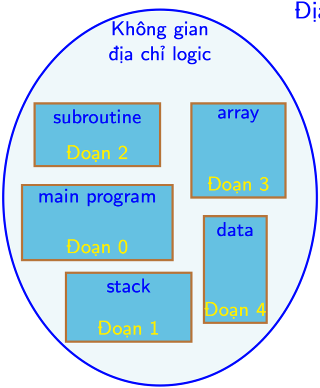
M	*A*	*L*
1	1400	1000
1	6300	400
1	4300	400
1	3200	1100
1	4700	1000

SCB



Memory segmentation (cont.)

Example: Consider the segmentation strategy of the following system and calculate the following logical address:
$< 3, 345 >, < 1, 240 >, < 2, 120 >, < 2, 450 >$

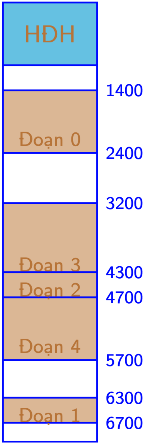


Địa chỉ <2,450> = ?

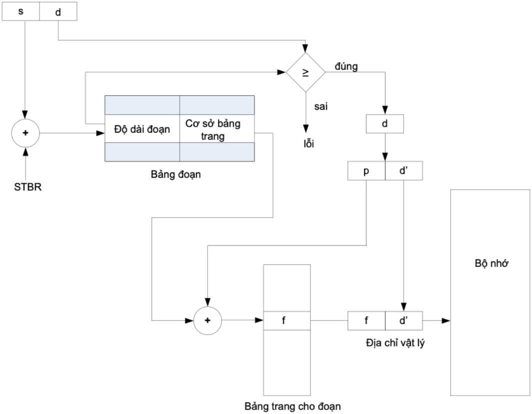
Lỗi truy nhập!

M	*A*	*L*
1	1400	1000
1	6300	400
1	4300	400
1	3200	1100
1	4700	1000

SCB



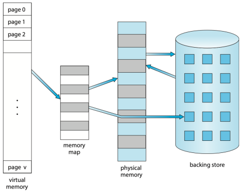
- Program segments, each segment will perform paging
- The address includes: paragraph number, page number, page offset
- The process has 1 segment table, each segment has 1 page table



Virtual memory

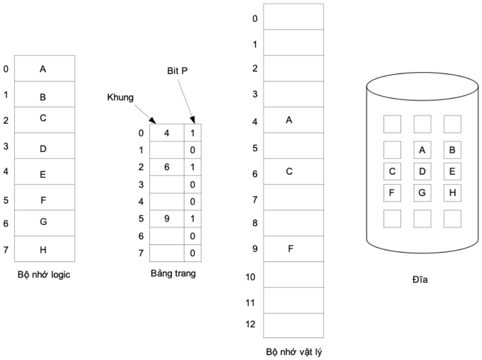
- The purpose of a computer system: to execute programs
- Program and data (*whole or partial*) must reside in main memory, which can be divided into small parts scattered throughout memory during execution
- Not every running process uses all instructions and data with the same frequency
  - It is not necessary that all pages/segments of a process must be present in memory at the same time when the process runs
  - Pages or segments can be swapped from disk to memory when accessed.
- The part of the program that has not been put into main memory is stored in secondary memory (eg: *hard disk*) => **Virtual memory**
  - Allows programmers not to worry about physical memory limits

- Technique for using secondary memory to store processes
- Process sections move in and out between main memory and secondary memory
- Executing the process does not need to load the entire process into main memory

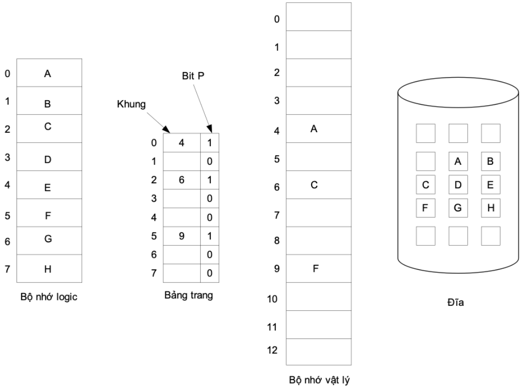


- There are two methods of installing virtual memory techniques:
  - Load pages as needed (*Demand paging*)
  - Load segments as needed (*Demand segmentation*)

