CS 340
Computer Systems

#7: Memory Allocation and malloc

Sep. 12, 2023 · G Carl Evans

System Memory:

To help us to begin to organize this RAM, we divide the RAM up into chunks called __________.

On Linux, find the size of a page:

```
# getconf PAGESIZE
```

...on most modern systems, a page is ______ KB.

Virtual Memory:

Modern systems provide an abstraction between _________ and ____________________:

1. A ____________________ translates a _________________ into a **physical address**. *It's just a pointer!*

2. Every memory address is made up of the ______________ and the ______________.

3. Virtual Memory is **NOT shared** between processes/apps.

4. **EVERY memory address** ________________________ is a virtual memory address!!

Virtual Memory Example:

P1 Page Table:	RAM:	P2 Page Table:	P3 Page Table:
[0]:	[0]:	[0]:	[0]:
[1]:	[1]:	[1]:	[1]:
[2]:	[2]:	[2]:	[2]:
[3]:	[3]:	[3]:	[3]:
[4]:	[4]:	[4]:	[4]:
[5]:	[5]:	[5]:	[5]:
[6]:	[6]:	[6]:	[6]:
[7]:	[7]:	[7]:	[7]:
[8]:	[8]:	[8]:	[8]:
[9]:	[9]:	[9]:	[9]:
[10]:	[10]:	[10]:	[10]:
[11]:	[11]:	[11]:	[11]:
[12]:	[12]:	[12]:	[12]:
[13]:	[13]:	[13]:	[13]:
[14]:	[14]:	[14]:	[14]:
[15]:	[15]:	[15]:	[15]:

OS Logs:

```
P1: 3 pages (a)
P3: 5 pages (b)
P1: 2 pages (c)
P3 exits
P2: 4 pages (d)
P2: 5 pages (e)
P1:
  Extend a to
  5 pages
  (ex: realloc)
```

P1 Page Table:	RAM:	P2 Page Table:	P3 Page Table:
[0]:	[0]:	[0]:	[0]:
[1]:	[1]:	[1]:	[1]:
[2]:	[2]:	[2]:	[2]:
[3]:	[3]:	[3]:	[3]:
[4]:	[4]:	[4]:	[4]:
[5]:	[5]:	[5]:	[5]:
[6]:	[6]:	[6]:	[6]:
[7]:	[7]:	[7]:	[7]:
[8]:	[8]:	[8]:	[8]:
[9]:	[9]:	[9]:	[9]:
[10]:	[10]:	[10]:	[10]:
[11]:	[11]:	[11]:	[11]:
[12]:	[12]:	[12]:	[12]:
[13]:	[13]:	[13]:	[13]:
[14]:	[14]:	[14]:	[14]:
[15]:	[15]:	[15]:	[15]:

OS Logs:

```
P1: 3 pages (a)
P3: 5 pages (b)
P1: 2 pages (c)
P3 exits
P2: 4 pages (d)
P2: 5 pages (e)
P1:
  Extend a to
  5 pages
  (ex: realloc)
```

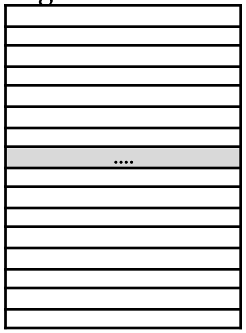
Memory Allocation

At a system level, the page table is a series of pointers to RAM (or other storage). From a process level, we organize our private page table to store data:

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**

Q1: 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

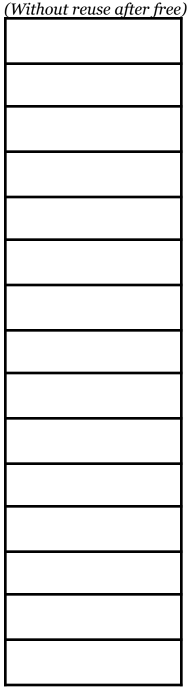
During the lifetime of a single process, we will allocate and free memory many times. Consider a simple program:

06/heap.c

```
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?