Newer is Better!

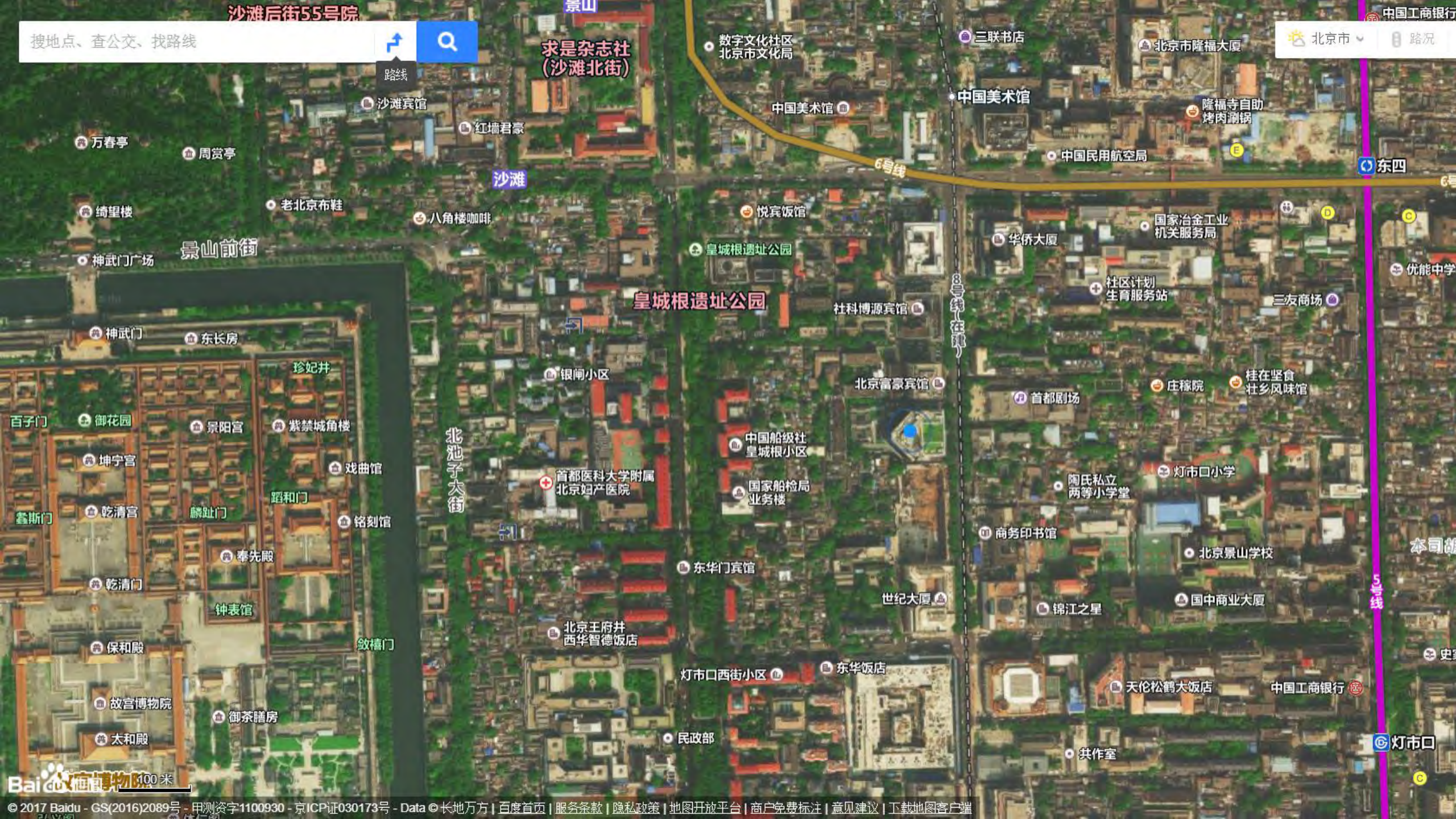
2017 CPP-Summit

Linux应用内存管理与错误诊断

在调试器里看glibc堆和Valgrind

张银奎

《软件调试》和《格蠹汇编》作者



张银奎，Raymond Zhang，格蠹老雷，《软件调试》和《格蠹汇编》作者
http://advdbg.org http://weibo.com/dbgger 格友公众号

内存问题是软件世界的住房问题

《戊申岁六月中遇火》
陶渊明
草庐寄穷巷，甘以辞华轩。
正夏长风急，林室顿烧燔。
一宅无遗宇，舫舟荫门前。
迢迢新秋夕，亭亭月将圆。
果菜始复生，惊鸟尚未还。
中宵伫遥念，一盼周九天。

陶渊明（约365年—427年）

杜甫（712年—770年）

沈从文（1902—1988

从万米高空鸟瞰

- 核心思想：
  - 分层管理
  - 批发与零售
  - 隐藏内部细节，用户接口尽可能简单
    - malloc
    - kmalloc
- 堆是针对小额分配而设计的，但是为了编程简单，也支持分配大块

malloc()

- The standard C runtime environment lets a user program allocate additional memory

```
void *malloc( unsigned long nbytes );
```

- This memory is released after being used

```
void free( void *vaddr );
```

- Using suitable kernel modules we can see how the kernel keeps track of this memory

ptmalloc

- 基于Doug Lea开发的dlmalloc，主要目的是增加多线程支持
  - Pt或为POSIX thread之缩写
- On Linux systems, ptmalloc has been put to work for years as part of the GNU C library.
- 迄今为止，共有三个版本
- 从glibc-2.3.x.开始使用的是ptmalloc2
- http://www.malloc.de/en/

ptmalloc - a multi-thread malloc implementation

版本1的readme文件

Wolfram Gloger (wg@malloc.de)

19 Dec 1999

Introduction

ptmalloc.c is a modified version of Doug Lea's malloc-2.6.4 implementation (available seperately from ftp://g.oswego.edu/pub/misc) that I adapted for multiple threads, while trying to avoid lock contention as much as possible. Many thanks should go to Doug Lea (dl@cs.oswego.edu) for the great original malloc implementation.

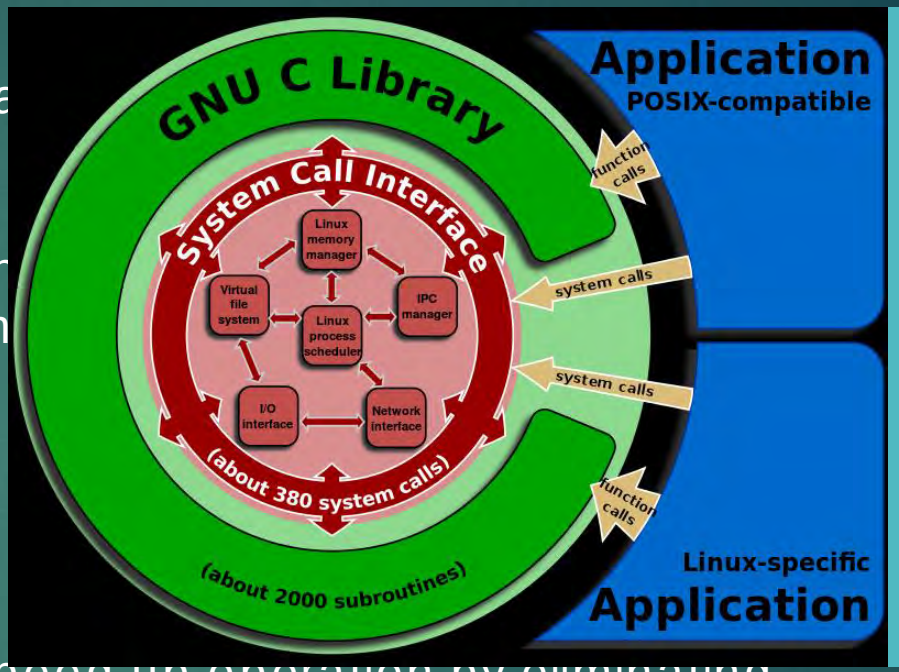
Version 2.3

* Masahide Washizawa contributed iconv modules for IBM1163 and IBM1164 charsets.

* iconv (the program and the interface) now a
options like //TRANSLIT) to mean "use charset

* localedef can now transliterate characters in
the provided charmap. The information from

* Prelinking support was added for ELF targets.
tools and recent versions of the GNU binutils.
Jelinek.



GNU C library始于1988年，2.3发布于2002-10-02

https://sourceware.org/glibc/wiki/Glibc%20Timeline

* Read-only stdio streams now use mmap to speed up operation by eliminating copying and buffer underflows. To use add 'm' to the mode string of the fopen/fdopen/freopen call. Implemented by Ulrich Drepper.

* The malloc functions were completely rewritten by Wolfram Gloger based on Doug Lea's malloc-2.7.0.c.

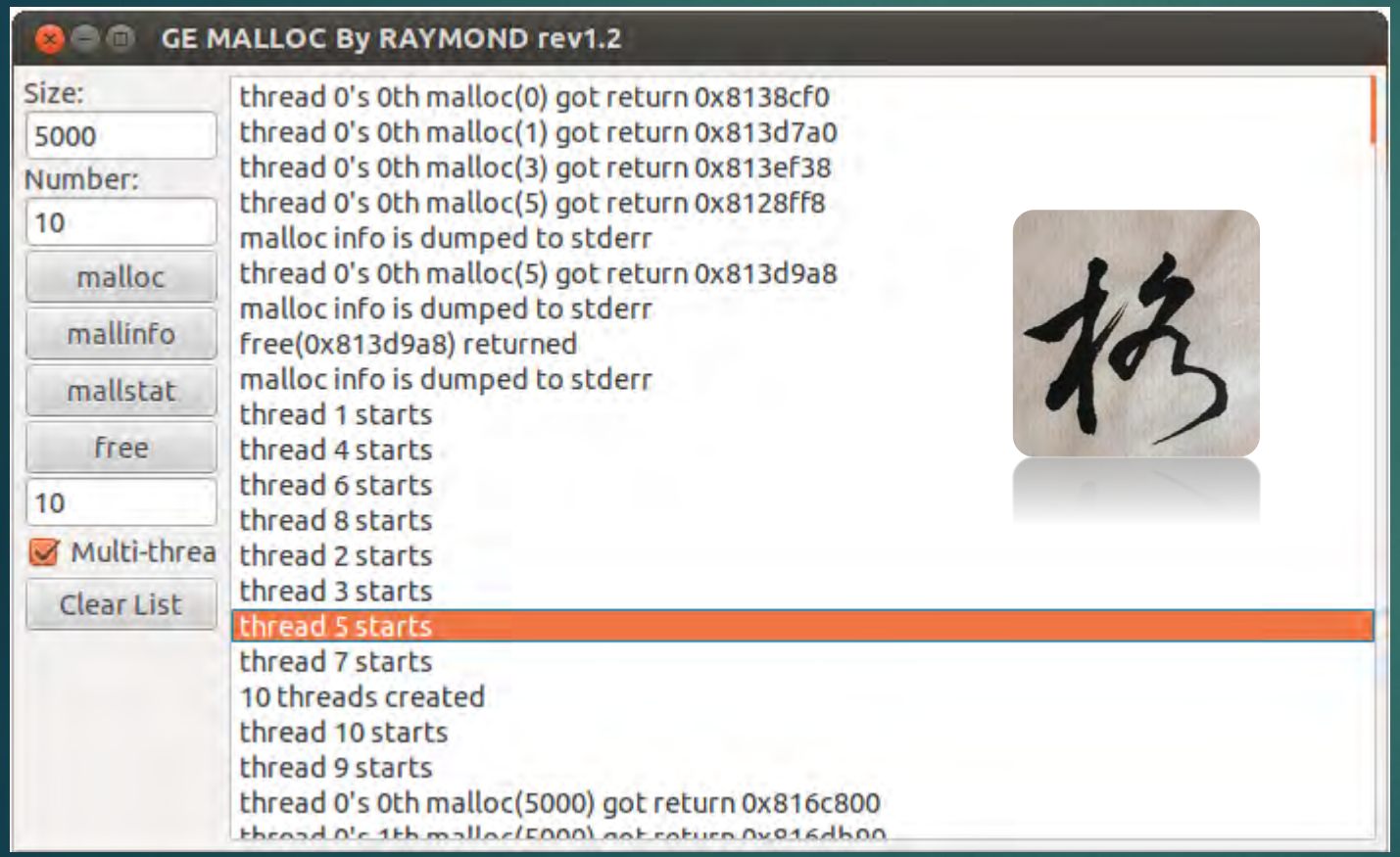
* Isamu Hasegawa contributed a completely new and POSIX-conformant implementation of regex.

源文件(ptmalloc2)

c:\dig\training\Linux\heap\ptmalloc2*.*

Name	Ext	↓Size	Date	Attr
[..]		<DIR>	2017/07/05 11:04	----
[sysdeps]		<DIR>	2004/07/26 02:31	----
malloc	c	171,889	2004/11/05 01:31	-a--
arena	c	21,734	2004/11/05 22:42	-a--
hooks	c	18,755	2004/11/05 22:42	-a--
malloc	h	9,993	2004/08/08 20:34	-a--
Readme		7,438	2004/11/06 20:36	-a--
Makefile		6,903	2006/06/05 19:13	-a--
ChangeLog		6,108	2006/06/05 19:07	-a--
t-test1	c	5,836	2004/11/04 22:58	-a--
malloc-stats	c	5,236	2004/08/09 04:35	-a--
t-test2	c	4,715	2004/11/04 23:01	-a--
t-test	h	2,815	2004/11/06 20:26	-a--
tst-mstats	c	2,717	2004/08/08 20:09	-a--
tst-mallocstate	c	1,917	2002/01/19 01:15	-a--
COPYRIGHT		976	2006/01/02 03:00	-a--
lran2	h	837	1996/12/06 00:42	-a--

gemalloc



- 交互式的堆探索辅助工具
- 支持指定块大小(默认十进制，0x前缀十六进制)和块个数
- 支持多线程
  - 线程动态创建和退出
- Mallinfo
- mallstat


概览
Arena
批发
分配和释放
Valgrind

Arena

```
static struct malloc_state main_arena =
 {
   .mutex = MUTEX_INITIALIZER,
   .next = &main_arena
 };
```

以静态变量形式定义主场地（main arena），与主堆（main heap）关联

```
(gdb) ptype *av
type = struct malloc_state {
    mutex_t mutex;
    int flags;
    mfastbinptr fastbinsY[10];
    mchunkptr top;
    mchunkptr last_remainder;
    mchunkptr bins[254];
    unsigned int binmap[4];
    struct malloc_state *next;
    struct malloc_state *next_free;
    size_t system_mem;
    size_t max_system_mem;
}
(gdb) p *av
$2 = {mutex = 1, flags = 0, fastbinsY = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, top = 0x0,
  last_remainder = 0x0, bins = {0x0 <repeats 254 times>}, binmap = {0, 0, 0, 0}, next = 0xb7fc7440,
next_free = 0x0,
  system_mem = 0, max_system_mem = 0}
```

```
(gdb) p &main_arena
$11 = (struct malloc_state *) 0xb7284440
(gdb) p main_arena
$12 = {mutex = 0, flags = 0, fastbinsY = {0x9b9cf18, 0x9ba0c40, 0x9ba1ad8, 0x9b9efd0, 0x9ba0718, 0x9ba4ff8, 0x0, 0x0, 0x0,
  top = 0x9bb5a68, last_remainder = 0x9b75660, bins = {0x9b75660, 0x9b75660, 0xb7284478, 0xb7284478, 0xb7284480, 0xb7
    0xb7284488, 0xb7284488, 0x9b93790, 0x9ab6358, 0xb7284498, 0xb7284498, 0x9ba5198, 0x9b9fd90, 0x9ba33b8, 0x9b988
    0x9ba31f8, 0x9b986b0, 0x9b94ef8, 0xb72844c0, 0xb72844c0, 0x9b9fd20, 0x9b9fd20, 0xb72844d0, 0xb72844d0, 0x9ba4b0
    0xb72844e0, 0xb72844e0, 0x9b9ce78, 0x9b9ce78, 0x9b743f8, 0x9b743f8, 0xb72844f8, 0xb72844f8, 0xb7284500, 0xb72845
    0xb7284508, 0xb7284508, 0xb7284510, 0xb7284510, 0x9ba50b0, 0x9ba50b0, 0xb7284520, 0xb7284520, 0xb7284528, 0xb72
    0xb7284530, 0xb7284530, 0xb7284538, 0xb7284538, 0xb7284540, 0xb7284540, 0xb7284548, 0xb7284548, 0xb7284550, 0xb7
    0xb72845                                                                                    63f60, 0x9b63f6
    0xb72845                                                                                    b72845a0, 0xb7
    0x9ba4dc                                                                                    xb72845c8, 0xb
    0x9bb567                                                                                    b5450, 0x9bb54
    0xb72845f8, 0xb7284600, 0xb7284600, 0xb7284608, 0xb7284608, 0xb7284610, 0xb7284610, 0xb7284618, 0xb7284618, 0xb7
    0xb7284620, 0xb7284628, 0xb7284628, 0xb7284630, 0xb7284630, 0xb7284638, 0xb7284638, 0xb7284640, 0xb7284640, 0xb7
    0xb7284648, 0xb7284650, 0xb7284650, 0xb7284658, 0xb7284658, 0xb7284660, 0xb7284660, 0xb7284668, 0xb7284668, 0xb7
    0xb7284670, 0xb7284678, 0xb7284678, 0xb7284680, 0xb7284680, 0xb7284688, 0xb7284688, 0xb7284690, 0xb7284690, 0xb7
    0xb7284698, 0xb72846a0, 0xb72846a0, 0xb72846a8, 0xb72846a8, 0xb72846b0, 0xb72846b0, 0xb72846b8, 0xb72846b8, 0x
    0xb72846c0, 0xb72846c8, 0xb72846c8, 0xb72846d0, 0xb72846d0, 0xb72846d8, 0xb72846d8, 0xb72846e0, 0xb72846e0, 0x
    0xb72846e8, 0xb72846f0, 0xb72846f0, 0xb72846f8, 0xb72846f8, 0xb7284700, 0xb7284700, 0xb7284708, 0xb7284708, 0xb72
    0xb7284710, 0xb7284718, 0xb7284718, 0xb7284720, 0xb7284720, 0xb7284728, 0xb7284728, 0xb7284730, 0xb7284730, 0xb7
    0xb7284738, 0xb7284740, 0xb7284740, 0xb7284748, 0xb7284748, 0xb7284750, 0xb7284750, 0xb7284758, 0xb7284758, 0xb7
    0xb7284760, 0xb7284768, 0xb7284768, 0xb7284770, 0xb7284770, 0xb7284778, 0xb7284778, 0xb7284780, 0xb7284780, 0xb7
    0xb7284788...}, binmap = {6553572, 205102, 0, 67108864}, next = 0xb5000010, next_free = 0x0, system_mem = 1257472,
  max_system_mem = 1433600}
```

An arena is a configuration of malloc_chunks together with an array of bins.

USE_ARENAS (default: the same as HAVE_MMAP)
Enable support for multiple arenas, allocated using mmap().

HAVE_MMAP (default: defined as 1)
Define to non-zero to optionally make malloc() use mmap() to allocate very large blocks.

当支持多线程时，总是需要多个Arenas，也就是将USE_ARENAS定义为1

Main Arena

只有一个

- Arena管理信息区是静态定义的
- 可分配区域是传统的堆区域
- 通过brk机制调整大小

Non-main Arena

0 - 2*(number of cpu cores)

- 使用mmap机制动态批发
- 可以有多个子堆
- 每个子堆一定是连续的

A heap is a single contiguous memory region holding (coalesceable) malloc_chunks.

-- arena.c

这里的堆最好理解为子堆，在ptmalloc里的heap大多时候是这个用法

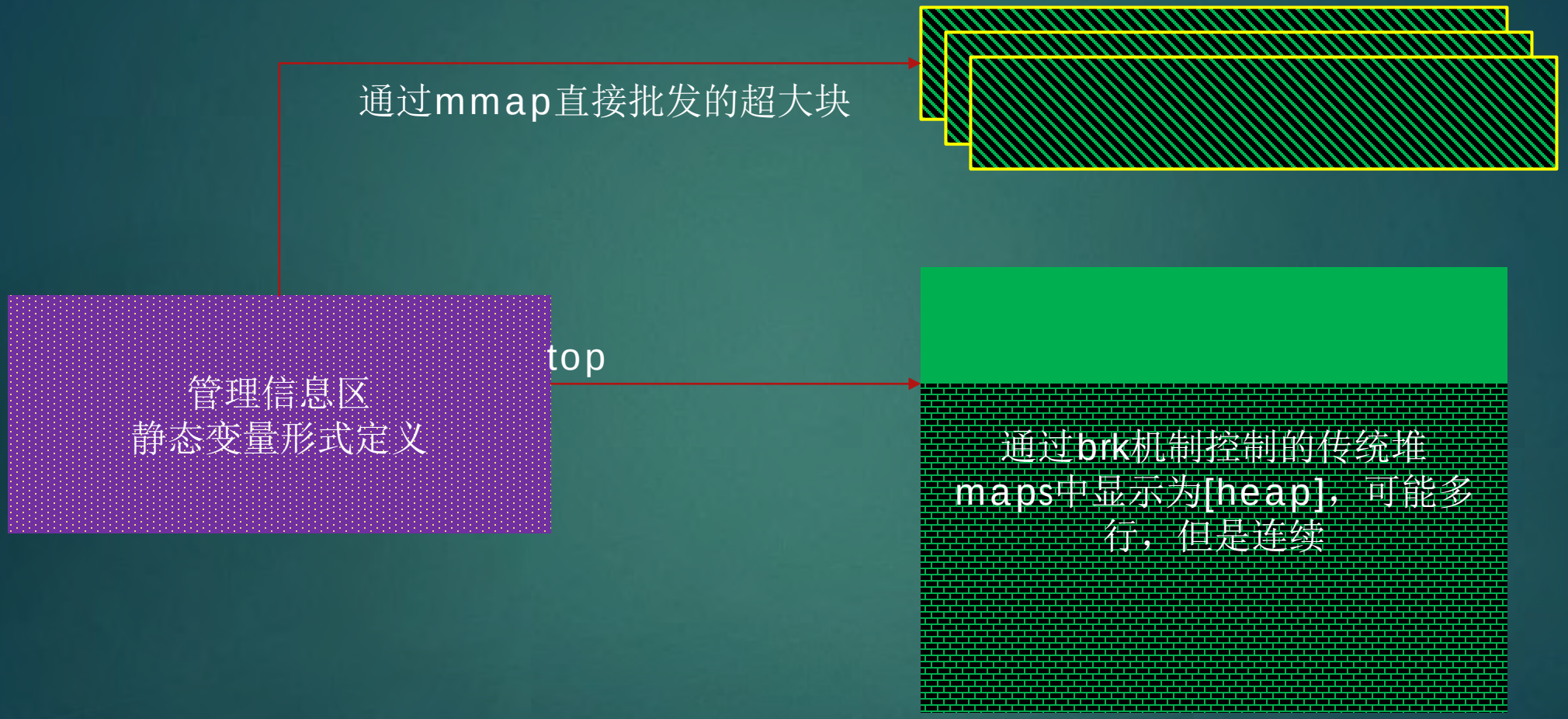
在ptmalloc中，arena是顶级的实体，一个arena可以有一或者多个（子）堆

