

Today: Segmentation

Segments take the user's view of the program and gives it to the OS.

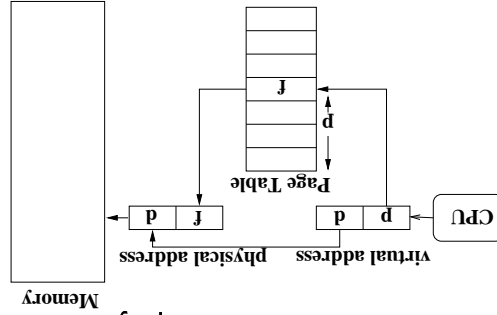
- User views the program in logical *segments*, e.g., code, global variables, stack, heap (dynamic data structures), not a single linear array of bytes.
- The compiler generates references that identify the segment and the offset in the segment, e.g., a code segment with offset = 399
- Thus processes thus use virtual addresses that are segments and segment offsets.

⇒ Segments make it easier for the call stack and heap to grow dynamically. Why?

⇒ Segments make both sharing and protection easier. Why?

Last Class: Paging

- Process generates virtual addresses from 0 to Max.
- OS divides the process onto pages; manages a page table for every process; and manages the pages in memory
- Hardware maps from virtual addresses to physical addresses.

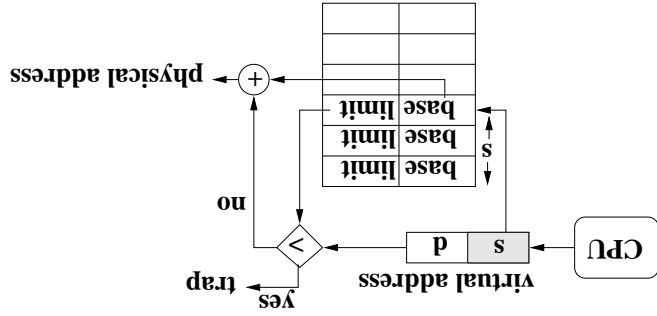


Let's combine the ease of sharing we get from segments with efficient memory utilization we get from pages.

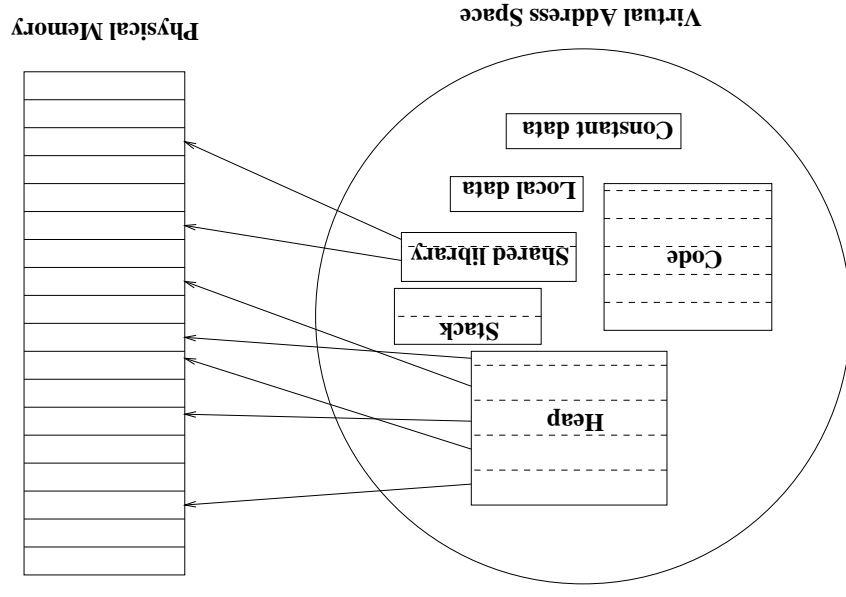
- Similar memory mapping algorithm as paging. We need something like the TLB if programs can have lots of segments
- Segmentation can be combined with a dynamic or static relocation system,
 - Each segment is allocated a contiguous piece of physical memory.
 - External fragmentation can be a problem again
- Compiler needs to generate virtual addresses whose upper order bits are a segment number.

Implementing Segmentation

- Segment table: each entry contains a base address in memory, length of segment, and protection information (can this segment be shared, read, modified, etc.).
- Hardware support: multiple base/limit registers.



Implementing Segmentation



Combining Segments and Paging

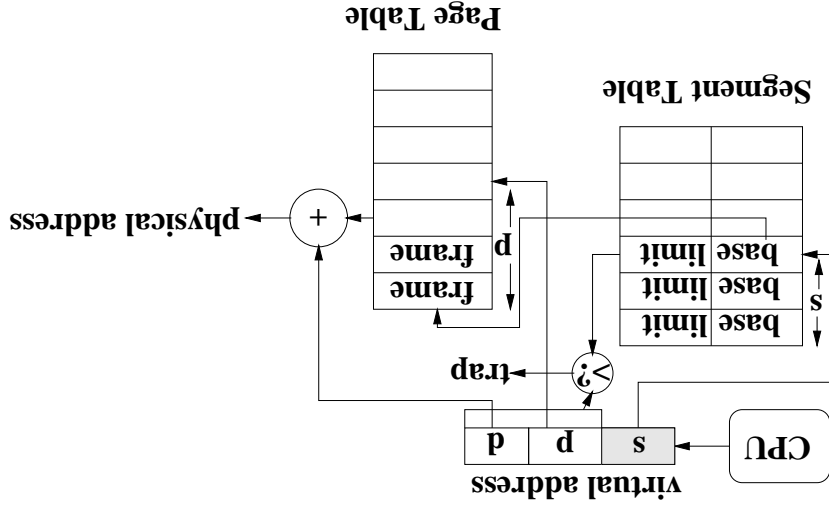
- Treat virtual address space as a collection of segments (logical units) of arbitrary sizes.
 - Treat physical memory as a sequence of fixed size page frames.
 - Segments are typically larger than page frames,
- ⇒ Map a logical segment onto multiple page frames by paging the segments

Combining Segments and Paging

Addresses in a Segmented Paging System

- A virtual address becomes a segment number, a page within that segment, and an offset within the page.
- The segment number indexes into the segment table which yields the base address of the page table for that segment.
- Check the remainder of the address (page number and offset) against the limit of the segment.
- Use the page number to index the page table. The entry is the frame. (The rest of this is just like paging.)
- Add the frame and the offset to get the physical address.

Addresses in a Segmented Paging System



1. How many bits is a physical address?
 2. How many bits is a virtual address?
 3. How many segment table entries do we need?
- Given a memory size of 256 addressable words,
 - a page table indexing 8 pages,
 - a page size of 32 words, and
 - 8 logical segments

Addresses in a Segmented Paging System: Example

- Share individual pages by copying page table entries.
- Share whole segments by sharing segment table entries, which is the same as sharing the page table for that segment.
- Need protection bits to specify and enforce read/write permission.
 - When would segments containing code be shared?
 - When would segments containing data be shared?

Sharing Pages and Segments

Sharing Pages and Segments: Implementation Issues

- Where are the segment table and page tables stored?
 1. Store segment tables in a small number of associative registers; page tables are in main memory with a TLB
 2. Both the segment tables and page tables can be in main memory with the segment index and page index combined used in the TLB lookup
 - (slower but no restrictions on the number of segments per program)
- Protection and valid bits can go either on the segment or the page table entries
- **Note:** Just like recursion, we can do multiple levels of paging and segmentation when the tables get too big.

Segmented Paging: Costs and Benefits

- **Benefits:** faster process start times, faster process growth, memory sharing between processes.
- **Costs:** somewhat slower context switches, slower address translation.
- Pure paging system \Rightarrow (virtual address space)/(page size) entries in page table. How many entries in a segmented paging system?
- What is the performance of address translation of segmented paging compared to contiguous allocation with relocation? Compared to pure paging?
- How does fragmentation of segmented paging compare with contiguous allocation? With pure paging?

Putting it all together

- **Relocation** using Base and Limit registers
 - simple, but inflexible

- **Segmentation:**

- compiler's view presented to OS
- segment tables tend to be small
- memory allocation is expensive and complicated (first fit, worst fit, best fit).
- compaction is needed to resolve external fragmentation.

- **Paging:**
 - simplifies memory allocation since any page can be allocated to any frame
 - page tables can be very large (especially when virtual address space is large and pages are small)

- **Segmentation & Paging**

- only need to allocate as many page table entries as we need (large virtual address spaces are not a problem).
- easy memory allocation, any frame can be used
- sharing at either the page or segment level
- increased internal fragmentation over paging
- two lookups per memory reference