CS 354 - Machine Organization & Programming
Tuesday Oct 10, and Thurs Oct 12, 2023

Project p3: Released DUE on or before Friday Oct 27

Activities A06 available

Homework 3: DUE on or before Monday Oct 22

Exam 1: Scores posted by Thursday (I hope)

Learning Objectives

- identify and describe options for implementing dynamic memory allocator
- write code that splits a free heap block into one alloc'd and one free block
- write code to create/update heap block header and add/update free block footer
- implement various placement policies to choice an available free block
- run and write tests to ensure correct implementation of heap library
- understand and describe the effect of various allocator design choices
- describe and explain the C/IA-32 memory hierarchy
- Use a Makefile to build a shared object file, and run tests to confirm its implementation

This Week

Placement Policies Free Block - Too Large/Too Small Coalescing Free Blocks Free Block Footers	Explicit Free List Explicit Free List Improvements Heap Caveats Memory Hierarchy
Next Week: Locality and Designing Caches B&O 6.4.2	

p3 Progress Dates (do expect to need multiple days and work sessions for p3)

- complete Week 6 activity as soon as possible
- review source code functions before lecture this week
- write code to see if you have computed the correct heap block size
- write and test code to determine size from size_status, and status from size_status field
- implement **balloc** by Friday this week and submit progress to Canvas (pass partA tests)
- implement **bfree** by Tuesday next week and submit progress to Canvas (pass partB tests)
- implement immediate coalescing by Thursday next week and submit progress
- test and debug to ensure that immediate coalescing and best-fit allocation occur as required.
- complete testing and debugging and complete final submission (partC&D)

Free Block - Too Large/Too Small

What happens if the free block chosen is bigger than the request?

- use the entire block

(–) mem util: more internal frag.

(+) thruput: fast and simple code

(–) must pre-divide heap into various sized smaller block

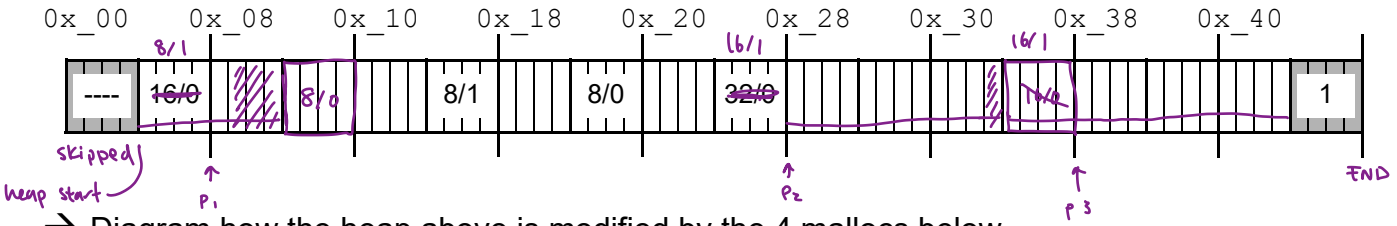
- Split the block, 1st is alloc'd, 2nd is FREE block

(+) mem util: less internal frag.

(–) thruput: slower, must search, and more heap blocks

(+) Can be fast O(1)

Run 4: Heap First-Fit Allocation with Splitting



→ Diagram how the heap above is modified by the 4 mallocs below.
For each, what address is assigned to the pointer?
If there is a new free block, what is its address and size?

	HDR + PAD	PTR:	FREE Block addr:	size:
1) `p1 = malloc(sizeof(char));`	1 + 4 + 3 = 8	0x_08	0x_0C	8
2) `p2 = malloc(11 * sizeof(char));`	11 + 4 + 1 = 16	0x_28	0x_34	16
3) `p3 = malloc(2 * sizeof(int));`	8 + 4 + 4 = 16	0x_38	no free	
4) `p4 = malloc(5 * sizeof(int));`	Alloc fail			

What happens if there isn't a large enough free block to satisfy the request?

1st. condense adj free blocks?

→ Can allocated blocks be moved out of the way to create larger free areas?

