Multicore Programming Project 3

담당 교수 : 박성용

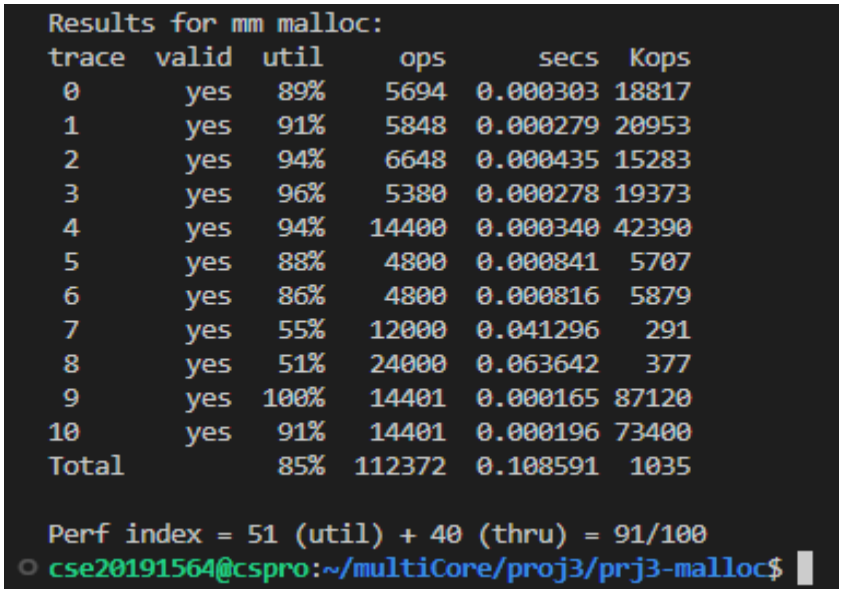
이름 : 고동헌

1. 개발 목표

- 이번 프로젝트의 목표는 수업 시간에 배운 내용을 토대로 Throughput과 peak memory utilization 을 최대한 높일 수 있도록, 자신만의 malloc 함수를 구현하는 것이다. malloc 함수의 구현 방법은 여러 가지가 있을 수 있다. 하지만 보통 free block 들을 어떻게 관리하는지에 따라 Implicit list, explicit list, segregated free list 등으로 나뉜다. Implicit List는 header에 포함된 size를 통해 block을 탐색하면서 free block 을 관리하고, explicit list는 free 상태인 block들 만을 linked list 구조로 저장한다. segregated free list는 explicit list 와 거의 유사하지만, free list 들을 size 별 class로 관리한다는 점이 다르다. 나는 이번 프로젝트에서 explicit list를 사용해서 malloc을 구현하였다.

2. 개발 범위 및 내용

- 이번 프로젝트에서 주요하게 보는 것은 크게 두 가지이다. 첫 번째는 Correctness, 두 번째는 Performance 이다. ./mdriver -V 명령어를 실행하면, 미리 정의된 tracefiles들을 input으로 받아 malloc, realloc, free 를 랜덤하게 수행한다. 이 때 모든 testcase 를 통과할 경우 correctness 는 만족된다. Performance는 memory utilization, throughput 각각에 가중치를 두어 점수를 측정하며, utilization의 최고점은 60, throughput의 최고점은 40이다. 만약 하나의 testcase 라도 통과하지 못하면 performance는 측정되지 않기 때문에, 우선 정확히 구현하는 것이 중요하다. ./mdriver -V 명령어의 실행 결과는 다음과 같다.



3. 개발 방법

- **매크로 및 변수**

- WSIZE, DSIZE

: 각각 one word, double word 의 byte 값을 저장하고 있다. 따라서 WSIZE = 4, DSIZE = 8 이다.

- CHUNCSIZE

: mm_init, mm_malloc에서 heap 영역이 부족해 heap을 확장할 때 사용하는 값이다. default 값은 (1<<12)로 설정 돼있었지만, utilization 을 높이기 위해 (1<<9)로 변경하였다.

- PACK(size, alloc)

: Block의 header, footer에 block의 size, allocation 의 여부를 저장해준다. 추후 GET_SIZE, GET_ALLOC에서 masking 을 통해 size 와 allocation 여부를 알 수 있다.

- GET(p), PUT(p, val)

: GET(p) 는 p 가 가리키는 곳에 있는 값을 1word 만큼 읽어 들인다. PUT(p,val)은 p 가 가리키는 곳에 1word val 값을 저장시킨다.

