System Programming Project 4

담당 교수 : 박성용

이름 : 임나현

학번 : 20211582

1. **개발 목표**

C프로그래밍에서 사용하는 동적 할당을 하는 방식을 구현한다. init, malloc, free, realloc하는 사용자 정의의 함수를 활용하여 알맞은 크기의 block을 할당한다. free block을 관리하는 방식은 implicit list, explicit list, segregated list 등의 방식이 있는데 최적화하여 관리할 수 있는 방식을 활용하는 것이 목표이다.

.

2. **개발 방법**

- **구현하기 위해 어느 소스코드에 어떤 요소를 추가 또는 수정할 것인지 설명. (함수, 구조체 등의 구현이나 수정을 서술)**

사용한 free block 관리 방식 : explicit free list

Global variable : 전체 block에 대한 정보인 heap_listp와 free block에 대해서만 연결되어 있는 free_listp가 필요하다.

Init 함수

```
int mm_init(void) {
    if ((heap_listp = mem_sbrk(6* WSIZE)) == (void*)-1)
        return -1;
    PUT(heap_listp, 0);
    PUT(heap_listp + (1 * WSIZE), PACK(DSIZE, 1));  // Prologue header
    PUT(heap_listp + (2 * WSIZE), PACK(DSIZE, 1));  // Prologue footer
    PUT(heap_listp + (3 * WSIZE), PACK(0, 1));      // Epilogue header
    heap_listp += 2 * WSIZE;
    free_listp = heap_listp;
    if (extend_heap(CHUNKSIZE / WSIZE) == NULL)
        return -1;

    return 0;
}
```

: 처음 heap_list를 생성할 때 제일 앞에 alignment padding, prologue의 header와 footer, epilogue의 header로 구성하고 free_listp를 headp_listp로 초기화한다. Extend_heap에서는 크기를 1024 word만큼 처음에 늘려준다.

Malloc & Extend_heap함수

```
void* mm_malloc(size_t size) //강의자료
{
    size_t asize;
    size_t extendsize;
    char* bp;
    if (size == 0) return NULL;
    if (size <= DSIZE)
        asize = 2 * DSIZE;
    else
        asize = DSIZE * ((size + (DSIZE)+(DSIZE - 1)) / DSIZE);

    if ((bp = find_fit(asize)) != NULL) {
        place(bp, asize);
        return bp;
    }
    extendsize = MAX(asize, CHUNKSIZE);
    if ((bp = extend_heap(extendsize / WSIZE)) == NULL)
        return NULL;
    place(bp, asize);
    return bp;
}

static void* extend_heap(size_t words) { //강의자료
    char* bp;
    size_t size;
    size = (words % 2) ? (words + 1) * WSIZE : words * WSIZE;
    if ((long)(bp = mem_sbrk(size)) == -1)
        return NULL;

    PUT(HDRP(bp), PACK(size, 0));
    PUT(FTRP(bp), PACK(size, 0));
    PUT(HDRP(NEXT_BLKP(bp)), PACK(0, 1));
    return coalesce(bp);
}
```

: 강의자료에 나와있는 implicit list와 동일한 코드를 사용한다. malloc 시 find_fit함수를 통해 free block중에서 할당할 수 있는 크기가 있는 경우 place 함수를 통해 allocate시킨다. 만약, 할당할 수 있는 크기가 없는 경우 extend_heap 함수를 통해 추가로 heap의 크기를 증가시켜 다시 할당시킨다.

Find_Fit 함수

```
static void *find_fit(size_t asize)
{
    void *bp;
    for(bp = free_listp; GET_ALLOC(HDRP(bp)) == 0; bp = NEXT_P(bp))
        if(GET_SIZE(HDRP(bp)) >= asize){
            return bp;
        }
    }
    return NULL;
}
```

: 전체 heapdmf 다 탐색하는 것이 아닌 free block들만을 탐색하여 자리가 있는 경우 해당 위치를 반환한다.

Place 함수

```
static void place(void *bp, size_t asize)
{
    size_t csize = GET_SIZE(HDRP(bp));
    delete_fb(bp);

    if((csize - asize) >= (2*DSIZE)){ //일부사용
        PUT(HDRP(bp), PACK(asize, 1));
        PUT(FTRP(bp), PACK(asize, 1));
        void* next_bp = NEXT_BLKP(bp);
        PUT(HDRP(next_bp), PACK(csize-asize, 0));
        PUT(FTRP(next_bp), PACK(csize-asize, 0));
        add_fb(next_bp);
    }
    else{ //전부사용
        PUT(HDRP(bp), PACK(csize, 1));
        PUT(FTRP(bp), PACK(csize, 1));
    }
}
```

: 해당 bp에 size만큼의 크기 이상이 존재한다면 size만큼 할당시켜준다. 만약, 남는 크기가 존재할 경우, 해당 free block을 새롭게 추가해준다. 만약, size만큼의 크기가 없다면 에러 사항이기 때문에 예외 처리한다.

