Virtual memory

E. Campo M. Knoblauch Ó. López J. Clemente

Departamento de Automática
Universidad de Alcalá

Index

What is the VM?

- It is a memory management scheme in which processes:
  - Run without being completely loaded into main memory (MM)
  - May be larger than the available MM
- It allows a decoupling between the physical address space and the virtual address space
- Uses secondary storage as an extension of the MM
- Based on the principle of locality of references $\Rightarrow$ Only the information needed at each moment is maintained in MM
- The control is performed by the operating system (OS) with the indispensable help of hardware

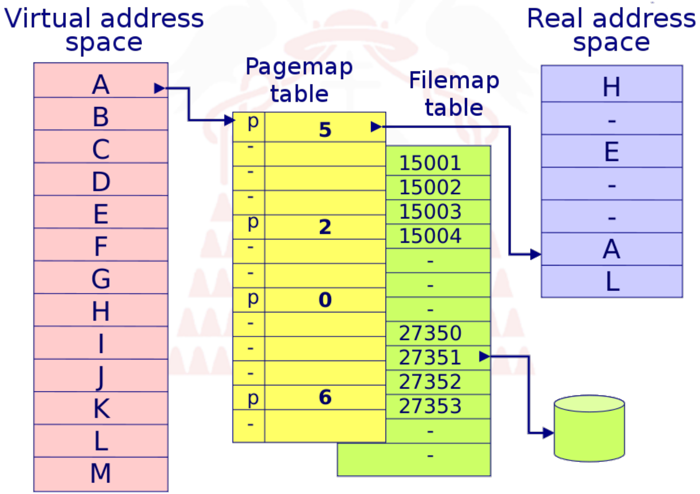
Advantages of VM

1. Transparency in transfers, storage device $\Leftrightarrow$ MM
2. The maximum size of processes depends on the virtual address space and the hard disc
3. Improves system performance $\Rightarrow$ increases multiprogramming degree
4. Reduces I/O $\Rightarrow$ only the needed parts of a program are loaded into MM

Hardware requirements of VM

- Paging and/or segmentation mechanisms are used
- Advantage of paging:
  - Simpler transfers with fixed-size blocks
    Disc $\Leftrightarrow$ MM
- Paging hardware:
  - Pagemap table management unit
  - Bits in page descriptors for: presence, modification *(dirty bit)* and reference
  - Auxiliary storage for the pages of the process
  - Support for interrupting instructions

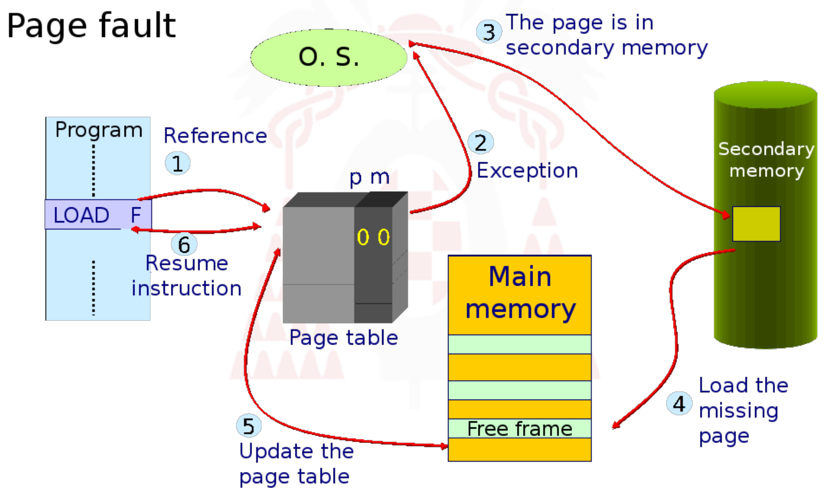
Hardware requirements of VM



Dynamic loading

- Pages transference from storage device to MM
- If the referenced page is not available in MM ⇒ Page fault (PF)
  - The page faults rate decreases as the number of frames grows
- The times that most affect the dynamic loading are:
  - Context changes
  - Storing a modified page in the disc (page out)
  - Loading a referenced into MM (page in)
- While the OS performs it, the process is blocked

