Virtual memory & MMU I

CompSys304 – Computer Architecture

A/Prof Oliver Sinnen

Department of Electrical, Computer, and Software Engineering

Content

Virtual memory

Motivation or problems solved

1. Safe and efficient sharing of memory among multiple programs
   - Secure sharing
2. Burden of programming for small, limited memories
   - Programs can exceed size of physical memory

Virtual memory

Motivation or problems solved

1. Safe and efficient sharing of memory among multiple programs
   - Secure sharing
2. Burden of programming for small, limited memories
   - Programs can exceed size of physical memory

Key elements

- Virtual address – address used by CPU
- Physical address – address in main memory
- Address translation – *mapping* virtual to physical address

Virtual memory

Motivation or problems solved

1. Safe and efficient sharing of memory among multiple programs
   - Secure sharing
2. Burden of programming for small, limited memories
   - Programs can exceed size of physical memory

Key elements

- Virtual address – address used by CPU
- Physical address – address in main memory
- Address translation – *mapping* virtual to physical address

OK, but why here, why not in OS course?

- Address translation done in CPU hardware (Memory Management Unit – MMU) and software
- Impact on CPU performance

Concepts

- Each program gets own address space (i.e. area)
  - Virtual memory exists in abundance
  - No other program is allowed to access this address space

  $\Rightarrow$ Protection of program's code and data from other programs

Concepts

- Each program gets own address space (i.e. area)
  - Virtual memory exists in abundance
  - No other program is allowed to access this address space

  $\Rightarrow$ Protection of program's code and data from other programs

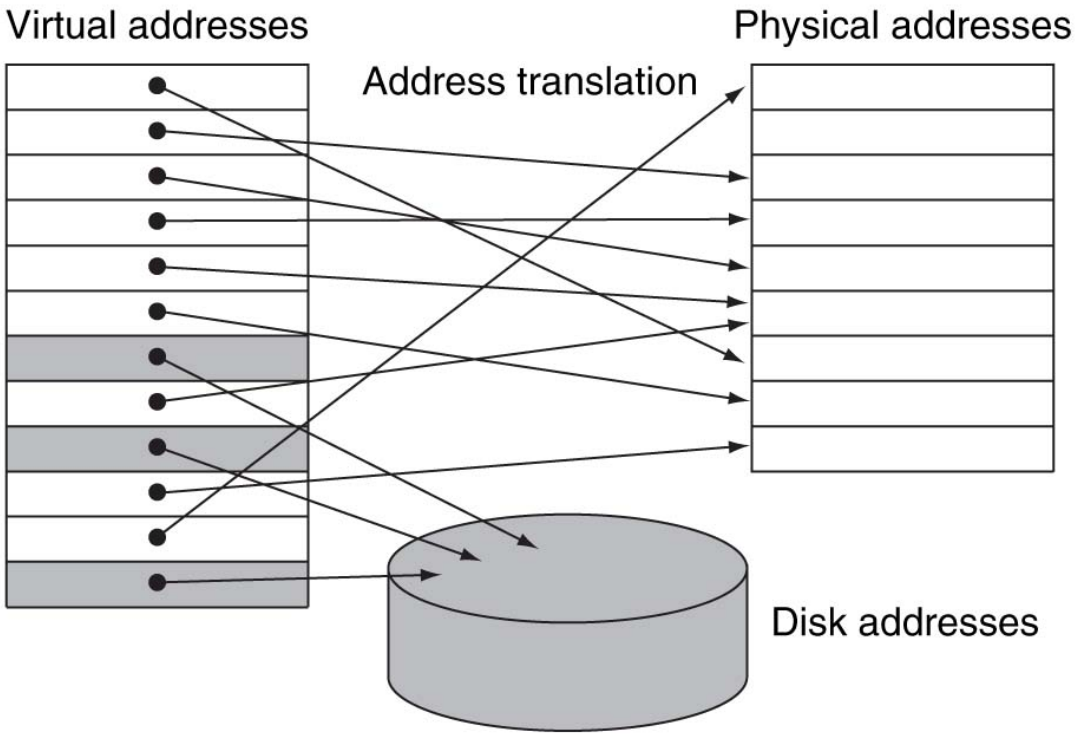
- Main memory becomes cache of secondary memory (disks)
  - Indeed very similar to cache concept

