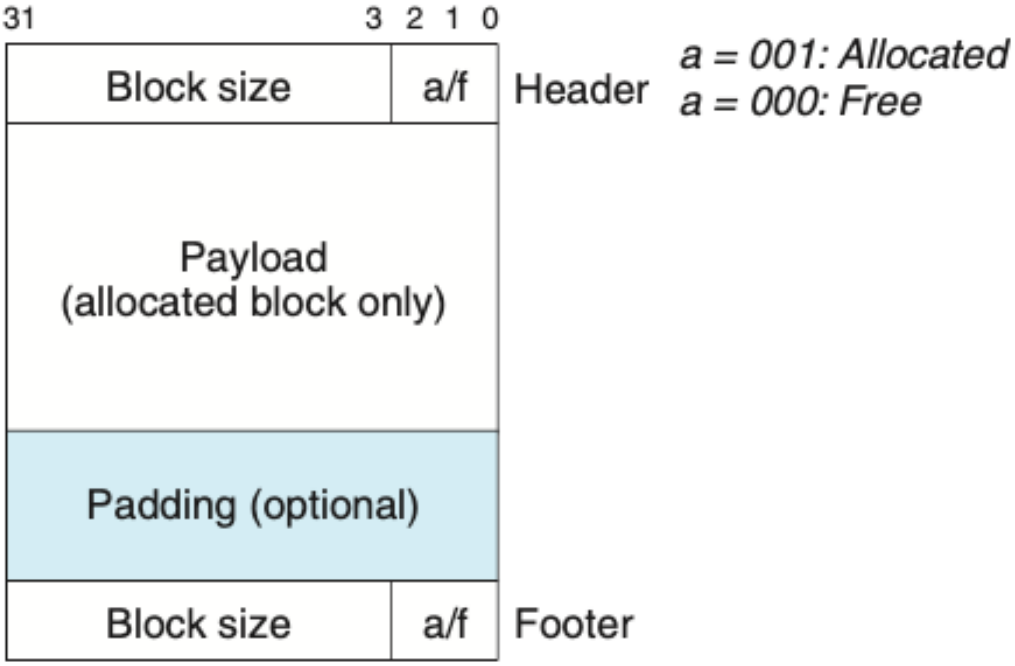
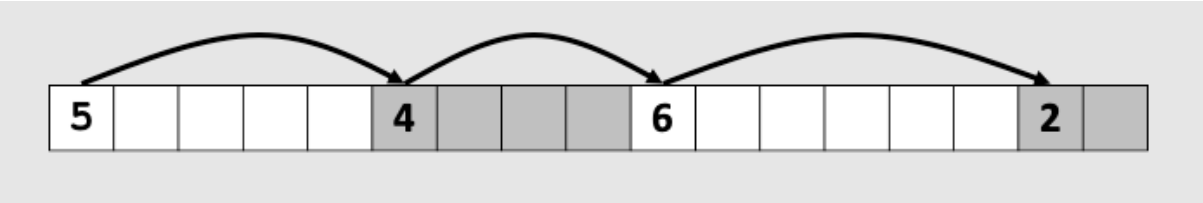
System Programming
Project 3
20181662 이건영

1. 설계

Dynamic Memory Allocator를 구현하기 위한 방식으로는 implicit list 구조를 이용하였다.



위 그림과 같이 하나의 블록에 Block size, 또 하위 3비트를 사용하여 해당 block의 allocate 상태와 free 상태를 나타낼 수 있도록 하였다.



위 그림처럼 모든 block이 연결되어 있다. Free block만을 관리하는 별도의 free list는 존재하지 않는 방식으로 구현하였다.

2. 구현

추가적으로 정의한 변수는 WSIZE, DSIZE, CHUNKSIZE이다. 각각 word와 Double word(4, 8byte), 그리고 heap을 확장할 size를 의미한다.

```
#define WSIZE       4       /* Word and header/footer size (bytes) */
#define DSIZE       8       /* Double word size (bytes) */
#define CHUNKSIZE   (1<<12) /* Extend heap by this amount (bytes) */
```

다음으로 macro들을 추가하였다.

MAX(x, y)는 x, y중 큰 값을 반환한다. PACK(size, alloc)는 block의 header부분에 들어갈 정보를 하나로 묶는다. 즉, size와 allocate된 정보를 하나로 합친다.

GET(p)와 PUT(p, val)은 각각 해당 pointer에서 하나의 word를 읽거나 쓴다.

```
#define WSIZE       4       /* Word and header/footer size (bytes) */
#define DSIZE       8       /* Double word size (bytes) */
#define CHUNKSIZE   (1<<12) /* Extend heap by this amount (bytes) */

#define MAX(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))
```

GET_SIZE와 GET_ALLOC은 header의 하위 비트를 사용하여 해당 block의 size와 allocate 상태를 알려준다. HDPR와 FTRP는 block pointer인 bp에 해당하는 block의 header와 footer의 위치를 알려준다.

