Review of Memory Management, Virtual Memory

CS448

Memory Management

- In a Multiprogramming System
  - Each user could run many processes
  - Each process has access to some section of memory
  - Memory requirements of the processes might exceed that of actual physical memory
  - Operating System plus the hardware has the task of managing memory
- Options
  - Swapping
  - Partitioning
  - Paging

Swapping

- Problem: I/O is so slow compared with CPU that even in multi-programming systems, the CPU can be idle most of the time
- Solutions:
  - Increase main memory
    - Expensive
    - Leads to larger programs
  - Swapping

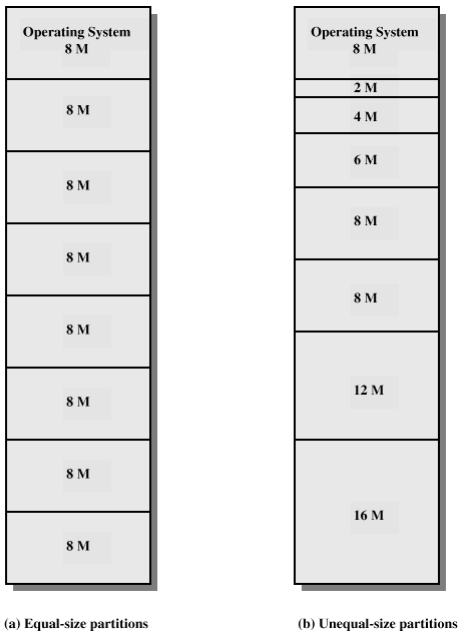
What is Swapping?

- Long term queue of processes stored on disk
- Processes “swapped” in as space becomes available
- As a process completes it is moved out of main memory
- If none of the processes in memory are ready (i.e. all I/O blocked)
  - Swap out a blocked process to intermediate queue
  - Swap in a ready process or a new process
- But…
  - Swapping is an I/O process... Potential to make things worse with big processes
  - Doesn’t allow us to execute programs larger than physical memory

Partitioning

- Splitting memory into sections to allocate to processes (including Operating System)
- Fixed-sized partitions
  - May not be equal size
  - Process is fitted into smallest hole that will take it (best fit)
  - Some wasted memory
  - Leads to variable sized partitions

Fixed Partitioning



(a) Equal-size partitions

(b) Unequal-size partitions

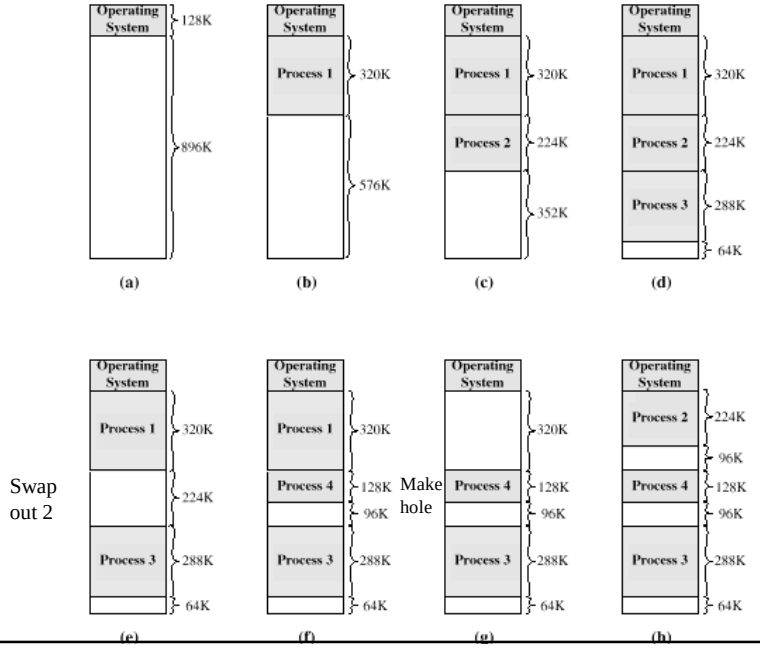
Variable Sized Partitions (1)

- Allocate exactly the required memory to a process
- This leads to a hole at the end of memory, too small to use
  - Only one small hole - less waste
- When all processes are blocked (or time-slice is up), swap out a process and bring in another
- New process may be smaller than swapped out process
- Another hole

Variable Sized Partitions (2)

- Eventually have lots of holes (fragmentation)
- Solutions:
  - Coalesce - Join adjacent holes into one large hole
  - Compaction - From time to time go through memory and move all hole into one free block (c.f. disk de-fragmentation)
  - Time consuming!

