Multicore Programming Project 3

담당 교수 : 최재승 교수님

이름 : 손채훈

학번 : 20181085

1. 개발 목표

교재 코드를 기반으로, 보다 개선된 throughput과 utilization을 보여주는 코드를 만드는 것이 이번 프로젝트의 개발 목표이다. 11개의 테스트 케이스(9개의 malloc과 2개의 realloc으로 구성된)를 통해 throughput 40점, utilization 60점, 총점 100점으로 코드의 성능을 측정한다. 한가지 metric에 편중되지 않고 두가지 metric 모두를 균형 있게 보완된 코드를 작성하는 게 이번 프로젝트의 주안점이다.

2. 개발 내용

A. 개발 내용

- **Explicit list 구현**

먼저 malloc시의 throughput을 개선하기 위해서 기존의 Implicit list에서 Explicit list로 변경하였다. Explicit list를 구현하기 위해서 기존의 mm_init 함수에서 mem_sbrk 함수를 통해서 총 4개의 word size 블록을 할당 받는데, 나는 총 6개의 블록을 할당 받았다. 그리고 기존의 prologue block 사이즈를 16으로 해주고 header와 footer를 제외한 가운데 두 블록을 Free block들을 저장하기 위한 더미 노드로 사용하였다. 4번의 그림을 참조하면 이해가 편할 것이다. 그리고 insert 함수와 delete함수를 통해서 Free block list를 관리하도록 하였다. Explicit list 구현을 위해서 insert 함수와 delete 함수를 추가하였는데, insert 함수에서는 더미 노드의 바로 뒤에 현재 블록을 insert해주고, delete 함수에서는 입력으로 받은 포인터 값이 가리키는 블록을 바로 링크드 리스트에서 제거해준다.

- **Best fit 구현**

Best fit을 구현하기 위해서 기존의 first fit으로 구현되어있는 find_fit함수를 수정하였다. 기존의 함수에서는 사용자가 요청한 블록 사이즈에 double-word size(header + footer)를 더하고 그 값을 double-word size로 align한 값 보다 free block의 크기가 크거나 같으면 바로 그 free block의 주소를 return해 주었는데, best fit 구현해서는 free block list를 전부 순회한 후에 가장 적합한 블록을 가리키는 포인터를 return해 준다.

- **공간 효율성을 높인 realloc 함수 구현**

  기존의 realloc 함수에서는 공간 효율성을 고려하지 않고 새롭게 mm_malloc함수의 호출을 통해 추가적인 공간을 할당받고, 그 전 공간에 있던 값들을 memcpy 함수를 통해서 메모리 복사해준 후에 새로운 mm_malloc의 리턴 값으로 받은 포인터 값을 return 해준다. 이는 코드의 utilization을 가장 심각하게 저해하는 요인 중 하나이다. 이를 개선하기 위해 이미 메모리 할당된 블록 뒤에 free block이 존재하고, 이게 요청 받은 사이즈를 수용하기에 충분하다면, 이미 할당된 블록과 뒤의 free block을 합친 후 입력으로 받았던 포인터를 return 해준다. 만약 그렇지 않을 때만 새로이 mm_malloc함수를 통해 공간을 할당 받은 뒤, 새로운 포인터를 return 해준다.

B. 개발 방법

- **Explicit list 구현**

  insert 함수에서는 2가지 케이스로 나누어서 Free block들을 링크드 리스트로 연결해주었다. 첫번째 경우는 기존의 더미 노드에 block들이 하나도 연결되지 않았던 경우이다. 이때는 그냥 더미 노드의 뒤에 바로 연결해주고 현재 추가된 블록의 next pointer는 null값을 가리키도록 하였다. 두번째 경우는 기존의 더미 노드에 블록이 하나 이상 연결된 경우이다. 이때는 현재 블록이 더미 노드의 바로 뒤에 연결되도록 변경해주고 이 위치에 있던 기존 블록이 현재 블록의 바로 다음에 오도록 포인터 연산을 해준다. 이를 통해 O(1)의 시간 복잡도로 block의 삽입이 가능하다.

  delete 함수에서도 역시 2가지 케이스로 나누었다. 첫번째 경우에는 인자로 받은 블록 포인터가 free block 리스트의 마지막 블록인 경우이다. 이때는 마지막 블록의 이전 블록이 next pointer가 null을 가리키도록 한다. 두번째 경우에는 delete하는 블록을 제외한 그 블록의 prev 블록과 next 블록끼리 연결시켜 준다. 역시 O(1)의 시간 복잡도로 block의 deletion이 이루어진다.

- **Best fit 구현**

  best fit을 구현하기 위해서 기존의 find_fit 함수를 수정하였다. free block 리스

트의 더미 노드의 다음 노드부터 순회하면서, 할당 요청 받은 블록사이즈와 크기 차이가 가장 적은 블록을 찾아서 리턴해준다. best fit 방식은 throughput을 상당히 저해하는 요인이기는 하지만, utilization의 개선에는 매우 유의미한 성능을 보여준다. best fit 방식은 free block list의 마지막 블록 까지 순회하기 때문에, O(n)의 시간 복잡도를 보여준다.

- **공간 효율성을 높인 realloc 함수 구현**

  realloc 함수의 수정을 위해 첫 번째로 한 것은 입력으로 들어온 사이즈를 실제 우리가 할당해줄 블록의 사이즈로 align해주는 작업이다. 기존의 mm_malloc 함수와 동일한 방식으로 align해 주었는데, 사용자가 요청한 블록의 크기가 double word size보다 작으면 최소 블록 크기인 16 바이트로 align하였고, 그게 아니라면 요청한 블록크기 + double word size(header + footer) 보다 큰 8의 배수 중 가장 작은 8의 배수로 align 해주었다. 이 align된 값을 asize라고 하였고, asize가 기존의 할당된 블록크기보다 작거나 같으면 함수의 첫번째 인자로 받은 포인터 값을 그대로 return 해주었다. 현재 할당된 블록의 크기가 asize보다 작은 경우에 현재 할당된 블록 다음 블록이 epilogue 블록이라면, 현재 할당된 블록 + 다음 free block의 크기가 asize보다 크거나 같을 때 까지 64바이트 만큼 힙 사이즈를 extend_heap 함수를 통해 증가시켜주었다. 그 후 현재 할당된 블록 다음 블록이 free block이며, 현재 할당된 블록의 크기 + 다음 free 블록의 크기가 asize보다 크거나 같다면 두 블록을 합쳐준 후 allocate bit를 1로 설정해주었다. 만일 그렇지 않다면 기존의 mm_realloc 함수와 동일하게 새로이 mm_malloc을 호출하여 메모리를 할당받고 새로운 포인터 값을 new_ptr 변수에 return 받는다. (이 경우를 제외하고 나머지 경우에는 기존 인자로 받은 포인터값을 그대로 반환해준다.) 그 후 새로운 블록에 memcpy를 통해 기존 할당된 블록의 값을 복사한 후, 기존 할당된 블록의 allocate bit를 0으로 바꿔주고 free block list에 coalesce 해준다. 마지막으로 block이 새로이 할당된 곳의 포인터를 return 해준다.

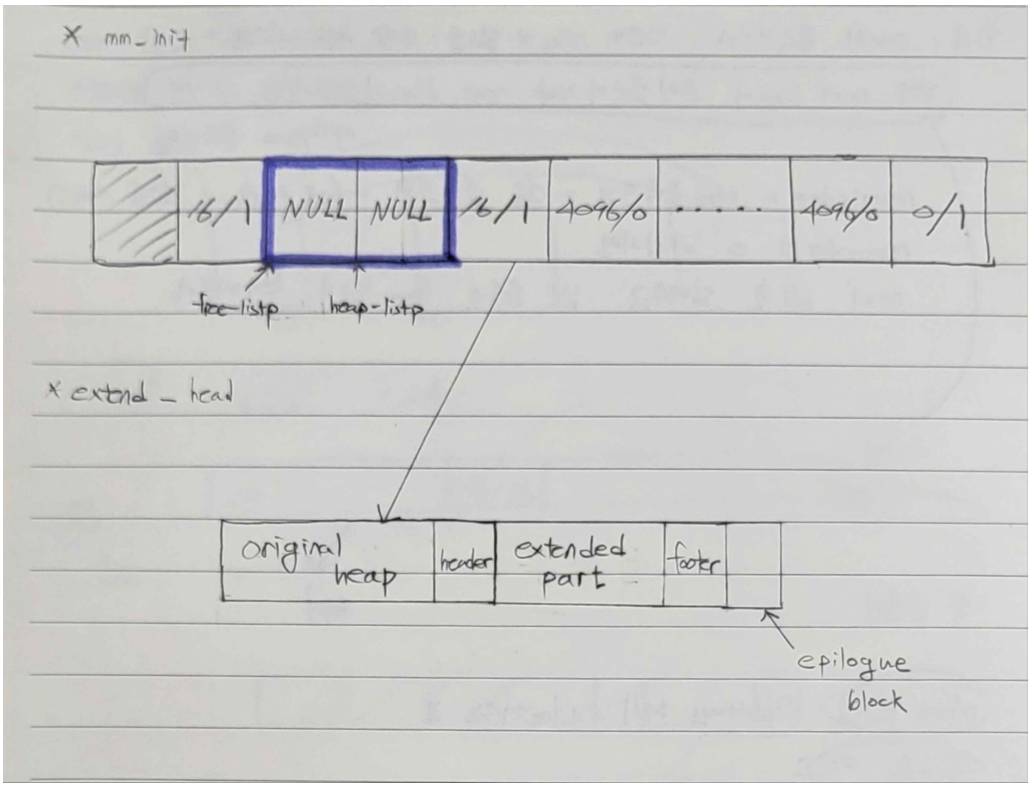
3. **구현 결과**

```
Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   99%    5694  0.010094   564
 1       yes   99%    5848  0.009194   636
 2       yes   99%    6648  0.019089   348
 3       yes   99%    5380  0.014300   376
 4       yes   99%   14400  0.000216 66605
 5       yes   95%    4800  0.003040  1579
 6       yes   95%    4800  0.002757  1741
 7       yes   95%   12000  0.018515   648
 8       yes   88%   24000  0.014823  1619
 9       yes   99%   14401  0.000264 54632
10       yes  100%   14401  0.000135106912
Total           97%  112372  0.092426  1216

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

위 그림은 mm.c 파일을 구현한 후에 mdriver 실행 파일을 -v 옵션으로 실행 시켰을 때의 결과이다.

4. 코드 내용 시각화



위의 그림은 처음 mm_init 함수가 실행되고 난 직후의 힙의 상태이다. free_listp 포인터가 가리키고 있는 NULL 값이 할당되어있는 두개의 word block이 free list block들을 관리하는 링크드 리스트의 더미 노드로 사용된다. 힙이 맨 처음 chunksize로 extend 된다고 할 때, 원래 초기 힙의 epilogue block의 자리가 확장된 free list block의 헤더로 덮어쓰여진다. 그리고 extend_heap 함수에서 확장된 부분의 마지막에 epilogue block을 덧붙여주고 이 블록이 역시 다음 힙 확장시 확장된 free block의 헤더로 덮어쓰여진다.