Glibc Heap Allocator

CS390R - UMass Amherst

Course Information

- Homework Released today

Today’s Content

- Glibc Heap
- Safely using the Heap
- Heap Chunks
- Free lists
- Allocation & Freeing procedures

Glibc

- “GNU C Library”
- Core libraries for many systems using the Linux kernel
- Standard functions and wrappers around syscalls
- Examples:
  - strlen
  - printf
  - read/write
  - malloc
  - free
  - …

Malloc

- GLIBC Memory Allocator
- Exploited since 20 years - new techniques keep coming out
- Provides process with dynamic memory

- malloc() - allocate some memory
- free()   - free some memory (give it back)
- realloc() - resize a previous allocation
- calloc() - allocate some memory and zero it out

Lets walk through an Example

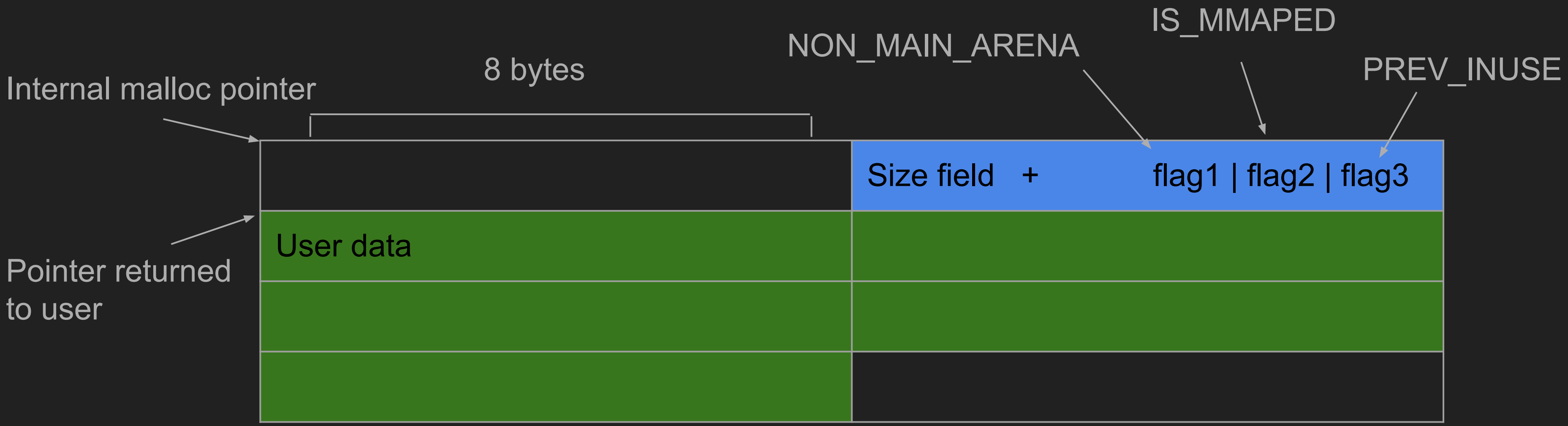
PWNDbg Heap Commands

- vis   - visualize heap chunks
- bins - print out various bins
- fastbins - print out just fast bins
- tcachebins - print out just tcache bins
- smallbins - print out just small bins
- largebins - print out just large bins
- unsortedbin - print out unsorted bin
- vmmap - print out memory mapping
- top_chunk - print out address/size of top chunk

