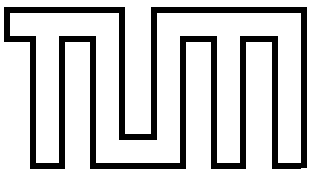

Technische Universität
München

Fakultät für Informatik
Forschungs- und Lehreinheit Informatik VI

Übung zur Vorlesung Echtzeitsysteme

Aufgabe 5 – Semaphoren

Dr. Christian Buckl
buckl@in.tum.de

Simon Barner
barner@in.tum.de

Michael Geisinger
geisinge@in.tum.de

Stephan Sommer
sommerst@in.tum.de

Wintersemester 2008/09

Aufgabe 5: Semaphoren

In der Vorlesung wurden Deadlocks anhand des *Dining Philosophers* Beispiels veranschaulicht. Ziel des aktuellen Aufgabenblattes ist es, diese bekannte Aufgabenstellung mit dem aus der 1. Übung bekannten Echtzeitbetriebssystem VxWorks umzusetzen.

Allgemeines

Alle Arbeiten zur Erstellung von Anwendungen für VxWorks finden im Kontext von Projekten statt. Verschiedene Projekte werden jeweils in einem Workspace verwaltet. Gehen Sie sicher, dass der Workspace auf dem Laufwerk Z: liegt (*File → Switch Workspace*). Jede Anwendung für VxWorks wird einem Projekt zugeordnet. Jedes Projekt benötigt sein eigenes Verzeichnis. Dort liegen der Quellcode, der auf mehrere Dateien verteilt sein kann, sowie Informationen zur Verwaltung des Projekts. Um ein neues Projekt ins Leben zu rufen, klickt man auf *File → New* und entscheidet sich anschließend für ein *VxWorks Downloadable Kernel Module Image*. Im nächsten Schritt gibt man einen Namen an und wählt die Option *Create project in workspace* aus. Im nachfolgenden *Build Specs* Dialog wählt man als *Active bild spec* die Option *SIMNTgnu*.

Für das Bearbeiten der nachfolgenden Aufgaben sind für die Semaphoren und Threads jeweils die Posix-Varianten zu verwenden. Verwenden Sie außerdem den VxWorks Simulator für Ihre Tests. Vom Rechner „atknoll133“ können Sie wieder ein Rahmenprogramm zum Bearbeiten der aktuellen Aufgabenstellung herunterladen.

Abbildung 1: Speisende Philosophen

Philosophen

Die Philosophen sitzen am Tisch 1 und denken über philosophische Probleme nach. Wenn einer hungrig wird, greift er zuerst die Gabel links von seinem Teller, dann die auf der rechten Seite und beginnt zu essen. Wenn er satt ist, legt er die Gabeln wieder zurück und beginnt wieder zu denken. Sollte eine Gabel nicht an ihrem Platz liegen, wenn der Philosoph sie aufnehmen möchte, so wartet er, bis die Gabel wieder verfügbar ist.

a) Implementieren Sie die Funktion *eating()* fertig, die von der Funktion *philosopher()* aufgerufen wird und folgenden Ablauf ausführen soll

```
eating(int i){ //i: Nummer des Philosophen
        P(gabel[i] % N); // Linke Gabel nehmen
        P(gabel[(i-1)]); // Rechte Gabel nehmen
        /*Der Philosoph ißt ... zufällige Zeit*/
        V(gabel[i] % N); // Linke Gabel zurücklegen
        V(gabel[(i-1)]); // Rechte Gabel zurücklegen
}
```

Verwenden Sie zum Schutz der kritischen Bereiche (=Gabeln) Semaphoren. Definieren Sie hierzu ein globales Array für die Semaphoren, dessen Größe der Anzahl der Philosophen (und dementsprechend auch der Gabeln) entspricht. Folgende Funktionen können Ihnen dabei hilfreich sein:

- sem_open()
- sem_post()
- sem_wait()

Solange nur einzelne Philosophen hungrig sind, funktioniert dieses Verfahren wunderbar. Es kann aber passieren, dass sich alle fünf Philosophen gleichzeitig entschließen zu essen. Sie ergreifen also alle gleichzeitig ihre linke Gabel und nehmen damit dem jeweils links von ihnen sitzenden Kollegen seine rechte Gabel weg. Nun warten alle fünf darauf, dass die rechte Gabel wieder auftaucht. Dies passiert aber nicht, da keiner der fünf seine linke Gabel zurücklegt. Die Philosophen verhungern (*Deadlock*). Deadlock oder Verklemmung bezeichnet in der Informatik einen Zustand, bei dem ein oder mehrere Prozesse auf Betriebsmittel warten, die dem Prozess selbst oder einem anderen beteiligten Prozess zugeteilt sind.

b) Da es in der Funktion eating() zu einem Deadlock kommen kann, müssen Sie die Funktion erweitern. Überlegen Sie sich, wie man diesen vermeiden kann und implementieren Sie Ihre Lösung.