Lecture 29, Apr 3, 2024

Memory Allocation

- When declaring normal variables on the stack, the compiler inserts `alloca()` calls, which allocate memory on the stack
  - The function that called `alloca()` frees the memory when it returns, so an explicit `free()` is not needed
- Dynamic allocation via `malloc()` can lead to *fragmentation*
  - A *fragment* is a small contiguous block of memory that is too small to be allocated; like a "hole" in memory which wastes space
    - Every allocation is permanent and contiguous, so between blocks of allocated memory there can be fragments
- Fragmentation requires the following:
  1. Different allocation lifetimes
     - e.g. stack allocation does not suffer from fragmentation since all variables live for the same time
  2. Different allocation sizes
     - e.g. page allocation does not have fragmentation because all pages are the same size, so any block of memory is the same
  3. Inability to relocate (defragment) previous allocations
     - e.g. Java does not have fragmentation since its garbage collector can move blocks of memory and update reference addresses
- *External fragmentation* occurs when allocating different sized blocks, and there is no more room for allocation between blocks
  - This is fragmentation between blocks
- *Internal fragmentation* occurs when allocating fixed blocks, and there is wasted space within a block
  - This is fragmentation within a block
  - e.g. if memory is only allocated in sizes of powers of 2
- To reduce fragmentation, we want to reduce the number of "holes" between blocks of memory
  - If we have holes, we want them to be as large as possible so we can fit future allocations in them
- Allocator implementations usually use a linked list of free blocks
  - When an allocation is needed, choose a free block large enough for the request, remove it for the free list and return it
    - Choosing which block to use requires a strategy
  - When freeing a block of memory, add it back to the free list
    - If it's adjacent to another free block, we can merge the two to get a larger chunk of free memory
- There are 3 general allocation strategies:
  1. Best fit: choose the smallest block that can satisfy the request
     - Requires searching through the entire list unless we come across an exact match
     - Too slow in practice
     - Tends to leave very large and very small holes; smaller holes may be useless
  2. Worst fit: choose the largest block
     - Also requires searching the entire list (can't stop early in this case)
     - Too slow in practice
     - Tends to be the worst in terms of storage utilization
  3. First fit: choose the fist block that can satisfy the request
     - Tends to leave "average" sized holes
     - Much faster and actually used in practice