System Programming Project 4

담당 교수 : 박성용 교수님

학번 : 20201654

이름 : 최호진

1. 개발 목표

메모리의 힙 영역 자료구조와 블록을 직접 구현하고 관련된 함수(malloc, realloc, free)를 구현한다. 또한, Throughput과 Utilization을 고려해 성능을 높일 수 있도록 Segregated list Allocate 방식을 이용해 구현한다. 이 방식은 여러 개의 linked list에 블록을 크기에 따라 구분해 저장하여, 필요한 블록을 찾을 때 걸리는 시간이 줄여 Throughput을 높일 수 있고, 또한 리스트 내에 블록들을 크기순으로 정렬하여, 메모리의 utilization도 높일 수 있도록 구현이 가능하다. 해당 프로젝트에서는 15개의 리스트를 만들어 2^i 크기 단위로 나누어 저장할 수 있게 하였다.

2. 개발(구현) 내용

A. 전역 변수

1. void* seg1 ~ seg15

블록들을 저장하기 위한 리스트들로 각 index는 저장할 수 있는 블록의 가장 큰 크기 (2^{i+4})를 나타낸다.

2. void* heap

heap 메모리 영역을 나타낸다.

3. size_t before

특정 테스트케이스에서 Utilization을 높이기 위해 도입한 변수로 ad-hoc하게 사용할 수 있도록 선언한 변수이다.

B. 함수

1. mm_init()

메모리 할당을 하기 전에 힙 영역의 공간을 만들어준다. Prologue와 epilogue 블록을 먼저 만들어주고, 미리 힙 영역에 적당한 영역을 할당한다(CHUNKSIZE(4096) words 할당). 또한 segregated list들(seg1~seg15)들을 초기화한다.

2. mm_malloc(size_t size)

인자로 주어진 size의 data영역을 가진 블록을 할당하기 위한 함수이다. 이때 블록의 사이즈는 alignment 규칙을 지킬 수 있도록 8의 배수로 적절하게 바꿔주어서 할당하게 되는데, 바뀐 사이즈를 할당할 수 있는 적절한 free 블록을 찾을 때

까지 seg1 ~ seg15 리스트에서 순서대로 찾는다. 만약 적절한 크기가 없고. 주어진 사이즈가 힙 영역보다 큰 경우에는 extend_heap 함수로 힙 영역을 확장한다.

3. mm_free(void* bp)

deallocate 하고자 하는 pointer를 인자로 받아서 메모리 할당을 해제한다. 만약, 주어진 pointer가 null(0)인 경우에는 그냥 리턴한다. 그 다음 coalesce 함수에 free된 블록과 그 사이즈를 인자로 넘겨주면서 호출하고, coalesce 함수에서 블록의 앞뒤를 확인한 후 필요한 coalescing을 한 뒤 해당 함수에서 insert 함수를 호출해 블록을 segregated list의 적절한 위치에 넣어준다.

4. extend_heap(size_t words)

인자 words에 해당하는 size를 alignment 규칙에 맞게 변형한 뒤 그 값만큼 힙 영역의 크기를 확장한다. 그 다음 coalesce 함수를 호출하여 확장된 영역의 블록을 segregated list에 추가한다.

5. coalesce(void* bp, size_t s)

인자로 bp와 해당 블록의 크기를 받는다. 인자로 받은 포인터의 앞뒤 블록의 상태를 확인해 4가지 케이스에 따라 알맞게 처리를 한다. 블록 앞뒤가 모두 allocated block일 경우 해당 블록을 바로 seg list에 insert하고, 그렇지 않은 경우 free block을 seg list에서 삭제하고, 인자로 받은 포인터와 해당 블록을 합쳐서 만든 새로운 크기의 블록을 다시 할당한다.

6. place(void* bp, size_t s)

인자로 bp와 할당할 블록의 크기를 받는다. 인자로 받은 블록의 전체 사이즈에서 할당할 사이즈를 제거하고 남은 크기의 free block을 seg list에 다시 넣어주기 위한 함수이다. 이때, utilization을 높이기 위해서 여러가지를 고려하여 구현하였는데, 먼저 전체 블록의 사이즈 (csize)와 할당할 블록 사이즈(asize)의 크기를 고려해 어떤 식으로 할당할지 결정하는데, allocated된 블록 앞쪽에 먼저 넣을지, 아니면 남은 부분을 앞쪽에 넣을지를 결정하였다. 이때, csize와 asize가 극단적으로 차이가 많이 나거나, 차이가 적을 때를 특별한 케이스로 고려하여 남은 블록을 먼저 넣을 수 있도록 정하였는데, 그 기준은 값을 바꾸며 테스트한 결과에 따라 휴리스틱하게 정하였다. 가장 최적의 결과가 csize와 asize를 뺀 값이 0.9csize 이상이거나, 0.1csize 이하일 때 가장 최적의 utilization을 보여 이와 같이 정하였다.

