



凌风探梅的专栏

图像处理，图像分割，特征提取，机器学习，模式识别，深度学习等

个人资料



凌风探梅

原创	粉丝	喜欢	评论
102	1075	237	284



等级： 博客 7

访问量： 210万+

积分： 2万+

排名： 358

Linux下glibc内存管理

2015年02月05日 17:52:28

分类： glibc (3)

版权声明：转载注明出处 http://blog.csdn.net/Real_Myth,如果有未完成的请
https://blog.csdn.net/Real_Myth/article/details/43531065
目录(?)

整理的参考文献，记不清了

1 背景简介

出现疑似“内存泄露”问题： malloc申请的内存， free释放以后没有归还操作系统，比如内存模块占用的内存为10GB，释放内存以后，通过TOP命令或者/proc/pid/status查看占用的内存有时仍然为10G，有时为5G，有时为3G, etc，内存释放的行为不确定。

2 malloc()/free(), mmap(), brk(), 用户程序-->glibc -->linux kernel之间的关系



0

目录视图

摘要视图

RSS 订阅



1333人阅读

评论(0)

收藏

举报



及时留言告知

联系方式

个人邮箱: xuxiduo@zju.edu.cn

QQ群:

1) OpenCV俱乐部

群1: 186168905 已满

群2: 421100161 可加

2) 视频/音频/图像/算法/ML

群1: 148111910 可加

群2: 157103105 可加

备注: 加群需要回答问题, 避免广告党。

如果你是博客看到后加的, 请注明“博客”并回答问题, 只注明“博客”不回答问题的恕不加入。答案为和群相关的任何技术名词, 不能出现1) 和2) 中的任何字眼

文章搜索

博客专栏



OpenCV专栏

文章: 32篇

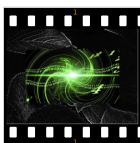
阅读: 73123



DeepID专栏

文章: 8篇

阅读: 12347



数字图像处理专题

文章: 16篇

阅读: 21846

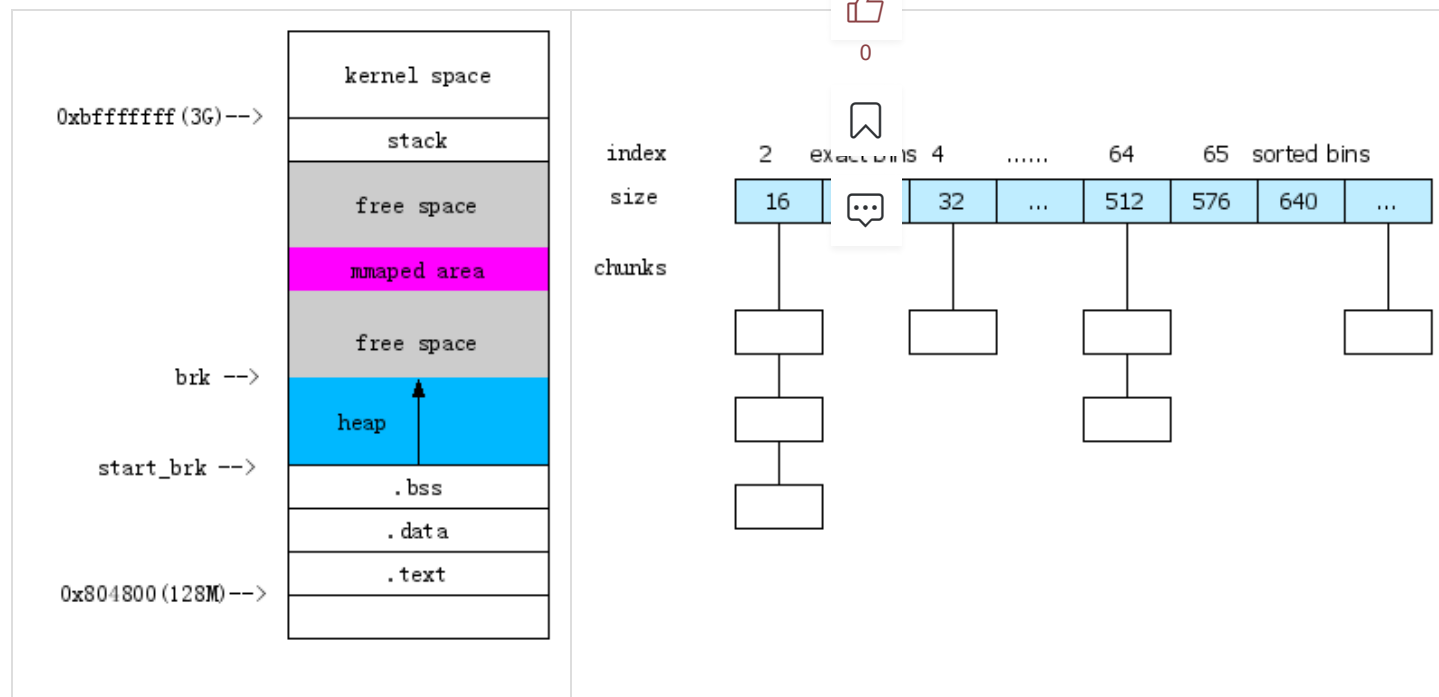
Caffe专栏

文章: 6篇

1) `malloc()/free()` 是C语言中常用的两个内存 分配/释放 函数, 但是ANSI C并没有指定它们具体应该如何实现。因此在各个系统级平台上 (windows, mac, linux等等), 函数的底层的内存操纵方式并不一样。

2) 在linux下, `malloc()/free()`的实现是由 **glibc库** 负责, 它会根据一定的策略与系统底层通信 (调用系统 API), 所以, 用户程序并不会直接和linux kernel进行交互, 而是交由glibc托管, 提供默认的内存管理器。关系为: 用户程序 ----> glibc----> linux kernel。

3) glibc使用了 **ptmalloc** 作为其内存管理器的实现。



* brk分配的内存 chunk list, 只能从top开始线性向下释放。中间释放掉的chunk, 无法归还给OS, 而是链入到了 bins/fast bins的容器中。

* mmap分配的内存, 等于直接从物理内存中映射了一块过来, 释放这块内存时, 可以直接归还给OS。

