



Inhoudsopgave:

Datum:

1	Hoofdstuk 1: Besturingssysteem	3
1.1	Wat is een besturingssysteem?	3
1.2	Soorten besturingssystemen	4
1.3	Concepten van besturingssystemen	5
1.4	Labo	5
2	Hoofdstuk 2: Virtualisatie & Cloud	6
2.1	Wat is virtualisatie?	6
2.2	Concepten van virtualisatie	7
2.3	Docker	8
2.4	Multi-Tenancy	9
2.5	Cloud Computing	10
3	Hoofdstuk 3: Van programma tot proces.....	13
3.1	Van programma tot proces	13
3.2	Opbouw van een process	14
3.3	Soorten processen.....	15
3.4	Beheer van processen	15
3.5	Scheduling.....	17
4	Hoofdstuk 4: Concurrency	23
4.1	Wat is concurrency.....	23
4.2	Wederzijdse uitsluiting (mutual exclusion)	24
4.3	Synchronisatie	25
4.4	Deadlocks	27
4.5	Quiz:.....	30
5	Hoofdstuk 5: File Systems	32
5.1	Persistente opslag	32
5.2	Files	32
5.3	Directories.....	33

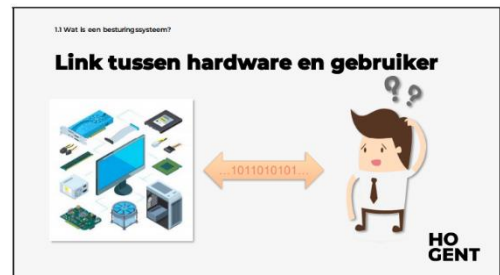
5.4	File systems.....	34
5.5	Partities.....	37
5.6	Booten	39
5.7	Voorbeelden FS	40
5.8	Opslag in Docker.....	42
6	Hoofdstuk 6: Threads.....	44
6.1	Wat zijn threads?	44
6.2	Soorten threads.....	45
6.3	Voor- en nadelen.....	46
6.4	Parallellisme	46
6.5	Voorbeelden.....	48

1 Hoofdstuk 1: Besturingssysteem

1.1 Wat is een besturingssysteem?

Wat is een besturingssysteem?

- Een **directe verbinding** tussen computer en gebruiker.
- Dit systeem praat (**geef instructies**) aan je **hardware** en die hardware praat ook tegen je software, ook geen commandlines om alles op te zetten.
- **Programma dat het mogelijk maakt de hardware van een computer te gebruiken**
 - Hardware = CPU, geheugen, secundaire opslagapparatuur, randapparatuur, ...
- Gebruikers geven geen **opdrachten** aan de computer/hardware, maar aan het **besturingssysteem**
- Het besturingssysteem geeft de hardware de opdracht om de gewenste taken uit te voeren
 - EN: Operating System (OS)

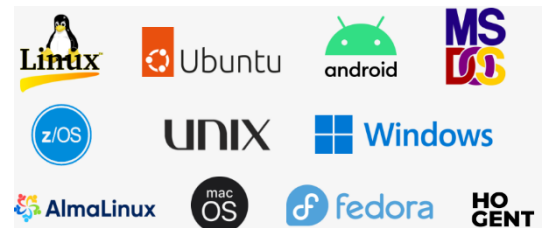


(Extra informatie: MSDOS = 1^e versie Windows)

Dit was een zwart scherm met slechts één enkele lijn.

Taken van een besturingssysteem:




- **Opslaan en ophalen van informatie**
 - Tijdelijk opslaan, voor altijd opslaan
- **Programma's afschermen**
 - Downloads, I/O
- **Gegevensstroom regelen**
- **Prioriteiten regelen**
 - Je download iets, en je kan iets anders doen ondertussen (meerdere taken gelijk uitvoeren)
- **Beheer en delen van bronnen**
 - printers, faxen, softwareapplicaties ...
- **Tijdelijke samenwerking tussen programma's mogelijk maken**
- **Reageren op fouten**
- **Tijdsplanning maken ...**



Nut van een virtuele machine:

MacOS 14 kan je niet installeren op een Mac van 14 jaar geleden, dit komt omdat het **besturingssysteem is verbonden met de hardware**. Op een **virtuele machine** kan dit wel.

Enkele voorbeelden

	Microsoft 	Apple 	Andere 
x86 / x64 laptop/desktop	Windows 10 Windows 11 Windows Server 2022	MacOS 14 Sonoma	Fedora 39 Ubuntu 22.04.1 LTS AlmaLinux 9*
Mobile devices	Windows 10 Mobile (end of support)	iOS 17 iPadOS 17	Android 14 Plasma Mobile Ubuntu Touch ...
ARM devices	Windows 11 on ARM	MacOS 14 Sonoma (for Apple silicon)	Raspberry Pi OS Fedora 39 ARM Ubuntu 22 Server ARM AlmaLinux 9 ARM ...

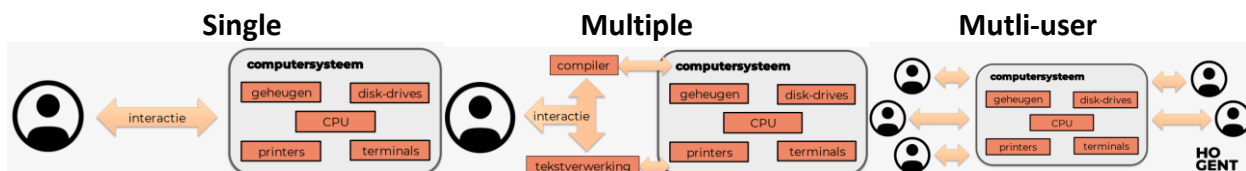
*AlmaLinux: een van de opvolgers van CentOS

Windowsserver = bedrijfsversie

1.2 Soorten besturingssystemen

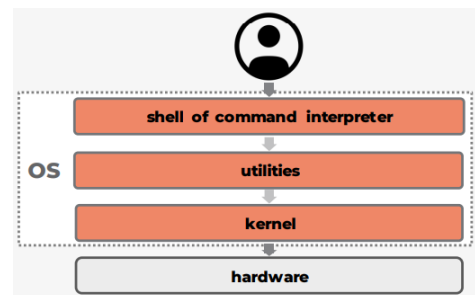
Soorten besturingssystemen:

- **Single-taskingsystemen**
 - Één taak uitvoeren en voor de rest niks, vb. MSDOS
- **Multitasking, single-user systemen**
 - Meerdere taken tegelijk actief zijn
- **Multi-user systemen**
 - Meerdere gebruikers kunnen tegelijk het systeem gebruiken



Verschillende lagen:

- **Kernel:** Softwarepakketten die hierop draaien via **code (C)**.
 - Hier worden **hogere programmeertalen niet gebruikt**, omdat die rechtstreeks in contact staan met je besturingssysteem zoals Java/python. Voordeel = je kan dit draaien op een wasmachine/broodrooster.
- **Utilities:** alles dat boven de hardware geplaatst wordt om het gebruiksvriendelijk te maken.

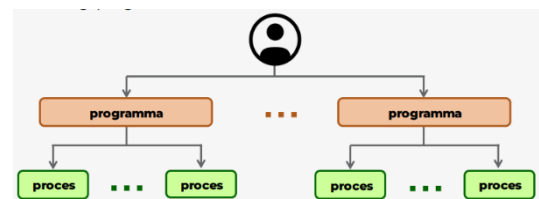


Programma's/taken:

- Een besturingssysteem zorgt er voor dat **taken (programma's) uitgevoerd worden**. We kunnen een onderscheid maken tussen:
 - **Interactieve programma's** (snelle respons)
 - **Batch programma's** (geen directe respons)
 - **Real-time programma's** (respons in een heel beperkte tijd)

Processen:

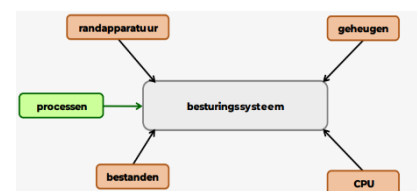
Eén of meer reeksen opdrachten die door een besturingsprogramma worden beschouwd als een **werkenheid**.



Resources:

In eerste instantie spreken processen **resources (bronnen)** aan, bij deze processen worden prioriteiten gesteld. Een PP die wordt voorgesteld is belangrijker dan updates die in de achtergrond zijn gevonden. de PPSlides moeten in Real Time getoond worden (zo snel mogelijk).

Resources die gebruikt worden kunnen ook niet verwijderd worden.



1.3 Concepten van besturingssystemen

Concurrency:

- Processen zijn meestal niet onafhankelijk, processen zijn concurrent Bijvoorbeeld:
- Processen willen dezelfde printer gebruiken
- Processen willen dezelfde file lezen of schrijven
- 2 Processen willen communiceren = conflicten!
- OS regelt in welke volgorde processen afgehandeld worden = synchronisatie (zie verder)

Ontwerp-criteria:

- Consistentie
- Flexibiliteit
- Overdraagbaarheid



1.4 Labo

- Installeren van VirtualBox.
- Virtuele computer opzetten (**naam: Ubunutu**)
 - **Gebruikersnaam: osboxes**
 - **Wachtwoord: osboxes.org**
 - Toetsenbord instellen op BEL AZERTY!!!!

2 Hoofdstuk 2: Virtualisatie & Cloud

2.1 Wat is virtualisatie?

Virtualisatie:

In de computerwereld **verwijst virtualisatie naar het maken** van een **virtuele** (in plaats van dan een echte) **versie** van **iets**, inclusief virtuele computer hardware platforms, opslag apparaten en computer netwerkbronnen.

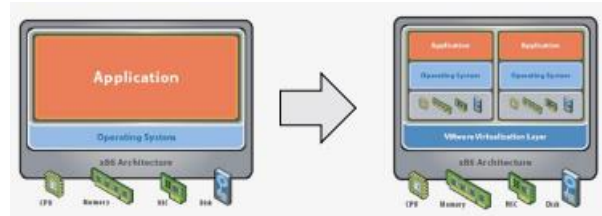
Virtuele architectuur (traditioneel vs virtuele architectuur):

- **Traditioneel** gebruiken we op een computer één besturingssysteem, zoals Microsoft Windows. Het besturingssysteem spreekt de hardware aan, en één of meerdere applicaties draaien bovenop het besturingssysteem. Het is wel mogelijk om op één computer verschillende besturingssystemen naast elkaar te installeren, maar deze kunnen niet tegelijk opgestart worden. Om te wisselen moet je de computer dus opnieuw opstarten. Door de opkomst van **virtualisatie** is het echter wel mogelijk om meerdere besturingssystemen gelijktijdig te gebruiken.
- **Traditioneel = één besturingssysteem**
- **Virtueel = meerdere besturingssystemen tegelijk gebruiken**



Virtuele Hardware:

Een traditionele computer bestaat uit verschillende hardware bronnen, zoals de processor, het geheugen, de netwerkkaart en harde schijven. Met behulp van virtualisatiesoftware gaan we een **virtuele versie maken van deze bronnen**. Elke virtuele machine heeft onder andere een **virtuele processor (vCPU)**, **virtueel geheugen (vRAM)**, één of meerdere **virtuele netwerkkaarten** en één of meer **virtuele harde schijven (virtual disks)**. De virtualisatiesoftware zorgt er voor dat de **fysieke hardware gedeeld** wordt over de virtuele hardware.



Waarom zou je een virtuele machine een bedrijf gebruiken?

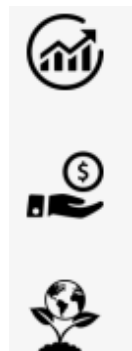
Stel je hebt maar één server en je wil Linux en Windows samen gebruiken dan kan door de andere die je niet hebt virtueel te installeren.

Voordelen van virtualisatie:

- **Efficiënter** gebruik van de beschikbare hardware
- **Goedkoper** dan aparte fysieke systemen
- **Lagere** ecologische voetafdruk

Nadelen van virtualisatie:

Performantie is redelijk wat slechter als normale besturingssystemene.



2.2 Concepten van virtualisatie

Virtuele Machine:

Een **virtuele machine (VM)** is een **computerprogramma** dat een **volledige computer nabootst**, waar andere (besturings-)programma's op kunnen worden uitgevoerd.



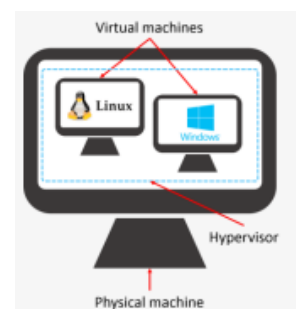
Soorten Virtuele Machines:

- **Programmeertaal-specifiek**
 - vb. Java Virtual Machine (JVM)
- **Emulator**
 - vb. VirtualBox, V MWare, ...
- **Applicatie-specifiek**
 - vb. Docker



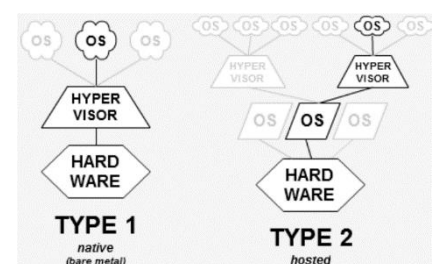
Hypervisor:

- Een **hypervisor** is de **software** die gebruikt wordt om **virtuele machines** aan te **maken**, op te **starten** en te **draaien**.
- Een hypervisor wordt ook vaak **Virtual Machine Monitor (VMM)** genoemd.
- Een hypervisor regelt de virtualisatie. Met een hypervisor kan één hostcomputer meerdere VM's gelijktijdig laten draaien door de (hardware) **bronnen** virtueel te **delen**, zoals het geheugen en de processor.



Type 1 vs. Type 2 Hypervisors:

- Type 1 hypervisor
 - Native (bare metal) hypervisor
 - Rechtstreeks op de hardware, er is **geen onderliggend OS**
 - Kan **rechtstreeks** de nodige **hardwarebronnen aanspreken**
 - Geen overhead door OS
 - **Voordeel: Efficiënter** maar ook **complexer**
 - Vooral gebruikt op **servers**, soms ingebouwd in hardware
 - **Nadeel: dure** contracten, moeilijker om te **managen/installeren**
 - **Hyper-V, VMWare ESXi, ...**
- Type 2 hypervisor
 - **Programma** bovenop een besturingssysteem (OS)
 - **Aanspreken** van **hardware** gebeurt via functies van het **besturingssysteem**
 - **Eenvoudiger** om te installeren (programma)
 - Vooral gebruikt op **persoonlijk toestel** (bv. Windows VM op Mac)
 - Niet zo krachtig en efficiënt als Type-1
 - **VirtualBox, VMWare Player, Parallels Desktop, ...**
 - Wij gebruiken type 2



2.3 Docker

Docker:

- Docker is virtualisatiesoftware die **containers** kan bouwen, uitvoeren en beheren.
- Het bootst **geen volledige computer** na maar de containers spreken rechtstreeks **de kernel van het besturingsysteem van de host aan**.



Docker commando's:

- optie -t: naam geven container anders geef random naam

Wat is een container?

- Uitvoerbare versie van een **container image**
- Volledig **gesandboxed proces** op jouw host, geïsoleerd van alle andere processen.
- Lijkt alsof het alleen draait op de host: **eigen file systeem, eigen procestabel...**
- Gebruikt kernel namespaces, cgroups
- Gedraagt zich op **elk systeem identiek hetzelfde**
- Uitvoerbaar op een **lokaal systeem, virtuele machine en in de cloud**.

Wat is een container image?

- **Blauwdruk voor een container**: bevat alles wat de container nodig heeft om te functioneren
- **Bevat het hele bestandssysteem van de container**:
 - Dependencies, configuraties, scripts, binaire bestanden ...
- Bevat ook andere **informatie**:
 - Omgevingsvariabelen, standaard startcommando, metadata ...

Dockerfile:

- De Dockerfile is het **belangrijkste bestand** van een **container image**. Dit bestand specificeert **hoe de image eruit zal zien**:
 - **Welke bestanden** naar de image gekopieerd worden
 - Welke **environment variables** er zijn
 - **Welke commando's** uitgevoerd moeten worden **wanneer de image gebouwd wordt**
 - **Welke commando** uitgevoerd moet worden **als de container start**, enz.

