|  |  |
| --- | --- |
| CS 341 | Lecture #10 Building an allocator II |

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

|  |
| --- |
| 1. typedef struct \_metadata\_entry\_t { 2. void \***ptr**; 3. size\_t **size**; 4. int **free**; //0(in use) or 1(available) 5. struct \_metadata\_entry\_t \***next**; 6. } entry\_t; 7. static entry\_t\* head = NULL; |

2. Implement an efficient realloc to avoid memory copying when possible?

Assume the above entry\_t structure is immediately before the user's pointer

1. void\* realloc(void \*old, size\_t newsize) {

3. Instrumenting malloc  
Case study: Fragmentation & Memory overhead & utilization?

How can we modify our malloc implementation so that we write an instrumentation function below to print how efficient our memory allocator is?

“123456 bytes allocated. 280 byte overhead. 352 unavailable bytes in 6 fragments”

1. void printMallocStats() {

4. Memory alignment and BUS Signals?   
… aka why malloc writers care about CPUs

… what is natural alignment?

… How can we round up allocations to nearest 16 bytes?

5. Block Coalescing & Thinking in sizeof(size\_t) blocks…

Goodbye bytes. Memory = one big “array” of size\_t entries

Use Knuth Boundary Tags:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 100 | .................... | 100 | 64 | ....... | 64 | 132 | ............................... | 132 |

malloc(size\_t request\_bytes){

int request\_blocks = request\_bytes / 8? Is this good?

// enough space -

void\* ptr = sbrk(

6. Implementing Canaries

How (and when can we detect buffer overflow/ underflow using boundary tags? Are there other canaries?

7. Fast Memory pools

static char buffer[10000];

size\_t used=0;

void\* malloc(size\_bytes) {

}

void free\_all\_the\_things() {

}

8. How can I beat malloc?

a) Efficiency of representation

b) Speed of allocation

c) Speed of "recycling"

d) Utilization of memory