
精彩會講
在调试器里看glibc堆
张银奎
2018/8/30更新
2017/7/12

1


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

2

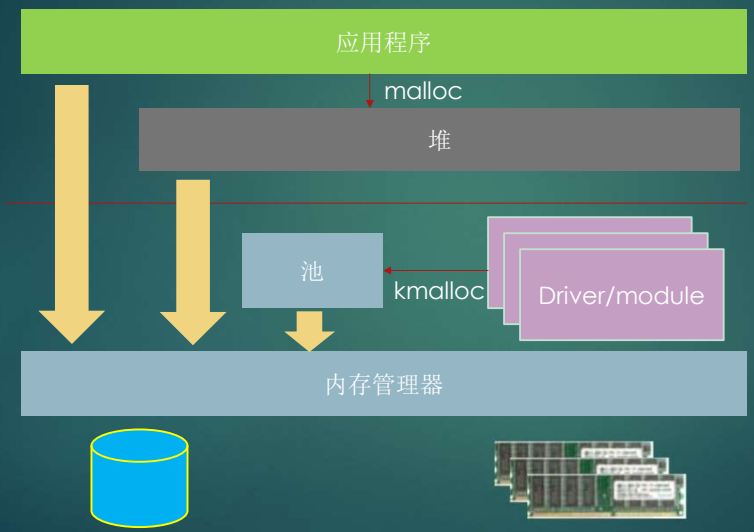
内存问题是软件世界的住房问题

3

《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。

4

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

5

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

6

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/

7

版本1的readme文件

ptmalloc - a multi-thread malloc implementation
==========

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.

As part of the GNU C library, the source files are available under the
GNU Library General Public License (see the comments in the files).
But as part of this stand-alone package, the code is available under
the (probably less restrictive) conditions described in the file
`COPYRIGHT'. In any case, there is no warranty whatsoever for this
package.

8

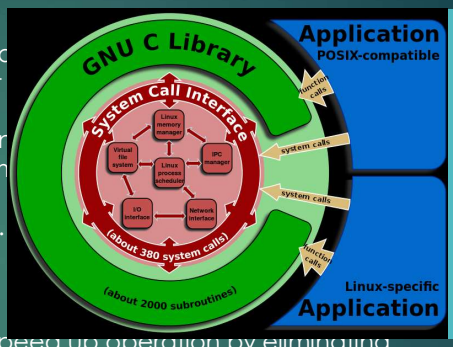
Version 2.3

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

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

* localedef can now transliterate characters ir
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.

9

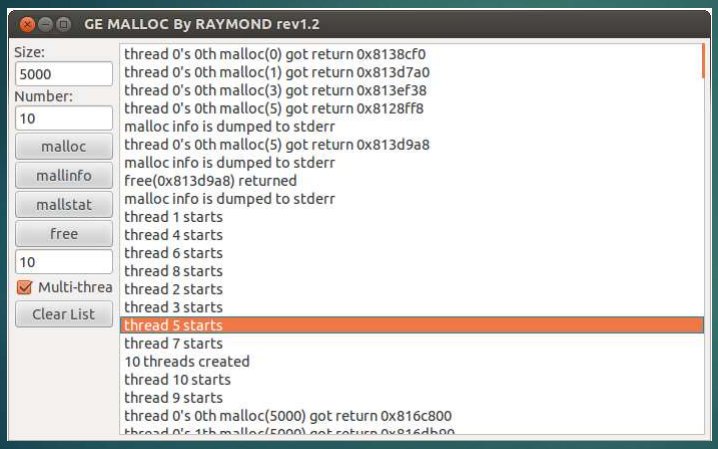
源文件(ptmalloc2)

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--
Iran2	h	837	1996/12/06 00:42	-a--

10

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

11

概览 Arena 批发 分配过程 释放 Valgrind Asan

12

Arena

13

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

14

```
(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}
```

15

```
(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, 0x9ba4b
    0xb72844e0, 0xb72844e0, 0x9b9ce78, 0x9b9ce78, 0x9b743f8, 0x9b743f8, 0xb72844f8, 0xb72844f8, 0xb7284500, 0xb72845
    0xb7284508, 0xb7284508, 0xb7284510, 0xb7284510, 0x9ba50b0, 0x9ba50b0, 0xb7284520, 0xb7284520, 0xb7284528, 0xb7
    0xb7284530, 0xb7284530, 0xb7284538, 0xb7284538, 0xb7284540, 0xb7284540, 0xb7284548, 0xb7284548, 0xb7284550, 0xb
    0xb72845
    0xb72845
    0x9ba4dc
    0x9bb567
    0xb72845f8, 0xb7284600, 0xb7284600, 0xb7284608, 0xb7284608, 0xb7284610, 0xb7284610, 0xb7284618, 0xb7284618, 0xb7
    0xb7284620, 0xb7284628, 0xb7284628, 0xb7284630, 0xb7284630, 0xb7284638, 0xb7284638, 0xb7284640, 0xb7284640, 0xb
    0xb7284648, 0xb7284650, 0xb7284650, 0xb7284658, 0xb7284658, 0xb7284660, 0xb7284660, 0xb7284668, 0xb7284668, 0xb
    0xb7284670, 0xb7284678, 0xb7284678, 0xb7284680, 0xb7284680, 0xb7284688, 0xb7284688, 0xb7284690, 0xb7284690, 0xb
    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, 0xb
    0xb7284738, 0xb7284740, 0xb7284740, 0xb7284748, 0xb7284748, 0xb7284750, 0xb7284750, 0xb7284758, 0xb7284758, 0xb
    0xb7284760, 0xb7284768, 0xb7284768, 0xb7284770, 0xb7284770, 0xb7284778, 0xb7284778, 0xb7284780, 0xb7284780, 0xb
    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.

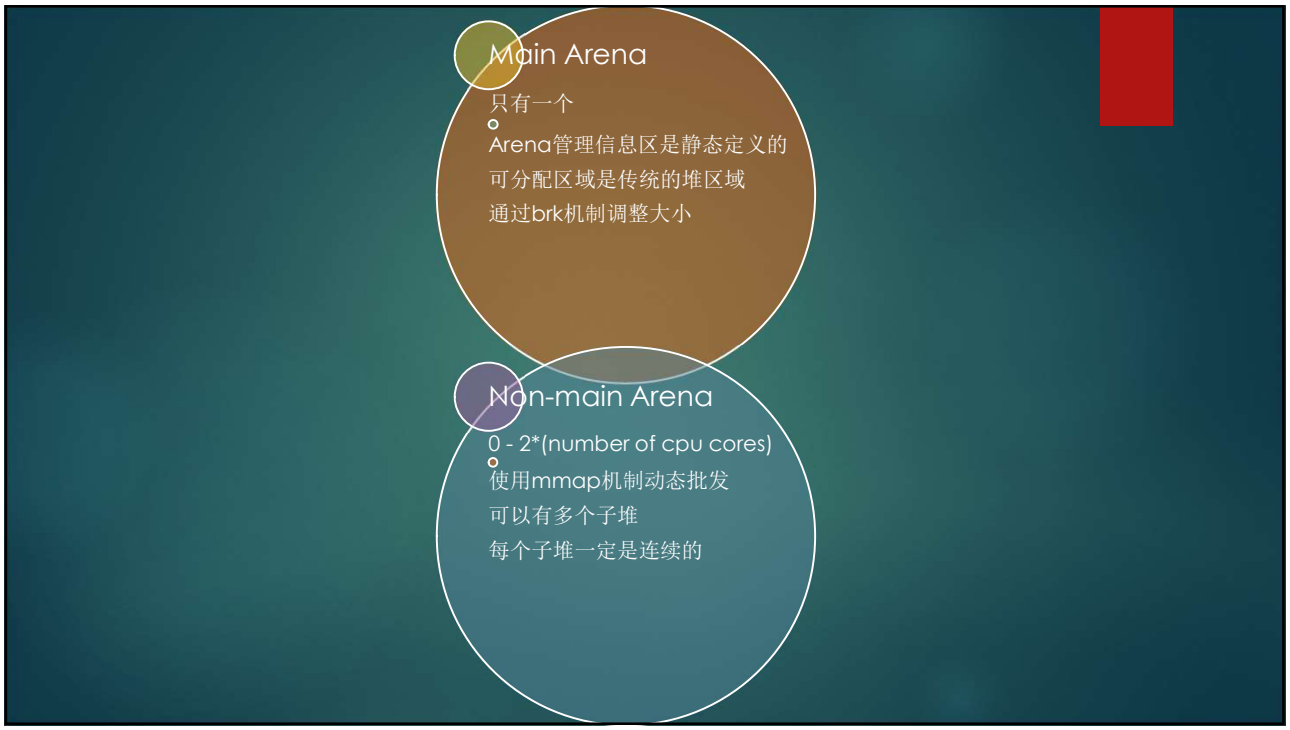
16

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

17



18

按需初始化

```
#0  ptmalloc_init () at arena.c:370
#1  0xb79c31fb in malloc_hook_ini (sz=352, caller=0xb79aee08) at hooks.c:32
#2  0xb79c30c7 in __GI___libc_malloc (bytes=352) at malloc.c:2917
#3  0xb79aee08 in __fopen_internal (filename=0xb710ef86 "/proc/filesystems", mode=0xb710ee7b "r", is
#4  0xb79b163b in _IO_fopen64 (filename=0xb710ef86 "/proc/filesystems", mode=0xb710ee7b "r") at iof
#5  0xb70fd2d3 in ?? () from /lib/i386-linux-gnu/libselinux.so.1
#6  0xb7fece5b in call_init (env=0xbffff41c, argv=0xbffff414, argc=1, l=<optimized out>) at dl-init.c:85
#7  call_init (l=<optimized out>, argc=1, argv=0xbffff414, env=0xbffff41c) at dl-init.c:35
#8  0xb7fecf44 in _dl_init (main_map=<optimized out>, argc=1, argv=0xbffff414, env=0xbffff41c) at dl-in
#9  0xb7fdf20f in _dl_start_user () from /lib/ld-linux.so.2
```


__malloc_initialized

- 全局变量
- 初始值为-1
- 开始初始化（ ptmalloc_init 函数开头）设置为0
- 初始化结束设置为1

```
static void*
malloc_hook_ini(size_t sz, const __malloc_ptr_t caller)
{
  __malloc_hook = NULL;
  ptmalloc_init();
  return public_mALLOc(sz);
}

__malloc_ptr_t weak_variable (*__malloc_hook)
    (size_t __size, const __malloc_ptr_t) = malloc_hook_ini;
```

21

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**

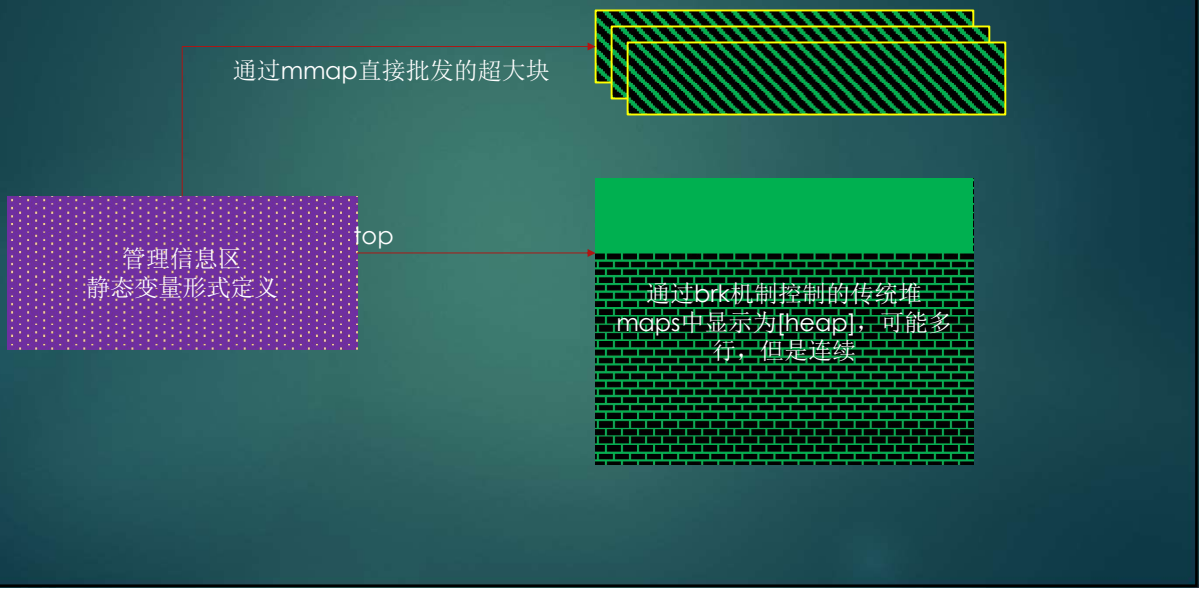
22

_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;
```

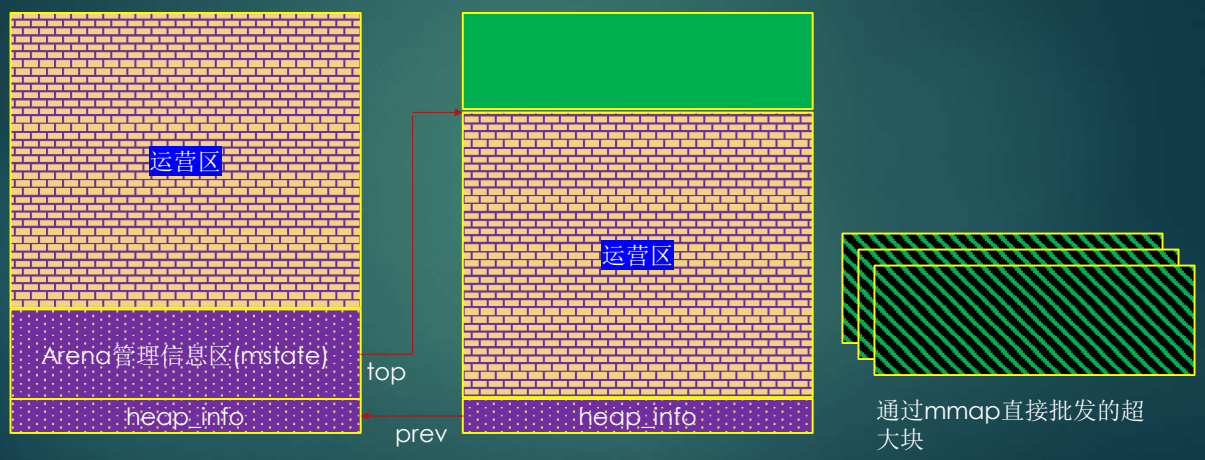
23

Main arena布局



24

Non-main arena布局



25

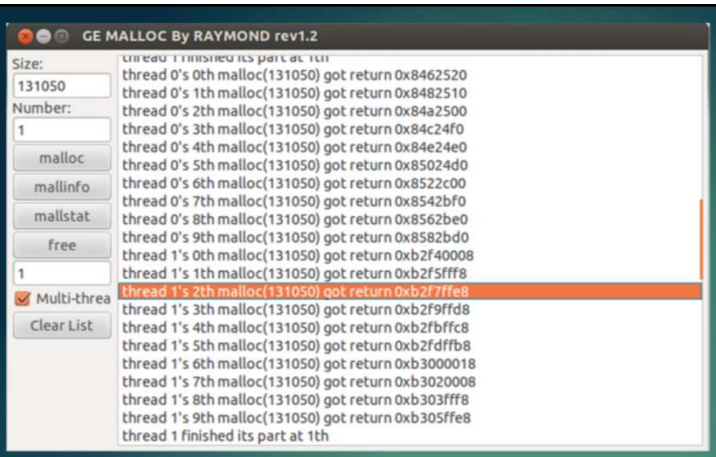
设计约束带来的好处，可以很简单地计算出（看出）一个来自堆的地址属于哪个子堆和arena

```
/* 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)


/* 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)
```

26

```
#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.  */
```

27



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,
```

28

获取arena和按需创建

29

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

30

```
/* 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)
```

31

```
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

32

```
#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

33

```
(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>
```

34

```
(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>
```

35

```
#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);

36

```
0xb7154f2f   2919        arena_lookup(ar_ptr);
   0xb7154f22 <__GI___libc_malloc+50>:   8b 93 78 fe ff ff   mov    edx,DWORD PTR [ebx-0x188]
   0xb7154f28 <__GI___libc_malloc+56>:   65 8b 0d 00 00 00 00 mov    ecx,DWORD PTR gs:0x0
=> 0xb7154f2f <__GI___libc_malloc+63>:   8b 3c 11       mov    edi,DWORD PTR [ecx+edx*1]
(gdb) i r
eax            0x0     0
ecx            0xb67f5840      -1233168320
edx            0xffffffd8  -40
ebx            0xb7283ff4      -1222098956
esp            0xbffc2f70      0xbffc2f70
ebp            0x24  0x24

(gdb) x /x $ecx+$edx
0xb67f5818: 0xb7284440

(gdb) p &main_arena
$16 = (struct malloc_state *) 0xb7284440
```

这次分配使用的刚好是主场地

37

```
#0 new_heap (size=3168, top_pad=131072) at arena.c:523
#1  0xb79c227f in _int_new_arena (size=2040) at arena.c:712
#2  arena_get2 (avoid_arena=0x0, size=2040, a_tsd=<optimized out>) at arena.c:856
#3 arena_get2 (a_tsd=<optimized out>, size=2040, avoid_arena=0x0) at arena.c:823
#4  0xb79c422d in __libc_calloc (n=1, elem_size=2040) at malloc.c:3291
#5  0xb7397753 in ?? () from /lib/i386-linux-gnu/libglib-2.0.so.0
#6  0xb7397e6b in g_malloc0 () from /lib/i386-linux-gnu/libglib-2.0.so.0
#7  0xb7360215 in ?? () from /lib/i386-linux-gnu/libglib-2.0.so.0
#8  0xb73ac445 in g_slice_alloc () from /lib/i386-linux-gnu/libglib-2.0.so.0
#9  0xb73ac796 in g_slice_alloc0 () from /lib/i386-linux-gnu/libglib-2.0.so.0
#10 0xb73a004b in g_queue_new () from /lib/i386-linux-gnu/libglib-2.0.so.0
#11 0xb73912d5 in g_main_context_push_thread_default () from /lib/i386-linux-gnu/libglib-2.0.so.0
#12 0xb6e5e11c in ?? () from /usr/lib/i386-linux-gnu/gio/modules/libdconfsettings.so
#13 0xb73b5673 in ?? () from /lib/i386-linux-gnu/libglib-2.0.so.0
#14 0xb7afcd4c in start_thread (arg=0xb6c2fb40) at pthread_create.c:308
#15 0xb7a3b87e in clone () at ../sysdeps/unix/sysv/linux/i386/clone.S:130
(gdb) info threads
  Id   Target Id         Frame
* 2    Thread 0xb6c2fb40 (LWP 13642) "dconf worker" new_heap (size=3168, top_pad=131072) at arena.c:5
  1    Thread 0xb7063840 (LWP 13639) "gemalloc" __GI__dl_debug_state () at dl-debug.c:77
```

38

```
(gdb) frame 2
#2  arena_get2 (avoid_arena=0x0, size=2040, a_tsd=<optimized out>) at arena.c:856
856 #else
(gdb) frame 4
#4  0xb79c422d in __libc_calloc (n=1, elem_size=2040) at malloc.c:3291
3291            av = arena_get2(av->next ? av : 0, sz);
```

```
if (mem == 0) {
  /* Maybe the failure is due to running out of mmapped areas. */
  if(av != &main_arena) {
    (void)mutex_lock(&main_arena.mutex);
    mem = _int_malloc(&main_arena, sz);
    (void)mutex_unlock(&main_arena.mutex);
  } else {
    /* ... or sbrk() has failed and there is still a chance to mmap() */
    (void)mutex_lock(&main_arena.mutex);
    av = arena_get2(av->next ? av : 0, sz);
    (void)mutex_unlock(&main_arena.mutex);
    if(av) {
     mem = _int_malloc(av, sz);
     (void)mutex_unlock(&av->mutex);
    }
```

Calloc的错误处理代码中触发创建

39

不忘初心

/* Malloc implementation for **multiple threads without lock contention**.
Copyright (C) 2001-2006, 2007, 2008, 2009, 2011 Free Software Foundation, Inc.
This file is part of the GNU C Library.
Contributed by **Wolfram Gloger** <wg@malloc.de>, 2001.

40

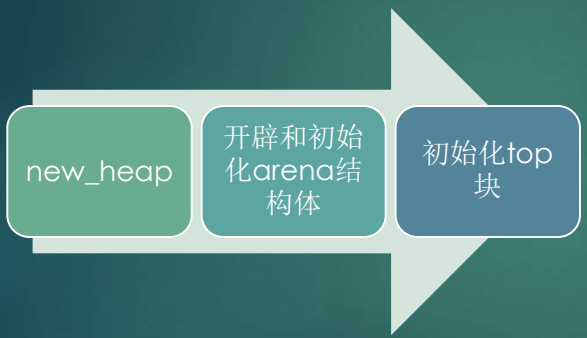
* Why use this malloc?

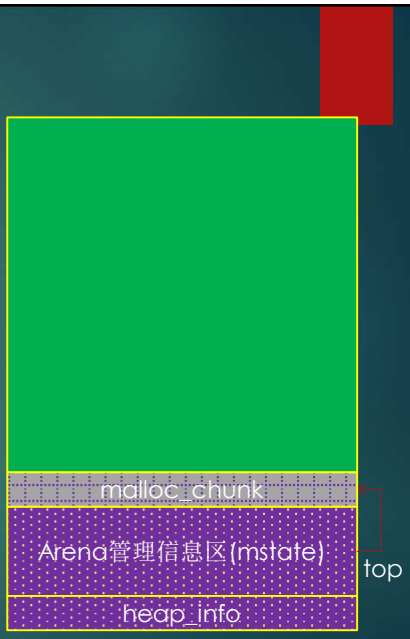
This is not the fastest, most space-conserving, most portable, or most tunable malloc ever written. However it is among the fastest while also being among the most space-conserving, portable and tunable. Consistent balance across these factors results in a good general-purpose allocator for malloc-intensive programs.

- Malloc.c

41

创建过程





42

源代码

```
mstate
_int_new_arena(size_t size)
{
  mstate a;
  heap_info *h;
  char *ptr;
  unsigned long misalign;

  h = new_heap(size + (sizeof(*h) + sizeof(*a) + MALLOC_ALIGNMENT),
               mp_.top_pad);
  if(!h) {
    /* Maybe size is too large to fit in a single heap.  So, just try
       to create a minimally-sized arena and let _int_malloc() attempt
       to deal with the large request via mmap_chunk().  */
    h = new_heap(sizeof(*h) + sizeof(*a) + MALLOC_ALIGNMENT, mp_.top_pad);
    if(!h)
      return 0;
  }
```

43

```
  a = h->ar_ptr = (mstate)(h+1);
  malloc_init_state(a);
  /*a->next = NULL;*/
  a->system_mem = a->max_system_mem = h->size;
  arena_mem += h->size;
#ifdef NO_THREADS
  if((unsigned long)(mp_.mmapped_mem + arena_mem + main_arena.system_mem) >
     mp_.max_total_mem)
    mp_.max_total_mem = mp_.mmapped_mem + arena_mem + main_arena.system_mem;
#endif

  /* Set up the top chunk, with proper alignment. */
  ptr = (char *)(a + 1);
  misalign = (unsigned long)chunk2mem(ptr) & MALLOC_ALIGN_MASK;
  if (misalign > 0)
    ptr += MALLOC_ALIGNMENT - misalign;
  top(a) = (mchunkptr)ptr;
  set_head(top(a), (((char*)h + h->size) - ptr) | PREV_INUSE);

  return a;
}
```

44

Arena小结

- Main_Arena是静态定义的
- 根据需要动态创建更多的Arena
  - 直到找到一个可用的
  - 最多每个线程一个，但可以通过配置参数设置上限
- 单CPU系统，常常有三个
- 增加CPU个数后（并发度增大），多线程的gemalloc小程序创建10个工作线程后，有6个Arena

45

他山之石，可以为错

Windows	Linux/Glibc
堆（Heap）	Arena
段（Segment）	Heap（子堆）
Entry	Chunk

46

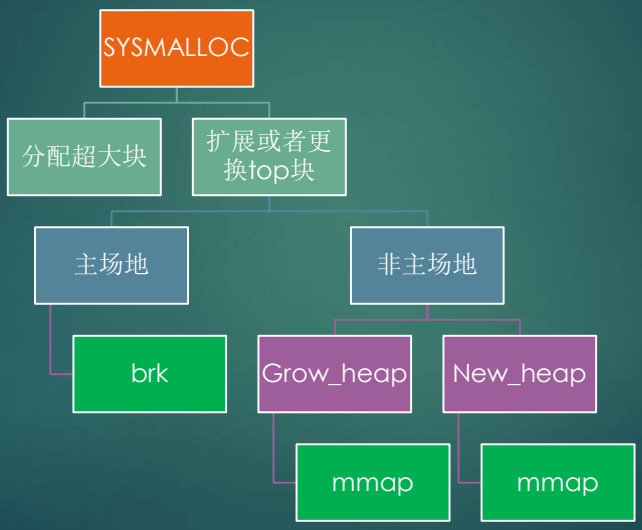
47

/*

sysmalloc handles malloc cases requiring more memory from the system.
On entry, it is assumed that av->top does not have enough
space to service request for nb bytes, thus requiring that av->top
be extended or replaced.
*/

48

三大块代码



49

主arena被触发扩容

```
(gdb) bt
#0  __brk (addr=0x0) at ../sysdeps/unix/sysv/linux/i386/brk.c:52
#1  0xb7a33571 in __GI___sbrk (increment=135168) at sbrk.c:44
#2  0xb79c4f4f in __GI___default_morecore (increment=135168) at morecore.c:49
#3  0xb79c0e82 in sYSMALLOc (av=0xffffefd0, nb=360) at malloc.c:2535
#4  _int_malloc (av=0xffffefd0, bytes=352) at malloc.c:3932
#5  0xb79c2f5c in __GI___libc_malloc (bytes=352) at malloc.c:2924
#6  0xb79c30c7 in __GI___libc_malloc (bytes=352) at malloc.c:2917
#7  0xb79aee08 in __fopen_internal (filename=0xb710ef86 "/proc/filesystems", mode=0xb710ee7b "r", is3
#8  0xb79b163b in _IO_fopen64 (filename=0xb710ef86 "/proc/filesystems", mode=0xb710ee7b "r") at iofo
#9  0xb70fd2d3 in ?? () from /lib/i386-linux-gnu/libselinux.so.1
#10 0xb7fece5b in call_init (env=0xbffff41c, argv=0xbffff414, argc=1, l=<optimized out>) at dl-init.c:85
#11 call_init (l=<optimized out>, argc=1, argv=0xbffff414, env=0xbffff41c) at dl-init.c:35
#12 0xb7fecf44 in _dl_init (main_map=<optimized out>, argc=1, argv=0xbffff414, env=0xbffff41c) at dl-in
#13 0xb7fdf20f in _dl_start_user () from /lib/ld-linux.so.2
```

50

```
/*
  MORECORE is the name of the routine to call to obtain more memory
  from the system.  See below for general guidance on writing
  alternative MORECORE functions, as well as a version for WIN32 and a
  sample version for pre-OSX macos.
*/

#ifndef MORECORE
#define MORECORE sbrk
#endif
```

51

brk

```
#include <unistd.h>

    int brk(void *addr);

    void *sbrk(intptr_t increment);
```

- change data segment size

pubs.opengroup.org/onlinepubs/7908799/xsh/brk.html

The Single UNIX ® Specification, Version 2
Copyright © 1997 The Open Group

NAME

brk, sbrk - change space allocation (**LEGACY**)

SYNOPSIS

```
#include <unistd.h>

int brk(void *addr);
void *sbrk(intptr_t incr);
```

DESCRIPTION

The *brk()* and *sbrk()* functions are used to change the amount of space allocated for the calling process. The change is made by resetting the process' break value and allocating the appropriate amount of space. The amount of allocated space increases as the break value increases. The newly-allocated space is set to 0. However, if the application first decrements and then increments the break value, the contents of the reallocated space are unspecified.

The *brk()* function sets the break value to *addr* and changes the allocated space accordingly.

52

更替top块

Hardware watchpoint 9: main_arena->top

Old value = (mchunkptr) 0xb7af2470

New value = (mchunkptr) **0x804c000**
Hardware watchpoint 10: main_arena->top

Old value = (mchunkptr) 0xb7af2470
New value = (mchunkptr) 0x804c000
sYSMALLOc (av=0xffffefd0, nb=360) at malloc.c:2697
2697 set_head(av->top, (snd_brk - aligned_brk + correction) | PREV_INUSE);

53

ge@gewubox:~$ cat /proc/2748/maps | grep "\[heap\]"
0804c000-0806d000 rw-p 00000000 00:00 0 [heap]

54

```
#0  __GI___sbrk (increment=143360) at sbrk.c:55
#1  0xb79c4f4f in __GI___default_morecore (increment=143360) at morecore.c:49
#2  0xb79c0e82 in sYSMALLOc (av=0xffffefd0, nb=32800) at malloc.c:2535
#3  _int_malloc (av=0xffffefd0, bytes=32796) at malloc.c:3932
#4  0xb79c2f5c in __GI___libc_malloc (bytes=32796) at malloc.c:2924
#5  0xb79fffc8 in __alloc_dir (fd=8, close_fd=<optimized out>, flags=<optimized out>, statp=0x0) at ../sysde
#6  0xb7a000f7 in __opendirat (dfd=<optimized out>, name=<optimized out>) at ../sysdeps/unix/opendir.c
#7  0xb7a0013d in __opendir (name=0xb75eed88 "/usr/lib/i386-linux-gnu/gio/modules") at ../sysdeps/unix/
#8  0xb737d0dc in g_dir_open () from /lib/i386-linux-gnu/libglib-2.0.so.0
.....
#18 0xb6e1bc00 in os_utils_is_blacklisted () from /usr/lib/liboverlay-scrollbar-0.2.so.0
#19 0xb7c962cc in ?? () from /usr/lib/i386-linux-gnu/libgtk-x11-2.0.so.0
#20 0xb739db12 in g_option_context_parse () from /lib/i386-linux-gnu/libglib-2.0.so.0
#21 0xb7c96988 in gtk_parse_args () from /usr/lib/i386-linux-gnu/libgtk-x11-2.0.so.0
#22 0xb7c96a13 in gtk_init_check () from /usr/lib/i386-linux-gnu/libgtk-x11-2.0.so.0
#23 0xb7c96a53 in gtk_init () from /usr/lib/i386-linux-gnu/libgtk-x11-2.0.so.0
#24 0x08049332 in main (argc=1, argv=0xbffff414) at gemalloc.c:145
```

55

```
 0xb7af2780, 0xb7af2780, 0xb7af2788, 0xb7af2788...}, binmap = {20, 33554432, 16, 0}, next = 0xb7af2440, r
0x0, system_mem = 135168, max_system_mem = 135168}
(gdb) watch main_arena->system_mem
Hardware watchpoint 15: main_arena->system_mem
(gdb) c
Continuing.
Hardware watchpoint 15: main_arena->system_mem

Old value = 135168
New value = 278528
sYSMALLOc (av=0xffffefd0, nb=32800) at malloc.c:2591
2591          if (brk == old_end && snd_brk == (char*)(MORECORE_FAILURE))
(gdb) p 135168+143360
$30 = 278528


  #0  sYSMALLOc (av=0xffffefd0, nb=32800) at malloc.c:2591
  #1  _int_malloc (av=0xffffefd0, bytes=32796) at malloc.c:3932
  #2  0xb79c2f5c in __GI___libc_malloc (bytes=32796) at malloc.c:2924
```

56

```
ge@gewubox:~/eglibc-2.15$ cat /proc/13795/maps | grep "heap"
08048000-0804a000 r-xp 00000000 08:01 1835228    /home/ge/work/geheap/gemalloc
0804a000-0804b000 r--p 00001000 08:01 1835228    /home/ge/work/geheap/gemalloc
0804b000-0804c000 rw-p 00002000 08:01 1835228    /home/ge/work/geheap/gemalloc
0804c000-0806d000 rw-p 00000000 00:00 0          [heap]
0806d000-08090000 rw-p 00000000 00:00 0          [heap]
```

所以，maps文件看到的heap块数与arena个数不是一会事，可能不一致

57

辅Arena用mmap从内核批发内存

```
#include <sys/mman.h>

void *mmap(void *addr, size_t length, int prot, int flags,
           int fd, off_t offset);
int munmap(void *addr, size_t length);
```

- 历史上，比brk机制要出现的晚

mmap and associated systems calls were designed as part of the Berkeley Software Distribution (BSD) version of Unix. Their API was already described in the 4.2BSD System Manual, even though it was neither implemented in that release, nor in 4.3BSD.[1] Sun Microsystems had implemented this very API, though, in their SunOS operating system. The BSD developers at U.C. Berkeley requested Sun to donate its implementation, but these talks never led to any transfer of code; 4.3BSD-Reno was shipped instead with an implementation based on the virtual memory system of Mach.[2] -- wikipedia

58

创建第一个子堆

#0 mmap () at ../sysdeps/unix/sysv/linux/i386/mmap.S:98
#1 0xb79be6c7 in new_heap (size=<optimized out>, top_pad=<optimized out>) at arena.c:554
#2 0xb79c227f in _int_new_arena (size=2040) at arena.c:712
#3 arena_get2 (avoid_arena=0x0, size=2040, a_tsd=<optimized out>) at arena.c:856
#4 arena_get2 (a_tsd=<optimized out>, size=2040, avoid_arena=0x0) at arena.c:823
#5 0xb79c422d in __libc_calloc (n=1, elem_size=2040) at malloc.c:3291

59

geheap.c arena.c

```
/* Create a new heap.  size is automatically rounded up to a multiple
   of the page size. */

static heap_info *
internal_function
new_heap(size_t size, size_t top_pad)
{
  size_t page_mask = GLRO(dl_pagesize) - 1;
  char *p1, *p2;
  unsigned long ul;
  heap_info *h;

  if(size+top_pad < HEAP_MIN_SIZE)
    size = HEAP_MIN_SIZE;
  else if(size+top_pad <= HEAP_MAX_SIZE)
    size += top_pad;
  else if(size > HEAP_MAX_SIZE)
    return 0;
  else
    size = HEAP_MAX_SIZE;
  size = (size + page_mask) & ~page_mask;

  /* A memory region aligned to a multiple of HEAP_MAX_SIZE is needed.
     No swap space needs to be reserved for the following large
     mapping (on Linux, this is the case for all non-writable mappings
     anyway). */
  p2 = MAP_FAILED;
  if(aligned_heap_area) {
    p2 = (char *)MMAP(aligned_heap_area, HEAP_MAX_SIZE, PROT_NONE,
                      MAP_PRIVATE|MAP_NORESERVE);
    aligned_heap_area = NULL;
    if (p2 != MAP_FAILED && ((unsigned long)p2 & (HEAP_MAX_SIZE-1))) {
      munmap(p2, HEAP_MAX_SIZE);
      p2 = MAP_FAILED;
    }
  }
```

