System Programming Project 4

담당 교수 : 이영민

이름 : 김정환

학번 : 20211522

1. 개발 목표

- **해당 프로젝트에서 구현할 내용을 간략히 서술.**

이 프로젝트에서는 C 프로그래밍에서 사용하는 dynamic memory allocator를 구현한다. mm.c에서 init, malloc, free, realloc에 대해서 각각에 맞는 기능을 구현하여 올바른 메모리 할당을 수행하도록 코드를 작성한다. Memory 할당과 해제에 있어서 적절한 방법을 사용하여 memory를 효율적으로 사용하고 빠르게 할당할 수 있도록 최적화하는 것이 본 프로젝트의 목표이다.

2. 개발 방법

- **프로젝트 항목을 구현했을 때 사용한 방법**

-사용한 free block의 관리 방식 : exiplicit free list

-사용한 block 탐색 방식 : Best-fit

-Global variable : block에 대한 정보를 저장하는 heap_list, free block을 저장하는 free_list로 두 개를 사용하였다.

-추가한 매크로

#define MINBLOCKSIZE (4 * WSIZE) : explicit free list를 사용하면서 크기를 맞춰주기 위해 추가하였다.

#define SUCC(bp) (*(char **)(bp)) : 다음 free block을 가리키는데 이용하기 위해 추가하였다.

#define PRED(bp) (*(char **)((char *)(bp) + WSIZE)) : 이전 free block을 가리키는데 이용하기 위해 추가하였다.

이외에는 교재에 나오는 매크로들을 이용하여 구현하였다.

-구현한 함수 설명

mm_init(void) : 사용할 list들을 초기화하는 함수이다.

```c
int mm_init(void){
    /* Create the initial empty heap */
    if ((heap_list = mem_sbrk(4*WSIZE)) == (void *)-1)
        return -1;
    PUT(heap_list, 0);                          /* Alignment padding */
    PUT(heap_list + (1*WSIZE), PACK(DSIZE, 1));     /* Prologue header */
    PUT(heap_list + (2*WSIZE), PACK(DSIZE, 1));     /* Prologue footer */
    PUT(heap_list + (3*WSIZE), PACK(0, 1));         /* Epilogue header */
    heap_list += (2*WSIZE);

    free_list = NULL; // initialize free_list

    /* Extend the empty heap with a free block of CHUNKSIZE bytes */
    if (extend_heap(CHUNKSIZE/WSIZE) == NULL)
        return -1;
    return 0;
}
```

처음에 4개의 word 사이즈로 heap_list를 초기화한다. 이 때 맨 앞에는 padding 처리를 해주고, prologue header, footer, 그리고 맨 뒤가 될 epilogue footer로 구성해준다. free list는 NULL로 초기화해준 후에 extend_heap에서 생성된 초기 free block이 추가되는 방식으로 초기화된다.

```c
static void *extend_heap(size_t words){
    char *bp;
    size_t size;

    /* Allocate an even number of words to maintain alignment */
    size = MAX((words % 2 ? (words + 1) : words) * WSIZE, MINBLOCKSIZE);
    if ((long)(bp = mem_sbrk(size)) == -1)
        return NULL;

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

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

extend_heap 함수는 교재에 나오는 것과 동일한 형태로 사용하였다.

void insert_free_block(void* bp) : free block list에 block 삽입하는 함수

void remove_free_block(void *bp) : free block list에서 block 제거하는 함수

```
static void insert_free_block(void *bp){
    SUCC(bp) = free_list;
    PRED(bp) = NULL;
    if (free_list != NULL)
        PRED(free_list) = bp;
    free_list = bp;
}

static void remove_free_block(void *bp) {
    if (PRED(bp))
        SUCC(PRED(bp)) = SUCC(bp);
    else
        free_list = SUCC(bp);
    if (SUCC(bp))
        PRED(SUCC(bp)) = PRED(bp);
}
```

free list에 insert하는 경우에는 list의 제일 앞에 삽입하도록 하는 방식을 이용하여 구현하였다. List의 제거의 경우에는 제거할 pointer를 전달하므로 해당 위치에 대해서 해당 block이 제일 앞인지만 구분하여 제거하도록 구현했다.

void *find_fit(size_t asize) : block을 선택하는 함수

```
static void *find_fit(size_t asize){
    // Best-fit: 가장 낭비가 적은 블록 선택
    void *ptr = free_list; // search위한 initialize
    void *best_fit = NULL; //return할 value
    size_t min_diff = (size_t)-1; // 최대값으로 초기화

    while (ptr != NULL) {
        size_t block_size = GET_SIZE(HDRP(ptr));
        if (block_size >= asize) {
            size_t diff = block_size - asize;
            if (diff < min_diff) {
                min_diff = diff;
                best_fit = ptr;