Coalesce 함수

```
static void *coalesce(void *bp)
{
    size_t prev_alloc = GET_ALLOC(FTRP(PREV_BLKP(bp))) || PREV_BLKP(bp) ==
    size_t next_alloc = GET_ALLOC(HDRP(NEXT_BLKP(bp)));
    size_t size = GET_SIZE(HDRP(bp));

    if(prev_alloc && next_alloc){//case1
        add_fb(bp);
        return bp;
    }
    else if(prev_alloc && !next_alloc){//case2
        delete_fb(NEXT_BLKP(bp));
        size += GET_SIZE(HDRP(NEXT_BLKP(bp)));
        PUT(HDRP(bp), PACK(size, 0));
        PUT(FTRP(bp), PACK(size, 0));
    }
    else if(!prev_alloc && next_alloc){//case3
        delete_fb(PREV_BLKP(bp));
        size += GET_SIZE(HDRP(PREV_BLKP(bp)));
        PUT(FTRP(bp), PACK(size, 0));
        bp = PREV_BLKP(bp);
        PUT(HDRP(bp), PACK(size, 0));
    }
    else{ //case4
        delete_fb(PREV_BLKP(bp));
        delete_fb(NEXT_BLKP(bp));
        size += GET_SIZE(HDRP(PREV_BLKP(bp))) + GET_SIZE(FTRP(NEXT_BLKP(bp)
        PUT(FTRP(NEXT_BLKP(bp)), PACK(size, 0));
        bp = PREV_BLKP(bp);
        PUT(HDRP(bp), PACK(size, 0));
    }
    add_fb(bp);
    return bp;
}
```

: 이 함수에서는 크게 4가지 case에 따라 다르게 동작하게 된다. Case1에서는 만약 현재 bp의 앞뒤의 block이 전부 할당이 되어있으면 해당 bp의 block만 free해준다. Case2에서는 bp의 앞에는 할당이 되어 있지만 뒤의 block은 free block이라면 해당 bp와 뒤의 free block을 하나로 합쳐서 큰 free block을 생성한다. Case3에서도 마찬가지로 뒤의 block에는 할당이 되어 있고 앞의 block이 free block이기 때문에 앞의 블록과 현재 블록을 합쳐서 free block을 생성한다. Case4에서는 해당 bp의 앞뒤 block이 전부 free block 상태이기 때문에 bp를 앞의 block으로

이동시켜주고 세개의 block을 한꺼번에 묶어 free block에 add해준다. 이렇게 하는 이유는 free block에 할당할 때, 남은 free space가 있음에도 불구하고 allocator가 찾지 못하는 문제를 해결하기 위해서이다.

Add& Delete 함수

```
static void delete_fb(void *bp)
{
    if(PREV_P(bp) == NULL) { //첫번째
        free_listp=NEXT_P(bp);
        PREV_P(free_listp) = NULL;
        return;
    }
    if (NEXT_P(bp) == NULL) {//마지막
        NEXT_P(PREV_P(bp)) = NEXT_P(bp);
        return;
    }
    PREV_P(NEXT_P(bp)) = PREV_P(bp);
    NEXT_P(PREV_P(bp)) = NEXT_P(bp);
}

static void add_fb(void *bp)
{
    void *temp = free_listp;
    free_listp = bp;
    PREV_P(bp) = NULL;
    NEXT_P(bp) = temp;
    if (temp != NULL){
        PREV_P(temp) = bp;
    }
}
```

: 먼저, delete 함수같은 경우에는 해당 free block의 previous free block의 next 포인터를 free block의 next block으로 변경하고, free block의 next block의 prev 포인터를 free block의 previous block으로 변경한다.

Add 함수를 구현하는 방법은 LIFO, FIFO, address 등등의 많은 방식이 있지만 효율성을 높이기 위해 새로 추가되는 free block을 제일 앞으로 연결하는 LIFO 방식을 사용하였다.

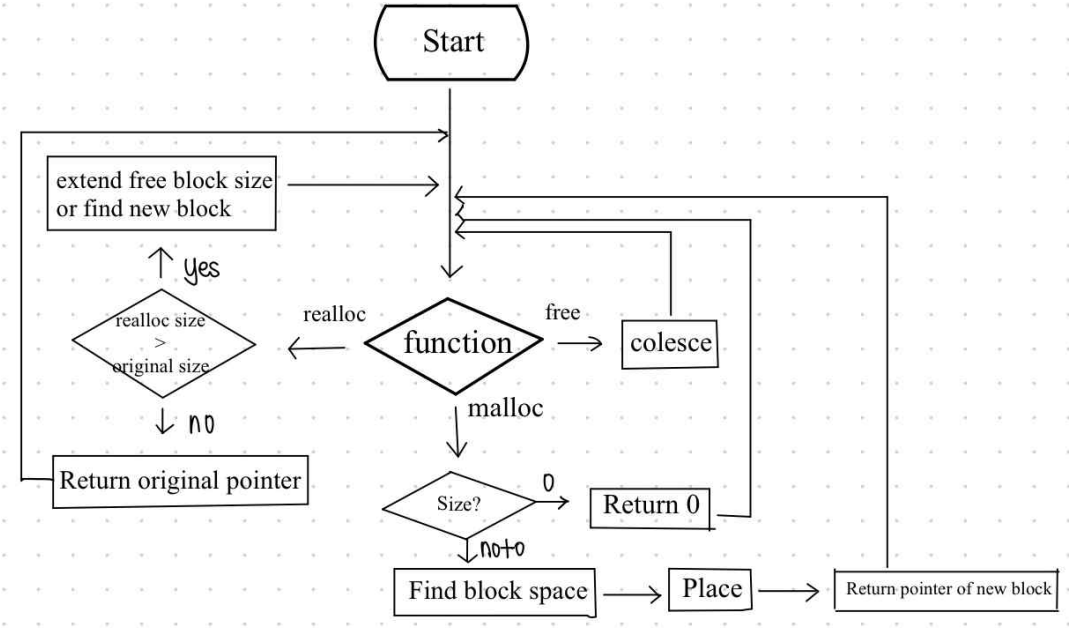
Realloc 함수

```c
void mm_free(void *ptr) //강의자료
{
    size_t size = GET_SIZE(HDRP(ptr));
    PUT(HDRP(ptr), PACK(size, 0));
    PUT(FTRP(ptr), PACK(size, 0));
    coalesce(ptr);
}

/*
 * mm_realloc - Implemented simply in terms of mm_malloc and mm_free
 */
void *mm_realloc(void *ptr, size_t size)
{
    size_t oldsize, newsize;
    void *oldptr = ptr;
    void *newptr;

    if (size <= 0) {
        mm_free(ptr);
        return NULL;
    }
    if (!oldptr) {
        return mm_malloc(size);
    }

    oldsize = GET_SIZE(HDRP(ptr));
    newsize = size + (2 * WSIZE);
    if (newsize <= oldsize) return ptr;
    if (!GET_ALLOC(HDRP(NEXT_BLKP(oldptr)))){
        size_t n_blk_size = GET_SIZE(HDRP(NEXT_BLKP(oldptr)));
        size_t co_fsize = oldsize + n_blk_size;
        delete_fb(NEXT_BLKP(oldptr));
        PUT(HDRP(oldptr), PACK(co_fsize, 1));
        PUT(FTRP(oldptr), PACK(co_fsize, 1));
        return oldptr;
    }
    else{
        newptr=mm_malloc(newsize);
        if(newptr == NULL) return NULL;

        place(newptr, newsize);
        memcpy(newptr, oldptr, oldsize);
        mm_free(oldptr);
        return newptr;
    }
}
```

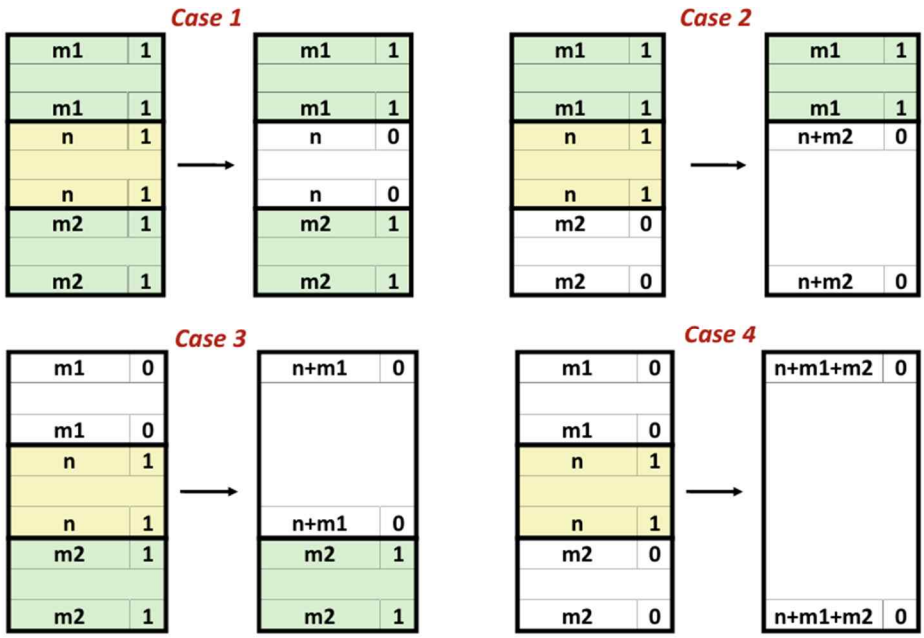
: realloc의 구현은 size가 0보다 작거나 같은 경우는 free와 같은 동작을 하게 했고, 그 외의 경우에 할당시킨다. 할당시킬 때 현재 oldptr의 next block이 free block이면 새로운 크기만큼 연결하여 할당하고, 그렇지 않은 경우에는 새로운 크기로 메모리를 할당하고 place 함수를 통해 적절하게 위치시킨다.

3. 구현 결과

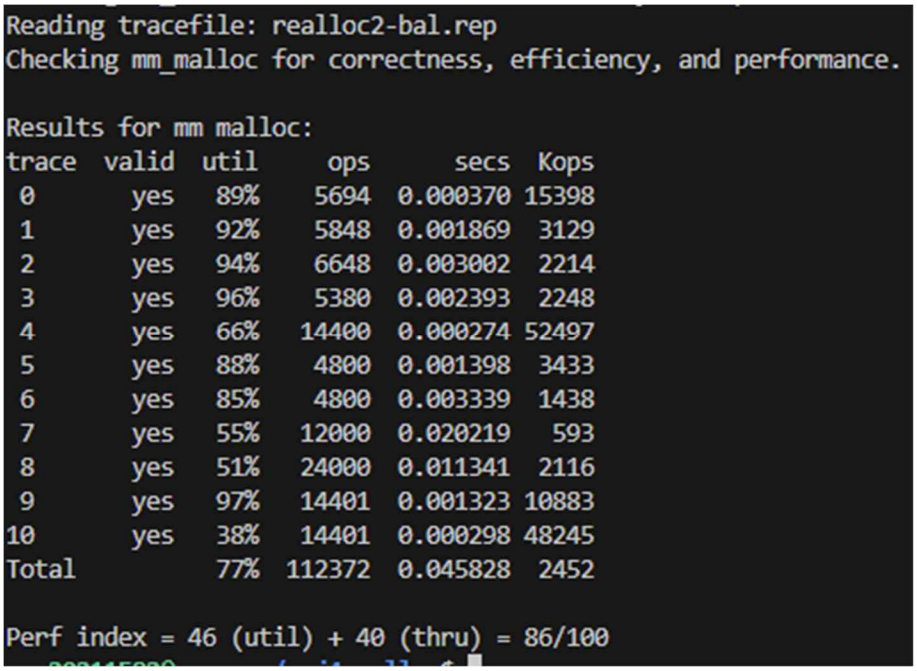
A. Flow Chart



Colesce의 경우 4가지의 case에 따라 구분하여 동작한다. (강의자료참고)



B. 성능 평가 결과



```
Reading tracefile: realloc2-bal.rep
Checking mm_malloc for correctness, efficiency, and performance.

Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   89%    5694  0.000370 15398
 1       yes   92%    5848  0.001869  3129
 2       yes   94%    6648  0.003002  2214
 3       yes   96%    5380  0.002393  2248
 4       yes   66%   14400  0.000274 52497
 5       yes   88%    4800  0.001398  3433
 6       yes   85%    4800  0.003339  1438
 7       yes   55%   12000  0.020219   593
 8       yes   51%   24000  0.011341  2116
 9       yes   97%   14401  0.001323 10883
10       yes   38%   14401  0.000298 48245
Total          77%  112372  0.045828  2452

Perf index = 46 (util) + 40 (thru) = 86/100
```

강의 자료에 나와있는 implicit free list의 방식을 활용하여 코드를 작성하면 성능이 55점 정도의 performance가 나왔다. 이에 반해, explicit free list 방식을 활용하면 성능 평가에서 86점이 나온다는 것을 확인할 수 있다. 두 코드에서 가장 큰 차이점은 free block끼리만을 연결한 free list 포인터를 활용했다는 점이다. Allocation시 free block을 찾는 find_fit이나 delete, add 함수에서 전체 heap block이 아닌 free block만을 사용하여 탐색하기 때문에 throughput에서 큰 차이를 보인 것을 확인할 수 있다.