Effect of Dynamic Partitioning



Add back 2 in a different memory location!

Need logical addresses

Relocation

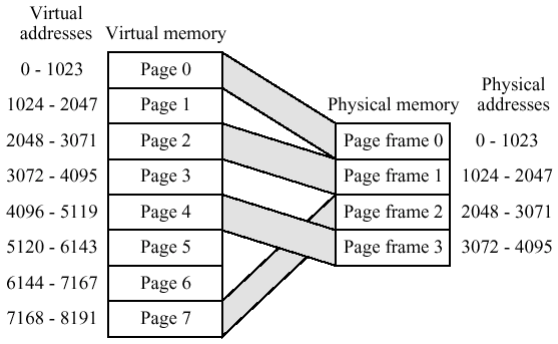
- No guarantee that process will load into the same place in memory
- Instructions contain addresses
  - Locations of data
  - Addresses for instructions (branching)
- Logical address - relative to beginning of program
- Physical address - actual location in memory (this time)
- Automatic conversion using base address

Paging

- Most common technique used today, superior to the previous two schemes

- Split memory into equal sized, small chunks
  - Called **pages**, **frames**, or **blocks**
- Split programs (processes) into equal sized pages
- Allocate some number of page frames to a process
- Operating System maintains list of free frames
- A process does not require contiguous page frames
  - Use page table to keep track
  - With **virtual memory**, pages could be on disk

Allocation of Pages



Allocation of Free Frames

Each process given their own Page Table. Here, add Process A:

Free Frame List
13
14
15
18
20

Process A:
Page 0
Page 1
Page 2
Page 3

Memory Frames

…	
13	Free
14	Free
15	Free
16	Used
17	Used
18	Free
19	Used
20	Free
21	Used
…	

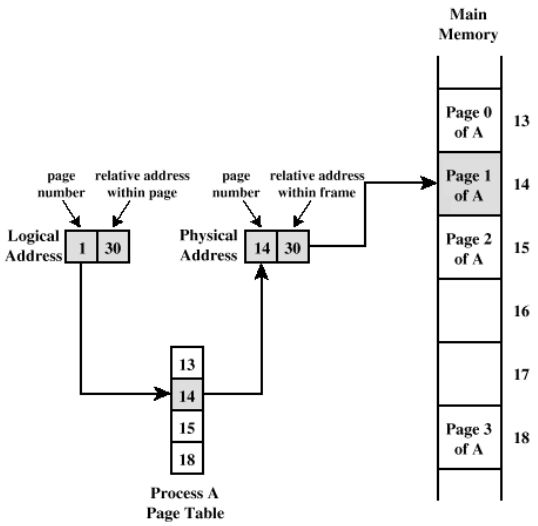
Free Frame List
20

Process A:
Page 0 : 13
Page 1 : 14
Page 2 : 15
Page 3 : 18

Memory Frames

…	
13	A: Page 0
14	A : Page 1
15	A : Page 2
16	Used
17	Used
18	A : Page 3
19	Used
20	Free
21	Used
…	

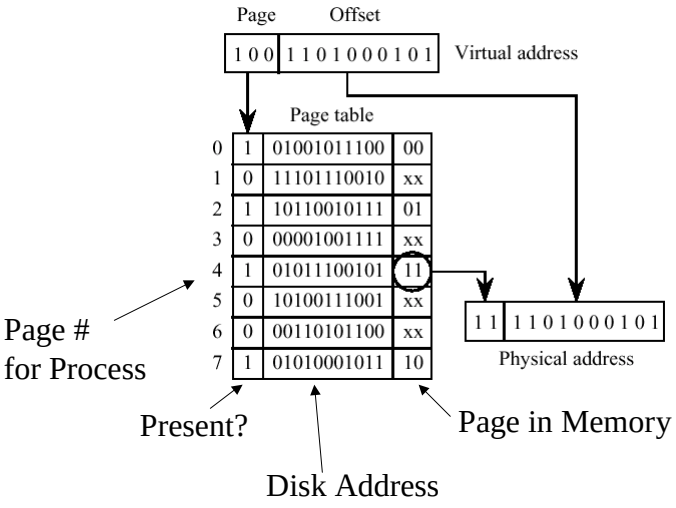
Logical and Physical Addresses - Paging



Virtual Memory

- Demand paging
  - Do not require all pages of a process in memory
  - Bring in pages as required
- Page fault
  - Required page is not in memory
  - Operating System must swap in required page
  - May need to swap out a page to make space
  - Select page to throw out based on recent history
    - Just like caching!
  - Needless to say this is very time consuming! Good thing we can process-switch to something else that is hopefully ready to do work, while we wait for a page to load from disk (many cycles)

Virtual Addressing



Thrashing

- What if we have a large number of processes running with too little memory?
  - Operating System spends all its time swapping
  - Little or no real work is done
    - All the time is spent in overhead in process switching
    - Could happen if you fork off too many processes uncontrollably
  - Disk light is on all the time
- Solutions
  - Good page replacement algorithms
  - Reduce number of processes running
  - Fit more memory

Bonus

- We do not need all of a process in memory for it to run
- We can swap in pages as required
- So - we can now run processes that are bigger than total memory available!
- Main memory is called real memory
- User/programmer sees much bigger memory - virtual memory

Problems with Virtual Memory

- Each process can see a big virtual memory
  - This means each process has its own Page Table
  - Page table can be big
    - E.g. Power PC has 32 bit virtual addresses, with 4K page size
    - 2^{12} = 4K, leaves 2^{32} / 2^{12} = 2^{20} pages per process
      - Storing 2^{20} entries per process takes a lot of memory
      - Actually only 2^{16}, because we'll use 4 bits for a segment
    - Solutions
      - Store page table itself in virtual memory; page table could be swapped in and out
      - Two-level schemes
      - Inverted schemes

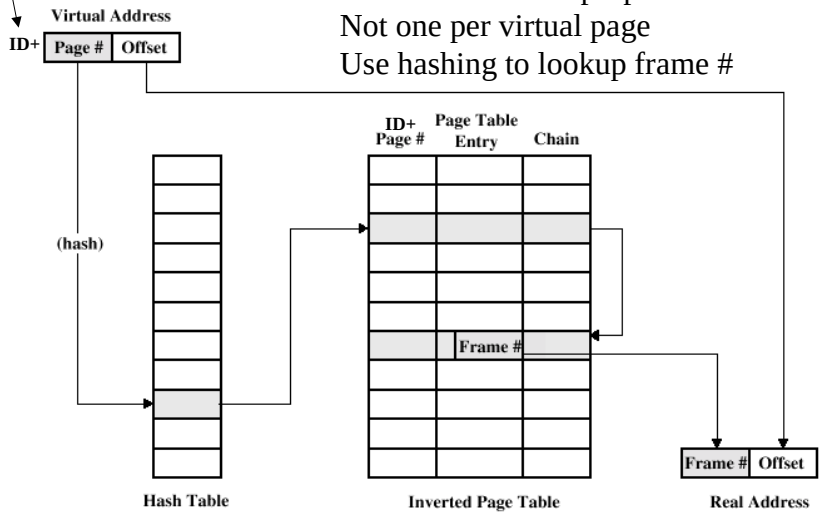
Inverted Page Table Structure

Add ID based on PID

Table for Real Memory Pages
- Stored in fixed proportion of real memory

Not one per virtual page

Use hashing to lookup frame #



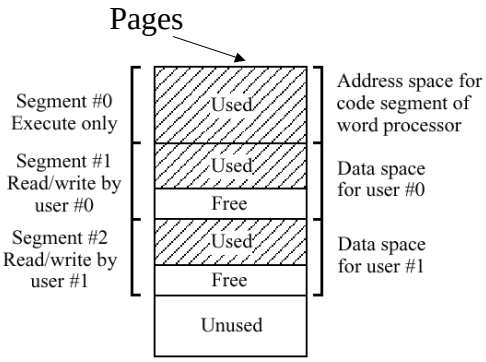
Translation Lookaside Buffer

- Virtual memory reference can cause two physical memory accesses
  - Fetch appropriate page table entry
  - Fetch desired data
  - Could double memory access time
- Solution
  - Create a special cache for page table entries
  - Called the **TLB** (Translation Lookaside Buffer)
  - Functions just like a memory cache
    - Contains page table entries most recently used
    - Complicates things even more for just a single memory reference
      - Look to see if virtual address is in the TLB
        - » If yes, use it. If not, we must fetch it from memory or disk
      - Decode the virtual address via the Page Table to a physical address
      - Send the physical address to the data cache to see if it's in the cache
        - » All our usual stuff with cache hits/misses

Segmentation

- Paging is not (usually) visible to the programmer
- Segmentation is visible to the programmer
- Usually different segments allocated to program and data
  - E.g., a segmented memory might be used to allow two users to share the same word processor code, with different data spaces
- May be a number of program and data segments

Segmentation Example



Means a virtual memory address with segmentation is split up into segment, page, offset

Allows sharing among processes, protection of segments (also where memory faults can be generated), growing of data structures if OS increases the segment size