Memory allocation in Glibc

The safeword is “heap going!”

GyM

2016.06.30.

So What is Malloc?

- dlmalloc – Daug Lea (http://gee.cs.oswego.edu/dl/html/malloc.html) no thread safety
- ptmalloc – v1/2/3 ptmalloc2 used in glibc multi areana/heap support (http://malloc.de/en/)
- jemalloc – FreeBSD and Firefox
- libumem - solaris
- tcmalloc – thread caching, Googles super highperformance implementation
- TLSF/nedmalloc/Hoard – all super fast/super good
- Kmalloc/vmalloc – linux kernel extra GFP flags (no context switch...)
- Windows - Nope

So What is in Libc?

- Most Linux distributions use glibc
- Used to be governed by the Glibc Steering Committee
- 2009 Debian switches to EGlibc (http://www.eglibc.org/home) until 2.19
- 2015 switch back to Glibc
- Ubuntu??
- Glibc src: http://www.gnu.org/software/libc/download.html
- Glibc docs (surprisingly useful): http://www.gnu.org/software/libc/manual/html_mono/libc.html#Memory-Concepts

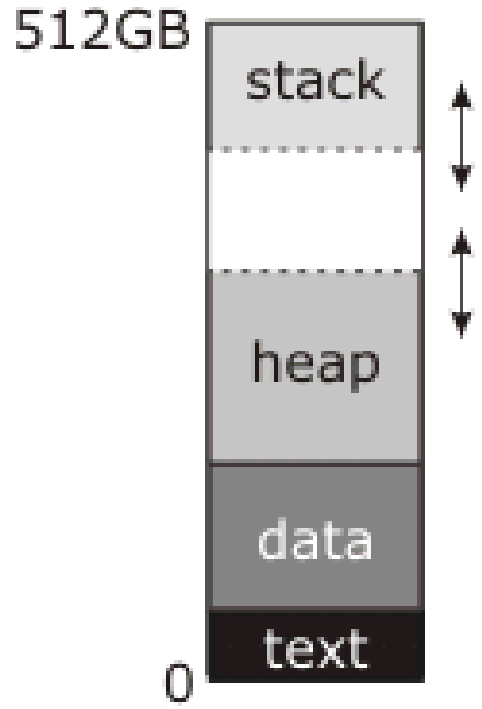
Content

- Kernel-libc responsibilities
  - Syscalls used by malloc
- General overview of malloc
  - Design goals
  - Basic concepts
- Detailed ptmalloc2 internals
  - Arenas
  - Binning
  - Chunks
  - Structures

Kernel/Loader Responsibilites

- Virtual memory management done by the kernel and hw (swapping…) except when page is locked
- Process execution (exec):
  - Create VA space
  - Map text and datat/bss segments
  - Set break address
  - Init stack
- Fork/vfork copies/shares entire address space
- Stack segment automatically grown by the OS – detect write to reserved address region
  - Page fault handled by the expand_stack() until RLIMIT_STACK (8MB) is reached
  - Stack segment does not shrink (usually)
- Permanently unmapped region between the stack and the heap when they try to grow into it → segfault (not always enforced)
- Everything else is done from the program

Far Away View of Memory



Img src: http://stackoverflow.com/questions/6988487/what-does-brk-system-call-do/6990428

Syscalls Used 1/2

- brk(addr) is used to set the high end of the heap segment to ADDR
  - Must be higher than the low address
  - Cannot overlap other segments
- sbrk(delta) is used to modify the high end address of the heap segment with delta
  - Can be +/-
  - Same rules apply
- Mmap(…) asks the OS for length bytes
  - Can hint where
  - Can give an FD to map
  - Set the protection flags
  - Anonymous mapping → no FD, page comes from the zero pool
  - Private mapping → no write back to FD
  - Actual address is returned

Syscalls Used 2/2

- Unmap: unmap a region
- Mprotect: change the protection flag (for pages)
- All implementd in: mm/map.c
- http://lxr.linux.no/linux+v2.6.28.1/mm/mmap.c#L248

