2018 시스템 프로그래밍

- Lab 09 -

제출일자	2018.12.14
분 반	01
이 름	함지희
학 번	201702087

1. Naïve

```
b201702087@2018-sp: ~/malloclab-handout
b201702087@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 82258 ./traces/malloc.rep
   yes    77%      17  0.000000106127 ./traces/malloc-free.rep
   yes   100%      15  0.000000 91199 ./traces/corners.rep
 * yes    71%    1494  0.000009157413 ./traces/perl.rep
 * yes    68%     118  0.000001158054 ./traces/hostname.rep
 * yes    65%   11913  0.000079151569 ./traces/xterm.rep
 * yes    23%    5694  0.000066 85969 ./traces/amptjp-bal.rep
 * yes    19%    5848  0.000071 82655 ./traces/cccp-bal.rep
 * yes    30%    6648  0.000092 72015 ./traces/cp-decl-bal.rep
 * yes    40%    5380  0.000067 80687 ./traces/expr-bal.rep
 * yes     0%   14400  0.000209 68874 ./traces/coalescing-bal.rep
 * yes    38%    4800  0.000053 90733 ./traces/random-bal.rep
 * yes    55%    6000  0.000069 86532 ./traces/binary-bal.rep
10         41%   62295  0.000716 86988

Perf index = 26 (util) + 40 (thru) = 66/100
b201702087@2018-sp:~/malloclab-handout$
```

소스 코드

```
mm-naive.c (~/malloclab-handout) - VIM
 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;
~/malloclab-handout/mm-naive.c [utf-8,unix][c]          0,121/145 79%
```

```
mm-naive.c (~/malloclab-handout) - VIM
 60  * malloc - Allocate a block by incrementing the brk pointer.
 61  *      Always allocate a block whose size is a multiple of the a
    lignment.
 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 }
~/malloclab-handout/mm-naive.c [utf-8,unix][c]          1,76/145 46%
```

구현 방법

<매크로>

#define ALIGNMENT 8 - double word로 할당하기 위해서 8로 설정

#define ALIGN(size) (((size) + (ALIGNMENT-1)) & ~ 0x7) - 받은 size를 8의 배수로 만들어주는 매크로

#define SIZE_T_SIZE (ALIGN(sizeof(size_t))) - 본 실습에선 64bit이므로 8이며, size 정보가 들어있는 공간의 크기이다.

#define SIZE_PTR(p) ((size_t*)(((char*)(p)) – SIZE_T_SIZE)) - 포인터 p가 가리키는 블록의 사이즈를 알려주는 매크로

<malloc>

size를 받아 ALIGN으로 8의 배수로 만들어 newsize를 만든 뒤, mem_sbrk함수로 힙에 공간을 할당해준다. 포인터p는 할당된 힙 영역의 첫번째 바이트 주소이다. p에 오류가 없다면, SIZE_T_SIZE만큼 p를 이동시키고, SIZE_PTR로 p가 가리키는 블록의 사이즈에 size를 넣는다. p를 반환한다.

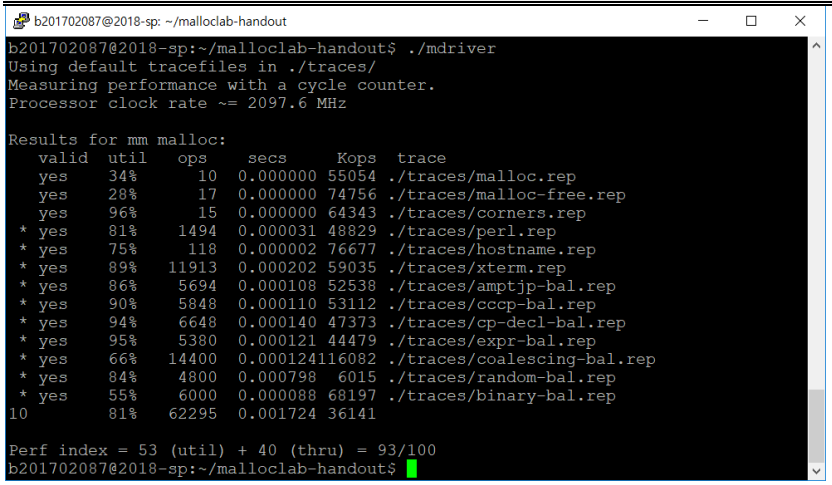
<realloc>

사이즈가 0이면 free한다는 뜻이고, 포인터가 NULL이면 malloc한다는 뜻이다. 이 두 경우에 속하지 않으면 malloc으로 새로 사이즈를 할당한 뒤 반환받은 포인터를 저장하고, 새 블록에 기존 블록를 복사한 뒤, 기존 블록을 free해준다. 새로운 포인터를 반환한다.

<calloc>

블록을 할당한 뒤, 0으로 채운다.

2. implicit



소스 코드

mm-implicit.c (~/malloclab-handout) - VIM

```
 44 /*매 크 로 */
 45 #define WSIZE 4
 46 #define DSIZE 8
 47 #define CHUNKSIZE (1 << 12)
 48 #define OVERHEAD 8
 49 #define MAX(x, y) ((x)>(y)?(x):(y))
 50 #define PACK(size, alloc) ((size)|(alloc))
 51 #define GET(p) (*(unsigned int*)(p))
 52 #define PUT(p, val) (*(unsigned int*)(p)=(val))
 53 #define GET_SIZE(p) (GET(p)&~0x7)
 54 #define GET_ALLOC(p) (GET(p)&0x1)
 55 #define HDRP(bp) ((char*)(bp)-WSIZE)
 56 #define FTRP(bp) ((char*)(bp)+GET_SIZE(HDRP(bp))-DSIZE)
 57 #define NEXT_BLKP(bp) ((char*)(bp)+GET_SIZE((char*)(bp)-WSIZE))
 58 #define PREV_BLKP(bp) ((char*)(bp)-GET_SIZE((char*)(bp)-DSIZE))
 59
 60 /*전 역 변 수 */
 61 static char *heap_listp = 0;
 62 static char *next_fit = 0;
~/malloclab-handout/mm-implicit.c [utf-8,unix][c]                13,44/260 17%
```

mm-implicit.c (~/malloclab-handout) - VIM

```
 64 void *coalesce (void *bp){
 65     size_t prev_alloc = GET_ALLOC(FTRP(PREV_BLKP(bp)));
 66     size_t next_alloc = GET_ALLOC(HDRP(NEXT_BLKP(bp)));
 67     size_t size = GET_SIZE(HDRP(bp));
 68
 69     if(prev_alloc && next_alloc){
 70         return next_fit = bp;
 71     }
 72     else if(prev_alloc && !next_alloc){
 73         size += GET_SIZE(HDRP(NEXT_BLKP(bp)));
 74         PUT(HDRP(bp), PACK(size, 0));
 75         PUT(FTRP(bp), PACK(size, 0));
 76     }
 77     else if(!prev_alloc && next_alloc){
 78         size += GET_SIZE(FTRP(PREV_BLKP(bp)));
 79         PUT(FTRP(bp), PACK(size, 0));
 80         PUT(HDRP(PREV_BLKP(bp)), PACK(size, 0));
 81         bp = PREV_BLKP(bp);
 82     }
 83     else {
 84         size += GET_SIZE(HDRP(PREV_BLKP(bp)));
 85         size += GET_SIZE(FTRP(NEXT_BLKP(bp)));
 86         PUT(HDRP(PREV_BLKP(bp)), PACK(size, 0));
 87         PUT(FTRP(NEXT_BLKP(bp)), PACK(size, 0));
 88         bp = PREV_BLKP(bp);
 89     }
 90     return next_fit = bp;
 91 }
~/malloclab-handout/mm-implicit.c [utf-8,unix][c]                 1,91/260 27%
```

mm-implicit.c (~/malloclab-handout) - VIM

```
 93 static void *extend_heap (size_t words) {
 94     char *bp;
 95     size_t size;
 96
 97     size = (words % 2) ? (words + 1) * WSIZE : words * WSIZE;
 98     if ((long)(bp = mem_sbrk(size)) == -1)
 99         return NULL;
100
101     PUT(HDRP(bp), PACK(size, 0));
102     PUT(FTRP(bp), PACK(size, 0));
103     PUT(HDRP(NEXT_BLKP(bp)), PACK(0, 1));
104
105     return coalesce(bp);
106 }
~/malloclab-handout/mm-implicit.c [utf-8,unix][c]                1,106/260 37%
```

mm-implicit.c (~/malloclab-handout) - VIM

```
111 int mm_init(void) {
112
113     if((heap_listp = mem_sbrk(4 * WSIZE)) == NULL)
114         return -1;
115
116     PUT(heap_listp, 0);
117     PUT(heap_listp + WSIZE, PACK(OVERHEAD, 1));
118     PUT(heap_listp + DSIZE, PACK(OVERHEAD, 1));
119     PUT(heap_listp + WSIZE + DSIZE, PACK(0, 1));
120     heap_listp += DSIZE;
121     next_fit = heap_listp;
122
123     if((extend_heap(CHUNKSIZE / WSIZE)) == NULL)
124         return -1;
125
126     return 0;
127 }
~/malloclab-handout/mm-implicit.c [utf-8,unix][c]                1,127/260 45%
```

