CS 354 - Machine Organization & Programming
Tuesday, October 8, 2019

Last Time

C's Heap Allocator (`stdlib.h`)
Posix `brk` (`unistd.h`)
Allocator Design

Today

Simple View of Heap
Free Block Organization
Implicit Free List
Placement Policies

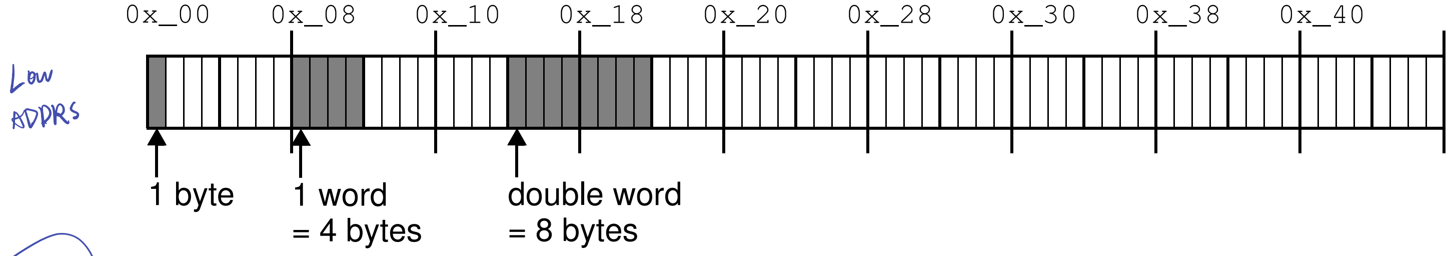
Exams Returned

Next Time

Splitting, Coalescing, Footers, Explicit Free Lists
Read: B&O 9.9.9 - 9.9.11, 9.9.13
Skim: B&O 9.9.12

Simple View of Heap

Linear Memory Layout



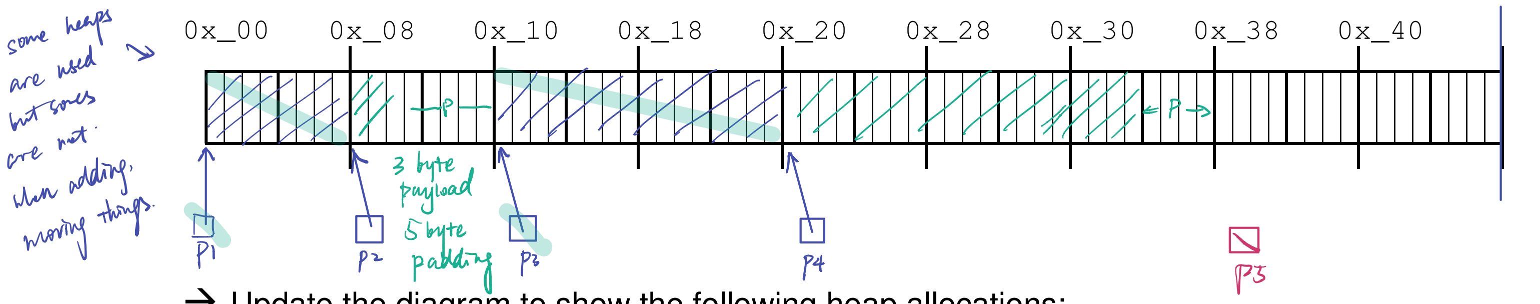
~32 bits

double word alignment: 1) block size must be a multiple of 8

2) Payload address must be a multiple of 8.

Heap Allocation Run 1 with a Simple View

assume the heap ends here



→ Update the diagram to show the following heap allocations:

```
1) p1 = malloc(2 * sizeof(int));
2) p2 = malloc(3 * sizeof(char)); 3+5=8      -P- means Padding
3) p3 = malloc(4 * sizeof(int));  16+0=16    no padding
4) p4 = malloc(5 * sizeof(int));  20+4=24
```

→ What happens with the following heap operations:

```
5) free(p1); p1 = NULL;
6) free(p3); p3 = NULL;
7) p5 = malloc(6 * sizeof(int)); Alloc Fail, and NULL is returned
```

External Fragmentation: when there is enough heap memory but it's divided into blocks that are too small to satisfy the request.