Concepts

- Each program gets own address space (i.e. area)
  - Virtual memory exists in abundance
  - No other program is allowed to access this address space
  - $\Rightarrow$ Protection of program's code and data from other programs
- Main memory becomes cache of secondary memory (disks)
  - Indeed very similar to cache concept
- Virtual address has usually more bits than physical address
  - Idea of concept!
  - But less is also possible (32-bit CPU with more than 4GB RAM)

Principle of address translation

- Virtual address can be mapped to any physical address or disk location
- Address translation is process of doing this



Features

- Enables relocation
  - Load program anywhere in main memory

Features

- Enables relocation
  - Load program anywhere in main memory
- Allows physical addresses not to be continuous
  - Only virtual addresses need to be

Features

- Enables relocation
  - Load program anywhere in main memory
- Allows physical addresses not to be continuous
  - Only virtual addresses need to be

Possible:

- 2 different virtual addresses can map to same physical address
  - To share code between programs

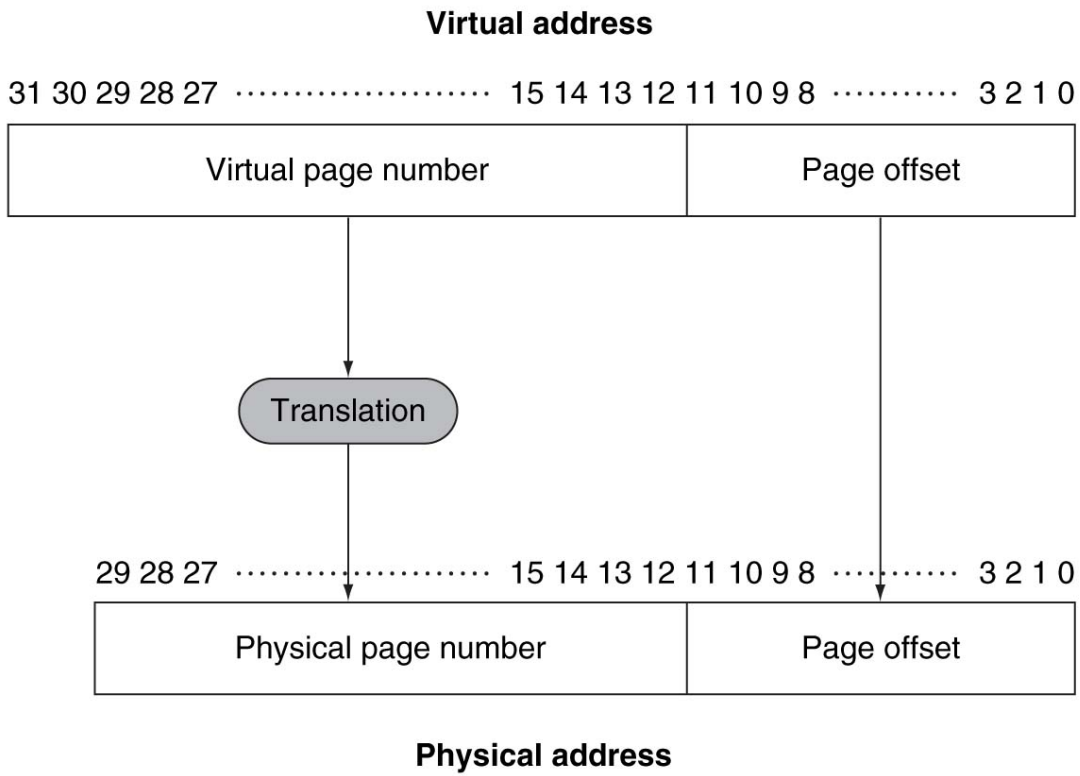
Content

Address translation notation

- Page – smallest block handled by virtual memory
  - Corresponds to cache line
- Page offset f – 2^f is size of page (in bytes)
  - On x86 CPUs (all?) $f = 12 \rightarrow$ page size 4KByte
  - Support for 2/4MB pages on x86
- Page number – number referencing page, index of page
- Page hit – page is in main memory
  - Translation handled by MMU
- Page fault – page is not in main memory
  - Virtual memory miss
  - Corresponds to cache miss
- Page fault handler – function to handle a page miss
  - SW: part of OS kernel

