

# Page Eviction, Replacement, Heap Management

The background of the slide features a photograph of a large, classical-style statue, likely the 'Alma Mater' statue at the University of Illinois, set in a park-like environment with trees. The entire image is overlaid with a semi-transparent orange color, which serves as a background for the white text.

**CS 240 - The University of Illinois**

Wade Fagen-Ulmschneider

September 9, 2021

# Segmentation Faults

A photograph of a crowd of people gathered around a statue of a woman in a long dress, set against a background of trees. The entire image is overlaid with a semi-transparent orange filter. The text "Segmentation Faults" is centered in white.

## RAM

0x0000	
0x1000	
0x2000	
0x3000	
0x4000	

## P1 Page Table

[0]
[1]
[2] @ DISK[240]
[3] @ DISK[241]
[4] @ DISK[242]
[5] @ DISK[243]
[6] @ DISK[244]
[7]
[8] @ 0x1000
[9] @ 0x3000
[10] @ 0x2000
[11] @ DISK[249]
[12]
[13]
[14]
[15]

## Disk Pages

...
[239]
[240] ./programCode
[241] ./programCode
[242] ./programCode
[243] ./programCode
[244] ./programCode
[245]
[246]
[247]
[248]
[249] hiddenImage.png
[250] hiddenImage.png
[251] hiddenImage.png
[252]
...



# Page Eviction and Replacement Strategies

# Page Eviction and Replacement

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:

Access: <u>RAM</u>	17	33	40	17	43	8	99	33	99	17
[0]										
[1]										
[2]										
[3]										



**I**

*Time*

# Strategy:

Access: <u>RAM</u>	17	33	40	17	43	8	99	33	99	17
[0]										
[1]										
[2]										
[3]										



**I**

*Time*

# Strategy:

Access: <u>RAM</u>	17	33	40	17	43	8	99	33	99	17
[0]										
[1]										
[2]										
[3]										



**I**

*Time*



# Strategy:

Access: <u>RAM</u>	17	33	40	17	43	8	99	33	99	17
[0]										
[1]										
[2]										
[3]										



**I**

*Time*

# Other Strategies:



# Heap and Stack Memory

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

```
5  int val;
6  printf("&val: %p\n", &val);
7
8  int *ptr = malloc(sizeof(int));
9  printf("&ptr: %p\n", &ptr);
10 printf(" ptr: %p\n", ptr);
11
12 int *ptr2 = malloc(sizeof(int));
13 printf("&ptr2: %p\n", &ptr2);
14 printf(" ptr2: %p\n", ptr2);
15
16 return 0;
```

A photograph of a crowd of people gathered around a statue, overlaid with a semi-transparent orange filter. The statue is the central focus, with people standing around it, some looking towards it. The background shows more people and trees. The text is overlaid in the center of the image.

# Exploration of Efficient Use of Heap Memory

# 06-heap-example.c

```
5  int *a = malloc(4096);
6  printf("a = %p\n", a);
7  free(a);
8
9  int *b = malloc(4096);
10 printf("b = %p\n", b);
11
12 int *c = malloc(4096);
13 printf("c = %p\n", c);
14
15 int *d = malloc(4096);
16 printf("d = %p\n", d);
17
18 free(b);
19 free(c);
20
21 int *e = malloc(5000);
22 printf("e = %p\n", e);
23
24 int *f = malloc(10);
25 printf("f = %p\n", f);
26
27 int *g = malloc(10);
28 printf("g = %p\n", g);
```

# 06-heap-example.c

```
5  int *a = malloc(4096);
6  printf("a = %p\n", a);
7  free(a);
8
9  int *b = malloc(4096);
10 printf("b = %p\n", b);
11
12 int *c = malloc(4096);
13 printf("c = %p\n", c);
14
15 int *d = malloc(4096);
16 printf("d = %p\n", d);
17
18 free(b);
19 free(c);
20
21 int *e = malloc(5000);
22 printf("e = %p\n", e);
23
24 int *f = malloc(10);
25 printf("f = %p\n", f);
26
27 int *g = malloc(10);
28 printf("g = %p\n", g);
```

## 06-heap-example.c

```
5  int *a = malloc(4096);
6  printf("a = %p\n", a);
7  free(a);
8
9  int *b = malloc(4096);
10 printf("b = %p\n", b);
11
12 int *c = malloc(4096);
13 printf("c = %p\n", c);
14
15 int *d = malloc(4096);
16 printf("d = %p\n", d);
17
18 free(b);
19 free(c);
20
21 int *e = malloc(5000);
22 printf("e = %p\n", e);
23
24 int *f = malloc(10);
25 printf("f = %p\n", f);
26
27 int *g = malloc(10);
28 printf("g = %p\n", g);
```





# Heap Management Strategies

# Heap Management Strategies

[No Reuse]:

# Heap Management Strategies

[Free Lists]:

# Heap Management Strategies

Free Lists Strategies:



# Memory Allocation Internals

# Heap Management Strategies

Initial Heap Size:

```
void *sbrk(intptr_t increment);
```

`brk()` and `sbrk()` change the location of the program break, which defines the end of the process's data segment (i.e., the program break is the first location after the end of the uninitialized data segment). Increasing the program break has the effect of allocating memory to the process; decreasing the break deallocates memory.

# 06-heap-example.c

```
5  int *a = malloc(4096);
6  printf("a = %p\n", a);
7  free(a);
8
9  int *b = malloc(4096);
10 printf("b = %p\n", b);
11
12 int *c = malloc(4096);
13 printf("c = %p\n", c);
14
15 int *d = malloc(4096);
16 printf("d = %p\n", d);
17
18 free(b);
19 free(c);
20
21 int *e = malloc(5000);
22 printf("e = %p\n", e);
23
24 int *f = malloc(10);
25 printf("f = %p\n", f);
26
27 int *g = malloc(10);
28 printf("g = %p\n", g);
```