System Programming Project 4

담당 교수 : 박성용

이름 : 조원빈

학번 : 20201644

이번 프로젝트에서는 segregated free list를 이용해서 dynamic memory allocation을 구현했다. segregated free list는 implicit list방법이나 explicit list 방법보다 더 효율적으로 작동한다.

segregated free list의 효율성

1. memory 할당을 위한 공간을 검색하는 데에 있어 implicit list나 explicit list은 최악의 경우 O(n)의 시간 복잡도를 가진다. 하지만 segregated free list는 size를 2의 제곱수마다 나누어 log개의 구간으로 더 효율적으로 관리하므로 또 빠른 시간에 탐색이 가능하다.

2. 이번 프젝에서 구현한 segregated free list에서는 bes-fit 방식을 사용하 메모리를 allocation했는데 그 덕분에 fragmentation이 감소하여 더 효율적으로 메모리를 할당할 수 있다.

3. 메모리를 할당(free)할 때에도 이전 블록과 이후 블록 두 깨만 고려하여 빠르게 free할 수 있다.

프로그램 실행 흐름

1. initialize : mm_init() 함수를 통해 segregated list를 위한 공간을 할당 해주고, prolog header와 epilog header 등 heap의 초기 메모리를 할당해준다.

2-1. malloc :mm_malloc() 함수를 이용해 메모리를 할당할 때에는 2가지 경우가 존재한다.

- segregated free list에 원하는 size만큼 메모리를 할당할 공간이 존재하는 경우 : find_fit()함수를 이용해 size보다 크거나 같은 free memory를 가진 block을 찾고, 그 block에 allocate한다.

- segregated free list에 원하는 size만큼 메모리를 할당할 공간이 존재하지 않는 경우 : extend_heap() 함수를 통해 추가로 memory 공간을 할당한 후 allocation을 진행한다.

2-2. free : mm_free() 함수를 이용해 동적 할당했던 메모리 공간을 해제할 수 있다. 공간을 해제한 후에는 header와 footer의 값을 갱신해주어야 하고, 양옆에 인접한 free block과 합쳐질 수도 있으므로 coalesc() 함수를 이용해 확인해준다.

3. realloc : realloc을 할 때에는 CHUNKSIZE를 기준으로 경우를 2가지로 나눈다. 만약 추가로 할당하는 공간의 size가 CHUNKSIZE보다 작은 경우에는 size보다 크거나 같은 가장 작은 2의 제곱수를 size로 잡고 할당해준다. 이렇게 했을 때 더 효율적으로 작동하는 이유는 malloc, free, realloc함수를 반복적으로 호출해 재할당해주는 경우 미리 여유 공간을 할당해 주는 것이 fragmentation을 줄일 수 있기 때문이다. 실제로 c++의 vector의 구현에서도 동적으로 공간을 할당해줄 때 메모리 공간을 2배씩 늘려주는 방식으로 작동하는 것으로 알고 있다.

segregated free list 작동 방식

이번 구현에서는 크게 allocated 또는 free memory block을 관리하는 implicit memory처럼 구현된 부분과, 원하는 size만큼의 크기를 갖는 free memory가 존재하는지 확인하기 위해 segregated free list 방식의 구현이 존재한다. segregated free list는 2의 제곱수를 기준으로 log개의 구간으로 나누어져 있으며, 각 구간에서 노드는 size 크기 순으로 내림차순 정렬된 상태를 유지한다. node는 add_node() 함수나 remove_node() 함수를 이용해 삽입, 삭제할 수 있는데, 이를 위해서는 각 구간에 linked_list로 연결된 노드들을 확인해야 하므로 최악의 경우 구간의 노드 개수에 비례하는 시간이 걸린다.

전역 변수와 함수들

- char** segregated_list: segregated free list의 head를 모아놓은 배열
- char** heap_ptr: 힙의 끝 포인터
- prev_size: 이전의 heap size를 저장하는 변수.
- mm_init: segregated free list를 초기화하고, 힙의 초기 상태를 할당해 준다..

- mm_malloc: 지정된 크기의 메모리 블록을 할당하며, 공간이 충분할 경우에는 바로 할당하고 아니면 extend_heap함수를 이용해 heap 크기를 늘린다.

- mm_free: 할당된 블록을 해제하고 앞뒤 free block고 합친후 segregated free list에 노드를 추가해준다.

- mm_realloc: 할당된 블록의 크기를 입력된 크기로 바꾸는 역할을 하며, segregated free list도 갱신해준다.

- extend_heap: 힙을 지정된 양만큼 확장하고 앞의 free block과 합쳐준다.

- coalesce: 4가지 case를 고려해 인접한 free block과 합쳐주어 fragmentation을 줄이고 segregated free list를 갱신해준다.

- add_node : free block을 segregated free list에 추가할 때 사용하는 함수이다. 우선 size를 보고 list의 어느 구간에 들어갈 지 확인한 후, 구간 내에서는 내림차순 정렬이 되어있어야 하므로 처음부터 탐색하면서 적절한 위치를 찾은 후 앞 뒤 노드와의 관계를 갱신해준다.(포인터 활용)

- remove_node : 입력으로 받은 block을 segregated free list에서 지우는 역할을 한다. node를 지울 때 각 구간은 linked list 자료구조를 이용해 구현되어 있으며, 앞의 노드와 뒤의 노드를 새로 연결해주는 작업이 필요하다.

- place : place 함수에서는 bp로 받은 block에 words 크기만큼의 공간을 할당하는 역할을 한다. 이때 주의할 점이 bp의 크기가 원하는 크기보다 더 클 수 있다는 점이다. 따라서 남는 공간도 고려를 해주어야 하는데, 남는 공간의 크기가 2 word(8 byte)보다 작은 경우에는 8-byte alignment 로 작동하는 이 프로그램에서는 의미있는 역할을 하지 못하므로 무시하고, 그 외에 경우에는 free block으로 갱신해주가 list에 node를 추가해준다.

- find_fit : size(word 단위) 이상의 크기를 갖는 free block을 찾는 함수이다. segregated free list에 free block을 저장해놨기 때문에 먼저 size이상의 크기를 갖을 수 있는 구간을 먼저 구한 후 탐색을 진행한다. 만약 구간 내에 원하는 크기의 block이 존재하지 않을 경우 더 큰 size를 갖는 구간으로 계속해서 탐색을 이어나간다.

- mm_realloc : ptr에 할당된 크기를 size로 바꾸는 함수이다. 이 때 더 효율적으로 공간을 할당하기 위해 size가 2^12보다 큰 경우와 작은 경우를 나누어 각각의 경우에 다르게 계산한다.

- coalesce : bp를 인자로 받아 앞 block과 뒤 block의 allocation 상태를 확인한 후 각각의 경우에 따라 case work를 하여 free block을 합쳐준다.

결론

이번 프로젝트에서는 동적 메모리 할당을 구현하기 위해 segregated free list를 사용했다. list에는 각 구간에 free block을 크기의 내림차순으로 저장하였다. 또한 malloc할 block의 size를 정할 때 size가 CHUNKSIZE(2^12)보다 작으면 size보다 크거나 같은 가장 작은 2의 제곱수로 size를 정했는데, 이를 통해 이후의 연산에서 fragmentation을 줄여 더 효율적으로 작동하여 더 높은 점수를 받을 수 있었다. 다만, 너무 size가 커질 경우 overhead가 커져 5번째 case를 통과하지 못했고 따라서 CHUNKSIZE를 통해 제한을 주었다.