CS 354 - Machine Organization & Programming
Tuesday Oct 8, and Thurs Oct 10, 2024

Project p3: Released DUE on or before Friday Oct 25

Activity A06 available

Homework 3: DUE on or before

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

Finish Implicit Free List, 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 **alloc** and submit progress to Canvas (pass partA tests)
- implement **free** 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?

-

  mem util:

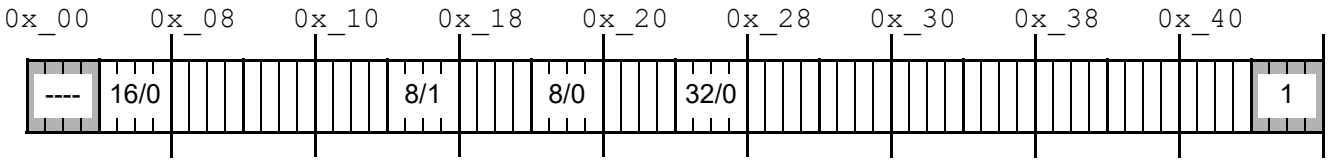
  thruput:

-

  mem util:

  thruput:

Run 4: Heap First-Fit Allocation with Splitting



→ 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));
2) p2 = malloc(11 * sizeof(char));
3) p3 = malloc(2 * sizeof(int));
4) p4 = malloc(5 * sizeof(int));
```

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

1st.

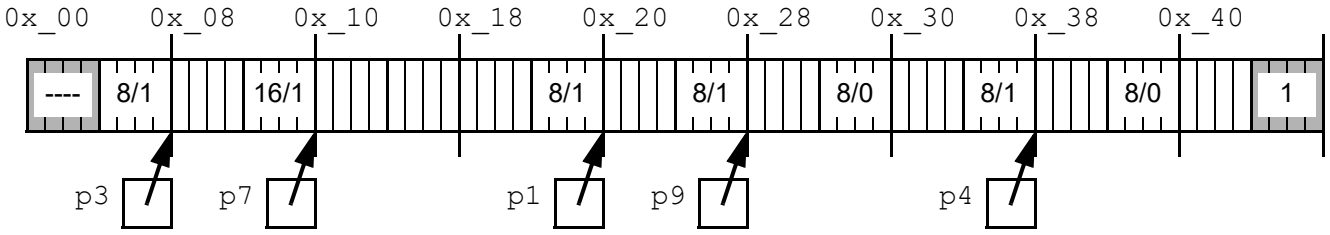
→ Can allocated blocks be moved out of the way to create larger free areas?

2nd.

3rd.

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));
```

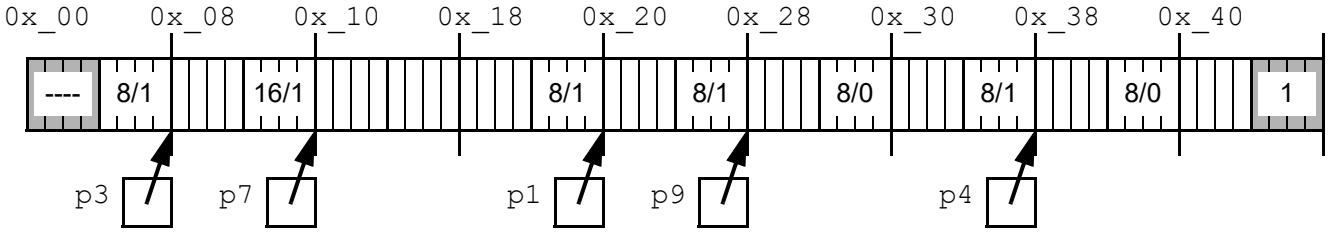
Problem?

Solution?

immediate:

delayed:

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;
```

→ Given a pointer to a payload, how do you find its block header?

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

❋ *Use type casting*

→ Given the modified heap above, what is the size in bytes of the freed heap block when immediate coalescing is used?

```
2) free(p3); p3 = NULL;
3) free(p1); p1 = NULL;
```

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

Free Block Footers

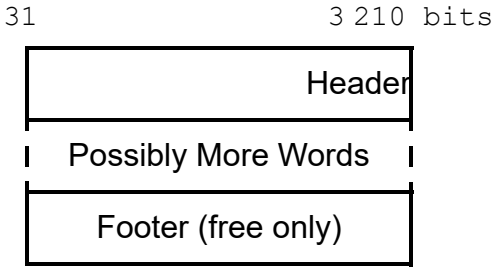
❋ *The last word of each free block*

→ Why don't allocated blocks need footers?

→ If only free blocks have footers, how do we know if previous block will have a footer?

❋ *Free and allocated block headers*

Layout 2: Heap Block with Headers & Free Block Footers



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

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

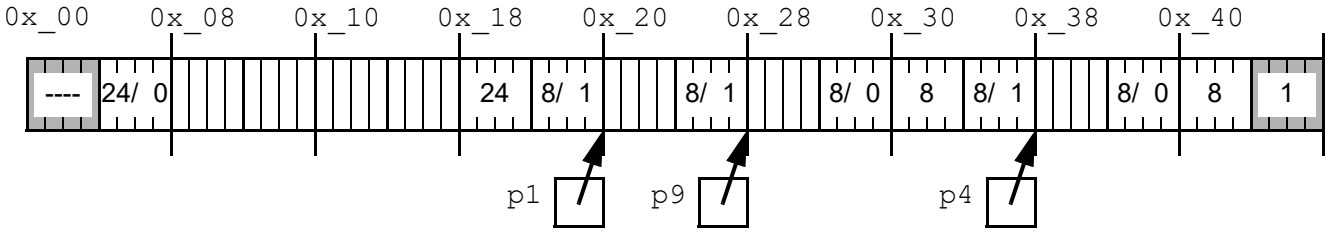
2) 8 bytes in size and prev. block is allocated?

3) 32 bytes in size and prev. block is allocated?

4) 64 bytes in size and prev. block is free?

→ Given a pointer to a payload, how do you get to the header of a previous block if it's free?

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



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

```
1) free(p1); p1 = NULL;
```

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

```
2) free(p4); p4 = NULL;
```

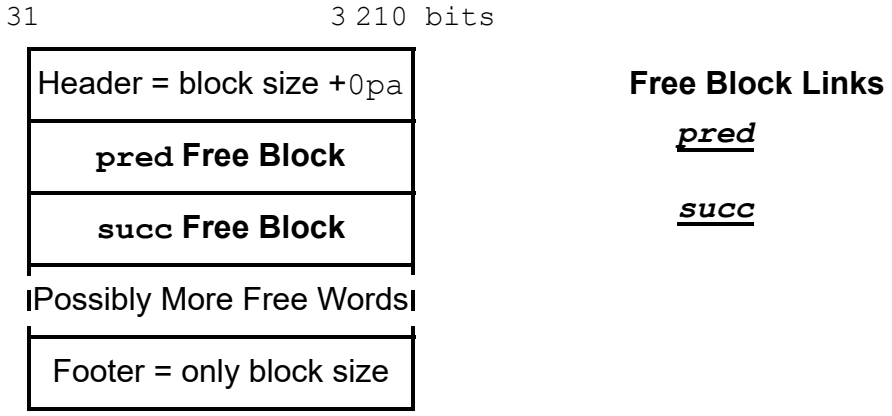
❋ *Don't forget to update*

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

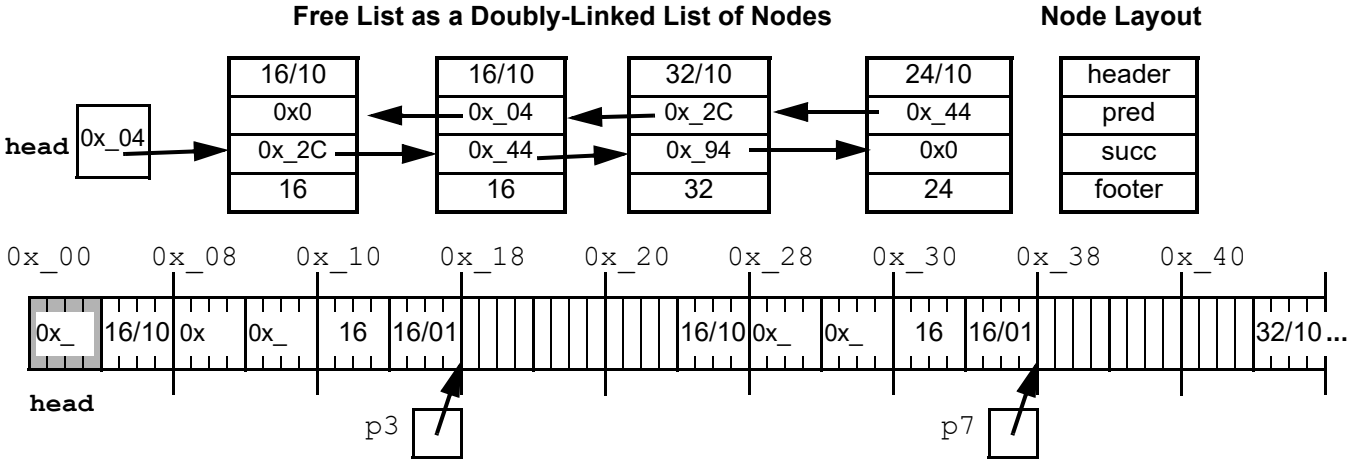
Explicit Free List

❋ *An allocator using an explicit free list*

Explicit Free List Layout: Heap Free Block with Footer



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



→ Why is a footer still useful?

→ Does the order of free blocks in the free list need to be the same order as they are found in the address space?

Explicit Free List Improvements

Free List Ordering

address order :

malloc with FF

free

last-in order:

malloc with FF

free

Free List Segregation

simple segregation:

structure

malloc

if free list is empty

free

problem

fitted segregation:

fitting

splitting

coalescing

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!

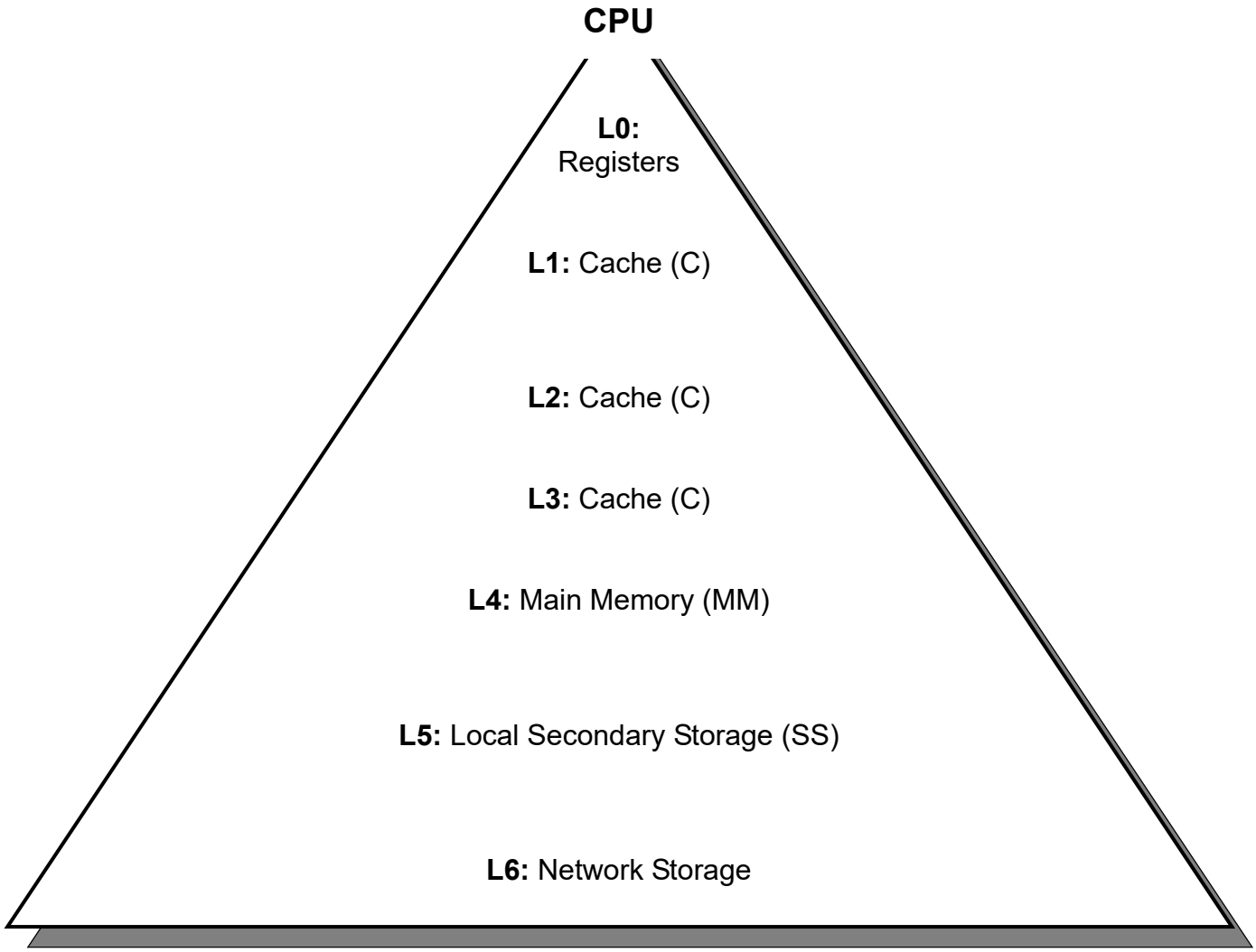
→ Why?

Do check if your memory intensive program has run out of heap memory!

→ How?

Memory Hierarchy

❋ *The memory hierarchy*



Cache

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: