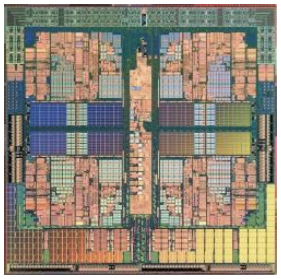
KAIST



Computer Architecture Lab.

Lab 6
Malloc lab

Contents

- **Introduction**
  - Dynamic memory allocation
- **Instructions**
  - Specification
  - Checking
  - Submission
- **Grading Policy**
- **Precautions**

Introduction

- The main goal: to implement a dynamic storage allocator, correctly, efficiently, and fast
- You will be implementing `malloc()`, `free()`, and `realloc()`
- Not on real memory, but on a simulated memory
  - Functions in `memlib.c` will be used for memory simulation
- Recommend you to study slides below
  - Lecture 16. Virtual Memory
  - Lecture 17. Memory Allocation

Introduction

- **Dynamic memory allocation**
  - **vs. static memory allocation**

Dynamic memory allocation	Static memory allocation

`int len = 30;`
`int *data =`
`(int *) malloc(len * sizeof(int));`
	`#define DATA_LEN 30`
	`int data[DATA_LEN];`

• Allocated in heap • Must call `free()` after using it	• Allocated in stack (local variable) • Automatically popped from stack when function returns

  - **Usually used to initialize array, when the size of array is not determined at compile time**
  - **Require `#include <stdlib.h>`**
  - **`malloc()`, `calloc()`, `realloc()`**

Specification

- Implement 4 functions in `mm.c`
  1. `int mm_init (void)`
  2. `void *mm_malloc (size_t size)`
  3. `void *mm_free (void *ptr)`
  4. `void *mm_realloc(void *ptr, size_t size)`
  - Refer to section 3 in the document for more details

- Implementation can be variable
  - No deterministic answer
  - Use any data structure that you can use
  - Implement it with creativity!

Specification

- You can use following functions in `memlib.c`
  - `void *mem_sbrk (int incr)`
  - `void *mem_heap_lo (void)`
  - `void *mem_heap_hi (void)`
  - `void *mem_heapsize (void)`
  - `void *mem_pagesize (void)`
  - Refer to section 4 in the document for more details

- You should not use following functions in `memlib`
  - `mem_init()`
  - `mem_deinit()`
  - `mem_reset_brk()`
  - Functions are already called by `mdriver`

Specification

- You must not use standard allocation functions
  - Using `malloc()`, `calloc()`, `realloc()`, `free()`, `sbrk`, `brk` is not allowed
- You must not define global/static compound data structurs, but allowed to declare global scalar var
  - `int a[4]`: not allowed
  - `struct element b`: not allowed
  - `struct element *bp`: allowed
  - `int b`: allowed
- You are not allowed to change the interface of `mm.c`
- Your memory allocator must always return pointers aligned by 8-byte boundaries.
  - `0x40c30018` (x)
  - `0x40c30020` (o)

Checking

- **We provide a trace-driven driver program**
  - **After typing `make, mdriver` program will be generated**
  - **Tests correctness, space utilization, and throughput**
  - **Test done with *a set of trace files* included in the same directory**

- **Trace files**
  - **`short1-bal.rep` and `short2-bal.rep` in malloclab-handout directory**
  - **We additionally provide more trace files that are will be used when we grade**

Checking

- **Run driver program**
  - Use one particular trace file for testing
    `$ ./mdriver (-V) -f short1-bal.rep`
  - Use default trace files for testing
    `$ ./mdriver (-V)`