Dynamic loading

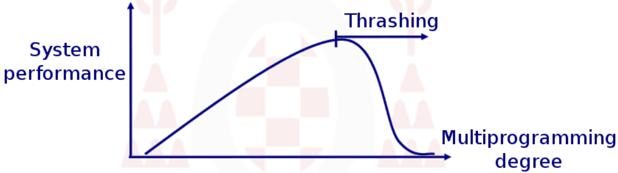


Pagers

- Part of the OS that moves pages between disc and MM
- Routines to make the transference when a PF happens
- Created and destroyed with the object mapped in VM
- Types of pager
  - File pagers
    - i.e.: `mmap, exec`
  - Anonymous objects pager or *swap pager* (for objects that don't have, themselves, an image in the file system)
    - *Swap* area management (persistent memory area for anonymous objects)
  - Device pagers
    - i.e.: *frame buffer* management
    - The memory zone used by the device is mapped (not in disc)

Thrashing (*hiperpaginación* or *vapuleo*)

- Performance drops dramatically when the degree of multiprogramming is increased
- When processes move pages: Disc ⇔ MM

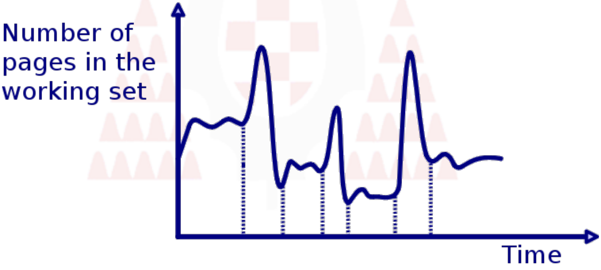


- Solutions:
  - Reduce multiprogramming degree
  - Use a local replacement algorithm
  - Provide each process the number of frames it needs
    - Working set model
    - Page fault frequency

Solutions to thrashing (i)

Working set model

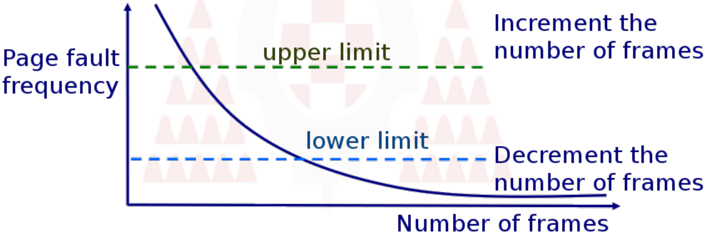
- Set of pages referenced by the process during a time interval
- Based on the principle of reference locality
- Supports high degrees of multiprogramming



Solutions to thrashing (ii)

Page fault frequency

- An upper threshold and a lower threshold are set, in order to decide when to assign frames to a process or when to take them back



Memory management algorithms (i)

- Goal ⇒ Minimize the percentage of PFs, with minimal overhead and maximum use of the MM
- Assignment policies
  What amount of MM is assigned one concrete process
  - Fixed assignment and variable assignment
  - Replacement scope: global and local
  - Free space management
- Placement policies
  Where is a MM block allocated?
- Fetching policies
  When and which pages are loaded into MM?
  - On-demand paging
  - Prefetch paging, anticipatory paging, or swap prefetch

Memory management algorithms (ii)

- Replacement policies
  Which pages should be replaced with others in MM?
  - Optimal algorithm
  - Algorithm first page arrived/first to leave *First In-First Out* (FIFO)
  - Least Recently Used page algorithm (LRU)
  - LRU approximation algorithms: global clock, FIFO second chance, Not Frequently Used page (NFU)

Assignment policies (i)

