Dynamic Memory Allocation (cont.) and Exceptions

CSCI 237: Computer Organization
34th Lecture, Wednesday, December 4, 2024

Kelly Shaw

Administrative Details

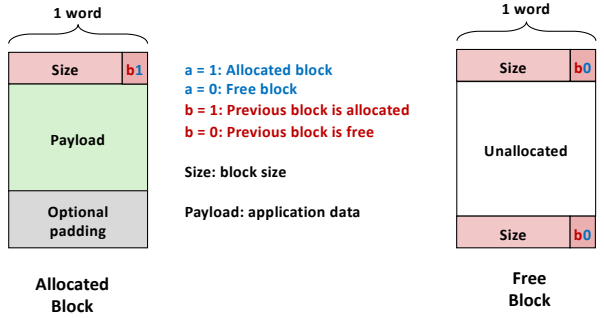
- Read CSAPP 9.9, 8.1-8.2
- Quiz open today at 2:30 until Friday at 2:30pm
- Lab #6 due Friday at 5pm
- Review session on Sun or Tues?
  - https://www.when2meet.com/?27868617-GeNrF
- Final exam
  - Wednesday, December 11, 9:30am – 11:30am (2 hours)
  - Clark Hall 105
- Colloquium talk on Friday at 2:35pm in Wege
  - "Systems research to address societal problems"
  - Aruna Balasubramanian, Stony Brook University

Today

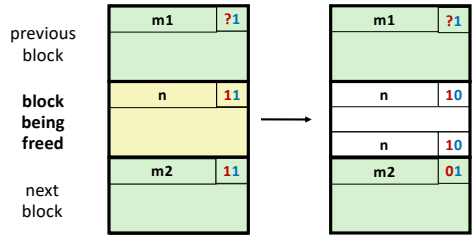
- Dynamic Memory Allocation (Ch 9.9)
  - Tracking Free Blocks
    - Implicit Lists
    - Explicit Lists
    - Segregated Lists
- Exceptional Control Flow
- Exceptions

Optimization: No Boundary Tag for Allocated Blocks

- Boundary tag is only needed for free blocks
- Insight: when sizes are multiples of 4 or more, have 2+ spare bits



No Boundary Tag for Allocated Blocks (Case 1)

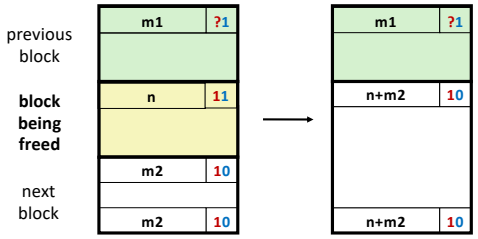


Header: Use 2 bits (always zero due to alignment):
(previous block allocated)<<1 | (current block allocated)

5

5

No Boundary Tag for Allocated Blocks (Case 2)

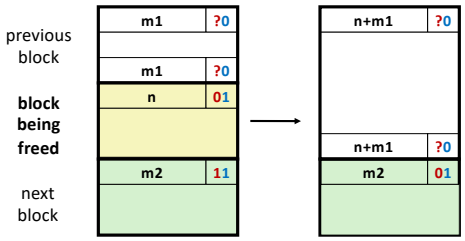


Header: Use 2 bits (always zero due to alignment):
(previous block allocated)<<1 | (current block allocated)

6

6

No Boundary Tag for Allocated Blocks (Case 3)

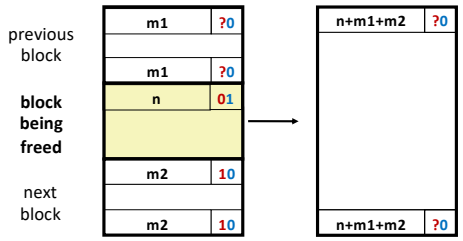


Header: Use 2 bits (always zero due to alignment):
(previous block allocated)<<1 | (current block allocated)

7

7

No Boundary Tag for Allocated Blocks (Case 4)



Header: Use 2 bits (always zero due to alignment):
(previous block allocated)<<1 | (current block allocated)

8

8

Summary of Key Allocator Policies

- Placement policy:
  - First-fit, next-fit, best-fit, etc.
  - Trades off lower throughput for less fragmentation
  - ***Interesting observation***: segregated free lists approximate a best fit placement policy without having to search entire free list
- Splitting policy:
  - When do we go ahead and split free blocks?
  - How much internal fragmentation are we willing to tolerate?
- Coalescing policy:
  - ***Immediate coalescing:*** coalesce each time **`free`** is called
  - ***Deferred coalescing:*** try to improve performance of **`free`** by deferring coalescing until needed. Examples:
    - Coalesce as you scan the free list for **`malloc`**
    - Coalesce when the amount of external fragmentation reaches some threshold

9

9

Implicit Lists: Summary

- Implementation: very simple
- Allocate cost:
  - linear time worst case
- Free cost:
  - constant time worst case
  - even with coalescing
- Memory usage:
  - will depend on placement policy
  - First-fit, next-fit or best-fit
- Not used in practice for `malloc/free` because of linear-time allocation
  - used in many special purpose applications
- However, the concepts of splitting and boundary tag coalescing are general to *all* allocators

10

10

Practice on Your Own

Consider a system where

- The memory is byte addressable.
- Memory accesses are to 1-byte words
- Virtual addresses are 16 bits wide.
- Physical addresses are 14 bits wide.
- The page size is 1024 bytes = 2^{10}
- The TLB is 4-way set associative with 16 total entries

Parameter	Value
VPN	0x
TLB Index	0x
TLB Tag	0x
TLB Hit? (Y/N)	
Page Fault? (Y/N)	
PPN	0x
PA	

For the following virtual addresses, fill out the table above

- 0x76BD
- 0x57EB

11

11

TLB			
Index	Tag	PPN	Valid
0	A	C	0
	4	9	1
	3	D	0
	9	3	0
1	2	0	1
	F	2	0
	7	4	0
	5	6	1
2	7	7	0
	2	9	0
	0	B	1
	6	F	1
3	6	A	0
	7	4	1
	C	1	0
	1	8	0

Page Table					
VPN	PPN	Valid	VPN	PPN	Valid
00	7	0	10	9	1
01	5	0	11	0	0
02	B	1	12	0	0
03	D	1	13	5	0
04	9	0	14	3	0
05	8	0	15	6	1
06	5	1	16	E	0
07	2	0	17	A	0
08	E	1	18	1	0
09	0	1	19	0	0
0A	F	0	1A	F	1
0B	D	0	1B	9	0
0C	C	0	1C	7	0
0D	4	0	1D	2	1
0E	3	0	1E	8	0
0F	1	0	1F	4	1

12

12

Practice on Your Own

- Assuming double-word alignment is used and the implicit free list format discussed in slides is used (i.e., 4 byte header). Block sizes are rounded up to nearest multiple of 8 bytes to maintain alignment.
- What would the block size be in bytes (including payload, header, and padding)? What would be stored in the block header (in hex)?
  - malloc(2)
  - malloc(9)
  - malloc(16)

13

13

Practice on Your Own

- When using implicit lists that only have a header, what is the smallest size of an allocated block (assuming double word alignment)?
- When using implicit lists that use headers and footers in all blocks, what is the smallest size of an allocated block (assuming double word alignment)?
- When using implicit lists that use headers in all blocks but footers only in free blocks, what is the smallest size of an allocated block (assuming double word alignment)?

14

14

Today

- Dynamic Memory Allocation (Ch 9.9)
  - Tracking Free Blocks
    - Implicit Lists
    - Explicit Lists
    - Segregated Lists
- Exceptional Control Flow
- Exceptions

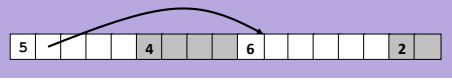
15

15

Keeping Track of Free Blocks

- Method 1: *Implicit list* using length—links all blocks

- Method 2: *Explicit free list* among the free blocks using pointers

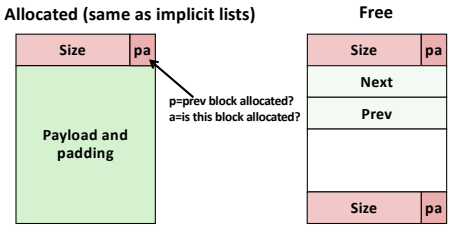


- Method 3: *Segregated free list*
  - Different free lists for different size classes
- Method 4: *Blocks sorted by size*
  - Can use a balanced tree (e.g. Red-Black tree) with pointers within each free block, and the length used as a key

16

16

Explicit Free Lists



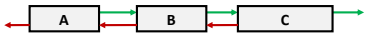
- Maintain list(s) of *free* blocks, not *all* blocks
  - The "next" free block could be physically anywhere (not necessarily adjacent block)
    - So we need to store forward/back pointers, not just sizes
  - Still need boundary tags (but only in free blocks!) for coalescing
  - Luckily we track only free blocks, so we can use payload area for pointers

17

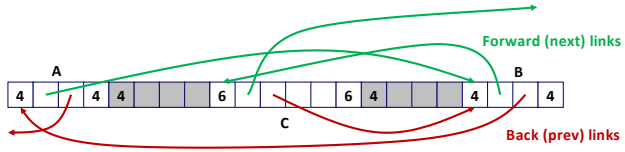
17

Explicit Free Lists

- Logically:



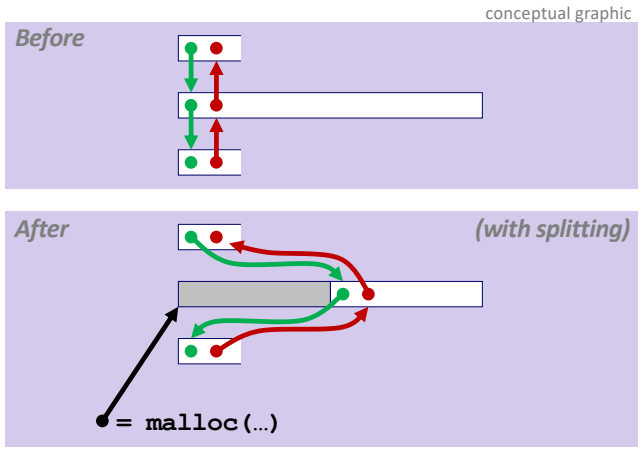
- Physically: blocks can be in any order



18

18

Allocating From Explicit Free Lists



19

19

Splitting Tasks

- Suppose we split a free block B into two: block A, and block F:
  - Allocated block A needs to be updated and removed from the list
  - Free block B needs to be updated so F is kept in the list
- `sizeAndTags` for both A and F needs to be adjusted
  - Shrink A to its allocation size
  - Create a `sizeAndTags` field in F to reflect new block's size (B – A)
- The next/previous pointers from A need to be added to F
- The next/previous blocks need their previous/next pointers to be updated to point to F
- Useful macros:
  - #define UNSCALED_POINTER_ADD(p,x) ((void*)((char*)(p) + (x)))
  - #define UNSCALED_POINTER_SUB(p,x) ((void*)((char*)(p) - (x)))

20

20

Freeing With Explicit Free Lists

- *Insertion policy:* Where in the free list do you put a newly freed block?

- **LIFO (last-in-first-out) policy**
  - Insert freed block at the beginning of the free list
  - ***Pro:*** simple and constant time
  - ***Con:*** studies suggest fragmentation is worse than address ordered

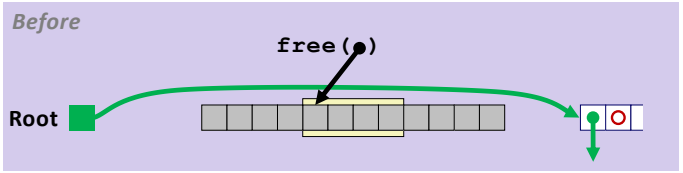
- **Address-ordered policy**
  - Insert freed blocks so that free list blocks are always in address order:
    addr(prev) < addr(curr) < addr(next)
  - ***Con:*** requires search
  - ***Pro:*** studies suggest fragmentation is lower than LIFO

21

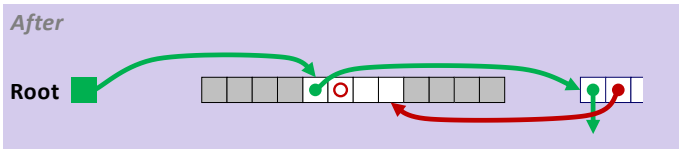
21

Freeing With a LIFO Policy (Case 1)

conceptual graphic



- Insert the freed block at the root of the list

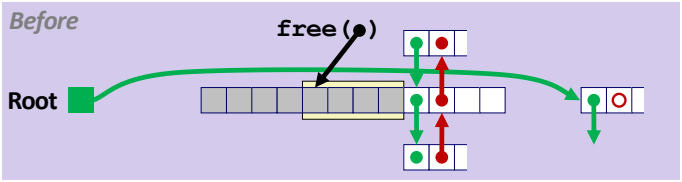


22

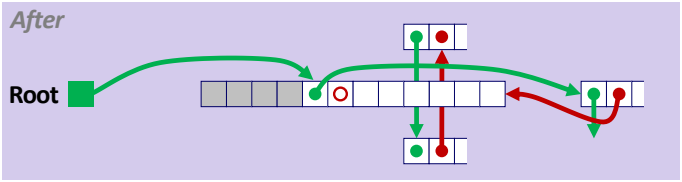
22

Freeing With a LIFO Policy (Case 2)

conceptual graphic



- Splice out successor block, coalesce both memory blocks and insert the new block at the root of the list

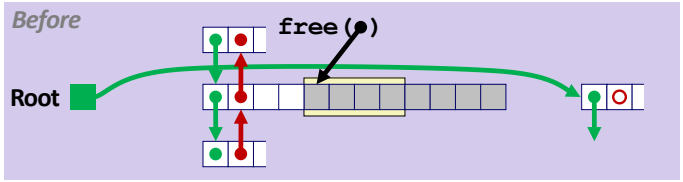


23

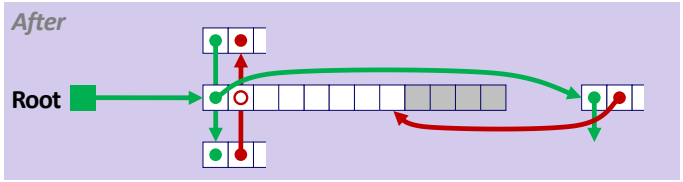
23

Freeing With a LIFO Policy (Case 3)

conceptual graphic



- Splice out predecessor block, coalesce both memory blocks, and insert the new block at the root of the list

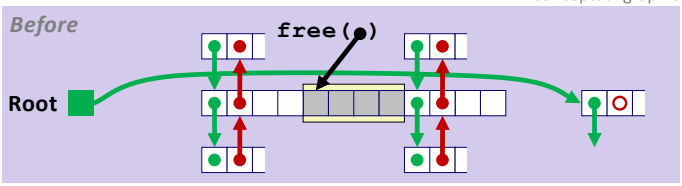


24

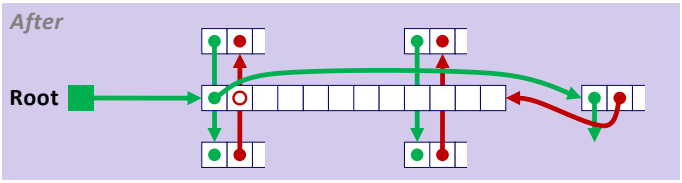
24

Freeing With a LIFO Policy (Case 4)

conceptual graphic



- Splice out predecessor and successor blocks, coalesce all 3 memory blocks and insert the new block at the root of the list



25

25

Explicit List Summary

- Comparison to implicit list:
  - Allocate is linear time in number of ***free*** blocks instead of ***all*** blocks
    - ***Much faster*** when most of the memory is full
  - Slightly more complicated allocate and free procedures since we need to splice blocks in and out of the list
  - Need some extra space for the links (2 extra words needed for each block)
    - Does this increase internal fragmentation? (maybe!)
- One of most common uses of linked lists is in conjunction with segregated free lists (up next)
  - Keep multiple linked lists of different size classes, or possibly for different types of objects

26

26

Today

- Dynamic Memory Allocation (Ch 9.9)
  - Tracking Free Blocks
    - Implicit Lists
    - Explicit Lists
    - Segregated Lists
- Exceptional Control Flow
- Exceptions

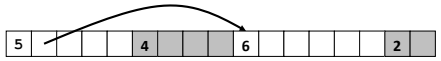
27

27

Keeping Track of Free Blocks

- Method 1: *Implicit list* using length—links all blocks

- Method 2: *Explicit list* among the free blocks using pointers



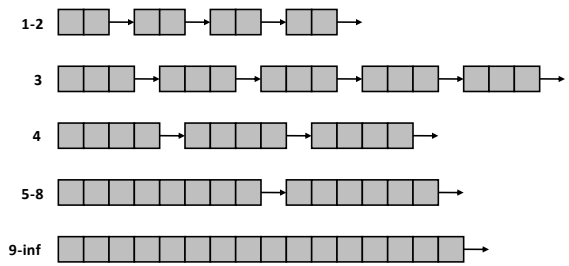
- Method 3: *Segregated free list*
  - Different (explicit) free lists for different size classes
- Method 4: *Blocks sorted by size*
  - Can use a balanced tree (e.g. Red-Black tree) with pointers within each free block, and the length used as a key

28

28

Segregated List (Seglist) Allocators

- Each *size class* of blocks has its own free list



- Often have separate classes for each small size
- For larger sizes: One class for each power-of-two size

29

29

Seglist Allocator

- Given an array of free lists, each one for some size class
- To allocate a block of size *n*:
  - Search appropriate free list for block of size *m* >= *n*
  - If an appropriate block is found:
    - Split block and place fragment on appropriate list (optional)
  - If no block is found, try next larger class
  - Repeat until block is found
- If no block is found:
  - Request additional heap memory from OS (using **`sbrk()`**)
  - Allocate block of *n* bytes from this new memory
  - Place remainder as a single free block in largest size class.

30

30

Seglist Allocator (cont.)

- To free a block:
  - Coalesce and place on appropriate list
- Advantages of seglist allocators
  - Higher throughput
  - Better memory utilization
    - First-fit search of segregated free list approximates a best-fit search of entire heap.
    - Extreme case: Giving each block its own size class is equivalent to best-fit.
- Seglist is used by GNU malloc provided in C std library

31

31

Today

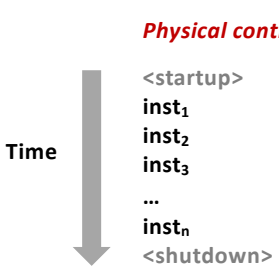
- Dynamic Memory Allocation (Ch 9.9)
  - Tracking Free Blocks
    - Implicit Lists
    - Explicit Lists
    - Segregated Lists
- Exceptional Control Flow
- Exceptions

32

32

Control Flow

- Processors do only one thing:
  - From startup to shutdown, a CPU simply reads and executes (interprets) a sequence of instructions, one at a time
  - This sequence is the CPU's *control flow* (or *flow of control*)



33

33

Altering the Control Flow

- Two mechanisms for changing control flow:
  - Jumps and branches
  - Call and return

  React to changes in ***program state***
- Insufficient for a useful system:

  Difficult to react to changes in *system state*
  - Data arrives from a disk or a network adapter
  - Instruction divides by zero
  - User hits Ctrl-C at the keyboard
  - System timer expires
- System needs mechanisms for "exceptional control flow"

34

34

Exceptional Control Flow

- Exists at all levels of a computer system
- Low level mechanisms
  - 1. **Exceptions**
    - Change in control flow in response to a system event (i.e., change in system state)
    - Implemented using combination of hardware and OS software
- Higher level mechanisms
  - 2. **Process context switch**
    - Implemented by OS software and hardware timer
  - 3. **Signals**
    - Implemented by OS software
  - 4. **Nonlocal jumps**: `setjmp()` and `longjmp()`
    - Implemented by C runtime library

35

35