Multicore Programming Project 3

담당 교수 : 최재승
이름 : 김연수
학번 : 20180501

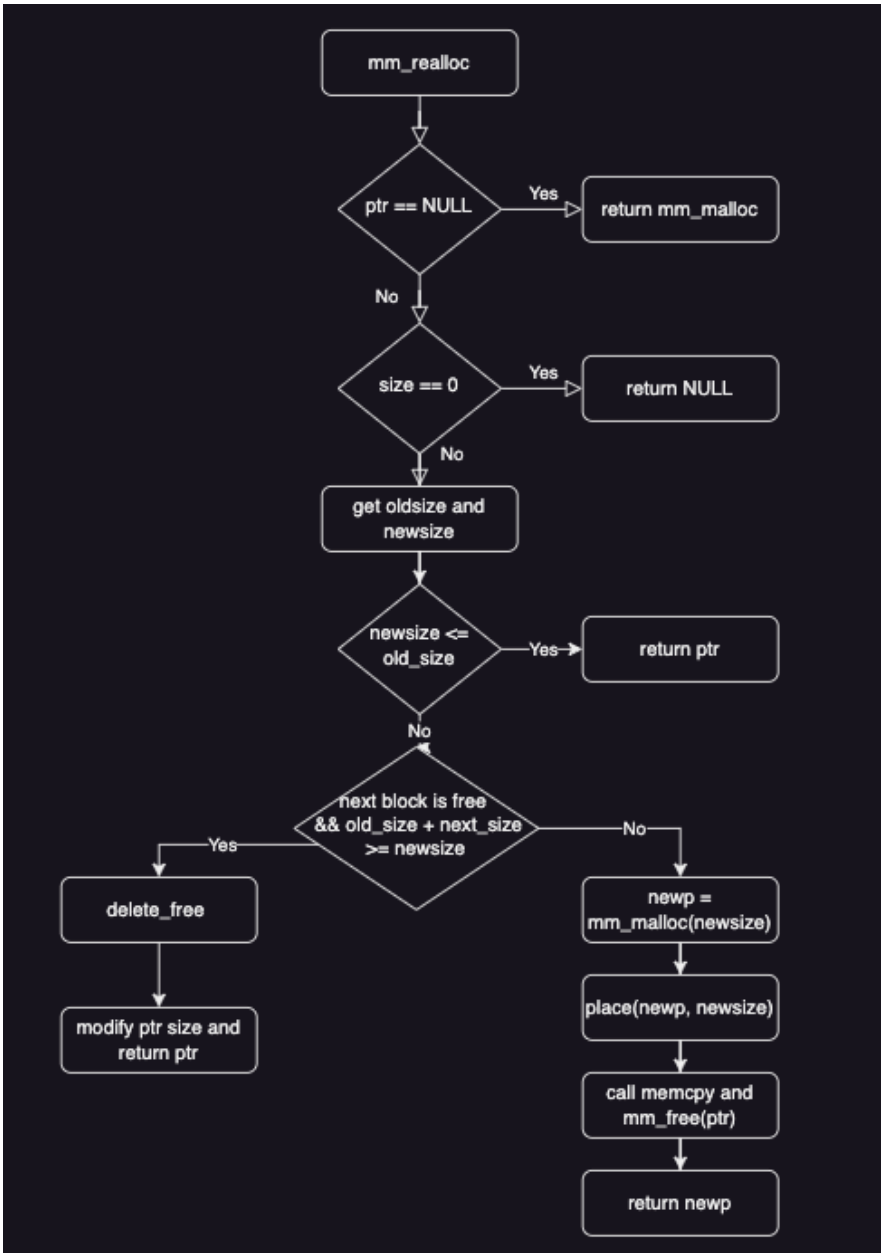
1. 개발 목표

utilization과 throughput을 최대로 만들어 performance를 증가시킬 수 있는 dynamic memory allocator를 구현한다. 본 프로젝트에서는 performance를 증가시키기 위해 explicit free list를 활용하도록 한다.

2. 내용

기존 csapp 교재에 있는 코드에서 수정한 부분을 위주로 설명하도록 한다.

mm_realloc : flowchart는 다음과 같다. realloc의 성능을 높이기 위해서 newsize가 old_size보다 작은 경우, 이미 할당된 메모리 블록을 그대로 반환한다. 그렇지 않은 경우, 인접한 빈 블록이 존재한다면, 인접한 빈 블록을 활용해 크기를 조정하고, 입력을 받은 포인터를 그대로 반환한다. 위 두 가지 경우에 해당이 안된다면, mm_malloc을 호출해 새로운 memoryblock을 할당하고, memory를 복사한 후, newp를 리턴한다.



bestfit : bestfit을 이용해 free list에서 적합한 블록을 찾도록 했다. findfit이 더 throughput의 측면에서는 유리하지만, explicit free list를 활용할 때는, bestfit을 활용해 utilization을 높이는 전략이 전체적인 performance를 높이는 데 더 우세한 것을 관찰했다.

