System Programming Project 3

Dynamic Memory Allocater

담당 교수 : 김영재

이름 : 이제현

학번 : 20170624

Segregated List 방식으로 mymalloc을 구성했다.

교재의 implicit allocater 코드를 기반으로 변경하여 segregated 방식을 만들었다.

먼저 전역변수와 정의는 다음과 같다.

```
/* Address of free block's predecessor and successor entries. */
#define PRED_PTR(ptr) ((char *)(ptr))
#define SUCC_PTR(ptr) ((char *)(ptr) + WSIZE)

/* Address of free block's predecessor and successor on the segregated list */
#define PRED(ptr) (*(char **)(ptr))
#define SUCC(ptr) (*(char **)(SUCC_PTR(ptr)))

/* p가 가리키는 memory에 ptr을 입력. */
#define SET_PTR(p, ptr) (*(unsigned int *)(p) = (unsigned int)(ptr))

void *free_lists[LISTS];

static void *extend_heap(size_t words);
static void insert_node(void *ptr, size_t size);
static void delete_node(void *ptr);
static void place(void *bp, size_t asize);
static void *coalesce(void *bp);
//static void *find_fit(size_t asize);
static void place(void *bp, size_t asize);
```

#define PRED(ptr) (*(char **)(ptr))

#define SUCC(ptr) (*(char **)(SUCC_PTR(ptr)))

seglist의 free block의 앞과 뒤에 있는 block entry의 주소이다.

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

포인터 p가 가리키는 메모리에 ptr을 입력한다. *p = *ptr을 매크로 형식으로 만든 것이다.

void *free_lists[LISTS];

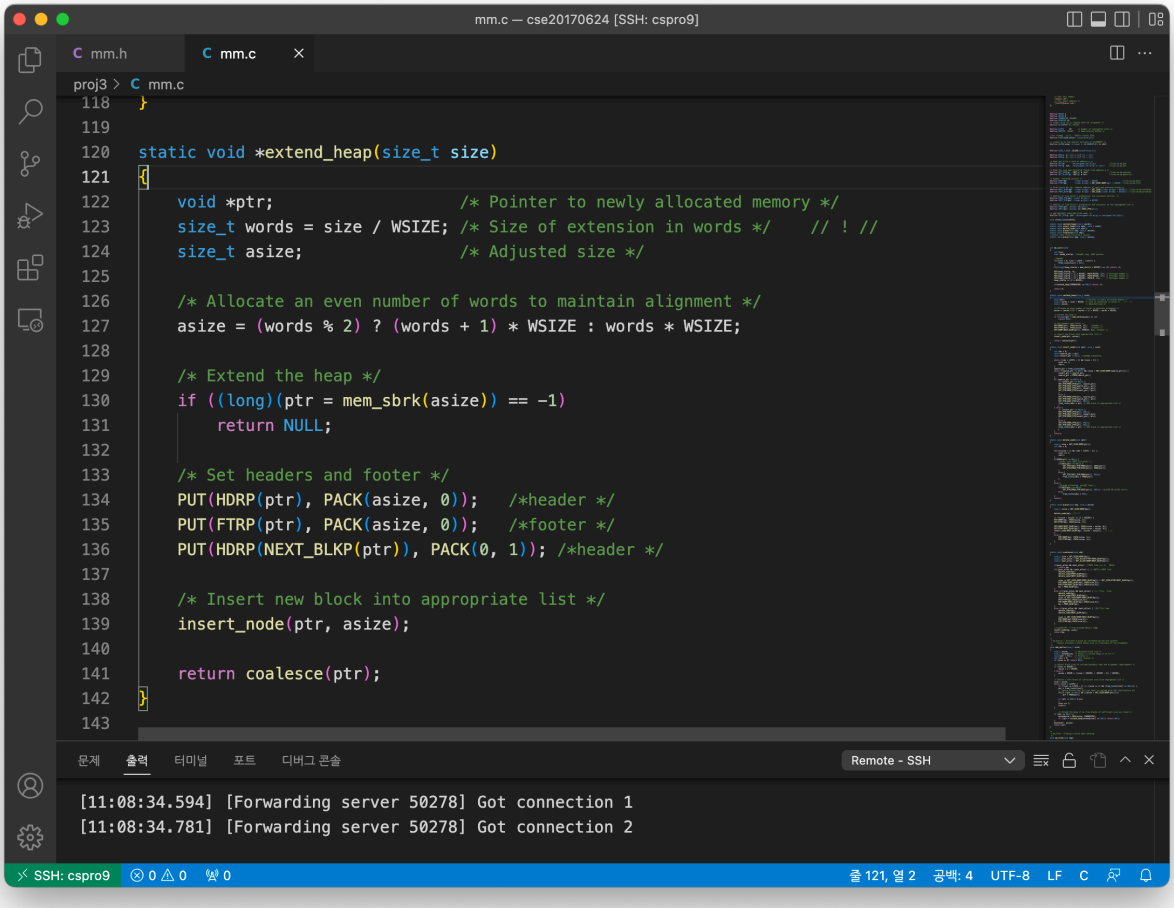
Segregated된 list를 저장할 배열이다. LISTS는 segregate된 list의 최대 개수이다.

```c
int mm_init(void)
{
    int list;
    char *heap_startp; //heap의 시작부분의 pointer

    //초기화
    for(list = 0; list < LISTS ; list++) {
        free_lists[list] = NULL;
    }
    if((long)(heap_startp = mem_sbrk(4 * WSIZE)) == -1) return -1;

    PUT(heap_startp, 0);
    PUT(heap_startp + (1 * WSIZE), PACK(DSIZE, 1)); /* Prologue header */
    PUT(heap_startp + (2 * WSIZE), PACK(DSIZE, 1)); /* Prologue footer */
    PUT(heap_startp + (3 * WSIZE), PACK(0, 1));     /* Epilogue header */
    heap_startp += (2 * WSIZE);

    if(extend_heap(CHUNKSIZE) == NULL) return -1;

    return 0;
```

mm_init

기본적으로 implicit malloc 방식과 동일한 초기화 방식이다.

seglist를 저장할 free_lists를 초기화하고 WSIZE단위로 DSIZE의 크기를 가지는 header, footer, header를 만든다. mem_sbrk는 4 * WSIZE byte만큼 heap을 증가시키는데, 음수가 return되면 증가에 실패했다는 의미로 -1을 return한다.



Extend_heap

free block을 합치는 함수이다.

여기서 word의 크기를 새로 정의해서 반올림할 asize를 만드는데, words = size / WSIZE 와 같이 WSIZE로 나누어야 한다. WSIZE로 나누지 않는 경우 측정한 결과 성능이 2만큼 저하되었다. internal fragmentation이 발생한것 같다. 반면 DSIZE로 나눈 결과 Segfault가 발생했다.

size = (words % 2) ? (words + 1) * WSIZE : words * WSIZE;

size_t size에서 요청받은 size를 2 word(8byte)로 반올림한 뒤에 heap공간을 요청하는 함수이다.

새로 정해진 크기 asize를 free list에 추가한다.

```
static void insert_node(void *ptr, size_t size)
{
    int idx = 0;
    void *search_ptr = ptr;
    void *insert_ptr = NULL; //node가 삽입되는곳.

    while ((idx < LISTS - 1) && (size > 1)) {
        size >>= 1;
        idx++;
    }
    search_ptr = free_lists[idx];
    while ((search_ptr != NULL) && (size > GET_SIZE(HDRP(search_ptr)))) {
        insert_ptr = search_ptr;
        search_ptr = PRED(search_ptr);
    }
    if (search_ptr != NULL) {
        if (insert_ptr != NULL) {
        SET_PTR(PRED_PTR(ptr), search_ptr);
        SET_PTR(SUCC_PTR(search_ptr), ptr);
        SET_PTR(SUCC_PTR(ptr), insert_ptr);
        SET_PTR(PRED_PTR(insert_ptr), ptr);
        }
        else {
        SET_PTR(PRED_PTR(ptr), search_ptr);
        SET_PTR(SUCC_PTR(search_ptr), ptr);
        SET_PTR(SUCC_PTR(ptr), NULL);
        free_lists[idx] = ptr; /* Add block to appropriate list */
        }
    } else {
        if (insert_ptr != NULL) {
        SET_PTR(PRED_PTR(ptr), NULL);
        SET_PTR(SUCC_PTR(ptr), insert_ptr);
        SET_PTR(PRED_PTR(insert_ptr), ptr);
        }
        else {
        SET_PTR(PRED_PTR(ptr), NULL);
        SET_PTR(SUCC_PTR(ptr), NULL);
        free_lists[idx] = ptr; /* Add block to appropriate list */
        }
    }
    return;
}
```

Insert_node

insert_ptr 포인터를 선언해서 node가 삽입되는 곳을 지정할 포인터를 만들었다.

seg list는 index가 2^n형태이므로, while((idx < LISTS - 1) && (size > 1)) 부분에서 size의 bit가 감소하면서 count를 하나씩 세면 count의 총합이 list의 index가 된다.

block을 삽입하기 위해서는 가용 블록이어야만 하기 때문에 블록의 앞뒤를 살펴보면서 free block인지 아닌지 판단하는 과정을 거쳐야 하기 때문에 아래부분의 if문을 구성했다. 앞뒤가 모두 allocated 인 경우, 앞만, 뒤만 allocated 된 경우, 모두 비어있는 경우 네가지를 구분한다. 여기서 SET_PTR을 이용한 삽입 과정은 linked list의 삽입과 같은 알고리즘이다.

Delete_node

delete_node도 insert_node와 동일한 구성을 가진다. seg list의 idx를 구한 뒤, linked list의 삭제와 같은 알고리즘으로 SET_PTR한다. 다만 이번에는 앞과 뒤의 순서가 바뀌어야 하므로 SUCC과 PRED의 순서는 insert와 반대이다.