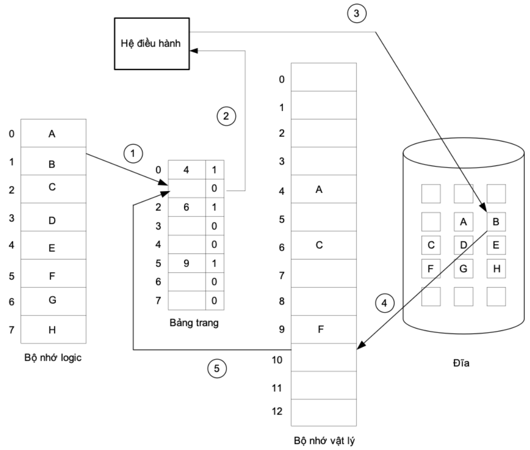
- Load pages on demand based on paging combined with memory-disk swap => Paged process and pages contained on disk
- When executing, load the process into MEM: only load the pages needed => Pages do not have to be loaded at the same time



- Process consists of pages on disk and in MEM: add P bit in page table entry to distinguish (P=1: loaded into MEM)



- Process of checking and loading pages:
  - The process accesses a page, checking the P bit. If P=1, the access occurs normally. If P=0, a missing page event occurs



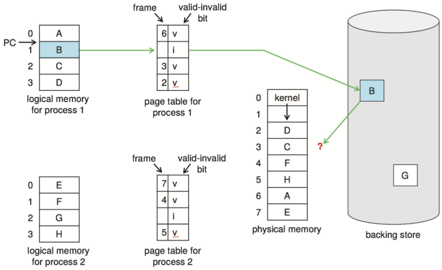
- Interrupt missing page handling:
  - Step 1: The OS finds an empty frame in the MEM
  - Step 2: Read the missing page into the page frame you just found
  - Step 3: Modify the corresponding entry in the page table: change bit P=1 and the number of frames allocated to the page
  - Step 4: Restore the process state and continue the interrupted command

- *Load pages completely as needed*:
  - Starts a process without loading any pages into memory
  - When the command pointer is moved by the operating system to the first command of the process to execute, a missing page event is generated and the corresponding page is loaded.
  - The process then proceeds normally until the next missing page
- *Loading pages in advance: different from loading pages on demand*
  - Unneeded pages are also loaded into memory
  - Not effective

- When a page shortage occurs: The OS must find an empty **frame** in memory, read the missing page into the frame, and the process then operates normally.
- Virtual memory > real memory and multiprogramming mode sometimes has no free **frame** to load a new page
- Solution given:
  - Terminate the process due to not satisfying memory needs
  - Swap the process to disk and wait for a more favorable time to reload the process into memory to continue executing
  - Use technique **Change page**

Page change technique

- If there are no free frames left, the OS selects an allocated but currently unused frame and releases this frame.
- The contents of the frame will be exchanged to disk, the memory page contained in the frame will be marked as not being in memory (by changing the P bit in the page table).
- The freed frame is allocated to a new page to be loaded



Page change process:

- *Step 1:* Determine the page on the disk that needs to be loaded into MEM
- *Step 2*: If there is an empty frame on the MEM, go to B4
- *Step 3*:
    - Select a frame on MEM to release, according to a certain algorithm
    - Write the changed frame content to disk (if necessary), update the page table and frame table
- *Step 4*: Read the page to be loaded into the newly released frame; Update the page table and frames table
- *Step 5*:Continue the process from the point where it was stopped before changing the page

Change the recorded page and change the unrecorded page:

- If the need to change pages occurs when loading a new page, the page load time will increase significantly.
- Helps identify pages that have not changed since loading and have not been written back to disk
- Use an additional revision bit M in the page entry to mark whether the page has been modified (1) or not (0).
- Locked frames:
  - When looking for pages to free and swap, the OS subtracts a number of frames
  - Some frames that will not be released during the search for frames to change pages are *locked frames*
  - For example: *Frame containing the OS kernel process, containing important control information structures*
  - Identified by a separate bit contained in the item

Page-changing strategies:

- OPT/MIN: Optimal page changing algorithm
- FIFO (First In First Out): First in, first out
- LRU (Least Recently Used): The page has the longest last use
- CLOCK: Clock algorithm
- LFU (Least Frequently used):Lowest frequency of use
- MFU (Most Frequently used): Highest frequency of use
- ...

