CS 354 - Machine Organization & Programming
Tuesday Oct 10, and Thurs Oct 12, 2024

Project p3: Released DUE on or before Friday Mar 8

Activities A06 available

Homework 3: DUE on or before Monday Mar 4

Exam 1: Scores posted by Thursday

Learning Objectives

- describe design choices 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
- shift bits and mask bits get size and status values from size_status integer
- choose an available free block based on placement policy, FF, NF, BF
- test implementation of shared object, heap
- describe the effect of various allocator design choices
- describe and explain the C/IA-32 memory hierarchy
- use make and Makefile to build a so object file, and run tests to show correctness

This Week

Placement Policies Free Block - Too Large/Too Small Coalescing Free Blocks Free Block Footers (ready for p3 now)	Explicit Free List (not in p3) Explicit Free List Improvements Heap Caveats (reminders) Memory Hierarchy Exam 1 Results - bring e1_error_report
Next Week: Locality and Designing Caches B&O 6.4.2	

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

- complete Week A06 activity as soon as possible
- review source code functions before lecture this week
- write code to compute the correct heap block size
- use GDB to examine "print" size from size_status, and status from size_status field
- implement **balloc** 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 placement policy are correct.
- complete testing and debugging and complete final submission (partC&D tests pass)

Free Block - Too Large/Too Small

"split" "coalesce", "ask", "fail"

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

- use entire block

-ve mem util: more internal fragmentation — waste of space

+ve thruput: fast and simple code

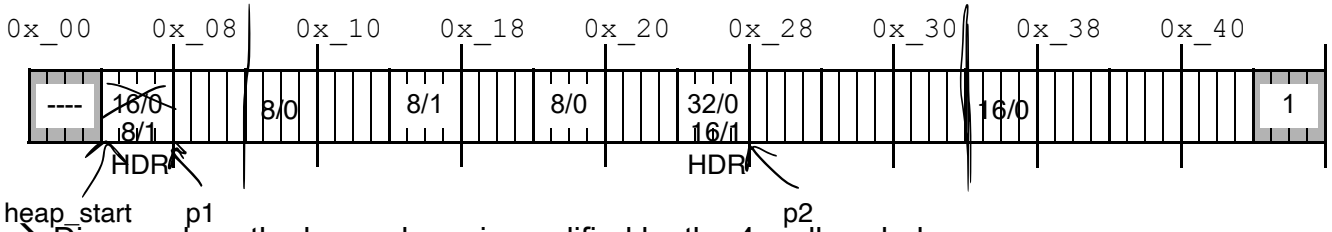
- split into 2 blocks - 1st block is allocated block, 2nd block remains as free block

+ve mem util: less internal fragmentation — less likely to have large wasted padding space

-ve thruput: slower to search, more blocks to search

*external fragmentation is caused by user, cannot be controlled here, so improve internal fragmentation

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?

		payload ptr	new free block
1) `p1 = malloc(sizeof(char));`	(1 + 4) + 3 = 8	0x08	0x0C
2) `p2 = malloc(11 * sizeof(char));`	(11 + 4) + 1 = 16	0x28	0x34
3) `p3 = malloc(2 * sizeof(int));`	(8 + 4) + 4 = 16	0x38	nil
4) `p4 = malloc(5 * sizeof(int));`	(20 + 4) + 0 = 24	nil	nil

reach end mark alloc fails

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

1st. coalesce adjacent free blocks

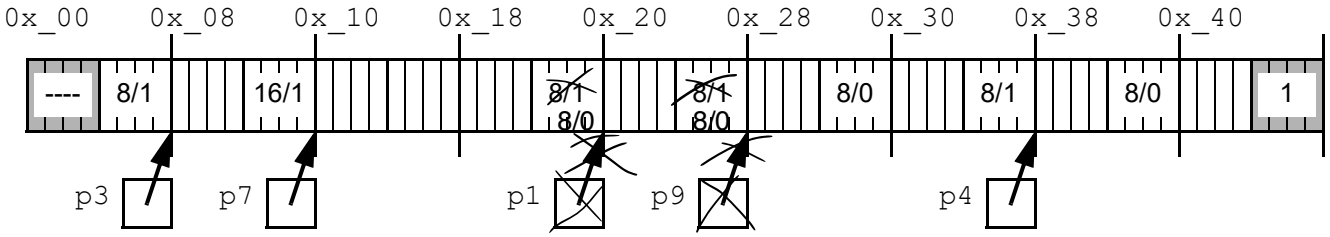
→ Can allocated blocks be moved out of the way to create larger free areas? NO!!!!!

2nd. ask kernel for more (not in p3)

3rd. return NULL (alloc fail)

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));
```

Text

(16 + 4) + 4 = 24 — ALLOC FAIL

Problem? FALSE FRAGMENTATION external fragmentation — user error ? in choosing 8 blocks ? can argue both ways

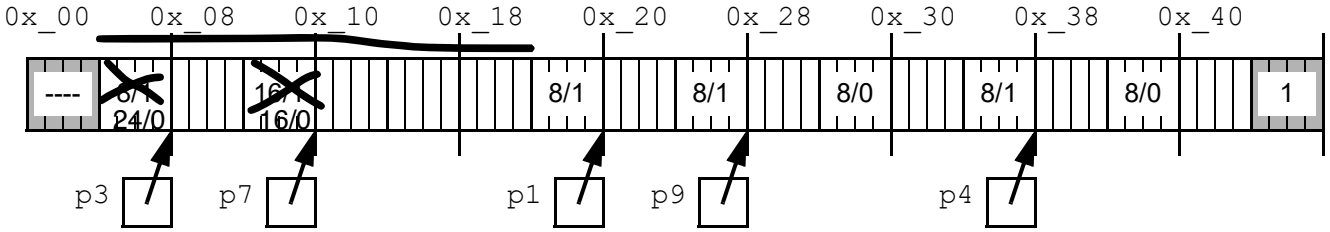
when there is a large enough contiguous free space but it is divided into blocks that are too small

Solution? COALESCE ADJACENT FREE BLOCKS

immediate: coalesce after freeing a block — check next and previous, coalesce if possible

delayed: do only if needed to satisfy request — in alloc function

Run 6: Heap Freeing with Immediate Coalescing



→ Given the heap above, what is the size in bytes of the freed heap block? 16 bytes, no coalesce

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

→ Given a pointer to a payload, how do you find its block header? ptr - 4 bytes (if ptr is void* or char*)

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

ptr - 4 + curr_block_size

❋ *Use type casting* to set scale factor

(void*) == scale factor 1
(char*) == scale factor 1
(blockHeader*) == scale factor sizeof(blockHeader*)

→ 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;
```

24 bytes — coalesce with next

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

32 bytes — coalesce with prev

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

((void*) ptr - 4) - prev_block_size)

Free Block Footers

❋ *The last word of each free block* is a FTR containing free block size

→ Why don't allocated blocks need footers? NOT "FREE", no coalescing allocated block

→ If only free blocks have footers, how do we know if previous block will have a footer?

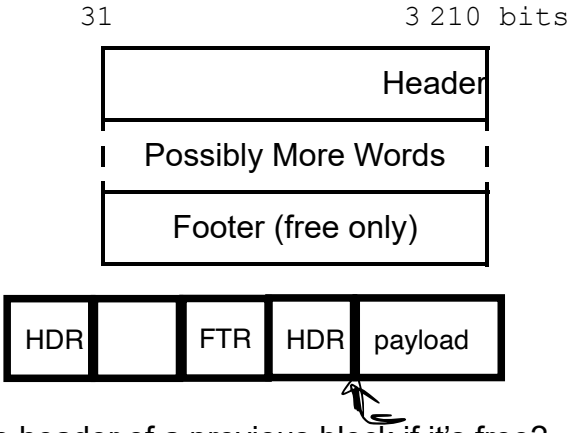
❋ *Free and allocated block headers*

also encode p-bit
if prev is allocated, then it is 1,
if prev is free, it is 0

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 + 0 + 1 = 9
2) 8 bytes in size and prev. block is allocated?
   8 + 2 + 1 = 11
3) 32 bytes in size and prev. block is allocated?
   32 + 2 + 1 = 35
4) 64 bytes in size and prev. block is free?
   64 + 0 + 1 = 65

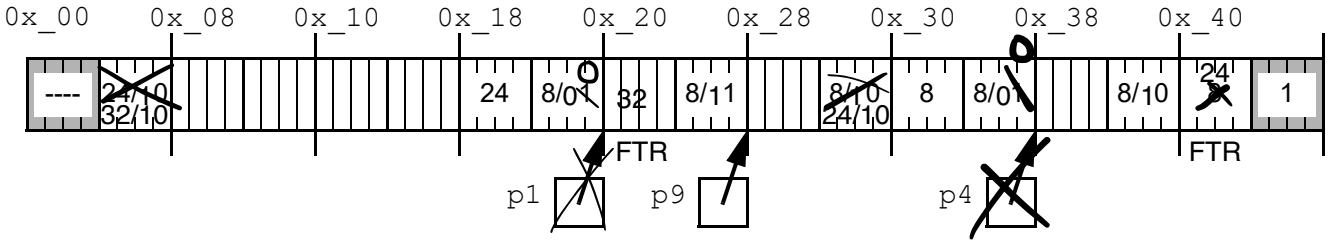


→ Given a pointer to a payload, how do you get to the header of a previous block if it's free?

1. ptr - 4 (get to HDR)
2. check p-bit
3. if p_bit = 0, HDR - prev_block_size

prev_block_header = ptr - 4 - (prev_block_size)
prev_block_size = *(HDR - 4) = *(ptr - 4 - 4)

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;
```

32 bytes

❋ *Don't forget to update* the next block p-bit when needed

➢ Is coalescing done in a fixed number of steps (constant time) or is it dependent on the number of heap blocks (linear time)?

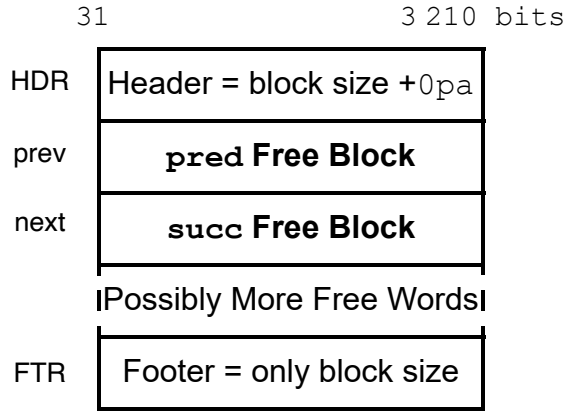
constant time

Explicit Free List

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

The EFL can be integrated into the heap by specifying a layout for free blocks

Explicit Free List Layout: Heap Free Block with Footer



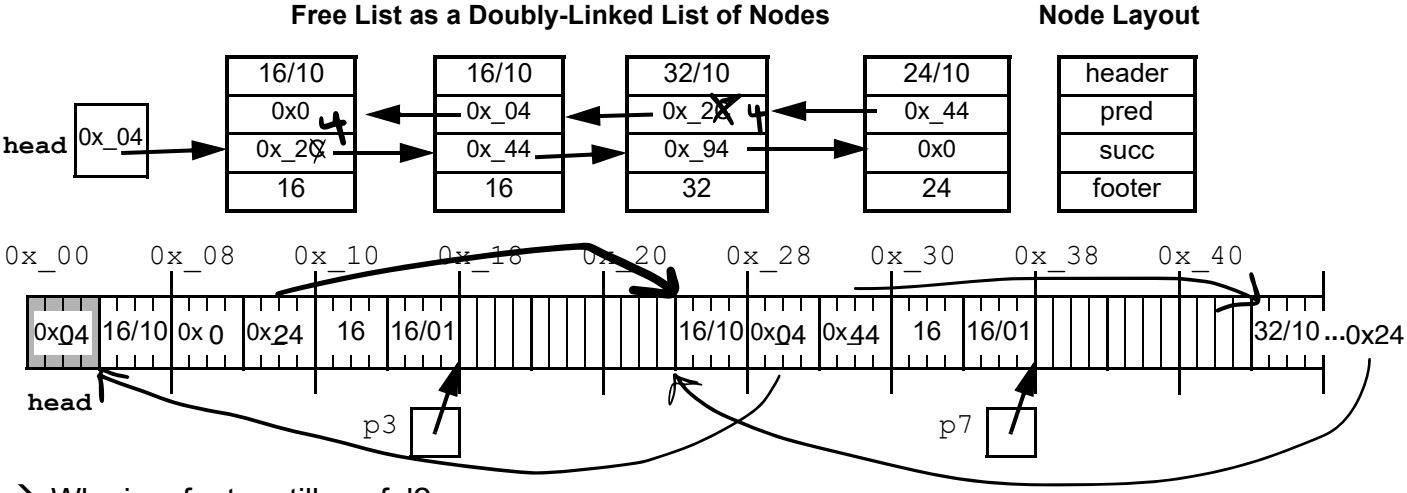
Free Block Links

pred addr of prev free block

succ addr of next free block

min size block is 16 bytes

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



→ Why is a footer still useful?

for faster coalescing

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

NO

Explicit Free List Improvements

Free List Ordering

address order : maintain list in order from low to high

+ve malloc with FF

-ve free slower but only linear O(n) and not O(n log n)

last-in order: place most recently freed block at end of the doubly linked list

-ve malloc with FF slower must go through most recetn

+ve free not trying to maintain the order — O(1)

Free List Segregation keep array of free list

a separate EFL for free block size

simple segregation: one EFL for each block size

structure

malloc

if free list is empty

free

problem

fitted segregation: one EFL for each size range — small, medium, large

+ve: memory utilisation - as good as best fit
+ve: throughput - search only part of heap

fitting use FF of appropriate EFL, if fails, go to next largest size

splitting put newly free block into appropriate EFL

coalescing put new coalesced free block into appropriate EFL

Heap Caveats

Consecutive heap allocations don't result in contiguous payloads!

→ Why?

Don't assume heap memory is initialized to 0!

Do free all heap memory that your program allocates!

→ Why are memory leaks bad?

→ Do memory leaks persist when a program ends?

Don't free heap memory more than once!

→ What is the best way to avoid this mistake?

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

→ What kind of error will result?

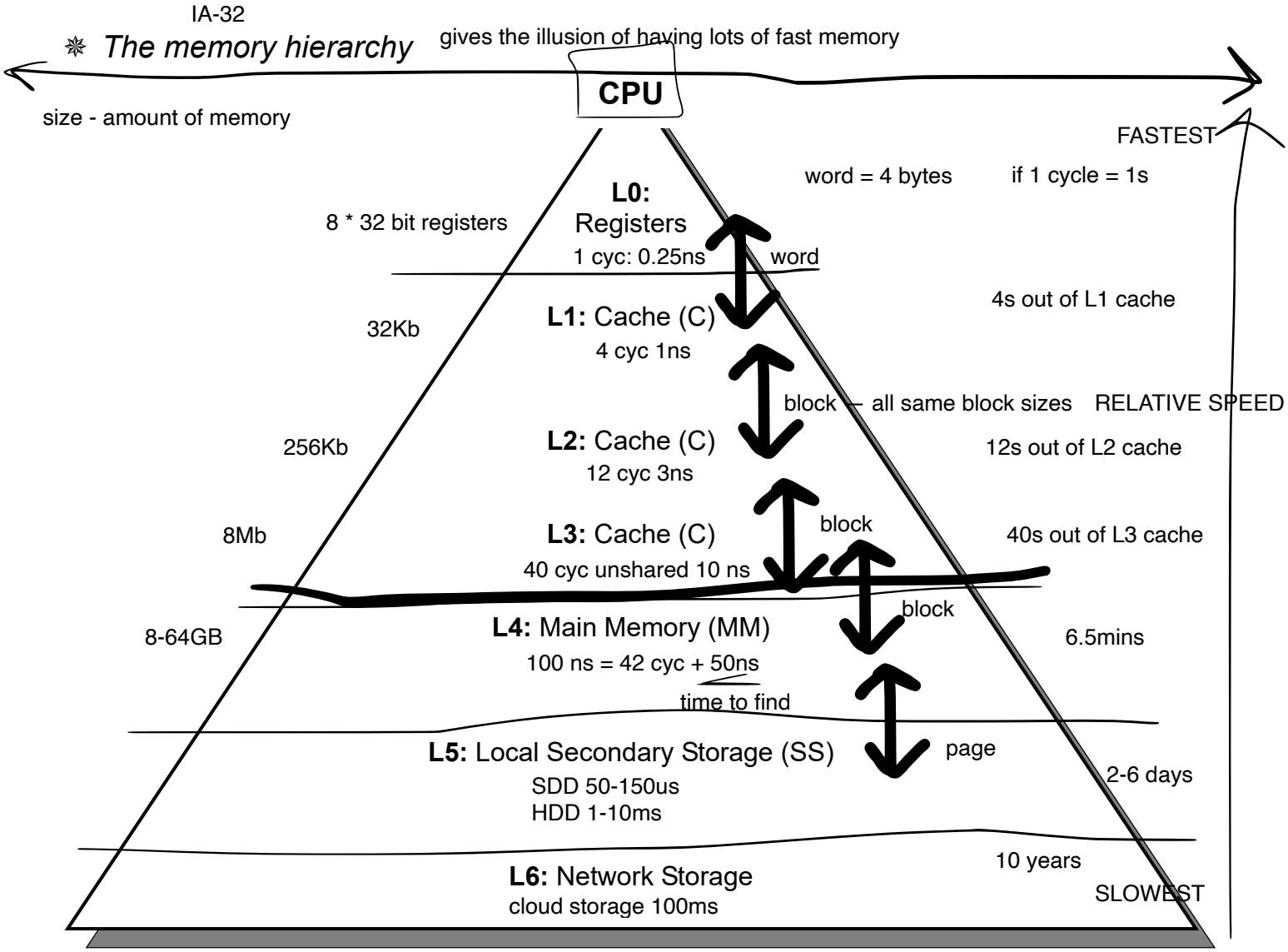
Don't change heap memory outside of your payload!

→ Why?

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

→ How?

Memory Hierarchy



Cache

is a smaller faster memory that acts as a staging area for data stored in a larger slower memory

Memory Units

word: size used by	CPU	transfer between	L1 and CPU	4 bytes
block: size used by	Cache Levels	transfer between	C levels and MM	32 bytes = 1 block unless otherwise defined
page: size used by	MM	transfer between	MM and SS	4096 bytes = 1 page = 4Kb

Memory Transfer Time

cpu cycles: used to measure time

latency: memory access time (delay)