Wouter Vande Velde

KU Leuven - Besturingssystemen 2

Process scheduler

Practicum 1

# Modellering

Het probleem bestaat uit 3 grote delen, namelijk als eerste het uitlezen, als tweede het verwerken van de data en als derde het weergeven van de data. Hierbij is het logisch om deze 3 stappen ook op te splitsen binnen de structuur van het programma.

Voor de eerste stap zal er gebruik gemaakt worden van een Node-lijst die het gehele Xml-bestand bevat en vervolgens wordt elke node uitgelezen naar een lijst in het programma.

De tweede stap wordt het verwerken van deze data, elke techniek wordt 1 voor 1 uitgewerkt in volgorde van makkelijk naar moeilijk waarbij (vermoedelijk) first come first serve het makkelijkste is en multilevel feedback het moeilijkste.

Eens deze strategieën zijn geïmplementeerd, komt stap 3 aan de beurt. Deze stap houdt in dat de data moet worden weergegeven op de grafiek. De resultaten worden omgevormd naar percentielen en vervolgens weergegeven op de grafiek aan de hand van de library genaamd JFreeChart

# Structuur

Het project is opgebouwd uit 4 verschillende packages, deze zijn de default package, een package voor de strategieën, een package met 2 Process objecten en een package met 2 klassen voor de grafieken op weer te geven. De reden waarom er 3 Process objecten zijn is omdat er 3 verschillende Comparables gebruikt worden die nodig zijn voor een Priorityqueue te kunnen gebruiken, onder andere voor highest response ratio next waarbij er op prioriteit gesorteerd wordt of shortest remaining time waarbij er op de remaining time wordt gesorteerd.

Om het programma te runnen en hiermee alle technieken te testen, moet enkel de main-klasse worden aangeroepen. In deze klasse worden eerst het Xml-bestand gelezen en in objecten gestoken, die op hun beurt dan weer worden doorgegeven naar elke techniek. Elke techniek zit omvat in zijn eigen klasse, die zich allemaal bevinden in de package Strategies

Eens de main gestart is, komt er een file Explorer waarbij het juiste bestand door de gebruiker wordt gekozen, vervolgens worden zoals eerder vermeld alle strategieën uitgevoerd op de data uit het gekozen Xml-bestand. Nadat deze zijn uitgevoerd, worden verschillende waarden afgeprint, deze waarden zijn de gemiddelde van turnaround time, waiting time en normalized turnaround time. De rest wordt tevens weggeschreven naar txt bestanden die te vinden zijn in de *src* folder.

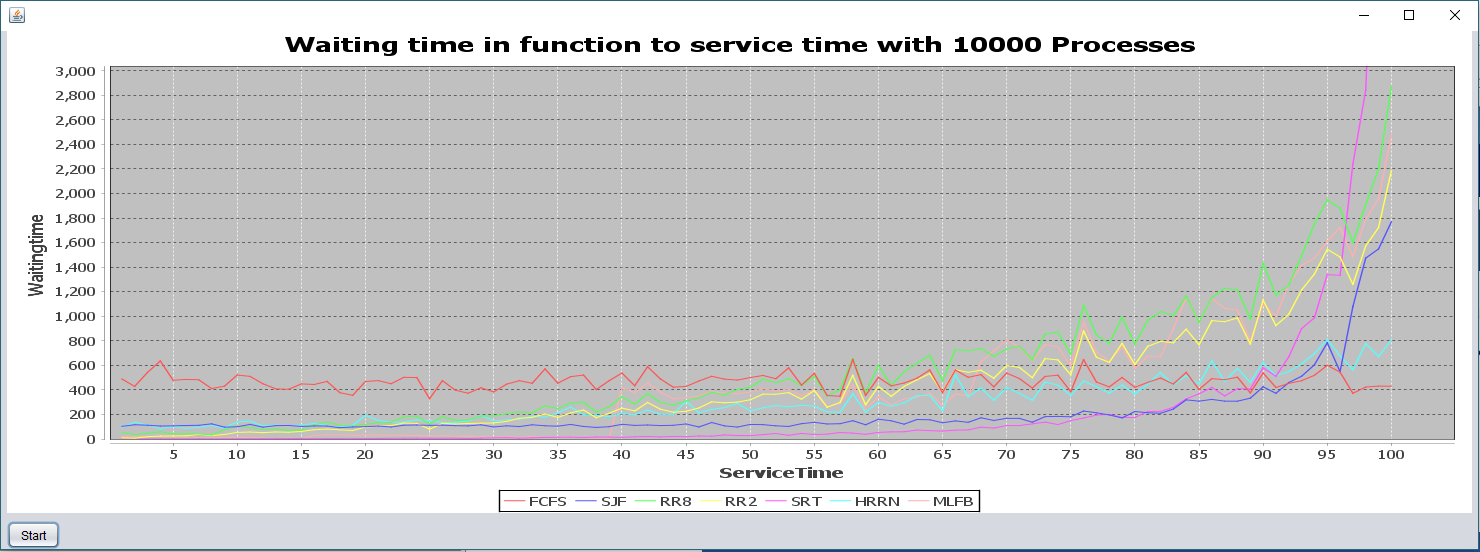
In de opdracht werd aangeraden om gebruik te maken van 10 milliseconden of ook wel de tijd tussen 2 system interrupts om 1 JIFFY voor te stellen. Wij hebben dit een beetje anders aangepakt door gebruik te maken van een simpele for-loop. In elke strategie maken we gebruik van een oneindige for-loop waarbij elke cyclus gelijk staat aan 1 Jiffy, er wordt uiteraard gezorgd dat er maar 1 process wordt uitgevoerd tijdens elke cyclus net zoals dit het geval is bij een processor.