60

创建第二个

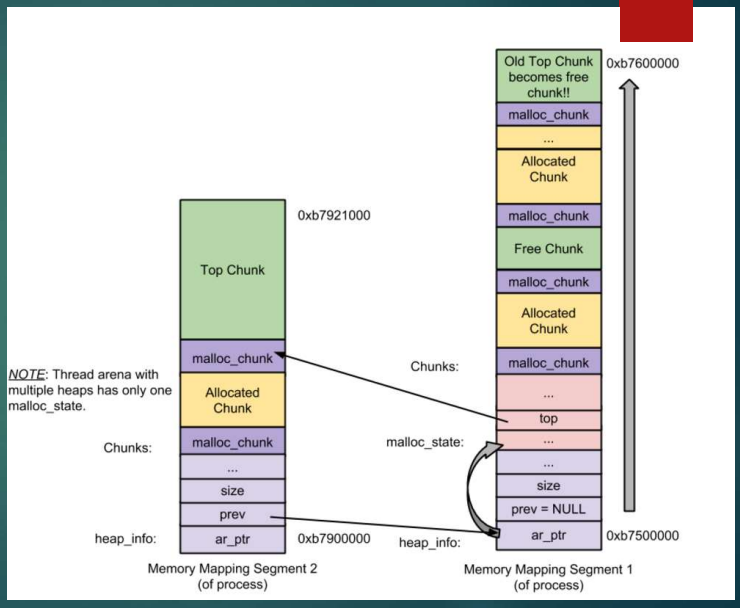
New_heap()函数返回后，观察

```
(gdb) p *heap
$27 = {ar_ptr = 0xb4f00010, prev = 0xb4f00000, size = 266240,
  mprotect_size = 266240, pad = 0xb3100010 ""}
(gdb) bt
#0  sYSMALLOc (av=0xb4f00010, nb=131064) at malloc.c:2476
#1  _int_malloc (av=0xb4f00010, bytes=131060) at malloc.c:3932
#2  0xb7818f5c in __GI___libc_malloc (bytes=131060) at malloc.c:2924
#3  0x080492e4 in do_malloc (nsize=131060, no=10, threadno=1) at gemalloc.c:49
#4  0x0804947c in thread_func (arg=0x1) at gemalloc.c:111
#5  0xb7952d4c in start_thread (arg=0xb5810b40) at pthread_create.c:308
#6  0xb789187e in clone () at ../sysdeps/unix/sysv/linux/i386/clone.S:130
```

创建新的heap，此后，一个场地具有多个heap

一个arena的两个子堆

- 老的top块变成空闲块
- 新分配出的运营区划出一块给当前的请求，余下的成为新的top块
- 两个子堆共用一个mstate结构体



```
(gdb) p *((heap_info*)0xb2f00000)
$40 = {ar_ptr = 0xb4f00010, prev = 0xb3100000, size = 1048576,
  mprotect_size = 1048576, pad = 0xb2f00010 ""}
(gdb) p *((heap_info*)0xb2f00000)->prev
$41 = {ar_ptr = 0xb4f00010, prev = 0xb4f00000, size = 1048576,
  mprotect_size = 1048576, pad = 0xb3100010 ""}
(gdb) p *((heap_info*)0xb2f00000)->prev->prev
$42 = {ar_ptr = 0xb4f00010, prev = 0x0, size = 978944, mprotect_size = 978944,
  pad = 0xb4f00010 ""}
```

后两个子堆的大小都为1MB，第一个略小，arena结构体有1104字节

63

分配超大块

```
/*
  If have mmap, and the request size meets the mmap threshold, and
  the system supports mmap, and there are few enough currently
  allocated mmapped regions, try to directly map this request
  rather than expanding top.
*/

if ((unsigned long)(nb) >= (unsigned long)(mp_.mmap_threshold) &&
    (mp_.n_mmaps < mp_.n_mmaps_max)) {
```

64

分配超大块(>=128KB)

#0 mmap () at ../sysdeps/unix/sysv/linux/i386/mmap.S:35
#1 0xb7817328 in sYSMALLOc (av=0xb4f00010, nb=131080) at malloc.c:2400
#2 _int_malloc (av=0xb4f00010, bytes=131072) at malloc.c:3932
#3 0xb7818f5c in __GI___libc_malloc (bytes=131072) at malloc.c:2924
#4 0x080492e4 in do_malloc (nsize=131072, no=1, threadno=1) at gemalloc.c:49
#5 0x0804947c in thread_func (arg=0x1) at gemalloc.c:111
#6 0xb7952d4c in start_thread (arg=0xb5810b40) at pthread_create.c:308
#7 0xb789187e in clone () at ../sysdeps/unix/sysv/linux/i386/clone.S:130


```
08725000-08745000 rw-p 00000000 00:00 0          [heap]
08745000-08785000 rw-p 00000000 00:00 0          [heap]
08785000-0879b000 rw-p 00000000 00:00 0          [heap]
0879b000-087db000 rw-p 00000000 00:00 0          [heap]
087db000-0881b000 rw-p 00000000 00:00 0          [heap]
0881b000-0885b000 rw-p 00000000 00:00 0          [heap]
0885b000-0889b000 rw-p 00000000 00:00 0          [heap]
0889b000-088db000 rw-p 00000000 00:00 0          [heap]
088db000-0891b000 rw-p 00000000 00:00 0          [heap]
0891b000-0895b000 rw-p 00000000 00:00 0          [heap]
0895b000-0899b000 rw-p 00000000 00:00 0          [heap]
0899b000-089db000 rw-p 00000000 00:00 0          [heap]
089db000-08a1b000 rw-p 00000000 00:00 0          [heap]
b2b00000-b2c41000 rw-p 00000000 00:00 0
b2c41000-b2d00000 ---p 00000000 00:00 0
b2d00000-b2f00000 rw-p 00000000 00:00 0
b2f00000-b3100000 rw-p 00000000 00:00 0
b3100000-b3200000 rw-p 00000000 00:00 0
b32f8000-b3319000 rw-p 00000000 00:00 0
b3319000-b333a000 rw-p 00000000 00:00 0
b333a000-b3fbb000 rw-p 00000000 00:00 0
b3fbb000-b4c3c000 rw-p 00000000 00:00 0
b4c3c000-b4d7d000 rw-p 00000000 00:00 0
b4d7d000-b4ebe000 rw-p 00000000 00:00 0
b4ebe000-b4edf000 rw-p 00000000 00:00 0
b4edf000-b4f00000 rw-p 00000000 00:00 0
b4f00000-b4fef000 rw-p 00000000 00:00 0
b4fef000-b5000000 ---p 00000000 00:00 0
```

概览 Arena 批发 分配过程 释放 Valgrind Asan

67

从外部接口到内部实现

```
/*------------------------ Public wrappers. --------------------------------*/

void*
public_mALLOc(size_t bytes)
{
  mstate ar_ptr;
  void *victim;

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

  arena_lookup(ar_ptr);

  arena_lock(ar_ptr, bytes);
  if(!ar_ptr)
    return 0;
  victim = _int_malloc(ar_ptr, bytes);
  if(!victim) {
    /* Maybe the failure is due to running out of mmapped areas. */
    if(ar_ptr != &main_arena) {
```

```
#0  _int_malloc (av=0xb7fc7440, bytes=10) at malloc.c:3485
#1  0xb7e97f5c in __GI___libc_malloc (bytes=10) at malloc.c:2924
#2  0xb7e980c7 in __GI___libc_malloc (bytes=10) at malloc.c:2917
#3  0x08048401 in main (argc=1, argv=0xbffff414) at geheap.c:9
```

68

内部分配函数

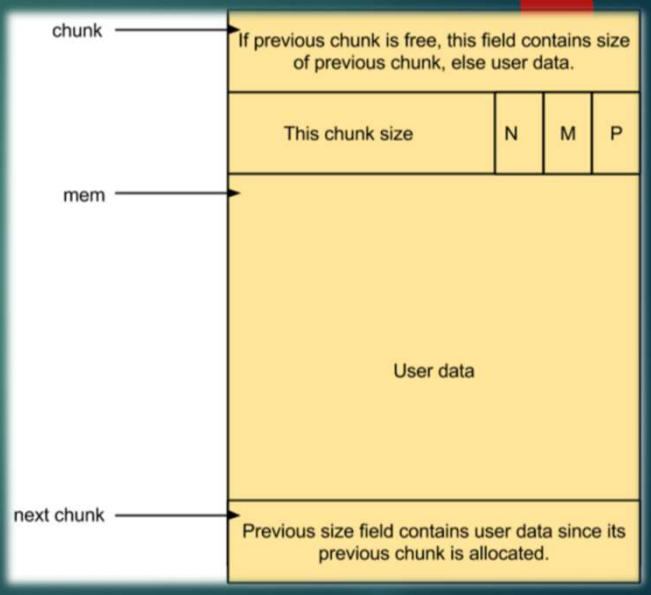
```
/*
  ---------------------------- malloc -----------------------------
*/
Void_t*
_int_malloc(mstate av, size_t bytes)
{
  INTERNAL_SIZE_T nb;               /* normalized request size */
  unsigned int    idx;              /* associated bin index */
  mbinptr         bin;              /* associated bin */
  mfastbinptr*    fb;               /* associated fastbin */

  mchunkptr       victim;           /* inspected/selected chunk */
  INTERNAL_SIZE_T size;             /* its size */
  int             victim_index;     /* its bin index */

  mchunkptr       remainder;        /* remainder from a split */
  unsigned long   remainder_size;   /* its size */

  unsigned int    block;            /* bit map traverser */
  unsigned int    bit;              /* bit map traverser */
  unsigned int    map;              /* current word of binmap */

  mchunkptr       fwd;              /* misc temp for linking */
  mchunkptr       bck;              /* misc temp for linking */
```


malloc_chunk

```
/*
  This struct declaration is misleading (but accurate and necessary).
  It declares a "view" into memory allowing access to necessary
  fields at known offsets from a given base. See explanation below.
*/

struct malloc_chunk {

  INTERNAL_SIZE_T      prev_size;  /* Size of previous chunk (if free).  */
  INTERNAL_SIZE_T      size;       /* Size in bytes, including overhead. */

  struct malloc_chunk* fd;         /* double links -- used only if free. */
  struct malloc_chunk* bk;

  /* Only used for large blocks: pointer to next larger size.  */
  struct malloc_chunk* fd_nextsize; /* double links -- used only if free. */
  struct malloc_chunk* bk_nextsize;
};
```

Allocated chunk

- prev_size: If the previous chunk is free, this field contains the size of previous chunk. Else if previous chunk is allocated, this field contains previous chunk's user data.
- size: This field contains the size of this allocated chunk. Last 3 bits of this field contains flag information.
  - PREV_INUSE (P) – This bit is set when previous chunk is allocated.
  - IS_MMAPPED (M) – This bit is set when chunk is mmap'd.
  - NON_MAIN_ARENA (N) – This bit is set when this chunk belongs to a thread arena.



71

```
/* size field is or'ed with PREV_INUSE when previous adjacent chunk in use */
#define PREV_INUSE 0x1

/* extract inuse bit of previous chunk */
#define prev_inuse(p)       ((p)->size & PREV_INUSE)


/* size field is or'ed with IS_MMAPPED if the chunk was obtained with mmap() */
#define IS_MMAPPED 0x2

/* check for mmap()'ed chunk */
#define chunk_is_mmapped(p) ((p)->size & IS_MMAPPED)


/* 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)
```

三个标志位

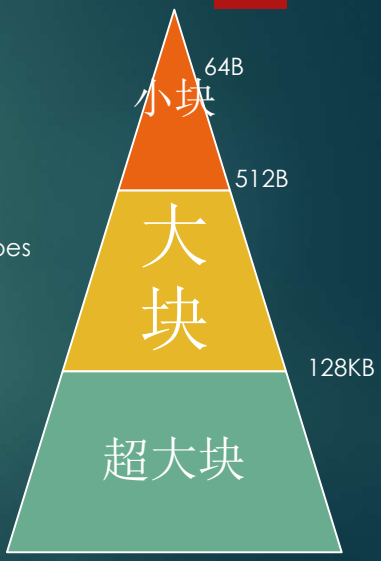
72

看人下菜碟儿

The main properties of the algorithms are:
* 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.

-- malloc.c

赵姨娘也不答话，走上来便将粉照着芳官脸上撒来，指着芳官骂道：“我家里下三等奴才也比你高贵些的，你都会看人下菜碟儿。宝玉要给东西，你拦在头里，莫不是要了你的了？拿这个哄他，你只当他不认得呢！”
--清·曹雪芹《红楼梦》第六十回 茉莉粉替去蔷薇硝 玫瑰露引来茯苓霜



73

图书馆目录的怀旧情结

- 在有目录的时代，读者见到的都是读者目录，放在目录室里，有一个专门的目录组负责，定期维护。记得先后担任《国家图书馆学刊》和《中国图书馆学报》常务副主编的蒋弘女士曾经在目录室工作过，南京图书馆《新世纪图书馆》常务副主编彭飞老师也当过目录组组长。采编工作人员负责的是公务目录有三套，包括采访目录、编目目录、书库目录，所有目录加起来至少有四大套，里面还包括著者目录、书名目录（或者著者和书名混合的排架目录）和分类目录等，卡片有7-8套，分别维护，很有难度。

节选自书蠹精的博客文章
http://blog.sina.com.cn/s/blog_495d62640102v5ol.html

74

细看卡片箱

75

卡片箱中的卡片

76

Bin（卡片箱）

Bin	卡片箱
索引空闲块	索引可借的书
有时也被称为空闲链表(free list)	读者卡片
为了加快释放和分配速度，释放时放入bin，分配时先在bin里寻找	提高查找书的速度
分成两大类，多种bin	“所有目录加起来至少有四大套，里面还包括著者目录、书名目录（或者著者和书名混合的排架目录）和分类目录等，卡片有7-8套”

77

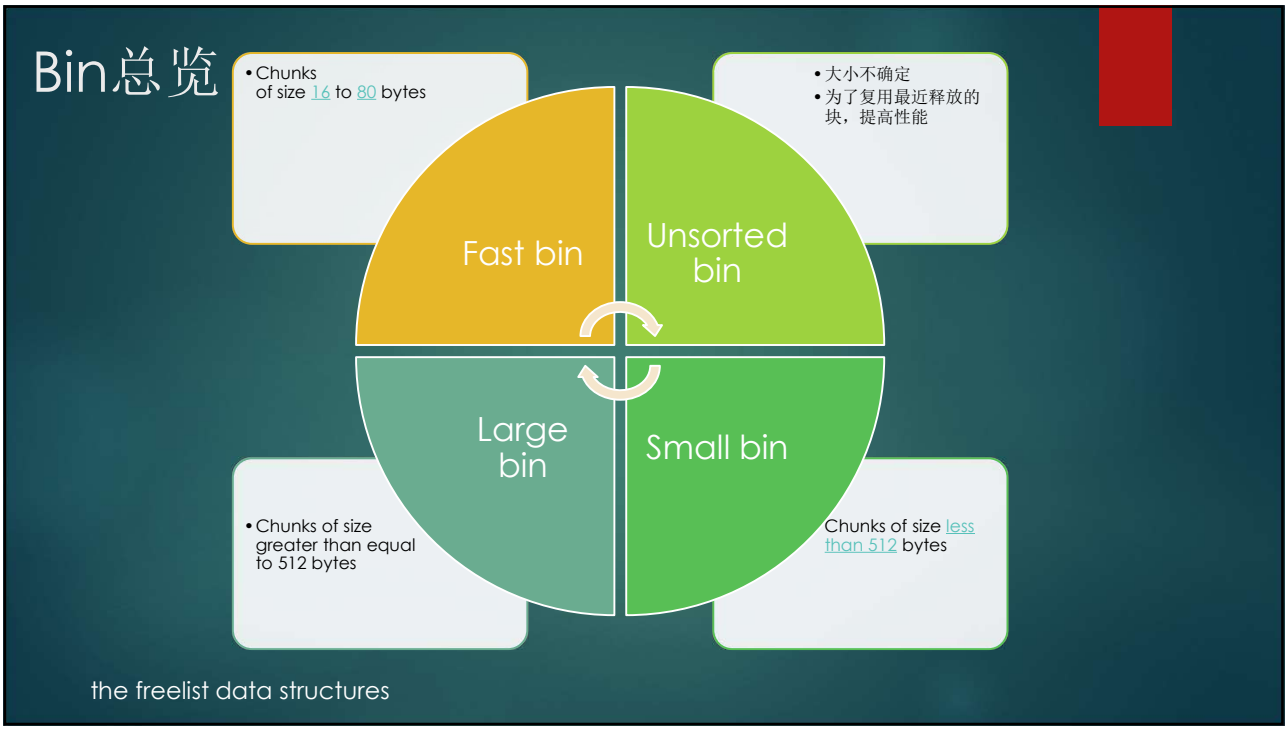
两类bin

fastbin
- 共有10个
- 只记录小于80字节的小块
- 每个bin里的块大小相同
- 单向链表

普通bin
- 分成三种
  - 无序，小块，大块
- 以双向链表索引

78

Bin总览



the freelist data structures

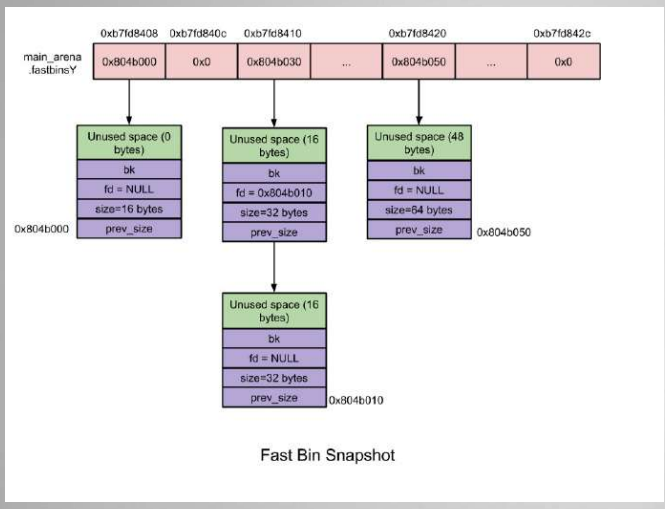
79

fastbin

- 里面的块不标记为空闲，保持busy状态
  - 释放和分配都快
- 单向链表
- 添加和删除都在头尾端进行，先进先出
- 按8字节单位划分
  - bin 0存放的都是16字节的块（用户区8字节）
  - Bin 1… 24
  - Bin 2… (2+2)*8 = 32
  - Bin 9… (9+2)*8 = 88 (用户区80字节)

```
/* The maximum fastbin request size we support */
#define MAX_FAST_SIZE     80

#define NFASTBINS  (fastbin_index(request2size(MAX_FAST_SIZE))+1)
```

80



Fastbin快照

从fastbin中分配堆块

```
/*
  If the size qualifies as a fastbin, first check corresponding bin.
  This code is safe to execute even if av is not yet initialized, so we
  can try it without checking, which saves some time on this fast path.
*/

if ((unsigned long)(nb) <= (unsigned long)(av->max_fast)) {
  fb = &(av->fastbins[(fastbin_index(nb))]);        //把fb指向对应的bin（卡片箱）
  if ( (victim = *fb) != 0) {                       //把箱子里的内容给victim，如果不为空说明有内容
    *fb = victim->fd;                               //把箱子里的内容更新为下一个
    check_remalloced_chunk(av, victim, nb);
    return chunk2mem(victim);
  }
}
```

```
#define fastbin(ar_ptr, idx) ((ar_ptr)->fastbinsY[idx])

/* offset 2 to use otherwise unindexable first 2 bins */
#define fastbin_index(sz) \
  ((((unsigned int)(sz)) >> (SIZE_SZ == 8 ? 4 : 3)) - 2)
```

Fastbin的块上限是可配置的

/*
M_MXFAST is the maximum request size used for "fastbins", special bins that hold returned chunks without consolidating their spaces. This enables future requests for chunks of the same size to be handled very quickly, but **can increase fragmentation**, and thus increase the overall memory footprint of a program.

This malloc manages fastbins very conservatively yet still efficiently, so fragmentation is rarely a problem for values less than or equal to the default. The maximum supported value of MXFAST is 80. You wouldn't want it any higher than this anyway. Fastbins are designed especially for use with many small structs, objects or strings -- the default handles structs/objects/arrays with sizes up
to 8 4byte fields, or small strings representing words, tokens, etc. Using fastbins for larger objects normally worsens fragmentation without improving speed.

M_MXFAST is set in REQUEST size units. It is internally used in chunksize units, which adds padding and alignment. You can reduce M_MXFAST to 0 to disable all use of fastbins. This causes the malloc algorithm to be a closer approximation of fifo-best-fit in all cases, not just for larger requests, but will generally cause it to be slower.
*/

83

av->max_fast为什么是72而不是88？

```
(gdb) p av->max_fast
$46 = 72
```

```
#ifndef DEFAULT_MXFAST
#define DEFAULT_MXFAST     64
#endif
```

```
(gdb) p global_max_fast
$8 = 64
```

84

格物

```
#0  _int_malloc (av=0x80521a0, bytes=8) at malloc.c:3851
#1  0x0804cf49 in malloc (bytes=8) at malloc.c:3350
#2  0xb7269474 in ?? () from /usr/lib/i386-linux-gnu/libxcb.so.1
#3  0xb72697db in ?? () from /usr/lib/i386-linux-gnu/libxcb.so.1
#4  0xb76a1a8a in ?? () from /usr/lib/i386-linux-gnu/libX11.so.6
#5  0xb76a1b8f in ?? () from /usr/lib/i386-linux-gnu/libX11.so.6
#6  0xb76a239f in _XEventsQueued () from /usr/lib/i386-linux-gnu/libX11.so.6
#7  0xb7693138 in XPending () from /usr/lib/i386-linux-gnu/libX11.so.6
#8  0xb7b07f3c in ?? () from /usr/lib/i386-linux-gnu/libgdk-x11-2.0.so.0
#9  0xb79adb3b in g_main_context_check () from /lib/i386-linux-gnu/libglib-2.0.so.0
#10 0xb79ae002 in ?? () from /lib/i386-linux-gnu/libglib-2.0.so.0
#11 0xb79ae52b in g_main_loop_run () from /lib/i386-linux-gnu/libglib-2.0.so.0
#12 0xb7c96bff in gtk_main () from /usr/lib/i386-linux-gnu/libgtk-x11-2.0.so.0
#13 0x0804a650 in main (argc=1, argv=0xbffff404) at gemalloc.c:301
```



```
(gdb) p nb
$55 = 16
(gdb) p fb
$56 = (mfastbinptr *) 0x80521bc
(gdb) p av->fastbins[0]
$57 = (mfastbinptr) 0x80f8f18
(gdb) p &av->fastbins[0]
$58 = (mfastbinptr *) 0x80521bc
(gdb) p *av->fastbins[0]
$59 = {prev_size = 1145324612, size = 17, fd = 0x80ce370, bk = 0x80f4f30}
(gdb) p *fb
$60 = (mfastbinptr) 0x80f8f18
(gdb) x /xw 0x80521bc
0x80521bc <main_arena+28>:    0x080f8f18
(gdb) p /t 17
$61 = 10001
(gdb) p victim
$62 = (mchunkptr) 0x80f8f18
(gdb) l
3848      if ((unsigned long)(nb) <= (unsigned long)(av->max_fast)) {
3849        fb = &(av->fastbins[(fastbin_index(nb))]);
3850        if ( (victim = *fb) != 0) {
3851          *fb = victim->fd;
3852          check_remalloced_chunk(av, victim, nb);
```

块总大小

把fb指向bin

Bin中目前的块

总大小为16字节，注意，第三位是标志

将要更新卡片箱

```
(gdb) p nb
$70 = 40
(gdb) macro expand fastbin_index(40)
expands to: ((((unsigned int)(40)) >> 3) - 2)
(gdb) p ((((unsigned int)(40)) >> 3) - 2)
$71 = 3
(gdb) p &av->fastbins[3]
$72 = (mfastbinptr *) 0x80521c8
(gdb) p fb
$73 = (mfastbinptr *) 0x80521c8
(gdb) p av->fastbins[3]
$74 = (mfastbinptr) 0x80f12d8
(gdb) p *fb
$75 = (mfastbinptr) 0x80f12d8
(gdb) p victim
$76 = (mchunkptr) 0x80f12d8
(gdb) p *victim
$77 = {prev_size = 0, size = 41, fd = 0x80cdb60, bk = 0x0}
(gdb) l
3848      if ((unsigned long)(nb) <= (unsigned long)(av->max_fast)) {
3849        fb = &(av->fastbins[(fastbin_index(nb))]);
3850        if ( (victim = *fb) != 0) {
3851          *fb = victim->fd;
3852          check_remalloced_chunk(av, victim, nb);
3853          return chunk2mem(victim);
```

```
#0  _int_malloc (av=0x80521a0, bytes=36) at mallo
#1  0x0804cf49 in malloc (bytes=36) at malloc.c:3350
#2  0xb7269394 in ?? () from /usr/lib/i386-linux-gnu/libxc
```

```
(gdb) n
3853          return chunk2mem(victim);
(gdb) p av->fastbins[3]
$79 = (mfastbinptr) 0x80cdb60
```

单步一行后，更新了bin里的内容

87

常规bin

- 共128个
  - 双向链表管理，每两个元素对应一个bin
  - Bin 1 – Unsorted bin
  - Bin 2 to Bin 63 – Small bin
  - Bin 64 to Bin 126 – Large bin

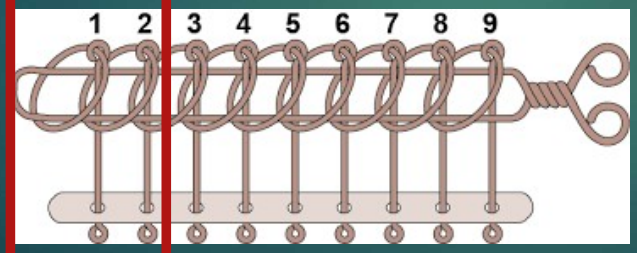
```
#define NBINS           128
#define NSMALLBINS       64
```

```
(gdb) pt main_arena
type = struct malloc_state {
    mutex_t mutex;
    long int stat_lock_direct;
    long int stat_lock_loop;
    long int stat_lock_wait;
    long int pad0_[1];
    size_t max_fast;
    mfastbinptr fastbins[10];
    mchunkptr top;
    mchunkptr last_remainder;
    mchunkptr bins[256];
    unsigned int binmap[4];
    struct malloc_state *next;
    size_t system_mem;
    size_t max_system_mem;
}
```

88

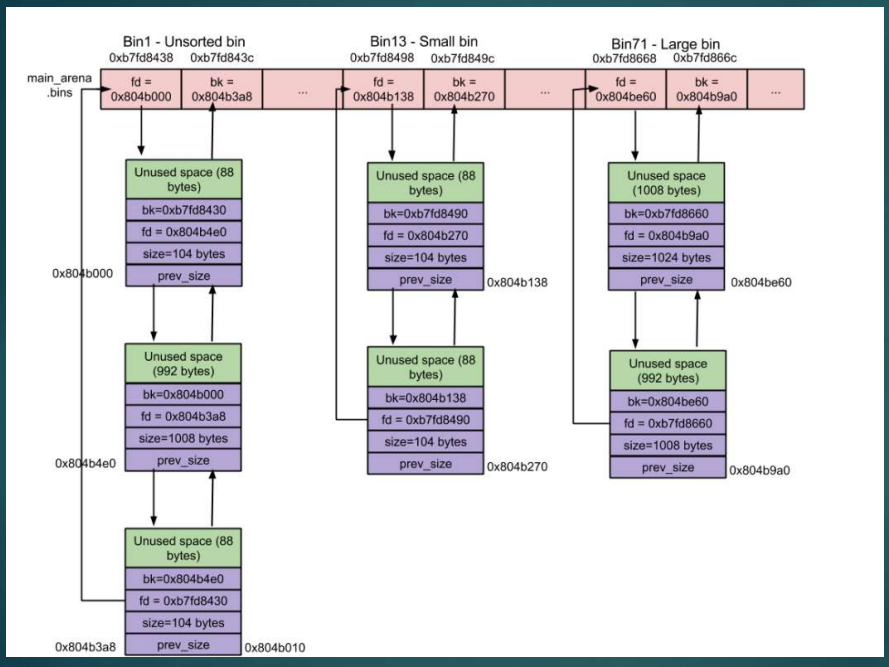
普通bin（regular bin）

- 双向链表
- 以数组形式定义，256元素，每个元素是malloc_chunk *
- 两个数组元素组成一个bin，与其它空闲块的fd和bk指针手拉手串起来




```
(gdb) ptype main_arena.bins[1]
type = struct malloc_chunk {
    size_t prev_size;
    size_t size;
    struct malloc_chunk *fd;
    struct malloc_chunk *bk;
} *

(gdb) p sizeof(main_arena.bins[1])
$8 = 4
(gdb) p sizeof(main_arena.bins)
$9 = 1024
```

89



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

90

从小箱分配

```
if (in_smallbin_range(nb)) {
  idx = smallbin_index(nb);
  bin = bin_at(av,idx);

  if ( (victim = last(bin)) != bin) {
    if (victim == 0) /* initialization check */
      malloc_consolidate(av);
    else {
      bck = victim->bk;
      set_inuse_bit_at_offset(victim, nb);
      bin->bk = bck;
      bck->fd = bin;

      if (av != &main_arena)
        victim->size |= NON_MAIN_ARENA;
      check_malloced_chunk(av, victim, nb);
      return chunk2mem(victim);
    }
  }
}
```

```
/*
  If a small request, check regular bin.  Since these "smallbins"
  hold one size each, no searching within bins is necessary.
  (For a large request, we need to wait until unsorted chunks are
  processed to find best fit. But for small ones, fits are exact
  anyway, so we can check now, which is faster.)
*/
```

91

格物

```
(gdb) p nb
$80 = 40
(gdb) p idx
$81 = 5

(gdb) p av->bins[10]
$85 = (mchunkptr) 0x805220c
(gdb) p av->bins[11]
$86 = (mchunkptr) 0x805220c
```

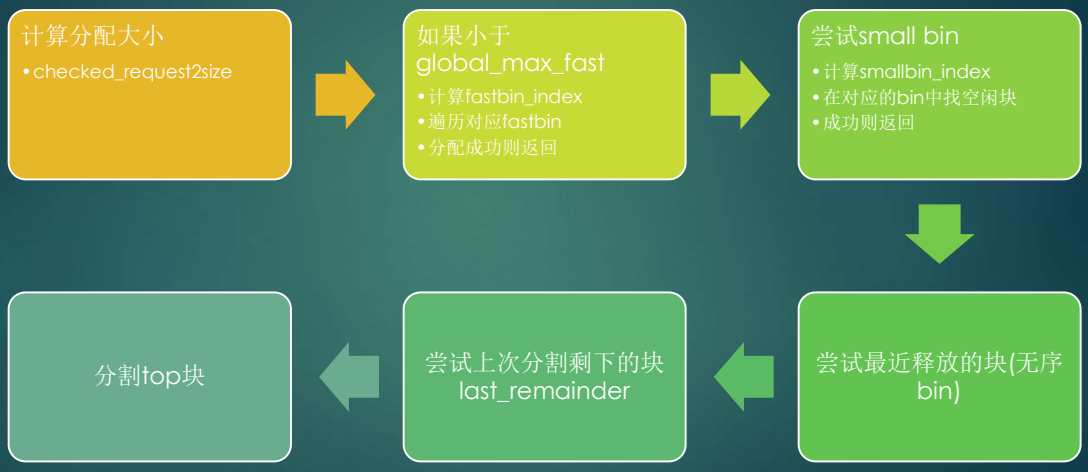
```
#0  _int_malloc (av=0x80521a0, bytes=36) at malloc.c:3918
#1  0x0804cf49 in malloc (bytes=36) at malloc.c:3350
#2  0xb7269394 in ?? () from /usr/lib/i386-linux-gnu/libxcb.so.1
#3  0xb72697db in ?? () from /usr/lib/i386-linux-gnu/libxcb.so.1
```

请求36字节，fastbin不能满足请求，则尝试smallbin，计算出是5号箱，但是为空，于是会尝试无序bin

```
#define SMALLBIN_WIDTH      8
#define MIN_LARGE_SIZE    512

#define in_smallbin_range(sz)  \
  ((unsigned long)(sz) < (unsigned long)MIN_LARGE_SIZE)

#define smallbin_index(sz)     (((unsigned)(sz)) >> 3)
```

92

小块分配过程



93

跟踪malloc

(gdb) bt
#0 _int_malloc (av=0xb7fc7440, bytes=2000) at malloc.c:3902
#1 0xb7e9af2c in __GI___libc_malloc (bytes=2000) at malloc.c:2924
#2 0xb7e9b097 in __GI___libc_malloc (bytes=2000) at malloc.c:2917
#3 0x08048730 in main (argc=3, argv=0xbffff404) at geheap.c:54

```
ge@gewubox: ~/work/heap
(gdb) n
3790            assert((unsigned long)(size) >= (unsigned long)(nb));
(gdb) n
3794            /* unlink */
(gdb) n
3791
(gdb) n
3792            remainder_size = size - nb;
(gdb) n
3793
(gdb) n
3799              set_inuse_bit_at_offset(victim, size);
(gdb) n
3801              victim->size |= NON_MAIN_ARENA;
(gdb) n
3803
(gdb) n
3801              victim->size |= NON_MAIN_ARENA;
(gdb) n
3899
```

94

```
============== After allocating blocks ==============
Total non-mmapped bytes (arena):       135168
# of free chunks (ordblks):            1
# of free fastbin blocks (smblks):     0
# of mapped regions (hblks):           0
Bytes in mapped regions (hblkhd):      0
Max. total allocated space (usmblks):  0
Free bytes held in fastbins (fsmblks): 0
Total allocated space (uordblks):      2016
Total free space (fordblks):           133152
Topmost releasable block (keepcost):   133152

============== After freeing blocks ==============
Total non-mmapped bytes (arena):       135168
# of free chunks (ordblks):            1
# of free fastbin blocks (smblks):     0
# of mapped regions (hblks):           0
Bytes in mapped regions (hblkhd):      0
Max. total allocated space (usmblks):  0
Free bytes held in fastbins (fsmblks): 0
Total allocated space (uordblks):      0
Total free space (fordblks):           135168
Topmost releasable block (keepcost):   135168
```

