# JSF31 – Opdrachten week 3

Als Software-engineer wil je zo goed mogelijke software afleveren, die niet alleen voldoet aan functionele eisen, maar ook aan kwaliteitseisen zoals performance. Inzicht in de performance van je software is hiervoor noodzakelijk. Hieraan gaan we deze week werken.

## Processen en procesbeheer in Linux

1. Maak een tekening van de proces “stamboom” van het Linux systeem vlak na het opstarten.   
   Dit kan met pen en papier, gebruikmakend van de output van het commando ps –eF (als root). Dit commando geeft van ieder proces het eigen process-id (PID) en het process-id van zijn parent (PPID) en veel andere detailinformatie per proces.   
   Je mag ook gebruik maken van een standaard aanwezig shell-commando om dit voor je uit te ‘tekenen’.
2. Start NetBeans op. Zoek het PID van het NetBeans proces (via ps en eventueel grep). Zoek het parent proces van NetBeans, welk proces is dit?

## Processen en Java

Vanuit Java is het mogelijk om processen op te starten, informatie over processen op te vragen en processen te managen. Dit gaan we onderzoeken in de volgende opdracht.

Gebruik JavaDoc of Google om informatie over klassen op te zoeken.

Maak hiertoe in NetBeans in je Linux VM een nieuwe Java Application aan met de naam 'JSF31\_w03\_OS\_process'. In de Wizard kun je bij “Create Main Class” aangeven in welke klasse de main methode moet komen; dit moet klasse 'JSF31\_w03\_OS\_Runtime' in package 'runtime' zijn.

Maak de hieronder gevraagde programma's door code toe te voegen aan de methode main, demonstreer ze aan de docent en zorg dat je hun werking kunt uitleggen.

1. Elke Java applicatie heeft een Runtime object dat de koppeling tussen de Java applicatie en het bijbehorende proces op het OS bijhoudt.   
   Vraag dit Runtime object op, en print de volgende gegevens:
   1. het aantal beschikbare processoren
   2. de totale hoeveelheid geheugen die het proces momenteel beschikbaar heeft
   3. de hoeveelheid geheugen die maximaal beschikbaar gesteld kan worden aan het proces
   4. de hoeveelheid vrij geheugen die het proces momenteel heeft
   5. de hoeveelheid geheugen die het proces momenteel gebruikt (dat is het verschil tussen het momenteel beschikbare geheugen en het vrije geheugen)
2. Het is ook mogelijk om aan de garbage collector van Java een verzoek te doen om objecten vrij te geven. Om dit goed te kunnen zien, moeten we eerst veel objecten maken. Dit doen we door in een loop veel (100000) verschillende String objecten aan te maken:  
   String s;  
   for(int i=0; i<100000; i++) {  
   s = “Hello”+i;  
   }  
   Print vóór en na de loop de hoeveelheid gebruikt geheugen (zie ook vraag 3e). Roep vervolgens de garbage collector aan via gc().   
   Print hierna weer het geheugen.   
   Verklaar wat je ziet.
3. Start een proces (bijvoorbeeld de “gnome-calculator”) op vanuit je applicatie. Wacht een aantal seconden in je main methode (met Thread.sleep) en sluit het proces weer.   
   Doe dit een maal door een ProcessBuilder object te gebruiken (de methode uit de sheets), en een maal door de methode exec uit de klasse Runtime te gebruiken. Bewaar en toon de code.
4. Het is vaak nodig dat je van een opgestart proces de uitvoer opvangt en die weergeeft. Je realiseert dit, door het proces aan een inputstream te hangen, en die uit te lezen. Als proc een Process is, kun je dat als volgt doen:   
   // obtain the input and output streams  
   InputStream is = proc.getInputStream();  
   InputStreamReader isr = new InputStreamReader(is);  
   BufferedReader br = new BufferedReader(isr);  
     
   String line;  
   while ( (line = br.readLine()) != null ) {  
   System.out.println(line);  
   }  
   br.close();  
     
   Test dit door een Linux commando te nemen waarvan je weet wat de output zou moeten zijn (bv “ls” of “pwd”).

## Performance meten (in tijd)

1. Run van bovenstaande project de bijbehorende jar file in een shell (zie week 2, stap 3).   
   Gebruik het Linux commando 'time' (H20.6 uit LinuxFun), om te meten hoelang dit programma runt.
2. Je programma gaan we nu meten vanuit de code zelf.   
   Maak hiertoe gebruik van het gegeven package 'timeutil' (zie SharePoint ). Voeg dit toe aan het NetBeans project (eenvoudigweg door met drag-and-drop de uitgepakte folder timeutil in NetBeans te slepen).  
   Bekijk en run de MetingDemo om te zien hoe je timeutil kunt gebruiken.  
   Voeg nu in de code uit onderdeel 5 methode-aanroepen toe die meten hoelang het creëren van een proces met ProcessBuilder duurt, en hoe lang het duurt met Runtime.exec.  
   Meet ook de tijdsduur voor het hele programma.   
   Vergelijk deze meting met de metingen die je in vraag 7 d.m.v. het Linux commando 'time' hebt gedaan.

## Threads en Java

We gaan nu de bovenstaande code opdelen, zodat het deel dat een proces opstart in een eigen thread kan gaan draaien. Dit doen we in een aantal stappen.

1. Breid je main methode zo uit, dat deze meerdere commando's kan uitvoeren. De uit te voeren commando's moeten meegegeven worden als command line arguments.  
   Gebruik 2 command line arguments per uit te voeren commando om het mogelijk te maken om commando's met 1 optie te kunnen uitvoeren.   
   Gebruik een lege string “” als een commando geen opties heeft.  
     
   Let op: ProcessBuilder en exec gaan op verschillende manieren met de command line arguments om: bij ProcessBuilder worden de commandonaam en de arguments ieder in aparte strings gezet, en bij exec worden commandonaam en arguments allemaal in één string gezet, gescheiden door spaties.  
     
   De commando's dienen na elkaar uitgevoerd te worden, dus elk commando wordt pas gestart als het voorafgaande klaar (terminated) is (gebruik hiervoor een van de methodes van klasse Process).   
   De output van elk commando moet getoond worden in het console, zoals in vraag 6.
   * Voorbeeld:  
     Met de volgende command line:  
     java -jar MyProg.jar ls -al gnome-calculator “” cat /etc/passwd  
     worden achtereenvolgens de commando's   
     ls -al  
     gnome-calculator  
     cat /etc/passwd  
     opgestart.   
     Let op: java moet eigenlijk aangeroepen worden met de volledige padnaam ervoor (zoals in week 2, stap 3); dat is hierboven weggelaten om het commando op 1 regel te laten passen.
2. Voeg code toe om de totale tijd hiervan te meten.  
   Bewaar en toon de code van vraag 9 en 10.
3. Maak nu voor elk met main meegegeven commando een eigen thread aan.   
   Maak hiervoor een nieuwe Runnable klasse, die parameters accepteert in een constructor:
   * het uit te voeren commando, en
   * het met dit commando meegegeven argument.

Als voorbeeld zou je dus 'ls' en '-al' mee moeten kunnen geven om   
'ls -al' te kunnen starten.  
De thread moet het commando starten, wachten tot het commando klaar is, en dan een tekst printen als “command ls -al is terminated” (maar dan natuurlijk niet altijd met 'ls -al', maar met de echte naam van het commando).  
Voeg aan het eind van je main methode een sleep(10000) toe, die er voor zorgt dat de main methode niet te snel klaar is.   
Meet ook nu weer de totale benodigde tijd.  
Demonstreer dit programma aan de docent en leg uit hoe het werkt.