Port bindings:

- **Containers krijgen een IP-adres in een Docker netwerk**
 - Een container kan in meerdere Docker netwerken zitten
 - Een docker netwerk is een intern virtueel netwerk
- Standaard worden geen poorten beschikbaar gemaakt op de host
- **Met port bindings kan je een poort van de container beschikbaar maken op de host**
- Je koppelt dus een poort van de container aan een poort op de host

```
docker run -p 8080:80 httpd
```

- Poort op host: 8080
- Poort in container: 80

Volumes:

- **Probleem:**
 - container heeft een eigen bestandssysteem
 - container verwijderen → data verdwenen
- Volumes worden gebruikt om data van containers te persisteren of om data te delen met andere containers
 - Backup van data in een databankcontainer
 - Code in container updaten tijdens development ...

Volumes: types

- **Volume**
 - Volledig beheerd door Docker
 - Bevindt zich in de "Docker area"
- **Bind mount**
 - Map op de host in de container plaatsen (~ snelkoppeling)
 - Map mag eender waar staan
- **Tmpfs mount**
 - Tijdelijke opslag in het geheugen van de host

```
docker run -v /my/own/website:/usr/local/apache2/htdocs/ \
-p 8080:80 httpd
```

```
- Pad op host: /my/own/website
- Pad in container: /usr/local/apache2/htdocs
```

Docker Compose:

- **Tool om meerdere (samenhorende) Docker containers te beheren**
- Definieer de containers in een **YAML**-bestand: `docker run -p 8080:80 httpd`
dockercompose.yml
- Start, stop... alle containers met **één commando**
 - **Individueel** ook nog **mogelijk**



2.4 Multi-Tenancy

Single-Tenant vs. Multi-Tenant:



Kenmerken van Multi-Tenancy:

- Bronnen worden **gedeeld**, in tegenstelling tot een dedicated of isolated omgeving
- Een **tenant** (huurder) is een **gebruiker of groep** van gebruikers met **gemeenschappelijke toegang**
- Multi-tenancy kan geïmplementeerd worden in **verschillende vormen**, zowel op niveau van hardware als software
- **Virtualisatie** speelt een belangrijke rol bij multi-tenancy
- Multi-tenancy is een belangrijk kenmerk van **Cloud Computing**

Voor- en nadelen Multi-Tenancy:

- AYES
 - **Efficiënter** gebruik van de beschikbare bronnen, want meerdere eindgebruikers kunnen bediend worden door één toestel of instantie van de applicatie
 - Daardoor leidt multi-tenancy tot **lagere operationele kosten dus goedkoper** voor de eindgebruiker
- NOES
 - **Minder isolatie en verhoogde beveiligingsrisico's**, in het geval van een inbreuk op de beveiliging op één enkele instantie kunnen meerdere tenants worden getroffen
 - **Minder prestatie-isolatie**, grote tenants kunnen de prestaties van kleinere tenants negatief beïnvloeden
- Een multi-tenant omgeving moet zowel een **duidelijke scheiding** van **prestaties** als **gegevens garanderen**. Hoewel één hardware- of software-instantie wordt gedeeld, moet deze zich ten opzichte van elke tenant als een aparte instantie gedragen.

2.5 Cloud Computing

Cloud computing:

Cloud Computing is het via een **netwerk**, vaak het internet, op **aanvraag beschikbaar** stellen van **hardware, software en gegevens**, ongeveer zoals elektriciteit uit het lichtnet. **Het beschikbaar zetten van resources over het netwerk.**

Kenmerken van Cloud Computing:

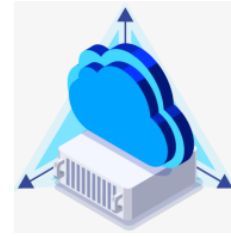
- **Bronnen** beschikbaar **op aanvraag** (on demand), vaak **zonder tussenkomst van een fysiek persoon**
 - **Bronnen = hardware, software en/of opslag**
 - Vaak: groot aantal servers in datacenter die door iedereen gehuurd kunnen worden, bijvoorbeeld als VM's
- Vaak via een **pay-as-you-go** pricing model (kostprijs afhankelijk van effectief verbruik)
 - ipv wanneer je server een piek moment krijgt (januari) een tweede server koopt (omdat hij dan na de piek stof staat te happen), doe je de pay-as-you-go.
- **Elasticiteit**: mogelijkheid om automatisch aan te passen in functie van vraag

Opkomst Cloud Computing;

- Eerste definitie **1997 (Ramnath K. Chellappa)**
- **2000** opkomst **Software as a Service**
- **2005-2007 opkomst Cloud Computing**
 - Amazon, Google, IBM en universiteiten
 - Hand in hand met opkomst virtualisatie
 - VMWare, VirtualBox, Microsoft en Citrix
- Analogie met mainframes (servers) en terminals (clients)
 - Geen centraal mainframe maar een gedistribueerde omgeving

Cloud:

- **Wolk van computers**
- Virtualisatie van **serveromgeving**
 - Eindgebruiker weet niet
 - **Waar** de instanties precies draaien
 - Soms ook niet op **hoeveel servers**
 - Eindgebruiker is
 - **Geen eigenaar** meer van hardware/software
 - **Niet verantwoordelijk** voor onderhoud
- Schaalbare, virtuele infrastructuur

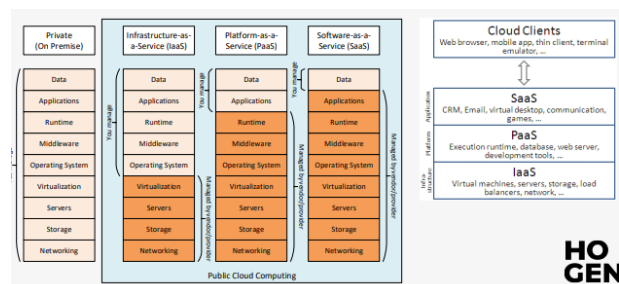


Deployment modellen:

- **Publieke** Cloud omgeving
 - Beschikbaar voor **iedereen**, via het internet
- **Private** Cloud omgeving
 - Toegang beperkt **tot één of meerdere organisaties**
 - Kan in **privaat (self-hosted)** of **publiek (collocated)** datacenter
- **Hybride** Cloud omgeving
 - **Combinatie van meerdere modellen**
- Er bestaan ook vele afgeleide vormen
 - Bv. Community Cloud, Distributed Cloud, Multi-Cloud, ...
- Soms ben je **niet eens baas van je eigen cloudmodel**. Als de 'baas' een beslissing maakt over de cloud effect het ook jouw cloud.

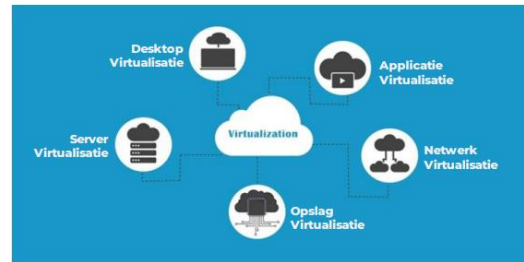
Service modellen (publieke cloud):

- **Infrastructure as a Service (IaaS)**
 - Infrastructuur **virtueel beschikbaar** voor gebruiker
 - Gebruiker heeft **controle over besturingssysteem**, software en (virtuele) hardware
 - Amazon EC2, ...
- **Platform as a Service (PaaS)**
 - Platform en diensten (bv. toegangsbeheer) aangeboden
 - Gebruiker heeft **controle over software**, maar **geen** controle over onderliggende **hardware**
 - Microsoft Azure App Service, Google AppEngine, ...
- **Software as a Service (SaaS)**
 - **Aanbieden applicaties**
 - Gmail, Office 365, Salesforce, ...
 - Zelf niets schrijven, zelf niets doen
 - handigste van alle services omdat de code hiervoor te schrijven niet makkelijk is
- Er bestaan ook vele afgeleide vormen (BPaaS, FaaS, DaaS, DBaaS, SECaaS, ...)



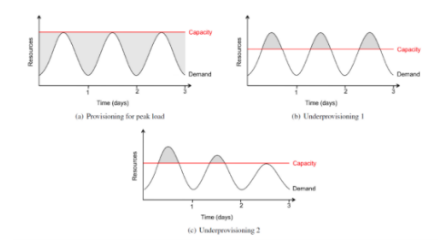
Cloud Computing = virtualisatie:

- Virtualisatie speelt een **belangrijke** rol binnen Cloud Computing. Bij cloud computing worden dan ook vaak virtuele **bronnen beschikbaar gesteld** aan de eindgebruiker, bijvoorbeeld in de vorm van virtuele machines.
- Zonder virtualisatie zouden er veel minder klanten gebruik kunnen maken van de Cloud omgeving**, en zouden er veel bronnen verspild worden.
- Naast het gebruik van **virtuele machines (server virtualisatie)** worden er binnen Cloud omgevingen ook **vele andere vormen van virtualisatie** gebruikt, zoals **virtualisatie van opslag, netwerk** virtualisatie, virtualisatie van **applicaties** en zelfs virtualisatie van **volledige desktopomgevingen**.



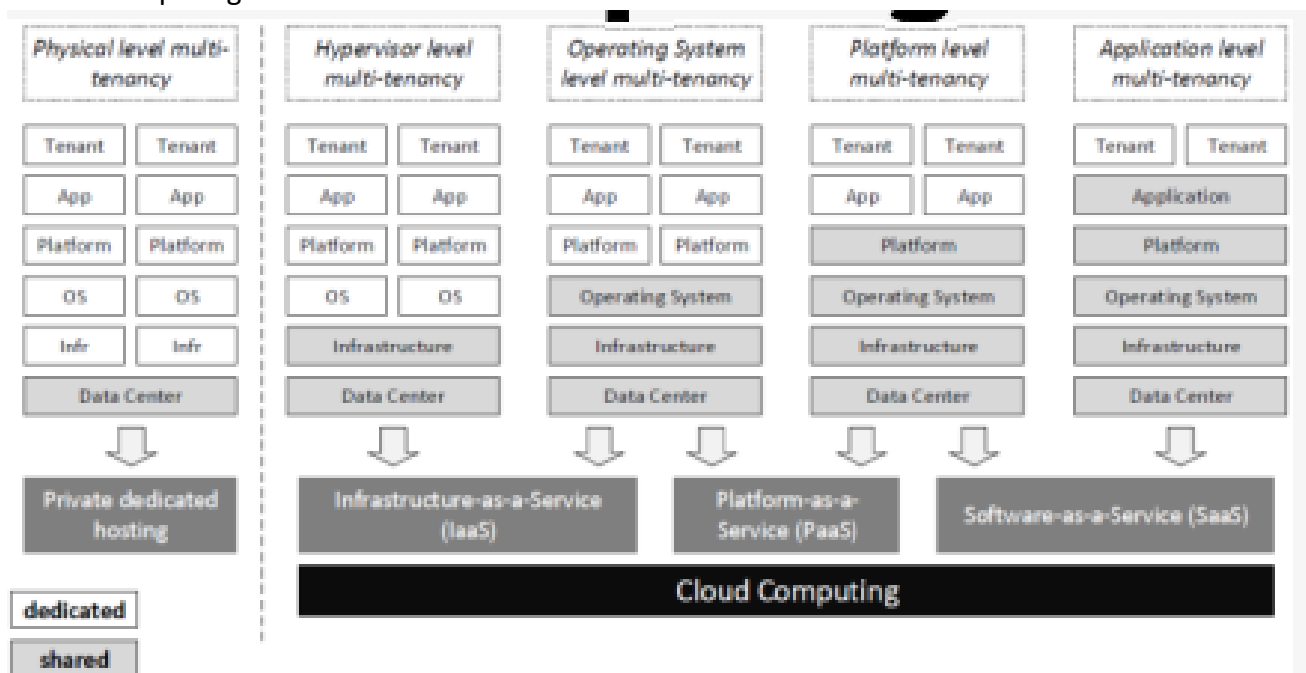
Elasticiteit van de Cloud:

- De mate waarin het **aanbod** zich **afstemt** op een stijgende of dalende **vraag**
 - Meer resources voorzien tijdens piekbelasting, minder bronnen tijdens daluren
- Automatische **provisie** van benodigd aantal **instanties**
 - Hoe voorspellen?
 - Autonoom, en dus zelflerend?
- Publieke Cloud: “**oneindige**” hoeveelheid **bronnen**
 - Elasticiteit is een belangrijk kenmerk van publieke cloud omgevingen!
- Een cloud kan nooit vol zitten in principe (google drive zal nooit zeggen “ik zit vol”)
- Amazon = cloud service provider (de dag van vandaag, vroeger niet)



Cloud Computing = Multi-Tenancy:

- Cloud computing is een mooi voorbeeld van multi-tenancy.**
- Zeker bij public Cloud computing maken dus vaak meerdere eindgebruikers gebruik van dezelfde set fysieke hardware.
- De figuur dient ter illustratie, maar toont mooi hoe verschillende vormen van multi-tenancy gerelateerd zijn aan de verschillende service modellen van public Cloud computing.



3 Hoofdstuk 3: Van programma tot proces

3.1 Van programma tot proces

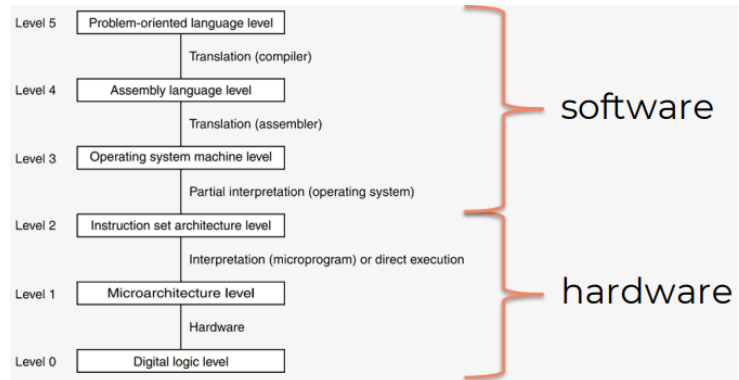
Compilers:

Niet elk programmeertaal verstaat onze hardware, elk programma moet dus worden omgezet **van programmeertaal tot een verzameling instructies uit die instructieset**.

Java stop vanaf Level 3.

Wat is een proces?

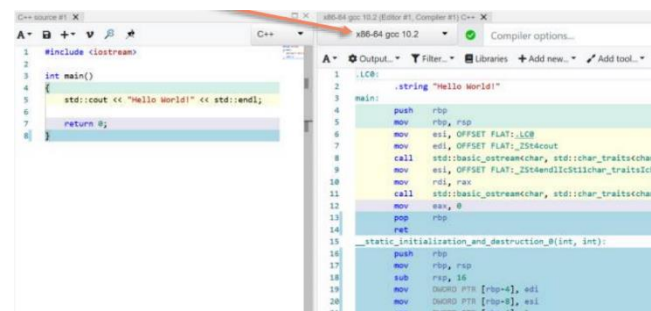
Een uitvoering op je programma (de live versie want je runt het).



voorbeeld compileren:

<pre>i = j + k; if (i == 3) k = 0; else j = j - 1;</pre>	<pre>1 ILOAD j // i = j + k 2 ILOAD k // i = j + k 3 IADD // i = j + k 4 ISTORE i // i = j + k 5 ILOAD i // if (i == 3) 6 BIPUSH 3 // if (i == 3) 7 IF_ICMPEQ L1 // if (i == 3) 8 ILOAD j // j = j - 1 9 BIPUSH 1 // j = j - 1 10 ISUB // j = j - 1 11 ISTORE j // j = j - 1 12 GOTO L2 // k = 0 13 L1: BIPUSH 0 // k = 0 14 ISTORE k // k = 0 15 L2:</pre>	<pre>0x15 0x02 0x15 0x03 0x60 0x36 0x01 0x15 0x01 0x10 0x03 0x9F 0x00 0x0D 0x15 0x02 0x10 0x01 0x64 0x36 0x02 0xA7 0x00 0x07 0x10 0x00 0x36 0x03</pre>
(a)	(b)	(c)

Figure 4-14. (a) A Java fragment. (b) The corresponding Java assembly language. (c) The JVM program in hexadecimal.



Instructieset:

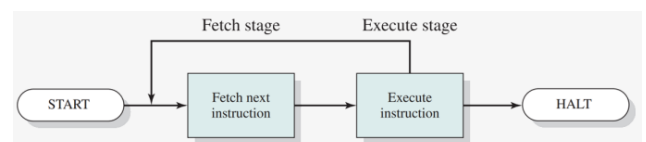
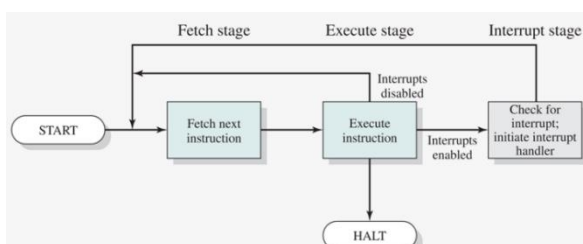
Uniek per type CPU Bv. x86, x86-64, ARM, MIPS, JVM, 8051, ...

Instructiecyclus:

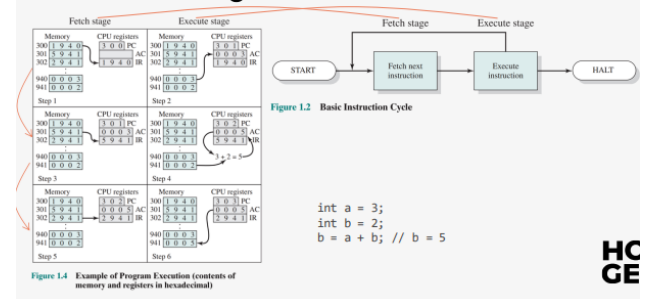
Eerst wordt een **instructie opgehaald (fetch)**, daarna wordt die **uitgevoerd (execute)**, dan gebeurt de **execute stage** en daarna de **fetch stage** uiteindelijk begint deze cyclus opnieuw tot alle instructies op zijn. Dit is het proces van de CPU op vroegere pc's.

Bij de tweede cyclus moet je enkel weten dat alles gebeurt via **fetch (links)** en **execute (rechts)**.

Interrupts:



Instructiecyclus



Een interrupt is bv. Ctr + c in Windows Powershell of het uitzetten van je pc.

Kortom, de opdracht/uitvoering stoppen op een manier.

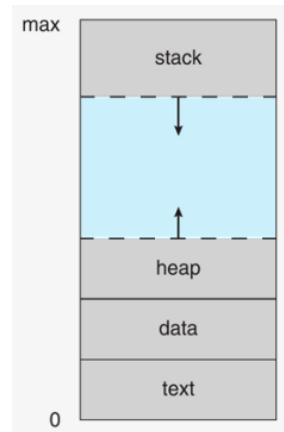
3.2 Opbouw van een process

Proces:

Een **proces** is een **instantie** van een programma dat uitgevoerd wordt op het systeem.

Proces Image:

- hoe het proces er uit ziet in het RAM geheugen
- Text: de uit te voeren **instructies**.
- Data: **globale variabelen**.
- Stack: tijdelijke opslag voor **variabelen, functie parameters**, adressen voor return uit functies, ...
- Heap: dynamisch gealloceerd geheugen – **een hoop waar allemaal data op wordt gegooit** (blijft beschikbaar tot het door de programmeur wordt opgekuist.)



Proces Control Block (inhoud van blokken niet kennen, figuur begrijpen is genoeg):

- Priority = kijkt naar de volgorde van je codefuncties
- Program counter = de code die je hebt geschreven
- Context data = variabelen
- Accounting information =
- Memory pointers =

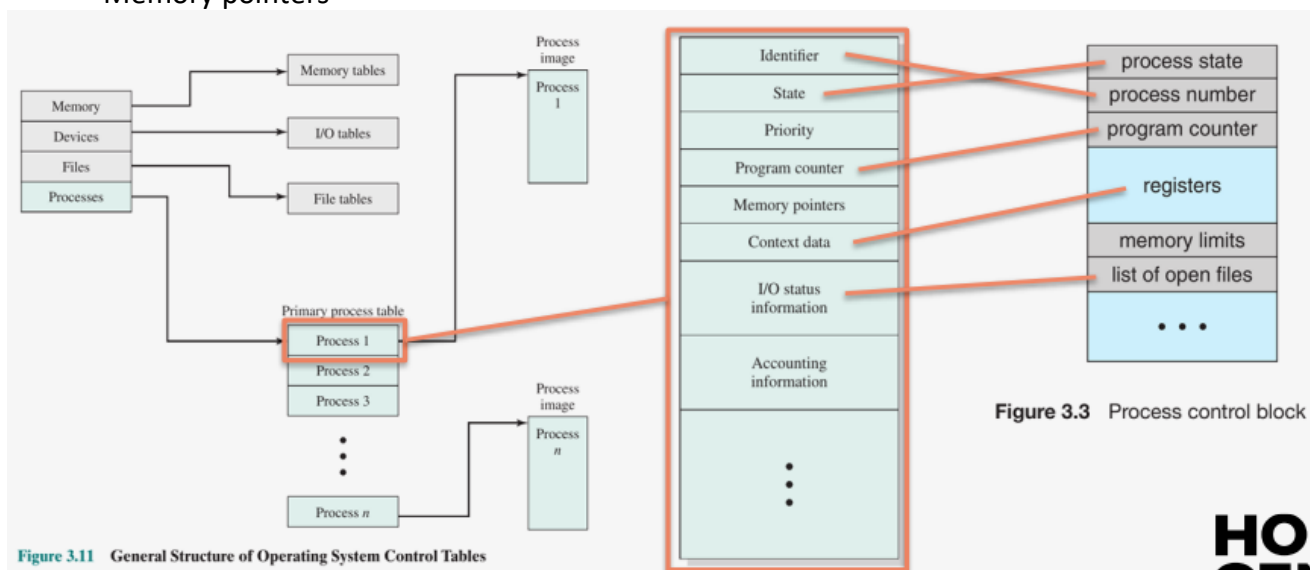


Figure 3.3 Process control block

Waar worden de instructies van een programma ingeladen wanneer het opgestart wordt?
RAM-geheugen

Context switchen:

De volgende instructie die moet worden geladen

3.3 Soorten processen

Soorten processen:

- **Interactief**
 - Processen die jij zelf bedient
- **Automatisch**
 - Processen die in een **wachtrij** worden geplaatst, wachten op uitvoering via LIFO
- **Daemons**
 - Achtergrondprocessen

Interactief proces:

- Opstarten en controleren vanuit een **terminal sessie**
- Kunnen zowel in voorgrond (foreground) als achtergrond (background) draaien
 - Foreground proces
 - **Blokkeert terminal** zolang het loopt
 - Background proces
 - **Blokkeert terminal enkel bij opstart van het proces**, nadien kan terminal andere taken uitvoeren

Automatische processen:

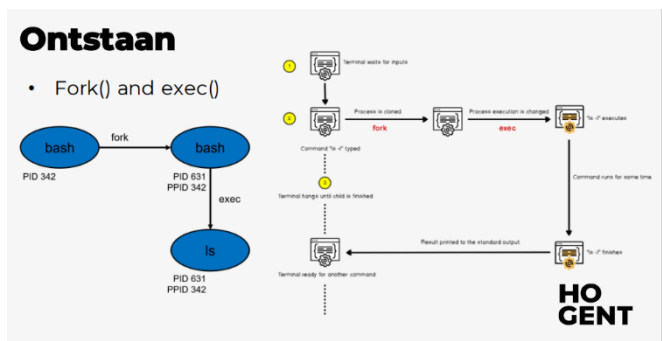
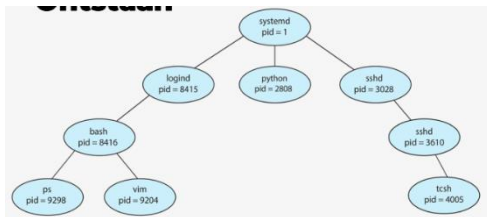
- Ook wel "**batch**" processen genaamd
- Verzameling van processen die in een **wachtrij** worden geplaatst **wachtend op uitvoering**
- Bij uitvoering worden alle processen b. Dit volgens het First In, First Out (FIFO) principe
 - Voorbeeld: automatisch een back-up maken, elke dag om middernacht

Daemons:

- **Processen die continu draaien**
- Veelal gestart bij opstarten van systeem
- Wachten in de achtergrond tot ze nodig zijn
- Ook gekend als services Voorbeeld: onedrive synchronisatie client, e-mail server, ...

3.4 Beheer van processen

Fork() en exec():



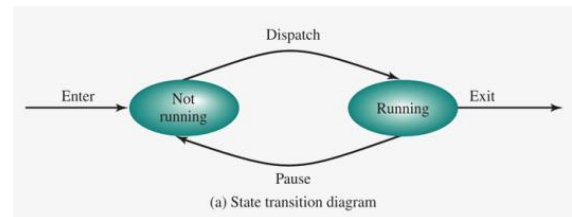
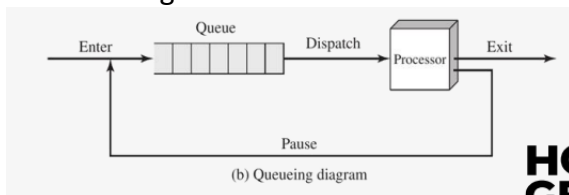
- **Services** zullen meestal een "**d**" achteraan het woord hebben, bv. **Mysqld**
- **Fork()**: Deze functie maakt een **exakte kopie** van het proces in het RAM geheugen (**een kopie van diens address space dus**) en vult de PID van het proces in met een nieuw ongebruikt procesnummer. Daarnaast vult de functie het PPID (Parent Process Identifier) in met het PID van het ouderproces.
- **Exec()**: Het nieuw aangemaakt kindproces wordt overschreven met de nodige waarden voor het gewenste proces. Zo worden bijvoorbeeld de juiste instructies, waarden, ... Ingelezen naar het procesbeeld van het kindproces.

Afbraak:

- Het proces is **afgewerkt** of er treedt een **fout** op:
 - Indien één van beide gebeurt wordt **Exit()** automatisch uitgevoerd.
- Het proces wordt afgebroken door een ander proces:
 - Bv. Kill
- Soms gaat er iets fout:
 - **Zombie proces** (een proces dat **mag geschrapt worden** uit je geheugen, maar wel nog in een procestabel zit)
 - **Orphan proces** (een proces dat geen parent meer heeft, **de returncode kan niet worden ontvangen**)

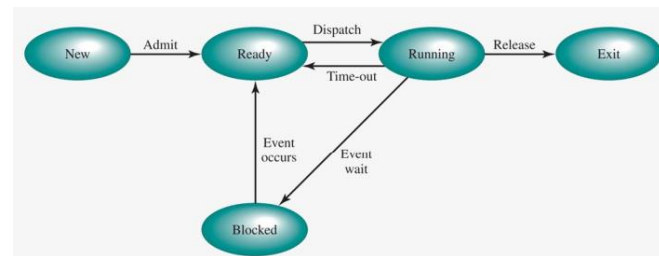
Processen hebben 2 States:

- Not Running
- Running



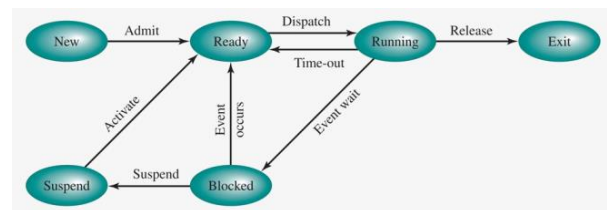
5 States:

- **New**
 - Een nieuw proces aangemaakt door het besturingssysteem
- **Ready**
 - Een proces dat wacht tot het op de CPU mag
- **Running**
 - Het proces wordt uitgevoerd op de CPU
- **Blocked**
 - Een proces dat staat te wachten op iets
- **Exit**
 - Een afgewerkt proces (afgewerkt natuurlijk of stopgezet)



6 states:

- **New**
 - Een nieuw proces aangemaakt door het besturingssysteem
- **Ready**
 - Een proces dat wacht tot het op de CPU mag
- **Running**
 - Het proces wordt uitgevoerd op de CPU
- **Blocked**
 - Een proces dat staat te wachten op iets
- **Suspend**
 - Alleen kan het soms gebeuren dat dat RAM geheugen vol geraakt. Het proces wordt dat in een tijdelijk rooster gestopt en wanneer er weer RAM vrij is, komt hij terug naar het RAM
- **Exit**
 - Een afgewerkt proces (afgewerkt natuurlijk of stopgezet)



Swapping (suspend wederom) + (bespaart RAM-geheugen):

- Op een systeem zie je bv. dat het RAM geheugen volloopt.
- Het systeem riskeert daardoor vast te lopen. Om dat te vermijden zal het systeem processen wegschrijven naar de harde schijf of SSD om **RAM geheugen vrij te maken** om ervoor te zorgen dat het systeem niet in de problemen komt.
- Op Windows wordt er weggeschreven naar de pagefile.sys en swapfile.sys bestanden op de C:\s

3.5 Scheduling

Multiprogramming:

- **CPU zo optimaal mogelijk gebruiken**
 - I/O is trager dan CPU
 - Sommige processen moeten wachten op iets (zoals bv. I/O)
 - **2 processen schakelen heel rap na elkaar (ze worden niet tegelijk uitgevoerd, maar heel snel)**
- Wanneer een proces moet wachten, kan een ander proces gebruik maken van de CPU

$$\text{CPU Utilization} = \frac{\text{CPU Usage Time}}{\text{Total Time}}$$

Streven naar 100 %
(Best met een marge: zie notes *)

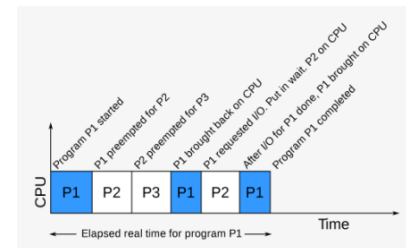
$$\text{CPU Idle} = \frac{\text{CPU Idle Time}}{\text{Total Time}}$$

Streven naar 0 %

Computer Action	Avg Latency	Normalized Human Time
3Ghz CPU Clock cycle 3Ghz	0.3 ns	1 s
Level 1 cache access	0.9 ns	3 s
Level 2 cache access	2.8 ns	9 s
Level 3 cache access	12.9 ns	43 s
RAM access	70 - 100ns	3.5 to 5.5 min
NVMe SSD I/O	7-150 µs	2 hrs to 2 days
Rotational disk I/O	1-10 ms	11 days to 4 mos
Internet: SF to NYC	40 ms	1.2 years
Internet: SF to Australia	183 ms	6 years
OS virtualization reboot	4 s	127 years
Virtualization reboot	40 s	1200 years
Physical system reboot	90 s	3 Millenia

Time sharing:

- Multitasking
- Multi-user
- Processen op de CPU wisselen elkaar (snel) af
- Geeft de illusie dat processen tegelijkertijd worden uitgevoerd



Context switch:

- Bij het wisselen van een proces treedt een **context switch** op: er wordt een **snapshot** genomen van het **volledige proces** en bewaard in het geheugen. Dan wordt de snapshot van het **andere proces** ingeladen en dat proces **uitgevoerd**.
- Als het terug de beurt is aan het eerste proces, wordt er opnieuw een snapshot van het tweede proces bewaard en de snapshot van het eerste proces ingeladen. Het proces doet verder vanwaar de snapshot is opgenomen alsof er niets is gebeurd.
- Het nemen van een snapshot heeft wel een kleine overhead: verschillende data moeten worden opgeslagen, zoals de stack, heap, data, registers (PC, ...), ... en dit neemt een klein beetje tijd in beslag. Hetzelfde geldt voor het terug inladen van een snapshot.

