Main Memory II

DM510 Operating Systems

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

Advanced page tables

Invalid flag

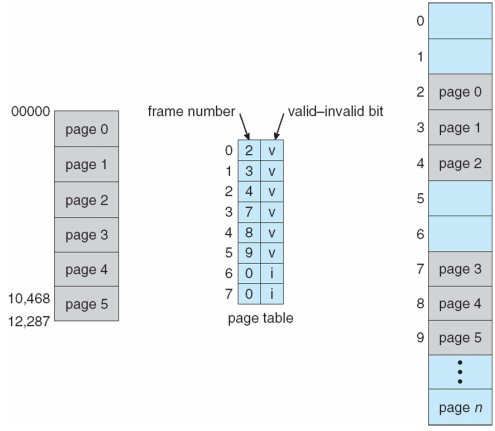
Motivational example: In a 32-bit system, the total addressable memory of a process is $2^{32}B = 4GB$. In a typical system, there is not enough physical memory to give every process $4GB$ of physical memory

Invalid flag

Motivational example: In a 32-bit system, the total addressable memory of a process is $2^{32} B = 4GB$. In a typical system, there is **not enough physical memory** to give every process $4GB$ of physical memory

- Not all addressable pages have actual frames mapped to them
- If a page is not mapped to a frame, an invalid bit is set in page table and access results in interrupt

  $\Rightarrow$ This allows us to add and remove memory from processes **dynamically**



Invalid flag

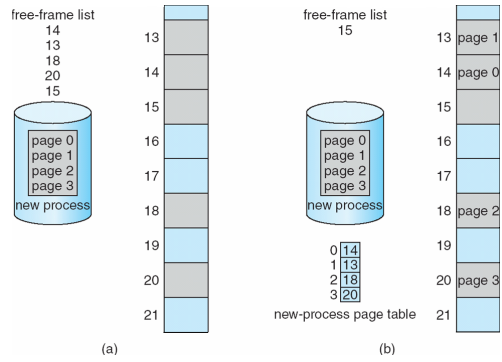
Motivational example: In a 32-bit system, the total addressable memory of a process is $2^{32} B = 4GB$. In a typical system, there is **not enough physical memory** to give every process $4GB$ of physical memory

- Not all addressable pages have actual frames mapped to them
- If a page is not mapped to a frame, an invalid bit is set in page table and access results in interrupt

  $\Rightarrow$ This allows us to add and remove memory from processes **dynamically**

Free frame list

- 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



Large address spaces

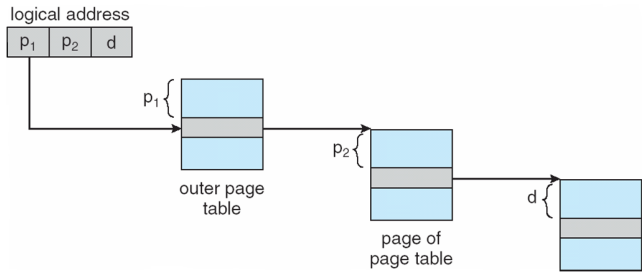
In modern systems, 64-bit addresses are most common. Here a simple page table is not sufficient:

- The page table needs to fit into memory, so the number of pages must be $\ll 2^{30}$ (very rough estimate)
- Each page is of size 2^{64}/<no. pages>, which would be Gigabytes with the previous estimate

In the next slides we look at possible solutions

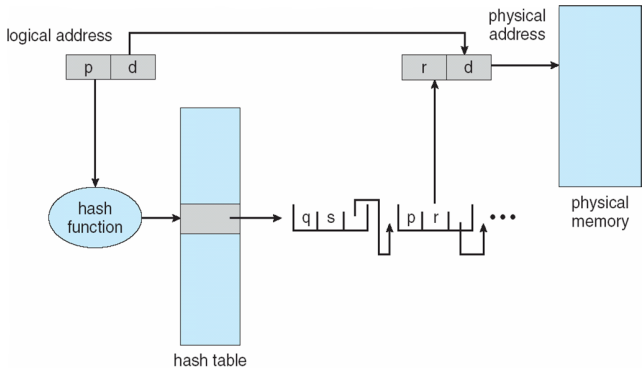
Hierarchical page table

- Use first bit section to index outer page table, which contains pointer to inner page table indexed by second bit section
- Tradeoff: The more nested tables the more physical memory accesses per logical memory access
- At each level, we can use an invalid bit to invalidate range of pages



