Universität Heidelberg
Sommersemester 2022

AG Parallele und Verteilte Systeme (PVS)
Artur Andrzejak

3. Übung zur Vorlesung „Betriebssysteme und Netzwerke“ (IBN)

Abgabedatum: 17.05.2022, 11:00 Uhr

Aufgabe 1 (1 Punkt)

Die Spinlocks (d.h. Locks mit aktivem Warten) werden in Betriebssystemen Solaris, Linux, und Windows nur auf Mehrprozessor-Systemen (bzw. Multi-Core-Systemen) verwendet. Warum macht das Sinn?

Aufgabe 2 (2 Punkte)

In der Vorlesung wurde erläutert, wie Locks oder Semaphore benutzt werden können, um den wechselseitigen Ausschluss zu gewährleisten. Hier ist die zugehörige Pseudocode:

```
S := 1;    // globale Initialisierung vor dem Start der Threads
// Per Thread
wait(S);

           // critical section

signal(S);
```

Beschreiben Sie, was passieren würde, wenn der Programmierer einen der folgenden Fehler begeht (geben Sie jeweils ein kurzes Beispiel einer unerwünschten Ausführung an):

a) die Reihenfolge von `signal()` und `wait()` wird vertauscht;

b) der Aufruf `signal()` wird durch den (erneuten) Aufruf `wait()` ersetzt;

c) der Aufruf von `wait()` und/oder von `signal()` wird ausgelassen.

Aufgabe 3 (3 Punkte)

In dieser Aufgabe werden Sie erneut das Online-Spiel *The Deadlock Empire*[1] verwenden, und folgende Aufgaben im Kontext der Prozesssynchronisation lösen.

a) Lösen Sie die Aufgabe „Insufficient Lock” aus dem Kapitel „Locks” und beschreiben Sie kurz, wie Sie ihre Lösung erreicht haben. Warum ist die entstandene Situation unerwünscht?

[1] https://deadlockempire.github.io/

b) Betrachten Sie die Aufgabe „Manual Reset Event" aus dem Kapitel „High-Level Synchronization Primitives" und lösen Sie diese (wie üblich geben Sie den Lösungsweg an). Weiterhin erläutern Sie via Pseudocode und Text, wie man mit der Abstraktion aus dieser Aufgabe die in der Vorlesung vorgestellte Funktionalität „Abhängigkeit" (d.h. Codeblock B darf erst nach Codeblock A ausführen) umsetzen kann.

c) Lösen Sie die Aufgabe „Semaphores" aus dem gleichnamigen Kapitel, und beschreiben Sie den Lösungsweg. Weiterhin recherchieren Sie das API der C#-Abstraktion *SemaphoreSlim* (siehe Link in der Aufgabe) und übersetzen Sie das in der Vorlesung vorgestellte API der Semaphore auf das API von *SemaphoreSlim*. Insbesondere erläutern Sie, wie Sie den Wert der Semaphorvariable setzen und wieder lesen können.

d) Lösen Sie die Aufgabe „Producer-Consumer" aus dem Kapitel „Semaphores" und geben ihre Schritte an. Warum vereinfacht die dort verwendete Datenstruktur *System.Collections.Generic.Queue* die Umsetzung des Producer-Consumer Schema im Vergleich zu der Vorlesung, und wie?

e) Lösen Sie die Aufgabe „Condition Variables" aus dem Kapitel gleichnamigen Kapitel, und beschreiben Sie ihre Schritte (ohne Erläuterungen).

Aufgabe 4 (4 Punkte)

Überlegen Sie, wie die Semaphor-Operationen wait() und signal() auf Mehrprozessor-Systemen mit Hilfe der Hardware-Befehle TSL bzw. XCHG umgesetzt werden können. Schreiben Sie die Lösung als einen C-ähnlichen Pseudocode auf. Ihre Lösung sollte das aktive Warten (d.h. das Ausführen einer Warteschleife) möglichst kurz halten. Hierbei wird die Operation TSL als Funktion `boolean TestAndSet(boolean *target)` dargestellt, bzw. die Operation XCHG als Funktion `void Swap(boolean *a, boolean *b)`.

Aufgabe 5 (2+3 Punkte)

Betrachten Sie die in der Vorlesung 6 eingeführten *Channels* der Programmiersprache Nim. Lösen Sie die folgenden Aufgaben:

a) Wie können Sie mit den Channels das Problem der *verallgemeinerten Abhängigkeit* lösen, bei dem ein Thread erst dann ausführt, wenn jeder der $k > 0$ anderen Threads den jeweils eigenen Codeabschnitt abgearbeitet hat? (Wir haben in der Vorlesung eine Lösung mit Semaphoren für $k = 1$ kennengelernt.) Geben Sie Ihre Lösung in Form von Pseudocode und einer kurzen Erläuterung ab.

b) Wie können Sie mit den Channels den wechselseitigen Ausschluss umsetzen? Auch hier reicht der Pseudocode mit einer Erläuterung aus.

c) (Bonusaufgabe, 3 Punkte) Implementieren Sie in Nim Ihre Lösung aus (a) und testen Sie Ihren Code an einem einfachen Szenario für $k > 2$. Reichen Sie Ihren Quelltext und die Ausgabe bei einem Beispiellauf ein.

Aufgabe 6 (2 Punkte)

Schauen Sie sich das Video von Mike Swift "Lecture 3, Unit 2: using condition variables"[2] an. Beantworten Sie anschließend folgende Fragen:

a) Wie unterscheiden sich Semaphore im Bezug auf „Gedächtnis" von Mutexen und Bedingungsvariablen? Was passiert, wenn ein Thread ein *signal* broadcastet, aber zu diesem Zeitpunkt kein Thread darauf wartet?

b) Warum muss der Mutex zu *done* aufrechterhalten werden, wenn man das signal broadcastet?

Aufgabe 7 (2 Punkte)

Schauen Sie das Video *Learning How to Learn - Introduction to Memory - Procrastination, Memory, and Sleep*[3]. Hier werden zwei Arten des Gedächtnisses eingeführt und erklärt.

- Benennen und vergleichen Sie die beiden Arten des Gedächtnisses.
- Vergleichen Sie die beiden Gedächtnissysteme mit Speichersystemen in der Speicherhierarchie (Vorlesung 2). Betrachen Sie dabei die Aspekte Größe, Geschwindigkeit und Flüchtigkeit.
- Wie groß sind die beiden Gedächtnissysteme, d.h. wie viele Einheiten können sie jeweils enthalten und wie werden diese Einheiten im Video genannt?
- Es kann Information von beiden Systemen ins jeweils andere bewegt werden. Worin unterscheiden sich diese Richtungen? Für eine der beiden Richtungen wird im Video eine Technik erklärt. Benennen und beschreiben Sie sie.

[2] http://goo.gl/stNNx5
[3] https://www.youtube.com/watch?v=0egaPpfFFLI