시스템프로그래밍 Lab3

자유전공학부
2018-16371 문보설

a. 실행결과

```
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   99%    5694  0.001964  2899
 1       yes   99%    5848  0.001728  3384
 2       yes   99%    6648  0.002688  2474
 3       yes   99%    5380  0.002532  2125
 4       yes   89%   14400  0.001516  9496
 5       yes   95%    4800  0.004181  1148
 6       yes   94%    4800  0.004405  1090
 7       yes   95%   12000  0.002837  4230
 8       yes   88%   24000  0.003329  7209
 9       yes   91%   14401  0.001890  7620
10       yes   51%   14401  0.000685 21011
Total          91%  112372  0.027757  4048

Perf index = 55 (util) + 40 (thru) = 95/100
2018-16371@sp4:~/malloclab/src$
```

b. 구현 방법

<참고사항>

- 본 과제는 Segregated list + Best fit policy + Immediate coalescing policy로 구현 되었다.
- 메모리 블락 구조

<allocated block>

(byte)

0 1 2 3 | ..... n| n+1 n+2 n+3 n+4

HEADER| <---------- Payload ----------->| FOOTER

|-> this is where the block pointer is allocated

<freed block>

0 1 2 3 | 5 6 7 8| 9 10 11 12|..... n| n+1 n+2 n+3 n+4

HEADER | PREV | NEXT |<----------freed --------->| FOOTER

|-> this is where the block pointer is allocated

- PREV : previous freed block on segregated list(4byte)
- NEXT : next freed block on segregated list (4byte)
- HEADER : includes allocated bit and the real size of block (4byte)
- FOOTER : includes allocated bit and the real size of block (4byte)

- 아래 설명에서 메모리 할당이란 header와 footer의 정보를 할당된 메모리로 바꾸고 allocated 비트를 1로 바꾸는 것을 말한다.
- 아래 설명에서 메모리 해제란 allocated 비트를 0으로 바꾸는 것을 말한다.

1. mm_init
   이 함수 안에서는 전역적으로 사용할 변수들을 초기화했다. segregated list로 구현했기 때문에 각 리스트를 초기화하고, heap의 시작점을 초기화한다. 또한 free block에 allocation을 하는 위치를 결정하는 threshold를 설정하기 위한 변수들을 초기화한다. 참고로, threshold보다 작은 메모리를 할당할 때는 free block의 앞부분에, threshold보다 큰 메모리를 할당할 때는 뒷부분에 할당한다. 이 때 threshold가 지금까지 alloc된 사이즈들의 평균값으로 결정되기 때문에 이 평균을 계산하는 변수들 역시 mm_init에서 초기화해야 하는 것이다.
2. mm_malloc
   이 함수에서는 요청된 메모리 할당과 관련한 작업을 처리한다. 먼저 앞서 설명한 threshold에 요청된 크기만큼을 반영하여 평균을 내는 작업을 수행한다. 그리고 best_fit 방식으로 할당할 크기에 맞는 freed block이 있는지 확인한다. 그러한 block이 있는 경우 해당 block에 메모리를 할당하고, 아닌 경우 힙을 확장하여 메모리를 할당한다.
3. mm_free

이 함수는 요청된 메모리를 해제하는 작업을 처리한다. 먼저 메모리를 해제하고 앞 뒤로 free block이 있는 경우 해당 free block과 방금 해제한 block을 합친다(coalescing). throughput이 크게 나쁘지 않았기 때문에 해제와 동시에 coalescing 하는 방식을 택했다. 이러한 과정의 결과로 생긴 free block을 적합한 segregate list에 삽입한다.

