Verslag Uppaal-model

Readers/writers-probleem

David Korsman (s4619579) Wietze Mulder (s4557557)

31 oktober 2016

Het readers/writers-probleem

Het readers/writers-probleem is als volgt gedefinieerd: de data is gedeeld met meerdere processen. De data kan van alles zijn, bijvoorbeeld een bestand, een blok in het geheugen of zelfs een registerbank van een processor. Er zijn twee groepen processen, de eerste groep leest alleen de data (readers) en de ander schrijft alleen naar de data toe (writers). Ook zijn er een aantal regels:

1. Er mogen meerdere readers tegelijk de data lezen.
2. Er mag maar een writer tegelijk naar de data schrijven
3. Als er een writer bezig is met de data, mag een reader het niet lezen

Uitwerking

Het boek gaat in op meerdere oplossingen. De eerste beschrijft een eenvoudige oplossing om door middel van een counter te kijken of er geen readers zijn zodat de writer kan schrijven. Deze oplossing is echter niet optimaal omdat dit voor starvation bij de writer kan zorgen als er veel readers zijn. De tweede oplossing, waar wij tevens een Uppaal-model van hebben uitgewerkt, geeft de writers voorrang om de data te gebruiken. Als er een writer wil schrijven mogen er geen nieuwe readers meer bij de data en komt de writer altijd een keer aan de beurt. Starvation zal daarom deze keer niet voor komen. Het boek geeft ook een derde oplossing door middel van message passing en een controller. Readers en writers kunnen bij de data door toegang te vragen bij een controller. Als er een writer toegang vraagt, laat de controller alle toegangvragende readers wachten en krijgt de writer als eerste toegang. Vanwege de complexiteit van Uppaal in deze oplossing en onze toch nog bescheiden ervaring met Uppaal hebben we er voor gekozen om deze niet uit te werken.

Onze uitwerking is identiek aan de uitwerking van het boek. Er worden meerdere semaforen gebruikt voor de readers. Dit heeft te maken met het dat we de rij voor `rsem` kort willen houden zodat writers snel bij de data kunnen. Voor de twee `readcounter` en de `writecounter` zijn ook beide extra semaforen nodig. Wij hebben het voorbeeld-semafoorbestand van deze site[1] en deze uitgebreid met meerdere semaforen en verschil gemaakt tussen readers en writers.

Synchronisatie

Synchronization properties van Downey:

- If there is only one thread that is ready to run, the scheduler has to let it run.
- If a thread is ready to run, then the time it waits until it runs is bounded.

[1] http://www.ita.cs.ru.nl/publications/papers/fvaan/MCinEdu/semaphores.html

- If there are threads waiting on a semaphore when a thread executes signal, then one of the waiting threads has to be woken.
- If a thread is waiting at a semaphore, then the number of threads that will be woken before it is bounded.

Onze Uppaaluitwerking voldoet aan deze vier punten van Downey. Dit is vooral omdat in het voorbeeld-semafoor-bestand de semaforen correct zijn. Ook is er bij ons geen starvation mogelijk. Wel weten wij door onze queries in Uppaal dat de processen netjes synchroon lopen. We controleren bijvoorbeeld of er een situatie is waar er meerdere readerprocessen (om exact te zijn 2) de data aan het lezen zijn. Ook controleren wij op overflow, of writers niet tegelijk aan het schrijven zijn en controleren we of er deadlocks zijn. De laatste drie queries zijn er voor om de correctheid van het Uppaal-model te bewijzen.

Reflectie

In dit project hebben we iets meer van semaforen opgestoken. Ook complexere modellen in Uppaal implementeren hebben we meer in de vingers gekregen. Het was prettig dat de deadline was verschoven zodat er wat druk van de ketel kon bij deze twee druk studerende mannen midden in hun tentamenweek. Samen hebben we de beginselen gelegd van het Uppaal-model en vooral het begrijpen van de template van de site[1]. David heeft daarna de uitwerking van het boek in ons Uppaalmodel geïmplementeerd. Wietze heeft vervolgens het verslag uitgewerkt.