Multicore Programming Project 3

담당 교수 : 최재승

이름 : 남현준

학번 : 20181625

1. Memory allocator

A. block

Padding	header	prev	next		Footer

Project3에서 기본으로 제공되는 mm.c 파일에서 alignment를 2word 단위로 진행하게 된다. 따라서, 이에 맞춰, 생성될 블록 또한 2 word alignment를 따라야 한다.

가장 처음 시작하며 free list에 할당될 initial block은 다음과 같은 구조로 생성된다. 이 때, header와 footer에는 해당 블록의 크기가 저장되고 allocate된 상태인 것을 알리기 위해 최하비트에 1을 OR연산을 통해 추가한다.

Mm_malloc() 함수를 사용하며 계속해서 memory allocator를 통해 memory를 사용하게 될 것이다. 이 때, 생성되는 block들 역시 마찬가지로 위와 같은 형태를 따르게 된다. 이 때, prev와 next에는 해당 block에서 이전 block에 위치하는 block의 next, 다음 block에 위치하는 block의 prev에 포인터를 통해 연결될 수 있도록 구현했다.

이렇게 생성되는 블록들은 allocateflag가 0, 즉 free인 상태라면 explicit list의 형태로 구현될 free list에 삽입된다.

2. 새로 추가된 함수

A. void removefromlist(void *node)

LIFO 형태로 구현되어 있는 free list에서 등록되어 있는 free block이 모종의 이유로 수정, 삭제되어야 할 때, 일관성을 위해 해당 node의 정보를 free list로부터 삭제한다.

B. Size_t allocateflag(void *node)

해당 함수는 특정 블록이 allocate 상태인지 free 상태인지를 반환한다.

C. Size_t returnblocksize(void *node)

해당 함수는 특정 블록 allocateflag를 확인하고 결과에 따라 크기를 반환한다.

D. Void *mm_malloc(size_t size)

Mm_fit()함수로 해당 size 크기의 블록을 할당할 수 있는 충분한 공간이 heap에 Size 크기의 블록을 할당할 수 있는 충분한 공간이 남아있는지 확인한다. 만약, 남아있는 여유공간이 size 크기의 블록을 할당 할 만큼 공간이 남아있지 않다면 mm_sbrk로 heap의 크기를 늘려 mm_malloc()함수의 작동이 완료될 수 있도록 한다.

이 때, 블록이 할당되고 나면 할당된 블록의 좌우를 기준으로 연속된 free block이 있을 경우 합쳐주게 된다.

E. Void split(void *node, size_t size)

현재 free상태인 블록보다 작은 크기의 블록이 mm_malloc()을 통해 할당될 때, 남은 공간을 재사용할 수 있게 free block을 사용될 부분과 사용되지 않는 부분으로 나눠줘야한다.

새롭게 할당되는 블록의 크기를 현재 free block의 크기에서 뺀 값이 블록의 최소 권장크기인 16 bytes (header, prev, next, footer) 보다 크다면, 해당 free block을 allocate된 부분과 free인 부분으로 나누게 되는 함수이다.

새롭게 생성되는 allocate된 block과 free block의 size 및 allocateflag의 마킹이 끝났다면, 새로이 생성된 free block의 주소를 free list의 제일 앞에 등록시킨다.

F. Void *mm_fit(size_t size)

특정 크기의 블록이 현재 heap을 기준으로 allocate될 수 있는 크기인지 판단하는 함수이다. Prev와 next로 연결되어 있는 free list를 탐색하며 allocateflag가 false이고 free block의 크기가 할당하고 싶은 블록의 크기보다 클 경우 해당 block의 주소를 반환하게 된다.

Free list의 처음부터 끝까지 탐색을 진행했음에도 조건을 만족하는 free block이 없었다면 NULL을 return하고 mm_sbrk()로 heap의 최대 크기를 늘려주어 그에 따른 할당을 진행하게 된다.

G. Void *merge(void *node)

인자로 넘겨진 node를 기준으로 전과 후에 free block이 있는지 확인한다.

양쪽에 allocate 된 블록만 있을 경우 추가적인 작동을 할 필요는 없고 새로 생

성된 블록의 할당만 끝내고 return한다.

다음과 같은 조건들에서는 free block들을 merge하여 하나의 큰 free block을 생성하고 새롭게 생성된 block을 free list의 가장 처음 node로 insert한다.

인자로 들어온 node와 이전 위치의 node가 free일 경우, 두 node를 합쳐 하나의 free block을 생성하고 이전 node의 주소를 free list의 제일 앞에 등록시킨다.

인자로 들어온 node와 다음 위치의 node가 free일 경우, 두 node를 합쳐 하나의 free block을 생성하고 현재 node의 주소를 free list의 제일 앞에 등록시킨다.

이전, 다음 node가 모두 free일 경우, 이전, 현재, 다음 node를 전부 합쳐 하나의 free block을 생성하고 이전 node의 주소를 free list의 제일 앞에 등록한다

H. Void *extend_heap(size_t size)

해당 함수는 memory allocator가 실행되며 allocate할 공간이 모자랄 때 size 크기만큼 heap의 크기를 확장하는 함수이다.

이후, merge함수를 통해 할당된 블록에 인접한 free block들을 연결하여 최대한 공간을 재사용할 수 있도록 한다.

I. Int mm_init(void)

Free list가 저장하게 될 initial block의 내용을 정하고 해당 주소를 초기 주소로 설정한다.

J. Void *mm_malloc(size_t size)

입력으로 들어온 size가 0이라면 NULL을 return한다.

Mm_fit()을 통해 해당 크기의 블록이 현재 할당될 수 있는 상태인지를 확인한다. Mm_fit을 통해 특정 node의 주소를 찾는 것에 성공했다면 해당 위치에 split함수를 통해 할당될 부분과 free일 부분으로 나눠주고 free list를 갱신한다.

Mm_fit의 결과가 NULL이었다면 extend_heap를 통해 heap의 최대 크기를 늘리고 새롭게 추가된 free block에 size 크기의 블록을 split을 통해 할당한다.

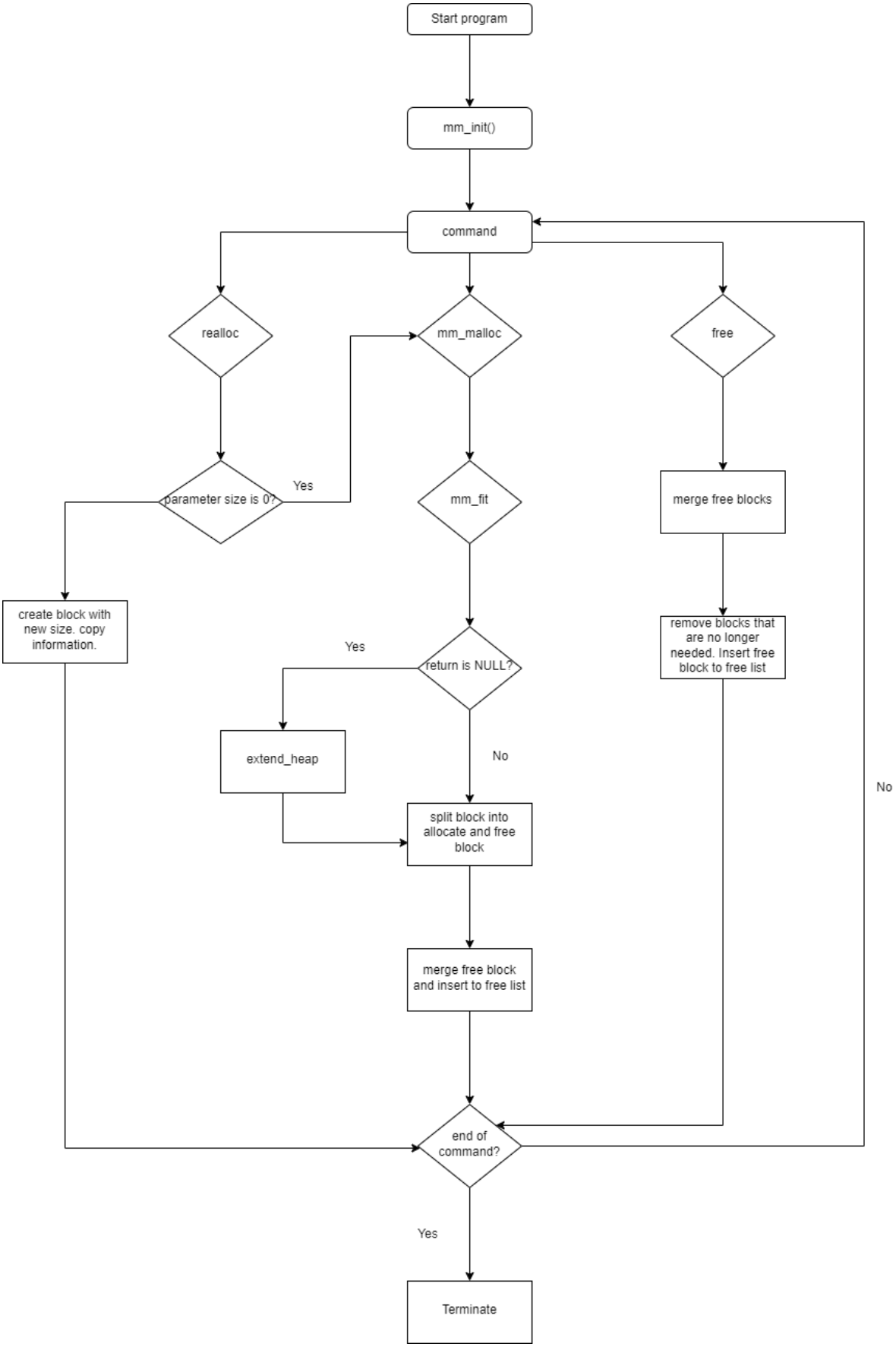
K. Void *mm_realloc(void *ptr, size_t size)

파라미터로 넘겨진 size가 0이면 NULL을 return한다.

블록이 할당되지 않은 상태라면 mm_malloc을 통해 블록을 할당하고 새롭게 생성된 블록의 주소를 return한다.

Ptr이 NULL이 아니고 인자로 들어오는 size가 ptr의 크기보다 작다면, ptr의 영역을 활용해서 size크기의 블록을 할당하게 된다. Ptr이 NULL이 아니고 size의 크기가 ptr의 크기보다 크다면 size의 크기를 이용하여 블록을 생성 및 할당하게 된다.

3. **Flow Chart**


Start program
mm_init()
command
realloc
mm_malloc
free
parameter size is 0?
Yes
mm_fit
merge free blocks
create block with new size. copy information.
remove blocks that are no longer needed. Insert free block to free list
Yes
return is NULL?
extend_heap
No
split block into allocate and free block
merge free block and insert to free list
end of command?
No
Yes
Terminate