Multicore Programming Project 3

담당 교수 : 최재승

이름 : 이우진

학번 : 20181669

1. 개발 목표

본 프로젝트에서는 C언어에서 제공되는 explicit allocator인 malloc package을 구현한다. Malloc package의 malloc(), realloc(), free()을 구현하고, 성능을 평가한다.

2. 개발 내용

본 프로젝트에서 user에게 제공해야 하는 함수는 다음과 같다.

- mm_malloc() : C 언어의 malloc()과 같은 기능을 수행한다. User가 size을 parameter로 넘겨 함수를 호출하면, 함수 내에서는 heap에 size 만큼의 공간을 확보하여 그 주소를 user에게 반환한다.
- mm_free() : C 언어의 free()와 같은 기능을 수행한다. User가 malloc을 이용해 반환 받은 주소를 free parameter로 넘겨 함수를 호출하면, 함수 내에서는 해당되는 공간을 추후에 다시 재사용 할 수 있도록 할당 해제한다.
- mm_realloc() : C 언어의 realloc()과 같은 기능을 수행한다. User가 할당 받은 공간의 포인터, size을 parameter로 넘겨 함수를 호출하면, 함수 내에서는 전달받은 포인터를 size만큼의 공간으로 다시 할당한다. 이 때, 전달받은 공간에 데이터가 있었을 경우 해당 데이터를 새로이 할당된 공간에 최대한 보존하여야 한다.

본 프로젝트에서, 데이터의 저장은 모두 1 word 단위, 여기서는 4 byte 단위로 이루어지며 user에게 반환되는 pointer는 2 word alignment가 이루어져야 한다.

3. 개발 방법

위 함수를 개발하기 위해, heap을 block 단위로 나누어 관리한다. Block의 종류에는 allocated block과 free block이 있는데, 각각 user에게 할당되어 사용될 수 있는 공간과 할당되지 않은 공간을 의미한다. 본 프로젝트에서, block의 크기는 8 byte 단위로 관리한다.

Heap의 효율적인 관리를 위해 free list을 사용한다. 일반적으로 사용되는 free list에는 구현 방법에 따라 implicit free list, explicit free list, segregated free list 가 존재한다. Free list에는 현재 heap에서 할당이 가능한 공간, 즉 user의 요청에 따라 할당되지 않았거나 할당이 해제된 공간인 free block 을 저장한다. 세 가지 방법 모두 linked list을 사용하여 구현한다는 점에서 유사한 점이 있다. 이를 위해 공통적으로 malloc package에서 block을 효율적으로 관리할 수 있도록 header, footer을 사용한다. 각각 1 word의 공간을 차지하며, header는 유저에게 할당된 공간의 앞 공간, footer는 유저에게 할당된 공간의 뒤 공간을 차지한다. Header와 footer에는 header와 footer가 포함된 block의 size와, 현재 block이 allocated block인지, free block인지를 저장한다. 저장된 block의 size을 이용해 next block의 pointer나 previous block의 header pointer을 다음과 같이 계산할 수 있으며, 이를 이용해 block간에 linked list을 유지한다.

$$Next\_head\_ptr = head\_ptr + size\_of\_block$$

Previous_head_ptr = head_ptr – size_of_previous_block

Block size와 할당 여부를 저장하기 위해 2 word을 사용 하는 것은 공간적으로 손해가 크므로, 1 word의 공간에 동시에 저장한다. Block의 size가 8 byte 단위로 정해지므로, 4 byte의 공간에 size을 저장하고 나면 least significant bit에는 아무런 데이터가 저장되어 있지 않다. 따라서, 해당 공간에 할당된 여부를 저장할 수 있다.

Implicit free, list, explicit free list, segregated free list는 상세 내용과 평균적인 성능에 차이가 있다. 각각의 구현 방법을 간략히 설명하면 다음과 같다.

- Implicit free list : implicit free list는 block단위로 관리되는 heap에서 모든 block을 linked list로 이어 관리한다. 새로이 할당할 block을 찾기 위해서 header와 footer에 저장된 정보를 이용해 block을 순회하면서 할당되지 않은, 충분한 크기의 block을 탐색한다.
- Explicit free list : explicit free list는 implicit free list와 달리 free block만을 관리하는 linked list을 관리한다. 이를 위해, header와 footer에 더해 previous link와 next link을 저장하는 공간을 추가한다. Implicit list와 달리 새로이 할당할 block을 찾기 위해 free block만을 탐색하기 때문에 성능적으로 우위에 있지만 link을 저장하는 공간이 필요하기 때문에 공간적으로 약간의 손해를 감수해야 한다.
- Segregated free list : segregated free list는 모든 free block을 하나의 linked list로 관리했던 explicit list와 달리, block의 크기에 따른 별개의 linked list을 사용한다. 원하는 크기의 공간을 보다 빠르게 찾을 가능성이 높기 때문에 explicit list에 비해 성능적으로 우위에 있고, 10개에서 20개 정도의 block만을 추가로 사용하기 때문에 공간 손해도 심하지 않다.

본 프로젝트에서는 implicit free list을 이용하여 먼저 구현하고, 구현된 내용을 기반으로 Segregated free list을 구현한다.

앞서 기술한 내용을 구현하기 위해, 다음과 같은 함수를 추가로 구현한다.

- mm_init : heap의 초기화를 위한 함수이다. Heap에 어느 정도의 공간을 할당하고, 할당된 공간을 추후 사용할 수 있도록 free list에 저장한다.
- extend_heap : heap에 추가적인 공간을 할당 받기 위한 함수이다. 추가적인 공간을 할당 받고 나서, 바로 앞의 공간과 block을 합쳐서 관리할 수 있다면 합쳐서 관리한다.
- Coalesce. : 현재 block의 앞부분과 뒷부분에 할당 되지 않은 block이 있을 경우 합쳐서 하나의 block으로 관리할 수 있게 한다.
- find_fit : malloc 또는 realloc에서 필요한 공간을 찾아 반환하는 함수이다.

- Place : find_fit을 이용해 찾은 block을 allocated block으로 변환하거나, size 만큼의 공간을 분리한 뒤 변환하는 함수이다. 남은 block은 다시 free list에서 관리한다.