CSED211 LAB7 report

20210054 정하우

<Lab 시간을 통해 배운점>

Malloc는 원하는 크기의 메모리 블록을 할당하고 해제하는 간단한 기능을 제공하는 함수이다. 우리가 구현할 환경인 linux에서는 제한된 동적 메모리 관리 함수인 brk와 sbrk를 제공한다.

utilization과 throughput 성능을 높이기 위해 여러 방식이 존재한다. First fit, next fit, best fit이 있으며, free block을 관리한느 방법에 따라 explisit와 implisit 방식이 있다.

이 함수들을 기반으로 utilization과 throughput 성능을 높힌 사용자지정 malloc 함수를 만드는게 본 랩의 목표이다.

1. Find_fit

   Best-fit기반으로 할당할 free block를 찾아 주소를 리턴해주는 함수이다. Best fit이므로 모든 block를 확인하고 조건에 맞는 사이즈를 갖는 free block중에서 가장 작은 사이즈를 갖는 free block의 주소를 리턴한다.

2. Coalesce

   Free block를 가리키는 주소를 인자로 받는다. 이 block전후로 할당여부를 판단해 block 전후로 할당여부를 판단해 앞 뒤 block와 합칠지 판단한다. 앞 뒤 block가 할당되어 있는지 가용블록인지 총 2가지 경우가 있으므로 총 2*2=4가지 경우가 있다. 이 경우에 대해 모두 if문을 통해 경우를 나누어 block를 합치는 과정을 구현하였다.

3. Extend_heap

   Mem_sbrk함수를 통해 heap를 확장하는 함수. Find_fit함수를 통해 할당할 free block를 찾지 못했을 때, 초기조건(mm_intt함수사용할 때) 에서 첫 heap을 정할 때 부른다. Header, footer 정보를 넣고, heap 범위가 확장되었으므로 epilogue header도 재할당 해준다.

4. Place

   할당할 block의 주소와 할당할 사이즈를 인자로 받는다. 할당할 free block이 충분히 크다면 두개의 block으로 나누고 앞 block는 할당하고, 뒷 block는 새로운 사이즈의 free block로 만들어 준다. 만약 할당할 free block의 사이즈 전부가 필요하다면 나누지 않고 모든 범위를 할당상태로 만들어 준다.

5. Mm_init

   메모리 할당기를 초기화해주는 함수. 이 함수에서 prologue header와 prologue footer, 그리고 epilogue header를 만든다. 그리고 extend_heap함수를 통해 초기 heap크기를 조정해 준다.

6. Mm_malloc

앞선 함수들을 이용해서 heap에 메모리 할당을 해주는 함수. 할당하고 싶은 사이즈를 인자로 받으면 find_fit함수를 통해 할당할 만한 최적의 block를 best-fit을 통해 찾고 place함수를 통해 할당해 준다. 만약 적절한 free block가 extend_heap함수를 통해 heap를 확장해서 확장된 뒷 부분에 할당해 준다.

7. Mm_free

   역시 앞선 함수들을 이용해 할당된 block를 할당해제해주는 함수이다. 할당하고 싶은 block의 주소를 인자로 받으면 그 block의 header와 footer에 할당되었다는 정보(1)을 할당해제되었다고(0)바꿔준다. 그리고 coalesce함수를 통해 free block를 합쳐준다.

8. Mm_realloc

   기존에 할당되어있는 block의 크기를 바꿔주는 함수. 사이즈를 바꾸고싶은 block의 주소와 바꾸고싶은 사이즈를 인자로 받는다. 사이즈 인자가 0인경우 mm_free함수를 통해 할당해제시켜주고, block의 주소가 NULL이면 mm_malloc함수를 호출한다. 이외의 경우에는 block의 사이즈를 바꿔주는데, 만약 확장할경우 확장할 공간이 충분하면 사이즈를 바꿔줄필요가 없으므로 그대로 받은주소를 리턴한다. 만약 부족할경우, 다음 block이 할당되어있는지를 본다. 만약 할당되어있지 않으면 그 block의 사이즈를 사용할수 있다. 현재 block와 다음 block의 사이즈를 모두 사용했을 때 충분하다면, 두블록을 합친다.(이 과정에서 header와 footer의 정보를 바꾼다). 그리고 시작주소는 변치 않았으므로 받은 주소 그대로 리턴한다. 만약 다음 block의 사이즈를 사용해도 부족하다면, extend_heap함수를 통해 heap을 확장하고, 확장된 곳에 새롭게 할당해준다.(이 과정은 mm_malloc함수가 알아서 처리해준다) 그리고 기존에 구현되어있는 함수인 memcpy함수를 통해 그전 block의 정보를 새롭게 할당된 block에 복사하고, 기존 block는 mm_free함수를 통해 할당해제해준다.

9. 시간복잡도

   find_fit함수는 best_fit이므로 O(size)만큼의 시간복잡도를 가지고 free함수에서 O(1)시간복잡도를 가진다.

<결과>

```
bash-4.2$ ./mdriver -v
Using default tracefiles in /home/std/jaechang/cs
Measuring performance with the interval timer.

Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   99%    5694  0.326587    17
 1       yes  100%    5848  0.261645    22
 2       yes   99%    6648  0.382331    17
 3       yes  100%    5380  0.289130    19
 4       yes   99%   14400  0.000094153029
 5       yes   96%    4800  0.522820     9
 6       yes   95%    4800  0.586881     8
 7       yes   55%   12000  2.880179     4
 8       yes   51%   24000  9.361950     3
 9       yes   39%   14401  0.000209 68970
10       yes   40%   14401  0.035584   405
Total          79%  112372 14.647411     8

Perf index = 48 (util) + 1 (thru) = 48/100
```

Space utilization은 꽤나 괜찮은 결과를 얻었다. 하지만 throughout는 좋은 점수를 얻지 못했는데, 이는 implicit list 기반 best-fit로 free block를 구하는 과정에서 시간적 손해를 많이 본 것 같다. 이를 해결하기위해서 explicit list 방식을 통해 구현하거나 더 나아가 segregated free list방식을 구현한다면 훨씬 좋은 throughout 값을 얻을 수 있을 것이다. 하지만 구현 과정에서 세그맨테이션오류를 해결하지 못해 기존 방식을 그대로 사용하였다.

<학습내용>

본 랩을 통해 할당기가 어떤방식으로 구현되는지 이해하게되었다. Implicit list 기반의 best fit방식으로 구현하면서 best fit 뿐만 아니라 first fit과 next fit 개념을 확실히 잡은 것 같다. 그리고 구현에는 실패하였으나 구현 과정에서 explicit list와 segregated free list 개념을 잡을 수 있었다.