从文本文件统计最高频单词

汇编语言实验报告

石曜铭 钱翰林

UCAS

2025 年 4 月 29 日

项目概述

- 需求分析：
  - 读取含英文/空格/标点的文本文件
  - 分割单词并统计频率
  - 输出最高频单词
- 一些问题：
  - 处理超大文件（超内存容量）：有时文件大小会超过缓冲区大小
  - 优化统计时间复杂度：暴力匹配是 $O(n^2)$ 的，怎样优化
  - 跨缓冲区单词完整性的处理

设计思路

分块读取策略：使用 10KB 大小的 buffer 对文件进行分块读取，分别处理每次读到 buffer 中的字符串。

数据管道设计：

- 主缓冲区 + 溢出缓冲区：使用 buffer + overflow_buf 分别存储每次读到的内容和最后一个有效字符连续段的内容
- 每次读 buffer 时先将上一轮中 overflow_buf 内的内容置于 buffer 的头部，拼接成完整的读入。

程序架构

- 文件处理层 - 系统调用封装
- 数据处理层 - 双缓冲区流水线
- 算法层 - Trie 树核心逻辑
- 内存管理层 - 堆分配控制

跨缓冲区处理机制

1. 逆向边界检测：从缓冲区尾部扫描，将最后一个有效字符连续段存到 Overflow_buf 中。
2. 溢出缓冲区管理
   - 100 字节环形缓冲区
   - 首尾相接验证

Trie 树的设计

- Trie 树上每个 Node 保存 26 个指向子结点的指针，整数类型 Count 表示节点作为字符串末尾出现的次数，char 指针保存该节点存的字符串。
- 具体地，我们为每个 Node 开辟 $8 \times 26 + 8 + 8 = 224$ 个 byte，在.bss 段开辟 100000 个 Node 节点（可以开更多），作为 Trie 树所用空间。

调试心得

在程序开发过程中，我们经历了数次棘手的调试挑战。我们通过 gdb 调试工具进行调试，设置断点，在重要的位置逐行运行，锁定 bug 根源所在。
以下列举一些在开发中发现的 bug：

- 在 process_loop 中，调用的 is_alpha 函数会将 rax 改为 0 或 1，但调用接下来的 to_lower 函数时没有将字符重新赋值给 rax；
- 在 trie_get_max 中的一个分支忘记将 rcx 自增，导致死循环；
- 在 memcpy 函数中，因对堆内存申请的不了解，写的代码实际上没有成功申请堆内存，导致 Segmentation fault；
- ……

经过长时间不眠不休的调试，我们终于得到了一个相对完整的健壮的程序。

项目总结

在本次大作业中，我们主要进行的设计有：

- 汇编级内存控制技术
- 跨缓冲区单词的处理方案
- 使用紧凑型 Trie 结构维护字典

未来优化方向

- 内存优化：为 Trie 树节点动态分配内存，而不是在.bss 段预加载
- 混合索引结构：
  - 使用 Patricia Trie 来维护，效率更高
  - 概率跳表加速
- 标准扩展：
  - Unicode 全字符支持
  - 分布式版本