调用malloc(1000)两次，块头占8字节

通过mallinfo接口获取信息

https://www.systutorials.com/docs/linux/man/3-mallinfo/

95

```
(gdb) p *ar_ptr
$5 = {mutex = 1, flags = 1, fastbinsY = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
    0x0}, top = 0x804b7d8, last_remainder = 0x0, bins = {0xb7fc7470, 0xb7fc7470,
    0xb7fc7478, 0xb7fc7478, 0xb7fc7480, 0xb7fc7480, 0xb7fc7488, 0xb7fc7488,
    0xb7fc7490, 0xb7fc7490, 0xb7fc7498, 0xb7fc7498, 0xb7fc74a0, 0xb7fc74a0,
    0xb7fc74a8, 0xb7fc74a8, 0xb7fc74b0, 0xb7fc74b0, 0xb7fc74b8, 0xb7fc74b8,
    0xb7fc74c0, 0xb7fc74c0, 0xb7fc74c8, 0xb7fc74c8, 0xb7fc74d0, 0xb7fc74d0,
    0xb7fc74d8, 0xb7fc74d8, 0xb7fc74e0, 0xb7fc74e0, 0xb7fc74e8, 0xb7fc74e8,
    0xb7fc74f0, 0xb7fc74f0, 0xb7fc74f8, 0xb7fc74f8, 0xb7fc7500, 0xb7fc7500,
    0xb7fc7508, 0xb7fc7508, 0xb7fc7510, 0xb7fc7510, 0xb7fc7518, 0xb7fc7518,
    0xb7fc7520, 0xb7fc7520, 0xb7fc7528, 0xb7fc7528, 0xb7fc7530, 0xb7fc7530,
    0xb7fc7538, 0xb7fc7538, 0xb7fc7540, 0xb7fc7540, 0xb7fc7548, 0xb7fc7548,
    0xb7fc7550, 0xb7fc7550, 0xb7fc7558, 0xb7fc7558, 0xb7fc7560, 0xb7fc7560,
    0xb7fc7568, 0xb7fc7568, 0xb7fc7570, 0xb7fc7570, 0xb7fc7578, 0xb7fc7578,
    0xb7fc7580, 0xb7fc7580, 0xb7fc7588, 0xb7fc7588, 0xb7fc7590, 0xb7fc7590,
    0xb7fc7598, 0xb7fc7598, 0xb7fc75a0, 0xb7fc75a0, 0xb7fc75a8, 0xb7fc75a8,
    0xb7fc75b0, 0xb7fc75b0, 0xb7fc75b8, 0xb7fc75b8, 0xb7fc75c0, 0xb7fc75c0,
    0xb7fc75c8, 0xb7fc75c8, 0xb7fc75d0, 0xb7fc75d0, 0xb7fc75d8, 0xb7fc75d8,
    0xb7fc75e0, 0xb7fc75e0, 0xb7fc75e8, 0xb7fc75e8, 0xb7fc75f0, 0xb7fc75f0,
    0xb7fc75f8, 0xb7fc75f8, 0xb7fc7600, 0xb7fc7600, 0xb7fc7608, 0xb7fc7608,
    0xb7fc7610, 0xb7fc7610, 0xb7fc7618, 0xb7fc7618, 0xb7fc7620, 0xb7fc7620,
    0xb7fc7628, 0xb7fc7628, 0xb7fc7630, 0xb7fc7630, 0xb7fc7638, 0xb7fc7638,
    0xb7fc7640, 0xb7fc7640, 0xb7fc7648, 0xb7fc7648, 0xb7fc7650, 0xb7fc7650,
    0xb7fc7658, 0xb7fc7658, 0xb7fc7660, 0xb7fc7660, 0xb7fc7668, 0xb7fc7668,
    0xb7fc7670, 0xb7fc7670, 0xb7fc7678, 0xb7fc7678, 0xb7fc7680, 0xb7fc7680,
    0xb7fc7688, 0xb7fc7688, 0xb7fc7690, 0xb7fc7690, 0xb7fc7698, 0xb7fc7698,
    0xb7fc76a0, 0xb7fc76a0, 0xb7fc76a8, 0xb7fc76a8, 0xb7fc76b0, 0xb7fc76b0,
    0xb7fc76b8, 0xb7fc76b8, 0xb7fc76c0, 0xb7fc76c0, 0xb7fc76c8, 0xb7fc76c8,
    0xb7fc76d0, 0xb7fc76d0, 0xb7fc76d8, 0xb7fc76d8, 0xb7fc76e0, 0xb7fc76e0,
    0xb7fc76e8, 0xb7fc76e8, 0xb7fc76f0, 0xb7fc76f0, 0xb7fc76f8, 0xb7fc76f8,
    0xb7fc7700, 0xb7fc7700, 0xb7fc7708, 0xb7fc7708, 0xb7fc7710, 0xb7fc7710,
    0xb7fc7718, 0xb7fc7718, 0xb7fc7720, 0xb7fc7720, 0xb7fc7728, 0xb7fc7728,
    0xb7fc7730, 0xb7fc7730, 0xb7fc7738, 0xb7fc7738, 0xb7fc7740, 0xb7fc7740,
    0xb7fc7748, 0xb7fc7748, 0xb7fc7750, 0xb7fc7750, 0xb7fc7758, 0xb7fc7758,
    0xb7fc7760, 0xb7fc7760, 0xb7fc7768, 0xb7fc7768, 0xb7fc7770, 0xb7fc7770,
    0xb7fc7778, 0xb7fc7778, 0xb7fc7780, 0xb7fc7780, 0xb7fc7788, 0xb7fc7788...},
  binmap = {0, 0, 0, 0}, next = 0xb7fc7440, next_free = 0x0, system_mem = 135168,
  max_system_mem = 135168}
```

96

概览 Arena 批发 分配过程 释放 Valgrind Asan

97

98

```
#0  free (mem=0x80f3d48) at malloc.c:3385
#1  0xb72682dd in ?? () from /usr/lib/i386-linux-gnu/libxcb.so.1
#2  0xb72697e6 in ?? () from /usr/lib/i386-linux-gnu/libxcb.so.1
#3  0xb76a1a8a in ?? () from /usr/lib/i386-linux-gnu/libX11.so.6
#4  0xb76a1b8f in ?? () from /usr/lib/i386-linux-gnu/libX11.so.6
…
#8  0xb79adb3b in g_main_context_check () from /lib/i386-linux-gnu/libglib-2.0.so.0
#9  0xb79ae002 in ?? () from /lib/i386-linux-gnu/libglib-2.0.so.0
#10 0xb79ae52b in g_main_loop_run () from /lib/i386-linux-gnu/libglib-2.0.so.0
#11 0xb7c96bff in gtk_main () from /usr/lib/i386-linux-gnu/libgtk-x11-2.0.so.0
#12 0x0804a650 in main (argc=1, argv=0xbffff404) at gemalloc.c:301
(gdb) n
3387        if (hook != NULL) {
(gdb)
3392        if (mem == 0)                              /* free(0) has no effect */
(gdb)
3395        p = mem2chunk(mem);
(gdb)
3398        if (chunk_is_mmapped(p))                       /* release mmapped memory. */
(gdb) p p
$90 = (mchunkptr) 0x80f3d40
(gdb) p *p
$91 = {prev_size = 55, size = 17, fd = 0x80f33c8, bk = 0x80ce818}
```

99

续

```
(gdb) n
3405        ar_ptr = arena_for_chunk(p);
(gdb) s
3407        if(!mutex_trylock(&ar_ptr->mutex))
(gdb) macro expand arena_for_chunk(p)
expands to: (((p)->size & 0x4) ? ((heap_info *)((unsigned long)(p) & ~((1024*1024)-1)))->ar_ptr : &main_a
(gdb) n
3408          ++(ar_ptr->stat_lock_direct);
(gdb)
3416        _int_free(ar_ptr, mem);
```

100

_int_free – 释放到fastbin

```
(gdb) s
_int_free (av=0x80521a0, mem=0x80f3d48) at malloc.c:4206
4206       if (mem != 0) {
(gdb) n
4207         p = mem2chunk(mem);
(gdb)
4208         size = chunksize(p);
(gdb)
4217         if ((unsigned long)(size) <= (unsigned long)(av->max_fast)
(gdb)
4228           set_fastchunks(av);
(gdb)
4229           fb = &(av->fastbins[fastbin_index(size)]);
(gdb)
4230           p->fd = *fb;
(gdb)
4231           *fb = p;
(gdb)
4347     }
```

链表更新好就OK了，不必设置为空闲块

101

set_fastchunks

Max_fast的最低两位是标志位

```
(gdb) macro expand set_fastchunks(av)
expands to: ((av)->max_fast &= ~(1U))
(gdb) p av->max_fast
$95 = 72
(gdb) n
4228           set_fastchunks(av);
(gdb)
4229           fb = &(av->fastbins[fastbin_index(size)]);
(gdb) p av->max_fast
$96 = 72
```

102

直接返还超大块

```
/*
    If the chunk was allocated via mmap, release via munmap(). Note
    that if HAVE_MMAP is false but chunk_is_mmapped is true, then
    user must have overwritten memory. There's nothing we can do to
    catch this error unless MALLOC_DEBUG is set, in which case
    check_inuse_chunk (above) will have triggered error.
  */

  else {
#if HAVE_MMAP
    int ret;
    INTERNAL_SIZE_T offset = p->prev_size;
    mp_.n_mmaps--;
    mp_.mmapped_mem -= (size + offset);
    ret = munmap((char*)p - offset, size + offset);
    /* munmap returns non-zero on failure */
    assert(ret == 0);
#endif
  }
```

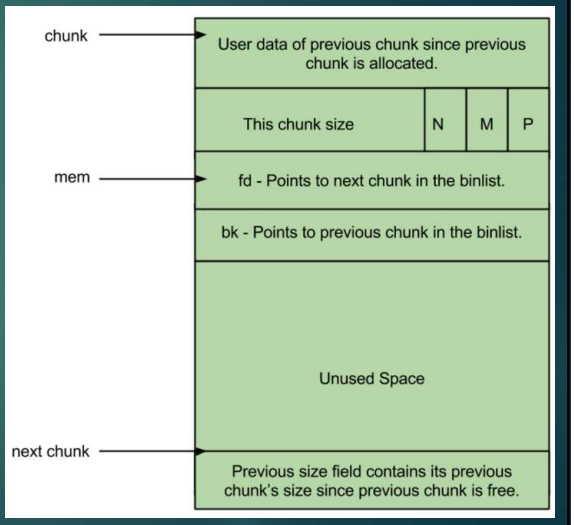
103

```
static void
munmap_chunk (mchunkptr p)
{
  INTERNAL_SIZE_T size = chunksize (p);

  assert (chunk_is_mmapped (p));

  /* Do nothing if the chunk is a faked mmapped chunk in the dumped
     main arena.  We never free this memory.  */
  if (DUMPED_MAIN_ARENA_CHUNK (p))
    return;

  uintptr_t block = (uintptr_t) p - prev_size (p);
  size_t total_size = prev_size (p) + size;
  /* Unfortunately we have to do the compilers job by hand here.  Normally
     we would test BLOCK and TOTAL-SIZE separately for compliance with the
     page size.  But gcc does not recognize the optimization possibility
     (in the moment at least) so we combine the two values into one before
     the bit test.  */
  if (__builtin_expect (((block | total_size) & (GLRO (dl_pagesize) - 1)) != 0, 0))
    malloc_printerr ("munmap_chunk(): invalid pointer");

  atomic_decrement (&mp_.n_mmaps);
  atomic_add (&mp_.mmapped_mem, -total_size);

  /* If munmap failed the process virtual memory address space is in a
     bad shape.  Just leave the block hanging around, the process will
     terminate shortly anyway since not much can be done.  */
  __munmap ((char *) block, total_size);
}
```

104

Free Chunk

- prev_size: No two free chunks can be adjacent together. When both the chunks are free, its gets combined into one single free chunk. Hence always previous chunk to this freed chunk would be allocated and therefore prev_size contains previous chunk's user data.
- size: This field contains the size of this free chunk.
- fd: Forward pointer – Points to next chunk in the same bin (and NOT to the next chunk present in physical memory).
- bk: Backward pointer – Points to previous chunk in the same bin (and NOT to the previous chunk present in physical memory).



105

```
nextchunk-> +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |             Size of chunk                                     |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+


    Where "chunk" is the front of the chunk for the purpose of most of
    the malloc code, but "mem" is the pointer that is returned to the
    user.  "Nextchunk" is the beginning of the next contiguous chunk.

    Chunks always begin on even word boundries, so the mem portion
    (which is returned to the user) is also on an even word boundary, and
    thus at least double-word aligned.

    Free chunks are stored in circular doubly-linked lists, and look like this:

    chunk-> +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |             Size of previous chunk                            |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    `head:' |             Size of chunk, in bytes                     |P|
      mem-> +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |             Forward pointer to next chunk in list             |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |             Back pointer to previous chunk in list            |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |             Unused space (may be 0 bytes long)                .
        .                                                               .
        .                                                               |
nextchunk-> +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    `foot:' |             Size of chunk, in bytes                           |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

    The P (PREV_INUSE) bit, stored in the unused low-order bit of the
    chunk size (which is always a multiple of two words), is an in-use
    bit for the *previous* chunk.  If that bit is *clear*, then the
    word before the current chunk size contains the previous chunk
    size, and can be used to find the front of the previous chunk.
    The very first chunk allocated always has this bit set,
    preventing access to non-existent (or non-owned) memory. If
```

- malloc.c

106

何时放入常规bin?

/*
Place the chunk in unsorted chunk list. Chunks are not placed into regular bins until after they have been given one chance to be used in malloc. */

- 释放时先放到无序bin
- 下次执行malloc时，放入所属的常规bin

```
/* place chunk in bin */

if (in_smallbin_range(size)) {
  victim_index = smallbin_index(size);
  bck = bin_at(av, victim_index);
  fwd = bck->fd;
}
else {
  victim_index = largebin_index(size);
  bck = bin_at(av, victim_index);
  fwd = bck->fd;

  if (fwd != bck) {
    /* if smaller than smallest, place first */
assert((bck->bk->size & NON_MAIN_ARENA) == 0);
    if ((unsigned long)(size) < (unsigned long)(bck->bk->size)) {
      fwd = bck;
      bck = bck->bk;
    }
    else if ((unsigned long)(size) >=
             (unsigned long)(FIRST_SORTED_BIN_SIZE)) {

      /* maintain large bins in sorted order */
      size |= PREV_INUSE; /* Or with inuse bit to speed comparisons */
  assert((fwd->size & NON_MAIN_ARENA) == 0);
      while ((unsigned long)(size) < (unsigned long)(fwd->size)) {
        fwd = fwd->fd;
  assert((fwd->size & NON_MAIN_ARENA) == 0);
  }
      bck = fwd->bk;
    }
  }
}

mark_bin(av, victim_index);
victim->bk = bck;
victim->fd = fwd;
fwd->bk = victim;
bck->fd = victim;
}
```

107

0804c000-0806d000 rw-p 00000000 00:00 0 [heap]
0806d000-08090000 rw-p 00000000 00:00 0 [heap]
08090000-**080b2000** rw-p 00000000 00:00 0 [heap]

#0 __GI___sbrk (increment=-4096) at sbrk.c:35
#1 0xb79c4f4f in __GI___default_morecore (increment=-4096) at morecore.c:49
#2 0xb79beedd in sYSTRIm (pad=<optimized out>, av=<optimized out>) at malloc.c:2810
#3 0xb79bfcb5 in _int_free (av=0xb7af2440, p=0x808fff8, have_lock=1) at malloc.c:4196

0804c000-0806d000 rw-p 00000000 00:00 0 [heap]
0806d000-08090000 rw-p 00000000 00:00 0 [heap]
08090000-**080b1000** rw-p 00000000 00:00 0 [heap]

108

109

```
ge@gewubox:~/work/heap$ cat /proc/meminfo
MemTotal:         766212 kB
MemFree:           90056 kB
Buffers:           12236 kB
Cached:           275296 kB
SwapCached:          348 kB
Active:           306888 kB
Inactive:         304720 kB
Active(anon):     156212 kB
Inactive(anon):   182932 kB
Active(file):     150676 kB
Inactive(file):   121788 kB
Unevictable:           0 kB
Mlocked:               0 kB
HighTotal:             0 kB
HighFree:              0 kB
LowTotal:         766212 kB
LowFree:           90056 kB
SwapTotal:        783356 kB
SwapFree:         778140 kB
Dirty:                28 kB
Writeback:             0 kB
AnonPages:        323768 kB
Mapped:            93660 kB
Shmem:             15068 kB
Slab:              44596 kB
SReclaimable:      32576 kB
SUnreclaim:        12020 kB
KernelStack:        3008 kB
PageTables:         7428 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:     1166460 kB
Committed_AS:    2400960 kB
VmallocTotal:     249912 kB
VmallocUsed:       20264 kB
VmallocChunk:     221168 kB
HardwareCorrupted:     0 kB
AnonHugePages:         0 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
DirectMap4k:       34752 kB
DirectMap2M:      751616 kB
```

