Matthias Simon, 1658438

Betriebssysteme, Übungsblatt 1, Winter 2025

Zur Ermittlung der Latenzen beim Lesen und Schreiben von Dateien (ORWC – Open Read Write Close) aus Sicht einer Anwendung wurde ein C-Programm entwickelt. Zur anschließenden Auswertung der Ergebnisse wurde Python verwendet.

Implementierung des Programms

Da die Wahl des Betriebssystems freigestellt war, wurde das Programm für Windows entwickelt. Daher wird die Windows-API für Dateioperationen und zur Zeitmessung verwendet.

Zeitmessung

```c
static inline uint64_t perf_freq_hz(void) {
    static uint64_t freq = 0;
    if (freq == 0) {
        LARGE_INTEGER f;
        QueryPerformanceFrequency(&f);
        freq = (uint64_t)f.QuadPart;
    }
    return freq;
}
static inline uint64_t now_us(void) {
    LARGE_INTEGER c;
    QueryPerformanceCounter(&c);
    // Umrechnung auf Mikrosekunden: (Zählerstand * 1.000.000) / Frequenz
    return (uint64_t)((c.QuadPart * 1000000ULL) / perf_freq_hz());
}
```

Zunächst wird eine Hilfsfunktion implementiert, die einmalig bei Programmstart die Frequenz des CPU-Zählers ermittelt.

Eine weitere Hilfsfunktion liefert den aktuellen Zeitstempel in Mikrosekunden (µs)

Programmstruktur und Ablauf

1. Initialisierung

Zu Beginn werden die Parameter des Programms festgelegt. Standardmäßig durchläuft das Programm 10000 Iterationen. Mit Hilfe eines Kommandozeilenarguments kann diese Anzahl jedoch angepasst werden. Anschließend wird der Pfad zur temporären Datei, in der die Dateioperationen stattfinden, sowie zur Ausgabedatei festgelegt, in der die Messergebnisse gespeichert werden. Darauffolgend wird ein Schreibpuffer mit dem Teststring „test“ definiert, der während des Experiments in die temporäre Datei geschrieben wird. Anschließend wird die CSV-Datei mit einer Kopfzeile vorbereitet.

2. Warm-Up-Phase

Um Messfehler durch den sogenannten „Kaltstart“-Effekt des Dateisystems zu vermeiden (z. B. beim erstmaligen Anlegen von Metadaten im Verzeichnis), wird ein Warm-Up durchgeführt. Dabei wird die temporäre Datei einmal erstellt und sofort wieder geschlossen. So wird sichergestellt, dass die anschließenden Messungen nicht verfälscht werden.

3. Messschleife

In der folgenden for-Schleife wird die Messung der Latenzen für die gegebene Anzahl an Iterationen durchgeführt. Zunächst wird die Latenz des Opens gemessen. Dazu werden zwei Zeitstempel gemessen – vor und nach dem Öffnen der Datei. Das Öffnen der Datei wird über *CreateFileA* realisiert, das mit dem Flag „OPEN_EXISTING“ die zuvor erstellt temporäre Date öffnet. Anschließend wird die Dauer als Zeitdifferenz vor und nach dem Öffnen gespeichert.

Darauf folgt die Latenzmessung der Write-Operation. Dazu wird in die zuvor geöffnete Datei das Teststring „test“ geschrieben. Dabei wird geprüft, ob alle Bytes korrekt geschrieben wurden. Im Fehlerfall wird eine Fehlermeldung ausgegeben. Die gemessen Zeit spiegelt dabei eher das Schreiben in den Dateisystem-Cache und nicht zwingend die tatsächliche Schreibgeschwindigkeit der Festplatte wider. Das liegt daran, dass moderne Betriebssysteme Schreibzugriffe zunächst im RAM puffern und erst später im Hintergrund auf die Festplatte übertragen.

Anschließend folgt die Messung der Leseoperation. Zunächst wird der Dateizeiger auf den Anfang der Datei zurückgesetzt, um jeden Lesevorgang konsistent vom Anfang der Datei zu starten. Anschließend beginnt die Zeitnahme. Wie bei der Schreiboperation wird überprüft, ob alle erwarteten Bytes korrekt gelesen wurden. Auch hier spiegelt die gemessene Zeit vor allem den Zugriff aus Anwendungssicht wider, da das Betriebssystem Lesezugriffe ebenfalls cachen kann und die Daten schon im RAM vorliegen.

