
CS 241 System Programming

Virtual Memory

Discussion Section 10
10 April – 13 April

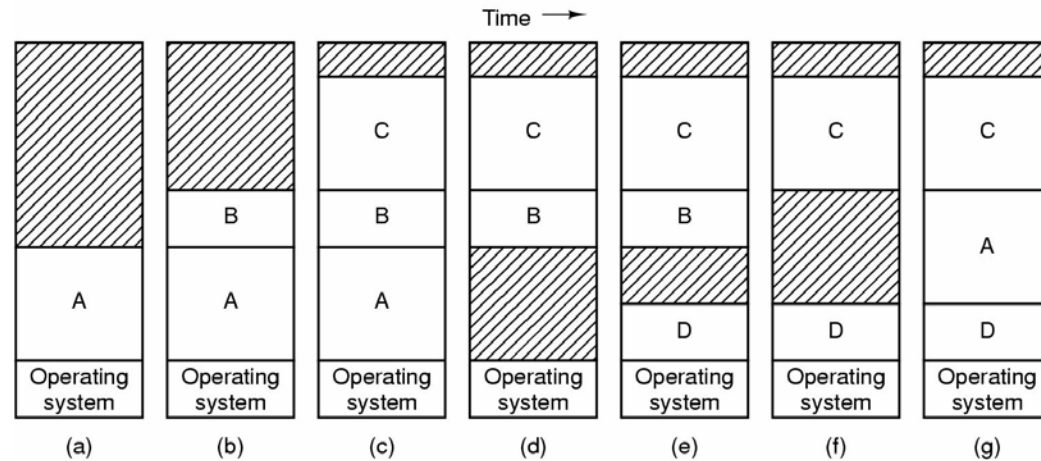
Outline

- MP4 Compaction
 - Uses of Virtual Memory
 - Virtual Memory Addressing
 - Examples
 - Page Replacement Algorithms
-

Compaction

- After numerous MyMalloc and MyFree calls, our memory will have many holes
 - Total free memory is much greater than that of any contiguous chunk
 - And for MP4 we're just doing contiguous allocations, not any kind of paging
 - We should compact our memory
 - Shift all allocations to one end of memory, and all holes to the other end
 - Temporarily rids of external fragmentation
-

Compaction (example)



- Lucky that A fit in there, may want to compact at (d), (e), or (f)
- Sadly, compaction is very costly
 - How costly?
 - How else can we eliminate external fragmentation?

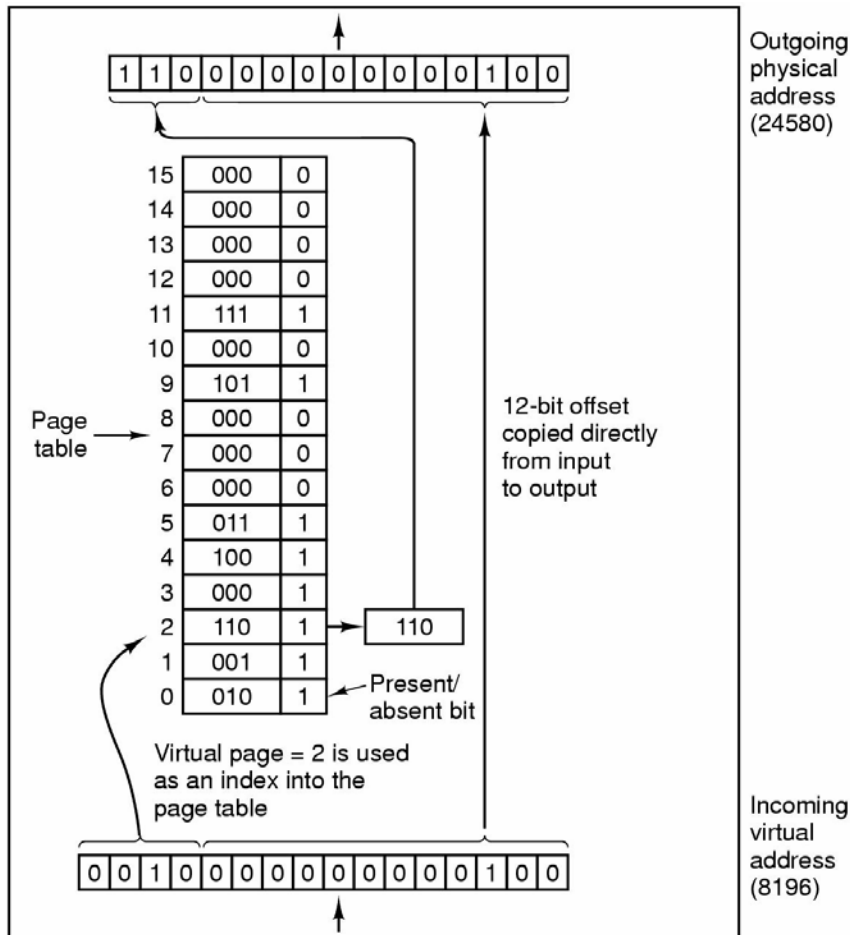
Paging

- Divide memory into pages, all of equal size
 - We don't need to assign contiguous chunks
 - Internal fragmentation can only occur on the last page assigned to a process
 - External fragmentation cannot occur at all
 - Process 4 may be using pages 3, 12, and 89
-

Virtual Memory

- RAM is expensive (and fast)
 - Disk is cheap (and slow)
 - Need to find a way to use the cheaper memory
 - Store memory that isn't frequently used on disk
 - "Swap"
 - Still uses pages
 - Process 3 (running) uses pages 1, 5, and 19
 - Process 4 (suspended) uses pages 3, 12, and 89
 - If we only can fit 4 pages in RAM, which should be there?
-

Page Table



- Keeps track of what pages are in memory
- Provides a mapping from virtual address to physical address

Handling a Page Fault

- Suspend process 3, run process 4
 - Process 4 wants all of its pages
 - How do we get these back into memory?
 - Page fault
 - Look for an empty page in RAM
 - May need to write a page to disk and free it
 - Load the faulted page into that empty page
 - Modify the page table
-

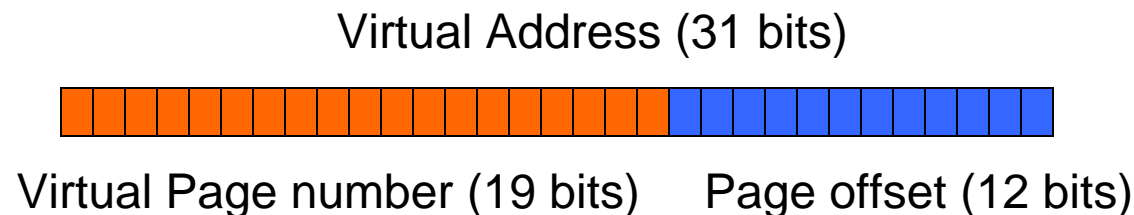
Addressing

- System has 64MB RAM (2^{26})
 - We have 32bit addresses
 - So we can address up to 2^{32} bytes of memory
 - (some 32bit systems use 64bit memory)
 - Page size of 4KB (offset of 4KB)

 - Using 31bits, how large can memory be?
 - How many bits for page address? How many for offset?
-

Addressing

- 64MB RAM (2^{26})
- 2^{31} (2GB) total memory
- 4KB page size (2^{12})
- So we need 2^{12} for the offset, we can use the remainder bits for the page
 - 19 bits, we have 2^{19} pages (524288 pages)

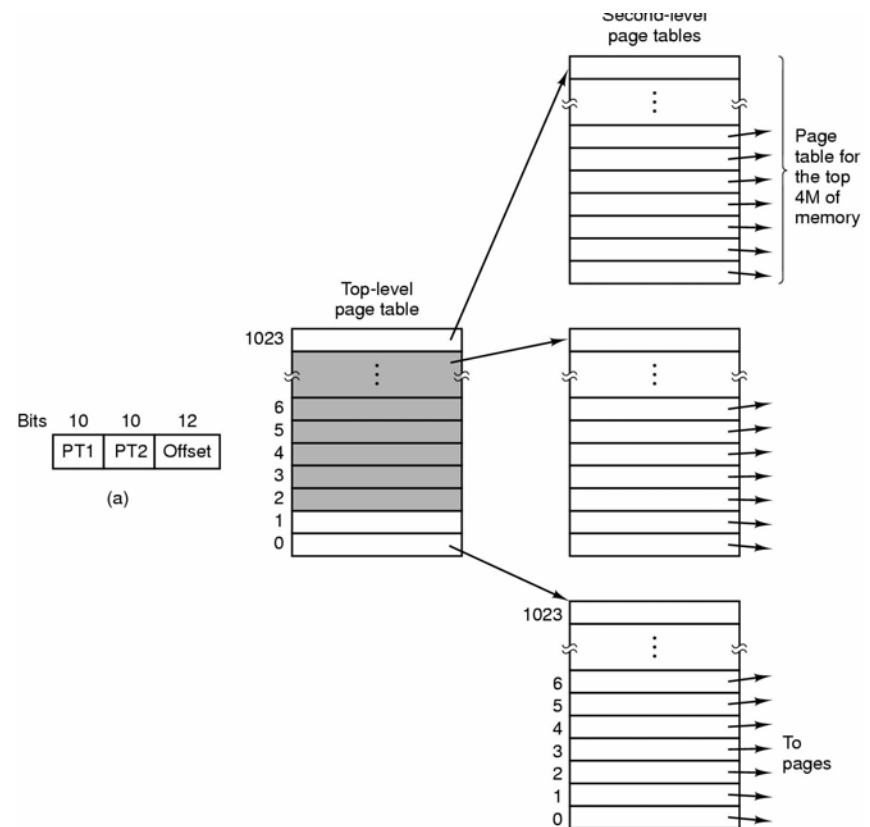


Address Conversion

- That 19bit page address can be optimized in a variety of ways
 - Translation Look-aside Buffer
 - m – memory cycle, α - hit ratio, ε - TLB lookup time
 - Effective access time (Eat)
 - $Eat = (m + \varepsilon)\alpha + (2m + \varepsilon)(1 - \alpha) = 2m + \varepsilon - m\alpha$
 - Multilevel Page Table
 - Similar to indirect pointers in I-nodes
 - Split the 19bits into multiple sections
 - Inverted Page Table
 - Much smaller, but is slower and more difficult to lookup
-

Multilevel Page Tables

- 20 bits of page addressing
 - First 10 indexes the top-level page table
 - Then we follow the pointer and index that table
 - Use the second 10 bits to find out the actual physical address
- Load page into memory accordingly



More Addressing

- Get comfortable with these calculations
 - 128MB RAM, 16KB page size
 - 32bit addresses, use all bits
 - 1GB RAM, 128KB page size
 - 64bit addresses
 - only use 40 of the 64 bits for addresses
 - How large of a chunk of hard disk do we need to set aside in each of these cases?
-

Page Replacement Policies

- Not every process can have their pages in memory at a time
 - But whose shall we keep?
 - Optimal – last to be used next = removed first
 - FIFO – Inserted first = removed first
 - LRU – used least recently = removed first
 - NRU – approximation using bit shifting
 - Second chance – approximation using single bit
 - Would also be useful to know the dirty pages
-

Page Replacement Strategies

- Takes two disk operations to replace a dirty page
 - We keep track of dirty bits, so we should attempt to replace clean pages first
 - Write dirty pages to disk during idle disk time
 - Then we can clear the dirty bit
 - Need to approximate optimal strategy, but can't because we never know what order a process will use its pages
 - Best we can do is run a program multiple times, and track what memory it uses
-

Summary

- Slow to use contiguous allocations + compaction
 - Very fast (and only one extra lookup) to use virtual memory
 - Can also address a much larger space than we have physical memory
 - No external fragmentation
 - Use the right page replacement algorithm
-