Malloc Debugging

15-213: Introduction to Computer Systems
Recitation 12: November 12, 2012

Andrew Audibert
Section D

Overview

- Common Errors
- Segmentation Faults
- Heap Checker
  - What it should do
  - What it should check
- gdb
  - Watch points

Common Errors

- If the driver complains about garbled bytes, that means you are overwriting part of an allocated payload.
    - Check your pointer arithmetic.

- If you waste too much space, some tests (particularly needle) will fail with out of memory errors.
    - This might happen if your allocator loses track of some blocks.

- Remember that you need to reinitialize everything when mm_init is called. We will call it between all traces.

Segmentation Faults

- printf is rarely the best way to debug these.

- A segfault on line 200 may be caused by a bug on line 70.

- segfaults are usually caused either by pointer arithmetic errors or violation of your invariants (corruption of the heap)

- checkheap can save you massive amounts of time in debugging the second type.

Heap Checker

- Your heap checker should not print things out unless it finds an error. This lets you sprinkle calls to it throughout your code.

- Once you know what you want your heap structure to look like, write a heap checker for that structure so that you can debug the rest of your malloc implementation.

- If you come to office hours with a nasty bug, the first thing we'll be interested in will be your heap checker.

What Makes a Good Heap Checker?

- Your heap checker should be detailed enough that the rest of your functions are guaranteed to work on any heap that your heap checker passes.

- What invariants do your heaps have?

Heap Checker Invariants

Invariants to think about:

- (Doubly) linked lists are pointed correctly?
- Headers and footers match up?
- No allocated blocks in your explicit list?
- No free blocks NOT in your explicit list?
- Any of YOUR OWN invariants! (address-ordering?)
- Seg lists: no big chunks in small lists / vice versa?
- Are there cycles in any of the lists? You can check this using the hare and tortoise algorithm.

Hare and Tortoise Algorithm

- Set two pointers "hare" and "tortoise" to the beginning of your list.

- During each iteration, move the hare pointer forward two nodes and move the tortoise forward one node. If they are pointing to the same node after this, the list has a cycle.

- If the tortoise reaches the end of the list, there are no cycles.

Useful gdb Techniques

- When you get a segfault, you can quickly find out which line it occurred on by doing 'gdb mdriver' and then 'run'.

- You can set watch points in gdb so that when a location in memory is written you are notified and execution is suspended just like for a break point. This can help you find the culprit when something is being corrupted.

- To break when the integer at address 0x12345678 is modified, you can do

watch *((int *) 0x12345678)

Questions?

(Don't be afraid to come to office hours if you are stuck*)

*We'll be able to help you more if you have a good heap checker already.