System Programming Project 4

담당 교수 : 김영재 교수님

이름 : 이 상 연

학번 : 20201617

1. 개발 목표

본 프로젝트에서는 C 프로그램의 heap 영역에서 동적 할당을 수행할 수 있는 Dynamic Memory Allocator를 구현한다. 구현하는 Memory Allocator는 기존 C 언어의 표준 라이브러리 libc의 malloc, realloc, free와 유사한 역할을 수행한다. Implicit list, Explicit list, Segregated free list 세 가지 Method를 직접 적용해 보면서 더 효율적으로 동적 메모리 할당을 수행할 수 있는 방법을 찾고 각 방식의 메모리 동적 할당이 어떻게 이루어지는지를 이해하는 것이 본 프로젝트의 목표이다.

본 프로젝트에서는 직접 구현하는 mm_malloc, mm_realloc, mm_free 함수가 제대로 수행되도록 하기 위해서 사용할 heap 메모리 영역을 초기화하는 mm_init 함수, heap의 사용 가능한 영역이 가득 찬 경우에 sbrk system call을 통해서 brk 포인터를 늘려주는 extend_heap 함수를 구현한다. 해당하는 주소에 메모리를 할당하는 place 함수와, 할당하는 메모리 공간의 앞 뒤에 비어 있는 메모리가 존재하면 합쳐서 하나의 빈 공간으로 만드는 coalesce 함수도 추가로 구현하도록 한다.

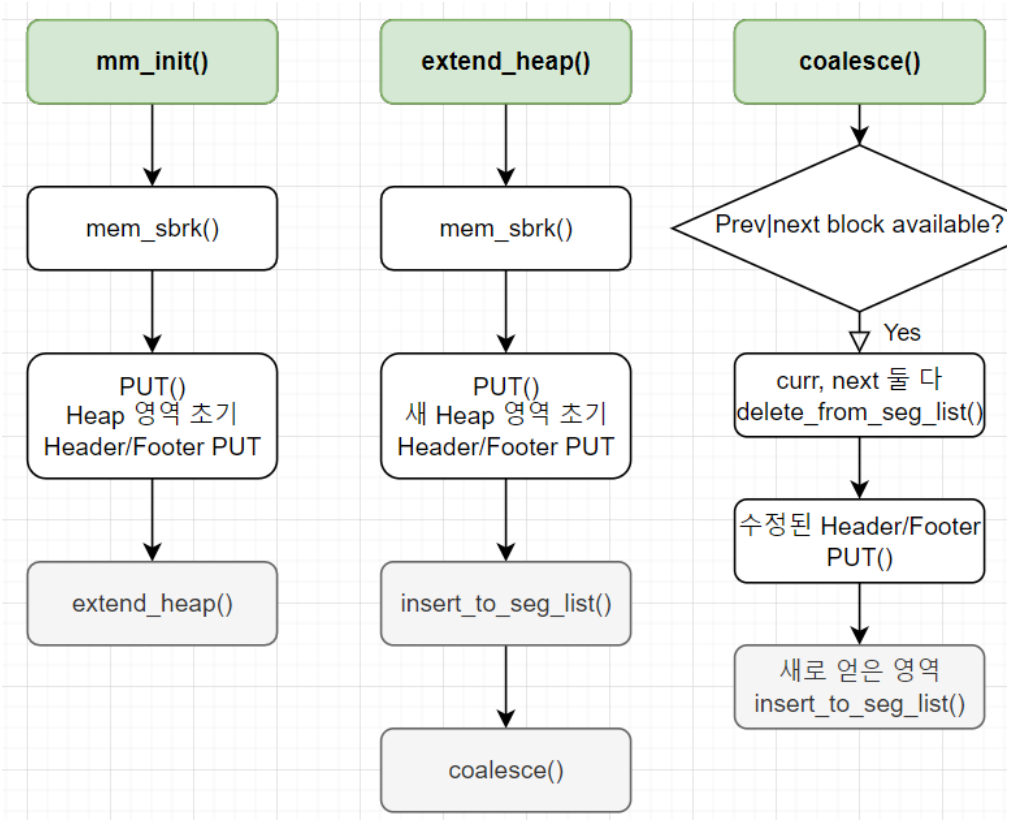
또한, 구현하는 method에 따라서 필요한 동작을 수행하는 함수도 구현한다. 본 프로젝트에서는 Segregated list 방식으로 구현했기 때문에, Segregated list에 할당 가능한 공간의 pointer를 연결하는 insert_node 함수, 그리고 데이터를 할당하면 Segregated list에서 해당 entry를 삭제하는 delete_node 함수를 구현한다.

