15-213 Recitation 10

Introduction to Computer Systems

Fahim Dalvi

7 November, 2013

Malloc Lab

- Due 20th November, 2013
- Submit on autolab constantly!

ONLY 318

HOURS LEFT!!!!

The Idea

- Create a general-purpose allocator that dynamically modifies the size of the heap as required.
- The driver calls your function on various trace files to simulate placing data in memory.
- Grade is based on:
  - Space utilization (minimizing fragmentation)
  - Throughput (processing requests quickly!)
  - Your heap checker
  - Style

You will implement:

- **`mm_init`:** initializes the heap
- **`malloc`:** returns a pointer to an allocated block
- **`calloc`:** same as malloc, but ***zeroes*** the memory first
- **`realloc`:** changes the size of a previously allocated block
- **`free`:** frees previously allocated memory
- **`mm_checkheap`:** debugging function → Something that will save your life

You can use:

- **mem_sbrk**
  - Used for expanding heap size
  - Allows you to dynamically change the heap size
- **mem_heap_lo:** Pointer to first byte of heap
- **mem_heap_hi:** Pointer to last byte of heap
- **mem_heapsize**
- **mem_pagesize**

Design

- You have a ton of decisions to make!
- Think about fragmentation
- Spend time thinking about each and every decision you make
- You have to decide on:
  - How will you manage free blocks → Implicit, Explicit or segregated
  - Policy for finding free blocks → First fit, Next fit, Best fit
  - Coalescing → Immediate, Lazy, etc…
  - And a lot more…..

Design

Think about each and every decision you make!

What happens if you use immediate coalescing with explicit lists?

Are implicit lists good enough?

Fragmentation

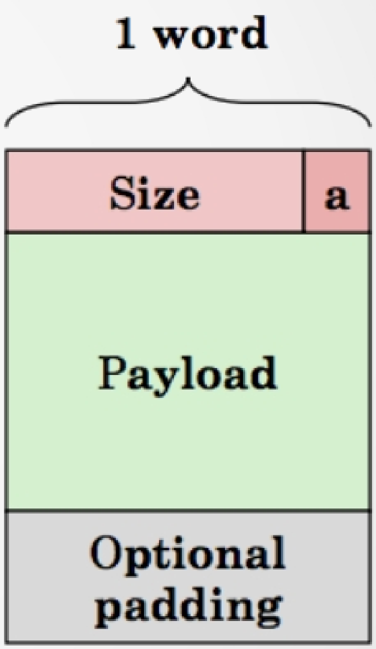
- Fragmentation affects the **space utilization** part of your grade
- Internal fragmentation
  - Results from the fact that payload's may be smaller than the block size itself
    - Header and Footer
    - Padding
    - Alignment
  - Quite unavoidable!

Fragmentation

- External fragmentation
  - Occurs when there is enough aggregate heap memory, but no free block is large enough
  - Some policies are better at minimizing this, you'll have to figure out what is your best option

Managing free blocks

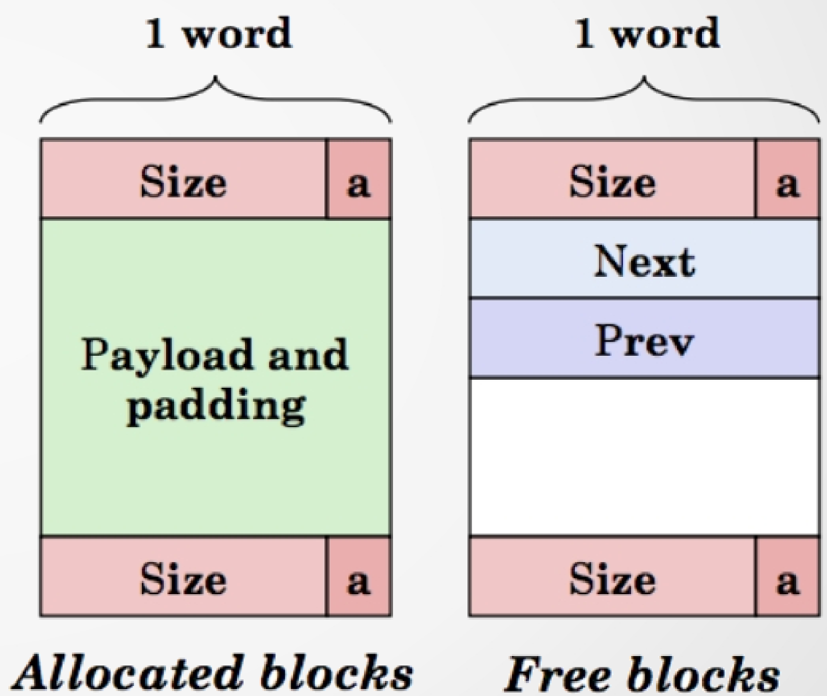
- This affects that **performance** part of your grade
- **Implicit list**
  - Uses block lengths to "find" the next block
  - Connects *all* blocks
  - Takes some extra space (for the "header")
  - Optional "footer" may take some more space