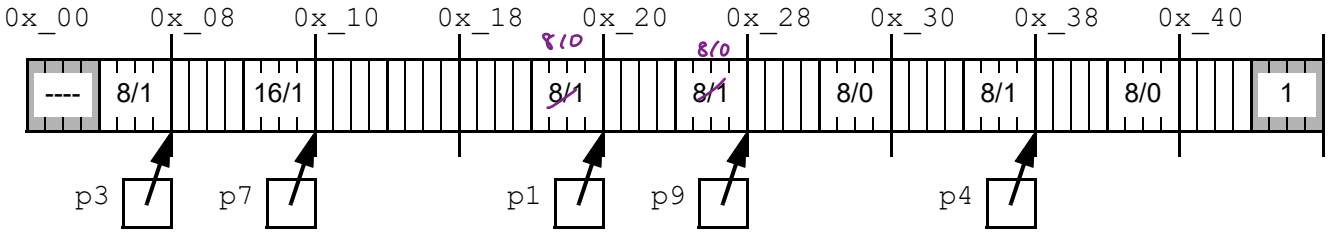
p3

2nd. ask kernel for more heap

3rd. return NULL, alloc fails

Coalescing Free Blocks

Run 5: Heap Freeing without Coalescing



→ What's the problem resulting from the following heap operations?

```
1) free(p9); p9 = NULL;
2) free(p1); p1 = NULL;
3) p1 = malloc(4 * sizeof(int));
```

16 + 4 + 4 = 24 alloc fails

Problem? false frag

have large enough contiguous free block

but divided into blocks that are small

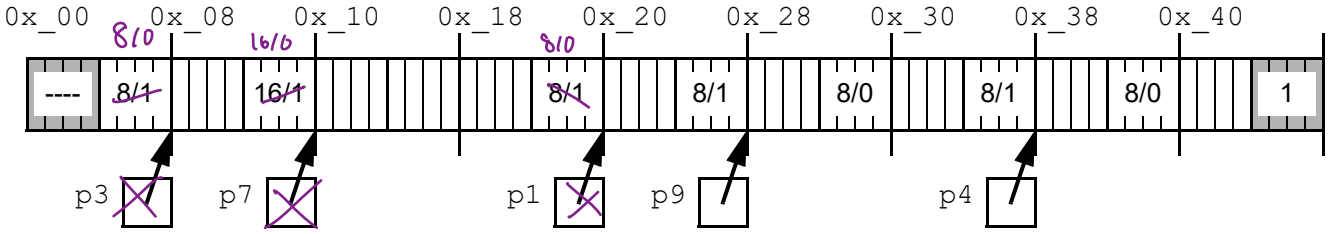
Solution? coalesce (merge) adj free blocks

immediate: coalesce w/ next and prev on free operation

delayed: coalesce only when needed by an alloc

called: coal. only if usr calls coal. fnc.

Run 6: Heap Freeing with Immediate Coalescing



→ Given the heap above, what is the size in bytes of the freed heap block?

```
1) free(p7); p7 = NULL;
```

free 16 bytes + next not free + prior not free

→ Given a pointer to a payload, how do you find its block header?

- 4 bytes or ptr - 4

→ Given a pointer to a payload, how do you find the block header of the NEXT block?

+ "size of curr block" - 4

❋ *Use type casting* to set correct scale factor

→ Given the modified heap above, what is the size in bytes of the freed heap block when immediate coalescing is used?

```
2) free(p3); p3 = NULL;
3) free(p1); p1 = NULL;
```

freed size:

24 / 0

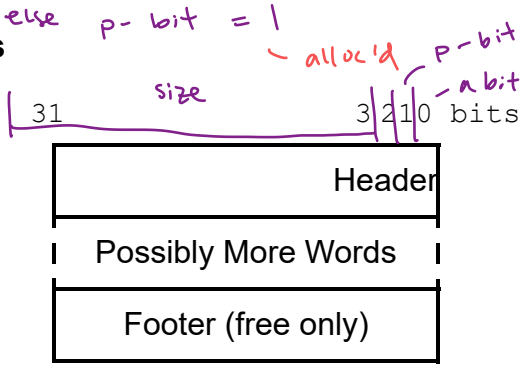
32 / 0

→ Given a pointer to a payload, how do you find the block header of the PREVIOUS block?