또한, 추가적으로 이렇게 하였을 때도 7, 8번 테스트 케이스에 대해서는 Utilization이 50%대로 낮게 나왔는데, 해당 케이스들은 두 종류의 사이즈를 가진 블록이 번갈아 가면서 먼저 할당되고, 그 다음 한 종류의 블록을 모두 free 시킨 후 그보다 큰 사이즈의 블록을 할당하는 경우였다. 따라서, 이러한 케이스를 처리해주기 위해 전역변수 before을 추가해, 해당 케이스의 경우도 할당되지 않은(남은) 블록을 먼저 할당하도록 바꿔주었다. 그렇게 하니, 두 케이스 모두 90% 이상의 Utilization을 얻을 수 있었다.

7. Insert(char* bp, size_t size)

인자로 전달받은 bp를 적절한 Segregated list에 할당시키는 함수이다. Size를 통해 bp가 들어갈 적절한 리스트를 선택하고, 해당 리스트(doubly linked list) 내에서 크기에 맞는(오름차순) 적절한 위치에 블록이 할당되도록 위치를 찾아 추가한다.

8. delete(char* bp)

리스트에서 제거하고자 하는 블록의 포인터를 인자로 전달받아 리스트 내에서 해당 블록을 찾아 제거한다. (Doubly linked list의 성질을 유지하도록 알맞게 제거)

9. mm_realloc(char* bp)

realloc도 utilization에 큰 영향을 미치는 함수로, 단순히 새롭게 위치를 할당할 경우 utilization이 크게 낮아질 수 있다. 따라서, 블록의 사이즈를 조정할 때 다음 블록을 확인하여, 다음 블록이 free block이고 그 블록을 이용하여 위치를 바꾸지 않고 할당이 가능할 경우 다음 블록을 seg list에서 지우고, 같이 이용해서 블록을 할당한다. 아닌 경우 새롭게 할당하고, 기존의 블록은 free 시킨다.

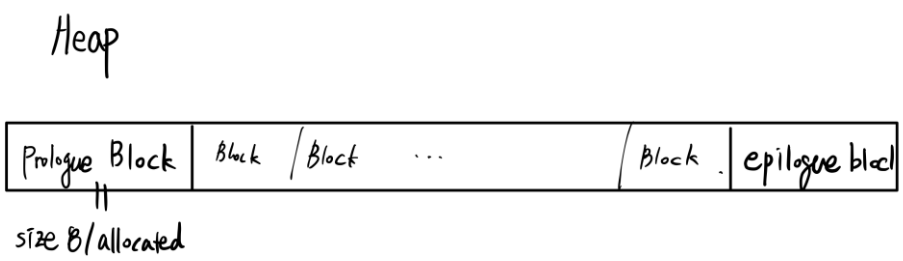
C. 주요 매크로

1. WSIZE/DSIZE: Word Size(4) and Double Word Size(8)
2. PACK(size, alloc): size와 allocatd bit을 word에 packing
3. ALIGN(size): Alignment 규칙에 맞도록 적절하게 size 변경
4. GET(p) / PUT(p, val) / GET_SIZE(p) / GET_ALLOC(p): Block에 관한 정보 읽기
5. HDRP/FTRP(bp): bp의 Header, Footer 위치 계산
6. NEXT_BLKP/PREV/BLKP 이전 블록 또는 다음 블록의 address 계산

