A CLOCK Page Replacement Implementation

Daniel Gonzalez

Using uint32_t As Virtual Pointers

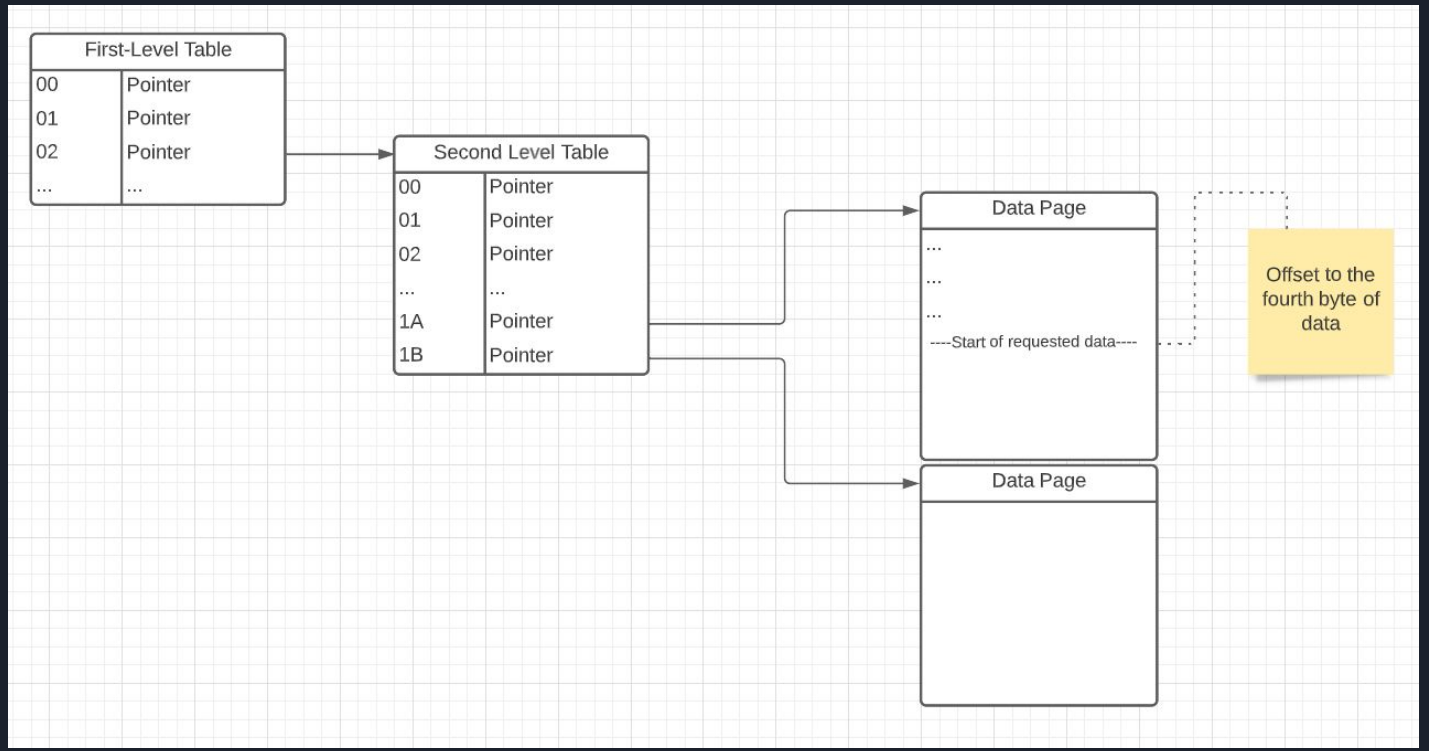
- 4 bytes of data
- Each byte is an offset into a page
- Page size must be 256 bytes to accommodate one byte offsets

Example: 137732

Bytes (in big endian):

0x00	0x02	0x1A	0x04
(Unused)	(Offset to first page)	(Offset to second page)	(Offset to data page)

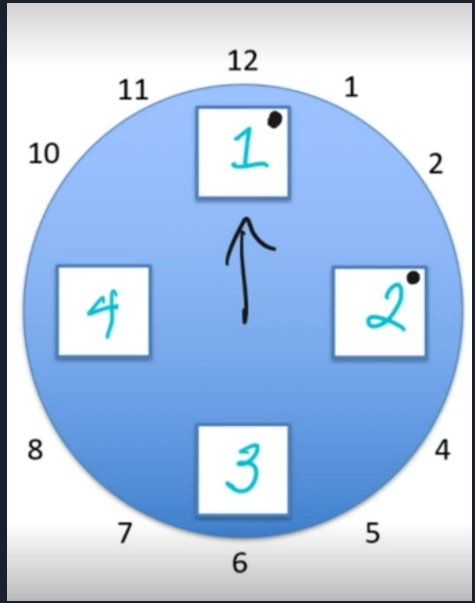
0x00 0x02 0x1A 0x04



CLOCK Page Replacement

Requirements:

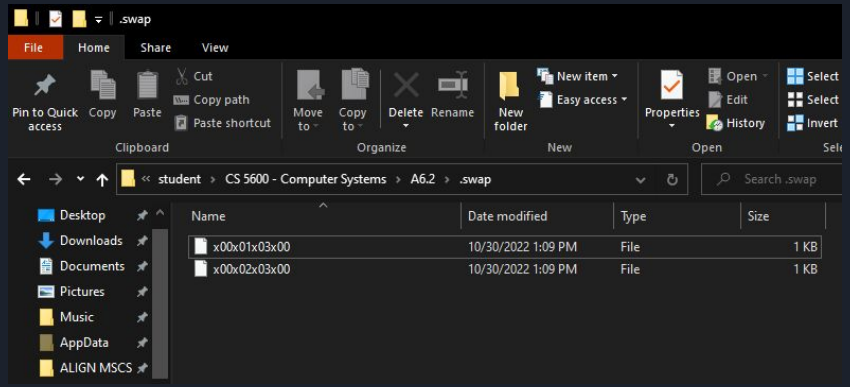
- Access boolean
- Implement a circular queue
- Track if a page is on disk or in memory
- Evict pages in the middle of the queue



Metadata

- 4 bytes of metadata (another uint32_t)
- Metadata: “Present byte”, “Next Page Allocated?”, “Access byte”, “Dirty byte”
- Doubly linked list
- Use offsets, not actual pointers
- Present byte = 1 in memory, 2 in disk, 0 if unused
- File names are page entry offsets (basically the virtual pointers!)

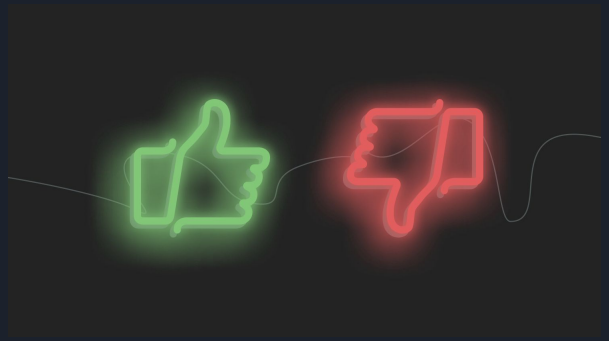
New Page Table Entries		
Metadata (4 bytes)	Queue Data (4 bytes)	Pointer (8 bytes)
0x01 0x01 0x01 0x00	0x01 0x01 | 0x01 0x05	0x6dfed4
0x01 0x00 0x01 0x00	0x01 0x04 | 0x01 0x00	0x6dffba
0x00 0x00 0x00 0x00	0x00 0x00 | 0x00 0x00	0x00000
0x02 0x01 0x00 0x01	0x01 0x01 | 0x00 0x00	0x00000
0x01 0x01 0x00 0x00	0x01 0x05 | 0x01 0x01	0x5dfeba
0x01 0x00 0x01 0x00	0x01 0x00 | 0x01 0x04	0x4dfecb



How Will This Work?

- Page is requested, page is added to the queue, points to itself
- New page added, page is added to the end of the queue, points to first page
- New pages continually added to the end in the same manner
- First page needs to be evicted, check access bit, iterate through queue until access bit is 0
- While iterating, if access bit is 1, set to 0
- Finally, evict a page. Update prev/next pointers in the circular queue
- Write page to disk
- The new entry that was requested can now be used in memory
- User requests use of page, check present bit - it is in disk, read file, write it to memory
- Set present bit to 1
- Rinse and repeat

Pros and Cons



Pros:

- Pointers still work intuitively
- Queue structure included as part of page table
- Multiple files are easy to demonstrate and debug
- Multiple files also prevents accidental crossover and overwriting

Cons:

- 16 entries, 256 byte page tables!
- Byte unused in pointers, dirty byte not utilized in metadata, two bytes of waste
- Could potentially accomplish with a singly linked list and save space
- Multiple files is less elegant of a solution than just one