

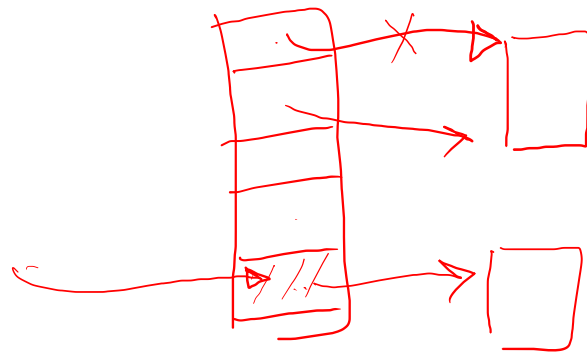
Design and Engineering of Computer Systems

Lecture 13: Demand Paging

Mythili Vutukuru

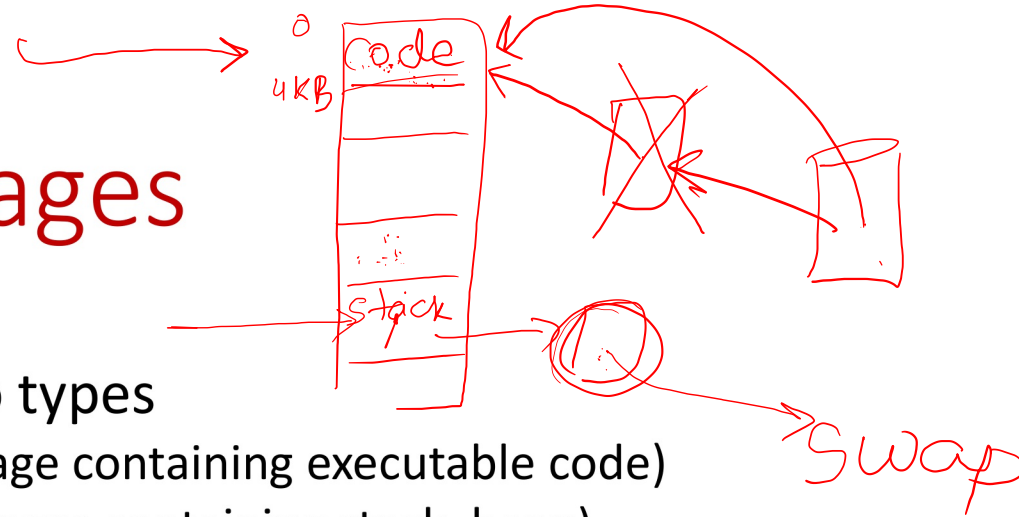
IIT Bombay

Demand Paging



- Story so far: OS allocates physical frames to logical pages of a process
- Modern operating systems provide virtual memory
 - Not all logical pages are assigned physical frames
 - OS allocates physical frames to logical pages only on demand, when process accesses the memory contents of the page
 - OS can reclaim some physical memory of process that is not in active use
 - For some pages in page table, physical frame number is not stored, page table entry is marked as “not present”
- When process accesses a logical page with no physical memory, MMU traps to OS = page fault
 - OS handles page fault, allocates physical frame, restarts process execution
- Virtual memory of processes can be much more than physical memory in the system, OS overcommits memory