Scheduler:

- Verantwoordelijk voor het bepalen wanneer welk proces CPU tijd krijgt
- Afweging tussen verschillende soorten processen:
 - Batch
 - Iets opslaan
 - Interactive
 - de vraag "Wil je dit installeren"
 - Real time
 - Iets dat gebeurt direct
- Hoe deze beslissing nemen? → Niet eenvoudig

Scheduler vervolg:

- De scheduler bepaalt wanneer **welk proces CPU-tijd krijgt**
- Geen eenvoudige beslissing: combinatie van
 - Prioriteit
 - Status
 - Volgorde in de wachtrij
- Dit kan sterk verschillen tussen processen van bv.:
 - Interactieve programma's
 - Batch programma's
 - Real-time systemen

Zelf een scheduler maken, hoe zou je dit doen?

- **Gebruik FIFO of First Come First Served (FCFS)**
 - **Makkelijk**
 - **Niet efficiënt**
- **Shortest first**
 - **Proces eindigen**
 - **Kleine eerst**
 - **Starvation**
- **Round Robin**
 - Elk proces evenveel tijd geven
- **Shortest Remaining Time**
 - Proces wordt onderbroken indien er een proces is met een kortere uitvoeringstijd
- **Shortest Proces Next**
 - Kleinst proces eerst

Preemptive:

- = **Onderbreken huidig proces**
- Nodig indien proces systeem kan monopoliseren
- Wisselen van proces zorgt voor extra overhead
- Huidige proces wordt terug in **wachtrij** gezet **indien niet voltooid**

Examenvraag: welk proces kies je eerst: een oude of een nieuw?

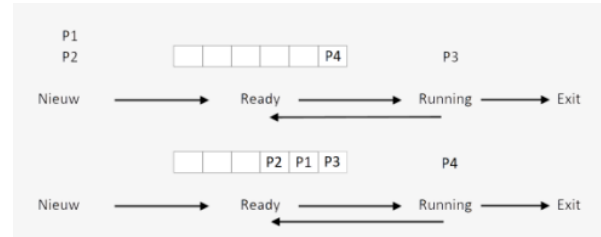
Oude processen komen eerst

Preemptive: voorrang?

- Theoretisch probleem: een nieuw proces komt toe net op het moment dat een running proces onderbroken wordt
 - Welk proces wordt eerst aan de **wachtrij** toegevoegd?
 - In de echte wereld is de kans klein dat dit probleem voorkomt, want er zal meestal één proces iets vroeger zijn dan het andere proces
 - In onze (vereenvoudigde) voorbeelden komt dit echter wel vaak voor omdat we werken met discrete tijdseenheden

Vorrang?

- Werkwijze 1
 - **Onderbroken proces voor nieuwe processen**
- Werkwijze 2
 - **Nieuwe processen voor onderbroken proces**



Starvation:

- In sommige gevallen kan het zijn dat **bepaalde processen** nooit **CPU** tijd krijgen
- Mogelijk scenario:
 - De scheduler geeft voorrang aan korte processen
 - Er komen steeds nieuwe korte processen in het systeem
 - Hierdoor worden lange processen telkens uitgesteld

Voorbeelden van type schedulers:

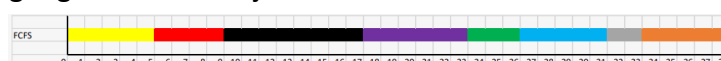
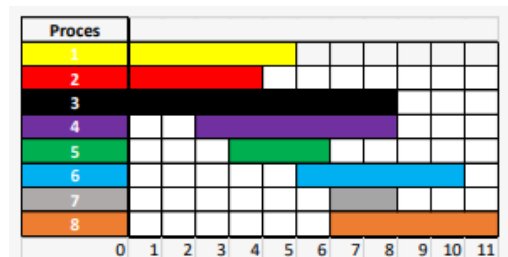
- **Zonder onderbrekingen**
 - First Come, First Serve (FCFS)
 - Shortest Process Next (SPN)
- **Met onderbrekingen**
 - Shortest Remaining Time (SRT)
 - Round Robin (RR)

First Come, First Served (FCFS):

- Proces wordt **volledig uitgevoerd**, geen onderbrekingen
- Processen worden in **wachtrij** geplaatst indien nodig
- Volgende proces? Eerste proces uit de wachtrij
- **+ Geen starvation, heel gemakkelijk**
- **- Korte processen kunnen lang moeten wachten, proces kan systeem monopoliseren**

FCFS:

- Tabel bevat **overzicht van alle processen**
- Horizontale as = tijdslijn
 - Startpunt = tijdstip dat proces aangeboden wordt
 - Je komt binnen in de wachtrij
 - Hoelang lopen? Tot einde proces
 - geel = 5 (turnaround time)
 - **Aantal gekleurde vakjes = tijd dat proces nodig heeft om uitgevoerd te worden**
 - B.v.: groene proces
 - Start op tijdstip 3
 - 3 tijdseenheden nodig om uitgevoerd te worden
- Onderstaande tabel bevat overzicht van alle processen na planning volgens FCFS methode volgens tijdslijn
- B.v.: groene proces
 - Effectief gestart na 23 tijdseenheden
 - Volledig afgerond na 26 tijdseenheden



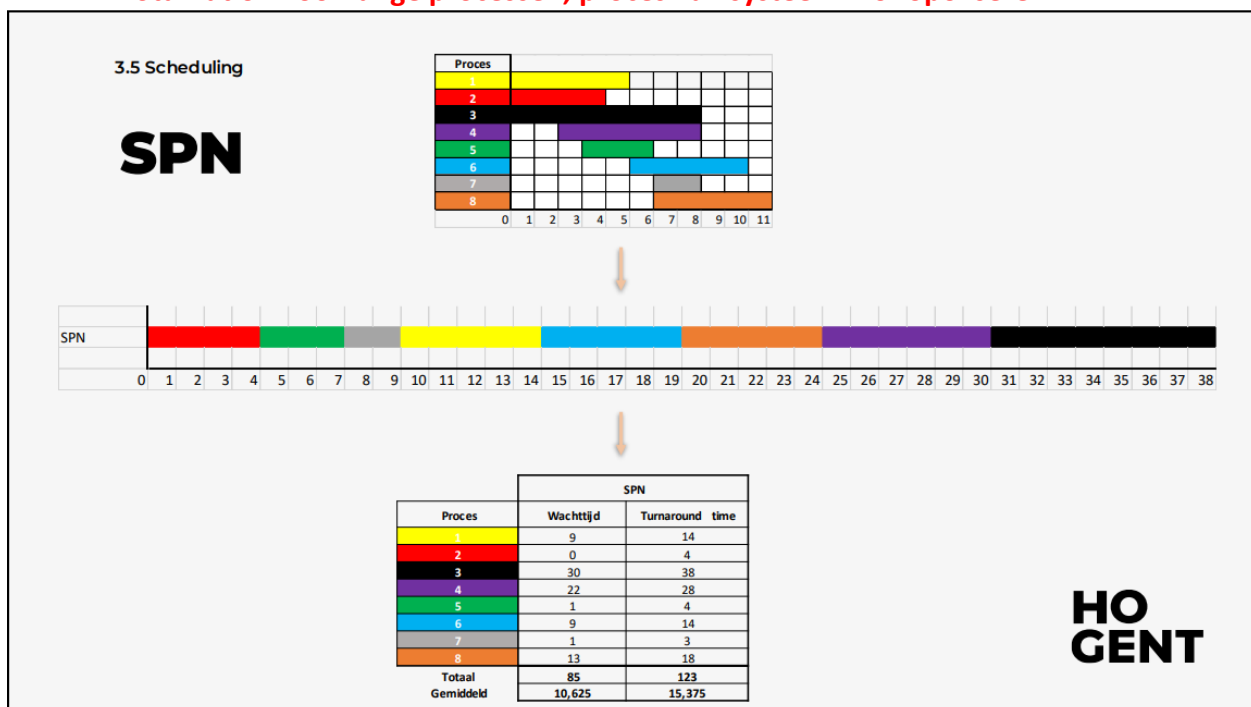
FCFS vervolg:

- Tabel rechts bevat voor de planner volgende informatie
- **Wachttijd:** tijd dat het proces in het systeem aanwezig was maar niet uitgevoerd werd
- **Turnaround time:** tijd dat het proces in het systeem aanwezig was vanaf aanbieden tot effectief uitgevoerd zijn (= wachttijd + uitvoeringstijd)

Proces	FCFS	
	Wachttijd	Turnaround time
1	0	5
2	5	9
3	9	17
4	15	21
5	20	23
6	21	26
7	25	27
8	27	32
Totaal	122	160
Gemiddeld	15,25	20

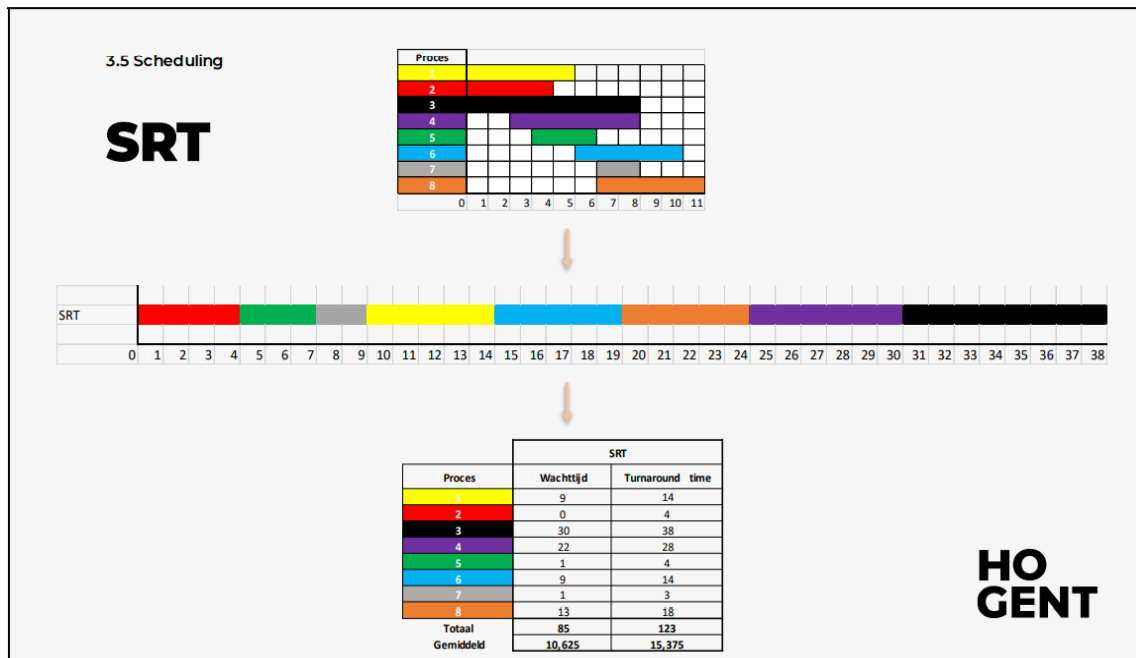
Shortest Process Next (SPN):

- Proces wordt volledig uitgevoerd, geen onderbrekingen
- Processen worden in wachtrij geplaatst indien nodig
- Volgende proces? Kortste proces uit de wachtrij
- **+ : Korte processen zijn snel uitgevoerd**
- **- : Starvation voor lange processen, proces kan systeem monopoliseren**



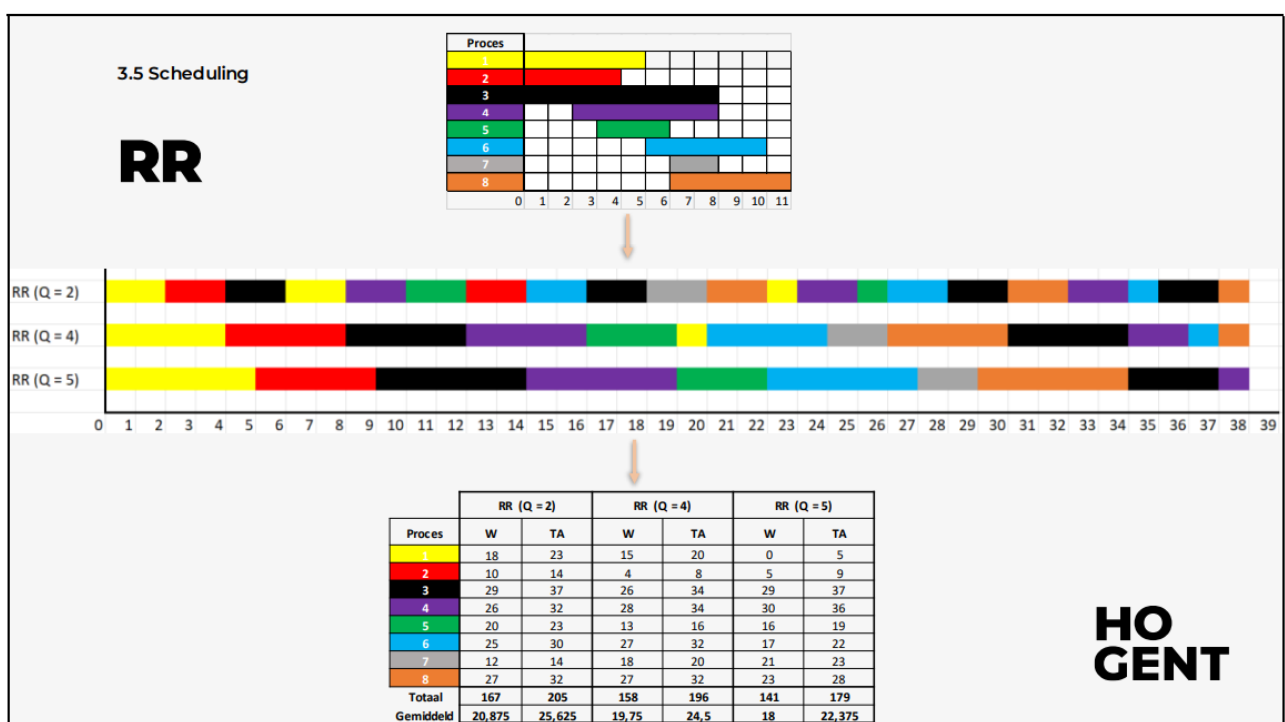
Shortest Remaining Time (SRT):

- Proces wordt onderbroken, indien er zich een beter (= korter) proces aandient
- Processen worden in wachtrij geplaatst indien nodig
- Volgende proces? Proces met kortste overgebleven uitvoeringstijd uit de wachtrij
- Indien beide processen even lang zijn worden de nieuwe achter de oude geplaatst
- + : Korte processen zijn snel uitgevoerd
- - : Starvation voor lange processen, overhead bij veel wisselen

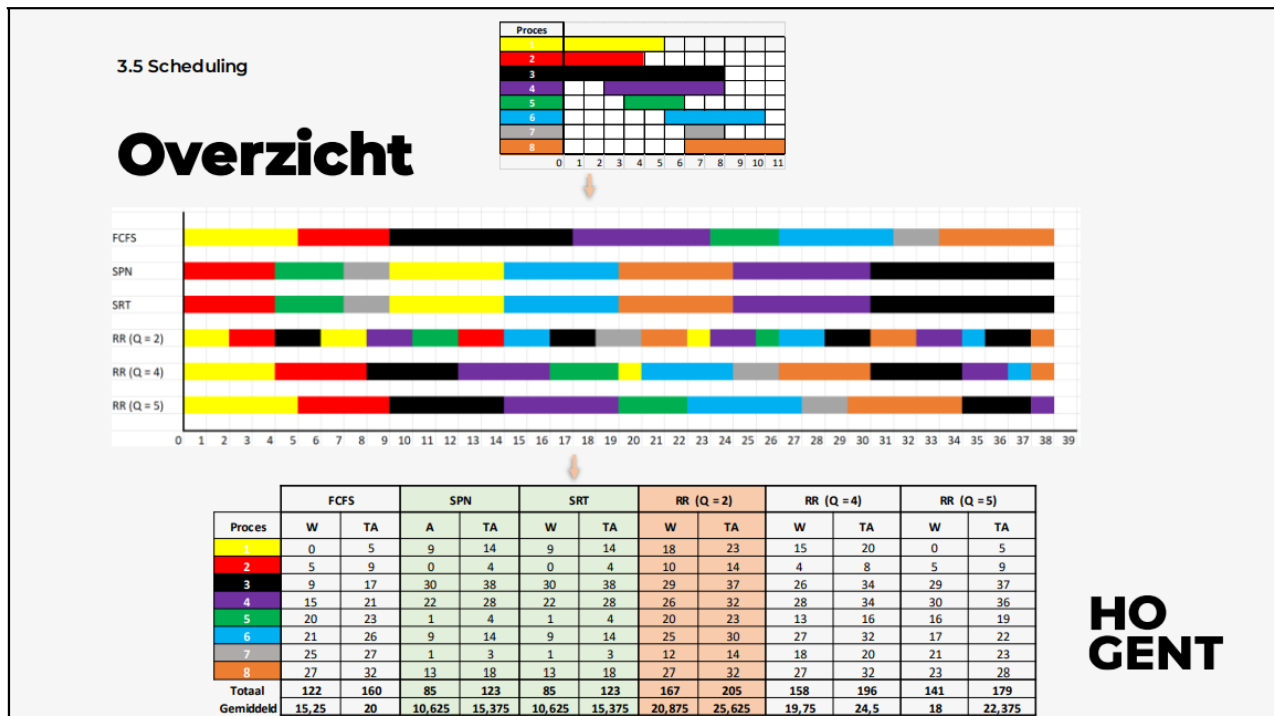


Round Robin (RR):

- Elk proces wordt beurtelings Q eenheden uitgevoerd
- Processen worden in wachtrij geplaatst indien nodig
- Volgende proces? Volgende proces uit wachtrij, laatst uitgevoerde wordt achteraan terug toegevoegd
- + : Eerlijk, geen starvation
- - : Waarde voor Q heel belangrijk, korte processen moeten soms lang wachten, overhead bij veel wisselen



Samenvatting:



Overzicht type planners:

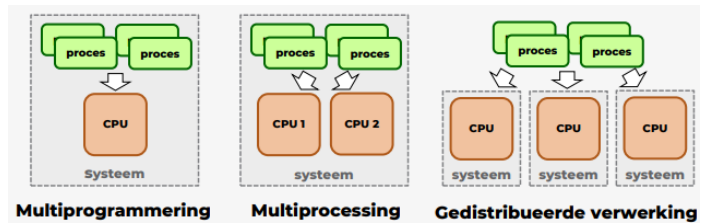
	FCFS	SPN	SRT	RR
Preemptive	Nee	Nee	Ja	Ja
Voordelen	<ul style="list-style-type: none"> Geen starvation Gemakkelijk 	Korte processen snel uitgevoerd	Korte processen snel uitgevoerd	<ul style="list-style-type: none"> Geen starvation Eerlijk
Nadelen	Korte processen moeten soms lang wachten	<ul style="list-style-type: none"> Starvation lange processen Monopolisatie systeem mogelijk 	<ul style="list-style-type: none"> Starvation lange processen Overhead bij vele wisselen 	<ul style="list-style-type: none"> Q waarde belangrijk Korte processen moeten soms lang wachten
Beste bij	Lange processen	Korte processen	Korte processen	Lange processen

4 Hoofdstuk 4: Concurrency

4.1 Wat is concurrency

Beheer van meerdere processen:

- **Multiprogrammering:** het beheer van meerdere processen in een systeem met 1 processor
- **Multiprocessing:** het beheer van meerdere processen in een systeem met meerdere processoren
- **Gedistribueerde verwerking:** het beheer van meerdere processen die worden uitgevoerd op een aantal verspreide (= gedistribueerde) computersystemen



Concurrency:

- Een **multiprocessor** is een systeem met 2 of meer CPU's
- Dergelijke systemen kunnen meerdere taken **gelijktijdig** uitvoeren = concurrency (parallele processen)
- Verhoogt productiviteit, maar zorgt ook voor uitdagingen:
 - Communicatie tussen processen
 - Delen van, en vechten om bronnen
 - Synchronisatie van meerdere procesactiviteiten
 - Verdelen van processortijd over processen ...

Concurrency treedt op in verschillende situaties:

- **Meerdere toepassingen:** dynamisch verdelen processortijd over aantal actieve toepassingen
- **Gestructureerde toepassing:** toepassingen geprogrammeerd als een verzameling gelijktijdige processen
- **Structuur van het besturingssysteem:** besturingssystemen geïmplementeerd als een verzameling processen

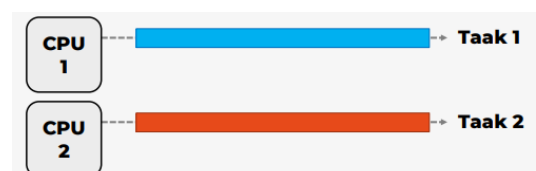
Concurrency - 1 CPU:

- Gelijktijdige uitvoering
- Toepassing boekt vooruitgang op meer dan één taak: (schijnbaar) gelijktijdig
- Bij systemen met één CPU: schakelen tussen verschillende taken tijdens uitvoering



Parallel execution (parallele uitvoering):

- Systemen met meer dan één CPU of CPU-kern
- Elke kern doet één taak
- Meer dan één taak van een toepassing worden gelijktijdig (parallel) uitgevoerd
- ≠ parallelisme (zie verder)
- Parallele uitvoering (EN: Parallel Execution) treedt op wanneer een computersysteem meer dan één CPU of CPU-kern heeft en tegelijkertijd vooruitgang boekt op meer dan één taak. Parallele uitvoering verwijst echter niet naar hetzelfde fenomeen als parallelisme. Parallele uitvoering wordt geïllustreerd in het diagram in deze slide.



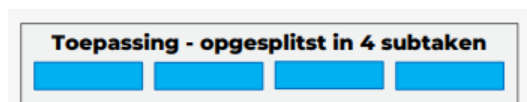
Parallel Concurrent Execution:

- Parallele gelijktijdige uitvoering
- Taken worden verdeeld over meerdere CPU's
- Binnen één CPU: schakelen tussen verschillende taken
- (schijnbaar) gelijktijdig - Taken op verschillende CPU's worden parallel uitgevoerd
- We kunnen ook de vorige 2 slides combineren, wat het principe is van parallelle gelijktijdige uitvoering (EN: parallel concurrent execution). Hierbij worden de verschillende taken verdeeld over meerdere CPU's. Taken die op dezelfde CPU worden uitgevoerd, worden gelijktijdig uitgevoerd, terwijl taken die op verschillende CPU's worden uitgevoerd parallel worden uitgevoerd.



Parallelism:

- Toepassing splitst zijn werk op in subtaken die parallel kunnen worden verwerkt
- Verwijst niet naar hetzelfde uitvoeringsmodel als parallelle gelijktijdige uitvoering
-
- Hoe parallelisme bereiken:
 - Meer dan 1 subtaak
 - Elke subtaak draait parallel op afzonderlijke CPU's of CPUcores
- **Kort: hoe ze effectief uitgevoerd worden**



4.2 Wederzijdse uitsluiting (mutual exclusion)

Wederzijdse uitsluiting:

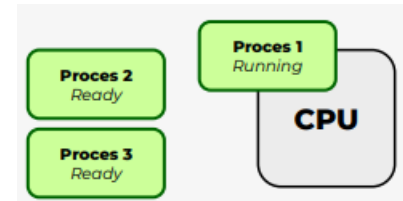
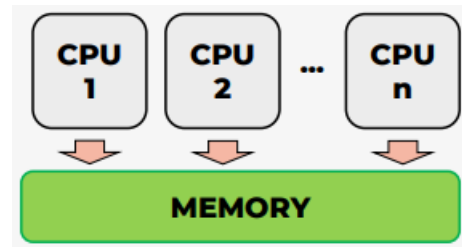
- Soms worden bronnen gedeeld over meerdere processen
- De code (instructies) die gebruikt wordt voor het aanspreken van gedeelde bronnen noemen we een **kritieke sectie**
- Het is belangrijk dat er op elk moment maar maximum 1 proces in een kritieke sectie zit
 - Nood aan **wederzijdse uitsluiting** (mutual exclusion)
 - Belangrijk probleem binnen informatica!

Wederzijdse uitsluiting voorbeeld:

- Stel: je hebt een **globale variabele getal** (geheel getal)
- 2 processen willen deze variabele aanpassen, via volgende instructies:
 - 1 Lees de huidige waarde van de variabele getal vanuit het geheugen
 - 2 Verhoog deze waarde met 1
 - 3 Schrijf de nieuwe waarde van getal weg naar het geheugen
- Als beide processen om beurt de instructies uitvoeren is er geen probleem, maar wat als het eerste proces onderbroken wordt na uitvoeren van de eerste instructie?
- De variabele getal zal, afhankelijk van de volgorde van uitvoering, verhoogd zijn met 1 of 2
- Een oplossing is om de 3 instructies te groeperen als **kritieke sectie**, en hiervoor wederzijdse uitsluiting af te dwingen

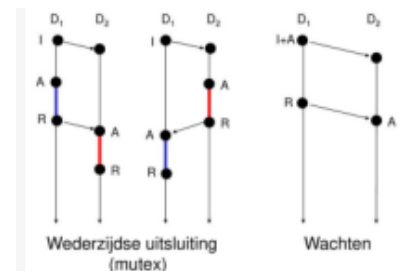
Wederzijdse uitsluiting bij multiprocessing:

- Niet alleen processen, maar ook **activiteiten binnen één proces** kunnen parallel worden uitgevoerd
- We zullen processen bespreken, maar de principes gelden eveneens voor activiteiten binnen één proces
- De moeilijkheden ontstaan wanneer de processen het **gemeenschappelijke geheugen** aanspreken
- Ook in een systeem met maar 1 CPU zijn gelijklopende processen mogelijk
- Processen kunnen niet tegelijkertijd worden uitgevoerd, maar ze kunnen wel op hetzelfde moment proberen de besturing van de CPU te krijgen
- Wanneer twee van zulke processen het gemeenschappelijk geheugen aanspreken, kunnen er nog steeds problemen ontstaan



Wederzijdse uitsluiting afdwingen:

- Niet evident om wederzijdse uitsluiting af te dwingen!
- Mogelijke oplossingen: algoritme van **Dekker** (2 processen) en wederzijds uitsluitingsalgoritme van **Peterson** (2 of meer processen)
- Alternatieven: wederzijds uitsluiting afdwingen via **semaforen** (Dijkstra) of **monitoren**



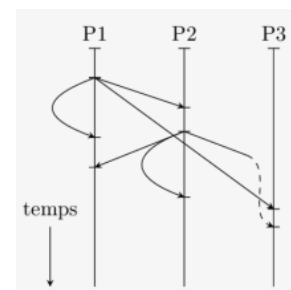
Meer dan toegang tot gedeeld geheugen:

- In vorige slides: wederzijdse uitsluiting om toegang te regelen van CPU naar gemeenschappelijk geheugen
- Andere vormen van wederzijdse uitsluiting:
 - Toegang tot **bestanden** en records (bv: meerdere processen willen gelijktijdig schrijven naar zelfde bestand)
 - Toegang tot **hardware** bronnen (bv: 2 processen willen gelijktijdig iets sturen naar printer) ...
- Het is de **taak** van het **besturingssysteem** om in deze situaties wederzijdse uitsluiting te garanderen

4.3 Synchronisatie

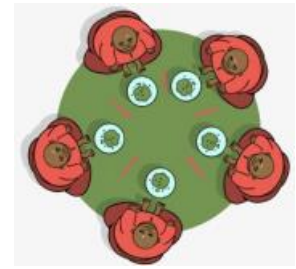
Wat is synchronisatie?

- **Synchronisatie** is het proces, of het resultaat van iets **gelijktijdig maken**
 - Ontstaan: 19e eeuw, treinen werden zo snel dat een verschil in lokale tijd opviel
 - Synchronisatie van klokken ook nodig om botsingen te voorkomen op enkelspoor
- Hier: het opleggen van een **dwingende volgorde** aan **events** die door concurrente, asynchrone processen worden uitgevoerd.
- Wij moeten garanderen dat processen in een bepaalde volgorde verlopen



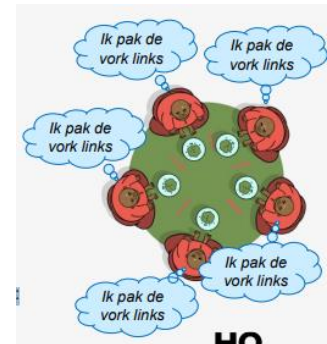
Het filosofen probleem:

- 5 Filosofen aan ronde tafel
- 5 Vorken op tafel, tussen elke filosoof
- Filosoof kan **denken** of **eten** (niet tegelijkertijd)
- Om te eten heeft een filosoof **2 vorken** nodig ($l + r$)
- Filosoof kan **vork oppakken** als die op tafel ligt
- Filosoof moet de vorken **één voor één** oppakken
- In welke **volgorde** moeten filosofen de vorken oppakken en hoe lang mag een filosoof eten, zodat geen enkele filosoof zal **verhongeren**?



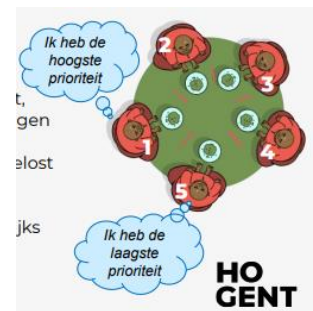
Een eerste poging:

- Stel: elke filosoof wil meteen eten, pakt eerst vork aan linkerkant op van tafel
- Elke filosoof heeft 1 vork vast, andere vork is reeds gepakt door filosoof rechts
- Geen enkele filosoof wil vork neerleggen, maar blijft wachten tot andere vork vrij is
- Filosofen blijven eeuwig op elkaar wachten: er is een deadlock



Een tweede poging:

- Stel: elke filosoof heeft **nummer**, een lager nummer heeft **voorrang**
- Als buur met lager nummer een vork heeft, moet filosoof wachten en vork(en) neerleggen
- Het probleem van een deadlock is nu opgelost
- Het systeem is echter **niet eerlijk**
- Filosoof met hoogste nummer kan mogelijks verhongeren: **starvation**



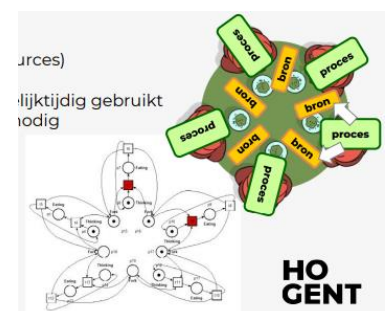
Een eerlijke poging:

- Filosoof mag **niet oneindig** blijven eten, moet na maximumtijd even stoppen om te filosoferen
- Wanneer filosoof **stopt** met eten krijgt hij **hoogste nummer + 1**
- Elke filosoof zal uiteindelijk kunnen eten: **geen** kans op **deadlock** of **starvation**
- Het systeem is **eerlijk**: elke filosoof heeft om beurt laagste nummer



Link naar processen:

- Filosofen = **taken** of processen Vorken = gedeelde **bronnen** (resources)
- Bron kan niet door 2 processen gelijktijdig gebruikt worden, **wederzijdse uitsluiting** nodig



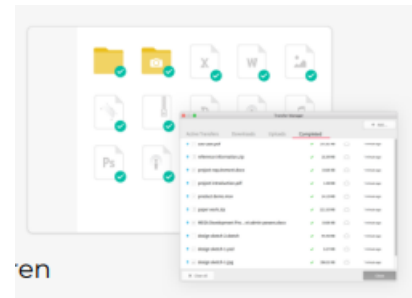
Bestandssynchronisatie:

