Virtual Memory

Lecture 16

Cache

- With fully associative and set associative cache, a *replacement policy* is invoked when it becomes necessary to evict a block from cache.
  - An optimal replacement policy would be able to look into the future to see which blocks won't be needed for the longest period of time.
- *Least recently used* (LRU) algorithm
  - keeps track of the last time that a block was assessed and evicts the block that has been unused for the longest period of time.
- The disadvantage of this approach is its complexity:
  - LRU has to maintain an access history for each block, which ultimately slows down the cache.

Replacement Policies

- *First-in, first-out* (FIFO) is a popular cache replacement policy.
  - The block that has been in the cache the longest, regardless of when it was last used.
- A *random* replacement policy does what its name implies:
  - It picks a block at random and replaces it with a new block.
  - Random replacement can certainly evict a block that will be needed often or needed soon, but it never.

Cache write Policies

- Cache replacement policies must also take into account *dirty blocks*,
  - Those blocks that have been updated while they were in the cache.
    - Dirty blocks must be written back to memory
  - A *write policy* determines how this will be done.
- There are two types of write policies,
  - *write through* and *write back*.
- Write through:
  - updates cache and main memory simultaneously on every write.

Cache write Policies

- Write back (also called *copyback*):
  - Updates memory only when the block is selected for replacement.
- The disadvantage of write through:
  - Memory must be updated with each cache write, which slows down the access time on updates.
    - This slowdown is usually negligible, because the majority of accesses tend to be reads, not writes.
- The advantage of write back:
  - Memory traffic is minimized,
- Its disadvantage:
  - Memory does not always agree with the value in cache, causing problems in systems with many concurrent users.

Virtual Memory

- Cache memory:
  - Enhances performance by providing faster memory access speed.
- Virtual memory:
  - Enhances performance by providing greater memory capacity, without the expense of adding main memory.
- Instead, a portion of a disk drive serves as an extension of main memory.

Virtual Memory

- The amount of real memory in a computer is limited to the amount of RAM installed.
  - Common memory sizes are 256MB, 512MB, and 1GB.
- Since it has a finite amount of RAM:
  - It is possible to run out of memory when too many programs are running at one time
    - if you load the operating system, an e-mail program, a Web browser and word processor into RAM simultaneously, RAM may not be enough to hold it all
- This is where virtual memory comes in:
  - Virtual memory increases the available memory of computer by enlarging the "address space," or places in memory where data can be stored

Virtual Memory Terminologies

- **Paging:**
  - One of the memory-management schemes by which a computer can store and retrieve data from hard disk for use in main memory
- The operating system retrieves data from hard disk in same-size blocks called *pages*
  - The advantage of paging:
    - It allows the physical address space of a process to be noncontiguous
- **Page Fault**
  - when a program tries to access pages that are not currently mapped to physical memory. This is known as a page fault.

Virtual Memory Terminologies

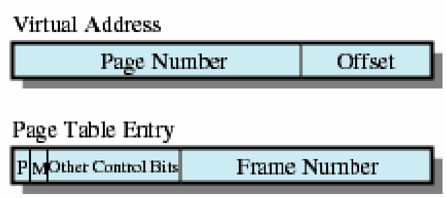
- Page Frame:
  - If a system uses paging, virtual memory partitions main memory into individually managed *page frames:*
    - That are written *(or paged)* to disk when they are not immediately needed.
- A *physical address:*
  - The actual memory address of physical memory.

Virtual Memory- Terminologies

- Programs create *virtual addresses:*
  - That are *mapped* to physical addresses by the memory manager.
- Page Table
  - Information concerning the location of each page, whether on disk or in memory, is maintained in a data structure called a page table.
    - There is one page table for each active process.

Page Table

- Each process has its own page table
  - Each page table entry contains the frame number of the corresponding page in main memory
  - A bit is needed to indicate whether the page is in main memory or not