Internal Fragmentation: is when mem in a block is used for overhead (E.G. Padding) instead of payload

Free Block Organization

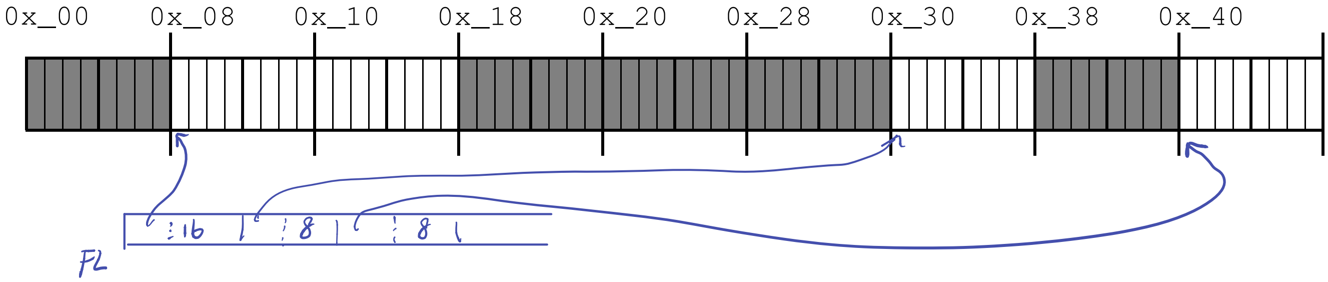
❋ Simple view of allocator has no way to determine the size and status of a block.

<u>size</u> number of Bytes in a block that is a payload + overhead.

<u>status</u> whether the block is allocated or false.

Explicit Free List

- allocator use a data structure (DS) containing just the free blocks.



code: only needs to track size for each block.

– space: potentially more mem for Data Structure.

+ time: a bit Faster, allocate is O(N) where n is the #free blocks.

Implicit Free List

✳
- Allocator uses the heap itself as Data Structure containing both allocated & free blocks.

code: must track both size & status for each block.

+ space: potentially less since just using heap blocks.

– time: allocate is O(N) where N is the #Allocated & Free Blocks.

Implicit Free List

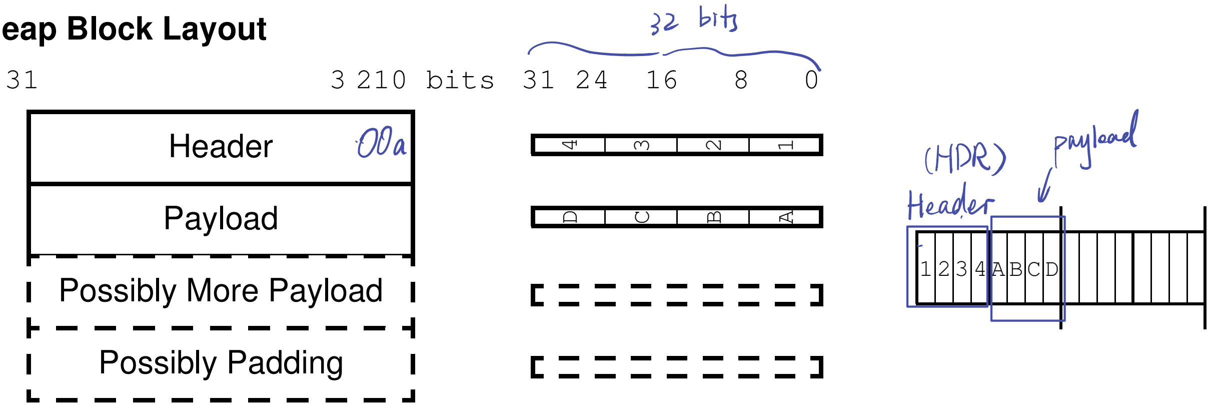
❋ *The first word of each block is* A header with blocks' size and status.

→ Since the block size is a multiple of 8, what value will the last three header bits always have?

8 : 01 | 000 ← zero
16 : 10 | 000
24 : 11 | 000

Status is stored in the a bit: a=1, allocated , a=0, freed.

