Multicore Programming Project 3

담당 교수 : 박성용

이름 : 김희진

학번 : 20211530

1. 개발 목표

수업 중 배운 explicit free list 방식을 이용해 dynamic memory allocator를 구현한다. Explicit free list는 free block을 관리하기 위한 방식 중 하나로, free block들을 이어주는 list를 통해 free block을 관리한다.

2. 개발 범위 및 내용

A. 개발 내용

Explicit free list를 이용하기 위해서 allocated block과 freed block의 설계를 다르게 해주었다.

-allocated block

(Header) size	payload	(Footer) size

Allocated block에는 다음 block의 시작 address를 알기 위한 header와 다음 block에게 자신의 시작 address를 알려주는 footer를 사용합니다. 같은 block의 header와 footer는 같은 값을 가집니다. 다음 block의 시작주소로 이동할 때는 현재 block의 header size만큼 뒤로 이동해주고, 이전 block의 시작주소로 이동할 때는 현재 block의 header바로 앞에 있는 footer의 size만큼 앞으로 이동해 줍니다.

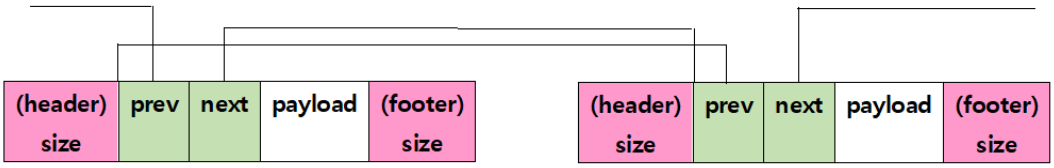
명세서에서 요구한 대로, block은 double word alignment 방식으로 할당됩니다. 그렇기 때문에 internal fragmentation이 발생합니다.

-freed block

(header) size	prev	next	payload	(footer) size

Freed block에는 allocated block과 마찬가지로 header, footer, payload가 존재할 뿐만 아니라 이전 free node의 주소값을 저장하는 prev, 다음 free node의 주소값을 저장하는 next가 존재합니다.

List를 연결하는 방식은 다양한데, 마지막으로 free된 block을 가장 앞에 연결하는 LIFO형태로 구현했습니다. 여러 free block들이 연결된 모습은 아래처럼 나타낼 수 있습니다.



B. 개발 방법

1. Global variables

   사용한 전역변수는 **static void* heap_listp** 한 가지 입니다. 이는 첫번째 block을 가리킵니다.

2. Subroutines

- mm_init

  메모리를 초기화하는 함수입니다. 초기 heap size는 6*WSIZE로 설정했습니다. Put이 총 6번이 나오는데 첫번째 PUT은 padding을, 두번째는 prologue header를, 세번째는 이전 free node 저장을 위한 것입니다. 네번째 PUT은 next free node 저장을 위한 것이고 다섯번째는 prologue footer를, 5번째는 epilogue header 저장을 위한 것입니다. Heap size를 늘리는 데 있어 에러가 발생하면 -1을 아니라면 0을 return 합니다.

- mm_malloc

  메모리를 동적할당 해주는 함수입니다. 할당하고자 하는 block size를 argument로 받습니다. Size가 0이라면 NULL을 return합니다. 해당 프로젝트에서는 double word alignment를 따르기에 size가 double word 이하의 크기라면 2*Dsize를, 초과하는 크기라면 DSIZE*((size+2*DSIZE-1)/DSISE)만큼 공간을 할당해줍니다. Find_fit함수를 통해 해당 size이상의 freed block을 찾아주고 place함수를 이용해 공간을 할당해줍니다. 할당할 수 있는 freed block이 없을 경우에는 extend_heap함수를 이용해 heap의 크기를 늘리고 palce함수를 통해 공간을 할당해 줍니다.

- mm_free

  malloc이나 realloc을 통해 할당한 공간을 해제해주는 함수입니다. Header와 footer에 tag 0을 할당해주어 free block임을 표시해주고

coalesce 함수를 호출해주어 합칠 수 있는 공간이 있으면 freed block끼리 합쳐줍니다.

- mm_realloc

  malloc 또는 realloc 함수를 통해 할당한 공간의 크기를 늘리고자 할 때 호출하는 함수입니다. 공간을 업데이트하고자 하는 블록의 주소와 크기를 argument로 받습니다. 주소가 null이라면 mm_malloc을 수행하고 size가 0이라면 mm_free를 수행합니다. 또한 새로 업데이트하려는 크기가 기존 크기보다 작다면 아무 작업도 수행하지 않습니다.

