Malloc Recitation

Recitation 12: November 15, 2016

Agenda

- Recap
- Data structures and Explicit List
- Debugging using GDB

Malloc Recap

Malloc basics

- **What is dynamic memory allocation?**

- **Terms you will need to know**
  - malloc/ calloc / realloc
  - free
  - sbrk
  - payload
  - fragmentation (internal vs. external)
  - coalescing
    - Bi-directional
    - Immediate vs. Deferred

Allocation Example

```
p1 = malloc(4)
```

```
p2 = malloc(5)
```

```
p3 = malloc(6)
```

```
free(p2)
```

```
p4 = malloc(2)
```

Fragmentation

- **Internal fragmentation**
  - Result of **payload** being smaller than block size.
  - `void * m1 = malloc(3); void * m2 = malloc(3);`
  - `m1,m2` both have to be aligned to 16 bytes…
- **External fragmentation**

External Fragmentation

- Occurs when there is enough aggregate heap memory, but no single free block is large enough

```
p1 = malloc(4)
p2 = malloc(5)
p3 = malloc(6)
free(p2)
p4 = malloc(6)
```

Oops! (what would happen now?)

- Depends on the pattern of future requests
  - Thus, difficult to measure

Implementation Hurdles

- **How do we know where the blocks are?**
- **How do we know how big the blocks are?**
- **How do we know which blocks are free?**
- **Remember: can't buffer calls to malloc and free… must deal with them real-time.**
- **Remember: calls to `free` only takes a pointer, not a pointer and a size.**
- **Solution: <u>Need a data structure to store information on the "blocks"</u>**
  - Where do I keep this data structure?
  - We can't allocate a space for it, that's what we are writing!

Malloc: Deep Dive

The data structure

- **Requirements:**
  - The data structure needs to tell us where the blocks are, how big they are, and whether they're free
  - We need to be able to CHANGE the data structure during calls to malloc and free
  - We need to be able to find the **next free block** that is "a good fit for" a given payload
  - We need to be able to quickly mark a block as free/allocated
  - We need to be able to detect when we're out of blocks.
    - What do we do when we're out of blocks?

The data structure

- **It would be convenient if it worked like:**

```
malloc_struct malloc_data_structure;
…
ptr = malloc(100, &malloc_data_structure);
…
free(ptr, &malloc_data_structure);
…
```

- **Instead all we have is the memory we're giving out.**
  - All of it doesn't have to be payload! We can use some of that for our data structure.

The data structure

- **The data structure IS your memory!**
- **A start:**
  - <h1> <pl1> <h2> <pl2> <h3> <pl3>
  - What goes in the header?
    - That's your job!
  - Let's say somebody calls free(p2), how can I coalesce?
    - Maybe you need a **footer**? Maybe not?

The data structure

- **Common types**
  - Implicit List
    - Root -> block1 -> block2 -> block3 -> …
  - Explicit List
    - Root -> free block 1 -> free block 2 -> free block 3 -> …
  - Segregated List
    - Small-malloc root -> free small block 1 -> free small block 2 -> …
    - Medium-malloc root -> free medium block 1 -> …
    - Large-malloc root -> free block chunk1 -> …

Explicit List

- **Improvement over implicit list implemented by mm-baseline.c**
- **From a root, keep track of all free blocks in a (doubly) linked list**
  - Remember a doubly linked list has pointers to next and previous
  - Do we therefore use more space than in implicit list implementation ?

Explicit List

- **Improvement over implicit list implemented by mm-baseline.c**
- **From a root, keep track of all free blocks in a (doubly) linked list**
  - Remember a doubly linked list has pointers to next and previous
  - Do we therefore use more space than in implicit list implementation ?
  - Perhaps not!
    - What data is common between allocated block and free block ?

Explicit List

- **Improvement over implicit list implemented by mm-baseline.c**
- **From a root, keep track of all free blocks in a (doubly) linked list**
  - Remember a doubly linked list has pointers to next and previous
  - Do we therefore use more space than in implicit list implementation ?
  - Perhaps not!
    - What data is common between allocated block and free block ?
      - Header, Payload, Footer
    - Does a free block need data to be stored in payload ? Can we reuse this space ?

