Malloc Lab

15/18-213: Introduction to Computer Systems
1011st Recitation, Nov. 7, 2011

TA: Pablo Chavez (pchavez)
Section E, 1:30pm – 2:20pm GHC 5222
Outline

• Overview
• Naïve Implicit List Allocator
• Getting Started
• Evaluation
• Debugging Tips, Misc
You will implement

- `malloc()`
- `calloc()`
- `realloc()`
- `free()`
- `mm_init()`

  - setup whatever you need to
  - called before any calls to `malloc`

- `mm_checkheap()`
  - a debug function

In other words, you will manage the heap yourself.
You have

- `mem_sbrk()`
- unused heap space
- a handful of *scalar* variables
  - no arrays
  - no structs
  - where will you keep your data structures?
mem_sbrk()

• a wrapper around sbrk()
  – sbrk(incr) asks the operating system for incr more bytes of heap space at the end of the current heap
  – has relatively high overhead
Outline

• Overview
• Naïve Implicit List Allocator
• Getting Started
• Evaluation
• Debugging Tips, Misc
Before `mm_init()`
During `mm_init()`

header/footer field format:

<table>
<thead>
<tr>
<th>size of block</th>
<th>alloc</th>
</tr>
</thead>
<tbody>
<tr>
<td>63 bits</td>
<td>1 bit</td>
</tr>
</tbody>
</table>

void *head_end

0x80000020

void *heap_start

0x80000000

64 bits wide
During `mm_init()`

header/footer field format:

<table>
<thead>
<tr>
<th>size of block</th>
<th>alloc</th>
</tr>
</thead>
<tbody>
<tr>
<td>63 bits</td>
<td>1 bit</td>
</tr>
</tbody>
</table>

void *head_end

“epilogue” block

<table>
<thead>
<tr>
<th>footer: size = 0, alloc = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>header: size = 0, alloc = 1</td>
</tr>
</tbody>
</table>

“prologue” block

<table>
<thead>
<tr>
<th>footer: size = 0, alloc = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>header: size = 0, alloc = 1</td>
</tr>
</tbody>
</table>

void *heap_start

64 bits wide

0x80000000

0x80000020

0x800000000

0x80000020
After `mm_init()`

header/footer field format:

<table>
<thead>
<tr>
<th>size of block</th>
<th>alloc</th>
</tr>
</thead>
<tbody>
<tr>
<td>63 bits</td>
<td>1 bit</td>
</tr>
</tbody>
</table>

We’ve set up our invariants – at this point we can implement `mm_checkheap()`

```
void *head_end
```

```
void *heap_start
```

```
| footer: size = 0, alloc = 1 |
| header: size = 0, alloc = 1 |
| footer: size = 0, alloc = 1 |
| header: size = 0, alloc = 1 |
```

0x80000000

0x80000000

64 bits wide

0x80000020

Aside: mm_checkheap()

- mm_checkheap()
  - called between malloc() / free() operations
  - should check invariants are obeyed
  - quickly find bugs

void *head_end

“epilogue” block

footer: size = 0, alloc = 1
header: size = 0, alloc = 1

“prologue” block

footer: size = 0, alloc = 1
header: size = 0, alloc = 1

void *heap_start

0x80000000
64 bits wide

0x80000020
Call to `malloc(0x100)`

header/footer field format:

```
<table>
<thead>
<tr>
<th>size of block</th>
<th>alloc</th>
</tr>
</thead>
<tbody>
<tr>
<td>63 bits</td>
<td>1 bit</td>
</tr>
</tbody>
</table>
```

Search through heap for a free block

Use the fact that header = start of previous block’s footer + 8 bytes

```
void *head_end

“epilogue” block
footer: size = 0, alloc = 1
header: size = 0, alloc = 1

“prologue” block
footer: size = 0, alloc = 1
header: size = 0, alloc = 1
```

nope, not free
nope, not free
Call to `malloc(0x100)`

header/footer field format:

<table>
<thead>
<tr>
<th>size of block</th>
<th>alloc</th>
</tr>
</thead>
<tbody>
<tr>
<td>63 bits</td>
<td>1 bit</td>
</tr>
</tbody>
</table>

Search failed to find free block. Now what?

- `void *head_end`
- "epilogue" block
  - footer: size = 0, alloc = 1
  - header: size = 0, alloc = 1
- "prologue" block
  - footer: size = 0, alloc = 1
  - header: size = 0, alloc = 1

- `void *heap_start`

64 bits wide

nope, not free

nope, not free
Call to `malloc(0x100)`

void *head_end

“epilogue” block

| Footer: size = 0, alloc = 1 |
| Header: size = 0, alloc = 1 |

0x80000130

void *heap_start

“prologue” block

| Footer: size = 0, alloc = 1 |
| Header: size = 0, alloc = 1 |

0x80000000

64 bits wide
Call to `malloc(0x100)`

void *heap_start

64 bits wide

“prologue” block

“payload”

“epilogue” block

return (void *) this

void *heap_end

0x80000000

0x80000130

ftr: size = 0x100, alloc = 1

0x80000118

0x80000018

“payload”

hdr: size = 0x100, alloc = 1

0x80000000

0x80000000

footer: size = 0, alloc = 1

0x80000018

0x80000018

header: size = 0, alloc = 1
Now, Call to malloc(0x50)

void *head_end

“epilogue” block

<table>
<thead>
<tr>
<th>Address</th>
<th>Size</th>
<th>Alloc</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x80000130</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0x80000118</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

“payload”

<table>
<thead>
<tr>
<th>Address</th>
<th>Size</th>
<th>Alloc</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8000018</td>
<td>0x100</td>
<td>1</td>
</tr>
</tbody>
</table>

“prologue” block

<table>
<thead>
<tr>
<th>Address</th>
<th>Size</th>
<th>Alloc</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x80000018</td>
<td>0x100</td>
<td>1</td>
</tr>
<tr>
<td>0x80000000</td>
<td>0x100</td>
<td>1</td>
</tr>
</tbody>
</table>

void *heap_start

64 bits wide
Now, Call to `malloc(0x50)`

```
void *head_end

"epilogue" block

footer: size = 0, alloc = 1
header: size = 0, alloc = 1
ftr: size = 0x50, alloc = 1

"payload"

hdr: size = 0x50, alloc = 1
ftr: size = 0x100, alloc = 1

"payload"

hdr: size = 0x100, alloc = 1
footer: size = 0, alloc = 1

return (void *) this
```

```
0x80000000
0x80000018
0x80000118
0x80000128
0x80000178
0x80000190
0x80000018
0x80000000
```

64 bits wide
suddenly, `free(0x800000128)`

- `void *head_end`
- `void *heap_start`
- "epilogue" block
- "prologue" block
- free(this), where this
- 64 bits wide
suddenly, free(0x80000128)

```
void *head_end

“epilogue” block

footer: size = 0, alloc = 1 0x80000190
header: size = 0, alloc = 1 0x80000178
ftr: size = 0x50, alloc = 1 0x80000128

“payload”

hdr: size = 0x50, alloc = 1 0x80000118
ftr: size = 0x100, alloc = 1 0x80000018

“prologue” block

void *heap_start

footer: size = 0, alloc = 1 0x80000190
header: size = 0, alloc = 1 0x80000178

64 bits wide
```

free(this), where this

suddenly, free(0x80000128)

void *head_end

“epilogue” block

free(this), where this

“prologue” block

void *heap_start

---

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Size</th>
<th>Alloc</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x80000000</td>
<td>ftr</td>
<td>0x100</td>
<td>1</td>
</tr>
<tr>
<td>0x80000010</td>
<td>hdr</td>
<td>0x100</td>
<td>1</td>
</tr>
<tr>
<td>0x80000012</td>
<td>ftr</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0x80000014</td>
<td>footer</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0x80000016</td>
<td>header</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0x80000018</td>
<td>ftr</td>
<td>0x50</td>
<td>0</td>
</tr>
<tr>
<td>0x80000020</td>
<td>header</td>
<td>0x50</td>
<td>1</td>
</tr>
<tr>
<td>0x80000022</td>
<td>footer</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0x80000024</td>
<td>head_end</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
suddenly, `free(0x800000018)`