Hashed page table

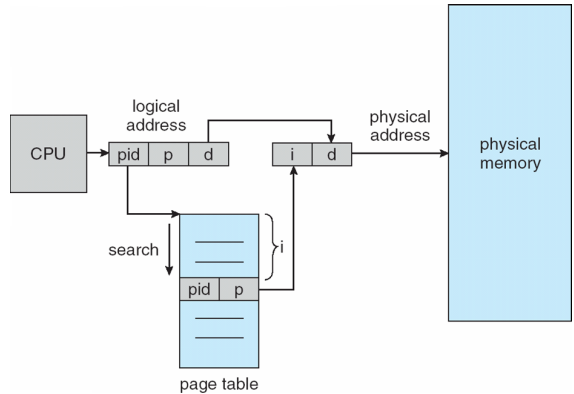
- Use hash table to map logical to physical addresses
- When a page is not found in the hash table, it is considered invalid



Inverted page table

The physical memory is often much smaller than the logical address space

- We can use a page table with one entry per frame instead of one per page and process. The entry needs to store the id of the process owning the frame
- Disadvantage: requires searching through table, since we cannot index it based on logical address



Shared memory

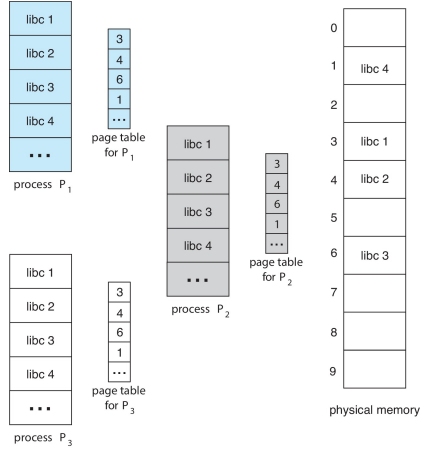
Shared frames

Using paging it is easy to implement shared memory sections between different processes:

- Each process has its own page table, but entries may point to same frame
- This easily implements communication via shared memory
- We discuss several variants of this next

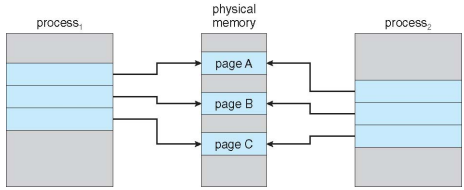
Shared code

- Different processes often share code, for example, shared libaries such as libc
- Keeping it only once in physical memory saves memory
- Page tables usually have **read-only** bit that can be used for protection

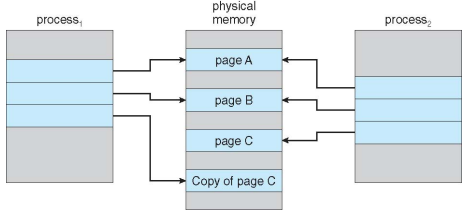


Fork

- System call `fork()` would normally need to copy entire memory of parent process
- Often `exec()` is called immediately after `fork()` without modifying memory, making copy seem unnecessary
- Page tables often have a 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)

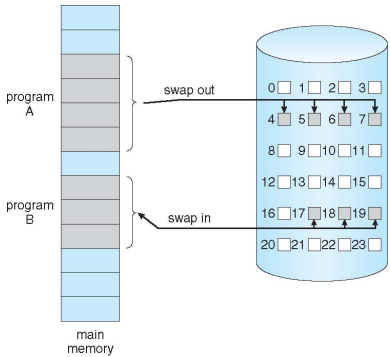
Virtual Memory

Swapping

Virtual memory is when memory used by processes does not necessarily exist in physical memory. A simple example is swapping

If RAM not large enough, we may move inactive processes to a backing store on a hard disk drive (HDD)

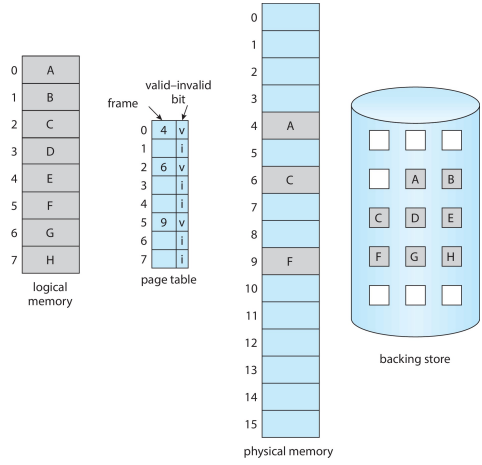
- HDD often has special **swap partition** for this purpose
- On context switch a process may need to be **swapped in** (brought back to RAM) or **swapped out**
- Transfer times (moving data between RAM and HDD) high, sometimes seconds. Typically employed only in extreme cases



Virtual memory with active processes

As a more aggressive variant of virtual memory than swapping, we may keep processes running even if part of their pages is not in physical memory

- Data of pages in physical memory, backing store (HDD/SDD), or both
- What happens when a process accesses a page not in physical memory?