Explicit List

- **Improvement over implicit list implemented by mm-baseline.c**
- **From a root, keep track of all free blocks in a (doubly) linked list**
  - Remember a doubly linked list has pointers to next and previous
  - Do we therefore use more space than in implicit list implementation ?
  - Perhaps not!
    - What data is common between allocated block and free block ?
      - Header, Payload, Footer
    - Does a free block need data to be stored in payload ? Can we reuse this space ?
      - How can we overlap two different types of data at the same location ?

Explicit List

- **Improvement over implicit list implemented by mm-baseline.c**
- **From a root, keep track of all free blocks in a (doubly) linked list**
  - Remember a doubly linked list has pointers to next and previous
  - Do we therefore use more space than in implicit list implementation ?
  - Perhaps not!
    - What data is common between allocated block and free block ?
      - Header, Payload, Footer
    - Does a free block need data to be stored in payload ? Can we reuse this space ?
      - How can we overlap two different types of data at the same location ?
    - Does an allocated block need next and previous pointers to be stored ?
    - Does an allocated block need a footer ?

Explicit List

- **Improvement over implicit list implemented by mm-baseline.c**
- **From a root, keep track of all free blocks in a (doubly) linked list**
  - Remember a doubly linked list has pointers to next and previous
- **When malloc is called, can now find a free block quickly**
  - What happens if the list is a bunch of small free blocks but we want a really big one?
  - How can we speed this up?

Segregated List

- **An optimization for explicit lists**
- **Can be thought of as multiple explicit lists**
  - What should we group by?
- **Grouped by size – let's us quickly find a block of the size we want**
- **What size/number of buckets should we use?**
  - This is up to you to decide

Instrumentation

- **Find aspects of the code which degrade performance**
- **Example: find_fit takes a lot of time**
  - What metric to collect? Compute the ratio of blocks viewed to calls

```
static block_t *find_fit(size_t asize)
{
    block_t *block;  call_count++;
    for (block = heap_listp; get_size(block) > 0;
                             block = find_next(block))
    {   block_count++;
        if (!(get_alloc(block)) && (asize <= get_size(block)))
        {
            return block;
        }
    }
    return NULL; // no fit found
}
```

Heap Checker

- **Part of the assignment is writing a heap checker**
  - This is here to help you.
  - **Write the heap checker as you go, don't think of it as something to do at the end**
  - A good heap checker will make debugging much, much easier
- **Heap checker tips**
  - Heap checker should run silently until it finds an error
    - Otherwise you will get more output than is useful
    - You might find it useful to add a "verbose" flag, however
  - Consider using a macro to turn the heap checker on and off
    - This way you don't have to edit all of the places you call it
  - There is a built-in macro called `__LINE__` that gets replaced with the line number it's on
    - You can use this to make the heap checker tell you where it failed
  - Call the heap checker at places that have a logical end. Eg: End of malloc(), free(), coalesce()
  - Call heap checker at the start and end of these functions

Design Considerations

- I found a chunk that fits the necessary payload… should I look for a better fit or not? (First fit vs. Best fit)
- Pros and Cons of First fit vs Best fit
- Can we speed up Best fit ?

Design Considerations

- **Free blocks: address-ordered or LIFO**
  - What's the difference?
  - Pros and cons?
- **Coalescing**
  - When do you coalesce?
- **You will need to be using an explicit list at minimum score points**
  - But don't try to go straight to your final design, build it up iteratively.

