SoSe 2017

Nichtsequentielle und verteilte Programmierung

7. Übungsblatt

Prof. Dr. Margarita Esponda

Ziel: Auseinandersetzung mit Scheduling-Algorithmen.

1. Aufgabe (insgesamt 10 P.)

Nehmen Sie an, es wird ein Thread mit 100 Minuten CPU-Ausführungszeit und 100 Threads mit jeweils einer Minute CPU-Ausführungszeit gleichzeitig gestartet.

a) (8 P.) Berechnen Sie die durchschnittliche Verarbeitungszeit und Wartezeit für die Threads mit folgenden Scheduling-Algorithmen:

a) nicht-präemptive FCFS
b) nicht-präemptive SJF
c) präemptive Shortest Remaining Time First. Quantum = 1 Minute
d) RR-Scheduling. Quantum = 1 Minute

b) (2 P.) Für welche Art von Anwendungen sollten die verschiedene Algorithmen bevorzugt werden? Begründen Sie Ihre Antwort.

2. Aufgabe (3 P.)

Welche der folgenden Scheduling-Algorithmen können dazu führen, dass Threads verhungern? Begründen Sie Ihre Antwort.

a) First-come, first-served
b) Shortest job first
c) Round robin
d) O(1) früheres Linux-Scheduling
e) Hybrid Lottery Scheduling
f) Completely Fair Scheduling

3. Aufgabe (8 P.)

Beschreiben Sie mit einem Bild den Abarbeitungsverlauf folgender Task-Tupels:

T1 = (0, 2, 7, 7)
T2 = (0, 3, 8, 9)
T3 = (0, 1, 9, 11)
T4 = (0, 1, 6, 6)

in einem Echtzeit-Betriebssystem mit folgenden Scheduling-Algorithmen:

a) RMS mit T2 = (0, 3, 9, 9) und T3 = (0, 1, 11, 11)
b) DMS
c) EDF
d) LLF

4. Aufgabe (3 P.)

Ein weiches Echtzeitsystem muss zwei Sprachverbindungen ausführen, die beide alle 5 ms laufen und 1 ms der zugeteilten CPU-Zeit verbrauchen, und ausserdem ein Video mit 25 Bildern pro Sekunde abspielen, wobei jedes Bild 20 ms CPU-Zeit benötigt. Ist dieses System mit einem Echtzeit-Scheduler realisierbar? Begründen Sie Ihre Antwort.

Aus den folgenden zwei Programmieraufgaben (5. und 6.) sollen Sie eine wählen und abgeben, aber wenn sie beide bearbeiten, werden die gewonnenen Punkte entsprechend als Bonuspunkte angerechnet.

5. Aufgabe (12 P.)

Simulieren Sie den **RMS**-Scheduling Algorithmus.

g) Definieren Sie zuerst eine **Task**-Klasse mit mindestens folgendem Konstruktor:

```
public Task( int release, int execution, int deadline, int period );
```

h) Die **Task**-Klasse soll dazu noch Eigenschaften wie **priority** und **id** beinhalten, die für den **RMS**-Algorithmus und für die Ausgabe wichtig sind.

i) Programmieren Sie eine **RealTimeScheduler**-Klasse, die in ihrem Konstruktor eine Liste von Tasks und die Simulationszeit als Parameter bekommt. In der **RealTimeScheduler**-Klasse sollen als erstes die Prioritäten der Tasks gesetzt werden und dann eine Simulation gestartet werden, die nach jeder Zeiteinheit den Task mit der höchsten Priorität um eine Einheit abarbeitet und alle anderen Tasks nach hinten verschiebt.

j) Die Simulation soll die Unterbrechungen und die verpassten Deadlines ausgeben.

k) (2 zusätzliche Bonuspunkte) Programmieren Sie mit Hilfe einer **Thread**-Klasse und der **Thinker**-Klasse eine Visualisierung des Abarbeitung-Verlauf.

6. Aufgabe (12 P.)

Vor kurzem sind Sie auf folgenden Link aufmerksam geworden:

http://q1k.de/uni_tutor/alp4/

Der Zugriff auf diese URL ist durch einen einfachen **.htaccess** Mechanismus geschützt. Der Verantwortliche war beim Einrichten des Webservers leider nicht sehr sorgfältig und hat die **.htpasswd** Datei (in der sich alle Nutzer und deren Passwort als Hashwert befinden) aus versehen als Textdatei auf den Webserver geladen (**http://q1k.de/uni_tutor/htpasswd.txt**).

Sie wissen, dass die Domain einem Alp4 Tutor gehört und befürchten, dass jemand Unbefugtes eindringen und sich einen unfairen Vorteil für die laufende Vorlesung verschaffen könnte. Um auf Nummer sicher zu gehen, möchten Sie vorsichtshalber selbst einen Blick auf die Daten werfen.

Damit Sie Zugriff auf die Daten erhalten, müssen Sie zu jedem Hashwert das entsprechende Klartext Passwort finden. Da sich Ihre Tutoren in der Vergangenheit als wenig fantasievoll erwiesen haben, vermuten Sie, dass sich die ursprünglichen Passwörter aus ein bis zwei Worten zusammensetzen (klein, zusammengeschrieben, englisch). Eine Liste, bestehend aus den 20 Tausend häufigsten Wörtern der englischen Sprache, sollte Sie daher schnell ans Ziel bringen.

e) (0 bis 2 zusätzliche Bonuspunkte) Importieren Sie das Framework „MD5 Cracking" in eine IDE Ihrer Wahl und lesen Sie die dazugehörige Dokumentation -ODER- implementieren Sie

eine eigene Benutzerschnittstelle zum Lösen der Aufgabe. Diese sollte mindestens folgende Funktionalitäten anbieten:

- Hinzufügen von neuen Hashwerten
- Auswählen einer Wörterliste
- Messen der Zeit vom Startzeitpunkt bis zu dem Zeitpunkt an dem alle Passwörter gefunden wurden

Für Lösungen mit grafischer Oberfläche werden zusätzliche Bonuspunkte vergeben.

f) (2 P.) Die Hashwerte werden mit dem Message-Digset Algorithm 5 (MD5) berechnet. Betrachten Sie folgende Java-Funktion, die für eine Zeichenfolge den dazugehörigen MD5 Hashwert berechnet.

(Quelle: http://stackoverflow.com/a/6565597)

Erläutern Sie den Zweck der Zeilen 40, 42 und 43 sowie 45. Veranschaulichen Sie die Funktionsweise der Zeilen 42 und 43 anhand eines kurzen Beispiels.

```
37 public String getMD5(String password) {
38     try {
39         java.security.MessageDigest md = java.security.MessageDigest.getInstance("MD5");
40         byte[] array = md.digest(password.getBytes());
41         StringBuffer sb = new StringBuffer();
42         for (int i = 0; i < array.length; ++i) {
43             sb.append(Integer.toHexString((array[i] & 0xFF) | 0x100).substring(1, 3));
44         }
45         return sb.toString();
46     } catch (java.security.NoSuchAlgorithmException e) {
47         e.printStackTrace();
48     }
49     return null;
50 }
```

g) (2 P.) Downloaden Sie die im KVV bereitgestellten Textdateien wordlist20 sowie easyHashes. Allen Hash-Werten aus der easyHashes.txt liegen je zwei Wörter aus der Wortliste zu Grunde, wobei das erste Wort unter den ersten 5.000 Wörtern zu finden ist.

Implementieren Sie ein naives Verfahren zum Cracken der Passwörter. Ihre Lösung sollte single- threaded arbeiten, keinerlei Optimierungen besitzen und das HashCracker Interface implementieren.

h) (5 P.) Entwickeln Sie nun ein optimiertes Verfahren zum Cracken von Passwörtern. Dazu sollen mehrere Threads verwendet werden. Außerdem ist auch das in Aufgabe b) gewonnene Wissen für die Optimierung sehr von Nutzen. (Weiterhin gilt: Alle Passwörter setzen sich aus je zwei Wörtern der Wortliste (wordlist20) zusammen.)

Folgende Kriterien sollten dabei erfüllt sein: Alle Threads des Systems werden bis zum Ende der Ausführung beansprucht; Informationen bzgl. Länge der Wörterliste, Anzahl zu knackender Hashwerte, Anzahl der vom System bereitgestellten Threads sind NICHT hard-coded; sobald alle Hashwerte gefunden wurden oder es zu einem Abbruch kommt, rechnet das System nicht unnötig lang weiter; zum Programmstart sind nur die originale Wörterliste sowie Hashwerte gegeben.

(2 P.) Listen Sie alle vorgenommen Optimierungen zusammen mit einer kurzen Erklärung auf und erläutern Sie, wie Sie die Arbeit auf mehrere Threads verteilt haben.

(1 P.) Verschaffen Sie sich Zugriff auf den Webserver (aufpassen – nicht alle Logins funktionieren). Listen Sie die zum Cracken benötigte Zeit, Ihre Systemspezifikationen sowie alle Benutzer zusammen mit dem jeweiligem Klartextpasswort auf.

Entwickeln Sie die schnellste Lösung des Jahrgangs!

Folgende Regeln sind einzuhalten:

- Die Lösung implementiert das HashCracker Interface
- Der Code ist ausschließlich in Java programmiert
- Nur in Java 8 bereits enthaltene Bibliotheken wurden genutzt

(Falls Sie einen Trick entdecken und sich nicht sicher sind, ob es nach den Regeln erlaubt ist, besprechen Sie es bitte vorher mit ihrem Tutor)

Möchten Sie am Wettbewerb teilnehmen, so treten Sie bitte im Vorfeld gegen mindestens eine andere (ambitionierten) Übungsgruppe an. Sollten Sie sich dabei durchsetzen, senden Sie Ihrem Tutor bitte am (am != bis) Freitag nach der Abgabe eine Mail mit folgendem Inhalt:

- Ihre HashCracker Klasse mit der Sie antreten (als .java-Datei)
- Die Namen aller Studierenden, die an der Implementierung beteiligt waren
- Die Namen aller Studierenden, die Sie im Vorfeld geschlagen haben (Gerne mit kurzer Stellungsname zum stattgefundenen „Duell")

Die anschließenden Tests finden auf einem der Windows Poolrechner statt. Verwendet wird die große Wortliste (wordlist58).

Allgemeine Wichtige Hinweise zur Übungsabgabe:

1. Es muss der Quellcode hochgeladen werden (.java-Dateien), nicht der kompilierte Bytecode (.class-Dateien).
2. Zusätzlich zur Online-Abgabe muss der Quellcode vollständig ausgedruckt werden. Der ausgedruckte Quellcode muss lesbar sein, insbesondere ist auf automatische Zeilenumbrüche zu achten.
3. Für die gute Verständlichkeit des Quellcodes sind sinnvolle Bezeichnernamen zu wählen und der Code ausreichend zu kommentieren.
4. Das Programm muss Testläufe enthalten, die die Aufgabe vollständig abdecken. Das heißt, zum Testen sollen keine Änderungen am Code durch die Tutorin/den Tutor mehr notwendig sein. Die Testläufe müssen in einer eigenen Klasse Main enthalten sein.
5. Das Programm muss mit Java 8 kompatibel sein.
6. Alle Übungsaufgaben, die nicht Programmieraufgaben sind, müssen ebenfalls digital und nicht handschriftlich sowohl online als auch ausgedruckt zu dem im KVV angegebenen Datum (in der Regel Dienstag um 11:55) abgegeben werden. Zu spät abgegebene Lösungen werden mit 0 Punkten bewertet.