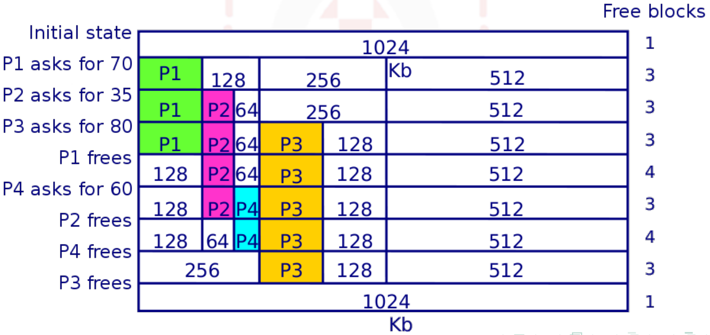
- They determine the amount of MM assigned to a process according to its needs
- Fixed assignment
  Number of frames decided on initial loading, and determined by the type of process
- Variable assignment
  The number of frames changes during the life of the process
  - Working set model
  - PFs frequency

Assignment policies (ii)

- Replacement scope
  - Global: All MM frames are candidates, regardless of the process to which they belong
    - Advantage: Better use of the MM
    - Drawback: Thrashing
    - Example: Unix
  - Local: Considers only the frames of the process that caused the PF
    - Advantage: The number of PFs is more deterministic
    - Drawback: Greater overhead $\Rightarrow$ Calculation of the frames to assign in each moment
    - Examples: VMS and Windows

Assignment policies (iii)

- Management of free space
  The OS maintains the state of free/assigned frames
  - Bitmap
  - Linked lists (Windows)
  - Buddy system (Linux, Unix)



Placement policies

- They determine where to place a block in MM
- Pure segmentation
  - First fit
  - Next fit
  - Best fit
  - Worst fit
- Paging
  - Irrelevant

Fetching policies

- They determine when and which pages are loaded into MM
- On-demand paging
  Only loaded into MM when referenced
  - Advantages
    - Only what is needed get to MM
    - Minimum overhead
- Prefetch paging, anticipatory paging, or swap prefetch
  Pages loaded into MM according to a prediction
  - Advantages
    - If the prediction is good, the execution time of processes is decreased
    - Useful when data in storage devices are accessed sequentially

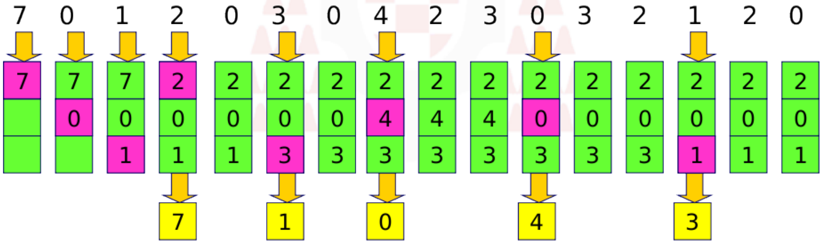
Replacement policies (i)

- They decide which pages should be replaced (with others) in MM
- Criteria
  1. Low overhead
  2. No tuning on machines with different configurations
  3. Approximation to LRU (Least Recently Used)
- Reference string
  - List of page references used as a test to evaluate the quality of these algorithms
  - Consecutive accesses to the same page count as one single reference
  - Can be obtained:
    - Artificially, with pseudo-random numbers
    - From an execution trace

Replacement policies (ii)

Optimal algorithm

- Replaces the page that will take longer to be used
- Advantage: It offers the lowest fault rate possible
  Establishes criteria for evaluation of other algorithms
- Drawback: Not implementable

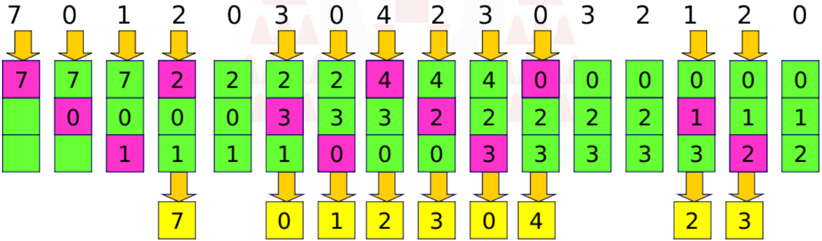


Replacement policies (iii)

FIFO algorithm

- Replaces the page that stayed longer loaded in MM
- Advantage: Easy to implement
- Drawbacks:
  - Low performance
  - Belady's anomaly ⇒ more PFs with more frames!

