System Programming Project 4

담당 교수: 김영재

이름: Jumagul Alua

학번: 20231632

1. 개발 목표

이번 프로젝트의 목표는 malloc, free, realloc 기능을 직접 구현하여 운영체제 수준의 메모리 관리자로서 동작하는 동적 메모리 할당자를 개발하는 것이었다. 수업에서 배운 네 가지 기법 implicit free list, explicit free list, segregated free list, sorted by size free list — 을 비교하며, 성능 향상과 단편화 최소화를 동시에 달성할 수 있는 방법을 탐색하였다.

여러 방법 중 segregated free list 방식이 공간 활용도(utilization)와 처리량(throughput) 모두에서 가장 우수한 성능을 보였기 때문에, 본 구현에서는 이 기법을 선택하였다. segregated list 구조를 통해 블록 검색 속도를 줄이고, 적절한 크기의 블록을 빠르게 할당하는 것이 가능해졌으며, realloc 함수도 가능한 한 in-place로 동작하도록 하여 불필요한 복사를 줄였다.

2. 개발 내용

이번 프로젝트의 성능 개선을 위해 가장 중점적으로 진행한 부분은 메모리 블록의 관리 방식을 암시적 리스트(implicit free list) 방식에서 크기 기반의 Segregated Free List 방식으로 변경한 점이다. 이를 통해 블록 탐색 시간을 획기적으로 줄일 수 있었으며, 공간 활용률(peak utilization)을 향상시키는 데에도 크게 기여하였다.

우선, 전역 배열을 직접 정의하는 방식은 과제의 제한 사항에 위배되므로, 전역 포인터 변수 void **segregated_lists 하나만을 선언하였다. 그리고 mm_init() 함수 내에서 mem_sbrk()를 사용해 힙 영역에 포인터 배열 공간(LIST_LIMIT 개수)을 동적으로 할당하였다. 이렇게 동적으로 확보한 포인터 배열을 통해 각 크기 구간별 free block들을 관리하였으며, segregated_lists[i] 형태로 배열처럼 접근 가능하도록 하였다.

리스트 인덱스를 계산하기 위해 get_list_index() 함수를 작성하였다. 이 함수는 주어진 블록의 크기를 2로 나누는 방식을 반복하여 log scale로 인덱스를 산출하며, 그 결과 크기가 클수록 상위 인덱스의 리스트에 배치되도록 설계되었다. 이를 통해 비슷한 크기의 블록끼리 리스트에 모여 검색 효율이 높아지도록 하였다.

free 블록을 리스트에 삽입할 때는 insert_node() 함수를 사용하며, 해당 블록은 리스트의 head에 삽입된다. 각 리스트는 양방향 연결 리스트로 구성되며, 삽입 시에는 이전/다음 블록 포인터를 조정하여 연결을 유지하도록 구현하였다. 반대로 remove_node() 함수는 리스트에서 블록을 제거할 때 사용되며, 포인터 연결을 안전하게 해제한 뒤 블록을 리스트에서 제거한다.

할당 요청이 들어오면 find_fit() 함수에서 적절한 free 블록을 탐색한다. 이 함수는 best-fit 전략

을 사용하여 블록 크기보다 크거나 같은 블록 중 차이가 가장 작은 블록을 선택한다. 이를 통해 내부 단편화를 줄이고, 전체적인 공간 활용률을 높이고자 하였다. 작은 블록일수록 낮은 인덱스 리스트에 위치하므로, 낮은 인덱스부터 순차적으로 검색하여 적합한 블록을 빠르게 찾을 수 있도록 하였다.

블록을 실제로 할당할 때는 place() 함수를 통해 필요한 크기만큼 분할하여 사용하고, 남은 부분은 새로운 free 블록으로 만들어 다시 적절한 리스트에 삽입한다. 단, 남은 공간이 최소 블록 크기보다 작은 경우에는 분할하지 않고 전체 블록을 할당하여 지나치게 작은 조각이 생기는 것을 방지하였다.

