Multicore Programming Project 3

담당 교수 : 최재승 교수님

이름 : 김동현

학번 : 20190345

1. 개발 목표

사용자의 입력을 받아 malloc, free, realloc과 같은 힙 영역을 관리할 수 있는 dynamic memory allocator를 C언어로 구현하고자 한다. 메모리 관리 방법은 implicit list를, 할당될 메모리를 찾는 방법은 next-fit을 사용하였다.

2. 개발 범위 및 내용

dynamic memory allocator 를 구현하는 방법은 다음과 같이 나뉠 수 있다. 메모리 블럭을 관리하기 위해 implicit list, explicit list, segregated list 의 방법을 사용할 수 있다. 뿐만 아니라 할당할 수 있는 free block 을 찾기 위해, first-fit, next-fit, best-fit 의 방법을 사용할 수 있다. 본 구현에서는 implicit list 와 next-fit 을 사용하여 dynamic memory allocator 를 구현하였다.

여러개의 macro 함수와 전역변수 그리고, 총 8 개의 함수를 구현하였다.

2-1) MACROS

WSIZE 는 word 크기로 4 바이트 크기를, DSIZE 는 double-word 크기로 8 바이트이다. CHUNKSIZE 는 2 의 12 제곱으로 힙의 크기를 늘릴 때 사용한다.

MAX 는 두 인자 중 큰 인자를 반환하는 함수를 나타내는 메크로이다. PACK 은 size 와 alloc 을 or 연산을 통해서 합치는 메크로이다. 이때 사이즈는 8 바이트로 alignment 되어 size 의 low 3 bit 가 0 으로 확정되어 있다. 또한 alloc 은 0 이면 free 상태, 1 이면 allocated 된 상태로 마지막 low 비트에 저장된다. 이를 따로따로 저장하지 않고 하나의 워드에 저장하기 위해 두 정보를 or 연산을 통해 저장한다. GET 은 해당 포인터 주소에 저장된 데이터 값을 얻어오는 메크로이다. PUT 은 a 로 입력된 주소에 val 값을 저장하는 메크로이다. GET_SIZE 와 GET_ALLOC 은 PACK 으로 합쳐진 하나의 블록에서 각각 블럭의 크기와 할당 여부를 알아내는 메크로이다. HDRP 는 인자로 들어온 블록의 header 블록의 주소를, FTRP 는 인자로 들어온 블록의 footer 블록의 주소를 받아오는 메크로이다. NEXT_BLKP 는 해당 블록의 다음 블록의 주소를, PREV_BLKP 는 해당 블록의 이전 블록의 주소를 얻어오는 메크로이다.

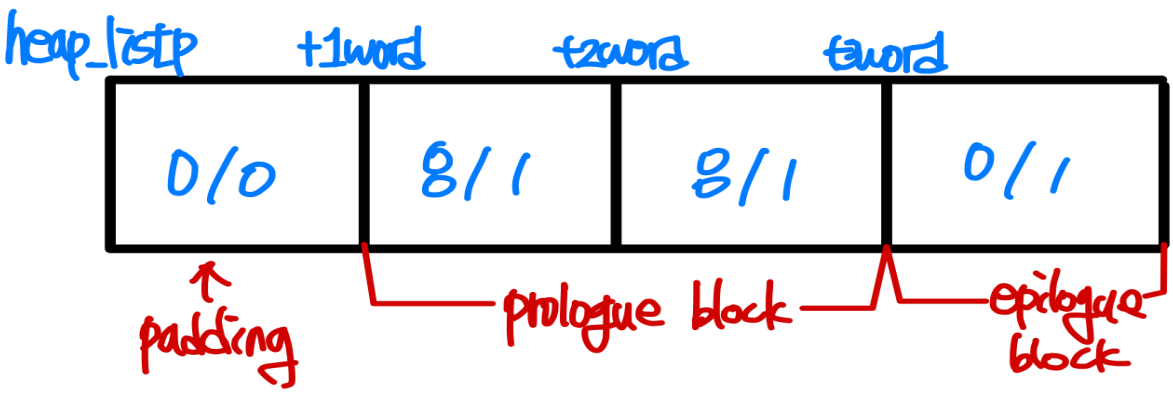
2-2) Global Variables

해당 코드에서는 2 개의 전역 변수를 사용했다. heap_list 는 첫번째 블록의 주소를 저장하고 있는 변수이다. 이를 메크로 함수를 이용하여 header 에 저장된 블록의