- 2 of meer identieke **kopieën** van bestand
 - Wijzigingen in één kopie zichtbaar in andere
 - Synchronisatie aan hand van vaste **regels**
- Vroeger: vaak **manueel** (bv. USB-stick) Nu: **automatisch** over netwerk / internet
 - Veel programma's beschikbaar, bv. DropBox, OneDrive, Google Drive, ...
 - Vaak met centrale kopie in de cloud
 - Ook gebruikt voor maken van backups



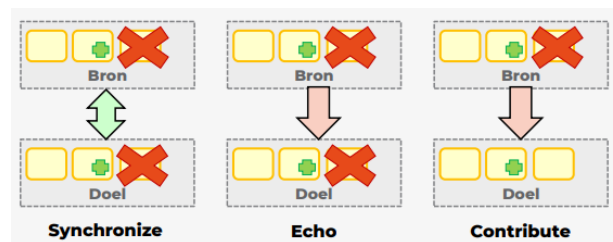
Bestandssynchronisatie: werking

- **Selectie** van mappen/bestanden
- Vaak: bron → doel
 - 1 bron (source)
 - 1 of meerdere doelen (lokaal of remote)
- Mogelijkheid om **regels** toe te voegen:
 - Bepaalde mappen uitsluiten?
 - Filter op type(s) bestanden?
 - Verborgenen bestanden?
- Synchronisatie kan meteen (**continu**) gebeuren of **periodiek** (op vastgelegde tijdstippen)
 - Online samenwerken ⇔ back-up maken



Bestandssynchronisatie: soorten:

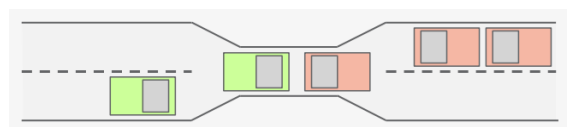
- Bij **synchronize** wordt de inhoud van de bron- en doelmap gelijk gehouden. Elke wijziging in een kopie zal dus altijd zichtbaar zijn in alle andere kopieën
- Bij **echo** worden nieuwe en gewijzigde bestanden gekopieerd van de bronmap naar de doelmap. Hernoemde en verwijderde bestanden in de bronmap worden in doelmap ook hernoemd of verwijderd. De synchronisatie gebeurt echter maar in één richting.
- De werking van **contribute** is gelijkaardig aan deze van **echo**, maar er worden geen bestanden verwijderd uit de doelmap als dit in de bronmap wel gebeurd is.



4.4 Deadlocks

Wat is een deadlock?

- Deadlocks bestaan er in vele maten en vormen. De figuur in deze slide toont bijvoorbeeld een eenvoudige situatie uit het verkeer waarbij een deadlock kan optreden. Een stuk weg bestaat uit 2 rijstroken, maar onderweg is er een versmalling, waardoor auto's in beide richtingen even over één enkele rijstrook moeten. Voor de eenvoud gaan we er hier van uit dat auto's enkel vooruit kunnen rijden. De auto's in het groen rijden van links naar rechts, de auto's in het rood van rechts naar links. Indien er veel verkeer is kan dit voor lastige situaties zorgen.
- Indien de auto's elkaar mooi afwisselen, en er dus telkens één rode en nadien één groene auto door de wegversmalling rijdt is er geen probleem, maar als een rode en groene auto

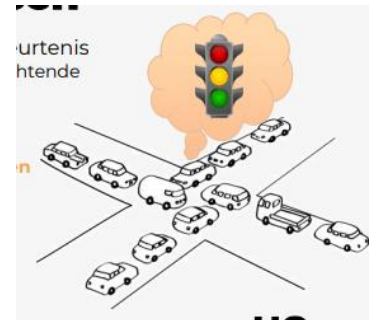


Deadlocks in de echte wereld: gridlock:

Deadlocks komen trouwens niet enkel voor binnen de computerwereld, maar ook binnen de echte wereld, bijvoorbeeld in het verkeer. Een mooi voorbeeld hiervan is de gridlock (in het Nederlands ook verkeersinfarct genoemd). Een gridlock is een situatie waarbij het verkeer zodanig gehinderd is door uitzonderlijke omstandigheden (file, weersomstandigheden, ...) dat voertuigen nog nauwelijks vooruit kunnen komen, meestal veroorzaakt doordat weggebruikers elkaar indirect blokkeren (kop van file sluit aan bij de staart).

Deadlocks bij processen:

- Twee of meer processen wachten op gebeurtenis
 - Gebeurtenis kan enkel door één van de wachtende processen worden veroorzaakt (gebeurt door een ander proces)
- Hoe deadlock behandelen?
 - Gebruik protocol om deadlock te **voorkomen** OF
 - Laat deadlocks toe, maar **detecteer** deadlock-situatie en **herstel** deze



Deadlocks behandelen:

- Deadlock **voorkomen**
 - OS beperkt gebruik van gemeenschappelijke bronnen
 - Doel: deadlock onmogelijk maken
- Deadlock **vermijden**
 - Alle aanvragen voor bronnen in detail onderzoeken
 - Kans op deadlock verminderen
- Deadlock **signaleren**
 - Hoe kan het OS weten dat er een deadlock is?
 - Processen zitten in wacht-toestand, is wachten permanent?
- Deadlock **herstellen**
 - Hoe kan OS een deadlock-situatie oplossen?



Voorwaarden deadlock:

- Deadlock kan **enkel** ontstaan bij volgende **voorwaarden**:
- Bron met **wederzijdse uitsluiting** (Mutual Exclusion)
- Proces zal bron **bezet houden** en **wachten** (Hold and Wait)
- Er is **geen voortijdig ontnemen** (No preemption)
- Processen **wachten** in een **kring** (Circular wait)

Deadlock preventie:

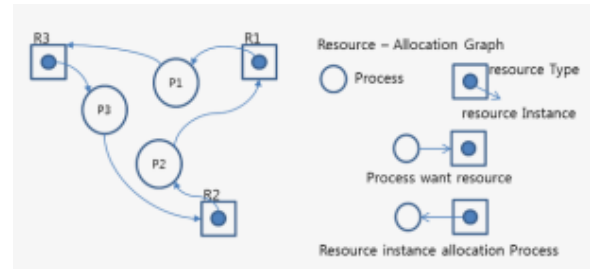
- Deadlock **voorkomen**: zorgen dat minstens één voorwaarde nooit optreedt?
- Proces zal bron bezet houden → **vrijgeven** indien nodig
- Er is voortijdig ontnemen → eerlijk? **volgorde** volgens **schema**
- Processen wachten in een kring

Deadlock vermijden:

- Deadlock **voorkomen**: geen kans op deadlock
- Deadlock **vermijden**:
- Kans op deadlock bijna **onmogelijk** maken, maar kan nog optreden
- Aanvragen die tot deadlock kunnen leiden **weigeren**
 - Omgeving blijft in veilige status
 - Moeilijk (onmogelijk?) om te implementeren: Hoe aanvragen beoordelen?

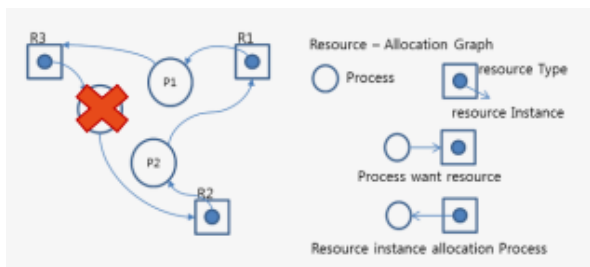
Deadlock signaleren:

- Hoe weet het OS dat er een deadlock is?
- Processen in BLOCKED toestand, is dit permanent?
- Deadlock opsporen via **Resource Allocation Graph**
- Deadlock = **cycclus** in RAG
- Algoritmes om cyclus in graaf op te sporen



Deadlock herstellen (Potentiële examenvraag):

- Wat doen als er een deadlock gevonden is?
- proces stoppen via **KILL** → bronnen vrijgeven ten koste van proces
- Doe **rollback** op proces
 - Deel van werk ongedaan maken en bronnen vrijgeven
 - Terugkeren naar starttoestand of **checkpoint**



Nadeel kill:

Alles kwijt

Voordeel kill:

Alles is direct weg

Voordeel rollback:

Je bent niet alles kwijt en keert terug naar een vorige 'snapshot'

Nadeel rollback:

Je moet echt je 'snapshots' hebben om naar terug te rollen

4.5 Quiz:

Met welke techniek(en) kan wederzijdse uitsluiting worden afgedwongen?

- Semaforen (poortwachters, integers, booleans)
- Monitoren (poortwachters, integers, booleans)

Welke manier van omgaan met deadlocks wordt hier beschreven: "Het besturingssysteem onderwerpt elke aanvraag van systeembronnen door processen aan een rigoreuze inspectie: als het risico op een deadlock niet te groot wordt, staat het systeem de aanvraag toe."

Deadlocks vermijden

Welke nodige voorwaarden voor een deadlock kunnen door het besturingssysteem gestuurd worden?

- Processen houden bronnen bezet tijdens het wachten
- Processen worden niet voortijdig beëindigd

Vermijden van een deadlock:

- Aanvragen tot systeembronnen worden gescreend en enkel toegestaan als ze veilig geacht worden.
- Het besturingssysteem scant voor processen die geblokkeerd staan.
- Via een RAG wordt nagegaan of processen in een cirkel op elkaar wachten.
- Processen worden eventueel beëindigd met 'kill' of 'rollback'.

RAG:

Resource Allocation Graph

Hoe heet het deel van de code waar de instructies uitgevoerd worden die gedeelde bronnen aanspreken? Hierbinnen geldt wederzijds uitsluiting.

Kritieke sectie

Verbind:

The diagrams show the following configurations:

- A. Multiprogramming:** A single system box containing two CPUs (CPU 1, CPU 2) and two processes (proces) running on them.
- B. Multiprocessing:** A single system box containing one CPU and two processes (proces) running on it.
- C. Distributed processing:** Three separate system boxes, each containing one CPU and one process (proces).

1^e 2 omdraaien, dan heb je het juiste antwoord

Welk begrip wordt hiermee omschreven: "Verschillende processen worden (fysiek) gelijktijdig uitgevoerd op verschillende CPUs"?

Parallele uitvoering

Wederzijdse uitsluiting is enkel nodig bij het gebruik van multiprocessing.

Vals

Wederzijdse uitsluiting is enkel vereist als er gemeenschappelijke systeembronnen zijn.

Waar

Het filosofenprobleem zou niet optreden als er geen wederzijdse uitsluiting nodig was.

Waar

Welk fenomeen treedt er op als we bij het filosofenprobleem elke filosoof een vast rangnummer geven, dat hun toegang tot het bestek bepaalt? (1 woord)

Concurrency

5 Hoofdstuk 5: File Systems

5.1 Persistente opslag

Persistente opslag:

- Processen hebben nood aan persistente opslag:
- Data bewaren na afsluiten van het proces.
- Grote hoeveelheden data opslaan.
- Data delen tussen processen.

Hard disk drive (HDD):

- Magnetisch opslagmedium
- Bevat draaiende schijven
- Heeft daardoor tijd (seek time) nodig om zich te positioneren
- Grote capaciteit
- Lage kost



Solid-state drive:

- Bevat geen bewegende onderdelen
- Opslag gebeurt in integrated circuits (ICs)
- Sneller dan HDD
- Hogere kost per GB



CD/DVD:

- Optisch opslagmedium.
- Beperkte capaciteit en performantie.
- Vaak read-only.
- Voornamelijk geschikt voor distributie of backup.



USB stick:

- Extern opslagmedium, aangesloten via USB
- Opslag gebeurt in ICs, net zoals SSD
- Beperkte capaciteit en performantie



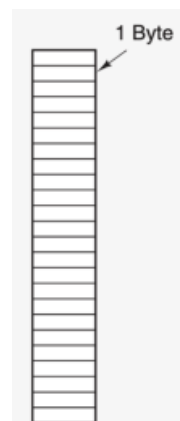
5.2 Files

Wat is een file?

- Een fysiek opslagmedium is onderverdeeld in blokken
- Een **file** (bestand) groepeer data uit één of meer blokken
- Een file is dus een **abstracte** eenheid die de complexiteit van de fysieke opslag verbergt
- De implementatie van files gebeurt door een **file system** (bestandssysteem)

Voorstelling van een file:

- Een file wordt voorgesteld als een opeenvolging van bytes.
- Dit is echter een abstractie: in realiteit kunnen deze bytes verspreid zijn over het opslagmedium.



Eigenschappen (wat is een file):

- Naam, eventueel gevolgd door een extensie
 - In Ubuntu is dit puur informatief. .txt en .docx is glad hetzelfde
- Huidige en maximale grootte
- Toegangsrechten
 - Belangrijk = permissies in Linux
- Datum van aanmaak
- Datum van laatste aanpassing
- Verwijzing naar de datablokken

Soorten (weten dat ze bestaan, details niet kennen):

- Op UNIX geldt de regel “**everything is a file**”:
- **Directories** bevatten andere bestanden of mappen
- **Links** maken een bestand op meerdere plaatsen in het bestandssysteem zichtbaar
- **Speciale bestanden** zijn gekoppeld aan hardware
- **Sockets** zorgen voor netwerkcommunicatie
- **Pipes** (FIFO) verbinden de output van een proces met de input van een ander proces
- De output van **ls -l** toont het type van elk bestand:
- De output van **ls -F** toont het type via een suffix:

Suffix	Type
/	map
*	uitvoerbaar bestand
@	link
=	socket
	named pipe

```
etc $ ls -l
total 912
-rw-r--r-- 1 root wheel 515 Jan 1 2020 afpovertcp.cfg
lwxr-xr-x 1 root wheel 15 Jan 1 2020 aliases -> postfix/aliases
-rw-r----- 1 root wheel 16384 Jan 1 2020 aliases.db
dwxr-xr-x 9 root wheel 288 Jan 1 2020 apache2
dwxr-xr-x 18 root wheel 576 Jan 1 2020 asl
```

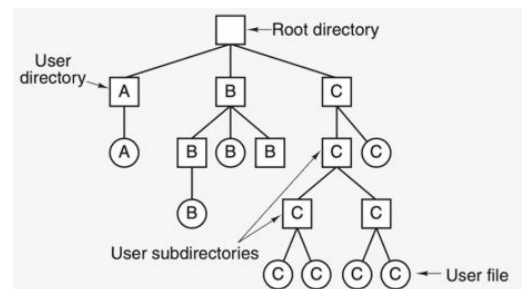
Bytes uitlezen:

- Ook de manier waarop we bestanden uitlezen kan verschillen:
- **Sequentieel** Bytes moeten in volgorde uitgelezen worden
- **Random access** Bytes kunnen in een willekeurige volgorde uitgelezen worden

5.3 Directories

Wat is een directory?

- Een **directory** (map) groepeer bestanden
- Een directory kan ook andere directories bevatten
- Dit creëert een **hiërarchische** structuur
- De implementatie van directories gebeurt door een file system
- Dit kan bv. d.m.v. een bestand
- /boot files kan je niet zomaar aan



Padnamen:

- Het **absoluut pad** naar een bestand of map start bij de root directory en doorloopt de hiërarchie,
 - bv. /home/hogent/os (Linux of Mac) C:\Users\hogent\os (Windows)
- Een **relatief pad** start vanuit een bestaande directory en kan de speciale verwijzingen . en .. gebruiken:
 - bv. ../hogent/os (Linux of Mac) ..\hogent\os (Windows)

5.4 File systems

Wat is een file system?

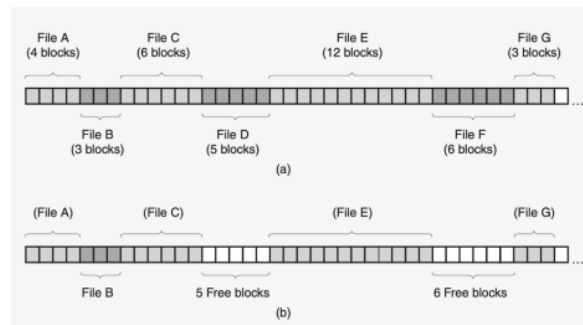
- Een **file system** (bestandssysteem) is een onderdeel van het besturingssysteem
- File systems beheren de fysieke opslagruimte en implementeren files en directories
- Een besturingssysteem kan meerdere file systems (tegelijk) ondersteunen

Implementatie van files:

- Files kunnen op verschillende manieren uitgewerkt worden:
 - Contiguous storage
 - Linked lists
 - File allocation table (FAT)
 - Index nodes (inodes)
- De volgende slides overlopen deze manieren.

Contiguous storage:

- Bij contiguous storage (samenhangende opslag) wordt elk bestand in één of meer aansluitende blokken opgeslagen. Dit is een eenvoudige methode met een goede leessnelheid en met ondersteuning voor random access. Er zijn echter ook belangrijke nadelen:
- Als een bestand groeit, dan moet het mogelijk verplaatst worden naar een grotere vrije ruimte.
- Als een bestand wordt verwijderd, en de vrije ruimte wordt ingenomen door een kleiner bestand, dan ontstaat er fragmentatie (= meer en meer kleine ruimtes die niet opgevuld geraken BB CAAA DD (je doet BB weg en wilt EE erbij plaatsten die 3 blokken innemen, dat gaat niet want de blokken zijn niet na elkaar beschikbaar). Een systeem dat deze techniek gebruikt moet dus regelmatig gedefragmenteerd worden om de kleine vrije ruimtes terug samen te voegen.



Voor- en nadelen contiguous storage:



Eenvoudig



Goede leessnelheid



Ondersteunt random access, want je weet het adres



snel (want hij loopt er in één keer door)



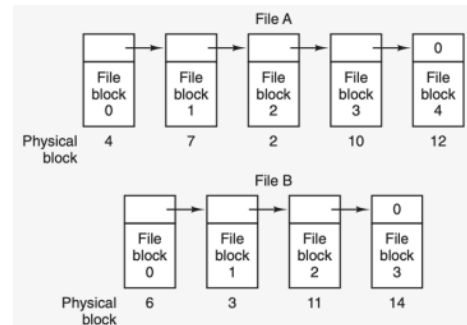
Bestanden die groeien moeten worden verplaatst



Leidt tot fragmentatie (groeiproblemen)

Linked lists:

- Bij linked lists (gelinkte lijsten) hoeven de datablokken van een bestand niet aan te sluiten. Elk blok bevat namelijk een verwijzing (link) naar het volgende blok, dat zich eender waar mag bevinden.
- Deze techniek lost de problematiek rond fragmentatie op, maar heeft een slechte performantie. Doordat de datablokken verspreid staan, daalt de leesperformantie, i.h.b. op opslagmedia met een seek time. Bovendien ondersteunt deze techniek geen random access, omdat de blokken in volgorde moeten worden uitgelezen.
- Elke blok verwijst naar een andere blok

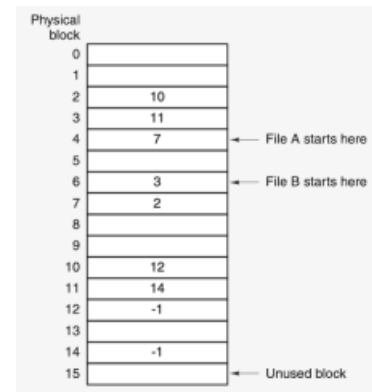


Voor- en nadelen linked lists:

- 👍 Geen fragmentatie
- 👍 Je mag ze overal plaatsen
- 🗨 Slechte leessnelheid
- 🗨 Ondersteunt geen random access
- 🗨 Fragmentatie heb je ook niet meer

File allocation table (FAT):

- Een **file allocation table (FAT)** (toewijzingstabel) is een verbetering van de linked list techniek. De datablokken van elk bestand vormen nog steeds een linked list, maar alle verwijzingen tussen de blokken worden samengebracht in één tabel.
- Door deze tabel in het RAM te bewaren, is het mogelijk snel een lijst te vinden van alle blokken van een bestand, zonder de blokken zelf te moeten uitlezen. Dit verhoogt de **leessnelheid** en maakt **random access** mogelijk.
- Het nadeel van deze techniek is dat de FAT een **hoog RAM-verbruik** kan hebben. Neem bvb. een harde schijf van 1TB, onderverdeeld in blokken van 4KB. Deze schijf heeft dan $1\text{TB} / 4\text{KB} = 256\text{M}$ blokken. Indien elke verwijzing 32 bits in beslag neemt, dan is de grootte van de FAT $256\text{M} \times 4\text{B} = 1\text{GB}$.

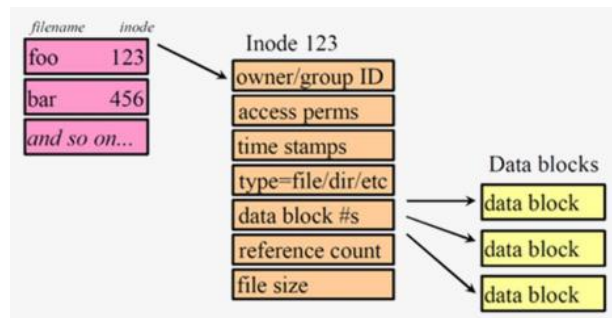


Voor- en nadelen table (FAT):

- 👍 Betere leessnelheid dan linked lists
- 👍 Ondersteunt random access
- 🗨 FAT kan erg veel RAM in beslag nemen

Een index node (inode);

- **Index nodes (inodes)** is een datastructuur die zowel de metadata van een bestand als verwijzingen naar de datablokken bevat. Een bestandssysteem dat inodes gebruikt, hoeft enkel de inodes van de geopende bestanden in het RAM te bewaren. Deze techniek combineert dus een **goede leessnelheid** met een **beperkt RAMverbruik**.
- Het verwijst naar adressen(data) in datablokken en verwijst ook naar de **metadata** daarvan (naam, locatie ...)



Voor- en nadelen inodes:

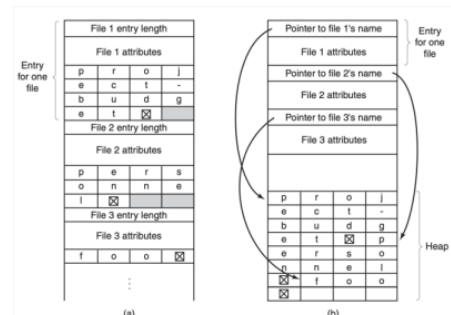
- 👍 Goede leessnelheid
- 👍 Ondersteunt random access
- 👍 Beperkt RAM-verbruik

Implementatie van directories:

- Een directory kan opgeslagen worden in een **bestand**
- Dit bestand bevat één **entry** per file of subdirectory
- Een entry bevat o.a. een verwijzing naar het eerste datablok, of de inode, van de file of subdirectory

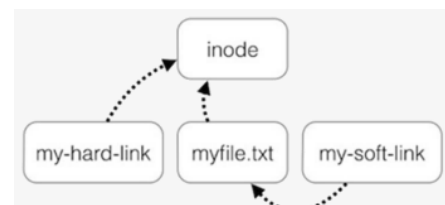
Implementatie van directories:

- Een belangrijke keuze die moet worden gemaakt is hoe/waar de naam van elke file of subdirectory wordt opgeslagen. In Figuur (a) wordt de naam opgeslagen als deel van de entry. In Figuur (b) worden de namen apart opgeslagen in een heap.
- Versie (a) heeft als nadeel dat directory entries een variabele grootte hebben, en er dus fragmentatie kan ontstaan binnen het directory-bestand. Versie (b) heeft dan weer als nadeel dat er een heap moet beheerd worden



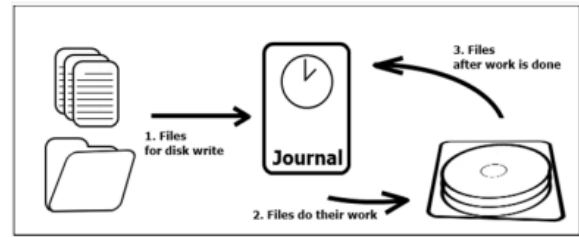
Implementatie van links:

- De meeste file systems ondersteunen **links**
- Bij een **hard link** worden datablokken of inodes gedeeld
- Een **soft link** heeft een eigen datablok of inode
- In dit voorbeeld delen **my-hard-link** en **myfile.txt** dezelfde inode. **my-soft-link** verwijst rechtstreeks naar het bestand **myfile.txt**.



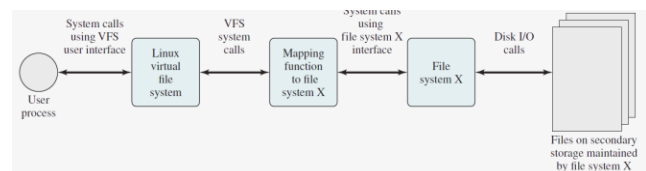
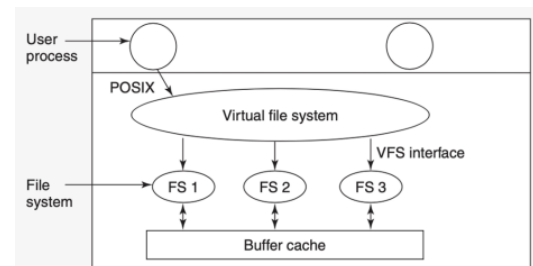
Journaling:

- Een crash tijdens een schrijfbewerking kan leiden tot corrupte of verloren data
- Het bijhouden van een journal (logboek van bewerkingen) kan hier tegen beschermen
- Wanneer het bestandssysteem of besturingssysteem crasht tijdens een schrijfbewerking, dan kan er data corrupt worden of verloren gaan. Een schrijfbewerking kan namelijk uit verschillende stappen bestaan, die allemaal moeten uitgevoerd worden. Zo komt het verwijderen van een file neer op:
 - 1. Verwijder de directory entry voor deze file.
 - 2. Verwijder de inode voor deze file.
 - 3. Markeer de datablokken van deze file als vrije ruimte.
- Het onderbreken van deze stappen kan het bestandssysteem in een ongeldige toestand brengen.
- Door het bijhouden van een journal (logboek van bewerkingen) kan het bestandssysteem zichzelf herstellen na een crash.
- waarom zou je dit niet op een USB-stick doen:
 - het maakt het trage



Virtual file systems:

- Op **Windows** krijgt elk bestandssysteem een **drive letter** toegewezen (C, D, ...)
- Linux en Mac hebben een **virtual file system** dat alle bestandssystemen presenteert als één hiërarchische structuur
- Het inladen van een bestandssysteem binnen de VFS-hiërarchie heet een **mount** bewerking
- Het uitladen van een bestandssysteem heet een **unmount**
- Bij het mounten geef je aan in welke map van de VFS-hiërarchie je de root directory van het bestandssysteem wil inladen



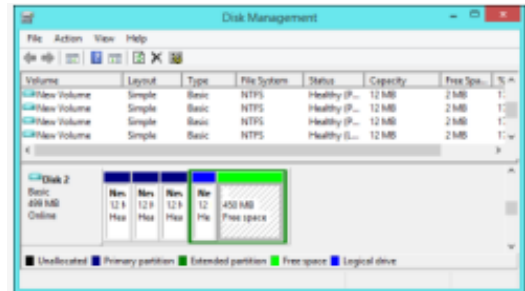
5.5 Partities

Partities:

- Een fysiek opslagmedium kan zijn capaciteit onderverdelen in **partities**
- Elke partitie heeft een eigen bestandssysteem
- Het opslagmedium voorziet dan ook een **partitietabel**, bvb:
 - Master Boot Record (MBR)
 - GUID Partition Table (GPT)
- Deze tabel bevat informatie over de partities en hun bestandssystemen

MBR vs GPT (GPT is opvolger):

- MBR: 1983 (DOS 2.0), belangrijke beperkingen:
 - Maximum grootte schijf: 2TB
 - Maximum 4 primaire partities
 - Workaround: Extended partitie maken die logische partities bevat
- GPT (1998): opvolger voor MBR – Elke partitie heeft (unieke) GUID
 - Ondersteuning voor schijven > 2TB
 - Theoretisch onbeperkt aantal partities (Windows max. 128)
 - Boot support voor Windows enkel op 64-bit systemen met UEFI.
 - Boot support voor Linux ook op BIOS firmware.

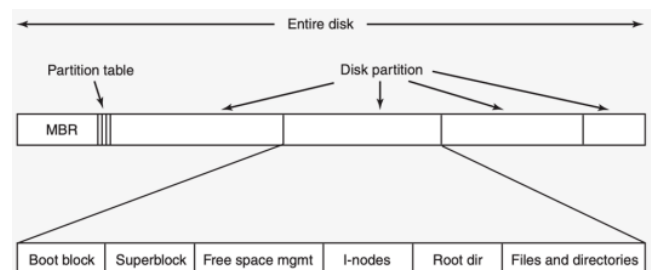


Partitietabel:

- Partitie:
 - Je harde schijf opdelen in stukken en je hebt een partitietabel nodig om te weten wat wat doet.
- boot partitie:
 - boot in een andere partitie zetten dan de main file system zodat hij niet per ongeluk overschreden kan worden
 - Als je *pc opstart* kan je makkelijk de boot partitie vinden om je OS op te starten
- Swappartitie:
 - soms is er *niet genoeg* RAM voor alle processen, en worden sommige tijdelijk opgeslaan in de swappartitie om later uit te voeren
 - zie [Suspending / swap van processen](#6-states)

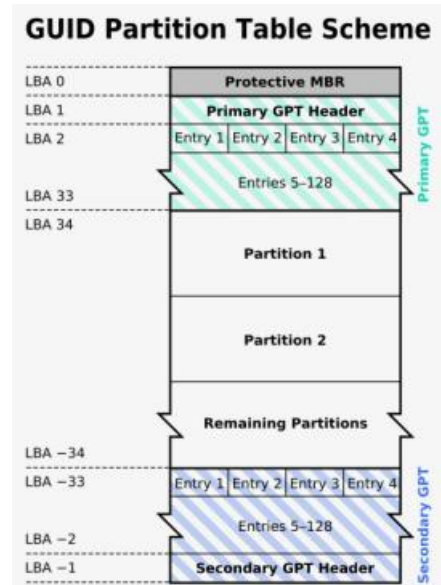
MBR (Master Boot Record):

- Dit voorbeeld toont een schijf die onderverdeeld is in partities. Aan het begin van de schijf zie je de Master Boot Record met de partitietabel. Daarna volgen de verschillende partities.
- Een Master Boot Record bevat naast de partitietabel ook een zogenaamde boot **loader**, die verantwoordelijk is voor het opstarten van het systeem. Deze **boot** loader gaat op zoek naar een partitie met een besturingssysteem en schakelt dan door naar de code in het **boot block** van deze partitie.



GPT (GUID Partition Table):

- Dit voorbeeld maakt gebruik van een GUID Partition Table (GPT). Deze GPT is onderdeel van de UEFI standaard, wat het verouderde BIOS vervangt. GPT is achterwaarts compatibel met MBR en kan ook worden gebruikt in combinatie met een BIOS. UEFI is dus niet verplicht, behalve als je Windows wil booten vanop een schijf met GPT.
- GPT plaatst de boot loader op een speciale **EFI System Partition**. Uiteraard kan er via de achterwaartse compatibiliteit met MBR nog steeds gebruik worden gemaakt van boot blocks.
- Waarom zou je de footer en header dubbel doen? Voor **fouttolerantie** als je niet weet waar je partities staan, ben je waardeloos.



