Multicore Programming Project 3

담당 교수 : 최재성 교수님

이름 : 임해진

학번 : 20180223

1. 개발 목표

본 프로젝트에서는 Dynamic Memory Allocator를 구현한다. 사용자의 요청에 따라서, 힙 공간의 메모리를 할당해주고, 사용하지 않는 메모리를 해제해주는 함수를 구현한다.

2. 개발 범위 및 내용

A. 개발 범위

mm_init(), mm_malloc(), mm_free(), mm_realloc() 함수를 구현하여, 좋은 성능의 Dynamic Memory Allocator를 구현한다

B. 개발 내용

메모리를 할당할 때는, 모든 free memory를 탐색하여 가장 적절한 공간에 할당해주는 best fit policy를 구현할 것이다. free memory를 doubly linked list로 관리하는 explicit free list방법을 사용해서, 빠르게 free memory를 탐색할 수 있도록 한다. realloc시에는, 기존의 메모리를 그대로 사용할 수 있는지 확인해서 데이터를 복사하지 않고 재할당해준다.

C. 개발 방법

[malloc()]

요청된 메모리 사이즈를, alignment에 맞도록 조절한 후에, free memory list를 순회해서 best fit 블럭을 찾는다. free memory가 있다면, 해당 메모리 공간을 할당한다. 없다면, 힙 공간을 추가 요청해서 해당 공간에 할당한다.

find_fit()함수를 구현해서, best fit 블럭을 찾는 로직을 구현했다. free list를 순회하면서, min_remain 블럭(size보다 큰 메모리 중 min size인 것), min_16_remain 블럭(size+16보다 큰 메모리 중 min size인 것), max_16_remain블럭(size+16보다 큰 메모리 중 max size인 것)을 찾는다. 이후 min_remain블럭이 요청된 사이즈+WSIZE보다 크기가 작거나 같으면, 해당 블럭을 사용한다. 이후, max_16_remain 블럭이 요청한 사이즈*3 보다 크기가 크면 해당 블럭을 사용한다. 그리고 min_16_remain, min_remain 순으로 블럭을 사용한다. 만약 가능한 free memory가 없다면 NULL을 리턴해, free memory가 없음을 알린다.

extend_heap()함수를 사용해서 힙 공간을 추가하는 로직을 구현했다. 함수에서는 힙 공간을 할당하는 시스템 콜인 mem_sbrk()를 호출해, 공간을 마련하고 prev 블럭이 free라면 합친 후에 이 공간을 free memory list에 추가한다

place()함수를 이용해서 메모리를 할당하는 로직을 구현했다. 함수에서는 먼저 free list에서 블럭을 제거하고, split해야하는지 확인한다. split해야하면, 남는 블럭의 헤더와 푸터 값을 업데이트하고 free list에 추가하고, 사용할 공간을 allocated로 업데이트 한다. split하지 않아도 되면 allocated로 업데이트 하고 리턴한다.

[free()]

만약 요청한 포인터가 널값이면 리턴한다. 아니라면 allocated bit을 0으로 표시한 후에, 양옆 블럭과 합친 후 free memory list에 추가한다.

coalesce()함수를 사용해서 양옆 블럭과 합치는 로직을 구현했다. prev, next 블럭의 할당 여부를 사용해 4개의 case로 분류했다. case1은 둘 다 할당되어 있는 경우로, 합치지 않고 그냥 리턴된다. case2는 next 블럭이 free인 경우로, next 블럭을 free list에서 삭제하고, 현재 블럭의 사이즈에 next 블럭 사이즈를 더해준다. case3은 prev 블럭만 free인 경우로, prev 블럭을 free list에서 삭제하고, prev 블럭의 사이즈에 현재 블럭 사이즈를 더해준다. case4는 둘 다 free인 상태로, 두 블럭 모두 free list에서 삭제하고, prev 블럭의 사이즈에 현재 블럭 사이즈와 next 블럭 사이즈를 더해준다

[realloc()]

요청한 사이즈가 0이라면, free()를 호출한다. 요청한 포인터가 널값이면 malloc()을 해준다. 요청된 메모리 사이즈를, alignment에 맞도록 조절한 후에 현재 메모리 블럭 사이즈와 비교한다. 현재 메모리 블럭 사이즈가 크거나 같아면 그대로 메모리를 사용할 수 있는 것이기 때문에, 그대로 사용한다. 만약 다음 메모리 블럭이 free라면, 두 메모리를 합쳐서 realloc할 수 있는지 확인한다. 가능하다면, next 블럭을 free list에서 삭제하고 두 블럭을 합친 후에 할당한다. 가능하지 않다면 malloc을 호출해 새 공간을 할당하고, 기존 데이터를 복사한 후에 기존의 블럭을 free한다.

[free list 관리]

free list의 헤더는 전역변수 free_listp를 사용해서 관리한다.

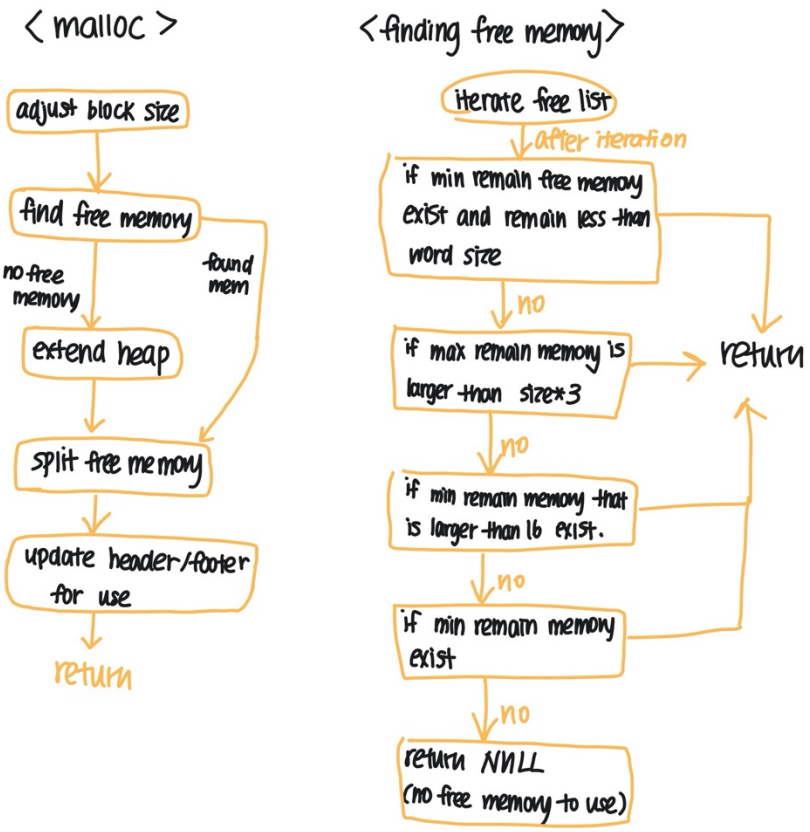
push_free_list()함수로 free list에 블럭을 추가하는 로직을 구현했다. 제일 앞에 블럭을 추가해준 후에, 포인터 값을 수정해준다.

remove_free_list()함수로 free list에서 블럭을 제거하는 로직을 구현했다. 블럭을 제거한 후에 양옆 블럭의 포인터 값을 업데이트 해준다.

