Onderzoeksverslag

– Process scheduling –

Student:	A.W. Janisse
Studentnummer:	2213829
Instelling:	Fontys hogescholen te Eindhoven
Opleiding:	ICT & Technology
Docent:	Dhr. Erik Dortmans

Opdracht:	6, Process scheduling
Datum:	9 april 2015

Inleiding

In deze opdracht wordt het begrip process scheduling bestudeerd. Process scheduling houdt in hoe het operating systeem de processortijd verdeeld tussen processen.

Taken

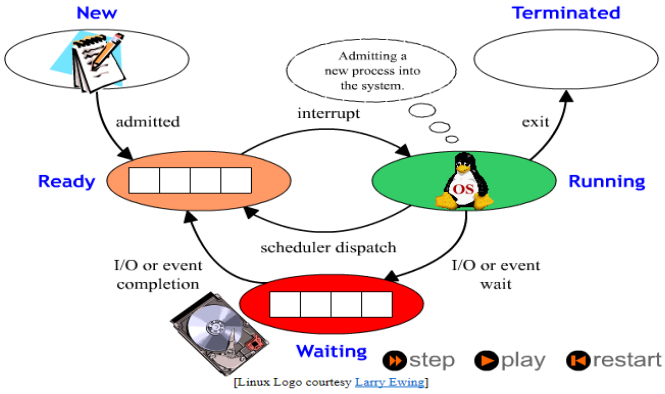
In deze opdracht wordt de uitwerking van de volgende taken beschreven:

1. Process schduling. Aan de hand van een animatie worden verschillende vragen beantwoord.
2. Process Scheduling Algorithms. Aan de hand hand van een aangeleverde applet worden een aantal algoritmes gemaakt en vragen beantwoord.
3. Timeslicing. Aan de hand van een programma wordt timeslicing onderzocht.

Uitwerking taak 1

Voor de uitwerking van deze taak is een applet zoals die in afbeelding 1 getoond is gestart. De applet is te vinden op de volgende link:

http://courses.cs.vt.edu/csonline/OS/Lessons/Processes



Afbeelding 1

Aan de hand van de applet moeten de volgende vragen beantwoord worden:

A: welk process schedulings algoritme wordt toegepast?

B: voor welke activiteiten komt het OS (de pinguïn "Tux") in de Running state?

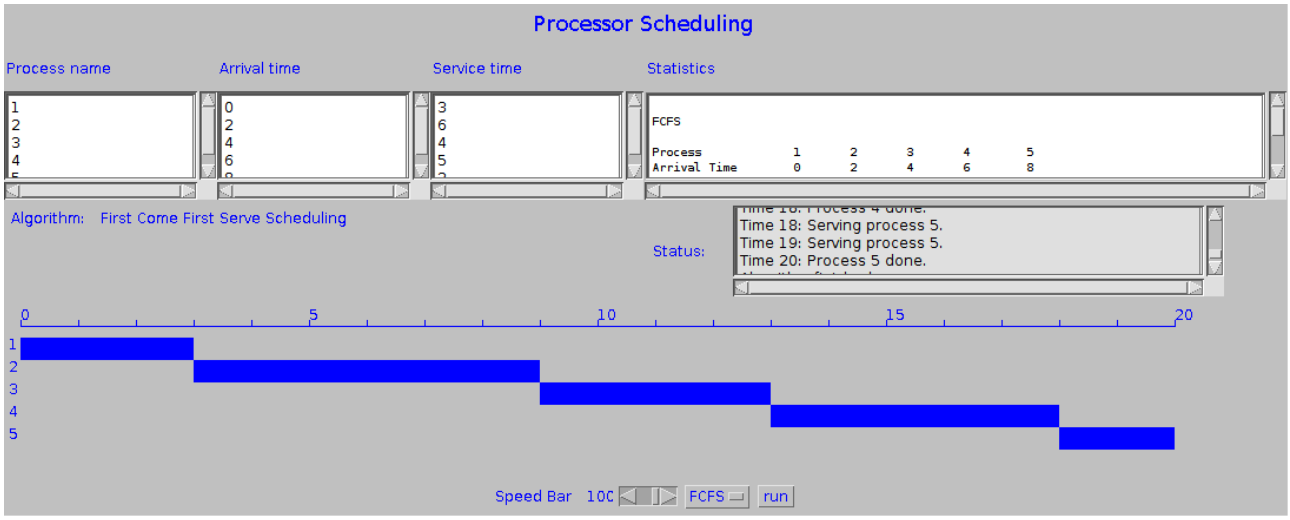
Antwoorden:

A: Het scheduling principe wat in de animatie wordt toegepast is *Round Robin*. Kenmerkend voor deze scheduling methode is dat op periodieke basis een *clock interrupt* gegenereerd wordt. Wanneer de *interrupt* plaats vindt zal het huidige proces in de *ready queue* worden geplaatst. Wanneer een proces geblokkeerd raakt zal het door het OS in de *waiting queue* worden geplaatst. De *ready queue* wordt afgehandeld op basis van *First Comes First Serves* (FCFS). De processen in de *waiting queue* komen weer achteraan in de *ready queue* als ze weer beschikbaar komen voor uitvoering. Deze methode is *preemptive*.

B: Het OS komt in de running state om een context switch uit te voeren. Een context switch vindt plaats op basis van de gegenereerde interrupt of op het moment dat een proces in een *blocked* toestand komt. Deze methode wordt ook wel *time slicing* genoemd.

Uitwerking taak 2

Voor de uitwerking van taak 2 is er een Java applet aangeleverd. Met deze applet is het mogelijk om scheduling simulaties uit te voeren. Afbeelding 2 laat deze applet zien.



Afbeelding 2

Opdrachten binnen deze taak:

A: Maak voor elk algoritme een scenario waarmee dit algoritme wint van de anderen. En maak *voor elk algoritme een scenario waarmee dit algoritme verliest van de anderen.*

B: Kies zelf een norm voor "verliezen" en "winnen": gaat het om de gemiddelde Tq/Ts, of om een *maximale Tq/Ts, of ...?*

C: *De Java applet is niet helemaal natuurgetrouw. Welk belangrijk aspect ontbreekt in de simulatie? Wat heeft dat voor gevolgen?*

Algemeen

Binnen de Applet kunnen de volgende scheduling simulaties worden uitgevoerd. Tabel 1 geeft deze weer:

Afkorting	Scheduling methode
FCFS	First Come First Serve
RR1	Round Robbin (q=1)
RR4	Round Robbin (q=4)
SPN	Shortest Process Next
SRT	Shortest Remaining Time

Tabel 1

De belangrijkste *scheduling* kenmerken worden in tabel 2 weergegeven:

	FCFS	Round Robin	SPN	SRT	HRRN	Feedback
Selection Function	max[w]	constant	min[s]	min[$s - e$]	$\max\left(\frac{w + s}{s}\right)$	(see text)
Decision Mode	Non-preemptive	Preemptive (at time quantum)	Non-preemptive	Preemptive (at arrival)	Non-preemptive	Preemptive (at time quantum)
Throughput	Not emphasized	May be low if quantum is too small	High	High	High	Not emphasized
Response Time	May be high, especially if there is a large variance in process execution times	Provides good response time for short processes	Provides good response time for short processes	Provides good response time	Provides good response time	Not emphasized
Overhead	Minimum	Minimum	Can be high	Can be high	Can be high	Can be high
Effect on Processes	Penalizes short processes; penalizes I/O bound processes	Fair treatment	Penalizes long processes	Penalizes long processes	Good balance	May favor I/O bound processes
Starvation	No	No	Possible	Possible	No	Possible

Tabel 2

Aanvulling op de *scheduling* kenmerken:

