CS 340
Computer Systems

#6: Heap Memory Allocation and malloc

Sept. 8, 2022 · Wade Fagen-Ulmschneider

With a virtual memory system:

- Can we meet all of the allocation requests?
- Are we limited to just RAM?

Simple Simulation of Page Tables with Disk Pages

RAM:	P1 Page Table:	Disk Pages:	
[0]:	[0]:	...	1: Load Program
[1]:	[1]:	./programCode (1/5)	
[2]:	[2]:	./programCode (2/5)	2: Run PC, pg1: - malloc(4000)
[3]:	[3]:	./programCode (3/5)	
	[4]:	./programCode (4/5)	3: Run PC, pg2: - malloc(10000) - Open hiddenImage.png - Read all of image
	[5]:	./programCode (5/5)	
	[6]:		
	[7]:		
	[8]:		
	[9]:		
	[10]:		
	[11]:	hiddenImage.png	4: Run PC, pg3: - Access OG 4 KB - Finish program
	[12]:	hiddenImage.png	
	[13]:	hiddenImage.png	
	[14]:		
	[15]:	...	

Q1: What is the range of possible file sizes for `hiddenImage.png`?

Q2: What is the range of possible file sizes for `./programCode`?

Q3: What is the size of the heap immediately before the program finishes?

Memory Allocation

Up until now, we have arbitrarily placed memory with the process page table – however, all modern Operating Systems (OSes) organize the memory of a process in a predictable way:

06/memory-addr.c

```
int val;
printf("&val: %p\n", &val);

void *ptr = malloc(0x1000);
printf("&ptr: %p\n", &ptr);
printf(" ptr: %p\n", ptr);

void *ptr2 = malloc(0x1000);
printf("&ptr2: %p\n", &ptr2);
printf(" ptr2: %p\n", ptr2);

int arr[4096];
printf("&arr: %p\n", &arr);

return 0;
```

Page Table:

....

As a programmer, we talk about these different regions of memory as different "types" of memory:

___________ **Memory**	___________ **Memory**

Q: What if we access memory beyond the end of our heap? (Or any other region not allocated in our page table?)

Memory Address Components:

Address:		

Efficient Use of Heap Memory

During the lifetime of a single process, we will allocate and free memory many times. Consider a simple program:

06/heap.c

```c
int *a = malloc(4096);
printf("a = %p\n", a);
free(a);

int *b = malloc(4096);
printf("b = %p\n", b);

int *c = malloc(4096);
printf("c = %p\n", c);

int *d = malloc(4096);
printf("d = %p\n", d);

free(b);
free(c);

int *e = malloc(5000);
printf("e = %p\n", e);

int *g = malloc(10);
printf("g = %p\n", g);

int *g = malloc(10);
printf("g = %p\n", g);
```

Heap v1:
(Without reuse after free)

Heap v2:
(With reuse after free)

Q2: How much memory is used if we **do not** reuse memory?

Q3: How much memory is used with **optimal** reuse of memory?

- What happens to our memory over time?

- When we have "holes" in our heap, how do we decide what hole to use?

Data Structures for Heap Management

When we manage heap memory, we need to use memory to help us store memory:

- Overhead:

- Allocated Memory:

Metadata-based Approach to Memory Storage

06/heap.c

```c
int *a = malloc(4096);
printf("a = %p\n", a);
free(a);

int *b = malloc(4096);
printf("b = %p\n", b);

int *c = malloc(4096);
printf("c = %p\n", c);

int *d = malloc(4096);
printf("d = %p\n", d);

free(b);
free(c);

int *e = malloc(5000);
printf("e = %p\n", e);

int *g = malloc(10);
printf("g = %p\n", g);

int *g = malloc(10);
printf("g = %p\n", g);
```

Heap w/ Data Structures:
(Without reuse after free)