

阅读目录

1. 结论
2. 具体内容
3. 内存分配的原理
4. 具体分配过程
 - 情况一：malloc 小于 128K 的内存，使用 brk 分配
 - 情况二：malloc 大于 128K 的内存，使用 mmap 分配 (munmap 释放)

摘要：偶尔看到面试题会问到 malloc 的底层原理，今天就来记录一下，毕竟学习要“知其所以然”，这样才会胸有成竹。

注：下面分析均是基于 linux 环境下的 malloc 实现。步骤是：先总结结论，再逐步展开

回到顶部

结论

- 1) 当开辟的空间小于 128K 时，调用 brk () 函数，malloc 的底层实现是系统调用函数 brk ()，其主要移动指针 _enddata(此时的 _enddata 指的是 Linux 地址空间中堆段的末尾地址，不是数据段的末尾地址)
- 2) 当开辟的空间大于 128K 时，mmap () 系统调用函数来在虚拟地址空间中（堆和栈中间，称为“文件映射区域”的地方）找一块空间来开辟。

回到顶部

具体内容

当一个进程发生缺页中断的时候，进程会陷入核心态，执行以下操作：

- 1) 检查要访问的虚拟地址是否合法
- 2) 查找/分配一个物理页
- 3) 填充物理页内容（读取磁盘，或者直接置0，或者什么都不做）
- 4) 建立映射关系（虚拟地址到物理地址的映射关系）
- 5) 重复执行发生缺页中断的那条指令

如果第3步，需要读取磁盘，那么这次缺页就是 majfit(major fault: 大错误),否则就是 minflt(minor fault: 小错误)

回到顶部

内存分配的原理

从操作系统角度看，进程分配内存有两种方式，分别由两个系统调用完成：brk 和 mmap（不考虑共享内存）

- 1) brk 是将数据段 (.data) 的最高地址指针 _edata 往高地址推
- 2) mmap 是在进程的虚拟地址空间中（堆和栈中间，称为“文件映射区域”的地方）找一块空闲的虚拟内存。

这两种方式分配的都是虚拟内存，没有分配物理内存。在第一次访问已分配的虚拟地址空间的时候，发生缺页中断，操作系统负责分配物理内存，然后建立虚拟内存和物理内存之间的映射关系。

回到顶部

具体分配过程

情况一：malloc 小于 128K 的内存，使用 brk 分配

将 _edata往高地址推(只分配虚拟空间，不对应物理内存(因此没有初始化)，第一次读/写数据时，引起内核缺页中断，内核才分配对应的物理内存，然后虚拟地址空间建立映射关系)，如下图：

<		202
日	一	二
30	31	1
6	7	8
13	14	15
20	21	22
27	28	29
4	5	6

导航

博客园
首页
新随笔
联系
管理

统计

随笔 - 144
文章 - 0
评论 - 3
引用 - 0

公告

昵称： 爱笑的张飞
园龄： 2年5个月
粉丝： 6
关注： 6
+加关注

搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

随笔分类

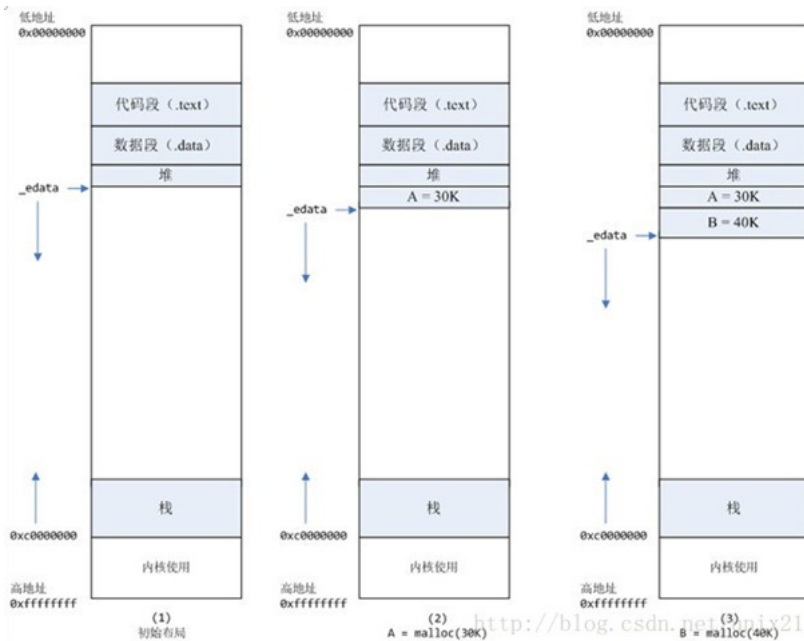
c/c++ (33)
database(13)
DesignPatten(1)
Errors(15)
Golang(12)
Internet(1)
interviews(11)
leetcode(10)
Nginx(5)
OAuth2.0(1)
python(18)
Ubuntu(16)
Virtualization(6)

随笔档案

2020年9月(3)
2020年8月(1)
2020年7月(3)
2020年6月(9)
2020年5月(1)
2020年4月(3)
2020年3月(4)
2020年2月(2)
2019年12月(3)
2019年10月(6)
2019年9月(10)
2019年8月(10)
2019年7月(4)
2019年6月(2)
2019年5月(7)
2019年4月(16)
2019年3月(15)
2019年2月(8)
2019年1月(17)
2018年12月(4)
2018年10月(1)
2018年9月(1)
2018年8月(4)
2018年4月(10)

最新评论

1. Re: malloc 底层实



1, 进程启动的时候, 其(虚拟)内存空间的初始布局如图1所示

2, 进程调用`A=malloc(30K)`以后, 内存空间如图2:

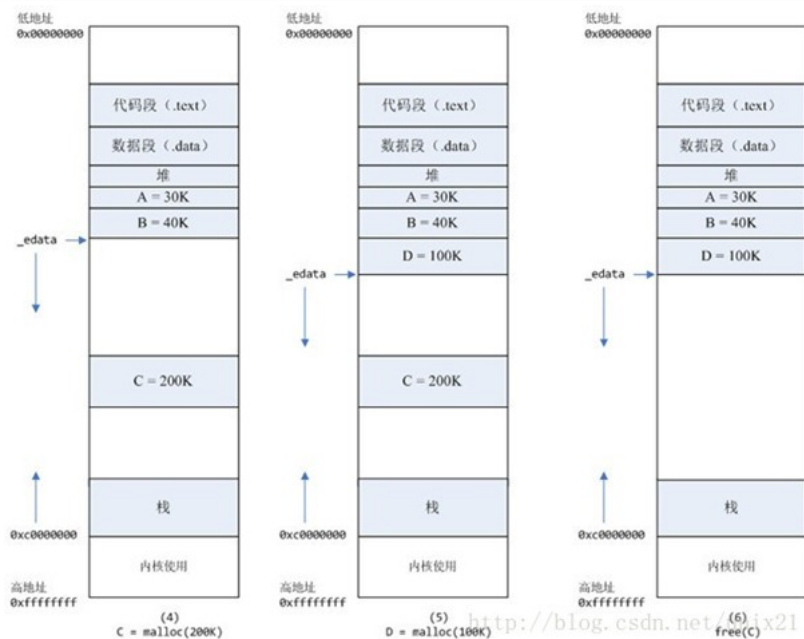
`malloc`函数会调用`brk`系统调用, 将`_edata`指针往高地址推30K, 就完成虚拟内存分配

你可能会问: 难道这样就完成内存分配了?

事实是: `_edata+30K`只是完成虚拟地址的分配, A这块内存现在还是没有物理页与之对应的, 等到进程第一次读写A这块内存的时候, 发生缺页中断, 这个时候, 内核才分配A这块内存对应的物理页。也就是说, 如果用`malloc`分配了A这块内容, 然后从来未访问它, 那么, A对应的物理页是不会被分配的。

3, 进程调用`B=malloc(40K)`以后, 内存空间如图3

情况二: `malloc` 大于 128K 的内存, 使用 `mmap` 分配 (mmap 释放)



4, 进程调用`C=malloc(200K)`以后, 内存空间如图4

默认情况下, `malloc`函数分配内存, 如果请求内存大于128K (可由`M_MMAP_THRESHOLD`选项调节), 那就不是去推`_edata`指针了, 而是利用`mmap`系统调用, 从堆和栈的中间分配一块虚拟内存

这样做主要是因为:

`brk`分配的内存需要等到高地址内存释放以后才能释放(例如, 在B释放之前, A是不可能释放的, 因为只有一个`_edata`指针, 这就是内存碎片产生的原因, 什么时候紧缩看下面), 而`mmap`分配的内存可以单独释放。

当然, 还有其它的好处, 也有坏处, 再具体下去, 有兴趣的同学可以去看glibc里面`malloc`的代码了。

5, 进程调用`D=malloc(100K)`以后, 内存空间如图5

6, 进程调用`free(C)`以后, C对应的虚拟内存和物理内存一起释放

“在B释放之前, A是一个`_edata`指针, 因此。”这句话和下面的理内存都没有释放, 指针, 如果往回推, 那:

2. Re:dockerfile---Unable to locate p @ xh龙渊哈哈, 一起

3. Re:dockerfile---Unable to locate p 最近也在学阳哥的do

阅读排行榜

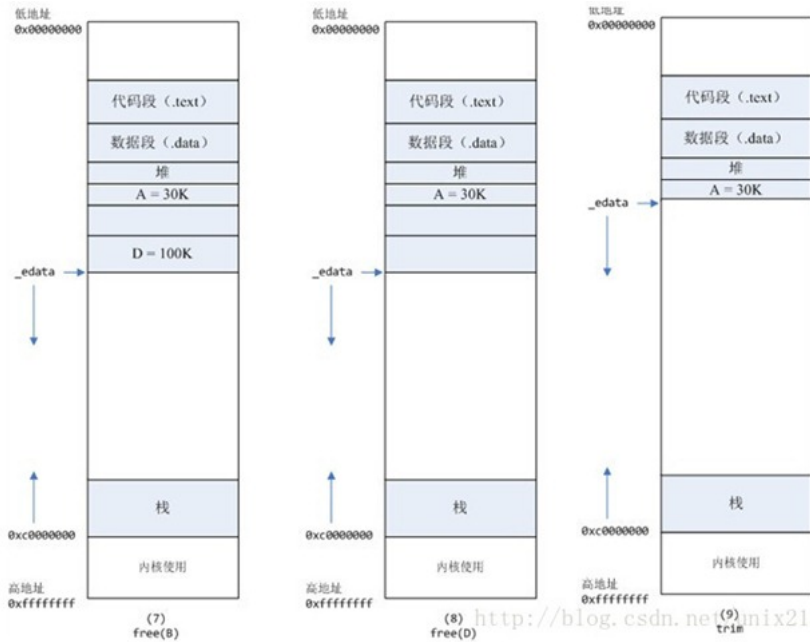
1. C++---类成员变
2. malloc 底层实现
3. error LNK2019 lass std::basic_os d::char_traits<ch
4. docker---Docke
5. c++ 中的数字和

评论排行榜

1. dockerfile---apt ble to locate packa
2. malloc 底层实现

推荐排行榜

1. malloc 底层实现
2. 如何在 main() 括 (4)
3. Error:stray '\24
4. dockerfile---apt ble to locate packa
5. one_day_one_li



- 7，进程调用free(B)以后，如图7所示
- B对应的虚拟内存和物理内存都没有释放，因为只有一个_edata指针，如果往回推，那么D这块内存怎么办呢？当然，B这块内存，是可以重用的，如果这个时候再来一个40K的请求，那么malloc很可能就把B这块内存返回回去了**
- 8，进程调用free(D)以后，如图8所示
- B和D连接起来，变成一块140K的空闲内存**
- 9，默认情况下：
- 当最高地址空间的空闲内存超过128K（可由M_TRIM_THRESHOLD选项调节）时，执行内存紧缩操作（trim）。在上一个步骤free的时候，发现最高地址空闲内存超过128K，于是内存紧缩，变成图9所示**

参考博客：
<https://www.cnblogs.com/dongzhiquan/p/5621906.html>

所有博文均为原著，如若转载，请注明出处！

分类: [c/c++](#)
标签: [malloc](#)

好文要顶

关注我

收藏该文

爱笑的张飞
关注 - 6
粉丝 - 6

4 0

« 上一篇: [C++---拷贝构造函数和赋值构造函数](#)
» 下一篇: [MySQL---MVCC机制](#)

posted on 2019-05-04 16:57 爱笑的张飞 阅读(4607) 评论(1) 编辑 收藏

评论

1楼 2020-08-17 20:08 打不死的黄妖精

“在B释放之前，A是不可能释放的，因为只有一个_edata 指针，这就是内存碎片产生的原因。”这句话和下面的“B对应的虚拟内存和物理内存都没有释放，因为只有一个_edata指针，如果往回推，那么D这块内存怎么办呢？”感觉有些矛盾啊，既然A不能先于B释放，那么B为何又能先于D释放呢？

支持(0) 反对(0)

[刷新评论](#) [刷新页面](#) [返回顶部](#)

注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问](#) 网站首页。

- 【推荐】超50万行VC++源码：大型组态工控、电力仿真CAD与GIS源码库
- 【推荐】7天蜕变！阿里云专家免费授课，名额有限！
- 【推荐】828企业上云节，亿元上云补贴，华为云更懂企业
- 【推荐】未知数的距离，毫秒间的传递，声网与你实时互动
- 【推荐】了不起的开发者，挡不住的华为，园子里的品牌专区
- 【推荐】深度回顾！30篇好文，解析历年双十一背后的阿里技术秘籍

相关博文:

- malloc
 - malloc, realloc和calloc的区别
 - vma
 - malloc/free与new/delete
 - [C]链接和生存周期
- » 更多推荐...

【推荐】电子签名认准大家签，上海CA权威认证

最新 IT 新闻:

- 为什么大家都讨厌「VIP 通货膨胀」？
 - 苹果若真的封杀Epic，就是和整个游戏界作对？
 - 在豆瓣上有4万人 每天都假装活在1980-2000年
 - 不含算法的TikTok值得买吗？专家：买家难以复刻其魔力
 - 理想汽车回应再次“断轴”：以51km/h时速撞到水泥花坛 已澄清
- » 更多新闻...

历史上的今天:

2019-05-04 C++---拷贝构造函数和赋值构造函数

Powered by:

博客园

Copyright © 2020 爱笑的张飞

Powered by .NET Core on Kubernetes