System Programming Project 3

담당 교수 : 김영재

이름 : 박성현

학번 : 20181632

1. 개발 목표

이번 프로젝트에서는 c언어를 사용해 나만의 malloc package를 만든다. Mm_malloc, mm_free, mm_realloc과 같은 기본 libc의 malloc과 같은 구성이다. 내가 만든 malloc을 통해 dynamic memory allocation의 메모리와 시간적인 관점에서 효율성을 극대화 시켜야 한다.

2. 개발 내용

A. Global variables

i. Static char* heap_listp

1. Heap을 가르키는 포인터이다. Mem_sbrk 를 실행해서 heap영역을 할당 받고 그 heap의 block들을 가르킨다.

ii. Size_t seg_lists

1. Segregate list의 index 개수이다. 12로 설정했다.

iii. Static char **list_p

1. Seg_list를 가르키는 포인터이다. List_p는 free list들을 관리하는 배열이며 각 인덱스에 따라 크기별로 free block들이 저장되어 있다.

B. Macros (책에 있는 코드에서 가져온 기본적인 매크로함수 제외 직접 추가한 매크로함수만 작성하였다.)

아래에 직접 정의한 모든 매크로는 seg_list, 즉 free-list를 관리하는 매크로이다.

i. PREV_BLK_RD(bp)

1. Seg_list에서 주어진 포인터의 전 block을 읽는다. Seg_list에서 읽으므로 항상 free block만 가져온다.

ii. **NEXT_BLK_RD(bp)**

1. Seg_list에서 주어진 포인터의 다음 block을 읽는다. PREV_BLK_RD(bp)와 같은 방식으로 작동한다.

iii. **PREV_BLK_WT(bp, ptr)**

1. Seg_list에서 주어진 bp block의 previous free block을 가르키는 block이 ptr이 되도록 설정한다. 이 함수를 실행하면 bp의 전 free block은 ptr이 된다.

iv. **NEXT_BLK_WT(bp, ptr)**

1. Seg_list에서 주어진 bp block의 next free block을 가르키는 block이 ptr이 되도록 설정한다. PREV_BLK_WT(bp,ptr)과 같은 방식으로 작동한다.

C. Functions

i. **Int mm_init(void)**

1. Malloc을 처음 실행 시 실행되는 함수이다. 가장 먼저 seg_list의 공간을 sbrk를 통해 정해진 index 개수만큼 할당한다. Size는 WSIZE이다. 그후, heap공간을 할당한다. heap공간은 Alignment padding, prologue header, footer, 그리고 Epilogue header의 공간이 필요하기에 최소 4*WSIZE(2*DSIZE)의 공간이 필요하다. Heap_listp가 prologue header 다음 block을 가르키게 한 후, seg_list에 초기값 NULL을 할당한다.

ii. **Void *mm_malloc(size_t size)**

1. malloc함수이다. 할당할 크기를 인자로 받는다. 각 block은 header와 footer를 가져야하기에 DSIZE보다 크기가 작으면 2*DSIZE로 크기를 설정한다. 다른 경우에는 추가로 header와

footer의 공간이 필요하기에 size를 적당히 조절한다. 그 후, size로 find_fit함수를 사용해 가장 적절한 크기를 가진 free block을 찾는다. 이 block으로 place함수를 실행해 block을 size 만큼 할당하고 남은 block을 free list에 넣는다. 그 후, 할당된 block을 반환한다. 만약 크기가 적당한 free block을 찾지 못했다면 extend_heap을 사용해 heap의 크기를 늘리고 할당 받은 block을 place함수를 사용해 자른 후 할당된 block을 반환한다.

iii. **Void mm_free(void* ptr)**

1. Ptr로 주어진 block을 free시키는 함수이다. Ptr 크기만큼 header와 footer에 적고 allocate bit을 0으로 설정한다. 그리고 그 block을 coalesce함수로 넘겨 앞뒤의 free한 block들과 합쳐준다.

iv. **Static int get_index(size_t size)**

1. Block의 사이즈를 인자로 받아 그 사이즈가 들어갈 seg_list의 인덱스 번호를 반환한다. Memory utilization을 위해 값을 큰 범위로 설정했다.

v. **Static void delete_(void *bp)**

1. 인자로 받은 block을 seg_list에서 제거시켜주는 함수이다. allocate되었을 때 사용된다. Free list에서는 하나의 block이 previous free block과 next free block을 가르키는 포인터를 가지고 있다. 앞뒤의 block의 유무에 따라 bp를 제거하고 앞과 뒤 block의 포인터들을 재설정해준다.

vi. **Static void insert_(void* bp)**

1. 인자로 받은 block을 seg_list에 삽입하는 함수이다. Bp의 크기에 알맞은 index를 찾고 free block을 list_p[index]의 제일 앞에 넣어준다.

vii. **Static void* coalesce(void *bp)**

1. 인자로 받은 bp를 앞과 뒤의 free block들과 합쳐주는 함수이다. 기본적으로 block들은 heap에 모두 저장되어 있고 seg_list가 따로 free한 block들만 pointer로 관리해준다. 그러기에 구조 상 현 block의 앞뒤에는 free block이 있을 수도, 없을 수도 있다. 메모리 관점에서 외부 단편화를 줄이기 위해서는 두개의 free block이 연속으로 존재하면 안된다.

2. 먼저 bp의 앞과 뒤의 block들이 allocate되어있는지 확인한다. 만약 둘다 allocate면 그냥 bp를 insert_함수를 통해 seg_list에 넣어준다. 뒤가 free block이면 그 block을 먼저 seg_list에서 제거한 후에 두개의 block을 합친다. Size와 header, footer를 업데이트 해주고 다시 그 block을 seg_list에 넣어준다. 전 block이 free block일 경우도 같은 작업을 해준다. 추가로 bp의 pointer를 전 block으로 옮겨 bp가 합쳐진 free block을 다 가르킬수 있도록 한다. 마지막으로 앞 뒤가 둘다 free block이라면 두 block다 제거한 후 합쳐준다. 그리고 다시 seg_list에 넣어준다.

viii. **Static void* find_fit(size_t asize)**

1. 이 함수는 best-fit으로 가장 적합한 free block을 탐색한다. 인자로 받은 size가 들어갈 seg_list의 index를 찾아 그 index부터 탐색을 시작한다. 그 index의 모든 free block을 탐색하면서 size와 가장 비슷한 크기의 block을 찾는다. 아래의 if (best!=NULL) 문은 첫번째로 설정한 index에서 찾았을 경우 바로 그 ptr을 반환한다. Index가 변했다는 것은 get_index으로 받은 적절한 index의 값에 free block이 없다는 것이고 다음 index에서 탐색한다는 의미인데 그렇게 되면 메모리 관점에서 내부 단편화가 크게 발생할 수 있다. 그러므로 index에 변화를 주기전에 적당한 block을 찾았으면 바로 반환한다.

ix. **Static void place(void* bp, size_t asize, int remove)**

1. 이 함수는 주어진 block을 알맞게 잘라서 사용할 부분을 제외한 나머지 부분을 seg_list에 넣어주는 역할을 한다. remove라는 추가 인자를 받는다. 이 값이 1이면 free list에서 빼서 자른다. 저음 mm_malloc함수 실행 시, find_fit으로 적당한 위치를 찾았을 경우에 remove값이 1로 할당된다. 이땐 그 free block이 size만큼 할당 되고도 다른 block을 allocate할수 있는 크기인지 먼저 확인한다. 적당한 공간이 남을 경우 그 free block을 list에서 제거한다. 그리고 block을 size만큼 나눠서 allocate해주고 남은 block은 다시 seg_list에 넣어준다. 적당한 공간이 남지 않을 경우 split하지 않고 다 할당한다. Remove가 0일 때는 extend heap 했을 때 이다. 이때는 seg_list에서 block을 가져오지 않는다. Delete_함수를 실행하지 않는다는 것이 remove 1일때와 다르다. 다음 과정은 똑같다. 함수를 split하고 남은 크기를 확인하고 적당한 크기이면 나누어서 seg_list에 넣어주고 아니면 block 전체를 할당해준다.

