

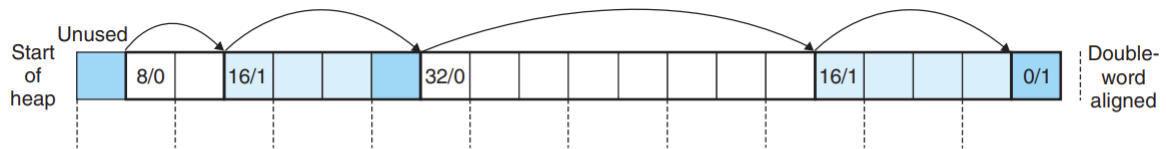
## 9.15

Determine the block sizes and header values that would result from the following sequence of `malloc` requests. Assumptions: (1) The allocator maintains double-word alignment and uses an implicit free list with the block format from Figure 9.35. (2) Block sizes are rounded up to the nearest multiple of 8 bytes.

Request	Block size (decimal bytes)	Block header (hex)
<code>malloc(4)</code>	_____	_____
<code>malloc(7)</code>	_____	_____
<code>malloc(19)</code>	_____	_____
<code>malloc(22)</code>	_____	_____

$$\text{malloc}(x) = x + 4$$

The additional 4 bits contain information to determine the block boundaries and whether a block is free or allocated.



**Figure 9.36** Organizing the heap with an implicit free list. Allocated blocks are shaded. Free blocks are unshaded. Headers are labeled with (size (bytes)/allocated bit).

Block header is the sum of the block size and the allocated bit. *Figure 9.36* shows the allocated bit for block.

For `malloc(4)`:

$$\text{malloc}(4) = 4 + 4 = 8$$

Since `malloc(4)` can fit within the first allocated block, the block size for `malloc(4)` will be 8.

Let block header be  $H_4$ :

$$H_4 = 8_{10} + 1_{10} = 9_{10} = 0x9_{16}$$

For `malloc(7)`:

$$\text{malloc}(7) = 7 + 4 = 11$$

Since `malloc(7)` can fit within the first two allocated blocks, the block size for `malloc(7)` will be 16.

Let block header be  $H_7$ :

$$H_7 = 16_{10} + 1_{10} = 17_{10} = 0x11_{16}$$

For `malloc(19)`:

$$\text{malloc}(19) = 19 + 4 = 23$$

Since `malloc(19)` can fit within the first two allocated blocks, the block size for `malloc(7)` will be 24.

Let block header be  $H_{19}$ :

$$H_{19} = 24_{10} + 1_{10} = 25_{10} = 0x19_{16}$$

For  $\text{malloc}(22)$ :

$$\text{malloc}(22) = 22 + 4 = 26$$

Since  $\text{malloc}(22)$  can fit within the first three allocated blocks, the block size for  $\text{malloc}(22)$  will be 32.

Let block header be  $H_{22}$ :

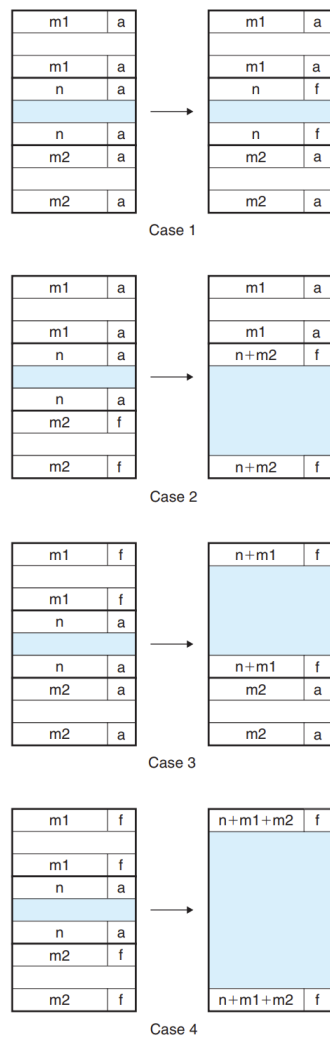
$$H_{22} = 32_{10} + 1_{10} = 33_{10} = 0x21_{16}$$

Request	Block size (decimal bytes)	Block header (hexadecimal)
$\text{malloc}(4)$	8	0x9
$\text{malloc}(7)$	16	0x11
$\text{malloc}(19)$	24	0x19
$\text{malloc}(22)$	32	0x21

## 9.16

Figure 9.40

**Coalescing with boundary tags.** Case 1: prev and next allocated. Case 2: prev allocated, next free. Case 3: prev free, next allocated. Case 4: next and prev free.



Conditions for determining minimum block size: next and previous values are free.

Alignment	Allocated block	Free block	Minimum block size (bytes)
Single word	Header and footer	Header and footer	$a$
Single word	Header, but no footer	Header and footer	$b$
Double word	Header and footer	Header and footer	$c$
Double word	Header, but no footer	Header and footer	$d$

a.  $(header + footer) \times 2 + (pred + succ) = (4 + 4) \times 2 + 4 = 20$

b.  $header + header + footer + (pred + succ) = 4 + 4 + 4 + 4 = 16$

c. 24

d. 16

$$(pred + succ) \times 2 + header + header + footer = 20$$

9.19

### Background information

**Buddy System:** apply segregated fits when the size class is a power of 2. When there is a heap containing  $2^m$  words, it is important to create a separate list with sizes:  $2^k$  where the block sizes will be approximated the nearest power.

In block allocation, find the first available block size  $2^j$  to store the block  $2^k$  in which  $j \geq k$  to ensure that the block size is either larger than or equal to the block needed to be stored. The block is split half each time an unsuitable memory location is found until the block is split small enough to be allocated to an available block.

Recursively split the block into half until the block can be allocated to an available block.

**Advantage:** Fast at searching and coalescing.

**Disadvantage:** Internal fragmentation: buddy system may allocate an available block with much larger memory than the actual block.

**First fit:** searches through the list sequentially and determines the first available block for allocation.

- **Advantage:** There are plenty of free memory towards the end of the list.
- **Disadvantage:** There are fragments of small memory blocks at the start of the list.

**Next fit:** plus, the first fit functions, next fit relates the previous search results and continues where it ends.

- **Advantage:** Next fit is faster than first fit at allocation.
- **Disadvantage:** Next fit is less effective at utilizing memory than first fit.

Next fit runs faster but has worse memory utilization than first fit. Best fit has better memory utilization than first fit. From the previous statement on next fit, it is deducted that best fit runs slower than first fit.

1. A

**Reason:** The buddy system recursively splits the block into half. First split results in 50% of the memory.