Chapter 17 Free Space Management.

- Managing free space can be easy:
    - When using paging.
    - When the space is divided into fixed-size chunks
        - Use an array listing the chunks.
- When free space managing becomes more complicated.
    - When the free space is of varying sizes.
        - Happens in user level allocations.
        - Happens when OS uses segmentation.
    - External Fragmentation:
        - The free space gets chopped up into little pieces of different sizes.
        - Subsequent requests for memory may fail because there is no single contiguous space of large enough size.

17.1 Assumptions

- Assume a basic interface that returns a pointer to a space of a requrested size in memory.
- When freeing the memory the library must know the size of memory to be freed.
    - The library must know how big a chunk of memory is just by being handed a pointer to it
- The generic data structure used to manage the free space on the heap is a free list.
    - Contains references to all the free chunks of space in the managed region of memory.
- Primarily concerned with external fragmentation.
- Internal fragmentation is when the allocator hands out chunks of

memory larger than requested.
    - Unasked for and unused memory is considered internal fragmentation.
    - It's internal to the allocation rather than external.
- Assume that once memory is handed out it cannot be relocated to another location in memory.
    - Eliminates the ability of compaction.
- Assume the allocator manages a contiguous region of bytes.
    - In some cases allowing for expansion.

17.2 Low-level Mechanisms

- **Splitting and Coalescing**
    - A free list contains a set of elements that describe the free space remaining in the heap.
    - Assume there is a request for a single byte of memory:
        - The allocator will perform splitting.
            - The allocator will find a free chunk of memory that can satisfy the request and split it in two.
            - The first chunk will return to the caller.
            - The second will remain on the free list.
        - The lsit essentially stays intact just missing the allocated portions.
    - Coalescing of free space.
        - If memory is freed returning space in the middle of the heap.
        - If not careful it may divide contiguous spaces into multiple segments.
            - Could have memory request fail because the space is divided into chunks that are too small.
        - If the newly freed space is next to previously freed space

then it is combined into one chunk.

- **Tracking the Size of Allocated Regions**
    - It is assumed that given a pointer the malloc library can determine the size of a region of memory.
    - Most allocators store a bit of extra information in the header block which is kept in memory just before the handed out chunk of memory.
        - The header minimally contains the size of the allocated region.
        - May contain additional pointers to speed up deallocation.
        - A magic number to provide additional integrity checking.
            - Used as a sanity check.
- **Embedding a Free List**
    - How do we build a list inside the free space?
        - Can't use malloc because it's being done inside the malloc library.
    - First the list has to be initialized.
    - The heap is built calling mmap().
    - Then splitting and coalescing is used to manage the memory.
- **Growing the Heap**
    - What should be done when the heap runs out of space?
        - The simplest approach is to let it fail.
        - Most traditional allocators start with a small heap and request more memory from the OS when it runs out.

17.3 Basic Stragies

- The ideal allocator is both fast and minimizes fragmentation.
- **Best Fit**
    - Search through the free list and find chunks of free memory that are as big or bigger than the requested size.

    - Return the one that is the smallest in the group.
    - Performance penalty for exhaustive block search.
- **Worst Fit**
    - Find the largest chunk and return the requested amount.
    - Still exhaustive block search.
    - Mostly shown to perform badly resulting in excess fragmentation.
- **First Fit**
    - Finds the first block that is big enough and returns requested amount.
    - Fast method.
    - Can pollute the beginning of the free list with small objects.
- **Next Fit**
    - Keeps a pointer to the location within the list where one was looking at last.
    - Peformance is similar to First Fit.
    - Designed to prevent the fragmentation of first fit.

17.4 Other Approaches

- **Segregated Lists**
    - If an application has one popular sized request that it makes keep a separate list just to manage that size object.
        - All other requests are forwarded to the main list.
    - Complications:
        - How much memory should be dedicated to the pool.
            - The slab allocator handles that problem.
                - at boot time it allocates a number of object caches for kernel objects likely to be requested frequently.
                - When a cache is running low on memory it

requests a slab of memory from the free list.
            - When the reference counts for a slab are zero the OS can reclaim the unused slab.
        - The slab allocator keeps the free objects on the list ina pre-initialized state.
- **Buddy Allocation**
    - Allocator designed around making coalescing simple.
    - Memory is first thought of as one big space of size 2^N.
    - When memory is requested a recursive search splitting space available in halves until the proper size block is found.
    - Scheme can suffer from internal fragmentation.
        - Blocks are only allowed to be alllocated in sizes of powers of 2.
    - Coalesces at the same time it frees memory.
        - checking is easy because it checks addresses that differ by single bits.
- **Other Ideas**
    - The ideas above suffer from a lack of scaling.
        - List searches can be slow so more advanced data structures are used.
            - Balanced binary trees, Splay trees, partially ordered trees, etc.