- GET_SIZE(p), GET_ALLOC(p)

: p는 block의 header 나 footer를 가리키고 있다. 위의 PACK macro 를 통해 header 와 footer에 size 와 allocation 여부를 저장하였고, 이를 각각 (x & ~0x7), (x & 0x1) 으로 masking해 값을 얻는다.

- *HDRP(bp), FTRP(bp)*

: bp는 block의 payload의 시작 지점을 가리키고 있다. HDRP(bp)는 해당 block 의 header, FTRP(bp) 는 해당 block의 footer 의 주소를 반환해준다.

- *NEXT_BLKP(bp), PREV_BLKP(bp)*

: bp 는 block의 payload의 시작 지점을 가리키고 있다. NEXT_BLKP(bp) 는 해당 block의 다음 block payload의 시작 주소를, PREV_BLKP(bp) 는 이전 block payload의 시작 주소를 반환해준다.

- *PFRBP(bp), NFRBP(bp)*

: explicit list에서, free block의 payload 시작 부분에는 free list에서의 이전 block payload 의 주소값, payload+1word 부분에는 free list에서의 다음 block payload의 주소값을 저장한다. PFRBP(bp) 는 free list에서 bp block 이전의 block pointer, NFRBP(bp) 는 다음 block pointer 를 반환해준다.

- *static char *heap_listp*

: 초기 heap 영역에서 prologue block payload를 가리킨다.

- *static void *first_list*

: free list의 첫 번째 block 을 가리키는 포인터이다.

- **함수**

- int mm_init(void)

: malloc, free, realloc 등의 명령어 수행 전 호출되는 함수이다. alignment 를 위한 padding, prologue/epilogue block 등 memory allocation 을 위한 heap 영역의 초기화를 해준다.

- static void coalesce(void*bp)*

: free block 이 생겼을 때, false fragmentation 을 방지하기 위해 물리적으로 인접한 앞, 뒤 free block 과 coalesce 를 해주는 함수이다. 이를 통해 새롭게 만들어진 free block 을 free list에 넣어주며 그 포인터를 return 해준다.

- static void extend_heap(size_t words)*

: heap 의 size를 증가시켜주는 함수이다. 내부적으론 mem_sbrk 함수를 사용한다. mem_sbrk 함수로 새롭게 할당된 free block 에 대한 포인터 값을 받아서, coalesce 함수를 호출한다.

- static void find_fit(size_t asize)*

: free list를 처음부터 살펴보면서, asize 만큼의 size를 갖는 free block 이 있는지 확인한다. 만약 있다면 해당 block에 대한 포인터를 return 해주고, 없다면 NULL 을 return 해 heap 의 size를 증가시키도록 한다.

*- static void DeleteFreeBlock(void *bp)*

: free list에서 bp 가 가리키는 block 을 제거해주는 함수이다.

*- static void InsertFreeBlock(void*bp, size_t size)*

: free list에 size 만큼의 크기를 갖는 free block 을 삽입해주는 함수이다. 보통 explicit list 의 구현은 LIFO 방식을 이용한다. 이러한 방식은 삽입을 O(1) 에 수행할 수 있기 때문에 Throughput은 올라갈 수 있지만, memory utilization 이 떨어질 수 있다. 이러한 단점을 해결하여 memory utilization 을 높이기 위해, block 을 삽입할 때 항상 size를 기준으로 오름차순이 되도록 하였다. 이렇게 하면, find_fit 함수에선 free list의 첫 번째 block 부터 찾기 때문에, segregated free list 만큼은 아니지만 best fit 과 유사한 성능을 낼 수 있다.

*- static void place(void*bp, size_t asize)*

: bp가 가리키는 free block 에 asize 만큼의 block 을 할당해주는 함수이다. 이 때 할당하고 남은 공간을 활용하기 위해 splitting을 해주어야 한다. 기존 교재에서는 할당하고 남은 공간이 block 의 최소 크기인 2*DSIZE보다 큰 경우에 splitting 하였다. 하지만 이렇게 하는 경우 size가 작은 block들이 계속 생겨, external fragmentation 이 매우 커질 수 있다. 그래서 splitting 하는 기준을 16*DSIZE 로 변경하였다.

- void mm_malloc(size_t size)*

: 최소 size byte 만큼의 payload를 갖는 block을 return 해주는 함수이다. find_fit 함수를 통해 적절한 free block 을 찾은 경우, 해당 block에 대한 포인터를 return 해준다. 만약 적절한 block을 찾지 못한 경우, extend_heap 함수를 호출해 heap 의 size를 증가시킨 뒤, extend_heap의 return으로 넘어온 새로운 free block 에 할당 해주고 그 block 포인터를 return 해준다.