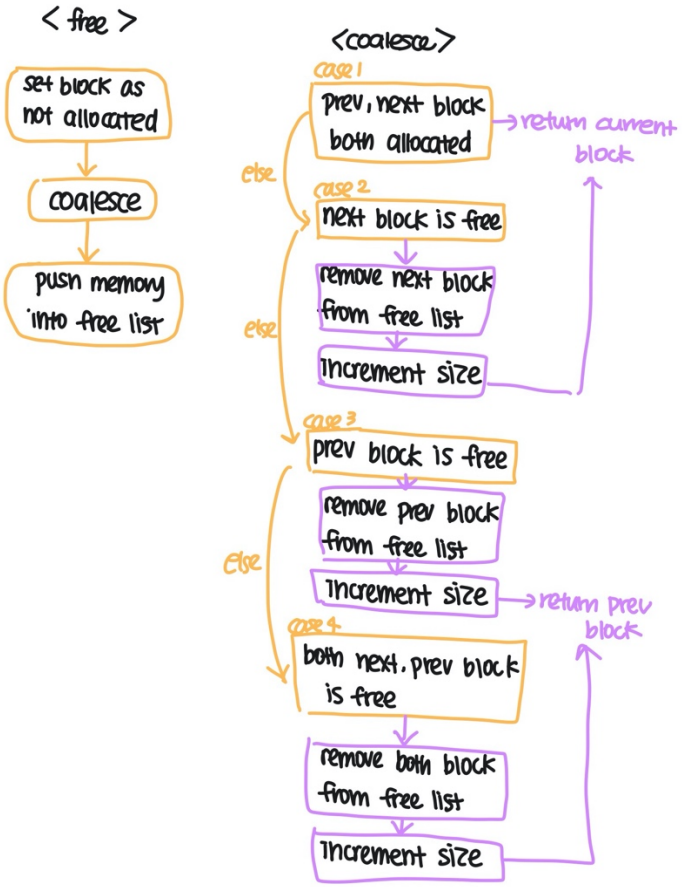
3. 구현 결과

init, malloc, free, realloc 함수와, 각종 sub routine, free memory list 조작 함수를 사용해서 성공적으로 dynamic memory allocator를 구현했다.

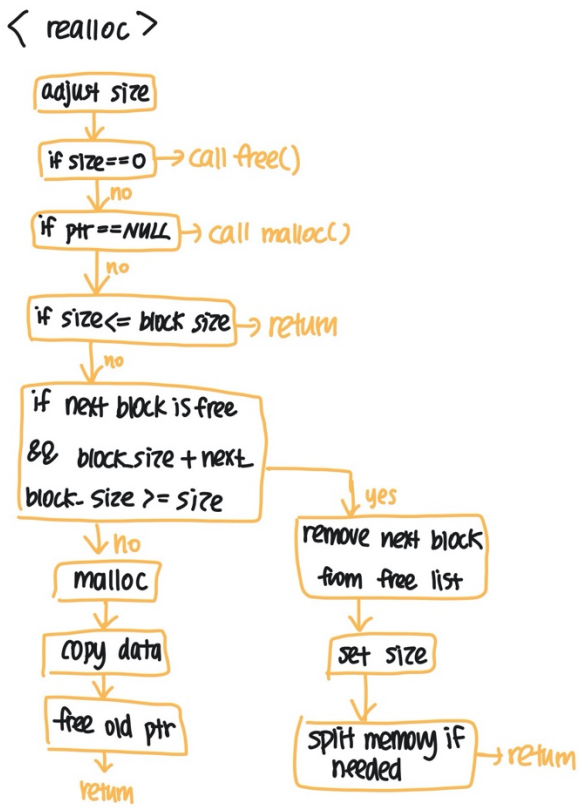
<malloc()의 flow chart>



< free의 flow char>



< realloc의 flow char>



4. 성능 평가 결과

mdriver -V를 사용해서 memory allocator의 정확도와 성능을 측정하였다. 그 결과 정확성 점수는 만점, performance index는 91/100이 나왔다.

```
Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   96%    5694  0.000635  8973
 1       yes   95%    5848  0.000841  6954
 2       yes   97%    6648  0.000889  7476
 3       yes   97%    5380  0.000679  7929
 4       yes  100%   14400  0.000393 36651
 5       yes   92%    4800  0.004747  1011
 6       yes   89%    4800  0.004977   964
 7       yes   55%   12000  0.032838   365
 8       yes   51%   24000  0.091883   261
 9       yes   80%   14401  0.000326 44202
10       yes   86%   14401  0.000214 67357
Total          85%  112372  0.138420   812

Perf index = 51 (util) + 40 (thru) = 91/100
```