alp4 - Übungszettel 3

Johannes Rohloff

3. Mai 2013

Aufgabe 1

Erläutern Sie die Funktionsweise binärer Semaphore zur Sicherstellung des gegenseitigen Ausschlusses mehrerer Prozesse an folgendem Beispiel, wobei Sie den jeweiligen Zustand in Tabellenform protokollieren:

Initialisierung mit 1; Prozess 1 ruft P auf; Prozess 2 ruft P auf; Prozess 3 ruft P auf; Prozess 1 ruft V auf; Prozess 4 ruft P auf; Prozess 3 ruft V auf; Prozess 5 ruft P auf.

Tabelle:

Prozess	Aufruf	Semaphore	Bemerkung
	I	1	Initialisierung
1	P	0	1. im k.A.
2	P	-1	2 blockiert
3	P	-2	3 blockiert
1	V	-1	2. im k.A.
4	P	-2	4 blockiert
3	V	-2	3. im k.A.
5	P	-3	5 blockiert

Aufgabe 2

a) Ein Prozess kan nicht auf zwei Semaphore gleichzeitig warten. Wartet ein Prozess auf einen Semaphore so wird sein Prozesszustand auf blockiert gesetzt. Somit kann er während er auf den ersten Semaphore wartet nich auf einen anderen Semaphore warten. Ein Prozess kann darauf warten, dass er von einem Semaphore aufgewacht wird und dannach auf einen weiteren Semaphore warten. Allerdings kann ernicht auf zwei verschiedenen Semaphore warten und somit keine zwei disjunkten Eintrittsbedingungen, die über Semaphore realisiert sind, haben.

b) Solange ein Prozess auf das Signal eienes Semaphores wartet ist er blockiert. Deshalb kann er aus sicht des Betriebsystems nicht rechenbereit sein. Sobald er dei Methode P() aufgerufen hat ist er entweder in der kritischen sektion oder blockiert. Das heißt aus der Sicht des Betriebsystems ist ein wartender Prozess nicht rechenbereit.

c) Ein solcher Ansatz ist wünschenswert. Es sind situationen denkbar in denen verschiedene ähnliche Ressurcen für eine Aufgabe zur verfügung stehen und der aufrufende Prozess das schnellstmögliche dieser Ressurcen aufrufen will. Hätten alle dieser Ressurcen nur einen Semaphore, so könnte man nicht auf das schnellste freiwerden warten sondern müsste man müsste sich eine einzige Ressurce aussuchen und auf diese warten. Das warten auf verschiedene Signale kann also durchaus sinnvoll sein! Allerdings sollte eine solche Implementierung dem aufrufer nicht nur aufwecken sondern ebenfalls mitteilen welches Signal ihn grade aufgeweckt hat. Eine solche Implementierung könnte allerdings auch Probleme mit sich brinegn. So bleiben Fragen der Priorisierung und der Gleichzeitigkeit beim eintritt von mehr als 2 Signalen. Hier ist es schwierig sich ein defi niertes verhalten zu überlegen, dass Korrektheit aufweist.

Aufgabe 3

Beweisen sie, das der folgende Algorithmus gegenseitigen Auschluss sichert ist:
Das abstrakte Datenmodell:

```
type Imp struct {
      val int
      cs, mutex sync.Mutex
}
```

Die Implementierung

```
func New (n uint32) *Imp {
      x:= new (Imp)
      x.val = n
      if x.val == 0 {
        x.cs.Lock()
      }
      return x
}

func (x *Imp) P() {
      x.cs.Lock()
      x.mutex.Lock()
      x.val–
      if x.val >0 {
        x.cs.Unlock()
      }
      x.mutex.Unlock()
}

func (x *Imp) V() {
      x.mutex.Lock()
      x.val++
      if x.val == 1 {
        x.cs.Unlock()
      }
      x.mutex.Unlock()
}
```

Gegenseitigen Ausschluss : Der hier vorgestellte Implementierung erfüllt den gegenseitigen Ausschluss. Dies soll im folgenden gezeigt werden. Der hauptsächliche Fokus

liegt hierbei auf den beiden mutexen: cs und mutes. Diese sichern zu, dass die Variable n geschützt ist, sowie implementieren sie das Blockieren des Semaphores.

Bei der Untersuchung der Auschlusseigenschaft konzentrieren wir uns auf 2 Prozesse (P1 und P2). Sollte es mehrere geben, kann dies mit rückführung auf alle Fälle die hier behandelt werden einfach gezeigtwerde, dass der Ausschluss immer noch gilt. Außerdem vereinbaren wir das n=1 bei der Initialisierung. Auch hier gilt das höhere Werte für n am Ausschlussverhalten nichts ändern. Unterscheiden wir folgenden Szenarien:

1. P1 im kritischen Abschnitt (nach P()) und P2 ruft P() auf.
2. P1 und P2 rufen zeitgleich P() auf

Fall 1: Es glit: cs ist Locked und mutex ist Unlocked, n ist auf 0. Der Prozess 2 durchläuft das Aufrufprotokoll und bleibt bei dem sperren von cs hängen. Er kann seinen eintritt erst beenden, wenn vorher V() aufgerufen wurden. Soweit v erhält er sich spezifisch dem Protokoll. Alle anderen Prozess bleiben an der gleichen stelle hängen

Fall 2: Hier kommt es nur darauf an welcher der beiden Prozesse den mutext cs als erstes Sperren darf. Selbst wenn beie Prozesse gleichzeitig durch diesen Mutex kommen (im fall n >1), so kann imer nur einer der Prozesse auf n schrieben, da dieser durch einen weiteren Mutext mutex geschütrzt ist.