CS 240
Computer Systems

#6: Heap Memory Allocation and malloc

Feb. 3, 2022 · ❄️ · Wade Fagen-Ulmschneider

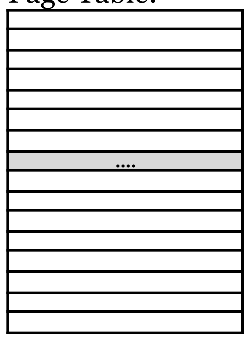
Heap Allocation

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

```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**

Q1: What if we access memory beyond the end of our heap? (Or any other region not allocated in our page table?)

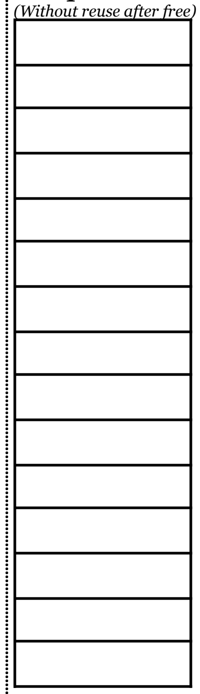
Efficient Use of Heap Memory

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

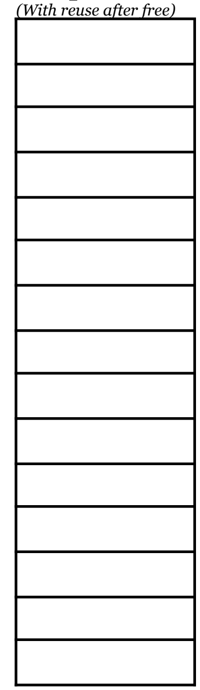
06/heap.c

```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:

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)

Metadata-based Approach to Memory Storage

Allocation Internals

Every process has a single heap starting point and a heap ending point in its virtual memory space that is provided by the Operating System.

- The initial heap size is: ___________________
  -

- A process grows/shrinks its heap using:
  `void *sbrk(intptr_t increment);`

- **MP3** ("mallocc") is released Friday and will have you build your own malloc, using the sbrk call, and require you to efficiently re-use memory just like the Linux kernel does!
  - EC Deadline:
  - Deadline #1:
  - Deadline #2:

Implementation Considerations

1. [Runtime]:

2. [Block Splitting]:

3. [Block Coellessing]: