시스템 프로그래밍 malloc lab 보고서

2017-12751 컴퓨터공학부 이동학

이번 lab에서는 mm.c 파일에 있는 4개의 함수 mm_init, *mm_malloc, mm_free, mm_realloc을 완성하여 정확하고 효율적이고 빠른 dynamic storage allocator를 구현하는 것을 목표로 하였습니다.

0. description

```
/*
2017-12751 Donghak Lee Malloc Lab

 < Allocated Block >

31 30  ......              2  1  0
┌──┬──┬──┬──┬──┬──┬──┬──┬──┬──┐
|   size of the block   |  |  |  | -> header, check allocated at 0 bit
├──┼──┼──┼──┼──┼──┼──┼──┼──┼──┤
......
├──┼──┼──┼──┼──┼──┼──┼──┼──┼──┤
|   size of the block   |  |  |  | -> footer, check allocated at 0 bit
└──┴──┴──┴──┴──┴──┴──┴──┴──┴──┘

 < Free block >

31 30 ......              2  1  0
┌──┬──┬──┬──┬──┬──┬──┬──┬──┬──┐
|   size of the block   |  |  |  | -> header, check allocated at 0 bit
├──┼──┼──┼──┼──┼──┼──┼──┼──┼──┤
|                                | -> pred_ptr, access by macro
├──┼──┼──┼──┼──┼──┼──┼──┼──┼──┤
|                                | -> succ_ptr, access by macro
├──┼──┼──┼──┼──┼──┼──┼──┼──┼──┤
......
├──┼──┼──┼──┼──┼──┼──┼──┼──┼──┤
|   size of the block   |  |  |  | -> footer, check allocated at 0 bit
└──┴──┴──┴──┴──┴──┴──┴──┴──┴──┘
 */
```

allocated block과 free block의 구현은 위와 같습니다. header와 footer의 0번 bit로 allocated 여부를 확인하고 free block의 경우에는 pred_ptr와 succ_ptr를 이용합니다.

1. macro & helper functions

```c
/* single word (4) or double word (8) alignment */
#define ALIGNMENT 8

/* rounds up to the nearest multiple of ALIGNMENT */
#define ALIGN(size) (((size) + (ALIGNMENT-1)) & ~0x7)

#define WSIZE     4
#define DSIZE     8
#define CHUNKSIZE (1<<8)

#define MAX(x, y) ((x) > (y) ? (x) : (y))
#define MIN(x, y) ((x) < (y) ? (x) : (y))

/* Pack a size and allocated bit into a word */
#define PACK(size, alloc) ((size) | (alloc))

/* Read and write a word at address p */
#define GET(p)            (*(unsigned int *)(p))
#define PUT(p, val)       (*(unsigned int *)(p) = (val))

/* Read the size and allocated fields from address p */
#define GET_SIZE(p)  (GET(p) & ~0x7)
#define GET_ALLOC(p) (GET(p) & 0x1)

/* Given block ptr bp, compute address of its header and footer */
#define HDRP(bp) ((char *)(bp) - WSIZE)
#define FTRP(bp) ((char *)(bp) + GET_SIZE(HDRP(bp)) - DSIZE)

/* Given block ptr bp, compute address of next and previous blocks */
#define NEXT_BLKP(bp) ((char *)(bp) + GET_SIZE((char *)(bp) - WSIZE))
#define PREV_BLKP(bp) ((char *)(bp) - GET_SIZE((char *)(bp) - DSIZE))

#define LIST_NUM 24

#define SET_PTR(p, bp) (*(unsigned int *)(p) = (unsigned int)(bp))

#define PRED_PTR(bp) ((char *)(bp))
#define SUCC_PTR(bp) ((char *)(bp) + WSIZE)

#define PRED(bp) (*(char **)(bp))
#define SUCC(bp) (*(char **)(SUCC_PTR(bp)))

void **free_seg_list;
char *heap_list;
static void *extend_heap(size_t size);
static void *coalesce(void *bp);
static void *place(void *bp, size_t asize);
static void add_node(void *bp, size_t size);
static void remove_node(void *bp);
```

교재에 있는 상수와 매크로, helper function들을 포함하여 힙 리스트 구현에 필요한 매크로와 helper function들을 추가하였습니다. set_ptr은 포인터를 저장하고 PRED_PTR, SUCC_PTR은 predecessor와 successor entry의 주소를, PRED와 SUCC은 segregated list의 predecessor와 successor의 주소를 저장합니다.

