Web服务器系统内存分配和释放管理

操作系统课程设计2023

鲁强

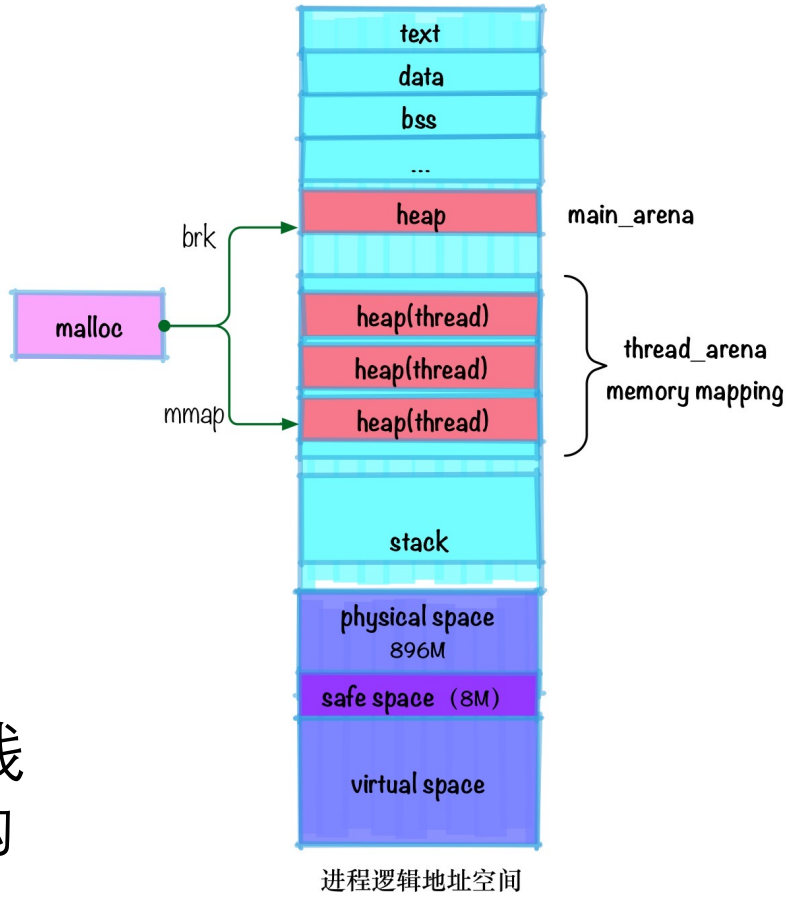
一、Linux用户库的内存管理方法

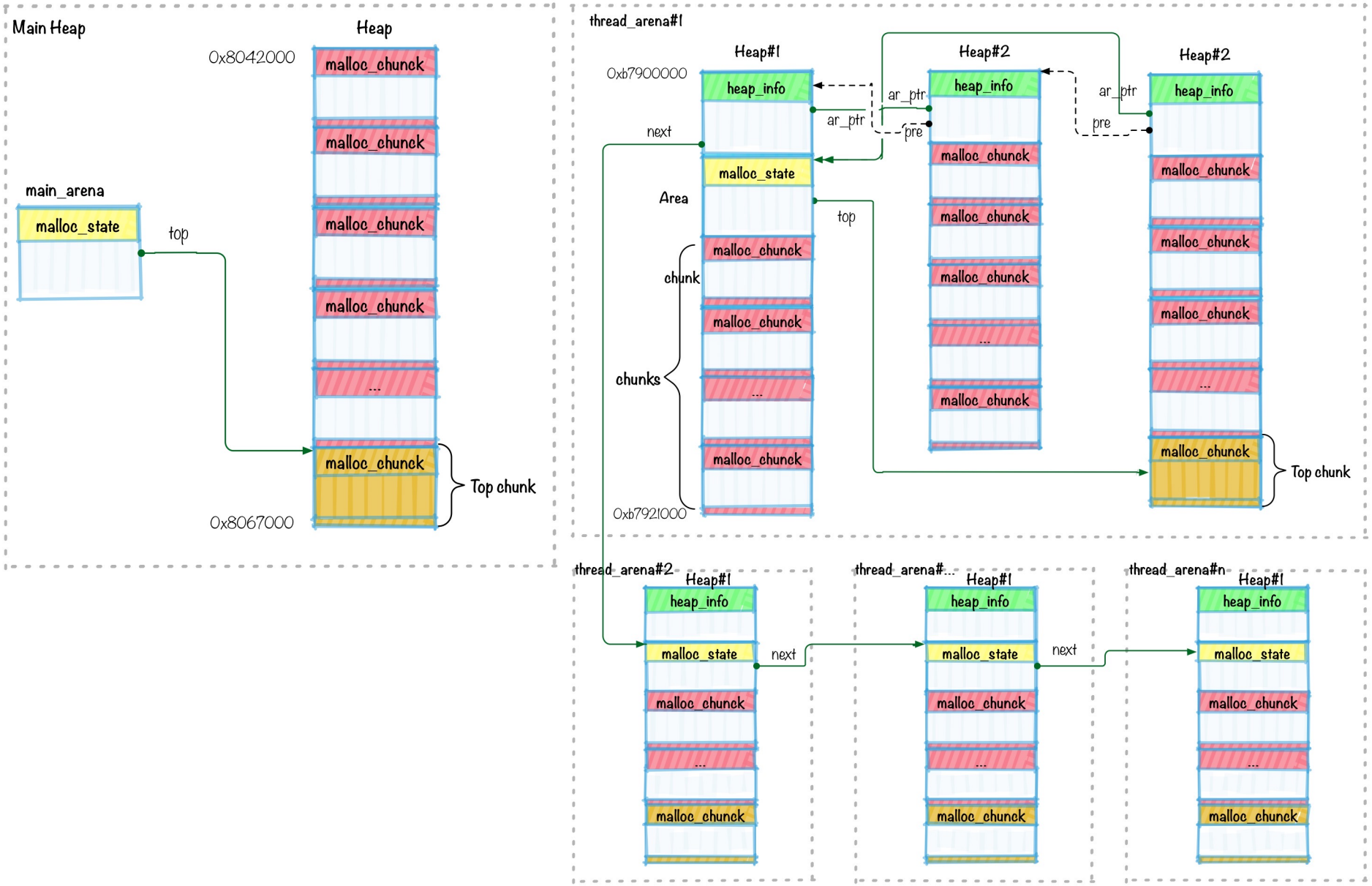
- 用户程序通过malloc和free等函数来分配和释放内存。
- 目前存在多个版本的C语言运行库，其中比较著名的有ptmalloc2，jemalloc，Hoard malloc和Thread-caching malloc（tcmalloc）等。
- 在linux中malloc和free存在于glibc库中，默认采用的是Wolfram Gloger和Doug Lea编写的ptmalloc2版本。

