Christian-Albrechts-Universität zu Kiel
Institut für Informatik
Prof. Dr. W.-P. de Roever
Jens Schönborn und Jan Waller

Nebenläufige und verteilte Programmierung

Wintersemester 2006/07

Serie 9

18.12.2006

Ausgabetermin: 18.12.2006

Abgabe: 8.1.2007 (11:00)

Aufgabe 1 (6 Punkte) The following problems refer to the Java programs in Section 5.4. The sources for the programs can be downloaded from the Website for this book.

1. Write a simple program that has two classes. One defines a thread and has a run method that prints a line; the second is the main class that creates a thread. In the main class, create and start the thread as described in the text. Then try calling the run method directly rather than indirectly via start. (Namely, use **s.run()** if **s** is the thread.) Describe what happens and why.

2. Develop more realistic simulations of the readers/writers programs. Use multiple readers and writers and modify them so that you can observe that they synchronize correctly. Perhaps modify the database to make it somewhat more realistic, or at least to have **read** and **write** take longer. Also have each thread sleep for a small random amount of time before (or after) every access to the database. Java provides several methods - such as **nap**, **age**, **random**, and **seed** - that you can use to construct your simulations. Write a brief report summarizing what you observe.

3. Modify the **ReadersWriters** class to give writers preference. Repeat your simulations form part 2, and summarize what you observe.

4. Modify the **ReadersWriters** class to make it fair. Repeat your simulations from part 2, and summarize what you observe.

Aufgabe 2 (4 Punkte) Develop a Java program to simulate the dining philosophers problem (Section 4.3). Your program should have 5 philosopher threads and a class that implements a monitor to synchronize the philosophers. The monitor should have two methods: **getforks(id)** and **relforks(id)**, where **id** is an integer between **1** and **5**. Have the philosophers eat and sleep for random amounts of time. Add print statements to your program to generate a trace of the activity of the program. Write a brief report summarizing what you observe.