2018 시스템 프로그래밍
- Lab 9 -

제출일자	2017.12.8
분 반	02
이 름	진승언
학 번	201404377

naive

```
c201404377@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 42634 ./traces/malloc.rep
   yes     77%      17  0.000000 64250 ./traces/malloc-free.rep
   yes    100%      15  0.000000 44253 ./traces/corners.rep
 * yes     71%    1494  0.000022 66619 ./traces/perl.rep
 * yes     68%     118  0.000002 63661 ./traces/hostname.rep
 * yes     65%   11913  0.000166 71689 ./traces/xterm.rep
 * yes     23%    5694  0.000090 62918 ./traces/amptjp-bal.rep
 * yes     19%    5848  0.000093 63014 ./traces/cccp-bal.rep
 * yes     30%    6648  0.000111 60132 ./traces/cp-decl-bal.rep
 * yes     40%    5380  0.000084 64368 ./traces/expr-bal.rep
 * yes      0%   14400  0.000231 62401 ./traces/coalescing-bal.rep
 * yes     38%    4800  0.000086 55771 ./traces/random-bal.rep
 * yes     55%    6000  0.000093 64846 ./traces/binary-bal.rep
10         41%   62295  0.000977 63745

Perf index = 26 (util) + 40 (thru) = 66/100
c201404377@2018-sp:~/malloclab-handout$
```

소스 코드

```
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(size) (((size) + (ALIGNMENT-1)) & ~0x7)
46
47
48 #define SIZE_T_SIZE (ALIGN(sizeof(size_t)))
49
50 #define SIZE_PTR(p)  ((size_t*)(((char*)(p)) - SIZE_T_SIZE))
```

naive에서 구현을 안해논 함수는 보고서 작성에 제외하였습니다.

구현 방법

naive에서 사용하는 매크로는 위와 같다
#define ALIGMENT 8은 말 그대로 ALIGMENT를 8의 값을 의미하게 한다.
#define ALIGN(size)는 size*8의 배수만큼 할당해준다. (옆에 붙어있는 매크로 정의는 생략하였습니다.)
#define SIZE_T_SIZE 매크로는 데이터타입의 크기를 할당하는 매크로이다. 우리 실습환경인 64bit환경에서는 8이다.
#define SIZE_PTR(p) 매크로는 해당 사이즈가 포함된 곳에 접근하는 매크로이다.

소스 코드

```
64 void *malloc(size_t size)
65 {
66   int newsize = ALIGN(size + SIZE_T_SIZE);
67   unsigned char *p = mem_sbrk(newsize);
68   //dbg_printf("malloc %u => %p\n", size, p);
69
70   if ((long)p < 0)
71     return NULL;
72   else {
73     p += SIZE_T_SIZE;
74     *SIZE_PTR(p) = size;
75     return p;
76   }
77 }
78
```

구현 방법

malloc 함수는 단순히 heap을 늘려가며 공간을 할당하는 방식을 사용한다.
mem_sbrk함수는 새로운 메모리를 heap 영역에 추가해주는 함수이다. 코드를 보면 newsize에 할당할 크기를 저장하고 mem_sbrk함수를 사용해서 해당 크기만큼의 새로운 메모리를 heap 영역추가하고 추가한 영역의 주소시작값을 p에 저장한다. 이 함수는 결국 계속 정해진 크기만큼의 공간을 연속적으로 할당하고 중간에 공간이 비게 된 경우가 있게 되도 해당공간을 사용하지 않고 연속해서 뒤에 블록을 할당해 사용하게 된다. 즉 현재 할당된 메모리 다음 주소로 연속적으로 메모리를 할당해준다.
size가 0이면 null을 반환하고 아니라면 최소 size 바이트의 메모리 블록의 포인터를 반환한다.

소스 코드

```
 92 void *realloc(void *oldptr, size_t size)
 93 {
 94   size_t oldsize;
 95   void *newptr;
 96
 97   /* If size == 0 then this is just free, and we return NULL. */
 98   if(size == 0) {
 99     free(oldptr);
100     return 0;
101   }
102
103   /* If oldptr is NULL, then this is just malloc. */
104   if(oldptr == NULL) {
105     return malloc(size);
106   }
107
108   newptr = malloc(size);
109
110   /* If realloc() fails the original block is left untouched  */
111   if(!newptr) {
112     return 0;
113   }
114
115   /* Copy the old data. */
116   oldsize = *SIZE_PTR(oldptr);
117   if(size < oldsize) oldsize = size;
118   memcpy(newptr, oldptr, oldsize);
119
120   /* Free the old block. */
121   free(oldptr);
122
123   return newptr;
124 }
```

구현 방법

realllloc()함수는 이전 블록의 크기를 바꾸고, 원본 값을 크기가 바뀐 블록에 다시 저장하고 이전 공간을 free()함수를 사용하여 해제후 memcpy로 데이터를 복사해와 재할당 해주는 것인데 naive에서는 free가 구현되지 않아 그냥

제자리걸음인 코드가 된다

소스 코드

```
129 void *calloc (size_t nmemb, size_t size)
130 {
131   size_t bytes = nmemb * size;
132   void *newptr;
133
134   newptr = malloc(bytes);
135   memset(newptr, 0, bytes);
136
137   return newptr;
138 }
```

구현 방법

nmemb * size 의 값을 bytes 변수에 저장한다, bytes변수를 매개변수로 malloc()호출해 메모리를 할당한다. 그 후, memset을 통해 0으로 초기화 시킨다. 즉 이 함수 calloc의 기능은 해당크기만큼의 메모리를 할당해주고 초기화해주어 해당 메모리를 가리키는 시작주소를 반환해주는 것이다.

implicit

```
c2014043777@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  0.000001 15469 ./traces/malloc.rep
   yes     28%      17  0.000001 27642 ./traces/malloc-free.rep
   yes     96%      15  0.000001 16861 ./traces/corners.rep
 * yes     86%    1494  0.003208   466 ./traces/perl.rep
 * yes     75%     118  0.000032  3647 ./traces/hostname.rep
 * yes     91%   11913  0.646187    18 ./traces/xterm.rep
 * yes     99%    5694  0.056549   101 ./traces/amptjp-bal.rep
 * yes     99%    5848  0.068244    86 ./traces/cccp-bal.rep
 * yes     99%    6648  0.093136    71 ./traces/cp-decl-bal.rep
 * yes    100%    5380  0.062210    86 ./traces/expr-bal.rep
 * yes     66%   14400  0.000489 29469 ./traces/coalescing-bal.rep
 * yes     93%    4800  0.051200    94 ./traces/random-bal.rep
 * yes     55%    6000  0.246941    24 ./traces/binary-bal.rep
10         86%   62295  1.228196    51

Perf index = 56 (util) + 2 (thru) = 58/100
c2014043777@2018-sp:~/malloclab-handout$
```

소스 코드

```
50 //ADD
51 #define WSIZE 4
52 #define DSIZE 8
53 #define CHUNKSIZE (1<<12)
54 #define OVERHEAD 8
55 #define MAX(x,y) ((x) >(y) ? (x) : (y) )
56 #define PACK(size ,alloc) ((size) | (alloc))
57 #define GET(p) (*(unsigned int*)(p))
58 #define PUT(p,val) (*(unsigned int*)(p) = (val))
59 #define GET_SIZE(p) (GET(p) & ~0x7)
60 #define GET_ALLOC(p) (GET(p) & 0x1)
61 #define HDRP(bp)     ((char *)(bp) - WSIZE)
62 #define FTRP(bp)     ((char*)(bp) + GET_SIZE(HDRP(bp)) - DSIZE)
63 #define NEXT_BLKP(bp) ((char*)(bp) + GET_SIZE((char * )(bp) -WSIZE))
64 #define PREV_BLKP(bp) ((char*)(bp) - GET_SIZE((char * )(bp) -DSIZE))
65 //static char *heap_listp = 0;
71 #define ALIGN(p) (((size_t)(p) + (ALIGNMENT-1)) & ~0x7)
72 #define SIZE_T_SIZE (ALIGN(sizeof(size_t)))
73 #define SIZE_PTR(p) ((size_t*)(((char*)(p)) - SIZE_T_SIZE))
74
75 static char *heap_listp = 0;
```

구현 방법

위에 코드가 있으므로 #define과 매크로내용은 생략하겠습니다.
WSIZE : word크기를 설정한다.
DSIZE : double word 크기를 설정한다.
CHUNKSIZE : 초기 heap 크기를 설정한다.
OVERHEAD : DSIZE와 마찬가지와 같은 크기를 설정한다. 이 매크로의 의미는 header + footer의 크기이고 실제 데이터가 저장되는 공간이 아니므로 overhead가 된다.
MAX(x,y) : x,y중 더 큰ㅍ값을 반환한다.
PACK(size,alloc) : PACK 매크로를 사용하여 size와 alloc와 값을 하나의 word로 묶는다. 쉽게 header와 footer에 저장할 수 있다.
GET(p) : 포인터 p가 가리키는 위치에서 word 크기의 값을 읽는다.
PUT(p, val) : 포인터 p가 가리키는 곳에 word 크기의 val 값을 쓴다.
GET_SIZE(p) : 포인터 p가 가리키는 곳에서 한 word를 읽은 다음 하위 3bit를 버린다. 즉, Header에서 block size를 읽는 것과 같다.
GET_ALLOC(p) : 포인터 p가 가리키는 곳에서 한 word를 읽은 다음 하위 1bit를 읽는다. block의 할당 여부를 0(NO), 1(YES)로 구분한다.
HDRP(bp) : 주어진 포인터 bp의 header의 주소를 계산한다.
FTRP(bp) : 주어진 포인터 bp의 footer의 주소를 계산한다.
NEXT_BLKP(bp) : 주어진 포인터 bp를 이용하여 다음 block의 주소를 계산한다.
PREV_BLKP(bp) : 주어진 포인터 bp를 이용하여 다음 block의 주소를 계산한다.

그 밑의 3개의 매크로는 naive에도 있던 매크로이다.
*heap_listp = 0 은 처음 블록포인터를 선언한거다.

소스 코드

```
80 int mm_init(void) {
81     if((heap_listp = mem_sbrk(4 * WSIZE)) == NULL){
82         return -1;
83     }
84 
85     PUT(heap_listp, 0);
86     PUT(heap_listp +(1*WSIZE), PACK(DSIZE, 1));
87     PUT(heap_listp + (2*WSIZE), PACK(DSIZE, 1));
88     PUT(heap_listp + (WSIZE*3), PACK(0,1));
89     heap_listp += DSIZE;
90     if((extend_heap(CHUNKSIZE/ WSIZE)) == NULL){
91         return -1;
92     }
93 
94     return 0;
95 
96 }
```

구현 방법

맨 처음 heap을 생성하는 함수로, 메모리 공간(CHUNKSIZE)을 생성한다. 메 heap_listp에 새로 생성되는 heap 영역의 시작주소를 담는다. 그리고 정렬을 위해 의미없는 값을 삽입하고 heap_listp의 위치를 header와 fotter의 사이로 이동시킨다. 그 후 , CHUNKSIZE 바이트의 free block 만큼 빈 힙을 확장한다. 생성된 빈 힙을 free block으로 확장한다. 마지막으로 WSIZe로 align 되어있지 않으면 에러를 나타내는 -1을 반환한다.

소스 코드

```
101 void *malloc (size_t size) {
102     size_t asize;
103     size_t extendsize;
104     char *bp;
105 
106 
107     if(size <= 0){
108         return NULL;
109     }
110 
111     if (size <= DSIZE) {
112         asize = DSIZE*2;
113     }
114     else {
115         asize = DSIZE * ((size + (DSIZE) + (DSIZE-1)) / DSIZE);
116     }
117 
118     /* Search the free list for a fit */
119     if ((bp = find_fit(asize)) != NULL) {
120         place(bp, asize);
121         return bp;
122     }
123     /* No fit found. Get more memory and place the block */
124     extendsize = MAX(asize,CHUNKSIZE);
125     if ((bp = extend_heap(extendsize/WSIZE)) == NULL) {
126         return NULL;
127     }
128 
129     place(bp, asize);
130     return bp;
131 }
132 
```

구현 방법

malloc함수는 size 바이트 메모리 블록을 할당해주는 함수이다. 먼저 size가 0보다 작으면 NULL을 리턴해준다. 만약 size가 DSIZE보다 크면 asize에 DISZE*8을 저장해주는데 이는 최소 16바이트의 크기의 블록을 구성하는거다. 8바잍는 정렬 요건을 만족시키기 위해, 추가적인 8바이트는 헤더와 풋터 오버헤드를 위해, 그 이외의 경우에는 8바이트를 넘는 요청에 대해서 일반적인 규칙은 오버헤드를 추가하고, 인접 8의 배수를 반올림한다.(115라인) 일단 할당기가 요청한 크기를 조정한 후에 적절한 가용 블록을 가용 리스트에서 검색하고 만일 맞는 블록을 찾으면 할당기는 요청한 블록을 배제하고, 옵션으

로 초과부분을 분할하고, 새롭게 할당한 블록을 리턴한다. 맞는 블록을 찾지 못한 경우에는 힙을 새로운 가용 블록으로 확장하고 요청한 블록을 이 새 가용 블록에 배치하고, 필요한 경우에 블록을 분하하며, 이후에 새롭게 할당한 블록의 포인터를 반환해준다.

소스 코드

```
136 void free (void *bp) {
137     if(!bp) return;
138
139     size_t size = GET_SIZE(HDRP(bp));
140
141     PUT(HDRP(bp), PACK(size, 0 ));
142     PUT(FTRP(bp), PACK(size, 0 ));
143     coalesce(bp);
144 }
145
```

구현 방법

free 함수는 입력 받은 블록을 가용 메모리로 바꾸어 주는 함수이다. 먼저 잘못된 free 요청인 경우는 함수를 종료한다. 그 후 GET_SIZE 매크로를 통해 bp의 헤더에서 블록사이즈를 읽어온다. 그 후 (141라인) 매크로를 통해 bp의 헤더에 블록 사이즈와 alloc =0을 저장하고 그 밑도 똑같이 footer에 저장해준다. 마지막으로 coalesce함수를 통해 주위에 빈 블록이 있을 시 병합해준다.

소스 코드

```
149 void *realloc(void *oldptr, size_t size) {
150     size_t oldsize;
151     void *newptr;
152 
153     /* If size == 0 then this is just free, and we return NULL. */
154     if(size == 0) {
155         free(oldptr);
156         return 0;
157     }
158 
159     /* If oldptr is NULL, then this is just malloc. */
160     if(oldptr == NULL) {
161         return malloc(size);
162     }
163 
164     newptr = malloc(size);
165 
166     /* If realloc() fails the original block is left untouched  */
167     if(!newptr) {
168         return 0;
169     }
170 
171     /* Copy the old data. */
172     oldsize = *SIZE_PTR(oldptr);
173     if(size < oldsize) {
174         oldsize = size;
175     }
176     memcpy(newptr, oldptr, oldsize);
177 
178     /* Free the old block. */
179     free(oldptr);
180 
181     return newptr;
182 }
```

구현 방법

realloc은 할당됐었던 블록을 새로운 사이즈로 변경하여 할당해주는 함수이다. naive와 동일하며 free함수가 implicit에서 구현되어있음이 다르다. 할당받을 사이즈가 0인 경우 free 시켜주고 0을 리턴한다. 기존 블록이 NULL이라면 malloc으로 할당 후 리턴 합니다. 알맞은 크기로 블록을 새롭게 생성한 후, 새로운 사이즈가 기존블록의 사이즈 보다 작으면 기존블록의 사이즈보다 큰 부분에 있는 데이터는 복사가 되어도 쓸모가 없으므로 새로운 사이즈의 크기만큼만 복사해 준다. 반대 경우는 원래 블록에 있는 데이터를 모두 복사해준다.

소스 코드

```
218 //extend heap
219 static void *extend_heap(size_t words)
220 {
221     char *bp;
222     size_t size;
223
224     size = (words % 2)? (words + 1) * WSIZE : words * WSIZE;
225     if((long)(bp = mem_sbrk(size)) == -1 ){ //
226         return NULL;
227     }
228
229     PUT(HDRP(bp), PACK(size, 0));
230     PUT(FTRP(bp), PACK(size, 0 ));
231     PUT(HDRP(NEXT_BLKP(bp)), PACK(0,1));
232
233     return coalesce(bp);
234 }
```

구현 방법

extend heap함수는 요청 받은 사이즈의 블럭을 할당 받아 heap에 추가해주는 함수 이다. 이 함수는 힙이 초기화 될때와 mm_malloc이 적당한 맞춤 fit을 찾지 못했을 때 호출한다. 정렬을 유지하기 위해서 이 함수는 요청한 크기를 인접 2워드의 배수 즉 8 바이트로 반올림하며, 그 후에 메모리 시스템으로부터 추가적인 힙 공간을 요청한다.

힙은 더블워드 경계에서 시작하고 이 함수로 가는 모든 호출은 그 크기가 더블워드의 배수인 블록을 리턴한다. 뚜라서 mem_sbrk로의 모든 호출은 에필로그 블록의 헤더에 곧이어서 더블 워드 정렬된 메모리 블록을 리턴한다. 229~3331라인의 메크로를 이용해서 이 헤더는 새 가용 블록의 헤더가 되고 블록의 마지막 dnjesmsm to 에필로그 블록의 헤더가 되고 이전 힙이 가용 블록으로 끝났다면, 두 개의 가용 블록을 통합하기 위해 coalesce함수를 호출하고 통합된 블록의 블록 포인터를 리턴해준다.

