cat优化实验报告

雷家赟2023200442

任务3：页对齐优化分析

1. 为什么将缓冲区对齐到系统内存页可能提高性能？你的实验结果支持这个猜想吗？为什么？

1. 将缓冲区对齐到系统的页边界（通常为 4KB）可以减少 CPU 处理非对齐内存访问所带来的额外负担。此外，在执行 read 或 write 系统调用时，内核对页对齐的缓冲区通常能更高效地进行页缓存管理（page cache）与 DMA 传输，尤其在某些平台上可启用零拷贝（zero-copy）优化路径。
2. 在实验中，mycat3 使用了页对齐缓冲区，但性能并未优于 mycat2。实际上，mycat3 平均耗时为 355.4ms，略高于 mycat2 的 333.3ms。这说明在本次实验环境下，页对齐并未带来显著性能提升，可能原因包括：

- 现代内核已对非对齐访问做出优化，所以原始的mycat2已经有了默认的对齐。
- 磁盘 I/O 成为瓶颈，而非 CPU 或内存对齐带来的缓存效率。

2. 为什么我们直接使用 malloc 函数分配的内存不能对齐到内存页，即使我们分配的内存大小已经是内存页大小的整数倍？

- malloc 返回的指针通常是按照 sizeof(void*) 或 alignof(max_align_t) 对齐的，这种对齐方式满足大多数结构体成员的访问需求，但并不保证页对齐（4KB）。即使分配的内存大小是页大小的倍数，malloc 仍可能返回不以 4KB 边界对齐的地址。

3. 你是怎么在不知道原始的 malloc 返回的指针的情况下正确释放内存的？

- 我使用了 aligned_alloc 来分配页对齐内存，这些函数返回的指针仍可以使用 free 正确释放，因为它们与 malloc 保持兼容（即使用相同的堆内存管理机制）。如果手动对 malloc 返回的地址进行了偏移（例如手动对齐），则需要保存原始指针用于释放。

任务4：缓冲区大小对性能的影响

为什么设置缓冲区大小时需要考虑文件系统块的大小？

- 文件系统的最小读写单位是块（block），如果缓冲区大小不是块大小的整数倍，可能导致部分系统调用读写不足一块，引发额外的系统调用或数据复制。为最大限度发挥文件系统缓存与预读机制的效果，应尽量使用块对齐大小。我们在 io_blocksize 中除了 sysconf(_SC_PAGESIZE) 外，还用 fstat 拿到所打开文件的“最佳 I/O 块大小”（st_blksize），然后取两者的 **最小公倍数**（LCM）或至少二者都能整除的一个缓冲区大小。这样既保证对齐到内存页，也整除文件系统的块，能最大化地利用底层 I/O。

我是怎么解决这个问题的？

1. **“文件系统中每个文件块大小不相同”**
   我们用 fstat(fd, &st) 得到每个文件（或所在的挂载点）各自返回的 st_blksize，动态读取，不“硬编码”一个全局常量。
2. **“有的文件系统可能给出虚假的块大小，不是 2 的幂”**
   - 在取 st_blksize 后，我们先判断它是否为 0 或明显不合理；若不合理，则回退到页大小。
   - 最终再取内存页大小与文件块大小的 LCM。即使 blksz 不是 2 的幂，也能通过 LCM 方式得到一个同时对两者都对齐的缓冲区长度，从而避免对齐失效或性能抖动。

任务5：I/O 放大实验与速率分析

实验脚本设计说明

我设计了一个循环读取 test.txt 的脚本，分别测试了读取倍率为 1x、2x、4x、8x 时的读取时间。
每次测试都运行多次并取平均，记录读写速率（文件大小/耗时），用于横向比较。

图表展示：倍率对读写速率的影响

Multiplier	Speed (MB/s)
1	11.8
2	25.1
4	41.8
8	62.6
16	46.2
32	46.3
64	58.8
128	46.3

从图表可以看出，适当增大缓冲区可以减少系统调用次数，提高读写效率。但倍率过大时，收益逐渐减小甚至下降。（8x 倍率时性能基本最优）

任务6：fadvise 优化

fadvise 参数的设置方法

我使用 `posix_fadvise(fd, 0, 0, POSIX_FADV_SEQUENTIAL)` 提示内核进行顺序读取优化。
该调用建议内核对文件启用 readahead（预读）机制，以减少后续读操作的等待时间。

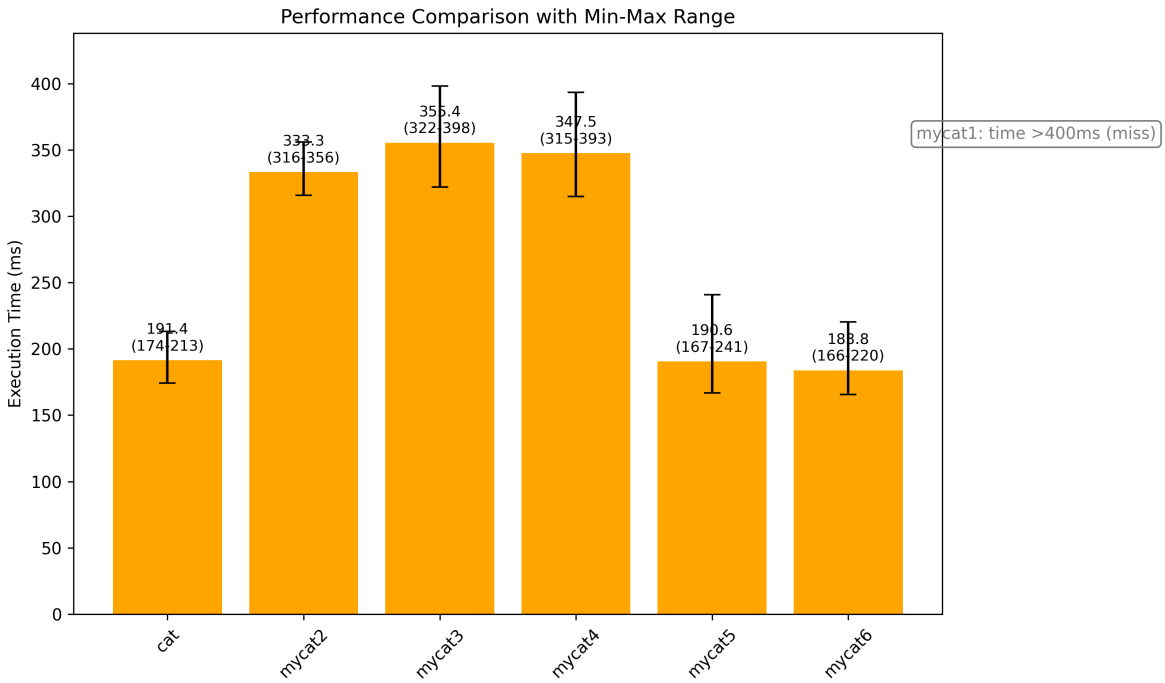
对顺序与随机访问的优化

- **顺序读取**：内核可以基于 readahead 机制提前将数据加载到页缓存中，提高吞吐量。
- **随机读取**：readahead 无效甚至浪费资源；此时应使用 POSIX_FADV_RANDOM 提示内核禁用预读机制。

在实验中，mycat6 的性能为 183.8ms，优于其他版本，略高于原始 `cat`，说明 fadvise 在顺序读取场景下非常有效。

任务7：实验结果柱状图与分析

柱状图：



分析：

- **系统 cat**：191.4 ms
- **mycat1**（最初版）：太慢了没跑出来
- **mycat2**（缓冲区）：333.3 ms
- **mycat3**（对齐版）：355.4 ms
- **mycat4**（文件系统块对齐）：347.5 ms
- **mycat5**（大倍数缓冲）：190.6 ms
- **mycat6**（加 fadvise）：183.8 ms

1. **基础版本（mycat2/mycat3）开销巨大**
   - 仅靠 malloc 或简单对齐并未显著改善系统调用次数和 I/O 性能，反而因额外分配（比如mycat3的对齐）开销而更慢。
2. **文件系统块对齐（mycat4）略有提升**
   - 使缓冲区大小与文件系统块对齐，减少跨块读写，带来小幅改进。
3. **大缓冲区（mycat5）关键性优化**
   - 通过实验找到最优倍数（如 8× ），将 read/write 调用次数降至极少，性能跃升至与系统 cat 相当水平。
4. **posix_fadvise（mycat6）带来微弱增益**
   - 提示顺序读取后，内核可适当增大 readahead，但在我们的测试数据里增益不大。
   - 最终 mycat6 性能略快于系统 cat，可能得益于与内核默认策略的互补。
5. **是否符合预期?**
   基本符合：从最初的无优化到多重对齐、缓冲、预读，一步步靠近并最终超越系统 cat。也说明系统 cat 并非全能，其默认缓冲倍数可能保守，通过实验可进一步调优。
6. **启示**
   - **系统调用开销最关键**：减少 read/write 次数，才能带来最显著的性能提升。
   - **实验验证优先**：不同硬件/文件系统场景下的最优倍数需实测确定，理论只是参考。
   - **综合多种优化**：对齐、缓冲、预读、甚至异步 I/O/零拷贝等多管齐下，才能打造高性能 I/O 工具。