Malloc Week – Day 3!

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

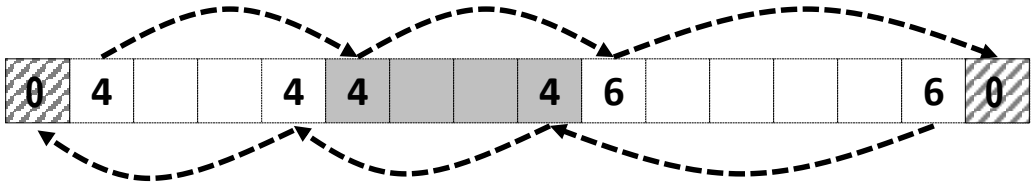
- **You have implemented explicit list representation**
  - You were using best fit with explicit lists
- **You experiment with segregated lists and best fit**
  - Is there a better fit for a given allocation?
  - What advantage(s) does segregated lists provide?

Structuring (meta)Data

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

Sketch out the Heap

- Start with a heap, in this case implicit list

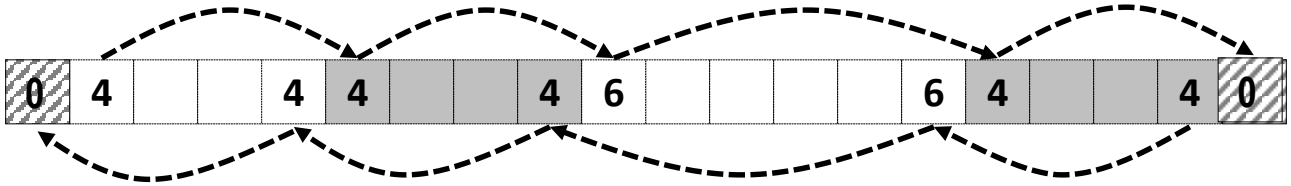


- Now try something, in this case, extend_heap

```
block_t *block = payload_to_header(bp);
write_header(block, size, false);
write_footer(block, size, false);
// Create new epilogue header
block_t *block_next = find_next(block);
write_header(block_next, 0, true);
```

Sketch out the Heap

- Start with a heap, in this case implicit list



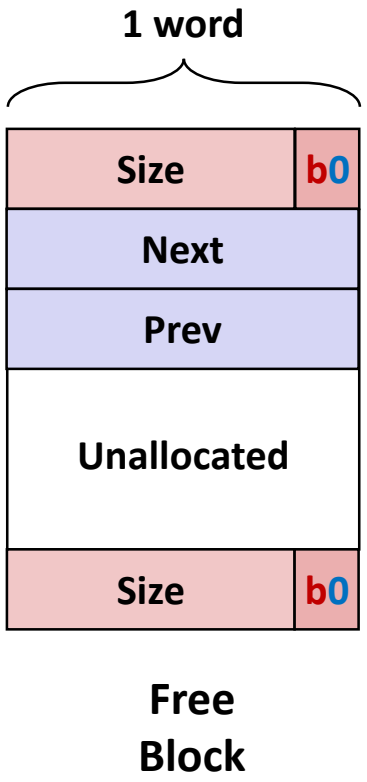
- Now try something, in this case, extend_heap

```
block_t *block = payload_to_header(bp);
write_header(block, size, false);
write_footer(block, size, false);
// Create new epilogue header
block_t *block_next = find_next(block);
write_header(block_next, 0, true);
```

Sketch out the Heap

- **Here is a free block based on lectures 19 and 20**
  - Explicit pointers (will be well-defined see writeup and Piazza)
  - Optional boundary tags

- **If you make changes to your design beyond this**
  - Draw it out.
  - If you have bugs,
    pictures can help the staff help you



Add Instrumentation

- **Remember that measurements inform insights.**
  - Add temporary code to understand aspects of malloc
  - Code can violate style rules or 128 byte limits, because it is temporary

- **Particularly important to develop insights into performance before making changes**
  - What is expensive throughput-wise?
  - How much might a change benefit utilization?

Add Instrumentation example

- **Looping in find_fit takes most of the time**

- **How efficient is your code? How might you know?**
  - 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
}
```

Add Instrumentation cont.

- **What size of requests?**
  - How many 8 bytes or less?
  - How many 16 bytes or less?
  - What other sizes?
- **What else could you measure? Why?**
- **Remember that although the system's performance varies**
  - The mdriver's traces are deterministic
  - Measured results should not change between runs

Use tools

- **Write your own – mm_checkheap()**
  - What conditions are true in a valid heap?
  - Discuss.

- **Use gdb**
  - Sometimes augmented with checkheap or printfs
  - Always valuable insights

GDB DEMO

Garbled Bytes

- **Malloc library returns a block**
  - mdriver writes bytes into payload (using memcpy)
  - mdriver will check that those bytes are still present
  - If malloc library has overwritten any bytes, then report garbled bytes

- **Now what?**

- **The mm_checkheap call is catching it right?**
- **If not, we want to find the garbled address and watch it**

Garbled Bytes and gdb

- Get out a laptop

- Login to shark machine
- wget http://www.cs.cmu.edu/~213/activities/recML.tar
- tar xf recML.tar

- This is an explicit list mdriver with a bug.
  - No source code is provided.

GDB Exercise

- gdb --args ./mdriver-garb -c ./traces/syn-array-short.rep

```
(gdb) r
// Sample output follows
Throughput targets: min=6528, max=11750, benchmark=13056
Malloc size 9904 on address 0x800000010.
...
ERROR [trace ././traces/syn-array-short.rep, line 12]:
block 0 has 8 garbled bytes, starting at byte 0
...
Terminated with 2 errors
[Inferior 1 (process 13470) exited normally]
(gdb)
```

GDB Exercise cont.

- **What is the first address that was garbled?**
  - Use gdb watch to find out when / what garbled it.

```
(gdb) watch * 0x800000010
(gdb) run
```

// Keep continuing through the breaks:
// mm_init()
// 4 x memcpy

```
Hardware watchpoint 1: *0x800000010

Old value = -7350814
New value = 0
0x0000000004041b7 in mm_malloc ()
```

We just broke in after overwriting