Project 6: Performance Evaluation of File I/O Operations

1. Introduction

This report presents the findings to compare the performance between multi-threaded memory mapped I/O and standard file I/O with different read sizes.

2. Testing Environment

The tests were performed on my personal HP Pavilion Aero Laptop 13z-be200 running Ubuntu 6.5.0-1019-oem with a 6.5.0 kernel version. The detailed specifications of the testing machine are as follows:

- **Processor**: AMD Ryzen 7 7735U with Radeon Graphics
  - CPU Family: 25
  - Model: 68
  - Cores: 8 cores per socket
  - Threads per Core: 2
  - Total Available CPUs: 16
  - Maximum Clock Speed: 4819 MHz
- **Memory**:
  - L1 Cache: 256 KiB (8 instances for data, 8 for instructions)
  - L2 Cache: 4 MiB (8 instances)
  - L3 Cache: 16 MiB (1 instance)
- **Virtualization**: AMD-V support for efficient virtualized environments.
- **Architecture**: x86_64, supporting 32-bit and 64-bit operations with little endian byte order.
- **File Size for Testing**: using the file **test_input/affordable-health.txt** that is 2,807,573 bytes.
- **Development and Compilation Environment**: The project directory contains source files for the testing application (**proj6**), including C source and header files, a **Makefile** for compilation, scripts for running tests (**run_tests.sh**) with various different configurations (e.g. number of threads or read sizes), and parsing output data (**parse_output_data.sh**) into csv file for ease of visualization, along with a Python script (**visualization.py**) for generating performance graphs.
- **I/O Operations Testing**: Executed via the provided **doit** executable implemented from project 1.

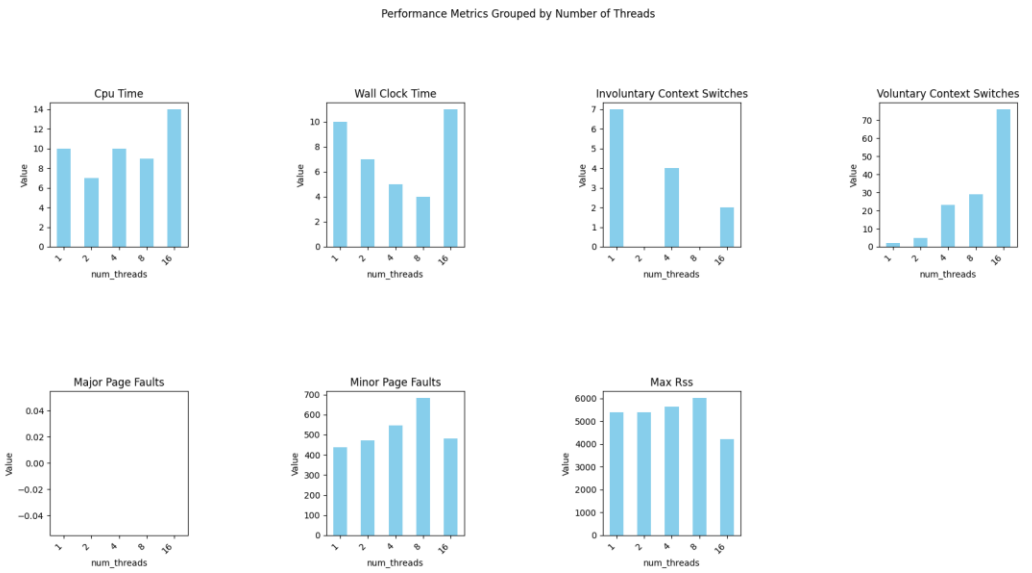
The system's multi-core and multi-threaded CPU setup is particularly relevant for assessing the performance of the program.

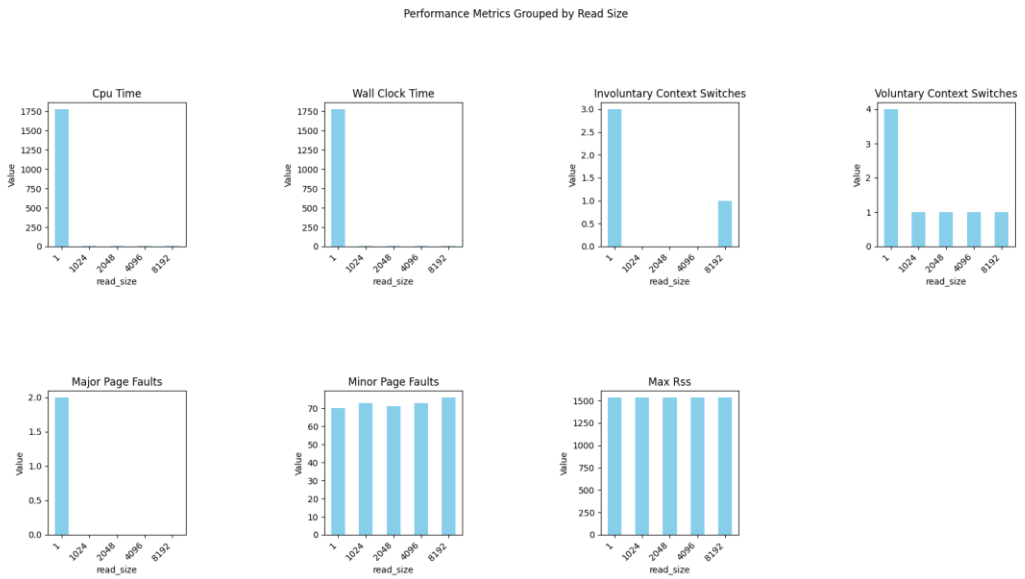
3. Methodology

Each configuration was tested on the same input file (i.e. test_input/affordable-health.txt), ensuring consistency across tests. The read sizes for standard file I/O were set to 1 byte, 1K, 4K, and 8K bytes. For memory-mapped I/O, the tests were single-threaded to focus on the efficiency of memory mapping itself. Subsequent tests for multithreading performance used 1, 2, 4, 8, and 16 threads.

The primary performance metrics assessed were CPU time used (user + system), elapsed wall-clock time, involuntary context switches, voluntary context switches, major page faults, minor page faults, and maximum resident set size.

4. Results and Discussion





From the tests conducted, it was observed that:

- **Major page faults:** Memory-Mapped I/O configurations showed no major page faults regardless of the number of threads used. So do most of the Standard I/O configurations. The only time where 2 major page faults happened was when we read the file one byte at a time.
- **Wall clock time:** the number of threads seem to correlate to the performance as performance improved up to a certain thresholds (e.g. 8 threads). However, the performance got worst at 16 threads, which could be due to context-switching overhead and CPU cache limitations. In addition, my implementation is probably not the most efficient since it needs dynamic allocation from the heap to store the detected ranges of strings. The read size had a notable impact on wall-clock time. Smaller read sizes resulted in longer operation times, while larger read sizes (4K and 8K bytes) performed better, aligning with the typical block size of the filesystem.
- **The CPU** time for multi-threaded memory-mapped I/O is much higher than standard I/O, especially at higher thread count (e.g. 16 threads) due to the number of context switches.

Conclusions

The testing indicates there is a performance benefit to using multiple threads, but only up to 8 threads. Beyond that, the overhead of context switching can negate the benefits of additional threads. In this testing environment, with only two processors, the optimal number of threads is closely aligned with the physical core count.