2018 시스템 프로그래밍
- Lab 09 -

제출일자	2018.11.26
분 반	00
이 름	노 효 근
학 번	201502049

1. mm-naive

Thanks for flying Vim

```
a201502049@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 79453 ./traces/malloc.rep
   yes     77%      17  0.000000108057 ./traces/malloc-free.rep
   yes    100%      15  0.000000 89640 ./traces/corners.rep
 * yes     71%    1494  0.000010155399 ./traces/perl.rep
 * yes     68%     118  0.000001158054 ./traces/hostname.rep
 * yes     65%   11913  0.000078151861 ./traces/xterm.rep
 * yes     23%    5694  0.000076 75319 ./traces/amptjp-bal.rep
 * yes     19%    5848  0.000070 83562 ./traces/cccp-bal.rep
 * yes     30%    6648  0.000084 79225 ./traces/cp-decl-bal.rep
 * yes     40%    5380  0.000060 89225 ./traces/expr-bal.rep
 * yes      0%   14400  0.000189 76061 ./traces/coalescing-bal.rep
 * yes     38%    4800  0.000052 93148 ./traces/random-bal.rep
 * yes     55%    6000  0.000063 95055 ./traces/binary-bal.rep
10         41%   62295  0.000683 91265

Perf index = 26 (util) + 40 (thru) = 66/100
a201502049@2018-sp:~/malloclab-handout$
```

소스 코드

```
40 /* single word (4) or double word (8) alignment */
41 #define ALIGNMENT 8  // double word8  
42
43 /* rounds up to the nearest multiple of ALIGNMENT */
44 #define ALIGN(size) (((size) + (ALIGNMENT-1)) & ~0x7)
45 //
46 //
47 //
48
49 #define SIZE_T_SIZE (ALIGN(sizeof(size_t)))
50
51 #define SIZE_PTR(p)  ((size_t*)(((char*)(p)) - SIZE_T_SIZE))
52 //
```

구현 방법

aasdasdasdas

#define ALIGNMENT 8

: malloc program은 double word의 방식을 이용하므로 payload를 8의 배수로 설정

#define ALIGN(p) (((size_t)(p) + (ALIGNMENT-1)) & ~0x7)

: 할당 사이즈에 &를 더하고, 뒤 3비트를 제거하여 가장 가까운 8의 배수로 올림을 하여 payload를 8의 배수로 만드는 매크로.

#defie SIZE_T_SIZE (ALIGN(sizeof(size_t)))
: size_t 타입의 크기를 ALGN한 매크로, size_t의 크기가 4byte이므로,
 8단위를 만들면 8byte가 만들어짐.

#define SIZE_PTR(p) ((size_t*)(((char*)(p)) - SIZE_T_SIZE))
: 해당 노드의 사이즈가 포함된 word로 이동시키는 매크로

int mm_init(void)
: heap을 초기화하는 함수(미구현 함수)

void *malloc(size_t size)
: size크기의 블록을 heap에 할당 ALIGN으로 입력받은 size를 SIZE_T SIZE. 즉, 8을 더한 후 이 값보다 크거나 같은 8의 배수를 구하여 newsize에 넣어준 후, 해당 newsize만큼의 저장 공간을 p에 넣어줌. p<0인 경우 저장 공간을 받지 못하였으므로 NULL을 리턴. 그렇지 않은 경우는 p에 8을 더하고, p의 사이즈 값에 입력받은 size를 넣어줌. p 리턴하는 함수

void free(void *ptr)
: 할당된 ptr 메모리를 해제하여 가용 메모리로 바꿔줌(미구현 함수)

void *realloc(void *oldptr, size_t size)
: 이미 할당되어 있는 oldptr 메모리의 크기를 size로 재 할당, 이전 메모리를 free하는 함수
1. size == 0인 경우
 - free와 같으므로 free 후 리턴
2. oldptr이 할당 받지 않은 메모리인 경우
 - malloc으로 입력받은 size만큼 할당
3. 위의 두 경우 다 아닌 경우
 - newptr에 malloc으로 입력받은 size만큼 할당, 할당이 잘 되었는지 확인 후
old data를 복사하기 위해, oldsize에 oldptr의 size 넣어주고 원래의 size와 할당받기 원하는 size의 크기를 비교, 작은 값을 oldsize에 넣어주고 memcpy 함수를 이용하여 원래 oldptr에 있었던 정보를 newptr에 oldsize만큼 복사해줌. 이때, 입력 받은 size가 작으면 입력 받은 size만큼 복사해주고, 입력받은 size가 더 크면 oldptr의 모든 정보를 복사함. 그 다음 oldptr free시켜준 후 할당받은 newptr 리턴

