**CS 354 - Machine Organization & Programming**

# Tuesday Feb 27th, and Thurs Feb 29th , 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

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

use entire block

(-) mem util: more internal fragmentation

(+) thruput: fast and simple code

split into 2 blocks 1st block allocated, 2nd block free

(+) mem util: less internal fragmentation

(-) thruput: slower to search since more blocks

**Run 4: Heap First-Fit Allocation with Splitting**

0x\_00 0x\_08 0x\_10 0x\_18 0x\_20 0x\_28 0x\_30 0x\_38 0x\_40



* 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?

* 1. p1 = malloc(sizeof(char));

1+4+3=8 //ptr 0x\_08 //New Free Block 0x\_0C

* 1. p2 = malloc(11 \* sizeof(char));

11+4+1 = 16 //ptr 0x\_28 //New Free Block 0x\_34

* 1. p3 = malloc(2 \* sizeof(int));

8+4+4=16 //ptr 0x\_38 //No Split

* 1. p4 = malloc(5 \* sizeof(int));

20+4 = 24 Alloc Fails

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

1st. Coalesce adjacent free blocks

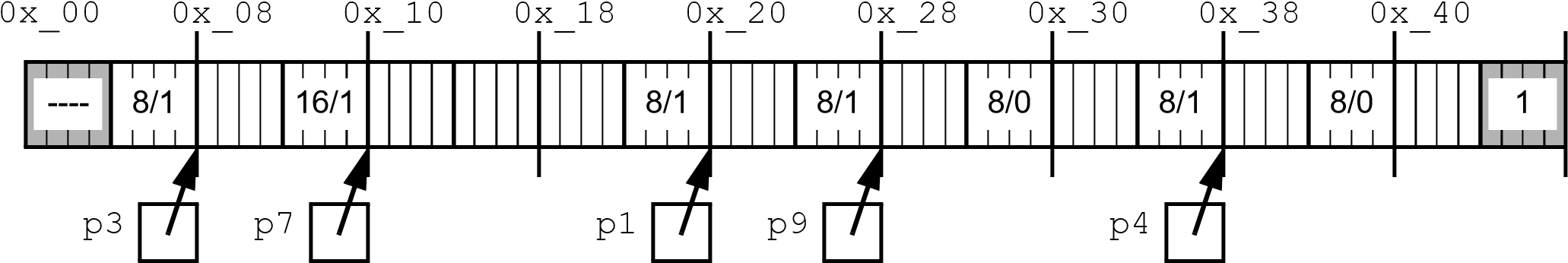
* Can allocated blocks be moved out of the way to create larger free areas? NO!

2nd. Ask kernel for more heap allocation

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)); 16+4+4=24 Alloc Fails

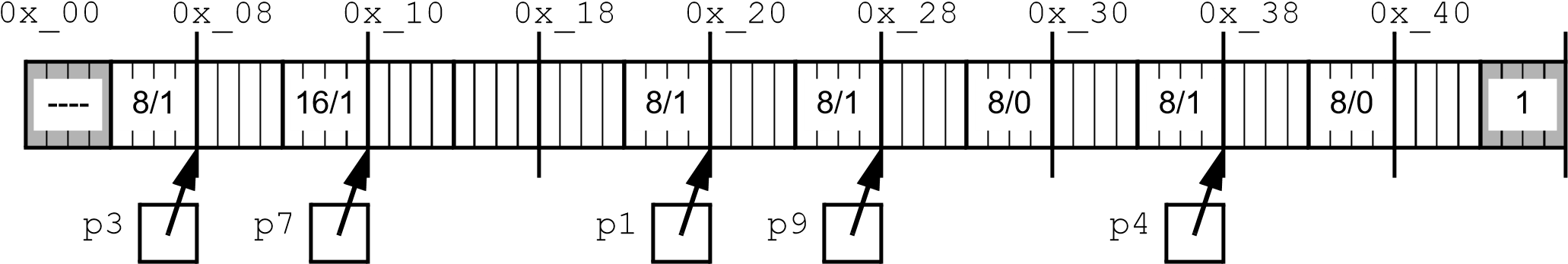
**Problem? False fragmentation – when there’s large enough contiguous free space but are divided by blocks too small.**

**Solution? Coalesce Adjecent free blocks**

*immediate*: coalesce after freeing a block – check next and previous

*delayed*: do only if needed to satisfy a 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?

1) free(p7); p7 = NULL; //16 bytes, no coal

* Given a pointer to a payload, how do you find its block header? ptr-4
* 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*

* Given the modified heap above, what is the size in bytes of the freed heap block when immediate coalescing is used?
  1. free(p3); p3 = NULL; //24 bytes, yes coal
  2. free(p1); p1 = NULL; //32 bytes, yes coal
* 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, coalescing alloced blocks
  + If only free blocks have footers, how do we know if previous block will have a footer?
* *Free and allocated block headers also encode a p bit, allocd = 1, free=0*

**Layout 2: Heap Block with Headers & Free Block Footers**

31 3 210 bits

|  |
| --- |
| Header |
|  |

* + What integer value will the header have for an allocated block that is:

1. 8 bytes in size and prev. block is free?

8(size) +0(prev)+1(allocated) = 9

|  |
| --- |
|  |
| Footer (free only) |

Possibly More Words

2) 8 bytes in size and prev. block is allocated?

8 + 2 + 1 = 11

* + 1. 32 bytes in size and prev. block is allocated?

32 + 2 + 1 = 35

* + 1. 64 bytes in size and prev. block is free?

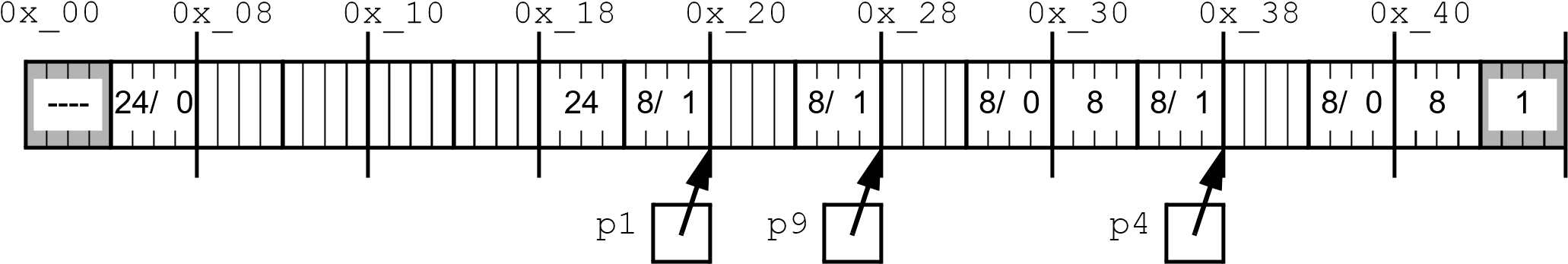
64 + 0 + 1 = 65

* + Given a pointer to a payload, how do you get to the header of a previous block if it’s free? 1. Ptr -4 2. Check p-bit 3. If p-bit = 0

prevBlockHdr = (ptr – 4)-(\*(ptr-8))

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

A close up of a graph

Description automatically generated

* + Given the heap above, what is the size in bytes of the freed heap block? 1) free(p1); p1 = NULL; //32

1. Set this block a bit to 0
2. Set the next block p bit to 0 (Make sure its not end mark)
3. Check next block a bit for coalescing
   1. It is 1 so we don’t coal
4. Check prev block a bit for coalescing
   1. If 0
      1. Subtract this block footer size (24) from hdr addr to get to last block
      2. Add this block size (24) to prev block size (8) and save to prev block size (32)
      3. Set footer to footer addr
         1. Footer addr = Prev block new size(32) + addr of prev block header – 4 (addr of prev block hdr + 28)
         2. Set prev block size(32) to this new footer addr
   * Given the modified heap above, what is the size in bytes of the freed heap block? 2) free(p4); p4 = NULL; //24
5. Set this block a bit to 0
6. Set next block p bit to 0 (Make sure its not end mark)
7. Check next block a bit for coalescing
   1. If 0
      1. Next block size gets added to this block size (8+8=16)
8. Check prev block a bit for coalescing
   1. If 0
      1. Subtract this block footer size (8) from hdr addr to get to last block
      2. Add this block size(Now its 16) to prev block size(p9 size = 8) and save to prev block size (24)
      3. Set footer to footer addr
         1. Footer addr = Prev block new size(now 24) + (addr of prev block header) – 4 (addr of prev block hdr + 20)
         2. Set prev block size(24) to this new footer addr