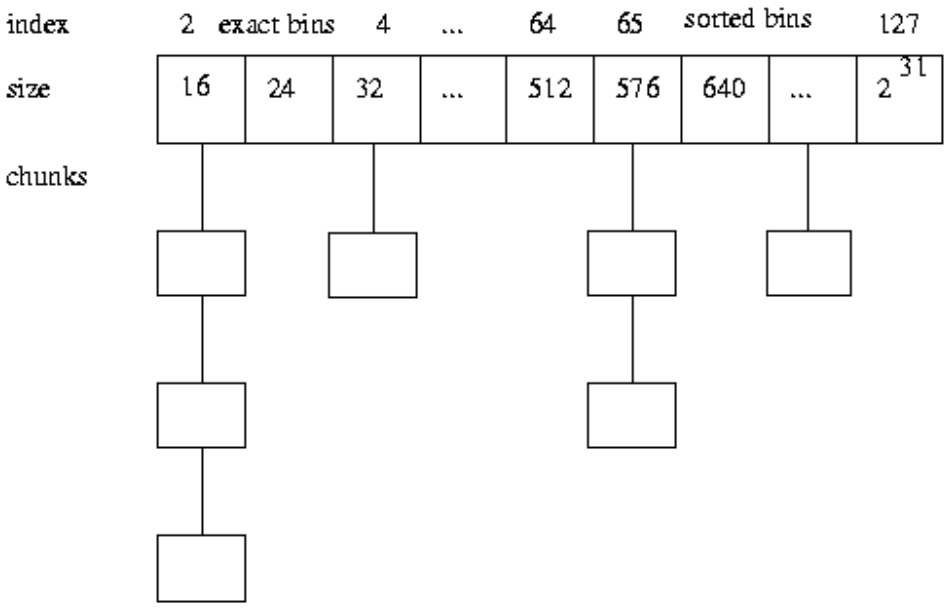
Basic Concepts of malloc 1/3

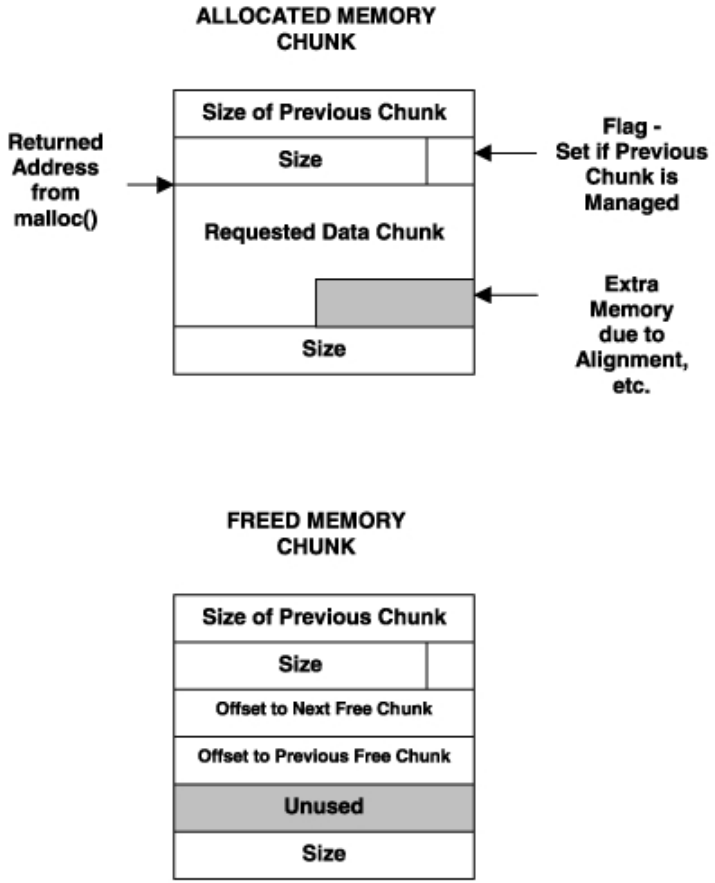
- Heap is governed by the allocator implementation
- Memory is allocated in chunks
- Programmers responsibility to keep track of allocated area – allocator only tracks the free chunks
- Uses a best fit strategy to allocate memory (reduce internal segmentation)
- Freed chunks are returned to the allocator and stored in lists (freelist)
- The heap metadata is stored on the heap itself

Basic Concepts of malloc 2/3

- Based on size the free chunks they are stored in different double linked lists (128)
- These lists are called bins
- The metadata of free chunks (list pointers) are stored in place
- Chunks store boundary tags and adjacent free chunks are coalesced (with one exception)
- If a request cannot be served from free chunks it is served from the end of heap (wilderness)
- The heap is grown (and shrinked) by sbrk

