

轻量级内存分配器 LWMalloc 技术报告 v0.1

参赛队员：池雨泉 王志扬

指导老师：付印金

所属机构：中山大学

摘要

1 背景、引言与动机

随着计算机系统性能的不断提升和多核架构的广泛应用，现代应用程序对内存管理的需求也日益复杂。优秀的内存分配器不仅需要提供 `malloc`, `free` 等标准接口，还必须在多线程环境下保证内存分配与释放操作的正确性、高效性与安全性。

传统的内存分配器（如 `ptmalloc2` [1]）在多线程并发场景下容易遇到锁竞争、内存碎片化以及分配效率下降等问题。为了解决这些问题，许多现代内存分配器引入了大小类系统（`size class system`）和线程本地缓存（`thread-local cache`）等技术。

在工业控制、航空航天等严苛的应用环境中，计算机硬件常会受到高低温、潮湿、电磁辐射、宇宙射线等恶劣因素的影响。进一步讲，计算机的主存储器在前述恶劣环境下可能会出现比特翻转的问题，导致系统出现错误甚至崩溃。传统的高可靠解决方案常采用硬件层面的升级改造，例如 `ECC memory` 和 `chipkill`，开发周期长且成本昂贵。如果能通过软件的解决方案，实现一个面向严苛场景的内存管理器，有助于节省硬件成本。

在传统的内存分配模式中，应用程序每次使用 `malloc` 分配内存之后，都需要显式地调用 `free` 将其释放。如果忘记了调用 `free` 函数，就会导致内存泄漏。如果错误地在调用 `free` 函数之后继续使用相应的内存区域，就可能发生释放后使用（`use-after-free`）或多次释放（`double-free`）等安全问题。为此，开源的 APR 项目¹引入了内存池管理相关接口：应用程序可以通过调用 `apr_pool_create` 接口创建一个或多个内存池，并使用 `apr_palloc` 在内存池中分配内存块；在内存池的生命周期结束时，只需要调用 `pool_destroy` 接口销毁内存池，即可释放之前通过该内存池分配的所有内存块，从而避免了前述内存泄漏、释放后使用、多次释放等效率和安全性问题。

基于以上背景，参照学术界和产业界的现有成果，我们设计了一款名为 LWMalloc²的轻量级内存分配器，具备以下特性：

- 支持标准的 `malloc`, `free`, `calloc`, `realloc` 接口；
- 适配多线程环境；

¹ 项目地址：<https://github.com/apache/apr>。

² LW 是英文单词 `lightweight` 的缩写。

- 支持抗单粒子翻转的可靠读写接口；
- 支持 pool create, pool clear, pool destroy, palloc, pcalloc 内存池管理接口。

本文余下内容的安排如下：（介绍本文余下内容的结构）

2 系统架构与接口设计

2.1 系统架构

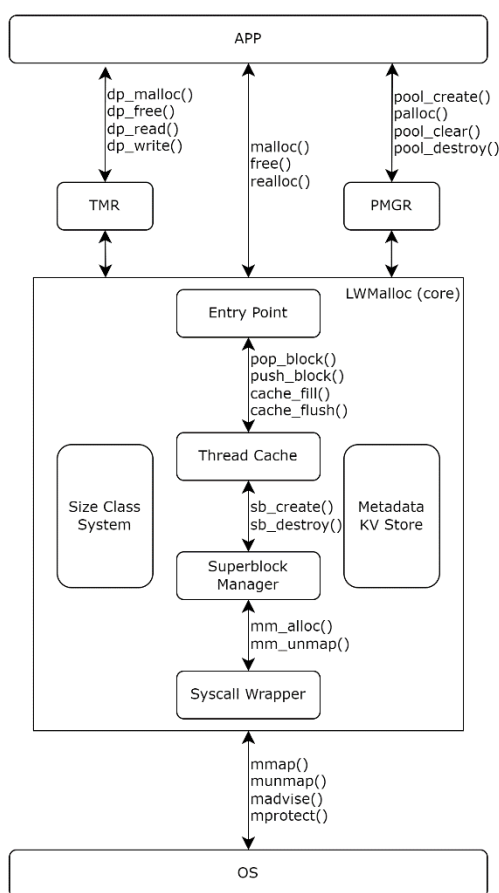


图 1 LWMalloc 系统架构

LWMalloc 的系统架构如图 1 所示。作为一个用户态的内存分配器, LWMalloc 本质上是操作系统和应用程序之间的一个中间件, 向下调用操作系统提供的 `mmap`, `munmap` 等系统调用, 向上提供 `malloc`, `free` 等库函数接口。LWMalloc 的核心由 6 个模块组成。其中, 最底层的 MM 模块用于封装 `mmap`, `munmap` 等系统调用, 以简化 LWMalloc 内部的系统调用流程; 最顶层的 LW 模块则是 LWMalloc 的入口点, 向下调用 LWMalloc 的内部模块, 向上提供标准的 `malloc`, `free` 接口。分配和释放操作的关键路径位于 TC 和 SBM 两个模块中。辅助模块 SC 用于执行大小类相关的计算。KV 模块则用于存放元数据。

在 LWMalloc 核心的基础上，我们还实现了 TMR 可靠读写模块和 PMGR 内存池管理模块。前者用于提供抗单粒子翻转的可靠存储和读写接口，后者则用于以内存池为单位管理程序运行过程中动态分配的内存区域。

2.2 接口设计

```
void *lw_malloc(size_t size);
void lw_free(void *ptr);
void *lw_calloc(size_t nmemb, size_t size);
void *lw_realloc(void *ptr, size_t size);
```

图 2 标准内存分配接口

图 2 展示了 LWMalloc 所提供的标准内存分配接口。其中：

- `lw_malloc` 用于分配一块**未初始化的**、大小等于 `size` 的内存区域，并返回该区域的起始地址。当 `size` 等于 0 时，`lw_malloc` 返回一个可以被 `lw_free` 释放的非空指针。当系统的可用内存不足时，`lw_malloc` 返回 `NULL`。
- `lw_free` 用于释放 `ptr` 参数所指向的内存区域，注意 `ptr` 参数必须是空指针 `NULL` 或者先前某次 `lw_malloc`、`lw_calloc` 或 `lw_realloc` 调用的返回值。如果传入的 `ptr` 参数不符合前述条件，或者使用相同的 `ptr` 参数连续调用两次及以上的 `lw_free`，将产生未定义的行为。
- `lw_calloc` 用于为一个由 `n` 个大小等于 `size` 的元素组成的数组分配内存空间。这片空间会被初始化为全 0。如果 `n` 或 `size` 等于 0，`lw_calloc` 将返回一个可以被 `lw_free` 释放的非空指针；**如果 `n` 和 `size` 的乘积超过了 `size_t` 类型所能表示的最大范围，`lw_calloc` 将返回 `NULL`**。如果系统的可用内存不足，`lw_calloc` 也返回 `NULL`。
- `lw_realloc` 函数将 `ptr` 指向的内存块的大小更改为 `size` 字节。在从内存块的起始位置到旧大小和新大小中较小值范围内的内容将保持不变。如果新大小大于旧大小，则增加的内存不会被初始化。如果 `ptr` 为 `NULL`，那么该调用等同于 `lw_malloc(size)`；如果 `size` 等于零且 `ptr` 不为 `NULL`，那么该调用等同于 `lw_free(ptr)`。除非 `ptr` 为 `NULL`，否则它必须是先前通过 `lw_malloc`、`lw_calloc` 或 `lw_realloc` 调用返回的指针。如果指向的内存块被移动，则会执行一次 `lw_free(ptr)`。如果系统的可用内存不足，`lw_realloc` 将返回 `NULL`，并且不会移动或释放 `ptr` 所指向的内存块。

（内存池管理接口）

（可靠读写接口）

3 关键技术与细节

3.1 使用线程本地缓存提高并发性

加速大概率事件（make common case fast）

3.2 使用大小类系统控制内部碎片

3.3 使用键值存储分离用户数据和元数据

3.4 使用三模冗余实现可靠存储

我们的可靠读写模块 TMR 采用三模冗余（Triple Modular Redundancy）机制，将每次写操作复制到三组存储单元中，通过多数表决（Majority Voting）的机制实现纠错读。这种机制在单比特翻转发生时，能够有效保证数据的正确性。

