|  |  |
| --- | --- |
| CS 341 | Lecture #9 Building an allocator |

1. The following allocator will use this linked list structure:

|  |
| --- |
| 1. typedef struct \_metadata\_entry\_t { 2. void \***ptr**; 3. int **size**; 4. int **free**; //0(in use) or 1(available) 5. struct \_metadata\_entry\_t \***next**; 6. } metadata\_entry\_t; |

*Global variable:*

|  |
| --- |
| 1. static metadata\_entry\_t \* head = NULL; |

*2. Complete malloc()*

|  |
| --- |
| 1. void \***malloc**(size\_t size) { 2. /\* See if we have free space of enough size. \*/ 3. metadata\_entry\_t \*p = head; 4. metadata\_entry\_t \*chosen = NULL; 5. while (p != NULL) { 6. if (p->free && \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_) { 7. if (chosen == NULL || (chosen && p->size < chosen->size)) { 8. chosen = p; 9. } 10. } 11. p = p->next; 12. } 13. if (chosen) { 14. chosen->free = 0; 15. return chosen->ptr; 16. } 17. /\* Add our entry to the metadata \*/ 18. chosen = sbrk(0); 19. sbrk(sizeof(metadata\_entry\_t)); 20. chosen->ptr = sbrk(0); 21. if (sbrk(size) == (void\*)-1) { 22. return NULL; 23. } 24. chosen->size = size; 25. chosen->free = 0; 26. chosen->next = head; 27. head = chosen; 28. return chosen->ptr; 29. } |

3. Complete free()

1. void free(void \*ptr) {
2. if (!ptr) return;
3. metadata\_entry\_t \*p = \_\_\_\_\_\_\_\_\_\_\_\_\_
4. while (p) {
5. if (p->ptr == ptr) {
7. }
8. p = p->next;
9. }
10. return;
11. }

Which placement algorithm does this malloc( )use?  
Is calling sbrk 4 tims necessary?   
What is the order of growth running time for this implementation of free?

4 i) Why does this implementation suffer from false fragmentation?  
 ii) When should we split blocks?

iii) Does this implementation use an explicit or implicit linked list?

5. How would you change malloc() to use a *first-fit* placement allocation?

1. while (p != NULL) {
2. if (p->free && \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_) {
3. if (chosen == NULL || (chosen && p->size < chosen->size)) {
4. chosen = p;
5. }
6. }
7. p = p->next;
8. }

8. Towards a better allocator

Implementing realloc & improving performance of free()

Hint: Can we ensure this structure is immediately before the user's pointer?

1. typedef struct \_metadata\_entry\_t {
2. void \*ptr;
3. int size;
4. int free;
5. struct \_metadata\_entry\_t \***next**;
6. } metadata\_entry\_t;

We want an O(1) deallocator!

1. void free(void\*user) {
2. if(user == NULL) return; // No-op
3. ?

End of the allocator challenge?

1. Block Spitting & Block Coalescing  
2. Memory pools  
3. Advanced: Slab allocator and Buddy allocator  
4. Internal vs External Fragmentation  
5. How we use Boundary Tags to implement coalescing?

9. Puzzle:

Complete this code to read in values from stdin into heap memory. Can you beat CS225 code by using C and realloc to increase the size of the array?  
Fix any errors you notice.

|  |
| --- |
| 1. #define quit(mesg) {puts(mesg); exit(1);} 2. size\_t capacity = 256; 3. size\_t count = 0; 4. int\* data = malloc( capacity ); 5. if( ! data ) quit("Out of memory"); 6. while( !feof(stdin) && !ferror(stdin)) { 7. if( count == capacity) { 8. capacity \*= 2;      1. } 2. if( fscanf(stdin, "%d", data+count) != 1) break; 3. count++; 4. } 5. // can now reduce capacity to the number actually read 6. printf("%d values read",(int) count); 7. data = realloc(data, count); |