System Programming Project 4

Mallocator

담당 교수 : 김영재 교수님

학 번 : 20211523

이 름 : 김 종 원

1. 개요

이번 보고서에는 프로젝트에서 개발한 동적 메모리 할당을 도와주는 Mallocator의 설계하고 구현한 방법에 대해 작성하였습니다. 메모리를 할당하는 mm_malloc과 할당된 메모리 사용 가능한 상태인 free 상태로 만들어주는 mm_free, 이미 할당된 메모리를 다른 크기로 할당해주는 mm_realloc 함수 등을 C언어로 구현했다. 또한, 프로젝트의 목표인 정확성인 valid, 공간 활용도인 utilization과 속도를 나타내는 처리율인 throughput을 최대한 높이면서도 다른 사항이 떨어지지는 않게 균형을 잡아가면서 개발을 진행했다..

구현의 핵심 전략으로는 Segregated Free Lists를 채택해 다양한 크기의 메모리 요청을 효율적으로 처리하고자 했다. 또한, Best-Fit 정책을 이용해서 정확히 필요한 만큼의 메모리 크기의 메모리만 할당하여 utilization를 높이는 전략을 선택했다. 마지막으로 coalescing와 splitting을 사용하여 메모리가 단편화되는 것을 최대한 막아 utilization가 떨어지는 것을 최소화했다.

2. 개발 범위 및 내용

1. Design of my Allocator

먼저 기본으로 제공된 mm-naive.c 코드에는 mm_malloc 요청 시에 단순하게 mem_sbrk를 호출하여 heap을 확장하고, mm_free는 아무 동작도 하지 않아서 메모리 재사용이 전혀 이루어지지 않는 비효율적인 구조였다. 그래서 mm.c에서는 이러한 문제를 해결하고 utilization와 throughput을 모두 최적화하기 위해서 Explicit Free List방식 중에서도 분리 free 리스트인 Segregated Free Lists를 아키텍쳐로 정해서 설계했다.

1.1. Segregated Free Lists

다양한 크기의 메모리 요청에 효율적으로 대응하기 위해서 free 블록들을 크기별로 분류하여 관리하는 Segregated Free Lists 방식을 채택했다.

이 방식으로 하면 mm_malloc 호출 시에 전체 heap을 탐색할 필요 없이, 요청된 크기가 속한 리스트만 탐색하면 되므로 throughput을 크게 향상시킨다. 동시에, 이후 설명할 Best-Fit과 같은 정책을 통해 utilization 또한 높일 수 있는 균형잡힌 구조이다.

어떻게 구현했는지 예시로 표현하려고 한다.

segregated_free_lists[0] -> [16byte free block] <=> [16B free block]

segregated_free_lists[1] -> [32byte free block]

segregated_free_lists[2] -> NULL

...

segregated_free_lists[19] -> [16*(2^19)byte free block]

1.2. 자료 구조 (Data Structures)

1.2.1. 블록 구조 (Block Structure)

블록의 재사용과 병합을 위해 모든 블록에 할당과 free할 때 사용할 Header와 Footer를 추가했다.

- **Header 및 Footer (4 bytes each)**:

  블록의 전체 크기인 size와 최하위 비트를 이용한 할당되어 있는지 여부를 allocated bit에 할당정보를 저장한다. Header는 현배 블록의 앞에 위치한 블록이 할당되어있는지를 알 수 있게 해주고, Footer는 현재 블록의 뒤에 위치한 블록이 이전 블록의 정보를 쉽게 파악하고 블록을 병합하는 이후에 말할 coalesce하는 데 핵심적인 역할을 하게 된다.

- **Allocated Block**:

  Header와 Footer 사이에 실제 데이터가 저장되는 Payload가 위치합니다.

- **free 블록**:

  할당되었던 메모리 공간을 free하면 Payload 공간을 재활용하여 free 리스트 상의 이전 블록(PRED)과 다음 블록(SUCC)의 주소를 저장하는 8바이트 포인터를 2개 배치했습니다. 이를 통해 Explicit Doubly Linked List를 구현했다.

어떻게 구현했는지 예시로 표현하려고 한다.

-Allocated Block: [**Header** (size|1) | Payload ... | **Footer** (size|1)]

-Free Block: [**Header** (size|0) | PRED Pointer | SUCC Pointer | ... | **Footer** (size|0)]

1.2.2. 전역 변수 (Global Variables)

전역 변수는 하나만을 사용했습니다.

- **static void** segregated_free_lists:**

  이 전역 변수는 segregated_free_lists의 시작점들을 저장하는 포인터 배열을 가리키는 전역 포인터입니다. segregated_free_lists의 메모리 공간은 mm_init 함수 내에서 mem_sbrk를 통해 heap 영역에 동적으로 할당된다.

2. 핵심 함수 및 동작 원리 (Description of Subroutines)

- **-1- mm_init**

단순히 0을 반환하던 기존 함수 mm_init을 Mallocator의 초기 상태를 설정하는 역할을 한다.

먼저 free 리스트 초기화해준다. mem_sbrk를 호출하여 segregated_free_lists를 위한 공간을 LIST_LIMIT * WSIZE 만큼의 크기로 heap에 할당하고, 모든 리스트의 시작점을 NULL로 초기화해준다.

