Betriebssysteme **WS 2007/2008**
Dr. Henrik Brosenne Georg-August Universität Göttingen
Institut für Informatik

Übung 6

Aufgabe 1

Gegeben sei folgendes Problem.

In einer Bäckerei arbeiten drei Lehrlinge sowie ein Meister. In endloser Folge bereitet jeder Lehrling den Teig für ein Blech Brötchen vor. Dafür benötigt er drei Zutaten: Mehl, Wasser und Hefe. Jeder Lehrling verfügt über einen unbegrenzten Vorrat genau einer Zutat, die die anderen Lehrlinge nicht besitzen. Der Meister hingegen verfügt über einen unbegrenzten Vorrat von allen Zutaten, von denen er jedoch immer nur ein zufälliges Paar von zwei verschiedenen Zutaten gleichzeitig anbietet und darauf wartet, dass einer der Lehrlinge diese entgegen nimmt. Jeweils der Lehrling mit der passenden dritten Zutat nimmt die beiden ihm fehlenden Zutaten vom Meister, bereitet den Brötchenteig für ein Blech zu und teilt dem Meister mit, wenn er damit fertig ist. Daraufhin bietet der Meister eine neue zufällige Kombination von zwei Zutaten an.
Erstelle basierend auf dem Semaphor-Konzept eine Lösung, die sicherstellt, dass die Verteilung der Zutaten korrekt erfolgt und die Lehrlinge sich nicht gegenseitig blockieren. Gebe hierzu kommentierten Pseudo-Code für

- die Initialisierung von gemeinsamen Variablen/Semaphoren,
- für den Meister
- sowie exemplarisch für einen der Lehrlinge

an.

Aufgabe 2

1. Modifiziere die in Aufgabe 2 von Übung 5 erstellten modularen *C*-Funktionen `void enter_criticalregion(int process)` und `void leave_criticalregion(int process)` sowie die globalen Variablen und deren Initialisierung so, dass nun POSIX-Semaphore statt des Peterson-Algorithmus' verwendet werden.

   Hinweis: Bei Bedarf können die Musterlösungen für Übung 5 (`http://www.stud.informatik.uni-goettingen.de/os/ws2007/uebung/besprechung05.tgz`) von der Vorlesungsseite heruntergeladen werden.

2. Lasse das Programm laufen. Stimmt der ausgegebene Wert von `in` mit dem doppelten Wert von `COUNT_MAX` überein?

   Sorge ggf. durch Erhöhen von `COUNT_MAX` dafür, dass Du, während Dein Programm läuft, auf der Kommandozeile das Programm `top` starten kannst. Zu wie viel Prozent lastet der Prozess bzw. die Threads die CPU aus?

3. Sorge dafür, dass sichtbar wird, welcher Thread sich gerade im kritischen Abschnitt befindet, indem jeder Thread im kritischen Abschnitt mittels `fprintf(stderr, ...)` eine '`0`' bzw. '`1`' als Thread-Identifikation auf der Standard-Fehlerausgabe ausgibt.

   Verschaffe Dir einen Eindruck, inwiefern sich die beiden Threads abwechseln. Füge den Betriebssystemaufruf `sched_yield()` in den kritischen Abschnitt ein bzw. entferne ihn aus diesem. Wie ändert sich die obige Ausgabe der Nullen und Einsen?

4. Wie müsste das Programm geändert werden, damit maximal 2 Threads gleichzeitig den kritischen Abschnitt betreten dürfen?

5. Sorge nun mittels Semaphoren für eine Bedingungssynchronisation der beiden Threads: diese sollen immer abwechselnd ausgeführt werden, d.h. erst soll Thread 0 den Wert der globalen Variable `in` erhöhen, danach Thread 1, danach wieder Thread 0 usw.

   Wie viel Semaphore werden hierfür benötigt?

   Wird der Semaphor für den wechselseitigen Ausschluss noch benötigt, wenn obige Bedingungssynchronisation sichergestellt ist?