CS 240
Computer Systems

#6: Page Eviction, Replacement, Heap Mgmt

Sept. 9, 2021 · Wade Fagen-Ulmschneider

Segmentation Faults

RAM:

[0]:
[1]:
[2]:
[3]:

P1 Page Table:

@ DISK[240]
@ DISK[241]
@ DISK[242]
@ DISK[243]
@ DISK[244]
@ 0x1000 (RAM[1])
@ 0x3000 (RAM[3])
@ 0x2000 (RAM[2])
@ DISK[249]

Disk Pages:

...
./programCode (PC1)
./programCode (PC2)
./programCode (PC3)
./programCode (PC4)
./programCode (PC5)
hiddenImage.png
hiddenImage.png
hiddenImage.png
...

1: Load Program
2: Run PC1
- malloc(4000)
3. Run PC2:
- malloc(10000)
- Open hiddenImage.png
- Read all of image
4: Run PC3
- Access OG 4 KB
- Finish program

Q: What happens if we access the address of the **bold** entry inside of our page table above?

Page Eviction/Replacement Strategies:

When we need to remove a page from RAM and store it on disk, how do we decide which page to remove given a **page access pattern**?

Strategy #1:

	17	33	40	17	43	8	99	33	99	17
RAM										

Strategy #2:

	17	33	40	17	43	8	99	33	99	17
RAM										

Strategy #3:

	17	33	40	17	43	8	99	33	99	17
RAM										

Strategy #4:

	17	33	40	17	43	8	99	33	99	17
RAM										

Other Strategies:

Sample Program:

Can we see the use of the heap and the stack in a real program?

06-see-heap-and-stack-usage.c

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

  int *ptr = malloc(sizeof(int));
  printf("&ptr: %p\n", &ptr);
  printf(" ptr: %p\n", ptr);

  int *ptr2 = malloc(sizeof(int));
  printf("&ptr2: %p\n", &ptr2);
  printf(" ptr2: %p\n", ptr2);

  return 0;
```

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-example.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:
(Without reuse after free)

Heap:
(With reuse after free)

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

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?

Heap Management Strategies

There are many strategies on the best way to allocate memory to the heap:

#1**:** [No Reuse]:

#2: [Free Lists]:

Free Block Allocation Strategies:

1.

2.

3.

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** (“the malloc MP”) is released tonight 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!