- **Compare your code performance with `malloc()` in standard C library (libc)**
  - `$ ./mdriver (-V) -l (-f <file>)`

```
Results for libc malloc:
trace  valid  util     ops      secs  Kops
 0       yes    0%    5694  0.000980  5808
 1       yes    0%    5848  0.000833  7020
 2       yes    0%    6648  0.001347  4934
 3       yes    0%    5380  0.001339  4019
 4       yes    0%   14400  0.000496 29056
 5       yes    0%    4800  0.002130  2253
 6       yes    0%    4800  0.002007  2391
 7       yes    0%   12000  0.001130 10624
 8       yes    0%   24000  0.001245 19282
 9       yes    0%   14401  0.000696 20691
10       yes    0%   14401  0.000379 38027
Total           0%  112372  0.012581  8932
```

```
Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   99%    5694  0.007249   785
 1       yes   99%    5848  0.006822   857
 2       yes   99%    6648  0.010762   618
 3       yes  100%    5380  0.007839   686
 4       yes   66%   14400  0.000222 64953
 5       yes   92%    4800  0.006662   720
 6       yes   92%    4800  0.006400   750
 7       yes   55%   12000  0.232974    52
 8       yes   51%   24000  0.419675    57
 9       yes   27%   14401  0.153943    94
10       yes   34%   14401  0.004962  2902
Total          74%  112372  0.857509   131

Perf index = 44 (util) + 9 (thru) = 53/100
```

Checking

- **Getting summary info from autograder**
  - `$ ./mdriver -g (-V)`

```
cs230_ta@canis08:~/malloclab/src$ ./mdriver -g
Using default tracefiles in /home/lab/traces/
Perf index = 44 (util) + 9 (thru) = 53/100
correct:11
perfidx:53
```

- **Maximum score**
  - **Correctness: 11**
  - **Performance index: 100**
  - **Actual score for performance index is *0.35x of the score**

- **You should both consider memory utilization and throughput**

$$P = wU + (1 - w)min\left(1, \frac{T}{T_{libc}}\right)$$

Checking

- **Throughput**
  - # operations completed per second

- **Memory space utilization**
  - Metric of how your allocator can manage fragmentation well
  - Example: `malloc(64B), malloc(48B), free(64B), malloc(32B)`
  - Method 1: allocate 32B next to 48B alloated memory

    - $U = \frac{48+32}{64+48+32} = 0.55\%$

  - Method 2: allocate 32B to memory that is freed before

    - $U = \frac{48+32}{64+48} = 0.71\%$

Submission

- You need to submit your mm.c

- Submit your `mm.c` with the following command, within your working directory
  - `$ submit Lab6 mm.c`

- Check with the following command
  - `$ submit check Lab6`

```
cs230_ta@canis01:~/malloclab-handout$ submit Lab6 mm.c

[CS230] System Programming Submit tool
=====================================================
 * Your ID :  cs230_ta
 * Lab Number :  Lab6
 * Making first submission for Lab6
 * Upload Status : Success
 * Upload Time :  2016-11-22 21:32:23
 * Upload Count (submission version) : 1

If there seems to be a problem with your submission
please email cs230_ta@calab.kaist.ac.kr.
=====================================================
```

- **No submission, no score**

Grading Policy

- Your shell will be graded based on 11 trace files
  - Exist in /home/lab/traces
  - You don't have to see trace files: `mdriver` will handle it
  - Total 55 points

- Autograder is provided: `./mdriver -g`

```
cs230_ta@canis08:~/malloclab/src$ ./mdriver -g
Using default tracefiles in /home/lab/traces/
Perf index = 44 (util) + 9 (thru) = 53/100
correct:11
perfidx:53
```

  - Correct: # valid trace (can be checked using –v)
  - Performance: equation written in the document

- Your score = $correct * \frac{20}{11} + perf * \frac{35}{100}$

Precautions

- Please check your code compiles successfully or does not crash the driver
  - You will receive zero points otherwise
- Not everything was covered on the slides
  - Read malloclab.pdf for useful hints
- Due date: Tuesday, Dec 6, 2016, 11:59 PM
- Only electronic handins (no paper reports)
- Results are graded on the submitted code
  - No submit, no grade
  - Will be graded on the most recently submitted code