一、Linux用户库的内存管理方法

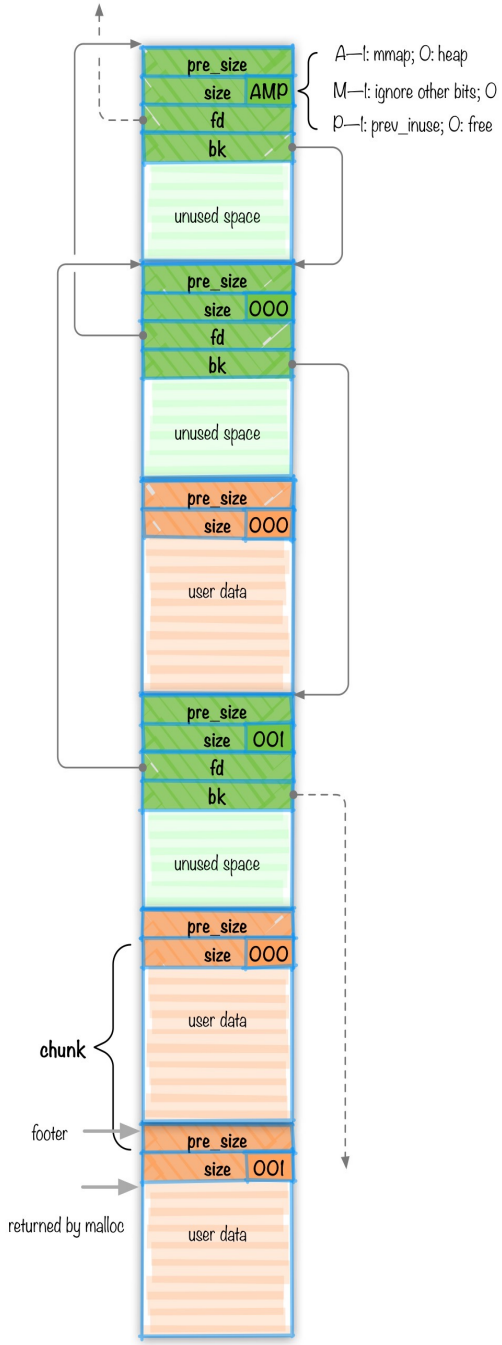
- 一个是通过brk系统调用来移动逻辑heap段的指针来分配或缩减地址空间；
- 另一个是通过系统调用mmap，在内存映射段开辟新的逻辑地址空间。

Arena包括两种形式，一种为main_arena，主要与用户进程空间中的heap段相对应；另一种为thread_aren，主要为每个或多个线程来管理其所需的内存。main_arena的结构形与thread_arena结构稍有不同，





Malloc 内存组织结构图



链表组织的问题？

为提高寻找合适chunk的速度，可以按照chunk的大小，对所有的空闲chunk进行组织。

根据空闲chunk大小，将其放入不同桶中

- **Fastbin**

用于存放空闲的“小”chunk（大小在16~80字节）

- Bin

unsorted bins、small bins和large bins三种区域，共占用了126个桶。其中unsorted bin只有一项，在bins[1];small bins共有62项，在bins[2]~bins[63]；large bins共有63项，在bins[64]~bins[126]。

