System Programming

Programming Assignment #3 Malloclab

2013-11826 임주경

- Structure

각 Block은 8-byte alignment를 따를 수 있도록, 최소 4 words 크기로 구현한다. Allocated block은 Header, Footer를 포함한 8 bytes와 4 byte payload의 짝수개로 구성되어 있고, Freed block은 마찬가지로 Header, Footer와 같은 Segregated list내의 prev, next free block을 가리키는 포인터로 이루어져 있다. Block의 Header와 Footer의 LSB는 allocation bit로 할당한다. Free list는 segregated list로 구현하며, 총 20개의 list가 존재한다. 각 list에는 2^n~2^n-1 크기의 블록 주소를 담도록 한다. Allocator는 best-fit 방법을 채택한다.

- Macro

수행 속도를 향상시키기 위해, Macro를 사용한다. 각 Macro는 code내부 주석으로 설명되어 있다.

- mm_innit

Mem_sbrk 함수 호출로 heap의 맨 아래에는 segregated list가 차지하도록 한다. 다음, 모든 list를 초기화 한다. 그 후에, 다시 mem_sbrk 호출로 Alignment padding, prologue header / footer, epilogue header를 할당하며, 첫 시작될 block pointer는 prologue header와 footer 사이를 가리킨다. 이때, 시작 포인터에 extend_heap 함수를 호출해서 heap 공간을 늘린다.

- extend_heap

인자로 words의 수를 받는다. 8-byte alignment를 준수하도록, words의 수를 짝수로 반올림하여, WSIZE를 곱한 size만큼을 할당하는 함수이다. 첫 할당 공간은 전부 free block이므로, free block header / footer에 allocation bit를 0으로, size값을 갖도록 값을 설정한다. 형성된 free block은 free_insert 함수 호출로 free list에 삽입한다. Return 값은 coalesce 함수를 수행해 coalesce 작업을 마친 block pointer를 반환한다.

- free_insert

Free_insert 함수는 인자로 받은 block pointer와 해당 block의 size를 이용해 합당한 segregated list에 pointer 삽입을 수행한다. Loop를 통해 list를 돌며, 들어가기에 알맞은 크기의 list를 찾고, list 내부에서 크기 순으로 정렬되도록 제 자리를 찾아 삽입한다. 각 삽입시에 list 내부 상황에 대한 분기는 code 내부에 설명되어 있다.

- coalesce

Free를 수행하고 나면, freed block의 Prev, Next block들에 대해 다음과 같이 4가지 상황이 올 수 있다. A를 allocated, F를 Freed, bp를 방금 freed 된 block이라고 한다면,

Prev – bp – Next

1. A - F - A
2. A - F - F
3. F - F - A
4. F - F - F

각 상황에 맞게 연속된 freed block 을 결합하며, 결합된 포인터는 free list에 삽입한다. 해당하는 block의 pointer를 return 한다.

- mm_malloc

Size가 0인 경우, NULL을 return 한다. Size는 Pay load의 size를 의미하므로, 여기에 Header / Footer의 overhead 8 bytes를 더한다. 해당하는 size가 free list에 존재하는지 free_find 함수를 호출하고, 존재할 경우 addblock 함수를 호출한다. Free list에 적당한 freed block를 찾지 못하면, heap의 크기를 증가시켜야 한다. 따라서, extend_heap 함수를 호출하고, addblock 함수를 호출한다. 할당한 뒤의 block pointer를 return 한다.

- free_find

Segregated lists를 돌면서, 인자로 받은 size가 할당 가능하며, 남는 공간이 최소가 되는 best-fit 전략을 채택한다. While loop로 list를 돌면서 해당하는 freed block의 pointer를 return 한다.

- addblock

Block pointer와 size를 인자로 받는다. 우선 해당하는 포인터 bp를 free list에서 free_remove 호출로 제거한다. 이때, 할당하고 남는 블록의 크기가 최소 블록의 크기보다 크다면, splitting으로 할당하고 남은 공간을 새롭게 free list에 삽입한다. 만약, 할당하고 남는 공간이 최소 블록의 크기보다 크지 않다면, 공간을 남기지 않고 전부 새로운 할당 블록으로 전환한다. 할당한 뒤, pointer를 return 한다.

- free_remove

지우고자 하는 freed block의 포인터를 인자로 받는다. 단순 loop를 통해, 해당하는 포인터를 list 내부에서 찾아내고 찾은 pointer를 list 내부에서 제거하는 함수이다.

- mm_free

해당하는 포인터의 header 와 footer의 allocation bit를 1에서 0으로 수정하고, free_insert를 호출해서 free list에 삽입한다. 다음, coalesce 수행으로 마친다.

- mm_realloc

인자로 받은 포인터가 NULL이면, mm_malloc(size)를 수행한다. Size가 0이면, mm_free를 수행한다. 할당하고자 하는 size가 기존 block의 size와 같으면, 아무것도 하지 않고 block pointer를 return 한다. 이러한 특별 case에 해당하지 않을 경우, 3가지 경우에 나뉘어 수행된다. 할당하고자 하는 크기가 기존의 크기보다 작을 경우, 기존 block에서 8-byte alignment를 준수하며, 뒤에 제거되어야 하는 block은 free를 수행하도록 한다. 할당하고자 하는 크기가 기존의 크기보다 크며, 이때, 기존의 block에 연속된 block이 free하며, 공간이 충분할 때, 필요한 공간만큼 새롭게 할당하고, 남은 공간은 free로 남긴다. 마지막으로, 완전히 새로운 공간을 할당 받아야만 하는 경우, mm_malloc 호출로 새로운 공간을 할당 받고, 기존 block의 contents는 memcpy 함수 호출을 이용해 수행한다.

- mm_check

Debugging을 위한 Heap consistency checker 함수 mm_check는 아래와 같은 부분을 check 하여, 에러가 발생하면, 해당 error 메시지를 출력하고 0을 return 한다. Error가 없으면, 1을 return 한다.

Free list의 모든 block이 free로 marking되어 있는지, 연속된 free block이 전부 coalescing되어 있

는지, 모든 free block이 free list에 존재하는지, header와 footer의 size와 allocation bit가 일치하는지, 8-byte alignment를 준수하지 않는 pointer가 존재하지 않는지를 확인한다.

- 마치며

이번 과제 구현에서 어려웠던 점은 포인터 연산하는 과정에서 타입이 알맞아야 연산 또는 결과에 에러가 발생하지 않는다는 점과, 8-byte alignment를 깨지 않도록 block구조와 heap구조의 모순점을 수정하는 부분이었다. 구현을 하며, 기존에 주어진 implicit list를 이용해서는 performance가 너무 떨어졌기 때문에, explicit list 또는, segregated list로 구현을 마음먹게 되었다. 수행과정에서 explicit 구현도 해보았지만, 생각보다 performance가 떨어져서 segregated list로 구현하였다. Segregated list 구현이 더 성능이 좋음을 확인하였으며, Red-Black tree를 이용한 구현은 더욱 성능이 좋을 것으로 기대된다. 본 프로젝트를 수행하며, 메모리 할당의 규칙과 내부 구조에 대한 이해에 큰 도움이 되었다.