

시스템프로그래밍 프로젝트3 보고서

20181619 김현서

1. design of allocator

기본적으로 예제 코드로 주어진 implicit free list의 구현 코드를 기반으로 각 block과 list의 구조를 explicit list로 바꾼 후, segregated free list를 구현하는 것이 목표였으나, 역량과 시간의 부족으로 implicit list를 구현하는 데에 그쳤다. 적절한 free block을 찾기 위해서는 next_fit 방식을 사용하였다.

mm_init() : malloc package에 필요한 resource들을 init할 때 사용하는 함수이다. mm의 malloc 함수들로 메모리를 할당받기 이전, 가장 처음에 호출된다. 먼저 null로 설정되어있던 전역변수 heap_list 포인터에 mem_sbrk() 함수를 이용하여 heap의 크기를 WORD size의 4배만큼 늘린다. 맨 첫 word는 alignment를 맞추기 위한 padding, 두 번째는 prologue header, 세 번째는 prologue footer, 마지막 word는 epilog header로 사용된다. next fit을 사용할 것이므로 현재까지 탐색했던 block을 저장해줄 전역변수 rover에 힙의 시작점을 저장한다.

mm_malloc() : 기본적인 구성은 implicit list를 find_fit 함수를 이용하여 next fit 방식으로 적절한 free 상태의 block을 찾아 return하고, 만약 적절한 free block이 없다면 mem_sbrk() 함수를 이용하여 heap 자체를 늘려 추가한 공간을 return한다.

코드를 한 줄씩 살펴보면, heap_size가 아직 0이라면, mm_init이 한번도 수행되지 않은 상태에서 mm_malloc이 수행된 것을 의미하며, heap 역시 초기화되지 않아 공간을 가지고 있지 않은 상황이다. 따라서 mm_init을 호출한다. 만약 인자로 전달받은 size가 0이라면 block을 할당할 수 없으므로 NULL을 return한다. size가 DSIZE (double word)보다 작다면 2*DSIZE로, 그렇지 않다면 정해진 ALIGN 사이즈만큼 할당하고, 해당 size만큼의 free block을 찾기 위해 find_fit 함수를 호출하고, return으로 받은 free block의 포인터를 return한다.

mm_free(): 이미 할당되어있는 block을 free block으로 바꾸는 함수이다. bp가 null이라면, 할당되어 있지 않은 block에 대한 mm_free()의 호출이고, invalid하므로 그대로 return한다. 만약 heap_list가 0이라면 애초에 mm_init()도 호출된 적 없는 상태이므로 mm_init()을 호출하고 return해준다. 만약 두 경우 모두에 해당되지 않는 정상적인 free 호출이라면 인자로 전달받은 free시킬 해당 block의 헤더와 푸터 모두 allocated를 0으로 초기화한다.

mm_realloc(): 이미 할당된 block의 사이즈를 변경하여 다시 할당하고 싶을 때 할당하는 함수이다. 전달받은 인자인 size가 0이라면 할당된 block의 크기를 0으로 만들어주면 되므로 mm_free()를 호출하여 해당 block을 free시킨다. 만약 인자로 null인 ptr을 전달받았다면 새롭게 mm_malloc 함수를 이용하여 size만큼의 공간을 할당하고 그 ptr을 return한다. 둘다 해당하지 않는다면 size만큼의 block을 mm_malloc 함수를 이용하여 새롭게 할당받는다. memcpy 함수를 이용하여 원래 ptr (oldptr)에 있던 payload를 newptr로 복사해준다. oldptr을 free시키고 size를 변경하여 새로 할당한 newptr을 return한다.

2. description of subroutines, structs, global variables

- global variable

static char* heap_listp : allocator에서 사용할 heap의 처음 주소를 저장하기 위한 전역 변수 char 포인터 변수이다. mm_init을 호출할 때 mem_sbrk() 함수를 통해 공간을 할당받으며, prologue header로 기능한다.

static char* rover : find_fit에서 현재까지 탐색했던 free block의 포인터를 저장하기 위해 사용하는 전역변수이다. 적절한 free block을 탐색할 때 무조건 free list의 맨 처음부터 탐색하는 first fit과 다르게 next fit은 현재까지 탐색한 지점을 저장해놓고 다음 수행 시 그 지점부터 탐색을 수행하므로 throughput이 올라가는 효과를 얻을 수 있다.

- subroutines

extend_heap() : mm_init으로 처음 초기화할 때나, implicit free list에 적절한 free block이 없을 경우 heap 영역을 늘려주는 함수이다.

확장을 원하는 word의 수가 홀수라면 하나 늘린 후 짝수로 맞춰준 후 WSIZE(4)를 곱하면 alignment (8)를 맞출 수 있다. mem_sbrk() 함수를 호출하여 heap의 크기를 확장하고, 확장한 block에 대한 header와 footer에 size를 채우고 allocated tag는 0으로 설정한다. 확장한 영역의 바로 앞에 있는 block이 free 상태일 수도 있으므로 coalesce를 수행하며 마무리 짓는다.

find_fit() : next fit 방식을 채택하였다. 현재까지 탐색했던 block의 pointer를 전역 변수인 rover에 저장해놓고, 다음 호출 시 해당 block에서부터 free block이 있는지 확인하며 탐색하는 방식이다. 반복문을 이용하여 rover부터 탐색하며 1. free되어있는 block && 2. 원하는 size보다 큰 size를 가지고 있는 block 두 조건을 모

두 만족하는 경우 바로 해당 block에 대한 포인터를 return한다. 만약 list의 끝까지 찾지 못했다면 heap_listp로 다시 돌아가서 처음부터 적절한 block을 탐색한다. 끝까지 찾지 못하였다면 null을 return한다. 그런 경우 mm_malloc 함수에서 mem_sbrk() 함수를 호출하여 heap의 공간을 새롭게 할당받을 것이다.

coalesce() : 어떤 block이 free될 때, 만약 앞이나 뒤의 block 역시 free block이라면, 아무 처리를 해주지 않고 free해줄 경우 실제로 free block의 공간이 충분함에도 다음 번에 검색할 때 남는 공간을 인식하지 못하는 false fragment 현상이 발생하게 된다. 따라서 앞뒤로 붙어 있는 free block을 하나의 block으로 합쳐주는 coalesce가 필요하다.

coalesce를 위해서는 현재 ptr가 가리키는 block의 앞, 뒤 block이 free인지, allocated 상태인지를 확인해야 한다. 다음과 같은 4가지 경우가 존재한다.

1. previous block, next block 모두 allocated

free 한 block이 이어지지 않으므로 coalesce가 필요없는 상황이다. 따라서 그냥 return 해주면 된다.

2. previous block : allocated && next block : free

앞의 block은 allocated되어 있으므로 합병할 수 없고, 현재 block과 뒤의 block만 합병하면 된다. 두 block의 size를 합쳐 계산한 후, 현재 block의 header와 다음 block의 footer 위치에 새로운 size를 저장한다.

3. previous block : free && next block : allocated

2의 경우와 반대의 경우로, 앞의 block의 헤더와 현재 block의 footer에 새로운 size를 저장하고, tag 역시 free로 모두 변경해준다.

4. previous block, next block 모두 free

앞과 뒤의 block 모두 free 상태이므로, 앞, 뒤의 block의 size와 현재 block의 size를 합친 새로운 size를 앞의 block의 header와 뒤의 block의 footer에 size로 저장한다.

