MeowLab 实验报告

姓名：宋艾轩 学号：2023200405

实验目标

本实验旨在通过实现不同版本的 `mycat` 命令，探索系统IO优化技术，并分析各种优化方法对性能的影响。

实验环境

- 操作系统：Linux 6.6.87.1-microsoft-standard-WSL2
- 测试文件大小：2GB
- 测试工具：hyperfine

实验内容

1. 基础版本 (mycat1)

第一个版本实现了最基本的文件复制功能，使用逐字节读取的方式：

```
while ((n = read(fd, &c, 1)) > 0) {
    if (write(STDOUT_FILENO, &c, 1) != 1) {
        print_error("Error writing to stdout");
    }
}
```

这个版本性能最差，因为每次只读取一个字节，导致大量的系统调用。

2. 使用系统页大小作为缓冲区 (mycat2)

第二个版本使用系统页大小作为缓冲区大小：

```
size_t io_blocksize(void) {
    long page_size = sysconf(_SC_PAGESIZE);
    if (page_size == -1) {
        return 4096;
    }
    return (size_t)page_size;
}
```

通过使用系统页大小作为缓冲区，显著减少了系统调用次数，性能得到明显提升。

3. 使用页对齐内存 (mycat3)

第三个版本在mycat2的基础上，添加了内存页对齐：

```
void* align_alloc(size_t size) {
    size_t page_size = io_blocksize();
    size_t total_size = size + page_size + sizeof(void*);
    void* raw_ptr = malloc(total_size);
    if (raw_ptr == NULL) {
        return NULL;
    }
    void* aligned_ptr = (void*)(((uintptr_t)raw_ptr + page_size + sizeof(void*) - 1) & ~(page_s
    ((void**)aligned_ptr)[-1] = raw_ptr;
    return aligned_ptr;
}
```

4. 考虑文件系统块大小 (mycat4)

第四个版本在mycat3的基础上，考虑了文件系统的块大小：

```c
size_t io_blocksize(int fd) {
    long page_size = sysconf(_SC_PAGESIZE);
    if (page_size == -1) {
        page_size = 4096;
    }
    struct stat st;
    size_t blk_size = 4096;
    if (fstat(fd, &st) == 0 && st.st_blksize > 0) {
        blk_size = st.st_blksize;
    }
    return (page_size > blk_size) ? page_size : blk_size;
}
```

5. 使用更大的缓冲区 (mycat5)

第五个版本使用系统页大小的8倍作为缓冲区：

```c
static size_t io_blocksize(void) {
    long page_size = sysconf(_SC_PAGESIZE);
    if (page_size == -1) {
        return 4096;
    }
    return page_size * 8;
}
```

6. 添加fadvise优化 (mycat6)

第六个版本在mycat5的基础上，添加了fadvise系统调用来提示系统我们将进行顺序读取：

```c
if (posix_fadvise(fd, 0, 0, POSIX_FADV_SEQUENTIAL) != 0) {
    fprintf(stderr, "Warning: fadvise failed: %s\n", strerror(errno));
}
```

实验结果

性能数据

版本	总时间(ms)	用户时间(ms)	系统时间(ms)
mycat1	时间过长	-	-
mycat2	381.8 ± 2.7	40.1	341.2
mycat3	382.9 ± 2.7	50.4	332.2
mycat4	381.0 ± 2.3	35.8	345.0
mycat5	214.5 ± 2.3	7.9	206.4
mycat6	229.6 ± 2.0	6.1	206.5

结果分析

1. **性能改进分析**：
   - mycat1（逐字节读取）：性能最差，时间过长
   - mycat2（使用系统页大小作为缓冲区）：性能显著提升，约381.8ms
   - mycat3（使用页对齐内存）：性能与mycat2相近，约382.9ms
   - mycat4（考虑文件系统块大小）：性能与mycat2相近，约381.0ms
   - mycat5（使用更大的缓冲区）：性能显著提升，约214.5ms
   - mycat6（添加fadvise优化）：性能略有下降，约229.6ms
2. **关键发现**：
   - 系统时间占比很高（约90%），说明IO操作是主要瓶颈
   - 用户时间在mycat5和mycat6中显著降低，说明优化减少了CPU使用
   - 缓冲区大小对性能影响最大，而其他优化效果相对较小
3. **结果是否符合预期**：
   - **符合预期的部分**：
     - 增加缓冲区大小带来的显著性能提升
     - mycat1的性能最差
     - 系统时间远大于用户时间
   - **不符合预期的部分**：
     - 内存对齐和文件系统块大小优化带来的提升不明显
     - fadvise优化反而导致性能略有下降

实验启示

1. **简单优化优先**：
   - 增加缓冲区大小这种简单的优化往往比复杂的优化更有效
   - 不要过度追求复杂的优化方案
2. **实际测试验证**：
   - 性能优化需要实际测试验证，不能仅凭理论推测
   - 理论上的优化可能在实际环境中产生反效果
3. **关注主要瓶颈**：
   - 系统IO是文件操作的主要瓶颈
   - 应该优先考虑IO相关的优化
4. **避免过度优化**：
   - 过度优化可能会带来反效果
   - 需要谨慎使用系统调用
5. **实践建议**：
   - 在文件操作中，优先考虑使用合适的缓冲区大小
   - 系统调用虽然强大，但也要考虑其开销
   - 性能优化时要注意实际测试，不能仅依赖理论分析

结论

通过本实验，我们深入理解了系统IO优化的各种技术，并验证了它们的效果。实验结果表明，在系统编程中，有时候最简单的优化方案可能是最有效的。同时，任何优化都需要通过实际测试来验证其效果，不能仅凭理论推测。这也提醒我们在进行性能优化时要保持谨慎，避免过度优化带来的反效果。