

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 Best Fit space allocation algorithm (based on the multilevel list implementation)

malloc()

- Part of the standard C library
- Linux employs the GNU implementation, [glibc](#)

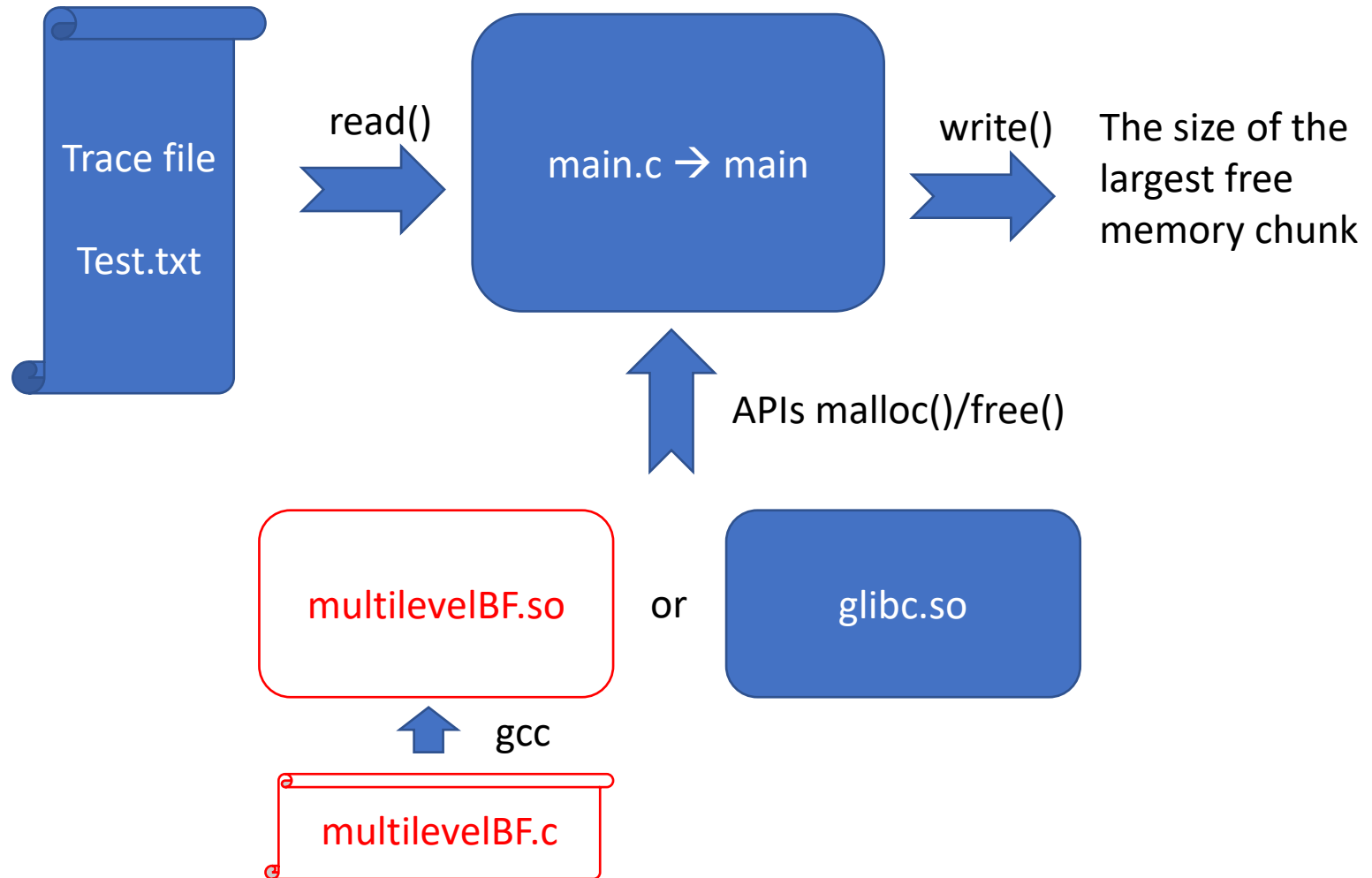
Implementation details about malloc() in glibc

- Small requests (< [M_MMAP_THRESHOLD](#), i.e., 128KB) are serviced using the heap. Heap is resized using brk()/sbrk() if necessary
- 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
 - main.c: A program that calls malloc() and free() using the operations in test.txt
- You write one file
 - multilevelBF.c: your malloc() & free() using Best Fit with multilevel free list

Assignment Overview



Test Flow

- 1) Compile main.c into main and put test.txt in the same dir.
- 2) Run `$/main`
 - Should be no problem
- 3) Compile multilevelBF.c into multilevelBF.so
`$ gcc -shared -fPIC multilevelBF.c -o multilevelBF.so`
- 4) Run `$LD_PRELOAD=/path/to/your/ multilevelBF.so ./main`
 - Print a result on the screen

