

Programming Assignment 4: malloc() Replacement

Introduction to Operating Systems
Prof. Li-Pin Chang @ NYCU

Objectives

- To replace the original libc implementation of malloc() and free() with your own version
- To evaluate the performance of two representative space allocation algorithms, namely, Best Fit and First Fit

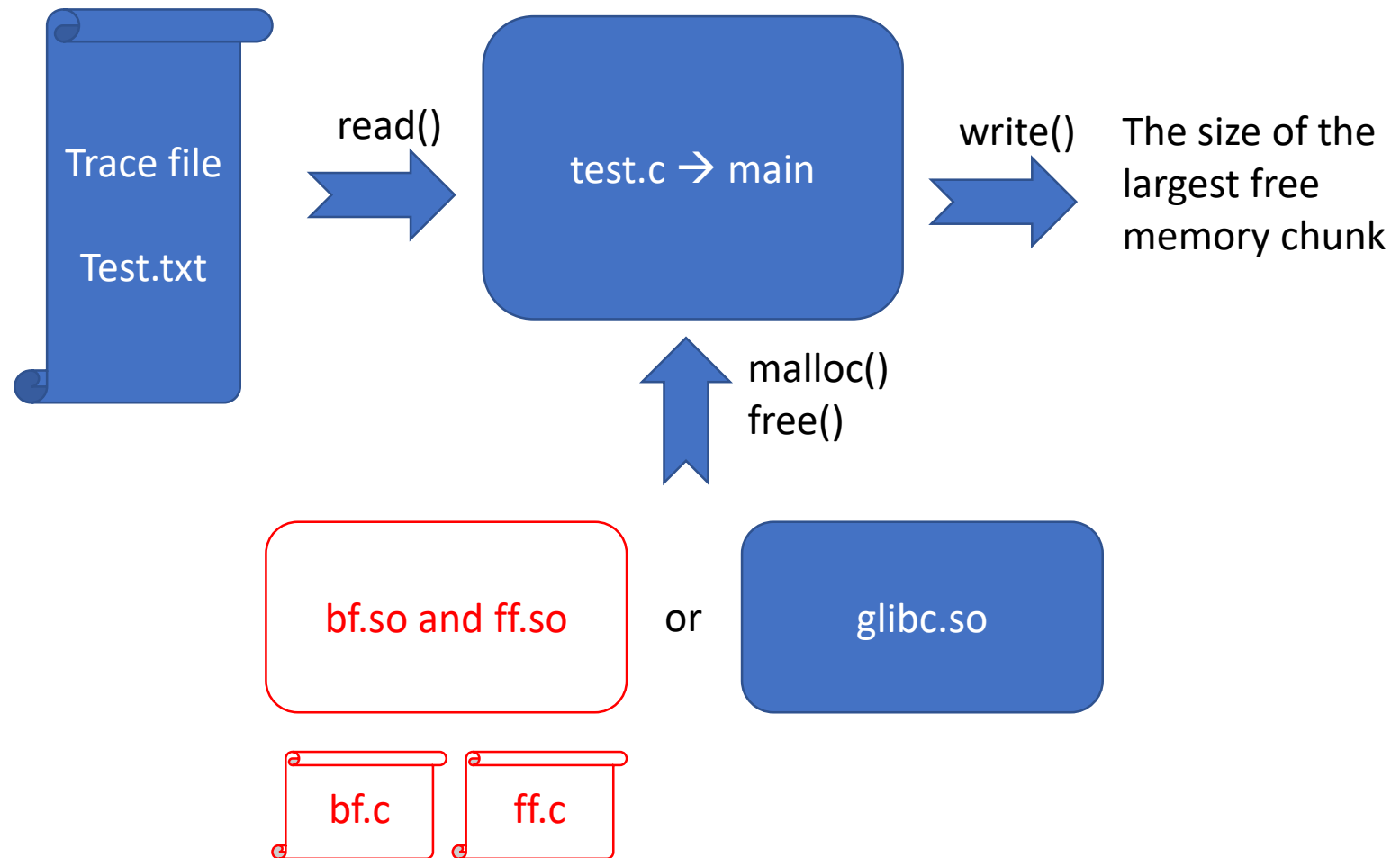
malloc()