Basic Rules when using the heap

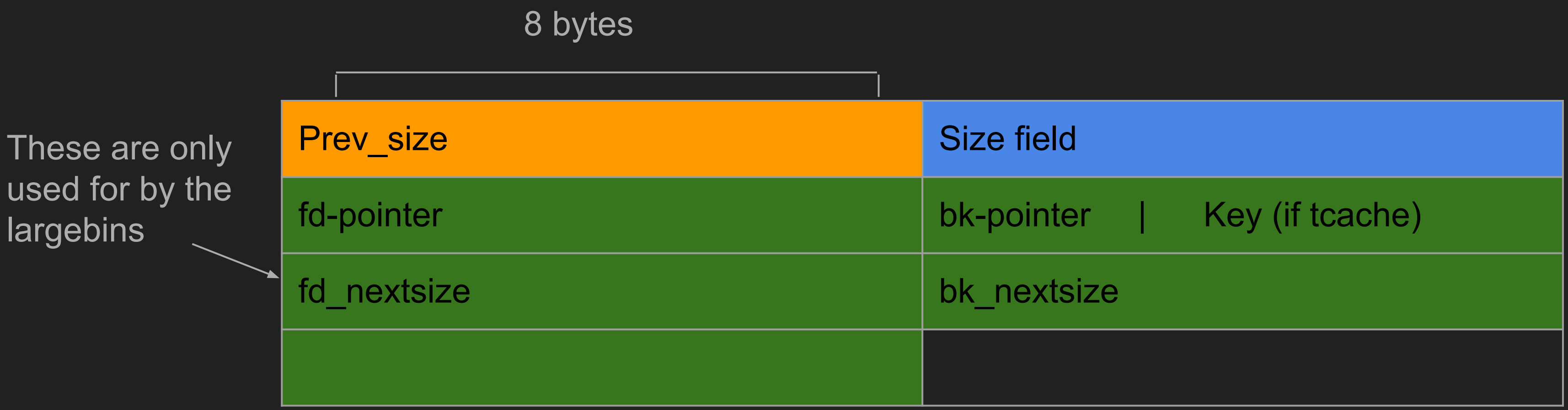
- Do not read/write to a pointer after its been free’d
  - Use after free vulnerability
- Do not use uninitialized heap space
  - Information leaks
- Do not read/write after the max size of an allocation
  - Heap Overflow
- Do not read/write before the beginning of an allocation
  - Heap Underflow
- Do not free a pointer more than once
  - Double Free
- Do not free a pointer that did not originate from malloc
  - Invalid Free

Allocated Heap Chunks



- Minimum chunk size is 32
- Chunk sizes increment by 16 bytes
- Allocations are always 16-byte aligned
- Chunks are in 1 of 2 states, in-use or free

Free'd Heap Chunks


8 bytes
These are only used for by the largebins
Prev_size
Size field
fd-pointer
bk-pointer | Key (if tcache)
fd_nextsize
bk_nextsize

- Some of the userdata-space is repurposed as metadata
- Metadata is used for linked lists
  - fd -> used by all types of free chunks
  - bk -> used by some types eg. smallbins (tcache uses this place for a key)
  - nextsize-fields -> only used by largebins
  - Prev_size field only used in chunks that support consolidation

Arenas

- For multithreaded applications, arena’s are used to give each thread a separate heap
- This lets different threads use their own heaps without needing locks
- These arena’s cannot use sbrk() to grow their heaps, so they make use of mmap() + mprotect() to emulate this behavior
- Chunks that are not in the main arena have the appropriate flag set in their metadata

Free lists

- Heap manager tracks free chunks in different free lists called “bins”
- Allocation request -> retrieve a chunk from one of these bins if possible
- Multiple different lists are used for performance reasons:
  - Tcache
    - 64 tcache bins with sizes [0x20-0x410]
  - Fastbins
    - 7 fastbins [0x20 - 0x80]
  - Unsorted bin
    - 1 unsorted bin
  - Small bins
    - 62 small bins with sizes [0x20 - 0x3f0]
  - Large bins
    - 63 large bins with sizes [0x400 - 0x80000]

Simplified Chunk Allocation Strategy

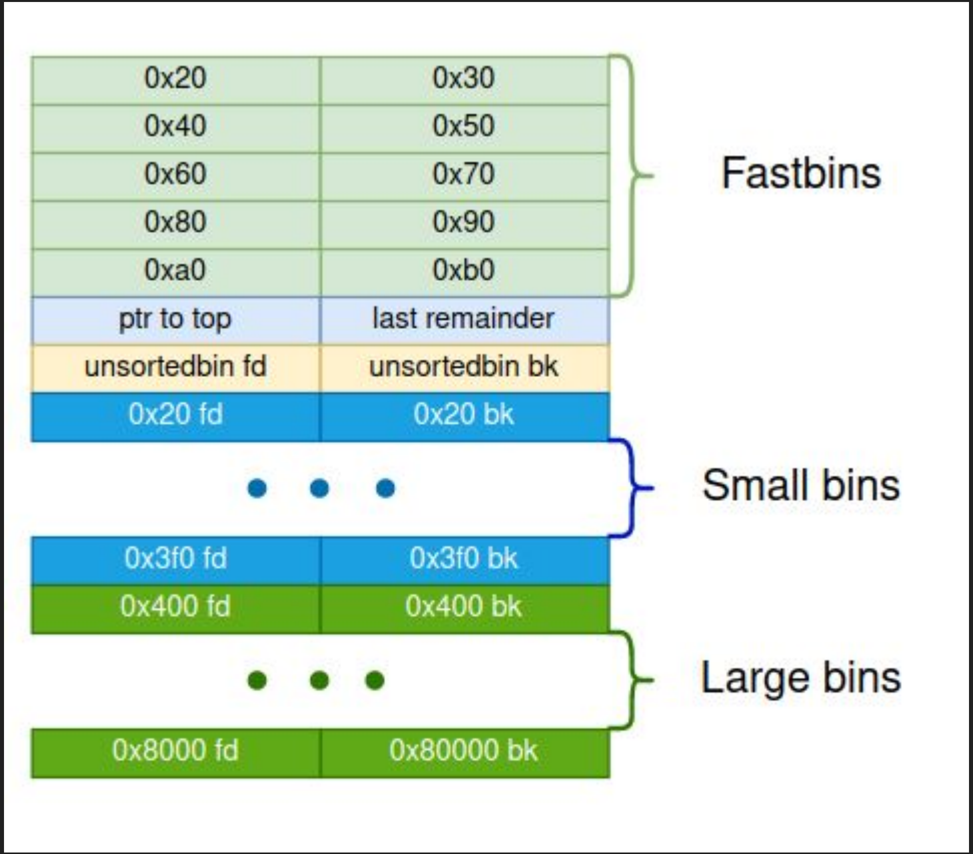
- If there is a big enough chunk in bins, use it for the new allocation
- No free chunks -> allocate memory from top chunk
  - Decreases the size value & place the new chunk at the end of the current heap region
- Top chunk does not have enough space -> kernel required
  - Sbrk() system call to ask kernel to grow the heap region
- If heap has reached the max possible size, mmap() -> create an entirely new heap somewhere else in memory
- If everything fails, return NULL

- Exception for very large allocations, immediately mmap() it. When they are later free’d, they are completely removed instead of being placed into any bins

Simplified Freeing Strategy

- If the IS_MMAPED flag is set, immediately unmap it
- If in tcache range, link into appropriate tcache bin
- If in fastbin range, link into appropriate fastbin
- Attempt to consolidate (merge with other free chunks next to it)
- Link into unsorted bin

Arena

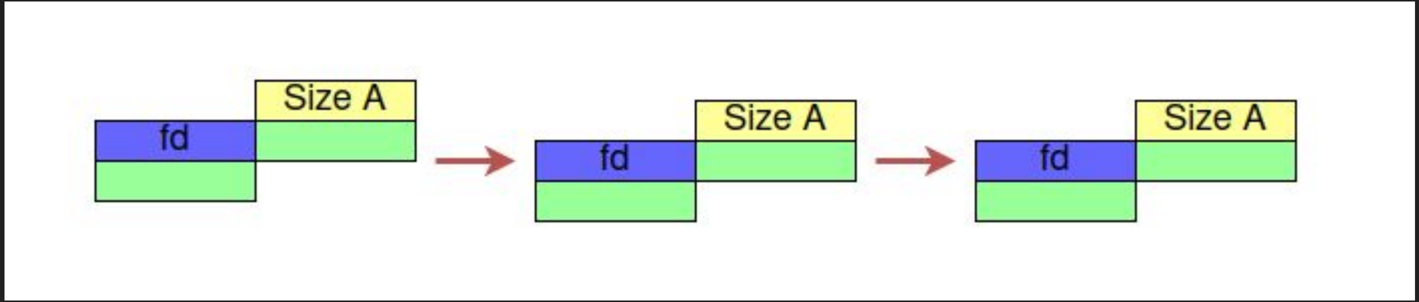

0x20
0x30
0x40
0x50
0x60
0x70
0x80
0x90
0xa0
0xb0
ptr to top
last remainder
unsortedbin fd
unsortedbin bk
0x20 fd
0x20 bk
0x3f0 fd
0x3f0 bk
0x400 fd
0x400 bk
0x8000 fd
0x80000 bk
Fastbins
Small bins
Large bins