- **Top** chunk

当申请内存大于Fastbin和Bin桶中的内存区域，在此部分分配

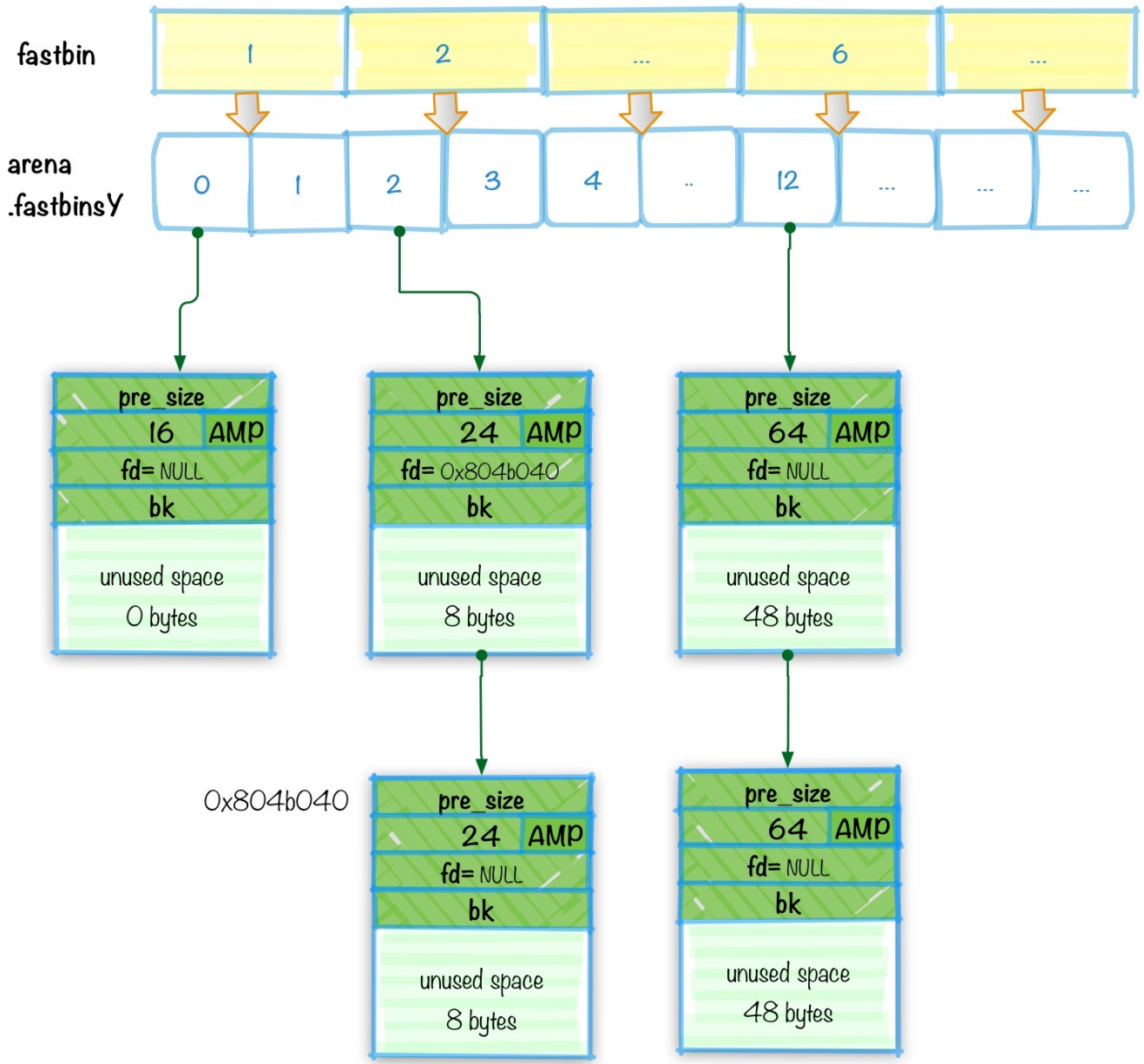
- Last Remainder Chunk

在分配一个small chunk时，如果在small bins中找不到合适的chunk，并且last remainder chunk满足要求，则将last remainder chunk分裂为两部分，一个用于用户请求的内存；剩余的将变为新的last remainder chunk。