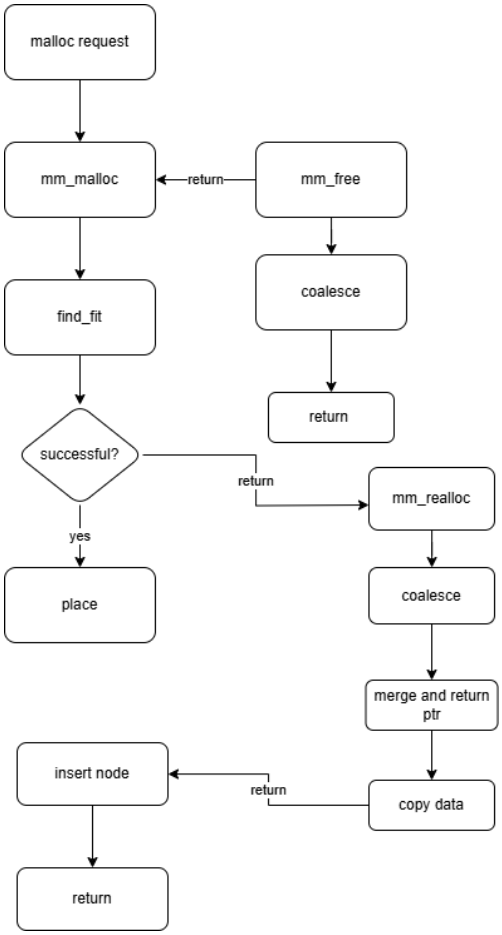
블록 해제는 mm_free() 함수에서 수행되며, 블록을 free 상태로 마킹한 뒤 coalesce() 함수를 호출하여 앞뒤 인접 블록이 free 상태일 경우 병합을 시도한다. 병합 가능한 블록들은 remove_node()를 통해 리스트에서 제거한 후, 병합된 새로운 블록을 다시 리스트에 삽입하여 단편화를 최소화하고, 큰 블록을 확보할 수 있도록 하였다.

또한 mm_realloc() 함수는 in-place 확장을 우선적으로 시도하도록 구현하였다. 재할당 시 다음 블록이 free 상태이고 충분한 공간을 확보할 수 있는 경우, 기존 블록과 병합하여 포인터를 이동시키지 않고 재할당을 처리한다. in-place 확장이 불가능한 경우에는 새로운 블록을 할당하고 데이터를 복사한 뒤, 원래 블록을 해제하는 방식으로 동작한다.

마지막으로 extend_heap() 함수는 힙의 공간이 부족할 때 추가 메모리를 확보하는 역할을 하며, 새로 생성된 블록은 초기화된 뒤 병합 과정을 거쳐 리스트에 등록된다.

이와 같은 방식으로 기존의 단순한 암시적 리스트 기반의 allocator 구조를 전면적으로 개선하였으며, 그 결과 성능 점수를 53점에서 90점까지 향상시킬 수 있었다. 특히 공간 활용률과 처리량 간의 균형을 중점적으로 고려하며 구현을 최적화하였다.

3. **플로우 차트**



4. 구현 결과

초기에는 가장 단순한 방식인 implicit free list 기반으로 구현하여 baseline 점수인 약 53점을 기록하였다. 이 방식은 구현이 간단하지만, 블록 검색 시 전체 힙을 선형 탐색해야 하므로 처리량이 떨어지고, 단편화가 심해 효율이 낮았다.

이후에는 성능 개선을 위해 segregated free list 방식으로 전환하였다. 이 방식은 free 블록들을 크기별로 분류하여 여러 개의 리스트에 나누어 저장함으로써, 할당 요청 시 전체 힙을 순회하지 않고도 빠르게 적절한 블록을 찾을 수 있다는 장점이 있다. 이를 통해 최종적으로 90점이라는 높은 점수를 기록할 수 있었다.

```
cse20231632@cspro:~/prj4-malloc$ ./mdriver -v
[20231632]::NAME: Alua Jumagul, Email Address: alua@sogang.ac.kr
Using default tracefiles in ./tracefiles/
Measuring performance with gettimeofday().

Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   99%    5694  0.000692  8227
 1       yes   99%    5848  0.000609  9603
 2       yes   99%    6648  0.000636 10453
 3       yes  100%    5380  0.000464 11597
 4       yes   66%   14400  0.000627 22970
 5       yes   96%    4800  0.001019  4709
 6       yes   95%    4800  0.001030  4661
 7       yes   55%   12000  0.000670 17916
 8       yes   51%   24000  0.001622 14799
 9       yes   87%   14401  0.000329 43732
10       yes   67%   14401  0.000246 58612
Total          83%  112372  0.007944 14146

Perf index = 50 (util) + 40 (thru) = 90/100
```