Possible Optimizations

- Eliminate footers in allocated blocks. But, you still need to be able to implement coalescing
- Decrease the minimum block size. But, you must then manage free blocks that are too small to hold the pointers for a doubly linked free list
- Reduce headers below 8 bytes. But, you must support all possible block sizes, and so you must then be able to handle blocks with sizes that are too large to encode in the header
- Set up special regions of memory for small, fixed-size blocks. But, you will need to manage these and be able to free a block when given only the starting address of its payload

Debugging

- **Debugging Tips using mm-baseline.c**
  - Using GDB
  - Using heapchecker
  - Using hprobes
- **We have injected a small bug in mm-baseline.c**
- **We attempt to trace it using the above debugging tools**

Debugging using GDB

- Set the optimization level to 0 before debugging
- Reset the optimization level back after debugging

```
#
# Makefile for the malloc lab driver
#
# Regular compiler
CC = gcc
# Compiler for mm.c
CLANG = clang
# Change this to -O0 (big-Oh, numeral zero) if you need to use a debugger on your code
COPT = -O0
CFLAGS = -Wall -Wextra -Werror $(COPT) -g -DDRIVER -Wno-unused-function -Wno-unused-parameter
LIBS = -lm -lrt
```

Bug Type I: Segmentation Faults

- Recollect the recitation on debugging using GDB
- Very useful to obtain the backtrace
- Examine values of variables

Segmentation Fault

```
bash-4.2$ gdb --args ./mdriver -c traces/syn-array.rep
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-80.el7
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /afs/andrew.cmu.edu/usr5/preetium/private/labs/malloclabcheckpoint-hand
(gdb) run
Starting program: /afs/andrew.cmu.edu/usr5/preetium/private/labs/malloclabcheckpoint-handout
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Found benchmark throughput 19868 for cpu type Intel(R)Xeon(R)CPUE5-2680v2@2.80GHz, benchmark

Throughput targets: min=9934, max=17881, benchmark=19868

Program received signal SIGSEGV, Segmentation fault.
0x000000000040634d in find_prev (block=0x800000000) at mm.c:628
628         size_t size = extract_size(*footerp);
Missing separate debuginfos, use: debuginfo-install glibc-2.17-106.el7_2.8.x86_64
(gdb) bt
#0  0x000000000040634d in find_prev (block=0x800000000) at mm.c:628
#1  0x0000000000405b92 in coalesce (block=0x800000000) at mm.c:417
#2  0x000000000040560f in extend_heap (size=4096) at mm.c:406
#3  0x00000000004054f0 in mm_init () at mm.c:219
#4  0x000000000040322a in eval_mm_valid (trace=0x61d4c0, ranges=0x61d480) at mdriver.c:1032
#5  0x00000000004015ad in run_tests (num_tracefiles=1, tracedir=0x60c1e0 <tracedir> "./", tr
#6  0x0000000000401d64 in main (argc=3, argv=0x7fffffffdfd8) at mdriver.c:506
(gdb) p footerp
$1 = (word_t *) 0x7fffffff8
(gdb) p mem_heap_hi()
$2 = (void *) 0x80000100f
(gdb) p mem_heap_lo()
$3 = (void *) 0x800000000
(gdb)
```

- Notice the footer value
- It is outside the range of the heap

Bug Type 2: Correctness error report by driver

```
-bash-4.2$ ./mdriver -p -V -D -f traces/syn-array.rep
Found benchmark throughput 17422 for cpu type Intel(R)Xeon(R)CPUE5-2680v2@2.80GHz, benchmark checkpoint

Throughput targets: min=3484, max=15680, benchmark=17422

Testing mm malloc
Reading tracefile: traces/syn-array.rep
Checking mm_malloc for correctness, ERROR [trace ./traces/syn-array.rep, line 8]: Payload (0x800000740:0x800001213) overlaps another payload (0x800000740:0x800002127)

Results for mm malloc:
  valid    util     ops   msecs    Kops  trace
 *  no      -         -      -         - ./traces/syn-array.rep
            -         -      -
```