다음으로, heap 초기화해준다. heap의 시작 부분에 정렬을 위한 패딩과 heap의 시작을 나타내는 프롤로그 블록과 끝을 나타내는 에필로그 블록을 생성한다. 이를 통해 heap의 시작과 끝을 알 수 있게 하여 나중에 heap의 크기를 늘려야 할 때 사용한다.

초기에 미리 heap을 확장해준다. extend_heap 함수를 호출하여 CHUNKSIZE 만큼 heap을 미리 확장하고, 생성된 free 블록을 free 리스트에 추가하여 malloc 요청에 즉시 대응할 수 있도록 준비해준다.

- **-2- mm_malloc**

요청 시마다 mem_sbrk를 호출하던 원래의 방식과 달리, 다음과 같은 절차에 따라 진행되게 만들었다.

1. 요청받은 size에 Header와 Footer를 보고 크기와 정렬 조건을 반영해서 실제로 할당받을 크기인 asize를 계산한다.

2. find_fit(asize) 함수를 앞서 얻은 asize를 이용해서 free 리스트에서 요청을 만족하는 블록을 탐색한다.
3. 적절한 블록을 찾으면 place(bp, asize)를 호출해서 해당 블록에 메모리를 할당하고, 해당 블록의 모든 부분을 할당하면 상관없지만 남은 부분이 있다면 남은 부분은 필요한 만큼만 분할하여 사용하고 새로운 free 블록을 만든다.
4. 만약에 적절한 free 블록이 없다면, extend_heap으로 heap을 확장한 뒤 새 공간에 블록을 할당한다.
5. 최종적으로 place 함수가 반환하는 할당된 블록의 정확한 주소를 사용자에게 반환해준다.

- **-3- mm_free**

아무 동작도 하지 않던 mm_free와 달리, 메모리 재사용을 위한 역할을 만들어 주었다.

1. 사용자로부터 받은 포인터 ptr에 해당하는 블록의 Header와 Footer에 할당 비트를 0으로 변경하여 free 상태로 만듭니다.
2. 그다음에 coalesce(ptr) 함수를 호출하여, 이 블록이 인접한 다른 free 블록과 합쳐질 수 있는지 확인하고 병합을 수행한다.

- **-4-coalesce (블록 병합)**

추가적으로 정의한 새로운 블록 병합용 보조 함수이다.

- bp 위치의 해제된 블록의 이전 블록(PREV_BLKP)과 다음 블록(NEXT_BLKP)의 할당 상태를 확인합니다.
- 다음 4가지 경우(1. 양쪽 모두 할당 / 2. 다음만 free / 3. 이전만 free / 4. 양쪽 모두 free)에 따라서 다르게 행동하게 된다. 인접한 free 블록들을 하나의 더 큰 free 블록으로 병합하여, 외부 단편화인 External Fragmentation 문제를 해결한다.
- 병합된 최종 free 블록을 add_to_free_list를 통해 free 리스트에 추가해준다.

- **-5- find_fit (블록 탐색)**

- utilization를 높이기 위해 최적 적합 방식인 Best-Fit 정책을 사용했다. 요청한 크기가 속한 그룹부터 탐색을 시작해서, 할당 가능한 블록 중에서 가장 크기가 작은 블록을 찾아 반환한다.
  이를 통해 최대한 낮은 용량만 사용하여 util 점수를 많이 받으려고 하였다.

- **-6- place (배치)**

- 찾은 free 블록을 할당하고 남는 공간이 최소 크기인 블록의 크기보다 크거나 같으면 분할하여 새로운 free 블록으로 만든다.
- 그런데 96byte보다 작은 크기의 메모리 할당 요청에 대해서는 큰 free 블록의 뒷부분을 할당해주는 방식을 적용해서 heap 앞부분의 큰 free 공간을 최대한 보존하도록 하였다. 이를 통해 utilization을 최대한 최적화하려고 했다.

- **-7- mm_realloc**

단순히 malloc, memcpy, free를 호출하던 비효율적인 원래 버전과 다르게 다음과 같은 단계를 거쳐서 재할당을 수행하도록 하였다.

1. **크기 축소**:

   요청된 크기가 기존보다 작으면, 현재 블록을 분할하여 공간 낭비를 줄인다.

2. **인접 free 블록 활용**:

   확장할 때 재할당하려는 블록의 다음 블록과 이전 블록 또는 양쪽 블록 모두가 free 블록이고 공간이 충분하다면, 이들을 병합하여 새로운 malloc 호출 없이 재할당을 완료한다.

3. **heap 끝 확장**:

   재할당하려는 블록이 heap의 가장 마지막에 위치한 경우에는 재할당하여 늘려줄 수 없으므로 mem_sbrk를 호출하여 필요한 만큼만 heap을 확장해준다. 이는 Fragmentation을 전혀 발생시키지 않는 방법이다.

4. **mm_malloc**:

   위의 과정의 모든 최적화가 불가능할 때만 mm_malloc으로 새 공간을 할당하고 데이터를 복사한 뒤 기존 블록을 mm_free 해준다.