Scribe Notes CS61 9/26/13

Garbage collection

- garbage collector should be able to clean up memory that is still reachable and not been freed
- need to find piece of memory that is not being used
- Cwoncept:
    - Start from ROOTS (objects accessible by definition)
        - stack, globals
    - Mark all accessible objects
    - Then, unmarked objects are accessible, so free them

How does a garbage collector work?

- Maybe go through static code to see which allocations are not being used?
    - No, because the garbage collector does not have access to this code (it's not the compiler)
- What it does have access to is the memory map inside of the heap.
    - So how can we track the memory blocks that have been allocated?
    - It starts from the assumption that all the memory allocations are there because there are pointers either in the stack frame or the globals (ROOTS) that point to them.
    - However, in benchmark, there are no globals, so we can just mark all accessible objects (through the stack) and the complement would be the pointers we have to free
    - In order to do this, we need to be able to find accessible objects.
        - But how do we find accessible objects?
            - We can go through the stack frame and check if any four bytes are valid pointers.
            - How we'd check if they're valid pointers is by comparing them to our list of pointers in memregion and marking the struct in memregion if it is accessible
            - This is called the sweep-mark method.
    - The garbage collector can be made to run faster by telling it to continue

whenever it encounters a 0 in the memory. This is called "optimizing for the common case"

How to find accessible objects?

- m61_find_allocations in m61.c in lecture 8
    - void m61_find_allocations(char* base, size_t sz)
- to do:mark all objects that have pointers stored in the memory from base to base+sz
- conservative garbage collectors - do less work than the precise amount of work
    - only frees some of the garbage but never frees something that is not a garbage
- when checking, do not assume the structs are properly aligned, instead check starting point at every byte of memory
- off by one error: make sure the last boundary is included in the for loop
- we have every memory pointer so iterate over the the list and see if any of the values of the pointer equals the values in the section of interest
- go through the entire stack, mark all accessible memory, then free everything else
    - marking is done recursively to ensure chasing of all objects that are pointed off of the original
- check to see if you have already marked it to avoid circular linked lists
- getting stack top and stack bottom
    - stack_bottom = (char*)&argc <-- because argc is in the stack and the address would be in the stack
    - 1 block remaining is the mem region - the array itself

Garbage collecting before end of program

- macro definition: #define m61_free(x) will mean that free will not run
- at its most extreme, a precise garbage collector means you will never need free
- taking a long time because recursively searching through all of the malloc objects
- optimizing for the common case - null is common case for memory
- not freeing any memory, just overwriting pointers
- what are the few left over pointers?
    - random collection of bytes in memory looked like a pointer so the conservative pointer didn't free it

Storage in general - what gets stored where and why?

- Money
    - A - cost of 1 MB of memory storage in 1955
    - B - cost of 1 MB hard disk in 2012
        - What is the ratio of A/B? 11 trillion
    - Cost of cache > cost of memory > cost of SSD > cost of hard disk
- Latency
    - Smaller = better
    - Register: 0.5ns
    - Hard disk: 5-9ms
    - Hard disk random access: really slow (~1 MB/s)
- Durability
    - Does data persist without power?
    - Hard disk - yes
    - Memory - no
- Demonstrations of data storage speed
    - w01-syncbyte
        - Writes a single byte to disk, waits for confirmation that byte has been written to disk, then writes the next byte
        - Writes at 3000 bytes/second to hard drive
    - Optimizations: batching
        - since writing to disk is so slow, fewer writes might make things go faster. Try batching writes in blocks of 512 bytes?
        - w02-syncblock
            - Does the same thing as w01-syncbyte, but writes in blocks of 512 bytes
            - Writes at speed of ~10^6 bytes/second
        - w03 -byte
            - Writes a single byte at a time without waiting for confirmation that

data was written to disk
        - Writes at ~900,000 bytes/second
    - w04 - block
        - same as wo3-byte, but writes in blocks of 512 bytes
        - Writes at ~ 9 * 10^7 bytes per second
- stdio seems to have some pretty efficient methods; goes much faster than syncbyte or syncblock
- Homework - beat stdio.