**1. What is a race condition?**

A "race condition" exists when multithreaded (or otherwise parallel) code that would access a shared resource could do so in such a way as to cause unexpected results.

You are planning to go to a movie at 5 pm. You inquire about the availability of the tickets at 4 pm. The representative says that they are available. You relax and reach the ticket window 5 minutes before the show. I'm sure you can guess what happens: it's a full house. The problem here was in the duration between the check and the action. You inquired at 4 and acted at 5. In the meantime, someone else grabbed the tickets. That's a race condition - specifically a "check-then-act" scenario of race conditions.

2.

1. **Why is it impossible to achieve Mutual Exclusion via disabling interrupts on a multi-core machine?**

Der gegenseite Ausschluss von interrupts ist nur auf einen EinzelSystemProzesser möglich.

1. **Why is it dangerous to give user processes the power to disable interrupts?**

Ohne Unterbrechugsanforderung entsteht eine „endlos-Schleife“ d.h. man kann nichts mehr mit dem PC machen.

3.

1. **Play through the two scenarios of the handout of Peterson's solution. Document how it works.**

✤ Scenario 1

* Process 0 calls enter\_region(), finishes it, and is within the CR
* Process 1 calls enter\_region()
* Process 0 calls leave\_region()

✤ Scenario 2

* Process 0 and Process 1 go through the whole loop nearly simultaneously.

1. **Play through the scenario which makes the strict alternation approach fail. Document how it fails.**

Process A has high priority, i.e., has big time slices Process B has a low priority, i.e, has small time slices

1. What is the meaning of the variable {\tt turn} in Peterson's solution? When does it have any effect?