FCFS	Presteert beter voor processen die langer duren dan voor processen die korter duren. Korte processen die lange processen opvolgen zullen nadeel ondervinden vanwege het feit dat ze een hoge *Tr* tijd hebben in relatie met een korte *Ts* tijd.
RR	Scheduling op basis van een *kwantumtijd*. Een zeer korte *kwantumtijd* zal voor veel *proces switches* zorgen waarbij de hierbij horende overhead zoals de interrupt tijd en de afhandeling van de scheduling en dispatching een nadelig effect. Hele kort *kwantumtijden* moeten dus worden gemeden. Deze scheduling presteert het best als er een *kwantumtijd* wordt gekozen die een fractie hoger is dan de procestijden waardoor processen hun taak kunnen afronden en niet opnieuw in de *queue* geplaatst hoeven te worden. In het bijzondere geval dat de *kwantumtijd* langer is dan de tijd van het langst durend proces, dan zal RR minder presteren dan FCFS.
SPN	Deze scheduling kiest het proces waarvan de verwachte procestijd het kortste is als het volgende proces. Deze keuze wordt gemaakt op het moment dat er een context wissel plaats moet vinden. Korte processen lopen sneller door de *queue* dan lange processen. Er is een noodzaak om voor ieder proces de procestijd te weten of te kunnen voorspellen. Risico van *starvation* is aanwezig.
SRT	Deze scheduling kiest het volgende proces op basis van de kortst resterende procestijd. Deze afweging wordt gemaakt op het moment dat een proces aan de *queue* wordt toegevoegd waarbij het mogelijk is dat een huidig proces wordt onderbroken en het nieuwe proces gestart wordt. Er is een noodzaak om van ieder proces zijn verstreken procestijd te weten. Dit zorgt voor meer overhead. Risico van *starvation* is aanwezig.

Scenario's:

Als eerste moet er worden bepaald wat winnen en wat verliezen is. Ik ga er vanuit dat dat het scenario welke de laagste T*r/Ts* oplevert, de winnaar is. De *Tr/*Ts geeft de verhouding weer tussen de *turn around* time en de *service time* en zal als laagste waarde 1.0 zijn. Deze waarde geeft daarmee feitelijk een relatieve waarde voor de delay van een proces weer. Dit zijn dan ook de tijden die in onderstaande tabellen genoteerd zijn. Om de vergelijking te maken is een gemiddelde waarde (mean) uitgerekend.

Uitwerking opdracht A

Winnend scenario voor FCFS:

FCFS scheduling komt het best tot zijn recht als er processen draaien die langer duren. Dit is een eigenschap die niet nadelig is voor SPN en SRT. Vanuit deze optiek zou juist de overhead er dus voor moeten zorgen dat FCFS een licht voordeel heeft t.o.v. deze andere twee. De simulatie omgeving laat als uitkomst gelijke waarden zien. Dit betekend dus dat er niet met overhead wordt gerekend. Onderstaande tabel laat wel zien dat deze keuze nadelig is voor RR scheduling.

Paramaters	**Process 1**	**Process 2**	**Process 3**	**Process 4**	**Process 5**	
Arrival time	0	1	2	3	4	
Service time	20	20	20	20	20	
FCFS	**Process 1**	**Process 2**	**Process 3**	**Process 4**	**Process 5**	**Mean**
Tr/Ts	1	1	2	3	4	**2,2**
RR, Q=1	**Process 1**	**Process 2**	**Process 3**	**Process 4**	**Process 5**	**Mean**
Tr/Ts	4	4	4	4	4	**4**
RR, Q=4	**Process 1**	**Process 2**	**Process 3**	**Process 4**	**Process 5**	**Mean**
Tr/Ts	4	4	4	4	4	**4**
SPN	**Process 1**	**Process 2**	**Process 3**	**Process 4**	**Process 5**	**Mean**
Tr/Ts	1	1	2	3	4	**2,2**
SRT	**Process 1**	**Process 2**	**Process 3**	**Process 4**	**Process 5**	**Mean**
Tr/Ts	1	1	2	3	4	**2,2**

Verliezend scenario FCFS

FCFC is een nadelig scheduling algoritme als korte processen lange processen opvolgen. De korte processen hebben daardoor een langere wachttijd en derhalve dus ook een hoge Tr/Ts. Onderstaand scenario geeft hiervan de resultaten. De uitkomsten laten mooi zien dat FCFS in het voordeel is van de langere processen en in het nadeel van de korte. Dit voordeel wordt gecompenseerd door SPN en SRT waarvoor deze algoritmes dan ook bedoeld zijn.

Paramaters	**Process 1**	**Process 2**	**Process 3**	**Process 4**	**Process 5**	
Arrival time	0	1	2	3	4	
Service time	50	2	50	2	50	
FCFS	**Process 1**	**Process 2**	**Process 3**	**Process 4**	**Process 5**	**Mean**
Tr/Ts	1	25	2	50	3	**16,2**
RR, Q=1	**Process 1**	**Process 2**	**Process 3**	**Process 4**	**Process 5**	**Mean**
Tr/Ts	3	3	3	3	3	**3**
RR, Q=4	**Process 1**	**Process 2**	**Process 3**	**Process 4**	**Process 5**	**Mean**
Tr/Ts	2	4	3	4	4	**4**
SPN	**Process 1**	**Process 2**	**Process 3**	**Process 4**	**Process 5**	**Mean**
Tr/Ts	1	25	25	3	3	**11,4**
SRT	**Process 1**	**Process 2**	**Process 3**	**Process 4**	**Process 5**	**Mean**
Tr/Ts	1	1	1	2	3	**1,6**

Winnend scenario RR, Q=1 en RR, Q=4:

Een winnend scenario zou moeten zijn dat de service tijd een fractie korter is dan de *kwantumtijd*. Omdat er slechts integere waarden ingevuld kunnen worden, maar ook weer vanwege het feit dat er niet met overhead wordt gerekend, is het niet mogelijk om dit tot uiting te laten komen.

Verliezend scenario RR, Q=1 en RR, Q=4

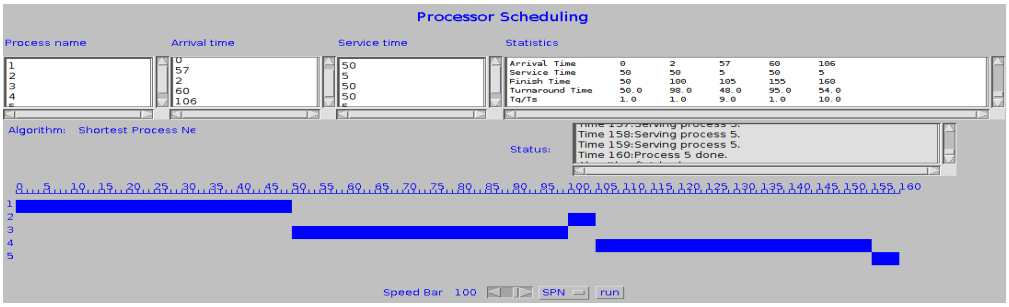
Een verliezend scenario zou tot uiting moeten komen als de service tijd een fractie langer wordt gekozen dan de *kwantumtijd*. In dat geval zal een proces meerdere malen actief moeten worden om ready te komen. In dit geval zou overhead een nadelig effect hebben op het RR scheduling algoritme.