사이즈를 통해 다음 블록으로 연결리스트와 동일하게 traversal 할 수 있다. 두번째 전역 변수 rover 은 next fit 구현을 위한 변수이다. 해당 변수는 find_fit 을 통해 찾은 블럭의 위치를 계속해서 저장하여 다음으로 블록을 찾을 때, 해당 변수를 이용해서 이전에 할당한 블록의 다음 블록에서부터 traversal 을 진행한다.

2-3) function

mm_init()은 처음으로 heap 영역을 할당하여 메모리 할당을 진행할 수 있도록 초기화 해준다. 우선 heap_listp 에 4 word size 만큼의 공간을 할당한다. 처음으로 할당된 영역은 alignment 를 위해, 이후 payload 의 크기가 0 인 블럭을 prologue block 을 선언하여 힙의 시작부분을 나타낸다. 힙의 마지막 부분은 크기가 0 이면서 할당된 부분으로 나타내어 나타낸다. 이후 heap_listp 를 prologue block 의 payload 를 가리키도록 하기 위해 2 word size 를 더해준다.



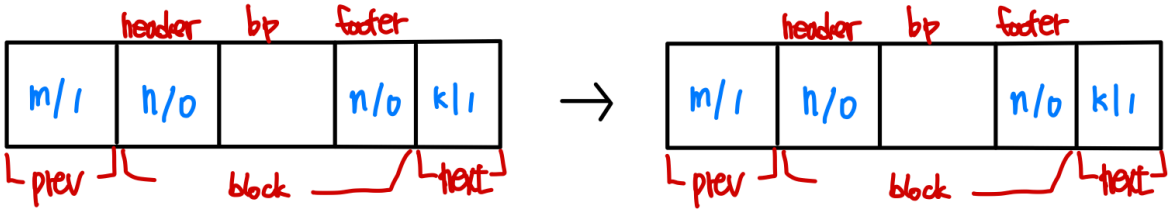
이후 extend_heap 을 호출하여 리스트 영역을 확보한다. 만약 이를 실패하면 초기화에 실패한 것으로 간주하여 -1 을, 그렇지 않다면 성공한 것으로 간주하여 0 을 반환한다.

mm_malloc()은 메모리 영역을 할당하는 함수이다. 우선 할당의 단위가 double word size 로 align 되어야 하기 때문에 이를 계산해 asize 에 저장한다. 이후 find_fit 함수를 호출하여 해당 크기만큼의 블럭이 있는 블록의 시작 주소를 반환받는다. 만약 NULL 값을 반환받는다면, 해당 크기만큼의 블럭이 없는 경우에는 extend_heap 함수를 호출해 힙 크기를 확장한다. 이후 place 함수를 호출해 해당 위치에 블록 영역을 할당한다.

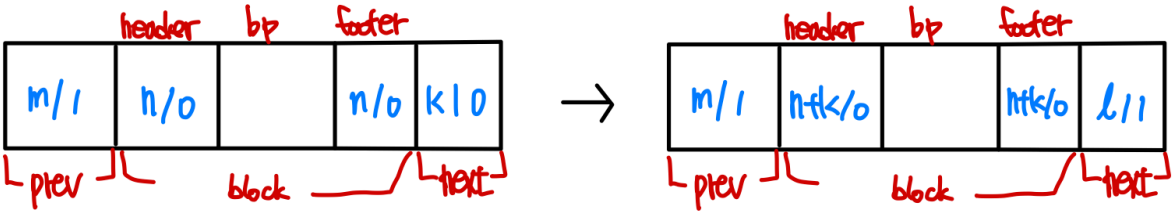
mm_free()는 메모리 영역을 반환하는 함수이다. 인자로 들어온 블럭의 시작 주소를 이용한다. 헤더에 저장된 블럭의 크기를 size 변수에 저장하고, 이 size 를 이용하여

header 와 footer 의 alloc 을 나타내는 가장 low 비트를 0 으로 변경한다. 이후 coalesce 함수를 호출하여 free 된 블럭 앞 뒤로, free 상태인 block 이 있는 경우 병합한다.

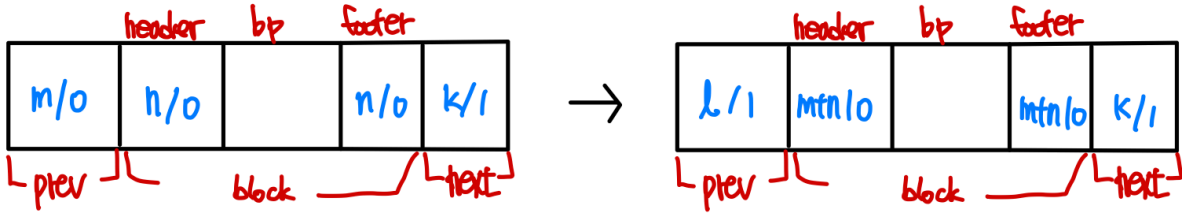
coalesce()함수는 free 가 된 블럭의 앞뒤로 free 되어 있는 블럭이 있다면 둘을 병합하는 함수이다. 총 4 가지 경우의 수가 있으며, 함수는 다음과 같이 동작한다. 우선 메크로 함수를 이용하여 이전 블럭의 할당 여부를 prev_alloc 과 next_alloc 에 저장한다. case1 번의 경우 앞뒤로 다 할당된 경우이므로(prev_alloc==1, nextalloc=1), 따로 병합해야할 과정이 없다.



case2번의 경우 뒤쪽 블럭이 free가 된 상태이므로(prev_alloc==1, nextalloc=0), 뒤 블럭과 병합을 한다. 뒤 블럭의 사이즈를 더하여 size 변수를 갱신하다. 이후 현재 블럭을 가리키는 포인터의 header, footer에 새로운 블럭의 크기를 저장해준다.

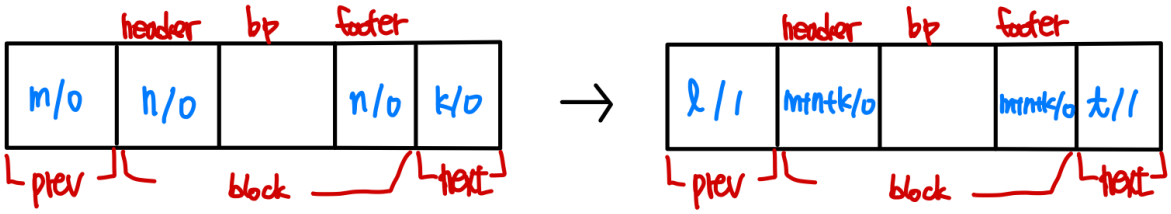


case3번의 경우 앞쪽 블럭이 free가 된 상태이므로(prev_alloc==0, nextalloc=1), 앞 블럭과 병합을 한다. 앞 블럭의 사이즈를 더하여 size 변수를 갱신하다. 이후 현재 블럭을 가리키는 포인터의 이전 블럭의 header, footer에 새로운 블럭의 크기를 저장해준다. 블럭을 가리키는 포인터는 이전 블럭을 가리키도록 한다.



case4번의 경우 앞뒤쪽 블럭이 free가 된 상태이므로(prev_alloc==0, nextalloc=0), 앞뒤 블럭과 병합을 한다. 앞과 뒤 블럭의 사이즈를 더하여 size 변수를 갱신하다. 이후 현재

블럭을 가리키는 포인터의 이전 블럭의 header, footer에 새로운 블럭의 크기를 저장해준다. 블럭을 가리키는 포인터는 이전 블럭을 가리키도록 한다.



이후 next-fit을 위한 rover 변수에도 coalesce을 통해 변경된 블럭포인터를 반영한다. 마지막으로 block pointer를 반환한다.

mm_realloc은 이미 할당되어 있는 메모리 영역의 크기를 변경하기 위한 함수이다. 재할당하고자 하는 크기가 0인 경우는 Free함수를 호출하여 메모리 영역을 반환해준다. 이후 새롭게 메모리 영역을 할당하여 새로운 메모리 영역으로 데이터를 복사해준다. 새롭게 할당된 주소를 반환한다. 이때 보다 작거나 같은 메모리를 재할당하고자 하는 경우에는 새롭게 블럭을 찾을 필요가 없어 해당 블록의 header와 footer의 정보를 변경하고 뒤에 블럭을 잘라 free 상태로 만들어 반환하여, 최적화를 진행하였다.

extend_heap()은 힙 영역을 확장하는 함수이다. alignment에 맞게 size를 저장한 뒤, sbrk 함수를 호출하여 힙 영역을 확장한다. 이후 해당 확장된 블럭을 free상태로 할당한 뒤, 힙의 끝을 나타내는 epilogue block을 새로 조정한다. 마지막으로 앞쪽 블럭이 free인 상태를 위해 coalesce를 수행한다.

place() 함수는 해당 위치에 블럭을 할당하고, 뒤에 free된 부분이 있다면 블럭을 쪼개주는 함수이다. 해당 블럭의 크기는 csize에 저장하고 할당하고자 하는 블럭과의 차이를 구해 diff_size에 저장한다. 만약 해당 크기가 2*DSIZE(header+footer)의 크기보다 작은 경우는 굳이 나눌 필요가 없기 때문에 해당 블럭 전체를 할당된 상태로 바꾼다. 그렇지 않은 경우 할당해야할 블럭의 크기만큼 할당한 뒤, 나머지 블럭을 free 상태로 쪼개준다.

find_fit()은 메모리에 블럭이 할당될 영역을 찾아주는 함수이다. 3가지 방법 모두 구현하였으며, 가장 성능이 좋은 next-fit을 이용해서 블럭을 찾는다. 나머지 2가지 방법은 주석

처리 하였다. next-fit은 전역 변수를 사용해서 블럭을 순회한다. rover는 블럭을 가리키는 포인터로 해당 블럭으로부터 순회하며 할당되지 않으면서, 크기가 충분한 블럭을 찾는다. 만약 끝까지 찾지 못한 경우, heap_listp가 처음을 가리키고 있기 때문에 처음으로 돌아가 계속해서 블럭을 찾는다. 만약 아무런 블럭도 맞는 경우가 없으면 NULL을 그렇지 않은 경우는 가장 먼저 찾은 블럭의 주소를 리턴한다.

3. 구현 결과

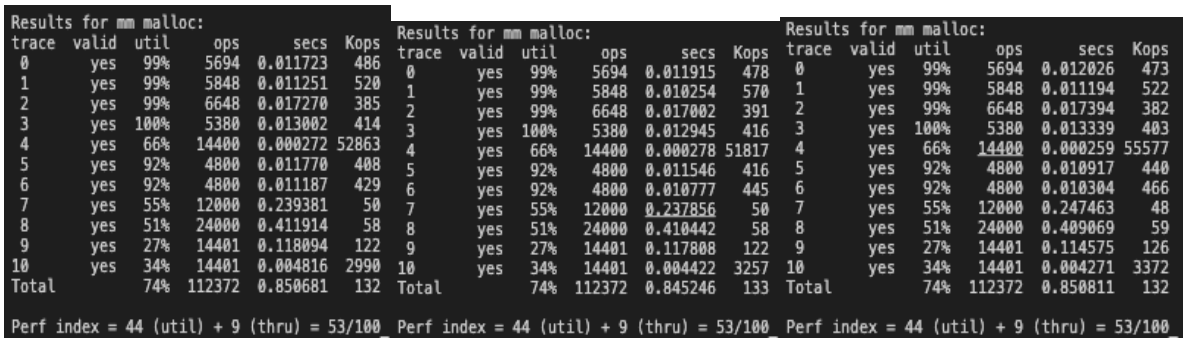
```
cse20190345@cspro:~/SP/proj3/prj3-malloc$ ./mdriver -V
[20190345]::NAME: Donghyun Kim, Email Address: kimdh0315@naver.com
Using default tracefiles in ./tracefiles/
Measuring performance with gettimeofday().

Testing mm malloc
Reading tracefile: amptjp-bal.rep
Checking mm_malloc for correctness, efficiency, and performance.
Reading tracefile: cccp-bal.rep
Checking mm_malloc for correctness, efficiency, and performance.
Reading tracefile: cp-decl-bal.rep
Checking mm_malloc for correctness, efficiency, and performance.
Reading tracefile: expr-bal.rep
Checking mm_malloc for correctness, efficiency, and performance.
Reading tracefile: coalescing-bal.rep
Checking mm_malloc for correctness, efficiency, and performance.
Reading tracefile: random-bal.rep
Checking mm_malloc for correctness, efficiency, and performance.
Reading tracefile: random2-bal.rep
Checking mm_malloc for correctness, efficiency, and performance.
Reading tracefile: binary-bal.rep
Checking mm_malloc for correctness, efficiency, and performance.
Reading tracefile: binary2-bal.rep
Checking mm_malloc for correctness, efficiency, and performance.
Reading tracefile: realloc-bal.rep
Checking mm_malloc for correctness, efficiency, and performance.
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   91%    5694  0.003600  1582
 1       yes   92%    5848  0.001660  3523
 2       yes   95%    6648  0.005238  1269
 3       yes   97%    5380  0.006137   877
 4       yes   66%   14400  0.000297 48469
 5       yes   91%    4800  0.006304   761
 6       yes   89%    4800  0.006682   718
 7       yes   55%   12000  0.026015   461
 8       yes   51%   24000  0.012208  1966
 9       yes   27%   14401  0.114423   126
10       yes   72%   14401  0.000194 74079
Total          75%  112372  0.182757   615

Perf index = 45 (util) + 40 (thru) = 85/100
```

implicit list, next-fit을 사용하여 dynamic memory allocator를 구현하였다. ./mdirver 명령어를 통해 성능을 측정하였다. 이때 -V 옵션을 주어 자세한 진행사항을 파악하였다. 총 점수는 100점 만점에 85점이 나왔으며, utilization점수는 45점 throughput 점수는 40점이 나와 총 85점이 나왔다.

4. 성능 평가 비교

free인 영역을 찾기위한 3가지 방법이 있다. first-fit, next-fit, best-fit이 있다. 해당 방법을 비교하여 implicit list와 어떤 방법이 가장 좋은 성능을 나타내는지 확인해보고자 한다.

first-fit은 블럭이 할당될 수 있는 가장 첫번째 블럭을 찾아서 반환하는 방법이다.



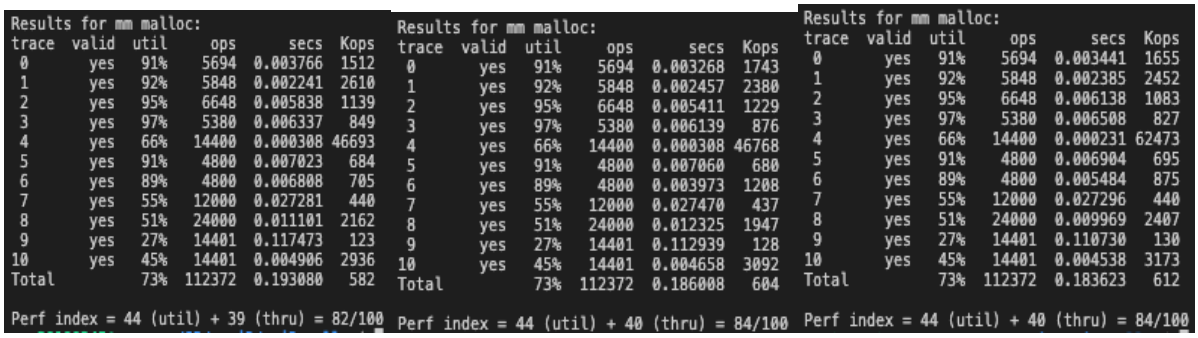
```
Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   99%    5694  0.011723   486
 1       yes   99%    5848  0.011251   520
 2       yes   99%    6648  0.017270   385
 3       yes  100%    5380  0.013002   414
 4       yes   66%   14400  0.000272 52863
 5       yes   92%    4800  0.011770   408
 6       yes   92%    4800  0.011187   429
 7       yes   55%   12000  0.239381    50
 8       yes   51%   24000  0.411914    58
 9       yes   27%   14401  0.118094   122
10       yes   34%   14401  0.004816  2990
Total          74%  112372  0.850681   132

Perf index = 44 (util) + 9 (thru) = 53/100
```

```
Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   99%    5694  0.011915   478
 1       yes   99%    5848  0.010254   570
 2       yes   99%    6648  0.017002   391
 3       yes  100%    5380  0.012945   416
 4       yes   66%   14400  0.000278 51817
 5       yes   92%    4800  0.011546   416
 6       yes   92%    4800  0.010777   445
 7       yes   55%   12000  0.237856    50
 8       yes   51%   24000  0.410442    58
 9       yes   27%   14401  0.117808   122
10       yes   34%   14401  0.004422  3257
Total          74%  112372  0.845246   133

Perf index = 44 (util) + 9 (thru) = 53/100
```

```
Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   99%    5694  0.012026   473
 1       yes   99%    5848  0.011194   522
 2       yes   99%    6648  0.017394   382
 3       yes  100%    5380  0.013339   403
 4       yes   66%   14400  0.000259 55577
 5       yes   92%    4800  0.010917   440
 6       yes   92%    4800  0.010304   466
 7       yes   55%   12000  0.247463    48
 8       yes   51%   24000  0.409069    59
 9       yes   27%   14401  0.114575   126
10       yes   34%   14401  0.004271  3372
Total          74%  112372  0.850811   132

Perf index = 44 (util) + 9 (thru) = 53/100
```

총 3번 실험을 진행하였고, 평균 53점이 나왔다.

next-fit은 블럭이 할당되고 해당 포인터를 저장하고 다음에 블럭을 찾을 때 해당 포인터로에서부터 찾는 방법이다.



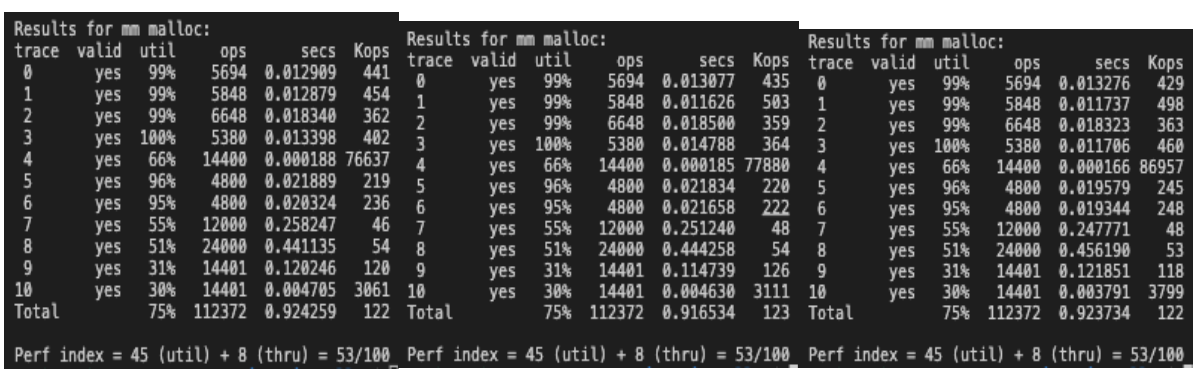
```
Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   91%    5694  0.003766  1512
 1       yes   92%    5848  0.002241  2610
 2       yes   95%    6648  0.005838  1139
 3       yes   97%    5380  0.006337   849
 4       yes   66%   14400  0.000308 46693
 5       yes   91%    4800  0.007023   684
 6       yes   89%    4800  0.006808   705
 7       yes   55%   12000  0.027281   440
 8       yes   51%   24000  0.011101  2162
 9       yes   27%   14401  0.117473   123
10       yes   45%   14401  0.004906  2936
Total          73%  112372  0.193080   582

Perf index = 44 (util) + 39 (thru) = 82/100
```

```
Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   91%    5694  0.003268  1743
 1       yes   92%    5848  0.002457  2380
 2       yes   95%    6648  0.005411  1229
 3       yes   97%    5380  0.006139   876
 4       yes   66%   14400  0.000308 46768
 5       yes   91%    4800  0.007060   680
 6       yes   89%    4800  0.003973  1208
 7       yes   55%   12000  0.027470   437
 8       yes   51%   24000  0.012325  1947
 9       yes   27%   14401  0.112939   128
10       yes   45%   14401  0.004658  3092
Total          73%  112372  0.186008   604

Perf index = 44 (util) + 40 (thru) = 84/100
```

```
Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   91%    5694  0.003441  1655
 1       yes   92%    5848  0.002385  2452
 2       yes   95%    6648  0.006138  1083
 3       yes   97%    5380  0.006508   827
 4       yes   66%   14400  0.000231 62473
 5       yes   91%    4800  0.006904   695
 6       yes   89%    4800  0.005484   875
 7       yes   55%   12000  0.027296   440
 8       yes   51%   24000  0.009969  2407
 9       yes   27%   14401  0.110730   130
10       yes   45%   14401  0.004538  3173
Total          73%  112372  0.183623   612

Perf index = 44 (util) + 40 (thru) = 84/100
```