Zum Schluss wird die Latenz des Schließens der Datei gemessen. Auch hier werden erneut zwei Zeitstempel erfasst – unmittelbar vor und direkt nach dem Aufruf von *CloseHandle.*

4. Speichern der Messergebnisse und Abschluss

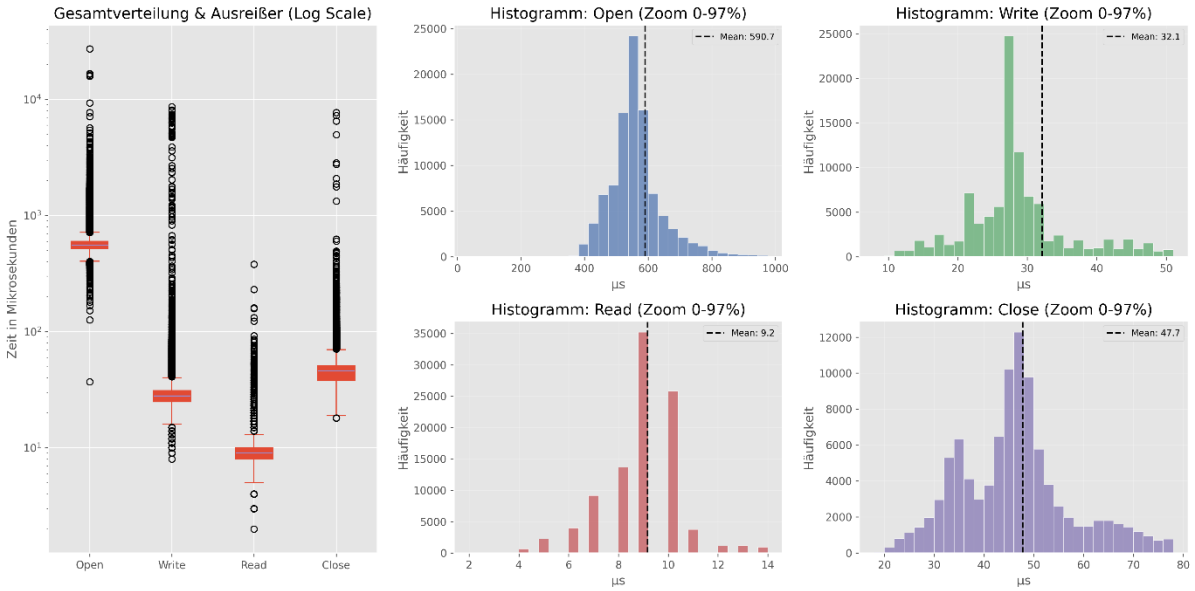
Zum Abschluss werden die Messergebnisse je Iteration in eine Zeile der CSV-Datei geschrieben. Nach Beendigung der Schleife wird die CSV-Datei geschlossen und die temporäre Datei wird mit *DeleteFile* entfernt.

Auswertung der Ergebnisse

```
=== Statistische Auswertung ===
  Operation   Mean  Median  Min    Max  StdDev  CI_95_Margin
0   open_us 590.66  556.00   37  27162  233.03          1.44
1  write_us  32.14   28.00    8   8652  116.38          0.72
2   read_us   9.17    9.00    2    381    3.15          0.02
3  close_us  47.74   46.00   18   7718   48.12          0.30
```

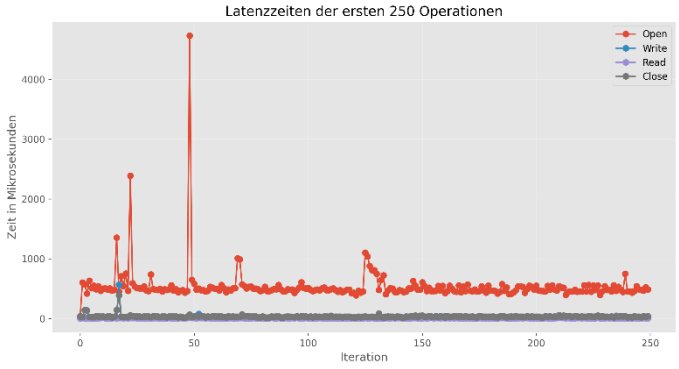
Die Auswertung der Ergebnisse erfolgt mit Hilfe eines Python-skripts. Zunächst werden die gängigen statistischen Größen für die Verweildauer im Kern ermittelt. Die durchschnittlichen Latenzen zeigen dabei eine klare Hierarchie der Kosten pro Operation. Die mit Abstand schnellsten Operation ist **Read** (9,17 µs), die mit einer Standardabweichung von 3,15 sehr stabil ist. Das **Schreiben** des Teststrings ist etwa 3x langsamer als das Lesen. Auffällig ist dabei die Standardabweichung von 116 und der im Vergleich sehr hohe maximale Wert. Das **Schließen** dauert etwas länger als das Schreiben, dauert aber deutlich kürzer als die **Open** Operation, die die teuerste Operation ist. Auch hier sind deutliche Ausreißer im maximalen Wert zu beobachten.

Matthias Simon, 1658438



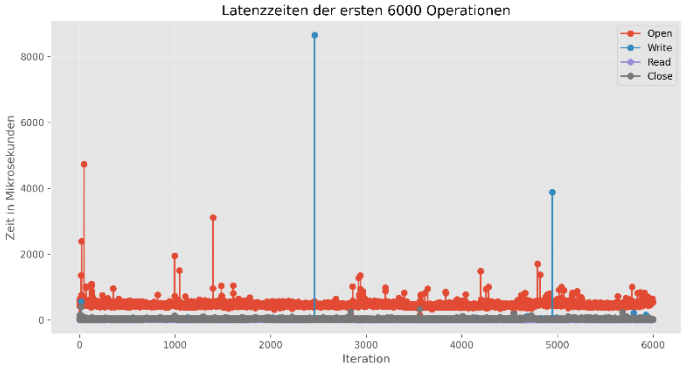
Die grafische Aufbereitung der Messungen unterteilt sich in eine globale Übersicht der Wertebereiche (Boxplot, links) und eine Betrachtung der Häufigkeitsverteilung der Messwerte (Histogramme, rechts). Das Boxplot nutzt eine logarithmische Skala auf der Y-Achse, die nötig ist, da die Werte in einem Werteberich von minimal 2 µs bis maximal 27162 µs liegen. Die roten Boxen, die die mittleren 50% der Daten repräsentieren, bestätigen die oben beschrieben Hierarchie der Operatoren. Die Kreise außerhalb der Boxen zeigen die Ausreißer, die vor allem bei Open, Write und Close ausgeprägt sind.

Die Darstellung in den Histogrammen wurde auf das 0-97% Perzentil beschränkt, damit die großen Ausreißer die Darstellung und Skala nicht verfälschen. Lässt man diese Ausreißer außer Betracht, zeigt sich, dass Open einer Normalverteilung annähert. Im Gegensatz dazu ist Write eher rechtsschief: Der höchste Peak liegt links neben dem Mittelwert. Da Read die kürzeste Dauer hat, stößt hier die Messung in µs an Grenzen und eine Messung in Nanosekunden wäre angebrachter, um die feineren Unterschiede besser deutlich zu machen.

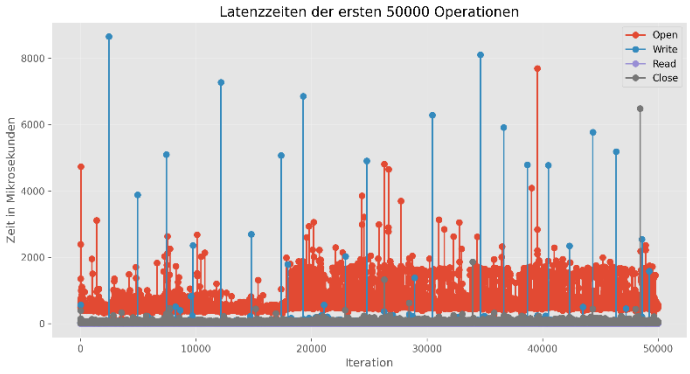


Ergänzend wurde der Verlauf der Latenzen über die ersten 250 Iterationen geplottet. Hier werden die ersten Ausreißer in Open bereits deutlich.

Matthias Simon, 1658438



Bei der Betrachtung der ersten 6000 Iterationen ist nach ca. 2500 Iterationen ein deutlicher Peak in der Write Operation sichtbar.



Bei der Betrachtung der ersten 50000 Iterationen wird sichtbar, dass die nach 2500 Iterationen sichtbare Spitze sich in regelmäßigen Abständen in verschiedener Ausprägung wiederholt. Zudem wird nach ca. 18000 Iterationen erkennbar, dass die Open-Operation länger dauert, was darauf hindeutet, dass ab diesem Zeitpunkt eine Drosselung greift, die für längere Zeiten sorgt.