Winnend scenario SPN

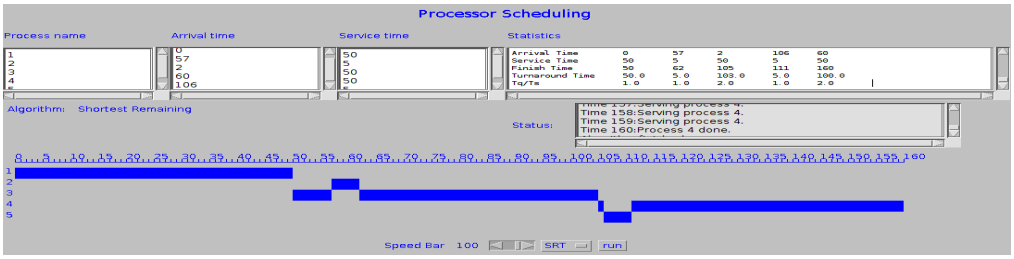
SPN zou een voordeel moeten hebben t.o.v. FCFS op het moment dat korte processen lange processen opvolgen. Zoals we gezien hebben bij FCFS heeft daar een fors nadeel bij. Omdat SPN zelf het volgende proces kiest zal dit nadeel dan worden opgeheven. Korte processen krijgen hierbij immers voorrang op lange processen. Voor zowel korte als lange processen heeft dit een gunstige invloed op de ratio Tr/Ts. SRT zou t.o.v. SPN een nadeel kunnen ondervinden als er korte processen in de *queue* komen op het moment dat een wat langer processen actief is. Hierbij zal het namelijk zo zijn dat het actieve proces wordt onderbroken en het binnenkomende korte proces actief gemaakt wordt.

Verliezend scenario SPN

Zoals inmiddels duidelijk is geworden is het voor korte processen nadelig om lang te moeten wachten. Als dit het geval is dan zal dus de ratio Tr/Ts behoorlijk toenemen. Om dus een verliezend scenario te hebben voor SPN zou het dus zo moeten zijn dat er kortdurende processen aan de *queue* worden toegevoegd als een wat langer durend proces actief is. Afbeelding 3 laat geeft dit scenario weer voor SPN en afbeelding 4 voor SRT. Duidelijk is te zien dat SPN hier de verliezer is.



Afbeelding 3



Afbeelding 4

Winnend scenario SRT

Afbeelding 4 laat als een winnend scenario zien. Bij SPN is het immers zo dat kortdurende processen direct aan de beurt kunnen komen en dat zelfs een langer durend proces hiervoor onderbroken wordt.

Verliezend scenario SRT

Zoals al eerder is benoemd moet voor een verliezend scenario er worden gezocht naar korte proces die relatief lang moet wachten om zodoende een hoge Tr/Ts ratio te krijgen. Nu is het zo dat juist SRT dit scenario tegen gaat. Hierdoor is het naar mijn mening niet haalbaar om een duidelijk verliezend scenario te bedenken.

Uitwerking opdracht B

De vraag is om zelf een norm voor "verliezen" en "winnen" bepalen en of het dan gaat het om de gemiddelde Tq/Ts, of om een *maximale Tq/Ts, of ...?*

Op deze vraag is niet een eenduidig antwoord te geven. Het zijn naar mijn mening juist de gebruiksomstandigheden die dit bepalen. Hierbij valt te denken aan een algeem systeem waar een gebruiker bijvoorbeeld video wil kijken of games wil spelen. Juist in deze situatie is het onwenselijk om de video of de game steeds weer kort te moeten pauzeren ten gunste van een ander proces.

Aan de andere kant zou het heel goed mogelijk zijn lagere processen wel vaker worden onderbroken. Een voorbeeld hiervan zou kunnen zijn dat een temperatuur sensor zijn meetwaarde op reguliere basis afgeeft. De waarde van deze sensor zou deel uit kunnen maken van een systeem welke een bewaking is voor een te hoge temperatuur. Juist dan is het dus belangrijk om deze waarde direct af te kunnen handelen.

In het kort kan dus gesteld worden dat afhankelijk van de situatie bepaald moet worden hoe het systeem het beste tot uiting kan komen.