In de package GUI bevinden zich 2 aparte GUI’s, de eerste GUI toont de genormaliseerde turnaround time ten opzichte van de service time. De tweede grafische interface toont de waiting time in functie van de service time. Op volgende pagina zijn deze grafieken weergegeven.

# 10000 processen

(De grafieken zijn mogelijk moeilijk te lezen, ze kunnen ook uitgevoerd worden in het bijgeleverde programma)

## Waiting time in functie van de service time



De waiting time in functie van de service tijden

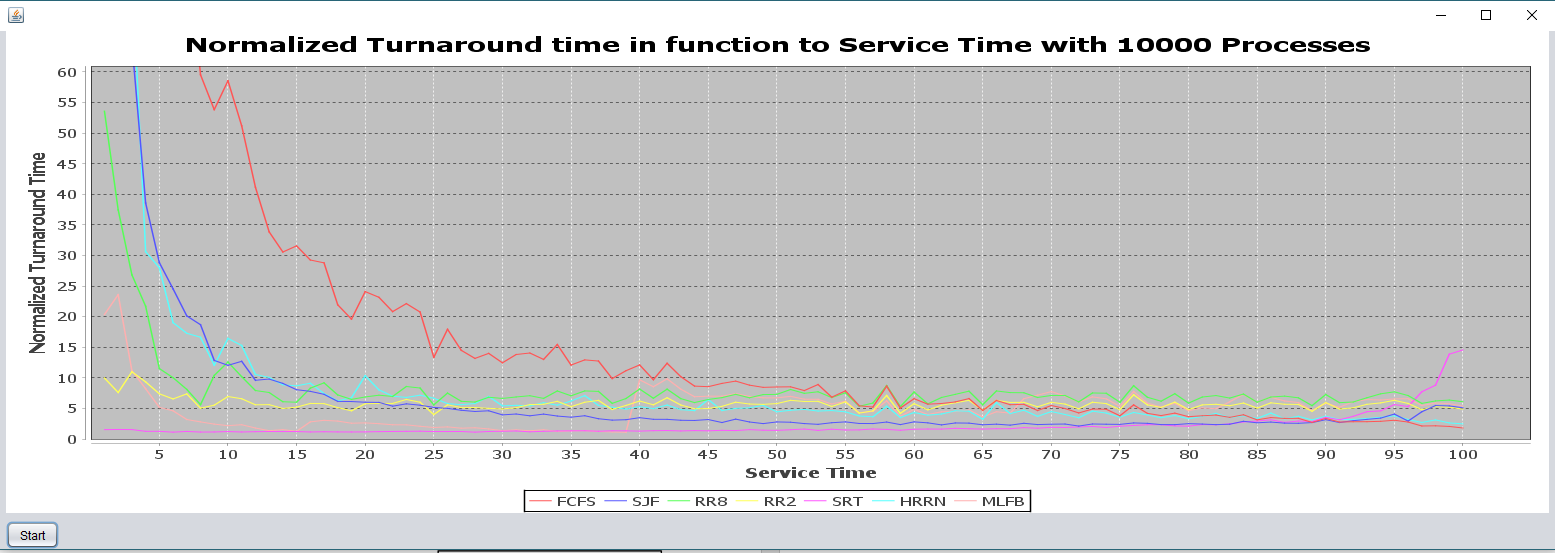
Op de grafiek is duidelijk te zien hoe bij Shortest Remaining Time (SRT) de wachttijden veel groter worden naarmate de processen groter worden. Dit komt omdat de kleinere processen voorrang zullen krijgen op de grotere en de grotere steeds langer en langer moeten wachten om aan de beurt te komen, er ontstaat starvation. Op deze afbeelding is ook te zien dat Shortest Remaining Time een wachttijd van 0 heeft bij kleine processen.

De rode lijn, die FCFS voorstelt, blijft over de hele lijn constant. Dit komt omdat alle processen even lang zullen moeten wachten op hun beurt en er geen enkel proces voorrang krijgt. Deze techniek is vooral slecht voor kleine processen aangezien de genormaliseerde turnaroundtime veel hoger zal zijn, wat zichtbaar is op de grafiek op de volgende pagina.

Voor de grote processen scoren bijna alle technieken slecht, de slechtste is duidelijk Shortest remaining time, maar ook round robin q8 en q2 en multilevel feedback niet goed scoren. De Beste techniek blijkt uiteindelijk FCFS voor de grootste processen. Highest response ratio next is nipt tweede.

Wat ook opvalt is dat bij Multi Level Feedback de wachttijden 0 zijn tot percentiel 40. Dit komt omdat alle processen die voor 40% zitten, afgehandeld konden worden in de eerste queue en de service times kleiner zijn dan de quantum van round robin van de eerste queue. Alle processen die boven 40 zitten zijn niet afgehandeld kunnen worden in hun eerste beurt en werden doorverwezen naar de 2de queue met een lagere prioriteit. Rond 65 is er een tweede kleine sprong omhoog

## Normalized turnaround time in functie van de service time



Genormaliseerde turnaround time bij 10000 processen

Het valt ook op dat de genormaliseerde turnaround time bij kleine processen bij shortest remaining time nul is, omdat elke process zal onderbroken worden zodra dit kleine process aankomt. Bij Shortest job first wordt het niet onderbroken en zullen de genormaliseerde turnaround times wel kunnen oplopen. Bij FCFS is deze tijd het hoogste, omdat hier geen rekening wordt gehouden met hoe groot het proces is, grote processen hebben dezelfde prioriteit als kleine.

Bij Round Robin valt er een groot verschil te constateren tussen quantum 2 en 8 in het begin. Dit komt omdat, indien er processen voor een heel klein proces komen, het kleine proces zal moeten wachten op zijn beurt en dus telkens 8 jiffy’s extra moet wachten van de processen er vlak voor. Bij kleine processen heeft dit heel snel een invloed op de genormaliseerde turnaround time.

Bij shortest remaining time valt op het einde een kleine stijging op te merken. Bij grote processen moeten ze al heel lang wachten voordat er een opvallende stijging zou zijn in de grafiek, maar bij SRT ontstaat er starvation van de grote processen en zullen deze dus soms heel lang moeten wachten voordat ze volledig afgewerkt zijn. Ook bij shortest job first valt een heel kleine stijging op, maar een pak minder opvallend.

Het valt hier dus duidelijk op dat First come First serve het slechtste is voor kleine processen en Shortest Remaining time het beste is voor kleine processen. Maar dit geld dan ook omgekeerd, First come First serve is volgens deze grafiek het beste voor grote processen en Shortest Remaining Time het slechtste

# 20000 processen

(De grafieken zijn moeilijk te lezen, ze kunnen ook uitgevoerd worden in het bijgeleverde programma)

## Waiting time in functie van de service time

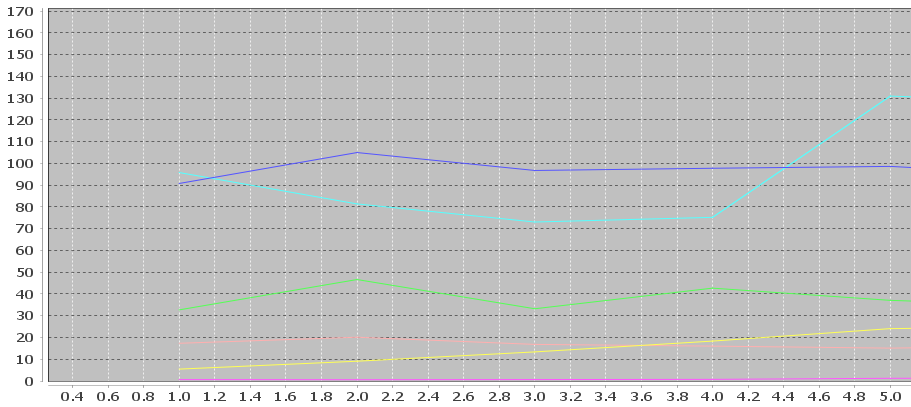
Waiting time in functie van de service time bij 20000 processen

Bij 20000 processen zijn de grafieken gelijkaardig aan de grafieken van 10000 processen.

Ook bij deze grafiek valt op dat FCFS constant blijft ongeacht de grootte van het proces, dit betekend dat het voor de grootste processen het meest voordelige en het minst voor de kleinere.

Verder valt er op dat bij Shortest remaining time de waiting time gelijk is aan 0 aangezien de kleinste processen hierbij altijd voorrang krijgen.

Hetgeen we ook zagen bij 10 000 is de sprong bij multilevel feedback rond 40%. Deze keer is het op procent 39, de overgang waarbij de processen te lang zijn voor de eerste queue.

Hiernaast is op de afbeelding een uitvergroting te zien van de eerste 5%. Hierop valt op dat round robin met quantum 2 beter scoort dan quantum 8. We kunnen dus besluiten dat zowel multilevel feedback, round robin (met een kleine quantum) en shortest remaining time de beste technieken zijn voor de kleine processen.

## Normalized turnaround time in functie van de service time

Genormaliseerde turnaround time bij 10000 processen

Ook hier zijn de besluiten ongeveer gelijk als die bij 10 000 processen. We kunnen zeggen dat FCFS het slechtste is voor kleine processen en het beste voor grote processen. De tweede en derde slechtste strategieën voor kleine processen zijn Shortest Job First en Highest Response Ratio Next voornamelijk omdat deze allebei niet preemptive zijn.

Shortest remaining time scoort dan weer het beste voor kleine processen met een heel lage genormaliseerde turnaround time, gevolgd door Round Robin met Quantum 2.

Op het einde van de grafiek valt weer op dat er starvation is voorgekomen bij Shortest Remaining time aangezien daar een kleine stijging op te merken is.

# Besluit

We konden zeker besluiten dat we veel hebben bijgeleerd gedurende dit project. Alhoewel we de meeste technieken al kenden van in de les was het wel nuttig om deze technieken om te zetten in de praktijk.

Een tweede pluspunt dat zeker verbonden is aan het project is dat we onze kennis van Java hiermee nog eens hebben kunnen opfrissen, dit was voor ons al geleden van onze 2de jaar in de bachelor.

Door het opnieuw te maken kan ik besluiten dat het uiteindelijk makkelijker was dan de eerste keer, het programmeren ging vlotter en op een paar kleine details na is alles een pak vlotter verlopen dan in april. De strategieën die wij vorige keer niet hadden gemaakt waren deze keer ook vlot geïmplementeerd.

Een werkpunt is zeker nog clean code, ik zou de code meer mogen onder verdelen en meer code hergebruiken in plaats van deze code te kopiëren.

Uiteindelijk kan ik stellen dat ik zeer tevreden ben met het resultaat aangezien alle technieken succesvol zijn geïmplementeerd.