Fallstudie

1. Einleitung

Bei der Nutzung von mehreren Threads kann ein weiteres Problem entstehen, wenn auf eine Datei gelesen und geschrieben werden soll. Dabei sollte die Situation des gleichzeitigen Lesen und Schreiben durch verschiedene Threads vermieden werden, da sonst unerwünschte Effekte entstehen können. Allerdings stellt das gleichzeitige Lesen durch mehrere Threads ohne einen Schreibzyklus kein Problem dar. Im Gegenzug sollte eine Datei nicht von mehreren Threads gleichzeitig geschrieben werden.

Besteht also die Sitation, dass ein Thread in eine Datei schreibt, sollten alle anderen lesenden **und** schreibenden Threads keinen Zugriff durchführen. Wenn jedoch der Schreibzyklus beendet ist, sollten **mehrere** Threads nun die möglichkeit haben, gleichzeitig auf die Datei lesen zu können oder ein **einzelner** Thread sollte die Möglichkeit haben, einen Schreibzugriff durchzuführen.

2. Adaptiver Semaphore

In dieser Fallstudie sollen Sie diese Problemstellung nachbilden und mit einem sogenannten **adaptiven Semaphore** lösen. Adaptive Semaphoren haben die Möglichkeit ihren Wert um einen bestimmten Wert zu vergrössern bzw. zu vermindern, anstelle wie bei den bisherigen Semaphor nur durch den Wert 1.

Erweitern Sie dazu Ihre selbst geschriebene Semaphore-Klasse. Verändern Sie dazu die Methode *p* und *v* wie folgt:

- Beide Methoden nehmen einen Integer-Wert auf. Dieser gibt an um wie viel sich der Semaphore-Wert verändern soll (reservieren / freigeben)
- Bei der *p*-Methode soll zuerst **dauerhaft** gefragt werden, ob der gewünschte Wert vom aktuellen Semaphore-Wert abgezogen werden kann. Ist dies der Fall, soll die Subtraktion durchgeführt werden. Ist dies nicht der Fall, soll der aktuelle Thread warten (wait) und die Abfrage bei einer Weiterführung nochmals durchführen.
- Bei der *v*-Methode kann direkt der Parameter-Wert zum Semaphore-Wert dazu gerechnet und wieder durch einen *notifyAll* alle Blockaden lösen werden.

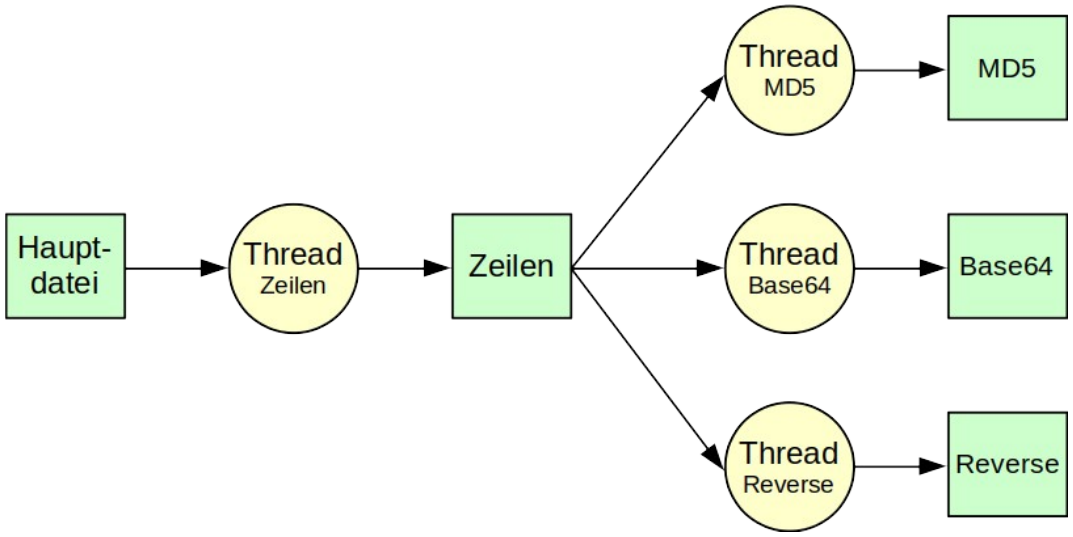
Testen Sie Ihre Implementierung:

1. Erstellen Sie einen Semaphore mit dem Initialwert 5.
2. Nun soll ein Thread 1 «Platz» reservieren (durchgeführt).
3. Ein weiterer Thread soll darauf 5 «Plätze» versuchen zu reservieren (blockiert).
4. Ein dritter Thread soll nun 4 «Plätze» reservieren (durchgeführt).
5. Nun soll der erste und der dritte Thread wieder ihre «Plätze» abgeben, wodurch der zweite Thread **erst jetzt** ausgeführt wird.

3. Problemstellung

Schreiben Sie ein Programm welches aus mehreren Threads besteht. Dabei soll ein Thread aus einer Hauptdatei absatzweise einen Text einlesen und diesen wortweise (ein Wort per Zeile) in eine Datei speichern. Der Inhalt in der Zeilen-Datei soll von 3 weiteren Threads gelesen und mit mit jeweiligem Algorithmus umgewandelt werden:

- Umwandlungsthread 1: Wort in MD5 umwandeln
- Umwandlungsthread 2: Wort in Base64 umwandeln
- Umwandlungsthread 3: Wort umkehren (reverse)



Als Hauptdatei steht ihnen die Text-Datei «bible_filtered.txt» zur Verfügung.

Um das Leser-Schreiber-Problem zu vermeiden, sollen Sie den adaptiven Semaphore verwenden. Dazu sollte man zunächst eine Zahl MAX bestimmen, die grösser oder gleich der maximalen Zahl von Lesern ist. Der Semaphor wird mit MAX initialisiert. Vor jedem Lesen wird der Semaphor um eins erniedrigt und nach dem Lesen um eins erhöht, während vor jedem Schreiben der Semaphor um MAX erniedrigt und nach dem Schreiben um MAX erhöht wird. Damit können alle Leser gleichzeitig zugreifen. Wenn aber ein Schreiber zugreift, kann kein anderer Leser und kein anderer Schreiber zugreifen. Ein Schreiber kann nur zugreifen, falls kein Leser aktiv ist. Solange ein Leser aktiv ist, können unbegrenzt lang weitere Leser ihren Datenzugriff beginnen und wartende Schreiber überholen.

Das Programm soll nach 5 Sekunden sich selber beenden.

Beachten Sie bei Ihrer Implementierung, dass die Umwandlungsthread einen flexiblen Aufbau aufweisen: Es soll mit wenig Aufwand möglich sein, weitere Umwandlungsthreads hinzuzufügen.

4. Lesen und Schreiben von Dateien

In Java können Sie mit den Klassen *FileReader*, *FileWriter* bzw. *BufferedReader* und *BufferedWriter* Dateien lesen und schreiben. Um eine Zeile aus einer Datei zu **lesen**, kann folgendes Beispiel betrachtet werden:

```
FileReader reader = new FileReader(input_path);
BufferedReader buffered_reader = new BufferedReader(reader);
String line = buffered_reader.readLine();
buffered_reader.close();

System.out.println(line);
```

Ein Beispiel für das **Schreiben** Zeichen in eine Datei sieht dabei wie folgt aus:

```
String line = "Text";

FileWriter writer = new FileWriter(output_path, true);
BufferedWriter buffered_writer = new BufferedWriter(writer);
buffered_writer.write(line);
buffered_writer.close();
```

5. MD5 und Base64

Für die Umwandlung von MD5 und Base64 können Sie folgende Quellen als Einstiegspunkt verwenden:

- https://www.baeldung.com/java-md5 (MD5 Using MessageDigest Class)
- https://www.baeldung.com/java-base64-encode-and-decode (Java 8 Basic Base64)

6. Ergebnis

Folgendes Ergebnis sollte bei diesem Programm erzeugt werden:

Zeilen	MD5	Base64	Reversed
In	EFEB369CCCBD560588A756610865664C	SW4=	nI
the	8FC42C6DDF9966DB3B09E84365034357	dGhl	eht
beginning	E3587F6620B552E78446D548A28392D9	YmVnaW5uaW5n	gninnigeb
God	AEB9573C09919D210512B643907E56B8	R29k	doG
created	E2FA538867C3830A859A5B17AB24644B	Y3JlYXRlZA==	detaerc
the	8FC42C6DDF9966DB3B09E84365034357	dGhl	eht
heaven	EB31870669F13FD8444C2BC918375F09	aGVhdmVu	nevaeh
and	BE5D5D37542D75F93A87094459F76678	YW5k	dna
the	8FC42C6DDF9966DB3B09E84365034357	dGhl	eht
earth.	1FD2D20C27AA7AF7EFF606B90F51A246	ZWFydGgu	.htrae
And	C33315685A0CBA3CE53BE378B3C7874B	QW5k	dnA
the	8FC42C6DDF9966DB3B09E84365034357	dGhl	eht
earth	852488DDD9570BC877783BF4397563E0	ZWFydGg=	htrae
was	A77B3598941CB803EAC0FCDAFE44FAC9	d2Fz	saw

7. Warnung: Zeilenumbruch Windows / Linux

Beachten Sie, dass Sie die Datei «bible_filtered.txt» nicht bearbeiten. Diese Datei besitzt für jede Zeile jeweils das Steuerzeichen «\n». Wenn Sie, vor allem unter Windows, diese Datei bearbeiten (auch aus Versehen), können zusätzliche Steuerzeichen, wie «\r» eingebaut werden. Die Abtrennung von neuen Zeilen kann dadurch erschwert werden. Versuchen Sie daher nur mit der Grunddatei zu arbeiten. Im Zweifelsfalle empfiehlt es sich, die Datei nochmals neu zu beziehen.