(a) Paging only

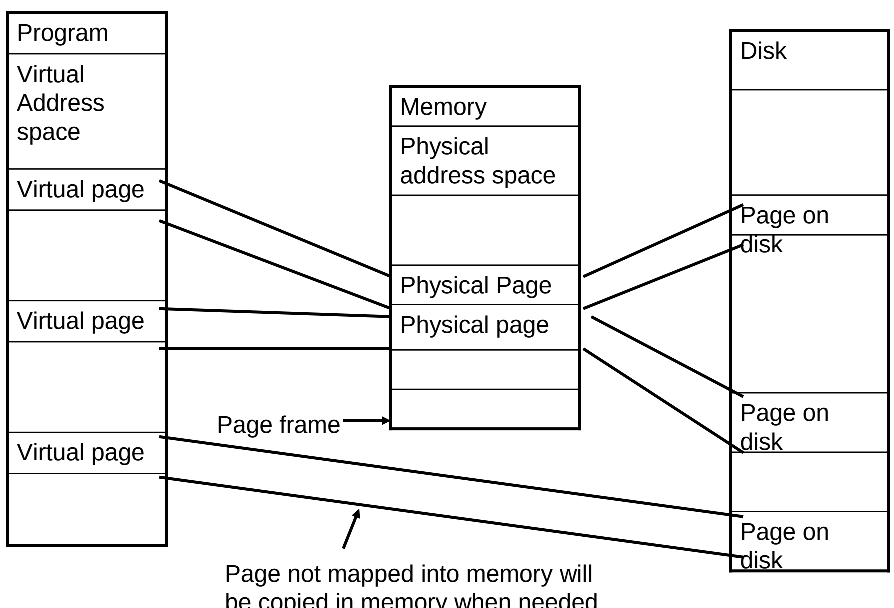
Virtual Memory

- A *physical address* is the actual memory address of physical memory.
  - Each program has its own virtual space, which is the set of addresses that a program uses for load and store operation
- The physical address space is the set of addresses used to reference location in the main memory
  - Programs create *virtual addresses* that are *mapped* to physical addresses by the memory manager.

Virtual Memory

- Virtual and physical address are used to describe address in virtual and physical address space
  - Virtual address space is divided in pages – hard disk
    - Virtual page
  - Some of the pages are copied into memory – **page frame**
    - Physical page

Virtual memory



Virtual Memory Organization

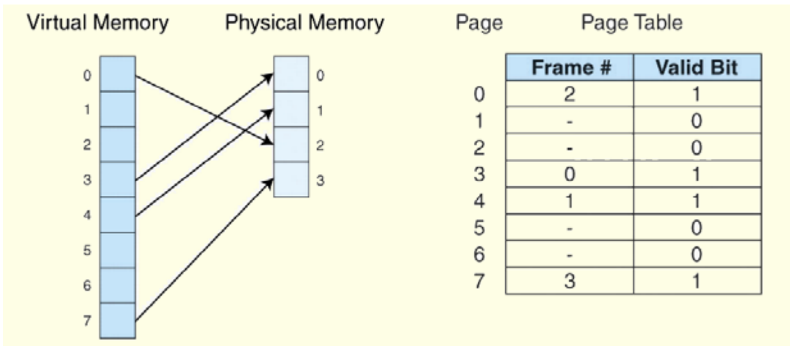
- Main memory and virtual memory are divided into equal sized pages.
- The entire address space required by a process need not be in memory at once.
  - Some parts can be on disk, while others are in main memory.
  - The pages allocated to a process do not need to be stored contiguously-- either on disk or in memory.
- Only the needed pages are in memory at any time,
  - the unnecessary pages are in slower disk storage.

Address Translation

- When a process generates a virtual address, the operating system translates it into a physical memory address.
- To accomplish this,
  - The virtual address is divided into two fields:
    - A *page* field, and an *offset* field.
  - The page field determines the page location of the address, and
  - The offset indicates the location of the address within the page.
- The logical page number is translated into a physical page frame through a lookup in the page table.

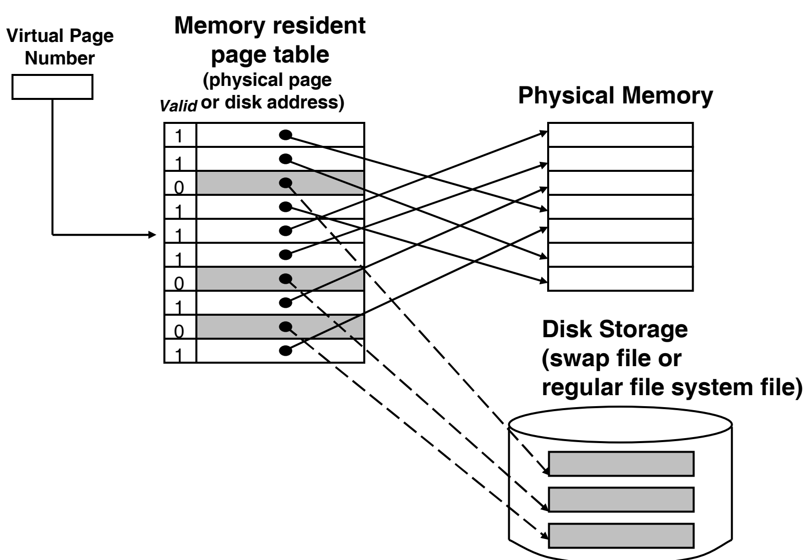
Page Table

- The entire page table may take up too much main memory
  - Page tables are also stored in virtual memory
- When a process is running, part of its page table is in main memory