Setting breakpoints

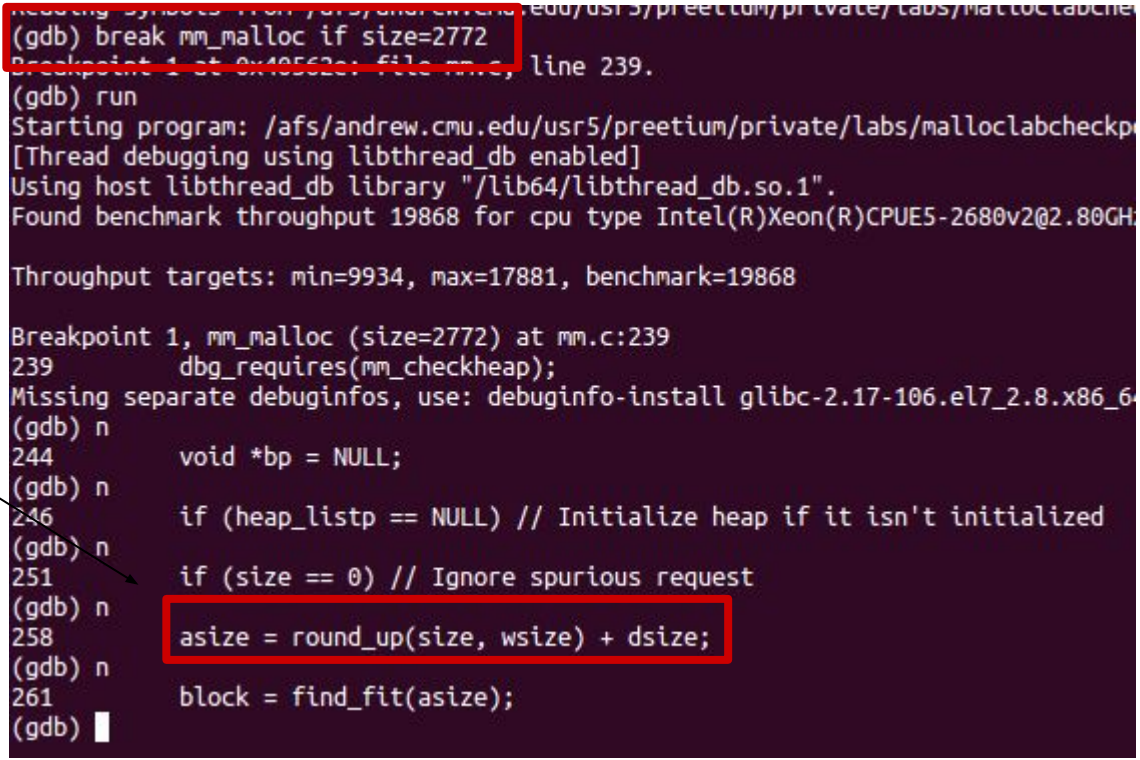
- The tracefile contains a lot allocations and few frees
- Most likely mm_malloc() has the issue
- Set breakpoint at every call to malloc

Setting breakpoints

```
(gdb) break mm_malloc
Breakpoint 1 at 0x40562c: file mm.c, line 235.
(gdb) run
Starting program: /afs/andrew.cmu.edu/usr5/preetium/private/labs/malloclabcheckpoint-handout/./mdriver -c traces/syn-array.rep
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Found benchmark throughput 19868 for cpu type Intel(R)Xeon(R)CPUE5-2680v2@2.80GHz, benchmark regular

Throughput targets: min=9934, max=17881, benchmark=19868

Breakpoint 1, mm_malloc (size=1820) at mm.c:235
235         void *bp = NULL;
Missing separate debuginfos, use: debuginfo-install glibc-2.17-106.el7_2.8.x86_64
(gdb) c
Continuing.

Breakpoint 1, mm_malloc (size=6632) at mm.c:235
235         void *bp = NULL;
(gdb) c
Continuing.

Breakpoint 1, mm_malloc (size=12) at mm.c:235
235         void *bp = NULL;
(gdb) c
Continuing.

Breakpoint 1, mm_malloc (size=2772) at mm.c:235
235         void *bp = NULL;
(gdb) c
Continuing.
ERROR [trace ./traces/syn-array.rep, line 8]: Payload (0x800000740:0x800001213) overlaps another payload (0x800000740:0x80000217

correctness check finished, by running tracefile "traces/syn-array.rep".
 => incorrect.

Terminated with 1 errors
[Inferior 1 (process 14430) exited normally]
(gdb)
```