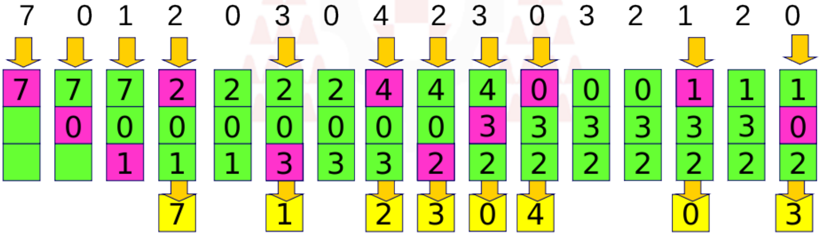
    Example: PFs in 1 2 3 4 1 2 5 1 2 3 4 5 with 3 and 4 frames



Replacement policies (iv)

LRU algorithm

- Approximates optimal. Uses recent past to predict the future
- Replaces the page less used in the recent past
- Drawbacks:
  - Difficult implementation
  - High overload
  - Solution: Use algorithms that approximate the LRU



Replacement policies (v)

LRU approximation algorithms - **Global clock**

- Uses a circular list of pages present in MM and a “reference” bit per page
  1. While the list is not full, load pages and clear their reference bit: $R = 0$
  2. When a page is referenced, set its reference bit: $R = 1$
  3. Every certain period, traverse the list with a rotatory pointer clearing ref. bits: $R = 0$
  4. If the list gets full:
     - If bit $R == 0$, replace page and advance with pointer
     - If bit $R == 1$, clear it ($R = 0$) and advance with pointer
- Example: 4.3 BSD

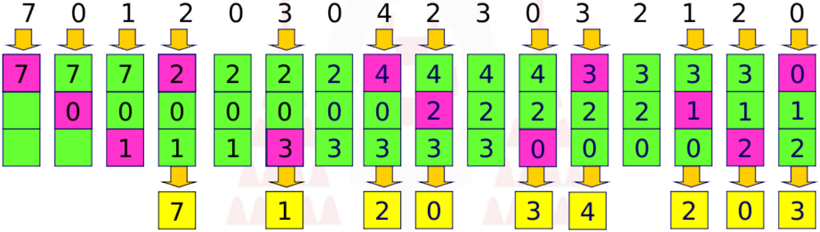
Replacement policies (vi)

LRU approximation algorithms - **FIFO with 2nd chance**

- It uses a "reference" bit associated with each page
  1. Choose a page with normal FIFO criteria
     - If R==1, just clear it (R=0)
     - If R==0, replace this page
  2. Advance with the pointer and go to step 1
- Usually implemented with a circular FIFO queue
- Advantage: Low overhead
- Drawback: Can degenerate into a normal FIFO if all pages have R==1

Replacement policies (vii)

LRU approximation algorithms - **FIFO with 2nd chance**



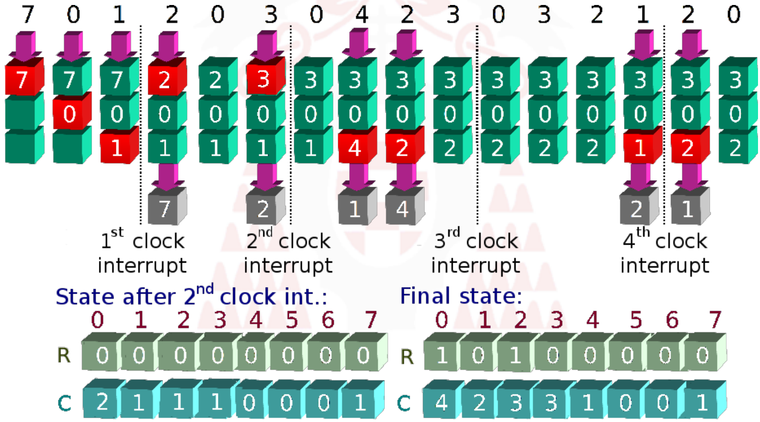
Replacement policies (viii)