Free Block Footers

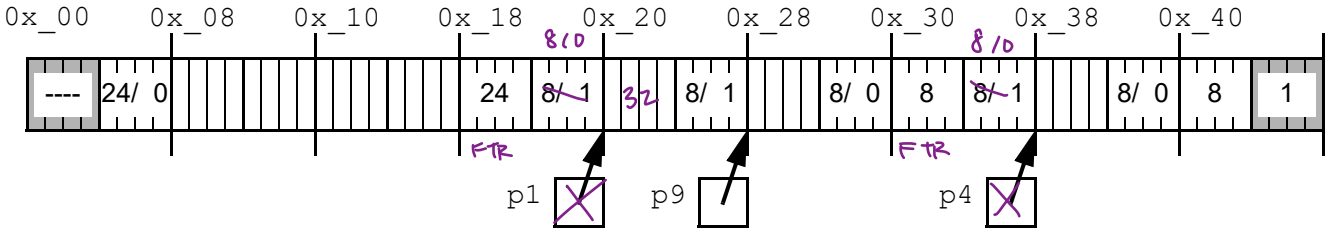
- *The last word of each free block* is faster containing free block size
  - → Why don't allocated blocks need footers? because NOT COALESCED
  - → If only free blocks have footers, how do we know if previous block will have a footer?
    need a p-bit (prev block alloc'd) in HDR
- *Free and allocated block headers* also create size + p-bit + a-bit
  if prev block's free p-bit = 0 else p-bit = 1

Layout 2: Heap Block with Headers & Free Block Footers



- → What integer value will the header have for an allocated block that is:
  1) 8 bytes in size and prev. block is free? 8+1 9
  2) 8 bytes in size and prev. block is allocated? 8+1+2 11
  3) 32 bytes in size and prev. block is allocated? 32+1+2 35
  4) 64 bytes in size and prev. block is free? 64+1 65
- → Given a pointer to a payload, how do you get to the header of a previous block if it's free?
  1. ptr -4 use sizeof to get to curr block HDR
  2. check p-bit if p-bit is 0, then prev block is free, ptr -8 to prev block footer
  3. ptr -4 - "prev block size"

Run 7: Heap Freeing with Immediate Coalescing using p-bits and Footers



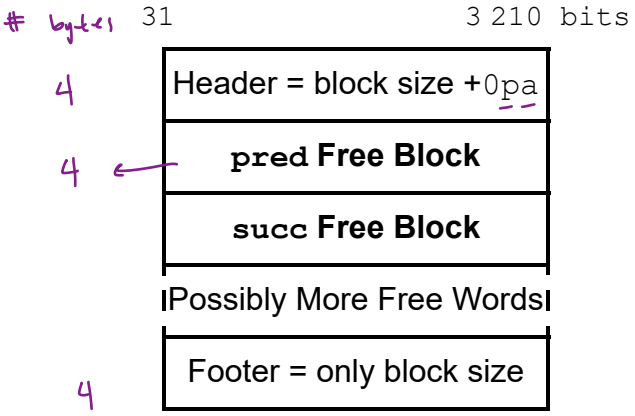
- → Given the heap above, what is the size in bytes of the freed heap block?
  1) `free(p1); p1 = NULL;` 32 bytes
- → Given the modified heap above, what is the size in bytes of the freed heap block?
  2) `free(p4); p4 = NULL;` 24 bytes
- *Don't forget to update* the next block's p-bit when needed (don't set p-bit end mark) that would be BAD
  - ➢ Is coalescing done in a fixed number of steps (constant time) or is it dependent on the number of heap blocks (linear time)?