* 对于request的一块内存, 到底是由brk分配, 还是由mmap分配, 这是由glibc策略机制决定的。有个 threshold, 可以调节这种策略, 默认小于threshold由brk分配, 大于等于则由mmap分配, threshold默认为 128kb。glibc具体从那个版本开始支持动态调节threshold需要调查。默认下, 在64位系统上, brk可以动态调整



阅读: 12429

文章分类

- 并行计算专题 (9)
- 数字图像处理专题 (27)
- 视频处理专题 (9)
- OpenCV专题 (70)
- DeepID专题 (7)
- ImageSearch (27)
- ImageProcess (106)
- AuidoProcess (4)
- AudioFeature (1)
- DeepLearning (175)
- 相机标定畸变校正 (19)
- 机器学习 (91)
- FFMpeg (15)
- ObjectDetect (36)
- 人脸检测 (45)
- 人脸识别 (42)
- 皮肤检测 (5)
- 性别识别 (1)
- C++ (38)
- C (35)
- Shell (24)
- OCR文字识别 (5)
- 实用工具 (40)
- CUDA (9)
- system (17)
- GearMan (3)

到从 128kb到32mb。调整策略基本可以概括为：发现对顶可以release的可用内存超过256kb的话，就将threshold调整到256kb，依次类推直到32mb。

glibc使用的两种用户程序内存管理机制，考虑了与系统底层通信代价，如果直接操纵大量小块内存，频繁地与系统调用进行通信，会降低程序的运行效率。将小块内存放入brk维护的堆中，等于实现了一块缓存（cache），用完了可以先攒起来，到时候可以一起归还给系统。但是由于它的实现，只维护了堆顶的一个指针。因此想要内存归还系统，必须从顶向下，依次归还。在这种机制下，如果堆顶内存一直被占用，底下所有内存虽然都已经没用了，但是这种设计决定内存此时不可以还给系统，因此出现“洞（Hole）”的问题。这种设计对一些由于业务需求，频繁申请/释放小块内存的用户程序而言，问题会比较突出。虽然**glibc**制定了这种有些“强硬”的内存管理方案，但是也提供了一些方法允许调节相关阈值（threshold）虽然不能干涉管理内存，但可以通过这些方法，决定“多大算大，多小算小”以及“攒到多少就归还”等这类问题。

3 mallopt() 与 malloc_trim(0)

1) mallopt是一个专门调节相关阈值的函数

URL: <http://man7.org/linux/man-pages/man3/mallopt.3.html>

```
#include < malloc.h >
int mallopt(int param, int value);
```

M_MMAP_THRESHOLD For allocations greater than or equal to the limit specified (in bytes) by **M_MMAP_THRESHOLD** that can't be satisfied from the free list,

the memory-allocation functions employ **mmap(2)** instead of increasing the program break using **sbrk(2)**.

Allocating memory using **mmap(2)** has the significant advantage that the allocated memory blocks can always be independently released back to the system.

(By contrast, the heap can be trimmed only if memory is freed at the top end.)

On the other hand, there are some disadvantages to the use of **mmap(2)**: deallocated space is not placed on the free list for reuse by later allocations;

memory may be wasted because **mmap(2)** allocations must be page-aligned; and the kernel must perform the expensive task of zeroing out memory allocated

via **mmap(2)**. Balancing these factors leads to a default setting of 128*1024 for the **M_MMAP_THRESHOLD** parameter. The lower limit for this parameter is 0.

WSGI (2)
Python (16)
CGI (1)
3D投影 (1)
GPU显卡 (4)
glibc (4)
手机卡 (1)
分布式 (1)
zookeeper (2)
互联网 (4)
条形码 (1)
二维码 (2)
图像特征 (26)
图像去雾 (8)
图像边缘检测 (3)
距离度量 (3)
图像分析 (2)
概率论知识 (12)
URL (23)
图像模糊 (6)
数学知识 (18)
TF-IDF (3)
face morphing (1)
Caffe (41)
硬盘维修 (1)
汇编 (1)
代码管理 (1)
ZMQ (4)
立体匹配 (9)
SpeechRecognition (3)
AudioSearch (2)
TextSearch (5)

The upper limit is `DEFAULT_MMAP_THRESHOLD_MAX`: 512x1024 on 32-bit systems or 4x1024x1024xsizeof(long) on 64-bit systems.

Note: Nowadays, glibc uses a dynamic mmap threshold by default. The initial value of the threshold is 128x1024, but when blocks larger than the current threshold and less than or equal to `DEFAULT_MMAP_THRESHOLD_MAX` are freed, the threshold is adjusted upwards to the size of the freed block.

When dynamic mmap thresholding is in effect, the threshold trimming the heap is also dynamically adjusted to be twice the dynamic mmap threshold.

Dynamic adjustment of the mmap threshold is disabled if any of the `M_TRIM_THRESHOLD`, `M_TOP_PAD`, `M_MMAP_THRESHOLD`, or `M_MMAP_MAX` parameters is set.

The `mallopt()` function adjusts parameters that control the behavior of the memory-allocation functions (see `malloc(3)`). The `param` argument specifies the parameter to be modified, and `value` specifies the new value for that parameter.

The following values can be specified for `param`:

`M_CHECK_ACTION`

Setting this parameter controls how glibc responds when various kinds of programming errors are detected (e.g., freeing the same pointer twice). The 3 least significant bits (2, 1, and 0) of the value assigned to this parameter determine the glibc behavior, as follows:

Bit 0 If this bit is set,

then print a one-line message on `stderr` that provides details about the error. The message starts with the string `*** glibc detected ***`, followed by the program name, the name of the memory-allocation function in which the error was detected, a brief description of the error, and the memory address where the error was detected.

Bit 1 If this bit is set,

SearchEngineSoftware (2)

自动摘要 (5)

Redis 数据库 (11)

NoSQL (2)

MongoDB (5)

H265 (2)

管理 (2)

编解码知识 (31)

日志库 (1)

Centos (2)

数据库 (9)

Makefile (4)

流媒体 (37)

自然语言处理 (6)

数据挖掘 (3)

推荐系统 (24)

valgrind (1)

inotify (1)

SQLite (1)

GPUImage (11)

WebP (4)

图像滤镜 (38)

颜色空间 (7)

数据结构 (1)

图像拼接 (23)

Charles (1)

排序算法 (7)

GLSL (13)

Android (13)

Spark (4)

s-plus (1)

KNN (4)

then, after printing any error message specified by bit 0, the program is terminated by calling abort(3). In glibc versions since 2.4, if bit 0 is also set, then, between printing the error message and aborting, the program also prints a stack trace in the manner of backtrace(3), and prints the process's memory mapping in the style of /proc/[pid]/maps (see proc(5)).

Bit 2 (since glibc 2.4),

This bit has an effect only if bit 0 is also set. If this bit is set, then the one-line message describing the error is simplified to contain just the name of the function where the error was detected and the brief description of the error.

The remaining bits in value are ignored.

Combining the above details, the following numeric values are meaningful for M_CHECK_ACTION:

0 Ignore error conditions; continue execution (with undefined results).

1 Print a detailed error message and continue execution.

2 Abort the program.

3 Print detailed error message, stack trace, and memory mappings, and abort the program.

5 Print a simple error message and continue execution.

7 Print simple error message, stack trace, and memory mappings, and abort the program.

Since glibc 2.3.4, the default value for the M_CHECK_ACTION parameter is 3. In glibc version 2.3.3 and earlier, the default value is 1. Using a nonzero M_CHECK_ACTION value can be useful because otherwise a crash may happen much later, and the true cause of the problem is then very hard to track down.

M_MMAP_MAX

This parameter specifies the maximum number of allocation requests that may be simultaneously serviced using mmap(2). This parameter exists because some systems have a limited number of internal tables for use by mmap(2), and using more than a few of them may degrade performance.

RANSAC (1)
垃圾文本过滤 (4)
IOS (4)
图像分割 (6)
ImageNet (3)
敏感词 (8)
Socket (6)
科研指标 (5)
word2vec (1)
中文分词 (6)
Weka (1)
贝叶斯分类器 (1)
LSTM (2)
图像质量评价 (6)
模糊检测 (1)
nginx (10)
图像增强 (3)
SIMD (6)
手背静脉识别 (1)
系统设计 (5)
alphago (2)
视频分割 (1)
scala (1)
Docker (15)
HDR (3)
Android (4)
虚拟现实 (7)
大数据 (3)
TensorFlow (9)
ADAS_辅助驾驶_自动驾驶 (74)
测距 (5)
立体视觉 (1)

The default value is 65,536, a value which has no special significance and which serves only as a safeguard. Setting this parameter to 0 disables the use of `mmap(2)` for servicing large allocation requests.

M_MMAP_THRESHOLD

For allocations greater than or equal to the limit specified (in bytes) by `M_MMAP_THRESHOLD` that can't be satisfied from the free list, the memory-allocation functions employ `mmap(2)` instead of increasing the program break using `sbrk(2)`.

Allocating memory using `mmap(2)` has the significant advantage that the allocated memory blocks can always be independently released back to the system. (By contrast, the heap can be trimmed only if memory is freed at the top end.) On the other hand, there are some disadvantages to the use of `mmap(2)`: deallocated space is not placed on the free list for reuse by later allocations; memory may be wasted because `mmap(2)` allocations must be page-aligned; and the kernel must perform the expensive task of zeroing out memory allocated via `mmap(2)`. Balancing these factors leads to a default setting of 128*1024 for the `M_MMAP_THRESHOLD` parameter. The lower limit for this parameter is 0. The upper limit is `DEFAULT_MMAP_THRESHOLD_MAX`: 512*1024 on 32-bit systems or 4*1024*1024*sizeof(long) on 64-bit systems. Note: Nowadays, glibc uses a dynamic `mmap` threshold by default. The initial value of the threshold is 128*1024, but when blocks larger than the current threshold and less than or equal to `DEFAULT_MMAP_THRESHOLD_MAX` are freed, the threshold is adjusted upward to the size of the freed block. When dynamic `mmap` thresholding is in effect, the threshold for trimming the heap is also dynamically adjusted to be twice the dynamic `mmap` threshold. Dynamic adjustment of the `mmap` threshold is disabled if any of the `M_TRIM_THRESHOLD`, `M_TOP_PAD`, `M_MMAP_THRESHOLD`, or `M_MMAP_MAX` parameters is set.

M_MXFAST (since glibc 2.3)

Set the upper limit for memory allocation requests that are satisfied using "fastbins". (The measurement unit for this parameter is bytes.) Fastbins are storage areas that hold deallocated blocks of memory of the same size without merging adjacent free blocks. Subsequent reallocation of blocks of the same size

FFT (3)
QT (1)
SSE (1)
Elasticsearch (3)
服务器架构 (4)
LBS (13)
开源项目 (2)
前背景分离 (2)
Windows (2)
Object-C (1)
魔方 (2)
显著图 (2)
手势识别 (2)
数码相机 (2)
词袋模型BOW (3)
LDA (1)
视频特征 (1)
视频音频处理 (4)
Lua (1)
双目立体视觉 (1)
wurenjiashi (1)
graphics rendering engines (1)
Git (22)
聊天机器人 (1)
色情识别 (2)
FPGA (1)
RabbitMQ (1)
机器人 (1)
SVM (1)
ARM (7)
工作生活 (15)
OpenMP (1)

can be handled very quickly by allocating from the fastbin, although memory fragmentation and the overall memory footprint of the program can increase. The default value for this parameter is $64 \cdot \text{sizeof}(\text{size_t})/4$ (i.e., 64 on 32-bit architectures). The range for this parameter is 0 to $80 \cdot \text{sizeof}(\text{size_t})/4$. Setting `M_MXFAST` to 0 disables the use of fastbins.

`M_PERTURB` (since glibc 2.4)

If this parameter is set to a nonzero value, then bytes of allocated memory (other than allocations via `calloc(3)`) are initialized to the complement of the value in the least significant byte of value, and when allocated memory is released using `free(3)`, the freed bytes are set to the least significant byte of value. This can be useful for detecting errors where programs incorrectly rely on allocated memory being initialized to zero, or reuse values in memory that has already been freed.

`M_TOP_PAD`

This parameter defines the amount of padding to employ when calling `sbrk(2)` to modify the program break. (The measurement unit for this parameter is bytes.) This parameter has an effect in the following circumstances:

- When the program break is increased, then `M_TOP_PAD` bytes are added to the `sbrk(2)` request.
- When the heap is trimmed as a consequence of calling `free(3)` (see the discussion of `M_TRIM_THRESHOLD`) this much free space is preserved at the top of the heap.

In either case, the amount of padding is always rounded to a system page boundary. Modifying `M_TOP_PAD` is a trade-off between increasing the number of system calls (when the parameter is set low) and wasting unused memory at the top of the heap (when the parameter is set high). The default value for this parameter is $128 \cdot 1024$.

`M_TRIM_THRESHOLD`

Face2Face-Reenactment (5)

Axure (1)

算法 (2)

编程 (1)

Paper-论文 (1)

安全 (2)

OpenStack (1)

Linux (29)

SUSE (2)

汽车 (3)

redmine (5)

语音识别 (3)

云直播 (1)

Bundler (1)

SLAM (4)

Web系统 (13)

Samba (6)

串口调试 (2)

无人驾驶 (2)

传感器 (1)

计算机病毒 (2)

数据加密 (11)

Unity3D (1)

VR-虚拟现实 (4)

AR-增强现实 (7)

MR-混合现实 (3)

行人检测 (5)

图像融合 (11)

Eigen (1)

微信开发 (1)

图像压缩 (1)

相机 (1)

When the amount of contiguous free memory at the top of the heap grows sufficiently large, `free(3)` employs `sbrk(2)` to release this memory back to the system. (This can be useful in programs that continue to execute for a long period after freeing a significant amount of memory.) The `M_TRIM_THRESHOLD` parameter specifies the minimum size (in bytes) that this block of memory must reach before `sbrk(2)` is used to trim the heap.

The default value for this parameter is `128*1024`. Setting `M_TRIM_THRESHOLD` to `-1` disables trimming completely. Modifying `M_TRIM_THRESHOLD` is a trade-off between increasing the number of system calls (when the parameter is set low) and wasting unused memory at the top of the heap (when the parameter is set high).

Environment variables

A number of environment variables can be defined to modify some of the same parameters as are controlled by `mallopt()`. Using these variables has the advantage that the source code of the program need not be changed. To be effective, these variables must be defined before the first call to a memory-allocation function. (If the same parameters are adjusted via `mallopt()`, then the `mallopt()` settings take precedence.) For security reasons, these variables are ignored in `set-user-ID` and `set-group-ID` programs.

