Malloc Lab

14520052 - Lâm Minh Bảo

14520470 - Phan Gia Linh

Mục lục

Yêu cầu

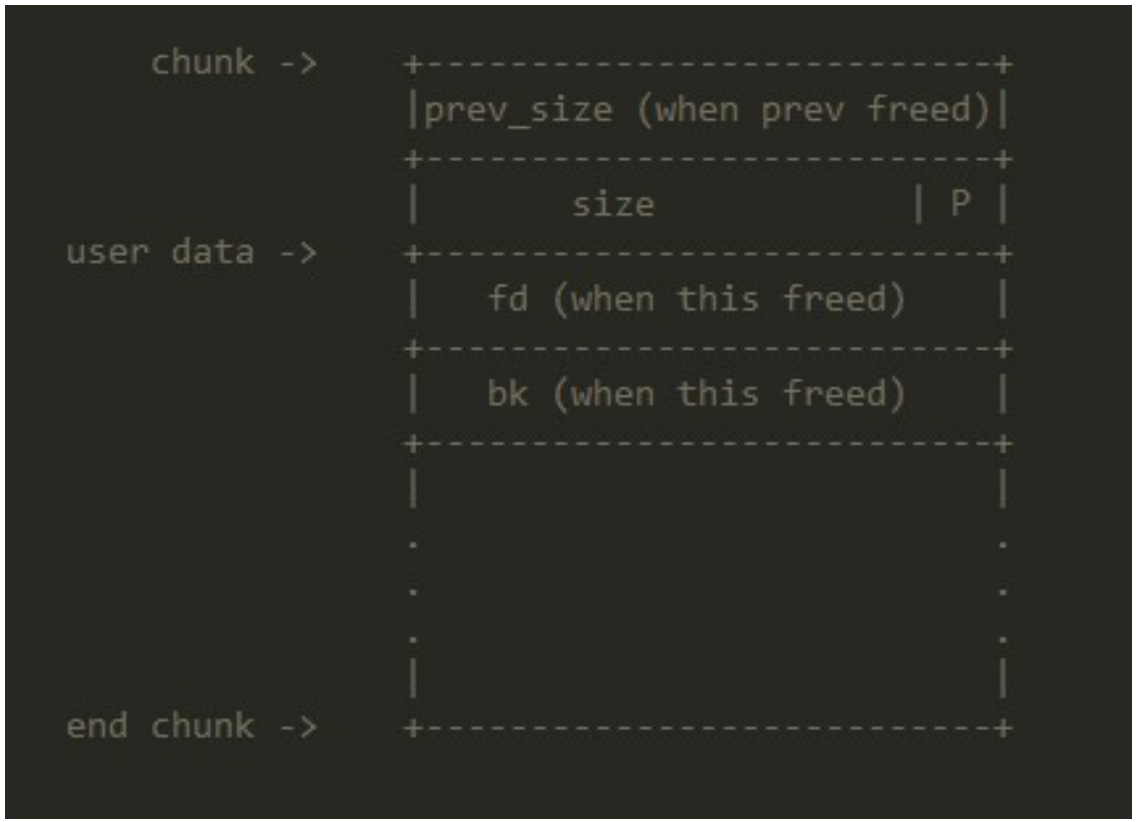
- Lập trình cấp phát động
  - `int   mm_init(void);`
  - `void  *mm_malloc(size_t size);`
  - `void  mm_free(void* ptr);`
  - `void  *mm_realloc(void* ptr, size_t size);`

Struct

- Memory được chia ra thành các chunk.
- Mỗi chunk sẽ có thông tin về:
  - Kích thước
  - Trạng thái đang sử dụng hay đã được free
  - Và các thông tin khác phục vụ cho mục đích của người lập trình.

Struct

- P (prev_inuse) = 1 nếu chunk trước đó đang được sử dụng, và ngược lại.
- fd và bk sử dụng để tạo nên double linking-list khi add chunk vào free-list tương ứng.