- insert_list

  freed block을 연결하는 list에 freed block을 추가하는 함수입니다. 추가하려는 block의 주소를 argument로 받아서 기본 첫번째 block인 heap_listp 다음에 넣어줍니다. 그 과정은 다음과 같습니다. heap_listp의 next block를 인자로 받은 block의 next에 저장해주고 heap_listp를 인자로 받은 block의 prev에 저장합니다. heap_listp의 다음 block의 prev에 인자로 받은 block의 주소를 저장합니다. 그리고 heap_listp의 next를 인자로 받은 주소로 변경해줍니다.

- remove_list

  freed block을 연결하는 listp에서 freed block을 삭제하는 함수입니다. 삭제하려는 block의 주소를 argument로 받습니다. 인자로 받은 block의 이전 node의 next에 인자로 받은 block의 next node의 주소값을 저장해줍니다. 인자로 받은 block의 next node의 prev에 인자로 받은 block의 이전 block의 주소값를 저장해줍니다

- Extend_heap

  현재 heap에 요구 size만큼 할당할 수 있는 공간이 없을 때 sbrk 함수를 통해 heap 크기를 증가시킵니다. 이 함수는 늘릴 크기를 argument로 받고, 이에 double word alignment를 적용한 크기를 구하고 sbrk를 통해 증가시키게 됩니다. 그리고 증가시킨 heap의 첫번째 block의 header와 footer에 크기를 저장하고 epilogue header를 할당해주어 init 해줍니다.

- place

  freed block에 block을 새롭게 할당하는 함수입니다. argument로 할당할 free block의 주소와 크기를 받습니다. Remove_list 함수를 호출해서 freed block list에서 인자로 받은 block을 삭제해줍니다. Freed block에 block을 할당하면 split되는 경우가 생기는데 이는 첫번째 if문이고 split 되지 않는 경우는 else문에 해당합니다. Split되지 않는 경우는, 기존 freed block의 hedaer와 footer에 할당할 block의 크기를 update 해주고 tag값을 1로 바꿔줍니다. Split되는 경우 split된 block은 크기를 header 와 footer에 저장하고 insert_list함수를 호출해 freed block list에 넣어주는 과정을 추가해줍니다.

- find_fit

  first fit 방식으로 할당할 free block을 찾는 함수입니다. argument로 할당이 필요한 size를 받습니다. Freed block list의 처음부터 탐색하며 size이상의 freed block이 있을 때 이 주소값을 return합니다.

- Coalesce

  Mm_free함수를 통해 공간을 해제하고 freed된 공간이 그 주변에 있다면 이를 합쳐주는 함수입니다. 첫번째 if문은 이전 block은 allocate되어 있고 다음 block은 freed된 상태입니다. 다음 block을 freed block list에서 삭제해주고 현재 block size에 다음 block의 size만큼 더하여 현재 block의 header와 footer를 update 해줍니다. 두번째 else if문은 이전 block은 freed 상태이고 다음 block은 allocate된 상태입니다. 이전 block을 freed block list에서 삭제해주고 이전 block size를 현재 block size에 더하여 이전 block의 header와 현재 block의 footer에 update합니다. 세번째 else if 는 앞 뒤 block 모두 free인 상태입니다. 앞, 뒤 block을 모두 freed block list에서 삭제해주고 이전 block, 다음 block 그리고 현재 block의 size를 모두 더한 값을 이전 block의 header와 footer에 update 해줍니다. 그리고 insert_list함수를 이용해 update된 block을 freed block list에 넣어줍니다.

3. **구현 결과**

```
Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   89%    5694  0.000419 13606
 1       yes   92%    5848  0.000251 23299
 2       yes   94%    6648  0.000482 13804
 3       yes   96%    5380  0.000383 14036
 4       yes   66%   14400  0.000259 55641
 5       yes   88%    4800  0.000639  7513
 6       yes   85%    4800  0.000647  7419
 7       yes   55%   12000  0.004050  2963
 8       yes   51%   24000  0.004333  5539
 9       yes   97%   14401  0.000151 95624
10       yes   38%   14401  0.000157 91668
Total          77%  112372  0.011770  9547

Perf index = 46 (util) + 40 (thru) = 86/100
```

11가지 경우에 있어 valid가 yes값이 나온 걸 확인할 수 있었으며 implict로 구현했을 때에 비교해 performance가 약 30점이 오른 것을 확인할 수 있었습니다.