CSED211: Lab 11

Malloc Lab

POSTECH

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

Speed evaluation

- Free memory block search
  - Linear search
  - Binary search
  - Etc

- Big-O notation
  - Linear search: O(n)
  - Binary search: O(log n) (may need sorting)
  - etc

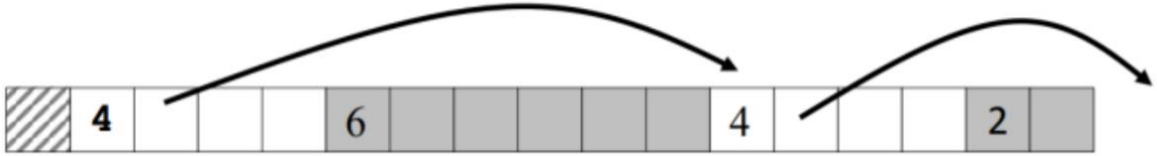
Space evaluation

- 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)

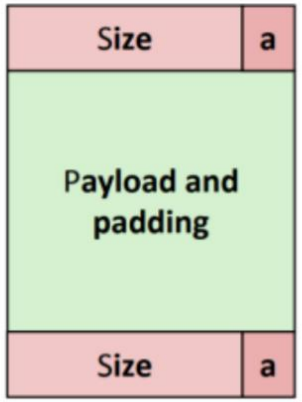
Free-list

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

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



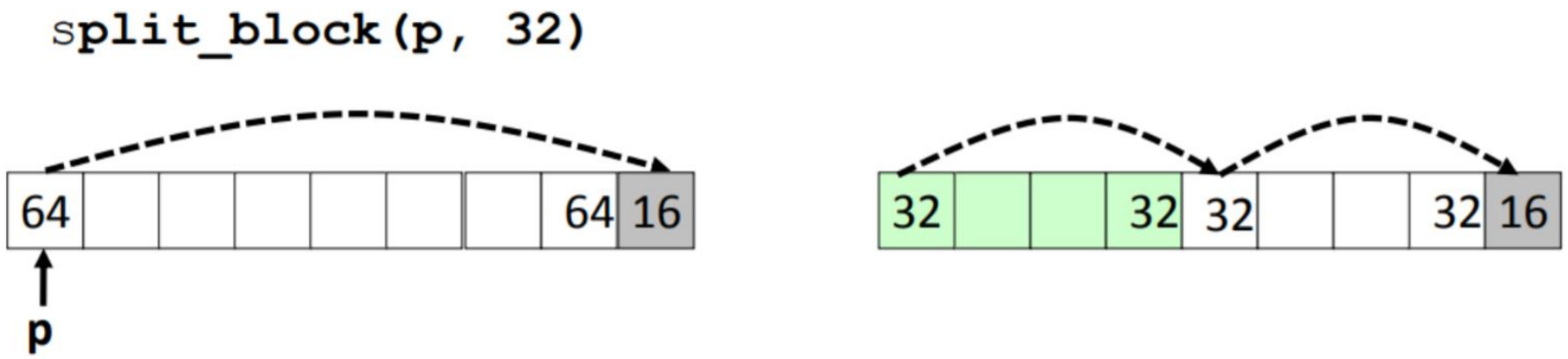
Allocated (as before)



Free

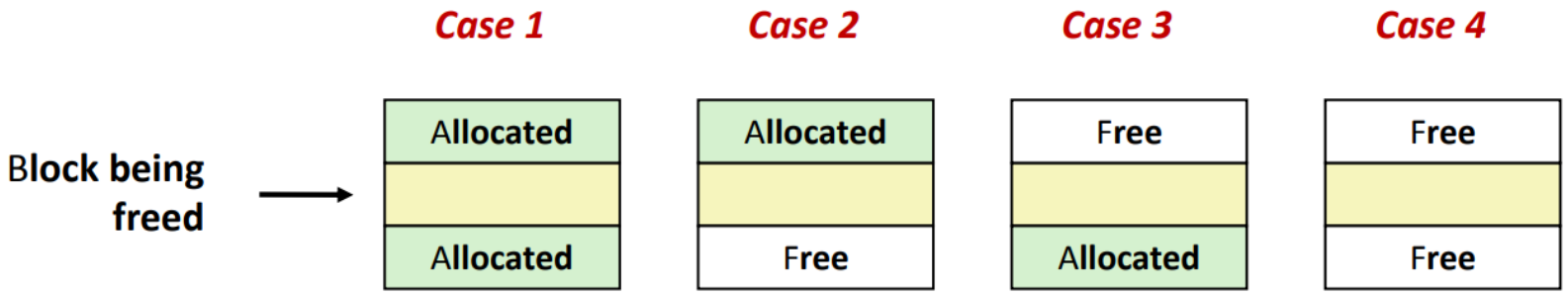
Implicit free-list (IFL)

- Allocation



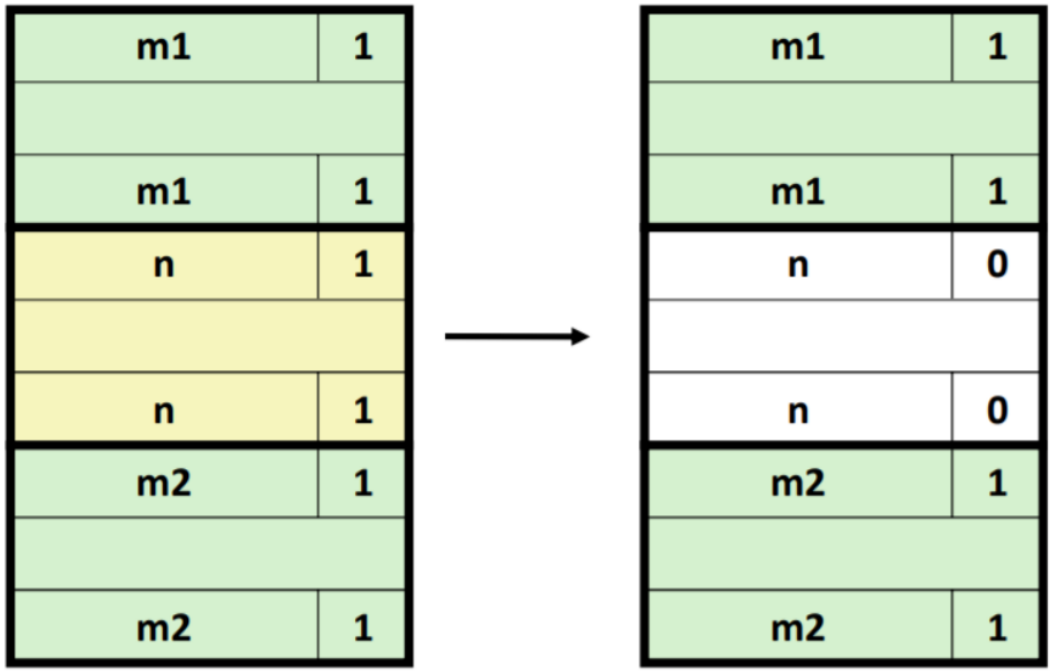
Implicit free-list (IFL)

- Free



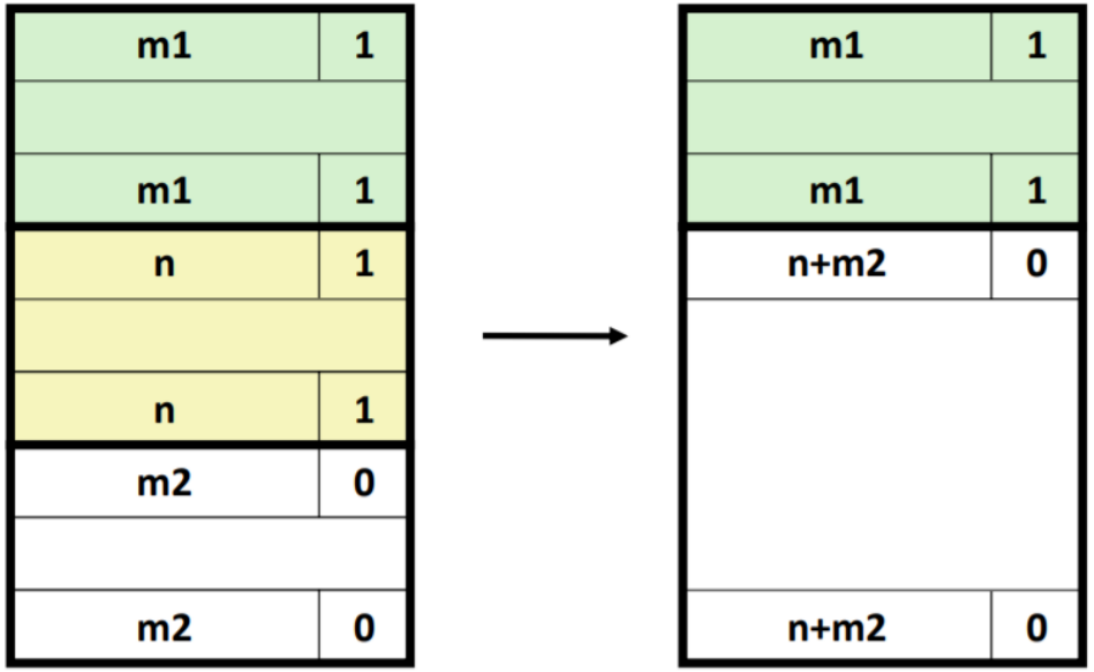
Implicit free-list (IFL)

- Free



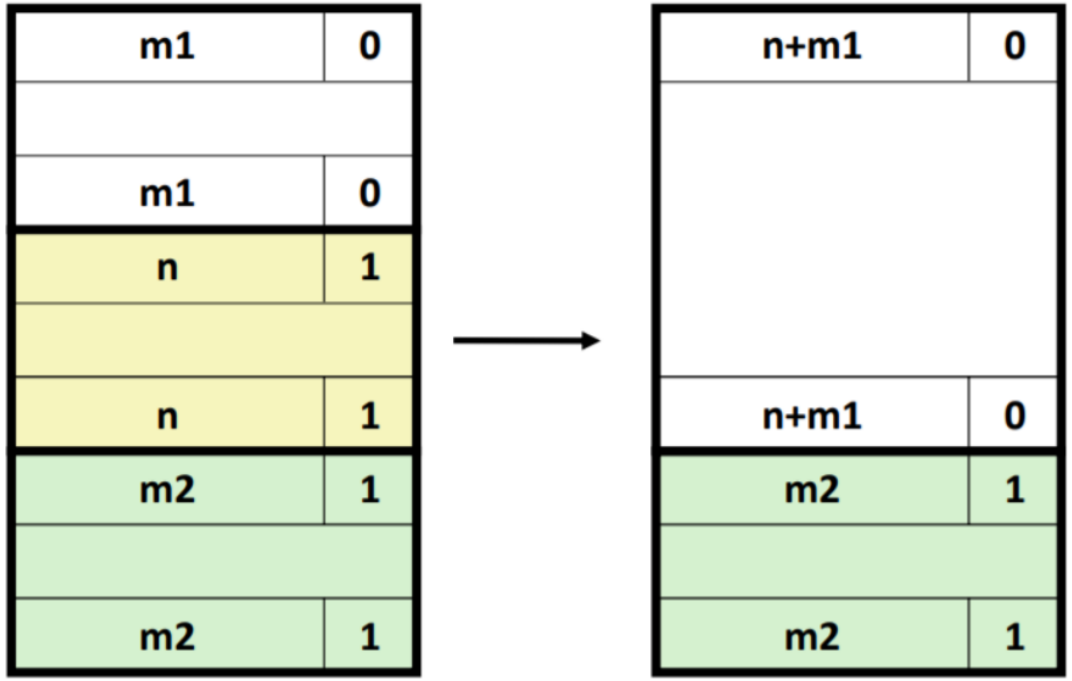
Implicit free-list (IFL)

- Free



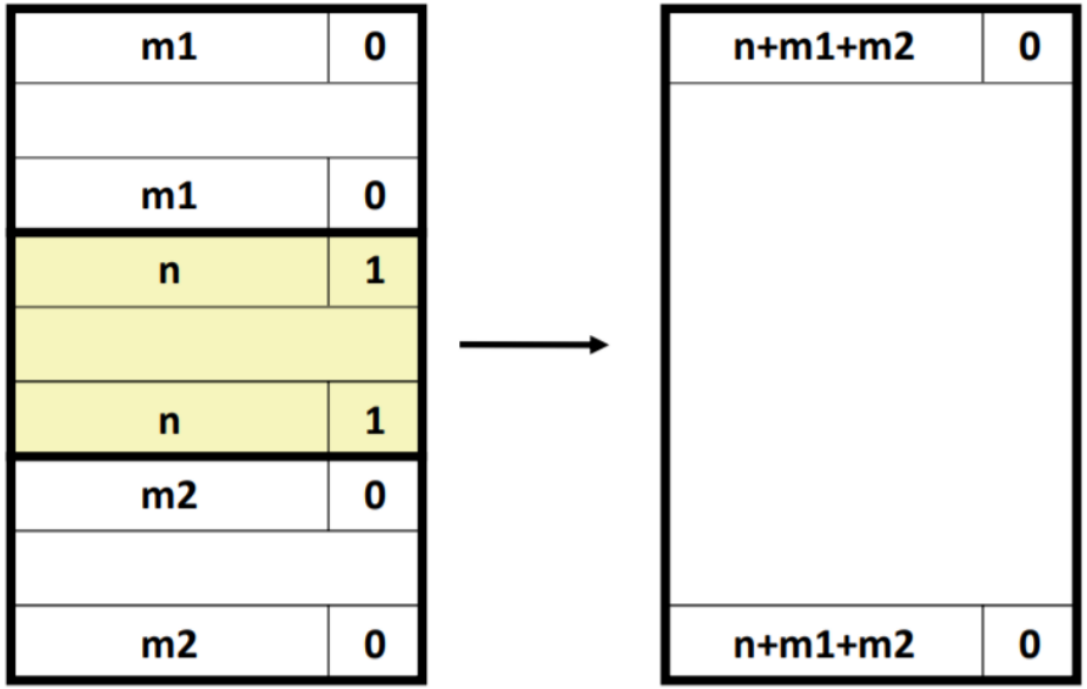
Implicit free-list (IFL)

- Free



Implicit free-list (IFL)

- Free



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

Assignment

- Write a dynamic memory allocator for C programs
- 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