2018 시스템 프로그래밍
- Lab 10 -

제출일자	2017.12.18
분 반	02
이 름	박상현
학 번	201702012

naive

결과 화면 캡처(mdriver 실행 화면)

```
c2017020012@2018-sp:~/malloclab-handout$ ./mdriver
Using default tracefiles in ./traces/
Measuring performance with a cycle counter.
Processor clock rate ~= 2097.6 MHz

Results for mm malloc:
   valid  util   ops    secs     Kops  trace
   yes     94%      10  0.000000 81301 ./traces/malloc.rep
   yes     77%      17  0.000000106127 ./traces/malloc-free.rep
   yes    100%      15  0.000000 90412 ./traces/corners.rep
 * yes     71%    1494  0.000010152740 ./traces/perl.rep
 * yes     68%     118  0.000001166005 ./traces/hostname.rep
 * yes     65%   11913  0.000079150691 ./traces/xterm.rep
 * yes     23%    5694  0.000068 84039 ./traces/amptjp-bal.rep
 * yes     19%    5848  0.000072 81640 ./traces/cccp-bal.rep
 * yes     30%    6648  0.000086 76995 ./traces/cp-decl-bal.rep
 * yes     40%    5380  0.000062 86812 ./traces/expr-bal.rep
 * yes      0%   14400  0.000189 76047 ./traces/coalescing-bal.rep
 * yes     38%    4800  0.000053 90126 ./traces/random-bal.rep
 * yes     55%    6000  0.000069 86345 ./traces/binary-bal.rep
10         41%   62295  0.000689 90367

Perf index = 26 (util) + 40 (thru) = 66/100
c2017020012@2018-sp:~/malloclab-handout$
```

소스 코드

1. 매크로 설명

```
40 /* single word (4) or double word (8) alignment */
41 #define ALIGNMENT 8
42
43 /* rounds up to the nearest multiple of ALIGNMENT */
44 #define ALIGN(size) (((size) + (ALIGNMENT-1)) & ~0x7)
45
46
47 #define SIZE_T_SIZE (ALIGN(sizeof(size_t)))
48
49 #define SIZE_PTR(p)  ((size_t*)(((char*)(p)) - SIZE_T_SIZE))
```

- #define ALIGNMENT 8 : allocation에 8byte(2word) alignment를 사용한다는 뜻.
- #define ALIGN(size) (((size) + (ALIGNMENT-1)) & ~0x7) : 7에 '~'연산을 하게되면 LSB부터 총 3자리가 0이 된다. 이 때, 이를 & 연산을 통해 8의 배수로 masking한다. 따라서 이 연산의 결과는 8의 배수이다.
- #define SIZE_T_SIZE (ALIGN(sizeof(size_t))) : size_t의 size를 구한 매크로이다. 본 실습에서는 8의 값을 갖는다.
- #define SIZE_PTR(p) ((size_t*) (((char*) (p)) - SIZE_T_SIZE)) : p에 8만큼을 뺀 주소값은 p로 하여금 header를 가리키게 한다.

2. 함수 설명

```
51 /*
52  * mm_init - Called when a new trace starts.
53  */
54 int mm_init(void)
55 {
56   return 0;
57 }
```

- mm_init() : 초기화를 위한 함수이며, 초기공간을 만들고, 사용할 공간을 만든다. naive에서는 구현하지 않았다.

-
```
59 /*
60  * malloc - Allocate a block by incrementing the brk pointer.
61  *      Always allocate a block whose size is a multiple of the alignment.
62  */
63 void *malloc(size_t size)
64 {
65   int newsize = ALIGN(size + SIZE_T_SIZE);
66   unsigned char *p = mem_sbrk(newsize);
67   //dbg_printf("malloc %u => %p\n", size, p);
68
69   if ((long)p < 0)
70     return NULL;
71   else {
72     p += SIZE_T_SIZE;
73     *SIZE_PTR(p) = size;
74     return p;
75   }
76 }
```

