System Programming Project 3

담당 교수 : 김영재 교수님

이름 : 홍주표

학번 : 20181702

1. 개발 목표

나만의 Dynamic Memory Allocator를 만드는 것이 이번 프로젝트의 목표이다. 다시 말해, 기존에 사용하던 malloc, realloc, free 함수와 같은 기능을 하는 나만의 함수를 만드는 것이다.

학기 중에 배운 Implicit Free List, Explicit Free List, Segregated Free List 등 여러 종류의 방법 중 가장 효율적으로(correct, efficient, fast) 좋은 방법을 택하여 개발한다.

2. 함수 설명

- 대부분의 매크로 및 함수의 기본적인 틀은 주교재를 참고하여 작성한다.
- 전역 변수로는 char *heap_listp와 char **seg_list 만을 사용한다.

■ int mm_init(void) (교재 참고)

mm_malloc, mm_realloc, mm_free를 호출하기 전에 mm_init 함수를 호출함으로써 heap을 초기화해준다.

memory system에서 4 words 만큼 받은 다음 empty free list를 만들어 초기화한다.

그리고 extend_heap 함수를 호출하여 heap을 CHUNKSIZE만큼 확장하고 initial free block을 생성한다. 이 때 allocator는 초기화되고 allocate나 free 상태를 받아들이게 된다.

위 과정에서 오류가 있는 경우 -1을, 그렇지 않으면 0을 return한다.

■ void *mm_malloc(size_t size) (교재 참고)

brk_pointer를 증가시키면서 block을 할당한다.

할당된 block의 size는 항상 ALIGNMENT의 배수이다.

allocated block payload의 pointer를 return한다.

할당된 전체 block은 heap 영역 내에 있어야한다.

■ void mm_free(void *bp) (교재 참고)

bp가 가리키고 있는 block pointer를 free한다.

- void *mm_realloc(void *bp, size_t size) (기존 코드 활용)

명세에 써있듯이, realloc을 실행하기 전에 bp가 NULL인 경우 mm_malloc을, size가 0인 경우 mm_free를 실행한다.

newptr을 mm_malloc을 활용하여 공간을 할당하고 memcpy를 활용하여 oldptr(bp와 동일)의 메모리를 복사한다. 이 때 그 size를 copySize로 하여 memcpy 함수를 사용한다.

- static void *extend_heap(size_t words) (교재 참고)
- static void *coalesce(void *bp) (교재 활용)

교재에 있는 Case 1~4를 활용한다.

Case 1: Prev allocated, Next allocated

insert_block을 실행하고 bp를 return한다.

Case 2: Prev allocated, Next free

next block에 대해 remove_block을 실행한다.

Case 3: Prev free, Next Allocated

prev block에 대해 remove block을 실행한다.

Case 4: Prev free, Next free

next block과 prev block 모두에 대해 remove_block을 실행한다.

Case 3과 4가 Case 1과 2와의 다른 점은 앞의 block이 free이기 때문에 block pointer 또한 변경해준다는 점이다.

Case 2, 3, 4가 끝난 후에는 bp에 대해 insert_block을 실행하고 bp를 return한다.

간단히 정리하자면 앞이나 뒤 block에 free 상태의 block이 있다면 해당 block을 remove하고, size를 다시 설정한 뒤, 앞에 free 상태의 block이 있다면 bp도 재설정 한 후 insert를 실행하는 것이다.

- static void *find_fit(size_t asize) (교재 활용)
- static void place(void *bpm size_t asize) (교재 참고)
- static void insert_block(void *bp, size_t size)

