Ontwerp en evaluatie van een process scheduler
Besturingsystemen 2

(B-KUL-JPI0AZ)

Gilles Peeters en Henri Vandeputte

March 30, 2022

Class: 3ELICTcs

Instructor: Naessens Vincent

1 Modellering en structuur van het programma

1.1 Algemene functies

Het inlezen van de datasets moet maar één keer gebeuren, dit doen we in het begin van het programma. We hebben ook drie functies gemaakt om elke strategie te evalueren. Een functie die de globale parameters berekent, een functie die de dataset omzet naar percentielen en ook een functie die de gewenste grafieken gaat plotten.

1.2 First Come First Served

Bij deze code gaan we de dataset sorteren op arrivaltime, waarna we doorheen de dataset itereren en steeds de dataset aanvullen met de bekomen starttijd en wachttijd. Daarna bereken we ook nog van elk proces de TAT en de genormaliseerde TAT.

1.3 SJF, SRT en HRRN

Voor deze strategieën is steeds itereren over de hele dataset geen goeie oplossing. We maken in plaats daarvan gebruik van een queue. Deze queue van processen die reeds aangekomen zijn kan dan waar nodig gesorteerd worden op de relevante parameter.
Bij SJF wordt er gesorteerd op service time (oplopend). Eenmaal een proces gekozen is, zal het volledig uitgevoerd worden.
Bij SRT wordt er gesorteerd op de *remaining* service time (oplopend). Telkens er een nieuw proces aankomt zal er opnieuw gesorteerd worden.
Bij HRRN wordt er gesorteerd op "response ratio" (aflopend). Telkens wanneer een proces aankomt, zullen de response ratios van alle processen in de queue berekend worden. Na het berekenen van alle response ratios zal het proces met de hoogste response ratio uitgevoerd worden.

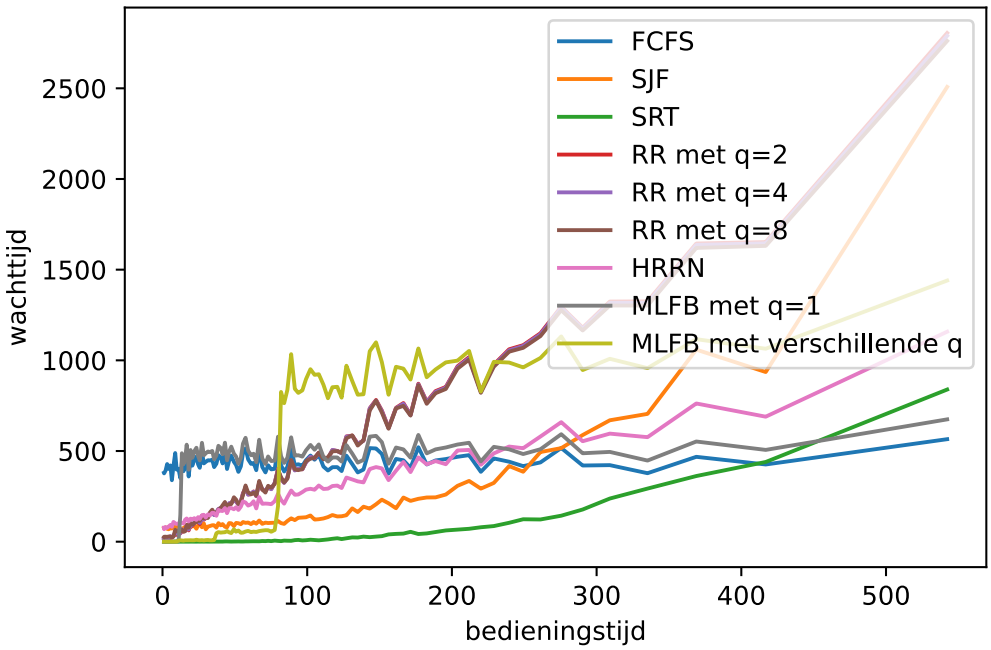
1.4 Round Robin

Ook hier wordt er steeds met een queue gewerkt. Om het volgende proces te kiezen word FCFS gebruikt, enkel wordt hier de processor tijd gelijk verdeeld over alle processen in de queue door te werken met time slices. In ons programma hebben we het algoritme getest voor de volgende time slices: $q = 2$, $q = 4$ en $q = 8$.

1.5 Multi level feedback

Dit scheduling algoritme kent veel variaties en is het meest complex van alle algoritmes die we gemodelleerd hebben. De implementatie kan terug gebracht worden tot een vijftal queues die volgens het RR principe worden afgewerkt. Een proces dat aankomt komt steeds eerst op de hoogste queue terecht. Deze queue zal afgewerkt worden volgens het FCFS principe telkens voor een gekozen tijd q. Eens deze tijd afgelopen is en het proces is nog niet afgerond dan zal dit proces naar een lagere queue geschoven worden. Dit gebeurt voor elke queue tot de laatste bereikt is, deze zal dan alle processen afwerken volgens het RR principe. Deze laatste queue kan wel enkel de processor gebruiken als alle andere queues leeg zijn. We maakten twee versies volgens dit principe. De eerste versie met voor elke queue dezelfde q, namelijk 1. De tweede versie met voor elke queue een andere q, waarbij we de waarden voor q bepaald hebben door 5 gemiddeldes te berekenen, en op basis hiervan 5 waarden te berekenen. We krijgen volgende waarden: 10,26,44,50,140.

2 Testresultaten



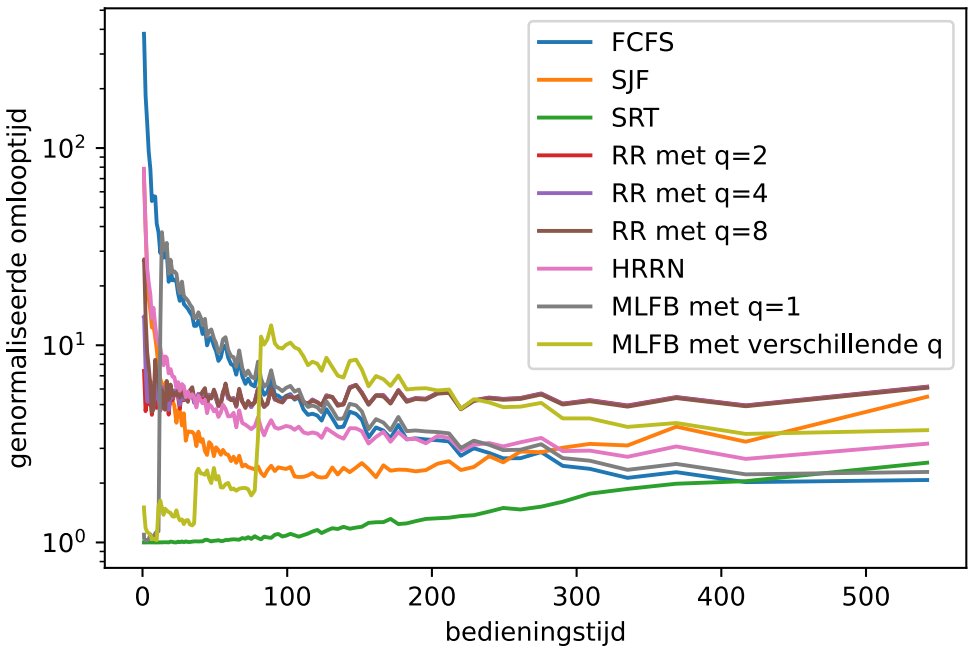


Figure 1: De gewenste grafieken bij 20 000 processen.

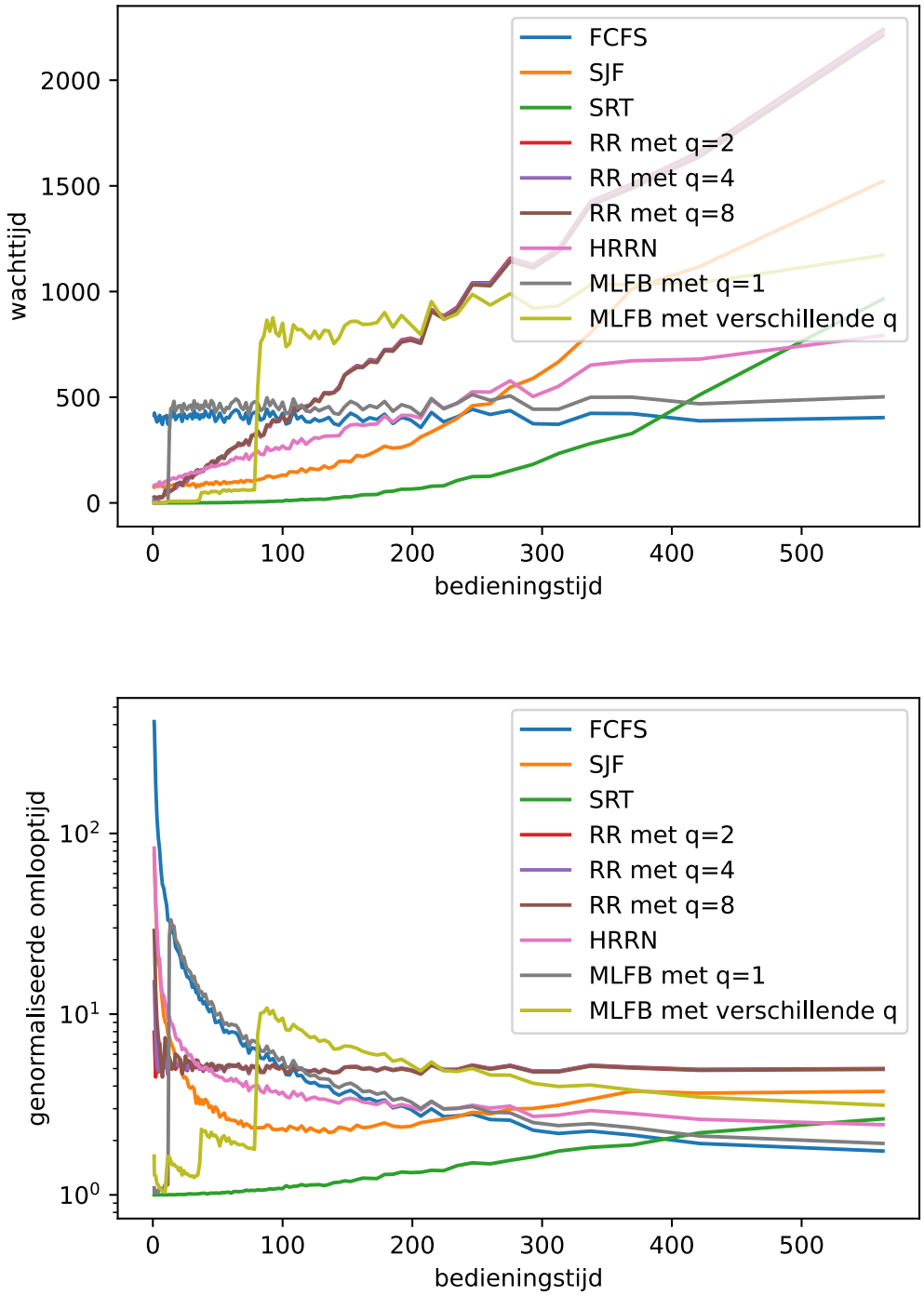


Figure 2: De gewenste grafieken bij 50 000 processen.

3 Evaluatie

Wat opvalt voor alle grafieken is dat er een zekere variatie zit op de gebruikte waarden die zorgt voor een zekere 'grilligheid' van de grafiek. Deze 'grilligheid' is kleiner wanneer een grotere dataset wordt gebruikt. Dit valt te verklaren door het feit dat wanneer een groter aantal waarden gebruikt is om elk percentiel te berekenen, dat dit de invloed van afwijkende waarden vermindert. Verder kunnen we hier stellen dat de grafieken voor beide datasets zeer gelijkaardig zijn.
We bespreken de grafieken nu voor elk onderdeel meer in detail.

3.1 Algemeen: genormaliseerde omlooptijd

Deze grafiek heeft voor veel strategieën in het begin een opvallend grote waarde. Dit valt te verklaren aan de hand van de gebruikte formule voor de genormaliseerde omlooptijd: NTAT = $\frac{s+w}{s}$. Hoe kleiner de service time, hoe groter de invloed van de wachttijd op de NTAT. Het is enkel bij die strategieën die de korte processen eerst gaan uitvoeren, dat dit niet het geval is. (vooral SRT en MLFB)

3.2 FCFS

Bij de FCFS strategie worden alle processen, van begin tot eind op een FIFO manier uitgevoerd. Theoretisch gezien zal de wachttijd ongeveer constant zijn. Als we kijken naar de grafiek voor de wachttijd zien we dat dit voor onze simulatie ook het geval is, met een gemiddelde van ongeveer 440 JIFFY's als wachttijd.

Ook de grafiek van de genormaliseerde TAT, voor FCFS, voldoet aan de verwachtingen. We zien een zeer hoge NTAT voor de laagste service times, met een sterke daling naargelang de service time groter wordt.

3.3 SJF

Voor SJF zien we een wachttijd die vrij linear toeneemt met de service time. We zien wel een iets lagere stijging in het begin, en een iets hogere stijging naar het einde toe. Voor een strategie die de kortste processen eerst zal uitvoeren is dit een logisch resultaat.
Wanneer we naar het verloop van de NTAT kijken voor deze strategie zien we wel dat voor de processen met een service time van minder dan 10 JIFFY's hier nog zeer hoge waarden bereikt worden. Voor deze processen presteert dit algoritme dus ondermaats.

3.4 SRT

Voor SRT zien we een wachttijd die eerst vrij constant lijkt. Eens de service time boven de 100 JIFFY's komt zien we dat de wachttijd meer en meer begint toe te nemen. Vanaf 300-400 JIFFY's krijgen we zelfs een vrij lineair 1 op 1 verband. Wanneer we dit verloop vergelijken met dat van SJF zien we een vrij gelijkaardig verloop, maar SRT presteert over de hele lijn duidelijk beter. Er zijn geen uitschieters voor zeer kleine service times, integendeel, voor de laagste service time is de wachttijd nagenoeg nul.
Zeker op de grafiek voor NTAT zien we dit bevestigd. Geen uitschieters, vrij constante stijging (op logaritmische schaal weliswaar) en een NTAT van ongeveer 1 voor de laagste service times. Voor beide parameters is SRT duidelijk de beste optie.

3.5 RR

Voor RR zouden er telkens 3 curves te zien moeten zijn voor de 3 verschillende waarden voor q. Bekijken we deze drie, dan zien we direct dat deze zeer dicht op elkaar liggen, en op de meeste plaatsen zelfs moeilijk te onderscheiden zijn.

Als eerste merken we op dat er een sterk lineair verband is tussen de service time en de wachttijd. We zien dat de grafieken hier vrijwel exact hetzelfde verloop hebben, met slechts voor de hoogste service times een klein verschil in de wachttijd voor de verschillende time slices. Aangezien de time slices allemaal vrij klein zijn ten opzichte van de service times is dit ook logisch aangezien we de kleine verschillen die zich zouden voordoen wegfilteren door alles nog eens om te zetten naar percentielen.
Voor de NTAT zien we ook dat alle curves helemaal lijken samen te vallen, behalve voor de kleinste service times. Dit valt te verklaren door het feit dat de NTAT groter zal zijn voor de gevallen waar service time kleiner is dan de gekozen q. Los hiervan zien we wel dat de NTAT voor elke q vrij constant is, met een gemiddelde waarde van 5.4 - 6.
We zien in dat RR dus een vrij constante NTAT geeft zonder al te grote pieken in het begin, maar over het algemeen zal RR niet goed presteren ten opzichte van de andere algoritmes, zeker niet voor de grotere service times.

3.6 HRRN

Voor HRRN zien we opnieuw een vrij lineair verband tussen de wachttijd en de service time. De wachttijd start voor de laagste service times op een waarde gelijkaardig aan die van SJF, waarna deze traag stijgt om uiteindelijk voor service times groter dan 500 JIFFY's zelfs een lagere wachttijd te bekomen dan SRT.
Voor de NTAT zien we dan ook een piek in het begin die snel daalt om dan schijnbaar uit te vlakken naargelang de service time groter wordt.
Over het algemeen lijkt HRRN dus geen slechte optie, zeker voor grotere service times kan HRRN zelfs beter presteren dan SRT. SRT blijft wel de betere optie over het algemeen.
Bij deze strategie zullen de lange processen op een bepaalt moment prioritair worden, tegenover de binnenkomende korte processen.

3.7 MLFB

Voor de MLFB versie waar elke q=1 kunnen we heel kort zijn: het gedrag is, logisch, gelijk aan RR met als time slice 1, eens alle processen in de laagste queue terecht zijn gekomen. De prestaties lijken iets beter voor service times tussen 50 en 200 JIFFY's, daarbuiten zijn zowel de NTAT als de wachttijd nog slechter dan voor RR met iets hogere time slices.
Voor de MLFB versie met verschillende time slices (10,26,44,50,140) zien we een uniek resultaat. De grafiek ziet er uit als een 'trap' met sprongen op 10, 36 en 70. Dit komt overeen met q0, q0+q1 en q0+q1+q2. We zien dat de grafiek na de laatste trap overeen komt met die van RR met een hoge q, zoals verwacht. Voor de laatste trap zien we een verloop dat in de buurt komt van dat van SRT. Vermoedelijk komt voor deze waarden voor q queue 3 niet vaak aan de beurt, alsook queue 4. We kunnen afleiden uit dit resultaat dat met ideaal gekozen waarden voor q een zeer goede werking bekomen kan worden, die zeker voor lange service times zelfs SRT zou kunnen verslaan. Het dynamisch bepalen van time slices is jammer genoeg iets te complex voor ons om te programmeren.

3.8 Algemeen: globale parameters

In onze evaluatie functie hebben we steeds een aantal globale parameters berekend. Om de algoritmes met elkaar te kunnen vergelijken hebben we deze op basis van de bekomen waarden gerangschikt. Deze waarden komen allemaal uit de dataset van 20 000 processen, aangezien er geen significante verschillen waren met deze bekomen voor de dataset met 10 000 of 50 000 processen. We laten ook even MLFB buiten beschouwing aangezien we dit algoritme onvoldoende hebben kunnen optimaliseren om gegronde conclusies te kunnen nemen.

Criterium \ Algoritme	RR2	RR4	RR8	FCFS	HRRN	SJF	SRT
Gemiddelde wachtijd in ms	4432	4431	4429	4356	2723	1965	391

Table 1: De verschillende processing algoritmes gerangschikt op de gemiddelde wachtijd.

Criterium \ Algoritme	FCFS	HRRN	RR8	SJF	RR4	RR2	SRT
Gemiddelde genormaliseerde omlooptijd	22.97	7.12	6.06	5.71	5.6	5.39	1.15

Table 2: De verschillende processing algoritmes gerangschikt op de gemiddelde genormaliseerde omlooptijd.

Criterium \ Algoritme	RR2	RR4	RR8	FCFS	HRRN	SJF	SRT
Gemiddelde omlooptijd in ms	5440	5439	5437	5364	3730	2973	1399

Table 3: De verschillende processing algoritmes gerangschikt op de gemiddelde omlooptijd.

$\Longrightarrow$ Op basis van deze globale parameters kunnen we opnieuw concluderen dat SRT, globaal gezien, het beste presteert.

4 Besluit

Bij FCFS is er geen enkele vorm van selectie, dus prioritaire processen kunnen langer wachten dan niet prioritaire. Dit vormt een nadeel. Afhankelijk van wat we het belangrijkste criterium vinden, kunnen we de verschillende strategieën gebruiken. Voor eerlijkheid FCFS, voor de kortste remaining time SRT, voor priorisatie naar de kortste processen SJF,...

SRT geeft de laagste wachttijden, en de laagste NTAT en klinkt dus ideaal. Lange processen zullen hier echter ook zeer lang moeten wachten. Dit is niet ideaal om in de praktijk toe te passen, waar er zeer lange en zeer korte processen kunnen voorkomen die van elkaar afhangen. Schedulers die steunen op verschillende niveau's van prioritaire processen, zullen het beste zijn voor de user experience van een computer en zullen dus in de praktijk gebruikt worden.

We merken dat niet elk algoritme voor elke situatie ideaal is. MLFB lijkt een aantal algoritmes te combineren maar dit zorgt wel voor een grotere complexiteit en bijgevolg ook overhead. Een

heilige graal bestaat niet maar wanneer bijvoorbeeld MLFB dynamisch time slices zou kunnen bepalen dan zou dit in theorie voor elke situatie een degelijke oplossing moeten vormen.

5 Reflectie

Dit labo gaf ons inzicht in de implementatie van de verschillende strategieën en hoe het mogelijk is om via de aangeleerde programmeer talen situaties te simuleren en te evalueren.

Er is ook veel kennis verworven over de soorten scheduling algoritmes die we gezien hebben in de hoorcolleges, en hoe we ze kunnen vergelijken met elkaar. MLFB blijft een zeer complex algoritme waar we nog niet alles van begrijpen, maar ook hier hebben we veel over bijgeleerd.

6 Tijdsbesteding

Gilles heeft zich hoofdzakelijk bezig gehouden met de code te schrijven en te perfectioneren. Henri heeft zich hoofdzakelijk bezig gehouden met het verslag en het kritisch observeren van de code en de resultaten. Doorheen heel het labo was er steeds goeie communicatie, we hebben beide aan alles bijgedragen maar we hebben uiteindelijk de code geschreven door Gilles gebruikt.