System Programming Project 3

담당 교수 : 김영재

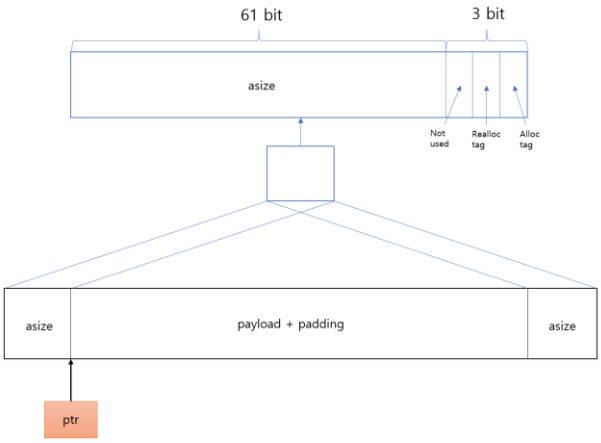
이름 : 조명재

학번 : 20192138

1. Design of allocator

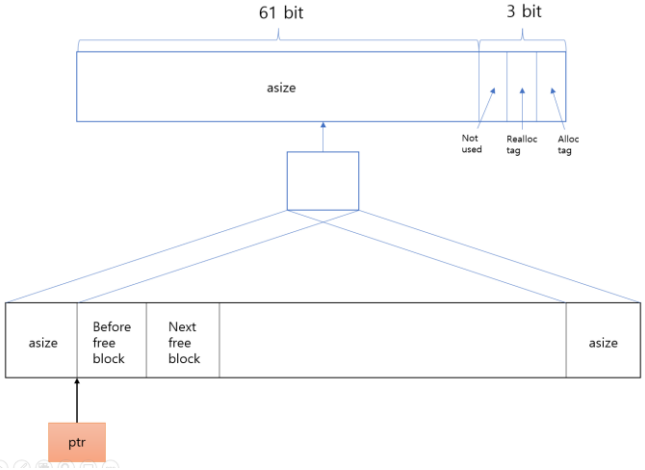
(1) allocated, free block 설계

- Allocated block 에 대한 설계



Realloc tag 를 추가하여 realloc 을 할 때 block 사이즈가 충분히 작을 경우 malloc 을 하지 않고 매개변수로 받은 할당 블록 내에서 처리를 가능하도록 하여 효율성을 극대화시킬 수 있다.

- Free block 에 대한 설계

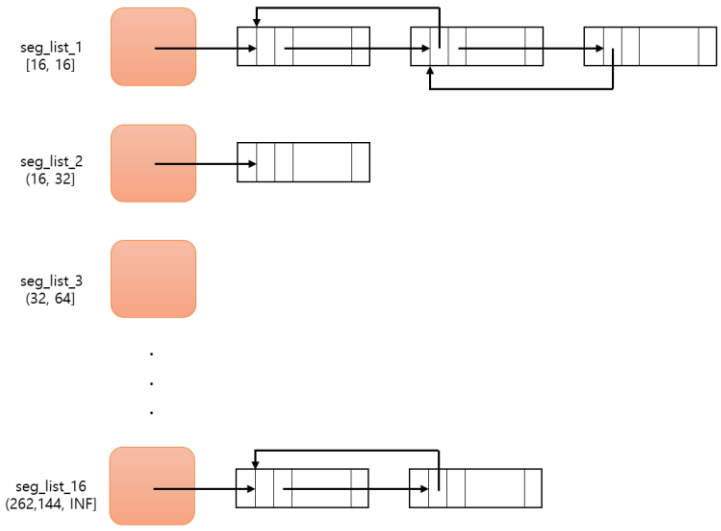


Before, next free block 은 동일한 클래스의 segregated list 에 속한 free block 들을 Double Linked List 구조로 이어주기 위한 용도이다.

(2) segregated list 설계

총 16 개의 segregated list 들이 존재한다.

seg_list_1, seg_list_2, ... , seg_list_16 와 같이 존재하게 되며, 각 list 에 허용 가능한 free block 의 사이즈와 어떤 식으로 block 을 담았는지는 다음과 같이 설계를 하였다.



Segregated list 는 세 가지 케이스로 분류할 수 있다.

1. Free block 의 사이즈가 minimum 인 경우

Header 와 footer 그리고 동일한 segregated list 내에 before, next 의 블록 주소만 담는 경우다.

즉, 최소로 16 byte 인 block 으로 이를 seg_list_1 으로 지정하였다.

2. Free block 의 사이즈가 maximum 인 경우

임의로 2^18 를 넘어선 free block 의 사이즈의 class 에 대해서 seg_list_16 에 전부 담아주도록 하였다.

3. Free block 의 사이즈가 minimum 도 maximum 도 아닌 경우

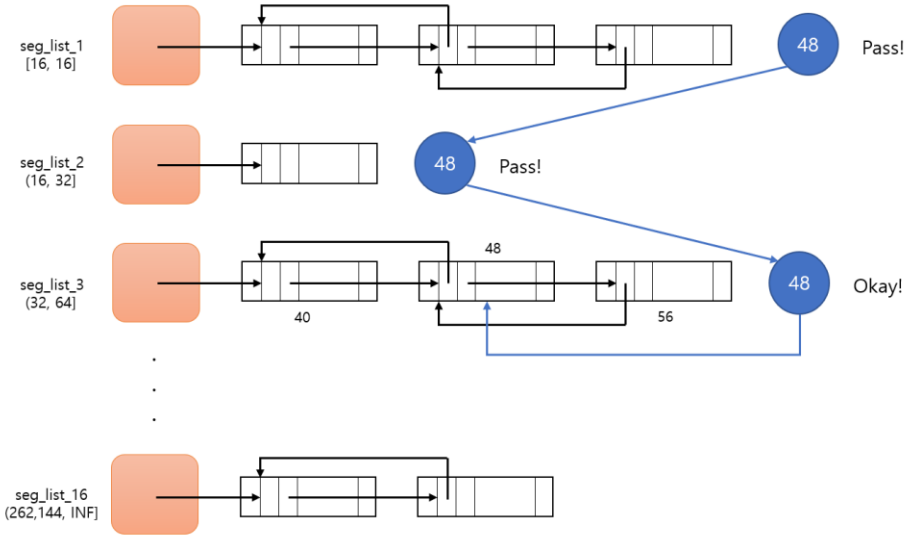
이 경우는 seg_list_i 에서 size 가 (2^(i+2), 2^(i+3)] 의 범위를 만족하는 free block 들을 담아주도록 하였다.

그리고 segregated list 내에 free block 들은 size 가 오름차순으로 정렬이 되도록 구현하였다.

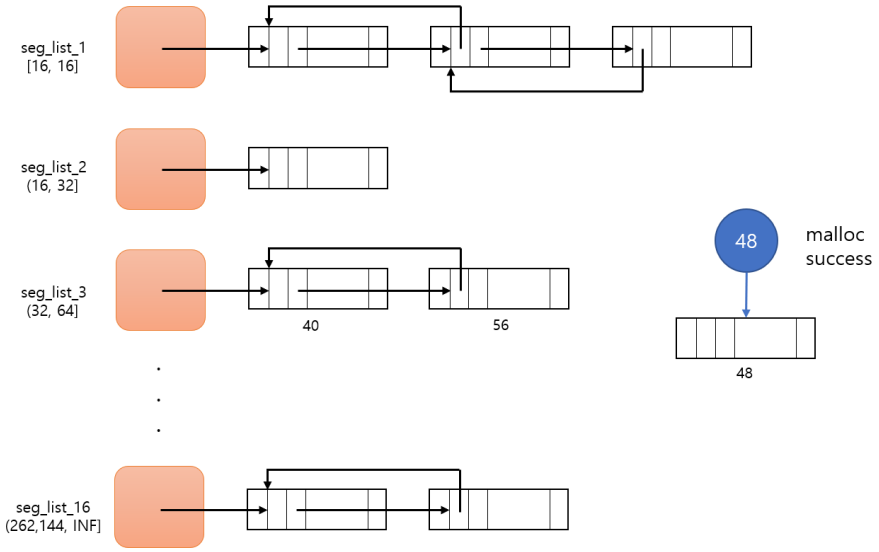
(3) mm_malloc(size_t size) 설계

매개변수로 받은 할당 사이즈 size 에 대하여 할당이 허용되는 segregated list 를 선택하여 해당 리스트를 순회하여 가장 적합한 free block 을 꺼내와서 place 작업을 수행한다.

예를 들어서 mm_malloc(43) 을 하게 될 경우, double word alignment 를 위해 할당 사이즈는 48 이며 segregated list 를 순회하여 가장 적합한 free block 을 찾는 과정은 다음과 같다.