- Part of the standard C library
- Linux adopts the GNU implementation, [glibc](#)
- Implementation details about malloc() in glibc
- Small requests (< [M_MMAP_THRESHOLD](#), i.e., 128KB) are serviced using the heap, which will be enlarged if necessary through brk()/sbrk()
- Large requests are serviced by asking the kernel to allocate a piece of anonymous memory using mmap()

Assignment Overview

- TA provides two files
 - test.txt: A input file that defines operations of memory allocation and de-allocation
 - test.c: A program tha calls malloc() and free() using the operations in test.txt
- You write two files
 - bf.c: your malloc() and free() using BEST FIT
 - ff.c: your malloc() and free() using FIRST FIT

Assignment Overview



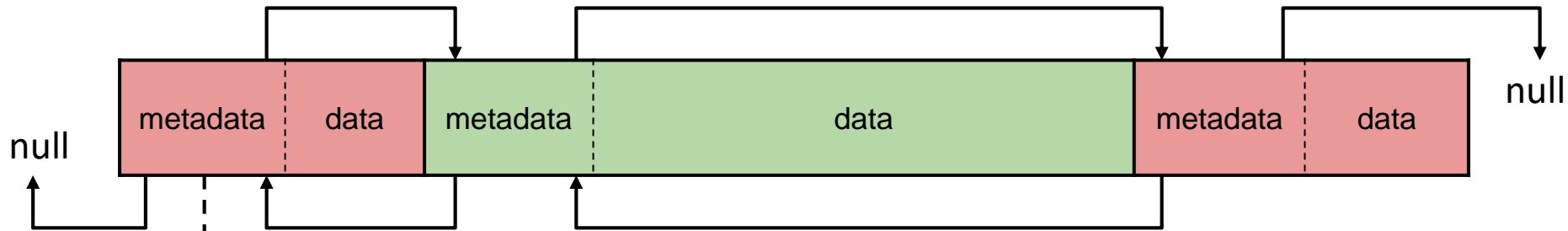
Test Flow

- 1) Compile test.c into main and put test.txt in the same dir.
 - 2) Run `$/main`
 - Should be no problem
 - 3) Compile bf.c and ff.c into bf.so and ff.so, respectively
 - 4) Run `$LD_PRELOAD=/path/to/your/bf.so ./main`
 - Print a result on the screen
 - 5) Run `$LD_PRELOAD=/path/to/your/ff.so ./main`
 - Print a result on the screen
-
- Remark: environment variable: LD_PRELOAD
 - A list of additional, user-specified, ELF shared objects to be loaded before all others
 - malloc() and free() in bf.so and ff.so override the original ones

Your Implementation (bf.c and ff.c)

- On the first malloc()
 - Pre-allocate a memory pool of 20,000 bytes from the kernel using mmap()
 - Initialize metadata for your memory pool
- On subsequent malloc() and free()
 - Process malloc() and free() within the memory pool
- On malloc(0)
 - A fake request that indicates end-of-test
 - Print the size of the largest free chunk
 - Call munmap() to release the memory pool

Metadata and Layout



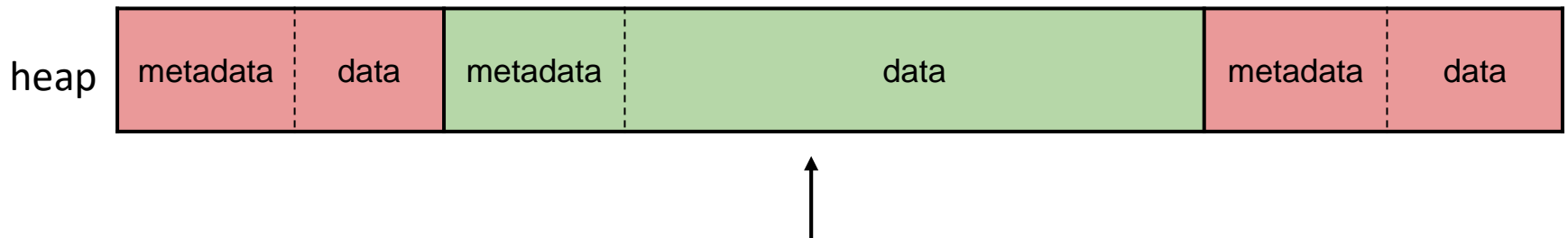
```
struct block {  
    size_t size;  
    int free;  
    struct block *prev;  
    struct block *next;  
};
```

Notice that your header must exactly use 32 bytes
(use padding if necessary)

