## OPDRACHT 6. PROCESS SCHEDULING

In deze opdracht raak je vertrouwd met het begrip process scheduling, d.w.z. hoe het operating systeem de processortijd verdeeld tussen processen.

### 1. Voorbereidingen

Bestudeer Stallings paragraaf 9.2, 10.2, 10.3.

### 2. Taken

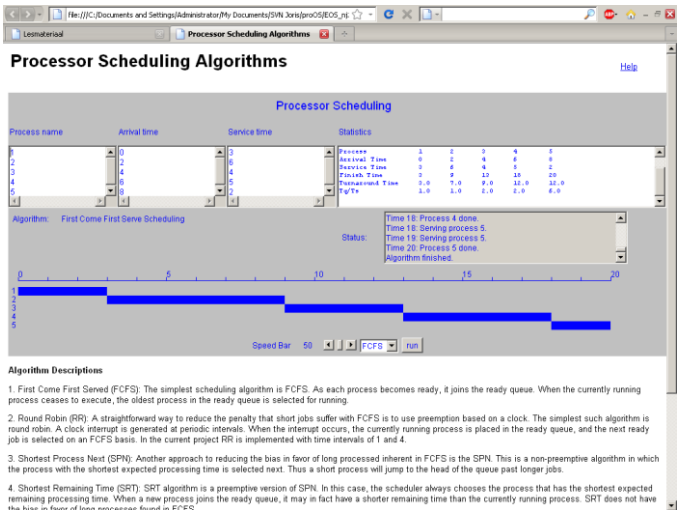
#### 2.1 Process Scheduling

Ga naar de volgende link:
http://courses.cs.vt.edu/csonline/OS/Lessons/Processes/ProcessStateDiagram.swf
en beantwoord de volgende vragen:

- welk process schedulings algoritme wordt toegepast?
- voor welke activiteiten komt het OS (de pinguïn "Tux") in de `Running` state?

#### 2.2 Process Scheduling Algorithms

Haal IPC32_pscheduler.zip van Sharepoint, en start de applet pscheduling.html (onderaan de html pagina staat uitleg) (Deze applet komt oorspronkelijk van elqui.dcsc.utfsm.cl/util/os)
Let er op dat je een grijs veld ziet zoals op dit screenshot, anders staat je beveiliging van Java Applets te streng.




Opdrachten:

- 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.
- Kies zelf een norm voor "verliezen" en "winnen": gaat het om de *gemiddelde* Tq/Ts, of om een *maximale* Tq/Ts, of …?
- De Java applet is niet helemaal natuurgetrouw.
  Welk belangrijk aspect ontbreekt in de simulatie?
  Wat heeft dat voor gevolgen?

Extra uitbreidingen:

- Verbeter de tekortkomingen van de Java applet (ik heb de Java source code liggen)

#### 2.3 Timeslicing

We nemen het programma `dutycycle` van de eerste week.
Verander het programma met allerlei slimme dingen en laat het draaien zodat je de volgende vragen kunt beantwoorden:

- hoe lang duurt een time slice op jouw machine
  - Wanneer je de SCHED_RR scheduling policy gebruikt?
  - Wanneer je de standaard Linux scheduler gebruikt (SCHED_OTHER)?

- wat is de context switch tijd op jouw machine?

Een idee om dit te bepalen is: het programma heeft een while loop waarin de huidige tijd wordt opgevraagd. Zolang dit programma met zijn time slice bezig is, zullen de tijden erg dicht bij elkaar liggen. Als er een gat is tussen twee opeenvolgende tijden dan heeft in de tussentijd een ander proces gedraaid. Op die manier is de grootte van een time slice te bepalen.

Let op: schrijf niet elk tijdstip met `printf` naar de output, want dat kost veel tijd en dat verstoort de meting. In plaats daarvan: sla alle tijdstippen in een (enorm groot) array, en verwerk de waarden na afloop.
En ook: de functies `get_time()` en `convert_time()` geven de tijd in *microseconds*; wijzig dit zodat de tijd in *nanoseconds* wordt ge-`return`-ed.

Als je twee van deze processen draait, dan kan je (na afloop) bij het ene proces het einde van een willekeurige time slice opzoeken, en het begin van de eerste daaropvolgende time slice van het andere proces. In de tijd daartussen is het OS bezig met een context swich.

Je hoeft dus niet in de Linux kernel te hacken; en de getallen hoeven ook niet tot op de laatste decimaal nauwkeurig te zijn.[1] [2]

Je kunt vanuit de shell een proces met een bepaalde scheduling policy en prioriteit opstarten door middel van het Linux commando *chrt* (zie: http://linux.die.net/man/1/chrt). Doe dit met sudo. Je kunt natuurlijk ook POSIX *sched_setscheduler* (http://linux.die.net/man/2/sched_setscheduler) aanroepen vanuit je code. De echte timeslice (time quantum) van de SCHED_RR scheduler kun je opvragen door middel van de *sched_rr_get_interval* call. Dit moet natuurlijk kloppen met de uitkomst van jouw programma. Hiermee kun je dus je programma ijken.

### 3. Opleveren

Je dient een document op te leveren waarin je de bovenstaande experimenten uitwerkt. Voeg, waar nodig, screendumps toe om een en andere duidelijk te maken. Lever ook de bijbehorende source code in van de programma's die je gemaakt of aangepast hebt.

[1] Voor de bepaling van de context switch tijd mag je verwaarlozen dat er in de Linux kernel ook allerlei processen draaien.
[2] Timeslices bij Linux zijn afhankelijk van prioriteiten en I/O gebruik. Voor dit experiment hoef je alleen te werken met enkele user processen met normale prioriteit op een machine die verder niet belast wordt.