Optimal Page Change (OPT):

- *Choose the page that will be unused for the longest period of time in the future to change or the page with the furthest next use*
- Allows you to minimize missing pages and optimize according to this standard
- The OS anticipates future use of pages
- Not applicable in practice only compared with other strategies

	2	3	2	1	5	2	4	5	3	2	5	2
OPT	2	2	2	2	2	2	4	4	4	2	2	2
		3	3	3	3	3	3	3	3	3	3	3
				1	5	5	5	5	5	5	5	5
					F		F			F		

Excercise: *Optimal Page Change (OPT):*
Suppose the process is given 3 frames, the process's memory space has 5 pages and the process's pages are accessed in the following order:

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

Determine the page loading order using the OPT optimization algorithm.

First in, first out (FIFO):

- Whichever page is loaded first will be moved first
- The simplest method
- The swap page is the page that stays in memory the longest

For example: Suppose the process is given 3 frames, the process's memory space has 5 pages and the process's pages are accessed in the following order: 2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2

	2	3	2	1	5	2	4	5	3	2	5	2
	2	2	2	2	5	5	5	5	3	3	3	3
FIFO		3	3	3	3	2	2	2	2	2	5	5
				1	1	1	4	4	4	4	4	2
					F	F	F		F		F	F

Virtual memory (cont.)

Page-changing strategies

Change page with longest last use (LRU):

- A changed page is the page that has had the longest time from the last visit to the present time (if you haven't visited in a long time, replace it)
- According to the principle of time locality, that is the page least likely to be used in the future
- In fact, LRU gives almost as good results as the optimal pagination method

	2	3	2	1	5	2	4	5	3	2	5	2
LRU	2	2	2	2	2	2	2	2	3	3	3	3
		3	3	3	5	5	5	5	5	5	5	5
				1	1	1	4	4	4	2	2	2
					F		F		F	F		

Change the page with the longest last use(LRU):

- Identify the page whose last visit occurred the longest ago?
- Using counter variable:
  - Each pagination table entry will have an additional field containing the time of last page access - the logical time
  - The CPU also has a logic timer added to this
  - The counter index increases each time a memory access occurs
  - Each time a memory page is accessed, the counter index is written to the access time field in that page's entry.
  - The time field always contains the time the page was last accessed
  - The converted page is the page with the smallest time value

Change the page with the longest last use(LRU):

- Use stacks
  - Special stack used to hold page numbers
  - Every time a memory page is accessed, the page number is moved to the top of the stack
  - The top of the stack will contain the most recently accessed page
  - The bottom of the stack is the LRU page, that is, the page that needs to be exchanged
  - Avoid searching in page tables
  - Suitable for implementation by software

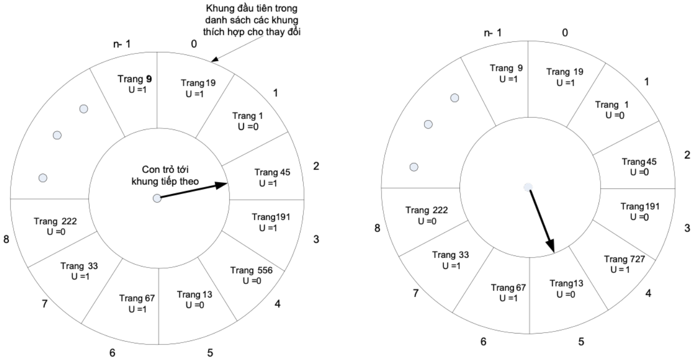
CLOCK: Clock Algorithm Improved FIFO to avoid replacing long loaded pages that are still usable

- Each page is appended with 1 bit using U
  - When page is accessed read/write: U = 1
  - As soon as the page is read into memory: U =1
- Frames/pages that can be changed are linked to a circular list
- When a page is changed, the cursor is moved to point to the next page
- When there is a need to change pages, OS checks the page being pointed to
  - If U=0: page will be changed immediately
  - If U=1: set U=0 and point to the next page, repeat the process

