# Multicore Programming Project 3

담당 교수 : 박성용

이름 : 백상욱

학번 : 20190388

## 1. 개발 목표

동적 메모리 할당에 사용되는 allocator알고리즘을 구현하고 이에 사용되는 malloc, free, realloc함수를 직접 구현한다. 이때 구현한 allocator에서 높은 throughput과 memory utilization을 높이는 것을 목적으로 한다.

## 2. 개발 범위 및 내용

### A. 개발 범위

1. 동적 메모리 할당을 위한 implicit list, explicit list, segregate list 중 implicit list 방식을 채택하여 구현한다. 그리고 이를 이용하여 malloc, free, realloc을 구현하기 위해서 다른 함수들을 함께 구현한다.
2. 위의 구현 결과를 가지고 mdrive를 실행시켜 memory utilization과 throughput에 대한 성능을 측정한다.

### B. 개발 내용 및 개발 방법

- **Macro :** 우선 다양한 포인터 연산과 워드 단위 연산을 하기 위해서 macro를 정의해둔다. 다만 교과서에 나온 내용과 동일한 매크로는 설명을 생략한다.
  ALIGNMENT는 double word align을 맞춰야하므로 8로잡는다. 그리고 이를 이용한 ALIGN이라는 매크로를 정의하는데, 이는 특정 사이즈를 줬을 때 align된 사이즈를 반환해주는 매크로로 (((size) + (ALIGNMENT-1)) & ~0x7)와 같이 구현한다.
  SIZE_T_SIZE라는 매크로는 ALIGN과 유사하게 size를 align되어서 반환하는 매크로다. 그리고 Explicit 리스트에서 prev와 next block에 대한 정보를 저장하는 블럭을 가리키는 매크로 NEXT_EX과 PREV_EX를 정의한다. PRE_EX는 GET(HDRP(bp) + WSIZE) PREV_EX는 GET(HDRP(bp) + DSIZE)로 정의한다.
- static variable : 스태틱 변수를 한가지 추가한 것이 있는데 free_listp이다. 이는 free_list를 가리키는 포인터 역할을 하게 된다.
- **mm_init 함수 :** mm_init은 초기 힙상태를 초기화하는 함수인데, 기존의 mm_init과 거의 유사하지만 explicit의 구현을 위해서 초반에 4개의 block이 아닌 6개의 블럭을 할당했 다. 그리고 2개를 explicit 리스트의 구현을 위해서 사용했는데, prologue header, next,prev prologue footer, expilogue로 사용하게 된다. 그리고 free_listp를 heap_listp +double word 로 설정한다. 이로서 free_list의 시작을 구현

할 수 있게 되었다.

- **mm_malloc :** explicit 리스트에 대한 관리는 find_fit, coalsce, place에서 구현하고 말록 함수의 경우 기존 교과서에 나온 코드와 동일한 코드를 사용했다.

- mm_free : mm_free함수의 malloc과 마찬가지로 마찬가지로 교과서 코드를 사용했다. 헤더와 푸터의 alloc에 0을 할당하고 다른 블럭과 합칠 수 있는지 확인하기 위해 coalesce를 호출한다.

- **mm_realloc :** 크기가 0이라면 free를 의미하는 것이므로 ptr을 free에 넣어준다. ptr이 NULL이라면 단순한 malloc을 의미하는 것이기 때문에 size만큼 malloc을 수행해서 포인터를 반환한다. 그리고 만약 할당해야하는 크기가 현재 할당된 크기보다 작다면 그냥 기존의 포인터를 반환한다. 이 외의 경우에 대해서 새롭게 할당해야하는데, 최적화를 위해서 현재 블럭의 다음 블럭을 확인한다.만약 다음 블럭이 할당되어있지 않다면 그리고 해당 블럭까지의 크기를 현재 블럭의 크기에 더했을 때, 할당하려는 크기보다 커서 할당이 가능하다면 다음블럭을 합쳐서 거기에 할당한다. 이렇게 하면 새롭게 할당하는 것보다 메모리 utilization을 높일 수 있는 것을 실험을 통해 확인했다. 이 외의 경우에는 그냥 새롭게 메모리를 할당한 후에, 새로운 포인터로 기존의 내용을 memcpy로 복사한 후 기존 포인터는 free하고 새로운 포인터를 반환한다.

- extend_heap() : 교과서에서 사용한 함수와 동일하다. malloc과 realloc이 fit을 찾기 못하고 추가 공간을 필요로하면 그때 호출한다.