- malloc() : size를 매개변수로 받아 newsize를 계산하는데, size에 8을 더한값을 ALIGN함으로써 double word alignment를 만족시킨다.이 때 8을 더하는 이유는 header를 추가로 만들기 위함이다. 이 8은 매크로에서 4의 값을 가지는 sizeof(size_t)값을 ALIGN을 이용하여 8로 만든 것이다. 결과적으로 header의 크기는 8을 가지게 된다. 이제 heap을 할당해야 하는데, mem_sbrk함수를 이용한다. 할당 후에 포인터인 p는 header를 가리키고 있을 것이다. 따라서, payload영역을 가리키게 하기 위해 8 즉 SIZE_T_SIZE를 더한다. 그 후 SIZE_PTR 매크로를 사용하여 header에 할당한 size만큼을 저장한다. 모두 끝난 후에 포인터 p를 리턴한다.

-
```
78 /*
79  * free - We don't know how to free a block.  So we ignore this call.
80  *      Computers have big memories; surely it won't be a problem.
81  */
82 void free(void *ptr)
83 {
84 }
```

- free() : 할당한 블록들을 반환해주는 함수이다. naive에서는 구현하지 않았다.

```
 86 /*
 87  * realloc - Change the size of the block by mallocing a new block,
 88  *      copying its data, and freeing the old block.  I'm too lazy
 89  *      to do better.
 90  */
 91 void *realloc(void *oldptr, size_t size)
 92 {
 93   size_t oldsize;
 94   void *newptr;
 95
 96   /* If size == 0 then this is just free, and we return NULL. */
 97   if(size == 0) {
 98     free(oldptr);
 99     return 0;
100   }
101
102   /* If oldptr is NULL, then this is just malloc. */
103   if(oldptr == NULL) {
104     return malloc(size);
105   }
106
107   newptr = malloc(size);
108
109   /* If realloc() fails the original block is left untouched  */
110   if(!newptr) {
111     return 0;
112   }
113
114   /* Copy the old data. */
115   oldsize = *SIZE_PTR(oldptr);
116   if(size < oldsize) oldsize = size;
117   memcpy(newptr, oldptr, oldsize);
118
119   /* Free the old block. */
120   free(oldptr);
121
122   return newptr;
123 }
```

- realloc() : 이 함수는 malloc으로 할당된 block의 size를 변경해주는 함수이다. malloc으로 반환된 포인터와 할당하고싶은 size를 매개변수로 넣고, 기존 malloc으로 할당한 size와 할당하고싶은 size(매개변수)를 비교한다. memcpy()를 사용하여 새로이 반환할 포인터에 malloc으로 반환된 포인터 값을 복사하고, malloc으로 반환된 포인터를 free시켜준다. naive에서 free는 물론 작동하지 않는다(구현을 안했으니). 마지막으로, 새로이 만든 포인터를 리턴한다.

```
125 /*
126  * calloc - Allocate the block and set it to zero.
127  */
128 void *calloc (size_t nmemb, size_t size)
129 {
130   size_t bytes = nmemb * size;
131   void *newptr;
132
133   newptr = malloc(bytes);
134   memset(newptr, 0, bytes);
135
136   return newptr;
137 }
```

- calloc() : 이 함수는 malloc을 함에 있어서 할당한 값들을 모두 0으로 설정해주는 함수이다. malloc의 간단한 래퍼함수라고 볼 수 있을 것이다.

```
139 /*
140  * mm_checkheap - There are no bugs in my code, so I don't need to check,
141  *      so nah!
142  */
143 void mm_checkheap(int verbose)
144 {
145 }
```

- mm_checkheap() : heap을 검사하는 함수같다. 구현되어있지 않다.

```
46 /*
47  * mem_sbrk - simple model of the sbrk function. Extends the heap
48  *    by incr bytes and returns the start address of the new area. In
49  *    this model, the heap cannot be shrunk.
50  */
51 void *mem_sbrk(int incr)
52 {
53     char *old_brk = mem_brk;
54
55     if ( (incr < 0) || ((mem_brk + incr) > mem_max_addr)) {
56     errno = ENOMEM;
57     fprintf(stderr, "ERROR: mem_sbrk failed. Ran out of memory...\n");
58     return (void *)-1;
59     }
60     mem_brk += incr;
61     return (void *)old_brk;
62 }
```

```
17 /* private variables */
18 static unsigned char heap[MAX_HEAP];
19 static char *mem_brk = heap; /* points to last byte of heap */
20 static char *mem_max_addr = heap + MAX_HEAP;  /* largest legal heap address */
```

- mem_sbrk() : 우선 mem_brk를 알아보자. mem_brk는 heap의 마지막 byte를 가리키는 static 포인터 변수이다. 이 mem_brk를 old_brk에게 준다. 그리고 mem_brk가 incr만큼 증가한다. 이러면 힙이 incr만큼 증가한다. 마지막으로, old_brk를 리턴한다.

구현 방법

naive는 구현되어있으므로, 구현 방법은 **생략**한다.

implicit

결과 화면 캡처(mdriver 실행 화면)

```
c2017020l2@2018-sp:~/malloclab-handout$ ./mdriver
Using default tracefiles in ./traces/
Measuring performance with a cycle counter.
Processor clock rate ~= 2097.6 MHz

Results for mm malloc:
   valid  util   ops    secs     Kops  trace
   yes    34%      10  0.000000 53784 ./traces/malloc.rep
   yes    28%      17  0.000000 74289 ./traces/malloc-free.rep
   yes    96%      15  0.000000 54910 ./traces/corners.rep
 * yes    81%    1494  0.000056 26905 ./traces/perl.rep
 * yes    75%     118  0.000002 77615 ./traces/hostname.rep
 * yes    91%   11913  0.000687 17332 ./traces/xterm.rep
 * yes    90%    5694  0.002008  2836 ./traces/amptjp-bal.rep
 * yes    93%    5848  0.001270  4604 ./traces/cccp-bal.rep
 * yes    94%    6648  0.007075   940 ./traces/cp-decl-bal.rep
 * yes    96%    5380  0.016302   330 ./traces/expr-bal.rep
 * yes    66%   14400  0.000128112536 ./traces/coalescing-bal.rep
 * yes    90%    4800  0.007063   680 ./traces/random-bal.rep
 * yes    55%    6000  0.012647   474 ./traces/binary-bal.rep
10         83%   62295  0.047237  1319

Perf index = 54 (util) + 40 (thru) = 94/100
```

소스 코드 / 함수 설명 / 동작 원리 / 구현 방법

1. 매크로 설명

```
39 /* MACROS */
40
41 /* single word (4) or double word (8) alignment */
42 #define ALIGNMENT 8
43
44 /* rounds up to the nearest multiple of ALIGNMENT */
45 #define ALIGN(p) (((size_t)(p) + (ALIGNMENT-1)) & ~0x7)
46
47 #define WSIZE 4
48 #define DSIZE 8
49 #define CHUNKSIZE (1 << 12)
50 #define OVERHEAD 8
51 #define MAX(x,y) ((x) > (y) ? (x) : (y))
52 #define PACK(size,alloc) ((size) | (alloc))
53 #define GET(p) (*(unsigned int*) (p))
54 #define PUT(p, val) (*(unsigned int*) (p) = (val))
55 #define GET_SIZE(p) (GET(p) & ~0x7)
56 #define GET_ALLOC(p) (GET(p) & 0x1)
57 #define HDRP(bp) ((char *)(bp) - WSIZE)
58 #define FTRP(bp) ((char *)(bp) + GET_SIZE(HDRP(bp)) - DSIZE)
59 #define NEXT_BLKP(bp) ((char *)(bp) + GET_SIZE((char *)(bp) - WSIZE))
60 #define PREV_BLKP(bp) ((char *)(bp) - GET_SIZE((char *)(bp) - DSIZE))
61
62 #define SIZE_T_SIZE (ALIGN(sizeof(size_t))) //  need for realloc
63 #define SIZE_PTR(p)  ((size_t*)(((char*)(p)) - SIZE_T_SIZE))    //  need for realloc
64
65 /* End of MACROS */
```

- ALIGNMENT, ALIGN(p), SITZE_T_SIZE, SIZE_PTR(p) 는 naive에서 설명했으므로 생략한다.

- #define WSIZE 4 : word 하나의 size를 4byte로 지정한다.
- #define DSIZE 8 : double word size를 8byte로 지정한다.
- #define CHUNKSIZE (1 << 12) : 초기 heap의 size를 설정하는데에 쓰인다.
- #define OVERHEAD 8 : header와 footer의 크기를 더한 8byte를 뜻한다.
- #define MAX(x, y) ((x) > (y) ? (x) : (y)) : x와 y 중 큰 값을 골라준다.
- #define PACK(size, alloc) ((size) | (alloc)) : size와 alloc을 입력하면 둘을 합쳐준다. 이를 활용하여 header에 size와 alloc정보를 입력한다.
- #define GET(p) (*(unsigned int*) (p)) : p포인터의 값을 받아온다.
- #define PUT(p, val) (*(unsigned int*) (p) = (val)) : p에 val값을 넣는다.
- #define GET_SIZE(p) (GET(p) & ~0x7) : 위의 GET매크로를 사용해 얻은 값을 masking하여 하위 3자리 bit를 제거한다. 즉, 상위 31bit만 얻어온다.
- #define GET_ALLOC(p) (GET(p) & 0x1) : GET_SIZE와 비슷한 방식으로 이번에는 LSB를 가져온다. 나머지는 0처리한다.
- #define HDRP(bp) ((char *)(bp) - WSIZE)) : bp(블럭 포인터변수)를 통해 그 block의 header의 처음을 가리키는 포인터를 얻는다.
- #define FTRP(bp) ((char *)(bp) + GET_SIZE(HDRP(bp)) - DSIZE) : HDRP와 비슷하게 이번에는 footer의 처음을 가리키는 포인터를 얻는다.
- #define NEXT_BLKP(bp) ((char *)(bp) + GET_SIZE((char *)(bp) - WSIZE)) : 다음 block의 payload를 가리키는 포인터를 얻는다.
- #define PREV_BLKP(bp) ((char *)(bp) - GET_SIZE((char *)(bp) - DSIZE)) : 이전 block의 payload를 가리키는 포인터를 얻는다.

2. 함수 설명

구현 중에 함수 원형을 선언하지 않아서 발생한 순서 오류를 함수 위치를 알맞게 재배열하여 해결하였습니다.

저는 next-fit을 이용하여 implicit을 구현하였습니다. best-fit을 시도하였지만, 비슷한 크기들을 합치는 과정에서 '비슷한' 이라는 것이 너무 모호하여 포기하였습니다.

```
68 static char *heap_listp = 0;    //  pointer of first block.
69 static char *next_listp = 0;
```

- 변수 선언 : next-fit 구현을 위해 필요한 static char * 변수입니다. 이 next_listp 포인터변수는 다음에 검색을 시작할 주소(혹은 위치)를 저장하는 역할을 합니다.
구별하기 쉽게 heap_listp의 앞부분만 next로 바꾸어 설정하였습니다.

```
72 /*
73  * coalesce
74  */
75 void *coalesce(void *bp){
76
77     size_t prev_alloc = GET_ALLOC(FTRP(PREV_BLKP(bp))); //  이전  블록의  할당  여부  0 = NO, 1 = YES
78     size_t next_alloc = GET_ALLOC(HDRP(NEXT_BLKP(bp))); //  다음  블록의  할당  여부  0 = NO, 1 = YES
79     size_t size = GET_SIZE(HDRP(bp));   //  현재  블록의  크기
80
81     //  case 1 : both side allocs == 1, no coalescing. return bp.
82     if(prev_alloc && next_alloc){
83         //      return bp;
84     }
85
86     //  case 2: prev_alloc == 1, next_alloc == 0. coalesce the next block and return bp.
87     else if(prev_alloc && !next_alloc){
88         size = size + GET_SIZE(HDRP(NEXT_BLKP(bp)));
89         PUT(HDRP(bp), PACK(size, 0));
90         PUT(FTRP(bp), PACK(size, 0));   //  why???????????????
91     }
92
93     //  case 3: prev_alloc == 0, next_alloc == 1. coalesce the prev block and return bp.
94     else if(!prev_alloc && next_alloc){
95         size = size + GET_SIZE(HDRP(PREV_BLKP(bp)));
96         PUT(HDRP(PREV_BLKP(bp)), PACK(size, 0));
97         PUT(FTRP(bp), PACK(size, 0));
98
99         bp = PREV_BLKP(bp);
100    }
101
102    //  case 4: prev_alloc == 0, next_alloc == 0. coalesce prev, present, next block. return bp.
103    else{
104        size = size + GET_SIZE(HDRP(PREV_BLKP(bp))) + GET_SIZE(FTRP(NEXT_BLKP(bp)));
105        PUT(HDRP(PREV_BLKP(bp)), PACK(size, 0));
106        PUT(FTRP(NEXT_BLKP(bp)), PACK(size, 0));
107        bp = PREV_BLKP(bp);
108    }
109
110    next_listp = bp;    //  give the pointer of current block payload to 'next_listp' for next-fit search!
111
112    return bp;
113 }
```

- coalesce() : free를 함에 있어서 이전block과 이후block을 검사하여 allocate되지 않은 block이라면 합쳐주는 함수이다.

case 1 : 양쪽 블록이 모두 할당되어있는 상태(즉, free block이 아님)

case 2 : 이전 block이 할당되어있고, 다음 block이 free인 경우

case 3 : 이전 block이 free이고, 다음 block이 할당되어있는 경우

case 4 : 양쪽 블록 모두가 free인 상태

먼저 size를 재계산한다. 그 후 각각의 header와 footer에 수정된 size정보와 alloc정보를 update해준다. case 2에서 많은 고민이 있었다. 위 함수 코드는 교과서의 코드를 참조한 것인데, 얼핏봐서는 현재 bp의 header와 footer의 정보를 update하는 잘못된 작업을 하는 것처럼 보인다. 하지만, 먼저 bp의 header size를 그것과 다음 block의 header size를 더한 값으로 update해준다. 그 후, FTRP 매크로를 사용하여 다음 block의 footer 정보를 바꿀수 있는데, 이는 FTRP 매크로가 header의 정보(size)를 기반으로 작성되었기 때문이다. 따라서, FTRP(bp)에는 bp의 update된 header의 size를 bp의 header에서부터 더한 값이 들어간다. 결론적으로 코드는 잘 수행됨을 알 수 있다.

```
115 /*
116  * place
117  */
118 static void place(void *bp, size_t asize){
119
120     size_t csize = GET_SIZE(HDRP(bp));
121     if((csize - asize) >= (2 * DSIZE)){ //  if size is more enough, place and make the rests a 'free block'!
122         PUT(HDRP(bp), PACK(asize, 1));
123         PUT(FTRP(bp), PACK(asize, 1));
124         bp = NEXT_BLKP(bp);
125         PUT(HDRP(bp), PACK(csize - asize, 0));
126         PUT(FTRP(bp), PACK(csize - asize, 0));
127     }
128     else{   //  if the difference between current_size and asize < 16bytes, just place!
129         PUT(HDRP(bp), PACK(csize, 1));
130         PUT(FTRP(bp), PACK(csize, 1));
131     }
132 }
```

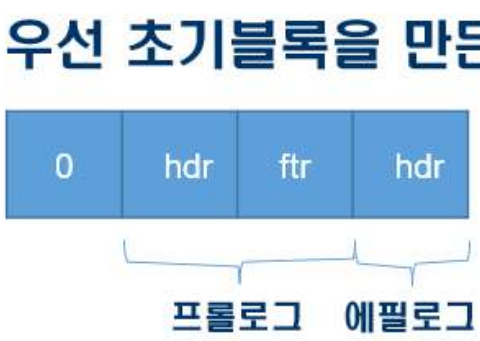
- place() : 할당하려는 크기를 asize라는 변수를 통해 받아온다. 현재 block의 header에 써있는 size를 currentBlockSize, 즉 csize라 선언해둔다. 만약 csize에서 asize를 뺀 값, 즉 현재 블록이 asize를 할당하고도 남는 크기가 16byte보다 크거나 같으면, 공간을 할당하고(alloc을 1로 설정), 다음 블록의 header와 footer를 free라는 것을 명시해준다(alloc을 0으로 설정). 만약 csize가 asize를 할당하고도 남는 크기가 16byte보다 작다면, 내부단편화를 감수하며 alignment를 위해 그 block의 header와 footer를 할당(alloc을 1로 설정)한다.
정리하자면, 16byte bp에서 asize만큼 쪼개어 할당을 해주어야 하는데, 가용공간과 할당하려는 공간의 차가 16byte보다 크냐 작냐를 기준으로 case를 구분한다.

```
134 /*
135  * extend_heap
136  */
137 static void *extend_heap(size_t words){
138
139     char *bp;
140     size_t size;
141
142     /* Allocate an even number of words to maintain alignment */    //  double word alignment!
143     size = (words % 2) ? (words + 1) * WSIZE : words * WSIZE;   //  size : byte단 위
144     if((long)(bp = mem_sbrk(size)) < 0)
145         return NULL;
146
147     /* Initialize free block header/footer and the epilogue header */
148     PUT(HDRP(bp), PACK(size, 0));   //  Free block header
149     PUT(FTRP(bp), PACK(size, 0));   //  Free block footer
150     PUT(HDRP(NEXT_BLKP(bp)), PACK(0, 1));   //  New epilogue header
151
152     /* Coalesce if the previous block was free */
153     return coalesce(bp);
154 }
```

- extend_heap() : heap이 부족할 경우 넓혀주는 함수이다. double word alignment를 사용하므로, 짝수개의 word * 4를 하여 size에 넣어준다. 그 후에 mem_sbrk(size)를 호출하는데, 이를 통해 힙을 넓힌 후에는 그 블록들의 header와 footer를 만들어줄 필요가 있다. 또한, init과정에서 보았던 에필로그 header도 만들어주어야 한다. 마지막으로, 이전 블록이 free상태였을 경우, 즉 이전블록의 크기가 할당하려는 사이즈보다 작아서 extend_heap을 수행한 경우 이 블록을 합쳐준다. 이때 coalesce() 함수를 사용한다.

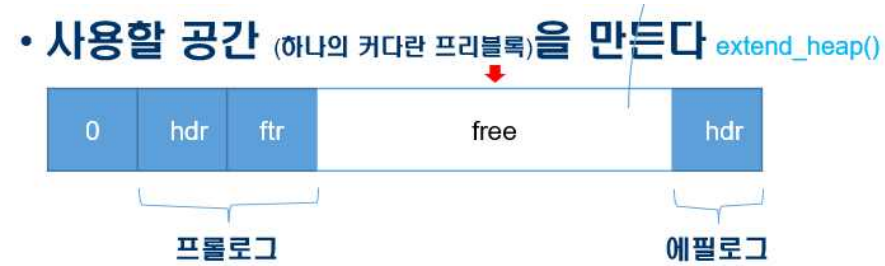
```
156 /*
157  * Initialize: return -1 on error, 0 on success.
158  */
159 int mm_init(void) {
160
161     if( (heap_listp = mem_sbrk(4 * WSIZE) ) == NULL)     //  초기  empty heap 생성
162         return -1;  //  heap_listp = 새로  생성되는  heap 영역의  시작  주소
163
164     PUT(heap_listp, 0); //  정렬을  위해서  의미없는  값 (4byte)을  삽입
165     PUT(heap_listp + WSIZE, PACK(OVERHEAD, 1)); //  prologue header
166     PUT(heap_listp + DSIZE, PACK(OVERHEAD, 1)); //  prologue footer
167     PUT(heap_listp + WSIZE + DSIZE, PACK(0, 1));    //  epilogue header
168     heap_listp += DSIZE;
169     next_listp = heap_listp;
170
171     if((extend_heap(CHUNKSIZE / WSIZE)) == NULL){   //  CHUNKSIZE 바이트의  free block 만큼  empty heap을  확장
172         //  생성된  empty heap을  free block으로  확장
173         return -1;
174     }   //  CHUNKSIZE는  1 << 12 일 .
175     return 0;
176 }
```

- mm_init() : 처음 heap을 만들어주고, 초기화해주는 함수이다. 우리가 필요한 것은 프롤로그 헤더, 프롤로그 푸터, 에필로그 헤더 뿐이다. 그러나, 이는 double word alignment에 좋지 않다. 따라서, 맨 앞에 의미없는 4byte(1word) 정수값 하나를 넣어 정렬을 맞춘다. 이후 포인터를 8byte만큼 옮겨주고, 그 값을 next_listp에 넘긴다. 초기에 총 16byte를 할당받고, 내용은 각각 위와 같다. 이를 그림으로 표현하면,



다음과 같다. (실습자료 참고)

이후에, 프롤로그와 에필로그 사이에 extend_heap()을 사용하여 영역을 넓혀야 한다. 이때 CHUNKSIZE / WSIZE만큼 확장을 하는데, CHUNKSIZE는 1 << 12 bytes이다. 최종 결과는 다음 그림과 같다. (실습자료 참고)



현재 heap_listp는 프롤로그의 footer를 가리키고 있을 것이다.

```
178 /*
179  * find_fit
180  */
181 static void *find_fit(size_t asize){
182
183     void *bp;
184     void *t = next_listp + WSIZE;   //  next_listp를  프 롤 로 그 에 서 free를 쪽 으 로   이 동 시 킨 다 .
185
186     for(bp = next_listp; bp != t; bp = NEXT_BLKP(bp)){  //  bp가  t까 지  갈  때  까 지  탐 색
187         if( !GET_SIZE(HDRP(bp)) && GET_ALLOC(HDRP(bp))){    //  에 필 로 그 를  찾 았 으 면
188             bp = PREV_BLKP(heap_listp); //  heap_listp는  프 롤 로 그 의  footer를  가 리 킴 .
189                                         //  bp는  그  전 블 럭 인  프 롤 로 그 의  header를  가 리 키 게 됨 .
190             t = next_listp;             //  t는  프 롤 로 그  footer, 즉  next_listp를  가 리 키 게  됨 .
191         }
192         else if( !GET_ALLOC(HDRP(bp)) && (asize <= GET_SIZE(HDRP(bp)))){
193             next_listp = bp;    //  alloc되 어 있 지  않 고 , size가  충 분 한  공 간 을  찾 으 면 ,
194             return bp;          //  next_listp에  해 당  포 인 터 를  저 장 해 주 고 , 리 턴 한 다 .
195         }
196     }
197     return NULL;
198 }
```

- find_fit() : 할당하고자 하는 size를 입력받아 적당한 공간이 있는지 탐색하는 함수이다. next-fit을 구현하였다.

먼저, 임시변수 *t에 next_listp + WSIZE를 한 값을 넣어준다. 그리고 탐색을 시작한다. next_listp에서부터 한블럭씩 bp가 t까지 이동할 때 까지 탐색을 진행한다.
만일 bp가 에필로그에 걸렸다면, bp를 프롤로그의 header를 가리키게 한다. 그리고 t를 프롤로그의 footer를 가리키게 한다. 이는 에필로그까지 갔음에도 못찾았다면, 처음부터 탐색을 진행하기 위함이다.
그럼에도 찾지 못하였다면, NULL을 리턴하게되고, malloc 함수는 밑에서 설명하겠지만, extend_heap()을 이용하여 힙을 늘리는 작업을 수행한다.
만일 탐색에 성공하였다면, 즉 alloc도 0이고, 블록 크기도 할당하고자 하는 size보다 크다면, 그 블록 포인터를 next_listp에게 전달하고, 리턴한다.

```
200 /*
201  * malloc
202  */
203 void *malloc (size_t size) {
204     size_t asize;    //  Adjusted block size
205     size_t extendsize;  //  Amount to extend heap if no fit
206     char *bp;
207
208     /* Ignore spurious requests */
209     if(size <= 0)
210         return NULL;
211
212     /* Adjust block size to include overhead and alignment reqs. */
213     if(size <= DSIZE){
214         asize = 2 * DSIZE;
215     }else{
216         asize = DSIZE * ((size + (DSIZE) + (DSIZE - 1)) / DSIZE);
217     }
218     /* Search the free list for a fit */
219     if((bp = find_fit(asize)) != NULL){
220         place(bp, asize);
221         return bp;
222     }
223
224     /* No fit found. Get more memory and place the block */
225     extendsize = MAX(asize, CHUNKSIZE);
226     if((bp = extend_heap(extendsize / WSIZE)) == NULL)
227         return NULL;
228
229     place(bp, asize);
230     return bp;
231 }
```

- malloc() : <line 209>의 조건을 교과서와 살짝 다르게 수정하였다. 교과서대로 == 으로 하니 89점이 나왔고, 수정하여 <= 으로 하니 94점이 나오는 것을 알 수 있다.

malloc 함수는 여태 설명한 모든 함수들을 종합한 함수라 보아도 된다. 동적메모리를 할당해주는 함수이다. 동작과정을 보겠다. 먼저, size를 입력받아 원하는 size(단위 : byte)만큼 할당을 하는 것을 목표로 한다.
size가 0보다 작거나 같을때에는 NULL을 리턴한다. size가 8보다 작거나 같을때에는 16으로 할당을 해준다. 또한, 8의 배수 단위로 할당을 해주도록 돕는다.

