Malloc Lab Report

2019-12333 정윤서

1. Implementation

- **mm_init:** segregation class에 따른 free list를 구현하기 위한 void *free_listp를 initialize하는 함수이다. Free_listp는 padding word로 시작해 4바이트의 prologue header, 14*4 바이트의 seg list 포인터, 4바이트의 prologue footer, 4바이트의 epilogue header로 초기화된다. 따라서 총 18*4 바이트 만큼의 힙 사이즈를 mem_sbrk() 함수를 통해 할당받고, free_listp가 prologue의 payload 시작점을 가리키도록 설정한다. 그 뒤 extend_heap() 함수를 통해 heap 사이즈를 default 크기만큼 늘려줌으로써 heap space를 초기화한다.
-
- **mm_malloc:** memory allocation은 해당 size를 위한 best fit을 heap 안에서 찾거나, 메모리가 부족한 경우 힙 사이즈를 늘려줌으로써 이루어진다. Best fit은 segregation class 안에서 가장 작은 사이즈를 가지고 있으면서도 필요한 사이즈를 충족하는 free block을 의미한다. 만약 필요한 size에 해당하는 class부터 시작해 가장 큰 class의 free list까지 모두 탐색했는데도 fit이 없다면 힙 사이즈를 늘려준다. 전자의 경우, 필요한 만큼 allocate한 후 남는 공간이 16바이트(최소 block 크기) 이상이라면 block을 split하여 생긴 새로운 free block을 add_first() 함수를 통해 해당하는 class의 free list 가장 앞에 삽입해 준다. 후자의 경우 extend_heap() 함수를 이용한다. 이 함수에서는 새롭게 생긴 free block을 coalesce() 함수를 통해 앞 free block과 합치고(만약 앞 block도 free block이라면), add_first()를 통해 해당하는 class의 free list 가장 앞에 삽입해 준 뒤 새로운 pointer를 반환한다. 그 뒤 mm_malloc에서는 place() 함수를 사용하여 전자와 같은 일을 수행한다. Place() 함수에서는 추가적으로 boundary tag의 allocated bit를 바꿔 주고 해당 free block에 연결되어 있던 pointer들을 처리하는 일을 한다.
-
- **mm_free:** free되는 header와 footer의 allocated bit을 바꿔 주고 coalesce한다. Coalesce되어 반환된 pointer는 해당 크기에 맞는 free list 가장 앞에 삽입된다. Coalesce() 함수에서는 추가적으로 합쳐지는 free block에 연결되어 있던 Pointer들을 처리하는 일을 한다.
-
- **mm_realloc:** 주어진 사이즈에 해당하는 block 사이즈가 원래 block 사이즈보다 같거나 작다면 바로 return한다. 더 큰 사이즈가 필요한데, 사이즈를 늘리려는 block의 바로 다음 block이 free block이고, 합쳐졌을 경우 size가 만족된다면 두 block을 합쳐서 새로운 block을 만들고 return한다. 바로 다음 block이 epilogue header인 경우에는 필요한 만큼 heap 사이즈를 늘린 뒤 기존 block과 합치고 return한다. 이러한 조건이 충족되지 않는 경우 어쩔 수 없이 mm_malloc()을 불러서 새로운 메모리를 할당하고, 기존 내용을 복사한 뒤 기존 block을 free한다.

2. What was difficult

- 로직이 돌아가게 하는 것뿐만 아니라 일정 수준의 성능을 맞추도록 해야 하는 것이 가장 힘들었습니다. 성능은 utilization과 throughput으로 측정되기 때문에, 그냥 implicit list과 같이 heap 전체를 돌면서 fit을 찾고 allocate하는 방식은 utilization은 나름 괜찮았으나 throughput이 낮게 나왔습니다. Explicit list로 구현했을 때는 thoughput이 높았으나 utilization이 높지 않아서 여전히 90점을 넘기기 힘들었습니다.

- Segregation list 방식으로 구현한 이후에도 mm_realloc()을 단순 malloc()+free() 조합으로 구현해 놓으니 throughput이 굉장히 낮게 측정이 되었습니다. 바로 malloc()을 사용하지 않고 다음 block을 peek하는 방식을 통해 constant 시간으로 재할당할 수 있도록 했어야 하는데, 처음 짠 로직에서는 이전 block까지 peek해 coalescing하도록 잘못 최적화했어서 고치는 데 시간이 걸렸습니다.

3. What was surprising

- 수업시간에 utilization과 thoughput은 주로 함께 가져가기 어려운 conflicting하는 가치라고 배웠는데, implicit → explicit → segregated list 순으로 구현해 보니 변화하는 수치를 눈으로 확인할 수 있어서 놀라웠습니다.

- First fit에 비해 best fit을 활용하는 것이 utilization이 조금 더 높게 나오고 throughput에는 별 영향을 미치지 않아서 놀라웠습니다. Heap 공간의 free block들을 크기에 따라 다른 free list로 나누다 보니, best fit을 찾기 위해 하나의 free list를 순회하는 오버헤드가 그렇게 크지 않았던 것 같습니다.