Slide 1

Administrivia

- Reminder: Homeworks 2a, 2b due Thursday.
- Attendance reports generated and uploaded. Current through noon 10/11.

Slide 2

Managing Free Space

- Problem of managing varying-size chunks of memory arises in more than one context — space within process (`malloc()` and `free()`), space among processes.

  (Note in passing that if using the textbook's version of base/bound method of allocating space to processes, problem doesn't really arise — all chunks are same size and we only have to keep track of which are free.)

- Since chunks vary in size, "external fragmentation" a problem. Compaction can sometimes help, but not always:

  Consider whether you can do that with space managed by `malloc()` and `free()`. How can you find all references?

Slide 3

Managing Free Space — Mechanisms

- Manage in terms of a "free list" (list of free regions, each with location and size).
- Basic operations on this list: "split", "coalesce".
- Splitting happens when space is allocated, if we assign part of a free region.
- Coalescing is kind of optional, but hard to imagine a practical system without it — idea is that if we have two free regions next to each other, merge into one.
- Since `free()` doesn't allow specifying size, actual allocated space typically includes header specifying size.

Slide 4

Managing Free Space — Mechanisms, Continued

- Free list itself has to be kept somewhere! Linux puts it within the space being managed(!). Details interesting but could be skipped/skimmed.
- Where does the space come from in the first place? Call to `mmap()`. Interesting function in that it can either copy contents of a file (all or part) directly into memory or just reserve an area of memory.

  Worth noting that `mmap()` is specific to Linux and some other UNIX-like systems. But other operating systems likely have something alone the same lines.

Slide 5

Managing Free Space — Policy/Strategy

- Idea of free list, split / coalesce — mechanisms.
- Unaddressed so far — question of *which* block of free list to split.
- Several strategies, some relatively simple (name almost tells you everything you need to know), others less so.

Slide 6

Simple Strategies

- Best fit.
- Worst fit.
- First fit.
- Next fit.

Slide 7

Less Simple Strategies — Segregated Lists

- One approach — keep several free lists for chunks of various commonly-used sizes, plus a general list for all other sizes.
- Idea behind "slab allocator", which the textbook authors seem to admire. Does sound good.
- (Interesting tangential observation about what "great" means in some fields.)

Slide 8

Less Simple Strategies — Buddy Allocators

- Basic idea here is to make coalescing simple.
- Binary buddy allocator allocates space in sizes that are powers of two (and presumably of some minimum size).
- When a chunk is free, only one other chunk it could be merged with (its "buddy"). The result might also need to merged with *its* buddy — recurse until buddy is not free (or until everything free?).

Slide 9

Other Strategies

- Textbook says ways to solve this problem (managing collection of varying-size chunks of free space) an active area of research.
- One point to keep in mind — good strategy will "scale" well. So "list" may in fact be some much more exotic(?) data structure (like the red-black trees used by the Linux scheduler?).

Slide 10

Minute Essay

- Questions?
- Had you thought previously about how complicated it might be for `malloc()` to get memory for you?