System-Level Programming

30 Multiprocessors

Peter Wägemann

Lehrstuhl für Informatik 4
Systemsoftware

Friedrich-Alexander-Universität
Erlangen-Nürnberg (FAU)

Summer Term 2025

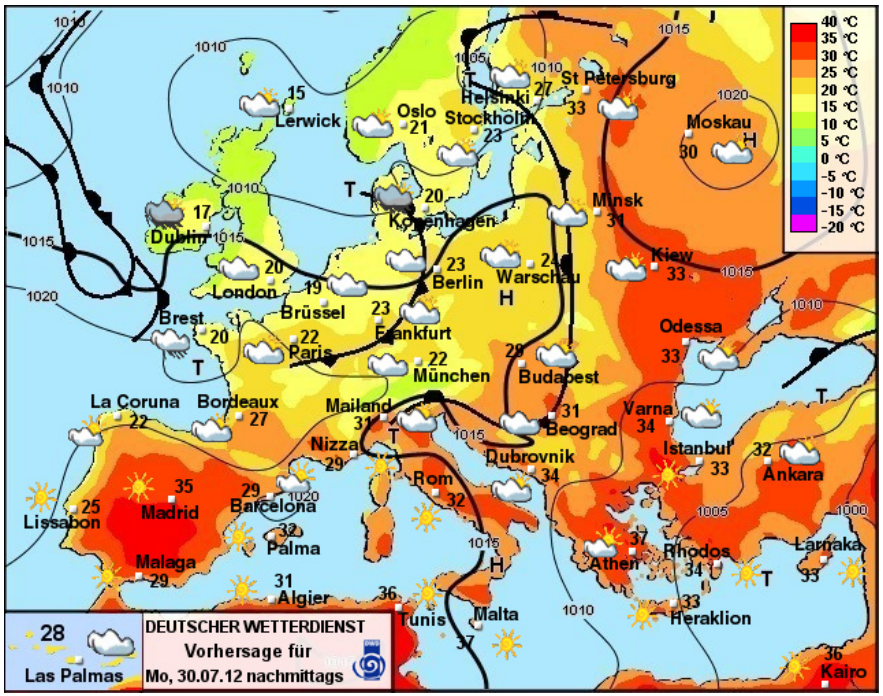
http://sys.cs.fau.de/lehre/ss25

Processes

- Multiple **processes** for structuring of solutions
- Tasks of an application can be modeled easier when divided into **multiple cooperating subprocesses**
  - e. g., applications with multiple windows (one process per window)
  - e. g., applications with many concurrent tasks (web browser)
  - e. g., client server applications;
    for each request a new process gets started (web server)
- **Multiprocessor** systems can only be used efficiently with multiple processes running in parallel
  - in the past this was only viable for high-performance computers (aerodynamics, weather prediction)
  - with today's multi-core systems very common

Example: Computing of Weather Map

- Computation of a weather map has to be as fast as possible

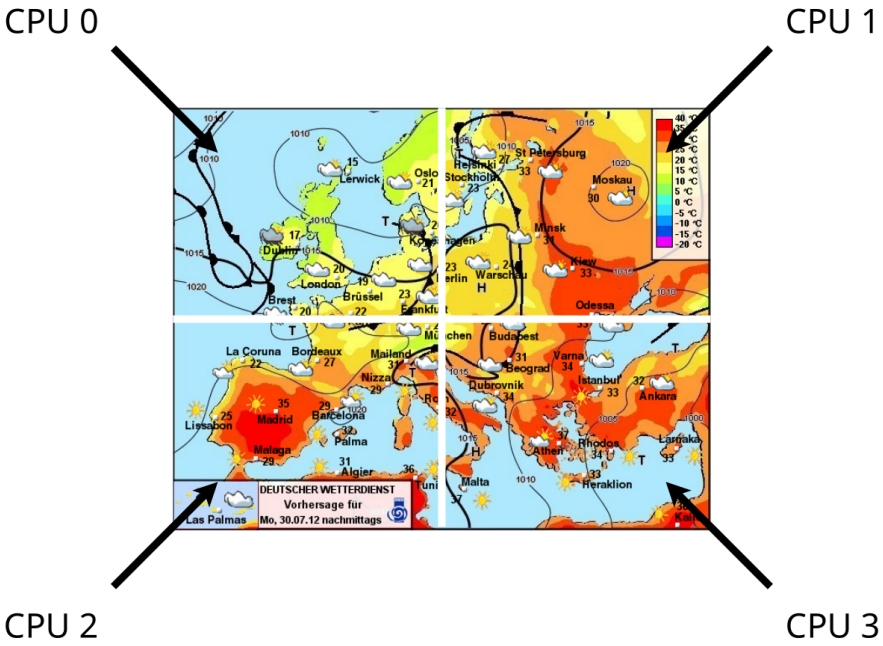


Source: www.wetterdienst.de

- Approach: multiple processes compute distinct parts of the map

Example: Computing of Weather Map (2)

- E. g., computation of a weather map split up between 4 processors:



- All processes access a shared memory area for computing the result

Processes with Shared Memory

- Use of *shared memory* by multiple processes

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

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

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

Example: Length of a Vector

- Calculation of the length/norm of a vector with N elements in one process:

```
#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);
}
```


Example: Length of a Vector (2)

- Compute of the length/norm of a vector with N elements with 4 processes:

```
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);
}
```

Example: Length of a Vector (3)

- Note that example is incomplete
  - `#include` instructions missing
  - error handling missing
  - ...
- Nonetheless, example illustrates
  - programming is more complex
  - program structure less straight-forward
  - parallel algorithm is harder to understand
- Benefits can be unintuitive
  - Significant overhead expenses for forking processes
  - The additional overhead for forking is only beneficial for very large vectors, that is, for values of N greater than 100 000 (depending on the actual machine)

Processes with Shared Memory (2)

Advantage of the solution above: in multiprocessor systems, **physically parallel execution** is possible

BUT

each process needs its own resources

- memory mapping
- permissions
- open files
- root and working directory
- ...

⇒ **creation, termination, and switching of processes is expensive**