- place() : place 함수는 malloc에서 호출되며 asize만큼의 블록을 free에서 찾아내어 할당하는 함수다. 해당 크키만큼 할당했을 때의 블럭 사이즈인 csize를 계산한한ㄷ. 그리고 할당했을 때, 남아있는 블럭의 크기를 rsize로 저장한다. 할당하고 남은 부분은 split해서 저장하기 위함이다. 이때 경우는 3가지가 존재하는데, 첫 번째, 남는 공간이 minimal block size인 4word보다 작은 경우, 이때는 남는 공간을 할당하는게 무의미하기 split하지 않는다. 두번째, 세번째 경우를 고려하기 위해서 기준이 되는 크기를 지정해야한다. explicit과 관련된 자료를 찾아봤을 때, explicit의 메모리 util을 높이지만, 너무 느리지는 않게 높이는 방법으로 이를 선택하는데 가령 50byte를 기준으로 잡는다면, 할당하려는 블럭의 크기가 50바이트 이상인지 이하인지 판단하여 이보다 크다면 risze만큼의 free block을 앞에 배치하고, 이보다 작다면, free block을 뒤로 할당한다. 다양하게 시도해본 결과 100바이트 정도에서 성능이 가장 높게 나오는 것 같아 구현에서는 100을 기준으로 삼았다.

그래서 100을 기준으로 판단하고 free block에 대해서는 헤더와 푸터 정보를 삽입한다.

- find fit : 찾는 방법으로 first fit을 구현했다. first fit을 구현하기 위해서 explicit list의 시작 수조인 free_listp부터 시작해서 헤더를 확인하여 get alloc이 0인 동안 while문을 반복하는데, 내부에서는 getsize로 크기를 구하고, 할당하려는 크기인 asize보다 크다면 해당 위치에 할당하면 되므로그때의 포인터를 반환한다. 그리고 만약 크기가 작다면, NEXT_EX로 다음 free list로 넘어간다.

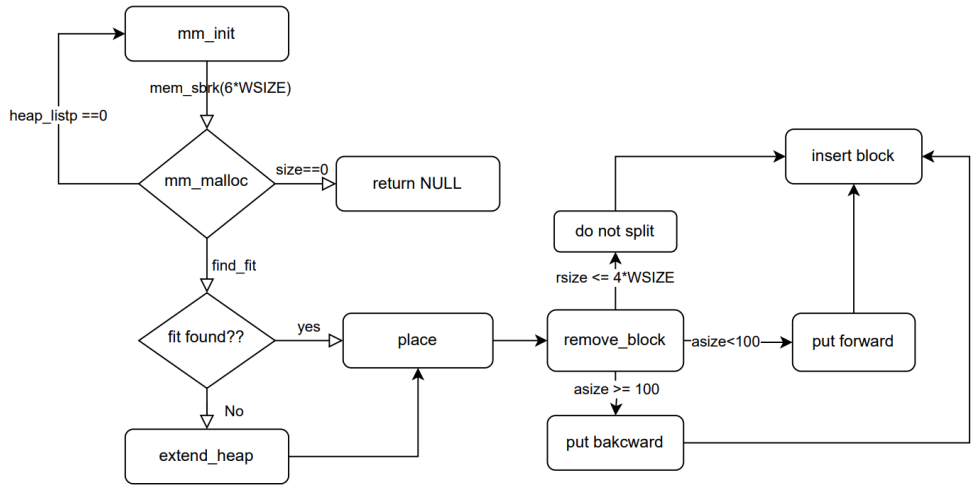
- coalesce : coalesce에서는 4가지 경우를 고려할 수 있는데, 앞뒤로 할당된 경우, 뒤만 할당된 경우, 앞만 할당된 경우, 앞뒤로 할당되지 않은 경우, 이때에 각각 맞춰 합춰준다. 그리고 이때 합춰주는 과정에서 푸터와 헤더의 정보를 수정하는 것 뿐 아니라 할당이 되어있지 않은 free block을 free list에서 제거하는 함수인 remove_block을 호출해야한다. 그리고 새롭게 할당된 하나의 블럭을 다시 free list에 넣기 위해 insert_block을 호출한다.

- insert_block : 블럭을 삽입하는 알고리즘으로는 LIFO를 사용한다. 즉 가장 마지막에 삽입한애가 가장 앞이고, free_listp와 연결되어야한다. 이를 위해서 해당 포인터의 PRE_EX로는 null을 삽입하고 NEXT_EX는 Free_listp(현재 freelistp가 가리키는 블럭)을 할당하고 free_listp의 PRE_EX를 해당 블럭으로 연결한다. 그리고 free_listp에 현재 블럭을 할당하여 가장 앞이 되도록 한다.

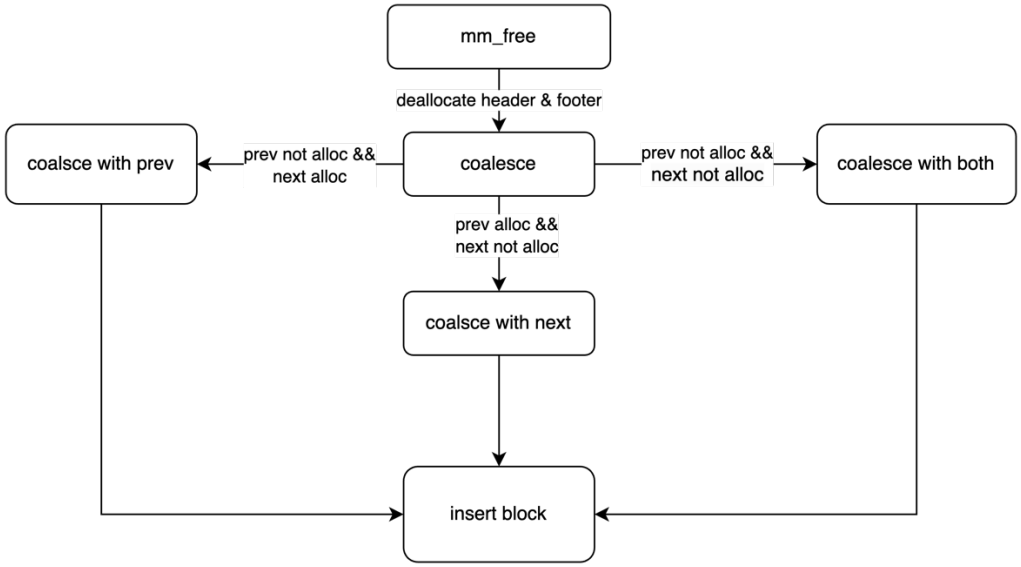
- remove_block : 해당 블럭이 할당되었을 때, free에서 없애주는 함수다. 이때 경우를 두가지로 나누어야하는데, 첫 번째는 해당 블럭이 free list의 첫번째 블럭인 경우이다. 이는 해당 블럭의 PRE_EX가 null인지 체크해서 확인한다. 이경우라면 free_listp가 가리키는 부분을 다음인 NEXT_EX로 바꾸고, NEXT_EX의 PRE_EX를 null로 만들어서 맨처음으로 만들어준다.

## 3. Flow chart

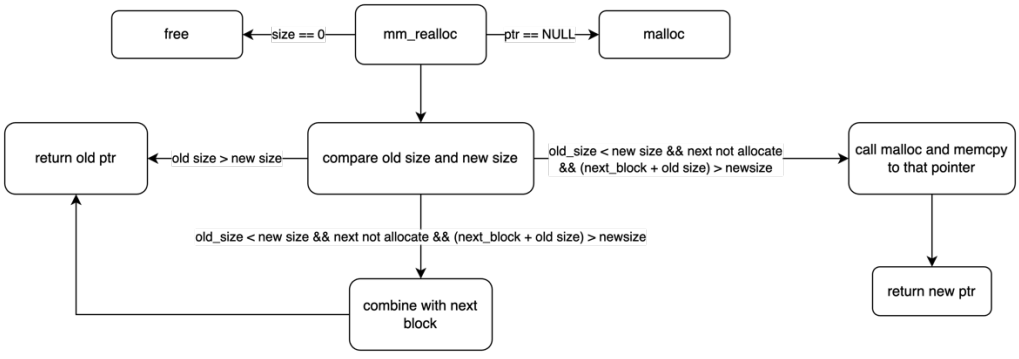
### <malloc>의 flow chart




### <free>의 flow chart




### <realloc>의 flow chart



## 4. 구현 및 결과 및 성능 평가

free block을 구현하고 추적하기 위해서 explicit list를 구현했고, 이를 위해서 mm_init,mm_malloc,mm_realloc,mm_free외에도place,coalesce,insert_block, remove_block등을 구현했고, mdriver를 수행한결과는 다음과 같다.

```
Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   89%    5694  0.000200 28413
 1       yes   92%    5848  0.000189 31024
 2       yes   95%    6648  0.000385 17254
 3       yes   96%    5380  0.000275 19542
 4       yes   66%   14400  0.000242 59627
 5       yes   88%    4800  0.000666  7203
 6       yes   86%    4800  0.000966  4969
 7       yes   86%   12000  0.002362  5080
 8       yes   89%   24000  0.000967 24829
 9       yes   63%   14401  0.000290 49676
10       yes   45%   14401  0.000198 72659
Total          81%  112372  0.006740 16672

Perf index = 49 (util) + 40 (thru) = 89/100
```

throughput 같은 경우는 간단한 수준의 알고리즘인 explicit free list를 선택했다는 점과 find 알고리즘도 비교적 빠르고 간단한 first fit알고리즘을 사용했다는 점에서 만점을 받았다. 하지만 특정 범위를 정해서 배열을 관리하는. seglist와 다르게 일괄로 관리하는 explicit list의 특징과 throughput을 위해서 best fit을 사용하지 않고 first fit을 사용했기 때문에 external fragment가 비교적 자주 발생했을 수 있고 util 점수는 49점에 그치는 한계를 보였다.