2. 개발 범위 및 내용

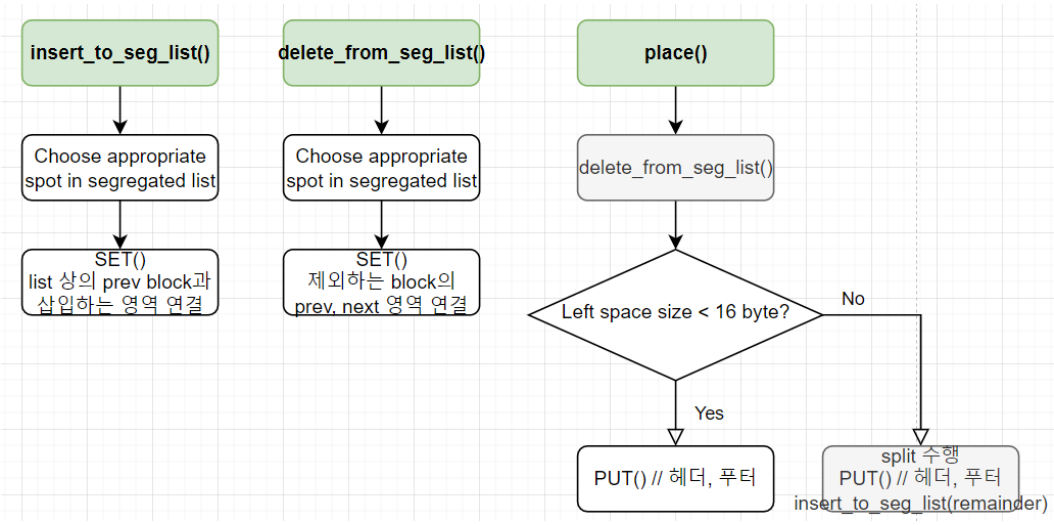
Segregated list method를 활용해서 직접 구현한 Dynamic allocator를 구성하는, 9가지의 서브루틴에 대한 동작의 flowchart는 각각 다음과 같다.

각 서브루틴 별 동작은 다음 항목에서 서술하도록 한다.