_heap_info

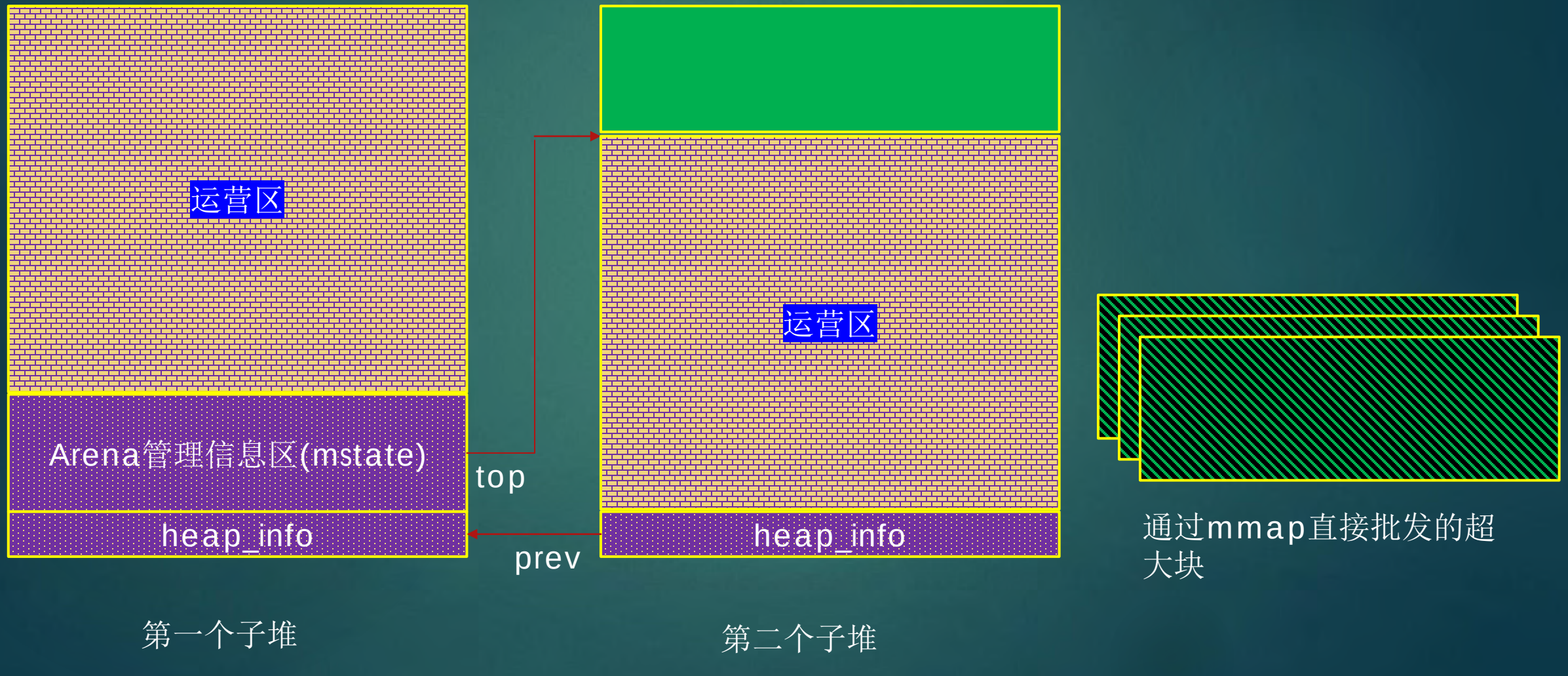
```
/* A heap is a single contiguous memory region holding (coalesceable)
   malloc_chunks.  It is allocated with mmap() and always starts at an
   address aligned to HEAP_MAX_SIZE.  */

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

Main arena布局



Non-main arena布局



/* find the heap and corresponding arena for a given ptr */

```
#define heap_for_ptr(ptr) \
 ((heap_info *)((unsigned long)(ptr) & ~(HEAP_MAX_SIZE-1)))
#define arena_for_chunk(ptr) \
 (chunk_non_main_arena(ptr) ? heap_for_ptr(ptr)->ar_ptr : &main_arena)
```

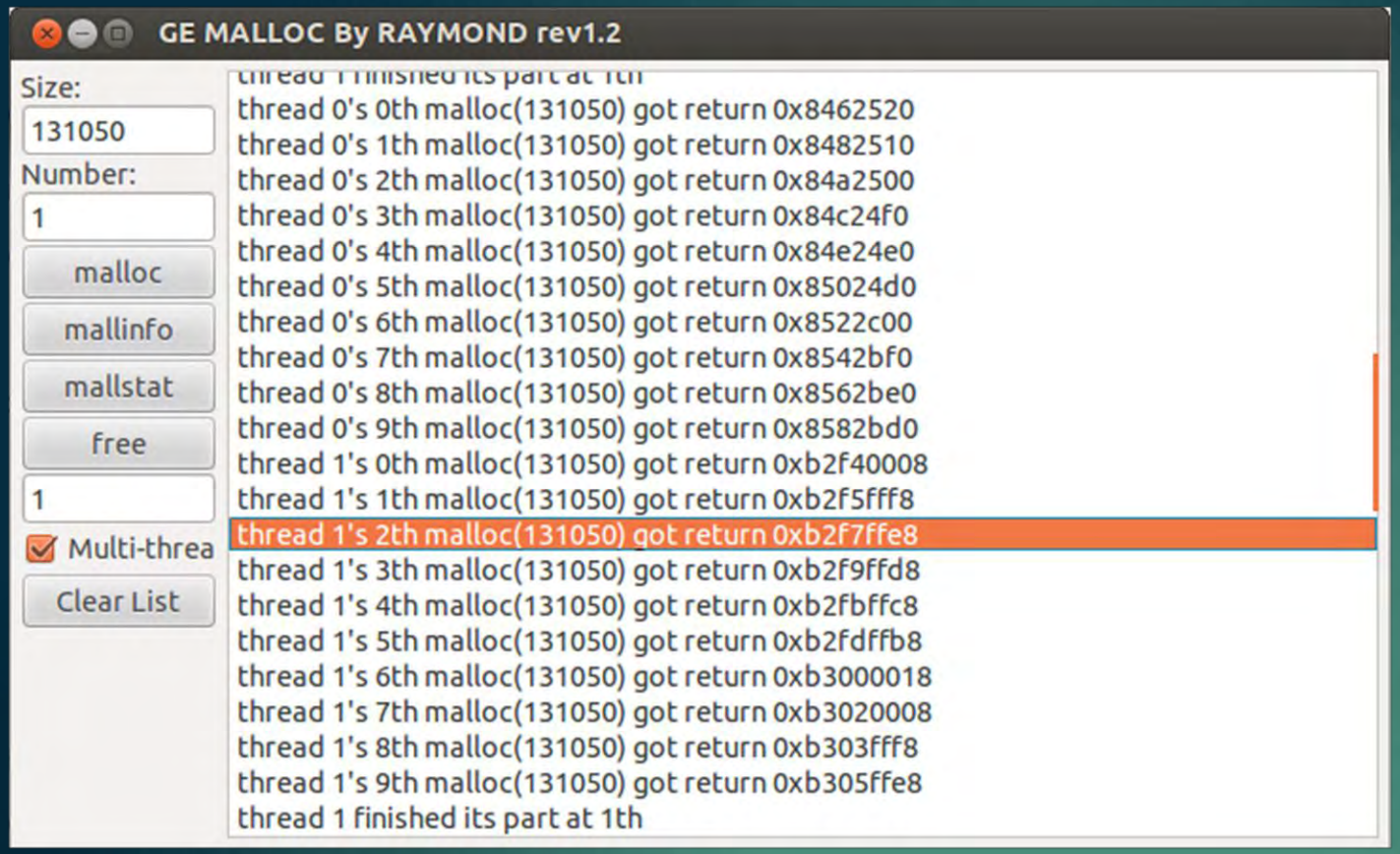
设计约束带来的好处，可以很简单地计算出（看出）一个来自堆的地址属于哪个子堆和arena

```
/* size field is or'ed with NON_MAIN_ARENA if the chunk was obtained
   from a non-main arena.  This is only set immediately before handing
   the chunk to the user, if necessary.  */
#define NON_MAIN_ARENA 0x4

/* check for chunk from non-main arena */
#define chunk_non_main_arena(p) ((p)->size & NON_MAIN_ARENA)
```

```
#define HEAP_MIN_SIZE (32*1024)
#ifndef HEAP_MAX_SIZE
#define HEAP_MAX_SIZE (1024*1024) /* must be a power of two */
#endif

/* HEAP_MIN_SIZE and HEAP_MAX_SIZE limit the size of mmap()ed heaps
   that are dynamically created for multi-threaded programs.  The
   maximum size must be a power of two, for fast determination of
   which heap belongs to a chunk.  It should be much larger than the
   mmap threshold, so that requests with a size just below that
   threshold can be fulfilled without creating too many heaps.  */
```



WinDBG: “我有强大的dt命令”

GDB： “我的p加类型强转就可以了”

```
(gdb) p *(heap_info*)0xb2f00000
$32 = {ar_ptr = 0xb4f00010, prev = 0xb3100000, size = 1048576,
  mprotect_size = 1048576, pad = 0xb2f00010 ""}

(gdb) p *((heap_info*)0xb2f00000)->ar_ptr
$38 = {mutex = 0, flags = 2, fastbinsY = {0x0, 0x0, 0x0, 0xb4fe7170,
    0xb4f042a8, 0xb4fe71c0, 0x0, 0x0, 0x0, 0x0}, top = 0xb2e3fff0,
  last_remainder = 0xb4f02878, bins = {0xb4fe6df8, 0xb4f01ef8, 0xb4f02a58,
    0xb4f02f10, 0xb4f02768, 0xb4f043c8, 0xb4f05330, 0xb4f05118, 0xb4f00060,
```

获取arena和按需创建

public_mALLOc(size_t bytes)

```
/*------------------------ Public wrappers. --------------------------------*/
Void_t*
public_mALLOc(size_t bytes)
{
  mstate ar_ptr;
  Void_t *victim;

  __malloc_ptr_t (*hook) __MALLOC_P ((size_t, __const __malloc_ptr_t)) =
    __malloc_hook;
  if (hook != NULL)
    return (*hook)(bytes, RETURN_ADDRESS (0));

  arena_get(ar_ptr, bytes);
  if(!ar_ptr)
    return 0;

  victim = _int_malloc(ar_ptr, bytes);
```

内部分配都是围绕arena进行的

来自Gloger网站的代码

```
/* arena_get() acquires an arena and locks the corresponding mutex.
   First, try the one last locked successfully by this thread.  (This
   is the common case and handled with a macro for speed.)  Then, loop
   once over the circularly linked list of arenas.  If no arena is
   readily available, create a new one.  In this latter case, `size'
   is just a hint as to how much memory will be required immediately
   in the new arena. */