Bins and Chunks

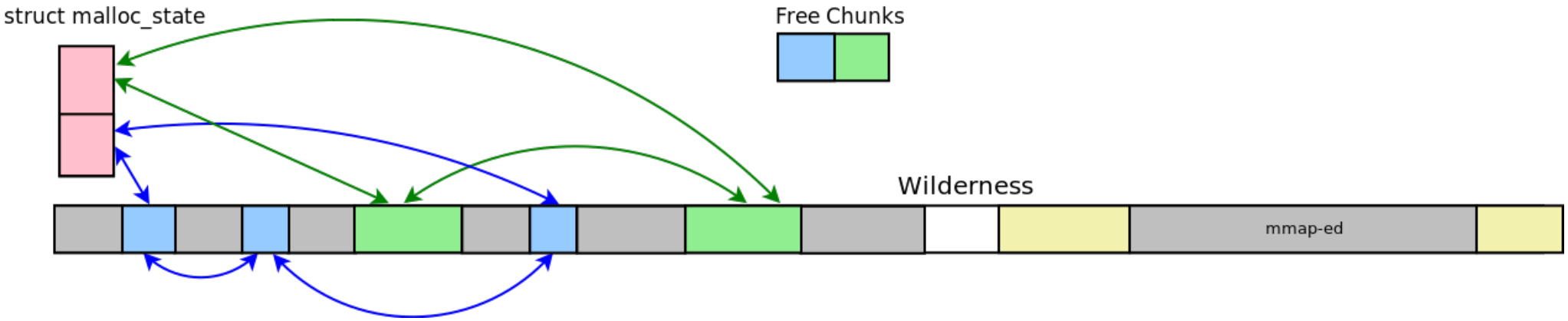




http://gee.cs.oswego.edu/dl/html/malloc.html

http://www.linuxjournal.com/article/4681

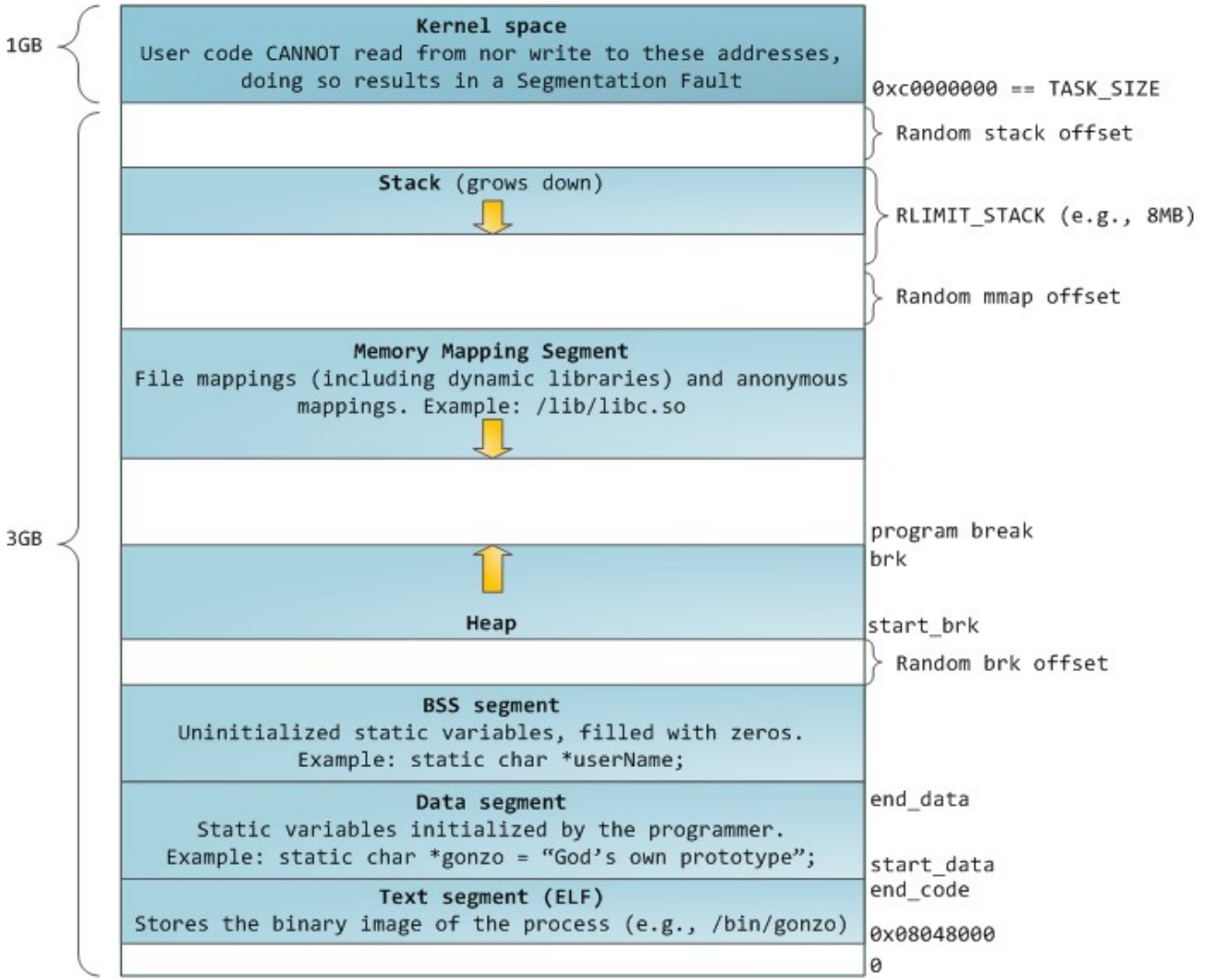
Heap Overview



Basic Concepts of malloc 3/3

- Large chunks (>= 128kb by default) are anonymous mmaped
- They are given back to the OS immediately upon free (unmapped)
- No special metada for them (uses the normal chunk structure)
- Malloc opts:
  - M_MMAP_MAX: max # of mmap chunks
  - M_MMAP_TRESHOLD: larger chunks than this value will be allocated with mmap (smaller ones might be too)
  - M_PERTURB: overwrite freed memory with unknown data -> debug uaf
  - M_TOP_PAD: set sbrk amount when growing/shrinking
  - M_TRIM_TRESHOLD: minimum size of free wilderness that is required for shrinking -> hysteresis

Not as Far Away View of Memory



http://duartes.org/gustavo/blog/post/anatomy-of-a-program-in-memory/

Implementation of malloc in Glibc

- Based on the ptmalloc2 implementation
- Which is based on dlmalloc with extra threading support
- Dlmalloc blocks on multiple access
- Ptmalloc has multiple heaps (per thread)
- Current version is 2.23
- Get source:
  - apt-get source libc6
  - http://www.gnu.org/software/libc/download.html
  - git clone git://sourceware.org/git/glibc.git
  - git checkout --track -b local_glibc-2.23 origin/release/2.23/master

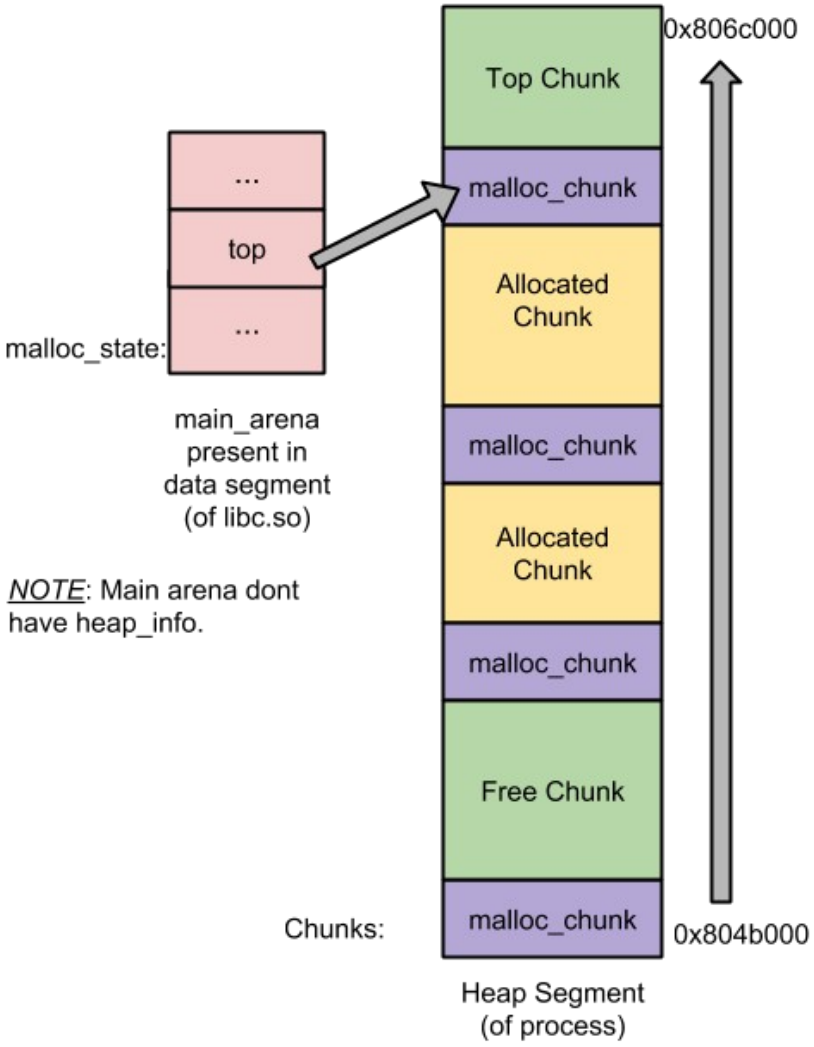
Arenas 1/2

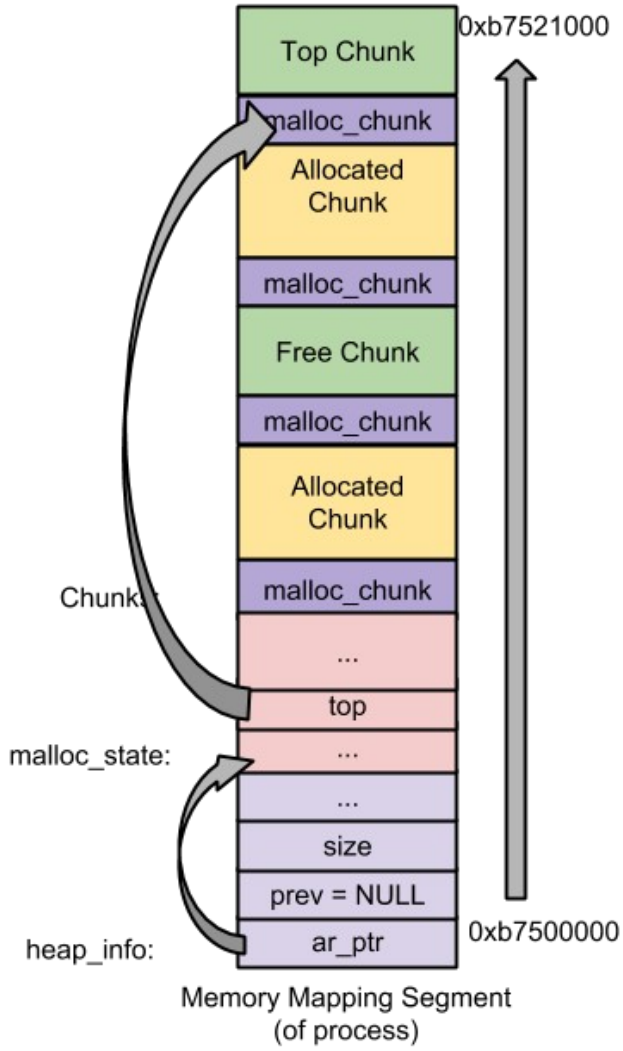
- represent a heap instance
- one process can have multiple heaps (for multiple threads)
- main_arena is created when a process begins
- per thread arena mmaped (1 MB)
- only the main arena is sbrk-d
- no blocking for the critical section (unlike dlmalloc)
- arena limit:
  - 32bit 2 * # of cores
  - 64bit 8 * # of cores

Arenas 2/2

- When limit is rached:
  - loop over arenas
  - try to lock them
  - if locked successfully return arena to thread
  - block until one is free
  - affinity set to the given arena
- one arena can have multiple heaps (when thread arena runs out of place new region is mmapped)
- main arena only have one single heap (has no heap_info)
- unlike thread arena main arena's arena header isn't stored on the heap it's a global variable in libc

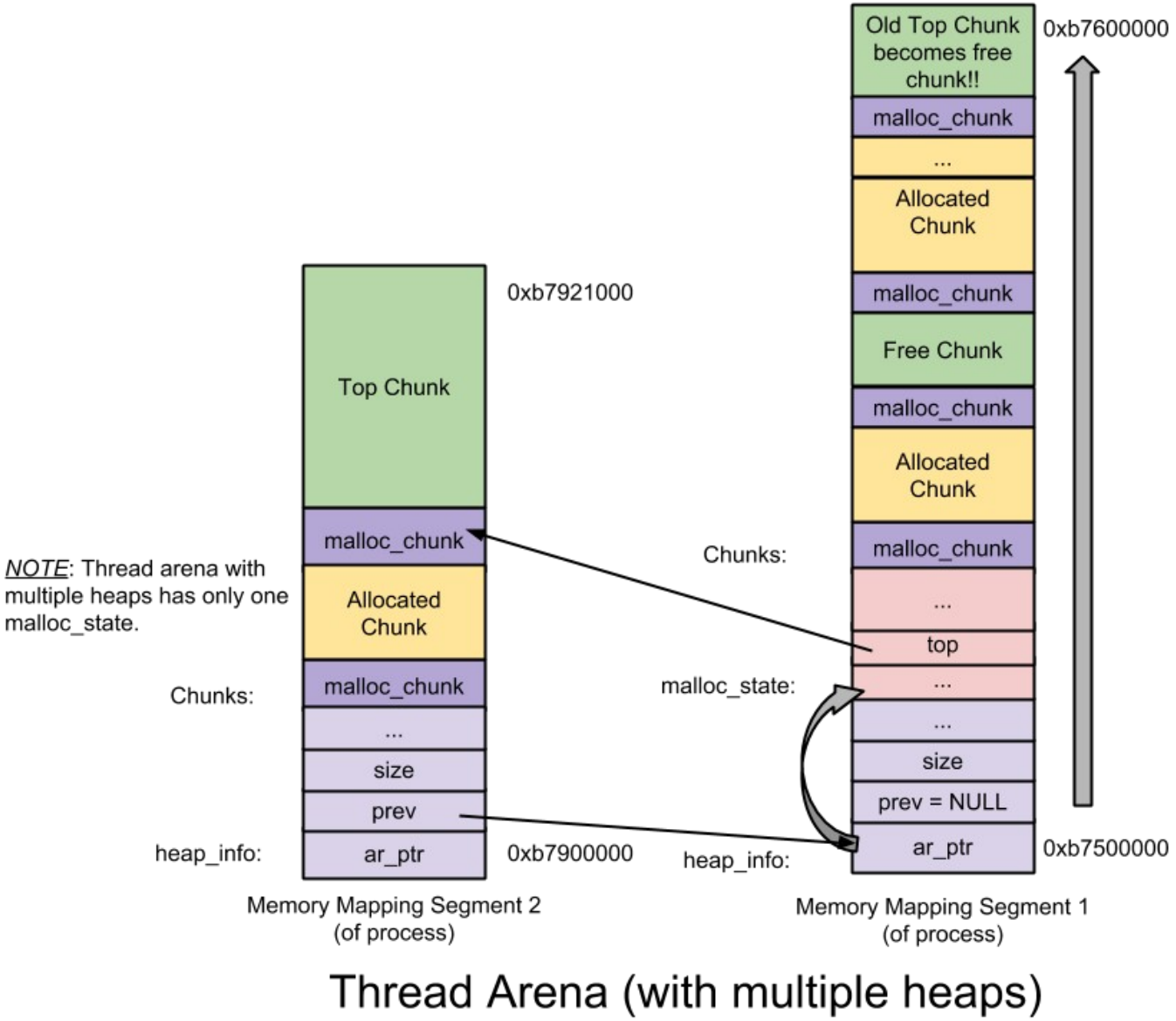
Main Arena





https://sploitfun.wordpress.com/2015/02/10/understanding-glibc-malloc/

Multi Heap



Thread Arena (with multiple heaps)

https://sploitfun.wordpress.com/2015/02/10/understanding-glibc-malloc/

Struct heap_info

```
48  typedef struct _heap_info
49  {
50    mstate ar_ptr; /* Arena for this heap. */
51    struct _heap_info *prev; /* Previous heap. */
52    size_t size;   /* Current size in bytes. */
53    size_t mprotect_size; /* Size in bytes that has been mprotected
54                             PROT_READ|PROT_WRITE.  */
55    /* Make sure the following data is properly aligned, particularly
56       that sizeof (heap_info) + 2 * SIZE_SZ is a multiple of
57       MALLOC_ALIGNMENT. */
58    char pad[-6 * SIZE_SZ & MALLOC_ALIGN_MASK];
59  } heap_info;
```

https://github.com/gymgit/glibc-2.23-tmp/blob/master/malloc/arena.c#L48

Struct malloc_state 1/2

```
1683  struct malloc_state
1684  {
1685    /* Serialize access.  */
1686    mutex_t mutex;
1687
1688    /* Flags (formerly in max_fast).  */
1689    int flags;
1690
1691    /* Fastbins */
1692    mfastbinptr fastbinsY[NFASTBINS];
1693
1694    /* Base of the topmost chunk -- not otherwise kept in a bin */
1695    mchunkptr top;
1696
1697    /* The remainder from the most recent split of a small request */
1698    mchunkptr last_remainder;
1699
1700    /* Normal bins packed as described above */
1701    mchunkptr bins[NBINS * 2 - 2];
```

https://github.com/gymgit/glibc-2.23-tmp/blob/master/malloc/malloc.c#L1683

Struct malloc_state 2/2

```
1703    /* Bitmap of bins */
1704    unsigned int binmap[BINMAPSIZE];
1705
1706    /* Linked list */
1707    struct malloc_state *next;
1708
1709    /* Linked list for free arenas.  Access to this field is serialized
1710       by free_list_lock in arena.c.  */
1711    struct malloc_state *next_free;
1712
1713    /* Number of threads attached to this arena.  0 if the arena is on
1714       the free list.  Access to this field is serialized by
1715       free_list_lock in arena.c.  */
1716    INTERNAL_SIZE_T attached_threads;
1717
1718    /* Memory allocated from the system in this arena.  */
1719    INTERNAL_SIZE_T system_mem;
1720    INTERNAL_SIZE_T max_system_mem;
1721  };
```