x. **Void* mm_realloc(void *ptr, size_t size)**

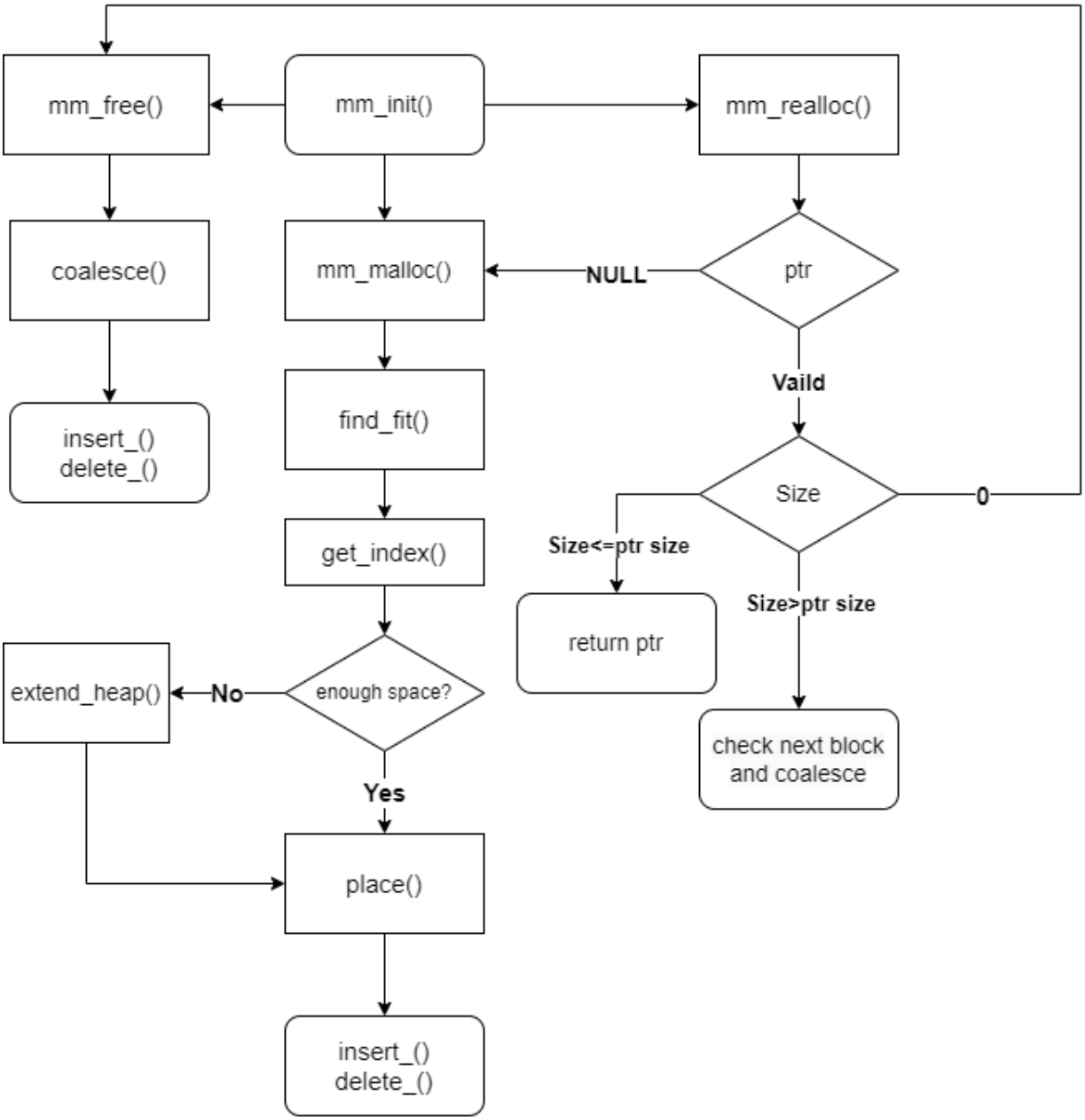
1. 이 함수는 realloc함수와 같은 역할을 한다. 먼저 인자로 받은 size는 순수하게 payload의 크기이므로 padding을 추가한 크기로 변경해 준다. 그 후, size가 0이면 free 와 같으므로 free를 해주고 함수를 종료 시킨다. 만약 요청한 크기가 원래 block의 크기보다 작거나 같으면 원래의 block을 반환한다. 만약 크기가 더 크다면 다음 block을 확인한다. 만약 다음 block이 free한 block이라면 현재 내 block과 다음의 free block을 합쳐서 realloc한 크기를 가질 수 있는지 확인한 후, 그 free block을 seg_list에서 제거하고 합친 후, 합쳐진 block을 반환한다. Previous block이랑 합치지 않는 이유는 만약 전 block이랑 합친다면 데이터를 앞으로 옮겨 줘야한다. 하지만 이 과정에서 data overlapping이 발생하기 때문에 뒤의 block과만 합쳐준다. Next block이 free block이 아닐 경우, 그 size만큼 malloc을 하