4. mm_realloc

이 함수는 이미 할당된 메모리의 사이즈를 바꾸는 작업을 처리한다. 만약 요청된 사이즈가 0이면 free에 해당하므로 mm_free를 호출한다. 새로운 사이즈가 0보다 작은 경우에는 에러에 해당하므로 아무런 작업 없이 NULL을 리턴한다. 새로운 사이즈가 0보다 크지만 기존 사이즈보다 작은 경우, 사이즈를 다시 조정할 필요가 없으므로 인자로 받은 포인터 그대로 리턴한다. 앞의 경우들에 해당하지 않는 경우, 3가지 케이스가 있다. 첫번째 케이스는 재조정해야 할 메모리 블락과 그 바로 다음 블락을 합쳤을 때 새로운 사이즈를 수용할 수 있는 경우이다. 이 경우 재조정해야 할 메모리 블락과 그 다음 블락을 합쳐서 메모리를 재할당하고, 만약 합친 블락이 새로운 사이즈를 담고도 충분히 남을 정도로 여유롭다면 남은 사이즈만큼은 free block으로 간주하여 해제하고 list에 추가한다. 두번째 케이스는 재조정해야 할 메모리 블락과 그 바로 다음 블락, 그 바로 앞 블락을 합쳤을 때 새로운 사이즈를 수용할 수 있는 경우이다. 이 역시 마찬가지로 세개의 블락을 합쳐서 메모리를 재할당하고, 합친 블락이 충분히 여유로울 경우 남은 사이즈를 free 블락으로 간주하여 해제하고 list에 추가한다. 마지막 케이스는 앞의 어느 경우에도 해당하지 않는 경우로, 필요한 사이즈의 블락을 mm_malloc을 통해 새로 할당하고 기존 내용을 복사한 뒤 기존 블락을 해제해야 한다.

c. 어려웠던 점

1. 힙 구조

처음에 교과서를 읽지 않아 에필로그 헤더와 프롤로그 블락 등을 잡지 않고 자의적으로 메모리를 구성했기 때문에 어려움을 겪었다. 교과서를 잘 읽고 기본 구조를 교과서와 같이 잡고 시작하니 훨씬 수월하게 과제를 마무리할 수 있었다.

2. allocation policy

Allocation policy를 정하는 것이 고민되었지만, 메모리 활용에 중점을 맞췄다. 처음에 implicit list를 기본 구조로 잡고 best/first/next fit을 구현했다. 그 결과 총점은 next fit이 가장 높게 나타났으나, utilization만 놓고 보자면 best fit을 사용하는 것이 더 좋았다. Next fit이 best fit보다 보통 utilization이 낮고 throughtput이 좋다는 것을 배웠기 때문에 당황스러워서 테스트 케이스를 분석했다. 그 결과 next fit의 총점이 좋은 이유는 주어진 테스트 케이스가 비슷한 블락들을 연속적으로 요청하는, 즉 next fit의 utilization에 유리한 테스트 케이스기 때문이라는 판단을 내렸다. 당장 점수가 좋아서 next fit으로 마무리할 생각도 했지만, 이 경우 여러 테스트케이스에 범용적으로 사용할 수 없다는 치명적인 단점이 있었다. 그리고 throughput의 경우 free list를 다른 식으로 변경하여 충분히 개선할 수 있다고 생각했기 때문에 best fit을 선택하고 list를 재구성하기로 했다.

3. list 구조

list는 implicit, segregate, explicit 세가지 구현 법이 있다. 이 중 하나를 결정하는 것도 고민

했던 점 중 하나인데, 앞서 설명했듯 throughtput을 극대화하는 것에 초점을 맞춰 segregated list로 구현했다. Implicit 구현은 best fit을 찾을 때 모든 블락을 순회해야 하므로 제외했고, segregated list가 사이즈별로 리스트를 관리하기 때문에 explicit보다 best fit block을 찾는 시간이 더 적게 걸릴 것이라 판단하여 이 방법을 선택했다.

d. 새로웠던 점

최적화를 하는 과정에 대해서 알 수 있는 과제였다. 먼저 기본적인 구조를 잡고, 성능에 trade-off(이 경우 utilization와 throughput)가 있는 경우 각 policy가 개선하고자 하는 성능을 명확히 하여 두 factor 모두 만족시키는 결과를 얻기 위해 노력해야 한다.