
# Memory Management

A.Y. 2017-18

ACSO Tutoring

MSc Eng. Michele Zanella

## Basic notions

- Main memory, cache and the registers built into the processor are the only storage that the CPU can access directly.
- Each process has a separate memory space -> range of **legal access**:
    - *Base register*: smallest legal physical memory address
    - *Limit register:* the size of the range
    - If the program executing in user mode tries to access OS memory -> trap to the OS and fatal error
    - Only OS can load the two registers

# Basic notions (cont'd)

- To be executed a program must be brought into memory and placed within a process
- **Binding**: between source symbolic address and the absolute addresses:
  - **Compile time**
  - **Load time**
  - **Execution time**

# Basic notions (cont’d)

- **Logical (Virtual) Address**: Address generated by the CPU
    - ➢ Logical Address Space
- **Physical Address:** Address seen by the memory unit
    - ➢ Physical Address Space
- **Memory-Management Unit (MMU):** run-time mapping from virtual to physical addresses
    - **Relocation Register:** base register
    - Converts logical addresses into physical ones

# Basic notions (cont'd)

- **Dynamic Loading:** A routine is not loaded until it is called
- **Dynamic Linked (Shared) Libraries:** system libraries that are linked to user programs when the programs are run
- **Static Linked Libraries:** system libraries are treated like any other module and combined by the loader into the binary programe image

# Swapping

- When a process is moved temporary out of memory to a **backing store** and then brought back into memory for continued execution.
- **Standard Swapping:** From main memory to a backing store
  - **Ready queue**: all processes whose memory images are on the backing store and that are *ready to run.*
  - The dispatcher checks to see whether the next process in the queue is in memory, otherwise if there is no free memory region -> it swaps out a process from memory and swaps in the desired process
- We must be sure that the process to swap out is completely idle.
- Not used in modern OS.
- In Mobile systems, if memory is needed:
  - No swapping
  - Applications are forced to release resources or to terminate

# Memory protection

- It prevents a process from accessing memory which is not own
- The MMU maps the logical address dynamically by adding the value in the relocation register
- When the *scheduler* selects a process for execution:
    - The *dispatcher* loads the relocation and limit registers during the context switch.
- The relocation allows the OS's size to change dynamically

# Memory allocation

- Dividing memor into several fixed-sized **partitions**.
- Each partition contains exactly one process
- The OS keeps a table indicating which parts of memory are available or not
- The OS have a list of available block sizes and an input queue
- The memory blocks available comprise a *set* of hole of various sizes scattered throughout memory -> problem

# Memory allocation (cont'd)

- **Memory allocation**: is a particulr instance of the general *dynamic storage allocation problem*.
  Solutions to select a free hole:
  - **First fit:** Allocata the first hole that is big enough
  - **Best fit:** Allocate the smallest hole that is big enough
  - **Worst fit:** Allocate the largest hole

# Fragmentation

- **External Fragmentation:** the free memory is broken into little pieces. The available space is not contiguous.
  - 50% rule
  - Solutions:
    - **Compaction:** shuffle the memory contents to place all free memory in one large block
    - Allow not contiguous logical address space: **Segmentation and paging:** breaking the physical memory into fixed-sized blocks and allocatio memory in units based on block size -> it leads to
- **Internal Fragmentation:** there is unused memory that is internal to a partition

# Segmentation

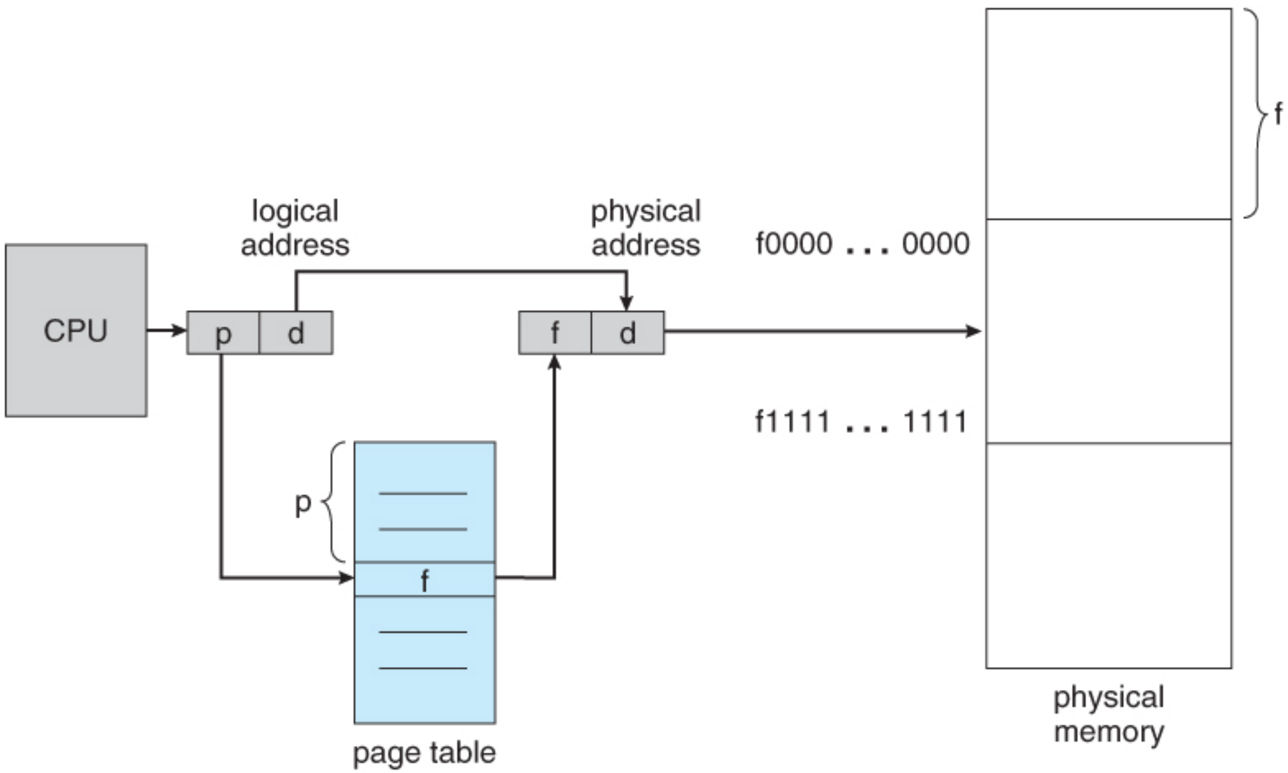
- The logical address space is a collection of segments
- Each segment ha a **name** and a **length**.

  `<segment-number, offset>`
- Different *segments* generated by the compiler:
  - The code
  - Global variables
  - Heap
  - Stack
  - Standard C library
- **Segmentation table**

# Paging

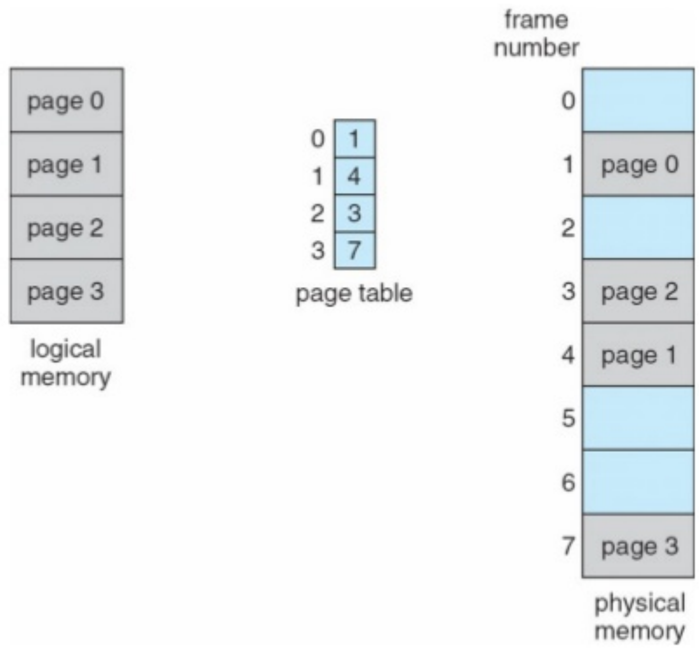
- Breaking physical memory into fixed-sized blocks -> **Frames**
- Breaking logical memory into blocks of the same size-> **pages**
- It avoids external fragmentation and the need for compaction
- Each generated address:
  - **Page number**: index into the *page table*.
  - **Page offset**: it defines with the page number the physical memory address
- Size is defined by the hardware -> power of 2.
- The logical address has *m-n* bits for the page number and *n* low-order bits for the page offset (space $2^m$, size $2^n$)

# Paging (cont'd)



# Paging (cont'd)

## Paging Model of Logical and Physical Memory

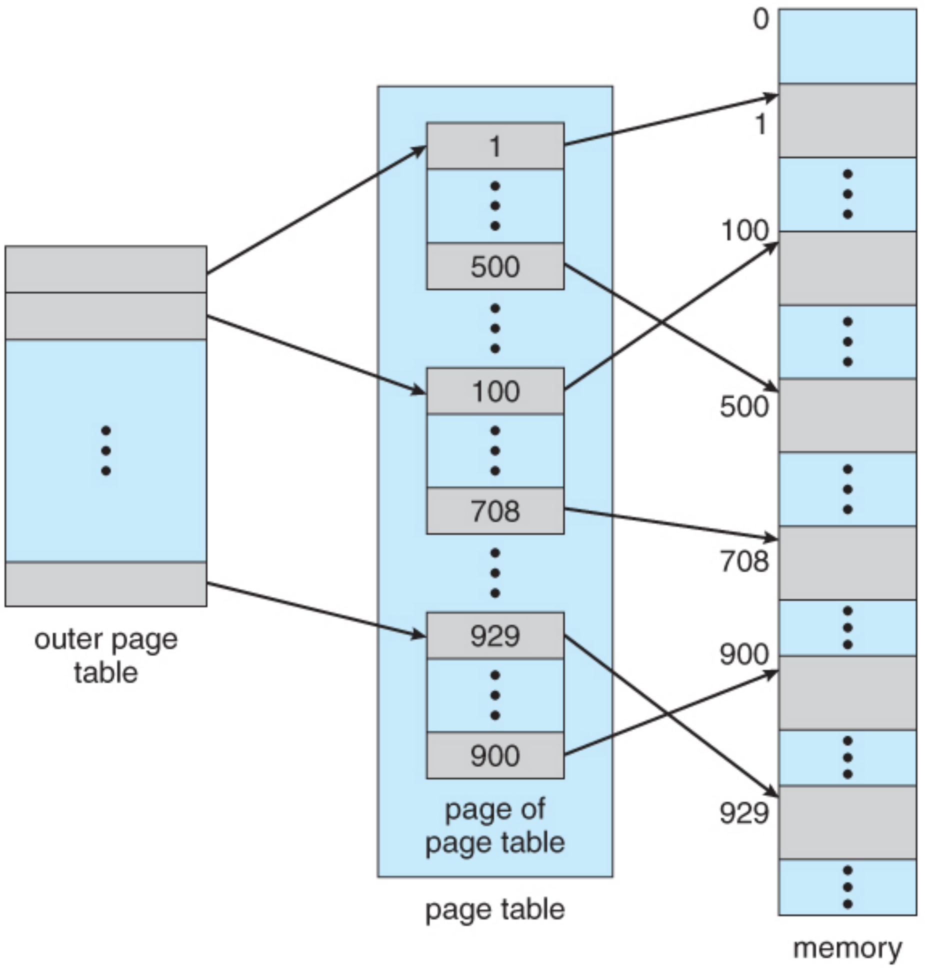



Unit-4 OS 15

# Paging (cont'd)

- **Protection**, ensured by *protection bits*:
  - Read-write or read only bit
  - Valid-invalid bit
- **Shared pages**: method of interprocess communication
- Structures:
- **Hierarchical Paging**: dividing the page table into smaller pieces -> two-level paging algorithm: page table itself is paged

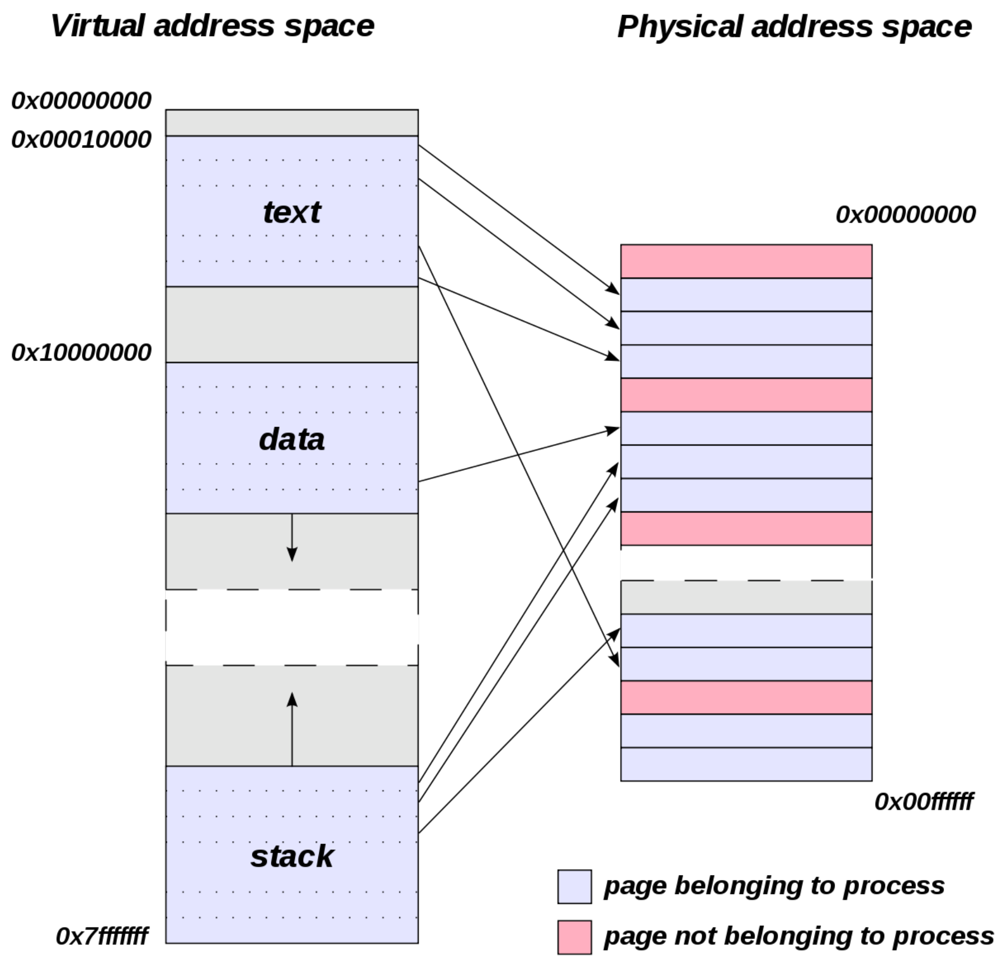
# Paging (cont’d)



# Virtual Memory

- It allows the execution of processes that are not completely in memory
- It abstracts the main memory
- It allows processes to share files easily and to implement shared memory
- It is an efficient mechanism for process creation
- It involves the separation of logical memory from physical memory

# Virtual Address space



# Page Sharing

- System libraries can be shared: the actual pages where the libraries reside in physical memory are shared by all processes
- Processes can communicate through the use of shared memory
- Pages can be shared during process creation