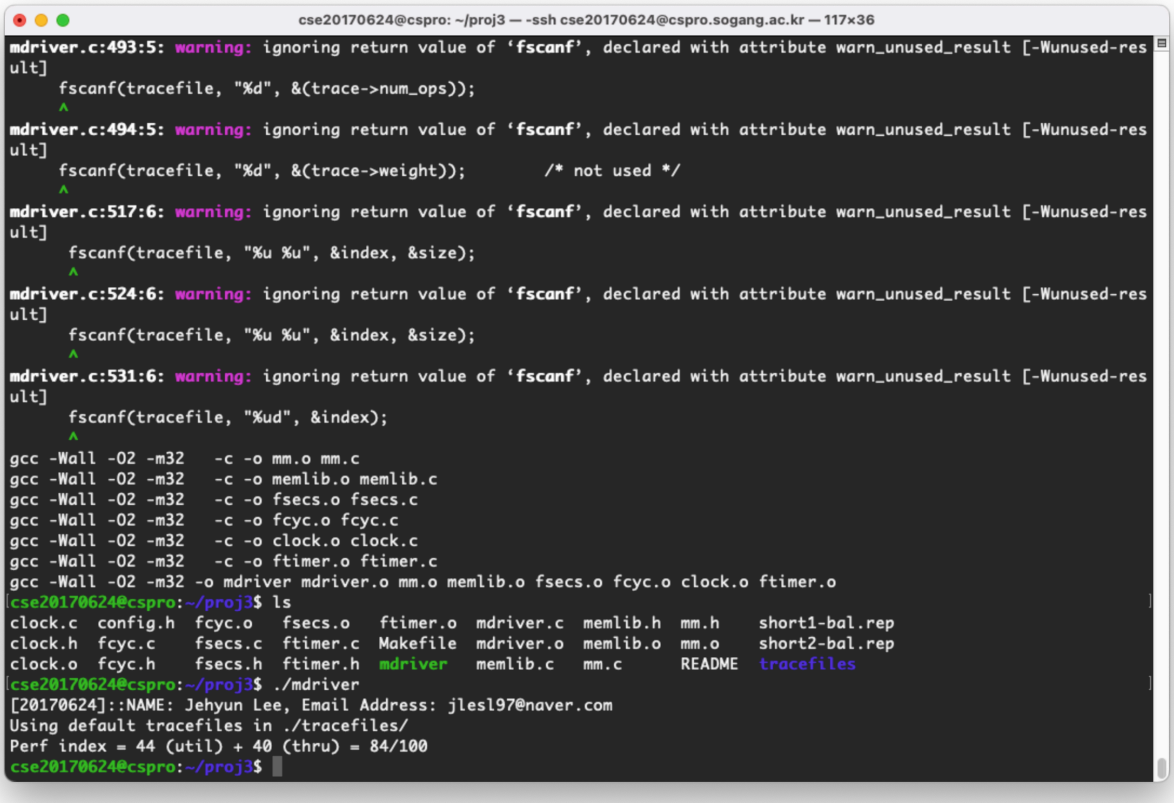
mm_malloc

```
void *mm_malloc(size_t size)
{
    size_t asize;      /* Adjusted block size */
    size_t extendsize; /* Amount to extend heap if no fit */
    void *ptr = NULL;  /* Pointer */
    int list = 0;      /* List counter */
    if (size == 0) return NULL;

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

    /* Select a free block of sufficient size from segregated list */
    size = asize;
    while (list < LISTS) {
        if ((list == LISTS - 1) || ((size <= 1) && (free_lists[list] != NULL))) {
        ptr = free_lists[list];
        // Ignore blocks that are too small or marked with the reallocation bit
        while ((ptr != NULL) && ((asize > GET_SIZE(HDRP(ptr)))))
            ptr = PRED(ptr);

        if (ptr != NULL) break;
        }
        size >>= 1;
        list++;
    }

        /* Extend the heap if no free blocks of sufficient size are found */
    if (ptr == NULL) {
        extendsize = MAX(asize, CHUNKSIZE);
        if ((ptr = extend_heap(extendsize)) == NULL) return NULL;
    }
    place(ptr, asize);
    return ptr;
}
```

먼저 할당하고자 하는 size를 인자로 받고 DSIZE를 기준으로 사이즈를 조정하여 asize라는 변수를 만든다. 다음으로 seglist에서 적당한 크기의 free block을 선택하도록 한다. ptr이 NULL이어야 이미 allocate되지 않았으므로 ptr != NULL이여야 하며, 선택하는 free block의 size가 가용공간 asize보다 작아야 하기 때문에 해당하는 경우에만 ptr에 PRED(ptr), 즉 ptr의 next를 할당함으로 list를 연결한다.

Coalesce

extend_heap에서 block을 합칠때 사용하며 합치는 경우 seglist에서 free block node를 삭제해준다.



./mdriver 결과 44(util) + 40(thru) = 84/100이 나왔다.