Fast Bins - 1

- Fast bins are the second bins accessed after the tcache bins during allocation
- Keep recently free’d chunks on a fast turnaround queue via singly linked list
- Each fastbin is only responsible for a single sized chunk (0x20, 0x30, ... 0x80)
- Since everything is sorted, insertions and removals are very fast
- Chunks linked into the fastbins are not consolidated with other chunks through standard means (There are some exceptions)
- The head of each fastbin resides in its arena

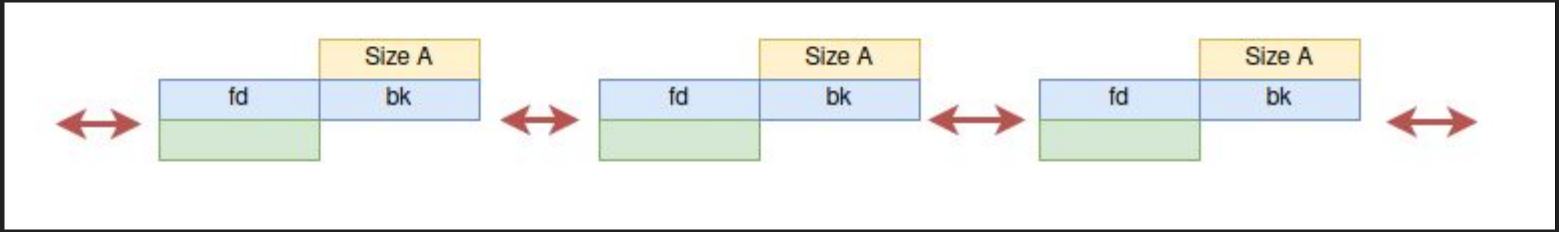
Fast Bins - 2

- Linked list pointers are stored as inline metadata within the chunk
- First 8 bytes of a chunk’s userdata are used as a forward pointer
- LIFO structure
- Unlinking a chunk from singly linked fastbin
  - copy victim chunks fd into the head of the list



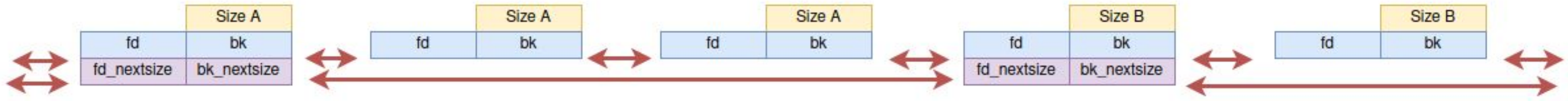
Small Bins

- 62 of them with sizes [0x20 - 0x3f0]
- One size/bin for fast insertions/deletions
- Head of each small bin in arena but inline links
- Small bins are searched after tcache, fastbin, and unsorted bins
  - if the request falls in small bin size range
- Doubly linked circular list - FIFO



