ESSAY SCHEDULING POLICIES_

OSM-S

Marc Vervoorn (1654525)

28 juni 2024

Docent: Bram Knippenberg

HAN_UNIVERSITY
OF APPLIED SCIENCES

INHOUDSOPGAVE

INLEIDING

Voor het vak OSM-S is mij de opdracht gegeven om verschillende algoritmes met elkaar te vergelijken. In dit document leest u welke algoritmes ik heb vergeleken, hoe ze op de verschillende factoren scoren, en welk algoritme in welke situatie goed te gebruiken is.

1 FAIR SCHEDULING VS PRIORITY SCHEDULING

In dit hoofdstuk zal ik ingaan op het verschil tussen Fair scheduling policies en Priority scheduling policies. Dit zijn 2 manieren om taken in te plannen, maar ze doen dit verschillend.

1.1 Fair scheduling policies

Fair scheduling gaat op basis van een gelijkmatige verdeling van taken. Dit houd in dat voor het algoritme elke taak even belangrijk is. Daardoor heeft elke taak evenveel kans om als eerste te starten, en heeft een taak zelf geen invloed op zijn starttijd.
Bijvoorbeeld een programma dat random een taak uit een lijstje moet halen, elke taak is even belangrijk, en maakt dus net zoveel kans om gekozen te worden.

1.2 Priority scheduling policies

Waar bij fair scheduling elke taak dezelfde kans maakt om als eerste te starten is dat bij priority scheduling zeker niet het geval. Bij priority scheduling is er enige vorm van prioritering van taken, bijvoorbeeld op basis van tijdsbestek per taak of de tijd wanneer een taak binnenkomt. Deze prioritering kan dus op verschillende manieren gedaan worden (afhankelijk van gekozen algoritme), 2 van deze manieren zullen uitgelegd worden in volgende hoofdstukken.
Neem bijvoorbeeld een brood dat gebakken word, eerst moet het deeg gemaakt worden, dan moet er een broodvorm van gekneed worden en vervolgens gaat het in de oven, in een vaste volgorde. Deze volgorde bepaald dan de prioritering.

In het kort: bij fair scheduling heeft elke taak dezelfde “kans” om als eerste uitgevoerd te worden. Bij priority scheduling heeft elke taak een prioriteringsscore, waardoor de belangrijkste taak eerst uitgevoerd wordt.

2 BEGRIPPEN

2.1 Stable

Een stabiel planningsalgoritme is een algoritme dat hetzelfde eindigt als er met dezelfde waardes opnieuw geplant word. Met andere woorden: “als je 10x (of vaker) precies hetzelfde in het algoritme stopt, komt er elke keer hetzelfde uit.” Ook word ermee bedoelt dat bij kleine wijzigingen niet direct de uitkomst geheel veranderd, maar alleen daar waar dat nodig is.
Bijvoorbeeld een algoritme dat berekent hoeveel bakstenen er nodig zijn voor een gebouw op basis van een 3D tekening. Dan wil je wel dat wanneer je 3x dezelfde tekening geeft, het algoritme ook 3x dezelfde uitkomst geeft voor wat betreft het aantal bakstenen. Als dit telkens verschilt weet je namelijk nog niet zeker hoeveel je er nodig gaat hebben.

2.2 Optimal

De letterlijke betekenis voor optimaal is: “zo gunstig mogelijk”. Voor een algoritme betekend dit dat er zo efficiënt mogelijk gebruik word gemaakt van machines. Dus dat de machines zo min mogelijk tijd stil staan. Denk aan een fabriekshal, je wilt dat er zoveel mogelijk productie gedraaid word, en daarom wil je dat elke machine zo min mogelijk stil staat. Dat kan je doen door heel efficiënt te plannen en op die manier alle machines (bijna) constant te laten draaien.

2.3 Responsive

Een responsive algoritme is een algoritme dat binnenkomende taken niet te lang laat wachten, maar deze binnen een bepaalde tijdsspan uitvoert. Dit voorkomt dat een taak veel te laat (of zelfs niet) wordt afgehandeld. Denk bijvoorbeeld aan het koelen van een machine, als een taak is om intern de ventilator aan te zetten moet dit wel binnen een bepaalde tijdsspan gebeuren, om ervoor te zorgen dat de machine niet oververhit.

2.4 Robust

Een robuust algoritme is een algoritme dat tot een bepaald niveau met fouten om kan gaan. Dit wilt zeggen, als er in 1 van de tijdlijnen/machines een fout ontstaat, heeft dit geen gevolgen voor andere tijdlijnen/machines. Dit zijn fouten in taken die uitgevoerd worden, stel dat er in taak 1 een fout zit, waardoor deze taak niet (volledig) kan worden uitgevoerd (bijvoorbeeld deel 1 op 3 verlijmen, terwijl dit niet past), word er bij een robuust algoritme voor gezorgd dat de taken 2,3,4, etc. wel uitgevoerd (kunnen) worden. De andere machines kunnen dus gewoon doorwerken om ervoor te zorgen dat er niet een te grote achterstand van taken ontstaat. Denk aan een fabriekshal die een foutief gemaakt item uit de fabrieksband haalt en de rest door laat gaan, op die manier hoeft niet elk item te wachten totdat de fout is opgelost, maar kan de rest gewoon verder het proces vervolgen.

Bron:

scheduling_20210906%20(1).pdf

3 2 VARIANTEN UITGEWERKT

3.1 Roud Robin – Fair Policy

3.1.1 Algemene Beschrijving

Het Round Robin algoritme werkt op basis van “timesharing”. Dit houd in dat de beschikbare tijd verdeeld word over de taken die uitgevoerd moeten worden. Elke taak krijgt op deze manier een tijdsslot. Als een taak nog niet klaar is na dit tijdsslot, word de taak afgebroken. De taak word weer verder afgehandeld als de tijdsloten weer opnieuw worden verdeeld. Als een machine eerder klaar is met zijn taak dan dat het tijdsslot is afgelopen, word er direct een nieuwe taak ingepland voor die machine. Dit proces word herhaald totdat alle taken zijn verwerkt/ uitgevoerd. Round Robin word ook wel time slicing genoemd, omdat er per tijdvak gewerkt word.

Bron:

(UNIVERSITY OF WISCONSIN–MADISON, 2008)

3.1.2 Score

Stable: Laag, afhankelijk van de drukte van de processor, zal er meer of minder tijd/ rekenkracht naar een ingeplande taak gaan. Daardoor is de uitkomst niet altijd hetzelfde als de invoer wel gelijk blijft, zeker op een computer die meerdere processen draait, kan het prima zijn dat de processor load nu zo is dat de taken anders worden verdeeld over de tijd.
Optimal: Laag, een taak word niet direct achter elkaar afgehandeld, een taak kan opgesplitst worden, waardoor het langer duurt dan dat je de taak in 1 keer helemaal afhandelt. Dit omdat je elke “ronde” een klein stukje van de taak uitvoert, omdat je de rekenkracht verdeeld over alle taken. Alle taken samen zijn hierdoor sneller afgerond, maar 1 losse taak duurt langer.
Responsive: Hoog, een binnenkomende taak word direct meegenomen in het volgende tijdsblok. Daardoor hoeft een taak niet tot nauwelijks te wachten voordat er begonnen wordt met het uitvoeren van deze taak en is er dus een hele lage response tijd, dat duidt op een hoge score.
Robust: Middel, als er een fout ontstaat in een taak, gaat dit natuurlijk elke ronde fout bij deze taak. Maar, omdat er elke ronde meerdere taken tegelijkertijd worden uitgevoerd, heeft deze fout geen directe gevolgen voor andere taken, deze kunnen gewoon door blijven draaien zoals ze deden.

3.1.3 Geschiktheid

Geschikt voor situaties waarin **responsiveness** erg belangrijk is. Denk aan het besturen van een auto, je wilt dat als jij aan het stuur trekt, de auto snel reageert, net als wanneer jij gas geeft of remt. Daarvoor moet het algoritme wel snel reageren en de berichten die daarvoor verstuurd moeten worden snel versturen naar de centrale regelcomputer van de auto.

Ongeschikt voor situaties waarin de uitkomst altijd hetzelfde moet zijn, denk aan een roostermaker die drie keer op genereren drukt, maar zonder wijzigingen toch echt hetzelfde rooster verwacht. (**stable**)

3.2 FCFS - Priority Policy

3.2.1 Algemene Beschrijving

Het FCFS algoritme plant nieuwe taken in op basis van hun “arrival time” (aankomsttijd), dit is de tijd wanneer het bekend is dat de taak ingeplant moet worden. De taak die het eerste aankomt, wordt als eerste uitgevoerd, de taak die als 2^{e} aankomt daarna, en zo verder. Er word als het ware gewerkt met een queue, als er te veel taken op enig moment zijn, dan worden ze toegevoegd aan de queue en deze queue word van langst wachtende taak, naar korst wachtende taak afgehandeld.

Voordelen:

1. Eenvoudig
2. Altijd eerlijk (eerste die erin komt, word ook als eerste behandeld)

Nadelen:

1. Er is geen prioritering mogelijk.

Bron: (javatpoint, n.d.)

3.2.2 Scores

Stable: Hoog; FCFS is een stabiel algoritme, omdat het kijkt naar de aankomsttijd van een taak, als je dus elke keer een vast rijtje taken geeft aan dit algoritme, en de volgorde veranderd niet, dan blijft de uitkomst altijd hetzelfde.
Optimal: Hoog; als een taak afgerond is, word er direct gekeken naar een volgend wachtende taak, waardoor een machine/processor niet stil hoeft te staan. Er word dan wel veel gebruik gemaakt van processor capaciteit, maar het is de meest optimale manier van het afhandelen van taken.
Responsive: Middel; als alle taken even lang duren, dan hoeven taken niet lang op elkaar te wachten en is het dus redelijk responsive, maar als er een hele lange taak op de planning staat en een korte taak wilt snel starten, dan moet deze korte taak alsnog erg lang wachten op zijn voorganger, dan is dit algoritme dus even niet responsive.
Robust: Laag; FCFS kan niet omgaan met fouten, als er in een eerdere taak fouten optreden, waardoor het systeem en/of het algoritme vastlopen of blijven hangen, dan is dat zo en worden overige taken niet of veel later (dan verwacht) uitgevoerd. Je kan dus niet goed vertrouwen op dit algoritme als het gaat om fouten.

3.2.3 Geschiktheid

Geschikt voor pakketsorteersystemen, omdat je wilt dat elk pakketje op tijd word afgehandeld, en het liefst in dezelfde volgorde als dat de pakketten het systeem in gaan. FCFS is hier geschikt voor omdat het een **stable** algoritme is. Ook is dit een **optimal** algoritme, wat ervoor zorgt dat pakketjes niet lang hoeven te wachten.

Ongeschikt voor medische gegevens die op tijd ergens afgeleverd moeten worden, omdat FCFS niet om kan gaan met fouten (**niet robust**) en je dus in theorie gegevens zou kunnen verliezen. Dit is in een medische setting natuurlijk uitermate vervelend en onwenselijk.

3.3 Completely Fair Scheduler – Fair Policy

3.3.1 Algemene Beschrijving

Het completely fair scheduler algoritme (later aangegeven als CFSA) is een fair policy algoritme voor het indelen van taken. CFSA kan taken groeperen en behandelen als 1 taak. Deze groep word een “task_struct” genoemd. Elke “task_struct” bevat een lijst van taken die daarbij horen. Binnen deze groepen worden taken gesorteerd op tijdsbestek. En, elke task_struct slaat op welke taak de kleinste execution time heeft. Er word ook een “maximum execution time” van elk proces berekend. Dit is de tijd die het kost voor het uitvoeren van een taak op een ideale processor wanneer alle processorkracht naar deze taak komt.

Als er een nieuwe taak gestart moet worden:

1. De taak met de kortste execution time word uitgevoerd.
2. Als de taak is afgerond, word deze verwijderd van de nog uit te voeren taken.
3. Als het lopende proces de maximale execution tijd heeft bereikt, en de taak nog niet uitgevoerd is dan word hij terug toegevoegd aan de uit te voeren taken, maar met een nieuwe execution time.
4. De taak met de kortste execution time wordt uitgevoerd.
5. (Algoritme treedt opnieuw in werking)

Bronnen:

(Molnár, 2021)

(ece.ubc.ca, 2016)

3.3.2 Scores

Stable: Hoog; Er word gerekend met tijden van taken, waardoor de uitkomst altijd gelijk is als het aantal taken en de tijdsduur van elke taak gelijk blijft en je het opnieuw uitvoert.
Optimal: Middel; Er word per ronde gewerkt, waardoor een taak steeds heel kort onderbroken word om opnieuw in te delen. Natuurlijk is dit in een software oplossing minimaal, omdat stilstand gaat om ms of zelfs nog kleiner, maar toch scoort dit algoritme daarom niet hoog op dit onderdeel.
Responsive: Heel hoog; doordat beschikbare tijd verdeeld word over openstaande taken hoeven taken praktisch nooit te wachten op elkaar, natuurlijk word dit een beetje vermomd, omdat het eigenlijk gewoon een spreiding is van uitvoeren van 1 taak. Maar er word wel extreem snel begonnen aan een taak.
Robust: Laag; Als er een fout ontstaat in een nu taak die nu uitgevoerd word, zal deze taak niet als “klaar” worden gelabeld, dus gaat dezelfde taak nog een keer op de “uitvoerstapel”. De fout ontstaat weer, en zo ontstaat er een loop.

3.3.3 Geschiktheid

Geschikt voor situaties waarin de uitkomst altijd hetzelfde moet zijn, denk aan een roostermaker die drie keer op genereren drukt, maar zonder wijzigingen toch echt hetzelfde rooster verwacht **(stable).**

Ongeschikt voor situaties waarin het erg belangrijk is dat er altijd doorgewerkt wordt, doordat dit algoritme erg slecht om kan gaan met fouten (**robust**), kan het zijn dat een taak waar een fout optreed opnieuw word ingeplant en er op die manier een loop ontstaat. Dat is zeer ongewenst en dus is dit algoritme niet geschikt om in te zetten op cruciale plekken binnen een organisatie/systeem.

3.4 SJN (Shortest Job Next) – Priority Policy

3.4.1 Algemene Beschrijving

Het SJN algoritme kijkt continu naar alle “execution times” van de verschillende taken die gedaan moeten worden. De taken worden gesorteerd op basis van deze tijden, dit sorteren gaat van hoog naar laag. De taak met de kortste execution time (tijd die nodig is om de taak uit te voeren) word als eerste uitgevoerd, daarna de taak met de op 1 na kortste execution time en zo verder. Totdat er geen taken meer over zijn.

Voordelen:

- Het is erg eenvoudig te implementeren.
- Een machine/ processor hoeft nooit stil te staan, want alles word na elkaar uitgevoerd.

Nadelen:

- Een processor kan snel overbelast raken, als er snel achter elkaar korte taken worden toegevoegd.

Bronnen:

(geeksforgeeks, 2022)

(Wikipedia authors, 2021)

3.4.2 Scores

Stable: Hoog; Dit algoritme is erg stabiel. Omdat het algoritme werkt op basis van tijdsduur van taken. Als je dus 10x dezelfde set taken meegeeft aan dit algoritme (met 10x dezelfde waardes), dan zal de uitkomst ook 10x hetzelfde zijn. Omdat het algoritme extreem afhankelijk is van de inputwaardes (tijdsduur)

Optimal: Heel Hoog; Een machine hoeft nooit stil te staan, omdat er nadat een taak klaar is, direct gekeken word naar een eventuele nieuwe taak die gestart kan worden, als die er is wordt deze onmiddellijk uitgevoerd, zodat de machine niet stil hoeft te staan.

Responsive: Laag, als er een nieuwe taak bij komt, word de taken set wel weer gesorteerd van laag naar hoog (tijdsduur). Dit betekend dat een lange taak heel lang moet wachten als er veel korte taken worden toegevoegd tussentijds.

Robust: Laag, als er een fout optreed in een taak, en er moet een nieuwe taak bepaald worden, word er alleen gekeken naar de lengte van de taken. Er word **niet** gekeken naar het feit of een taak al eerder is uitgevoerd. Dit zorgt ervoor dat als er een fout optreed in een uitvoerende taak, deze taak daarna weer start, omdat hij niet heeft kunnen eindigen en nog steeds de taak is met de hoogste prioriteit. Dat gaat natuurlijk fout, er ontstaat een loop.

3.4.3 Geschiktheid

Geschikt voor situaties waarin je elke keer dezelfde uitkomst wilt hebben, denk aan een programma dat jij als bedrijf gebruikt, en de uitvoer voor een langere periode wilt gebruiken, maar personen kunnen wel vaker de informatie opnieuw opvragen van het algoritme, dan wil je wel dat de informatie elke keer gelijk is voor iedereen, anders krijg je verschillende roosters.

Ongeschikt voor situaties waarin **responsiveness** erg belangrijk is, omdat SJN elke keer sorteert op tijdsbestek van een taak, kan het zijn dat lange taken erg lang moeten wachten voordat ze worden uitgevoerd. Dit is bijvoorbeeld in een processor van een microcontroller onwenselijk, omdat je wilt dat de taken die jij geeft snel worden afgehandeld door de processor.

4 BRONNEN

ece.ubc.ca. (2016). Opgehaald van web.archive.org: https://web.archive.org/web/20180205095009/http://www.ece.ubc.ca/~sasha/papers/eurosys16-final29.pdf

geeksforgeeks. (2022, Feb 08). *Program for Shortest Job First (or SJF) CPU Scheduling | Set 1 (Non- preemptive)*. Opgehaald van geeksforgeeks: https://www.geeksforgeeks.org/program-for-shortest-job-first-or-sjf-cpu-scheduling-set-1-non-preemptive/

javatpoint. (sd). *os-fcfs-scheduling* . Opgehaald van javatpoint.com: https://www.javatpoint.com/os-fcfs-scheduling

Molnár, I. (2021, November 17). *Completely Fair Scheduler*. Opgehaald van wikipedia: https://en.wikipedia.org/wiki/Completely_Fair_Scheduler

UNIVERSITY OF WISCONSIN–MADISON. (2008). Opgehaald van wisc.edu: https://pages.cs.wisc.edu/~remzi/OSTEP/cpu-sched.pdf

Wikipedia authors. (2021, April 22). *Shortest_job_next* . Opgehaald van wikipedia: https://en.wikipedia.org/wiki/Shortest_job_next

OPEN UP
NEW HAN_UNIVERSITY OF APPLIED SCIENCES
HORIZONS.