Setting breakpoints

Should have been:
`asize = round_up(size, dsize) + dsize;`



Heapchecker

- The above problem is easy to identify using heap checker

```
bash-4.2$ ./mdriver -p -V -D -f traces/syn-array.rep
Found benchmark throughput 17422 for cpu type Intel(R)Xeon(R)CPUE5-2680v2@2.80GHz, benchmark checkpoint

Throughput targets: min=3484, max=15680, benchmark=17422

Testing mm malloc
Reading tracefile: traces/syn-array.rep
Checking mm_malloc for correctness, Line 0, Heap error in block 0x800000738.  Header (0x19f1) != footer (0x19f9)
ERROR [trace ./traces/syn-array.rep, line 7]: mm_checkheap returned false


Results for mm malloc:
  valid    util     ops    msecs     Kops  trace
 *  no      -         -        -         - ./traces/syn-array.rep
            -         -        -

Terminated with 1 errors
```

Using Hprobes

- Use hprobes as mentioned in the handout on the defaulting block
- Useful to check the contents of the heap

(gdb) break place if block = 0x800000738

Examine header and footer

(gdb) print hprobes(block, 0, asize)

```
Reading symbols from /afs/andrew.cmu.edu/usr5/preetium/private/labs/malloclabcheckpoint-handout/mdriver...done.
(gdb) break place if block=0x800000738
Breakpoint 1 at 0x405940: file mm.c, line 463.
(gdb) run
Starting program: /afs/andrew.cmu.edu/usr5/preetium/private/labs/malloclabcheckpoint-handout/./mdriver -c traces/syn-array.rep
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Found benchmark throughput 19868 for cpu type Intel(R)Xeon(R)CPUE5-2680v2@2.80GHz, benchmark regular

Throughput targets: min=9934, max=17881, benchmark=19868

Breakpoint 1, place (block=0x800000738, asize=1840) at mm.c:463
463         size_t csize = get_size(block);
Missing separate debuginfos, use: debuginfo-install glibc-2.17-106.el7_2.8.x86_64
(gdb) print hprobe(block, 0, 8)
Bytes 0x80000073f...0x800000738: 0x0000000000000000
$1 = void
(gdb) print hprobe(block, 0, 16)
Bytes 0x800000747...0x800000738: 0x00000000000000000000000000000000
$2 = void
(gdb) n
465         if ((csize - asize) >= min_block_size)
(gdb) n
468             write_header(block, asize, true);
(gdb) n
469             write_footer(block, asize, true);
(gdb) print hprobe(block, 0, 8)
Bytes 0x80000073f...0x800000738: 0x0000000000000731
$3 = void
(gdb) n
471             block_next = find_next(block);
(gdb) print hprobe(block, 0, 8)
Bytes 0x80000073f...0x800000738: 0x0000000000000731
$4 = void
(gdb) print asize
$5 = 1840
(gdb) print (block->payload) + get_size(block) - dsize
$6 = 0x800000e60 "A\a"
(gdb) print hprobe(block, 0, asize)
Bytes 0x800000e67...0x800000738: 0x0000000000007310
```

Using watchpoints

- **Now use watchpoints to observe when the header and footer values change**
  - watch *0x800000e67, where 0x800000e67 is the address of the header as shown by hprobes
  - watch *0x800000738, where 0x800000738 is the address of the footer as shown by hprobes
- **Exercise: Try to see if you can catch the error that we caught earlier by stepping through the code**

Summary

- **You can use dbg_printf and friends for more verbose debugging**
- **Use GDB, heapchecker and hprobes generously**
- **Write the heapchecker in parallel with the code**
- **Read the handout carefully**
- **Encapsulate complexity within helper functions:**
  - add_free_block(), remove_free_block()
  - find_next_blk(), find_prev_blk()
  - find_bucket() for segregated lists….