구현은 간단하다. explicit free list를 돌면서 요청된 크기를 수용하면서 크기가 가장 작은 블록을 선택한다. 해당 조건을 만족하는 블록이 없는 경우 flag가 1로 변하지 않아 NULL을 반환한다. 코드는 아래와 같다.

```
static void* best_fit(size_t asize) {
    void *bp;
    void *best;
    size_t flag = 0;
    size_t min_size = SIZE_MAX;

    for (bp = free_listp; GET_ALLOC(HDRP(bp)) == 0; bp = GET(NEXT_FREE_BLKP(bp))) {
        // 요청된 크기를 수용 가능하면서 가장 작은 블록을 선택
        if ((asize <= GET_SIZE(HDRP(bp))) && (GET_SIZE(HDRP(bp)) < min_size)) {
            min_size = GET_SIZE(HDRP(bp));
            flag = 1;
            best = bp;
        }
    }
    if (flag)
        return best;

    return NULL;
}
```

insert_free : explicit free list에 블록을 집어넣는 역할을 한다. 인자로 들어온 bp의 이전 빈 블록 포인터를 NULL로 설정하고, free_listp(free list의 헤더)의 이전 빈 블록 포인터를 bp로 설정하고, bp의 다음 빈 블록 포인터에 free_listp를 넣는다. free_listp를 bp로 설정해주면, linked list에 새로운 블록을 추가하는 작업이 완료된다.

```
static void insert_free(void *bp) {
    // 'bp'의 이전 빈 블록 포인터를 NULL로 설정
    PUT(PREV_FREE_BLKP(bp), NULL);
    // 현재 빈 블록 리스트의 첫 번째 블록의 이전 빈 블록 포인터를 'bp'로 설정
    PUT(PREV_FREE_BLKP(free_listp), bp);

    // 'bp'의 다음 빈 블록 포인터를 현재 빈 블록 리스트의 첫 번째 블록으로 설정
    PUT(NEXT_FREE_BLKP(bp), free_listp);

    // 'bp'를 새로운 빈 블록 리스트의 첫 번째 블록으로 설정
    free_listp = bp;
}
```

delete_free : 이전 빈 블록 포인터가 NULL인 경우와 그렇지 않은 경우를 나눠서 처리한다. 이전 빈 블록 포인터가 NULL인 경우, 해당 bp가 linked list의 헤더에 위치한다는 뜻이므로, free_listp의 위치를 bp의 다음 빈 블록으로 바꿔주고, 다음 빈 블록의 전 빈 블록을 NULL로 설정해준다. bp가 헤더가 아닌 경우에는, 전 빈 블록과 다음 빈 블록을 연결시켜주고 함수를 종료한다. 코드는 아래와 같다.

```
static void delete_free(void *bp) {
    // 'bp'의 이전 빈 블록 포인터를 가져옴
    char *prev = GET(PREV_FREE_BLKP(bp));
    // 'bp'의 다음 빈 블록 포인터를 가져옴
    char *next = GET(NEXT_FREE_BLKP(bp));

    // 이전 빈 블록 포인터가 NULL인 경우
    if (!prev) {
        // 'bp'를 새로운 빈 블록 리스트의 첫 번째 블록으로 설정
        free_listp = next;

        // 다음 빈 블록의 이전 빈 블록 포인터를 NULL로 설정
        PUT(PREV_FREE_BLKP(next), NULL);
    }
    else {
        // 이전 빈 블록의 다음 빈 블록 포인터를 'next'로 설정
        PUT(NEXT_FREE_BLKP(prev), next);

        // 다음 빈 블록의 이전 빈 블록 포인터를 'prev'로 설정
        PUT(PREV_FREE_BLKP(next), prev);
    }
return;
}
```

coalesce : 기존 csapp 교재에 쓰여있는 coalesce코드에 몇가지 코드를 추가했다.

case 1에서는 바로 insert_free(bp)를 호출하고 반환하도록 했다.

case 2,3에서는 delete_free를 호출해 다음블록 또는 전 블록이 free상태인 경우, free list에서 삭제 했다.

case 4에서는 이전 블록과 다음블록 모두 free list에서 삭제한다.