Struct

- Các marco để lấy/set thông tin cho chunk.

```
#define get_prev_size(p)        (*(size_t *)(p))
#define get_size(p)             (*(size_t *)((void*)p + INTERNAL_SIZE) & ~0x7)
#define get_prev_inuse(p)       (*(size_t *)((void*)p + INTERNAL_SIZE) & 0x1)

#define set_size(p, size)                   \
        (*(size_t *)((void*)p + INTERNAL_SIZE) = (size | get_prev_inuse(p)))
#define set_prev_inuse(p, prev_inuse)       \
        (*(size_t *)((void*)p + INTERNAL_SIZE) = (get_size(p) | prev_inuse))
#define set_prev_size(p, size)              \
        (*(size_t *)((void*)p) = size)
```

Policy

- Tùy vào size của chunk mà sẽ có policy khác nhau.
- Chia làm 2 cách tiếp cập:
  - Các chunk có size <= 512 (#define SMALL_RANGE 512)
  - Và các chunk còn lại.

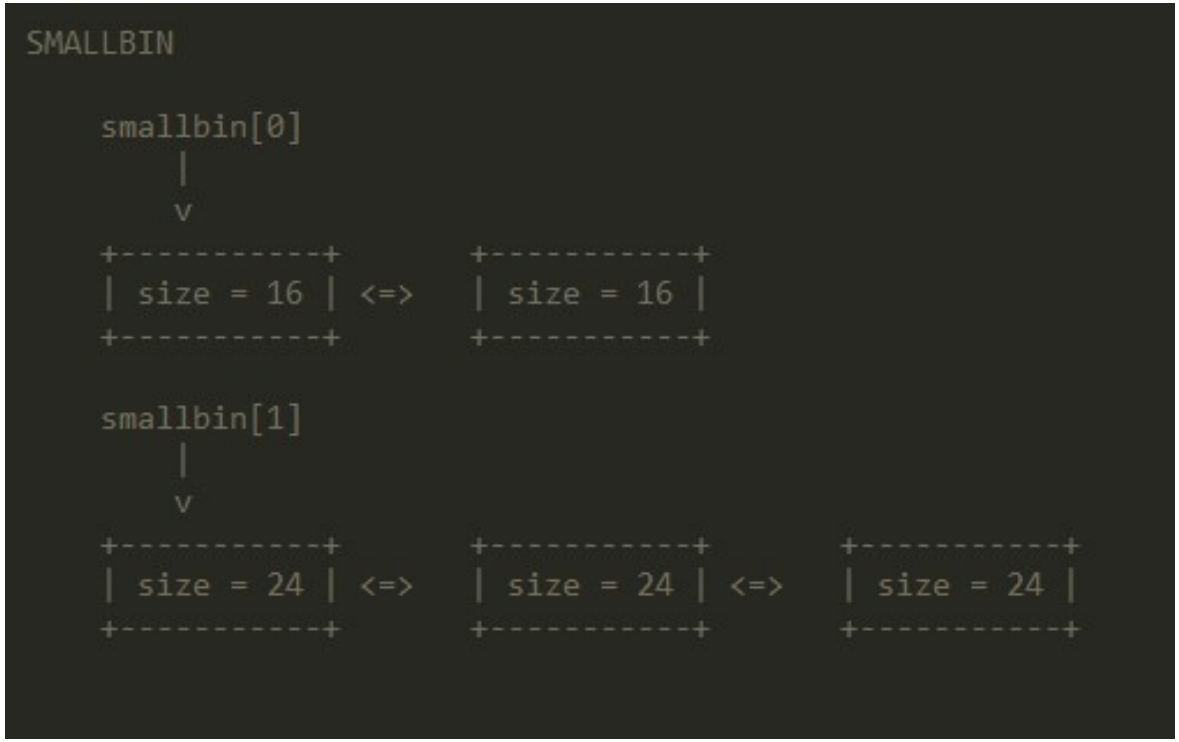
```
#define MIN_SIZE            (INTERNAL_SIZE*4)

#define SMALL_RANGE         512

#define NUM_SMALL_BIN       (((SMALL_RANGE - MIN_SIZE)>>3) + 1)
```

```
void* smallbin[NUM_SMALL_BIN];
void* largebin;
```

Policy – IN SMALL_RANGE

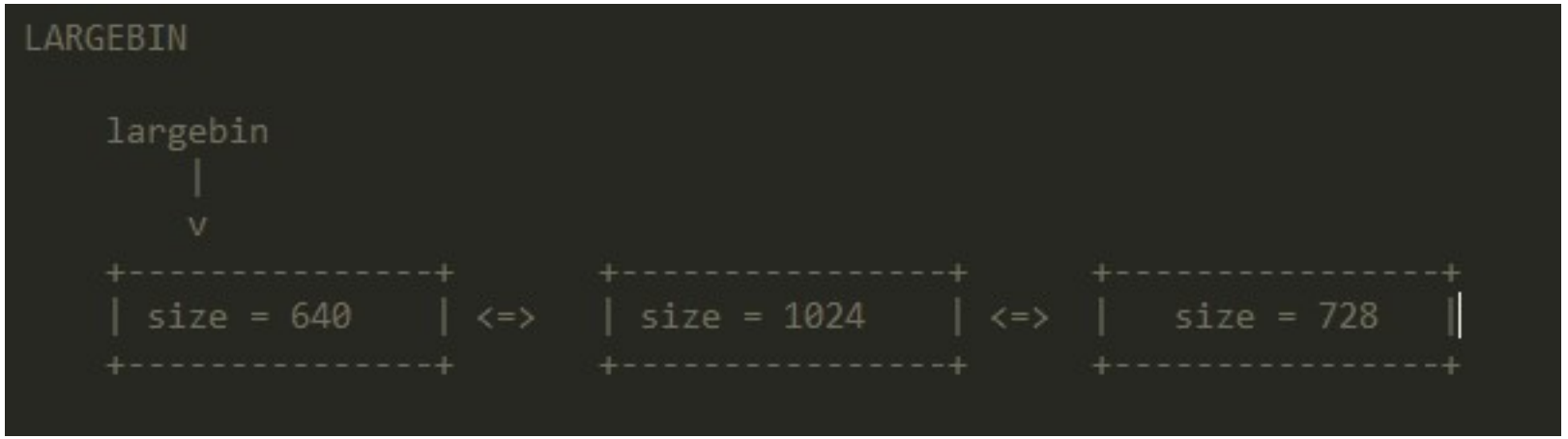
- Tạo ra các double linking-list để lưu trữ các chunk đã được free với size tương ứng, gọi là smallbin.
- Mỗi list cách nhau 8 bytes.
- VD:
  - smallbin[0] lưu các chunk có size 16 (MINSIZE 16).
  - smallbin[1] lưu các chunk có size 24.
  - ...
  - smallbin[(n-MINSIZE)>>3] lưu các chunk có size n.

Policy - IN SMALL_RANGE



Policy – NOT IN SMALL_RANGE

- Có một double linking-list lưu tất cả các chunk đã được free với size > 512, gọi là largebin.



Init

- Khởi tạo các list = NULL: smallbin và largebin.
- Và khởi tạo heap với hàm expand_heap()
- expand_heap()
  - Nếu topchunk == NULL thì sẽ khởi tạo topchunk với size = page_size() của linux. (4096 bytes)
  - Ngược lại, mở rộng topchunk với page_size() bytes.
- topchunk là chunk có địa chỉ cao nhất trong heap.

Init

- `void mm_init(void)`

```
/* init global value */
count_action = 0;
topchunk = NULL;
count_init++;

for (i=0; i < NUM_SMALL_BIN; i++)
    smallbin[i] = NULL;

largebin = NULL;

/* init topchunk by extending heap*/
extend_heap();
```

Init

- expand_heap()

```
void extend_heap() {
    if (!topchunk) {
        topchunk = mem_sbrk(PAGE_SIZE);
        /* prev_size = 0 */
        set_prev_size(topchunk, 0);
        set_head(topchunk, PAGE_SIZE, 1);
    }
    else {
        mem_sbrk(PAGE_SIZE);
        set_head(topchunk, get_size(topchunk) + PAGE_SIZE, 1);
    }
}
```

Malloc

- Khi có request với size nhất định, chúng ta sẽ xét size đó nằm trong SMALL_RANGE hay không.
- Nếu trong SMALL_RANGE
  - Tìm tới smallbin tương ứng với size.
  - Nếu smallbin không rỗng, thì lấy một chunk ra, gọi là victim.
  - Return victim cho user.
- Ngược lại (không nằm trong SMALL_RANGE)
  - Tìm first-fit chunk trong largebin.
  - Nếu có thể chia nhỏ chunk đó ra?
    - Chia làm 2 chunk -> victim và new
    - New chunk sẽ add vào list tương ứng.
  - Return victim cho user.