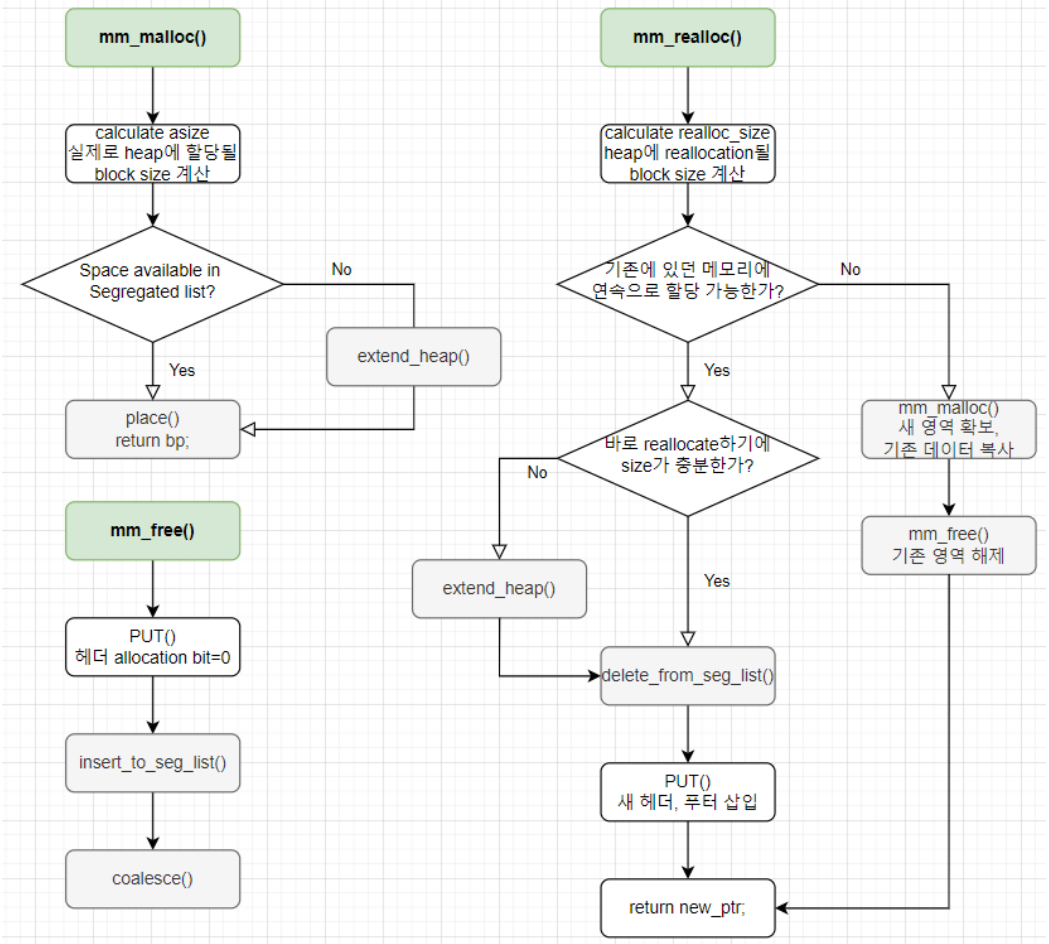
- Flowchart for mm_init(), extend_heap(), coalesce()



- Flowchart for insert_node(), delete_node(), place()



- Flowchart for mm_alloc(), mm_free(), mm_realloc()



본 프로젝트를 수행하면서 구조체는 명시된 인적사항을 기재하는 team struct만 사용했다. 전역변수는 아래와 같이 segregated free list 를 저장하기 위한 포인터 변수만을 선언해서 사용했다.

```
/*********************************************************
 * NOTE TO STUDENTS: Before you do anything else, please
 * provide your information in the following struct.
 *********************************************************/
team_t team = {
    /* Your student ID */
    "20201617",
    /* Your full name*/
    "Sangyeon Lee",
    /* Your email address */
    "sangyeone0163@gmail.com",
};
```

```
/* Global variables */
static char **segregated_free_lists; /* Array of segregated free lists */
```

사용한 전역변수는 character 형 double pointer 변수로, Free Space들을 크기별로 관리하는 segregated list의 시작 주소지를 가리키는 segregated_free_lists 변수이다. 아래에서 소개하는 insert_node, delete_node 함수를 통해서 segregated list를 조작하여 메모리 동적 할당을 수행한다.

프로젝트를 수행하며 구현한 서브루틴, 즉 C 함수에 대한 역할과 기능, 구현 방법에 대한 설명은 아래와 같다.

mm_init()

사용할 heap 공간에 대한 초기화를 수행한다. mem_sbrk() 함수를 호출해서 사용할 공간의 크기만큼 heap 영역의 pointer를 확장하고, 8byte alignment를 맞춰 주기 위한 가장 앞의 block과 프롤로그, 에필로그 header 및 footer를 삽입한다. 이후 extend_heap 함수를 호출해서 heap의 크기를 매크로로 정한 CHUNK의 사이즈만큼 확장시킨다.

extend_heap()

Heap 공간을 얼마만큼 확장할 지 word 단위로 입력을 받은 뒤, mem_sbrk() 함수를 호출해서 brk pointer 를 확장한다. 확장한 영역의 가장 앞과 뒤에 Header, Footer Block을 삽입하고, insert_node() 함수를 호출해서 새로 할당된 영역의 시작 포인터를 segregated list에 삽입한다.