File-backed vs. anonymous pages

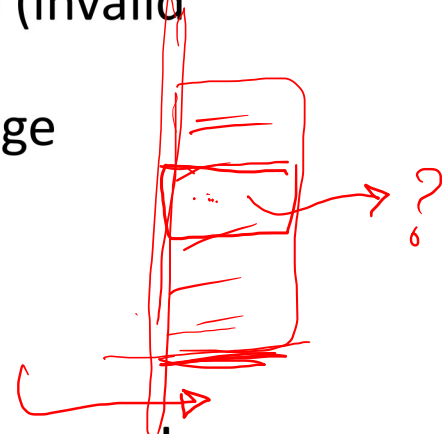


- Pages in the memory image of a process are of two types
 - File-backed pages contain data from files on disk (e.g., page containing executable code)
 - Anonymous pages are not backed by files on disk (e.g., pages containing stack, heap)
- On-demand allocation of file backed pages
 - Can be fetched from disk into empty frame only when accessed for first time
 - Physical frame can be reclaimed when not in use by process (copy exists on disk)
- On-demand allocation of anonymous pages
 - Empty physical frame can be allocated when page is used for first time
 - Once page is modified, cannot simply reclaim physical frame (data can be lost)
- Swap space: space on hard disk used to store copies of modified “dirty” anonymous pages (different from file storage)
 - Dirty anonymous pages are written to swap space when not in use by process, read back from swap into main memory when required


Information in page table entry



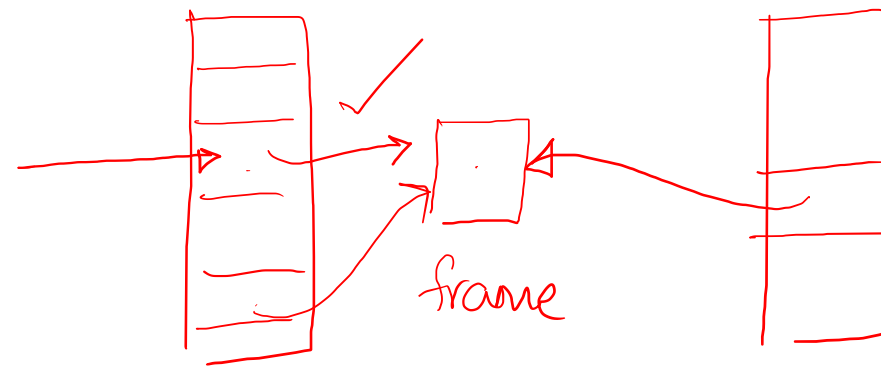
- Page table entry maps page number to physical frame number
- Page table entry also contains several bits to indicate status of page
 - Valid bit indicates if the page is in use by process in its virtual address space (invalid addresses should never be accessed by a process)
 - Present bit indicates if a physical frame number is assigned to the logical page
- Process accesses page with
 - Valid bit not set → illegal memory access (segmentation fault)
 - Valid bit set, present bit not set → OS has not allocated memory yet
- Other bits set by MMU to indicate usage of page to OS (since OS does not know of every memory access)
 - Whenever a process writes to a page, MMU sets dirty bit
 - Whenever a process accesses a page, MMU sets accessed bit



Page fault handling (1)

- When MMU walks page table to translate a virtual address to physical address, the various bits in page table entry are also examined
 - MMU traps to the OS (page fault) in case of any unexpected behavior
 - Illegal access, e.g., process tries to write to a read-only page
 - Invalid access, e.g., process tries to access an entry with valid bit not set
 - Valid bit is set but present bit is not set, e.g., due to on-demand memory allocation not done yet by OS
 - For illegal/invalid accesses, OS may terminate the process when servicing the page fault
 - If the virtual address is valid but not present in physical memory, OS will service page fault by assigning a free frame to the page
- 

Page fault handling (2)



