# System Programming Project 3

담당 교수 : 김 영 재

이름 : 이 호 성

학번 : 20171680

1. **개발 목표**

기존 stdlib.h의 malloc 함수를 직접 구현해보며 메모리 할당과 관리 방식에 대한 이해를 높인다.

2. **개발 범위 및 내용**

**A. 개발 범위**

malloc() , free(), realloc()을 직접 구현하였다.

**B. 개발 내용**

해당 함수들을 구현하기 위해 free block을 관리하는 method 들 중 explicit list 방법을 활용하여 개발하였다. mm_init() 함수로 맨 처음 heap을 초기화하고 그 다음 들어오는 다양한 조합의 mm_malloc(), mm_free(), mm_realloc() 요청들을 처리하는 구조이다.

**C. 개발 방법**

- **새로 추가한 주요 MACRO 코드**

  SET_PTR(p,ptr) : 포인터(p) 에 주소의 정보(ptr)를 할당 한다.

  NEXT_PTR(ptr) / PREV_PTR(ptr) : heap 영역에서 ptr 이전/이후 포인터의 값을 구한다.

  LIST_NEXT(ptr) / LIST_PREV(ptr) : free list에서 ptr 이전/이후 에 연결된 주소의 값을 구한다.

- **새로 추가한 주요 자료구조**

  void *exp_listp : free list를 포인터이다. linked list로 free list를 저장한다.

- **새로 추가한 주요 함수**

- **mm_init()**을 통해 첫 heap 영역을 할당한다. 안전한 heap 의 사용을 위해 prologue 와 epilogue block 을 생성한다. 그 후 초기 heap 영역을 extend_heap() 함수를 통해 할당 받는다.

- **extend_heap(size)**의 경우 요청 받은 size 를 mem_sbrk()를 통해 할당 받는다. 이후 block 의 header/footer 에 size 정보를 할당하고 insert_node()함수를 통해 free list 에 추가한다. 그 다음 coalesce()를 통해 list 에 새로이 추가한 block 에 대하여 주소적으로 인접한 block 들을 검사하여 free block 들을 병합한다.

- **coalesce(bp)**는 free block 이 생성되는 경우 호출이 되는 함수로 주소적으로 인접한 block 들이 free 일 경우 병합을 하는 함수이다. 앞/뒤 블록의 allocated 여부를 확인한 다음 4 가지 경우 (앞/뒤 * free/allocated)에 따라 block 병합을 진행한다.

- **insert_node(ptr,size)**의 경우 빠른 free block 할당을 위해 block 크기의 오름차순으로 정렬하였다. free list 내부에서 크기에 맞는 적절한 위치를 찾게 되면 그 위치에 따라 (맨앞,맨뒤,중간,유일) NEXT_PTR 과 PREV_PTR 를 할당 해준다. list 의 맨앞/유일 노드를 추가하는 경우 exp_listp 가 가리키는 노드를 바꾸어 준다.

- **delete_node(ptr)**의 경우 insert_node()와 동일하게 4 가지 경우에 따라 free list 에서 해당 노드를 삭제하는 작업을 한다. list 의 맨앞/유일 노드를 삭제하는 경우 exp_listp 가 가리키는 노드를 바꾸어 준다.

- **mm_malloc(size)**의 경우 size 만큼 heap 영역을 할당 해준다. 들어온 size 를 align 을 해준 뒤 정렬되어 있는 free list 를 탐색하며 적절한 크기의 free block 를 찾는다. 만약 찾지 못했다면 extend_heap()을 통해 heap 영역 확장을 요청하여 받은 다음 place() 함수를 통해 free block 에서 size 만큼만 사용을 하고 나머지 부분을 free list 에 추가한다.

- **place(bp,asize)**의 경우 free block 에 새로운 정보를 할당할 때 사용된다. 할당하고 남은 부분을 free list 에 추가하게 되는데 이때 성능 향상을 위해 할당한 영역의 크기가 100 을 넘어갈 경우 block 의 뒷부분에 할당하고 앞부분을 free block 으로 남겨두고 영역의 크기가 100 보다 작을 경우 앞부분에 할당하고 뒷부분을 free block 으로 남겨둔다. 남는 부분이 없을 경우 free list 에 추가하지 않는다.

- **mm_free(ptr) :** ptr 의 header 와 footer 의 내용을 초기화 하고 free list 에 insert_node()를 통해 추가해준다. 그 이후 coalescing 을 통해 해당 ptr 과 인접한 free block 들을 병합한다.