Uitwerking opdracht C

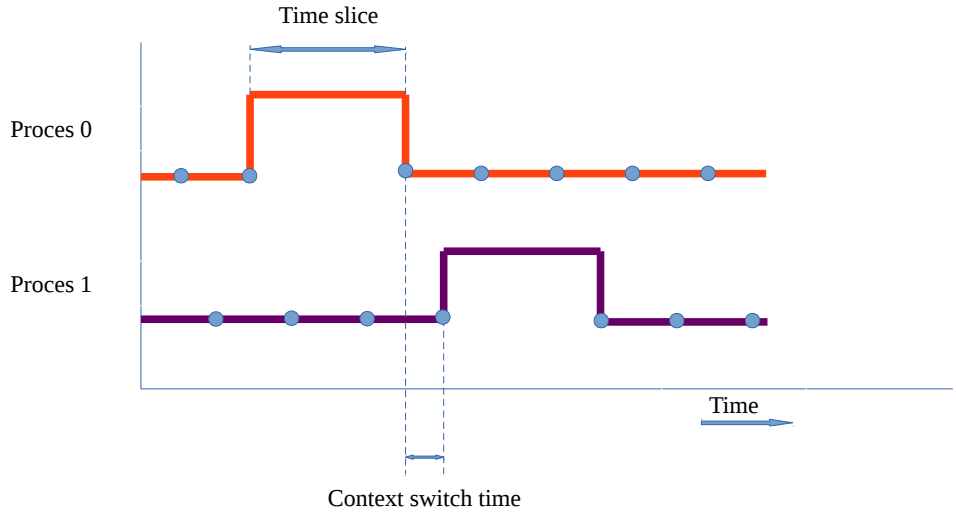
Een belangrijk aspect welke ontbreekt in de simulatie is overhead. Overhead komt voort uit het feit dat de er een *interrupt* afhandeling moet zijn maar ook door de *scheduling* en *dispatching* functie. Al deze zaken kosten immers tijd. Omdat deze naar mijn mening niet in de simulatie wordt meegewogen wordt er zeker geen reëel beeld weergegeven.

2.3 Time slicing

Voor deze opdracht moet er antwoord worden gegeven op de volgende twee vragen:

- Hoe lang duurt een time slice op mijn machine?
- Wat is de context switch tijd op mijn machine?

Om te verduidelijken naar welke tijden we precies zoeken zal ik dit eerst zichtbaar maken in een figuur. In het het figuur wordt door de blauwe bolletjes het moment weergegeven dat een proces een kloktijd opvraagt en opslaat in een array. Als tussen twee meetpunten een langere tijd wordt gemeten dan ' normaal' het geval is dan zou dit dus de gezochte tijd voor de *timeslice* moeten zijn. De *context switch time* kan worden gevonden door het einde van bijvoorbeeld Proces 0 op te zoeken en het begin van Proces 1. Het tijdsverschil tussen deze tijdstippen is de c*ontext switch time*.



Het programma:

Het programma zal in een loop een array vullen met kloktijden. Het array heeft een vooraf gedefinieerde grootte. Nadat het array volledig is gevuld zullen alle tijden middels *printf* worden uitgeprint. De uitvoer van het programma wordt omgeleid naar een file zodat deze later te bekijken is in een editor.

Om de slicetime te kunnen vinden moeten de processen uitgevoerd worden met het Round Robin (RR) scheduling algoritme. Om dit te doen is er in het programma gebruikt gemaakt van de functie *int sched_setscheduler(pid_t pid, int policy, const struct sched_param *param).* Voor het proces id (PID) is de waarde 0 gebruikt waarmee het aanroepende proces wordt geïdentificeerd. Voor de policy is de constante *SCHED_RR* gebruikt. In de *param* parameter kan de prioriteit worden ingesteld. Deze waarde mag variëren tussen 1 en 99. In mijn programma is 25 gebruikt.

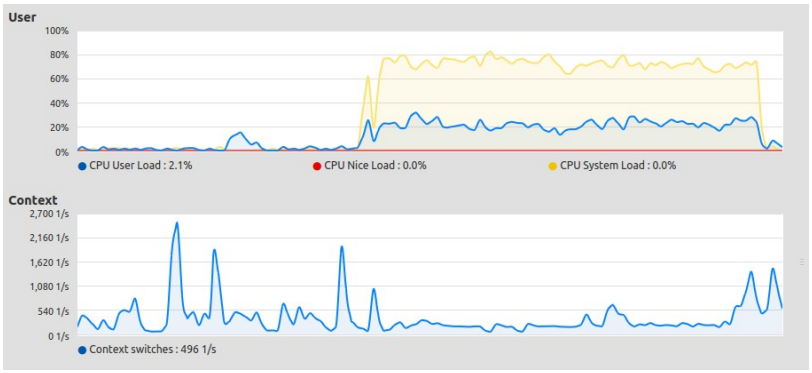
Aan het einde van het programma wordt met de functie *int sched_rr_get_interval(pid_t pid, struct timespec * tp)* de werkelijke waarde van de timeslice opgevraagd. De waarde wordt terug gegeven in de *struct timespec*. In deze struct zit *tv_nsec* welke de timeslice tijd in nanoseconden weergeeft. De gevonden waarde is 100ms en de gemeten waarden zullend dus in de orde van grootte van deze tijd moeten liggen.

Om er voor te zorgen dat beide processen ongeveer gelijktijdig draaien is er een semafoor gebruikt. Proces 0 beheert de semafoor en proces 1 gebruikt hem alleen maar. De processen worden vanaf de command line als volgt opgestart: sudo ./sched 0 > p0 en sudo ./sched 1 > p1. Op deze wijze wordt de output van de programma's naar respectievelijk file p0 en file p1 geschreven.

```
962537   1428517111.473481388, delta = 146
962538   1428517111.473481535, delta = 147
962539   1428517111.473481681, delta = 146
962540   1428517111.473481829, delta = 148
962541   1428517111.473484098, delta = 2269  <------------------------
962542   1428517111.473484242, delta = 144
962543   1428517111.473484386, delta = 144
962544   1428517111.473484531, delta = 145
962545   1428517111.473484676, delta = 145
962546   1428517111.473484819, delta = 143
```

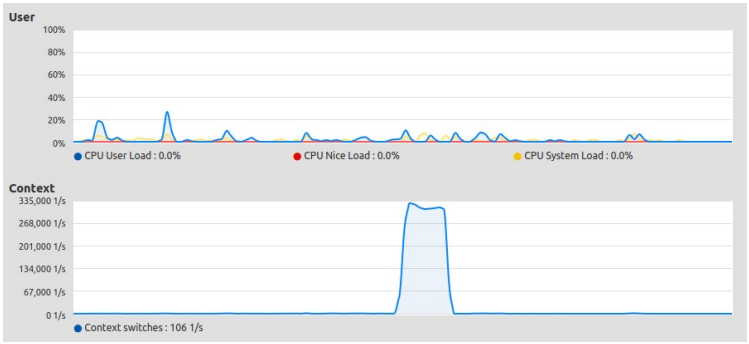
Afbeelding 5

Naderhand heb ik de gegevens geanalyseerd. In afbeelding 5 is een deel van de resultaten te zien. Deze komen dus bij lange na niet bij de verwacht 100ms. Onderzoek met KsysGuard geeft aan dat er nagenoeg geen context switches zijn. Afbeelding 6 laat dus nagenoeg geen context switches zien.



Afbeelding 6

Om context switches te krijgen kijk ik terug naar opdracht 2. Hierin zijn verschillende proeven gedaan waarbij er wel veel context switches. IK heb nu het programma zodanig aangepast door in de loop waarin de tijd wordt opgenomen, een *usleep(1)* te plaatsen. Zoals in afbeelding 7 te zien is zijn de context switches flink toegenomen. Echter afbeelding 8 laat zien dat we nog steeds niet in de buurt van de 100ms zitten



Afbeelding 7

```
548   1428605172.261972434, delta = 7304
549   1428605172.261979748, delta = 7314
550   1428605172.261987052, delta = 7304
551   1428605172.261996493, delta = 9441
552   1428605172.263664427, delta = 1667934
553   1428605172.264668553, delta = 1004126
554   1428605172.264710580, delta = 42027  <
555   1428605172.264807274, delta = 96694  <
556   1428605172.264816475, delta = 9201
557   1428605172.264834260, delta = 17785  <
558   1428605172.264842254, delta = 7994
559   1428605172.264850047, delta = 7793
```

Afbeelding 8