UNIVERSITY of WISCONSIN-MADISON
Computer Sciences Department

CS 537
Intro to Operating Systems

A. Arpaci-Dusseau
Spring 2000

Segmentation and Paging

Questions answered in these notes

- What are the advantages and disadvantages of segmentation?
- What are the advantages and disadvantages of paging?
- How can the two approaches be combined?
- How to make page translation faster?

Segmentation

- **Divide address space into logical segments**
  - Each logical segment can be in separate part of physical memory
  - Separate base and bound for each segment (+ protection bits)
    - Read and write bits for each segment
- **How to designate segment?**
  - Use part of logical address
    - Top bits of logical address select segment
    - Low bits of logical address select offset within segment
  - Implicitly by type of memory reference
    - Code vs. data segment
  - Special registers

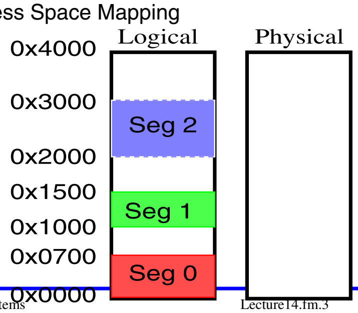
Segment Table

- **Segment Table: Base and bounds for every segment in process**

Translation: Indirection --> Table lookup before Add and Compare

Segment	Base	Bounds	RW
0	0x4000	0x06FF	1 0
1	0x0000	0x04FF	1 1
2	0x3000	0x0FFF	1 1
3			0 0

Address Space Mapping



Where is:

0x0240
0x1108
0x265c
0x3002

Managing Processes with Segments

Process Creation

- Find contiguous space for each segment
- Fill in each base and bound value in segment table

Additional memory allocation when no contiguous space

- Compact memory (move all segments, update bases)
- Swap one or more segments to disk

Context-Switch

- Add segment table to PCB

Process Exit

- Return segments to free list

Pros and Cons of Segmentation

Advantages

- Different protection for different segments
    - read-only status for code
- Enables sharing of selected segments
- Easier to relocate segments than entire address space
- Enables sparse allocation of address space

Disadvantages

- Still expensive/difficult to allocate contiguous memory to segments
- External fragmentation: Wasted memory

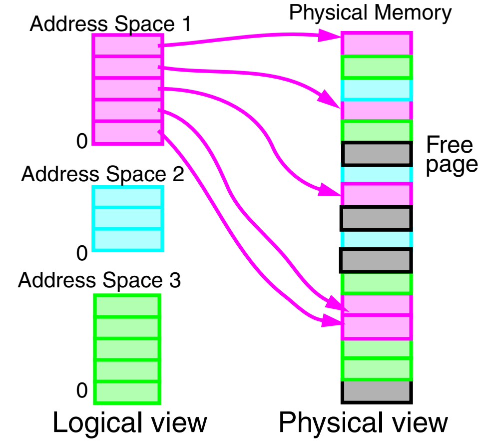
Next approach: Paging

- Allocation easier
- Reduces fragmentation

Paging

Memory divided into fixed-sized pages

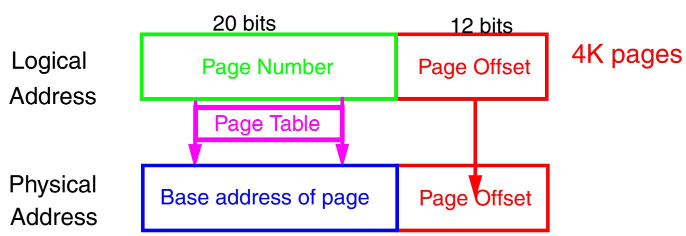
- Typical page size: 512-16K bytes



Translation of Pages

How do translate logical address to physical address?

- Upper bits of address designate page number



No comparison or addition: Just table lookup and bit substitution

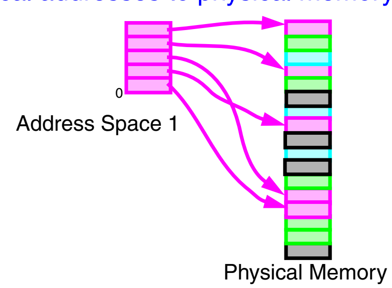
Page table per process: One entry per page in address space

- Base address of each page in physical memory
- Read/Write protection bits

How many entries in page table?

Page Table Example

Mapping of logical addresses to physical memory



- Page Table for process 1

Base Address	Protection
3	1 0
9	1 0
4	1 1
13	1 1
16	1 1

Advantages of Paging

Fast to allocate and free

- **Alloc:** Keep free list of free pages and grab first page in list
  - No searching by first-fit, best-fit
- **Free:** Add page to free list
  - No inserting by address or size

Easy to swap-out memory to disk

- Page size matches disk block size
- Can swap-out only necessary pages
- Easy to swap-in pages back from disk

Disadvantages of Paging

Additional memory reference --> Inefficient

- Page table too large to store as registers in MMU
  - Page tables kept in main memory
  - MMU stores only base address of page table

Storage for page tables may be substantial

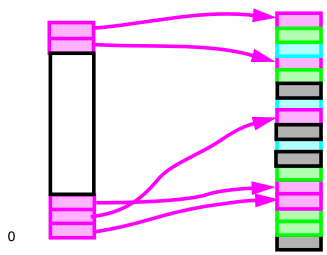
- Simple page table --> Require entry for all pages in address space
  - Even if actual pages are not allocated
- Partial solution: Base and bounds for page table
  - --> Only processes with large address spaces need large page tables
  - Does not help processes with stack at top and heap at bottom

Internal fragmentation: Page size does not match allocation size

- How much memory is wasted (on average) per process?
- Wasted memory grows with larger pages

Paging with Large Address Spaces

Mapping of logical addresses to physical memory



- Page Table for process 1

Base Address	Protection
3	1 0
9	1 0
4	1 1
... skipping many entries ...	**0 0**
13	1 1
16	1 1

Combine Paging and Segmentation

Structure

- Segments correspond to logical units: code, data, stack
  - Segments vary in size and are often large
- Each segment contains one or more (fixed-size) pages

Two levels of mapping to make tables manageable (2 look-ups!)

- Page table for each segment
- Base (real address) and bound (size) for each page table

Logical address divided into three portions: System 370

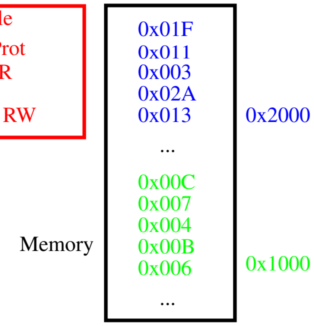
Seg # (4bits)	Page # (8 bits)	Page offset (12 bits)

12 bit physical address base in page table entries -->

Example: Segments + Pages

Translate logical to physical addresses

Segment table		
Base	Bound	Prot
0x2000	0x14	R
0x0000	0x00	
0x1000	0x0D	RW



0x002070 read:

0x202016 read:

0x104C84 read:

0x010424 read:

0x210014 write:

Segments + Pages Advantages

Advantages of Segments

- Supports sparse address spaces
  - If segment is not used, no need for page table
  - Decreases memory required for page tables

Advantages of Paging

- Eliminate external fragmentation
- Segments to grow without any reshuffling

Advantages of Both

- Increases flexibility of sharing
  - Share at two levels: Page or segment (entire page table)

Segments + Pages Disadvantages

Internal fragmentation increases

- Last page of every segment in every process

Increases overhead of accessing memory

- Translation tables in main memory
- 1 or 2 overhead references for every real reference

Large page tables

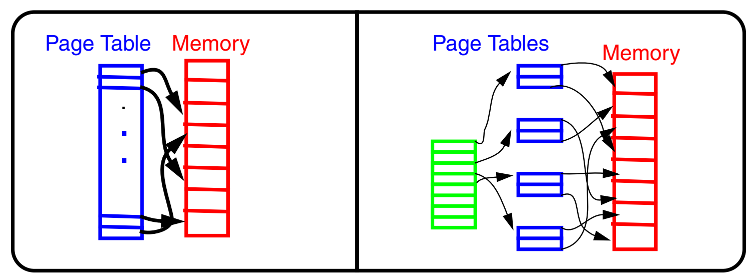
- Do not want to allocate page tables contiguously
- More problematic with more logical address bits

- Two potential solutions:
  - Page the user page tables
  - Inverted page table

Paging Page Tables

Multiple levels of page tables

- Page tables can be scattered (only outer-level must be contiguous)
- Only allocate page tables for pages in use



Implications for logical address structure

Outer Page (x bits)	Inner Page (10 bits)	Page offset (12 bits)

- Inner-level should require exactly one page per table (4 bytes / entry)

Inverted Page Table

Less physical memory than logical memory for all processes

One page table entry for each real page of physical memory

- Each entry: Logical address of page and owning process
- On memory access: Search through physical memory for logical page

Efficient lookup: Hash on logical page number

- Index into hash table to find corresponding page table entry
- Return physical page number

Advantages

- Fast lookup
- Page table space proportional to physical memory

Disadvantages

- Difficult to share pages (one logical addr for each physical addr)

Translation Look-Aside Buffer (TLB)

Locality: Process references few unique pages in time interval

TLB (cache) stores a few translation table entries

- Typical sizes: 64 to 2K entries
- Index by logical segment+page --> Physical base address of page

On each memory reference: Check TLB for page

- If present, fetch physical base address and append page offset
- Else, go through segment and page tables to get base address

  Update TLB for next access

  (TLB discards one of old entry)