Memory Management II

Tom Kelliher, CS 318

Apr. 13, 2012

1 Administrivia

Announcements

Assignment

See reading assignment on class web site.

From Last Time

Memory Management I.

Outline

1. Non-contiguous allocation.
2. Paging.
3. Segmentation.

Coming Up

Linux kernel modules.

2 Non-Contiguous Allocation

Problems with contiguous allocation:

1. Problems with fixed partitions:
   (a) Limited degree of multiprogramming.
   (b) Internal fragmentation.
   (c) Placement policies.
2. Problems with variable partitions:
   (a) External fragmentation, compaction.
   (b) Swapping is difficult, if not impossible (address binding).

Alternative: *non-contiguous* allocation.

3 Paging

The idea:

1. Entire process in memory.
2. Partition memory into frames of size 2^m.
   (a) Typical frame sizes: 512 to 8K bytes.
   (b) Frame size constrained by MMU design.
3. Logical address space is broken into pages.
   (a) Page size = frame size.

   (b) Pages can be *arbitrarily* mapped onto frames.

   (c) **However**, process “sees” a contiguous, flat address space.

4. MMU splits logical address:

   (a) Assume logical address is n bits.

   (b) Page number field is $n - m$ *most significant* bits.

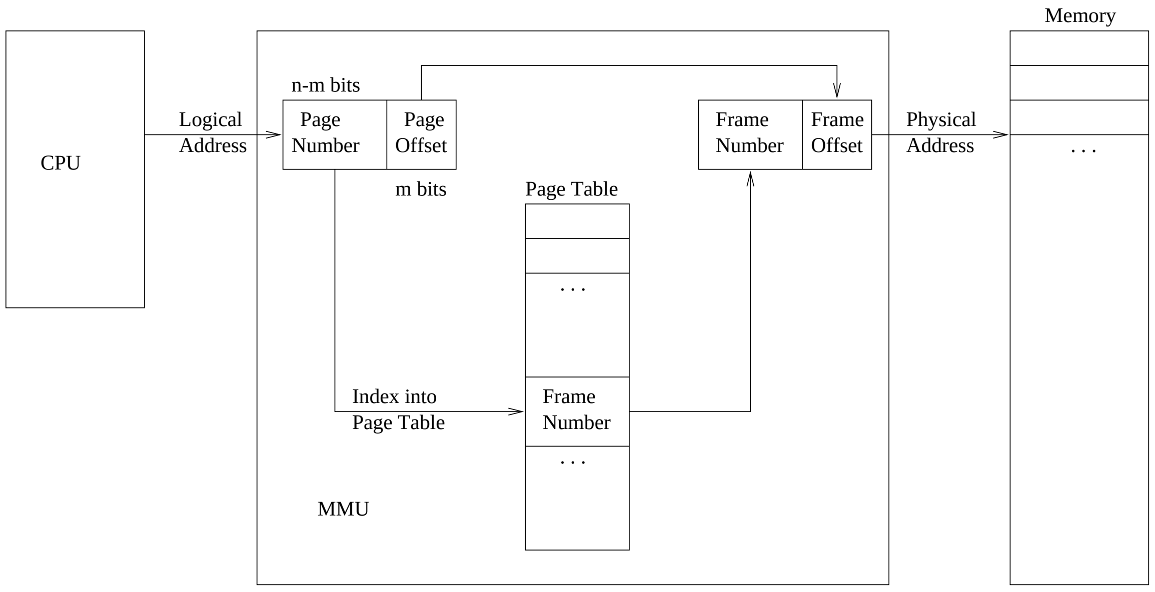
   (c) Page offset field is m *least significant* bits.

   (d) Using table look-up, convert logical page number to physical frame number.

   (e) Frame offset = page offset.