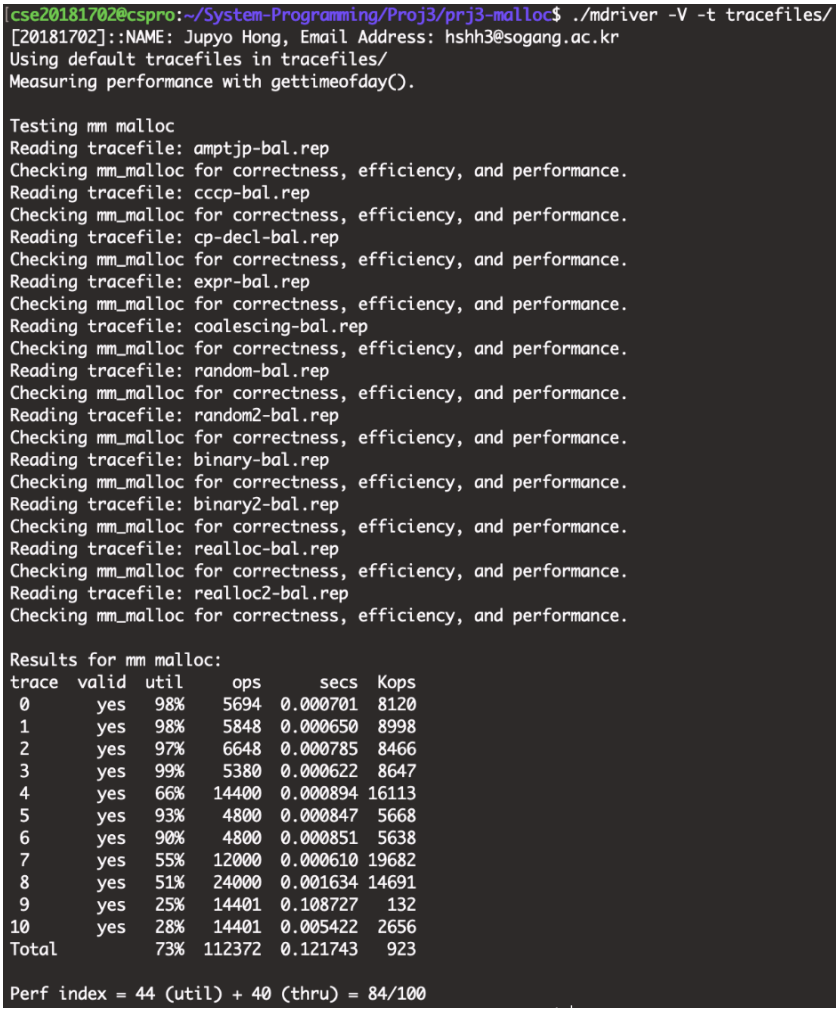
free 상태의 block을 seg_list에 삽입한다. 이 때 search_ptr과 insert_ptr이라는 void pointer 변수를 사용하는데, 각 변수의 NULL 상태 여부에 따라 Case를 나누어 새로운 block의 predecessor와 successor를 설정한다.

- static void remove_block(void *bp)

insert_block 함수와 유사하게 SUCC_FREE(bp)와 PRED_FREE(bp)의 NULL 상태 여부에 따라 Case를 나누어 block pointer를 bp로 가지는 block을 remove한다.

3. 구현 결과

./mdriver -V -t tracefiles를 실행했을 때 다음과 같은 결과를 얻을 수 있다.



```
cse20181702@cspro:~/System-Programming/Proj3/prj3-malloc$ ./mdriver -V -t tracefiles/
[20181702]::NAME: Jupyo Hong, Email Address: hshh3@sogang.ac.kr
Using default tracefiles in tracefiles/
Measuring performance with gettimeofday().

Testing mm malloc
Reading tracefile: amptjp-bal.rep
Checking mm_malloc for correctness, efficiency, and performance.
Reading tracefile: cccp-bal.rep
Checking mm_malloc for correctness, efficiency, and performance.
Reading tracefile: cp-decl-bal.rep
Checking mm_malloc for correctness, efficiency, and performance.
Reading tracefile: expr-bal.rep
Checking mm_malloc for correctness, efficiency, and performance.
Reading tracefile: coalescing-bal.rep
Checking mm_malloc for correctness, efficiency, and performance.
Reading tracefile: random-bal.rep
Checking mm_malloc for correctness, efficiency, and performance.
Reading tracefile: random2-bal.rep
Checking mm_malloc for correctness, efficiency, and performance.
Reading tracefile: binary-bal.rep
Checking mm_malloc for correctness, efficiency, and performance.
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   98%    5694  0.000701  8120
 1       yes   98%    5848  0.000650  8998
 2       yes   97%    6648  0.000785  8466
 3       yes   99%    5380  0.000622  8647
 4       yes   66%   14400  0.000894 16113
 5       yes   93%    4800  0.000847  5668
 6       yes   90%    4800  0.000851  5638
 7       yes   55%   12000  0.000610 19682
 8       yes   51%   24000  0.001634 14691
 9       yes   25%   14401  0.108727   132
10       yes   28%   14401  0.005422  2656
Total          73%  112372  0.121743   923

Perf index = 44 (util) + 40 (thru) = 84/100
```