Understanding Malloc

Zachary Porter

2022-10-31

1 Problem

I need to understand the codebase for the glibc memory allocator.

2 Starting knowledge

The comment in malloc.c says that this uses ptmalloc2. We also know from this comment that there are three main ways blocks can be allocated.

```
* For large (>= 512 bytes) requests, it is a pure best-fit allocator,
  with ties normally decided via FIFO (i.e. least recently used).
* For small (<= 64 bytes by default) requests, it is a caching
  allocator, that maintains pools of quickly recycled chunks.
* In between, and for combinations of large and small requests, it does
  the best it can trying to meet both goals at once.
* For very large requests (>= 128KB by default), it relies on system
  memory mapping facilities, if supported.
```

3 Things I want to know

- What code is run for each of the four different sizes of blocks to be allocated
- What code is run when running free() on chunks of each different size
- What code causes a problem in a double free
- Where the check for allocating less than 32 bytes is (minimum allocated size is 32 bytes)

4 Major concepts

The major tools that malloc uses are: tcache, fastbins, smallbins, largebins, unallocated linked list, sbrk and mmap. The first five are tools that allow malloc to reuse memory that was just free()d but that cannot be given back to the system because it lies in a congiguous heap between used blocks. sbrk and mmap are tools that allow malloc to expand when needed. sbrk can grow the heap and mmap is best for long lived objects.

4.1 Tcache

Tcache (or thread-cache) is a tool used my malloc to avoid the expensive synchronization step between multiple threads when accessing a single arena. This works similarly to the fastbins.

4.2 Bins

TODO

4.3 unallocated linked list

TODO

4.4 Sbrk

Malloc is able to use sbrk to grow the heap contiguously as far as it wants. This allows malloc to assume that the last (special) reserved block is infinite and malloc grows the stack when data is consumed from this block.

4.5 Mmap

The problem with sbrk is that memory cannot be given back to the operating system when it is freed. For example, imagine allocating blocks of sizes [1Kib, 512Mib, 1Kib] and then freeing the 512 Mib block. There is no way for malloc to give this space back to the operating system (currently). To address this problem, malloc uses mmap to generate a mapped region for every long-lived allocation.

However, mmap is not perfect as the kernel must zero all of the bits in the block before it is given to the requesting process. As such, malloc tries to:

- Use mmap for large long lived allocations
- Use mmap for all really large allocations
- Use sbrk for transient (short allocations)

This is done with a sliding threshold between 128Kib and 32Mib. The threshold always goes up whenever free() is called on a block that was mmap'd (proving to the allocator that mmap was likely the wrong decision)