110

free

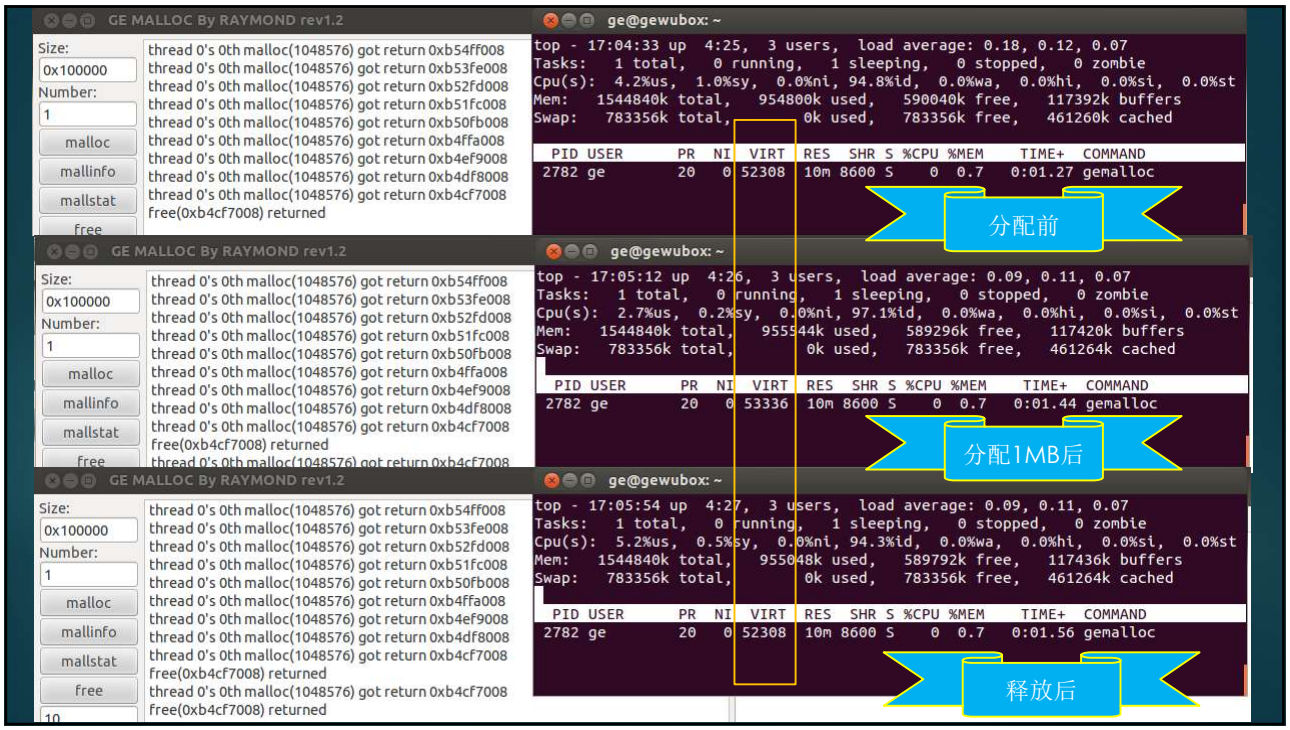
```
ge@gewubox:~/work/heap$ free -l -t
             total       used       free     shared    buffers     cached
Mem:        766212     676192      90020          0      12348     275304
Low:        766212     676192      90020
High:            0          0          0
-/+ buffers/cache:     388540     377672
Swap:       783356       5216     778140
Total:     1549568     681408     868160
```

111

top

```
top - 17:05:20 up  9:10,  3 users,  load average: 0.13, 0.09, 0.28
Tasks: 162 total,   1 running, 161 sleeping,   0 stopped,   0 zombie
Cpu(s):  3.1%us,  0.7%sy,  0.0%ni, 96.2%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:    766212k total,   677876k used,    88336k free,    12392k buffers
Swap:   783356k total,     5216k used,   778140k free,   275400k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 1065 root      20   0  142m  94m  20m S  4.0 12.6   1:32.29 Xorg
 2406 ge        20   0 96536  16m  10m S  2.3  2.2   0:18.07 gnome-terminal
 1917 ge        20   0  119m  12m 9984 S  0.3  1.7   0:06.95 metacity
 1936 ge        20   0  254m  53m  29m S  0.3  7.1   0:32.82 unity-2d-shell
 3804 ge        20   0  152m  35m  27m S  0.3  4.7   0:02.81 unity-2d-spread
    1 root      20   0  3520 1784 1228 S  0.0  0.2   0:03.21 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kthreadd
    3 root      20   0     0    0    0 S  0.0  0.0   0:00.29 ksoftirqd/0
    5 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 kworker/0:0H
    7 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/0
    8 root      20   0     0    0    0 S  0.0  0.0   0:00.00 rcu_bh
    9 root      20   0     0    0    0 S  0.0  0.0   0:00.84 rcu_sched
   10 root      RT   0     0    0    0 S  0.0  0.0   0:00.59 watchdog/0
   11 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 khelper
   12 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kdevtmpfs
   13 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 netns
   14 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 writeback
   15 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 kintegrityd
   16 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 bioset
   17 root       0 -20     0    0    0 S  0.0  0.0   0:00.10 kworker/u3:0
   18 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 kblockd
   19 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 ata_sff
   20 root      20   0     0    0    0 S  0.0  0.0   0:00.05 khubd
```

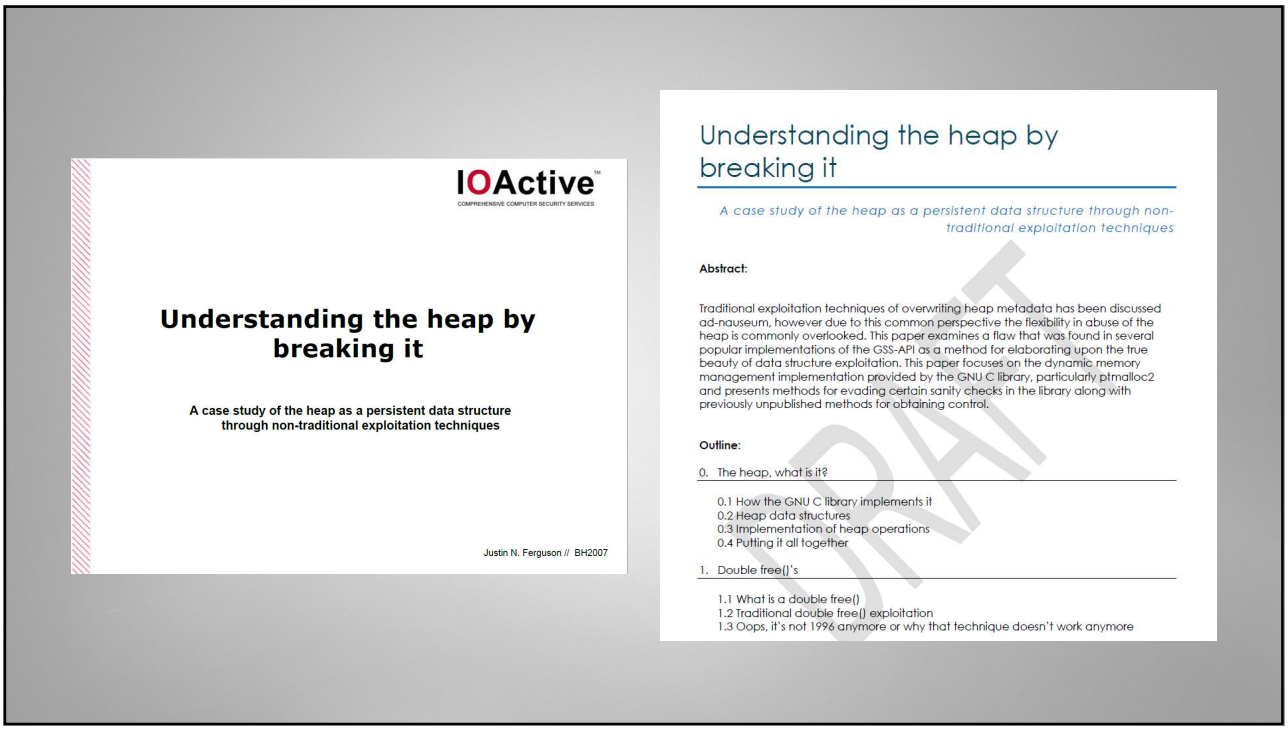
112



堆陷阱

- 多次释放
- 使用释放后的块
- 释放野指针
- 溢出

- 刀刀致命



115

Double Free

```
*** glibc detected *** ./geheapd: double free or corruption (!prev): 0x099ba008 ***
======= Backtrace: =========
/lib/i386-linux-gnu/libc.so.6(+0x74fd2)[0xb75f1fd2]
./geheapd[0x80487ba]
/lib/i386-linux-gnu/libc.so.6(__libc_start_main+0xf3)[0xb7596533]
./geheapd[0x8048471]
======= Memory map: ========
08048000-08049000 r-xp 00000000 08:01 19         /home/ge/work/heap/geheapd
08049000-0804a000 r--p 00000000 08:01 19         /home/ge/work/heap/geheapd
0804a000-0804b000 rw-p 00001000 08:01 19         /home/ge/work/heap/geheapd
099ba000-099db000 rw-p 00000000 00:00 0          [heap]
b754e000-b756a000 r-xp 00000000 08:01 132094     /lib/i386-linux-gnu/libgcc_s.so.1
b756a000-b756b000 r--p 0001b000 08:01 132094     /lib/i386-linux-gnu/libgcc_s.so.1
b756b000-b756c000 rw-p 0001c000 08:01 132094     /lib/i386-linux-gnu/libgcc_s.so.1
b757c000-b757d000 rw-p 00000000 00:00 0
b757d000-b7720000 r-xp 00000000 08:01 154504     /lib/i386-linux-gnu/libc-2.15.so
b7720000-b7722000 r--p 001a3000 08:01 154504     /lib/i386-linux-gnu/libc-2.15.so
b7722000-b7723000 rw-p 001a5000 08:01 154504     /lib/i386-linux-gnu/libc-2.15.so
b7723000-b7726000 rw-p 00000000 00:00 0
b7734000-b7735000 rw-p 00000000 00:00 0
b7735000-b7736000 rw-p 00000000 00:00 0
b7736000-b7738000 rw-p 00000000 00:00 0
b7738000-b7739000 r-xp 00000000 00:00 0          [vdso]
b7739000-b7759000 r-xp 00000000 08:01 154510     /lib/i386-linux-gnu/ld-2.15.so
b7759000-b775a000 r--p 0001f000 08:01 154510     /lib/i386-linux-gnu/ld-2.15.so
b775a000-b775b000 rw-p 00020000 08:01 154510     /lib/i386-linux-gnu/ld-2.15.so
bf77c000-bff20000 rw-p 00000000 00:00 0          [stack]
Aborted (core dumped)
```

```
for (j = freeBegin; j < freeEnd; j += freeStep)
    free(alloc[j]);

if(argv[3] != NULL && strcmp(argv[3],"df")==0 )
{
    printf("doing double free now\n");
    free(alloc[0]);
}
```

116

第二次释放80字节的块

Program received signal SIGSEGV, Segmentation fault.
0x0804dc8e in _int_malloc (av=0x80521a0, bytes=46) at malloc.c:3876
warning: Source file is more recent than executable.
3876 bck->fd = bin;
(gdb) bt
#0 0x0804dc8e in _int_malloc (av=0x80521a0, bytes=46) at malloc.c:3876
#1 0x0804d734 in calloc (n=1, elem_size=46) at malloc.c:3633
#2 0xb79b3753 in ?? () from /lib/i386-linux-gnu/libglib-2.0.so.0
#3 0xb79b3e6b in g_malloc0 () from /lib/i386-linux-gnu/libglib-2.0.so.0
#4 0xb73a7cae in ?? () from /usr/lib/i386-linux-gnu/libpango-1.0.so.0
…
#15 0x0804987a in append_list (szMsg=0xbfffe17c) at gemalloc.c:27
#16 0x080498e0 in d4d (format=0x804faa2 "free(%p) returned") at gemalloc.c:40
#17 0x08049a94 in button_free_clicked (data=0x0) at gemalloc.c:101

- 释放返回后
- 再分配时非法访问

117

二次释放8字节的块

#0 0x0804e749 in malloc_consolidate (av=0xb6000010) at malloc.c:4415
#1 0x0804dd48 in _int_malloc (av=0xb6000010, bytes=513) at malloc.c:3900
#2 0x0804ea68 in _int_realloc (av=0xb6000010, oldmem=0xb60102b8, bytes=512) at malloc.c:4541
#3 0x0804d2b2 in realloc (oldmem=0xb60102b8, bytes=512) at malloc.c:3489
…
#20 0xb7a8a2cc in g_signal_emit_valist () from /usr/lib/i386-linux-gnu/libgobject-2.0.so.0
#21 0xb7e3dcaa in gtk_signal_emit_by_name () from /usr/lib/i386-linux-gnu/libgtk-x11-2.0.so.0
#22 0xb7e082d9 in ?? () from /usr/lib/i386-linux-gnu/libgtk-x11-2.0.so.0
#23 0xb7e13e71 in ?? () from /usr/lib/i386-linux-gnu/libgtk-x11-2.0.so.0
#24 0xb7e0efa0 in gtk_clist_append () from /usr/lib/i386-linux-gnu/libgtk-x11-2.0.so.0
#25 0x0804987a in append_list (szMsg=0xbfffe17c) at gemalloc.c:27
#26 0x080498e0 in d4d (format=0x804faa2 "free(%p) returned") at gemalloc.c:40
#27 0x08049a94 in button_free_clicked (data=0x0) at gemalloc.c:101

- 释放返回
- 再分配时合并块时死循环

118

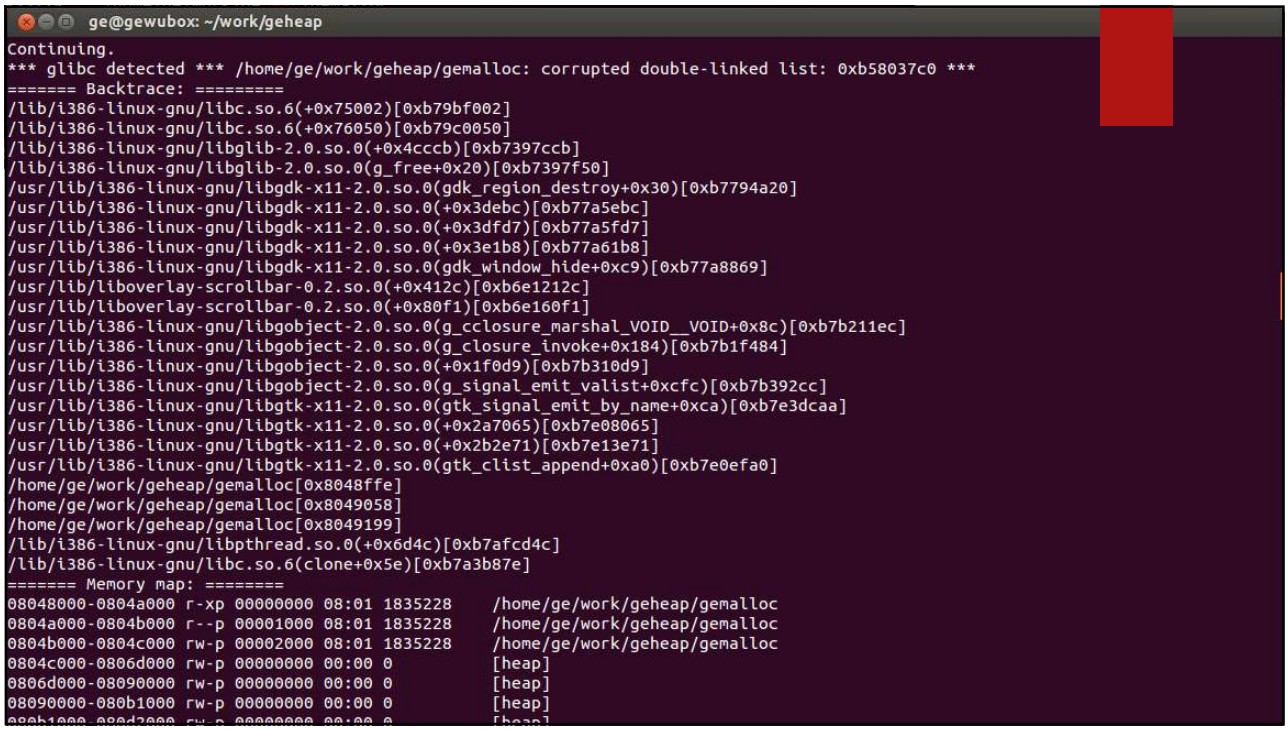
```
Program received signal SIGSEGV, Segmentation fault.
0x0804d049 in free (mem=0xb4cf7008) at malloc.c:3398
3398      if (chunk_is_mmapped(p))                      /* release mmapped memory. */
(gdb) bt
#0  0x0804d049 in free (mem=0xb4cf7008) at malloc.c:3398
#1  0x08049a7f in button_free_clicked (data=0x0) at gemalloc.c:99
```

