Praktikum Betriebssysteme – Übung 4.1

Aufgabe 1 – Verständnisfragen

a) Warum wurden „Leichtgewichtige Prozesse“ (Threads) eingeführt?

b) Wodurch unterscheiden sich Threads von Prozessen?

c) Welcher Nachteil besteht bei federgewichtigen Threads im Vergleich zu leichtgewichtigen Threads?

Aufgabe 2 – pthreads

Bisher haben wir uns Prozesse angesehen, die einen eigenständigen Adressraum besitzen und eigene Betriebsmittel (z.B. offene Dateien) nutzen. Es ist daher nur durch zusätzlichen Aufwand möglich, dass mehrere Prozesse kooperativ arbeiten (z.B. durch den Austausch von Zwischenergebnisse per IPC).

Dagegen nutzen Threads die Ressourcen (Adressraum, geöffnete Dateien, usw.) eines Prozesses (also eines laufenden Programms) gemeinsam. Verschiedene Threads können also auf die gemeinsamen Resourcen zugreifen. Sie können beispielsweise eine Referenz auf eine Variable (z.B. ein Array) an die Threads ihres Programms übergeben und jeden Thread einen Teilbereich bearbeiten lassen. Dabei sind Sie jedoch selbst dafür verantwortlich, die Threads so arbeiten zu lassen, dass sie sich nicht gegenseitig „stören“ (z.B. gegenseitig Ergebnisse überschreiben).

a) Übersetzen Sie das Beispielprogramm (→ gitlab) und führen Sie es aus. Das Programm erzeugt mehrere Threads, gibt deren Thread-ID und eine laufende Nummer aus, wartet auf Beendigung der Threads und beendet sich dann selbst.

b) Das Programm aus Aufgabe a besitzt einen Designfehler im Bezug auf die Datenstruktur auf dem Heap-Speicher (dies hat nichts mit Threads zutun). Erkennen Sie wo das Problem ist und wie man es lösen würde?

c) Modifizieren Sie das Programm aus Aufgabe a so, dass die Threads für einige Zeit schlafen (z.B. 200 Sekunden). Sehen Sie sich anschließend die Prozessliste an. Finden Sie ihren Prozess und seine Threads? Wie können Sie die Threads in der Prozessliste anzeigen lassen?

d) Modifizieren Sie das Programm weiter. Rufen Sie im Hauptprogramm fork() auf, während die Threads schlafen. Was passiert?

Aufgabe 3 – Parallele Berechnung von π mit Threads

Um den Effekt von Threads sichtbar zu machen, benötigen wir eine Berechnung, die durch parallele Ausführung sichtbar beschleunigt wird, mathematisch aber nicht zu kompliziert für ein einfaches Beispiel ist und nicht zu viel Overhead erzeugt. Die Berechnung der Kreiszahl π bietet sich hier an.

Es gibt eine ganze Reihe verschiedener Möglichkeiten, die Kreiszahl π zu berechnen. Eine Möglichkeit basiert auf einer einfachen Integralformel und ist somit gut für unser Beispiel geeignet.

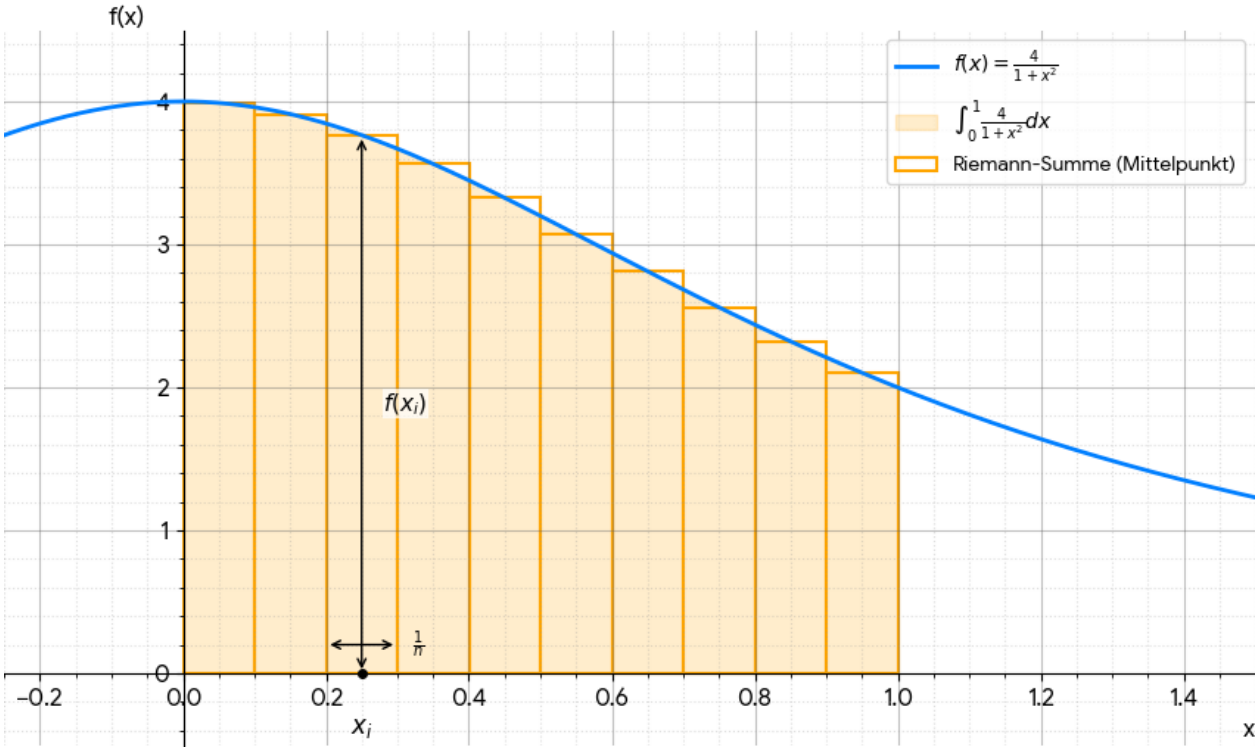
$$\pi = \int_0^1 \frac{4}{1+x^2} dx$$

Mathematischer Hintergrund

Die Stammfunktion von $f(x) = \frac{4}{1+x^2}$ lautet $F(x) = 4 \cdot \arctan(x)$

Somit gilt für die Fläche unter der Kurve: $\int_0^1 \frac{4}{1+x^2} dx = 4 \cdot (\arctan(1) - \arctan(0)) = 4 \cdot \frac{\pi}{4} = \pi$

Mit Hilfe des Riemannschen Integral können wir uns an diese Fläche annähern. Dazu teilen wir die x-Achse im Intervall [0,1] in n Quadrate und berechnen die Summe der Flächen dieser Quadrate.



Es gilt also $\pi \approx \sum_{i=0}^{n-1} f(x_i) \cdot h$ mit $x_i = x_{start} + \frac{(i+0{,}5)}{n}$ und $h = \frac{1}{n}$.

a) Den Programmcode für die Berechnung ohne Threads finden Sie in **Git**. Laden Sie das Programm herunter und führen Sie es aus. Je größer Sie die Zahl n wählen, umso genauer nähert sich das Ergebnis an die Kreiszahl π an.

Hinweis:
Bereits mit einem Wert für n >= 100.000 erhält man ein auf 9 Nachkommastellen korrektes Ergebnis, allerdings läuft das Programm – auf meinem Rechner – dann nur 0.001 Sekunden. Wir müssen also einen etwas größeren Wert für n verwenden um mit Threads einen Geschwindigkeitsvorteil erkennen zu können.

Bei einem Wert n = 10.000.000.000 läuft das Programm auf meinem Notebook für etwa 22 Sekunden. Das ist ideal um die Verbesserung durch Threads deutlich sichtbar zu machen. Für Ihren eigenen Rechner müssen Sie einen passenden Wert für n finden.

b) Modifizieren Sie das Programm aus a so, dass die Berechnung der Kreiszahl π mit Hilfe von POSIX-Threads parallel geschieht. Nutzen Sie mindestens zwei Threads oder schaffen Sie die Möglichkeit, die Anzahl der Threads als Parameter übergeben zu können.

Hinweis: Bei Nutzung von zwei Threads kann zum Beispiel Thread 1 die Summe aller Rechtecke im Intervall [0, 0.5] berechnen, während Thread 2 die Summe aller Rechtecke im Intervall]0.5, 1] berechnet. Zum Schluss müssen die beiden Summen nur noch zu einer Gesamtsumme addiert werden.

c) Statt die Zeit – wie im Beispielprogramm – mittels eigenem C-Code zu messen (weitere Beispiele in Git) kann auch das Programm `time` genutzt werden. Dieses zeigt Ihnen die genutzte CPU-Zeit und die verstrichene Realzeit an. Vergleichen Sie die Laufzeit der beiden Programme. Was fällt Ihnen auf?
→ `time ./ihr_programm_name`