DM510: Main Memory (cont.)

Lars Rohwedder

Disclaimer

These slides contain (modified) content and media from the official Operating System Concepts slides: https://www.os-book.com/OS10/slide-dir/index.html

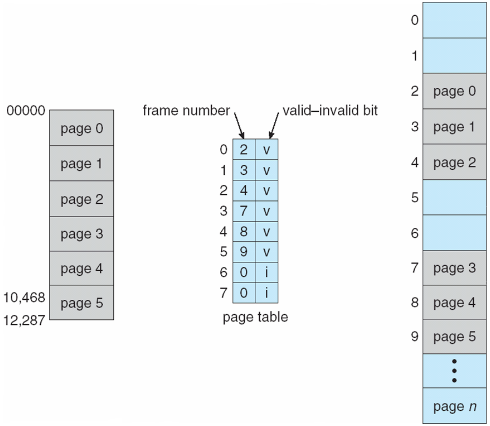
Today's lecture

- Chapter 9+10 of course book

Paging (cont.)

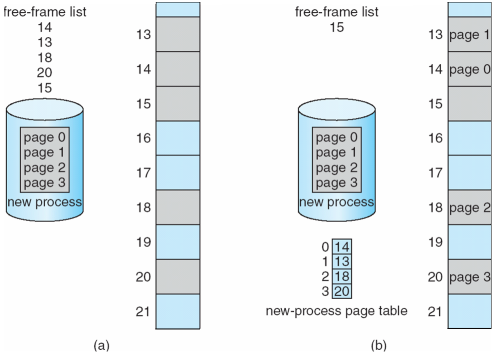
Allocation of pages

- Not all addressable pages have allocated frames (too many!). If it isn't, an invalid bit is set in page table and access results in trap $\Rightarrow$ variable and dynamic size of processes' memory



Allocation of pages

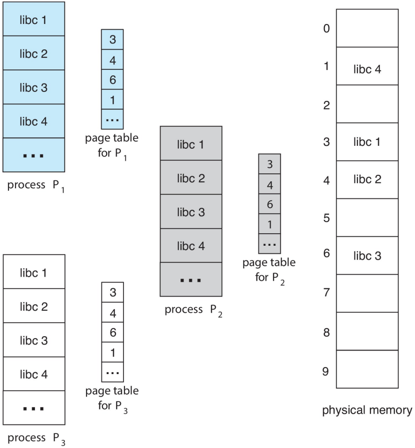
- Not all addressable pages have allocated frames (too many!). If it isn't, an invalid bit is set in page table and access results in trap $\Rightarrow$ variable and dynamic size of processes' memory
- When process is created or requests additional pages, frames are taken from a free-frame list
- When process terminates or releases pages, frames are added to a free-frame list



Before allocation　　After allocation

Shared pages

Each process has own page table, but entries may point to same frame.

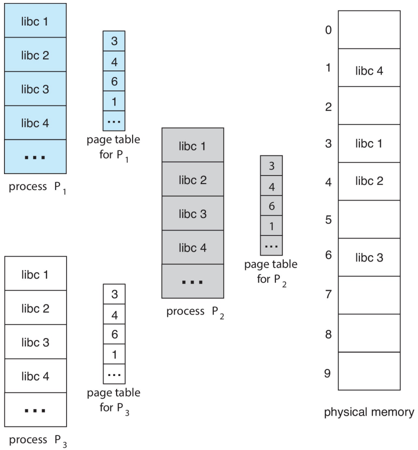


Shared pages

Each process has own page table, but entries may point to same frame.

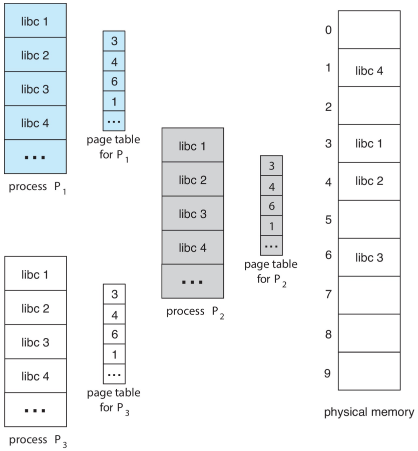
Examples 1: Shared code

- Different processes often share code, for example, standard libaries such as libc
- Save space by keeping it only once in physical memory
- Page tables usually have “read-only” bit that can be used for protection



Shared pages

Each process has own page table, but entries may point to same frame.



Examples 2: Shared memory

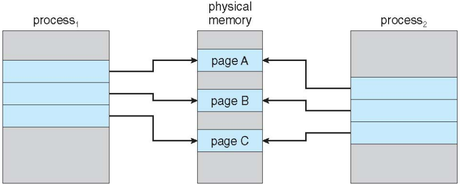
- For communication between processes, frames can be shared
- Then of course "read-only" bit should not be set

Shared pages

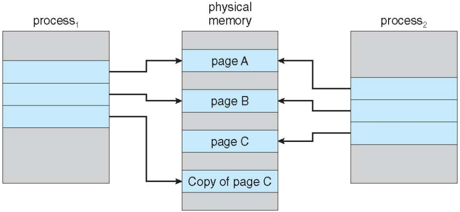
Each process has own page table, but entries may point to same frame.

