Bericht uit 

Voor de verandering trakteer ik u eens op een column met technische diepgang. Het onderwerp dat ik ter hand neem is de matige implementatie van thread-synchronisatie in Java en twee oplossingen hiervoor die voortkomen uit het werk van de bekende Britse informaticus Tony Hoare.

Foutjes in Java (1)

Zoals u allen weet is een van de voordelen van Java dat er in de taal faciliteiten aanwezig zijn voor "multi-threading": het tegelijkertijd kunnen uitvoeren van meerdere stukken Java-code. Het aardige van multi-threading is dat applicaties kunnen worden geschreven om onafhankelijke deeltaken in aparte "threads" uit te voeren, die ieder zelfstandig recht hebben op de CPU. Dit komt de architectuur van de applicatie ten goede, en verbetert in de meeste gevallen ook het externe gedrag van de applicatie (vanuit de gebruiker gezien "hangt" de applicatie minder vaak). Echter, in een "multi-threaded"-omgeving horen ook faciliteiten om threads onderling te coördineren. Die coördinatie is bijvoorbeeld nodig als twee of meer threads gezamenlijk een datastructuur gebruiken. Terwijl de ene thread die datastructuur wijzigt, moeten de andere threads die de data willen raadplegen even wachten totdat de wijziging klaar is. Een ander voorbeeld van de noodzaak tot coördinatie is als de threads gebruik willen maken van een gedeelde hulpbron waarvan slechts een beperkt aantal voorhanden is (bijvoorbeeld een poel met databaseconnecties). Een thread die een connectie uit een lege poel wil pakken zal moeten wachten totdat een andere thread een connectie terugstopt in de poel. Het coördineren van threads is gelukkig een opgelost probleem. De vier meest bekende methodes van thread-synchronisatie zijn semaforen, monitoren, rendez-vous en CSP (Communicating Sequential Processes). Sun heeft er in haar oneindige wijsheid voor gekozen om Java uit te rusten met een variant op het 'monitor'-principe. Dit is om twee redenen jammer. De belangrijkste van die redenen is dat Sun er om onduidelijke redenen voor heeft gekozen om op een tamelijk belangrijk punt af te wijken van Hoare's monitor-specificatie. En dat was nu eens geen goed plan, want de door Sun ingebrachte variatie brengt problemen met zich mee die in de "echte" monitoren niet voor kunnen komen! Peter Welch van de Engelse universiteit in Kent heeft dit op werkelijk briljante wijze aangetoond met een scenario genaamd:"Wot, No Chickens!" (zie http://www.cs.ukc.ac.uk/projects/ofa/java-threads/0.html). Het blijkt dat in specifieke gevallen waarin threads "strijden" om toegang tot een beperkte hoeveelheid hulpbronnen bepaalde threads continue buiten de boot vallen en nooit met een van de schaarse resources aan de haal kunnen gaan. Dit betekent dat die thread in een oneindige lus komt van wachten en in de rij aansluiten. Voor waar geen prettig vooruitzicht! In een productiesysteem kan dit betekenen dat een thread die een verzoek van een klant aan het afhandelen is nooit aan de beurt komt om bijvoorbeeld een databaseconnectie uit de poel te halen. De kans dat dit gebeurt is niet groot, maar hij is zeker aanwezig! Wat nu te doen? Gelukkig biedt de academische wereld hier uitkomst. Na monitoren is Tony Hoare aan de slag gegaan met een wiskundige aanpak voor het modelleren van onderling communicerende threads (processen). Het resultaat hiervan is CSP: "Communicating Sequential Processes". In de CSP-algebra kunnen threads slechts op voorgeschreven manieren met elkaar communiceren. Een met CSP gemodelleerd proces kan wiskundig worden geanalyseerd en er kunnen uitspraken worden gedaan over de correctheid van die communicatie. Teams aan de universiteiten van Twente en Kent hebben Java libraries ontwikkeld waarmee Java threads op CSP-wijze met elkaar kunnen communiceren. Die libraries kennen (natuurlijk) niet de nadelen van de standaard Java-monitoren, en het verdient naar mijn mening dan ook aanbeveling voor Java software engineers om die libraries eens aan een nader onderzoek te onderwerpen (zie http://www.rt.el.utwente.nl/javapp). Maar ja, wie ben ik?

Jos Visser
is de schoonheidsfout in het team van Open Solution Providers en te bereiken via josv@osp.nl.