|  |  |
| --- | --- |
| CS 241 | #8 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?

#2 What value will be printed?

|  |
| --- |
| 1. int a = 10; 2. int\* ptr = &a; 3. pid\_t child = fork(); 4. if(child == 0) { \* ptr = 20; ptr = NULL;} 5. else { 6. waitpid(child, NULL,0); 7. printf("%d", \* ptr ); 8. } |

#3 What does sbrk do?

"sbrk increases the process's data segment by n bytes"

... but what does this mean?

#4 A very simple heap memory allocator

|  |
| --- |
| 1. void\* malloc(unsigned int numbytes) { 2. printf("Top of heap was %p\n", sbrk(0) ); 3. void\* ptr = sbrk(numbytes); 4. if(ptr == (void\*) -1) return NULL; // no mem for you! 5. printf("Now you have some mem at %p\n",ptr ); 6. return ptr; 7. } 8. void free(void\*mem) { } |

What are the limitations of the above allocator?

How can we improve it?

#5 How do I use calloc?

void \* calloc(size\_t count, size\_t size);

#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) {

#7 How does I use realloc?

void \* realloc(void \*oldptr, size\_t size);

**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 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

Starting address of pD= \_\_\_\_ K and pE = \_\_\_\_\_ K

***Worst Fit:***

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

Starting address of pD = \_\_\_\_ K and pE = \_\_\_\_\_ K

***First Fit:***

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

Starting address of pD = \_\_\_\_ K and pE = \_\_\_\_\_ K

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

**Best Fit outcome?**

**Worst Fit outcome?**

**First Fit outcome?**