Malloc Lab

Implementing your own malloc/free/realloc

libc malloc/free

void *malloc(size_t size)

- Allocate *size* bytes and return a pointer to the address allocated address
- my_type *my_obj = (my_type *)malloc(sizeof(my_type));

void free(void *ptr)

- Free the memory space pointed by *ptr*
- free(my_obj);

For more detail, https://linux.die.net/man/3/malloc

What is malloc?

- malloc is designed to provide a simple and portable way to allocate/deallocate a memory block of desired size

- Linux kernel itself also provides very limited dynamic memory management primitives (brk, sbrk)
  → They can only expand/shrink the end of data segment (just like a stack)

- libc, a user-level library, provides malloc implementation using those primitives

Challenges in malloc design

Execution speed

- Finding a free memory block
- Releasing a memory block

Memory space consumption

- Data structure overhead
- Internal fragmentation
- External fragmentation

→ Therefore, many different algorithms are there

Speed evaluation

Remember? Big-O notation

- Bubble sort: O(n^2)
- Merge sort: O(n log n)
- Linear search: O(n)
- Binary search: O(log n)
- DFS/BFS: O(#edges + #vertices)
- Hashtable with collision list: O(max_collision_length)
- Red-black tree search/insertion/deletion: O(log n)

Space evaluation

A N-byte request at least consumes N-byte

Data structure overhead:

- Ex) Doubly-linked list: next and prev pointer (two words)

Internal fragmentation

- Ex) 3-Byte is requested, but 4-Byte is returned (1-Byte wasted)

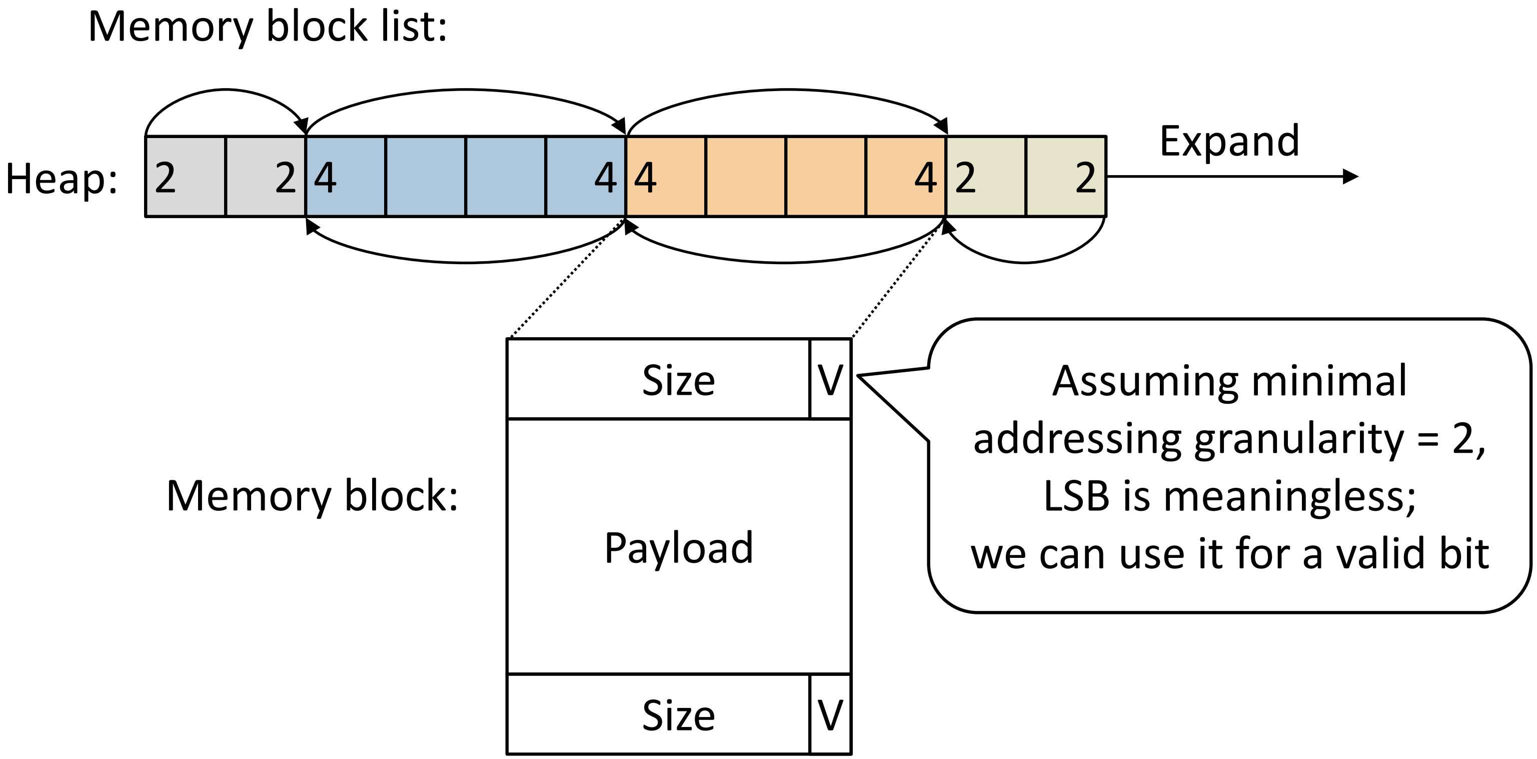
External fragmentation

- Ex) There is total 4-Byte of free memory, but increased the heap to satisfy 4-Byte malloc request (4-Byte wasted)

Example – Implicit free-list (IFL)

Data structure

Summary: maintain a doubly linked list of memory blocks



Memory block list:

Heap: 2 2 4 4 4 4 2 2

Expand

Memory block:

Size V

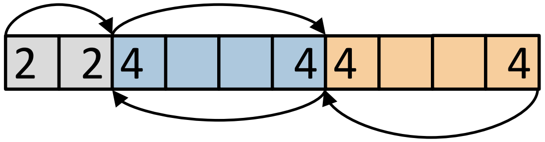
Payload

Size V

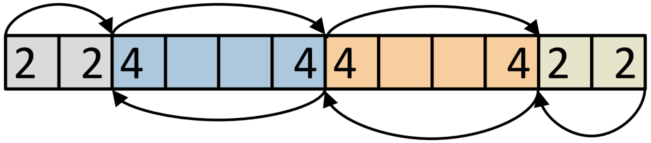
Assuming minimal addressing granularity = 2, LSB is meaningless; we can use it for a valid bit

Operations

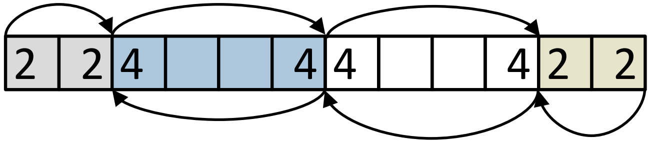
(1) Iintial



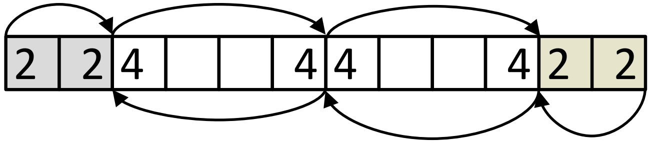
(2) malloc(2); expand



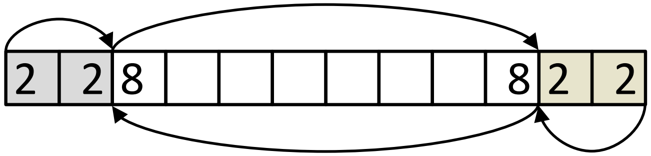
(3) free(a); invalidate



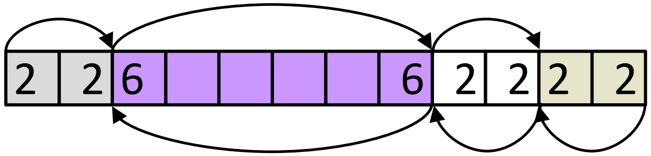
(4-1) free(b); invalidate



(4-2) merge (a.k.a. coalesce)



(5) malloc(6); split



(You may find the first-fit or the best-fit)

Implementation

malloc:

- Linearly search for an invalid memory block
- If nothing is found, expand the heap
- Mark the block as valid, and return the address

free:

- Mark the memory block as invalid
- Merge the adjacent blocks if they are also invalid
- ~~If the last block becomes invalid, shrink the heap~~
  - For this assignment, we assume the heap never shrinks

Performance

Speed

- ☹ malloc: **O(#memory_blocks); linear search**
- ☺ free: O(1); set invalid & coalesce

Space

- ☺ Overhead (2-word): next and prev displacement; valid bit may reuse next and prev's LSB
- ☺ Internal fragmentation: fine
- External fragmentation:

  ☹ First-fit → **severe** / ☺ Best-fit → fine

Assignment

- Write a dynamic memory allocator for C programs
  - Functions including mm_init, mm_malloc, mm_free, mm_realloc
- Assure them to work correctly and efficiently
- Hand in only one source code file (mm.c) and your report
- For more details, please refer to the README