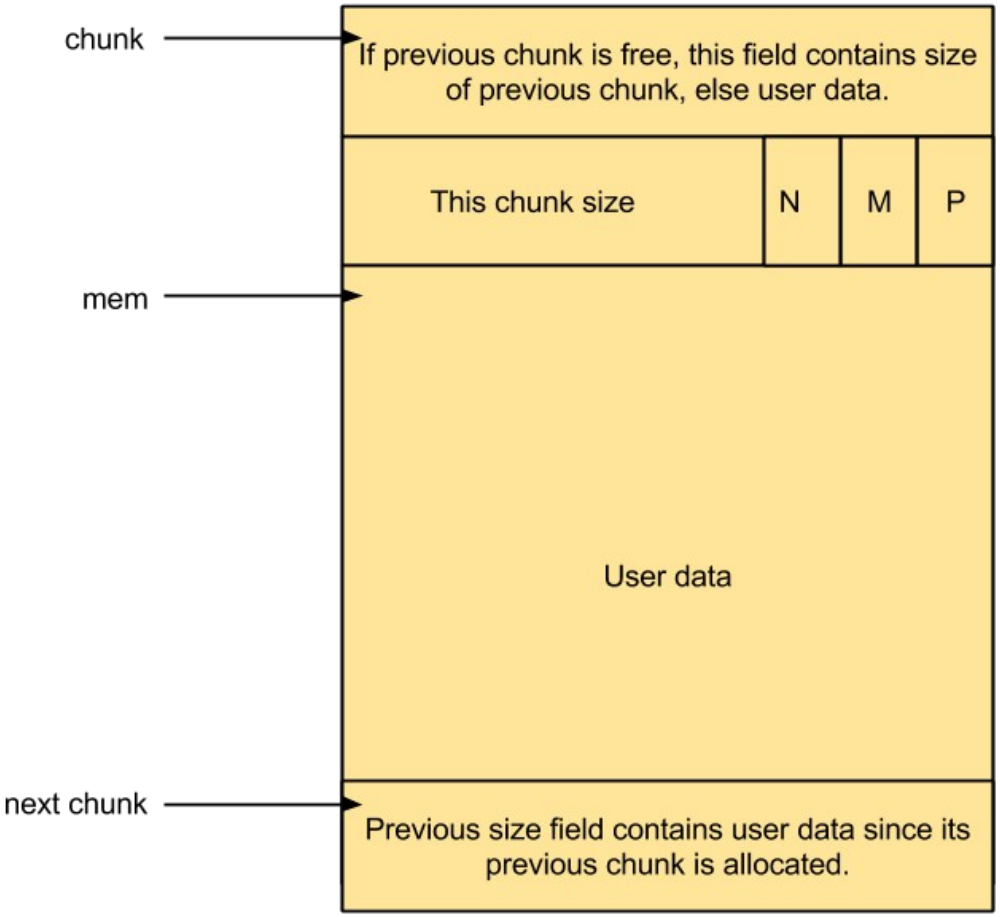
Chunks

- Basically 4 types of chunks
- All use the same struct
- https://github.com/gymgit/glibc-2.23-tmp/blob/master/malloc/malloc.c#L1108
- Malloc freelist operates on it

Struct malloc_chunk

```
1108  struct malloc_chunk {
1109
1110    INTERNAL_SIZE_T      prev_size;  /* Size of previous chunk (if free).  */
1111    INTERNAL_SIZE_T      size;       /* Size in bytes, including overhead. */
1112
1113    struct malloc_chunk* fd;         /* double links -- used only if free. */
1114    struct malloc_chunk* bk;
1115
1116    /* Only used for large blocks: pointer to next larger size.  */
1117    struct malloc_chunk* fd_nextsize; /* double links -- used only if free. */
1118    struct malloc_chunk* bk_nextsize;
1119  };
```