이후 새로 할당한 메모리 공간에 대해서, 앞의 block이 allocated 되지 않은 free block이었을 가능성이 있기 때문에 coalesce()를 호출한다.

coalesce()

할당하는 메모리 영역의 앞과 뒤 block에 allocated 되지 않은 free block이 있다면 합치는 역할을 수행하는 함수이다. 합치지 않았을 경우, 나누어져 있는 block의 크기보다 큰 block에 대한 할당 요청이 온다면 공간이 남지만 그 공간에 할당하지 못하기 때문에, utilization 측면의 개선을 위해서 coalesce() 함수를 호출한다.

함수 내에서는, previous block과 next block의 allocation 여부를 파악해서, 어느 위치에 있는 인접한 block이 free 되어 있는지를 판단하고 존재한다면 병합한다.

place()

place() 함수는 할당할 메모리를 해당하는 영역에 배치하는 동작을 하는 함수이다. Place 함수를 호출하기 전에, 메모리를 heap 영역에 할당하기 위해서 필요한 block 개수를 미리 계산하고 파라미터로 넘긴다. 함수 내부에서는 delete_node() 함수를 호출해서 segregated list 상에서 사용할 메모리 영역만큼을 빼고, 배치하는 메모리 공간의 header/footer를 해당하는 값으로 채운다. 필요한 영역보다 사용하는 공간이 크다면, split을 수행해서 header와 footer에 적절한 값을 채운다.

insert_node()

Segregated list를 관리하며 사용 가능한 Free space가 생기면 해당 공간의 시작 주소 포인터를 Segregated list에 삽입하는 동작을 하는 함수이다. 구현한 Segregated list는 1칸, 2칸, 4칸 … 과 같은 순서로 2의 거듭제곱 단위 크기로 할당되고, insert_node() 함수에서는 파라미터로 받은 free space의 크기를 Segregated list 크기 기준에 맞춰서 어느 index에 삽입할지, 그리고 구체적으로 그 리스트 안에서 어느 지점에 삽입할지를 결정한다. 이를 바탕으로 free space를 가리키는 포인터를 segregated list 상에 삽입한다.

delete_node()

insert_node() 함수와는 반대로, 메모리 공간에 할당할 데이터가 있을 경우에 segregated list에서 사용할 free space를 선택하고, 해당 공간을 list로부터 삭제하는 역할을 한다. 코드 구성은 insert_node()와 유사하고, size에 따라서 필요한 index를 판단해서 해당하는 index의 가장 첫 번째 free space를 segregated list에서 제외한다. 이후에, SET 매크로를 이용해서 segregated list에서 관리하는 free space 들의 NEXT_PTR, PREV_PTR Block들을 해당하는 메모리 영역과 이어서 segregated list를 이후에도 계속 사용할 수 있도록 한다.

mm_malloc()

lib 라이브러리의 malloc 처럼, heap에서 사용하고자 하는 크기만큼의 메모리를 할당받는 역할을 하는 함수이다. 메모리의 크기를 size로 입력을 받고, 실제로 heap 영역에 할당될 block 크기를 결정한다. 이 크기는 Header, Footer, 8 byte alignment를 만족하는 크기를 고렇나다. 이후에 segregated list에서 충분한 크기의 free space가 있는지를 탐색하고 만

약 있는 경우에는 해당 영역의 포인터의 Header, Footer를 수정해서 리턴한다.

만약 해당하는 크기만큼의 메모리가 존재하지 않으면 extend_hep() 를 호출해서 heap 공간의 크기를 확장하고 해당 영역을 리턴한다.

mm_realloc()

이미 할당한 메모리 영역의 크기를 수정해 reallocation하는 경우에 사용하는 함수이다. 파라미터로 수정할 메모리의 포인터를 넘겨받아서 heap 영역에 배치할 수정된 메모리 영역의 block 수를 계산한다. 현재 위치한 메모리 영역에서 연속해서 더 큰 메모리 영역을 사용할 수 없는 경우에는, mm_malloc() 함수를 호출해서 새로 수정된 크기만큼의 메모리 영역을 찾아 할당받는다. 이전에 사용하던 메모리 공간을 free 하고 리턴한다.

