Sharing Main Memory: Segmentation and Paging (cont)

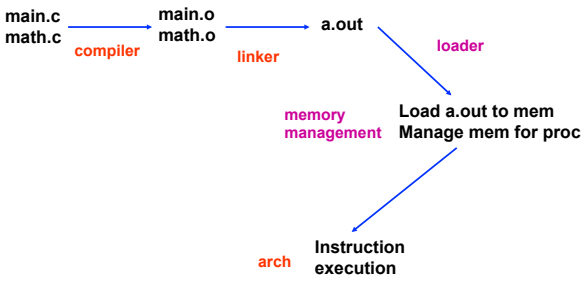
ECE595
Oct 9

(Y. Charlie Hu)
Jay Meng

Review: sharing main memory

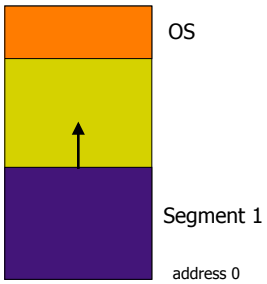
- Linker generated a.out, assuming address starting from 0
- OS needs to load it into physical memory
  - Code
  - Data
  - Stack
  - Heap

[lec13] The big picture



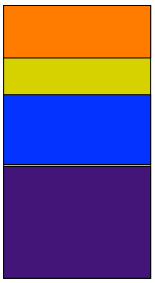
Review: sharing main memory

- Simple uniprogramming



Review: sharing main memory

- Simple multiprogramming



4 drawbacks?

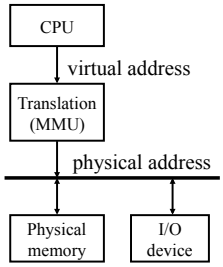
Review: sharing main memory

- Simple multiprogramming – 4 drawbacks
  1. No protection
  2. Low utilization -- Cannot relocate dynamically
     - Cannot do anything about holes
  3. No sharing -- Single segment per process
  4. Entire address space needs to fit in mem
     - Need to swap whole, very expensive!

Fix drawback #1 & #2: Dynamic memory relocation

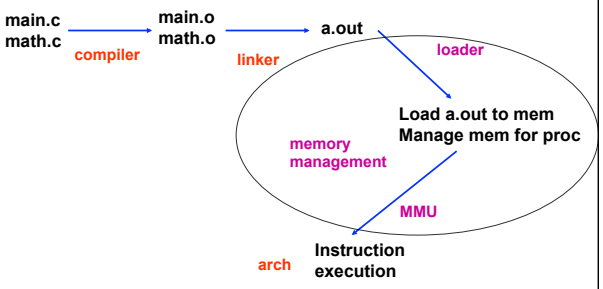
- Instead of changing the address of a program when it's loaded, change the address dynamically *during every reference*
  - Under dynamic relocation, each program-generated address (called a *logical address* or *virtual address*) is translated in hardware to a *physical* or *real address*

Dynamic memory relocation

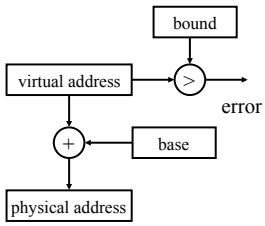


- Actual translation is in hardware (MMU)
  - why?
- Controlled in software
- CPU view
  - what program sees, virtual addresses
- Memory view
  - physical memory

[lec13] The big picture



Fix drawback # 1 & #2 – base and bound

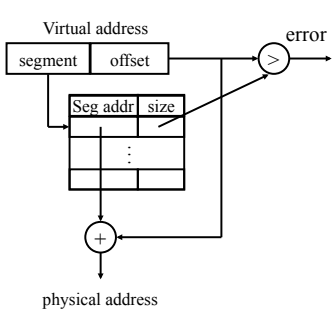


Indirection!

Review: sharing main memory

- Simple multiprogramming – 4 drawbacks
  - No protection
  - Low utilization -- Cannot relocate dynamically
    - *Can relocate now; but is frequent relocation desirable?*
  - No sharing -- Single segment per process
  - Entire address space needs to fit in mem
    - Need to swap whole, very expensive!

Fix drawback # 3 – allow multiple segments



- A table of (seg, size)
- Protection: each entry has
  - (nil, read, write, exec)
- On a context switch: save/restore the table or a pointer to the table to/from PCB

[lec3] Process Control Block (Process Table)

- Process management info
  - State (ready, running, blocked)
  - PC & Registers, parents, etc
  - CPU scheduling info (priorities, etc.)
- Memory management info
  - Segment table, page table, stats, etc
- I/O and file management
  - Communication ports, directories, file descriptors, etc.