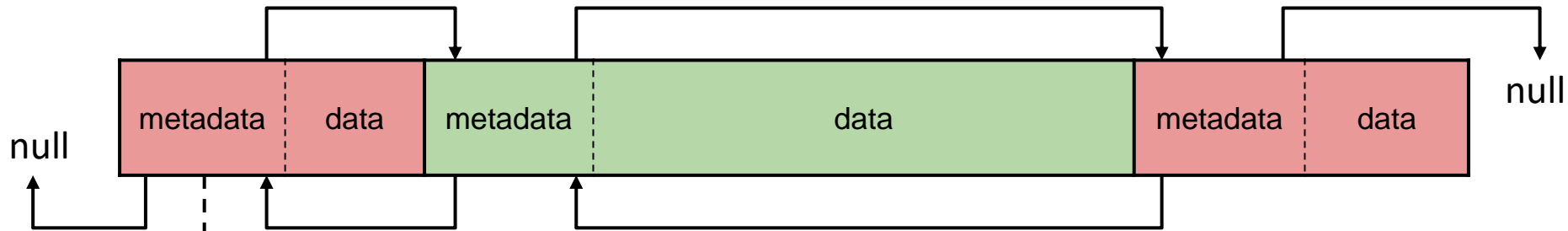
Remark: environment variable: LD_PRELOAD

- A list of additional, user-specified, ELF shared objects to be loaded before all others
- malloc() & free() in multilevelBF.so override the original ones

Your Implementation (multilevelBF.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



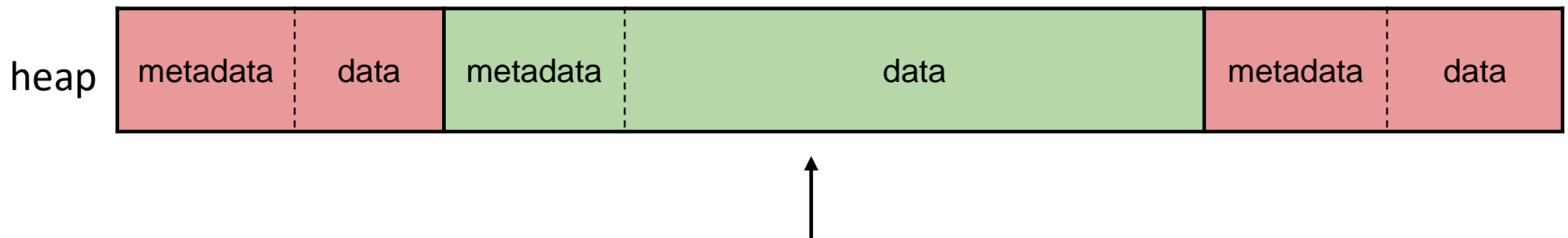
```
struct block {  
    size_t size;  
    int free;  
    struct block *next;  
    struct block *prev;  
};
```

Notice that your header must exactly use 32 bytes. Except for the prev, size, free, and next must in the metadata.(use padding if less than 32 bytes.)