                // perfect match면 바로 리턴
                if (diff == 0) break;
            }
        }
        ptr = SUCC(ptr); // move to successor
    }

    return best_fit; // NULL이면 no-fit
}
```

find_fit의 경우에는 best-fit 방식을 이용하도록 구현하였다. Free block list를 순회하면서 요청한 size에 딱 맞는 block인 경우에만 곧바로 할당하고 이외에는 전부 순회한 후에 가장 남는 크기가 작은 경우를 돌려준다.

void place(void *bp, size_t asize) : bp에 asize의 크기 이상이 있으면 할당해주는 함수

```
static void place(void *bp, size_t asize){
    size_t csize = GET_SIZE(HDRP(bp));
    remove_free_block(bp); // 가용 리스트에서 제거

    // 블록을 나눌 수 있을 만큼 크다면 분할
    if ((csize - asize) >= MINBLOCKSIZE) {
        // 현재 블록을 asize만큼 할당
        PUT(HDRP(bp), PACK(asize, 1));
        PUT(FTRP(bp), PACK(asize, 1));

        // 남은 부분을 새 free block으로 설정
        void *next_bp = NEXT_BLKP(bp);
        size_t remaining = csize - asize;

        PUT(HDRP(next_bp), PACK(remaining, 0));
        PUT(FTRP(next_bp), PACK(remaining, 0));
        insert_free_block(next_bp);
    } else {
        // 블록이 너무 작아서 분할하지 않고 통째로 할당
        PUT(HDRP(bp), PACK(csize, 1));
        PUT(FTRP(bp), PACK(csize, 1));
    }
}
```

우선 bp는 free block list에서 제거해주고, MINBLOCKSIZE 이상으로 남는 경우에는 나눠서 free block list에 넣어주고 아닌 경우에는 통째로 할당한다.

void *coalesce(void *bp) : bp의 앞 뒤에 free block 존재 시 합치는 함수

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

    if (!prev_alloc) remove_free_block(PREV_BLKP(bp)); // 이전 block이 free block이면 우선 free_list에서 제거
    if (!next_alloc) remove_free_block(NEXT_BLKP(bp)); // 다음 block이 free block이면 우선 free_list에서 제거

    if (prev_alloc && next_alloc) { /* Case 1 */ // prev and next are both allocated
        insert_free_block(bp);
        return bp;
    }
    else if (prev_alloc && !next_alloc) { /* Case 2 */ // only next block is free
        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 */ // only previous block is free
        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 */ // prev and next are both free
        size += GET_SIZE(HDRP(PREV_BLKP(bp))) + GET_SIZE(FTRP(NEXT_BLKP(bp)));
        PUT(HDRP(PREV_BLKP(bp)), PACK(size, 0));
        PUT(FTRP(NEXT_BLKP(bp)), PACK(size, 0));
        bp = PREV_BLKP(bp);
    }

    insert_free_block(bp); // insert free block to free block list

    return bp;
}
```

강의자료에서 볼 수 있었던 것처럼 앞 뒤 block이 free block인지 여부에 따른 4가지 경우로 나눠서 합치는 과정을 수행한다. 우선 이전과 다음 block이 할당되지 않은 경우에는 bp만 free block list에 넣어주고 바로 종료한다. 이 외의 나머지 3개의 case에서는 size에 앞 뒤의 block size를 맞게 넣어주고, 이전 block과도 합쳐야 하는 경우에만 이전 block에 size를 넣어주고 bp를 이전 block으로 옮긴 뒤에 최종적으로 free block list에 넣어준다.

void *mm_malloc(size_t size) : malloc을 수행하는 함수

```c
void *mm_malloc(size_t size){
    size_t asize; /* Adjusted block size */
    size_t extendsize; /* Amount to extend heap if no fit */
    char *bp;
    /* Ignore spurious requests */
    if (size == 0)
        return NULL;

    /* Adjust block size to include overhead and alignment reqs. */
    if (size <= DSIZE)
       asize = MINBLOCKSIZE;
    else
       asize = DSIZE * ((size + (DSIZE) + (DSIZE-1)) / DSIZE);

    if (asize < MINBLOCKSIZE)
        asize = MINBLOCKSIZE;

    /* Search the free list for a fit */
    if ((bp = find_fit(asize)) != NULL) {
       place(bp, asize);
       return bp;
    }

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

교재에 나온 것과 동일한 코드를 사용하였으나 최소 size를 MINBLOCKSIZE에 맞춘 것에만 차이가 존재한다.

void mm_free(void *ptr) : free를 수행하는 함수