void *calloc(size_t nmemb, size_t size)
: malloc과 같이 저장 공간을 할당해 주고 0으로 초기화 malloc과 비슷하지만 memset을 이용하여 모두 0으로 초기화

void mm_checkheap(int verbose)
: heap을 체크해줌(미구현 함수)

2. mm-implicit

```
a201502049@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 40651 ./traces/malloc.rep
   yes    28%      17  0.000000 69106 ./traces/malloc-free.rep
   yes    96%      15  0.000000 52969 ./traces/corners.rep
 * yes    81%    1494  0.000260  5737 ./traces/perl.rep
 * yes    50%     118  0.000014  8542 ./traces/hostname.rep
 * yes    87%   11913  0.017828   668 ./traces/xterm.rep
 * yes    97%    5694  0.000856  6649 ./traces/amptjp-bal.rep
 * yes    95%    5848  0.002781  2103 ./traces/cccp-bal.rep
 * yes    94%    6648  0.001027  6475 ./traces/cp-decl-bal.rep
 * yes    93%    5380  0.000481 11183 ./traces/expr-bal.rep
 * yes    66%   14400  0.000127113239 ./traces/coalescing-bal.rep
 * yes    90%    4800  0.003391  1415 ./traces/random-bal.rep
 * yes    55%    6000  0.000118 50960 ./traces/binary-bal.rep
10        81%   62295  0.026884  2317

Perf index = 52 (util) + 40 (thru) = 92/100
```

소스 코드

```
29 /* do not change the following! */
30 #ifdef DRIVER
31 /* create aliases for driver tests */
32 #define malloc mm_malloc
33 #define free mm_free
34 #define realloc mm_realloc
35 #define calloc mm_calloc
36 #endif /* def DRIVER */
37
38 /* single word (4) or double word (8) alignment */
39 #define ALIGNMENT 8
40
41 /* rounds up to the nearest multiple of ALIGNMENT */
42 #define ALIGN(p) (((size_t)(p) + (ALIGNMENT-1)) & ~0x7)
43
44 /*Basic constants and macros*/
45 #define WSIZE 4
46 #define DSIZE 8
47 #define CHUNKSIZE (1<<12)
48 #define OVERHEAD 8
49
50 #define MAX(x,y) ((x)>(y) ? (x):(y))
51 #define PACK(size,alloc) ((size)| (alloc))
52 #define GET(p) (*(unsigned int*)(p))
53 #define PUT(p,val) (*(unsigned int*)(p) = (val))
54 #define GET_SIZE(p) (GET(p) & ~0x7)
55 #define GET_ALLOC(p) (GET(p) & 0x1)
56
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 static void *extend_heap(size_t words);
63 static void *coalesce(void *bp);
64 static void place(void *bp,size_t asize);
65 static void *find_fit(size_t asize);
66 static char *heap_listp = 0;
67 static char *oldbp;
```

구현 방법

MACRO :

```
#define WSIZE 4 //word의 크기를 4로 설정
#define DSIZE 8 //double word 크기를 8로 설정
#define CHUNKSIZE (1<<12) //초기 heap의 크기를 설정, 확장 사이즈 4096를 매크로로 정의
#define OVERHEAD 8 //head(4 byte) + footer(4 byte) = 8byte 설정

#define MAX(x,y) ((x)>(y) ? (x):(y)) // x와 y 값 중에서 더 큰 값 고름
#define PACK(size,alloc) ((size)| (alloc)) //size와 alloc을 하나의 word로 묶어서 쉽게 header와 footer에 저장
#define GET(p) (*(unsigned int*)(p)) //포인터 p가 가리키는 위치에서 word크기의 값을 읽음
#define PUT(p,val) (*(unsigned int*)(p) = (val)) //포인터 p가 가리키는 곳에 word크기의 val 값을 씀
#define GET_SIZE(p) (GET(p) & ~0x7) //포인터 p가 가리키는 곳에서 한 word를 읽은 다음 하위 3bit를 버림
#define GET_ALLOC(p) (GET(p) & 0x1) //포인터 p가 가리키는 곳에서 한 word를 읽은 다음 하위 1bit를 읽음

#define HDRP(bp) ((char*)(bp) - WSIZE) //주어진 포인터 bp의 header 주소를 계산
#define FTRP(bp) ((char*)(bp) + GET_SIZE(HDRP(bp)) - DSIZE) //주어진 포인터 bp의 footer 주소를 계산
#define NEXT_BLKP(bp) ((char*)(bp) + GET_SIZE(((char*)(bp) - WSIZE))) //주어진 포인터 bp를 이용하여 다음 block의 주소를 계산
#define PREV_BLKP(bp) ((char*)(bp) - GET_SIZE(((char*)(bp) - DSIZE))) //주어진 포인터 bp를 이용하여 이전 block의 주소를 계산

static void *extend_heap(size_t words); // heap 크기 증가 함수
static void *coalesce(void *bp); // free 블록 메모리 합성 함수
static void place(void *bp,size_t asize); // bp를 받아 asize 만큼 할당하는 함수
static void *find_fit(size_t asize); // asize 만큼 메모리 할당을 위해 free 블록 탐색 함수
static char *heap_listp = 0; //heap 영역의 시작 주소 저장
static char *oldbp;
```

FUNCTION :

```
int mm_init(void) { //heap 생성 및 초기화

        if((heap_listp = mem_sbrk(4 * WSIZE)) == NULL)
                return -1;
        //새로 생성되는 heap 영역 시작 주소 저장
        PUT(heap_listp,0); // 정렬을 위해 0 값 삽입
        PUT(heap_listp + WSIZE, PACK(DSIZE, 1)); // 프롤로그 header에 9(8+1) 저장
        PUT(heap_listp + DSIZE, PACK(DSIZE, 1)); // 프롤로그 footer에 9(8+1) 저장
        PUT(heap_listp + WSIZE + DSIZE, PACK(0,1)); // 에필로그 header 에 size 0 할당
        heap_listp += DSIZE; // 세번째 블록

        oldbp = heap_listp;

        if ((extend_heap(CHUNKSIZE / WSIZE))== NULL)
                return -1;
        //CHUNKSIZE 의 free block만큼 empty heap 확장 및 empty heap을 free block으로 확장

        return 0;
}
```

```
void *malloc (size_t size) {
        size_t asize;
        size_t extendsize;
        char *bp;

        if(size==0)
                return NULL;  // size ==0 할당하지 않음

        if(size<=DSIZE)
                asize = 2*DSIZE;  // size < 8 이하 인경우   asize = 16
        else
                asize = DSIZE*((size+(DSIZE)+(DSIZE-1))/DSIZE);  size + 8 보다 큰 8의 배수 저장

        if((bp = find_fit(asize)) != NULL){
                place(bp,asize);
        return bp;
        } // find_fit을 이용하여 bp의 위치 탐색

        extendsize = MAX(asize,CHUNKSIZE);
        if((bp = extend_heap(extendsize/WSIZE)) == NULL){
                return NULL;
        }
        place(bp,asize); // asize 만큼 할당
        oldbp = NEXT_BLKP(bp);
        return bp;
}
```

```
void free (void *ptr) {
  if(!ptr) return;
         size_t size = GET_SIZE(HDRP(ptr)); //ptr header 에서 block size 탐색

         PUT(HDRP(ptr), PACK(size,0)); //header , block size,  alloc = 0
         PUT(FTRP(ptr), PACK(size,0)); // footer, block size, alloc = 0

         coalesce(ptr);  // block 병합
}
```

void *realloc(void *oldptr, size_t size) : naive와 동일 구현

void *calloc (size_t nmemb, size_t size) : naive와 동일 구현

```
static void *extend_heap(size_t words){
        char *bp;
        size_t size;
        size = (words % 2) ? (words + 1) * WSIZE : words * WSIZE;

        // size가 짝수인 경우와 그렇지 않은 경우 각각 size 값을 설정

        if((long)(bp = mem_sbrk(size)) == -1)
                return NULL;

        // size 만큼 힙 영역 확장

        PUT(HDRP(bp), PACK(size,0));
        PUT(FTRP(bp), PACK(size,0));
        PUT(HDRP(NEXT_BLKP(bp)), PACK(0,1));

        //header, footer, next_block_header 각각 맞는 값으로 설정

        return coalesce(bp); //bp가 가리키는 free block 주소로 coalesce 함수 호출
}
```

```
//asize 만큼 메모리 할당을 위해 free block 탐색 함수
static void *find_fit(size_t asize){

        // Next fit 으로 구현,  oldbp를 이용하여 포인터의 다음 위치부터 시작, 책 참고
        char *p;
        for(p = oldbp; GET_SIZE(HDRP(p))>0; p = NEXT_BLKP(p)){
                if(!GET_ALLOC(HDRP(p)) && (asize <= GET_SIZE(HDRP(p)))){
                        return p;
                }
        }
        return NULL;
}
```

```
// bp를 받아 asize 만큼 할당 함수
static void place(void *bp, size_t asize){
        size_t csize = GET_SIZE(HDRP(bp)); // ptr이 가르키는 블록의 사이즈 저장

        if((csize - asize) >= (2*DSIZE)){
                //할당할 사이즈와 free블록의 사이즈 > 차가 16이상 인경우
                PUT(HDRP(bp), PACK(asize,1)); //header에 asize저장, 마지막비트1
                PUT(FTRP(bp), PACK(asize,1)); //footer에 asize저장, 마지막비트1

                bp = NEXT_BLKP(bp);//다음 블록저장
                PUT(HDRP(bp), PACK(csize-asize, 0));
                PUT(FTRP(bp), PACK(csize-asize, 0));
                // 남은크기를 header,footer에 저장하고 마지막 비트0
        }
        else{
                PUT(HDRP(bp), PACK(csize,1));
                PUT(FTRP(bp), PACK(csize,1));
                //header, footer에 csize 저장하고 마지막 비트 1
        }
}
```

```
//free block 메모리 합병 함수
static void *coalesce(void *bp){
        size_t prev_alloc = GET_ALLOC(FTRP(PREV_BLKP(bp))); //이전블록의 할당여부
        size_t next_alloc = GET_ALLOC(HDRP(NEXT_BLKP(bp))); //다음블록의 할당여부
        size_t size = GET_SIZE(HDRP(bp)); // bp의 블럭사이즈

        if(prev_alloc && next_alloc){ //둘다 할당되었을 시 종료하고 끝냄
                return bp;
        }
        else if(prev_alloc && !next_alloc){ //다음블록만 free인 경우
                size += GET_SIZE(HDRP(NEXT_BLKP(bp))); //다음 free 블록 사이즈를 원래에 더함
                PUT(HDRP(bp), PACK(size, 0));
                PUT(FTRP(bp), PACK(size, 0));
                //새로운 header,footer에 변경된 size를 넣어줌, 마지막 비트 0으로 셋팅
        }
        else if(!prev_alloc && next_alloc){ //이전 블록만 free인 경우
                size += GET_SIZE(HDRP(PREV_BLKP(bp)));
                PUT(FTRP(bp), PACK(size, 0));
                PUT(HDRP(PREV_BLKP(bp)), PACK(size,0));
                bp = PREV_BLKP(bp);//이전 블록에 값을 더했으므로 새로운 free이 이전블록을 가르킴
        }
        else if(!prev_alloc && !next_alloc){ //둘다 free인 경우
                size += GET_SIZE(HDRP(PREV_BLKP(bp)))+ GET_SIZE(FTRP(NEXT_BLKP(bp)));
                // 이전, 다음 free 블록을 원래 사이즈에 더함
                PUT(HDRP(PREV_BLKP(bp)), PACK(size, 0));
                PUT(FTRP(NEXT_BLKP(bp)), PACK(size, 0));
                bp = PREV_BLKP(bp);
        }
        if(oldbp > bp)
```