중요한 것은 bp를 return할 때 병합된 block들 중 뒤의 block을 point하는 포인터를 return하면 안된다. 따라서 previous block이 free인 3,4 경우에는 if문을 빠져나오기 전에 bp를 bp의 previous pointer로 바꿔주는 구문이 필요하다.

place() : 어떤 block이 allocated 될 때 남은 공간이 free block으로 처리할 수 있을만큼의 size(여기서는 2* doubleword size이다)를 가지고 있는지 판별하고, 그렇다면 새로운 block의 header와 footer를 0으로 설정하여 free block임을 기록하고, 남은 공간이 충분치 않으면 그냥 전체 block의 header와 footer를 allocated 되었다고 기록한다. (남은 공간은 internal fragment라고 볼 수 있다.)

3. mm_check 함수

프로젝트 설명서에서 주어진 기능들은 다음과 같다.

- Is every block in the free list marked as free?
- Are there any contiguous free blocks that somehow escaped coalescing?
- Is every free block actually in the free list?
- Do the pointers in the free list point to valid free blocks?
- Do any allocated blocks overlap?
- Do the pointers in a heap block point to valid heap addresses?

여기에서 첫 번째부터 세 번째 질문은 애초에 implicit free list를 사용하므로 free block 뿐만 아니라 모든 block이 list에 저장되어서 확인하는 것이 크게 의미는 없다. 따라서 다섯번째, 여섯번째 질문을 확인할 수 있는, 구현되어있는 checkheap 함수를 참고하여 사용하였다.

checkheap 함수에서는 먼저 prologue header에 제대로 된 size 값이 저장되어 있는지를 확인한 후, for문으로 탐색하며 모든 block에 대해 checkblock()을 호출하여 해당 block이 valid한 헤더와 푸터를 가지고 있는지를 확인한다. 마지막으로 epilogue block에 대해서도 확인한다.

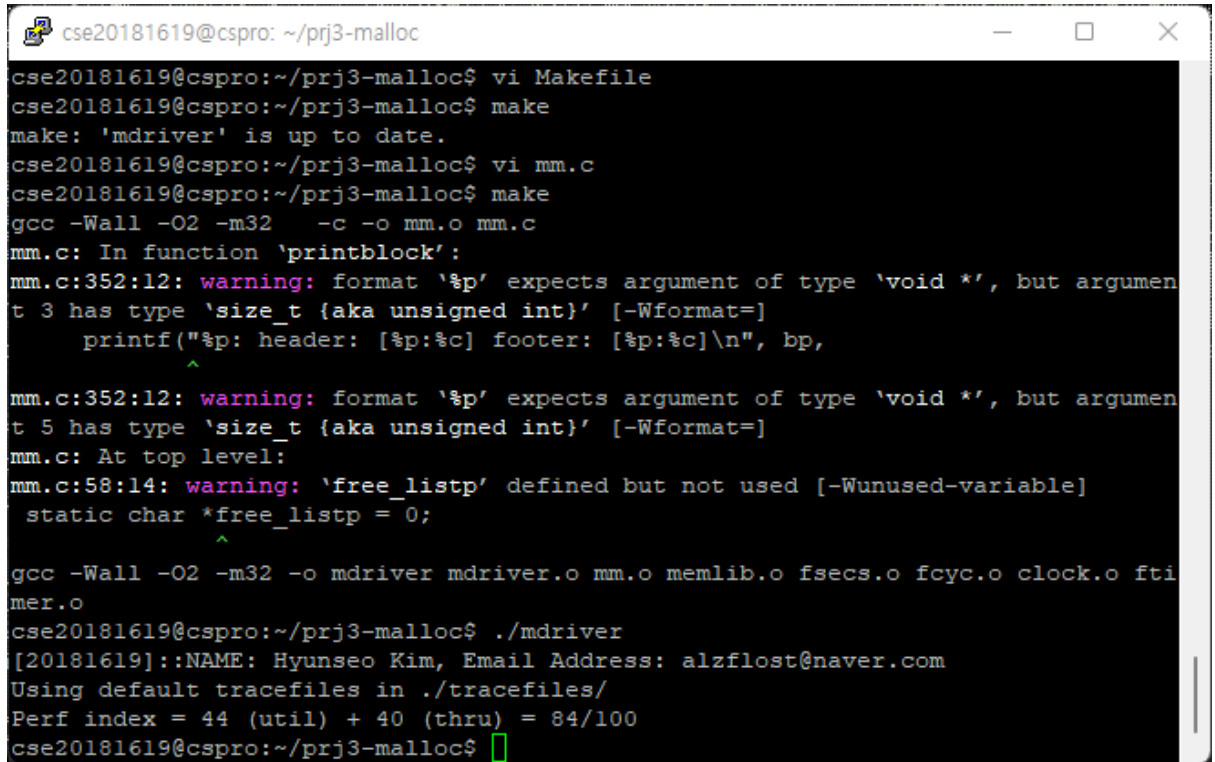
checkheap 함수 내부에서 사용되는 함수들은 다음과 같다.

printblock() : 오류가 있는 block을 출력하기 위한 함수로, header와 footer의 allocated 여부와 size를 출력한다.

checkblock() : header의 size를 읽어서 만약 8로 나누어 떨어지지 않는다면 (word-alignment를 지키지 않는다면) 위의 마지막 질문에 해당하는 비정상적인 경우이므로 오류메세지를 출력한다. 만약 header의 내용과 footer의 내용이 다르다면 위의 5번째 질문에 해당하는 allocated된 block이 overlap된 경우이므로 오

류 메시지를 출력한다.

4. 구현 결과



```
cse20181619@cspro: ~/prj3-malloc
cse20181619@cspro:~/prj3-malloc$ vi Makefile
cse20181619@cspro:~/prj3-malloc$ make
make: 'mdriver' is up to date.
cse20181619@cspro:~/prj3-malloc$ vi mm.c
cse20181619@cspro:~/prj3-malloc$ make
gcc -Wall -O2 -m32 -c -o mm.o mm.c
mm.c: In function 'printblock':
mm.c:352:12: warning: format '%p' expects argument of type 'void *', but argumen
t 3 has type 'size_t {aka unsigned int}' [-Wformat=]
    printf("%p: header: [%p:%c] footer: [%p:%c]\n", bp,
           ^
mm.c:352:12: warning: format '%p' expects argument of type 'void *', but argumen
t 5 has type 'size_t {aka unsigned int}' [-Wformat=]
mm.c: At top level:
mm.c:58:14: warning: 'free_listp' defined but not used [-Wunused-variable]
    static char *free_listp = 0;
               ^
gcc -Wall -O2 -m32 -o mdriver mdriver.o mm.o memlib.o fsecs.o fcyc.o clock.o fti
mer.o
cse20181619@cspro:~/prj3-malloc$ ./mdriver
[20181619]::NAME: Hyunseo Kim, Email Address: alzflost@naver.com
Using default tracefiles in ./tracefiles/
Perf index = 44 (util) + 40 (thru) = 84/100
cse20181619@cspro:~/prj3-malloc$
```

performance 측정 결과 utilization에서 44, throughput에서 40의 점수를 얻음을 확인할 수 있었다. 참고로 first fit 방식을 채택하였을 때는 $45 + 10 = 55$ 의 결과가 나왔다.

같은 next-fit 방식에서도 초기 heap을 늘려주는 값인 CHUNKSIZE를 설정하는 값에 따라 결과가 달라짐을 확인할 수 있었다. 2^{10} 으로 바꾸었을 때는 72점이, 2^{14} 로 바꾸었을 때는 82점의 결과가 나오는 등, 몇 번의 실험 끝에 2^{12} 가 가장 최적의 결과를 도출한다는 것을 알 수 있었다. 12보다 큰 값일 때는 한 번에 메모리를 너무 크게 가져오기 때문에 공간의 utilization이 떨어지고, 작은 값일 때는 한 번에 작은 값을 sbrk로 늘려주므로 여러 번 수행하게 만들어 throughput이 떨어짐을 알 수 있었다.