Malloc

- Nếu các free-list trên rỗng.
- Ta sẽ sử dụng topchunk.
  - Chúng ta sẽ expand_heap() cho đến khi topchunk có size lớn hơn request size.
  - Cắt topchunk ra và return cho user.

Malloc

- Nếu trong SMALL_RANGE

```
/* if in small range */
if (MIN_SIZE <= n && n <= SMALL_RANGE) {

    bin = &smallbin[small_index(n)];

    /* if have chunk */
    if (*bin) {

        victim = *bin;
        remove_from_corespond_bin(victim);
        /* mark as used */
        set_prev_inuse(chunk_at_offset(victim, get_size(victim)), 1);

        return chunk_to_mem(victim);
    }
```

Malloc

- không nằm trong SMALL_RANGE, tìm first-fit.

```
/* find in largebin */
else {
    bin = &largebin;
    size_t founded = 0;

    /* find first fit */
    if (*bin) {
        victim = *bin;

        do {
            if (get_size(victim) >= n){
                founded = 1;
                break;
            }
            victim = get_fd(victim);
        } while (victim != *bin);
    }
```

Malloc

- Sau đó, nếu có thể chia nhỏ?

```
if (founded) {

    remove_from_corespond_bin(victim);

    if (get_size(victim) >= MIN_SIZE + n) {
        /* split and add to corespond bin. */
        /*
            +----------------+-------------+
            | victim    | new |   next      |
            +----------------+-------------+
        */

        void* new = chunk_at_offset(victim, n);
        size_t new_size = get_size(victim) - n;
        void* next = chunk_at_offset(victim, get_size(victim));

        set_size(victim, n);
        set_size(new, new_size);

        /* new is freed chunk */
        set_prev_size(next, new_size);
        add_to_corespond_bin(new);
    }
```

Malloc

- Sau khi chia nhỏ, return victim.

```
    /* mark as used */
    set_prev_inuse(chunk_at_offset(victim, get_size(victim)), 1);

    return chunk_to_mem(victim);
}
```

Malloc

- Sử dụng topchunk

```
/* use topchunk */
while (get_size(topchunk) < (n + MIN_SIZE))
    extend_heap();

/* split topchunk */
victim = topchunk;
topchunk = chunk_at_offset(topchunk, n);
set_head(topchunk, get_size(victim) - n, 1);
set_head(victim, n, get_prev_inuse(victim));

return chunk_to_mem(victim);
```

Free

- Trước khi free một chunk, ta sẽ thực hiện consolidate() chunk đó.
- Consolidate() là sẽ hàm kết hợp các free chunk liền kề nhau trong memory đảm bảo fragmentation nhỏ nhất.

Free

- Sau đó sẽ đưa chunk đang xử lý vào free-list tương ứng bằng hàm add_to_correspond_bin()
- VD:
  - chunk size = 24 -> smallbin[1]
  - chunk size = 1024 -> largebin

Free

```
void mm_free(void *ptr)
{
    if (ptr) {

        void* p = mem_to_chunk(ptr);
        consolidate(&p);

        if (p != topchunk) {
            size_t size = get_size(p);

            /* mark this chunk is freed */
            set_prev_inuse(chunk_at_offset(p, size), 0);
            set_prev_size(chunk_at_offset(p, size), size);

            add_to_corespond_bin(p);
        }
    }
}
```