LRU approximation algorithms - **NFU**

- Requires a timer (clock interrupt). Uses, per page, a counter and a "reference" bit
  - At every clock interrupt
    - If R==1, increment the counter
    - Clear all ref. bits (R=0)
  - For every PF, replace the page with the lowest counter value
- Drawback: If a page was used a lot, it might never be replaced, even though it is never used again
- Solution: Apply 'aging' mechanisms to the counters Instead of simply incrementing them
  - Shift the counters one bit to the right
  - Insert the ref. bit at the left end of the counter

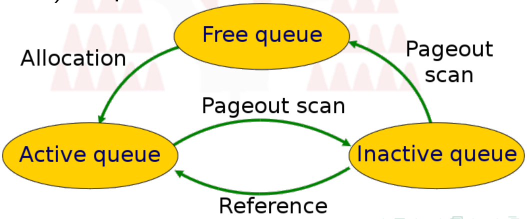
Replacement policies (ix)

LRU approximation algorithms - **NFU**



Mach 3.0

- LRU approximation with FIFO 2^{nd} chance
- Uses three page queues
  - Free pages (frames!) queue, set to 0. Replaced upon PF
  - Active pages queue. They hold the working set
  - Inactive pages queue. Buffer of non-referenced pages
- The *pageout* daemon frees pages, moving them from the active queue to the inactive queue when the number of free pages (frames) drops below a certain threshold



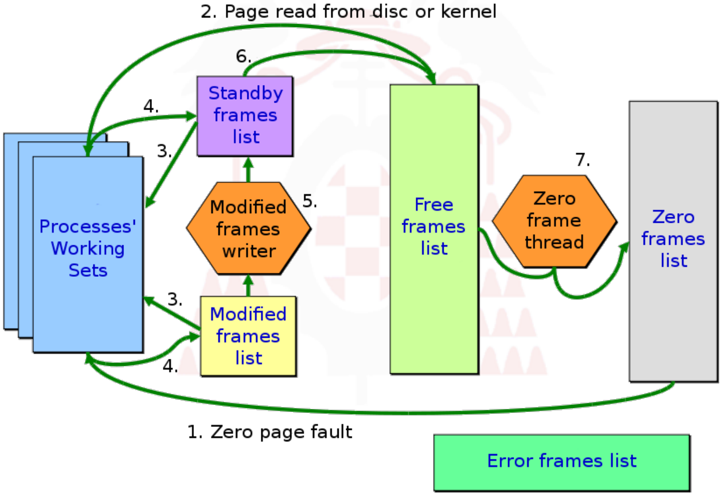
W2K (i)

- *Clustering* mechanism
  - Upon PF, page on-demand but bring several contiguous pages (a cluster) to MM
  - The number depends on the amount of MM and the type of the object that provoked the PF
  - On desktop systems the *clustering* value is 8 frames for code, 4 for data and 8 for the rest
- Management of the working set (WS) $\Rightarrow$ Number of pages of the executing process loaded in MM
  - Variable number of frames assignment; local scope
  - When a process starts, a minimum WS is assigned
  - The WS size varies dynamically between 50 and 345 pages
    - Readjusts the processes' WS and decides how many pages can be freed
    - Increments the WS size if the process causes PFs

W2K (ii)

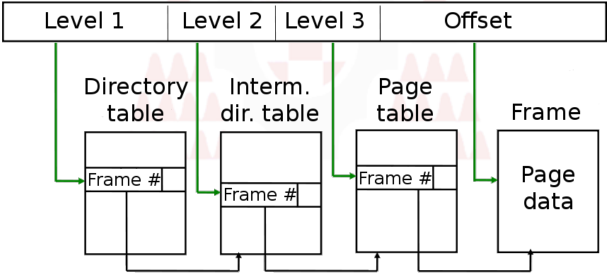
- State of a frame
  - Error - Physically damaged
  - Free - Not assigned, but can't be assigned until written with 0s
  - Empty - Initialized with 0s
  - Active - Present in a WS or blocked in memory
  - Transition - Being updated with information from disc (*collided PFs*)
  - Standby - Not in a WS anymore, but still contains data and the PMT points to it
  - Modified - Contains data that must be saved to disc
  - Modified, can't save yet - Must wait until it can be saved to disc (*metadata*)

W2K (iii)


2. Page read from disc or kernel
6.
Standby frames list
4.
3.
Processes' Working Sets
Modified frames writer
5.
Free frames list
7.
Zero frame thread
Zero frames list
3.
Modified frames list
4.
1. Zero page fault
Error frames list

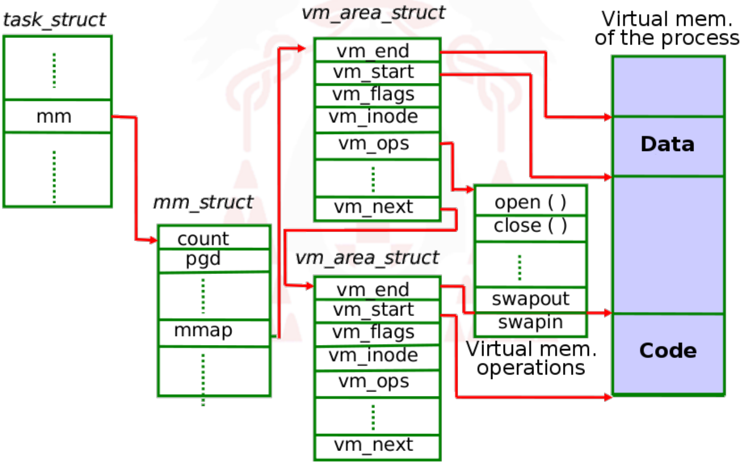
Linux (i)

- Virtual memory addressing
  - Page table with three levels and four fields: page directory, intermediate directory, page table and offset
  - Each page table occupies 1 page
  - Designed for Alpha 64 bits. Adapted to x86 (32 bits) by defining intermediate directory size $= 1$



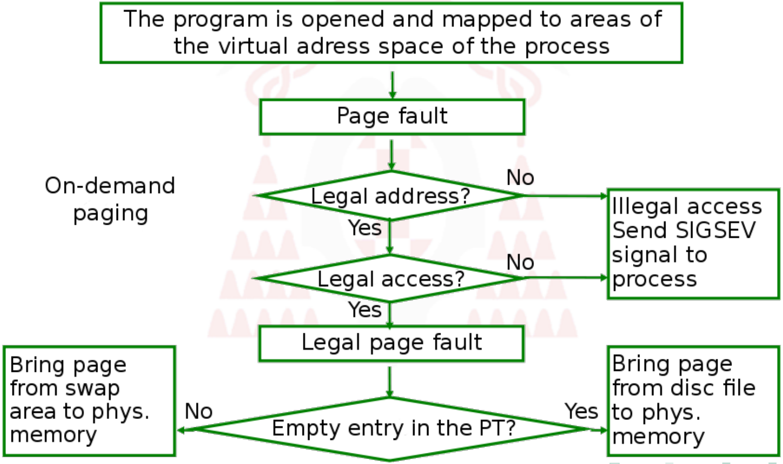
Linux (ii)

- Data structures



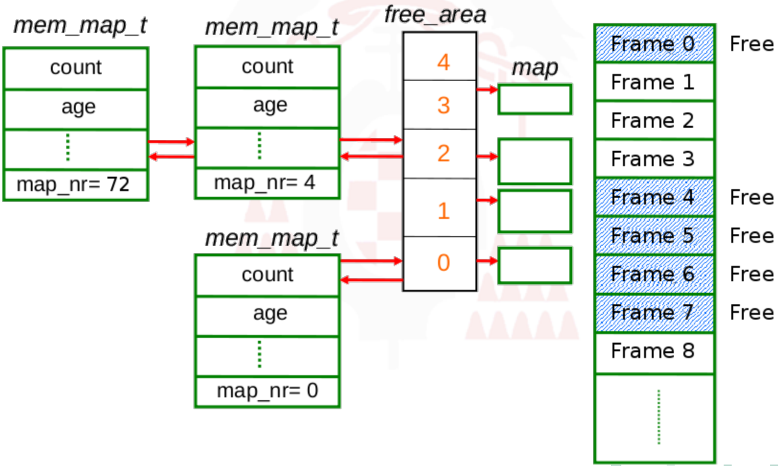
Linux (iii)

- Page faults



Linux (iv)

- Frames assignment using the buddy system



Linux (v)

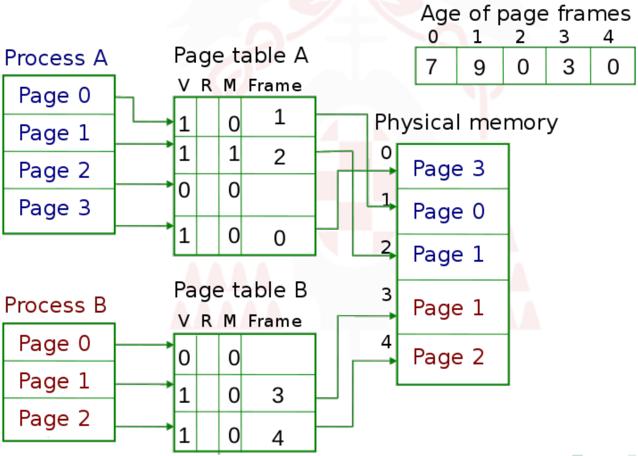
- Replacement algorithm
  - Replacement scope: global
  - Swap area is managed by daemon `kswapd`
    - At every second, it checks the number of free frames and searches for those that might be replaced
    - It's an LRU approximation algorithm with aging
    - Based on the age of the pages

Linux (vi)

- Replacement algorithm
  - Technique of pages aging
    - Every page is initialized with age 3
    - If a page is accessed (R==1), increment its age by 3, up to a maximum of 20
    - If `kswapd` is executed, decrement by 1 the age of the pages that were not used (R==0)
  - If a modified page is brought to disc
    - Mark the PMT entry as invalid
    - Include information for later recovery of the page
    - Free it, adding it to the free frames list
  - Unmodified pages are simply marked as free and added to the free frames list
  - If enough pages were recovered, the daemon will go back to sleep
  - Otherwise, it continues with the next process

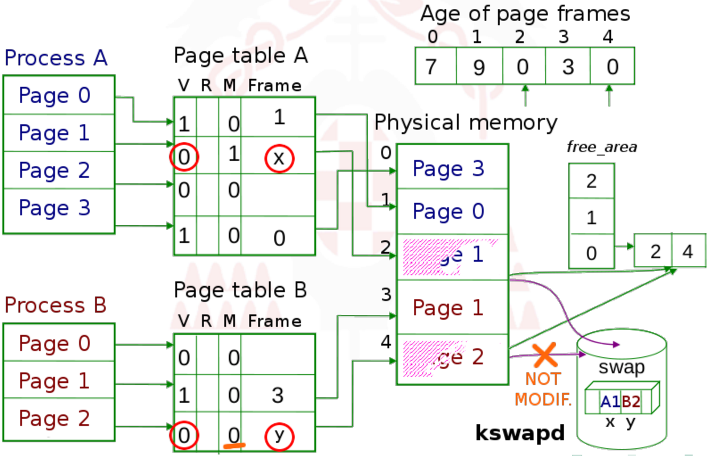
Linux (vii)

- Replacement algorithm



Linux (viii)

- Replacement algorithm



Bibliographic references

- [Sánchez, 2005] S. Sánchez Prieto.
  Sistemas Operativos.
  Servicio de Publicaciones de la UA, 2005.
- [Stallings, 1999] W. Stallings.
  Organización y arquitectura de Computadores.
  Ed. Prentice Hall, 1999.
- [Silberschatz, 2006] A. Silberschatz, P. B. Galván y G. Gagne
  Fundamentos de Sistemas Operativos.
  McGraw Hill. 2006
- [Tanenbaum, 2009] A. Tanenbaum.
  Sistemas Operativos Modernos.
  Ed. Pearson Education, 2009.