```c
void mm_free(void *ptr){
    size_t size = GET_SIZE(HDRP(ptr));
    PUT(HDRP(ptr), PACK(size, 0));
    PUT(FTRP(ptr), PACK(size, 0));
    coalesce(ptr);
}
```

교재와 동일한 코드를 사용하였다.

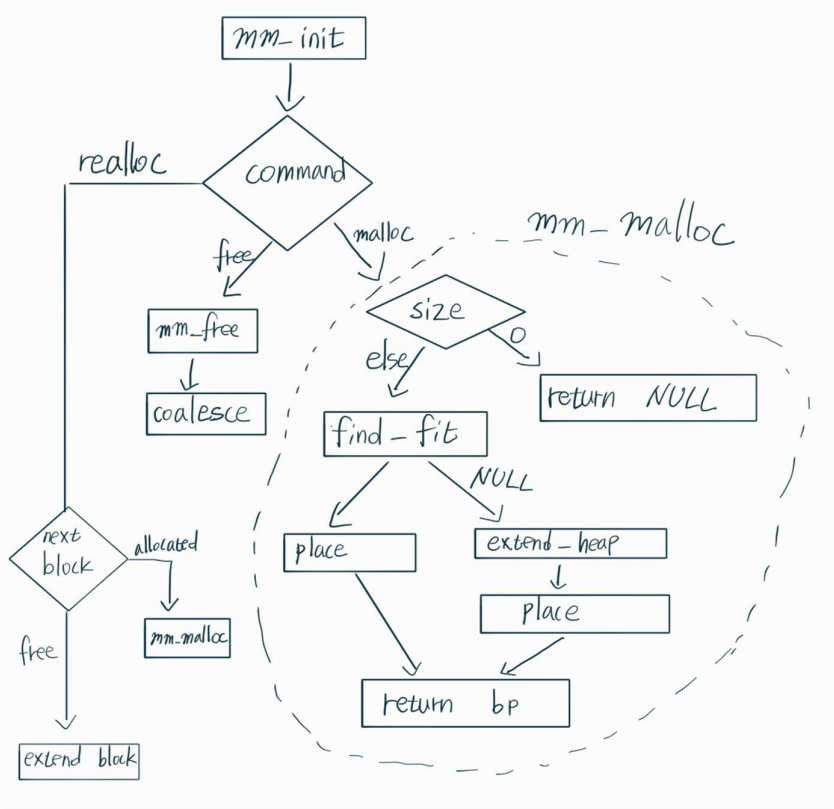
void *mm_realloc(void *ptr, size_t size) : realloc을 수행하는 함수

```c
void *mm_realloc(void *ptr, size_t size){
    if (ptr == NULL) return mm_malloc(size);
    if (size == 0) {
        mm_free(ptr);
        return NULL;
    }

    size_t oldsize = GET_SIZE(HDRP(ptr));
    size_t newsize;

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

    if (newsize <= oldsize) {
        return ptr;  // 기존 블록 크기로 충분히 커버 가능한 case
    }

    void *next_bp = NEXT_BLKP(ptr);
    if (!GET_ALLOC(HDRP(next_bp))) {
        size_t next_size = GET_SIZE(HDRP(next_bp));
        if (oldsize + next_size >= newsize) {
            size_t total_size = oldsize + next_size;
            remove_free_block(next_bp);
            PUT(HDRP(ptr), PACK(total_size, 1));
            PUT(FTRP(ptr), PACK(total_size, 1));
            return ptr;
        }
    }

    void *newptr = mm_malloc(size);
    if (newptr == NULL) // malloc이 안 된 경우
        return NULL;

    size_t copySize = oldsize - DSIZE; // copy할 size
    if (size < copySize)
        copySize = size;

    memcpy(newptr, ptr, copySize);
    mm_free(ptr);
    return newptr;
}
```

ptr이 NULL인 경우에 대해서는 mm_malloc으로 할당해주고, size가 0인 경우에는 free 처리해준다. 크기가 줄어들거나 같은 경우에는 기존의 ptr을 그대로 반환해주고, 늘리는 경우에 대해서 조정이 필요하다. 우선 다음으로 이어지는 block과 합쳤을 때 size에 맞는 block을 만들 수 있는 경우에는 합치는 방식을 이용한다. 아닌 경우에는 mm_malloc으로 할당하고 값들을 memcpy로 복사하고 기존의 ptr을 free 처리하는 방식을 이용한다.

3. Flow Chart

- 프로젝트 항목을 구현한 흐름도



4. 테스트 결과

- 서버에서 mdriver로 테스트한 결과

```
cse20211522@cspro:~/prj4-malloc$ ./mdriver -v
[20211522]::NAME: Junghwan Kim, Email Address: jh0814@sogang.ac.kr
Using default tracefiles in ./tracefiles/
Measuring performance with gettimeofday().

Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   99%    5694  0.000423 13477
 1       yes   99%    5848  0.000413 14146
 2       yes   99%    6648  0.000420 15814
 3       yes  100%    5380  0.000344 15658
 4       yes   66%   14400  0.000268 53812
 5       yes   96%    4800  0.003714  1292
 6       yes   95%    4800  0.003780  1270
 7       yes   55%   12000  0.031022   387
 8       yes   51%   24000  0.092693   259
 9       yes   87%   14401  0.000210 68576
10       yes   67%   14401  0.000122117655
Total          83%  112372  0.133409   842

Perf index = 50 (util) + 40 (thru) = 90/100
cse20211522@cspro:~/prj4-malloc$
```