CS 326: Operating Systems

Custom Memory Allocator

Lecture 15

Manual Memory Allocation - C

- In C, we use **malloc** and **free** to allocate and free up memory, respectively
- When you took your first C class, you may have heard this referred to as “manual memory allocation”
- **However**, the C library is actually doing some of the work for you
  - (believe it or not)

malloc

- When you call malloc, several things need to happen
- The first thing we need to do is ask the OS for the memory we are about to use
  - As we discussed in the last couple weeks, memory isn't just static start + end pointers
  - The OS is in charge of who gets memory and takes care of address translation
    - Logical → physical memory addresses

Getting Some Space

- There are a few ways to request memory…
- One is an ancient, crufty syscall: **sbrk**
  - …and its cousin, **brk**
- brk – set the size of the heap
- sbrk – increase the size of the heap

How Old, and How Crufty?

- **macOS**: "The brk and sbrk functions are historical curiosities left over from earlier days before the advent of virtual memory management."
- **FreeBSD**: "The brk() and sbrk() functions are legacy interfaces from before the advent of modern virtual memory management."
- **Linux**: "brk() and sbrk() change the location of the **program break**, which defines the end of the process's data segment. Increasing the program break has the effect of allocating memory to the process; decreasing the break deallocates memory."

An Alternative: **mmap**

- We can create **anonymous** mappings with mmap to have the OS allocate chunks of memory for us
- When we don't need it anymore? **munmap**
- Why can't we just translate these functions?
  - malloc → mmap
  - free → munmap
- The main reason: free() takes a single pointer, while munmap takes a pointer and mapping size

Replacing **malloc**

- We can use mmap as a basic version of malloc. Whenever, we need space, we'll just allocate it as necessary by passing in the size we need!
  - `int *x = (int *) mmap( ... );`
  - And really, who even uses ***malloc***?! We use small-batch, artisanal memory allocation in San Francisco
- Of course, your algorithm has to be a bit smarter. We'll also allocate a **page** at a time
  - You'll need to split pages for subsequent allocations

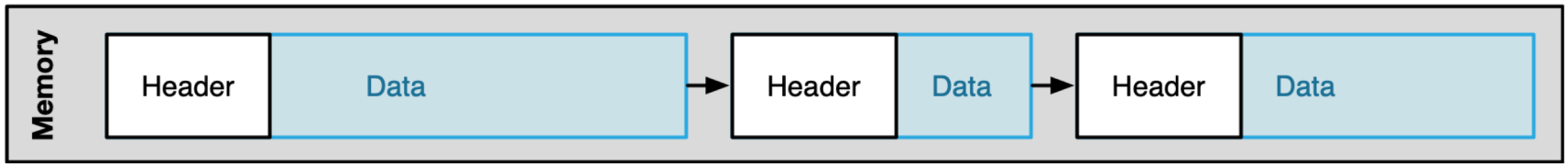
Replacing **free**

- We've allocated memory with **mmap**… now, how do we free it?
- What we need to be able to do is mark some of the memory as invalid and reuse it
- How can we track what memory is in use and what has been freed?

Tracking Free Space

- We could keep a table somewhere with memory information stored in it
  - What would be the downsides of this approach?
- An easier solution: let's prefix each region of memory that we allocate with a struct
  - The struct will tell us how large the region of memory is and whether it is available

Struct Prefix



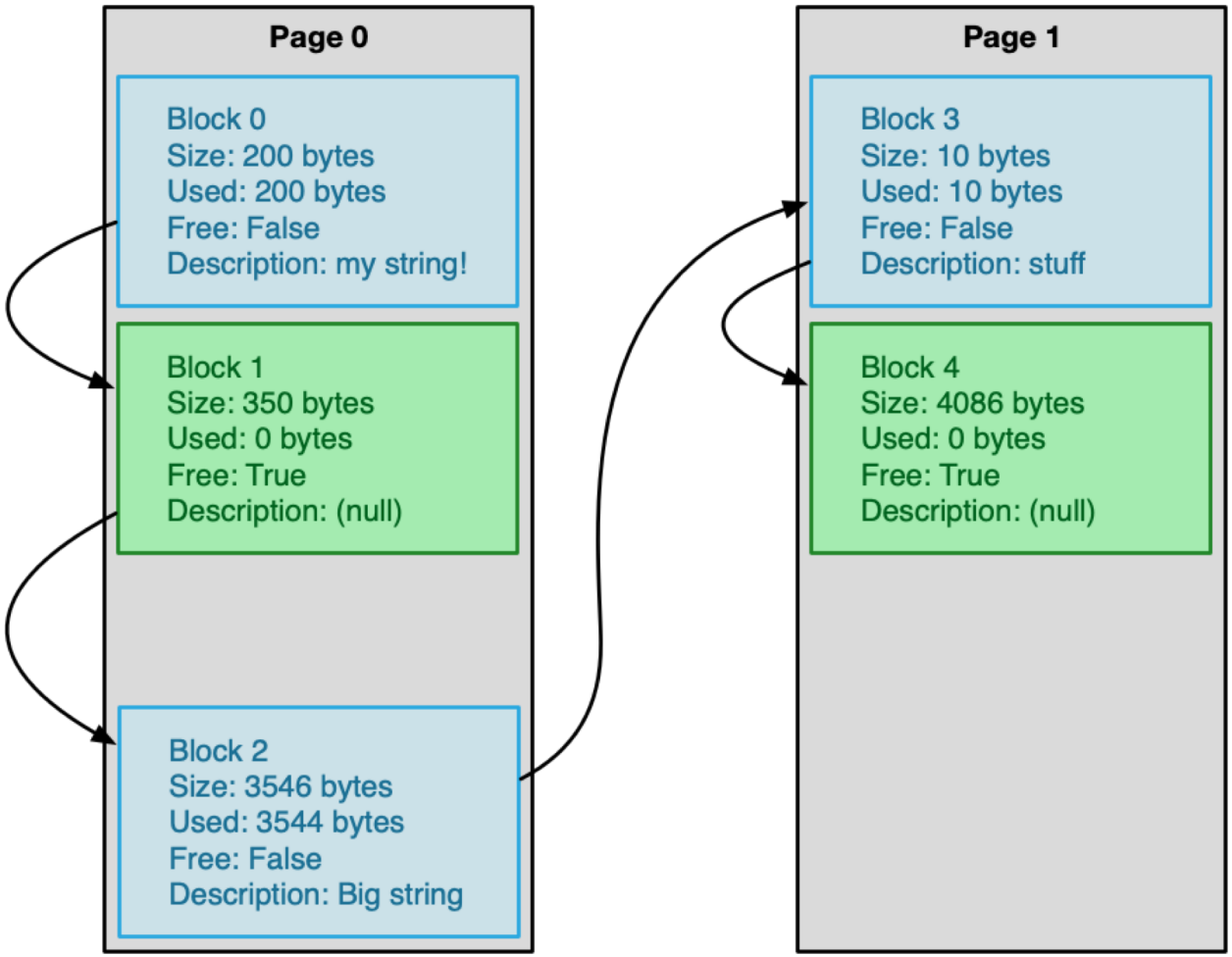
Finding Free Space

- The struct approach is great because now we can find out how much memory to eliminate and mark some of the blocks as unused
- However, how do we find these unused blocks?
- The final piece of the puzzle: linking each struct to the next
  - We'll create a linked list of memory locations
  - This way we can scan through to find free blocks!

Free Space Management

- You will support 3 free space management algorithms in Project 3:
  - First fit
  - Best fit
  - Worst fit
- You can configure these with environment variables (see the spec for example code)

Pages


Page 0
Block 0
Size: 200 bytes
Used: 200 bytes
Free: False
Description: my string!
Block 1
Size: 350 bytes
Used: 0 bytes
Free: True
Description: (null)
Block 2
Size: 3546 bytes
Used: 3544 bytes
Free: False
Description: Big string
Page 1
Block 3
Size: 10 bytes
Used: 10 bytes
Free: False
Description: stuff
Block 4
Size: 4086 bytes
Used: 0 bytes
Free: True
Description: (null)

Project 3

- We'll implement our own malloc!
- Unlike regular malloc, ours has several neat features:
  - It can log each memory allocation/deallocation/etc. to a file
  - We can name blocks of memory
  - We can iterate through memory to print each block and its metadata
  - And more…

Getting Started

- If you want some guidance, watch the P3 video
- Two of the questions on Q4 will be based on memory allocator basics
- You will need to work on this while you wait for P2 interactive grading to happen
- You can work with a partner! You just need to tell us your team members by Tuesday
  - After that, you are required to work alone
- This is a smaller project. You will need to turn it around quickly. Don't wait until next week to start!