Double free超大块

119

```
/*
  Debugging:

  Because freed chunks may be overwritten with bookkeeping fields, this
  malloc will often die when freed memory is overwritten by user
  programs.  This can be very effective (albeit in an annoying way)
  in helping track down dangling pointers.

  If you compile with -DMALLOC_DEBUG, a number of assertion checks are
  enabled that will catch more memory errors. You probably won't be
  able to make much sense of the actual assertion errors, but they
  should help you locate incorrectly overwritten memory.  The checking
  is fairly extensive, and will slow down execution
  noticeably. Calling malloc_stats or mallinfo with MALLOC_DEBUG set
  will attempt to check every non-mmapped allocated and free chunk in
  the course of computing the summmaries. (By nature, mmapped regions
  cannot be checked very much automatically.)

  Setting MALLOC_DEBUG may also be helpful if you are trying to modify
  this code. The assertions in the check routines spell out in more
  detail the assumptions and invariants underlying the algorithms.

  Setting MALLOC_DEBUG does NOT provide an automated mechanism for
  checking that all accesses to malloced memory stay within their
  bounds. However, there are several add-ons and adaptations of this
  or other mallocs available that do this.
*/
```

120



121

Malloc Parameter

```
(gdb) ptype mp_
type = struct malloc_par {
    long unsigned int trim_threshold;
    size_t top_pad;
    size_t mmap_threshold;
    size_t arena_test;
    size_t arena_max;
    int n_mmaps;
    int n_mmaps_max;
    int max_n_mmaps;
    int no_dyn_threshold;
    size_t mmapped_mem;
    size_t max_mmapped_mem;
    size_t max_total_mem;
    char *sbrk_base;
}
```

默认为0，可能动态调整大虚拟块的阈值

可配置性的实现
可以通过mallopt函数来设置

int mallopt(int param, int value);

128KB

```
(gdb) p mp_
$2 = {trim_threshold = 131072, top_pad = 131072, mmap_threshold = 131072,
  arena_test = 2, arena_max = 0, n_mmaps = 0, n_mmaps_max = 65536,
  max_n_mmaps = 0, no_dyn_threshold = 0, mmapped_mem = 0, max_mmapped_mem = 0,
  max_total_mem = 0, sbrk_base = 0x804d000 ""}
```

122

Top_pad

调用sbrk扩展top时，故意多申请的数量

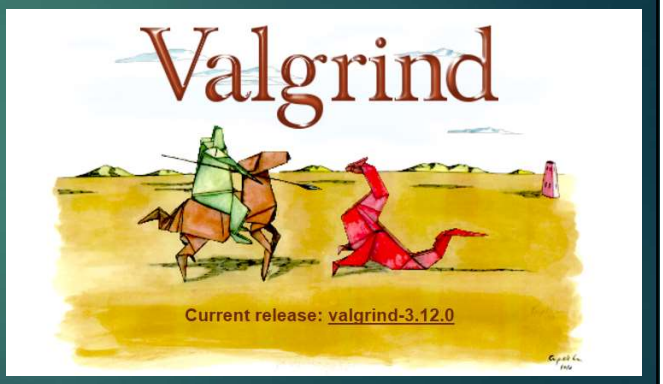
```
/*
  M_TOP_PAD is the amount of extra `padding' space to allocate or retain whenever sbrk is called. It
is used in two ways internally:

  * When sbrk is called to extend the top of the arena to satisfy a new malloc request, this much
padding is added to the sbrk request.

  * When malloc_trim is called automatically from free(),  it is used as the `pad' argument.

  In both cases, the actual amount of padding is rounded so that the end of the arena is always a
system page boundary.

  The main reason for using padding is to avoid calling sbrk so often. Having even a small pad greatly
reduces the likelihood that nearly every malloc request during program start-up (or after trimming)
will invoke sbrk, which needlessly wastes time.

  Automatic rounding-up to page-size units is normally sufficient to avoid measurable overhead, so
the default is 0.  However, in systems where sbrk is relatively slow, it can pay to increase this value, at
the expense of carrying around more memory than  the program needs.
*/
```

123

124

Valgrind

- an instrumentation framework for building dynamic analysis tools
- There are Valgrind tools that can automatically detect many memory management and threading bugs, and profile your programs in detail
- You can also use Valgrind to build new tools
- 安装：sudo apt-get install valgrind



125

基本原理

- A program running under Valgrind is not executed directly by the CPU. Instead it runs on a synthetic CPU provided by Valgrind.
- Your program is then run on a synthetic CPU provided by the Valgrind core. As new code is executed for the first time, the core hands the code to the selected tool. The tool adds its own instrumentation code to this and hands the result back to the core, which coordinates the continued execution of this instrumented code.
- This is why a debugger cannot debug your program when it runs on Valgrind.

126

包一层，二进制插桩

```
int foo ( int x, int y ) { return x + y; }
```

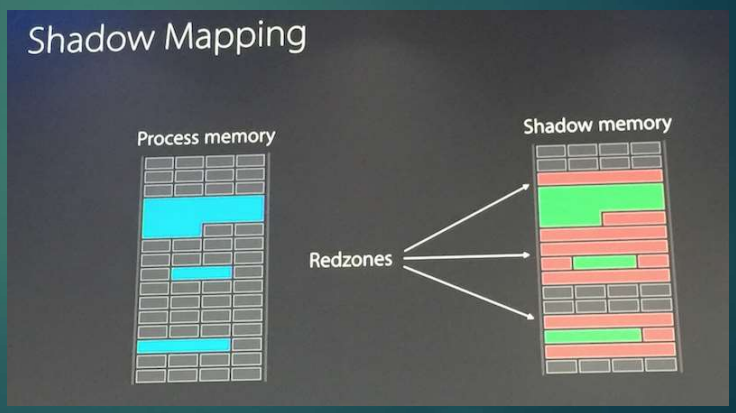
```
#include <stdio.h>
#include "valgrind.h"
int I_WRAP_SONAME_FNNAME_ZU(NONE,foo)( int x, int y )
{
  int  result;
  OrigFn fn;
  VALGRIND_GET_ORIG_FN(fn);
  printf("foo's wrapper: args %d %d\n", x, y);
  CALL_FN_W_WW(result, fn, x,y);
  printf("foo's wrapper: result %d\n", result);
  return result;
}
```

- http://valgrind.org/docs/manual/manual-core-adv.html#manual-core-adv.wrapping

127

Shadow Memory

- 使用影子内存区来存储每个内存块的元数据
- 判断某个内存访问是否合法



128

兼容gdb

- valgrind --vgdb=yes --vgdb-error=0 prog
- gdb prog
- (gdb) target remote | vgdb

129

六大工具

130

Memcheck: a memory error detector

- Accessing memory you shouldn't, e.g. overrunning and underrunning heap blocks, overrunning the top of the stack, and accessing memory after it has been freed.
- Using undefined values, i.e. values that have not been initialised, or that have been derived from other undefined values.
- Incorrect freeing of heap memory, such as double-freeing heap blocks, or mismatched use of malloc/new/new[] versus free/delete/delete[]
- Overlapping src and dst pointers in memcpy and related functions.
- Passing a fishy (presumably negative) value to the size parameter of a memory allocation function.
- Memory leaks.

131

检测泄露

```
==4724== Warning: client switching stacks?  SP change: 0xbe5fa040 --> 0xbed9b270
==4724==          to suppress, use: --max-stackframe=8000048 or greater
==4724==
==4724== HEAP SUMMARY:
==4724==     in use at exit: 2,000 bytes in 2 blocks
==4724==   total heap usage: 2 allocs, 0 frees, 2,000 bytes allocated
==4724==
==4724== 2,000 bytes in 2 blocks are definitely lost in loss record 1 of 1
==4724==    at 0x402BE68: malloc (in /usr/lib/valgrind/vgpreload_memcheck-x86-linux.so)
==4724==    by 0x804872F: main (geheap.c:54)
==4724==
==4724== LEAK SUMMARY:
==4724==    definitely lost: 2,000 bytes in 2 blocks
==4724==    indirectly lost: 0 bytes in 0 blocks
==4724==      possibly lost: 0 bytes in 0 blocks
==4724==    still reachable: 0 bytes in 0 blocks
==4724==         suppressed: 0 bytes in 0 blocks
==4724==
==4724== For counts of detected and suppressed errors, rerun with: -v
==4724== ERROR SUMMARY: 51 errors from 32 contexts (suppressed: 0 from 0)
```

- valgrind --tool=memcheck --leak-check=yes ./geheapd 2 1000 leak

132

泄露报告

```
==3145== 720 bytes in 9 blocks are definitely lost in loss record 2,943 of 3,217
==3145==    at 0x402BE68: malloc (in /usr/lib/valgrind/vgpreload_memcheck-x86-linux.so)
==3145==    by 0x8049343: do_malloc (gemalloc.c:57)
==3145==    by 0x8049434: button_malloc_clicked (gemalloc.c:78)
==3145==    by 0x456C242: g_cclosure_marshal_VOID__VOIDv (in /usr/lib/i386-linux-gnu/libgobject-2.0.so
==3145==    by 0x456A726: ??? (in /usr/lib/i386-linux-gnu/libgobject-2.0.so.0.3200.4)
==3145==    by 0x4583A18: g_signal_emit_valist (in /usr/lib/i386-linux-gnu/libgobject-2.0.so.0.3200.4)
==3145==    by 0x4584442: g_signal_emit (in /usr/lib/i386-linux-gnu/libgobject-2.0.so.0.3200.4)
==3145==    by 0x40BF289: gtk_button_clicked (in /usr/lib/i386-linux-gnu/libgtk-x11-2.0.so.0.2400.10)
==3145==    by 0x40C069F: ??? (in /usr/lib/i386-linux-gnu/libgtk-x11-2.0.so.0.2400.10)
==3145==    by 0x456C1EB: g_cclosure_marshal_VOID__VOID (in /usr/lib/i386-linux-gnu/libgobject-2.0.so.
==3145==    by 0x45692FC: ??? (in /usr/lib/i386-linux-gnu/libgobject-2.0.so.0.3200.4)
```

133

多次释放

```
==4727==
==4727== Invalid free() / delete / delete[] / realloc()
==4727==    at 0x402B06C: free (in /usr/lib/valgrind/vgpreload_memcheck-x86-linux.so)
==4727==    by 0x8048800: main (geheap.c:69)
==4727==  Address 0x41ef028 is 0 bytes inside a block of size 1,000 free'd
==4727==    at 0x402B06C: free (in /usr/lib/valgrind/vgpreload_memcheck-x86-linux.so)
==4727==    by 0x8048800: main (geheap.c:69)
==4727==
==4727==
==4727== More than 10000000 total errors detected.  I'm not reporting any more.
==4727== Final error counts will be inaccurate.  Go fix your program!
==4727== Rerun with --error-limit=no to disable this cutoff.  Note
==4727== that errors may occur in your program without prior warning from
==4727== Valgrind, because errors are no longer being displayed.
```

- valgrind --tool=memcheck --leak-check=yes ./geheapd 2 1000 df

134

```
==3164== Invalid free() / delete / delete[] / realloc()
==3164==    at 0x402B06C: free (in /usr/lib/valgrind/vgpreload_memcheck-x86-linux.so)
==3164==    by 0x80494AE: button_free_clicked (gemalloc.c:107)
==3164==    by 0x456C242: g_cclosure_marshal_VOID__VOIDv (in /usr/lib/i386-linux-gnu/libgobject-2.0.so.0.32
==3164==    by 0x456A726: ??? (in /usr/lib/i386-linux-gnu/libgobject-2.0.so.0.3200.4)
==3164==    by 0x4583A18: g_signal_emit_valist (in /usr/lib/i386-linux-gnu/libgobject-2.0.so.0.3200.4)
==3164==    by 0x4584442: g_signal_emit (in /usr/lib/i386-linux-gnu/libgobject-2.0.so.0.3200.4)
==3164==    by 0x40BF289: gtk_button_clicked (in /usr/lib/i386-linux-gnu/libgtk-x11-2.0.so.0.2400.10)
==3164==    by 0x40C069F: ??? (in /usr/lib/i386-linux-gnu/libgtk-x11-2.0.so.0.2400.10)
==3164==    by 0x456C1EB: g_cclosure_marshal_VOID__VOID (in /usr/lib/i386-linux-gnu/libgobject-2.0.so.0.320
==3164==    by 0x45692FC: ??? (in /usr/lib/i386-linux-gnu/libgobject-2.0.so.0.3200.4)
==3164==  Address 0x6f68e00 is 0 bytes inside a block of size 80 free'd
==3164==    at 0x402B06C: free (in /usr/lib/valgrind/vgpreload_memcheck-x86-linux.so)
==3164==    by 0x80494AE: button_free_clicked (gemalloc.c:107)
==3164==    by 0x456C242: g_cclosure_marshal_VOID__VOIDv (in /usr/lib/i386-linux-gnu/libgobject-2.0.so.0.32
==3164==    by 0x456A726: ??? (in /usr/lib/i386-linux-gnu/libgobject-2.0.so.0.3200.4)
==3164==    by 0x4583A18: g_signal_emit_valist (in /usr/lib/i386-linux-gnu/libgobject-2.0.so.0.3200.4)
==3164==    by 0x4584442: g_signal_emit (in /usr/lib/i386-linux-gnu/libgobject-2.0.so.0.3200.4)
==3164==    by 0x40BF289: gtk_button_clicked (in /usr/lib/i386-linux-gnu/libgtk-x11-2.0.so.0.2400.10)
==3164==    by 0x40C069F: ??? (in /usr/lib/i386-linux-gnu/libgtk-x11-2.0.so.0.2400.10)
==3164==    by 0x456C1EB: g_cclosure_marshal_VOID__VOID (in /usr/lib/i386-linux-gnu/libgobject-2.0.so.0.320
==3164==    by 0x45692FC: ??? (in /usr/lib/i386-linux-gnu/libgobject-2.0.so.0.3200.4)
==3164==
```

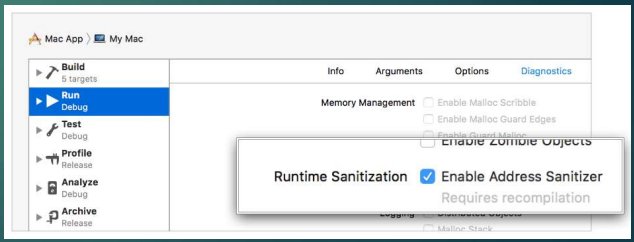
135

概览 Arena 批发 分配过程 释放 Valgrind Asan

136

Address Sanitizer

- 谷歌的几位工程师开发
  - 莫斯科site
  - Konstantin Serebryany, Derek Bruening, Alexander Potapenko, Dmitry Vyukov
- 2011年5月发布，最早用在Chromium open-source browser项目
- Linux kernel 4.0引入，称为ksan
- Clang 3.1 GCC4.8开始支持
- 2015年，苹果的Xcode7开始引入
- https://github.com/google/sanitizers



137

Address/Thread/MemorySanitizer

Slaughtering C++ bugs

Dmitry Vyukov, Google
dvyukov@
Feb 2015 @C++ User Group, Russia

ASan *marketing* slide

- 2x slowdown (Valgrind: 20x and more)
- 1.5x-3x memory overhead
- 3000+ bugs found in Chrome in 3 years
- 3000+ bugs found in Google server software
- 1000+ bugs everywhere else
  - Firefox, FreeType, FFmpeg, WebRTC, libjpeg-turbo, Perl, Vim, LLVM, GCC, MySQL

138

工作原理

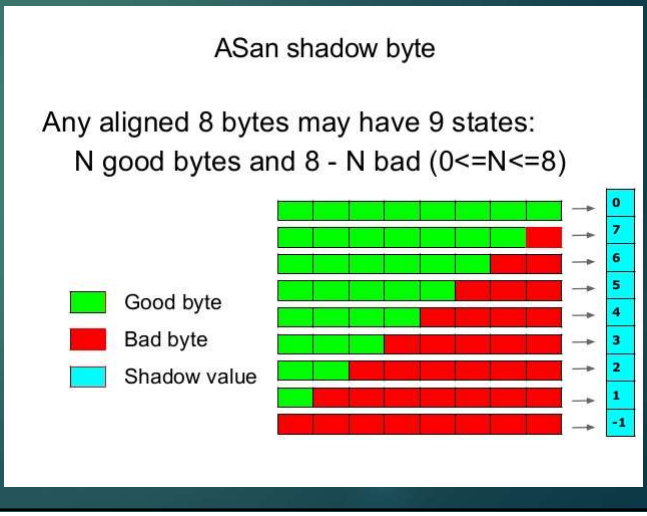
- 编译阶段插桩
  - 支持clang，gcc
  - 对所有load和store插桩监视
  - 73%
- 替换malloc和free的实现

```
ShadowAddr = (Addr >> 3) + Offset;
if (*ShadowAddr != 0)
      ReportAndCrash(Addr);
```

139

Shadow方法

- 影子内存是原始内存的1/8
- 每8个字节对应1个字节
- 比Valgrind更高效



140

访问检查

ASan instrumentation: 8-byte access

```
*a = ...
```

```
char *shadow = a >> 3;
if (*shadow)
  ReportError(a);
*a = ...
```

ASan instrumentation: N-byte access (1, 2, 4)

```
*a = ...
```

```
char *shadow = a >> 3;
if (*shadow &&
    *shadow <= ((a&7)+N-1))
  ReportError(a);
*a = ...
```

141

插桩示例

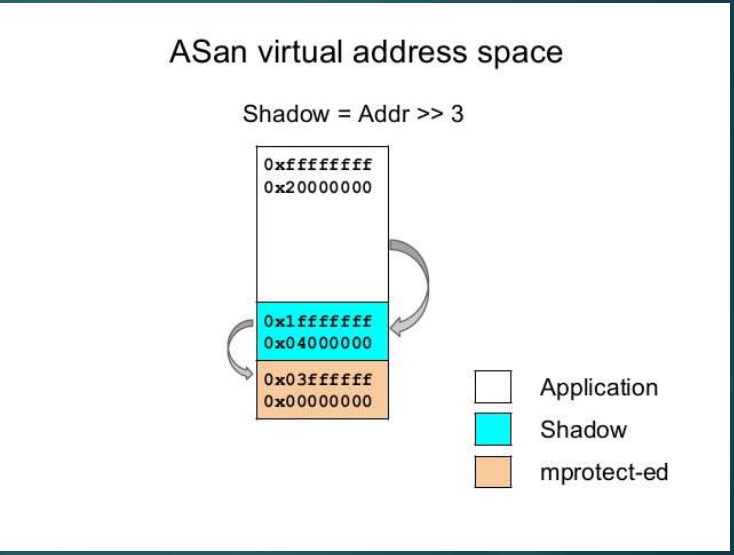
Instrumentation example (x86_64)

```
mov    %rdi,%rax
shr    $0x3,%rax          # shift by 3
cmpb   $0x0,(%rax)        # load shadow
je 1f  <foo+0x1f>
ud2a                      # generate SIGILL*
movq   $0x1234,(%rdi)     # original store

* May use call instead of UD2
```

142

地址空间

- Shadow = Addr >> 3



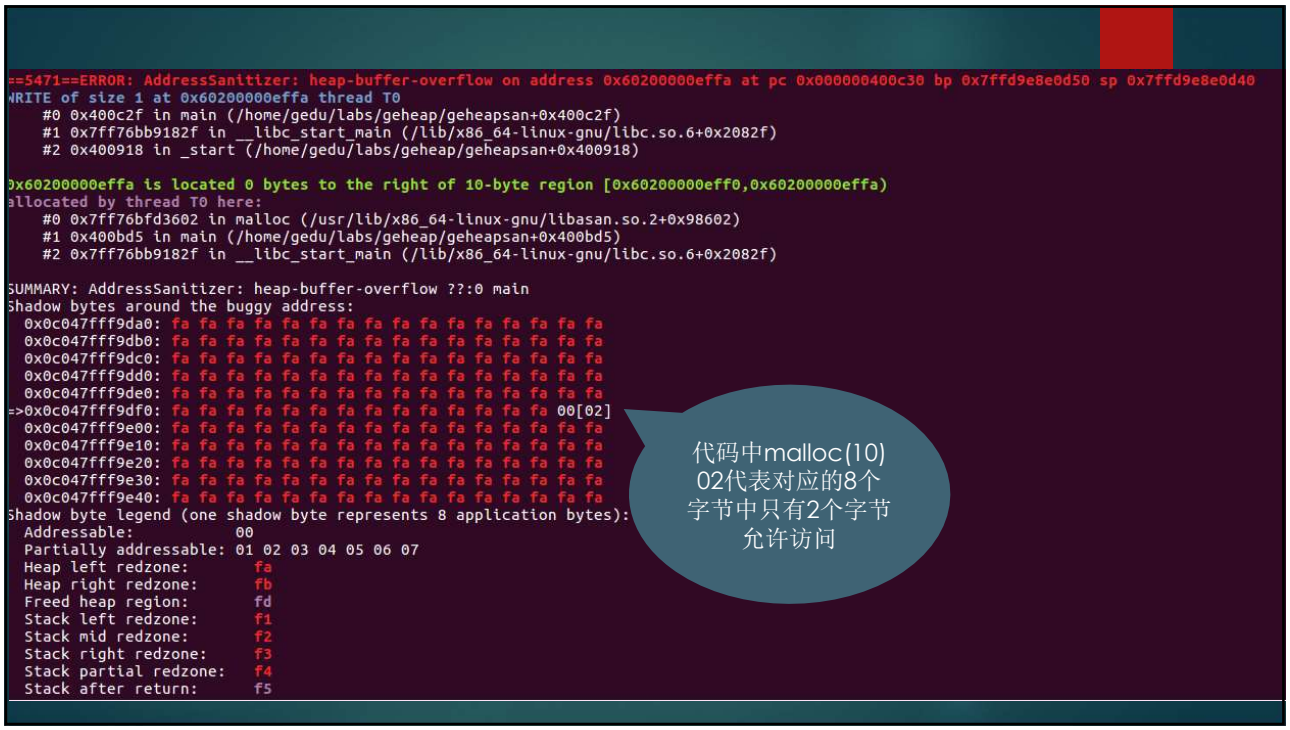
143

在Ubuntu中使用

- GCC 4.8版本开始已经包含asan
- Ubuntu 16.04默认的GCC已经支持
  - GCC 5.4
- gcc -fsanitize=address -lasan geheap.c -o geheapsan

```
gedu@gedu-VirtualBox:~/labs/geheap$ gcc --
version
gcc (Ubuntu 5.4.0-6ubuntu1~16.04.10) 5.4.0 20160609
```

144



==5471==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x60200000effa at pc 0x000000400c30 bp 0x7ffd9e8e0d50 sp 0x7ffd9e8e0d40

WRITE of size 1 at 0x60200000effa thread T0

#0 0x400c2f in main (/home/gedu/labs/geheap/geheapsan+0x400c2f)

#1 0x7ff76bb9182f in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x2082f)

#2 0x400918 in _start (/home/gedu/labs/geheap/geheapsan+0x400918)

0x60200000effa is located **0** bytes to the right of **10-byte** region [0x60200000eff0,0x60200000effa)
allocated by thread T0 here:

#0 0x7ff76bfd3602 in malloc (/usr/lib/x86_64-linux-gnu/libasan.so.2+0x98602)

#1 0x400bd5 in main (/home/gedu/labs/geheap/geheapsan+0x400bd5)

#2 0x7ff76bb9182f in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x2082f)

```
(gdb) x /16b (((long)p)>>3)+0x7fff8000-8
0xc047fff9df6:   0xfa    0xfa    0xfa    0xfa    0xfa    0xfa    0xfa    0xfa
0xc047fff9dfe:   0x00 0x02 0xfa    0xfa    0xfa    0xfa    0xfa    0xfa

(gdb) x /15i 0x0000000000400bff
   0x400bff <main+78>:    mov    rdx,rax
=> 0x400c02 <main+81>:    shr    rdx,0x3
   0x400c06 <main+85>:    add    rdx,0x7fff8000
   0x400c0d <main+92>:    movzx  edx,BYTE PTR [rdx]  ;; 读出影子内存中的编码值
   0x400c10 <main+95>:    test   dl,dl
   0x400c12 <main+97>:    setne  sil
   0x400c16 <main+101>:   mov    rdi,rax
   0x400c19 <main+104>:   and    edi,0x7
   0x400c1c <main+107>:   cmp    dil,dl
   0x400c1f <main+110>:   setge  dl
   0x400c22 <main+113>:   and    edx,esi
   0x400c24 <main+115>:   test   dl,dl
   0x400c26 <main+117>:   je     0x400c30 <main+127>
   0x400c28 <main+119>:   mov    rdi,rax
   0x400c2b <main+122>:   call   0x4008c0 <__asan_report_store1@plt>
   0x400c30 <main+127>:   mov    BYTE PTR [rcx],r8b  ;; 用户代码的写动作
   0x400c33 <main+130>:   add    DWORD PTR [rbp-0xc],0x1  ;; 循环变量做++
   0x400c37 <main+134>:   cmp    DWORD PTR [rbp-0xc],0x13 ;; 比较循环条件

(gdb) p /x $rdx
$2 = 0x6020000eff0
```

147

与编译器的加宽加载冲突

```
struct X { char a, b, c; };
void foo() {
X x; ...
... = x.a + x.c; }
```

- 上面代码，X的成员是三个char型，内存中占3个字节
- 访问x.a和x.c时，编译器为了提高速度会优化为一次加载四个字节，越界访问，使Asan误报越界

148

内存开销

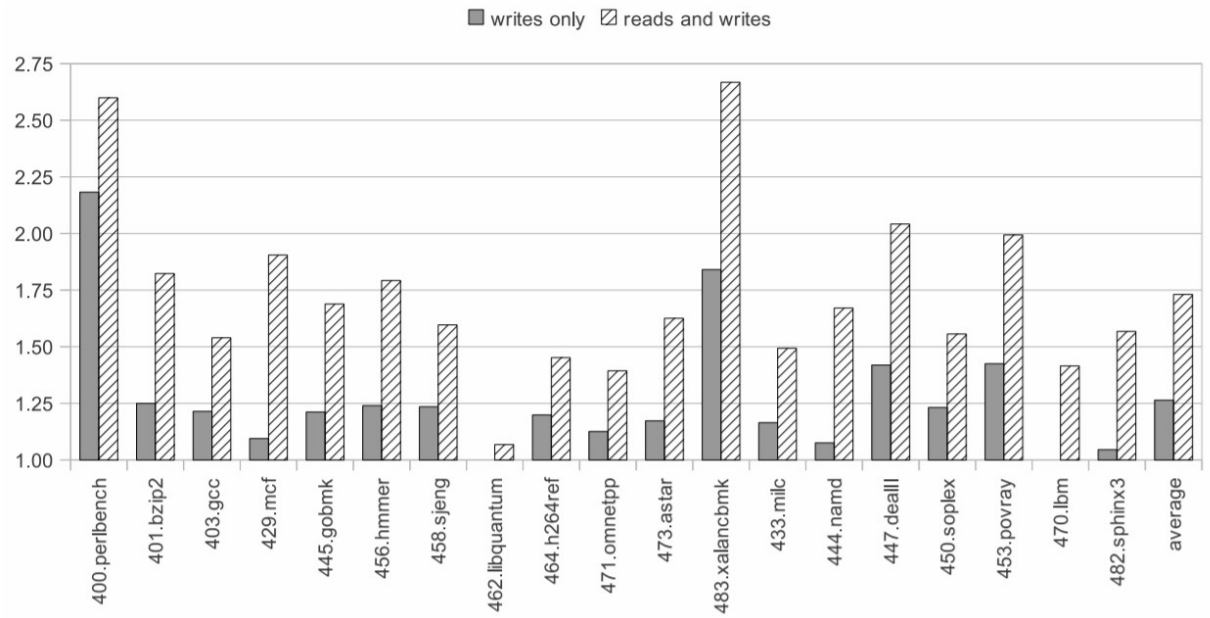
- 来自

Table 1: Memory usage with AddressSanitizer (MB)

Benchmark	Original	Instrumented	Increase
400.perlbench	670	2168	3.64x
401.bzip2	858	1618	2.12x
403.gcc	893	4133	5.21x
429.mcf	1684	2098	1.40x
445.gobmk	37	369	11.22x
456.hmmer	33	582	19.84x
458.sjeng	180	249	1.56x
462.libquantum	104	930	10.06x
464.h264ref	72	439	6.86x
471.omnetpp	181	787	4.89x
473.astar	343	1214	3.98x
483.xalancbmk	434	1688	4.38x
433.milc	694	1618	2.62x
444.namd	58	146	2.83x
447.dealII	807	2602	3.63x
450.soplex	637	2479	4.38x
453.povray	17	371	24.55x
470.lbm	417	550	1.48x
482.sphinx3	52	426	9.22x
total	8171	24467	**3.37**x

149

在64位Linux系统的平均变慢（slowdown）使用SPEC CPU2016测试



https://www.usenix.org/system/files/conference/atc12/atc12-final39.pdf

150

Source-code organization

High-level 'mm' code is processor-independent
(It's in the '/usr/src/linux/mm' subdirectory)

Low-level 'mm' code is processor-specific
(It's in the '/usr/src/linux/arch/' subdirectories)

i386/mm **ppc/mm** **mips/mm** **alpha/mm** . . .

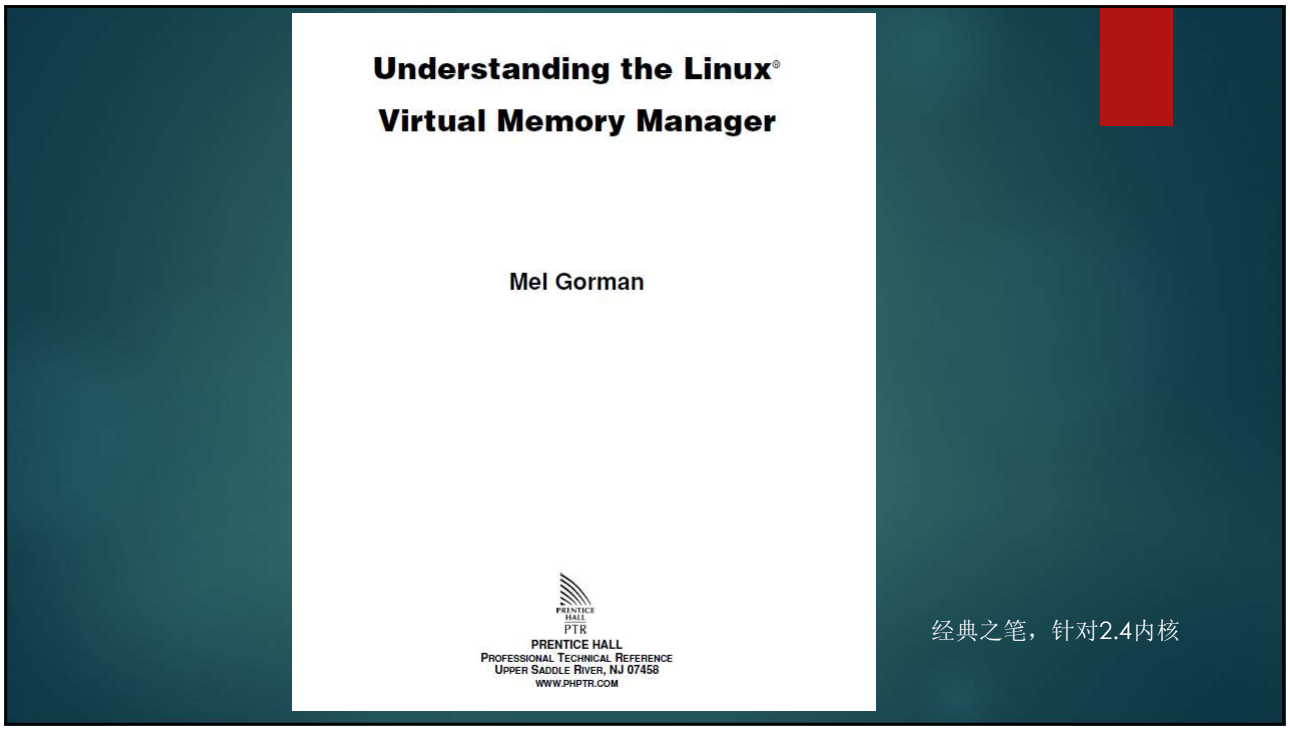
151

文档

d:\bench\linux-4.4.14\Documentation\vm*.*

Name	Ext	Size	Date	Attr
[..]		<DIR>	06/25/2016 01:18	——
.gitignore		20	06/25/2016 01:18	-a--
00-index		1,359	06/25/2016 01:18	-a--
active_mm	txt	3,806	06/25/2016 01:18	-a--
balance		5,325	06/25/2016 01:18	-a--
cleancache	txt	14,111	06/25/2016 01:18	-a--
frontswap	txt	15,384	06/25/2016 01:18	-a--
highmem	txt	5,809	06/25/2016 01:18	-a--
hugetlbpage	txt	17,163	06/25/2016 01:18	-a--
hwpoison	txt	5,975	06/25/2016 01:18	-a--
idle_page_tracking	txt	4,916	06/25/2016 01:18	-a--
ksm	txt	5,540	06/25/2016 01:18	-a--
numa		8,913	06/25/2016 01:18	-a--
numa_memory_policy	txt	23,098	06/25/2016 01:18	-a--
overcommit-accounting		2,573	06/25/2016 01:18	-a--
page_migration		6,601	06/25/2016 01:18	-a--
page_owner	txt	3,701	06/25/2016 01:18	-a--
pagemap	txt	6,970	06/25/2016 01:18	-a--
remap_file_pages	txt	1,554	06/25/2016 01:18	-a--
slub	txt	13,179	06/25/2016 01:18	-a--
soft-dirty	txt	1,782	06/25/2016 01:18	-a--
split_page_table_lock		3,619	06/25/2016 01:18	-a--
transhuge	txt	18,422	06/25/2016 01:18	-a--
unevictable-lru	txt	29,388	06/25/2016 01:18	-a--
userfaultfd	txt	7,114	06/25/2016 01:18	-a--
zsmalloc	txt	3,093	06/25/2016 01:18	-a--
zswap	txt	5,282	06/25/2016 01:18	-a--

152


Understanding the Linux®
Virtual Memory Manager
Mel Gorman
PRENTICE HALL
PTR
PRENTICE HALL
PROFESSIONAL TECHNICAL REFERENCE
UPPER SADDLE RIVER, NJ 07458
WWW.PHPTR.COM
经典之笔，针对2.4内核

153


在调试器里看
glibc堆
张银奎
2017/7/12
卡帕多奇亚（Cappadocia）

154

切问而近思

欢迎关注格友公众号

155

拍案惊奇——软件调试实战训练营

软件调试高级研习班2017庐山秀峰站

HTTP://001001.ORG/GEDU/

156