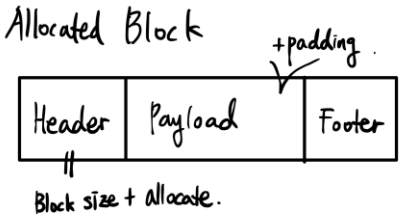
3. 플로우 차트 및 다이어그램

A. 자료구조 다이어그램

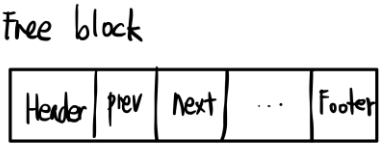
1) Heap



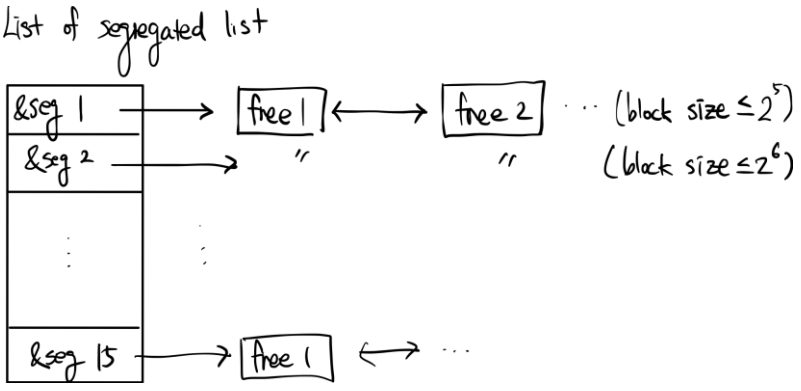
2) Allocated Block



3) Free Block

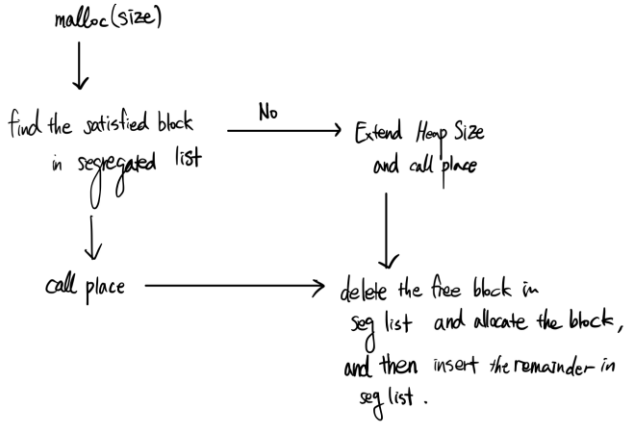


4) Segregated list

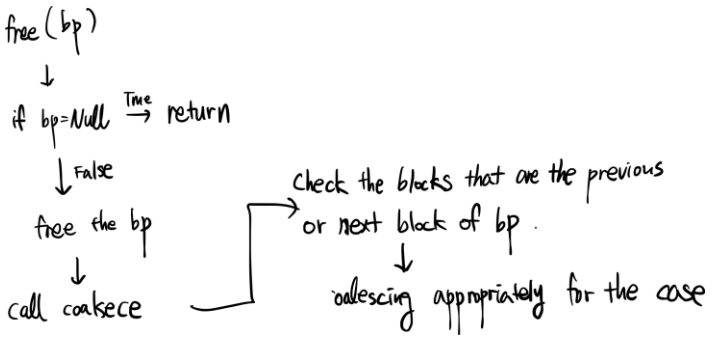


B. Malloc, free, realloc 플로우차트

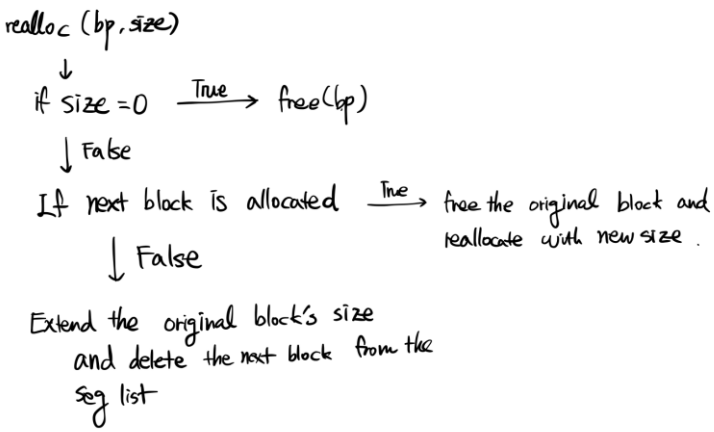
1) malloc



2) free



3) realloc



4. **구현 결과**

```
./cse20201654@cspro:~/sp/prj4-malloc$ ./mdriver -v
[20201654]::NAME: Hojin Choi, Email Address: kaler1234@sogang.ac.kr
Using default tracefiles in ./tracefiles/
Measuring performance with gettimeofday().

Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   99%    5694  0.004657  1223
 1       yes   99%    5848  0.003547  1649
 2       yes   99%    6648  0.003960  1679
 3       yes   99%    5380  0.003376  1594
 4       yes  100%   14400  0.006639  2169
 5       yes   96%    4800  0.005856   820
 6       yes   95%    4800  0.007584   633
 7       yes   96%   12000  0.023313   515
 8       yes   90%   24000  0.019165  1252
 9       yes   99%   14401  0.003182  4525
10       yes   98%   14401  0.003207  4490
Total          97%  112372  0.084487  1330

Perf index = 58 (util) + 40 (thru) = 98/100
```