- U of all pages in the list =1 the cursor will rotate exactly 1 revolution, set U of pages =0 and the currently pointed page will be changed

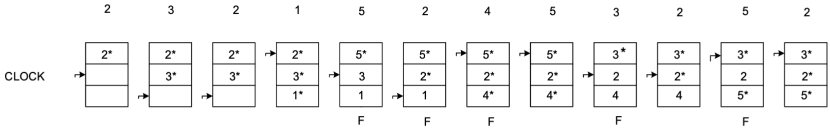
CLOCK: Clock algorithm

Need to load page 727



CLOCK: Clock algorithm

For example: the process is given 3 frames, the process's memory pages are accessed in order: 2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2



- Based on 2 pieces of information to make the decision to change pages:
  - The time the page was loaded, expressed as the page position in a FIFO-like list
  - Whether the page has been used recently is indicated by the U bit
- Checking the extra U bit is similar to giving the page more capacity to be held in memory

Improved clock algorithm:

- Uses additional information about whether the page content has been changed using the M bit
- Combining U and M bits, there are 4 possibilities:
  - U=0, M=0: the page has not been accessed recently and the content has not been changed, very suitable to be changed out
  - U=0, M=1: page has changed content but has not been accessed recently, also a candidate to change out
  - U=1, M=0: the page was recently accessed and therefore according to the time-locality principle may be about to be accessed again
  - U=1, M=1: page with changed content and recently accessed, not suitable for change

Improved clock algorithm:

- Steps to change pages:
- Step 1:
  - Starting from the current cursor position, examine the pages
  - The first page with U=0 and M=0 will be changed
  - Only check without changing the contents of U and M bits
- Bước 2:
  - If you go all the way around and can't find a page with U and M equal to 0, scan the list a second time.
  - The first page with U=0, M=1 will be changed
  - Set the U bit of scanned but ignored pages to 0
- If not found, repeat step 1 and step 2 if necessary

- The operating system reserves a number of empty frames connected into a linked list called buffer pages
- The page is changed as usual, but the page content is not immediately deleted from memory
- The frame containing the page is marked as an empty frame and added to the end of the buffer page list
- The new page will be loaded into the top frame in the buffer page list
- At the appropriate time, the system will write the content of the pages in the buffer list to disk

- Page caching techniques allow for improved speed:
- If the changed page has content that needs to be burned to disk, the HDH can still load the new page immediately
  - Burning to disc will be postponed to a later date
  - Burning to disc can be performed simultaneously with multiple pages in the list marked as blank.
- The changed page remains in memory for a while:
  - If there is an access request during this time, the page will be retrieved from the cache list and used immediately without reloading from disk.
  - The buffer acts like a cache

Allocate page frames

Frame limit

- With virtual memory, the process does not have to reside entirely in the computer's memory. Some pages of the process are allocated memory frames while others reside temporarily on disk.
- How many frames does the OS allocate to each process? When the maximum number of frames allocated to a process decreases to a certain level, page missing errors occur frequently
- Sets the minimum limit of frames allocated to the process
- When the number of frames allocated to a process increases to a certain level, adding more frames to the process does not significantly reduce the frequency of missing pages anymore.
- Allocate a fixed number of frames and a variable number of frames.

Allocate a fixed number of frames

- Gives the process a fixed number of frames to hold memory pages
- The quantity is determined at the time of process creation and does not change while the process exists
- Equal allocation:
  - Processes are allocated the same maximum number of frames
  - The number is determined based on the MEM size and the desired level of multiprogramming
- Unequal allocation:
  - Processes are allocated different maximum frames
    - The number of frames is proportional to the process size
    - Has priority

Allocate a variable number of frames

- The maximum number of frames allocated to each process may change during execution
- Changes depend on the implementation status of the process
- Allows more efficient use of memory than the fixed method
- Need to monitor and process information about the process's memory usage

- All allocation:
  - Allows a process to change a new page into any (non-locked) frame, including frames already allocated to another process
- Local allocation:
  - The page can only be changed into the frame currently allocated to that process
- Allocation range is closely related to the maximum number of frames:
  - Fixed number of frames corresponding to the local allocation range
  - The number of frames varies relative to the overall allocation range

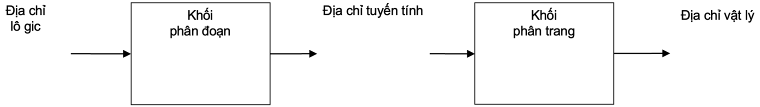
- Stagnation is the condition of constantly changing pages due to insufficient memory
- The page change time of the process is greater than the execution time
- Occurs when:
    - Limited memory size
    - The process requires concurrent access to multiple memory pages
    - The system has a high degree of multiprogramming
- When the process becomes stagnant, the frequency of missing pages increases significantly
- Used to detect and resolve stagnation problems

Controls the frequency of missing pages

- Monitor and record the frequency of missing pages
- It is possible to set an upper and lower bound on the frequency of missing pages for a process
  - Frequency exceeding the upper limit:
    - Grants new frame process
    - If it cannot find frames to add, the process will hang or be terminated
  - Frequency of missing pages is lower than the lower limit: some frames of the process are recovered

Intel Pentium memory management

- Intel's Pentium processors support a memory management mechanism: segmentation combined with paging
- The process's memory space consists of many segments, each segment can have a different size and is paged before being placed in memory.
- Address mapping: 2 stages



Intel Pentium memory management (cont.)

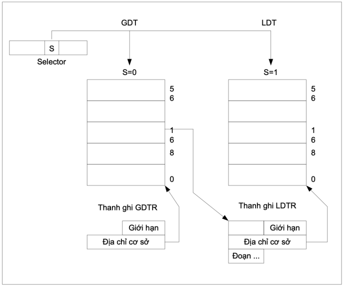
- Allows processes to have a maximum of 16KB (more than 16000) segments, each segment has a maximum size of 4GB
- The logical memory space is divided into two parts:
  - Part 1: dedicated to the process, includes a maximum of 8KB segments
  - Part 2: common to all processes, including the Operating System, and also contains a maximum of 8KB segments
- Process management information belongs to the first and second parts of the tables LDT (Local Descriptor Table) and GDT (Global Descriptor Table): contain management information:
  - Each cell is 8 bytes in size: contains the base address and limit of the corresponding segment
- To speed up address mapping, Petinum has 6 segment registers: allowing the process to access 6 segments simultaneously.
- Segment information is contained in six 8-byte registers

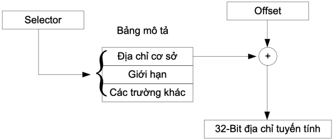
- Logical addresses include: (selector, offset):

s	g	p
13 bit	1 bit	2 bit

  - Selector: select the corresponding cell from the two tables describing LDT and GDT
  - S: is the paragraph number
  - G: indicates whether the segment belongs to GDT (g=0) or LDT(g=1)
  - P: indicates protected mode (p=0 is kernel mode, p=3 is user mode)
  - Offset: offset in segment, size 32bit

- Convert logical addresses to linear addresses:
  - First, the selector is used to find the corresponding cell in the GDT, the LDT contains the description of paragraph (a);
  - The segment descriptor is then combined with the offset to produce a linear address (b).





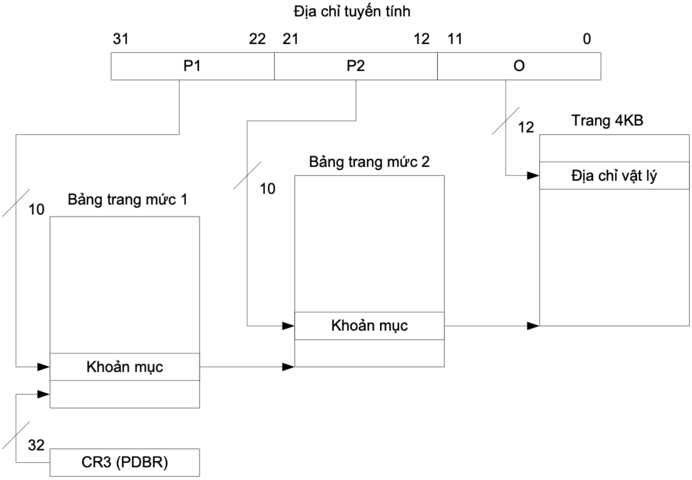
Intel Pentium memory management (cont.)