NEXT_BLKP와 PREV_BLKP는 해당 bp에 해당하는 block의 다음과 이전 block의 주소를 알려준다.

```
/* Read the size and allocated fields form 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)))
```

전역 변수로 heap을 가리키는 pointer인 heap_listp를 선언하였다. 추가적으로 구현한 함수는 다음과 같다.

```
static void *extend_heap(size_t words);
static void *coalesce(void *bp);
static void *find_fit(size_t asize);
static void place(void *bp, size_t asize);
```

extend_heap 함수는 sbrk를 호출하여 현재 heap에 할당된 메모리의 크기를 늘려주는 역할을 한다. Coalesce 함수는 false fragmentation을 방지하기 위해 free나 extend_heap 시에 해당 block의 앞과 뒤 block을 확인하여 free block이라면 합쳐주는 역할을 한다. Find_fit 함수는 first fit 방식으로 malloc시 적절한 위치를 찾는 역할을 한다. Place 함수는 해당 block에 메모리를 할당할 때, 적당한 size만큼만 block을 나누어 할당하도록 하는 역할을 한다.

Mm_malloc(size_t size)

```
void *mm_malloc(size_t size)
{
    /*int newsize = ALIGN(size + SIZE_T_SIZE);
    void *p = mem_sbrk(newsize);
    if (p == (void *)-1)
    return NULL;
    else {
        *(size_t *)p = size;
        return (void *)((char *)p + SIZE_T_SIZE);
    }*/
    size_t asize;
    size_t extendsize;
    char *bp;

    if(size == 0)
        return NULL;

    if(size <= DSIZE)
        asize = 2*DSIZE;
    else
        asize = DSIZE * ((size + (DSIZE) + (DSIZE-1)) / DSIZE);

    if((bp = find_fit(asize)) != NULL) {
        place(bp, asize);
        return bp;
    }

    extendsize = MAX(asize, CHUNKSIZE);
    if((bp = extend_heap(extendsize/WSIZE)) == NULL)
        return NULL;
    place(bp, asize);
    return bp;
}
```

Malloc은 인자로 넘겨받은 size를 단위에 맞게 round한 후, find_fit을 통해 최적의 위치를 찾은 경우엔 place로 위치시킨다. 만약 찾지 못했다면 extend_heap을 통해 sbrk를 이용하여 heap의 사이즈를 늘린 후 위치시킨다.

Mm_free(void *bp)

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

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

Free 함수는 bp에 해당하는 block의 header와 footer 정보를 초기화해준 후, coalesce 함수를 통해 앞, 뒤의 free block과 합쳐준다.

Mm_realloc(void *bp, size_t size)

```
void *mm_realloc(void *bp, size_t size)
{
    /*void *oldptr = ptr;
    void *newptr;
    size_t copySize;

    newptr = mm_malloc(size);
    if (newptr == NULL)
      return NULL;
    copySize = *(size_t *)((char *)oldptr - SIZE_T_SIZE);
    if (size < copySize)
      copySize = size;
    memcpy(newptr, oldptr, copySize);
    mm_free(oldptr);
    return newptr;*/

    if(size <= 0) {
        mm_free(bp);
        return NULL;
    }
    if(bp == 0) {
        return mm_malloc(size);
    }
    void *newbp = mm_malloc(size);
    if(newbp == NULL) return NULL;

    size_t oldsize = GET_SIZE(HDRP(bp));
    if(size < oldsize) {
        oldsize = size;
    }
    memcpy(newbp, bp, oldsize);
    void *oldbp = bp;
    mm_free(oldbp);
    return newbp;
}
```

Realloc 함수는 우선 인자로 받은 size가 0보다 작거나 같을 경우 해당 bp의 block을 free한다. 만약 넘겨받은 bp가 유효한 값이 아니라면 malloc과 같은 작업을 수행한다. 이 외엔 새로운 block을 할당하여 해당 block에 이전 block의 메모리의 내용물을 복사한 후, 이전 block을 free한다.

Mm_check()

```
int mm_check()
{
    char* bp = heap_listp;
    int flag = -1;
    while(bp) {
        if(!GET_ALLOC(HDRP(bp))) {
            if(flag == 0) {
                printf("Free blocks escaped colaescing exist\n");
                return 1;
            }
            else
                flag = 0;
        }
        else
            flag = 1;
        bp = NEXT_BLKP(bp);
    }
    return 0;
}
```

Check 함수는 implicit list 방식의 특성을 고려하여, 현재 coalesce 되지 않은 block이 존재하는지, 즉 heap에 false fragmentation이 존재하는지 파악하는 함수이다. 만약 존재한다면 에러 메세지와 함께 1을 return하도록 구현하였다.