

**HelloGcc Workshop China 2010**

**2010年11月 北京**

**ChinaUnix**

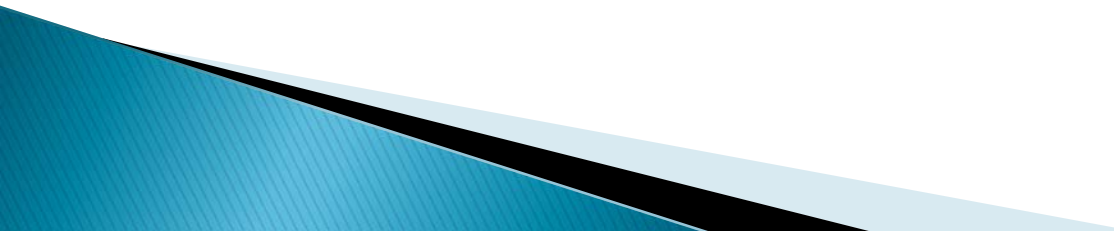


# 内存管理机制与优化

袁鹏

yuanpeng@malloc.cn

# 概述

- ▶ 内存管理的层次
  - ▶ malloc的设计策略
  - ▶ DLmalloc的设计
  - ▶ 多线程环境中的malloc
  - ▶ 如何选择malloc?
- 

# 内存管理的层次

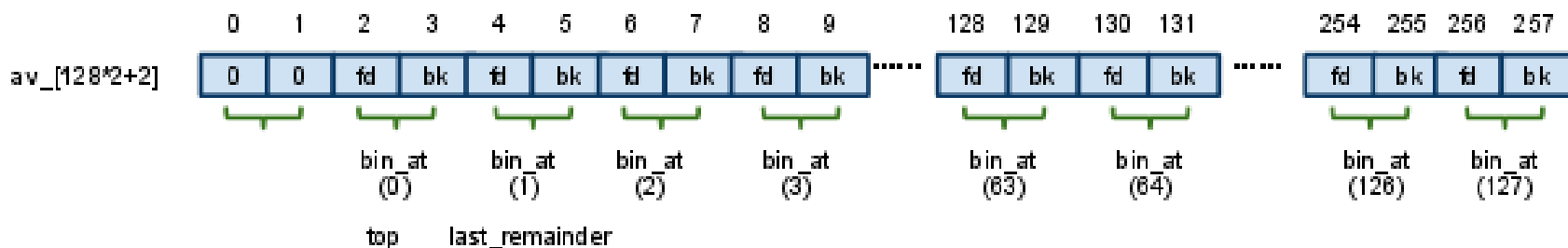
- ▶ 程序如何使用内存？ 插在主板上的那块内存！
  - C: malloc/free
  - C++: new/free (libstdc++中最终调用malloc/free)
  - Java: new (Garbage Collection机制)
  - 汇编: 系统调用 (brk or sysmap)
  - Kernel: virtual memory, page, TLB
- ▶ 这些都是库或运行时提供的函数
  - 不同的库中，这些内存管理函数的实现机制不一样。
    - Glibc: DL malloc (<http://gee.cs.oswego.edu/dl/>)
    - TCMalloc, Hoard, nedmalloc
    - Apache: pool allocator
  - Java中有很多的GC机制供开发者选择和调优

# Malloc的设计策略

- ▶ Malloc级别看到的是虚拟存储。
  - 对内存的申请是通过brk或sysmap系统调用实现。
- ▶ 速度
- ▶ 存储需求
- ▶ 可扩展性
- ▶ 伪共享
  - Cache line上的。不同线程间操作同一Cache line上的不同对象引起，强制内存更新以维持一致性。
- ▶ Cache性能
  - 如何更有效的提高Cache局部性？

# Dlmalloc的设计

- 分配和释放都是基于 *存储块* 进行。存储块是一块虚拟存储区域。
- 在系统中，有两种存储块。一种是malloc()给用户的 *分配块*，另一种是free()释放的 *有效块*。有效块通过箱（bin）进行管理。
- 有效块基于其分配大小放入对应的箱中



# 多线程环境中的Malloc

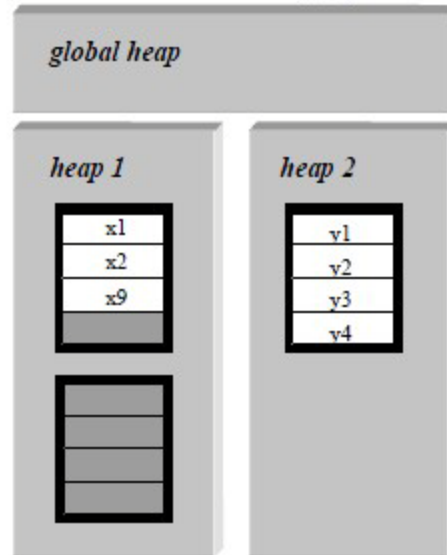
## ▶ Hoard

- 维护一个全局堆。每个线程有用一个私有堆。
- 和Tcmalloc类似的机制。

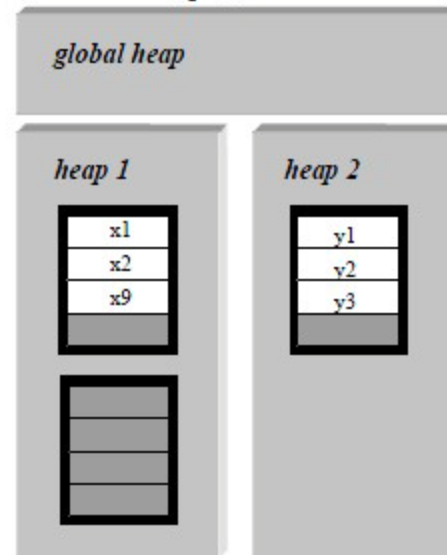
## ▶ Cache的性能

- 不同块之间的映射冲突。更关注的是最后一级的Cache。
- 这级cache通常是物理索引。因此仅靠malloc函数层次的设计不够。还需要OS的参与。

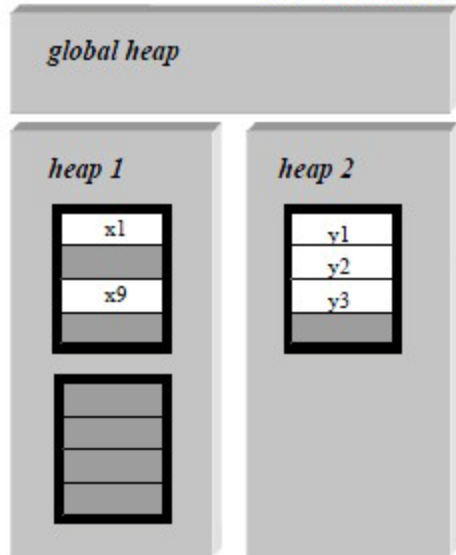
```
t1: x9 = malloc(s);
```



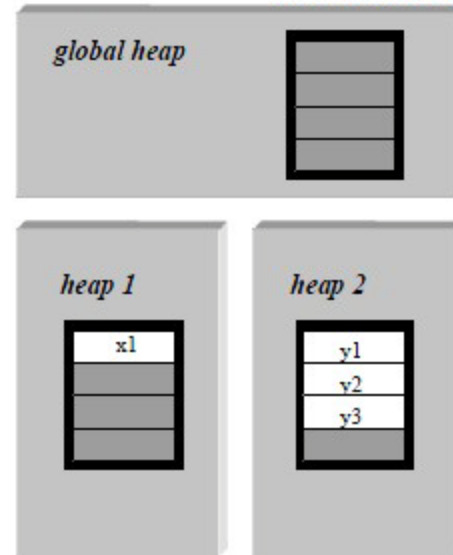
```
t1: free(y4);
```



```
t2: free(x2);
```



```
t2: free(x9);
```



# 如何选择Malloc

- ▶ Hoard或tcmalloc一定好么？池存储一定高效么？
- ▶ 基于程序的特征以及具体的评测。
  - 多线程程序
  - 小对象还是大对象
  - 频繁分配释放？
- ▶ apache, 197.parser, 175.vpr, 176.gcc
  - Hoard不比DLmalloc好。
  - 对于apache这样的程序，每个请求都会分配对象，而且生存期不一致。池存储会增加存储消耗。
- ▶ 具体的程序和运行平台评测+逻辑的分析



# 借助工具

- ▶ Valgrind是很好的检测程序中malloc bug的工具。
- ▶ 一些编译器实现了自动化的池存储
  - 比如LLVM, Open64, GCC（进行中）
- ▶ 基本思想
  - 同一个分配点的对象放到同一个池中
    - 注意分配点是否是真的malloc还是封装
  - 同一类型的对象放到同一个池中
  - 小对象！
- ▶ 实现的都没人维护了。。。



# 谢谢！ 欢迎提问。

12月份，我会将malloc系列的文章（包括具体的技术实现和代码）放到HelloGCC的网站，以及另一个技术网站上。