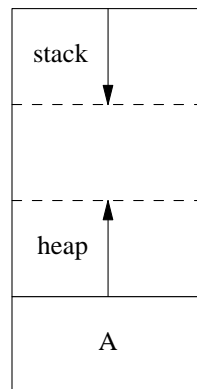


Segmentation (and Paged Segmentation)

Segmentation

Programs generally divide up their memory usage by function. Some memory holds instructions, some static data, some dynamically-located data, some execution frames. All of these memory types have different protection, growth, and sharing requirements. In the monolithic memory allocation of classic VM systems, this model isn't well supported.



Segmentation addresses this by providing multiple sharable, protectable, growable address spaces that processes can access.



In a pure segmentation architecture, segments are allocated like variable partitions, although the memory management hardware is involved in decoding addresses. Pure segmentation addresses replace the page identifier in the virtual address with a segment identifier, and find the proper segment (not page) to which to apply the offset.

Segment	Offset
0101	110101110101
5	d75

The segment table is managed like the page table, except that segments explicitly allow sharing. I'm not going to explicitly discuss protection methods, but the protections reside in the segment descriptors, and can use keying or explicit access control lists to apply them.

Of course, the segment name space must be carefully managed, and the OS must provide a method of doing this. The file system can come to the rescue here - a process can ask for a file to be mapped into a segment and have the OS return the segment register to use. This is known as *memory mapping* files. It's slightly different from memory mapping devices, because one file system abstraction (a segment) is providing an interface to another (a file). Memory mapped files may be reflected into the file system or not and may be shared or not at the process's discretion.

The biggest problem with segmentation is the same as with variable sized real memory allocation: managing variable sized partitions can be very inefficient, especially when the segments are large compared to physical memory. Checkerboarding, or external fragmentation, can easily result in expensive compaction when a large segment is loaded, and swapping large segments (even when compaction is not required) can be costly.

Paged Segmentation

One solution to the problems with pure segmentation is the same one applied to variable partitions: paging. In this case each segment is paged just like address spaces are in pure paging systems. To resolve a memory reference, the MMU:

- masks off the segment ID and uses it to find the relevant segment table. The segment table points to the page table for the segment.
- The page ID is masked off and used to find the physical frame on which the page resides.
- If the frame is not in memory, it is faulted in.
- The offset is added to the page and the relevant physical address fetched (subject to permission constraints).

Of course multilevel page tables (and even multi-level segment tables) can make the memory usage more efficient.

The Multics system implemented such a paged segmentation system. (Multics' strategy was "all the world's a segment."¹ Files, libraries, programs were all memory segments.)

¹ Most interesting research operating systems have an "all the world's a _____" philosophy. Multics → segment, UNIX® → file, Mach → port.