TECHNISCHE UNIVERSITÄT CHEMNITZ

Department of Computer Science
Chair for Practical Computer Science
Prof. Dr. G. Rünger
Dr. Jens Lang
Thomas Jakobs, M.Sc.

2nd Task Sheet

The tasks 1 and 2 are solved together during the tutorial. Task 3 is supposed to be solved until the next tutorial session. Please hand in your solution before the tutorial or upload it in OPAL.

Task 1

Consider the following C programme:

```
int ressource = 0;

void inc_r(unsigned int inc)
{
  ressource = ressource + inc;
}

void main(void)
{
  inc_r(1); inc_r(2); inc_r(3); inc_r(4);
  printf("%d\n", ressource);
}
```

The following parallel programme sketches a parallel implementation of the programme using threads.

```
int ressource = 0;

void inc_r(unsigned int inc)
{
  ressource = ressource + inc;
}

void thread_function1()
{
  inc_r(1); inc_r(2);
}

void thread_function2()
{
  inc_r(3); inc_r(4);
}

void main(void)
{
  // starte Thread 1, der thread_function1 ausführt und sich danach beendet
  // starte Thread 2, der thread_function2 ausführt und sich danach beendet
  // warte auf die Beendigung beider Threads
  printf("%d\n", ressource);
}
```

In the parallel programme, two threads are started which execute the functions `thread_function1()` and `thread_function2()`, respectively. Both threads access the shared variable `ressource` when executing `inc_r`.

a) Give reason why the result of the sequential implementation is not necessarily equal to the result of the parallel implementation. Identify race conditions.

b) Eliminate the race conditions in the parallel version by using a lock variable `s` and the functions `lock(s)` and `unlock(s)`.

Task 2

The following three functions use lock variables `S1`, `S2` and `S3`. They are executed in parallel by three different threads, which are running on a separate core, each. During the exection, a problem occurs.

a) Explain the problem and the situation in which it arises.

b) Propose a solution which avoids the problem.

c) Would the problem also occur on a dual-core system (with 2 cores)?

```
void f1()                void f2()                void f3()
{                        {                        {
  lock(S1);                lock(S2);                lock(S3);
  lock(S2);                lock(S3);                lock(S1);

  // critical section      // critical section      // critical section

  unlock(S2);              unlock(S2);              unlock(S1);
  unlock(S1);              unlock(S3);              unlock(S3);
}                        }                        }
```

Task 3

Three threads T_1, T_2 and T_3 are executed simultaneously where each of them iterates a loop from 1 through n. How can you assure that each thread only starts whith executing the iteration $i+1$ if all other threads have finished the iteration i, $1 \leq i < n$? Give two solutions which use different synchronisation methods.