소스 코드

```
236 static void *coalesce(void *bp){
237     size_t prev_alloc = GET_ALLOC(FTRP(PREV_BLKP(bp)));
238     size_t next_alloc = GET_ALLOC(HDRP(NEXT_BLKP(bp)));
239     size_t size = GET_SIZE(HDRP(bp));
240 
241     if(prev_alloc && next_alloc){
242         return bp;
243     }
244 
245     else if (prev_alloc && !next_alloc)
246     { /* Case 2 */
247         size += GET_SIZE(HDRP(NEXT_BLKP(bp)));
248         PUT(HDRP(bp), PACK(size, 0));
249         PUT(FTRP(bp), PACK(size,0));
250     }
251     else if (!prev_alloc && next_alloc) { /* Case 3 */
252         size += GET_SIZE(HDRP(PREV_BLKP(bp)));
253         PUT(FTRP(bp), PACK(size, 0));
254         PUT(HDRP(PREV_BLKP(bp)), PACK(size, 0));
255         bp = PREV_BLKP(bp);
256     }
257     else { /* Case 4 */
258         size += GET_SIZE(HDRP(PREV_BLKP(bp))) +  GET_SIZE(FTRP(NEXT_BLKP(bp)));
259         PUT(HDRP(PREV_BLKP(bp)), PACK(size, 0));
260         PUT(FTRP(NEXT_BLKP(bp)), PACK(size, 0));
261         bp = PREV_BLKP(bp);
262     }
263     return bp;
264 }
```

구현 방법

coalesce 함수는 빈 공간을 합쳐주는 함수이다. 즉 free 된 블록을 해당 블록의 앞과 뒤에 있는 블록이 free 인지에 따라 free인 블록이 있으면 해당 블록과 합쳐준다. 4가지 경우의 수로 나눌 수 있는데 free된 블록의 이전블록과 다음블록의 할당여부에 따라 알맞게 병합한 뒤 그 포인터를 반환해준다. 첫 if 문은 앞뒤 블록이 모두 사용중이므로 병합이 불가한 상황이고 그다음 else if 문은 뒤 블록이 비므로 뒤 블록과 병합한다. 그 뒤 else if문은 앞블록과 병합한다. 마지막 case 4인 else문은 앞뒤가 다 비므로 앞뒤 free블록들과 병합해준다.

소스 코드

```
266 static void *find_fit(size_t asize){
267     void *bp;
268 
269     for(bp = heap_listp; GET_SIZE(HDRP(bp))>0; bp = NEXT_BLKP(bp))
270     {
271         if(!GET_ALLOC(HDRP(bp)) && (asize <= GET_SIZE(HDRP(bp)))){
272             return bp;
273         }
274     }
275     return NULL;
276 }
```

구현 방법

find_fit 함수는 입력받은 사이즈에 맞는 할당받을 공간을 찾아주는 함수이다. 마지막으로 사용했던 블록부터 블록 사이즈가 0이 될 때까지 블록을 검색한다. 그리고 해당 블록이 free블록이면서 요청 사이즈에 맞거나 크면 해당 블록을 리턴해주면 된다. 만약 끝까지 맞는 블록을 못찾으면 NULL을 리턴해준다.

소스 코드

```
278 static void place(void *bp, size_t asize){
279     size_t csize = GET_SIZE(HDRP(bp));
280     if((csize - asize) >= (DSIZE + OVERHEAD)){
281         PUT(HDRP(bp), PACK(asize, 1));
282         PUT(FTRP(bp), PACK(asize, 1));
283         bp = NEXT_BLKP(bp);
284         PUT(HDRP(bp), PACK(csize - asize, 0 ));
285         PUT(FTRP(bp), PACK(csize- asize, 0));
286     }
287     else{
288         PUT(HDRP(bp), PACK(csize , 1));
289         PUT(FTRP(bp), PACK(csize , 1));
290     }
291 }
```

구현 방법

place함수는 입력된 블록과 사이즈에 따라 블록을 그대로 할당할지 나누어 할당할지 정해서 알맞게 블록을 할당해주는 함수이다. 즉 찾은 자리에 값을 저장해준다. 먼저 csize에 입력 받은 블록의 사이즈를 받아온다. 그리고 첫 if 문은 블록을 나누는 경우이고 else문은 블록을 나누지 않는 경우이다.