Explicit Free List

❋ *An allocator using an explicit free list* keeps a list of free blocks

the EFL can be integrated into the heap

Explicit Free List Layout: Heap Free Block with Footer



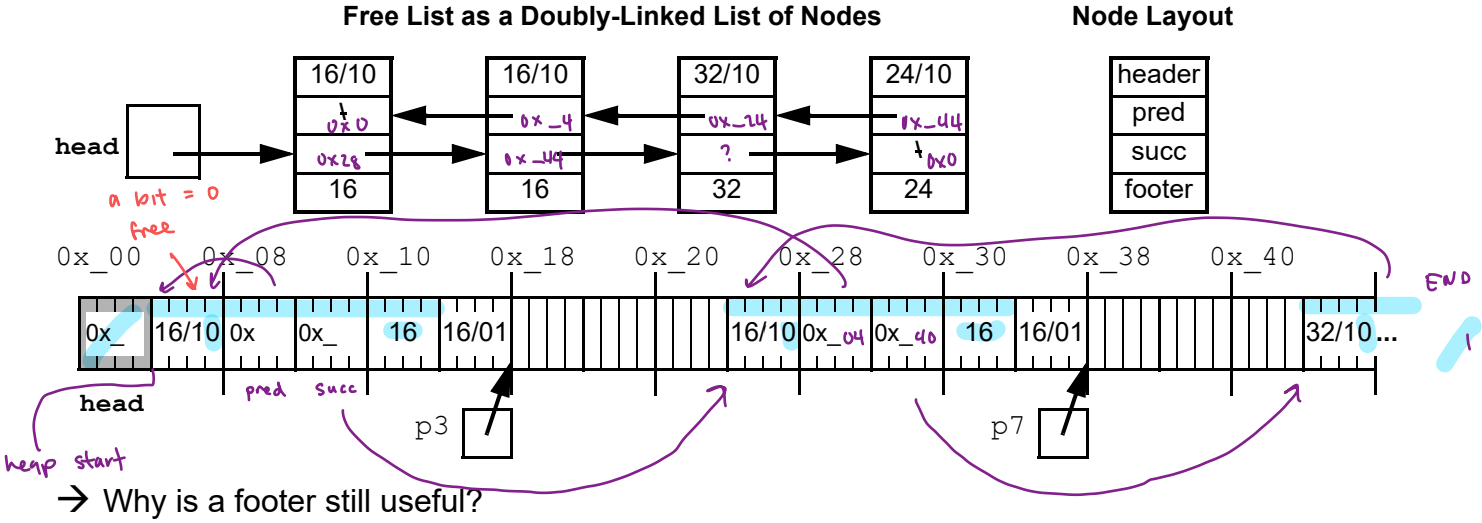
Free Block Links

pred addr of free block before

succ " " " after

min size : 16 bytes

→ Complete the addresses in the partially shown heap diagram below.



→ Why is a footer still useful?

for faster coalescing with prev free block

→ Does the order of free blocks in the free list need to be the same order as they are found in the address space?

Explicit Free List Improvements

Free List Ordering

address order: order free list from low to high addr

better than

(+) malloc with FF has better mem util than "last-in" order

(-) free slower O(N) where N is # free blocks

last-in order:

(-) malloc with FF place most recently freed block at front of F.F.L

(+) free ~ looks at most recently free'd blocks first (+) for progs that request same size

(+) O(1) → just link at head of F.F.L.

(+) O(1) → coalesce with footers

Free List Segregation

use an array of free lists segregated by size

malloc chooses appropriate free list based on req. size

simple segregation: 1 free list per block size

structure simple - no header, blocks only need succ ptr

malloc fast O(1)? choose block at front of approp. list

if free list is empty, get more mem from O.S., divide into blocks, add to EFL

free fast O(1), link to front of approp. free list. NO COALESCING

problem? fragmentation, internal splitting or coalescing

fitted segregation: one list for each range sm, med, lg.

(+) mem util, as good as best fit

(+) thruput, since search only part of heap

fitting do F F search of approp free list, if FAIL search next lgr size list

splitting yes, puts rem free block in approp free list of smaller blocks

coalescing yes. puts new free block in approp free list larger sizes

Heap Caveats

Consecutive heap allocations don't result in contiguous payloads!

→ Why? payloads are interspersed by heap structures & padding

placement policies and heap structure can scatter alloc. thruout heap

Don't assume heap memory is initialized to 0!

O.S. initially clear for security

? but recycled heap will have old data unless calloc

Do free all heap memory that your program allocates!

→ Why are memory leaks bad? they slowly kill performance by cluttering heap w/ garbage block

→ Do memory leaks persist when a program ends? No, heap pages are returned to O.S. when prog ends

Don't free heap memory more than once! UNDEFINED

→ What is the best way to avoid this mistake?

set ptr = NULL

Don't read/write data in freed heap blocks!

→ What kind of error will result? intermittent errors

Don't change heap memory outside of your payload!

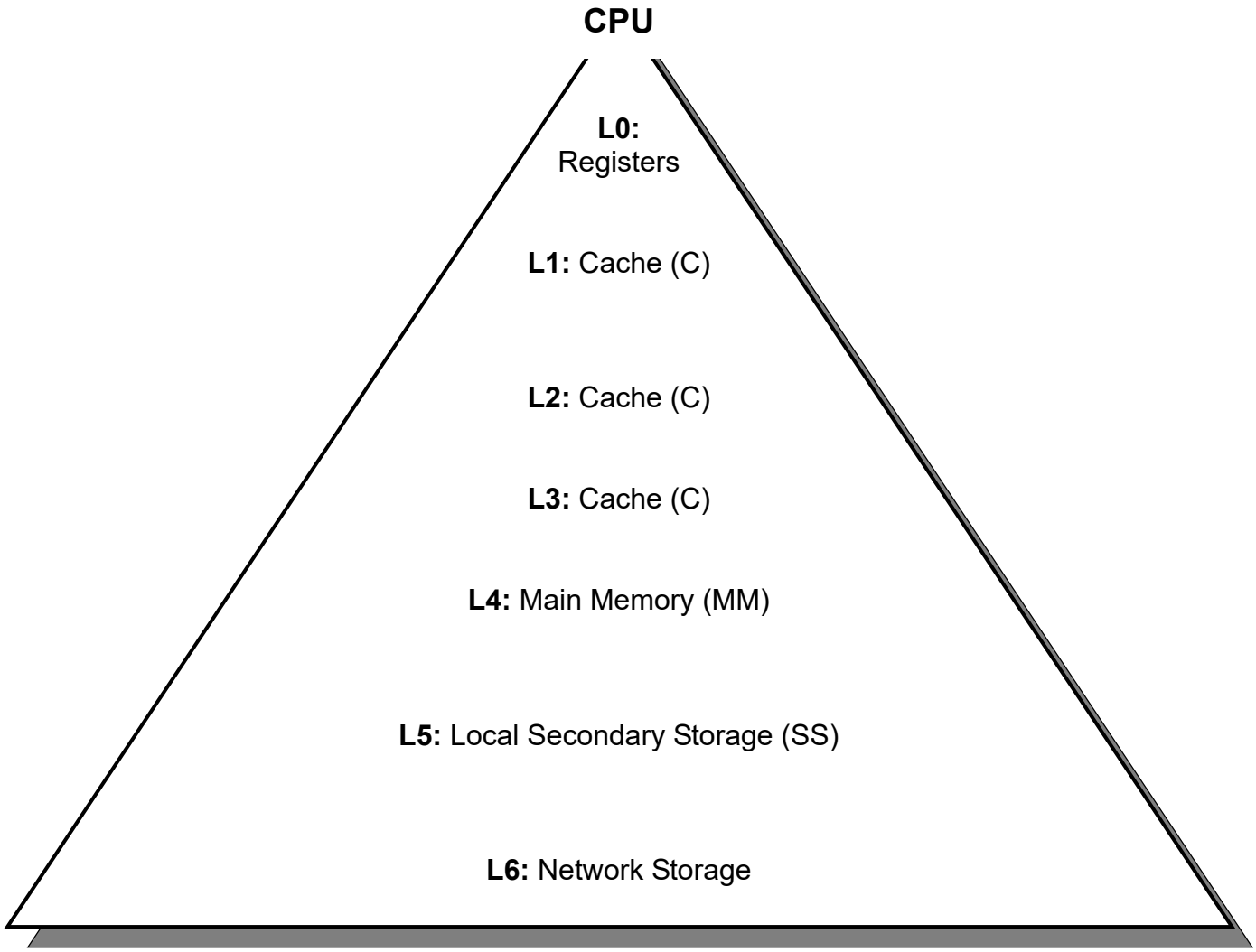
→ Why? can trash the internal structures

Do check if your memory intensive program has run out of heap memory!

→ How? always check return value of malloc

Memory Hierarchy

❋ *The memory hierarchy*



Cache

Memory Units

word: size used by transfer between

block: size used by transfer between

page: size used by transfer between

Memory Transfer Time

cpu cycles:

latency: