CS 450 Module R5
Memory Management

West Virginia University

Spring 2023

WestVirginiaUniversity.

Goals

Goals

- Implement a heap manager for dynamically allocating and deallocating memory in MPX
- Due to its complexity and ability to render MPX unusable, we'll use a phased approach:
  - Implement the heap manager and test it via user commands
  - Integrate the heap manager into MPX

Background

Introduction

- We will keep track of memory by using Memory Control Blocks (MCB)
- We will maintain blocks in two lists
  - Free list
  - Allocated list
- This can also be done with a single list
  - Neither approach is objectively “better” - it depends on what makes sense to *you*

Memory Control Blocks

- The heap consists of a number of *Memory Blocks*
- Each Memory Block will begin an *Memory Control Block* (MCB) – don’t confuse the two types of blocks
- MCBs contain:
    - Start Address - The base address of the usable memory in the block (the first byte after the MCB)
    - Size - The size of the block in bytes, not including the MCB
    - Pointers to the next and previous MCB in the list
    - A single-list implementation would also need a flag indicating whether a block is free or allocated

`void *kmalloc(size_t, int, void  **);`

- Declared in `<mpx/vm.h>`
- Allocates memory from the low-level kernel heap
- First parameter is the size (in number of bytes) of the allocation request
- Second and third parameters are only used in VM initialization – your call to
  `kmalloc()` **must** use `0` and NULL
- Returns a pointer to the newly allocated memory

Reminder About Pointer Arithmetic

- This module is pointer intensive
- You’re going to have to do pointer arithmetic at some point
- Remember that pointer arithmetic is done in units of the underlying type:

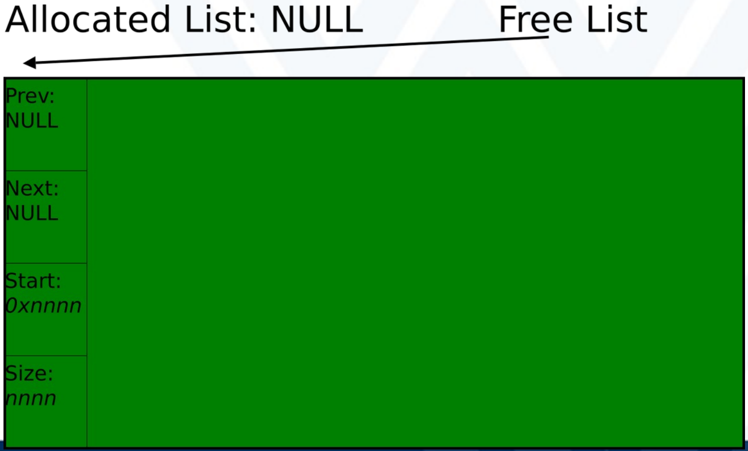
```
int * foo = some_valid_initialization();
int * bar = foo + 1; /* bar is greater than foo by sizeof(int), * not* 1 */

struct mcb * mcb = some_other_valid_initialization();
void * ptr = mcb + 1;  /* ptr is greater than mcb by sizeof( struct mcb ) */

char * str = " some string ";
str ++;          /* str now points to " some string " */

void * null = another_more_different_valid_initialization();
void * non_null = null + 1;  /* INVALID ! Pointer arithmetic on void * is UNDEFINED */
```

Library Functions

`void initialize_heap(size_t);`

- Allocates all memory available to your memory manager as a single, large free block using **kmalloc()**
- Creates an MCB for this block and places it on the free list
- Initializes the allocated list to be empty
- Parameters:
  - The total size of the heap (does *not* include the size of the initial MCB)
- Returns:
  - None

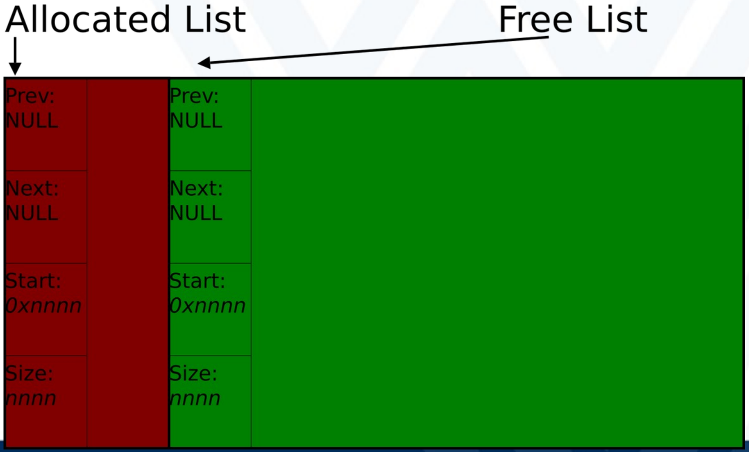
A Freshly Initialized Heap



`void *allocate_memory(size_t);`

- Allocates memory from the heap (demonstrated using first-fit)
- Splits a free memory block in two if necessary, initializing and/or updating the corresponding MCBs
- Places the allocated block on the allocated list
- Parameters:
  - The size, in bytes, of the requested allocation
- Returns:
  - `NULL` on error
  - A pointer to the start address of the newly allocated block (**not** the MCB address)

Allocating a Block of Memory



Allocated List

Free List

Prev: NULL

Next: NULL

Start: *0xnnnn*

Size: *nnnn*

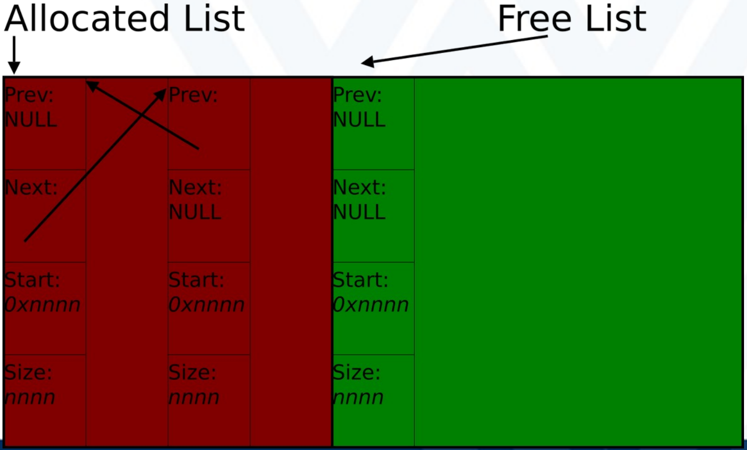
Prev: NULL

Next: NULL

Start: *0xnnnn*

Size: *nnnn*

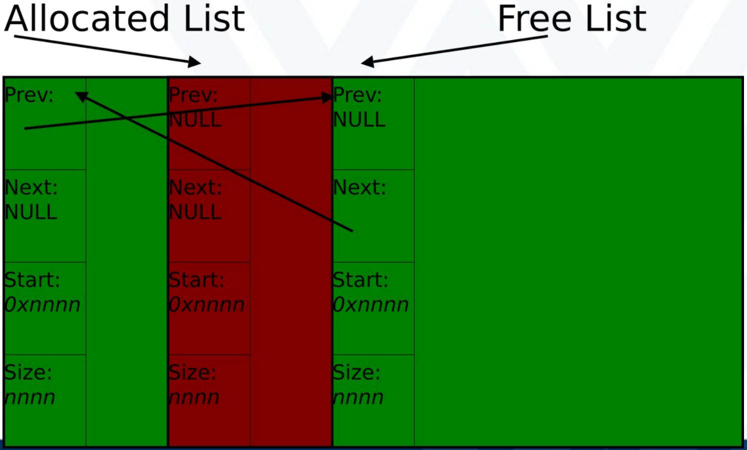
Allocating Another Block of Memory



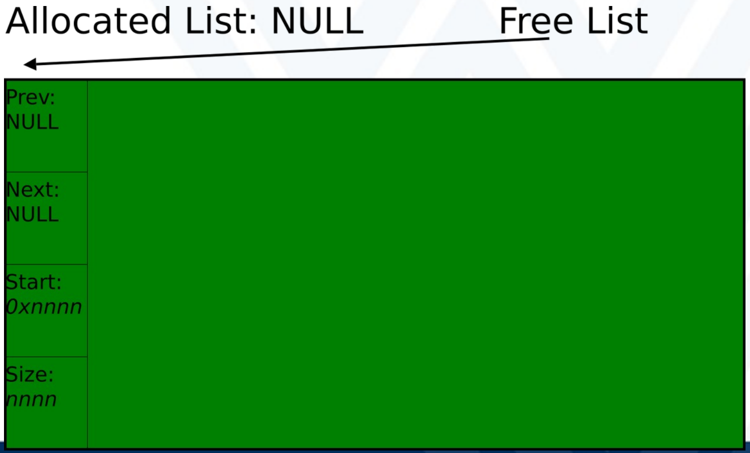
int free_memory(void *);

- Frees allocated memory, placing the associated block on the free list
- If the freed block is adjacent to any other free blocks, they **must** be merged into a single free block
- Parameters:
  - A pointer to the start address (**not** the MCB address) of an allocated block
- Returns:
  - 0 on success
  - non-zero on error

Freeing a Block of Memory



Freeing Another Block of Memory



Allocated List: NULL

Free List

Prev: NULL

Next: NULL

Start: *0xnnnn*

Size: *nnnn*

User Commands

Allocate Memory

- Allocates heap memory by calling `allocate_memory()` and prints (in hexadecimal) the address of the newly allocated block (**not** the MCB address), or an error message if allocation fails
- Parameters:
  - The size of the allocation request (in decimal)

Free Memory

- Frees heap memory by calling `free_memory()`
- Prints an error message if freeing fails
- Parameters:
  - The address of the memory block (**not** MCB) to free (in hexadecimal)

Show Allocated Memory and Show Free Memory

- Each command walks through the corresponding list, printing information for each block of memory
- Information needs to include:
    - The start address of the block (**not** the MCB address) (in hexadecimal)
    - The size of the block (in decimal)

Phased Approach

Phases

- Phase 1:
  - Implement the **library functions and user commands**
  - Initialize your heap from `kmain()` (in the “Initializing MPX modules” section): `initialize_heap(50000);`
  - Test the heck out of the library functions using the user commands
- Phase 2:
  - Replace the default MPX heap manager with yours by adding the following to `kmain()` (just after the initialization):

    ```
    sys_set_heap_functions(allocate_memory, free_memory);
    ```
  - Cross your fingers that MPX still boots

Important Note on Phase 2

- If any of your other modules require initialization from `kmain()`, that initialization **MUST** come **<u>AFTER</u>** you initialize your heap and replace the heap manager
- Put another way, R5 **MUST** be the **FIRST** module you initialize

Final Notes

Final Notes

- Remember to update:
    - Commands: Version and Help
    - Documentation: User’s and Programmer’s Manuals; Contributions
- START EARLY! This module can be very difficult to debug
- If you see values “mysteriously” changing, it’s very likely a pointer error
- Sometimes it helps to diagram things
- Library (like kernel) functions, must **not** directly perform any unspecified I/O
- Get comfortable with the debugger