<table>
<thead>
<tr>
<th>void *head_end</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x80000190</td>
</tr>
<tr>
<td>0x80000178</td>
</tr>
<tr>
<td>0x80000128</td>
</tr>
<tr>
<td>0x80000118</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>“epilogue” block</th>
</tr>
</thead>
<tbody>
<tr>
<td>footer: size = 0, alloc = 1</td>
</tr>
<tr>
<td>header: size = 0, alloc = 1</td>
</tr>
<tr>
<td>ftr: size = 0x50, alloc = 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>free for you to use</th>
</tr>
</thead>
<tbody>
<tr>
<td>hdr: size = 0x50, alloc = 0</td>
</tr>
<tr>
<td>ftr: size = 0x100, alloc = 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>“payload”</th>
</tr>
</thead>
<tbody>
<tr>
<td>hdr: size = 0x100, alloc = 1</td>
</tr>
<tr>
<td>footer: size = 0, alloc = 1</td>
</tr>
<tr>
<td>header: size = 0, alloc = 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>free(this), where this</th>
</tr>
</thead>
<tbody>
<tr>
<td>free(0x800000018), where this</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>“prologue” block</th>
</tr>
</thead>
<tbody>
<tr>
<td>footer: size = 0, alloc = 1</td>
</tr>
<tr>
<td>header: size = 0, alloc = 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>void *heap_start</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x80000000</td>
</tr>
<tr>
<td>0x8000000000</td>
</tr>
</tbody>
</table>
Suddenly, `free(0x800000018)`

void *head_end

“epilogue” block

- footer: size = 0, alloc = 1
- header: size = 0, alloc = 1
- ftr: size = 0x50, alloc = 0

free for you to use

- hdr: size = 0x50, alloc = 0
- ftr: size = 0x100, alloc = 0

free for you to use

- hdr: size = 0x100, alloc = 0
- footer: size = 0, alloc = 1
- header: size = 0, alloc = 1

void *heap_start

64 bits wide

`void *heap_start` 0x80000000

`0x80000018`

`free()` where this

“prologue” block
suddenly, free(0x800000018)

Are we done?

```
void *head_end

"epilogue" block

footer: size = 0, alloc = 1
header: size = 0, alloc = 1
ftr: size = 0x50, alloc = 0

free for you to use

hdr: size = 0x50, alloc = 0
ftr: size = 0x100, alloc = 0

free for you to use

hdr: size = 0x100, alloc = 0
footer: size = 0, alloc = 1
header: size = 0, alloc = 1

free(this), where this

"prologue" block

void *heap_start

0x80000190
0x80000178
0x80000128
0x80000118
0x80000128
0x80000178
0x80000190
0x80000000
0x80000018
64 bits wide
```
suddenly, `free(0x800000018)`

void *head_end

“epilogue” block

nope. try to combine previous and next blocks.

free(this), where this

“prologue” block

void *heap_start

64 bits wide
suddenly, free(0x800000018)

void *head_end

“epilogue” block

footer: size = 0, alloc = 1
header: size = 0, alloc = 1
ftr: size = 0x50, alloc = 0
free for you to use
hdr: size = 0x50, alloc = 0
ftr: size = 0x100, alloc = 0
free for you to use
hdr: size = 0x100, alloc = 0
footer: size = 0, alloc = 1
header: size = 0, alloc = 1

free(this), where this

“prologue” block

0x80000000

0x80000018

0x800000190
0x800000178
0x800000128
0x800000118
0x800000128
0x800000190
0x800000018
0x800000000

64 bits wide
suddenly, `free(0x800000018)`

void *head_end

“epilogue” block

Can only combine these two blocks

free(this), where this

“prologue” block

void *heap_start

---

<table>
<thead>
<tr>
<th></th>
<th>Address</th>
<th>Size</th>
<th>Alloc</th>
</tr>
</thead>
<tbody>
<tr>
<td>footer: size</td>
<td>0x80000018</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>header: size</td>
<td>0x80000018</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>ftr: size</td>
<td>0x80000118</td>
<td>0x50</td>
<td>0</td>
</tr>
<tr>
<td>free for you</td>
<td>0x80000190</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ftr: size</td>
<td>0x80000178</td>
<td>0x100</td>
<td>0</td>
</tr>
<tr>
<td>free for you</td>
<td>0x80000128</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ftr: size</td>
<td>0x80000118</td>
<td>0x100</td>
<td>0</td>
</tr>
<tr>
<td>free for you</td>
<td>0x80000190</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>hdr: size</td>
<td>0x80000180</td>
<td>0x100</td>
<td>0</td>
</tr>
<tr>
<td>free for you</td>
<td>0x80000018</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>footer: size</td>
<td>0x80000018</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>header: size</td>
<td>0x80000018</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

64 bits wide
suddenly, free(0x800000018)

void *head_end

“epilogue” block

free for you to use

Ok?

free(this), where this

“prologue” block

void *heap_start

64 bits wide
suddenly, free(0x800000018)

void *head_end

“epilogue” block

Nope. Common source of errors.

free(this), where this

“prologue” block

free for you to use

hdr: size = 0x150, alloc = 0
footer: size = 0, alloc = 1
header: size = 0, alloc = 1
ftr: size = 0x150, alloc = 0

0x80000190
0x80000178
0x80000128
0x80000118
0x80000108
0x80000018
0x80000000

void *heap_start

64 bits wide
suddenly, free(0x800000018)

void *head_end

“epilogue” block

footer: size = 0, alloc = 1
header: size = 0, alloc = 1
ftr: size = 0x160, alloc = 0

free for you to use

free(this), where this

“prologue” block

void *heap_start

64 bits wide

0x80000190
0x80000178
0x80000128
0x80000118
0x80000100
0x80000018
0x80000010
0x80000000
Outline

• Overview
• Naïve Implicit List Allocator
• Getting Started
• Evaluation
• Debugging Tips, Misc
How to Start (suggestion)

• Read the 32 bit implicit list in your textbook and wade through the sea of macros.
  – Don’t copy and paste from the CSAPP website (you won’t learn as much).
• Implement a naïve 64 bit implicit list
  – you can leave out coalescing, just for now
• Implement `mm_checkheap()` for this heap pattern.
How to Finish

• Turn an implicit list into an explicit list
  – update mm_checkheap(), debug, tune

• Turn your explicit list into segregated free lists
  – update mm_checkheap(), debug, tune
Outline

• Overview
• Naïve Implicit List Allocator
• Getting Started
• Evaluation
• Debugging Tips, Misc
[student@makoshark]$ ./mdriver
Using default tracefiles in ./traces/
Measuring performance with a cycle counter.
Processor clock rate ~= 2260.7 MHz

............................
Results for mm malloc:

<table>
<thead>
<tr>
<th>valid</th>
<th>util</th>
<th>ops</th>
<th>secs</th>
<th>Kops</th>
<th>trace</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>89%</td>
<td>100000</td>
<td>0.004959</td>
<td>20165</td>
<td>./traces/alaska.rep</td>
</tr>
<tr>
<td>* yes</td>
<td>75%</td>
<td>11991</td>
<td>0.001168</td>
<td>10266</td>
<td>./traces/chrome.rep</td>
</tr>
<tr>
<td>u yes</td>
<td>59%</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>./traces/exhaust.rep</td>
</tr>
<tr>
<td>* yes</td>
<td>77%</td>
<td>200</td>
<td>0.000015</td>
<td>13613</td>
<td>./traces/lrucd.rep</td>
</tr>
<tr>
<td>u yes</td>
<td>99%</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>./traces/needle.rep</td>
</tr>
<tr>
<td>p yes</td>
<td>--</td>
<td>6495</td>
<td>0.004902</td>
<td>1325</td>
<td>./traces/seglist.rep</td>
</tr>
<tr>
<td>yes</td>
<td>99%</td>
<td>12</td>
<td>0.000003</td>
<td>4105</td>
<td>./traces/short2.rep</td>
</tr>
<tr>
<td>17</td>
<td>16</td>
<td>85%</td>
<td>149036</td>
<td>0.024727</td>
<td>6027</td>
</tr>
</tbody>
</table>

Perf index = 52 (util) & 11 (thru) = 63/100
if(size == 213) /* large_trace.rep */
{
    // skip searching through free
    // list, because there isn’t any
    // block that fits – I checked
    // the trace.
    extend_heap(size);
}

Not a general purpose allocator. You will be penalized for this.
Outline

• Overview
• Naïve Implicit List Allocator
• Getting Started
• Evaluation
• Debugging Tips, Misc
Misc

*p + 0xff | 0x08  \rightarrow  SIZE(HD(ptr))

??????  \rightarrow  oh.
#define MAX(x, y) x > y ? x : y
– MAX(x++, y) -> x++ > y ? x++ : y
  • Not what you’d expect

static inline int max(x, y) {
    return x > y ? x : y;
}
– max(x++, y)?

#define DOUBLE(x) 2 * x
– DOUBLE(x+1) -> 2 * x + 1 (wrong)

#define DOUBLE(x) 2 * (x)

• struct __attribute__((__packed__))
  – neat, but gcc specific (not standard)
Debugging Tips, Misc

1. write \texttt{mm\_checkheap()}. 

2. write \texttt{mm\_checkheap()} well.

3. write coalescing to make bugs more apparent, fix bugs using \texttt{mm\_checkheap()}. 

4. Start now.

5. Accelerate neutrinos past the speed of light, enabling you to start three days ago.

6. Good luck!