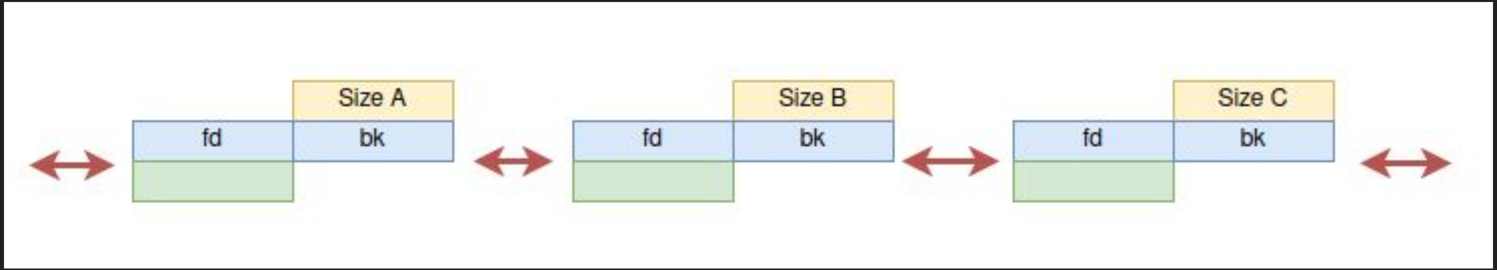
Large Bins

- 63 of them with sizes [0x400 - 0x80000]
- Store chunks within size ranges
- Insertions require traversing the list -> slower than other bins
- Head of each small bin in arena but inline links
- Large bins are searched after tcache, fastbin, and unsorted bins
  - if the request falls in large bin size range
- Doubly linked circular list - Since the list has potentially multiple copies of various sizes, and the list has to be traversed, a skip list is used so multiple chunks of one size aren't all traversed



Unsorted Bin

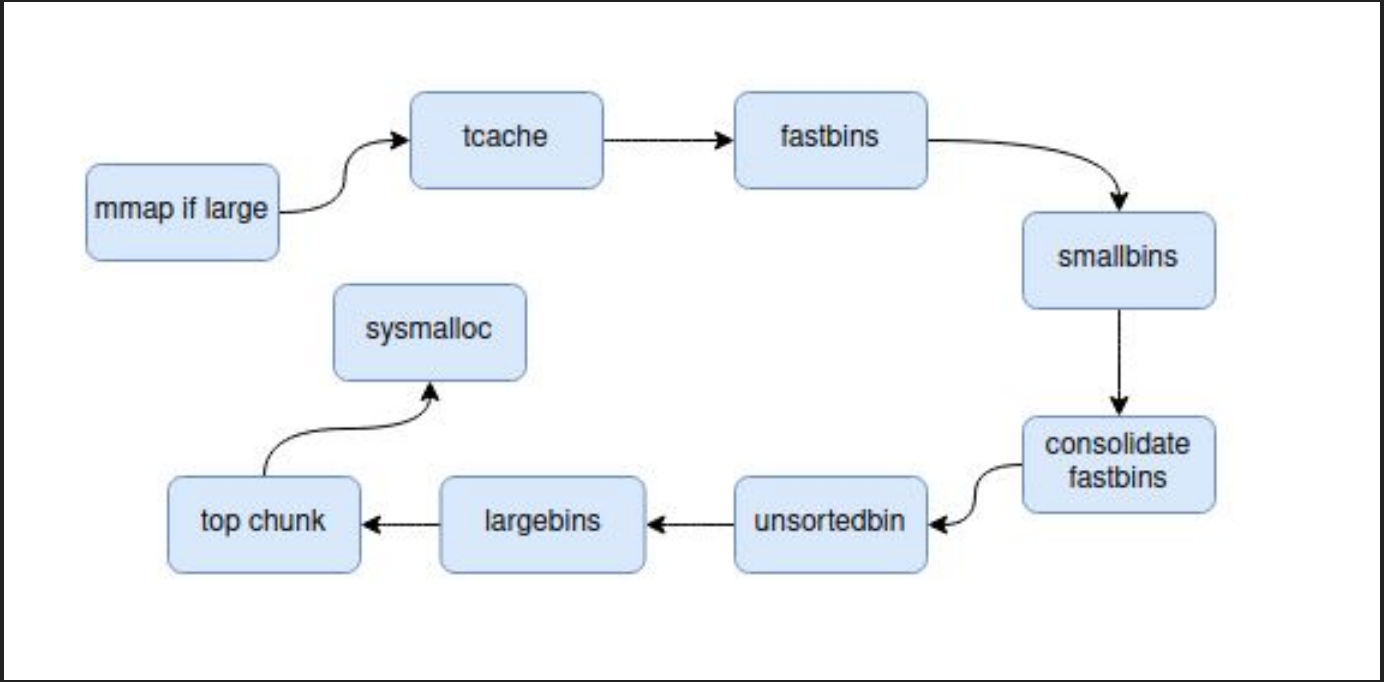
- Free’s that would go into small/large bins go into unsorted bin first
- Allocations traverse unsorted bin looking for fitting chunk, before looking in small/large bins depending on size
- During this traversal, all chunks are sorted into small or large bins
- Circular doubly linked list