mm-implicit.c (~/malloclab-handout) - VIM

```
129 static void *find_fit (size_t asize) {
130     void *bp;
131     bp = next_fit;
132     while(GET_SIZE(HDRP(bp)) > 0){
133         if( !GET_ALLOC(HDRP(bp)) && (asize <= GET_SIZE(HDRP(bp))) )
134             return bp;
135         bp = NEXT_BLKP(bp);
136     }
137     return NULL;
138 }
~/malloclab-handout/mm-implicit.c [utf-8,unix][c]                1,138/260 51%
```

mm-implicit.c (~/malloclab-handout) - VIM

```
140 static void place (void *bp, size_t asize) {
141     size_t fsize = GET_SIZE(HDRP(bp));
142     if((fsize - asize) >= (DSIZE * 2)){
143         PUT(HDRP(bp), PACK(asize, 1));
144         PUT(FTRP(bp), PACK(asize, 1));
145         bp = NEXT_BLKP(bp);
146         PUT(HDRP(bp), PACK(fsize - asize, 0));
147         PUT(FTRP(bp), PACK(fsize - asize, 0));
148     }
149     else {
150         PUT(HDRP(bp), PACK(fsize, 1));
151         PUT(FTRP(bp), PACK(fsize, 1));
152     }
153 }
~/malloclab-handout/mm-implicit.c [utf-8,unix][c]                1,153/260 56%
```

mm-implicit.c (~/malloclab-handout) - VIM

```
159 void *malloc (size_t size) {
160     size_t asize;
161     size_t extendsize;
162     char *bp;
163
164     if(size == 0)
165         return NULL;
166
167     if(size <= DSIZE)
168         asize = 2 * DSIZE;
169     else
170         asize = DSIZE * ((size + (DSIZE) + (DSIZE - 1)) / DSIZE);
171
172     if ((bp = find_fit(asize)) != NULL){
173         place(bp, asize);
174         next_fit = bp;
175         return bp;
176     }
177
178     extendsize = MAX(asize, CHUNKSIZE);
179     if((bp = extend_heap(extendsize/WSIZE)) == NULL)
180         return NULL;
181     place(bp, asize);
182     next_fit = bp;
183     return bp;
184 }
~/malloclab-handout/mm-implicit.c [utf-8,unix][c]                1,184/260 67%
```

mm-implicit.c (~/malloclab-handout) - VIM

```
189 void free (void *bp) {
190
191     if (bp == 0) return;
192     size_t size = GET_SIZE(HDRP(bp));
193
194     PUT(HDRP(bp), PACK(size, 0));
195     PUT(FTRP(bp), PACK(size, 0));
196
197     coalesce(bp);
198 }
~/malloclab-handout/mm-implicit.c [utf-8,unix][c]                1,198/260 75%
```

realloc은 naive와 동일

구현 방법

<매크로>

ALIGNMENT, ALIGN(p)는 naive와 동일

#define WSIZE 4 - word 크기

#define DSIZE 8 - double word 크기

#define CHUNKSIZE (1<<12) - 힙을 늘릴시에 사용 , 2^12

#define OVERHEAD 8 - header + footer 사이즈

#define MAX(x, y) ((x) > (y) ? (x) : (y)) - 더 큰 수를 찾아줌

#define PACK(size, alloc) ((size) | (alloc)) - size와 alloc을 한 byte로 묶음

#define GET(p) (*(unsigned int*) (p)) - 해당 위치의 정보를 반환

#define PUT(p, val) (*(unsigned int*) (p) = (val)) - p에 val를 삽입

#define GET_SIZE(p) (GET(p) & ~0x7) - 하위 3비트를 제외하고 반환하는데 사이즈를 읽어오는 것

#define GET_ALLOC(p) (GET(p) & 0x1) - 하위 1비트를 반환하는데 alloc정보를 읽어오는 것

#define HDRP(bp) ((char*) (bp) - WSIZE) - 블록에서 4바이트 뒤에 있는 header의 위치

#define FTRP(bp) ((char*) (bp) + GET_SIZE(HDRP(bp)) - DSIZE) - 다음 블록에서 8바이트 뒤에 있는 footer의 위치

#define NEXT_BLKP(bp) ((char*) (bp) + GET_SIZE((char*) (bp) - WSIZE)) - 해당 블록의 header를 읽어 사이즈를 알아낸 뒤, 사이즈를 더하여 다음 블록의 위치를 알아냄.