Page	Frame #	Valid Bit
0	2	1
1	-	0
2	-	0
3	0	1
4	1	1
5	-	0
6	-	0
7	3	1

Page Tables

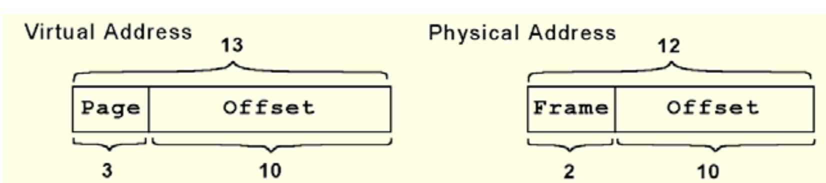


Page Fault

- If the valid bit is zero in the page table entry for the logical address, this means that the page is not in memory and must be fetched from disk.
  - This is a page fault.
  - If necessary, a page is evicted from memory and is replaced by the page retrieved from disk, and the valid bit is set to 1.
- If the valid bit is 1, the virtual page number is replaced by the physical frame number.
- The data is then accessed by adding the offset to the physical frame number.

Virtual Memory-Example

- suppose a system has a virtual address space of 8K and a physical address space of 4K, and the system uses byte addressing and the page size is 1024.
  - So $2^{13}/2^{10} = 2^{3}$ virtual pages.
- A virtual address has 13 bits ($8K = 2^{13}$) with 3 bits for the page field and 10 for the offset,
- A physical memory address requires 12 bits, the first two bits for the page frame and the trailing 10 bits the offset.



Virtual Memory-Example

- If we have the page table shown below.
  - What happens when CPU generates address 5459_{10} = 1010101010011_2?

	Frame	Valid Bit
Page 0	–	0
1	3	1
2	0	1
3	–	0
4	–	0
5	1	1
6	2	1
7	–	0

Page Table

Addresses	
Page 0 :	0 - 1023
1 :	1024 - 2047
2 :	2048 - 3071
3 :	3072 - 4095
4 :	4096 - 5119
5 :	5120 - 6143
6 :	6144 - 7167
7 :	7168 - 8191

Virtual Memory

- The address 1010101010011_2 is converted to physical address 010101010011 because the page field 101 is replaced by frame number 01 through a lookup in the page table.

	Frame	Valid Bit
Page 0	–	0
1	3	1
2	0	1
3	–	0
4	–	0
5	1	1
6	2	1
7	–	0

Page Table

Addresses	
Page 0 :	0 - 1023
1 :	1024 - 2047
2 :	2048 - 3071
3 :	3072 - 4095
4 :	4096 - 5119
5 :	5120 - 6143
6 :	6144 - 7167
7 :	7168 - 8191

Virtual Memory- Example

- What happens when the CPU generates address 100000000100_2?

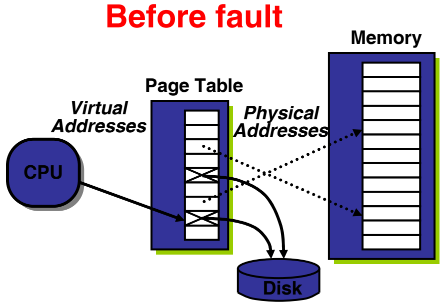
Page Table

Page	Frame	Valid Bit	Addresses
0	–	0	Page 0 : 0 – 1023
1	3	1	1 : 1024 – 2047
2	0	1	2 : 2048 – 3071
3	–	0	3 : 3072 – 4095
4	–	0	4 : 4096 – 5119
5	1	1	5 : 5120 – 6143
6	2	1	6 : 6144 – 7167
7	–	0	7 : 7168 – 8191

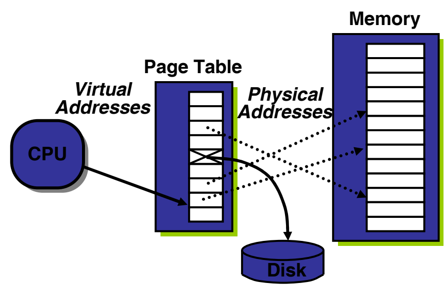
Page Faults (like "Cache Misses")

- What if an object is on disk rather than in memory?
  - Page table entry indicates virtual address not in memory
  - OS exception handler invoked to move data from disk into memory
    - current process suspends, others can resume
    - OS has full control over placement, etc.

Before fault



After fault



Virtual Memory

- virtual memory is also a factor in the calculation, to consider page table access time. Example
- Suppose a main memory access takes 200ns, the page fault rate is 1%, and it takes 10ms to load a page from disk. We have:

  EAT = 0.99(200ns + 200ns) 0.01(10ms) = 100, 396ns.

- Even if there had no page faults, the EAT would be 400ns because memory is always read twice:
  - First to access the page table, and second to load the page from memory.