Aufgabenblatt 6

Aufgabe - Multi-Threading und Synchronisation

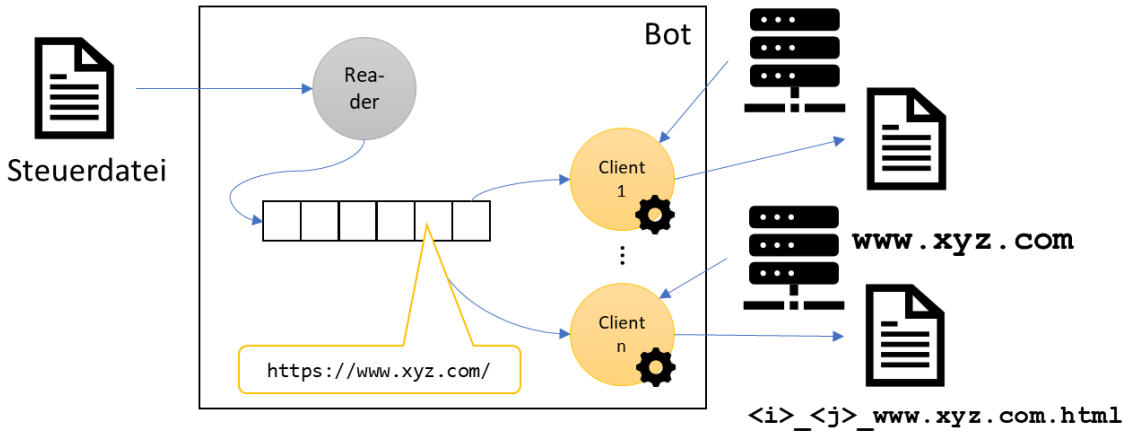
Systemkonzepte zum synchronisierten Zugriff sind nicht nur im Zusammenhang mit Betriebssystemen wichtig, sondern auch in der Anwendungsprogrammierung. Sie sollen in dieser Aufgabe ein C/C++-Programm (einen sog. *Bot*) entwickeln, das aus einer Steuerdatei URLs ausliest. Die Server im Netz sollen kontaktiert, die Seiten per Http **`GET`** Request ausgelesen und lokal gespeichert werden.

Aufgabenstellung:

Die Steuerdatei listet zeilenweise URLs auf:

```
https://www.google.de/
https://www.bing.com/
```

Leerzeilen und fehlerhafte Einträge werden ignoriert.



Der Name dieser Steuerdatei kann variiert werden und soll Ihrem Programm über die Kommandozeile als Parameter übergeben werden. Ansonsten arbeitet das Programm wie folgt:

- Das Auslesen der Seiten soll durch **mehrere Threads** geschehen:
  - Ein Thread (*Reader-Thread*) liest die Zeilen aus einer Steuerdatei und schreibt einen entsprechenden Job in eine **Queue** mit einer (zur Compilezeit) konfigurierbaren Größe.
  - Eine konfigurierbare **Anzahl von Threads** (*Client-Threads*) liest die Jobs aus der Queue und kontaktiert die jeweiligen Web-Server.
- Die Zugriffe auf die Queue müssen **synchronisiert**, d. h. gegeneinander geschützt erfolgen.
- Ein Client-Thread bekommt bei seiner Erstellung als Parameter eine Instanznummer zugewiesen, die ihn z. B. bei Debug-Ausgaben eindeutig identifiziert.
- Die Antworten der Web-Server werden in Dateien gespeichert. Die Dateinamen sollen durchnummeriert werden: wurden *n* Threads konfiguriert und sind *m* Einträge in der Steuerdatei vorhanden, so sollen die Antworten jeweils in einer Datei

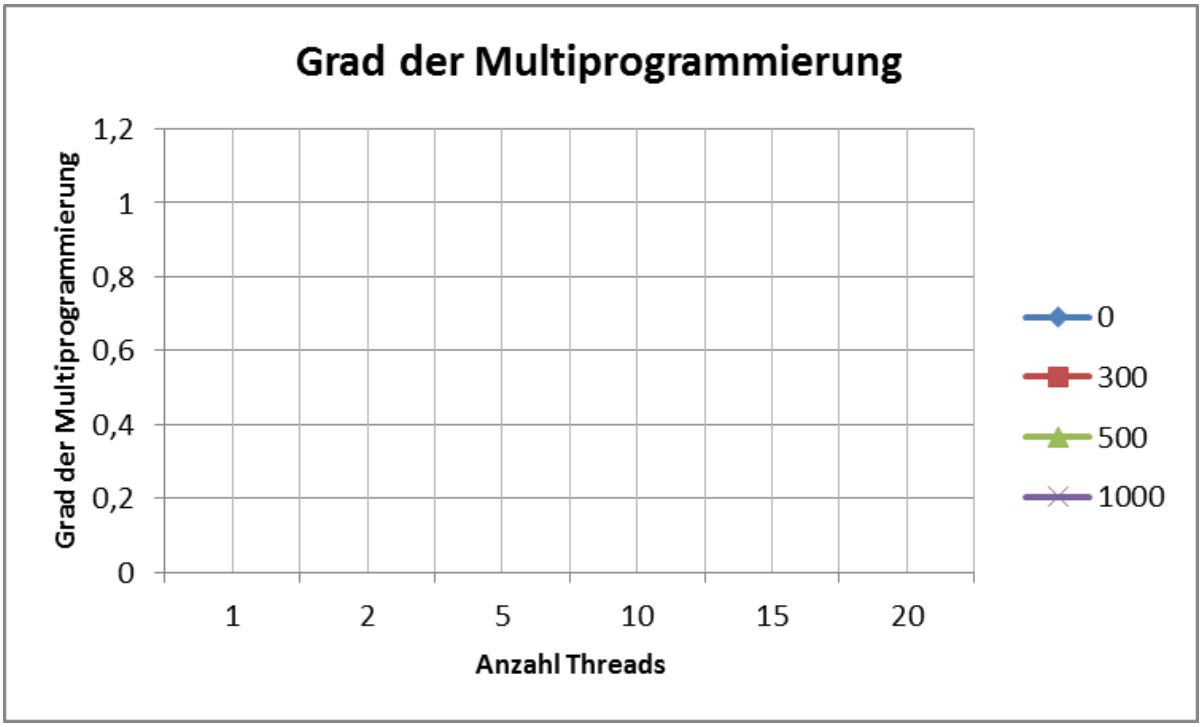
`<i>_<j>_<server host>.html` gespeichert werden (für Threads mit Index $1 \leq i \leq n$ und Dateien mit Index $1 \leq j \leq m$).

Bestimmen Sie nun die **Laufzeiten** des Bots für die Steuerdatei bezüglich unterschiedlicher Konstellationen. Um sinnvolle Ergebnisse zu bekommen sollen die Zugriffe auf die Seiten künstlich um einen einstellbaren Zeitoffset verzögert werden können. Dies ist über eine im Rahmen des Praktikums mitgelieferte Bibliothek möglich (entsprechende Erläuterungen werden dort gegeben).

Messen Sie für das Protokoll jeweils die Zeit zur Abarbeitung der URLs in der Steuerdatei, wobei Sie die Anzahl der Reader-Threads und Zugriffsverzögerung gemäß folgender Tabelle variieren (die Queue-Länge für die Verwaltung der Reader-Threads soll konstant auf 10 eingestellt werden):

	Proxy Verzögerung [ms]			
#Threads	0	300	500	1000
1				
2				
5				
10				
15				
20				

Aus dieser Tabelle kann nun der Grad der Multi-Programmierung bestimmt werden: man bildet dazu für eine feste Verzögerung das Verhältnis der minimalen Laufzeit zur jeweils betrachteten Laufzeit (für die minimale Laufzeit ist der Wert also 100%, sonst kleiner) und trägt dieses Verhältnis über die Anzahl der Threads auf. Zeichnen Sie für die jeweiligen Verzögerungen die Funktionsgraphen. Wie interpretieren diese?



Hinweise:

- Eigentlich müsste man die CPU-Auslastung messen, die den maximal erreichbaren Grad der Multi-Programmierung und damit den höchstmöglichen Durchsatz vorgibt: bei 100% CPU-Auslastung kann die Parallelität offensichtlich nicht mehr erhöht werden. Eine solche Messung ist aber nicht einfach möglich, denn es dürfte streng genommen nur der Bot auf dem System laufen, also nicht einmal das Betriebssystem und die zugehörigen Basis-Prozesse. Deshalb findet hier eine Approximation statt: sei $T(t_d , n)$ die Laufzeit bei einer Verzögerung t_d und bei einer Anzahl von n Threads. Dann wird die minimal benötigte Zeit für das Laden aller Seiten $T_{min} (t_d) = min \{ T(t_d , n), n = 1,\ldots,\infty\}$ zugrundegelegt und der Kehrwert mit dem maximal erreichbaren Durchsatz gleichgesetzt. Die Auslastung $A(t_d , n)$ ist dann definiert durch das Verhältnis: $A(t_d , n) = T_{min} (t_d) / T(t_d , n)$.
- Laufzeiten kann man durch Auslesen der Systemuhr beim Start und beim Ende der Ausführung des Bots messen. Einen Zeitstempel mit Millisekunden-Auflösung [ms] erstellt man per:

```
struct timeval  tv;
gettimeofday(&tv, NULL);
double start = (tv.tv_sec) * 1000 + (tv.tv_usec) / 1000;
```

- Führen Sie die Zeitmessungen anhand der fertigen Lösungen auf den Poolrechnern durch, um zu vergleichbaren Ergebnissen zu kommen.
- Die Funktion zum Zugriff zum Auslesen und Speichern von URLs finden Sie im Verzeichnis **`Web_Request`**. In dem Projekt sehen Sie auch, wie Sie die benötigten Bibliotheken zu dem zu entwickelnden Programm hinzubinden (**`-lm -lssl -lcrypto -pthread`**).
- Mit dem o. g. Projekt wird eine Steuerdatei mit einigen URLs zur Verfügung gestellt, die Sie vor Beginn Ihrer Untersuchungen einmal einzeln abrufen sollten. Es kann nicht ausgeschlossen werden, dass einzelne URLs temporär nicht zur Verfügung stehen. Modifizieren Sie gegebenenfalls diese Datei im Hinblick auf alternative Server.

Für das Testat ist ein **Protokoll** (Formatierung: siehe Blatt 1) des erstellten Programms (denken Sie an hinreichende Kommentierung) inklusive der durchgeführten Tests und Antworten auf die gestellten Fragen vorzulegen.