Multicore-Programming Project 3

20191611
유종선

1. 함수 설명
   A. Malloc
      i. 입력으로는 우리가 할당할 크기를 받아준다.
      ii. 사이즈가 2WORDS 즉, 8byte가 되지 않는 것은 최소 16byte로 할당 요청을 한다.
      iii. Find-fit 함수로 할당할 수 있는 공간이 들어갈 수 있는 공간을 찾는다. 찾았다면 그 부분을 분할하고, 새롭게 할당한다.
      iv. 새롭게 할당 할 수 있는 공간이 없을 수 있다. 그런 경우 우리는 extend_heap 함수를 호출하여 mem_sbrk를 사용하여 heap의 공간을 늘려줄 것을 요청한다. Place 함수는 할당한 공간의 위치를 조정해주는 도움을 주는 함수이다.
      v. 공간이 늘어났다면 늘어난 공간에 메모리를 할당하고 return 한다.
   B. Free
      i. 우리가 free해줄 공간의 header와 footer에 모두 PACK을 사용하여 0으로 만들어준다. 그리고 0이 된다면, 옆의 공간과 병합할 수 있는 4가지의 경우가 생기는데, 이를 처리하는 coalesce함수를 호출하여 4가지의 경우에 대해서 공간을 합쳐준다.
   C. Realloc
      i. Realloc은 기본적으로 malloc으로 할당한 것의 size를 바꿔주는 함수이다.
      ii. 먼저 새롭게 할당받을 size를 입력으로 받아서 mm_malloc을 사용하여 할당해준다.
      iii. 새롭게 할당받을 사이즈가 원래 사이즈보다 작을 경우와 클 경우를 나누어 copySize에 크기를 넣어준 후에, 그 만큼 포인터를 옮겨준다.
      iv. 할당이 끝난 후에는 옛날 포인터는 free해주고, 새롭게 할당받은 주소를 return 해준다.

2. Explicit
   A. Implicit와의 차이점 : 기본적으로 explicit와 implicit의 차이점은 속도에 있다. Explicit는 애초에 free block만 travel하면 되기 때문에, implicit와 비교했을 때 속도면에서 현저한 차이가 난다. Explicit에서 더 요구되는 점이라면, 먼저 implicit에 비해서 더 복잡하게 구성이된다. 또, 연결고리가 2개 생기기 때문에, 2 extra words가 존재하고 더 많은 공간을 할당해야한다. 하지만, explicit의 단점보다 장점이 더 크기 때문에 이렇게 구성하게 되었다. 실제로 implicit로 구성했을 때보다 explicit로 구성했을 경우 performance 점수가 대략 30점 가량 높게 나왔다.
   B. Explicit
      i. explicit는 free block을 따로 travel 하기 때문에, free_listp라는 포인터를 하나 더 만들어 주었다. 또한, 우리가 새로운 공간을 할당받고, 삭제하는

경우에 이 free_listp에 대한 포인터 재 정렬이 필요하기 때문에, 이와 관련한 함수 add_free와 remove_free를 선언해주었다.

ii. Mm_init : 초기화 함수로, 여기서 heap_listp를 선언해준다. 여기서 Implicit와 다른 점은 두 공간(나는 여기서 pre, pos 라는 이름으로 할당하였다.) 을 추가로 더 할당해주어야 한다.

iii. Extend_heap : heap의 공간을 늘려주는 함수로, size를 입력받아 그 크기 만큼(홀수인 경우, +1 만큼 더해서) 크기를 늘려준다. 만약 늘려주고 나서 free blocks들이 연속해서 존재한다면, 이는 coalesce함수를 호출하여 합쳐준다.

iv. Add_free : 이 함수는 새로운 free block을 연결해주는 함수이다. 이 free list는 LIFO 구조를 따르고 있어서 새로운 block은 가장 앞으로 들어간다.

v. Remove_free : 이 함수는 블록을 삭제하는 함수이다. 이 상황에서는 2가지 경우가 존재하는데, 가장 앞에 있는 block을 삭제하는 경우와 그 외의 경우가 있다. 가장 앞에 있는 블록을 삭제하는 경우에는 free_listp에서 바로 다음 블록이 연결되고, 그 외의 경우에는 삭제 블록 앞 뒤의 포인터를 연결해준다.