Address translation

- Virtual address 32 bits, physical address 30 bits, page offset 12 bits



Page fault

- Page fault is very expensive
  - Load 4KB from disk
  - Latency of random disk access very high in comparison to RAM
    - much better these days with SSDs

Page fault

- Page fault is very expensive
  - Load 4KB from disk
  - Latency of random disk access very high in comparison to RAM
    - much better these days with SSDs
- High latency

  $\Rightarrow$ Pages should not be small

Page fault

- Page fault is very expensive
  - Load 4KB from disk
  - Latency of random disk access very high in comparison to RAM
    - much better these days with SSDs
- High latency

  $\Rightarrow$ Pages should not be small
- Write back (copy back)
  - Write through takes long
  - Writing single word/cache line to disk is inefficient (latency/bandwidth is large)
  - Dirty bit – set when written to any word of page

Page fault

- Page fault is very expensive
  - Load 4KB from disk
  - Latency of random disk access very high in comparison to RAM
    - much better these days with SSDs
- High latency

  $\Rightarrow$ Pages should not be small
- Write back (copy back)
  - Write through takes long
  - Writing single word/cache line to disk is inefficient (latency/bandwidth is large)
  - Dirty bit – set when written to any word of page
- Page faults can be handled by software
  - Disk access slow anyway, hardware not necessary
  - Advantage, clever placement algorithms can reduce page fault rate

Placement and retrieval

- Reduce page fault rate
  - $\Rightarrow$ Fully associative placement of pages in memory
    - Virtual page can go into any physical page
- But, locating a page must be fast
- Use indexed table

