**CS 354 - Machine Organization & Programming**

**Thursday, February 22, 2018**

# Project p2 (6%): DUE at 10 pm THIS SUNDAY, February 25th

Note: It is better to submit a working program using indexing than a non-working program attempting to use address arithmetic.

**Project p3 (6%):** Assigned Tomorrow

**Homework hw3 (1.5%):** Assigned Tomorrow

# Last Time

C’s Heap Allocator (stdlib.h)

Posix brk (unistd.h)

Allocator Design

Simple View of Heap

Free Block Organization

Implicit Free List

# Today

Implicit Free List (from last time)

Placement Policies

Free Block - Too Much/Too Little

Coalescing Free Blocks

Footers

# Next Time

Heap Caveats

**Read:** B&O 9.9.11, 9.9.13

**Skim:** B&O 9.9.12

# Placement Policies

## Placement Policies

Are algo used to search for a free block to satisfy the request

Assume heap has been divided into varying sized blocks or ordered from smallest to largest

First Fit (FF): start search from beginning stop at first free block that’s big enough fail if the end mark reached

mem util: better likely to find an appropriate size block

thruput: slower or for large block requests

Next Fit (NF): start search from block most recently allocated stop at first free block that’s big enough

fail if starting block is reached

mem util: loop or since small block requests after large chooses an appropriately large block

thruput: faster

Best Fit (BF): start search from beginning for block that’s closed to reconstruction size stop at end mark or if exact size match found

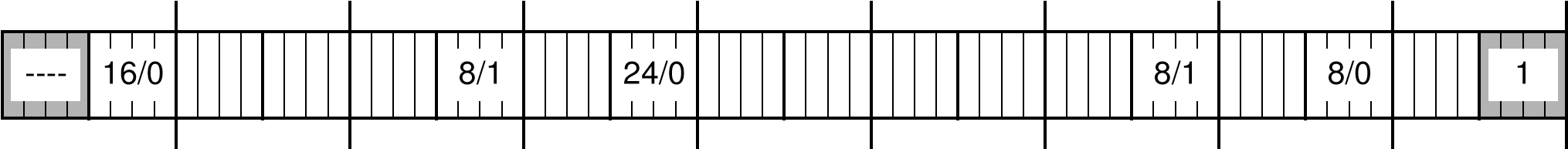
fail if no block found that’s big enough

mem util: best

thruput: slower since in worst case you need to loop though the entire

**Heap Allocation Run 3** using a Placement Policy:

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



Given the original heap above and the placement policy, what address is ptr assigned?

ptr = malloc(14 \* sizeof(char)); //FF? 0x20 BF? 0x20

ptr = malloc(3 \* sizeof(char)); //FF? 0x08 BF? 0x40

Given the original heap above and the address of block most recently allocated, what address is ptr assigned using NF?

|  |  |  |
| --- | --- | --- |
| ptr = malloc(2 \* sizeof(char)); | //0x\_14? 0x20 | 0x\_34? |
| 0x40  ptr = malloc(3 \* sizeof(int)); | //0x\_14? 0x20 | 0x\_34? |

# Free Block - Too Much/Too Little

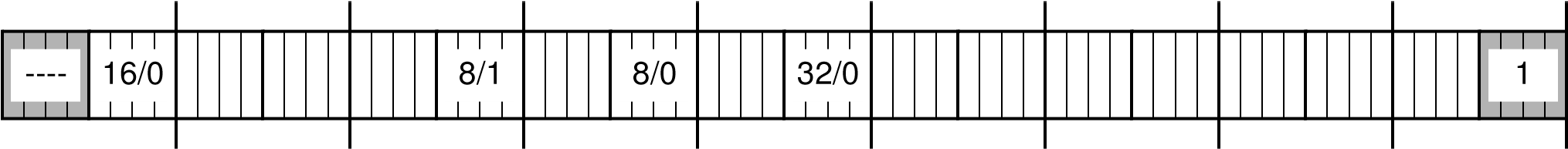
**What happens if free block chosen is bigger than the request?**

Use the entire block. Thruput: faster, simpler code. Mem UTIL: more internal fragmentation. Must divide up heap into varying size blocks before you can use it.

Split block into allocated and new free block. Thruput: slower 1. Code needs to splitting. 2. Increases number blocks slowing the free block search. MEM UTIL: better since it reduces internal fragmentation.

**Heap Allocation Run 4** using Splitting and using FF

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



Given the heap to be modified by the 4 mallocs below, what address is assigned to each pointer?

If there is a new free block, what is its address and size in bytes?

1. ptr1 = malloc(sizeof(char)); 0x.08 0x.\_0c 8
2. ptr2 = malloc(11 \* sizeof(char));0x.28 0x.34 16
3. ptr3 = malloc(2 \* sizeof(int)); 0x38 none
4. ptr4 = malloc(5 \* sizeof(int));

**What happens if no free block is large enough to satisfy the request?**

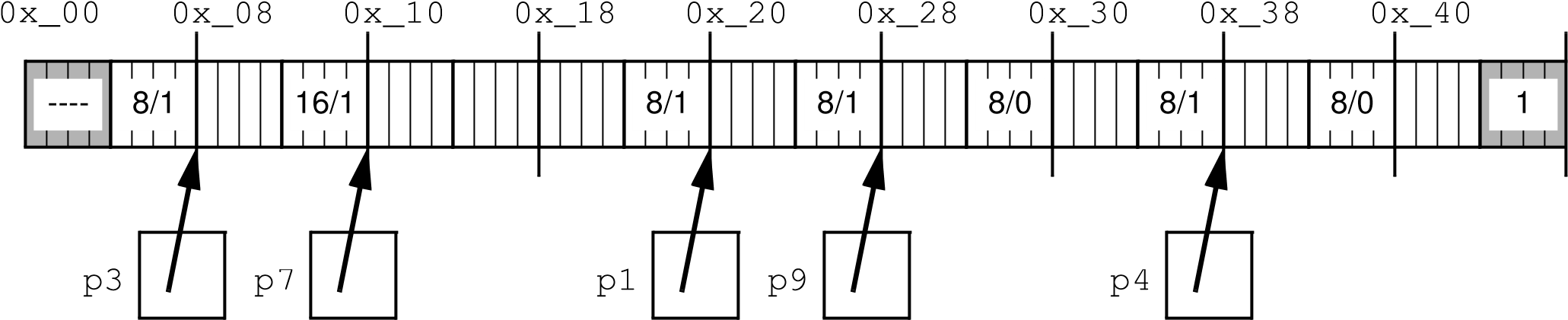
1st. coalesce neighboring free blocks

2nd. Ask kernel for more heap memory

3rd. return null

# Coalescing Free Blocks

## Heap Allocation Run 5



What’s the problem resulting from the following heap operations using FF?

1. free(p9); p9 = NULL;
2. free(p1); p1 = NULL;
3. p1 = malloc(4 \* sizeof(int)); fails due to

**Problem: false (external) fragmentation when a heap cannot satisfy a request despite having a large enough continuous free region that has been divided into smaller blocks**

**Solution: coalesce adjacent free block**

*immediate* coalesce freed block with its free neighbor

*delayed* coalesce only when request for a larger block is made.

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

1. free(p7); p7 = NULL; 16

Given a pointer to a payload, how do you find its block header? Ptr – 4 bytes

Given a pointer to a payload, how do you find the block header of the next block? Ptr-4 bytes+ size in bytes of freed block

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;
2. free(p1); p1 = NULL;

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

# Footers

## Heap Free Block Layout with Header and Footer

31 3|210 bits **Footer** (AKA Boundary Tag)

|  |
| --- |
| Header = block size |0pa |
|  |

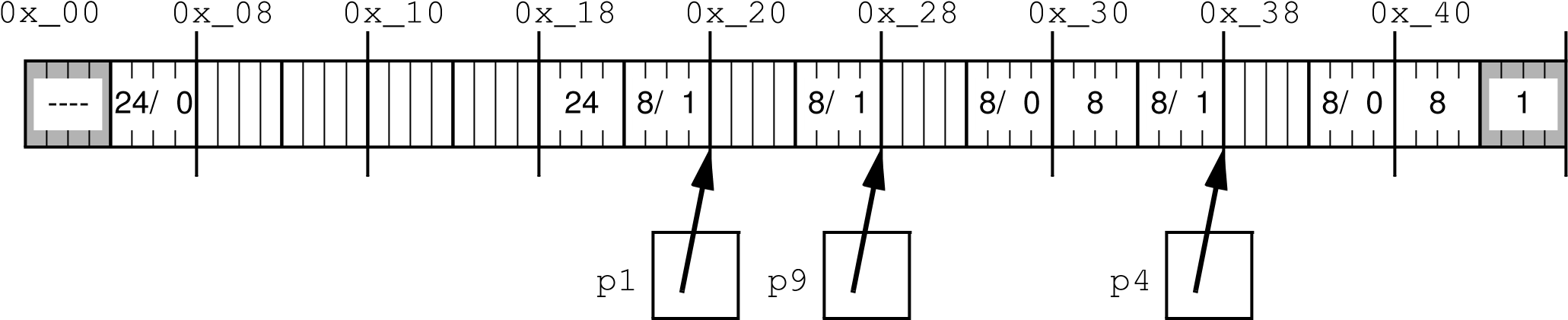
Possibly More Free Words

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

Why don’t allocated blocks need footers?

Given a pointer to a payload, how do you get to the header of a previous block that’s free?

**Heap Allocation Run 6** with given Free List using Immediate Coalescing and Free Block Footers



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

1. free(p1);

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

1. free(p4);

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