#define PREV_BLKP(bp) ((char*) (bp) - GET_SIZE((char*) (bp) - DSIZE)) - 해당 블록에서 8바이트 뒤로 이동하여 이전 블록의 footer를 읽어 사이즈를 알아낸 뒤, 사이즈를 빼서 이전 블록의 위치 알아냄

<mm_init>

mem_sbrk함수로 heap의 크기를 16만큼 증가시켜 공간을 할당한다. heap_listp는 새로 생성되는 heap의 시작주소가 되고, 이 값이 null일 경우엔 -1을 반환한다. (오류) null이 아닐 경우엔 초기 heap을 생성해준다. 먼저 정렬을 위해서 시작주소에 0을 삽입시킨다. 그리고 프롤로그와 에필로그를 삽입시키는데, 시작주소+4에 prologue header를 넣고 시작주소+8에 prologue footer를 넣고 시작주소+12에 epilogue header를 넣어준다. 그 담엔 heap_listp의 위치를 header와 footer사이로 이동시킨다. 마지막으로 CHUNKSIZE만큼 heap을 확장해준다.

<malloc>

실제 할당하는 사이즈(asize)는 사용자가 할당하려는 사이즈(size)보다 크다. (할당에 필요한 정보들이 저장되는 공간이 필요함) 할당하려는 사이즈가 0이면 NULL반환 (오류), 할당하려는 사이즈가 8byte이하면 실제로는 16byte를 할당해준다. (할당하려는 사이즈 + header, footer) 그게 아니면 (할당하려는 사이즈 + header, footer(DSIZE)) 를 8의 배수로 바꿔준 뒤, 만들어진 사이즈를 할당해준다. 할당할 땐 find_fit 함수로 free블록을 탐색한 뒤, place함수를 호출하여 할당시킨다. (단편화 줄이기 위해) 만약 free블록을 못찾았으면 힙공간을 늘려 다시 할당해준다. 할당이 끝나면 블록포인터를 리턴한다.

<place>

할당했을때 일정 사이즈(2*DSIZE)이상 빈 공간이 생길 것 같으면, header와 footer에 할당하려는 사이즈, alloc = 1을 넣어서 할당을 시켜준 뒤, NEXT_BKLP로 포인터를 이동시킨다. (할당하고 남은 공간으로 이동됨) 거기서 남은공간을 사이즈로 하고 alloc = 0으로 해서 free한 가용블록을 새로 만들어 준다.(블록쪼개기) 일정 사이즈미만으로 공간이 남을 것 같으면 그냥 그 공간을 다 할당시켜준다.

<extend_heap>

사이즈만큼 새로 힙을 할당한 후에, 새로운 free블럭의 header와 footer 그리고 epilogue header를 초기화 해준다. 그 뒤에 coalesce함수를 호출한다. 최종적인 bp를 반환한다.

<find_fit>

bp를 이동시키면서 할당되지 않았고, 할당에 필요한 여유 사이즈가 있는 블록을 찾는다. 찾으면 해당 포인터를 반환한다. 찾지 못하면 NULL리턴. free하는 위치에서부터 찾으면 first fit의 헛손질을 줄일 수 있으므로, next fit 사용

<mm_free>

먼저 잘못된 free요청인지 확인한 후에 잘못된 요청이면 함수를 종료한다. 문제없으면 ptr의 헤더에서 block의 size를 가져온다. 다음엔 PUT매크로로 ptr의 header와 footer에 block size와 alloc = 0을 저장해준다. 실제로 데이터를 지우지는 않았지만 최하위 비트인 alloc을 0으로 바꿔줌으로써 해당 block을 할당되지 않은 상태로 만들 수 있다.

주위에 빈 블록이 있으면 통합해주기 위해 coalesce함수를 호출한다.

<coalesce>

현재 블록을 기준으로 이전 블록의 footer의 alloc정보를 받아 prev_alloc에 저장하고, 현재 블록을 기준으로 다음 블록의 header의 alloc정보를 받아 next_alloc에 저장한다. 그리고 현재 블록의

header를 읽어 size도 저장한다.

(1) 앞, 뒤 블록이 다 할당되어있을땐 아무것도 하지 않고 바로 현재 블록의 포인터를 반환한다.

(2) 앞은 할당이 되어있고 뒤는 프리할땐 뒷 블록의 사이즈를 얻어 현재 사이즈에 더한다음 현재 블록의 header와 footer에 size와 alloc = 0을 저장해준다.

(3) 앞은 프리한데 뒤는 할당이 되어있을땐 앞 블록의 사이즈를 얻어 현재 사이즈에 더한다음 앞 블록의 header와 footer에 size와 alloc = 0을 저장해준다. 그 후엔 ptr을 앞블록의 위치로 옮겨준다.

(4) 앞, 뒤 블록이 모두 프리할땐 앞, 뒤 블록의 사이즈를 얻어 현재 사이즈에 모두 더해준다음 앞 블록의 header와 footer에 size와 alloc = 0을 저장해준다. 그 후엔 ptr을 앞불록의 위치로 옮겨준다.

<realloc>

naive와 동일하게 구현