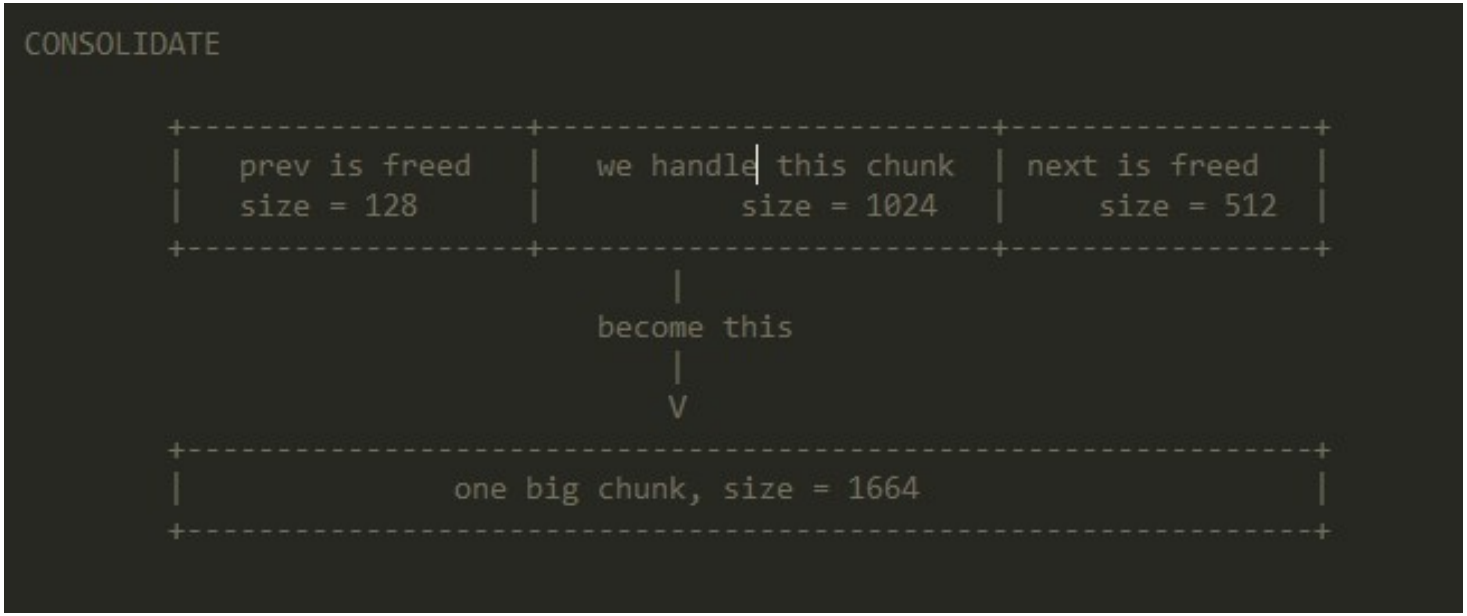
Free

- Consolidate()

```
/* consolidate next */
if (next == topchunk) {
    set_size(this, get_size(this) + get_size(next));
}
else if (!get_prev_inuse(chunk_at_offset(next, get_size(next)))) {
    set_size(this, get_size(this) + get_size(next));
    remove_from_corespond_bin(next);
}

/* consolidate previous */
if (!get_prev_inuse(this)) {
    prev = chunk_at_offset(this, -get_prev_size(this));
    remove_from_corespond_bin(prev);
    set_size(prev, get_size(prev) + get_size(this));
    *p = prev;
}

if (next == topchunk) {
    topchunk = *p;
}
```

Free

- Consolidate()

