[堆利用入门]malloc_chunk结构及宏定义

HAPPYers 关注

0.349 2019.04.04 23:53:33 字数 1,463 阅读 2,605

malloc_chunk

先来看一下 `malloc_chunk` 的结构

```
/*
  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;
};
```

一般来说， `size_t` 在 64 位中是 64 位无符号整数，32 位中是 32 位无符号整数。

每个字段的具体的解释如下

- `prev_size`, 如果该 chunk 的物理相邻的前一地址 chunk（两个指针的地址差值为前一 chunk 大小）是空闲的话，那该字段记录的是前一个 chunk 的大小 (包括 chunk 头)。否则，该字段可以用来存储物理相邻的前一个 chunk 的数据。这里的前一 chunk 指的是较低地址的 chunk 。
- `size` ，该 chunk 的大小，大小必须是 2 * SIZE_SZ 的整数倍。如果申请的内存大小不是 2 * SIZE_SZ 的整数倍，会被转换满足大小的最小的 2 * SIZE_SZ 的倍数。32 位系统中，SIZE_SZ 是 4；64 位系统中，SIZE_SZ 是 8。 该字段的低三个比特位对 chunk 的大小没有影响，它们从高到低分别表示
  - `NON_MAIN_ARENA` ，记录当前 chunk 是否不属于主线程，1 表示不属于，0 表示属于。
  - `IS_MAPPED` ，记录当前 chunk 是否是由 mmap 分配的。
  - `PREV_INUSE` ，记录前一个 chunk 块是否被分配。一般来说，堆中第一个被分配的内存块的 size 字段的 P 位都会被设置为 1，以便于防止访问前面的非法内存。当一个 chunk 的 size 的 P 位为 0 时，我们能通过 prev_size 字段来获取上一个 chunk 的大小以及地址。这也方便进行空闲 chunk 之间的合并。
- `fd` ， `bk` 。 chunk 处于分配状态时，从 fd 字段开始是用户的数据。chunk 空闲时，会被添加到对应的空闲管理链表中，其字段的含义如下
  - `fd` 指向下一个（非物理相邻）空闲的 chunk
  - `bk` 指向上一个（非物理相邻）空闲的 chunk
  - 通过 fd 和 bk 可以将空闲的 chunk 块加入到空闲的 chunk 块链表进行统一管理
- `fd_nextsize` ， `bk_nextsize` ，也是只有 chunk 空闲的时候才使用，不过其用于较大的 chunk（large chunk）。
  - **fd_nextsize** 指向前一个与当前 chunk 大小不同的第一个空闲块，不包含 bin 的头指针。
  - **bk_nextsize** 指向后一个与当前 chunk 大小不同的第一个空闲块，不包含 bin 的头指针。
    一般空闲的 large chunk 在 fd 的遍历顺序中，按照由大到小的顺序排列。这样做可以避免在寻找合适 chunk 时挨个遍历。

一个已经分配的 `chunk` 的样子如下。我们称前两个字段称为 `chunk header`，后面的部分称为 `user data`。每次 malloc 申请得到的内存指针，其实指向 `user data` 的起始处。

当一个 chunk 处于使用状态时，它的下一个 chunk 的 `prev_size` 域无效，所以下一个 chunk 的该部分也可以被当前 chunk 使用。这就是 chunk 中的空间复用。

```
chunk-> +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |             Size of previous chunk, if unallocated (P clear)  |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |             Size of chunk, in bytes                     |A|M|P|
  mem-> +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |             User data starts here...                          .
        .                                                               .
        .             (malloc_usable_size() bytes)                      .
next    .                                                               |
chunk-> +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |             (size of chunk, but used for application data)    |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |             Size of next chunk, in bytes                |A|0|1|
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

被释放的 chunk 被记录在链表中（可能是循环双向链表，也可能是单向链表）。具体结构如下

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

```
        .                                                               .
 next   .                                                               |
chunk-> +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
`foot:' |             Size of chunk, in bytes                           |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |             Size of next chunk, in bytes                |A|0|0|
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

一般情况下，物理相邻的两个空闲 chunk 会被合并为一个 chunk 。堆管理器会通过 prev_size 字段以及 size 字段合并两个物理相邻的空闲 chunk 块。

chunk 相关宏

chunk 与 mem 指针头部的转换

- mem 指向用户得到的内存的起始位置。

```
/* conversion from malloc headers to user pointers, and back */
#define chunk2mem(p) ((void *) ((char *) (p) + 2 * SIZE_SZ))
#define mem2chunk(mem) ((mchunkptr)((char *) (mem) -2 * SIZE_SZ))
```

- 最小的 chunk 大小

```
/* The smallest possible chunk */
#define MIN_CHUNK_SIZE (offsetof(struct malloc_chunk, fd_nextsize))
```

这里，offsetof 函数计算出 fd_nextsize 在 malloc_chunk 中的偏移，说明最小的 chunk 至少要包含 bk 指针。

最小申请的堆内存大小

用户最小申请的内存大小必须是 2 * SIZE_SZ 的最小整数倍。

注：就目前而看 MIN_CHUNK_SIZE 和 MINSIZE 大小是一致的，个人认为之所以要添加两个宏是为了方便以后修改 malloc_chunk 时方便一些。

```
/* The smallest size we can malloc is an aligned minimal chunk */
//MALLOC_ALIGN_MASK = 2 * SIZE_SZ -1
#define MINSIZE                                                       \
    (unsigned long) (((MIN_CHUNK_SIZE + MALLOC_ALIGN_MASK) &          \
                      ~MALLOC_ALIGN_MASK))
```

检查分配给用户的内存是否对齐

2 * SIZE_SZ 大小对齐。

```
/* Check if m has acceptable alignment */
// MALLOC_ALIGN_MASK = 2 * SIZE_SZ -1
#define aligned_OK(m) (((unsigned long) (m) & MALLOC_ALIGN_MASK) == 0)

#define misaligned_chunk(p)                                           \
    ((uintptr_t)(MALLOC_ALIGNMENT == 2 * SIZE_SZ ? (p) : chunk2mem(p)) &       \
     MALLOC_ALIGN_MASK)
```

请求字节数判断

```
/*
   Check if a request is so large that it would wrap around zero when
   padded and aligned. To simplify some other code, the bound is made
```

```
   low enough so that adding MINSIZE will also not wrap around zero.
 */

#define REQUEST_OUT_OF_RANGE(req)                                 \
    ((unsigned long) (req) >= (unsigned long) (INTERNAL_SIZE_T)(-2 * MINSIZE))
```

将用户请求内存大小转为实际分配内存大小

```
/* pad request bytes into a usable size -- internal version */
//MALLOC_ALIGN_MASK = 2 * SIZE_SZ -1
#define request2size(req)                                         \
    (((req) + SIZE_SZ + MALLOC_ALIGN_MASK < MINSIZE)              \
         ? MINSIZE                                                \
         : ((req) + SIZE_SZ + MALLOC_ALIGN_MASK) & ~MALLOC_ALIGN_MASK)

/*  Same, except also perform argument check */

#define checked_request2size(req, sz)                             \
    if (REQUEST_OUT_OF_RANGE(req)) {                              \
        __set_errno(ENOMEM);                                      \
        return 0;                                                 \
    }                                                             \
    (sz) = request2size(req);
```

当一个 chunk 处于已分配状态时，它的物理相邻的下一个 chunk 的 prev_size 字段必然是无效的，故而这个字段就可以被当前这个 chunk 使用。这就是 ptmalloc 中 chunk 间的复用。具体流程如下

1. 首先，利用 REQUEST_OUT_OF_RANGE 判断是否可以分配用户请求的字节大小的 chunk。
2. 其次，需要注意的是用户请求的字节是用来存储数据的，即 chunk header 后面的部分。与此同时，由于 chunk 间复用，所以可以使用下一个 chunk 的 prev_size 字段。因此，这里只需要再添加 SIZE_SZ 大小即可以完全存储内容。

3. 由于系统中所允许的申请的 chunk 最小是 MINSIZE，所以与其进行比较。如果不满足最低要求，那么就需要直接分配 MINSIZE 字节。
4. 如果大于的话，因为系统中申请的 chunk 需要 2 * SIZE_SZ 对齐，所以这里需要加上 MALLOC_ALIGN_MASK 以便于对齐。

个人认为，这里在 request2size 的宏的第一行中没有必要加上 MALLOC_ALIGN_MASK。
需要注意的是，通过这样的计算公式得到的 size 最终一定是满足用户需要的。

标记位相关

```
/* 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)->mchunk_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)->mchunk_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 main arena.  */
#define chunk_main_arena(p) (((p)->mchunk_size & NON_MAIN_ARENA) == 0)

/* Mark a chunk as not being on the main arena.  */
#define set_non_main_arena(p) ((p)->mchunk_size |= NON_MAIN_ARENA)

/*
   Bits to mask off when extracting size
   Note: IS_MMAPPED is intentionally not masked off from size field in
   macros for which mmapped chunks should never be seen. This should
```

```
   cause helpful core dumps to occur if it is tried by accident by
   people extending or adapting this malloc.
 */
#define SIZE_BITS (PREV_INUSE | IS_MMAPPED | NON_MAIN_ARENA)
```

获取 chunk size

```
/* Get size, ignoring use bits */
#define chunksize(p) (chunksize_nomask(p) & ~(SIZE_BITS))

/* Like chunksize, but do not mask SIZE_BITS.  */
#define chunksize_nomask(p) ((p)->mchunk_size)
```

获取下一个物理相邻的 chunk

```
/* Ptr to next physical malloc_chunk. */
#define next_chunk(p) ((mchunkptr)(((char *) (p)) + chunksize(p)))
```

获取前一个 chunk 的信息

```
/* Size of the chunk below P.  Only valid if !prev_inuse (P).  */
#define prev_size(p) ((p)->mchunk_prev_size)

/* Set the size of the chunk below P.  Only valid if !prev_inuse (P).  */
#define set_prev_size(p, sz) ((p)->mchunk_prev_size = (sz))

/* Ptr to previous physical malloc_chunk.  Only valid if !prev_inuse (P).  */
#define prev_chunk(p) ((mchunkptr)(((char *) (p)) - prev_size(p)))
```

当前 chunk 使用状态相关操作

```
/* extract p's inuse bit */
#define inuse(p)                                                              \
    ((((mchunkptr)(((char *) (p)) + chunksize(p)))->mchunk_size) & PREV_INUSE)

/* set/clear chunk as being inuse without otherwise disturbing */
#define set_inuse(p)                                                          \
    ((mchunkptr)(((char *) (p)) + chunksize(p)))->mchunk_size |= PREV_INUSE

#define clear_inuse(p)                                                        \
    ((mchunkptr)(((char *) (p)) + chunksize(p)))->mchunk_size &= ~(PREV_INUSE)
```

设置 chunk 的 size 字段

```
/* Set size at head, without disturbing its use bit */
// SIZE_BITS = 7
#define set_head_size(p, s)                                                   \
    ((p)->mchunk_size = (((p)->mchunk_size & SIZE_BITS) | (s)))

/* Set size/use field */
#define set_head(p, s) ((p)->mchunk_size = (s))

/* Set size at footer (only when chunk is not in use) */
#define set_foot(p, s)                                                        \
    (((mchunkptr)((char *) (p) + (s)))->mchunk_prev_size = (s))
```

获取指定偏移的 chunk

```
/* Treat space at ptr + offset as a chunk */
#define chunk_at_offset(p, s) ((mchunkptr)(((char *) (p)) + (s)))
```

指定偏移处 chunk 使用状态相关操作

```
/* check/set/clear inuse bits in known places */
#define inuse_bit_at_offset(p, s)                                       \
    (((mchunkptr)(((char *) (p)) + (s)))->mchunk_size & PREV_INUSE)

#define set_inuse_bit_at_offset(p, s)                                   \
    (((mchunkptr)(((char *) (p)) + (s)))->mchunk_size |= PREV_INUSE)

#define clear_inuse_bit_at_offset(p, s)                                 \
    (((mchunkptr)(((char *) (p)) + (s)))->mchunk_size &= ~(PREV_INUSE))
```