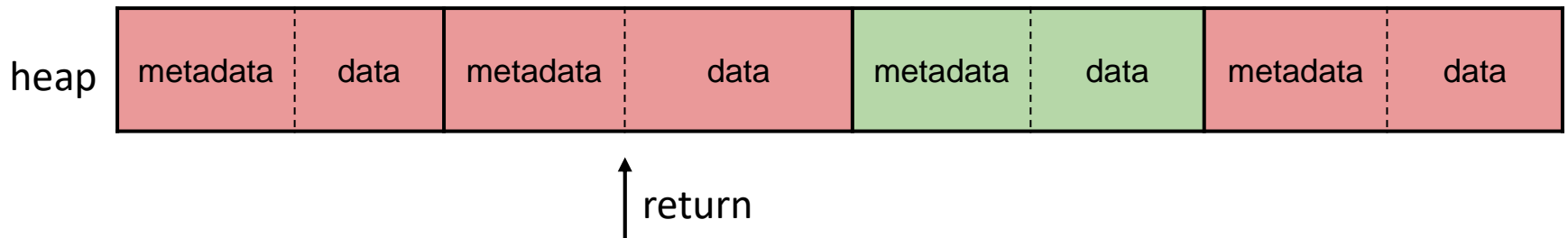
Memory Pool Management

- `void *malloc(size_t size);`

1. choose and split



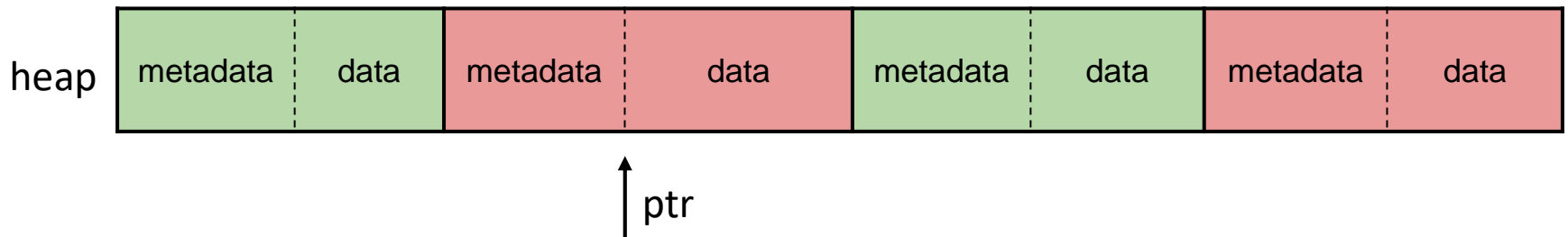
2. return the pointer



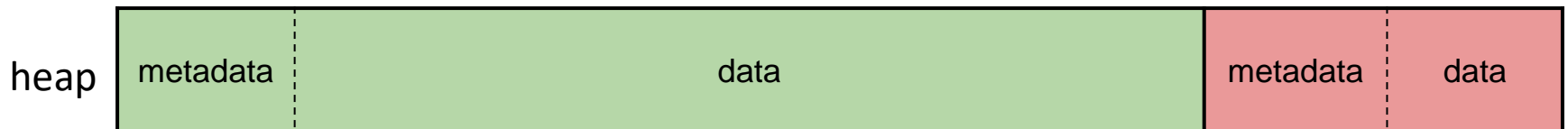
Memory Pool Management

- `void free(void *ptr);`

1. free the memory block



2. merge with free neighbor(s)



Implementation Details (!)

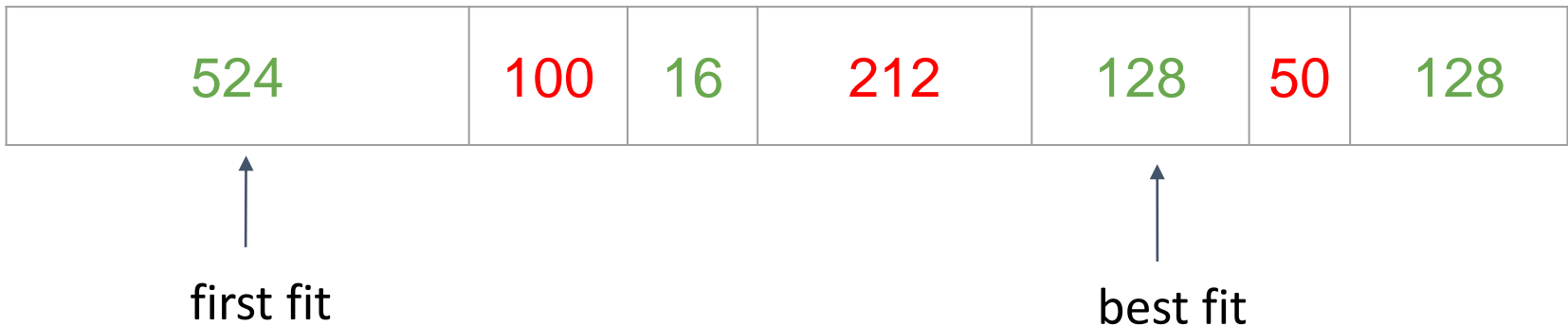
- Chunk list (chunk = space)
 - A list manages all memory chunks, both used and free
 - Initially has only one free memory chunk
- The header of a chunk is of exactly 32 bytes
 - Including paddings
- Memory alignment
 - The starting address of the memory pool must be aligned to 4 KB (this is guaranteed by `mmap()`)
 - The allocation size must be rounded to a multiple of 32
- The memory address returned by `malloc()` must all be aligned to 32 bytes. for example:
 - The starting memory address of the memory pool is 8192
 - The return address of the first `malloc(31)` is $8192 + 32$
 - The return address of the second `malloc()` is $8192 + 32 + 32 + 32$

Allocation Policies

- Write 2 different malloc libraries using these 2 policies:
- First Fit
- Best Fit
 - Use the first (leftmost) one if there are multiple choices

Example: calling malloc() for 100 bytes

Memory: free / allocated



APIs

- `<sys/mman.h>`
 - `mmap()` - creates a new mapping in the virtual address space of the calling process
 - `munmap()` - deletes the mappings
-
- <https://man7.org/linux/man-pages/man2/mmap.2.html>
 - <https://man7.org/linux/man-pages/man8/ld.so.8.html>

mmap()

- `void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);`
 - `addr` : NULL for system to choose suitable address
 - `length` : the length of the mapping
 - `prot` : `PROT_READ` | `PROT_WRITE` for read and write
 - `flags` : `MAP_ANON` since our mapping is not backed by any file, `MAP_PRIVATE` let updates invisible to other processes
 - `fd` : -1 for ignored (in conjunction with `MAP_ANON`)
 - `offset` : 0

`munmap()`

- `int munmap(void *addr, size_t length);`
 - `addr` : The starting address to be unmap (must be a multiple of the page size)
 - `length` : the length to be unmap

Remark: malloc() called within glibc APIs

- You may notice that test.c avoids using fopen(), scanf(), and printf() because these APIs call malloc() internally and will affect your result
 - fopen() -> open()
 - fread() -> read()
 - fclose() -> close()
- In your bf.c and ff.c
 - printf() -> write()
 - 在自己的malloc呼叫printf會造成deadlock喔

Input and Output

- Input filename: test.txt
- Input line format: [A or D] [id] [size]\n
 - A: Allocate, D: Deallocate
 - id: an integer identifier
 - size: bytes
- Output: size of the largest free space
 - Format: Max Free Chunk Size = \$size in bytes\$\n
 - Excluding the header
- We will provide you test.c and test.txt
- Your implementation must reproduce **exactly the same results** shown below

```
izskon@izskon-VirtualBox:~/Code/Malloc$ LD_PRELOAD=./bf.so ./main
Max Free Chunk Size = 416
izskon@izskon-VirtualBox:~/Code/Malloc$ LD_PRELOAD=./ff.so ./main
Max Free Chunk Size = 960
```

Grading Policy

- Produce correct answers for
 - The test.txt that TA give to you
 - Some other input files prepared by TA
- Submit bf.c and ff.c to E3

Testing OS Environment

- Ubuntu 18.04
- Install as a VM or on a physical machine

Header of your .c or .cpp

```
/*
```

```
Student No.: 31415926
```

```
Student Name: John Doe
```

```
Email: xxx@yyy.zzz
```

```
SE tag: xnxctxuxoxsx
```

```
Statement: I am fully aware that this program is not  
supposed to be posted to a public server, such as a  
public GitHub repository or a public web page.
```

```
*/
```

Credits

- 沈林緯、周益全 helped design this project
- Questions should be directed to TAs, **not** Mr. 沈/周