alp4 - Übungszettel 4

Johannes Rohloff

16. Mai 2013

Aufgabe 1

Entwickeln Sie mit universellen kritischen Abschnitten eine faire Lösung für das Leser-Schreiber-Problem.

Für diese Aufgabe nehem ich an, dass ein universeller Kritischer Abschnitt, wie er im Buch implementiert ist, zur verfügung steht. Es ist noch eien Invariante anzugeben, die das Problem charakterisiert und zu zeigen, wie diese Invariante sich auf die Fairness des Problems auswirkt.

Folgede Invariante wurde im Buch angeben:

```
func c(k uint) bool {
      if k == r {
        return nW == 0 && Blocked(w) == 0
      }
      return nR == 0 && nW == 0 // w
}
```

Diese Eintrittsedinugn ist ohne weiteres nicht fair. Sie gibt einem der Parteien immer das recht noch einen weiteren zugriff auf den kritischen Abschnitt zu starten und verweigert dies der anderen, so lange eine die Hoheit über den kA hat. (so können sich 2 abwächeselnde schreiber für immer die Konrtrolle an sich reißen). Dieses Problem lässt sich lösen indem der Zugriff nicht gewährt, wenn ein anderer Prozess aus einer anderen Klasse zugriff fordert.
Eine einfache Variation:

Prozess	Bedingung	<IN >	<OUT >
Leser	nW == 0 && (prio == leser || bW == 0)	nW++	$nW--$; prio = schreiber
schreiber	nS == 0 && (prio == leser || bS == 0)	nS++	$nS--$; prio = leser

Diese Veränderungen ermöglichen es einem Prozess einer Klasse nur dann ebenfalls in den kA einzutreten, wenn kein Prozess der anderen Klasse ebenfalls eintritt begehrt. Die Variablen bW und bS werden vom universellen kritischen Abschnitt bereitgestellt.

Aufgabe 2

Bei dieser Aufgabe soll das Problem der speisenden Philosophen mithilfe des universellen kritischen Abschnitts gelöst werden.

Prozess	Bedingung	<IN >	<OU
p	state[left(p)] != dining &&	state[p] == dining	state[p] == satisfied;
	state[rigth(p)] != dining &&		state[rigth(p)] == starv
	(state[left(p)] != starving &&		state[left(p)] == starvi
	state[rigth(p)] != starving)		

Somit ist es möglich, dass die anderen Philosophen den vorzug erhalten, nachdem man selber gespeist hat.

Aufgabe 3

Eine großes Probelm das eine Lösung des Speisend Philosophen Problems mit hilfe von Semaphoren mit sich bringt ist das verhungern. Ein einzelner Philosoph kann unter umständen nicht mehr an seine linke gabel kommen, da sie immer von seinem Nachbarn in besitz gehalten wird und niemals zeitglich zu seiner Rechten freigegeben wird.

Aufgabe 4

a) Ein Semaphore der sich durch eine FIFO Queue realisiert hat folgendes Interface.
type Semaphore struct val int q Queue

Der Wert Wert val ist der entsprechende Wert mit dem der Semaphore initialisiert wird und die Menge der freien Ressurcen verwaltet werden. Die Queue q ist die Warteschlange mit der alle blockierten Prozesse hinterlegt werden. Für die Funktionen gilt folgendes Verhaletn:

P() Das verhalten von P hängt davon ab welchen wert die Varialbe val hat. Die wird in jedem Fall inkrementiert (in machen implementationen wird sie auch um eine übergebenen Wert inkrementiert). Dannch wird überprüft ob der wert nun kleiner oder gleich 0 ist. Falls dies der Fall ist, so wird er Prozess vom Betriebsystem blockiert und in die FIFo Schlange hinten hineingeworfen.

V() Hier wird val zunnächst erhöht. Ist sein Wert dannach immer noch kleiner oder gleich 0. So sind noch andere Prozesse in der Warteschlange. Nun wird der erste Prozesse in der Warteschlange deblockiert und auf rechenbereit gesetzt. (oder eine äquivalente Zahl an Prozessen.)

b) Das Konvoi Probelm kann dardurch behoben werde, dass nicht nach der FIFO Reihenfolge deblockiert wird, sondern in der Deblokade der Prozesse ein stochastisches element eingefügt wird. So kann z.B einfach ein willkürlicher Prozess bei einem aufruf von V() auf bereit gesetzt werden.

c) Die ist in Go realisierbar. Die Go Bibliotheken ermöglichen es auf die low-level Systemcalls zuzugreifen und ein Signal wie SIGTSTP zu senden. Dies unterbricht die Ausführung des aktuellen hreads. Allerdings gehen die meisten Vorteile der Sprache die goroutinen verloren. Mit diesen ist so etwas schwieriger zu implementieren, aber hier könnte so etwas mit Channels gemacht werden.