Basic Heap Block Layout

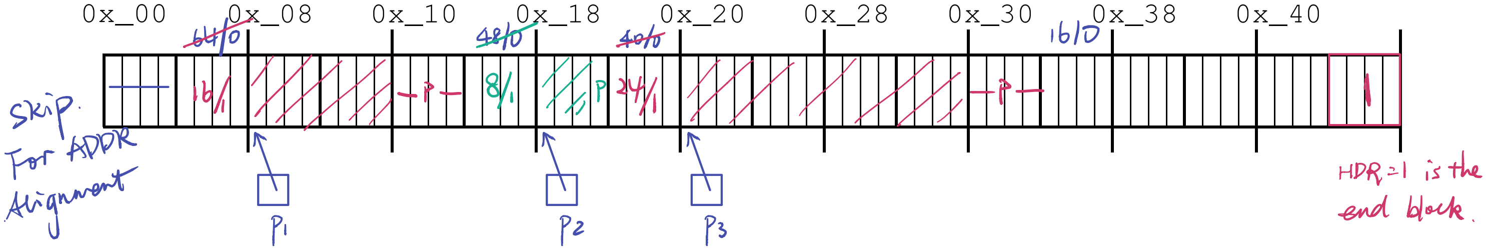


→ What integer value will the header have for a block that is

1) allocated and 8 bytes in size? 8 + 1 = 9. ≡ 8/1 ↑a
2) free and 32 bytes in size? 32 + 0 = 32. ≡ 32/0
3) allocated and 64 bytes in size? 64 + 1 = 65. ≡ 64/1

❋ *The header is an integer* that encodes both size and status.

Heap Allocation Run 2 with Block Headers



→ Update the diagram to show the following heap allocations:

```
1) p1 = malloc(2 * sizeof(int));
2) p2 = malloc(3 * sizeof(char));
3) p3 = malloc(4 * sizeof(int));
4) p4 = malloc(5 * sizeof(int));
```

1) 4 + 8 + 4 = 16
2) 4 + 3 + 1 = 8
3) 4 + 16 + 4 = 24.
4) 4 + 20 = 24 Alloc Fails

➢ Why does it make sense that Java doesn't allow primitives on the heap?

Placement Policies

What? *Placement Policies* are Algorithms used to search the heap for a free block to satisfy the request.

Assume the heap is pre-divided into various-sized free blocks ordered from smaller to larger.

- First Fit (FF): start from beginning of the heap.
  stop at first free block that is big enough.
  fail if the end mark is reached

+ mem util: Likely to choose the block close to the desired size

- thruput: request for large blocks must step through smaller blocks at front

- Next Fit (NF): start from the block most recently allocated
  stop at first free block that is big enough.
  fail if str block is reached

- mem util: might choose a block larger than needed

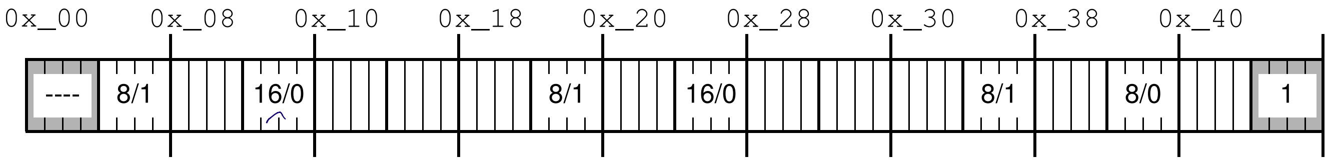
+ thruput: faster than FF since each request doesn't need to step through small blocks at front.

- Best Fit (BF): start from the beginning of the heap
  stop at end mark & choses the block closest to its size
  or stop early if exact size match found
  fail if no block that is big enough.

+ mem util: chooses the best block size

- thruput: slowest in generall since typically must seach entire heap

Heap Allocation Run 3 using a Placement Policy



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

`ptr = malloc(sizeof(int));` 8 `//FF?` 0x-10 `BF?` 0x-40

`ptr = malloc(10 * sizeof(char));` 16 `//FF?` 0x_10 `BF?` 0x_10

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

`ptr = malloc(sizeof(char));` 8 `//0x_04?` 0x-10 `0x_34?` 0x-40

`ptr = malloc(3 * sizeof(int));` 16 `//0x_1C?` 0x-28 `0x_34?` 0x-10

→ Given a pointer to the first block in the heap, how is the next block found?