2018 시스템 프로그래밍
- Lab 9 -

제출일자	2017.12.16
분 반	00
이 름	장수훈
학 번	201402414

naive

```
c201402414@2018-sp:~/malloc/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  0.000009158776 ./traces/perl.rep
 * yes    68%     118  0.000001183072 ./traces/hostname.rep
 * yes    65%   11913  0.000072165119 ./traces/xterm.rep
 * yes    23%    5694  0.000071 80285 ./traces/amptjp-bal.rep
 * yes    19%    5848  0.000074 78603 ./traces/cccp-bal.rep
 * yes    30%    6648  0.000092 72645 ./traces/cp-decl-bal.rep
 * yes    40%    5380  0.000066 81484 ./traces/expr-bal.rep
 * yes     0%   14400  0.000200 72155 ./traces/coalescing-bal.rep
 * yes    38%    4800  0.000056 85098 ./traces/random-bal.rep
 * yes    55%    6000  0.000076 78763 ./traces/binary-bal.rep
10        41%   62295  0.000717 86857

Perf index = 26 (util) + 40 (thru) = 66/100
c201402414@2018-sp:~/malloc/malloclab-handout$
```

소스 코드

```
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))
```

구현 방법

- ALIGNMENT 8
ALIGNMENT를 8로 정의한다. 메모리를 8n 의 크기로 쓰기위하여 만든 것
- ALIGN(size) (((size) + (ALIGNMENT-1)) & ~0x7)
입력받은 size를 size보다 큰 8n 중 가장 작은수자로 바꿔준다.
- SIZE_T_SIZE (ALIGN(sizeof(size_t)))
size_t 의 크기를 ALIGN 해준 값을 정의한다. size_t는 unsigned int 이기 때문에 4, ALIIGN을 하면 4보다큰 8n은 8이다.
- SIZE_PTR(p) ((size_t*)(((char*)(p)) - SIZE_T_SIZE))
p에 들어갈 포인터에 SIZE_T_SIZE의 값을 앞으로 이동한 주소의 값을 반환한다.

- void *malloc(size_t size)
동적 메모리를 할당해주는 함수이다. size를 받은뒤 위에 메크로들을 통해 8n의 값을 (가장 작은) 저장하고, 포인터 변수에 newsize만큼 mem_sbrk를 통해 할당 한 후 반환한다.
- void *realloc(void *oldptr, size_t size)
호출되어 새로운 공간을 할당하고, 그 전의 값을 새로운 공간에 복사를 한다. 이전 공간은 free함수를 통해 해제한다.
- void *calloc (size_t nmemb, size_t size)
블록을 할당하고 0으로 블록을 세팅하는 함수

implicit

```
c201402414@2018-sp:~/malloc/malloclab-handout$ ./mdriver
Using default tracefiles in ./traces/
Measuring performance with a cycle counter.
Processor clock rate ~= 2097.6 MHz
Segmentation fault
```

안타깝게도 실패했다.

소스 코드 / 구현 방법

- WSIZE
word 크기 결정

- DSIZE
double word 크기 결정

- CHNKSIZE
초기에 heap의 크기를 설정

- OVERHEAD
header + footer의 크기, 실제 데이터가 아니다.

- MAX(x,y)
x,y,중 max값

- PACK(size, alloc)
size와 alloc의 값을 하나의 word로 합쳐서 사용하기 편리하게 해준다.

- GET(p)
포인터 p가 가리키는 word 값을 읽는다.

- PUT(p, val)
포인터 p가 가리키는 word에 val 값을 쓴다.

- GET_SIZE(p)
포인터 p가 가리키는 word를 읽은 다음 하위 3bit 버린다.

- GET_ALLOC(p)
포인터 p가 가리키는 word를 읽은 후 하위 1bit 읽는다.

- HDRP(bp)
주어진 포인터 bp의 header 주소를 계산한다.

- FTRP(bp)
주어진 포인터 bp의 footer 주소를 계산한다.

