System Programming Project 3

담당 교수 : 김영재 교수님

이름 : 전용본

학번 : 20181683

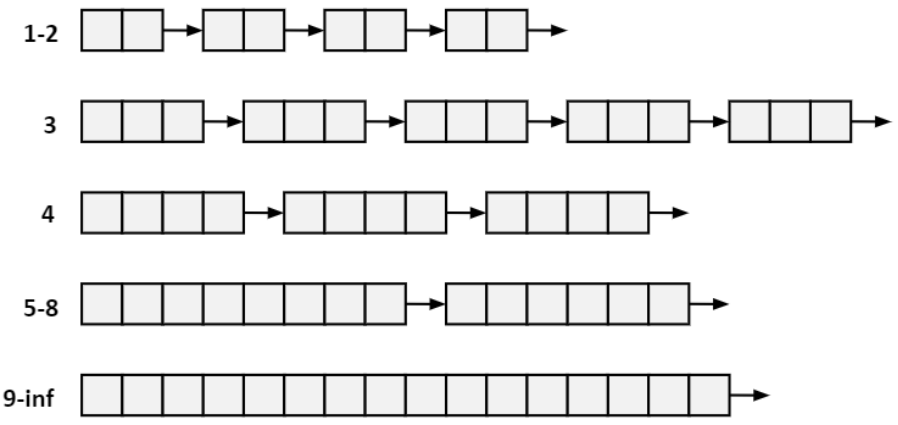
1. 개발 목표

My dynamic malloc allocator를 개발하는 것이 목표이다. dynamic malloc allocator는 libc에서 제공하는 malloc, free, realloc 등의 package로 heap 영역을 관리하는 도구이다. 강의 중에 배웠던 Implicit free list, explicit free list, segregated free list, buddy system 중에서 Implicit free list, explicit free list는 memory utilization, throughput이 균형있게 만족할만한 performance가 나오지 못함을 알고 segregated free list 방식으로 allocator를 구현하려한다.

2. 개발 범위 및 내용

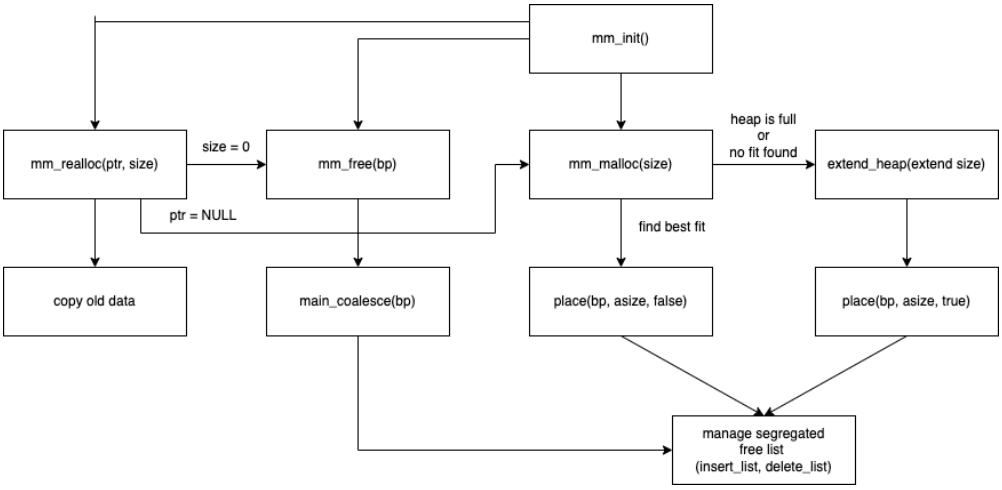
A. 개발 범위

Segregated free list는 각각 다른 size의 block들로 이루어진 여러 개의 linked list를 사용한다. 각각의 linked list는 하나의 explicit free list처럼 이루어진다. segregated free list는 할당, 해제를 constant time에 해결하는 성능을 보인다. 이 방식의 best fit은 다른 방식의 best fit보다 빠르다. 요청한 할당 size를 이용해 free list의 index를 찾아 할당할 수 있기 때문이다.



B. 개발 내용

1) Design of allocator with flow chart



2) Description for subroutines, structs, any global variables

1. 전역 변수

```
size_t segregated_lists = 10;     /* segregated free list 는 10가지 size class를 갖고 있음 */
static char **seg_listp;          /* segregated free list pointer */
```

segregated free list를 전역변수로 정의했다. segregated_lists는 list의 class의 개수를 나타낸다.

2. 함수 매크로

```
/* Free list 상에서 next, previous block pointer를 반환하는 함수 매크로 */
#define NEXT_FREE_BLKP(bp) GET(bp)
#define PREV_FREE_BLKP(bp) GET((char *)(bp)+WSIZE)

/* Free list 에 next, previous block pointer를 설정하는 함수 매크로 */
#define SET_NEXT_FREE_BLKP(bp, next_block_ptr) PUT(bp, next_block_ptr)
#define SET_PREV_FREE_BLKP(bp, prev_block_ptr) PUT((char *)(bp)+WSIZE, prev_block_ptr)
```

segregated free list안에서 block의 이동, 삽입을 다루는 함수 매크로이다.

3. 함수

A. index_list(size_t size)

size를 이용해 segregated free list의 index를 반환하는 함수이다.

B. delete_list(void *bp)

segregated free list에서 free block을 삭제하는 함수이다.

C. insert_list(void *bp)

segregated free list에서 free block을 해당하는 index의 list에 처음 block으로 삽입하는 함수이다.

D. coalesce(void *bp)

segregated free list를 관리하기 위해 새로 정의한 coalesce 함수이다. 기존의 coalesce 함수에서 segregated free list에 free block들을 추가, 삽입하는 case를 추가했다.

E. find_fit(size_t asize)

segregated free list에 맞는 find_fit을 새로 정의했다. size에 맞는 index를 획득한 뒤 그 이상의 index의 free list를 순회하며 적절한 block을 찾는다.

F. place(void *bp, size_t asize, bool extend)

extend 여부를 알 수 있는 parameter를 추가해서 정의했다. extend가 false인 경우 기존의 free list에서 가져온 block이므로 할당 후에 block을 삭제해야하고 extend가 true인 경우에는 extend_heap으로 새로 가져온 block이므로 block을 삭제하지 않아도 된다.

G. mm_check(void)

Heap space의 consistency를 검사하는 함수이다. 현재 heap에 대해서 모든 free block들이 free로 마킹되어 있는지, free list의 pointer들이 제대로 free block들을 가리키고 있는지, 모든 block들이 double-word로 align되어 있고 header와 footer가 일치하는

지, coalesce되지 않은 free block들이 존재하는지, heap 영역의 모든 free block이 free list에 들어가있는지 검사한다.

H. checkfreebock(void *bp)

free block의 유효성을 검사하는 함수이다. free block들의 next, previous pointer가 제대로 향하고 있는지 확인한다.

3) Heap consistency checker

1. Is every block in the free list marked as free ?

```
for(int i = 0; i < segregated_lists; i++){
    if(seg_listp[i]!=NULL){
        for(bp = seg_listp[i]; bp!=NULL; bp = NEXT_FREE_BLKP(bp)){
            /* Is every block in the free list marked as free ? */
            if(GET_ALLOC(bp)){
                printf("Error1 : Allocated block exists in free_list\n");
                return 0;
            }
```

segregated free list를 순회하면서 allocate된 block이 있는지 확인

2. Is every block in the heap Double-word aligned ?

```
for(bp = heap_listp; GET_SIZE(HDRP(bp)) > 0; bp = NEXT_BLKP(bp)){
    /* Is every block in the heap Double-word aligned ? */
    if((unsigned int)bp % DSIZE){
        printf("Error2 : Not Double-word aligned\n");
        return 0;
    }
```

Heap 영역의 모든 block들을 순회하면서 8로 나눠보며 확인

3. **Is every block in the heap matched Header and footer ?**

```
/* Is every block in the hep matched Header and footer */
if(GET(HDRP(bp)) != GET(FTRP(bp))){
    printf("Error3 : Header does not match footer\n");
    return 0;
}
```

Heap 영역의 block들을 순회하면서 header 와 footer 같은지 확인

4. **Are there any contiguous free blocks that somehow escaped coalescing ?**

```
/* Are there any contiguous free blocks that somehow escpaed coalescing? */
if(!(GET_ALLOC(HDRP(bp))) && !(GET_ALLOC(FTRP(PREV_BLKP(bp))) && GET_ALLOC(HDRP(NEXT_BLKP(bp))))){
    printf("Error4 : consecutive free blocks\n");
    return 0;
}
```

Heap 영역의 free block들을 순회하면서 coalesce되지 않은 free block이 있는지 확인

5. **Is every free block actually in the free list ?**

```
for(bp = heap_listp; GET_SIZE(HDRP(bp)) > 0; bp = NEXT_BLKP(bp)){
if(!GET_ALLOC(HDRP(bp)))
    free_heap_num++;
```

```
for(int i = 0; i < segregated_lists; i++){
    if(seg_listp[i]!=NULL){
        for(bp = seg_listp[i]; bp!=NULL; bp = NEXT_FREE_BLKP(bp)){
free_list_num++;
```

```
/* Is every free block actually in the free list */
if(free_heap_num != free_list_num){
    printf("Error5 : some free blocks are not in free_list\n");
    return 0;
}
```

Heap 영역의 free block들을 순회하면서 개수를 확인하고 segregated free list의 free block들을 순회하면서 개수를 확인한 뒤 둘이 같은지 확인한다.

6. **Do the pointers in a heap block point to valid heap addresses ?**

```
for(int i = 0; i < segregated_lists; i++){
    if(seg_listp[i]!=NULL){
        for(bp = seg_listp[i]; bp!=NULL; bp = NEXT_FREE_BLKP(bp)){
```

```
/*Checks if the pointers in the free list point to valid free blocks*/

/* previous free block이 next free block으로 current block을 가리키고 있는가 */
if(PREV_FREE_BLKP(bp))
{
    if(NEXT_FREE_BLKP(PREV_FREE_BLKP(bp)) != bp)
    {
        printf("Error 6: Free list inconsistent");
        return 0;
    }
}

/* next free block이 previous free block으로 current block을 가리키고 있는가 */
if(NEXT_FREE_BLKP(bp) && NEXT_FREE_BLKP(bp) != heap_listp)
{
    if(PREV_FREE_BLKP(NEXT_FREE_BLKP(bp)) != bp)
    {
        printf("Error 6:Free list inconsistent");
        return 0;
    }
}
```

segregated free list의 free block들을 순회하면서 free block들의 pointer들이 서로 유효하게 가리키고 있는지 확인

3. 성능 평가

A. Implicit free list(교재 코드)

```
cse20181683@cspro:~/prj3-malloc$ ./mdriver
[20181683]::NAME: YongBon Jeon, Email Address: bon0057@naver.com
Using default tracefiles in ./tracefiles/
Perf index = 44 (util) + 10 (thru) = 55/100
```

utilization memory와 throughput 사이의 균형이 맞지 않는 형태임을 알 수 있다.

B. segregated free list(구현)

```
cse20181683@cspro:~/prj3-malloc$ ./mdriver
[20181683]::NAME: YongBon Joen, Email Address: bon0057@naver.com
Using default tracefiles in ./tracefiles/
Perf index = 47 (util) + 40 (thru) = 87/100
```

utilization memory와 throughput 사이의 균형이 implicit free list 대비 상당히 좋음을 알 수 있다.

C. libc malloc package 와 비교

```
cse20181683@cspro:~/test$ ./mdriver -vl
[20181683]::NAME: YongBon Joen, Email Address: bon0057@naver.com
Using default tracefiles in ./tracefiles/
Measuring performance with gettimeofday().

Results for libc malloc:
trace  valid  util     ops      secs  Kops
 0       yes    0%    5694  0.001859  3063
 1       yes    0%    5848  0.001448  4038
 2       yes    0%    6648  0.002331  2852
 3       yes    0%    5380  0.002077  2590
 4       yes    0%   14400  0.000432 33310
 5       yes    0%    4800  0.002851  1684
 6       yes    0%    4800  0.002546  1885
 7       yes    0%   12000  0.001280  9379
 8       yes    0%   24000  0.001172 20485
 9       yes    0%   14401  0.000669 21539
10       yes    0%   14401  0.000315 45761
Total           0%  112372  0.016978  6619

Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   99%    5694  0.000638  8932
 1       yes   99%    5848  0.000641  9118
 2       yes   99%    6648  0.000584 11386
 3       yes   99%    5380  0.000487 11049
 4       yes   99%   14400  0.000529 27237
 5       yes   95%    4800  0.001057  4542
 6       yes   94%    4800  0.001198  4008
 7       yes   60%   12000  0.000582 20633
 8       yes   53%   24000  0.006899  3479
 9       yes   27%   14401  0.108637   133
10       yes   30%   14401  0.005359  2687
Total          78%  112372  0.126610   888

Perf index = 47 (util) + 40 (thru) = 87/100
```

9번 tracefile(realloc-bal.rep)에서 좋지 못한 성능이 보였다. realloc을 더 보완할 수 있으면 좋은 malloc package가 될 수 있을 듯하다.