4 工作流

4.1 大额分配、释放

4.2 小额分配、释放

4.3 可靠读写

4.4 内存池管理

5 实验评估与讨论

5.1 实验环境

- 机型 - Dell Precision 7820 Tower
- 处理器 - Intel(R) Xeon(R) Gold 6226R CPU @ 2.90GHz
- 物理线程数 - 32
- 内存 - 64 GB
- 系统架构 - x86_64
- 内核版本 - Linux 5.15.167.4-microsoft-standard-WSL2
- 操作系统 - Ubuntu 22.04.2 LTS

5.2 实验设计

表 1 展示了本次实验所采用的基准测试集。

名称	描述
linux scalability (modified) [2]	● 每个线程连续分配 10,000,000 个 16 字

	<p>节的对象，然后依次释放，释放的顺序与分配的顺序相同</p> <ul style="list-style-type: none"> ● 用途：测量最好情况（线程间相互独立）下分配器的吞吐量
threadtest [3]	<ul style="list-style-type: none"> ● 与 linux scalability 类似；不同之处在于线程的分配、释放模式 ● 每个线程重复 100 轮如下操作：100,000 次分配 + 100,000 次释放
larson [2], [4]	用途：模拟长时间运行的服务器进程

表 1 基准测试集

表 2 展示了本次实验设计所包含的分配器。

名称	描述
ptmalloc2 [1]	由 Wolfram Gloger 于 2006 年发布的一款经典内存分配器，支持多线程
glibc 2.35	实验环境所采用的 Ubuntu 操作系统自带的内存分配器，可以理解为 ptmalloc2 的优化版
jemalloc 5.3.0 [5]	jemalloc 开发团队于 2022 年发布的 release 版本，代表产业界的顶尖水平
TRMalloc	2021 年中国科学技术大学相同赛题参赛作品
LWMalloc	本文所评估的内存分配器

表 2 参与比较的分配器

5.3 实验结果

线程数	1	2	4	8	16	32	64	128
glibc 2.35	16.49	29.53	51.15	98.05	176.10	267.91	349.56	383.24
jemalloc	21.45	36.51	58.58	111.43	224.45	349.51	476.14	552.85
LWMalloc	20.18	38.95	72.15	115.26	200.58	305.82	396.53	435.85

表 3 运行 larson 基准测试的结果（指标：吞吐量；单位：百万次操作每秒）

运行 larson 基准测试的结果如表 3 所示。可以看到，当线程数小于等于 8 时，LWMalloc 的表现优异，其吞吐量甚至超过了 jemalloc。不过，随着线程数进一步增加，LWMalloc 在扩展性方面的劣势逐渐显现，并被 jemalloc 反超。

6 总结与展望

（理论分析 MT-safe 问题）

参考文献

- [1] G. Wolfram, *ptmalloc2*. (2006). C. Accessed: Dec. 28, 2024. [Online]. Available: <http://www.malloc.de/malloc/ptmalloc2-current.tar.gz>
- [2] D. Boreham, “Malloc() Performance in a Multithreaded Linux Environment,” in *2000 USENIX Annual Technical Conference (USENIX ATC 00)*, San Diego, CA: USENIX Association, Jun. 2000. [Online]. Available: <https://www.usenix.org/conference/2000-usenix-annual-technical-conference/malloc-performance-multithreaded-linux>
- [3] E. D. Berger, K. S. McKinley, R. D. Blumofe, and P. R. Wilson, “Hoard: A scalable memory allocator for multithreaded applications,” *ACM Sigplan Not.*, vol. 35, no. 11, pp. 117–128, 2000.
- [4] P.-Å. Larson and M. Krishnan, “Memory allocation for long-running server applications,” in *Proceedings of the 1st International Symposium on Memory Management*, in ISMM ’98. New York, NY, USA: Association for Computing Machinery, 1998, pp. 176–185. doi: 10.1145/286860.286880.
- [5] *jemalloc* 5.3.0. (2022). Accessed: Nov. 09, 2024. [Online]. Available: <https://github.com/jemalloc/jemalloc/releases/tag/5.3.0>