그 후에 place 작업을 수행하게 될 경우 segregated list 에 선택된 free block 은 사라지게 되며 사라지는 블록의 전후 관계를 수정하면 다음과 같다.



(4) mm_free(void *ptr) 설계

매개변수로 받은 ptr 에 대해 free block 으로 변환을 해줘야 한다.

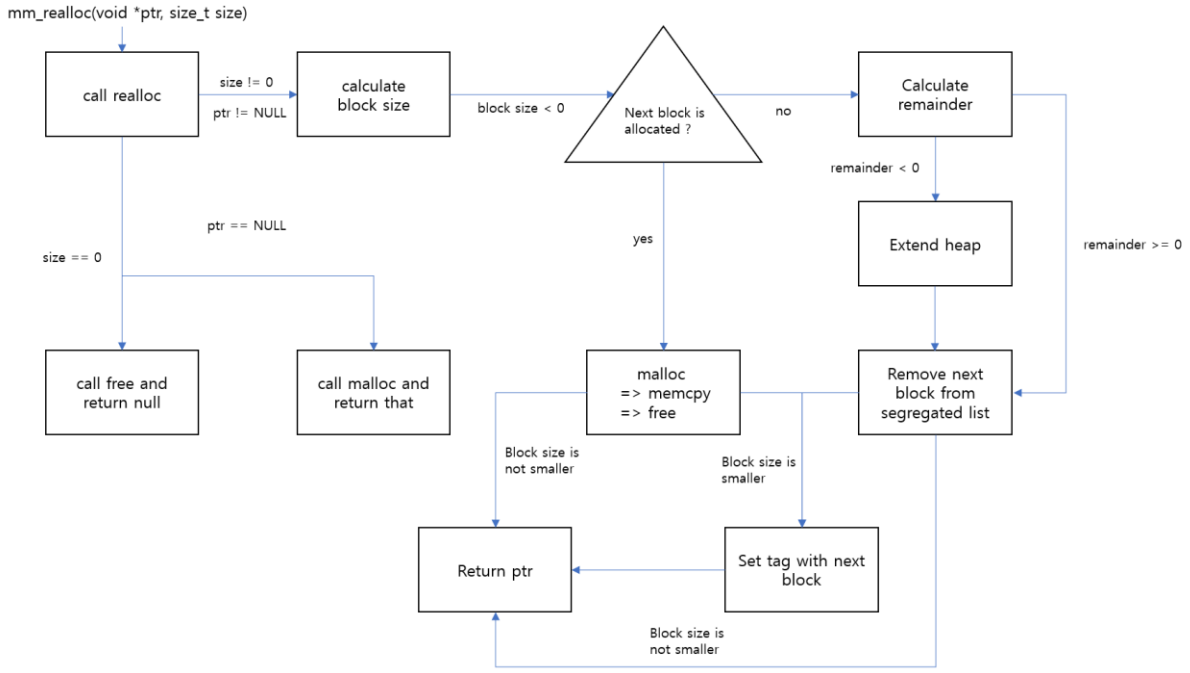
즉, header 와 footer 에 allocated bit 를 0으로 변경하며 동시에 reallocated bit 는 1로 변경한다.

그리고 segregated list 에 ptr 을 추가하는 작업을 수행한 다음 coalesce 작업을 수행한다.

(5) mm_realloc(void *ptr, size_t size) 설계

매개변수로 받은 ptr 의 사이즈가 충분히 클 경우 새로 malloc 을 하지 않고, 바로 ptr 을 리턴하도록 하여 메모리 util 을 극대화 하도록 설계하였다.

mm_realloc 함수에 대한 플로우를 그리면 다음과 같다.



2. Description of subroutines, global variables

(1) 전역변수 설명

static void * 타입으로 seg_list_1 부터 seg_list_16 까지 총 16 개의 segregated list 들을 선언하였고, 위에서 segregated list 설계에서 설명한 것처럼 동일한 class 내에 size 가 지정이 되있으며 그 지정된 size 에 맞는 free block 들이 오름차순으로 정렬이 되어있다.

(2) add_seglist(char *bp, size_t size) 설명

seg_list_1 부터 seg_list_16 까지 허용 가능한 free block size 는 위에서 언급하였다.

우선, 매개변수로 받은 size 로부터 seg_list_1 부터 seg_list_16 까지 탐색해가며 속할 수 있는 적합한 seg_list 를 선택하도록 한다.

그리고 선택된 seg_list 를 오름차순으로 매개변수로 받은 bp 를 삽입할 수 있도록 하는 위치를 선별한다.