Format of allocated and free blocks

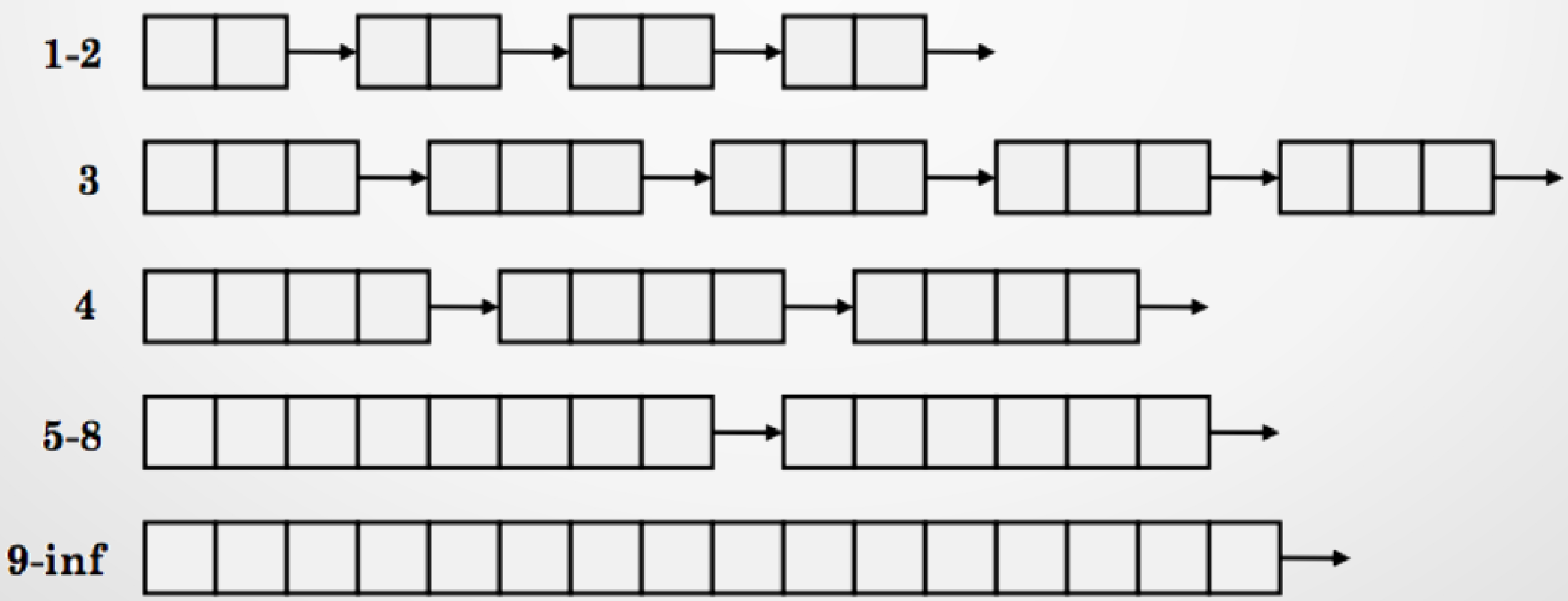
Managing free blocks

- **Explicit list**
  - A list of free blocks
  - Use the "payload" part of the free block
  - Why is this better than implicit lists?



Managing free blocks

- **Segregated list**
  - “size” classes
  - Very easy to find the “appropriate” free block



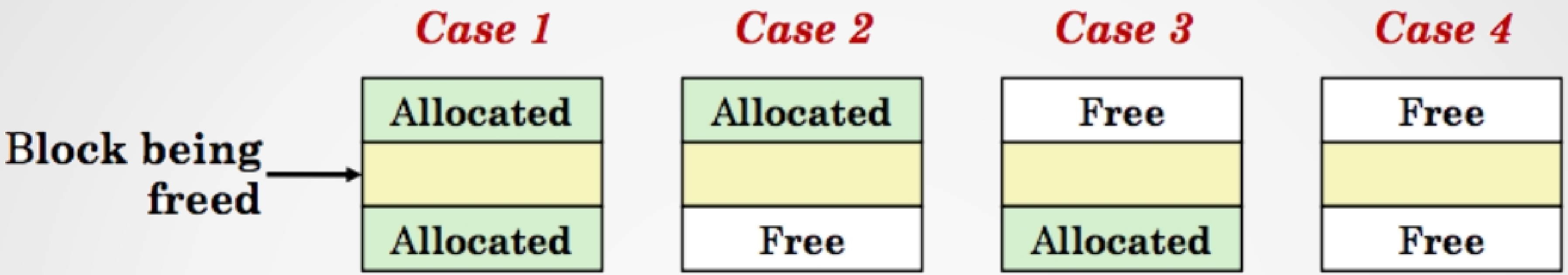
Finding free blocks

- First fit
  - Start from the beginning everytime
  - Find the first free block
- Next fit
  - Continue searching where last search ended
  - Is atmost as slow as first fit, but generally better
- Best fit
  - Choose a block such that it minimizes fragmentation
  - Great for space utilization, bad for performance (Why?)
- What if you don't find a free block thats big enough?
  - Extend the heap

But what about new free blocks..?

- Do you insert in it the beginning of each list? End of each list?
  - Really fast, but may result in slower “free block finding”
- Do you insert it in the “logical” place?
  - Slower, but research shows this may decrease fragmentation

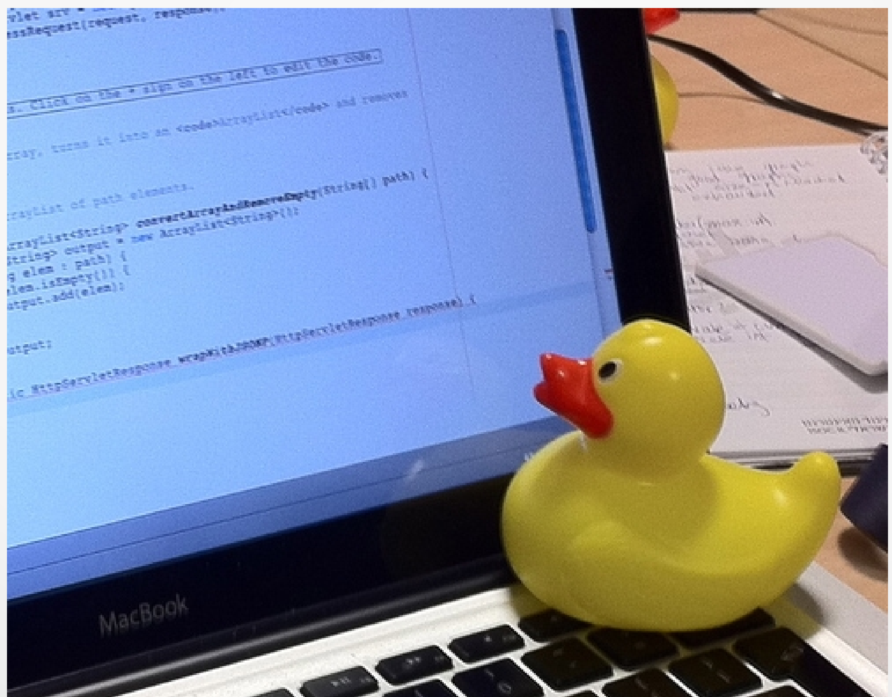
Coalescing policy



- Implicit lists
  - Write new size in the header of first block & footer of last block
- Explicit lists
  - You have to add this to your "free" list
- Segregated lists
  - Like explicit, but you must insert it in the right "size" class

Debugging

- The idea behind malloc is quite simple
- On the other hand, the debugging is very hard



Debugging

- The idea behind malloc is quite simple
- On the other hand, the debugging is very hard
- mm_checkheap would be your best friend!
  - Most problems arise from the fact that your "headers or footers" are not properly maintained
  - Will help you solve "simple" problems and avoid unnecessary hypertension

mm_checkheap

- Don't worry about efficiency here, this function is not tested for performance
- Make sure you concentrate on correctness
- Check for things like:
  - Consistency of pointers
  - Consistency of headers/footers
  - Address alignment
  - Whether free blocks are coalesced when they are supposed to
- The more checks the merrier!

Suggested plan

- Implement the basic “implicit” list first
- Get to **60%** performance, don't worry about being smart
- ONCE you have this safe, think about better implementations and ideas

Some helpful stuff

- Pointer casting can be tricky

```
int *ptr     =  0x10203040
char *ptr2   =  (char *)ptr + 2 → ?
char *ptr3   =  (char *) (ptr + 2) → ?
```

Some helpful stuff

- Macros
  - `#define some-text replace-text`
  - These are “faster” than functions
  - Basically search-and-replace before the compiler starts compiling
  - Hopefully you have used these before like:

```
#define MAGIC_NUMBER 513
```

Some helpful stuff

- Macros
  - But there are more powerful, can act like "functions"
  - `#define ADD2(x) (x+2)`
  - `#define SUM(x,y) (x+y)`
  - **Use parenthesis** all the time!
    - `#define MULT1(x) 2*x`
    - `#define MULT2(x) 2*(x)`
    - What happens if `x=5+1`

Some helpful stuff

- Inline functions
  - Real functions, but still faster than normal functions
  - Compiler replaces each call with the actual "code"
    - Avoids overhead of stack setup
  - Useful for small functions

```
inline int max(int a, int b) {
   return (a > b ? a : b);
}
```

Any Questions?