ulm university universität
uulm

Institut für Numerische Mathematik

Dr. Andreas F. Borchert

30. April 2019
Blatt 1

Systemnahe Software II (SS 2019)

Abgabe bis zum 7. Mai 2019, 14:00 Uhr

Lernziele:

- Einfache Parallelisierung mit dem Fork-and-Join-Pattern

Hinweis: Kein fork() auf den Servern!

Führen Sie Programme, die *fork()* enthalten, **nie** auf einem der Server (wie etwa der Theon) aus. Der Aufwand, der notwendig ist, um eine (unbeabsichtigte) Fork-Bombe wegzuräumen kann bedeutend sein.

Aufgabe 1: Konstellationen von Primzahlen

Zu den offenen Fragen der Zahlentheorie gehört, ob bestimmte aufeinanderfolgende Konstellationen von Primzahlen unendlich oft auftreten und wenn ja, wie sie verteilt sind. Diese Fragestellung wird gerne empirisch untersucht und lässt sich hervorragend parallelisieren.

Schreiben Sie eine Anwendung, die ein Intervall der natürlichen Zahlen $[N_1, N_2]$ und eine vorgegebene Primzahlkonstellation $\{n_i\}_{i=1}^{k-1}$ mit $n_i < n_{i+1}$ der Länge $k > 1$ erhält. Gesucht und zurückzuliefern sind dann alle Primzahlkonstellationen $(p, p + n_1, \ldots p + n_{k-1})$ mit $N_1 \leq p \leq N_2$. Primzahlpaare werden beispielsweise mit $\{2\}$ gesucht, Primzahlvierlinge mit $\{2, 6, 8\}$.

Wir gehen davon aus, dass alle Prozesse gleich leistungsstark sind, daher teilen wir das Gesamtintervall auf die Zahl der zu erzeugenden Prozesse auf.

Hier ist ein einfaches Beispiel, das nach Primzahlvierlingen in $[1, 1000]$ sucht:

```
theon$ primes -p 10 1 1000 2 6 8 | tr '\n' ' '; echo
5 11 101 191 821
theon$
```

Die Option *-p* spezifiziert die Zahl der zu erzeugenden Prozesse, dann folgen die Intervallgrenzen und danach die relativen Offsets der gewünschten Primzahlkonstellation, wobei die 0 weggelassen wird.

Das folgende Beispiel sucht mit Hilfe von 24 Prozessen in $[2^{48}, 2^{48} + 2^{24}]$ nach Primzahlfünflingen und benötigte (bei unserer Lösung) auf der Theon 1,3 Sekunden:

```
theon$ primes -p 24 281474976710656 281474993487872 2 6 8 12
281474977256771
281474980559231
281474981633891
281474992639271
theon$
```

(Bitte verwenden Sie selbst nicht die Theon für Tests.)

Hinweise:

Für das Rechnen mit großen Zahlen empfiehlt sich die Verwendung der *GNU Multiple Precision Arithmetic Library*, kurz *GMP*. Die Dokumentation dazu steht direkt auf der Webseite `http://gmplib.org/` zur Verfügung. Funktionen im Kontext von Primzahlen sind unter dem Abschnitt *Number Theoretic Functions* beschrieben. Es ist besonders wichtig, darauf zu achten, dass alle verwendeten Variablen mit *mpz_init* oder einer der Alternativen zu Beginn initialisiert wurden! Beim Übersetzen Ihres Programms ist dann zusätzlich die Option *-lgmp* mit anzugeben.

Sie sollten in Ihrem ersten Schritt mit der Implementierung einer Funktion beginnen, die ohne Parallelisierung das oben genannte Problem für ein Intervall löst.

Die Ausgabe der gefundenen Zahlenkonstellationen soll für jeden Prozess in eine temporäre Datei erfolgen. Hierbei genügt es, die jeweils erste Zahl einer Konstellation auszugeben. Nachdem alle erzeugten Prozesse beendet sind, soll das Hauptprogramm die gefundenen Zahlenkonstellationen in der richtigen Reihenfolge auf der Standardausgabe ausgeben.

Das Programm lässt sich gut modularisieren und entsprechend im Team aufteilen. Hier ein Vorschlag dazu:

- *primes.c/primes.h*: Enthält eine Funktion, die in einem gegebenen Intervall nach der ersten Primzahlenkonstellation sucht.
- *worker.c/worker.h*: Enthält eine Funktion, das ein gegebenes Intervall nach den gewünschten Primzahlenkonstellationen absucht und die gefundenen Konstellationen (jeweils repräsentiert durch die erste Zahl) auf einem gegebenen Dateideskriptor ausgibt (jeweils mit einem Zeilentrenner nach der Zahl).
- *tmpfile.c/tmpfile.h*: Stellt eine Funktion zur Verfügung, um eine temporäre Datei zu erzeugen. Dabei ist nur der Dateideskriptor zurückzugeben; der Dateiname selbst ist mit *unlink* zu entfernen. Die Funktion sollte sicherstellen, dass der Name innerhalb von */tmp* eindeutig ist (wozu ggf. mehrere Versuche notwendig sind).

- *main.c*: Liest die Intervallgrenzen und die gewünschte Zahlenkonstellation als Kommandozeilenargumente ein, teilt das Intervall gleichmäßig auf und erzeugt die Prozesse, die die Teilintervalle durchsuchen. Nachdem diese Prozesse alle beendet sind, gibt das Hauptprogramm die Ergebnisse geordnet aus.

Verpacken Sie all Ihre Quellen wiederum mit *tar* in ein Archiv und reichen Sie dies auf der Theon ein:

```
tar cvf primes.tar *.c *.h [mM]akefile
submit ss2 1 team [notes] primes.tar
```

Viel Erfolg!