- void mm_free(void ptr)*

: ptr 은 이전에 malloc, realloc 에 의해 할당된 block의 포인터이다. mm_free 함수는 해당 block의 header, footer의 alloc flag를 0으로 바꿔주고, coalesce 함수를 호출해 freed block을 free list에 넣어준다.

- void mm_realloc(void*bp, size_t size)*

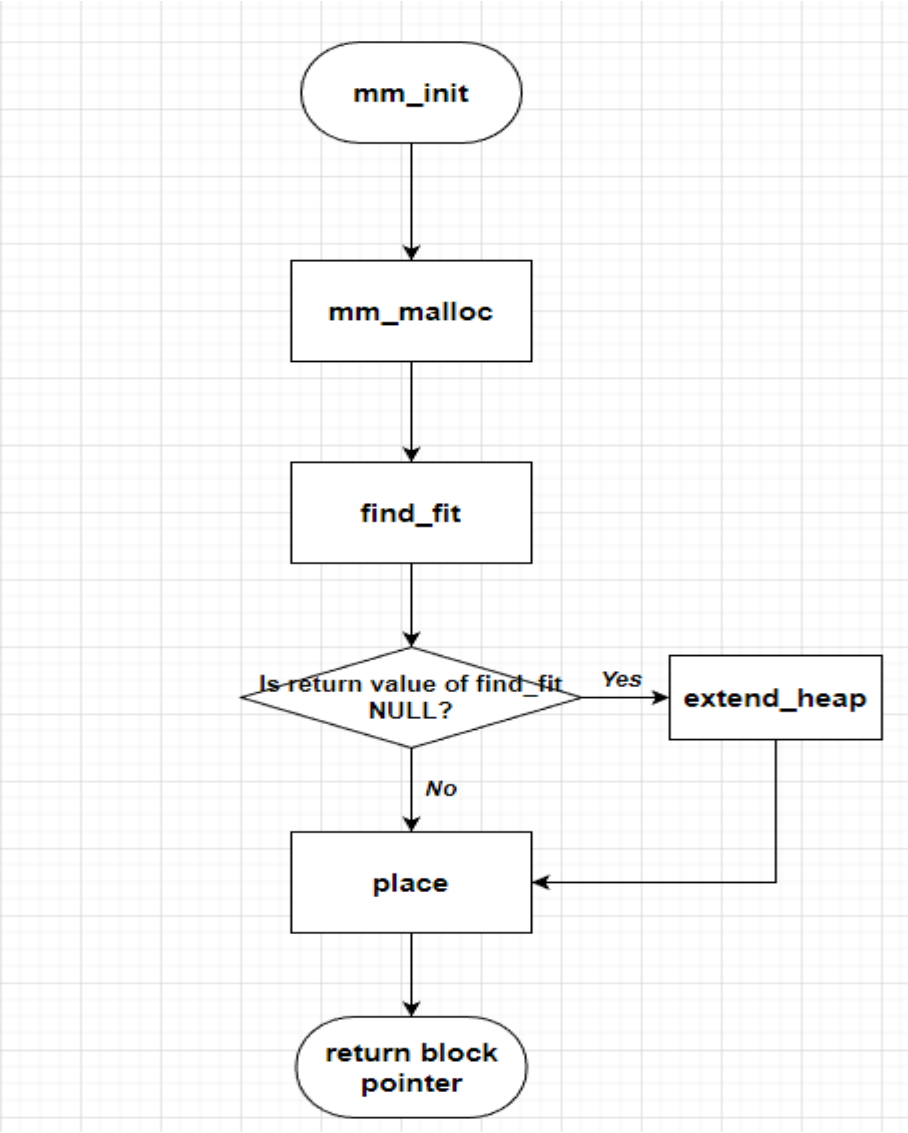
: bp는 이전에 할당된 block에 대한 포인터이다. bp가 NULL 일 경우, mm_malloc(size)와 동일한 기능을 수행한다. size가 0일 경우, mm_free(bp)와 동일한 기능을 수행한다. 만약 둘 다 아닌 경우, mm_realloc 함수는 bp가 가리키고 있는 block의 크기를 size로 바꾸고, 새로운 block에 대한 포인터를 return 해준다.

기존 bp가 가리키고 있던 block의 size를 csize, size를 alignment에 맞도록 조정한 size를 asize 라고 하자. 이 때 asize <= csize 인 경우, 새롭게 할당하고자 하는 block의 size가 더 작은 것이므로 다시 새로운 block 을 할당할 필요가 없다. 따라서 그냥 bp 를 return 해준다.