이제 find_fit을 통해 알맞은 공간을 찾는다. 만일 find_fit이 NULL이 아니라면, 즉 알맞은 공간을 찾았다면, 그 공간블럭의 포인터를 place()함수에 넣어 그 공간을 분할하고 할당표시를 해준다.

만약 find_fit이 실패하였다면, extend_heap을 호출하여 heap을 더 늘려준다. 그 후에, 늘린 공간을 분할(쪼개)하고 할당표시를 해준다.

```
/*
 * free
 */
void free (void *ptr) {
    if(!ptr) return;    // 잘못된 free 요청인 경우 함수를 종료한다. 이전 프로시져로 return
    size_t size = GET_SIZE(HDRP(ptr));  // bp의 헤더에서 block size를 읽어온다.

    // 실제로 데이터를 지우는 것이 아니라
    // header와 footer의 최하위 1 bit (1, 할당된 상태) 만을 수정

    PUT(HDRP(ptr), PACK(size, 0));
    PUT(FTRP(ptr), PACK(size, 0));  // header와 footer는 같은 것이기 때문에 size도 같음.

    coalesce(ptr);  // 주위에 빈 블록이 있을 시 병합한다.
}
```

- free() : 이 함수는 할당받은 공간을 다시 복구하는 함수이다. alloc을 0으로 바꾸고자 하는 포인터를 매개변수로 받아와 그 header와 footer의 alloc을 0으로 바꿔주고, coalesce를 호출하여 합칠 free block이 있으면 합쳐준다. 이후 과정은 coalesce에서 설명하였다.

```
/*
 * realloc - you may want to look at mm-naive.c
 */
void *realloc(void *oldptr, size_t size) {  // same as mm-naive.c


    size_t oldsize;
    void *newptr;

    /* If size == 0 then this is just free, and we return NULL. */
    if(size == 0) {
        free(oldptr);
        return 0;
    }

    /* If oldptr is NULL, then this is just malloc. */
    if(oldptr == NULL) {
        return malloc(size);
    }

    newptr = malloc(size);

    /* If realloc() fails the original block is left untouched  */
    if(!newptr) {
        return 0;
    }

    /* Copy the old data. */
    oldsize = *SIZE_PTR(oldptr);
    if(size < oldsize) oldsize = size;
    memcpy(newptr, oldptr, oldsize);

    /* Free the old block. */
    free(oldptr);

    return newptr;
}
```

- realloc() : naive와 동일한 함수다. 특이점은, 이 함수를 사용하기 위해 두 매크로를 추가하였다는 것이다. 이외의 설명은 naive에서 대신한다.

```
287 /*
288  * calloc - you may want to look at mm-naive.c
289  * This function is not tested by mdriver, but it is
290  * needed to run the traces.
291  */
292 void *calloc (size_t nmemb, size_t size) {
293     return NULL;
294 }
```

- calloc() : 구현하지 않았다. 설명은 naive에서 진행하였다.

이외의 함수들 :

```
297 /*
298  * Return whether the pointer is in the heap.
299  * May be useful for debugging.
300  */
301 static int in_heap(const void *p) {
302     return p < mem_heap_hi() && p >= mem_heap_lo();
303 }
304
305 /*
306  * Return whether the pointer is aligned.
307  * May be useful for debugging.
308  */
309 static int aligned(const void *p) {
310     return (size_t)ALIGN(p) == (size_t)p;
311 }
312
313 /*
314  * mm_checkheap
315  */
316 void mm_checkheap(int verbose) {
317 }
```

디버깅을 진행해주는 함수들로 판단된다.