vi. Coalesce : 이 함수는 free block을 가용할 수 있도록 합쳐주는 함수이다. 여기서는 4가지 경우가 존재한다. 앞, 뒤 블록 모두 할당 되어 있는 경우, 앞 블록은 할당 뒤 블록은 free 인 경우, 그 반대의 경우, 그리고 앞 뒤 블록 모두 free 인 상태인 경우가 있다. 경우가 많기는 하지만 처리하는 방법은 크게 다르지 않다. Free block만큼 사이즈를 늘리고 pack 매크로를 사용하여 free인 상태로 바꾸어주면 된다.

vii. Find_fit : Explicit의 장점인 free block만을 travel 하면 되기 때문에 속도가 훨씬 빨라진다. Free_listp에서 출발하여 내가 사용해야하는 사이즈보다 큰 free block이 나오는 순간 그 포인터를 리턴한다. 찾지 못하면 null을 리턴한다.

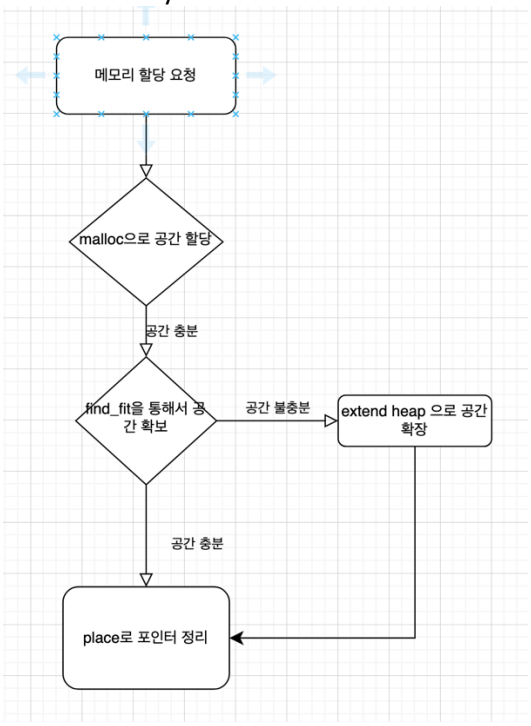
viii. PRE_LINK, POS_LINK : 이는 explicit 구현을 위해 선언한 매크로이다. PRE는 bp위치를 받고, POS는 4 byte 다음 위치를 받는다.

3. 처음에 implicit로 구현했을 때는 util 점수가 대략 10점 정도로 아주 낮게 나왔다. 모든 block을 travel하기 때문에, 속도도 느리고 공간을 효율적으로 사용하기도 힘들었기 때문일 것이다. 하지만 explicit로 구현 후에는 util이 크게 상승하였는데, 이는 free blocks만 travel할 수 있다는 장점 덕분에, 더 빠르고 빈 공간을 더 효율적으로 사용하게 되어서 더 좋은 점수가 나왔음을 예상해볼 수 있다.

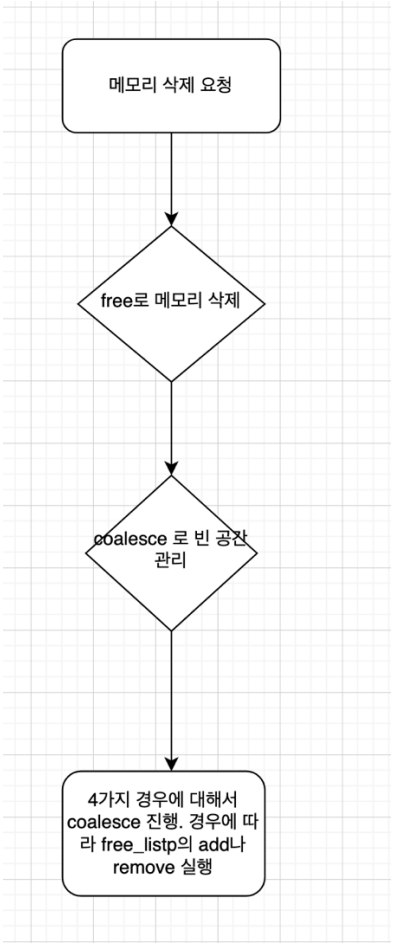
```
 8        yes    51%    24000  0.004097  5858
 9        yes    26%    14401  0.130846   110
10        yes    34%    14401  0.003732  3859
Total            71%   112372  0.145669   771

Perf index = 42 (util) + 40 (thru) = 82/100
```

4. Flow chart
   A. Memory allocate



B. Free


메모리 삭제 요청
free로 메모리 삭제
coalesce 로 빈 공간 관리
4가지 경우에 대해서 coalesce 진행. 경우에 따라 free_listp의 add나 remove 실행