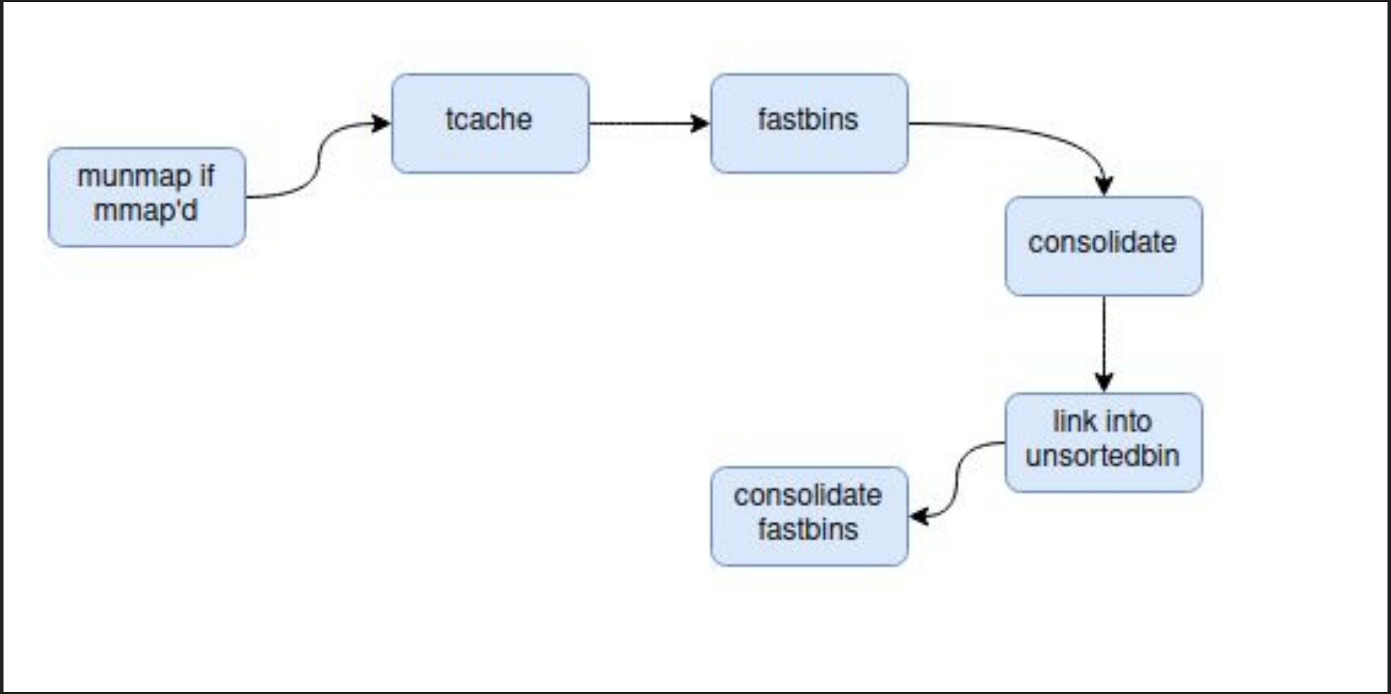
Tcache (thread-cache)

- Relatively new addition to glibc (Glibc versions >= 2.26)
- Limit on concurrent arenas, but each thread gets its own tcache
  - Reduces time spent waiting on other threads
- 64 of them with sizes [0x20 - 0x410]
- Holds up to 7 chunks per size field
- During allocations, the tcache is the first set of bins to be checked
- Similarly to fast bins, not regularly coalesced - LIFO
- Has a “count” bitmap-like structure with a field for each tcache size, and a pointer within itself that points to first entry of each bin

Full Allocation Strategy



Full Freeing Strategy


munmap if mmap'd
tcache
fastbins
consolidate
link into unsortedbin
consolidate fastbins