Allocated Chunk

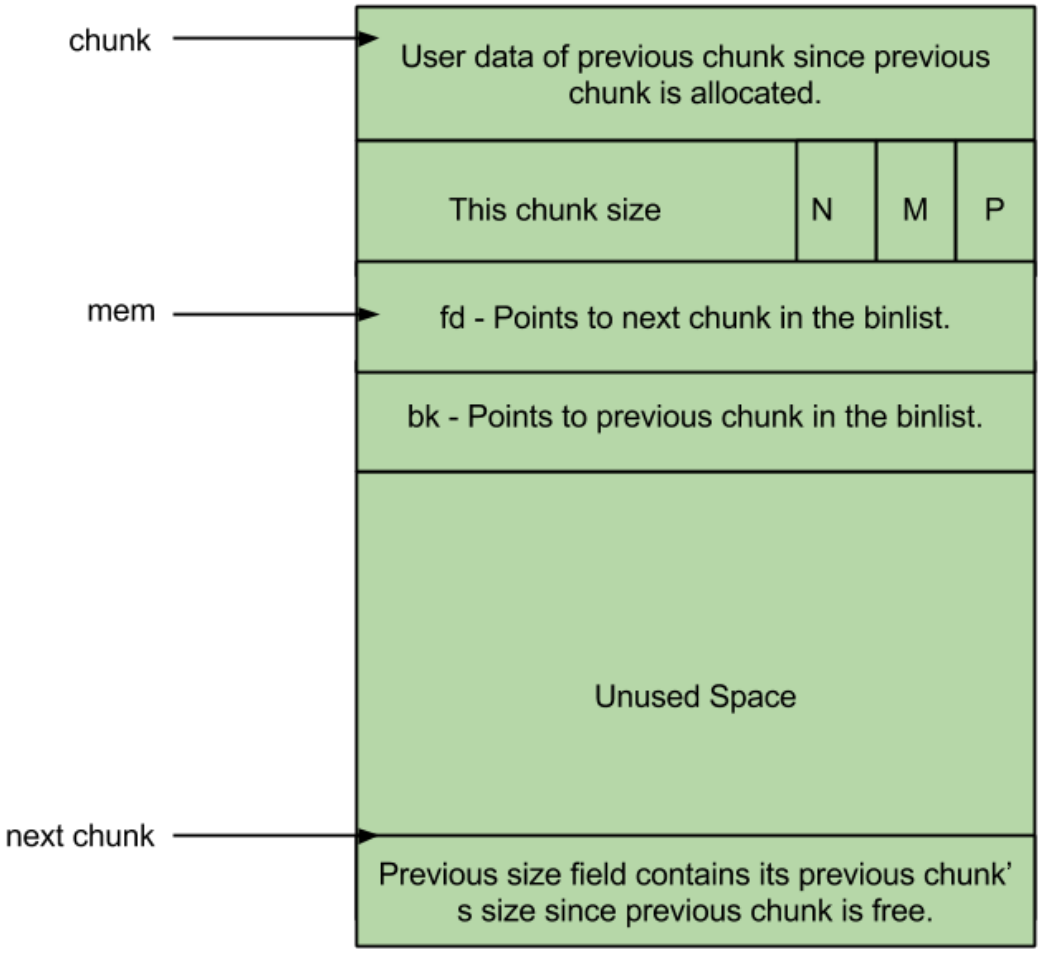


Allocated Chunk

- Chunk size min 16 last 4 bits not used (as size)
- request2size()
- PREV_INUSE (P) – This bit is set when previous chunk is allocated.
- IS_MMAPPED (M) – This bit is set when chunk is mmapped.
- NON_MAIN_ARENA (N) – This bit is set when this chunk belongs to a thread arena.

https://sploitfun.wordpress.com/2015/02/10/understanding-glibc-malloc/

Free Chunk



Free Chunk

- **Fd – points to the next free chunk in the same bin**
- **Bk – previous chunk in the same bin**

https://sploitfun.wordpress.com/2015/02/10/understanding-glibc-malloc/

Top Chunk

- Also known as wilderness
- Serves requests when there are no free chunks
- Its size field is checked when deciding if sbrk needed to server a request – HoF
  - https://github.com/gymgit/glibc-2.23-tmp/blob/master/malloc/malloc.c#L3793
- Not in any bin
- If larger than requested size split in two
- Remainder becomes the new top chunk and extended with sbrk or mmap if required
  - Done in sysmalloc

Last Remainder

- Remainder from the most recent split
- Put into the unsorted bin
- Next small request is served from it (if possible)
- Helps to achieve locality
- https://github.com/gymgit/glibc-2.23-tmp/blob/master/malloc/malloc.c#L3484

Bins

- Freelist data structure, they hold the free chunks
- Different bins based on size
- Speed/order of serving requests:

  fast > unsorted > small > large
- They are arrays of malloc_chunk*
- 126 + 10 bins
- Initialized during the first call to malloc
- https://github.com/gymgit/glibc-2.23-tmp/blob/master/malloc/malloc.c#L1797

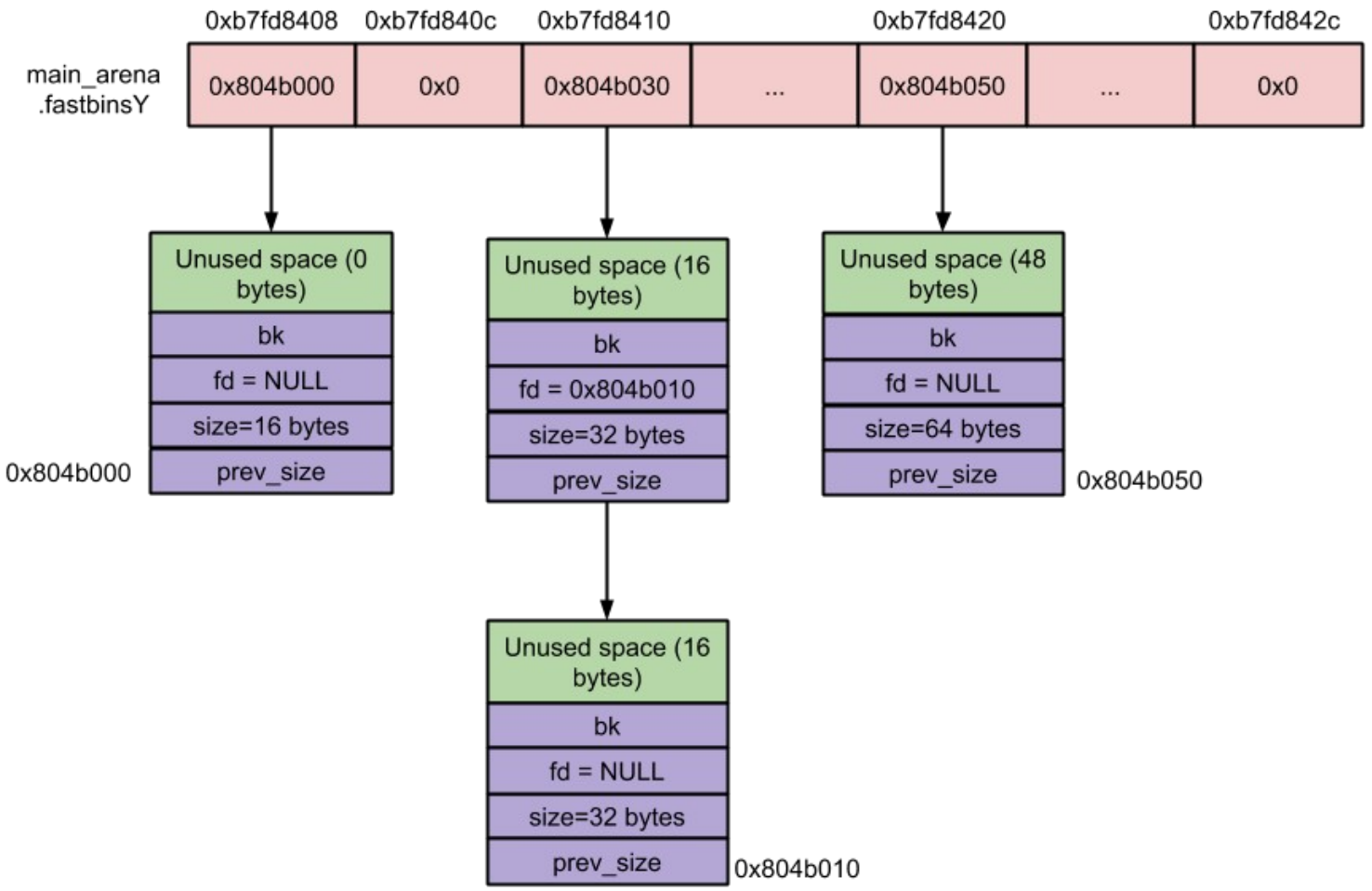
Fast Bins 1/2

- 16-80b → fast chunks
- 10 bin by default, incremented by 8 bytes
- Each contains a single linked list (chunks are not removed from middle)
- Chunks in the same bins have the same size
- No coalescing – free chunks can be next to each other
- Maximum size of memory handled by fastbins is predetermined (static INTERNAL_SIZE_T global_max_fast;)
- https://github.com/gymgit/glibc-2.23-tmp/blob/master/malloc/malloc.c#L3365

Fast Bins 2/2

- Allocation:
  - When empty handled by the smallbin code
  - Fastbin index is calculated from request size to get the appropriate binlist
  - First chunk from the binlist is removed and returned
- Free:
  - Index is calculated
  - Added to the front
- Works as LIFO

Fast Bins



Fast Bin Snapshot

https://sploitfun.wordpress.com/2015/02/10/understanding-glibc-malloc/

Unsorted Bin

- when small or large chunks are freed they are added to here
- they are given a second chance to be reused
- if not, they are sorted to appropriate bins
- The first bin with a circular double linked list
- chunks with any size
- Unlink: https://github.com/gymgit/glibc-2.23-tmp/blob/master/malloc/malloc.c#L1411

Small Bins

- Less than 512 bytes
- 62 bins (from the 2nd) with circular double linked list
- Add to the front delete from back (FIFO)
- Bins 8 bytes apart, all chunks are the same size in a bin
- Two adjecent chunks are combined
- Allocation:
  - Initially -> all null (points to itself), unsorted bin tries to service it
  - last chunk from its list is removed and returned
- Free:
  - check if prev or next is free
  - if so: unlink those from their list coalesce them and add to unsorted bin
- https://github.com/gymgit/glibc-2.23-tmp/blob/master/malloc/malloc.c#L3402

Large Bins 1/2

- >= 512 bytes
- 63 bins with a circular double linked list
- random access
- 32 bins with size 64 bytes apart: bin 65 - 512-568 bin 66 - 576-632 ...
- 16 bins 512 bytes apart
- 8 bins 4096 bytes apart
- 4 bins 32768 bytes apart
- 2 bins 262144 bytes apart
- 1 bin for the rest
- chunks are not the same size in the same bin they are stored in a **decreasing order**
- adjecent chunks are combined into a single free chunks
- https://github.com/gymgit/glibc-2.23-tmp/blob/master/malloc/malloc.c#L3601

Large Bins 2/2

- Allocation:
  - initially: all empty
    - next largest bin code serves it
    - https://github.com/gymgit/glibc-2.23-tmp/blob/master/malloc/malloc.c#L3674
    - If fails served from wilderness
  - bin selected than walked from the rear to front to find the best match
  - once found the chunk is split and the remainder chunk is added to the unsorted bin
  - if largest chunk in bin is too small, serve the request from the next bin
  - if all empty, served from the top chunk
- Free is done same as for small bins

Debugging malloc

- **Ltrace**
- **GDB**
- **LD_PRELOAD**
- **Malloc hooks**
  - http://www.gnu.org/software/libc/manual/html_node/Hooks-for-Malloc.html
- **Villoc**
  - https://github.com/wapiflapi/villoc

Heap Based Attacks (next week?)

- Dlmalloc unlink - http://phrack.org/issues/61/6.html
- Shellphish how2heap - https://github.com/shellphish/how2heap
- House of * - https://sploitfun.wordpress.com/2015/03/04/heap-overflow-using-malloc-maleficarum/
- Understanding the heap by breaking it - https://www.blackhat.com/presentations/bh-usa-07/Ferguson/Whitepaper/bh-usa-07-ferguson-WP.pdf

Recap

- Syscalls used by malloc and prog loading
- Basic concepts of malloc
- Arenas and thread safety
- Structures on the heap
- Chunks
- Bins and freelists

References

- Libc memory concepts - https://www.blackhat.com/presentations/bh-usa-07/Ferguson/Whitepaper/bh-usa-07-ferguson-WP.pdf
- Anatomy of progmem - https://www.blackhat.com/presentations/bh-usa-07/Ferguson/Whitepaper/bh-usa-07-ferguson-WP.pdf
- Syscalls used - https://sploitfun.wordpress.com/2015/02/11/syscalls-used-by-malloc/
- https://www.blackhat.com/presentations/bh-usa-07/Ferguson/Whitepaper/bh-usa-07-ferguson-WP.pdf
- http://phrack.org/issues/67/8.html
- Libc - http://www.gnu.org/software/libc/download.html
- Understanding glibc malloc - https://sploitfun.wordpress.com/tag/ptmalloc/