총 3번 실험을 진행하였고, 평균 83.3점이 나왔다.

best-fit은 블럭이 가장 효율적인 곳에 할당되기 위해 모든 리스트를 순회하여 찾는 방법이다.



```
Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   99%    5694  0.012909   441
 1       yes   99%    5848  0.012879   454
 2       yes   99%    6648  0.018340   362
 3       yes  100%    5380  0.013398   402
 4       yes   66%   14400  0.000188 76637
 5       yes   96%    4800  0.021889   219
 6       yes   95%    4800  0.020324   236
 7       yes   55%   12000  0.258247    46
 8       yes   51%   24000  0.441135    54
 9       yes   31%   14401  0.120246   120
10       yes   30%   14401  0.004705  3061
Total          75%  112372  0.924259   122

Perf index = 45 (util) + 8 (thru) = 53/100
```

```
Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   99%    5694  0.013077   435
 1       yes   99%    5848  0.011626   503
 2       yes   99%    6648  0.018500   359
 3       yes  100%    5380  0.014788   364
 4       yes   66%   14400  0.000185 77880
 5       yes   96%    4800  0.021834   220
 6       yes   95%    4800  0.021658   222
 7       yes   55%   12000  0.251240    48
 8       yes   51%   24000  0.444258    54
 9       yes   31%   14401  0.114739   126
10       yes   30%   14401  0.004630  3111
Total          75%  112372  0.916534   123

Perf index = 45 (util) + 8 (thru) = 53/100
```

```
Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   99%    5694  0.013276   429
 1       yes   99%    5848  0.011737   498
 2       yes   99%    6648  0.018323   363
 3       yes  100%    5380  0.011706   460
 4       yes   66%   14400  0.000166 86957
 5       yes   96%    4800  0.019579   245
 6       yes   95%    4800  0.019344   248
 7       yes   55%   12000  0.247771    48
 8       yes   51%   24000  0.456190    53
 9       yes   31%   14401  0.121851   118
10       yes   30%   14401  0.003791  3799
Total          75%  112372  0.923734   122

Perf index = 45 (util) + 8 (thru) = 53/100
```

총 3번 실험을 진행하였고, 평균 53점이 나왔다.

위 3가지 방법 중 implicit list에 효과적인 방법은 next fit이다. first-fit은 계속해서 처음부터 노드를 순회하다보니 중복된 순회를 계속하여 속도가 느릴 뿐만 아니라, 가장 처음으로 발견된 블럭을 선택하다보니 효율적인 배치를 기대할 수 없다. next-fit은 처음부터가 아닌 이전에 할당된 이후부터 순회하여 중복된 순회를 줄일 수 있어 높은 속도를 기대할 수 있다. 마지막으로 best-fit은 모든 블럭을 순회하며 가장 효율적인 블럭을 찾는다. 이로인해 util점수는 first-fit에 비해 근소하게 높을 수 있지만, 모든 노드를 순회한다는 점에서 속도 저하를 불러온다.