System Programming Project 3

담당 교수 : 김영재 교수

이름 : 강 민 석

학번 : 20181589

1. 개발 목표

이 프로젝트에서는 C 프로그램을 위한 동적 스토리지 할당기, 즉 자신의 malloc, free, realloc 함수를 작성할 것이다. 디자인 공간을 창의적으로 탐색하는 것이 좋다.

2. 개발 범위 및 내용

- **개발 범위 및 내용**

동적 스토리지 할당기는 mm.h로 선언되고 mm.c로 정의된 다음 함수로 구성된다.

```
int mm_init(void);

void mm_malloc(size t size);

void mm_free(void *ptr);

void mm_realloc(void *ptr, size t size);
```

주어진 mm.c 파일은 모든 것을 정확하게 구현하지만 순진하지는 않다. 이를 시작점으로 사용하여 다음 의미에 따르도록 이러한 함수들을 수정한다(그리고 가능하면 다른 개인 정적 함수들을 정의한다).

• mm_init: mm_malloc mm_realloc 또는 mm_free를 호출하기 전에, 응용 프로그램(즉, 구현을 평가하기 위해 사용할 추적 구동 드라이버 프로그램)은 mm_init을 호출하여 초기 힙 영역 할당과 같은 필요한 초기화를 수행한다. 초기화를 수행하는 데 문제가 있으면 반환 값이 -1이어야 하며 그렇지 않으면 0이어야 한다.

• mm_malloc: mm_malloc 루틴은 최소 크기 바이트의 할당된 블록 페이로드에 포인터를 반환한다. 할당된 전체 블록은 힙 영역 내에 있어야 하며 할당된 다른 chunk와 겹치지 않아야 한다. 세부적인 구현 목표는 표준 C 라이브러리(libc)에서 제공되는 malloc 버전과 최대한 비슷하게 작동하도록 하는 것이다. libc malloc은 항상 8바이트에 정렬된 페이로드 포인터를 반환하므로, mm_malloc 구현도 마찬가지로 해야 하며 항상 8바이트 정렬 포인터를 반환해야 한다.

• mm_free: mm_free 루틴은 ptr이 가리키는 블록을 해제한다. 그것은 아무것도 돌려주지 않는다. 이 루틴은 전달된 포인터(ptr)가 mm_malloc 또는 mm_realloc에 대한 이전 호출에 의해 반환되고 아직 해제되지 않은 경우에만 작동한다.

• mm_realloc: mm_realloc 루틴은 다음과 같은 제약 조건을 가진 최소 크기 바이트의 할당된 영역으로 포인터를 반환한다.

1. ptr가 NULL이면 호출은 mm malloc(size)와 같다.

2. 크기가 0이면 호출은 mm free(ptr)와 같다.

3. ptr이 NULL이 아닌 경우 mm_malloc 또는 mm_realloc로 이전 호출로 반환되어야 한다. mm realloc 호출은 ptr(이전 블록)이 가리키는 메모리 블록의 크기를 크기로 변경한다.

바이트를 입력하고 새 블록의 주소를 반환한다. 새 블록의 주소가 이전 블록과 같거나, 구현 방법, 이전 블록의 internal fragmentation 양 및 재할당 요청 크기에 따라 다를 수 있다.

새 블록의 내용은 기존 ptr 블록과 동일하며, 최소 구/신 크기까지 동일하다. 다른 모든 것들은 초기화되지 않는다. 예를 들어 이전 블록이 8바이트이고 새 블록이 12바이트인 경우 새 블록의 처음 8바이트는 이전 블록의 처음 8바이트와 동일하고 마지막 4바이트는 초기화되지 않는다. 마찬가지로 이전 블록이 8바이트이고 새 블록이 8바이트인 경우 block은 4바이트이고 새 블록의 내용은 이전 블록의 처음 4바이트와 동일하다.

이것은 해당 libc malloc, realloc 및 free 루틴의 의미와 일치한다.

- **개발 방법**

memlib.c 패키지는 동적 메모리 할당기의 메모리 시스템을 시뮬레이션한다. memlib.c에서 다음 함수를 호출할 수 있다.

• void *mem_sbrk(intr): 힙을 incr 바이트 단위로 확장한다. 여기서 incr은 0이 아닌 양의 정수이며 새로 할당된 힙 영역의 첫 번째 바이트로 일반 포인터를 반환한다. 이는 mem_sbrk가 0이 아닌 양의 정수 인수만을 받아들인다는 점을 제외하면 유닉스 sbrk 함수와 동일하다.

• void *mem_message_lo(void): 힙의 첫 번째 바이트로 일반 포인터를 반환한다.

- void *mem_time_hi(void): 힙의 마지막 바이트로 일반 포인터를 반환한다.
- size_t mem_heapsize(void): 힙의 현재 크기(바이트)를 반환한다.
- size_t mem_pagesize(void): 시스템의 페이지 크기를 바이트 단위로 반환한다(Linux 시스템에서는 4K).

prj3-malloc.tar 배포판의 드라이버 프로그램 mdriver.c는 mm.c 패키지의 정확성, 공간 활용도 및 처리량을 테스트한다. 드라이버 프로그램은 prj3-malloc.tar 배포판에 포함된 추적 파일 집합에 의해 제어된다. 각 추적 파일에는 일련의 할당, 재할당 및 사용 가능한 방향이 포함되어 있다. 이 명령은 드라이버에게 mm_malloc, mm_realloc 및 mm_free 루틴을 순서대로 호출하도록 지시한다.

드라이버 mdriver.c는 다음 명령줄 인수를 사용한다.

• -t <tracedir>: config.h에 정의된 기본 디렉터리 대신 tracedir 디렉터리에서 기본 추적 파일을 찾는다.

• -f <tracefile>: 테스트에 기본 추적 파일 집합 대신 특정 추적 파일 하나를 사용한다.

• -h: 명령줄 인수 요약을 출력한다.

• -l: 학생의 malloc 패키지에 추가하여 libc malloc를 실행하고 측정한다.

• -v: 상세 출력. 압축 테이블의 각 추적 파일에 대한 성능 분석을 출력한다.

• -V: 더 자세한 결과이다. 각 추적 파일이 처리될 때 추가 진단 정보를 인쇄한다. malloc 패키지에 오류가 발생한 추적 파일을 확인하는 데 디버깅 중에 유용하다.

- **개발 시 제약조건**

• mm.c 단위의 인터페이스는 변경하지 않는다.

• 메모리 관리 관련 라이브러리 호출 또는 시스템 호출을 호출하면 안된다. 여기에는 코드에서 malloc, calloc, free, realoc, sbrk, brk 또는 이러한 호출의 변형이 사용되지 않는다.

• mm.c 프로그램에서 배열, 구조체, 트리 또는 목록과 같은 전역 또는 정적 복합 데이터 구조를 정의할 수 없다. 그러나 정수, 부동 소수점 및 포인터와 같은 전역 스칼라 변수를 mm.c 단위로 선언할 수 있다.

• 8바이트 경계에 정렬된 블록을 반환하는 libc malloc 패키지와의 일관성을 위해 할당자는 항상 8바이트 경계에 정렬된 포인터를 반환해야 한다. mdriver가 이 요구사항을 적용한다.

3. 구현 결과

- **<매크로 설명>**

```
34 /* single word (4) or double word (8) alignment */
35 #define ALIGNMENT 8
36
37 /* rounds up to the nearest multiple of ALIGNMENT */
38 #define ALIGN(size) (((size) + (ALIGNMENT-1)) & ~0x7)
39
40 #define WSIZE 4
41 #define DSIZE 8
42 #define CHUNKSIZE (1<<12)
43
44 #define MAX(x, y) ((x) > (y) ? (x) : (y))
45 #define MIN(x, y) ((x) < (y) ? (x) : (y))
46
47 /* Pack a size and allocated bit into a word */
48 #define PACK(size, alloc) ((size) | (alloc))
49
50 /*Read and write a word at address p */
51 #define GET(p) (*(unsigned int *)(p))
52 #define PUT(p, val) (*(unsigned int *)(p) = (val) | GET_TAG(p))
53 #define PUT_NOTAG(p, val) (*(unsigned int *)(p) = (val))
54
55 /* Read the size and allocated fields from address p */
56 #define GET_SIZE(p) (GET(p) & ~0x7)
57 #define GET_ALLOC(p) (GET(p) & 0x1)
58 #define GET_TAG(p) (GET(p) & 0x2)
59
60 /* Given block ptr bp, compute address of its header and footer */
61 #define HDRP(bp) ((char *)(bp) - WSIZE)
62 #define FTRP(bp) ((char *)(bp) + GET_SIZE(HDRP(bp)) - DSIZE)
63
64 /* Given block ptr bp, compute address of next and previous blocks */
65 #define NEXT_BLKP(bp) ((char *)(bp) + GET_SIZE((char *)(bp) - WSIZE))
66 #define PREV_BLKP(bp) ((char *)(bp) - GET_SIZE((char *)(bp) - DSIZE))
67
68 /* Address of free block's predecessor and successor entries */
69 #define PRED_PTR(bp) ((char *)(bp))
70 #define SUCC_PTR(bp) ((char *)(bp) + WSIZE)
71
72 /* Address of free block's predecessor and successor on the seg list */
73 #define PRED(bp) (*(char **)(bp))
74 #define SUCC(bp) (*(char **)(SUCC_PTR(bp)))
75
76 #define SIZE_T_SIZE (ALIGN(sizeof(size_t)))
```

ALIGNMENT: word alignment를 수행할 크기

ALIGN(size): word alignment를 수행
WSIZE, DSIZE: 워드사이즈, 더블사이즈 설정
CHUNKSIZE: 하나의 chunk size를 설정한다.

MAX, MIN: 최댓값, 최솟값 반환

PACK: 한 word 안에 size와 allocation status bit를 묶는다.

GET: 블록 포인터
PUT: 값에 TAG추가
PUT_NOTAG: TAG 없이 값만 추가

GET_SIZE: 사이즈만 가져오기
GET_ALLOC: allocation status bit만 가져오기
GET_TAG: tag bit만 가져오기

- **<전역변수 설명>**

```
78 /* Global Variables */
79 void *seg_list0;     //block size 1
80 void *seg_list1;     //block size 2 ~ 2^2-1
81 void *seg_list2;     //block size 2^2 ~ 2^3-1
82 void *seg_list3;     //block size 2^3 ~ 2^4-1
83 void *seg_list4;     //block size 2^4 ~ 2^5-1
84 void *seg_list5;     //block size 2^5 ~ 2^6-1
85 void *seg_list6;     //block size 2^6 ~ 2^7-1
86 void *seg_list7;     //block size 2^7 ~ 2^8-1
87 void *seg_list8;     //block size 2^8 ~ 2^9-1
88 void *seg_list9;     //block size 2^9 ~ 2^10-1
89 void *seg_list10;    //block size 2^10 ~ 2^11-1
90 void *seg_list11;    //block size 2^11 ~ 2^12-1
91 void *seg_list12;    //block size 2^12 ~ 2^13-1
92 void *seg_list13;    //block size 2^13 ~ 2^14-1
93 void *seg_list14;    //block size 2^14 ~ 2^15-1
94 void *seg_list15;    //block size 2^15 ~ 2^16-1
95 void *seg_list16;    //block size 2^16 ~ 2^17-1
96 void *seg_list17;    //block size 2^17 ~ 2^18-1
97 void *seg_list18;    //block size 2^18 ~ 2^19-1
98 void *seg_list19;    //block size 2^19 ~ 2^20-1
99 static char *heap_listp = 0;
```

- 각 block size별로 segregated list 포인터를 전역변수로 선언하였다. Seg_list 포인터는 list의 마지막 block을 포인트하고 있다.
- Heap_listp는 heap의 list를 포인트하고 있다.

- **<함수 설명>**
- Static void insert_node(void *bp, size_t size)
  - 알맞은 seglist를 찾아서 block size가 오름차순이 되도록 삽입한다. Searchp와 insertp 변수를 사용해서 list를 순회하면서 알맞은 위치를 찾는다. 이후에 삽입하려는 위치가 (1) list의 중간인 경우, (2) list의 마지막인 경우, (3) 맨 앞인 경우, (4) 빈 list에 삽입하는 경우로 나눈다. 이후에 seg_list 포인터를 업데이트한다.

- Static void delete_node(void *bp)
  - 삭제하려는 node가 있는 seglist를 찾고, node를 list에서 제거한다. 이후에 seg_list 포인터 업데이트가 필요한 경우에는 업데이트한다. 마찬가지로 제거하려는 node(bp)가 (1) list의 중간인 경우, (2) list의 마지막인 경우, (3) 맨 앞인 경우, (4) list에 있는 유일한 block인 경우로 나눠서 수행한다.
- Static void *coalesce(void *bp)
  - Bp의 앞뒤 block이 allocated/free 여부를 이용해서 coalesce를 수행한다. Case1, Case2, Case3, Case4 네가지 경우에 대해 각각 수행한다.
- Static void *extend_heap(size_t words)
  - mem_sbrk() 함수를 이용해서 heap 영역의 크기를 늘리고, free block의 포인터를 반환한다.
- Static void *place(void *bp, size_t asize)
  - malloc하려는 bp block을 해당 크기의 free block seglist에서 제거하고, splitting이 필요하다면 수행한다. Split 이후에 남은 나머지 block은 그 크기에 맞는 새로운 seglist에 추가한다.
- Int mm_init(void)
  - seglist를 모두 NULL로 초기화하고 mem_sbrk()를 통해 처음에 empty heap을 생성한다. 이후에 heap_list에 prologue header, prologue footer, 그리고 epilogue header를 추가한다.
- Void *mm_malloc(size_t size)
  - heap_listp가 NULL이면 mm_init()을 다시 수행한다. 이후 header/footer과 alignment padding을 위해 block size를 부분 수정한다. 이후에 malloc하려는 크기에 맞는 list를 찾는다. 만약에 찾았다면(성공했다면) ptr=place(ptr, asize);를 통해서 (필요 시) splitting을 수행한다. 만약에 fit을 찾지 못했다면, extend_heap을 통해서 힙 영역의 크기를 늘리고 새로운 빈 블록을 받아온다.

- Void mm_free(void *ptr)
  - free하려는 block의 header와 footer에 allocation status tag를 free로 바꾸고 free list에 추가한다. coalesce도 수행한다.
- Void *mm_realloc(void *ptr, size_t size)
  - 만약에 realloc size가 0인 경우에는 mm_free를 수행한다. 그리고 mm_malloc()과 똑같이 block size를 부분 수정한다. realloc을 수행하려는 block의 다음 block이 free block이거나 epilogue block인 경우에는 현재 block과 다음 block의 합친 크기가 새로 realloc하려는 size보다 작은지를 확인한다. 만약 작다면 CHUNKSIZE만큼 extend_heap을 수행해야 한다. 하지만 다음 block이 free block이거나 epilogue block이 아닌 경우에는 새로운 크기로 아예 새로 mm_malloc을 수행해 주어야 한다. 그리고 반드시 oldptr는 mm_free()해주어야 한다.
- **<출력 결과>**

```
cse20181589@cspro:~/prj3-malloc$ ./mdriver
[20181589]::NAME: Minseok Kang, Email Address: richkang715@gmail.com
Using default tracefiles in ./tracefiles/
Perf index = 57 (util) + 40 (thru) = 97/100
```