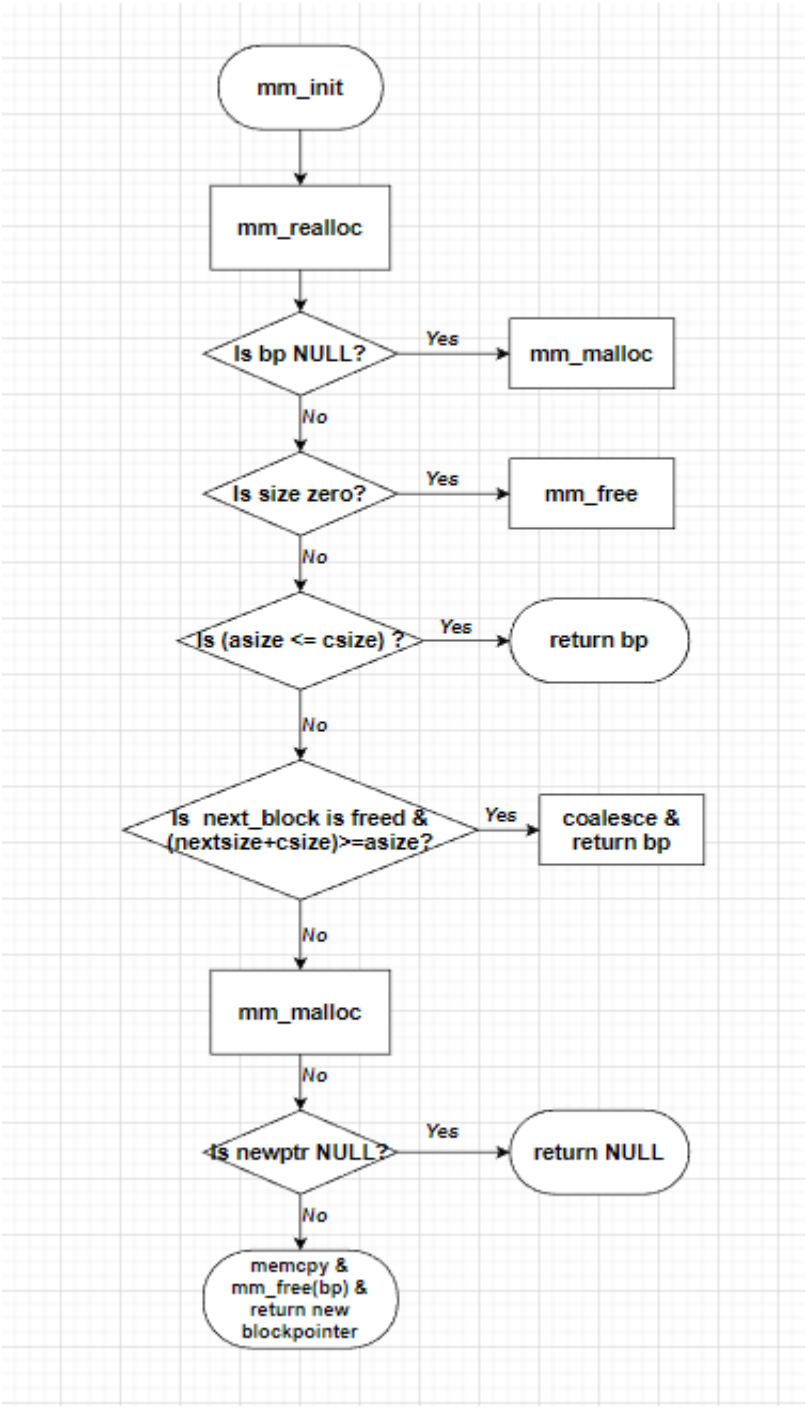
asize > csize 인 경우, 두 가지로 나뉜다. 만약 bp의 next block 이 free block 이고 next block과 coalesce 했을 때 asize 보다 크다면 새로운 block을 할당하지 않고, next block과 coalesce 하고 해당 block을 return 해주면 된다. 그렇지 않다면, mm_malloc 함수를 호출하여 새로운 block을 할당해주어야 한다. 이 때 min(old block, new block) 만큼의 payload는 realloc 된 block에 대해서 항상 old block의 내용을 저장하고 있어야 한다. 따라서 memcpy 함수를 통해 old block 의 내용을 new block으로 copy 해준다. 그 후 bp block 을 free 해주고, 새로운 block 에 대한 포인터를 return 해준다.

4. Flow Chart

(I) mm_malloc



(II) mm_realloc



(III) mm_free