#define arena_get(ptr, size) do { \
  Void_t *vptr = NULL; \
  ptr = (mstate)tsd_getspecific(arena_key, vptr); \
  if(ptr && !mutex_trylock(&ptr->mutex)) { \
    THREAD_STAT(++(ptr->stat_lock_direct)); \
  } else \
    ptr = arena_get2(ptr, (size)); \
} while(0)
```

```
2906    /*------------------------ Public wrappers. --------------------------------*/
2907
2908    void*
2909    public_mALLOc(size_t bytes)
2910    {
2911      mstate ar_ptr;
2912      void *victim;
2913
2914      __malloc_ptr_t (*hook) (size_t, __const __malloc_ptr_t)
(gdb)
2915        = force_reg (__malloc_hook);
2916      if (__builtin_expect (hook != NULL, 0))
2917        return (*hook)(bytes, RETURN_ADDRESS (0));
2918
2919      arena_lookup(ar_ptr);
2920
2921      arena_lock(ar_ptr, bytes);
2922      if(!ar_ptr)
2923        return 0;
2924      victim = _int_malloc(ar_ptr, bytes);
```

Glibc中的代码

```
#define arena_lookup(ptr) do { \
  void *vptr = NULL; \
  ptr = (mstate)tsd_getspecific(arena_key, vptr); \
} while(0)
```

做了一些简化，总是从线程局部存储中取，如果取到为空，稍后的lock函数会动态创建

```
(gdb) disassemble __GI___libc_malloc
Dump of assembler code for function __GI___libc_malloc:
   0xb7154ef0 <+0>:   sub    $0x3c,%esp
   0xb7154ef3 <+3>:   mov    %ebx,0x2c(%esp)
   0xb7154ef7 <+7>:   call   0xb72096c3 <__i686.get_pc_thunk.bx>
   0xb7154efc <+12>: add    $0x12f0f8,%ebx
   0xb7154f02 <+18>: mov    %ebp,0x38(%esp)
   0xb7154f06 <+22>: mov    0x40(%esp),%ebp
   0xb7154f0a <+26>: mov    %esi,0x30(%esp)
   0xb7154f0e <+30>: mov    %edi,0x34(%esp)
   0xb7154f12 <+34>: mov    -0xbc(%ebx),%eax
   0xb7154f18 <+40>: mov    (%eax),%eax
   0xb7154f1a <+42>: test   %eax,%eax
   0xb7154f1c <+44>: jne    0xb71550ba <__GI___libc_malloc+458>
=> 0xb7154f22 <+50>:      mov    -0x188(%ebx),%edx
   0xb7154f28 <+56>: mov    %gs:0x0,%ecx
   0xb7154f2f <+63>:  mov    (%ecx,%edx,1),%edi
   0xb7154f32 <+66>: test   %edi,%edi
   0xb7154f34 <+68>: je     0xb7155038 <__GI___libc_malloc+328>
```

```
(gdb) set disassembly-flavor intel
(gdb) disassemble __GI___libc_malloc
Dump of assembler code for function __GI___libc_malloc:
   0xb7154ef0 <+0>:   sub    esp,0x3c
   0xb7154ef3 <+3>:   mov    DWORD PTR [esp+0x2c],ebx
   0xb7154ef7 <+7>:   call   0xb72096c3 <__i686.get_pc_thunk.bx>
   0xb7154efc <+12>: add    ebx,0x12f0f8
   0xb7154f02 <+18>: mov    DWORD PTR [esp+0x38],ebp
   0xb7154f06 <+22>: mov    ebp,DWORD PTR [esp+0x40]
   0xb7154f0a <+26>: mov    DWORD PTR [esp+0x30],esi
   0xb7154f0e <+30>: mov    DWORD PTR [esp+0x34],edi
   0xb7154f12 <+34>: mov    eax,DWORD PTR [ebx-0xbc]
   0xb7154f18 <+40>: mov    eax,DWORD PTR [eax]
   0xb7154f1a <+42>: test   eax,eax
   0xb7154f1c <+44>: jne    0xb71550ba <__GI___libc_malloc+458>
=> 0xb7154f22 <+50>:      mov    edx,DWORD PTR [ebx-0x188]
   0xb7154f28 <+56>: mov    ecx,DWORD PTR gs:0x0
   0xb7154f2f <+63>:  mov    edi,DWORD PTR [ecx+edx*1]
   0xb7154f32 <+66>: test   edi,edi
   0xb7154f34 <+68>: je     0xb7155038 <__GI___libc_malloc+328>
```

```
#ifdef PER_THREAD
# define arena_lock(ptr, size) do { \
  if(ptr) \
    (void)mutex_lock(&ptr->mutex); \
  else \
    ptr = arena_get2(ptr, (size)); \
} while(0)
#else
# define arena_lock(ptr, size) do { \
  if(ptr && !mutex_trylock(&ptr->mutex)) { \
    THREAD_STAT(++(ptr->stat_lock_direct)); \
  } else \
    ptr = arena_get2(ptr, (size)); \
} while(0)
#endif
```

arena_get2内部如果找不到可用的arena则_int_new_arena (size);