Betriebssysteme
Übung 3

Optimistische Nebenläufigkeit mit ZFS-Snapshots

Als Betriebssystem wollte ich für diese Aufgabe wie zuvor bereits auch ein Ubuntu WSL auf Windows 11 verwendet. Leider funktioniert ZFS auf WSL2 leider nicht, weshalb ich eine VM aufsetzen musste.
Die VM ist eine Ubuntu 22.04 LTS die mit dem nativen Tool von Windows, dem Hyper-V-Manager.

Im Rahmen der Übung wurden zwei Projekte erstellt. Das erste Projekt ist die ZFS-Library und das zweite Projekt beinhaltet den Prototyp des Brainstorming-Tools und des Validierung-Tools.

Java-Bibliothek

Überblick über die Bibliothek

Die ZFS-Java-Bibliothek besteht aus drei Komponenten: FileOperation, TransactionManager und ZfsManager.

FileOperations

Die Klasse FileOperations bietet gundliegende Funktionen zum Manipulieren von Dateien. Sie ermöglicht Schreiben, Lesen und Löschen von Dateien.
Zusätzlich ist in dieser Klasse auch das Hashing der Dateien mit der Methode calculateHash umgesetzt. Mit dieser wird der SHA-256-Hash einer Datei berechnet, der für die Konflikterkennung innerhalb von Transaktionen genutzt wird. Dadurch kann kontrolliert werden, ob die Datei während einer Transaktion unkontrolliert geändert wurde.

ZfsManager

Der ZfsManager stellt die direkten Interaktionen mit ZFS bereit, indem die entsprechenden Linux-Befehle von Java ausgeführt werden.
Es ermöglicht Erstellung, Löschung und Rollback auf den letzten Snapshot. Beim Erstellen der Snapshots wird dieser ein Name entsprechend dem Zeitstempel in ms gegeben, um eindeutige Snapshots zu gewährleisten.
Zusätzlich ermöglicht executeCommand das ausführen beliebiger ZFS-Befehle und fängt mögliche dabei auftretende Fehler ab.

TransactionManager

Der TransactionManager handelt die Transaktionen (Anpassungen) auf die Dateien und kombiniert somit die Methode von FileOperations und ZfsManager.
Es besteht aus den folgenden Methoden:

- “startTransaction”: Die Methode startTransaction erstellt einen Snapshot des aktuellen Dateizustands und speichert den SHA-256-Hash der Zieldatei.
- “commitTransaction”: Mit commitTransaction werden Änderungen nur dann übernommen, wenn der aktuelle Hash mit dem initialen Hash übereinstimmt. Wird ein Konflikt erkannt, erfolgt ein Rollback.
- “rollbackTransaction”: Die Methode rollbackTransaction stellt die Datei auf den Zustand des zuletzt erstellten Snapshots zurück. Mit der Methode “getLastSnapshot” wird der letzte erstellte Snapshot ermittelt.
- “deleteSnapshotIfExists”: Mit deleteSnapshotIfExists können nicht mehr benötigte Snapshots entfernt werden.

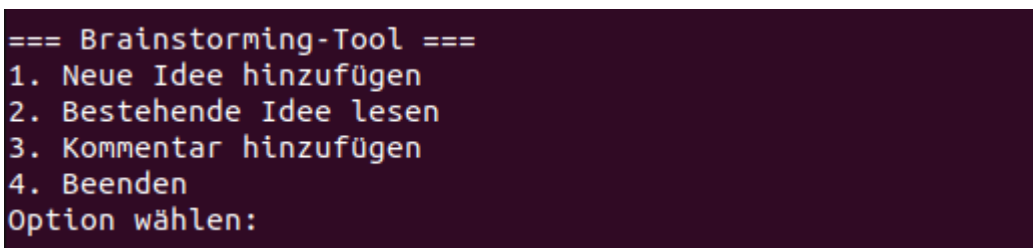
Somit werden mit dieser Klasse die grundlegenden Methoden zur Erstellung des BrainstormingTools umgesetzt.

BrainstormingTool

Das BrainstormingTool besteht aus zwei Klassen. Einmal dem IdeaManager und dem BrainstormingTool. Das BrainstormingTool verarbeitet alle Benutzerinteraktionen wohingegen der IdeaManager als Schnittstelle zwischen BrainstormingTool und der Java-Bibliothek dient.

Der IdeaManager speichert die Dateien/Ideen im Verzeichnis “ideas” und nutzt die Methoden der Java-Bibliothek um Kommentare für die Ideen zu ermöglichen.
Durch diese Auslagerung der Logik in eine extra Datei war es sehr einfach die Methoden zu nutzen um das ValidierungsTool zu erstellen und das BrainstormingTool zu testen.

Das BrainstromingTool gibt einem 4 verschiedene Auswahlmöglichkeiten, nach dem Öffnen:



Durch das tippen der Zahlen 1-4 wird dann die entsprechende Aktion ausgeführt.

Wählt man 1, so muss man anschließend den Namen der Idee erstellen und diese mit Inhalt füllen:

```
=== Brainstorming-Tool ===
1. Neue Idee hinzufügen
2. Bestehende Idee lesen
3. Kommentar hinzufügen
4. Beenden
Option wählen: 1
Name der Idee: TestIdee
Inhalt der Idee: Sollte man das nicht mal testen?
```

Wählt man 2, so kann man sich den Inhalt einer Idee anzeigen lassen, sofern man den entsprechenden Namen eingibt.
Wählt man 3, so erhält man die Möglichkeit einer zuvor erstellten Idee einen Kommentar hinzuzufügen.

Wählt man 4, dann wird das Programm beendet:

```
=== Brainstorming-Tool ===
1. Neue Idee hinzufügen
2. Bestehende Idee lesen
3. Kommentar hinzufügen
4. Beenden
Option wählen: 4
Programm beendet.
```

ValidierungsTool

Das ValidationTool wurde eingesetzt, um die Effizienz und Robustheit des gesamten Systems zu testen. Es simuliert verschiedene Zugriffsszenarien durch mehrere Threads, die zufällig Lese- und Schreiboperationen durchführen, und liefert entsprechende Metriken zur Systemleistung. Um mehr Komplikationen zu provozieren, werden alle Transaktionen auf einer Datei/Idee ausgeführt.
Dadurch kam es bei etwa 30% der Transaktionen zu Komplikationen.

Weitere Schwierigkeitenbei der Umsetzung

ZFS-Befehle

Das Ausführen der ZFS-Befehle führte zu mehr Problemen als gedacht, da die zunächst genutzte Methode executeZfsCommand des ProcessBuilder nicht funktionierte und keine ZFS-Befehle richtig ausgeführt werden konnten. Deswegen wurde später von mir mit Hilfe des executeCommand der gesamte, richtige ZFS-Befehl ausgeführt.

Reihenfolge der Hash-Funktionen

Ein weitere Problemquelle ist der Zeitpunkt gewesen, zu denen die Hash-Funktionen im Code ausgeführt werden. Ich hatte zunächst extra Funktionen hierfür eingebaut, jedoch schnell gemerkt, dass diese zu vielen Komplikationen führen würden, da durch die extra Methoden extra Zeit benötigt werden würde. Daher befinden sich die Kontroll-Hash-Funktion nun genau vor dem Schreiben in die Datei/Idee