Scribe Notes 9/24

Great time of year to go to fresh pond at sunset, subway to Alewife take short walk

Types of memory in a process

- stack (stores local variables), heap (stores dynamically allocated variables), code and data area (stores global variables)

Local and global variables are managed by the compiler

Heap is managed by you

Global variables last until exit, stack variables last as long as the current stack frame, dynamic variables have a dynamic lifetime

Malloc & free

Throughput – number of operations per unit time (higher is better)

- how fast malloc and free are

Utilization – amount of some resource given to useful work (higher is better)

- written as number between 0-1
- resource is memory, useful is amount asked for by user (application memory)

Fragmentation is a problem that can cause low utilization

C abstract machine assumes infinite memory, so not a constraint in actual implementation of malloc, but real life constraint on a machine

Constraints for Malloc

- Allocate any size
- Active allocated objects don't overlap
- Allocated objects have to be 8 byte aligned
- Frees can occur in an order

Testing implemenations of malloc and free with membench.c
Malloc vs tcmalloc vs jemalloc

-Malloc beat google and facebook's malloc with 1 thread
-Google and facebook's malloc beat malloc with more than 1 thread

Threads – simulating having multiple processors on a computer

Malloc libraries usually written in C, sometimes C++

Difficulties of malloc when dealing with multiple threads

- One thread might have to malloc, other have to free
- Active allocated objects don't overlap (threads have to abide by this)

Goal: beat google's malloc for a fixed size memory allocator

- don't have to do coalescing
- don't have to combine memory blocks

can take advantage of these facts to make fixed size allocator fast
can use batching to speed up our program

One thing to speed up our program is to allocate memnodes in groups (arenas) and use a linked list to link them together and keep track of what has been freed and what blocks of memory are still free

An arena is an object that encapsulates a set of allocations (commonly used term in fixed size allocators)

Fragmentation:

- specific type of overhead
- not headers/trailers, not linked lists
- linked to issues caused by variable size

- External fragmentation:
  - unusable space between allocations
  - ex:
    - x = malloc(5)
    - y = malloc(2000)
    - z = malloc(10) // right now, have memory chunks of 5, 2000, 10 in a row
    - free(y)
    - w = malloc(2001) // can't take the space where y was, so external fragmentation
- Internal fragmentation:
  - unusable space inside an allocation
    - ex. If you call malloc(1), you will still get at least 8 bytes since the next malloc call must be 8 byte aligned

- fixed-size allocators always avoid external fragmentation
  - space between two allocations all same, so new allocations can take that spot

Common for computer scientists to look down at speed ups by a factor of 2

Garbage collection:

- essentially just forget about free; goes through and frees unused memory
- In Java, memory is logically freed after the last time it is dereferenced
- Garbage collector goes around looking for leaked memory and frees it when it finds it
- Problems with calling free manually
    - Forget to do it; call free multiple times; nasal demons everywhere (security break ins, crashes, etc.)
- Garbage collection way easier to use, has all benefits of dynamic memory management, but don't have some of the costs like freeing at same time
- cost of garbage collection: time. (takes a lot of time to scan through all of memory)
    – any allocation could potentially cause the garbage collector to freak out
    - "Oh, now's a good time to go through literally all of memory…" – d'oh!

Garbage collector: manyalloc.c
How many allocations? 1001, 1000 allocations for each element of array and 1 for array itself

Garbage collectors scan memory for existing references. It then frees any allocated memory that is not referenced.