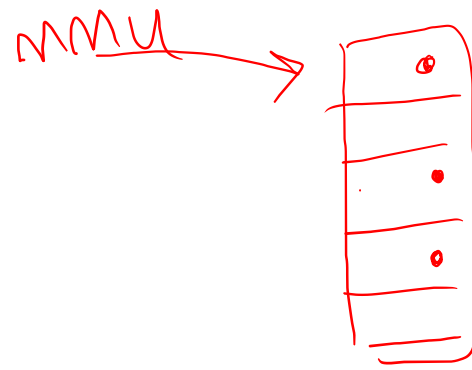
- If page not assigned a frame, OS finds free physical frame to assign to logical page
 - OS maintains list of free physical frames to assign during page faults
 - If there are no free physical frames, OS can evict a victim page (i.e., take away a physical frame assigned to another page) to free up a physical frame
 - Page replacement policy helps OS identify victim pages
 - Reclaiming victim page involves writing victim page contents to swap space (in case of modified anonymous page) and deleting old page table mappings
- Next, OS populates contents of free physical frame with new page content
 - If page is in swap space or file-backed, contents are read from disk into free page
 - Reading the page contents from disk may block the process
- Once physical frame is ready with page content, OS updates page table mapping with new physical frame number, MMU updated, process execution restarted
- Minor page fault: physical frame already in memory (e.g., shared library being used by another process), just need to add page table mapping and restart

major

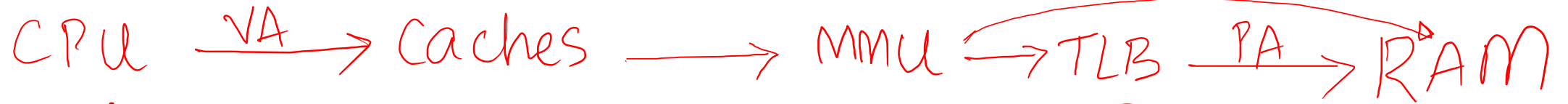
Page replacement policies

- Page replacement policy: which victim page should OS pick in order to free up a physical frame?
- Goal: Minimize page faults, evict pages that we are not likely to need immediately
- Simple policy: First In First Out (FIFO) evicts pages in the order in which they have been assigned frames
 - May be suboptimal, e.g., the first assigned pages may be important pages that are in use very often, leading to another page fault in near future
- Most commonly used policy: evict the Least Recently Used (LRU) page
 - Page has not been used for sometime now, so less likelihood that it will be immediately used in future

LRU implementation



- How does OS know which page is LRU?
 - OS is not involved in every memory access, so doesn't know which pages have been recently used
- Solution: MMU sets the accessed bit for every page table entry it accesses
 - Accessed bit is set implies page has been recently used
- Modern operating systems implement approximate LRU
 - Periodically, look at accessed bit of pages to classify pages into active and inactive pages
 - Pick pages that have been inactive for eviction
 - May also avoid dirty pages for eviction, since it requires extra disk write



What happens on a memory access?

- CPU has requested data (or instruction) at a certain memory address
 - CPU checks caches. If hit in **CPU cache**, data is directly available. Otherwise, CPU has to access memory via MMU
 - MMU checks **TLB**. If TLB hit, the physical address is available, the data is fetched from memory. Otherwise, MMU has to **walk page table**
 - If address is valid and present in page table, permission checks pass, translate address and access memory (MMU adds translation to TLB)
 - For any error (address is valid but not present, or if valid bit is not set, or permissions do not permit access) MMU traps to OS for page fault, OS services page fault
- Overheads: CPU cache misses, TLB misses, page faults, ...

Thrashing

- How much physical memory should OS assign a process?
- Every process has a **working set**: frequently used pages in memory image
 - Working set can change from time to time, based on code being executed
 - Working set is usually smaller than total virtual memory of process
 - If memory assigned to process is less than working set, frequent page faults, frequent interruptions to process execution
- **Thrashing** = system spends too much time servicing page faults and swapping back and forth from disk, and too little time doing useful application work
 - Significant slowdown in application performance will be noticed by users
- Thrashing can also happen for CPU caches and TLB, but the term is most commonly used with memory pages
- Solution: users can reduce working set of processes, OS can terminate some processes or clean up unnecessary memory, ...

Summary

- In this lecture:
 - Virtual memory: on demand memory allocation by OS
 - Page fault handling
 - Page replacement policies
 - End-to-end view of memory access overheads
- Look at any process in your system. How much virtual memory does it use, and how much physical memory is assigned? Output of “ps” will show these in Linux.