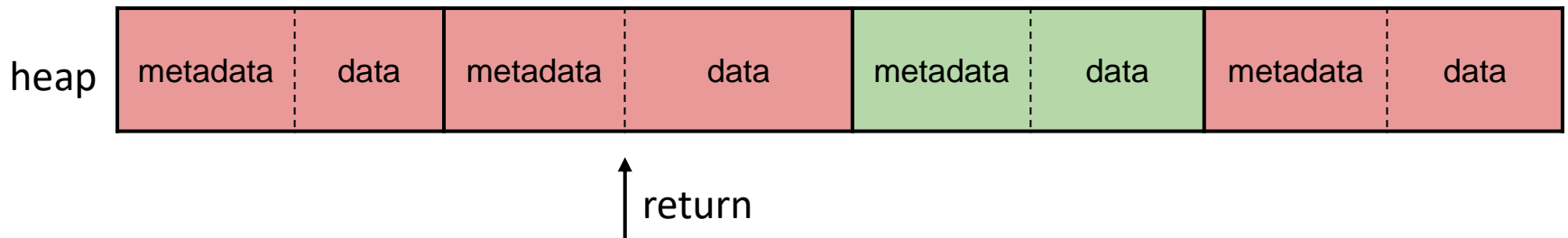
Memory Pool Management

- `void *malloc(size_t size);`

1. choose and split



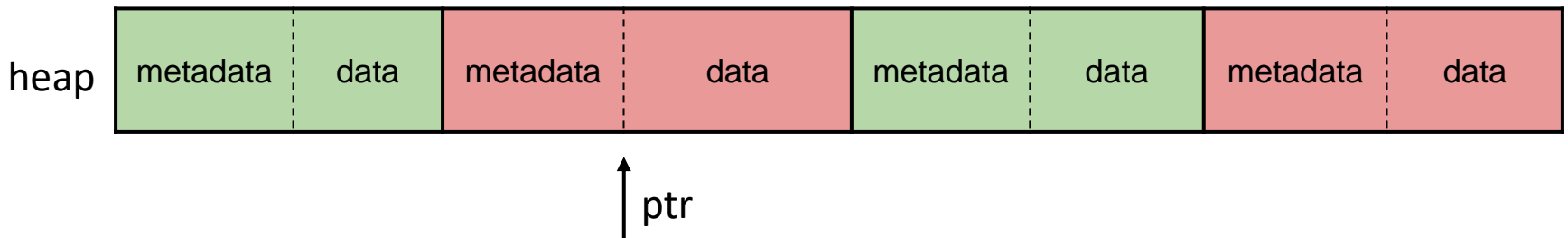
2. return the pointer



Memory Pool Management

- `void free(void *ptr);`

1. free the memory block



2. merge with free neighbor(s)

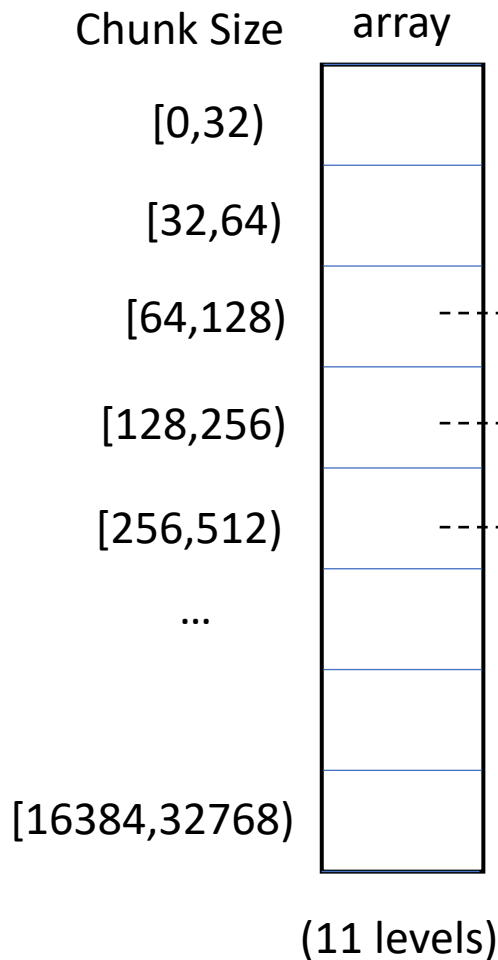


Implementation Details (!)

- Chunk list (chunk = space)
 - A list manages all memory chunks, both used and free
 - Initially has only one free memory chunk (size=20,000 bytes)
- The header of a chunk is of exactly 32 bytes
 - Including paddings (if necessary)
- 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

Multilevel Free List

First In First Out (FIFO)



- An array of pointers. A level has a pointer to a list of free blocks whose sizes are in $[2^i, 2^{i+1})$
- On allocation, select the **first** free block from the list. On release, append the block to the **end** of the list.

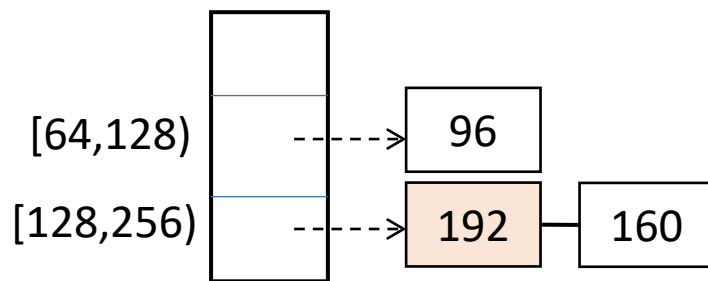
Details on malloc()

- Use the multilevel free list to find a free block
 - Find the best-fitting level (powers of 2)
 - If no free blocks, descend to the next level
 - Each level is an **FIFO**
 - Get a free block and split if necessary

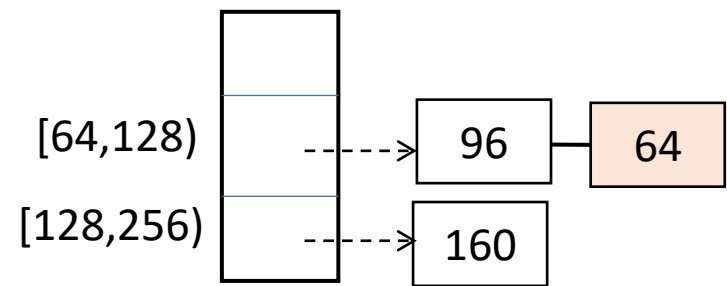
Example: malloc(100) → round to 128

Memory: free / allocated

We assume no header overhead in this example!



Free lists



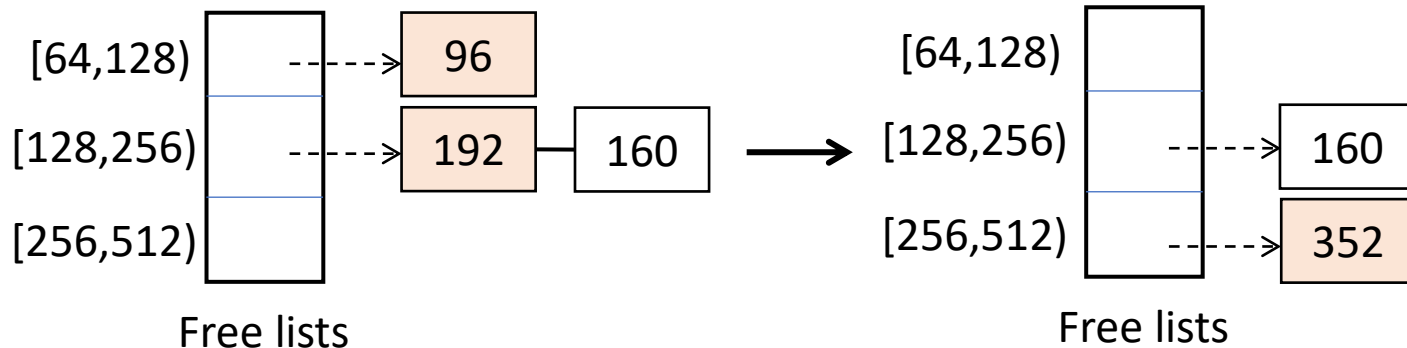
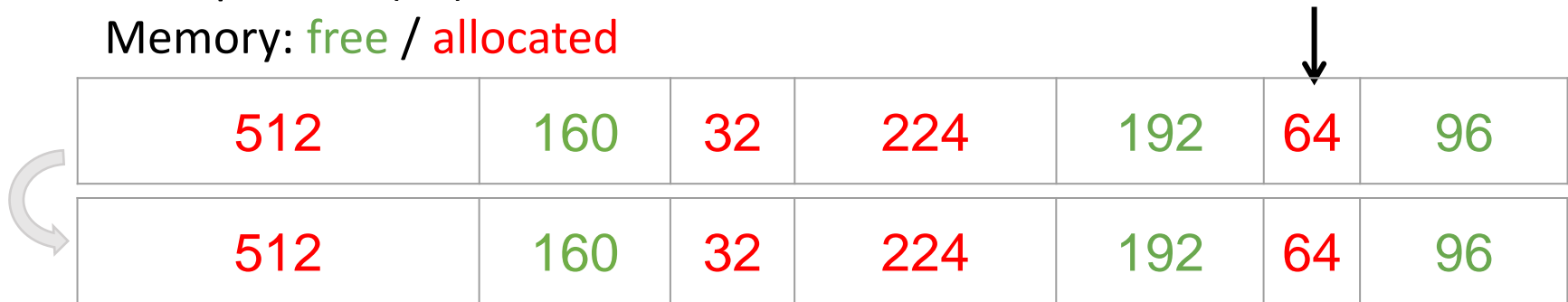
Free lists

Details on free()

- Return a block to the multilevel list
 - If neighbor(s) are also free blocks, merge them
 - Delete old blocks and append the new free block

Example: free(64)

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 main.c avoids using fopen(), scanf(), and printf() because these APIs call malloc() internally and will affect your result (or deadlock your program)
 - fopen() -> open()
 - fread() -> read()
 - fclose() -> close()
- To print out a string
 - Use a local variable string array
 - Use sprintf() to format your string
 - Use write(stdout, **) to output your string

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
 - Exclude the header
- We will provide you main.c and test.txt
- Your implementation must reproduce **exactly the same results** shown below

```
root@22954ec65807:/home/Lab/malloc/malloccode# LD_PRELOAD=$PWD/multilevelBF.so ./main
Max Free Chunk Size = 416
```

Grading Policy

- Produce correct answers for
 - The test.txt that TA give to you
 - Some other input files prepared by TA
- Submit your multilevelBF.c to E3
 - **Filename : hw4_<student_ID>.c**
 - Example: hw4_111550999.c
- Notice: It is recommended to write some testcases yourself to ensure there are no other issues (should be take care about how to free and merge) since the **provided testcase is the simplest.**

Testing OS Environment

- Ubuntu 22.04
- 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、陳虹蓓 help design this project
- Direct all questions to the current TAs