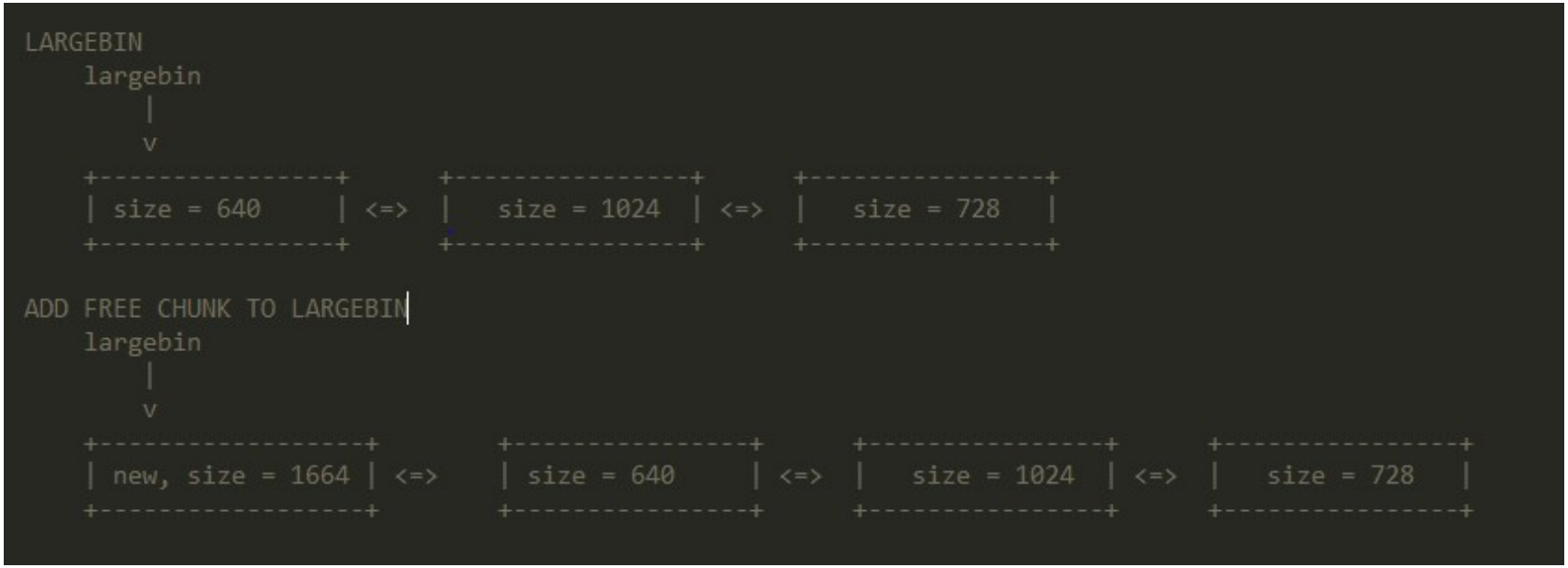


Free

- add_to_correspond_bin()

```
/* if in small range add to smallbin */
if (MIN_SIZE <= size && size <= SMALL_RANGE)
    /* bin is head of double linking-list */
    bin = &smallbin[small_index(size)];

/* if in large range add to largebin */
else
    bin = &largebin;


/* add to corespond bin */
if (!*bin) {
    /* bin is empty */
    set_fd(p, p);
    set_bk(p, p);
    *bin = p;
}
else {
    /* bin is not empty */
    bck = get_bk(*bin);

    /* insert into list */
    set_fd(p, *bin);
    set_bk(*bin, p);
    set_fd(bck, p);
    set_bk(p, bck);
    *bin = p;
}
```

Free


LARGEBIN
largebin
size = 640
<=>
size = 1024
<=>
size = 728
ADD FREE CHUNK TO LARGEBIN
largebin
new, size = 1664
<=>
size = 640
<=>
size = 1024
<=>
size = 728

Realloc

- Ý tưởng rất đơn giản.
- Ta chỉ việc free pointer được truyền vào, sau đó malloc lại chunk mới.
- Cùng với một số thủ thuật copy dữ liệu từ chunk cũ vào chunk mới.
  - Vì hàm mm_free(void* ptr) không xóa đi dữ liệu của ptr, mà chỉ thay đổi một chút ở con trỏ fd và bk nên ta chỉ việc backup data ở đó lại.

Realloc

```
memcpy(old_data, ptr, 2*INTERNAL_SIZE);

consolidate_prev_and_copydata(&p);
mm_free(chunk_to_mem(p));
result = mm_malloc(size);

memcpy(result, old_data, 2*INTERNAL_SIZE);

copysize = old_size - 2*INTERNAL_SIZE;

if (size < copysize)
    copysize = size;
copysize -= 2*INTERNAL_SIZE;

if (result != chunk_to_mem(p)) {
    memcpy(result + 2*INTERNAL_SIZE, chunk_to_mem(p) + 2*INTERNAL_SIZE, copysize);
}

return result;
```

Hết