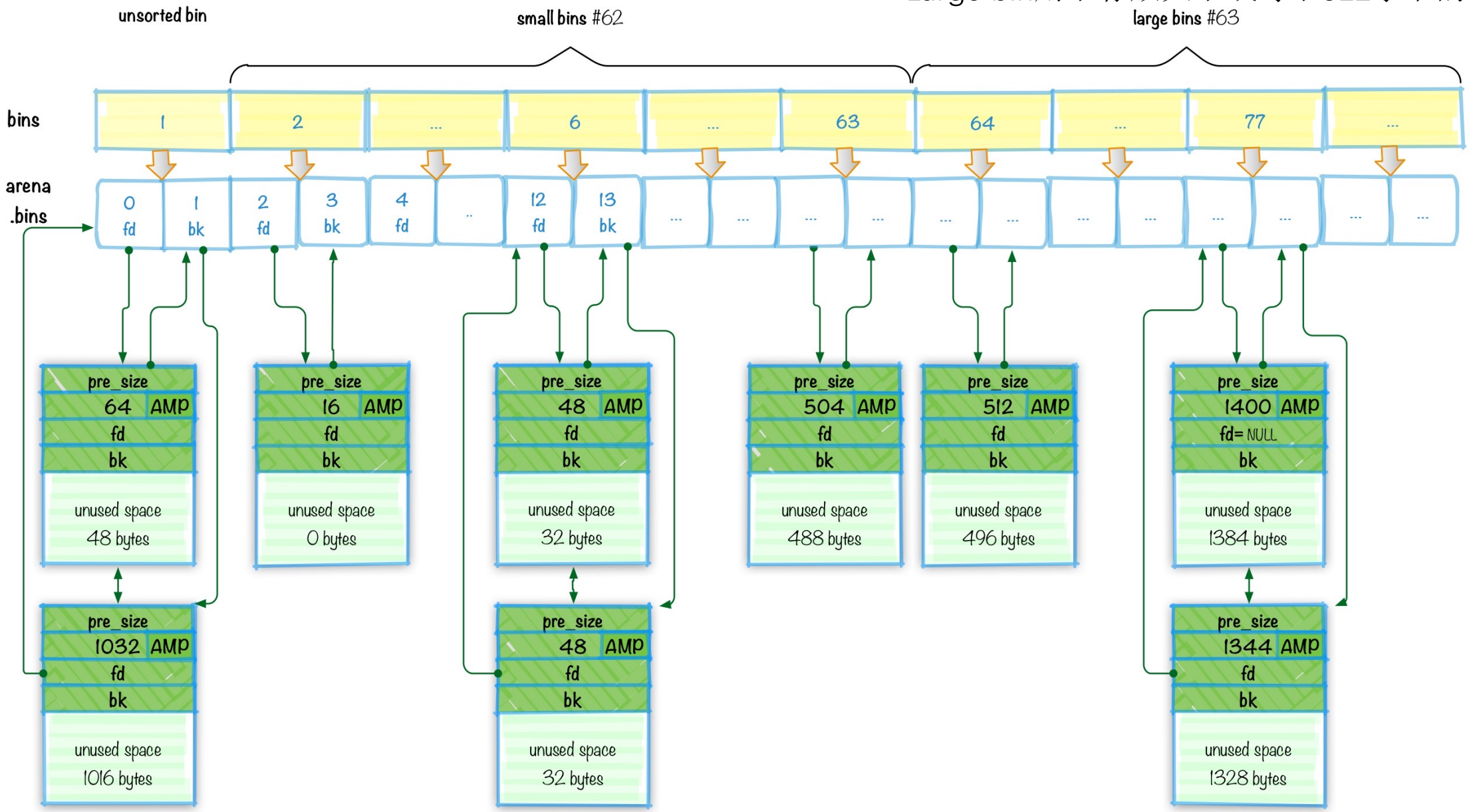
此项值能够提高malloc连续分配小chunk的速度。

- fastbinsY用于存放空闲的“小”chunk（size大小在16~80字节之间），用于小内存的分配和释放管理。在fastbin中，每个桶存放的chunk大小相差为8个字节

- 在分配内存和回收chunk时采用后进先出（LIFO）策略，即在每个桶的链表首部删除或者增加chunk。需要注意的是，为提高速度，在fastbin中不对地址相邻的空闲chunk进行合并。这虽然会加快外部碎片产生速度，但能够提高内存分配速度。



- Unsorted bin用于存放回收的chunk
- small bins主要用于小内存的分配
- Large bin用于存放大于或等于512字节的chunk。



- 分配算法

在进程初始状态，heap的大小为0。如果第一次用户请求的内存大小小于mmap分配阈值时，malloc会申请128kb+sizeof(malloc_chunk)的内存给heap；如果大于mmap分配阈值，则直接使用mmap分配内存。