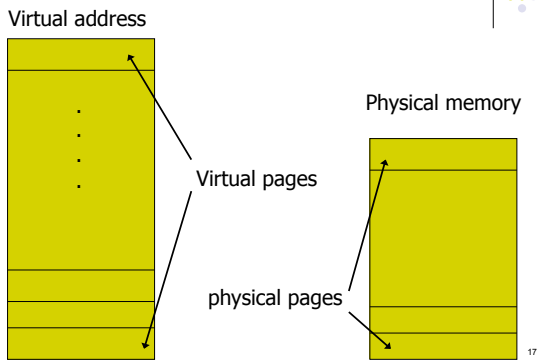
Pros/cons of segmentation

- Pros:
  - Process can be split among several segments
    - Allows sharing (how?)
  - Segments can be allocated/swapped independently
    - Still allocate/swap each segment as a whole
- Cons:
  - External fragmentation: many holes in physical memory
    - Also happens in base and bound schemes
    - Can relocate, but is it desirable / easy?

What fundamentally causes external fragmentation?

- Segments of many different sizes
- Each has to be allocated contiguously
- "Million-dollar" question:
  Physical memory is precious.
  Can we limit the waste to a single hole of X bytes?

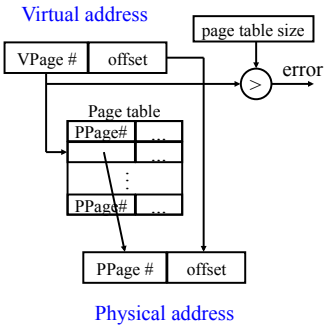
Virtual pages / physical pages



Paging

- Goal:
  - to make allocation and swapping easier (time)
  - to reduce memory fragmentation (space)
- Key idea:
  - Make all chunks of memory the same size, called *pages*
- Implementation:
  - For each process, a *page table* defines the base address of each of that process' pages along with existence and read/ write bits
  - Translation?

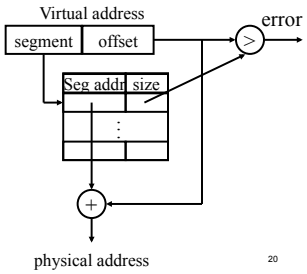
Paging



- Context switch
  - similar to the segmentation scheme
- Pros:
  - easy allocation, keep a free list
  - easy to swap
  - easy to share

Deek thinking: Paging implementation

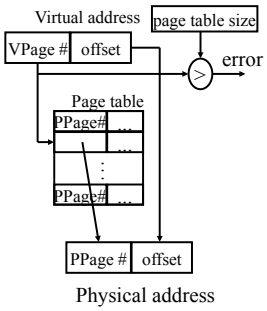
- Translation: table lookup and bit substitution
- Why is this possible?
- Why cannot we do the same in segmentation?



How many PTEs do we need?
(assume page size is 4096 bytes)

- Worst case for 32-bit address machine?
- What about 64-bit address machine?

Paging implementation – how does it really work?



- Where to store page table?
- How to use MMU?
  - Even small page tables too large to load into MMU
  - Page tables kept in mem and MMU only has their base addresses
    - What does MMU have to do?
- Page size?
  - Small page -> big table
    - 32-bit with 4k pages
  - Large page ->small table but large internal fragmentation

Paging vs. segmentation

- Segmentation:
  - External fragmentation
  - Complicated allocation, swapping
  - \+ Small segmentation table
- Paging
  - Internal fragmentation
  - \+ Easy allocation, swapping
  - Large page table

Deep thinking

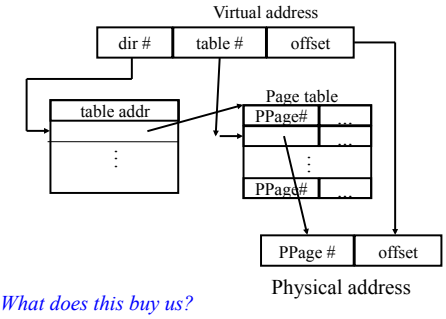
- Why does the page table have to be contiguous in the physical memory?
  - Why did a segment have to be contiguous in memory?
- For a 4GB virtual address space, we just need 1M PTE (~4MB), what is the big deal?
- My PC has 1 GB, why do we need PTEs for the entire 4 GB address space?

Page table

- The page table has to be consecutive in mem
  - Potentially large
  - Consecutive pages in mem hard to find
- How can we be flexible?

"All computer science problems can be solved with an extra level of indirection."

Two-level page tables



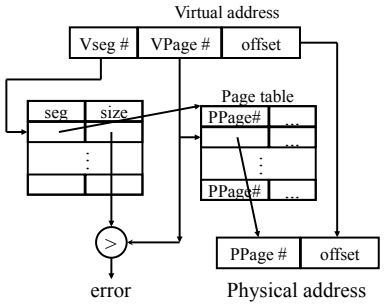
What does this buy us?

Multi-level page tables

- 3 Advantages?

The power of an extra level of indirection!

Segmentation with paging



Ex: IBM System 370 (24-bit, 4-bit segment #, 8-bit page #)

Segmentation + paging

- Use two levels of mapping to make tables manageable:
  - Each segment contains one or more pages
  - Segments correspond to logical units: code, data, stack
  - Segments vary in size and are often large
  - Pages are for easy of management by OS: fixed size -> easy to allocate/free

Reading assignment

- Chapter 8