Universiteit van Amsterdam

CPU-scheduling-algoritmen

Het *Round-Robin*-algoritme en diens varianten met effectieve prestatieverbeteringen

25-05-2018

Tutor:
W.R.J. Vermeulen

Practicumgroep:
A1 - ALGOL

Cursus:
Besturingssystemen

Student:
R.A.J. Wacanno
11741163

1 Introductie

In een tijd waar de samenleving steeds meer afhankelijk is geworden van computersystemen is er een continue zoektocht naar manieren om deze systemen beter te laten presteren. Er zijn veel facetten van een systeem die invloed hebben op de prestaties, maar het efficiënter maken van bepaalde facetten kan een grotere invloed hebben op de algemene prestatie dan verbetering bij anderen. Volgend de wet van Amdahl loont het om veelvoorkomende aspecten te verbeteren (Amdahl, 1967).

Een dergelijk veelvoorkomend onderdeel binnen een computersysteem is het *CPU-scheduling*-algoritme. Dit algoritme heeft als taak het coördineren van *CPU*-gebruik door meerdere processen. Binnen de groep van *CPU-schedulers* is de *Round-Robin* een interessante vanwege zijn gebruik als basis voor complexere *CPU-scheduling*-algoritmes. Een voorbeeld van een algoritme dat van de *Round-Robin* gebruik maakt is het *CFS*-algoritme (*Completely Fair Scheduling*) dit in het Linux besturingssysteem wordt gebruikt (Wong, Tan, Kumari & Wey, 2008). Het gebruik als basis voor andere algoritmen maakt dat een verbetering van *Round-Robin* grote effecten kan hebben op algemene prestatieverbeteringen; hetgeen conform de wet van Amdahl is.

Hierom zal in het volgende schrijven de volgende vraag centraal staan: Op welke wijze kunnen aanpassingen aan de standaard-implementatie van *Round-Robin* gedaan worden waardoor prestatieverbeteringen behaald kunnen worden? Ook zal er gekeken worden of deze verbeteringen in realistische scenario's haalbaar zijn en of er in bepaalde gevallen niet beter een ander en meer geschikt algoritme geïmplementeerd kan worden.

2 De *Round-Robin* met aanpassingen

Allereerst is de vraag natuurlijk hoe *Round-Robin* aangepast kan worden om verbeterde prestaties te verkrijgen.

2.1 Het *Round-Robin*-algoritme

Voordat verbeteringen van het *Round-Robin*-algoritme (*RR*) kunnen worden besproken, dient eerst kennis genomen te worden van de werking van *RR* in zijn simpele vorm. Een uitgebreide beschrijving hiervoor was te vinden in het boek *Operating System Concepts* (Peterson & Silberschatz, 1985).

RR vervult zijn taak als *CPU-scheduling*-algoritme door gebruik te maken van een uitvoeringsrij. In deze rij zet RR alle processen die op dat moment uitgevoerd moeten worden. De volgorde van deze processen is in dit geval niet van belang. Vervolgens wordt steeds het proces dat vooraan in de rij staat hieruit gehaald zodat deze voor een bepaalde tijd van de *CPU* gebruik kan maken. Hoeveel tijd een proces per keer krijgt wordt aangeduid als het tijdskwantum; hier soms afgekort als *T*. Dit tijdskwantum wordt, in deze standaardimplementatie van de *Round-Robin* vooraf bepaald en vastgelegd in de code. Zodra een proces voor de duur van *T* van de *CPU* gebruik heeft gemaakt wordt deze weer achteraan in de uitvoeringsrij gezet. Een voorbeeld hiervan is met de processen `1` tot en met `5` in figuren 1 en 2 weergegeven.

1-2-3-4-5

Figuur 1: Uitvoeringsrij op tijd 0

2-3-4-5-1

Figuur 2: Uitvoeringsrij op tijd $0+T$

De *Round-Robin* gaat op cyclische wijze door de lijst heen. Mocht een proces binnen een tijdskwantum klaar zijn met uitvoeren, dan zal deze stoppen met gebruikmaken van de *CPU*, maar *Round-Robin* zal tot het einde van het betreffende tijdskwantum wachten tot het een nieuw proces de kans geeft uit te voeren. Met de eerdergenoemde processen `1` tot en met `5`

1	2	3	4	5	1	2	3	4	5	2	3	5	2	5	...

Tabel 1: Voorbeeld van *CPU*-toewijzing per tijdskwantum *T* gedurende een gegeven periode

2.2 Verbeteringen van de *Round-Robin*

2.2.1 Verbetering bij volgordebepaling

Een allereerste verbeterde implementatie van *Round-Robin* is *SFRR* (*Shortest First Round-Robin*) (Yadav, Mishra, Prakash & Sharma, 2010). Dit algoritme is in vrijwel alles identiek aan de standaard *Round-Robin*, ware het niet dat hij de processen niet in de volgorde van binnenkomst in de uitvoeringsrij zet. In plaats daarvan worden processen van voor naar achter gezet op basis van een steeds grotere *burst*-tijd. Deze *burst*-tijd is een schatting van de tijd die een proces in totaal van de *CPU* gebruik moet maken.

2.2.2 Verbetering bij tijdskwantumbepaling met een bekende procestijd

Veel verbeteringsalgoritmes van de *RR* proberen efficiëntere prestaties te behalen door het tijdskwantum van de *Round-Robin* op een meer dynamische manier te bepalen. Dit betekend dat de grootte van het kwantum af hangt van de processen die in de uitvoeringsrij staan.

Voorbeelden van implementaties van dit principe zijn: *RARR* (*Re-Adjusted Round-Robin*) (Behera, Mohanty & Nayak, 2011), *SARR* (*Self-Adjustment Round-Robin*) (Matarneh, 2009) en *SBRR* (*Shortest Burst Round-Robin*) (Hiranwal & Roy, 2011).

In essentie zijn deze varianten gelijkend aan het eerdergenoemde *SFRR*-algoritme. Ook zij zetten de binnengekomen processen op volgorde van *burst*-tijd. Dit wordt echter niet gedaan voor directe prestatieverbetering, maar voor het bepalen van de mediaan van de *burst*-tijden van de in de uitvoeringsrij aanwezige processen. Met deze mediaan wordt het tijdskwantum bepaald. Het bepalen van tijdskwantum T met n processen P wordt in de volgende vergelijking geïllustreerd:

$$T = \overline{x} = \begin{cases} P_{\frac{n+1}{2}} & \text{als } n \text{ oneven is} \\ \frac{1}{2}(P_{\frac{n}{2}}) + (P_{\frac{n+1}{2}}) & \text{als } n \text{ even is} \end{cases}$$

Voor *SARR* wordt aan deze berekening nog een extra berekening toegevoegd (Matarneh, 2009). Mocht de berekende $\overline{x}$ onder de 25 ms liggen, dan wordt deze aan 25 ms gelijkgesteld; hetgeen in de volgende vergelijking is uitgedrukt:

$$T = \begin{cases} \overline{x} & \text{als } \overline{x} \geq 25 \\ 25 & \text{als } \overline{x} < 25 \end{cases}$$

Het is belangrijk om te beseffen dat alle drie de algoritmen bij iedere start van een tijdskwantum de lengte van het nieuwe kwantum herberekenen op basis van de op dat moment in de uitvoeringsrij aanwezige processen.

Al deze algoritmen gaan er op hun beurt van uit dat de *burst*-tijd van ieder proces bekend is.

2.2.3 Verbetering bij tijdskwantumbepaling met een onbekende procestijd

Wanneer een verbeteringen op het gebied van de tijdskwantumbepaling moet worden behaald in een context waar de *burst*-tijd niet zomaar gegeven wordt, moet gebruik worden gemaakt van een ander algoritme. Dit kan gedaan worden met *DTSRR* (*Dynamic Time Slice Round-Robin*) (Bakare & Yusuf, 2017). Dit algoritme is in staat om zelf een *burst*-tijd per proces te schatten op basis waarvan een tijdskwantum op basis van alle processen in de uitvoeringsrij bepaald kan worden.

DTSRR bepaalt de *burst*-tijd op basis van 3 eigenschappen:

- het aantal instructies van een bepaald proces;
- het aantal processorcycli per bepaalde instructie (dit kan namelijk per instructie verschillen)
- en de processortijd per cyclus.

Zodra deze *burst*-tijden berekend zijn kan wederom per start van een nieuw tijdskwantum diens duur berekend worden. Hoe *DTSRR* deze duur berekend verschilt wel van de drie eerdergenoemde algoritmen. *DTSRR* neemt als tijdskwantum het naar boven afgeronde gemiddelde van de *burst*-tijden van de in de uitvoeringsrij aanwezige processen. Hetgeen formeel volgens de deze vergelijking beschreven wordt bij n processen P en een *burst*-tijd B_{P_i} van proces P_i:

$$T = \left\lceil \frac{\sum_{i=1}^{n} B_{P_i}}{n} \right\rceil$$

3 Gevolgen van de verbeteringen

Nu is het natuurlijk de vraag in hoeverre de aanpassingen op *Round-Robin* een verbetering opleveren in vergelijking met het standaardalgoritme.

