CS 341

#8 a whole *heap* of trouble
malloc, calloc, realloc

#1 Review:

Why put the heap so far away from the stack?

What will you find below the end of the stack and above the top of the heap?

nothing, except some dynamic allocated libs.

#2 What value will be printed?

```
int a = 10;
int* ptr = &a;
pid_t child = fork();        copy on write
if(child == 0) { * ptr = 20; ptr = NULL;}
else {
   waitpid(child, NULL,0);
   printf("%d", * ptr );  10
}
```

Initially, heap has 0 byte.

#3 What does sbrk do?

"sbrk increases the process's data segment by n bytes"
... but what does this mean?

Just a ptr to some memory, can be any type

#4 A very simple heap memory allocator

```
void* malloc(unsigned int numbytes) {     -> return address
  printf("Top of heap was %p\n", sbrk(0) ); // safe??

  void* ptr = sbrk(numbytes);
  if(ptr == (void*) -1) return NULL; // no mem for you!

  printf("Now you have some mem at %p\n",ptr );
    -> uses malloc. So we write
  return ptr;
}
                          bad
void free(void*mem) { }
```

What are the limitations of the above allocator?

How can we improve it?

#5 How do I use calloc?

useful when we want zeroed memories

```
void*  calloc(size_t count, size_t size);
```

```
double * result
result = calloc( 100, sizeof(double))
```

#6 Implement your own calloc using memset and malloc:

```
// void *   memset(void *b, int c, size_t len);

void* mycalloc(size_t count, size_t size) {
   size_t total = count * size  -> overflow? TODO: check overflow
   void* result = malloc( total )
      if (!result) return NULL;
   memset(result, 0, total);
   return result
```

behaves like this, but kernel won't use this.

#7 How does I use realloc?

may move stuff!!!
if NULL, same as malloc

```
void *  realloc(void *oldptr, size_t size);
```

```
int capacity = 0, n = 0 ;
double * data = NULL;
while (...) {        if cap = 0, -> 1024
                     else cap = cap*2
  double v = ..
  if (n == capacity) { cap = cap ? cap*2 : cap = 1024
    data = realloc(data, cap * sizeof(double))
  }
  data[n++] = v
```

Placement Strategies - Best Fit. Worst Fit. First Fit Allocation

Suppose the heap is managed with a linked list. Each node in the list is either allocated or free. The list is sorted by address. When **`malloc()`** is called, the list is searched for a free segment that is big enough (depending on the allocation algorithm), that segment is divided into an allocated segment (at the beginning) and a free segment. When **`free()`** is called, the corresponding segment should merge with its neighboring segments, if they are also free. A process has a heap of 13KB, which is initially unallocated. During its execution, the process issues the following memory allocate/de-allocate calls (**`pA... pE`** are **`void*`** pointers). In all cases, break ties by choosing the earliest segment. Also, assume all algorithms allocate memory from the beginning of the free segment they choose.

```
pA = malloc(3KB)

pB = malloc(4KB)

pC = malloc(3KB)

free(pB)

pD = malloc(3KB)

free(pA)

pE = malloc(1KB)
```

For simplicity, assume the memory begins at address 0, and ignore the memory used by the linked list itself. Show the heap allocation after the above calls, using best-fit, worst-fit and first-fit algorithms respectively.

Best Fit:

0K	1K	2K	3K	4K	5K	6K	7K	8K	9K	10K	11K	12K

PE

Starting address of pD= ____ K and pE = _____ K

Worst Fit: Choose the longest space

0K	1K	2K	3K	4K	5K	6K	7K	8K	9K	10K	11K	12K

Starting address of pD = ____ K and pE = _____ K

First Fit: Once large enough, use that space

0K	1K	2K	3K	4K	5K	6K	7K	8K	9K	10K	11K	12K

Starting address of pD = ____ K and pE = _____ K

Spaces not continuous anymore

What is Fragmentation? What happens if heap memory is severely fragmented?

Best Fit outcome? Lots of tiny unusable space.

Worst Fit outcome? Eats largest space → Median areas.

First Fit outcome? Winner!