Recitation 11: MallocLab Part 1

Outline

- List Utilization
- Structuring (meta)Data
- GDB Exercises

Malloc Internals

- **The heap consists of blocks of memory**
  - Some are allocated
  - Some are free

- **What is responsible for tracking allocated blocks?**
- **What is responsible for tracking free blocks?**

List Utilization

- **The malloc package is responsible for tracking free blocks**
  - Blocks are tracked in a free list
  - Malloc tries reusing these blocks to satisfy future allocation requests

- **mm-baseline uses an implicit list**
  - What is its memory utilization in the lab?

Finding a block

- What fit algorithm does mm-baseline use?
- What other fit algorithms could be used?
- If you switch from an implicit to explicit list representation, how does this change memory utilization?

Finding a Best Block

- **You have implemented explicit list representation**
  - You were using best fit with explicit lists

- **You experiment with segregated lists and best fit**
  - Is there a better fit for a given allocation?
  - What advantage(s) does segregated lists provide?

Structuring (meta)Data

- **There are (at least) two different types of blocks:**
  - Allocated and free

- **What data is common between blocks?**

Structuring (meta)Data

- **There are (at least) two different types of blocks:**
  - Allocated and free
- **What data is common between blocks?**
- **What data might a free block need?**

Structuring (meta)Data

- **There are (at least) two different types of blocks:**
  - Allocated and free

- **What data is common between blocks?**

- **What data might a free block need?**

- **Is there any unused space in free blocks?**

Structuring (meta)Data

- **There are (at least) two different types of blocks:**
  - Allocated and free
- **What data is common between blocks?**
- **What data might a free block need?**
- **Is there any unused space in free blocks?**
- **How can we overlap two different types of data at the same location?**

GDB Practice

- **Using GDB well in malloclab can save you <u>HOURS</u>* of debugging time**
  - Average 20 hours using GDB for “B” on malloclab
  - Average 23 hours not using GDB for “B” on malloclab

- **Form pairs**
  - Login to a shark machine
  - wget http://www.cs.cmu.edu/~213/activities/rec11.tar
  - tar xf rec11.tar
  - cd rec11
  - make
- **Two buggy mdrivers**

***Average time is based on Summer 2016 survey results**

Debugging mdriver

```
$ gdb --args ./mdriver -c traces/syn-mix-short.rep
(gdb) run


(gdb) backtrace


(gdb) disassemble
```

1) What function did backtrace indicate? What function was disassembled? What happened?

2) What line of assembly has crashed?

Debugging mdriver cont.

(gdb) disassemble

```
=> 0x0...0404363 <+243>:  mov  %rcx,0x8(%rdi,%rsi,1)
```

(gdb) print /x <reg> // What register holds the memory location accessed?
// Does the address look valid?
// What about the other register?

Looking up from "=>", which x86 instructions generate these values? (Hint: The instructions are implementing parts of:

```
block_t *block_next = (block_t *)(((char *)block) +
get_size(block));
word_t *footerp = (word_t *)((block->payload) + get_size(block) -
dsize);
```

Which component is invalid? Review mm.c and identify the bug.

Debugging Mdriver-2

```
$ gdb --args ./mdriver-2 -c traces/syn-array-short.rep
(gdb) run
```

mm_checkheap will fail

Track the headers / footers for the blocks in the heap using

```
(gdb) watch *0x800000008 // And other addresses
```

5 Commands to Remember

- backtrace
- frame
- disassemble
- print <reg>
- watch

MallocLab Checkpoint

- Due Thursday
- Checkpoint should take approximately half of the time
- Read the writeup
- Use GDB
- Ask us for debugging help