De *SFRR* laat een prestatieverbetering zien als het gaat om wachttijden, maar dit is enkel beargumenteerd binnen een hypothetisch scenario (Yadav et al., 2010).

RARR (Behera et al., 2011) en *SBRR* (Hiranwal & Roy, 2011) — welke zoals uit de vorige sectie bleek in feite dezelfde algoritmen zijn — presteren beiden ook beter dan de standaard *Round-Robin*. *SARR* presteert in vergelijking tot de hiervoor genoemde twee beter bij processen met een kleine *burst*-tijd (Matarneh, 2009). Dit is een gevolg van de minimum tijdskwantum van 25 ms, welke minder proceswisselingen tot gevolg heeft.

DTSRR laat van alle variaties met verbeterde tijdskwantumbepaling een verbetering zien die ook het meest realistisch is. Dit algoritme gaat er namelijk niet van uit dat de *burst*-tijd van tevoren beschikbaar is en is qua implementatie dus ook het meest haalbaar (Bakare & Yusuf, 2017).

3.1 Vergelijking met andere algoritmen

Hoewel de behandelde algoritmen als *CPU-scheduler* in theorie beter presteren, zijn er, wanneer zij vergeleken worden met andere algoritmen (Goel & Garg, 2013), in bepaalde scenario's gunstigere algoritmen welke in de praktijk beter zullen presteren van de aangepaste *RR*'s.

Wat hierbij vooral een grote rol speelt zijn de extra tijdskosten die een in vergelijking complexer algoritme met zich mee kan brengen. In het geval van *SFRR* zullen deze extra kosten liggen in het op goede volgorde zetten van de processen bij het plaatsen in de uitvoeringsrij (Yadav et al., 2010). Bij *RARR* (Behera et al., 2011), *SARR* (Matarneh, 2009) en *SBRR* (Hiranwal & Roy, 2011) is de *scheduler* extra tijd kwijt met het continu berekenen van het tijdskwantum. *SARR* verminderd het aantal keren dat van proces gewisseld wordt door een minimum tijdskwantum van 25 ms te handhaven, maar dit neemt niet weg dat er tijd verloren wordt.

In een context waar steeds processen met ongeveer gelijke *burst*-tijf moeten worden verwerkt en het tijdskwantum nagenoeg gelijk is aan de *burst*-tijd, zal de *Round-Robin* zich meer gedragen als een *FIFO-scheduler* (First In First Out) (Goel & Garg, 2013). In tegenstelling tot een daadwerkelijke *FIFO* is de verbeterde *RR* echter nog steeds significant veel tijd krijt met administratieve berekeningen. In een dergelijk geval zou het dus lonen om in plaats van een verbeterde *RR* een *FIFO* te implementeren.

4 Discussie

4.1 Conclusie

Literatuur

Amdahl, G. M. (1967). Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the april 18-20, 1967, spring joint computer conference* (pp. 483–485). doi: 10.1145/1465482.1465560

Bakare, O. & Yusuf, N. (2017, jun). Improved dynamic time slice round robin scheduling algorithm with unknown burst time. *International Journal of Computer Applications*, *167*(7), 21–28. doi: 10.5120/ijca2017914305

Behera, H. S., Mohanty, R. & Nayak, D. (2011). A new proposed dynamic quantum with re-adjusted round robin scheduling algorithm and its performance analysis. *arXiv preprint arXiv:1103.3831*. doi: 10.5120/913-1291

Goel, N. & Garg, R. (2013). A comparative study of cpu scheduling algorithms. *arXiv preprint arXiv:1307.4165*.

Hiranwal, S. & Roy, K. (2011). Adaptive round robin scheduling using shortest burst approach based on smart time slice. *International Journal of Computer Science and Communication*, *2*(2), 319–323.

Matarneh, R. J. (2009). Self-adjustment time quantum in round robin algorithm depending on burst time of the now running processes. *American Journal of Applied Sciences*, *6*(10), 1831. doi: 10.3844/ajassp.2009.1831.1837

Peterson, J. L. & Silberschatz, A. (1985). *Operating system concepts*. Addison-Wesley Longman Publishing Co., Inc.

Wong, C. S., Tan, I., Kumari, R. D. & Wey, F. (2008). Towards achieving fairness in the linux scheduler. *ACM SIGOPS Operating Systems Review*, *42*(5), 34–43. doi: 10.1145/1400097.1400102

Yadav, R. K., Mishra, A. K., Prakash, N. & Sharma, H. (2010). An improved round robin scheduling algorithm for cpu scheduling. *International Journal on Computer Science and Engineering*, *2*(04), 1064–1066.