고 데이터를 옮겨 적어준다. 새로 할당된 block을 반환하고 예전 block은 free시켜준다.

xi. **Static int mm_check(void)**

1. Heap을 종합적으로 체크해주는 함수이다. 가장 먼저 seg_list를 돌며 free 하지 않은 block이 seg_list에 있는지 확인한다. 그 후, heap을 전체적으로 돌면서 header와 footer의 값이 일치하는지, 그리고 coalesce되지 않은 두개의 free한 block이 있는지 확인한다.

D. Flowchart



1. 이 프로그램의 flowchart이다. Mm_init 실행 후, mm_malloc, mm_free, mm_realloc을 호출하게 된다. Malloc을 호출하면 find_fit을 통해 적당한 크기의 free block을 찾는다 찾지 못하면 extend heap을 통해 heap의 크기를 늘려준다. 그 후, place함수를 통해 free block을 seg_list에서 가져와 할당한다. 만약 mm_free를 호출 했을 경우, 그 block을 coalesce해준다. 이 과

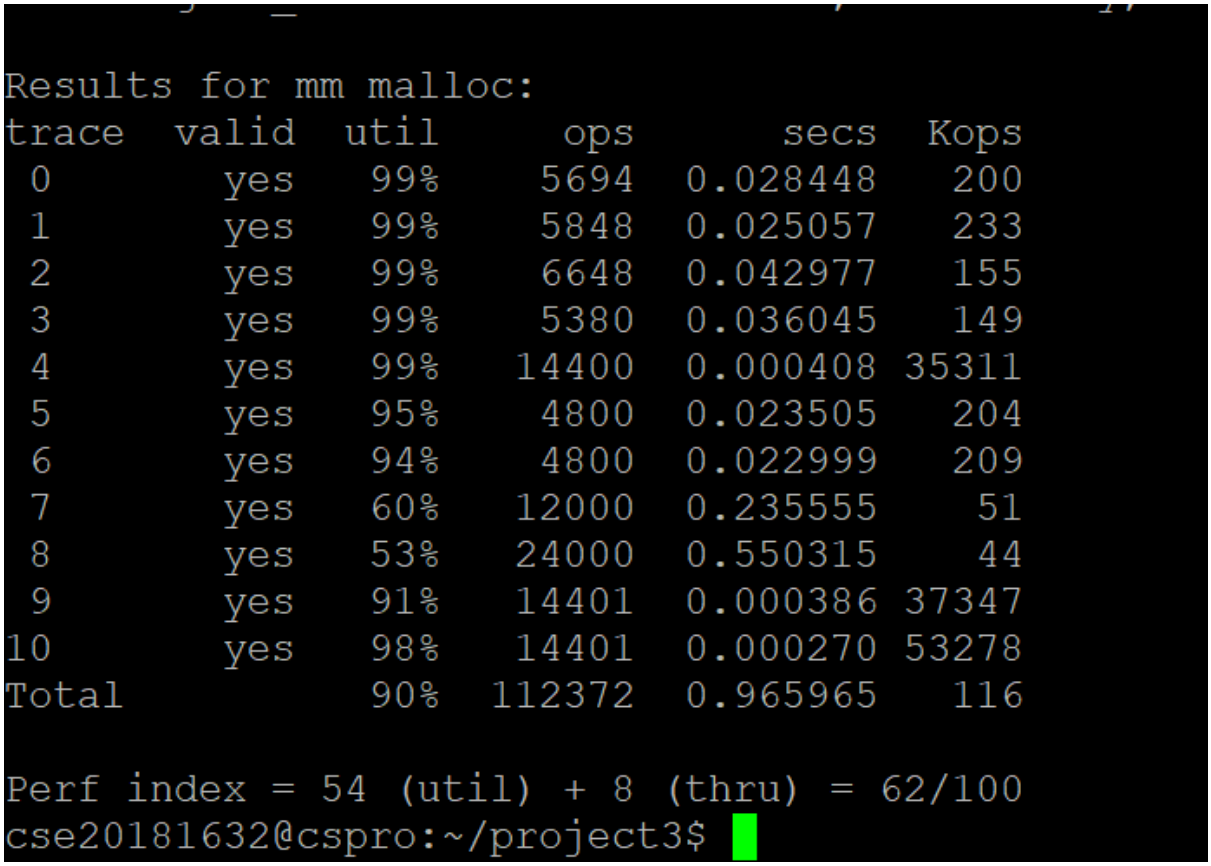
정에서 block들이 합쳐져 seg_list에 알맞은 index에 들어가게 된다. Realloc의 경우 ptr이 NULL이면 malloc을 호출한다. 다른 경우에는 realloc하는 size를 확인하고 그 사이즈에 따라 ptr을 그대로 반환하거나 다음 block과 합친다. 합치지 못할 경우 malloc을 호출해 새로 할당해준다.

3. 수행 결과

```
Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   99%    5694  0.000616  9245
 1       yes   99%    5848  0.000591  9902
 2       yes   99%    6648  0.000582 11423
 3       yes   99%    5380  0.000466 11553
 4       yes   99%   14400  0.000507 28397
 5       yes   95%    4800  0.001512  3175
 6       yes   94%    4800  0.001261  3808
 7       yes   60%   12000  0.000655 18321
 8       yes   53%   24000  0.006402  3749
 9       yes   91%   14401  0.000251 57466
10       yes   98%   14401  0.000254 56719
Total          90%  112372  0.013095  8581

Perf index = 54 (util) + 40 (thru) = 94/100
cse20181632@cspro:~/project3$
```

위 결과는 mm_check를 실행하지 않고 실행한 결과이다. Best-fit search이기 때문에 memory utilization이 상당히 높다. 하지만 그만큼 seg_list를 탐색하는데 시간이 걸리므로 throughput이 좋은 결과를 내진 않는다. 만약 throughput을 더 높이고 싶다면 best-fit이 아닌 first-fit이나 next-fit으로 수행하면 될 것 같다. 이 두 방법은 list의 끝까지 탐색하지 않고 가장 먼저 나오는 free block에 할당시켜주기때문에 시간 관점에서는 더 좋을 수 있다. 하지만 할당하는 메모리의 크기가 다양할 경우, 메모리 관점에서 매우 좋지 않은 성능을 낼 것이다.



다음 결과는 mm_check를 mm_free함수의 마지막에 넣고 돌린 결과이다. 에러메세지는 뜨지 않았으므로 모든 block들이 잘 할당되어 있다는 것을 알 수 있다. 하지만 free 실행시마다 heap전체를 체크하므로 시간이 많이 소요되고, 그만큼 throughput이 감소한다.