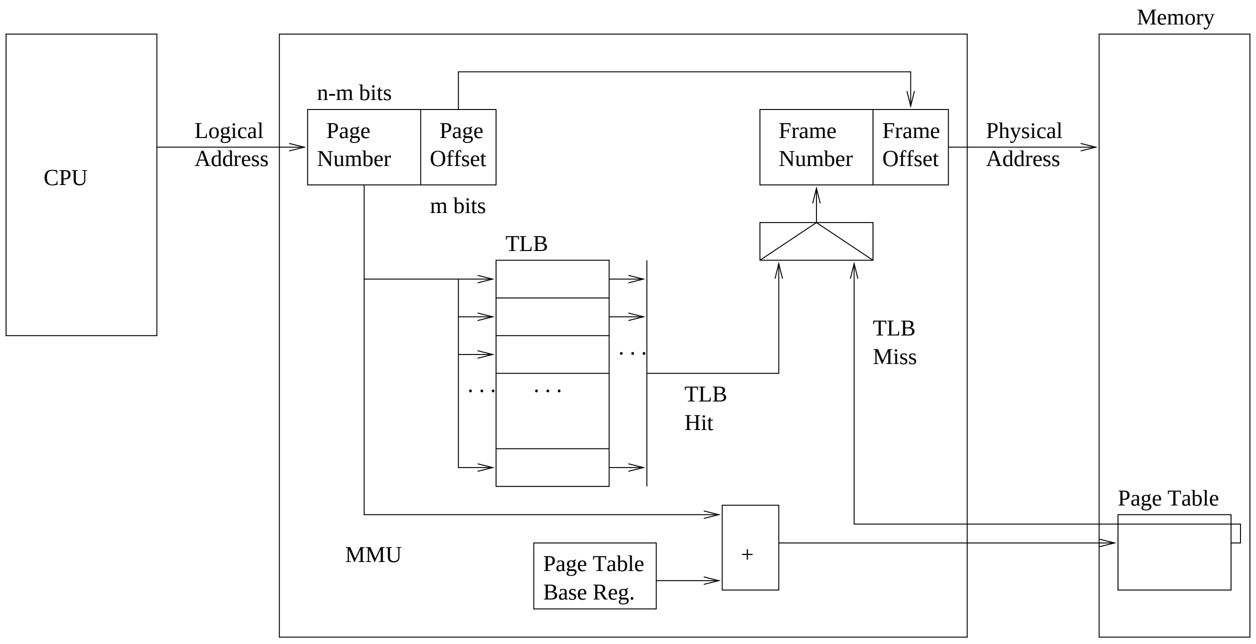
Why is frame, page size a power of two?

Paging hardware:



Design issues:

1. Page size:
    (a) Internal fragmentation.
    (b) Maximizing I/O transfer rate.
2. I/O — process passes *logical* address to kernel.
3. Implementation of the page table.
    (a) Small register file.
    (b) Array in memory.
4. Issues for memory implementation:
    (a) Page table must be in contiguous memory.
    (b) Page table base register.
    (c) "Logical memory access" requires *two* physical accesses.
        i. Translation look-aside buffer.
        ii. TLB entries contain: Page number, frame number pairs.
        iii. Issue: Context switches.
        iv. Only a few entries needed.



3.1 Size, Structure of Page Table

1. Problem: huge page tables. How did this happen?
2. Solutions:
   (a) Valid/invalid bit.

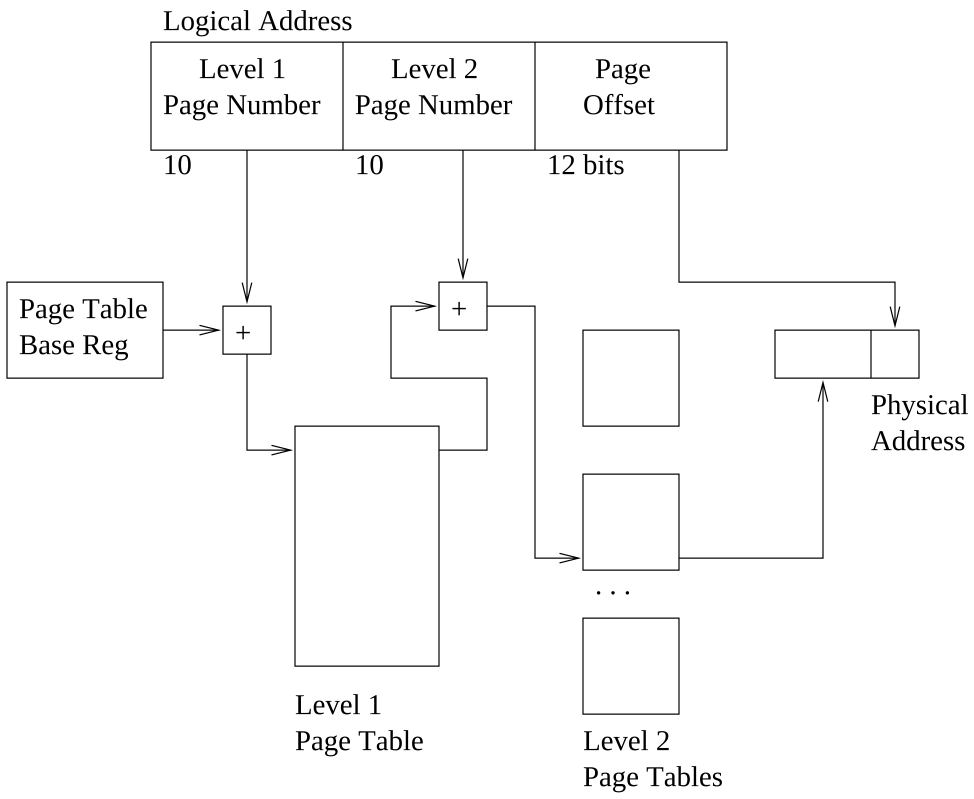
   (b) Page table limit register.

   (c) Multi-level paging.

3.1.1 Multi-Level Paging

Why not just use the limit register?

A two-level paging scheme for a 32-bit logical address:



Is the page table *really* any smaller?

What does this solve?

1. Non-contiguous (paged) page table.
2. "Holes" in page table (valid bit) shrink it.

Sun SPARC: 3 level paging.

64-, 128-bit logical addresses?

3.2 Protection

1. Is it possible for a process to access an arbitrary memory location?
2. Using valid bit to introduce "holes" into logical address space.

3.3 Page Sharing

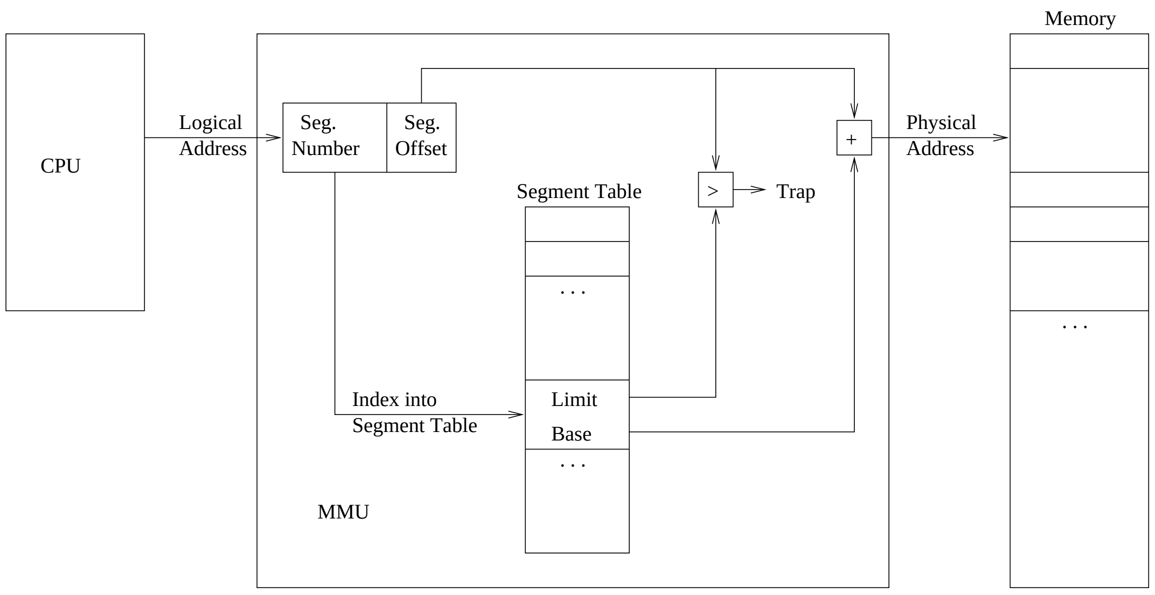
1. What can be shared?
2. Read-only pages.
3. Page alignment — segment the logical address space.
4. Page de-allocation.

4 Segmentation

1. "Object oriented" approach:
   (a) User views program as a set of objects:
      i. Stack.
      ii. Routines.
      iii. Arrays.
      iv. ...

   (b) Each object stored in a segment.
2. Generalization of paging.
3. Logical address is a segment "name" and an offset.
4. Variable-sized "page" having a base and length.
5. Frames start on 2^m memory boundaries. Segments start anywhere. Therefore, must *add* a segment base and offset.
6. Straight paging: external fragmentation.
7. Solution: Paged segmentation (x86 architecture).

Segmentation hardware:


Memory
CPU
Logical Address
Seg. Number
Seg. Offset
Segment Table
>
Trap
+
Physical Address
. . .
Index into Segment Table
Limit
Base
. . .
MMU
. . .