- Supports 4KB or 4MB page sizes, depending on the page size flag value
- Page size 4KB: organize the page table into 2 levels
  - Linear addresses are 32 bits in size

p1	p2	o
10 bit	10 bit	12 bit

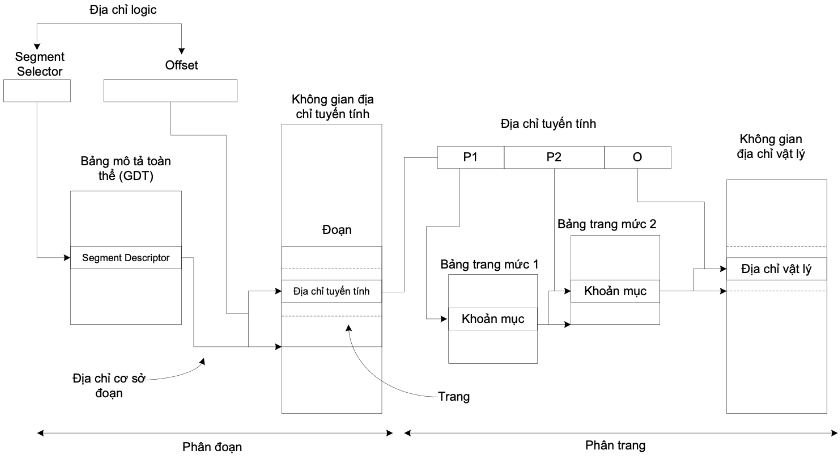
  - P1: Allows finding page table level 1
  - P2: find the corresponding cell in the level 2 page table combined with the offset o to create the physical address
- Page size 4MB: The page table has only one level
  - P :10bit
  - O: offset, 22bit size allows pointing to a specific location in a 4MB memory page

- Converts a linear address to a 4KB page size physical address



Intel Pentium memory management (cont.)

- Converts a linear address to a 4KB page size physical address



Memory management in Windows XP

- Allows processes to use up to 4GB of virtual memory
  - 2GB is reserved for the process
  - The following 2GB is shared by the system
- Virtual memory is implemented using the on-demand page fetch and page swap technique
  - Memory page size 4KB
  - Organize a 2-level page table
- Load pages in clusters: when a missing page occurs, load the entire cluster including a number of pages behind the missing page

Memory management in Windows XP (cont.)

- Control the number of pages: assign each process a maximum and minimum number of pages
- The maximum and minimum number of pages allocated to the process are changed depending on the free memory status
- The OS stores a list of free frames, and uses a safety threshold
  - Number of free frames less than threshold: The operating system considers the processes in progress.
  - Processes with more than the minimum number of pages will have their number of pages reduced until they reach their minimum number.
- Depending on the processor, Windows XP uses different pagination algorithms

Summary

Chapter 3 Memory management

- Memory segment
- Virtual memory

Chapter 4 Process management

- Concepts related to the process
- Luồng (thread)
- Moderate the process
- Synchronize concurrent processes
- Deadlock and hunger

1. **Question 1**: The memory is 1MB in size. Use the buddy system to allocate processes in turn with the following sizes: A: 112KB, B: 200KB, C: 150KB, D: 50KB
2. **Question 2**: The memory frame size is 4096 bytes. Let's convert logical addresses 8207, 4300 to physical addresses knowing that the page table is as follows:

Page	Frame
0	3
1	5
2	4
3	30
4	22
5	14

3. **Question 3**: The process's logical address space consists of 11 pages, each page of size 2048B mapped into physical memory of 20 frames.
   3.1 How many bits are needed to represent a logical address?
   3.2 How many bits are needed to represent a physical address?
4. **Question 4**: Physical memory has 3 frames. The order of accessing the pages is 1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6. Draw a diagram memory allocation and How many page missing events occur if using:
   - Optimization algorithm
   - FIFO
   - LRU
   - Clock

Questions and exercises (cont.)

5. **Question 5**: Suppose the process is given 4 physical memory frames, the process's pages are accessed in the following order: 1,2,3,4,5,3,4,1,6,7 ,8,7,8,9,7,8,9,5. Please specify the order of loading and changing pages if using:
    - Optimization algorithm
    - FIFO
    - LRU
    - Clock