Systemnahe Programmierung in C (SPiC)

30 Multiprozessoren

Jürgen Kleinöder, Daniel Lohmann, Volkmar Sieh

Lehrstuhl für Informatik 4
Verteilte Systeme und Betriebssysteme

Friedrich-Alexander-Universität
Erlangen-Nürnberg

Sommersemester 2022

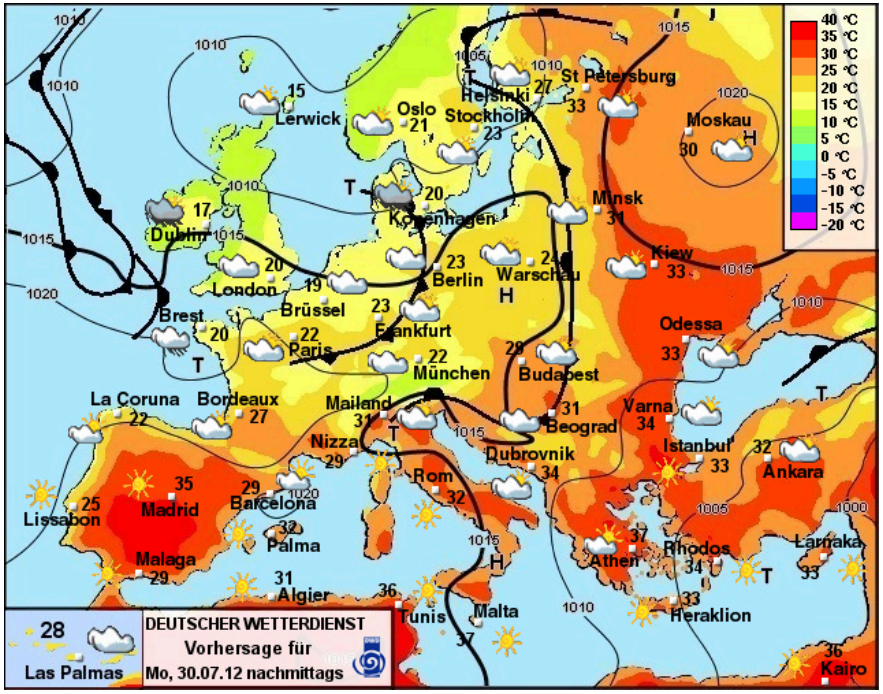
http://sys.cs.fau.de/lehre/SS22/spic

Prozesse

- Mehrere Prozesse zur Strukturierung von Problemlösungen
  Aufgaben einer Anwendung leichter modellierbar, wenn sie in mehrere kooperierende Prozesse unterteilt wird
  - z.B. Anwendungen mit mehreren Fenstern (ein Prozess pro Fenster)
  - z.B. Anwendungen mit vielen gleichzeitigen Aufgaben (Web-Browser)
  - z.B. Client-Server-Anwendungen;
    pro Anfrage wird ein neuer Prozess gestartet (Web-Server)
- Multiprozessorsysteme werden erst mit mehreren parallel laufenden Prozessen ausgenutzt
  - früher nur bei Hochleistungsrechnern (Aerodynamik, Wettervorhersage)
  - durch Multicore-Systeme jetzt massive Verbreitung

Beispiel: Berechnung einer Wetterkarte

- Berechnung der Wetterkarte muss so schnell wie möglich erfolgen

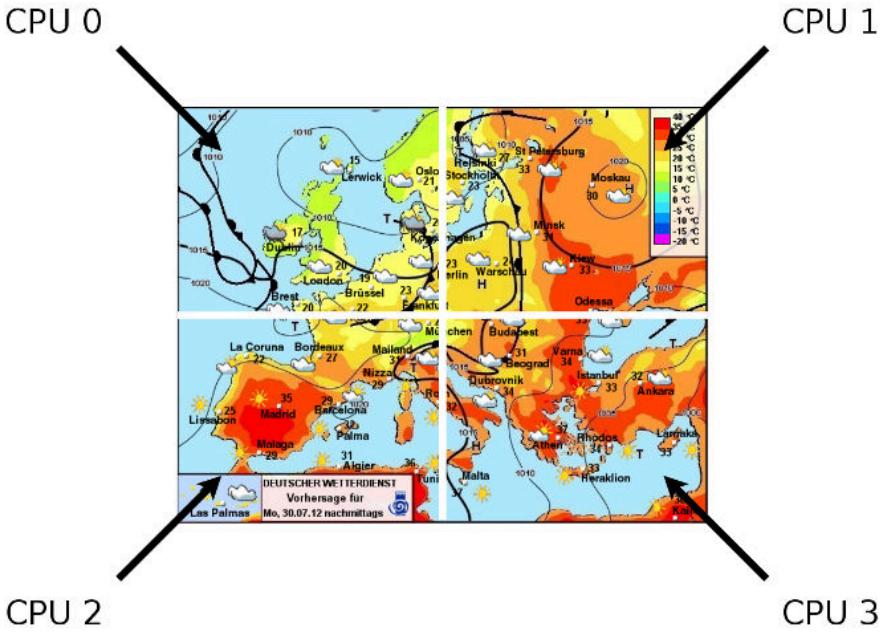


Quelle: www.wetterdienst.de

- Ansatz: Mehrere Prozessoren berechnen jeweils einen Teil der Karte

Beispiel: Berechnung einer Wetterkarte (2)

- Z.B. Berechnung der Wetterkarte aufgeteilt auf 4 Prozessoren:



- Alle Prozessoren greifen auf einen gemeinsamen Speicher zu, in dem das Ergebnis berechnet wird.

Prozesse mit gemeinsamen Speicher

- Nutzung von gemeinsamen Speicher durch mehrere Prozesse

```
char *ptr = mmap(NULL, NBYTES, PROT_READ | PROT_WRITE,
                 MAP_SHARED | MAP_ANONYMOUS, -1, 0);
if (ptr == MAP_FAILED) ... // Fehler

for (i = 0; i < NPROCESSES; i++) {
    pid[i] = fork();
    switch (pid[i]) {
    case -1: ...              // Fehler
    case 0:
        do_work(i, ptr);
        _exit(0);
    default:;
    }
}
for (i = 0; i < NPROCESSES; i++) {
    ret = waitpid(pid[i], NULL, 0);
    if (ret < 0) ...          // Fehler
}

ret = munmap(ptr, NBYTES);
if (ret < 0) ...              // Fehler
```

Beispiel: Vektorlänge

- Berechnung der Länge/Norm eines N-Elemente-Vektors mit einem Prozess:

```
#include <math.h>

double
veclen(double vec[])
{
    double sum = 0.0;

    for (int i = 0; i < N; i++) {
        sum += vec[i] * vec[i];
    }

    return sqrt(sum);
}
```

Beispiel: Vektorlänge (2)

- Berechnung der Länge eines N-Elemente-Vektors mit vier Prozessen:

```
double veclen(double vec[]) {
    pid_t pid[4];
    double *ptr = mmap(NULL, 4096, PROT_READ | PROT_WRITE,
                       MAP_SHARED | MAP_ANONYMOUS, -1, 0);
    for (int p = 0; p < 4; p++) {
        if ((pid[p] = fork()) == 0) {
            double sum = 0.0;
            for (int i = p * N / 4; i < (p + 1) * N / 4; i++)
                sum += vec[i] * vec[i];
            ptr[p] = sum;
            _exit(0);
        }
    }
    for (int p = 0; p < 4; p++)
        waitpid(pid[p], NULL, 0);
    double sum = 0.0;
    for (int p = 0; p < 4; p++)
        sum += ptr[p];
    munmap(ptr, 4096);
    return sqrt(sum);
}
```

Beispiel: Vektorlänge (3)

- Hinweis: Beispiel unvollständig
  - `#includes` fehlen
  - Fehlerbehandlung fehlt
  - ...
- Trotzdem sieht man
  - Programmierung sehr viel aufwändiger
  - Programm sehr viel unübersichtlicher
  - eigentlicher Algorithmus kaum noch erkennbar
- Ergebnis ernüchternd
  - Aufwand lohnt sich bei aktuellen Rechnern erst ab etwa $N = 100000$

Prozesse mit gemeinsamen Speicher (2)

Vorteil der obigen Lösung: in Multiprozessorsystemen sind **echt-parallele Abläufe möglich**

aber

jeder Prozess hat eigene Betriebsmittel

- Speicherabbildung
- Rechte
- offene Dateien
- Wurzel- und aktuelles Verzeichnis
- ...

=> **Prozess-Erzeugung, Prozess-Terminierung und Prozess-Umschaltungen sind teuer**