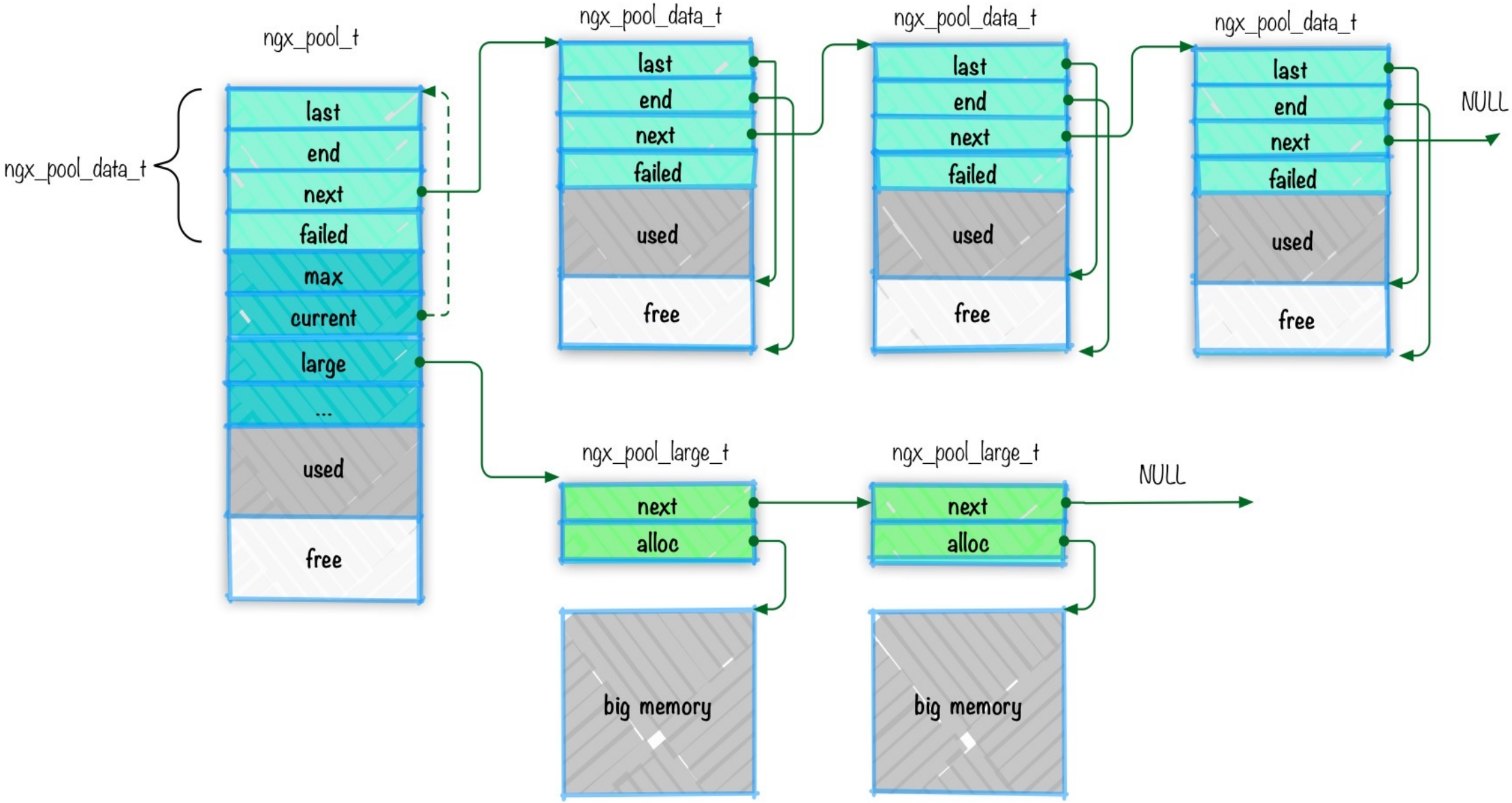
- **如果用户请求的内存大小小于mmap阈值**，malloc首先在fastbin中进行查找，如果找到将找到的chunk返回给用户；
- **如果找不到合适的chunk**，则在small bins中进行查找。如找到则将chunk返回给用户；
- **若还找不到**，则将fast bins所有的chunk加入到unsorted bin中，并进行chunk合并；
- **然后在unsorted bin中进行查找**。如果找到合适的chunk，则返回给用户；如果在unsorted bin中没有找到合适chunk，则将unsorted bin中的所有chunk加入到small bins或large bins中，并进行chunk合并；
- **然后在large bins中进行查找**。如果找到，则分裂此chunk为两部分，一部分返回给用户；另一部分加入到unsorted bin中；
- **如果在large bins中也未找到**，malloc则会查看top chunk。如果top chunk满足用户要求，则将top chunk分裂为两部分，一部分返回给用户；另一部分成为新的top chunk；
- **如果top chunk大小也不满足要求**，则根据mmap阈值和用户请求内存大小，来决定采用mmap增加内存映射区以在对应arena中生成新的heap；还是应用sbrk进行heap内存扩展以增加arena中top chunk的大小。

- **回收算法**

用户通过free函数所释放的内存，在一般情况下并没有被操作系统回收，而是被重新打包成chunk，并放入到malloc相应的内存结构中，以供重用。但是如果释放的内存紧邻top chunk，使得它们合并起来足够大时，将通过munmap系统调用将这些内存返回给操作系统。具体释放过程如下：

- 如果回收的chunk足够小，则将它放入合适的fastbin中；
- 如果回收的chunk是经过mmap得到的大数据块，则将它通过munmap释放给操作系统；
- 查看此chunk是否有相邻的处于空闲的chunk，如果有则合并它们成为新的chunk；
- 如果此chunk为top chunk则根据top chunk的大小来决定是否将内存释放给操作系统；如果此chunk不是top chunk，则将它放入unsorted bin中；
- 如果此chunk足够大，则合并fastbin中所有的chunk；然后看top chunk是否有足够大的空间，以给操作系统返回一些内存。由于性能原因，此步可能会被推迟到malloc函数或其它函数调用时才完成。

二. Nginx


ngx_pool_t
ngx_pool_data_t
last
end
next
failed
max
current
large
...
used
free
ngx_pool_data_t
last
end
next
failed
used
free
ngx_pool_data_t
last
end
next
failed
used
free
ngx_pool_data_t
last
end
next
failed
used
free
NULL
ngx_pool_large_t
next
alloc
big memory
ngx_pool_large_t
next
alloc
big memory
NULL

三、实验7 Web服务器的内存管理

题目1. 查询tcmalloc相关材料，写出其组织和管理内存的结构，并说明其为什么在多线程环境下管理内存（分配和释放内存）的效率比ptmalloc高？

题目2. 根据上面对Nginx中内存池的描述，实现与此描述类似的内存池（可以参考Nginx相关源代码）。注意此内存池要支持内存对齐和多线程。

题目3. 设计测试代码，在不同内存申请和释放情况下，对比malloc/free和内存池的内存申请和释放效率。（比如，连续分配300000个小内存，两种方法所需要的时间；连续分配和释放300000个不同大小的内存，malloc/free函数所消耗时间，以及内存池完成连续分配300000个不同大小的内存和一次释放这些内存所消耗的时间；…）

题目4. 通过指定数量的线程或任务共享一个内存池的方式，来修改前面Web服务器中申请和释放内存的代码。每个任务都从指定的内存池申请内存。当使用一个内存池的任务全部完成后，才释放这个内存池。例如，指定k个任务共享一个内存池，当服务器中存在10k个任务时，服务器就会创建10个内存池。当内存池中k个任务完成后，这个内存池就是被释放。

题目5.根据前面介绍的linux内核内存模型和ptmalloc2用户内存管理模型，设计合理的内存管理结构，以支持实验7中缓存Web页面的hash结构对内存的使用和释放。在设计此内存管理结构时，需要考虑性能问题和碎片问题，以及两种之间的折中和权衡。

四、考核及实验报告撰写要求

- 1. 每人独立完成每个题目
- 2. 内容包含完成每个题目的思考、设计方案、源代码（带注释）、实验实现过程（附相关抓图）说明、实验运行结果展示及相应的分析

  ---只提交代码，没有分数！！！

  ---提交实验报告雷同、成绩为零
- 4. 最终考核包括答辩成绩和实验报告成绩

  答辩 40% + 报告60%