Examples 3: fork()

- System call `fork()` would need to copy process' entire memory
- Often `exec()` is called immediately after `fork()` without modifying memory, making copy seem unnecessary
- Can use "copy-on-write" bit. If set, the page will be copied before it is modified.



Before write to page C (with "copy-on-write" bit)



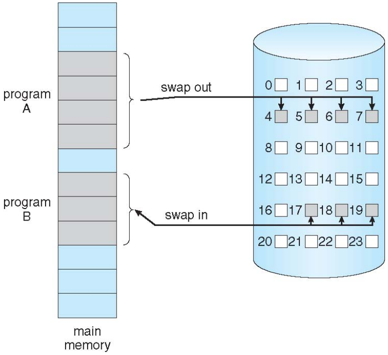
After write to page C (with "copy-on-write" bit)

Swapping

Swapping

If RAM not large enough, can move inactive processes to backing store (HDD)

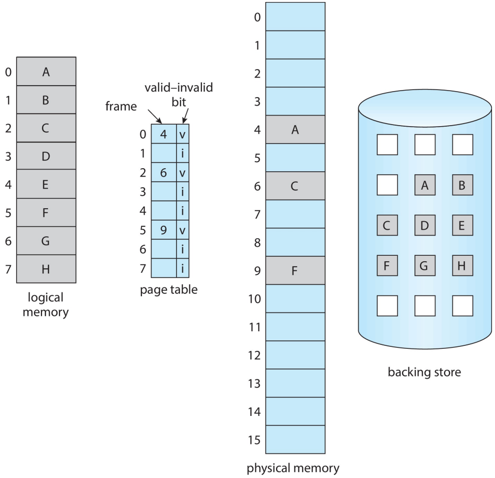
- HDD often has special **swap partition** for this purpose
- On context switch a process may need to be **swapped in** (brought back to RAM) or **swapped out**
- Transfer times (moving data between RAM and HDD) high, sometimes seconds. Typically employed only in extreme cases
- Rarely used with SDDs (e.g. on mobile devices) since SDDs wear off with many writes



Virtual Memory

Overview of virtual memory

- As in swapping, total memory of processes may be larger than available physical memory and backing store is used
- Data of pages in physical memory, backing store (HDD/SDD), or both

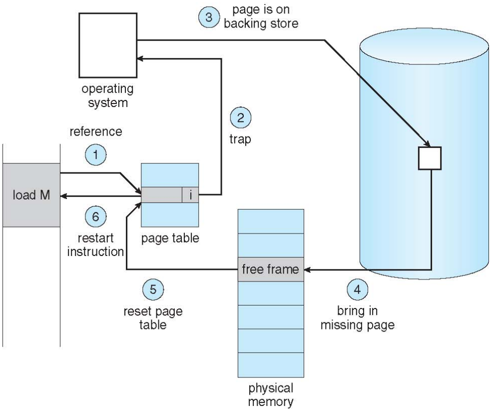


Overview of virtual memory

- As in swapping, total memory of processes may be larger than available physical memory and backing store is used
- Data of pages in physical memory, backing store (HDD/SDD), or both

Page fault

- If page's invalid bit is set, access leads to **page fault** trap
- Then either page does not exist or needs to be moved from backing store to a free frame



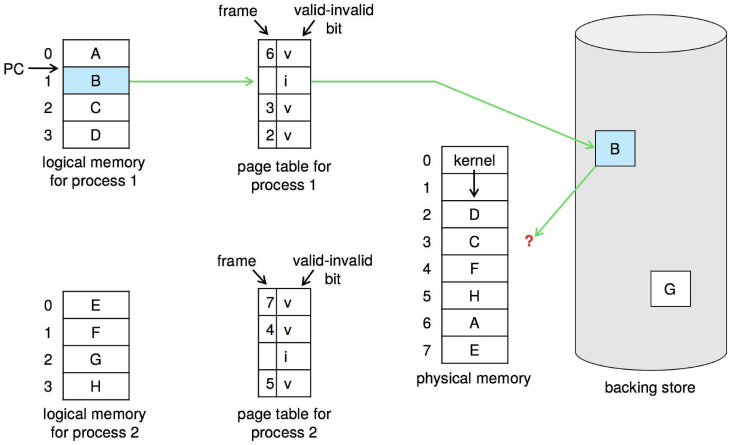
Demand paging

- When program is loaded, some data (e.g. instruction code) usually need to be brought into memory
- **Pure demand paging:** bring page into memory only when accessed (upon page fault)
- **Prepaging:** bring some pages already into memory to avoid large number of page faults initially

Page replacement

Need of page replacement

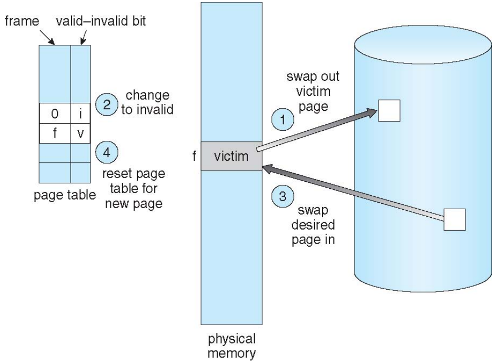
Page may be needed in main memory, but no free frame available



Need of page replacement

Page may be needed in main memory, but no free frame available

- Need to select **victim** page that is swapped out
- If victim page was not modified, might not be necessary to write it to backing store. This can be checked using a **dirty** bit in page table that is set when page is written to
- Requested page then takes the victim's frame



But how to choose the victim page?

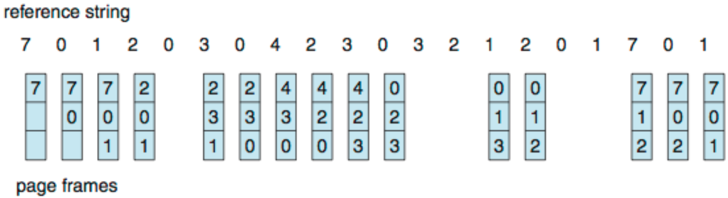
Page replacement algorithm

- **Goal:** choose victim pages to minimize page faults
- Evaluated for a fixed number of frames and a **reference string** containing a list of page numbers referenced by process, e.g., 3 frames and reference string

$$7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1$$

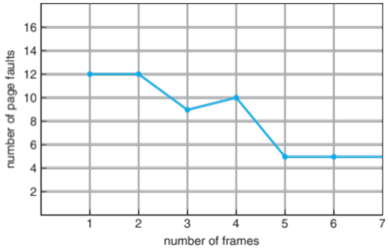
First-in-first-out (FIFO)

- Victim is the page that has been longest in memory
- Example (3 frames):



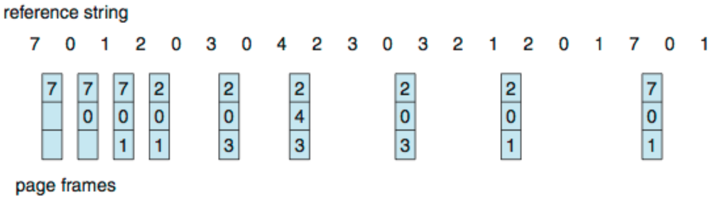
Belady's Anomaly

- More frames may lead to more page faults
- Example: 1,2,3,4,1,2,5,1,2,3,4,5 (see graph on the right)



Optimal algorithm

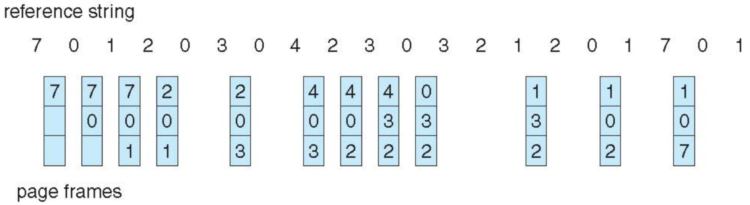
- victim is page that will not be used for the longest ⇒ minimizes page faults
- Example (3 frames):



- Impossible to implement (no knowledge of future)
- Used to judge performance of other algorithms
- Does not suffer from Belady's Anomaly: the pages in memory when using more frames are always a superset of those when using fewer frames

Least-recently-used (LRU)

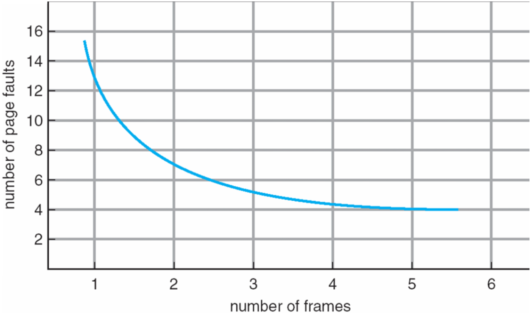
- Victim is the page that has not been used for the longest
- Example (3 frames):



- Generally performes well
- Not very efficient to implement $\Rightarrow$ often faster approximations of LRU used
- (As in optimal algorithm) Does not suffer from Belady's Anomaly: the pages in memory when using more frames are always a superset of those when using fewer frames

Frame allocation

- Each process typically has a minimum number of frames, algorithm for determining exact number varies greatly
- **Global replacement:** victim page is chosen from all frames
- **Local replacement:** victim page is chosen only from process's own frames



Frame allocation

- Each process typically has a minimum number of frames, algorithm for determining exact number varies greatly
- **Global replacement:** victim page is chosen from all frames
- **Local replacement:** victim page is chosen only from process's own frames

Thrashing

- When number of processes increases, number of frames per process decreases
- Recently swapped out page are quickly requested again
- Most time transferring between RAM and backing store $\Rightarrow$ low CPU utilization