Parallel Programming Concepts
WS 2013 / 2014

Assignment 1 (Submission deadline: Nov 8th 2013, 23:59 CET)

General Rules

The assignment solutions have to be submitted at:

https://www.dcl.hpi.uni-potsdam.de/submit/

Our automated submission system is intended to give you feedback about the validity of your file upload. A submission is considered as accepted if the following rules are fulfilled:

- You did not miss the deadline.
- Your file upload can be decompressed with a zip / tar decompression tool.
- Your submitted solution contains only the source code files and a Makefile for Linux 2.6 64-bit. Please leave out any Git / Mercurial repository clones or SVN / CVS meta-information.
- Your solution can be compiled using the "make" command, without entering a separate sub-directory after decompression.
- You program runs without expecting any kind of keyboard input or GUI interaction.
- Our assignment-specific validation script accepts your program output / generated files.

If something is wrong, you will be informed via email (console output, error code). Re-uploads of corrected solutions are possible until the deadline.

50% must be solved correctly in order to pass the assignment. Documentation should be done inside the source code.

Students can submit solutions either **alone or as team of max 2 persons**.

Assignment 1

The first assignment covers the usage of basic synchronization primitives in a thread-based shared memory system. You have to solve the given programming exercises in C / C++ with the POSIX PThread API[1]. Additional threading libraries are not allowed for this assignment.

[1] `man pthread`

Hasso Plattner Institute Operating Systems and Middleware Group
Dr. Peter Tröger, Frank Feinbube

Task 1.1

Implement a program that sums up a range of numbers in parallel. The general algorithmic problem is called “parallel reduction”.

Input

Your application has to be named “parsum” and accept three parameters: The *number of threads to use*, the *start index* and the *end index* (64bit numbers) of the range to compute. For example, the command line

```
Example: ./parsum 30 1 10000000000
```

has to result in a parallel summation of the numbers 1,2,...,10.000.000.000, based on 30 threads running in parallel.

Output

Your program has to produce an output file with the name “output.txt” in the same directory. This file has to contain only the computed sum.

Validation

The solution is considered correct if a true parallelized computation takes place (no Gauss please), if the solution scales based on the number of threads, and if the application produces correct results for all inputs. We will evaluate your solution with different thread counts / summation ranges.

The student achieving the lowest average runtime will be announced in the lecture.

Task 1.2

Implement the dining philosophers with a freely chosen deadlock-free solution strategy[2].

Each philosopher has to be represented by a thread. You are free to map also other stake holders (e.g. waiters) or resources (e.g. forks) to threads if necessary.

Input

Your application has to be named “dinner” and accept two parameters, the *number of philosophers* resp. forks (min. 3) and the *maximum run time* in seconds.

```
Example: ./dinner 3 10
```

Output

After the given run time is exceeded, the program must terminate with exit code 0 and produce an output file with the name of “output.txt” in the same directory. This file has to contain nothing but the number of “feedings” per philosopher as semicolon separated values.

```
Example: 5000;5000;5000
```

[2] http://en.wikipedia.org/wiki/Dining_philosophers_problem