1. Übungsblatt

Nichtsequentielle und verteilte Programmierung

Boyan Hristov, Sergelen Gongor

Aufgabe 1.

Datenparallelität – Viele Daten können parallel bearbeitet werden. Das heißt, die Datenoperationen sind voneinander unabhängig (z.B. wenn man alle Einträge in einem Array aufsummieren will, ist es egal welche man zuerst zusammenaddiert, hier können Operationen parallel ausgeführt werden).

Funktionsparallelität – Verschiedene Aufgaben können parallel ausgeführt werden. Hier geht es nicht um die Daten in einer Aufgabe, sondern um mehrere Aufgaben, die total unabhängig von einander sind. Zum Beispiel, wenn man ein Videokonferrenz macht sind Viedoaufnahme und Videoübergabe zwei parallel laufende Aufgaben.

Aufgabe 2.

Die Prozesse haben ein eigenes Adressraum (eigene Umgebung) und werden durch das Betriebssystem verwaltet, wobei Threads existieren innerhalb eines Prozesses, haben ein gemeinsames Adressraum (dieses des Prozesses) und werden durch das Prozess verwaltet.

Aufgabe 3.

a) Ein Algorithmus ist determiniert, fall dieses bei gleiche Eingabe immer die gleiche Ausgabe liefert, oder anders gesagt falls dieses keine Seiteneffekte hat.

b) Ein Algorithmus ist deterministisch, falls es eine eindeutige Reihenfolge der Zustände gibt und ein Zustand und eine Eingabe immer zu einen eindeutigen Zustand führen. Dabei ist der Ablauf des Algorithmus immer reproduzierbar.

c) Ja, deterministische Algorithmen sind immer determiniert. Falls man für jede Eingabe immer die Zustandsreihenfolge und diese fest ist, dann muss auch für jede Eingabe die gleiche Rückgabe geliefert werden.

d) Nein. Ein gutes Beispiel dagegen wurde in der Vorlesung erwähnt – Quicksort mit zufällig gewählten Pivot. Dabei ist die Zustandsreihenfolge immer unterschiedlich, die Rückgabe ist aber für jede Eingabe immer gleich – ein sortiertes Array.

   Weiter könnte man die Aufsummierung aller Elemente in einer Liste durch mehrere Threads als Beispiel geben. Welche Indizes in der Liste zuerst zusammenaddiert werden ist nicht bekannt, das Ergebniss ist aber immer die aufsummierte Liste.

Aufgabe 4.

Eine nebenläufige, asynchrone Ausführung von Prozessen führt in der Regel immer zu nicht deterministischen Abläufen, wie schon in der Vorlesung erwähnt wurde. Es ist so, weil die Verwaltung der Prozessen von dem Betriebssystem gemacht wird, wobei mehrere Fakten für die Entscheidung welches Prozess vor welches ausgeführt wird, diese Entscheidung ist für jede Ausfuhrung unterschiedlich. Man kann aber immer ein Scheduler konstruieren, der die nebenläufige Prozessen immer in der selben Reihenfolge ausführen kann, deswegen ist das theoretisch keine feste Regel, obwohl in der Praxis immer so ist.

Aufgabe 5.

a) Wir haben ein kurzes Script mit JavaScript hergestellt, der automatisch alle mögliche Varianten herstellt. Dafür muss man die letzte Version von Node.js haben und das folgende in dem selben Ordner ausführen

   node generator.js > out.txt

   Das führt das Programm aus und schreibt das Ergebniss in einer Datei out.txt.

   Ergebniss:

p1 -> p2 -> p3 -> r1 -> r2 -> r3 -> (x = 1, y = 1, z=2)

p1 -> p2 -> r1 -> p3 -> r2 -> r3 -> (x = 2, y = 2, z=4)

p1 -> p2 -> r1 -> r2 -> p3 -> r3 -> (x = 2, y = 2, z=4)

p1 -> p2 -> r1 -> r2 -> r3 -> p3 -> (x = 2, y = 2, z=1)

p1 -> r1 -> p2 -> p3 -> r2 -> r3 -> (x = 0, y = 0, z=0)

p1 -> r1 -> p2 -> r2 -> p3 -> r3 -> (x = 0, y = 0, z=0)

p1 -> r1 -> p2 -> r2 -> r3 -> p3 -> (x = 0, y = 0, z=1)

p1 -> r1 -> r2 -> p2 -> p3 -> r3 -> (x = 0, y = 2, z=2)

p1 -> r1 -> r2 -> p2 -> r3 -> p3 -> (x = 0, y = 2, z=1)

p1 -> r1 -> r2 -> r3 -> p2 -> p3 -> (x = 0, y = 2, z=1)

r1 -> p1 -> p2 -> p3 -> r2 -> r3 -> (x = 1, y = 1, z=2)

r1 -> p1 -> p2 -> r2 -> p3 -> r3 -> (x = 1, y = 1, z=2)

r1 -> p1 -> p2 -> r2 -> r3 -> p3 -> (x = 1, y = 1, z=1)

r1 -> p1 -> r2 -> p2 -> p3 -> r3 -> (x = 1, y = 2, z=3)

r1 -> p1 -> r2 -> p2 -> r3 -> p3 -> (x = 1, y = 2, z=1)

r1 -> p1 -> r2 -> r3 -> p2 -> p3 -> (x = 1, y = 2, z=1)

r1 -> r2 -> p1 -> p2 -> p3 -> r3 -> (x = 1, y = 2, z=3)

r1 -> r2 -> p1 -> p2 -> r3 -> p3 -> (x = 1, y = 2, z=1)

r1 -> r2 -> p1 -> r3 -> p2 -> p3 -> (x = 1, y = 2, z=1)

r1 -> r2 -> r3 -> p1 -> p2 -> p3 -> (x = 1, y = 2, z=1)

b) Verschiedene verzahnte Ausführungen: 20. Da Prozess P und Prozess R nebenläufig ausgeführt werden, kann zu jeder Zeit das im Moment ausgeführte Prozess gewechselt werden. Es kann aber nicht sein, dass z.B. p2 ausgeführt wird, bevor vorher schon p1 ausgeführt wurde.

Aufgabe 6.

a) Das kann man nicht mit Sicherheit sagen, da es total von dem Prozess-Scheduler abhängig ist. Falls dieser total fair ist, dann kann es sogar sein, dass kein Prozess überhaupt beendet wird.

Falls der Counter aber schon z.B. <= -7 ist und Thread 2 als erstes ausgeführt wird, dann wird dieses auch zuerst beendet.

b) Analog wie bei a) ist es vom Scheduler abhängig. Falls dieser total fair ist, dann werden mit Sicherheit die beide Prozessen nie beendet (Falls nur bis zu 7 Schritte je Thread ausgeführt werden).

   In der Praxis aber sind die Schedulers nicht so fair und führere mehrere Schritte eines Threads auf einmal, deswegen werden irgendwann beide Prozesse beendet.
   Man kann das aber nie beweisen.

Aufgabe 8.

a) Wir brauchen maximal n/2 Prozessoren, da wir am schnellsten paarweise die Zahlen aufsummieren können. Dass sind n/2 Paare initial, dann n/4 am nächsten Schritt usw.

b) Am ersten Schritt werden n/2 Additionen ausgeführt, n/4 am 2. und so weiter, bis es nur ein Paar gibt das am Ende aufsummiert wird. Das sind also log n Schritte, damit hat dieses Algorithmus eine Komplexität von O(logn)