The environment variables are as follows (note the trailing underscore at the end of the name of each variable):

`MALLOC_CHECK_`

This environment variable controls the same parameter as `mallopt()` `M_CHECK_ACTION`. If this variable is set to a nonzero value, then a special implementation of the memory-allocation functions is used. (This is accomplished using the `malloc_hook(3)` feature.) This implementation performs additional error checking, but is slower than the standard set of memory-allocation functions. (This implementation does not detect all possible errors; memory leaks can still occur.)

HTML (2)
跟踪算法 (1)
Vi-Vim (2)
Svn (1)
Dsp (1)
线性代数 (1)
嵌入式 (2)
播放器 (10)
V4L (2)
Matlab (1)
OpenGL (1)
最强大脑讨论 (3)
Screen (2)
golang (21)
NLP (7)
Kubernetes (8)
yaml (3)
Etcd (7)
唯一ID (1)
traefik (1)
numpy (6)
Matplotlib (1)
kinect (5)
爬虫 (1)
Protobuf (1)

文章存档

2018年3月 (1)
2018年2月 (7)
2018年1月 (4)
2017年12月 (4)

The value assigned to this environment variable should be a single digit, whose meaning is as described for `M_CHECK_ACTION`. Any characters beyond the initial digit are ignored. For security reasons, the effect of `MALLOC_CHECK_` is disabled by default for set-user-ID and set-group-ID programs. However, if the file `/etc/suid-debug` exists (the content of the file is irrelevant), then `MALLOC_CHECK_` also has an effect for set-user-ID and set-group-ID programs.

`MALLOC_MMAP_MAX_`

Controls the same parameter as `mallopt()` `M_MMAP_MAX`.

`MALLOC_MMAP_THRESHOLD_`

Controls the same parameter as `mallopt()` `M_MMAP_THRESHOLD`.

`MALLOC_PERTURB_`

Controls the same parameter as `mallopt()` `M_PERTURB`.

`MALLOC_TRIM_THRESHOLD_`

Controls the same parameter as `mallopt()` `M_TRIM_THRESHOLD`.

`MALLOC_TOP_PAD_`

Controls the same parameter as `mallopt()` `M_TOP_PAD`.

2) `malloc_trim()`

负责告诉**`glibc`**在**`brk`**维护的堆队列中，堆顶留下多少的空余空间（free space），其他往上的空余空间全部归还给系统。它不能归还除堆顶之外的内存。

URL: http://man7.org/linux/man-pages/man3/malloc_trim.3.html



2017年11月 (13)

展开

阅读排行

NVIDIA GPU 运算能力列表	(37986)
人脸识别活体检测的一些方法	(21843)
矩阵特征值与行列式、迹的关系	(19979)
cuDNN 下载地址	(18897)
CNN笔记：通俗理解卷积神经...	(17644)
image stitch(国外开源的图像...	(17029)
无参考图像的清晰度评价方法	(13958)
port常用和不常用端口一览表	(12545)
Caffe 编译安装	(11874)
如何用TensorFlow和TF-Slim...	(11467)

最新评论

在Python中使用LDA处理文本
Johline : 您好! 我通过LDA训练出主题模型, 我要输入一篇文本能否利用这个模型进行主题分布, 以及获得主题下的词...

opencv实现图像的拼接功能
凌风探梅 : [reply]qq_36583260[/reply] printUs age() 和126行, 代码还...

opencv实现图像的拼接功能
qq_36583260 : imge 获取图像的路径应该加在哪里?

Tensorflow 自动文摘: ...
guo070110053 : [reply]Real_Myth[/reply] 你好, 博主, 语料库的数字还有英文用正则都不能替换...

CNN笔记: 通俗理解卷积神经网络
qq_1615443838 : 给博主点赞



使用OpenCV进行人脸识别的三种...
js784381664 : 学习了, 谢博主分享

Tensorflow 自动文摘: ...

```
#include <malloc.h>
int malloc_trim(size_t pad);
```

The **malloc_trim ()** function attempts to release free memory at the top of the heap (by calling [sbrk\(2\)](#) with a suitable argument).



The *pad* argument specifies the amount of free space to leave untrimmed at the top of the heap. If this argument is 0, only the minimum amount of memory is  tained at the top of the heap (i.e., one page or less). A nonzero argument can be used to maintain some trailing space at the top of the heap in order to allow future allocations to be made without hav  to extend the heap with [sbrk\(2\)](#).

4 解决方法

从操作系统的角度看, 进程的内存分配由两个系统调用完成: brk和mmap。brk是将数据段(.data)的最高地址指针 _edata往高地址推, mmap是在进程的虚拟地址空间中找一块空闲的。其中, mmap分配的内存由munmap释放, 内存释放时将立即归还操作系统; 而**brk分配的内存需要等到高地址内存释放以后才能释放**。也就是说, 如果先后通过brk申请了A和B两块内存, 在B释放之前, A是不可能释放的, 仍然被进程占用, 通过TOP查看疑似“内存泄露”。默认情况下, 大于等于128KB的内存分配会调用mmap/mummap, 小于128KB的内存请求调用sbrk (可以通过设置M_MMAP_THRESHOLD来调整)。Glibc的: **M_MMAP_THRESHOLD可以动态调整**。M_MMAP_THRESHOLD的值在128KB到32MB(32位机)或者64MB(64位机)之间动态调整, 每次申请并释放一个大小为2MB的内存后, M_MMAP_THRESHOLD的值被调整为2M到2M + 4K之间的一个值. 例如:

```
char* no_used = new char[2 * 1024 * 1024]; memset(no_used, 0xfe, 2 * 1024 * 1024); delete[] no_used;
```

这样,M_MMAP_THRESHOLD的值调整为2M到2M + 4K之间的一个值, 后续申请 <= 2 * 1024 * 1024的内存块都会走sbrk而不是mmap, 而sbrk需要等到高地址内存释放以后低地址内存才能释放。

可以显式设置M_MMAP_THRESHOLD或者 M_MMAP_MAX来关闭M_MMAP_THRESHOLD动态调整的特性, 从而避免上述问题。

guo070110053 : [reply]Real_Myth[/reply] 你好，博主，这个数据预处理替换数字还有日期用正则...

cuDNN 下载地址
tony_3 : 一样要进官网。。。

k8s docker集群搭建
GounX : 总结得很详细，然来还能用yum直接安装。

基于形态学操作提取水平和垂直线条(...
KittyRong002 : 你好，我想请教一下，图像处理后的结果要在哪里读取？谢谢

联系我们




请扫描二维码联系客服

✉ webmaster@csdn.net

☎ 400-660-0108

💬 QQ客服 🗨 客服论坛

关于 招聘 广告服务  百度

©1999-2018 CSDN版权所有

京ICP证09002463号

经营性网站备案信息

网络110报警服务

中国互联网举报中心

北京互联网违法和不良信息举报中心

当然，mmap调用是会导致进程产生缺页中断的，为了提高性能，常见的做法如下：

- 1) 将动态内存改为静态，比如采用内存池技术或者启动的时候给每个线程分配一定大小，比如8MB的内存，以后直接使用；
- 2) 禁止mmap内存调用，禁止Glibc内存缩紧将内存归还系统，Glibc相当于实现了一个内存池功能。只需要在进程启动的时候加入两行代码：

```
mallopt(M_MMAP_MAX, 0); // 禁止malloc调用mmap分配内存  
  
mallopt(M_TRIM_THRESHOLD, 0); // 禁止内存缩进，sbrk内存释放后不会归还给操作系统
```

- [上一篇](#) glibc (ptmalloc) 内存暴增问题解决
- [下一篇](#) Advanced Memory Allocation 内存分配进阶

您还没有登录,请[\[登录\]](#)或[\[注册\]](#)

Linux虚拟内存管理(glibc) mmap sbrk

 origin_lee 2015年01月15日 14:03 1031

在使用mysql作为DB开发的兑换券系统中，随着分区表的不断创建，发现mysqld出现了疑似“内存泄露”现象，但通过 valgrind 等工具检测后，并没发现类似的问题(最终原因是由于glibc的内存...

Glibc 内存管理知识点总结

这几天在看Glibc 内存管理模块的内容，感觉收获颇多，在此做个简单的总结，以便知识点回顾。先介绍一下相关的背景。有个项目组在研发一个类似数据库的NoSql 系统时，遇到了Glibc 内存暴增问题...

转载-----GLIBC内存分配机制引发的“内存泄露”



转载: http://blog.sina.com.cn/s/blog_a584f2e20101505f.html Linux下Glibc的内存管理机制大致如下: 从操作系统的角度看, 进程的内存分配由...



donghanhang 2016年06月28日 16:36 657

Linux glibc 的 mallopt 海量小内存回收问题



wscdylzjy 2015年03月13日 18:04 1826

一.http://blog.sina.com.cn/s/blog_4673e6030101haxg.html 最近使用ACE的Message_Block时发现, 程序运行一段时间之后内存越吃越多, ...

Advanced Memory Allocation 内存分配进阶



AMWIHHC 2012年04月21日 03:25 4931

May 01, 2003 By Gianluca Insolvibile in Embedded Software Call some useful fuctions...

Linux内存分配小结--malloc、brk、mmap



gfgdsg 2015年01月14日 14:54 2783

Linux 的虚拟内存管理有几个关键概念: 1、每个进程都有独立的虚拟地址空间, 进程访问的虚拟地址并不是真正的物理地址; 2、虚拟地址可通过每个进程上的页表(在每个进程的内核虚拟地址空间)与物理地...

glibc内存管理

 shiludeng 2017年11月23日 11:42 50

我们调用free函数释放之前分配的内存，其实是将内存还给glibc，但是glibc却并不一定会将内存还给操作系统，导致使用top命令查看进程占用的内存时，出现类似内存泄漏的现象。关于glibc的内存管...

glibc内存管理器

 wangwei 2014年07月28日 12:22 594

那么我们每次调用malloc来分配一块内存，都进行相应的系统调用呢？答案是否定的，这里我要引入一个新的概念，glibc的内存管理器。我们知道malloc和free等函数都是包含在gli...

内存管理器（七）Glibc malloc 实现（三）--多线程思想(先占个坑)

基本方式1.使用锁的方式来支持多线程 我们之前介绍过，库函数的实现方式是通过一个主分配区和多个非主分配区来组织的，每一个分配区中又有一个bins.我们先说对于大的分配区是如何保证线程安全的。首...

 zmrlinux 2015年10月24日 21:08 687

Linux之glibc内存管理malloc和free

 donghanhang 2016年06月28日 17:14 1920

转载：<http://blog.csdn.net/phenics/article/details/7770531> 前言 C语言提供了动态内存管理功能, 在C语言中, 程序员可以使用 ma...

glibc内存管理

 u012275397 2014年12月20日 16:23  742

X86平台Linux进程内存布局如下：上面个段的含义如下：text：存放程序代码的，编译时确定，只读；data：存放程序运行时就能确定的数据，可读可写；bss：...

malloc()--控制 内存分配的函数

 3920085 2016年10月18日 11:35  1713

malloc函数可以控制 内存分配的函数：int malloc(int param,int value)//控制 内存分配的函数。param 的取值可以为M_CHECK_ACTION、M...

pyclutter内存不释放解决记录

 AMWIIHC 2012年04月20日 15:35  1773

由于程序最终是在一款嵌入式平台上运行.由 python+clutter 的程序在pc上运行内存可以非常理想的释放,而在嵌入式平台上却不能释放.开始时,一直怀疑是内存泄露的原因,但是前面的工作将内存...

GLIBC内存分配机制引发的“内存泄露”

 hintonic 2013年04月07日 10:35  1569

<http://www.longyusoft.com/detail.php?id=295> 内容提要：我们正在开发的类数据库系统有一个内存模块，出现了一个疑似“内存泄露”问题，现象如下：内存模块...

glibc下的内存管理

 xocoder 2016年03月03日 14:45  402

<http://www.cnblogs.com/lookof/archive/2013/03/26/2981768.html>

glibc内存碎片【转】

 ferrarild 2013年01月04日 21:05  1328

Glibc内存管理--ptmalloc2源代码分析

 yazhouren 2014年08月14日 10:17  1201

原文地址: <http://mqzhuang.iteye.com/blog/1005909>



 0 heiyeshuwu 2014年05月28日 12:06  2193



[转]glibc 内存池管理 ptmalloc

来源: ptmallocphenix* 2006-06-07目录 1 前言 2 x86平台Linux程序内存分布 3 Allocator 4 chuck的组织 ...



glibc内存管理ptmalloc源代码分析



2016年03月23日 14:22 1.82MB [下载](#)

glibc内存管理ptmalloc源代码分析PDF



2013年05月20日 15:20 2.5MB [下载](#)