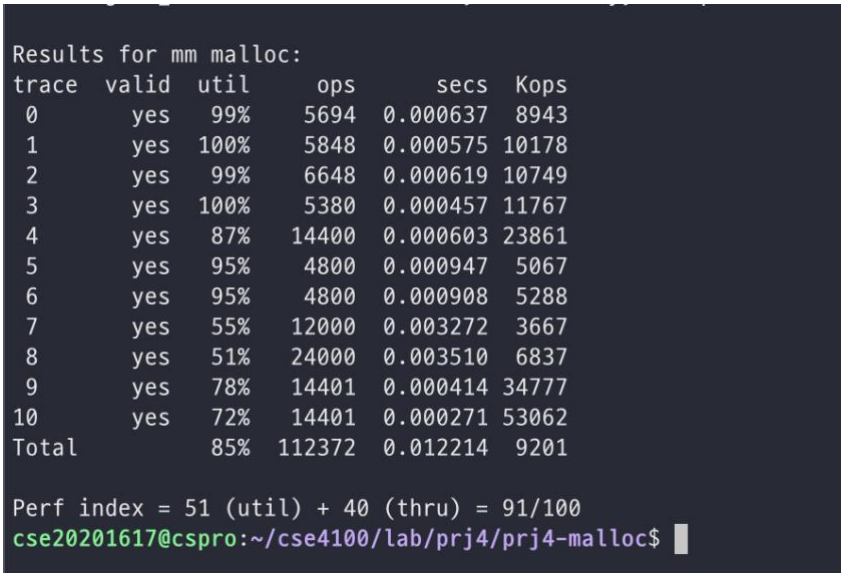
만약 다음 Block의 size와 allocation 여부를 모두 고려했을 때 연속된 메모리 공간에서 사용할 수 있다면, 해당 Block에 연속해서 배치하고, 만약 메모리 영역의 크기가 모자라다면 extend_heap()를 호출한다. extend_heap() 함수 내부에서 segregated list에 새로운 free space를 삽입했기 때문에, 이 경우에는 곧바로 delete_node() 함수를 호출해서 확장된 heap 영역에 block을 삽입한다.

mm_free()

위에서 구현한 mm_malloc(), mm_realloc() 함수를 통해서 동적 할당한 메모리를 해제해주는 역할을 하는 함수이다. 호출된 포인터가 속한 메모리 영역의 Header, Footer에서 allocation bit을 0으로 만들고, 해당 영역을 segregated list에 삽입한다. 메모리 해제 후에 앞 뒤 메모리 영역이 비어 있다면 coalesce를 수행하도록 한다.

3. 구현 결과

결론적으로 91점의 퍼포먼스 점수에 도달할 수 있었다. 다른 테스트 케이스에서는 모두 높은 utilization을 기록했지만, binary-bal 테스트 케이스에서 50퍼센트 대의 낮은 utilization이 나왔다.


Results for mm malloc:
trace valid util ops secs Kops
0 yes 99% 5694 0.000637 8943
1 yes 100% 5848 0.000575 10178
2 yes 99% 6648 0.000619 10749
3 yes 100% 5380 0.000457 11767
4 yes 87% 14400 0.000603 23861
5 yes 95% 4800 0.000947 5067
6 yes 95% 4800 0.000908 5288
7 yes 55% 12000 0.003272 3667
8 yes 51% 24000 0.003510 6837
9 yes 78% 14401 0.000414 34777
10 yes 72% 14401 0.000271 53062
Total 85% 112372 0.012214 9201
Perf index = 51 (util) + 40 (thru) = 91/100
cse20201617@cspro:~/cse4100/lab/prj4/prj4-malloc$