* *Don’t forget to update next block p bit when needed*

 Is coalescing done in a fixed number of steps (constant time) Yes or is it dependent on the number of heap blocks (linear time)? No

# Explicit Free List

 *An allocator using an explicit free list only keeps a list of free blocks. The Explicit free list 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***

31 3 210 bits

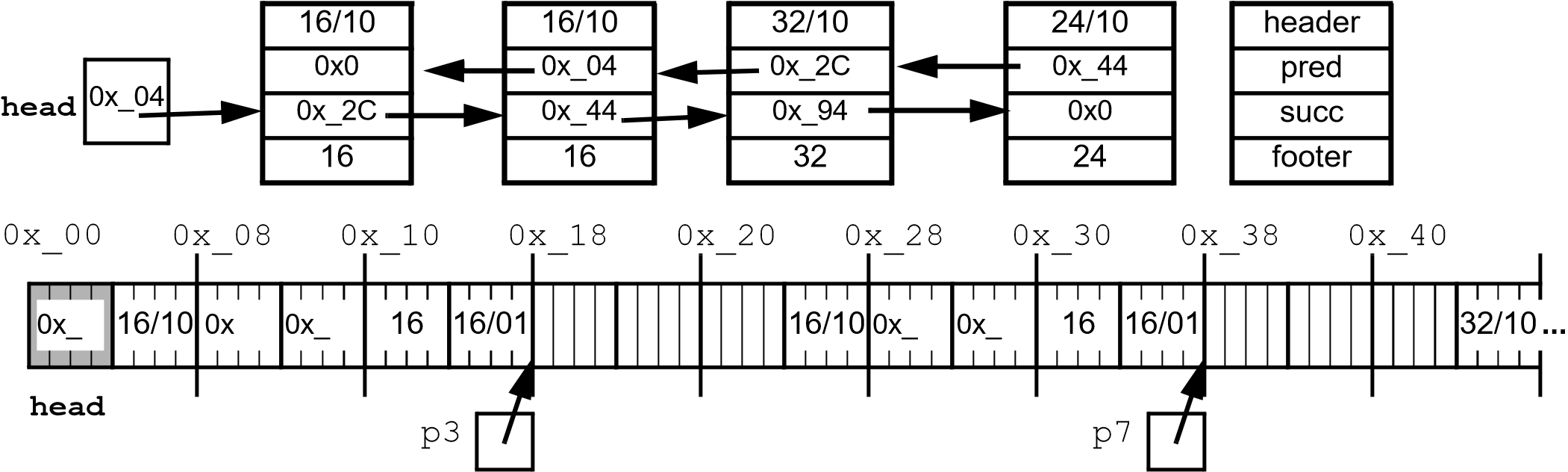
|  |
| --- |
| Header = block size +0pa |
| **pred Free Block** |
| **succ Free Block** |
|  |

Possibly More Free Words

|  |
| --- |
|  |
| Footer = only block size |

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

**Free List as a Doubly-Linked List of Nodes Node Layout**

A diagram of a diagram

Description automatically generated with medium confidence

* 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**

Maintain list in order from low to high addr

*address order* : malloc with FF

faster

Slower O(n)

free

Place most recent freed block at end of doubly linked list

*last-in order*: malloc with FF

faster, not trying to maintain order: O(1)

Slower, must go through most recent

free

Keep an array of free list – a separate E.F.L for free block size

**Free List Segregation**

One E.F.L for each block size

*simple segregation*:

structure malloc

if free list is empty free problem

One E.F.L for each size range: sm, med, large

Helps memory utilization – as good as best fit

Helps thruput – search only part of heap

*fitted segregation*:

fitting splitting coalescing

Put new coalesced free block into appropriate E.F.L

Put newly freed block in appropriate E.F.L

Use FF of appropriate E.F.L, if fails, look in next large size

# 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



*The memory hierarchy*

***Cache***

**CPU**

**L0:**

Registers

**L1:**

Cache (C)

**L2:**

Cache (C)

**L3:**

Cache (C)

**L4:**

Main Memory (MM)

**L5:**

Local Secondary Storage (SS)

**L6:**

Network Storage

**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*: