M1522.000800 System Programming
Fall 2018

System Programming MemoryLab Report

Park seungil
2016-10760

1. <lab> Memory lab

이번 memory lab의 목표는 dynamic storage allocator을 C 프로그램을 이용해 구현하는 것이 목표이다. Malloc, init, free, exit, realloc을 자가 구현해야하며, 이 때 memory allocate 방식 implicit, explicit, segregated 등의 방법 중 space utilization과 time utilization을 고려해 최적의 allocator을 구현해야 한다.

2. Implementation

전체적인 구현방식은 segregated list를 이용한 방식을 이용했다. Segregated list는 모든 가능한 블록 크기를 size class별로 나누어 비슷한 크기를 가진 블록들을 동일 free 클래스의 집합으로 분리하는 것이다. 이번 구현에서는 2의 제곱씩 크기를 증가시켜 클래스를 비교하였다. 만약 이 segregated list에서 원하는 크기의 free을 찾지 못한다면, extend heap과정을 진행한다. 이번 과제에서는 heap의 20개의 WSIZE의 포인터를 이용하여 각각의 class 의 head pointer을 저장해 놓는다. Free가 발생하면 해당하는 class을 찾고 free 된 payload개수에 따라 오름차순으로 저장하게 된다. Free가 될 상황에서 coalesce을 항상 체크하며, segregated list을 이용함으로써 원하는 size의 free block을 찾을 때 size을 이용하여 찾는 시간을 최대한 단축시킬 수 있다.

2.1 mm_init

Mm_init은 dynamic memory allocator을 초기화하기 위한 작업으로 크게 다음과 같이 진행된다. 1.우선 base구조 작업을 위해 24개의 포인터를 heap에 mem_sBRK 을 이용하여 할당받고, 2. 앞 20개의 포인터에 segregate list class을 위한 pointer을 초기화시킨다. 3. 나머지 4개의 포인터에는 각각 prologue 의 header, footer, epilogue의 header을 설정한다.

```
1.heap_list_pointer = mem_sbrk(24*WSIZE)
2. for(i = 1; i <= LIST_MAX; i++)
    *(char **)(heap_list_pointer + i*WSIZE) = NULL;
3. PUT(heap_list_pointer + (21*WSIZE), PACK(DSIZE, 1));
```

2.2 mm_malloc

Mm_malloc은 알맞은 free block을 찾고, allocate시키는 작업으로 크게 다음과 같이 진행된다. 1. Alloc_size을 구현방식에 맡게 정해준다. 2. Segregate list에서 맞는 seg free block을 찾고, segregate

list에서 그 블락을 지우고, allocate작업을 진행한다. 3. 만약 segregate list에서 맞는 seg free block이 없으면 extend heap 작업을 진행하고 똑 같은 작업을 진행한다.

```
if(size <= DSIZE)
   alloc_size = 2*DSIZE;
else
   alloc_size = DSIZE * ((size + (DSIZE) + (DSIZE-1)) / DSIZE);
if((bp = find_seg(alloc_size)) != NULL){
   segregate_delete(bp);
   allocate(bp, alloc_size);
   return bp;
}
extendsize = MAX(alloc_size, CHUNKSIZE);
if((bp = heap_extend(extendsize / WSIZE)) == NULL){
   return NULL;
}
segregate_delete(bp);
allocate(bp, alloc_size);
```

2.3 mm_free

Mm_free는 주어진 포인터를 free시키는 작업을 진행한다. Footer와 header을 설정하고 coalesce을 호출한다.

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

2.4 mm_exit

Mm_exit는 heap_list_pointer로부터 모든 alloc 된 block을 free시킨다.

```
char *p = NEXT_BLKP(heap_list_pointer);
while(GET_SIZE(HDRP(p)) != 0){
   if(GET_ALLOC(HDRP(p))){
      mm_free(p);
   }
   p = NEXT_BLKP(p);
}
```

2.5 mm_check

Mm_check는 dynamic memory allocator을 구현하면서 발생할 수 있는, 그리고 실제로 발생할 수 있는 다양한 error들을 잡을 수 있는 부분이다. 이번 과제에서는 coalescing되지 않은 free block의 여부를 확인하는 부분과 seg list 에 들어가 있는 seg가 실제 free된 valid block인지 확인하는 부분을 코드로 첨가했다.

2.6 mm_realloc

Mm_realloc은 ptr, size두가지 input을 받는데 만약 ptr이 null이면 mm_malloc(size)과 같은 size가 0이면 mm_free(ptr)과 같은 기능을 하고, 둘다 아니라면 현재 ptr을 주어진 size에 맞게 resize하는데 만약 resize할 수 없으면 다른 주소에 new block을 할당한다. 안에 저장되어있던 내용은 정확히 보존되어야한다.

주어진 내용을 구현하기 위해 다양한 function들을 선언했다.

```
static void *heap_extend(size_t words);
static void *coalesce(void *bp);
static void allocate(char *bp, size_t size);
static void *find_seg(size_t alloc_size);
static void segregate_insert(char *bp);
static void *segregate_delete(void *bp);
```

2-1-1. heap-extend

Extend heap when there is no free space to malloc. Allocate have to be even, use mem_sbrk and set free block header and footer and call coalesce

2-1-2. coalesce
If block is freed, check there is front or next block which is free. If front or next block is free coalesce it and insert it in the segregated list.

2-1-3. allocate
After find appropriate block or extend heap and find appropriate block delete block from segregated list, and now allocate new block.

2-1-4. find-seg
Find appropriate free block from segregated list. find fast use size

2-1-5. segregate_insert
insert free block to segregate list after find appropriate segregate class insert is sort by size.

2-1-6. segregate_delete
delete segregate node from segregate list before alloc

3. Conclusion

이번 memory lab을 진행하면서 많은 것을 배울 수 있었다. malloc, free의 정확한 구현방식에 대해서 공부했고, malloc과 free의 과정에서 주의해야할 점에 대해서 인지할 수 있었다. heap위에서 어떠한 구조로 data을 저장하며 효율적으로 memory 과정을 진행할 수 있을까에 대해 많이 고민하면서, implicit list, explicit list, segregated list 등 다양한 구현 방식을 함께 공부했다. 처음 프로젝트를 진행할 때 explicit list로 구현을 진행했다. 구현을

진행하면서 free전체를 따라 가는 first fit, 사용가능한 공간들 중에서 가장 작은 것을 택하는 best-fit, 사용가능한 공간들 중에서 가장 큰 것을 선택하는 worst-fit등 다양한 전략들에 대해서 공부할 수 있었다. 그리고 explicit list로 구현하는 것보다 segregate list로 구현하는 것이. 블록당 오버헤드가 거의 없고, 블록 할당 반환이 모두 빠른 상수시간연산에 이루어지며 할당된 블록들은 헤더와 풋터가 필요 없다는 점에서 큰 장점이 있다는 것을 알 수 있었다. Segregate list로 구현할 때 global variable이 아니라 어떻게 segregate class의 pointer을 저장해야하나 고민을 많이 했던 것 같다.

Tips:
- do not copy-paste screenshots of your code. Format it properly as text
- length: do not exceed 5 pages. There is no lower limit, but one page is probably not detailed enough.