Placement and retrieval

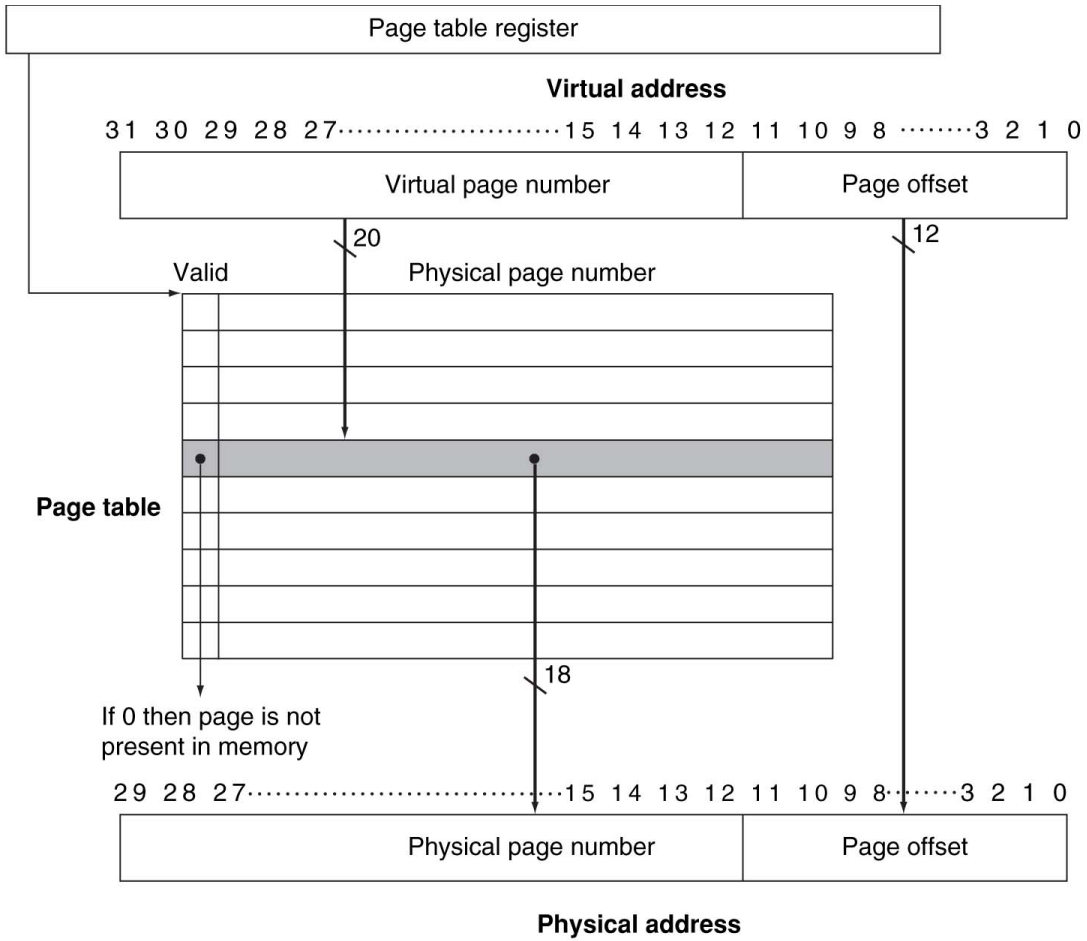
- Reduce page fault rate
  - $\Rightarrow$ Fully associative placement of pages in memory
    - Virtual page can go into any physical page
- But, locating a page must be fast
- Use indexed table

Page table

- In main memory
- Every program (process) has own page table
- Virtual-physical pairs
  - also entries for not in memory (valid bit is `false`)
- Indexed with page number of virtual address
  - No searching necessary
- Page table register
  - MMU register: points to start of page table

Page table

- Table indexed by page number
  - $2^{20} = 1$M entries, 4 bytes each $\Rightarrow$ 4MB



Memory requirement of page tables

- On 32-bit system, page table would require 4 MB
  - Remember, one table per program (process)
  - Would consume too much memory
  - Even worse on systems with larger virtual address

Memory requirement of page tables

- On 32-bit system, page table would require 4 MB
  - Remember, one table per program (process)
  - Would consume too much memory
  - Even worse on systems with larger virtual address
- Solutions
  - E.g. Multi-level page table
  - *or* Only keep used part of table
    - Only possible to allocate consecutive parts of virtual memory
  - *or* ...

Memory requirement of page tables

- On 32-bit system, page table would require 4 MB
  - Remember, one table per program (process)
  - Would consume too much memory
  - Even worse on systems with larger virtual address
- Solutions
  - E.g. Multi-level page table
  - *or* Only keep used part of table
    - Only possible to allocate consecutive parts of virtual memory
  - *or* ...
- Not discussed here

$\Rightarrow$ studied in Operating System design

Replacement policy

- When physical memory exhausted, evict which page?

Replacement policy

- When physical memory exhausted, evict which page?
- LRU – Least Recently Used
  - Again, as with caches

Replacement policy

- When physical memory exhausted, evict which page?
- LRU – Least Recently Used
  - Again, as with caches
- Table contains many pages
  - Exact LRU would be too costly

Replacement policy

- When physical memory exhausted, evict which page?
- LRU – Least Recently Used
  - Again, as with caches
- Table contains many pages
  - Exact LRU would be too costly

Approximation

- Use reference bit (ref bit)
  - Set when page is accessed
  - Periodically reset
  - Software takes snapshots before reset
    - Can build set of recently used pages
  - Evict page to swap space whose ref bit has not been set for some time