System Programming Project 4

담당 교수 : 이영민 교수님

이름 : 김지섭

학번 : 20201572

1. 개발 목표

런타임에 크기가 결정되는 데이터를 위해 libc에서 제공되는 malloc, realloc, free와 같이 heap에 dynamic memory allocation을 하는 함수를 구현한다.

2. 개발 범위 및 내용

A. 개발 범위

```
int mm_init(void)
void *mm_malloc(size_t size)
void *mm_realloc(void *ptr, size_t size)
void mm_free(void *bp)
```

위의 함수들을 구현하여 heap에 메모리 동적 할당을 효율적으로 하도록 구현하였다.

B. 개발 내용

- mm_init

기타 함수들을 호출하기 전에 mem_sbrk를 호출하여 초기 heap 영역을 할당한다.

- mm_malloc

인자로 받은 크기에 적합한 free block을 찾아 영역을 할당한 후, 해당 영역의 포인터를 반환한다. 이때 포인터는 항상 word-aligned 상태이다.

- mm_realloc

인자로 받은 포인터가 null 일 경우, malloc 과 동일한 작동을 하며, 그렇지 않을 경우, 새로운 크기의 메모리 영역을 할당 후 기존 영역의 데이터를 복사한다.

- mm_free

인자로 받은 포인터의 메모리 영역을 할당 해제 후, 필요 시 접해 있는 다른 free block 과 합쳐, fragmentation 이 발생하지 않도록 한다.

C. 개발 방법

```
static void *heap_listp;
```

할당되지 않은 free block들을 linked list로 관리하기 위해 선언한 전역 변수 포인터로, 리스트의 첫 번째 블록의 주소값을 저장한다.

```
static char *init_heap_space(void);
```

mem_sbrk를 호출하여 heap 영역을 4 * WSIZE만큼 초기화한다.

```
static void create_heap(char *heap_s);
```

힙 포인터를 인자로 받아 프롤로그, 에필로그를 초기화하여 heap 영역을 이후 함수들이 참조하고 관리할 수 있도록 한다.

```
static void *extend_heap(size_t words);
```

힙 영역에 새로운 free block을 추가하여 확장하며, 새로 할당할 크기를 인자로 받고, 할당된 주소를 반환한다.

```
static char *extend_heap_if_needed(char *bp, size_t asize, int
*left);
```

주어진 블록 포인터와 크기를 인자로 받아 남은 영역의 크기 및 현재 블록의 크기를 통하여 필요할 경우 extend_heap을 호출한다.

```
static char *fit_block(size_t asize);
```

크기에 맞는 free block을 반복문을 통해 찾고 반환한다.

```
static void *place(void *bp, size_t asize);
```

인자로 받은 크기만큼을 free block에 할당 후, 남은 공간이 block의 최소 크기보다 작을 경우 분리하여 리스트에 추가한다. 리스트에 추가 시에는 크기가 클 경우 탐색이 비효율적으로 진행되므로 뒤에 연결하도록 한다.

```
static void *coalesce(void *bp);
```

현재 블록에 대해 물리적으로 이웃하는 빈 블록들 (리스트가 아닌 실제 주소 상 양 옆의 블록들) 에 대해 4가지 경우로 나누어 현재 블록과 병합한다.

```
static void insert(size_t size, char *ptr);
```

Doubly linked list 형태로 연결된 free block들을 탐색하며 크기가 적절한 영역에 인자로 받은 포인터가 가리키는 블록을 삽입 후, 이에 맞게 리스트를 수정한다.

```
static void delete(char *ptr);
```

인자로 받은 포인터가 가리키는 블록을 리스트에서 제거 후, 이에 맞게 리스트를 수정한다.

```
int mm_init(void)
```

앞서 구현한 init_heap_space, create_heap을 호출하여 heap 영역을 초기화한다.

```
void *mm_malloc(size_t size)
```

인자로 받은 크기만큼의 heap 영역을 찾아 할당한다. 이는 fit_block으로 크기에 알맞은 블록을 찾고, extend_heap_if_needed로 필요 시 힙 영역을 확장한다.

```
void *mm_realloc(void *ptr, size_t size)
```

인자로 받은 포인터가 가리키는 메모리 블록의 크기를 수정한다. 이때 수정한 크기가 원래 크기보다 클 경우에만 재할당을 진행하며, 기존 영역은 할당 해제를 한다. 인자로 받은 포인터가 NULL일 경우, mm_malloc과 동일하게 작동하며, 인자로 받은 크기가 0일 경우, mm_free와 동일하게 작동한다.

```
void mm_free(void *bp)
```

인자로 받은 포인터가 가리키는 메모리 블록을 할당 해제하고 free block list에 해당 블록을 다시 추가한다.

연결 리스트에서의 free block은 아래와 같은 구조를 가지고 있다.

Header (4bytes)	PREV_FREEP(4bytes)	NEXT_FREEP(4bytes)	Footer(4bytes)

Header와 Footer는 각 블록의 크기인 SIZE와 할당 여부인 ALLOC (1bit)를 OR 연산을 통해 합쳐서 저장하며, payload가 들어갈 공간에는 리스트의 다른 블록들의 주소를 저장하여 공간을 효율적으로 사용한다.

#define PACK(size, alloc), GET_SIZE(p), GET_ALLOC(p)

Header와 footer에 저장되는 정보를 생성 및 추출하기 위해 선언한 매크로로, word-aligned 된 주소에서는 하위 비트가 항상 0으로 고정되기 때문에 할당된 경우 1, 아닌 경우 0을 하위 비트에 저장하여 공간을 아낀다.

#define HDRP(bp), FTRP(bp)

블록 포인터 bp는 항상 payload의 주소를 가지고 있으므로, header는 4 byte를 뺀 주소, footer는 블록의 크기에서 8 byte를 뺀 주소에 위치한다.

#define NEXT_BLKP(bp), PREV_BLKP(bp)

물리적으로 이웃하는 블록들의 주소는 블록의 크기를 바탕으로 계산이 된다. 따라서 이웃 블록들의 header, footer 정보를 활용하여 계산한다.

#define NEXT_FREEP(ptr), PREV_FREEP(ptr)

연결 리스트를 탐색하기 위해 사용하며, payload 위치에 저장된 포인터 값을 dereference하여 앞 뒤의 free block 포인터의 주소를 얻는다.

3. 구현 결과

명세서에 나온 공식대로 제공된 mdriver 프로그램을 통해 성능 평가를 진행하였다.

```
cse20201572@cspro:~/prj4/20201572$ ./mdriver -v
[20201572]::NAME: JiSeop Kim, Email Address: smartflame@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  0.000745  7640
 1       yes   99%    5848  0.000417 14021
 2       yes   99%    6648  0.001032  6441
 3       yes   99%    5380  0.000523 10287
 4       yes   99%   14400  0.000289 49793
 5       yes   95%    4800  0.011534   416
 6       yes   95%    4800  0.010693   449
 7       yes   61%   12000  0.075461   159
 8       yes   88%   24000  0.021022  1142
 9       yes   99%   14401  0.000134107631
10       yes   98%   14401  0.000129112070
Total          94%  112372  0.121978   921

Perf index = 56 (util) + 40 (thru) = 96/100
cse20201572@cspro:~/prj4/20201572$
```

프로젝트의 구조도