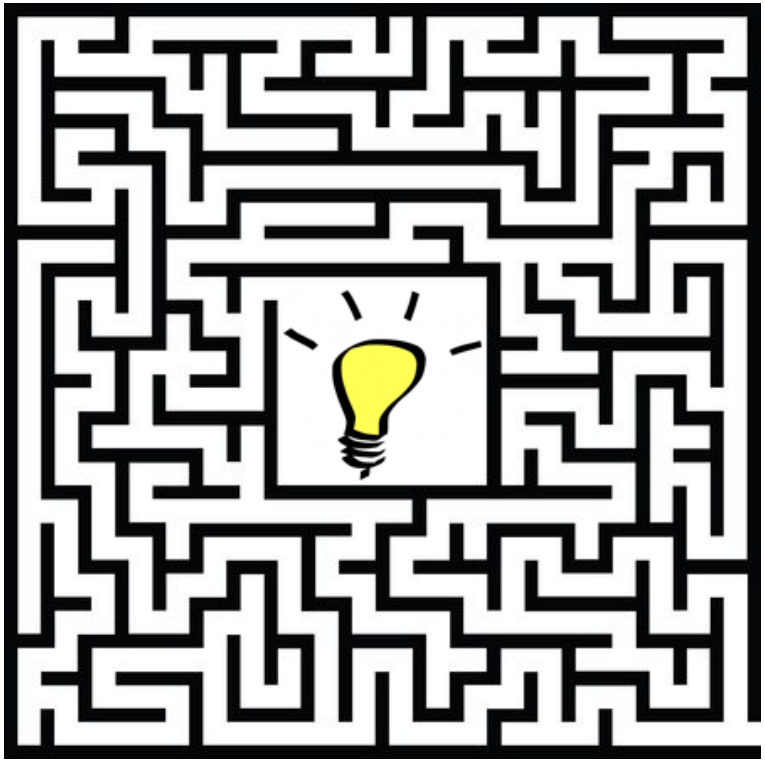
Prof. Dr. Henning Pagnia

Advanced IT
(Übungsaufgaben)



Herbst 2023

DHBW Mannheim

Aufgabe 1

Aufgabe 1 (Bearbeitungszeit: max. 10 min)

(a) Welche Aufgaben erfüllt der Befehl `cat`?

(b) Erzeugen Sie mittels `cat` zunächst zwei kleine Dateien mit jeweils 4 Zeilen und anschließend eine dritte als Zusammenschluss der ersten beiden Dateien. Dabei sollen fortlaufende Zeilennummern vorangestellt werden!

Aufgabe 2

Aufgabe 2 (Bearbeitungszeit: max. 10 min)

(a) Geben Sie unter Verwendung des Befehls `echo` den folgenden Text exakt wie angegeben auf dem Bildschirm aus: (Tipp: 'man bash' und dort nach 'echo' suchen)

```
Dear friend!



    Welcome!
```

(b) Überlegen Sie, welche Ausgabe die folgende Befehlszeile erzeugt:

```
echo >testf; echo >>testf; wc testf
```

Aufgabe 3

Aufgabe 3 (Bearbeitungszeit: max. 10 min)

Sortieren Sie die Zeilen einer Datei, so dass gleiche Zeilen in der Ausgabedatei nur ein einziges Mal vorkommen, wobei Groß- / Kleinschreibung ignoriert werden soll.

Beispieldatei `Namen` zum Testen:

```
Mueller fehlt
Maier fehlt
Schmidt fehlt
Maier fehlt
Mueller fehlt
Schmidt fehlt
maier fehlt
Schmidt fehlt
Mueller fehlt
schmidt      fehlt
mueller FehLt
```

Die korrekte Ausgabe ist:

```
Maier fehlt
Mueller fehlt
schmidt      fehlt
Schmidt fehlt
```

Aufgabe 4

Aufgabe 4 (Bearbeitungszeit: max. 15 min)

Finden Sie die am häufigsten auftretende Zeile einer Datei (ohne Berücksichtigung der Groß- / Kleinschreibung)!
Tipp: Verwenden Sie die Tools `sort`, `uniq` und `head`

Aufgabe 5

Aufgabe 5 (Bearbeitungszeit: max. 15 min)

(a) Schreiben Sie ein Shell-Script `mydel`, das eine angegebene Datei löscht, jedoch zuvor eine Sicherungskopie in Ihrem Verzeichnis `$HOME/Papierkorb/` anlegt.
Achten Sie darauf, dass auch sog. *Wildcards* ('*', '?') korrekt verarbeitet und auch Verzeichnisse gelöscht werden können!
Tipp: Sie können im Shell-Script die auf der Kommandozeile übergebenen Parameter referenzieren:
$1 bezeichnet den ersten, $2 den zweiten, ..., $@ alle Parameter.

(b) Erstellen Sie in Ihrem Home-Verzeichnis ein Verzeichnis `bin/` und kopieren Sie Ihr Script dorthin in die Datei `rm`. Verändern Sie Ihren Datei-Suchpfad so, dass bei der Eingabe des Kommandos `rm` Ihre Datei aufgerufen wird. Wie können Sie nun den Papierkorb leeren?

Aufgabe 6

Aufgabe 6 (Bearbeitungszeit: max. 20 min)

Erstellen Sie ein Script, das solange von der Tastatur einen Namen einliest, bis ein 'q' oder ein 'quit' eingegeben wird. Für jeden Namen aus der nachstehenden Liste soll die passende Note angezeigt werden. Für alle anderen Namen soll '5.0' ausgegeben werden.

carla	1.0	uta	4.0
hans	1.0	georg	4.0
berta	4.0		

Aufgabe 7

Aufgabe 7 (Bearbeitungszeit: max. 20 min)

Schreiben Sie ein `awk`-Programm, das zunächst vom Benutzer zwei Wörter einliest und dann in einer Eingabedatei alle Vorkommen des ersten Wortes durch das zweite ersetzt. Geben Sie das Ergebnis auf der Standardausgabe aus!

Tipp: Mit `getline` *`str`* `< "-"` können Sie eine Eingabe von der Tastatur in die Variable *str* einlesen.

Aufgabe 8

Aufgabe 8 (Bearbeitungszeit: max. 30 min)

Bitte ändern Sie die Klasse *MyThread1* von Folie 31 so, dass der *main*-Thread unmittelbar nach Beendigung aller Threads die Nachricht "Alle fertig." auf dem Bildschirm ausgibt. Achten Sie dabei darauf, dass auch weiterhin alle Threads nebenläufig ablaufen!

Aufgabe 9

Aufgabe 9 (Bearbeitungszeit: max. 90 min)

(a) Schreiben Sie ein Java-Programm, das ein Array der Größe 2 097 152 ($= 2^{21}$) mit Integer-Werten > 0 füllt. Bilden Sie dann die Gesamtsumme und geben sie diese aus.
Messen Sie die Zeit in Millisekunden, die das Programm nur für die Summenbildung benötigt hat.

(b) Wiederholen Sie das Vorgehen mit 2, 4, 8, 16, 32, 64 und 128 Worker-Threads, die jeweils die Summe über einen eigenen Teil des Arrays bilden und an den Haupt-Thread zurückgeben und messen Sie ebenfalls jeweils die für die Summenbildung benötigte Zeit.

> Hinweis: Übergeben Sie eine fortlaufende ID an die Worker, so dass jeder Worker berechnen kann, welchen Ausschnitt des Arrays er bearbeiten muss.

(c) Diskutieren Sie die Ergebnisse!

Aufgabe 10

Aufgabe 10 (Bearbeitungszeit: max. 120 min)

Beim nebenläufigen Zugriff auf gemeinsam genutzte Datenstrukturen kann es zu Inkonsistenzen kommen. In dieser Aufgabe sollen Sie diese einmal am "lebenden System" beobachten.

(a) Implementieren Sie in JAVA eine Klasse *FifoQueue* als eine einfach verkette Liste, in der String-Objekte verwaltet werden können. Definieren Sie hierzu die beiden Methoden *put(String s)* sowie *String get()*.

(b) Erzeugen Sie zwei Threads, die parallel auf der Queue arbeiten, und zeigen Sie, dass dabei durch die nebenläufigen Zugriffe von Zeit zu Zeit Fehler auftreten.
(Hinweis: Verwenden Sie ggf. *Thread.yield()*-Aufrufe um eine Thread-Umschaltung zu forcieren.)

(c) Sichern Sie die Methoden mittels eines Semaphors ab, und zeigen Sie, dass nun alles korrekt ausgeführt wird.

Aufgabe 11

Aufgabe 11 (Bearbeitungszeit: max. 90 min)

5-Philosophenproblem (Dijkstra 1971)

Fünf Philosophen sitzen gemeinsam an einem runden Tisch, an dem sie *unabhängig voneinander* von Zeit zu Zeit Spaghetti essen. Jeder Philosoph hat rechts neben seinem Teller nur eine Gabel, benötigt aber zum Essen zwei Gabeln, also auch die seines linken Nachbarn. Aus diesem Grund können nicht alle Philosophen gleichzeitig essen, sondern sie müssen sich bezüglich ihrer kritischen Abschnitte *Essen* synchronisieren.

(a) Formulieren Sie die Synchronisationsbedingungen mittels eines Synchronisationsgraphen.

(b) Geben Sie Java-Lösungen unter Verwendung des Semaphorkonzepts an, wobei Sie die Philosophen jeweils als Threads implementieren sollten.

(i) Entwerfen Sie zunächst eine Lösung, bei der die Gabeln als Betriebsmittel aufgefasst werden.

(ii) Entwerfen Sie eine zweite Lösung unter Verwendung des Konzepts der privaten Semaphore. Gehen Sie davon aus, dass ein Philosoph zyklisch die Zustände *denkend*, *hungrig*, *essend* durchlebt.

Beurteilen Sie Ihre Lösungen jeweils anhand der Kriterien Verklemmungsfreiheit, Aushungerungsfreiheit und erreichbarer Parallelitätsgrad.

Aufgabe 12

Aufgabe 12 (Bearbeitungszeit: max. 90 min)

Lösen Sie das 5-Philosophenproblem mit dem Java-Monitorkonzept.

Tipp: Richten Sie für die Philosophen und den Monitor jeweils eine eigene Klasse ein.
Achten Sie insbesondere darauf, dass alle Philosophen denselben Monitor verwenden!

Aufgabe 13

Aufgabe 13 (Bearbeitungszeit: max. 60 min)

(a) Schreiben Sie ein einfaches UDP-basiertes Chat-Programm. Das Programm soll aus jeweils einem Client und einem Server bestehen. Jeder Client liest von der Tastatur eine Eingabezeile in Form eines Strings ein, der dann sofort zum Server gesendet wird. Der Server wartet auf Port 4999 und empfängt die Meldungen beliebiger Clients, die er dann – zusammen mit der IP-Adresse des Absenders – auf dem Bildschirm ausgibt.

(b) Starten Sie den Server auf Port 4998 und senden Sie Client-seitig **Broadcasts** in das lokale Netz.

Lösungsvorschlag:

Aufgabe 14

Aufgabe 14 (Bearbeitungszeit: max. 150 min)

In dieser Aufgabe soll ein einfacher File-Server für Textdateien entwickelt werden.

Vereinfachend gehen wir davon aus, dass dem Server ein festes Basisverzeichnis zugeordnet ist, in dem sich alle verwalteten Dateien befinden und dass er die notwendigen Zugriffsrechte besitzt. Die Textdateien sind dabei zeilenweise organisiert und beginnen mit Zeilennummer 1.

(a) Der Server soll auf Port 5999 Aufträge in Form von Strings mit "**READ filename,line_no**" entgegennehmen, wobei **line_no** eine positive ganze Zahl sein muss. Daraufhin wird vom Server die Datei **filename** geöffnet, die Zeile **line_no** ausgelesen und zurückgeschickt. Falls sich im Basisverzeichnis des Servers keine solche Datei befindet oder keine entsprechende Zeile vorhanden ist, soll an den Client eine Fehlermeldung zurückgesendet werden.

Implementieren Sie den Server sowie einen kleinen Test-Client (**keine Broadcasts senden!**). Verwenden Sie Java und UDP.

(b) Erweitern Sie den Server so, dass er auch das Kommando "**WRITE filename,line_no,data**" versteht, bei dem die Zeile **line_no** durch **data** ersetzt werden soll.

◇ **Tipps:**

(1) Implementieren Sie die Methoden `read` und `write` für die Verarbeitung der Dateizugriffe und lagern Sie diese in eine eigene Klasse aus (z. B. `class MyFile`)

(2) Verwenden Sie die folgenden Konstruktionen für die Dateizugriffe:

```
BufferedReader fileIn = new BufferedReader (
                            new FileReader(filename) );
...
String zeile = fileIn.readLine();
```

bzw.

```
PrintWriter fileOut =  new PrintWriter (
                           new FileWriter(filename) );
...
fileOut.println(data);
```

Lösungsvorschlag:

Aufgabe 15

Aufgabe 15 (Bearbeitungszeit: max. 90 min)

In dieser Aufgabe soll unser bisheriger File-Server verbessert werden.

Modifizieren Sie Ihre bisherige Implementierung des File-Servers, sodass der Server den *Thread-per-Request* Ansatz (ohne Synchronisation) umsetzt!

Zeigen Sie durch ein geeignetes Szenario (und durch das Einfügen von sleep-Aufrufen an kritischen Stellen), dass der Server parallele Requests ausführt! Was passiert mit Ihren Dateien bei parallelen Schreibzugriffen?

Lösungsvorschlag:

Aufgabe 16

Aufgabe 16 (Bearbeitungszeit: max. 30 min)

Skizzieren Sie, wie der File-Server mit den folgenden Implementierungsvarianten aussehen würde:

(a) *Thread-per-File*

(b) *Worker-Pool*

Lösungsvorschlag: