Betriebssysteme
bs@ibr.cs.tu-bs.de
Wintersemester 2025/26

Technische Universität Braunschweig

Übungsblatt 5: Synchronisation

Abgabetermin: Donnerstag, 15.01.2026 (23:59 Uhr)

Allgemeine Hinweise zu den BS Übungen

- Die Aufgaben sind alleine zu bearbeiten, allerdings ist Partnerarbeit (maximal zu zweit) möglich. Beide Partner müssen aber in ihrem Git-Repository abgeben und eine `PARTNER`-Datei mit dem Name des Partners im Repository anlegen.
- Jeder Teilnehmende muss dabei in der Lage sein, die gesamte Lösung zu erläutern.
- Zur Versionsverwaltung wird jedem Teilnehmenden ein Git-Repository zur Verfügung gestellt, welches auch zur Abgabe verwendet wird. Der Zugriff auf das Repository erfolgt über das IBR GitLab unter `https://gitlab.ibr.cs.tu-bs.de`.
- Die Abgabe ist spätestens zur Deadline durch ein `git push` zu tätigen. Es sollte sichergestellt werden, dass die Änderungen vorher auch durch ein `git commit` gesichert wurden.
- Eine Aufgabe kann beliebig oft abgegeben werden; es gilt die letzte *rechtzeitige* Abgabe (der letzte rechtzeitige Commit auf dem `main` Branch).
- Die abgegebenen Programme werden auf Ähnlichkeit mit anderen Programmen desselben Semesters und früherer Semester überprüft. Bei starken Übereinstimmungen behalten wir uns ein Ausschluss vom Übungsbetrieb und eine Meldung an den Prüfungsausschuss vor.
- Die Lösungen der Programmieraufgaben müssen im GitLab CI/CD lauffähig sein. Zusätzlich kann das System `x1.ibr.cs.tu-bs.de` zum Testen genutzt werden. Auf diesem können Sie sich per SSH mit ihrem IBR Account anmelden.
- Für die Bearbeitung kann auf einem beliebigen Linux-System gearbeitet werden. Allerdings empfehlen wir das frühzeitige Testen der Abgabe durch ein `git push` und die Inspektion der Test Pipeline im GitLab.
- Die Programmieraufgaben werden automatisch durch das bereitgestellte Testsystem per GitLab CI/CD ausgewertet. Es müssen mindestens $50\,\%$ der verfügbaren Testcase-Punkte erreicht werden.
- Die Testcase-Punkte können kontinuierlich über die Ausgabe des `run-tests` Jobs aus der GitLab CI/CD Pipeline eingesehen werden. Gewertet wird der letzte Pipeline-Status auf dem von uns zur Abgabe angelegten Abgabebranch (z.B. `abgabe1`), überprüfen Sie diesen gegebenenfalls.
- Sollten Sie $50\,\%$ der Punkte erreicht haben werden Sie ggf. für ein Code-Interview ausgewählt. Sollten Sie zum Code-Interview nicht erscheinen oder ihre Lösung nicht ausreichend erklären können wird der Zettel als „ungenügend“ bewertet.
- **KI Richtline**: Der Einsatz von KI-gestützten Tools ist bei der Bearbeitung der Hausaufgaben streng untersagt, da die eigenständige Erarbeitung der Lösungen inhärentes Lernziel ist. Es ist zu beachten, dass wir alle Abgaben als Prüfungsdokumente archivieren und es auch bei einer nachträglichen Prüfung zu einer rückwirkenden Aberkennung der Studienleistung kommen kann.

Aufgabe 1: Parallel Triangle Counter

Schreiben Sie ein Programm `patric` in der Datei `patric.c`, welches eine einfache Aufgabenverteilung von Arbeitspaketen an Arbeiterthreads implementiert. Ziel der Aufgabe ist es, die Verteilung auf die Arbeiterthreads und die Synchronisierung der Ausgabe zu implementieren. Als Arbeitspakete werden dem Programm Eckpunkte von Dreiecken auf der Standardeingabe zur Verfügung gestellt. Die Arbeiterthreads sollen die Anzahl der Punkte auf *ganzzahligen Koordinaten* zählen, die innerhalb des Dreiecks (*interior points*, im Beispielbild •) oder auf dessen Begrenzungslinien liegen (*boundary points*, ×).

Das Programm wird wie folgt aufgerufen: `patric <Maximale Anzahl der Arbeiterthreads>`

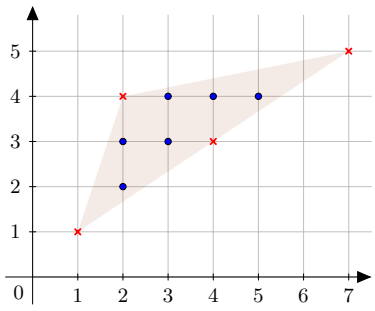

Abbildung 1: Dreieck mit den Koordinaten *(1,1),(2,4),(7,5)* und seinen *interior-* & *boundary points*.

(a) **Der Hauptthread:** Einlesen der Arbeitspakete.

Auf der Standardeingabe werden zeilenweise Dreiecke zur Verarbeitung angegeben. Eine Eingabezeile besteht aus drei, durch Komma getrennte Koordinatenpaaren. Ein Koordinatenpaar wird im Format *(x,y)* angegeben; bei x und y handelt es sich jeweils um Ganzzahlen.

Zeilen, die nicht dem erwarteten Format entsprechen, sollen unter Ausgabe einer Warnung verworfen werden und die Ausführung des Programms fortgeführt werden. Zur Vereinfachung dürfen Sie annehmen, dass valide Eingabezeilen keine Leerzeichen enthalten.

Die Eingabezeile *(1,1),(2,4),(7,5)* symbolisiert das in obenstehender Beispielgrafik dargestellte Dreieck und soll zu einer Erhöhung der gefundenen Punkte um 4 boundary points, bzw. 6 interior points führen.

(b) **Die Arbeitsthreads:** Abarbeiten der Arbeitspakete.

Für jede Eingabezeile soll ein Arbeiterthread erzeugt werden (**pthread_create(3)**), der das entsprechende Dreieck verarbeitet. Die Arbeitsthreads sollen jeweils losgelöst (detached) arbeiten, sodasss ihre Resourcen automatisch nach Beendigung freigeräumt werden. Die maximale Anzahl der gleichzeitig existierenden Arbeiterthreads soll auf die auf der Kommandozeile angegebene Zahl begrenzt werden. Bei einer darüber hinausgehenden Anzahl an Arbeitspaketen soll **passiv** gewartet werden, bis diese Grenze wieder unterschritten ist, bevor neue Arbeiterthreads erzeugt werden. Scheitert das Erzeugen eines Arbeiters, beendet sich das Programm mit Fehlermeldung. Die Anzahlen der gefundenen Punkte sollen über alle Dreiecke hinweg aufsummiert werden. Nutzen Sie die Funktion `countPoints` **aus dem vorgegebenen Modul** `triangle` (`triangle.c`, `triangle.h`) zum Zählen der Punkte.

Die Funktionsweise des Moduls ist in der zugehörigen Headerdatei beschrieben.

(c) **Der Ausgabethread:** Statusausgabe

Während der Ausführung soll das Programm folgende Informationen in einer Zeile ausgeben:

- Anzahl der insgesamt gefundenen Punkte, die auf Dreiecken liegen (*boundary points*).
- Anzahl der insgesamt gefundenen Punkte, die innerhalb von Dreiecken liegen (*interior points*).
- Anzahl der momentan aktiven Arbeiterthreads (d. h. Threads, die gerade `countPoints` ausführen).
- Anzahl der Threads, die die Bearbeitung ihres Arbeitspaketes abgeschlossen haben.

Die Ausgabe soll in einem getrennten Kontrollfluss (*Ausgabethread*) stattfinden und dabei kontinuierlich aktualisiert werden. Die Aktualisierung soll immer angestoßen werden, wenn sich ein Datum geändert hat. Zwischenzeitlich wartet der Ausgabethread **passiv**. Verwenden Sie folgendes Ausgabeformat:

```
"Found %d boundary and %d interior points, %d active threads, %d finished threads"
```

(d) **Synchronisierung**.

Bei Zugriffen auf globale Datenstrukturen und Koordination zwischen Threads muss auf geeignete Synchronisierung geachtet werden. Nutzen Sie hierfür Semaphoren (**sem_overview(7)**). Warten eines Threads muss jeweils passiv geschehen. Auch hierzu sind Semaphoren zu verwenden. Die notwendige Signalisierung von Ereignissen ist ebenfalls mittels Semaphoren durchzuführen.

Fehlerbehandlung

- Für jeden Funktionsaufruf, der fehlschlagen kann, muss eine Fehlerbehandlung geschehen.
- Es soll eine Fehlerausgabe mittels **perror(3)** getätigt werden.
- Das Programm soll sich mit Fehlerstatus (**exit(3)**) beenden.

Implementierungshinweise:

- Beginnen Sie die Implementierung mit einer Singlethread Variante. Starten Sie also unabhängig von der geforderten Zahl an Arbeitsthreads stets nur einen zum Berechnen. Erst wenn diese Logik wie erwartet funktioniert, starten Sie weitere Arbeitsthreads.
- Die Threads müssen `detached` laufen, damit sie nach Beendigung automatisch aufgeräumt werden.
- Der Cursor kann durch Ausgabe des Zeichens `\r` an den Anfang der Zeile zurückgesetzt werden.
- Orientieren Sie sich am Ausgabeformat der Referenzimplementierung.
- Langsame (I/O) Funktionen (z. B. **printf(3)**) dürfen **nicht** in kritischen Abschnitten ausgeführt werden.
- Die Anzahl der Zeilen einer Eingabedatei passt stets in einen `int`.

Hinweise zur Aufgabe

- Hilfreiche *Manual-Pages*: **scanf(3)**, **fflush(3)**, **sem_overview(7)**, **sem_init(3)**, **sem_wait(3)**, **sem_post(3)**, **pthread_create(3)**, **pthread_detach(3)**.
- Eine Referenzimplementierung und einfache Testdaten gibt es in `/ibr/courses/ws2526/bs/blatt5/`.
- Zum Übersetzen des Programmes ist das zusätzliche Compiler-Flag `-pthread` notwendig.
- Unterprogramme und globale Variablendefinitionen sind ausreichend zu kommentieren. Achten Sie bitte außerdem auf saubere Gliederung des Quellcodes!