2. mm_init

```
int mm_init(void)
{
    char *heap_start;

    if((heap_start = mem_sbrk(LIST_NUM * WSIZE)) == NULL)
        return -1;

    free_seg_list = (void**) heap_start;

    for(int i=0; i<LIST_NUM; i++)
        free_seg_list[i] = NULL;

    /* Create the initial empty heap */
    if((heap_start = mem_sbrk(4 * WSIZE)) == NULL)
        return -1;

    PUT(heap_start, 0);                             /* Alignment padding */
    PUT(heap_start + (WSIZE * 1), PACK(DSIZE, 1)); /* Prologue header */
    PUT(heap_start + (WSIZE * 2), PACK(DSIZE, 1)); /* Prologue footer */
    PUT(heap_start + (WSIZE * 3), PACK(0, 1));     /* Epilogue header */

    /* Extend the empty heap with a free block of CHUNKSIZE bytes */
    if(extend_heap(CHUNKSIZE) == NULL)
        return -1;

    return 0;
}
```

교재에 있는 mm_init을 참고하여 구현하였습니다. mm_malloc, mm_free, mm_realloc을 call하기 전에 mm_init을 call하여 initial heap area를 allocate 합니다.

3. *mm_malloc

```
void *mm_malloc(size_t size)
{
    /* Adjust block size to include overhead and alignment reqs. */
    size_t asize = MAX(2 * DSIZE, ALIGN(size + DSIZE));
    size_t extendsize;  /* Amount to extend heap if no fit */

    /* Ignore spurious requests */
    if (size == 0)
        return NULL;

    /* Search the free list for a fit */
    void *bp = NULL;
    size_t searchsize = asize;
    for (int seg_size = 0; seg_size < LIST_NUM; seg_size++)
    {
        if (((searchsize <= 1) && (free_seg_list[seg_size] != NULL))
            ||(seg_size == LIST_NUM - 1)) {
            bp = free_seg_list[seg_size];

            while ((bp != NULL) && ((asize > GET_SIZE(HDRP(bp)))))
                bp = PRED(bp);

            if (bp != NULL)
                break;
        }
        searchsize /= 2;
    }

    /* No fit found. Get more memory and place the block */
    if (bp == NULL)
    {
        extendsize = MAX(asize, CHUNKSIZE);
        if ((bp = extend_heap(extendsize)) == NULL)
            return NULL;
    }

    bp = place(bp, asize);
    return bp;
}
```

교재에 있는 mm_malloc을 참고하여 구현하였습니다. block size를 align한 후 segregated list에서 free block을 찾되, 너무 작은 경우에는 무시합니다. 만약 free block을 찾지 못하면 heap을 extend합니다. 그 후 block을 place하고 새로 allocate된 block의 포인터를 return합니다.

4. mm_free

```
void mm_free(void *bp)
{
    size_t size = GET_SIZE(HDRP(bp));

    PUT(HDRP(bp), PACK(size, 0));
    PUT(FTRP(bp), PACK(size, 0));
    add_node(bp, size);
    coalesce(bp);
}
```

교재에 있는 mm_free를 참고하여 구현하였습니다. block pointer의 block을 free합니다.

5. *mm_realloc

```
void *mm_realloc(void *bp, size_t size)
{
    void *new_ptr = bp;
    size_t asize = MAX(ALIGN(size + DSIZE), 2 * DSIZE);

    if (size == 0)
        return NULL;

    if (GET_SIZE(HDRP(bp)) < asize) {
        if (!GET_ALLOC(HDRP(NEXT_BLKP(bp))) || !GET_SIZE(HDRP(NEXT_BLKP(bp)))) {
            int surplus = GET_SIZE(HDRP(NEXT_BLKP(bp))) + GET_SIZE(HDRP(bp)) - asize;

            if (surplus < 0) {
                if (extend_heap(MAX(-surplus, CHUNKSIZE)) == NULL)
                    return NULL;
                surplus += MAX(-surplus, CHUNKSIZE);
            }

            remove_node(NEXT_BLKP(bp));
            PUT(HDRP(bp), PACK(asize + surplus, 1));
            PUT(FTRP(bp), PACK(asize + surplus, 1));
        }
        else {
            new_ptr = mm_malloc(asize - DSIZE);
            memcpy(new_ptr, bp, MIN(size, asize));
            mm_free(bp);
        }
    }

    return new_ptr;
}
```

block size를 align한 후 overhead가 align값에 미치지 못하면 다음 block이 free block이거나 epilogue block인지 확인한 후 heap을 extend하고 메모리에 write합니다. 아니라면 mm_malloc과 mm_free를 사용하여 새로 메모리를 할당해줍니다.

6. *extend_heap

```
static void *extend_heap(size_t size)
{
    void *bp;
    size_t asize = ALIGN(size);

    /* Allocate an even number of words to maintain alignment */
    if ((bp = mem_sbrk(asize)) == (void *)-1)
        return NULL;

    /* Initialize free block header/footer and the epilogue header */
    PUT(HDRP(bp), PACK(asize, 0));          /* Free block header */
    PUT(FTRP(bp), PACK(asize, 0));          /* Free block footer */
    PUT(HDRP(NEXT_BLKP(bp)), PACK(0, 1));   /* New epilogue header */
    add_node(bp, asize);

    /* Coalesce if the previous block was free */
    return coalesce(bp);
}
```

교재에 있는 *extend_heap을 참고하여 구현하였습니다. size를 align하고 그만큼 heap area에 allocate한 후 header와 footer, epilouge header를 설정합니다.

7. coalesce

```
static void *coalesce(void *bp)
{
    size_t prev_alloc = GET_ALLOC(HDRP(PREV_BLKP(bp)));
    size_t next_alloc = GET_ALLOC(HDRP(NEXT_BLKP(bp)));
    size_t size = GET_SIZE(HDRP(bp));

    if (prev_alloc && next_alloc) {                         /* Case 1 */
        return bp;
    }

    else if (prev_alloc && !next_alloc) {                   /* Case 2 */
        remove_node(bp);
        remove_node(NEXT_BLKP(bp));
        size += GET_SIZE(HDRP(NEXT_BLKP(bp)));
        PUT(HDRP(bp), PACK(size, 0));
        PUT(FTRP(bp), PACK(size, 0));
    }

    else if (!prev_alloc && next_alloc) {                   /* Case 3 */
        remove_node(bp);
        remove_node(PREV_BLKP(bp));
        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);
    }

    else {                                                  /* Case 4 */
        remove_node(bp);
        remove_node(PREV_BLKP(bp));
        remove_node(NEXT_BLKP(bp));
        size += GET_SIZE(HDRP(PREV_BLKP(bp))) + GET_SIZE(HDRP(NEXT_BLKP(bp)));
        PUT(HDRP(PREV_BLKP(bp)), PACK(size, 0));
        PUT(FTRP(NEXT_BLKP(bp)), PACK(size, 0));
        bp = PREV_BLKP(bp);
    }

    add_node(bp, size);

    return bp;
}
```

교재에 있는 coalesce를 참고하여 구현하였습니다.

prev와 next가 모두 allocate된 경우 coalesce할수 없으므로 넘깁니다.

next가 free인 경우 curr가 next에 merge 되고 curr의 header와 next의 footer를 업데이트 합니다.
prev가 free인 경우 prev가 curr에 merge 되고 prev의 header와 curr의 footer를 업데이트 합니다.
prev와 next가 둘다 free인 경우 셋 다 merge 되고 prev의 header와 next의 footer를 업데이트 합니다.

8. place

```
static void *place(void *bp, size_t asize)
{
    size_t ptr_size = GET_SIZE(HDRP(bp));
    size_t surplus = ptr_size - asize;

    remove_node(bp);

    if (surplus <= DSIZE * 2) {
        PUT(HDRP(bp), PACK(ptr_size, 1));
        PUT(FTRP(bp), PACK(ptr_size, 1));
    }
    else {
        PUT(HDRP(bp), PACK(asize,1));
        PUT(FTRP(bp), PACK(asize,1));
        PUT(HDRP(NEXT_BLKP(bp)), PACK(surplus, 0));
        PUT(FTRP(NEXT_BLKP(bp)), PACK(surplus, 0));
        add_node(NEXT_BLKP(bp),surplus);
        return bp;
    }

    return bp;
}
```

교재에 있는 place를 참고하여 구현하였습니다. 만약 split한 나머지가 minimum block size보다 크다면 block을 split합니다.

9. add_node

```
static void add_node(void *bp, size_t size) {

    int seg_size = 0;

    while ((seg_size < LIST_NUM - 1) && (size > 1)) {
        size /= 2;
        seg_size++;
    }

    void *head = free_seg_list[seg_size];
    void *prev = NULL;
    while ((head != NULL) && (size > GET_SIZE(HDRP(head)))) {
        prev = head;
        head = PRED(head);
    }

    if (head != NULL) {
        if (prev != NULL) {
            SET_PTR(PRED_PTR(bp), head);
            SET_PTR(SUCC_PTR(head), bp);
            SET_PTR(SUCC_PTR(bp), prev);
            SET_PTR(PRED_PTR(prev), bp);
        }
        else {
            SET_PTR(PRED_PTR(bp), head);
            SET_PTR(SUCC_PTR(head), bp);
            SET_PTR(SUCC_PTR(bp), NULL);
            free_seg_list[seg_size] = bp;
        }
    }
    else {
        if (prev != NULL) {
            SET_PTR(PRED_PTR(bp), NULL);
            SET_PTR(SUCC_PTR(bp), prev);
            SET_PTR(PRED_PTR(prev), bp);
        }
        else {
            SET_PTR(PRED_PTR(bp), NULL);
            SET_PTR(SUCC_PTR(bp), NULL);
            free_seg_list[seg_size] = bp;
        }
    }
}
```

seg_list에 node를 추가하는 함수입니다. segregated list를 선택하고 크기에 대한 오름차순으로 search한 후 predecessor와 successor를 설정합니다.

10. remove_node

```
static void remove_node(void *bp) {

    size_t size = GET_SIZE(HDRP(bp));
    int seg_size = 0;

    while ((seg_size < LIST_NUM - 1) && (size > 1)) {
        seg_size++;
        size /= 2;
    }

    if(SUCC(bp) != NULL) {
        if(PRED(bp) != NULL) {
            SET_PTR(SUCC_PTR(PRED(bp)), SUCC(bp));
            SET_PTR(PRED_PTR(SUCC(bp)), PRED(bp));
        }
        else {
            SET_PTR(PRED_PTR(SUCC(bp)), NULL);
        }
    }
    else {
        if(PRED(bp) != NULL) {
            SET_PTR(SUCC_PTR(PRED(bp)), NULL);
            free_seg_list[seg_size] = PRED(bp);
        }
        else {
            free_seg_list[seg_size] = NULL;
        }
    }
}
```

seg_list에서 node를 지우는 함수입니다. segregated list를 선택하고 node를 삭제합니다.

후기: 지금까지 편리하게 사용해왔던 malloc 을 직접 구현해본다는 점에서 흥미웠습니다. textbook 을 이해하고 example 을 참고하여 구현을 시작하니 어느정도 수준까지는 수월하게 구현할 수 있었지만, 완벽에 가깝게 구현하는 것은 어려울 것 같습니다.

textbook 을 참고한 것을 감안해도 전체적인 난이도가 link lab 이나 shell lab 에 너무나도 높게 느껴졌고 실제로도 많은 시간을 할애했지만, 그만큼 결과를 내니 뿌듯함을 느낄 수 있었습니다.