- NEXT_BLKP(bp)
주어진 포인터 bp를 이용하여 다음 block의 주소계산

- PREV_BLKP(bp)
주어진 포인터 bp를 이용하여 다음 block의 주소를 계산한다.

```
74 int mm_init(void) {
75     if ((heap_listp = mem_sbrk(4 * WSIZE)) == NULL)
76         return -1;
77     PUT(heap_listp, 0);
78     PUT(heap_listp + WSIZE, PACK(OVERHEAD, 1));
79     PUT(heap_listp + DSIZE, PACK(OVERHEAD, 1));
80     PUT(heap_listp + (WSIZE+ DSIZE), PACK(0, 1));
81     heap_listp += DSIZE;//
82
83     //next_bp = heap_listp;
84     //curheap = heap_listp;
85     if (extend_heap(CHUNKSIZE / WSIZE) == NULL)
86         return -1;
87     return 0;
88 }
```

처음에 heap을 생성해주는 함수. 메모리를 할당해줄 heap을 초기화 해준다.

```
 93 void *malloc (size_t size) {
 94
 95     size_t asize;
 96     size_t extendsize;
 97     char *bp;
 98
 99     if (size == 0) {
100         return NULL;
101     }
102     if (size <= DSIZE)//footer(4)  header(4) and pay,padding (8)
103         asize = DSIZE + OVERHEAD;
104     else {
105         asize = DSIZE * ((size + (DSIZE)+(DSIZE - 1)) / DSIZE);
106     }
107     if ((bp = find_fit(asize)) != NULL) {
108         place(bp, asize);//divide place
109         return bp;
110     }
111
112     extendsize = MAX(asize, CHUNKSIZE);
113     if ((bp = extend_heap(extendsize / WSIZE)) == NULL)
114         return NULL;
115     place(bp, asize);
116     //  curheap=bp;
117     //  curheap = NEXT_BLKP(bp);
118     //  curheap=heap_listp;
119     return bp;
120 }
```

size를 받은만큼 메모리를 할당한다. 사이즈가 0이면 null을 리턴 size가 dsize보다 작거나 같으면 asize에 위와같이 저장, 작은 경우 필요한만큼 저장한다.

if else문을 통해 결과가 place 함수를 통해 메모리를 할당하고, 포인터를 반환한다.

```
121 static void *coalesce(void *bp)//merge free block
122 {
123     size_t prev_alloc = GET_ALLOC(FTRP(PREV_BLKP(bp)));
124     size_t next_alloc = GET_ALLOC(HDRP(NEXT_BLKP(bp)));
125     size_t size = GET_SIZE(HDRP(bp));
126 
127     if (prev_alloc && next_alloc) {
128         return bp;
129     }
130 
131     else if (prev_alloc && !next_alloc) {
132         size += GET_SIZE(HDRP(NEXT_BLKP(bp)));
133         PUT(HDRP(bp), PACK(size, 0));
134         PUT(FTRP(bp), PACK(size, 0));
135     }
136 
137     else if (!prev_alloc && next_alloc) {
138         size += GET_SIZE(HDRP(PREV_BLKP(bp)));
139         PUT(FTRP(bp), PACK(size, 0));
140         PUT(HDRP(PREV_BLKP(bp)), PACK(size, 0));
141         bp = PREV_BLKP(bp);
142     }
143     else {
144         size += GET_SIZE(FTRP(PREV_BLKP(bp))) + GET_SIZE(HDRP(NEXT_BLKP(bp)));
145         PUT(HDRP(PREV_BLKP(bp)), PACK(size, 0));
146         PUT(FTRP(NEXT_BLKP(bp)), PACK(size, 0));
147         bp = PREV_BLKP(bp);
148     }
149     return bp;
```

빈 공간을 합쳐주는 함수다 이전블럭, 다음 블록의 최하위 bit이 둘다 1인 경우 bp를 리턴
이전 블록 최하위 biit가 1이고 다음블럭은 0인경우는 다음 블록과 합친 뒤 리턴
0,1 인 경우 이전블럭과 합친뒤 리턴 , 0,0인 경우 이전블럭 다음블럭을 전부 합친뒤에 리턴한다.

```
151 /*
152  * free
153  */
154 /*void free (void *ptr) {
155     if(!ptr) return;
156     size_t size = GET_SIZE(HDRP(ptr));
157     PUT(HDRP(ptr), PACK(size, 0));
158     PUT(FTRP(ptr), PACK(size, 0));
159
160     next_bp = coalesce(ptr);
161 }*/
162 void mm_free(void *bp) {
163     if (bp == 0) return;
164     size_t size = GET_SIZE(HDRP(bp));
165     PUT(HDRP(bp), PACK(size, 0));
166     PUT(FTRP(bp), PACK(size, 0));
167
168     coalesce(bp);
169 }
```

기존에 할당된 데잍터를 해제하는 함수.

```
174 void *realloc(void *oldptr, size_t size) {
175     size_t oldsize;
176     void *newptr;
177 
178     /* If size == 0 then this is just free, and we return NULL. */
179     if (size == 0) {
180         free(oldptr);
181         return 0;
182     }
183 
184     /* If oldptr is NULL, then this is just malloc. */
185     if (oldptr == NULL) {
186         return malloc(size);
187     }
188 
189     newptr = malloc(size);
190 
191     /* If realloc() fails the original block is left untouched  */
192     if (!newptr) {
193         return 0;
194     }
195 
196     /* Copy the old data. */
197     oldsize = *SIZE_PTR(oldptr);
198     if (size < oldsize) oldsize = size;
199     memcpy(newptr, oldptr, oldsize);
200 
201     /* Free the old block. */
202     free(oldptr);
203 
204     return newptr;
205 }
```

이미 할당되어 있는 메모리의 크기를 다시 할당한다. 새로운 블록을 malloc함수를 통해 동적 메모리 할당을 해준 뒤, 원래 있던 데이터를 복사하고 기존의 블록을 free 함수를 통해 해제 한 뒤에 새로 할당된 메모리에 기존의 데이터를 복사하는 형식이다.

```
244 static void place(void *bp, size_t asize)
245 {size_t rsize = GET_SIZE(HDRP(bp));
246
247     if ((rsize - asize) >= (OVERHEAD + WSIZE)) {
248         PUT(HDRP(bp), PACK(asize, 1));
249         PUT(FTRP(bp), PACK(asize, 1));
250         bp = NEXT_BLKP(bp);
251         PUT(HDRP(bp), PACK(rsize - asize, 0));
252         PUT(FTRP(bp), PACK(rsize - asize, 0));
253     }
254     else {
255         PUT(HDRP(bp), PACK(rsize, 1));
256         PUT(FTRP(bp), PACK(rsize, 1));
257     }
258
259 }
260 static void *extend_heap(size_t words)
261 {
262     char *bp;
263     size_t size;
264
265     size = (words % 2) ? ((words + 1)*WSIZE) : (words * WSIZE);
266     if ((long)(bp = mem_sbrk(size)) == -1)
267     {
268
269         return NULL;
270     }
271
272     PUT(HDRP(bp), PACK(size, 0));
273     PUT(FTRP(bp), PACK(size, 0));
274     PUT(HDRP(NEXT_BLKP(bp)), PACK(0, 1));
275
276     return coalesce(bp);
277 }
```

place는 해당 위치에 사이즈만큼 메모리를 위치시켜주는 함수고
extend heap은 요청 받은 크기의 빈 블록을 만들고, 이전 블록을 검사하여 상황별로 free 시켜주는 함수이다.

```
278 static void *find_fit(size_t asize)
279 {
280
281     char *bp;
282
283     for (bp = next_bp; GET_SIZE(HDRP(bp))>0; bp = NEXT_BLKP(bp))
284     {
285
286         if (!(GET_ALLOC(HDRP(bp))) && (asize <= GET_SIZE(HDRP(bp))))
287             next_bp = bp;
288         return bp;
289     }
290     return NULL;
291
292 }
```

free block을 검색하고 first, next, best fit 중 선택해서 구현하는 함수이다.