Malloc Lab Report

-2019-17577 경제학부 정지후-

1. 구현 방법

implicit list에 next_fit + best fit method를 이용했다.

- 기본 상수
  - 워드 사이즈 WSIZE, 더블 워드 사이즈 DSIZE, 초기 가용 블록 설정 위한 INIT_CHUNKSIZE, extend heap 때 사용할 CHUNKSIZE 와 같은 기본 상수를 정의했다.
- macro
  - 연산을 빠르게 하기 위해 매크로를 이용했다.
  - GET 매크로는 매개변수 포인터가 참조하는 워드를 리턴
  - PUT 매크로는 매개변수 포인터가 가리키는 워드에 매개변수 value를 넣는다.
  - PACK 은 size와 alloc 여부를 pack한다.
  - GET_SIZE(GET_ALLOC)는 매개변수 포인터에 저장된 사이즈(allocated 여부)를 읽어온다. 즉, header나 footer에 대한 포인터가 매개변수로 쓰인다.
  - HDRP(FTRP)는 블록 헤더(푸터)를 가리키는 포인터를 리턴함
  - NEXT_BLKP(PREV_BLKP)는 next(previous) block 포인터를 리턴함
- core 함수 : mm_init, mm_malloc, mm_realloc, mm_free
  - mm_init : 메모리에서 4워드를 가져와 빈 가용 리스트를 만들고, 여러 가지 초기화 시 필요한 alignment padding, Prologue header, Prologue footer, Epilogue header 등을 넣는다. 이후, extend_heap을 이용해서 INIT_CHUNKSIZE 만큼 heap size를 늘려준다. 그리고, last_bp에 heap_listp를 저장한다.
  - mm_malloc : 가장 먼저, size를 재조정한다. alignment에 맞게 재조정한 size를 asize에 넣고, next fit 함수를 call해서 어떤 곳에 allocate할 지 올바른 장소를 정한다. 그리고, 그 장소를 찾으면, 그곳에 place 함수를 이용해 allocate을 하고 last_bp를 설정한다.
  - mm_free : free할 block의 header와 footer에 unmark를 하고

coalesce함수를 이용해 coalescing 이 필요할 경우 coalescing한다.

- mm_realloc : mm_alloc 함수와 마찬가지로, asize를 먼저 계산한다. 만약 current block의 size가 asize보다 크거나 같다면, 아무 것도 하지 않는다. 그렇지 않으면, next block이 free이거나 에필로그 블록이 아닌지 확인한다. 만약 그렇다면, 현재 블록과 다음 블록을 이용해서 realloc한다. 이 때, extend_heap을 할 필요가 있다면, 해준다. next_block을 이용하지 못할 경우, 새로운 block을 만드는데, 이에는 mm_malloc함수를 이용한다. 그리고 기존의 내용을 copy해주고, 기존 block은 free 해준다.

- helper function : next_fit,place, extend_heap, coalesce 함수를 구현했다.
  - next_fit : 기본적으로는 next_fit method를 이용하는데, 더 utility를 올리기 위해 best fit tech를 약간 차용한다. best_fit으로 하면 시간 복잡도가 올라가기 때문에, 적정한 범위에서 Best를 찾아 이를 쓴다. next_fit mehod이므로 last_bp에서 시작해서 적당한 크기를 찾는다. 만약 이렇게 해서 못찾았을 경우에는, 처음부터 리스트 last_bp까지 리스트를 훑으면서 적절한 곳을 찾는다.
  - place : current block size와 필요한 size를 비교해서, 그 차이가 2*DSIZE 보다 크거나 같으면, 현재 블록을 split해서 앞의 asize만큼은 alloc하고, 뒤는 free를 유지한다. 그렇지 않다면, 그냥 블록 전체를 이용하면 된다. 간단히, header와 footer를 mark해주면 된다.
  - coalesce : previous block과 next block이 free인지에 따라 case를 4개로 나누어 처리한다. 만약 둘 다 allocated되었다면, coalescing 이 필요없다. 그저 last_bp만 재설정해준다. 둘 다 free이면 앞 뒤를 coalescing 해준다. size를 previous block size와 next block size를 합친만큼 더 해주고, previous의 header와 next의 footer를 unmark 해주고, 이의 사이즈를 변경한다. 만약 previous block은 free, next는 alloc이면 previous block size를 더해주고, 현재 block의 footer와 previous의 header의 사이즈를 조정하고 unmark해준다. 반대의 경우는, next block size를 더해주고 현재 block의 header와 footer의 사이즈를 변경하고, unmark한다.
  - extend heap : alignment에 맞도록 적절한 사이즈를 설정한다. initialize해주고, 앞의 블록이 free였다면 coalescing해준다.

2. 어려웠던 점

- 전반적으로는 교과서의 implementation을 참고했지만, reallloc의 경우는 그렇지 못해 이것이 살짝 어려웠다. 그리고, coalescing code는 교과서의 코드 중 case 2의 경우를 잘 이해하지 못해 살짝 어려웠다. 또, 성능을 올리기 위해, next fit에다가 best fit적인 요소를 섞었는데, next fit 자체를 구현하는 것도 살짝 까다로웠다. last

bp뒤를 먼저 보고, 이후 앞도 봐야하는데, 이를 깨닫기가 오래 걸렸다. 사실 explicit list나 segregated list구현도 시도했었으나, 너무 복잡해 포기했던 것도 사실이다. array 등을 쓸 수 있었다면 상당히 간편했을 테지만, 이를 쓰지 못해 포인터 연산만을 이용해야 했어서 그것도 까다로운 편이었다.

3. 새롭게 배운 점

- 포인터에 대해 많이 이해하게 되었다. 또, 매크로가 평소에 약간 복잡해서 잘 쓰지 않았는데, 이번 기회에 매크로의 장점에 대해 알게 되어, 앞으로도 빠른 프로그래밍이 필요할 때, 자주 이용할 것 같다. 직접 dynamic memory allocation을 구현해보니, 매우 뜻깊은 경험이었고, dynamic memory allocation에 대해 더 깊이 이해한 것 같은 느낌이다. 성능을 최적화하기 위해 다양한 fit함수를 구현했었는데, 이의 성능을 직접 비교해보는 경험이 되었다. 각각 fit의 장단점을 잘 이용해 trade off를 적절히 한 next_fit + best_fit을 이용했다. 비록, segregated list와 explicit list를 구현하지는 못했으나, 이를 구현하다가 실패하는 과정에서 이에 대한 이해도 높아졌다.