위 과정을 거치고 나서 병합 된 bp블록을 insert_free를 호출해 free list에 집어넣도록 한다. 코드는 아래와 같다.

```c
static void *coalesce(void *bp)
{
    // 이전 블록이 할당되었는지 확인하고, 첫 번째 블록을 가리키는 경우 처리
    size_t prev_alloc = GET_ALLOC(FTRP(PREV_BLKP(bp))) || PREV_BLKP(bp) == bp;
    // 다음 블록이 할당되었는지 확인
    size_t next_alloc = GET_ALLOC(HDRP(NEXT_BLKP(bp)));
    // 현재 블록의 크기를 가져옴
    size_t size = GET_SIZE(HDRP(bp));

    // 이전 블록과 다음 블록 모두 할당된 경우 (Case 1)
    if (prev_alloc && next_alloc) {
        insert_free(bp);
        return bp;
    }
    // 이전 블록은 할당되고 다음 블록이 비어있는 경우 (Case 2)
    else if (prev_alloc && !next_alloc) {
        // 다음 블록을 빈 블록 리스트에서 제거
        delete_free(NEXT_BLKP(bp));
        // 현재 블록과 다음 블록의 크기를 결합
        size += GET_SIZE(HDRP(NEXT_BLKP(bp)));
        // 헤더와 푸터에 결합된 크기와 할당되지 않음을 표시
        PUT(HDRP(bp), PACK(size, 0));
        PUT(FTRP(bp), PACK(size, 0));
    }
    // 이전 블록이 비어있고 다음 블록이 할당된 경우 (Case 3)
    else if (!prev_alloc && next_alloc) {
        // 이전 블록을 빈 블록 리스트에서 제거
        delete_free(PREV_BLKP(bp));
        // 현재 블록과 이전 블록의 크기를 결합
        size += GET_SIZE(HDRP(PREV_BLKP(bp)));
        // 이전 블록의 헤더와 현재 블록의 푸터에 결합된 크기와 할당되지 않음을 표시
        PUT(FTRP(bp), PACK(size, 0));
        PUT(HDRP(PREV_BLKP(bp)), PACK(size, 0));
        // 포인터를 이전 블록으로 옮긴다.
        bp = PREV_BLKP(bp);
    }
    }
    // 이전 블록과 다음 블록이 둘 다 비어있는 경우 (case 4)
    else {
        // 다음 블록과 이전 블록을 빈 블록 리스트에서 제거
        delete_free(NEXT_BLKP(bp));
        delete_free(PREV_BLKP(bp));
        // 현재 블록과 이전 블록, 다음 블록의 크기를 결합
        size += GET_SIZE(HDRP(PREV_BLKP(bp))) +
            GET_SIZE(FTRP(NEXT_BLKP(bp)));
        // 이전 블록의 헤더와 다음 블록의 푸터에 결합된 크기와 할당되지 않음을 표시
        PUT(HDRP(PREV_BLKP(bp)), PACK(size, 0));
        PUT(FTRP(NEXT_BLKP(bp)), PACK(size, 0));
        // 포인터를 이전 블록으로 옮긴다.
        bp = PREV_BLKP(bp);
    }
    // 결합된 블록을 빈 블록 리스트에 삽입
    insert_free(bp);

    return bp;
}
```

global variable : bp는 다음 빈 블록의 위치를 가리키고, bp+4는 이전 빈 블록의 위치를 가리키도록 설정했다.

전역변수로 explicit free list의 헤더를 포인터로 선언했다.

```
#define NEXT_FREE_BLKP(bp) ((char *)(bp))
#define PREV_FREE_BLKP(bp) ((char *)(bp) + WSIZE)

static char *heap_listp;
static char *free_listp;
```

3. 결과

위 내용을 구현해 "./mdiver –V"를 실행한 결과는 아래와 같다. first_fit을 사용할 때 88점에서 bestfit을 사용하니 91점으로 점수가 오르는 것을 관찰할 수 있었다.

```
Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   99%    5694  0.000448 12698
 1       yes   99%    5848  0.000374 15620
 2       yes   99%    6648  0.000492 13518
 3       yes  100%    5380  0.000444 12106
 4       yes   99%   14400  0.000466 30915
 5       yes   96%    4800  0.004516  1063
 6       yes   96%    4800  0.004346  1105
 7       yes   55%   12000  0.030409   395
 8       yes   51%   24000  0.095665   251
 9       yes   93%   14401  0.000190 75715
10       yes   43%   14401  0.000294 48933
Total          85%  112372  0.137645   816

Perf index = 51 (util) + 40 (thru) = 91/100
```