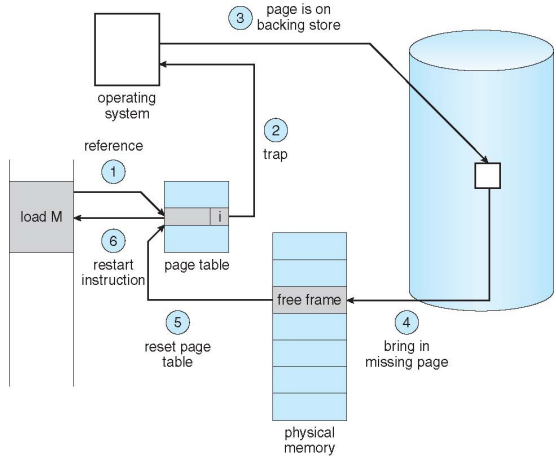
Virtual memory with active processes

As a more aggressive variant of virtual memory than swapping, we may keep processes running even if part of their pages is not in physical memory

- Data of pages in physical memory, backing store (HDD/SDD), or both
- What happens when a process accesses a page not in physical memory?

Page fault

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



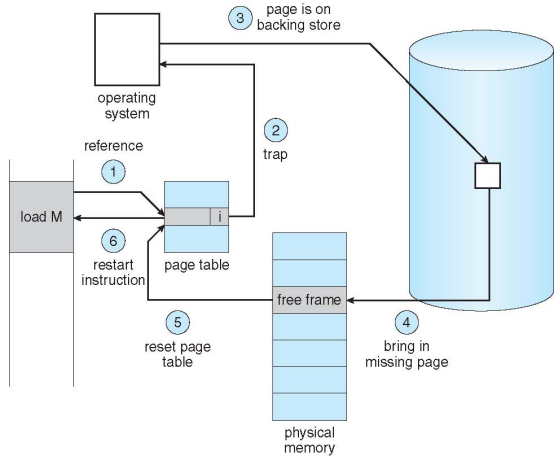
Virtual memory with active processes

As a more aggressive variant of virtual memory than swapping, we may keep processes running even if part of their pages is not in physical memory

- Data of pages in physical memory, backing store (HDD/SDD), or both
- What happens when a process accesses a page not in physical memory?

Page fault

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



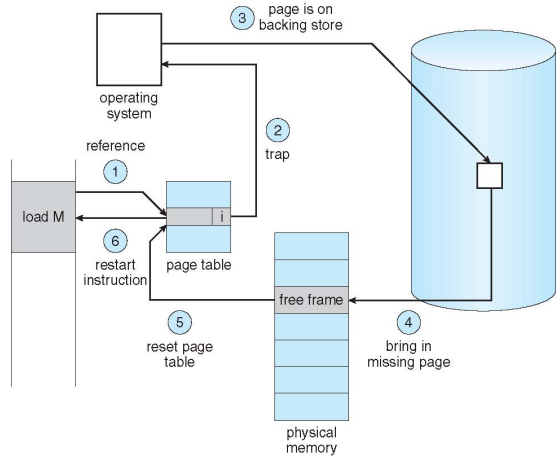
Virtual memory with active processes

As a more aggressive variant of virtual memory than swapping, we may keep processes running even if part of their pages is not in physical memory

- Data of pages in physical memory, backing store (HDD/SDD), or both
- What happens when a process accesses a page not in physical memory?

Page fault

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



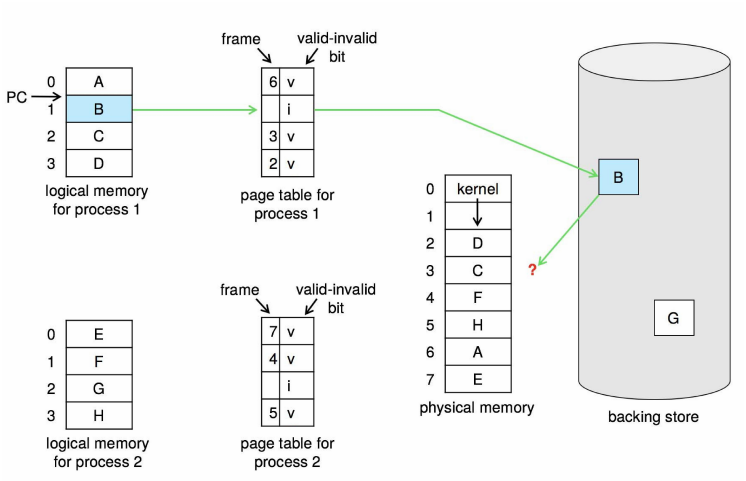
This way we can save space on rarely used memory. It also allows for rapid program loading:

- When program is loaded, some data (for example, instruction code) usually need to be brought into memory
- **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

Motivation

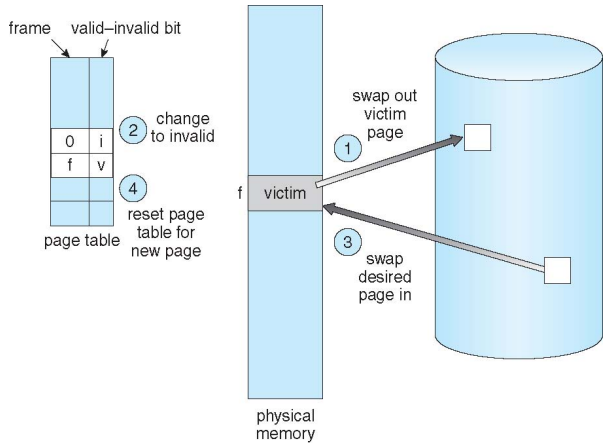
Page may be needed in main memory, but no free frame available



Motivation

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, it may 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?

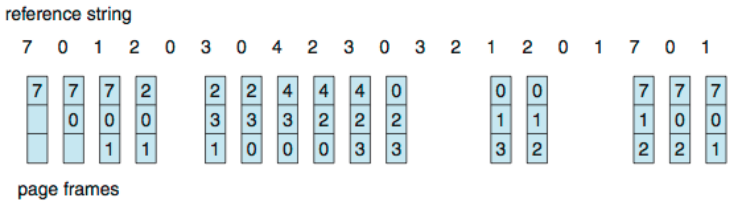
Page replacement algorithms

- **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, for example, 3 frames and reference string

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

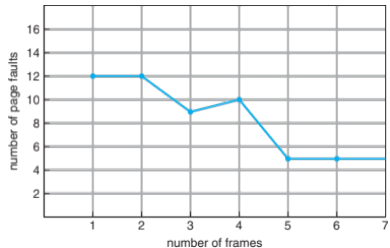
First-in-first-out (FIFO)

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



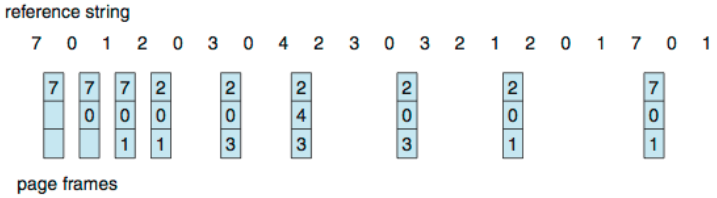
Belady's Anomaly

- 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

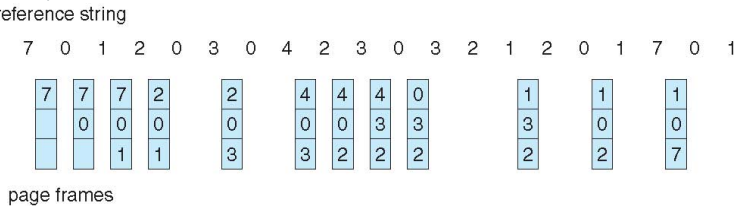
- victim is page that will not be used for the longest ⇒ minimizes page faults
- Example (3 frames):



- 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)

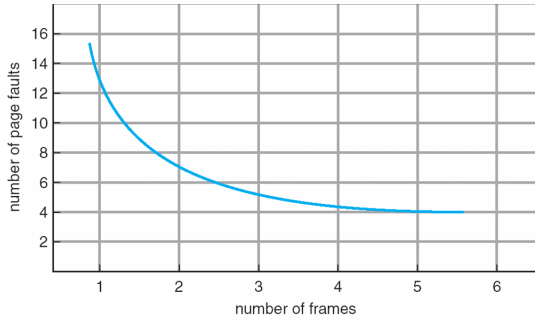
- Victim is the page that has not been used for the longest
- Example (3 frames):



- Generally performes well
- Not very efficient to implement ⇒ 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 pages are quickly requested again
- Most time transferring between RAM and backing store ⇒ low CPU utilization