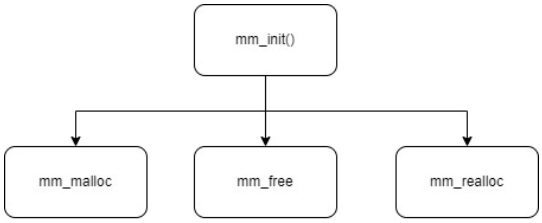
- **mm_realloc(ptr,size)** : size 의 크기로 heap 영역을 재할당 한다. 우선 크기를 align 한 다음 REALLOC_BUFFER 를 더한다. 만약 현재 포인터의 크기가 이 사이즈보다 작다면 realloc 을 해준다. 만약 기존의 NEXT 블록이 allocated 되어 있다면 새로운 영역을 mm_malloc 을 통해 구한 다음 memcpy 를 통해 내용을 복사한 후 mm_free 를 통해 기존의 영역을 free 시킨다. 만약 기존의 다음 영역이 free 영역이거나 epilogue 부분일 경우 기존의 block 에서 크기만 증가시켜주면 되기 때문에 영역이 모자란 경우 extend_heap()까지 활용하여 기존의 영역에 덧붙인다.

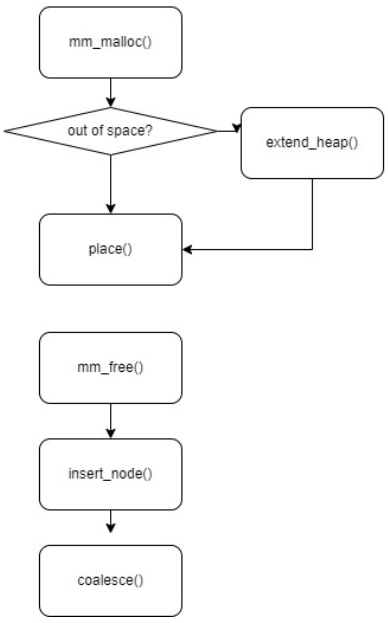
- **mm_checker()**

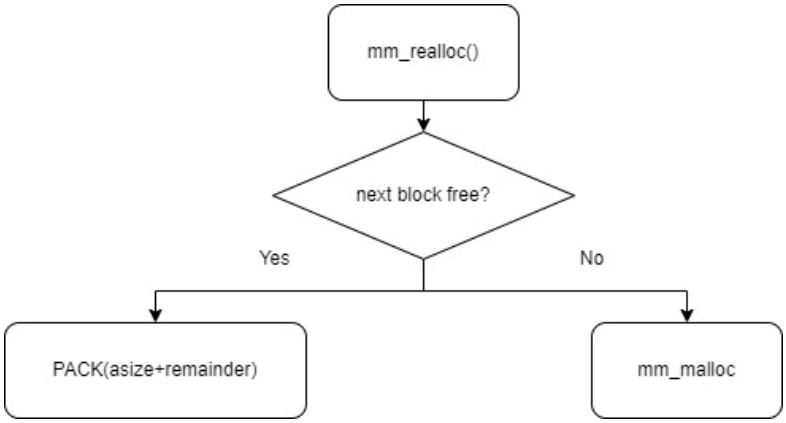
  #define MM_CHECKING 을 활용 하여 MM_CHCKING 이 정의되어 있는 경우만 mm_checker 함수를 호출 할 수 있도록 하였다.

  free list 를 순회하며 free block 이 아닌 node 가 있는지 확인해 보았으며 조건문에 걸리지 않고 정상적으로 작동하였다.

**Flow Chart**






3. **구현 결과**

```
Reading tracefile: binary2-bal.rep
Checking mm_malloc for correctness, efficiency, and performance.
Reading tracefile: realloc-bal.rep
Checking mm_malloc for correctness, efficiency, and performance.
Reading tracefile: realloc2-bal.rep
Checking mm_malloc for correctness, efficiency, and performance.

Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   99%    5694  0.000737  7728
 1       yes   99%    5848  0.000450 13007
 2       yes   99%    6648  0.000896  7417
 3       yes   99%    5380  0.000533 10090
 4       yes   99%   14400  0.000288 49931
 5       yes   95%    4800  0.011634   413
 6       yes   95%    4800  0.010702   449
 7       yes   95%   12000  0.021691   553
 8       yes   88%   24000  0.019796  1212
 9       yes   99%   14401  0.000143100706
10       yes   98%   14401  0.000096150638
Total          97%  112372  0.066966  1678

Perf index = 58 (util) + 40 (thru) = 98/100
cse20171680@cspro:~/prj3-malloc$
```