이 때, segregated list 에 free block 을 추가할 수 있는 3가지 case 가 존재하는데 다음과 같이 분류할 수 있다.

1. list 의 head 에 부착 : 이 케이스는 list 가 전부 비어있거나 그렇지 않은 경우 케이스가 또 나뉠 수 있다.

우선, list 가 전부 빌 경우 단순히 head 에 부착을 하면 된다.

비어있지 않을 경우엔 head 에 부착 후, 이전에 head 에 해당한 free block 의 이전 블록을 bp 로 설정을 해준다.

2. list 의 가장 마지막에 삽입 : bp 의 이전 블록을 list 의 가장 마지막인 블록으로 설정하고 bp 의 다음 블록을 NULL 로 설정한다.

그리고 list 의 가장 마지막에 있었던 블록의 다음 블록을 현재 삽입하고자 하는 bp 로 설정한다.

3. list 의 중간에 삽입 : Double Linked List 의 삽입 과정과 동일하게 현재 bp 의 이전 블록을 before 로 설정하고 before 의 다음 블록을 bp 로 설정한다.

또한, bp 의 다음 블록을 next 로 설정하고 next 의 이전 블록을 bp 로 설정한다.

(3) remove_seglist(char *bp) 설명

이 함수도 add_seglist 함수와 동일한 매커니즘으로 적용된다.

우선, 매개변수로 받은 bp 의 사이즈로부터 seg_list_1 부터 seg_list_16 까지 탐색해가며 어떤 리스트에서 bp 블록을 삭제할 수 있는지를 선택하도록 한다.

bp 블록을 삭제할 적합한 리스트를 선택하게 된다면, add_seglist 와 동일하게 3 가지 케이스로 나눠서 블록을 제거할 수 있다.

1. 첫 번째 블록인 경우 : 이 케이스는 다음 블록이 존재하는 경우(블록이 2개 이상)와 다음 블록이 존재하지 않은 경우(블록이 단 1개)로 나눌 수 있다.

다음 블록이 존재하는 경우 다음 블록의 이전 블록을 NULL 로 지정하고 segregated list 의 head 를 bp 의 다음 블록으로 지정하면 된다.

다음 블록이 존재하지 않을 경우 단순히 segregated list 의 head 를 NULL 로 지정하면 된다.

2. 마지막 블록인 경우 : bp 의 이전 블록의 다음 블록을 NULL 로 지정하면 된다.

3. 중간 블록인 경우 : bp 의 이전 블록의 다음 블록을 bp 의 다음 블록으로 지정하고 bp 의 다음 블록의 이전 블록을 bp 의 이전 블록으로 지정하면 된다.

(4) coalesce(void *bp) 설명

블록 bp 를 기준으로 하여 이전 블록과 다음 블록의 할당 여부에 따라서 처리할 게 다르다.

우선 4가지 케이스로 분류가 가능하다.

1. 이전 블록과 다음 블록 전부 할당된 경우

Coalesce 작업을 할 필요가 없으므로 단순히 bp 를 반환한다.

2. 이전 블록이 free 이며 다음 블록이 할당된 경우

이전 블록과 현재 bp 블록의 coalesce 작업이 필요하다.

따라서 이전 블록과 현재 bp 블록을 segregated list 에서 제거하는 작업을 수행하고 두 블록을 합쳐서 새로운 segregated list 에 추가하는 작업을 수행한다.

3. 이전 블록이 할당되며 다음 블록이 free 인 경우

현재 bp 블록과 다음 블록의 coalesce 작업이 필요하다.

따라서 현재 bp 블록과 다음 블록을 segregated list 에서 제거하는 작업을 수행하고 두 블록을 합쳐서 새로운 segregated list 에 추가하는 작업을 수행한다.

4. 이전 블록과 다음 블록 전부 free 인 경우

이전 블록과 현재 bp 블록과 다음 블록 전부 coalesce 작업이 필요하다.

따라서 이전 블록과 현재 bp 블록과 다음 블록을 segregated list 에서 제거하는 작업을 수행하고 세 블록을 합쳐서 새로운 segregated list 에 추가하는 작업을 수행한다.

(5) place(void *bp, size_t asize) 설명

bp의 사이즈와 asize 의 값에 따라서 block 을 분할할 것인지, 분할하지 않을 것인지를 결정할 수 있다.

우선, bp의 사이즈와 asize 의 차이가 16 미만일 경우 block 분할을 하지 않는다.

그러나 차이가 16 이상인 경우 block 을 분할하도록 하여 free 공간을 남겨두도록 하였다.

3. Explanation of mm_check()

mm_check() 함수에서는 free 블록들이 정상적으로 segregated list 에 저장되어있는지를 확인하도록 구현하였다.

각각의 segregated list 를 탐색해가며 2 가지 조건을 확인하도록 하였다.

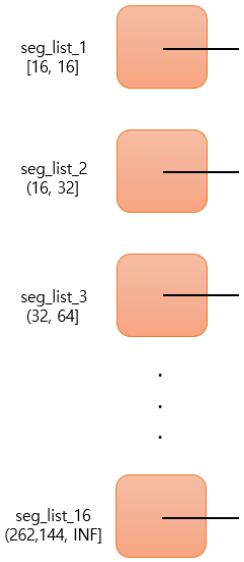
(1) Header 사이즈와 Footer 사이즈가 동일한지를 검사

Segregated list 에 free block 을 추가하고 제거하는 작업과 coalesce 작업을 처리할 때 free block 의 header 사이즈와 footer 사이즈가 다른 경우가 존재할 수 있다.

따라서, 정상적으로 header 와 footer 에 동일한 size 가 할당이 되었는지를 전부 확인하고 하나라도 다를 시 에러 문구를 띄우고 강제로 종료한다.

(2) Header 사이즈가 segregated list 의 사이즈 범위 내에 있는지를 검사

위에서 봤던 segregated list 들에 대한 정보를 살펴보도록 한다.



Segregated list 를 위와 같이 세 가지 케이스로 분류하였다.

이 세 가지 케이스에 따라 free 블록의 header 사이즈가 segregated list 의 사이즈 범위 내에 속하는지를 확인하도록 하였다.

1. Free block 의 사이즈가 minimum 인 경우

seg_list_1 에 속해있는 모든 free 블록들을 탐색하며 사이즈가 전부 16 인지 검사한다.

이 때 하나라도 사이즈가 16이 아닐 경우 에러로 강제 종료를 시킨다.

2. Free block 의 사이즈가 maximum 인 경우

seg_list_16 에 속해있는 모든 free 블록들을 탐색하며 사이즈가 (2^16, INF] 에 속하는지 검사한다.

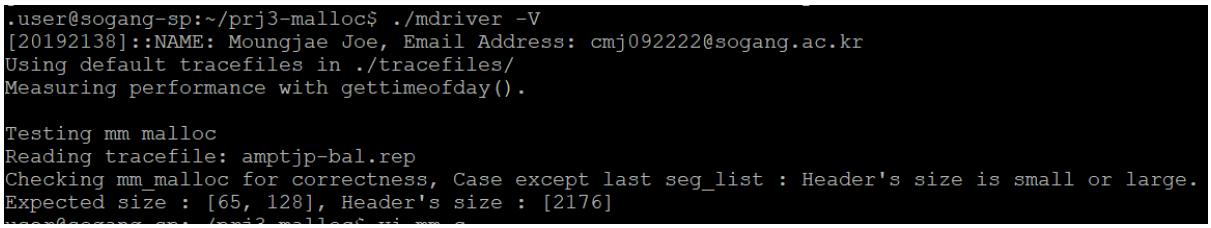
이 때 하나라도 사이즈가 (2^16, INF] 에 속하지 않을 경우 에러로 강제 종료를 시킨다.

3. Free block 의 사이즈가 minimum 도 maximum 도 아닌 경우

seg_list_2 부터 seg_list_15 에 속해있는 모든 블록들을 탐색하며 i 를 2 부터 15까지의 자연수라고 할 때, seg_list_i 의 free 블록의 사이즈가 (2^(i+2), 2^(i+3)] 에 속하는지 검사한다.

이 때 seg_list_i 의 free 블록이 하나라도 사이즈가 (2^(i+2), 2^(i+3)] 에 속하지 않을 경우 에러로 강제 종료를 시킨다.

아래의 사진은 mm_check() 를 통해서 비정상적으로 free 블록이 segregated list 에 속한 경우를 찾은 경우이다.



직접 mm_check() 함수를 커스텀하여 mm_malloc 함수를 통해 place 를 한 다음에 수행하거나 mm_free() 함수가 끝나는 시점에 수행하여 정상적으로 segregated list 에 추가되거나 제거되었는지, 또는 coalesce 작업이 정상적으로 수행되었는지를 확인할 수 있었다.