5.6 Booten

Booten:

- Booten = opstarten van de computer en inladen operating system (Windows, Linux ...)
- Nood aan apart programma opgeslagen in een bootable partitie = **bootloader**
 - Een stuk software dat in je partities zit dat zorgt dat je OS wordt ingeladen
- Instellen volgorde bootable partities in de **BIOS**
- Bootloader is meestal OS afhankelijk
 - Linux: grub, lilo, rEFInd, ...
 - Mac: BootX
 - Windows: Windows Boot manager

Bootloader:

- Werking bootloader:
 - 1. Uitpakken gecomprimeerde bestanden op bootpartitie
 - 2. Inladen kernel in het geheugen
 - 3. Inladen root file system in het geheugen
 - 4. Controle doorgeven aan kernel om OS verder in te laden en te starten
- Elke kernel update OS => updaten files bootpartitie Zet je computer dus nooit uit tijdens OS update!
 - Kernel = basis van je systeem, I/O-controller zorgt voor cachestructuren ...
- Corrupte bootpartitie en/of files = onmogelijk om OS te laden

Multi-boot:

- Meerdere OS op eenzelfde computer
 - OS in juist volgorde installeren want voorgaande bootloader wordt overschreven
 - B.v. bij Linux en Windows samen
 - Eerst Windows installeren
 - Dan Linux installeren
 - Linux bootloader laten doorverwijzen Windows Boot Manager
- Chain loaden bootloaders

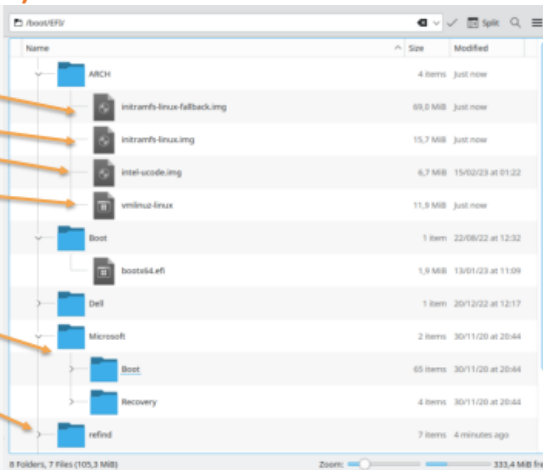
Voorbeeld boot loader (Multi-boot):

```
rEFInd Boot Loader
menuentry "Arch Linux" {
volume "Arch Linux"
loader /EFI/ARCH/vmlinuz-linux
initrd /EFI/ARCH/initramfs-linux.img
options "root=UUID=6287d387-be27-4e98-b453-
6fb0ea3154fa rw
add_efi_memmap          initrd=\EFI\ARCH\intel-
ucode.img"
submenuentry "Boot using fallback initramfs" {
initrd /EFI/ARCH/initramfs-linux-fallback.img
}
}
menuentry "Windows 11 (Education) {"
loader \EFI\Microsoft\Boot\bootmgfw.efi
}
```



Voorbeeld bootpartitie (Multi-boot):

- Arch Linux Boot files
 - Root file system (Back-up)
 - Root file system
 - Intel CPU patch (Optioneel)
 - Kernel
- Windows Boot Files
- Bootloader (rEFInd)



5.7 Voorbeelden FS

Voorbeelden FS:

- Windows:
 - NTFS
 - FAT32
 - exFAT
- Mac:
 - HFS+
 - APFS
- Linux:
 - ext4
 - ZFS
 - trfs
- Andere:
 - ISO 9660 / UDF

NTFS:

- New Technology File System
- Standaard file system op **Windows**
 - Ook bruikbaar op Mac en Linux via de NTFS-3G driver
- Gebruikt journaling
- Vervangt oudere file systems zoals FAT32

FAT32:

- File Allocation Table
- Ontwikkeld door Microsoft maar werkt op verschillende OS
- Opvolger FAT12 en FAT16
- 32 bit voor adressering clusters ($0 - 2^{32}-1$)
 - Maximum bestandsgrootte: 4GB
 - Maximum grootte bestandssysteem: 2TB

exFAT:

- Extensible File Allocation Table
- Ontwikkeld door Microsoft
 - Goede ondersteuning op Mac en Linux
- Gericht op USB-sticks en SD-kaarten
 - Vervangt FAT32 voor deze toepassingen
- Maakt gebruik van een file allocation table
- Gebruikt geen journaling

HFS+:

- Hierarchical File System Plus
- Was het standaard file system op **Mac** tot enkele jaren terug
 - Beperkte ondersteuning op Windows en Linux
- Gebruikt journaling
- Vervangen door APFS en niet langer aan te raden

APFS:

- Apple File System
- Standaard file system op **Mac**
 - Beperkte ondersteuning op Windows en Linux
- Sterke focus op SSDs en encryptie
- Gebruikt GPT partitionering met containers en volumes

Ext4:

- Fourth extended file system
- Standaard file system op vele **Linux**-distributies
- Beperkte ondersteuning op Windows en Mac
- Gebruikt journaling

ZFS:

- Zettabyte File System
- Ontwikkeld door Sun Microsystems voor Solaris
- Populair op **Linux** en **FreeBSD**
- Heel geavanceerd: volume management, snapshots maken, klonen, integriteitscontrole, caching, ...
- Niet zo flexibel als andere file systems
- Heel sterk tegen bitrot en data corruptie

Btrfs:

- B-tree file system / Butter FS / Better file system
- Ontwikkeld door Oracle
- Antwoord op ZFS
- Standaard file system op **Fedora**
- Sterk tegen bitrot en datacorruptie

ISO 9660 / UDF:

- File systems voor **optische schijven** (resp. CD en DVD)
- Vooral gericht op write-once media
- Ondersteund op Windows, Mac, en Linux

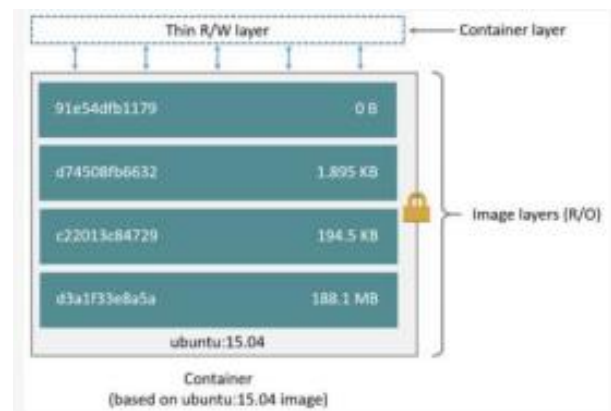
5.8 Opslag in Docker

VM vs container:

- Een virtuele machine bevat één of meerdere **virtuele disks**
 - Typisch een bestand, opgeslagen op de fysieke schijf van de host bv. in Virtualbox bestand in VDI-formaat met extensie .vdi
 - Deze virtuele disk stelt een virtueel blokapparaat voor
 - Een virtuele disk heeft eigen partitietabel, en elke partitie heeft eigen file system
- Een docker container heeft geen virtuele disk
 - Bestanden in Docker container worden opgeslagen in **writable container layer**
 - Beheer van deze layer gebeurt via een **storage driver**

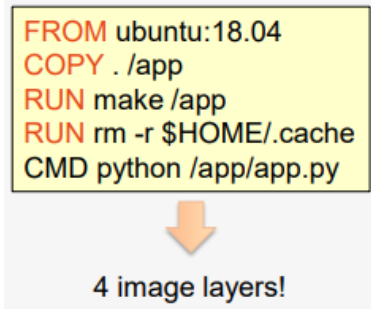
Storage driver en layers:

- Storage driver en layers Een docker container bestaat typisch uit:
 - Meerdere image layers (readonly)
 - Eén schrijfbare container layer (writeable)
- De onderste layer is de basis image, en elke layer houdt wijzigingen bij ten opzichte van de onderliggende layer.
- Bij aanmaken van een nieuwe container maak je een nieuwe writable layer, vaak de "container layer" genoemd. Dit is de enige layer waarin de container wijzigingen kan wegschrijven.
- De storage driver is verantwoordelijk voor het beheer van deze layers.
- Basislaag van een dockerfile van apache, Ubuntu → Apache → index.html. Dit alles maakt deel uit van een dockerimage (statisch) en daarboven zitten je logs (wat wordt gebruikt als je de image runt)



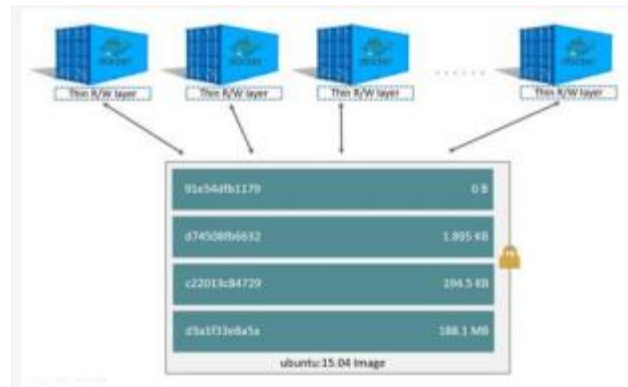
Voorbeeld image layers:

- Het **FROM** statement maakt een layer aan op basis van de ubuntu:18.04 image
- Het **COPY** commando kopieert enkele bestanden naar de container, en slaat dit op in een nieuwe layer
- Het eerste **RUN** commando compileert de applicatie en schrijft het resultaat naar een nieuwe layer
- Het tweede **RUN** commando verwijdert tijdelijke bestanden, en schrijft het resultaat naar een nieuwe layer
- De **CMD** instructie zegt wat de container moet doen bij opstarten, dit voegt geen nieuwe image layer toe (maar wijzigt de metadata van de image)



Container Layer:

- Bij aanmaken van een container maak je een **writable layer** bovenop de image layers
 - Deze layer noemen we de **container layer** (= thin R/W layer)
 - Alle **wijzigingen** van **data** tijdens **uitvoeren** van de container worden bijgehouden in deze container layer
- Bij verwijderen container wordt de container layer verwijderd
 - Onderliggende image layers worden niet verwijderd!
- Meerdere containers kunnen dezelfde onderliggende image (layers) gebruiken
 - Elke container houdt wijzigingen ten opzichte van image bij in eigen container layer
- Container layer is uniek (de logs)



Persistente opslag in Docker:

- Aangezien de container layer verwijderd wordt bij verwijderen container, is dit niet aangeraden voor persistente opslag (bv. bewaren data in databank).
- De layers worden weliswaar “ergens” opgeslagen op het fysieke filesystem van de Linux host, maar het is niet de bedoeling om bestanden en mappen in deze layers manueel te wijzigen vanaf de host.
- Voor persistente opslag is het dus beter om gebruik te maken van **volumes** en **bind mounts** (zie H2.3 - Docker).
 - Volumes zijn nuttig om data te delen tussen verschillende containers
 - Bind mounts zijn nuttig als je vanaf de host ook toegang wil tot de data

6 Hoofdstuk 6: Threads

6.1 Wat zijn threads?

Processen: 2 concepten:

- Een proces bestaat uit 2 concepten:
 - **Eigendom** van bronnen
 - **Uitvoering** van het programma
- Deze 2 concepten worden **onafhankelijk behandeld** door het besturingssysteem
 - Bronnen zoals bestanden blijven toegewezen aan proces, ongeacht of dit proces actief of geblokkeerd is

Proces: eigendom bronnen:

- Elk proces krijgt controle of **eigenaarschap** over **bronnen** (locking)
- **Afgeschermd** van andere processen door het besturingssysteem
- Voorbeelden van bronnen:
 - Address space (geheugen voor procesbeeld: instructies, data, ...)
 - Geheugen (in RAM, HDD, SSD, ...)
 - Bestanden
 - Apparaten (bv. CPU, ...)

Proces: uitvoering programma:

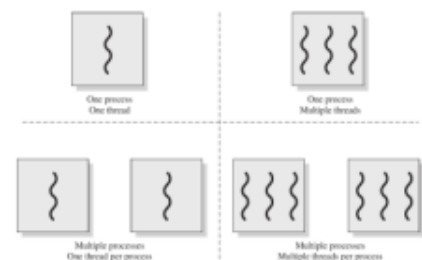
- Uitvoeren van **instructies**
 - Via fetch-execute cyclus
- Bijhouden **registers** en **stack**
 - Bv. program counter (PC)
- Scheduling
 - Bijhouden toestand/staat
 - Is proces actief/geblokkeerd/... ?
- Je voert 20000 lijnen Java uit, middenin stopt hij voor een aantal nanoseconden en dan doet hij weer voort, maar hij onthoudt die 10000 lijnen die hij al heeft gedaan met al zijn variabelen/berekeningen ... Dit wordt gezet op de stack.

Program counter:

Een soort bladwijzer in je code die bijhouden wat je threads doen.

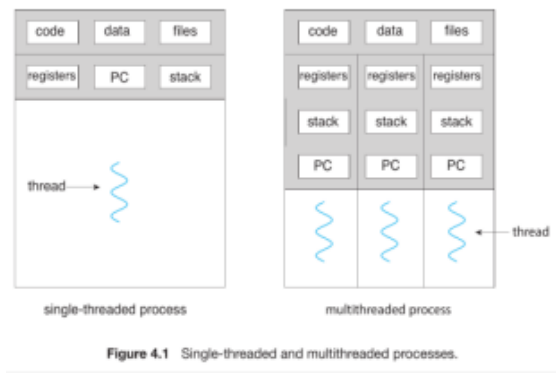
Processen vs. threads:

- Uitvoeringsgedeelte proces = **thread**
- Binnen een proces:
 - Enkele thread: single-threaded
 - Meerdere threads: multi-threaded
- Multi-threading:
 - Parallele taken binnen een proces mogelijk (via meerdere threads)



Opbouw threads:

- Een thread **bestaat uit**:
 - Thread ID
 - Registers (Program Counter, ...)
 - Stack
 - Toestand (status)
- Threads in een proces **delen**:
 - Instructies
 - Data
 - Toegang tot bronnen



6.2 Soorten threads

User-Level Threads (ULT):

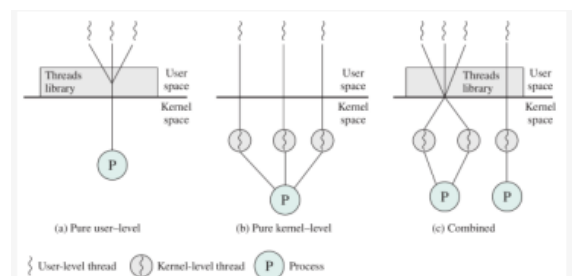
- Programma gebruikt **bibliotheek** voor threading
- OS beschouwt programma als **single-threaded** proces
- Lichte threads binnen de user space (bv. coderen in Java = dat niveau)
- **Voordelen**
 - Geen system calls van het OS nodig voor threading (**sneller, licht**)
 - Proces heeft zelf controle over scheduling threads
 - **Portable (draagbaar)**
 - Onafhankelijk van OS
- **Nadelen**
 - Als thread blokkeert op I/O, wordt hele process geblokkeerd
 - Geen gebruik van **multiprocessing**

Kernel-Level Threads (KLT):

- De logica voor threads zit in het besturingssysteem
- Zware threads op niveau van de kernel
- **Voordelen**
 - Scheduling mogelijk op threadniveau
 - Blokkeren van één thread geen invloed op andere threads process
 - **Multiprocessing mogelijk**
 - Besturingssysteem gebruikt waarschijnlijk zelf threads
- **Nadelen**
 - Trager (met wisselen tussen programma en besturingssysteem)
 - **Zwaar**
 - **OS-afhankelijk (Niet portable)**

Combinatie ULT en KLT:

- Combinatie van User-Level en Kernel-Level threads
- Beste van de twee werelden
- Kan op verschillende manieren



6.3 Voor- en nadelen

Voordelen multi-threading:

- Niet noodzakelijk om hele proces te blokkeren
- Eenvoudiger om bronnen te delen
- Lichter en sneller dan processen (creatie, context switch, ...)
 - Bv. wisselen van cursussen is makkelijker (want switchen van pdf is makkelijk), maar wisselen van locatie + bureau voor wisselen van cursus is veel moeilijker
- Efficiënt gebruik van multi-core CPU's (**parallelisme**)
- **Kan meer dingen tegelijk doen (applicatie sneller maken, want werkt op meerdere CPU's)**
- **Het zal de gebruikers ervaring beter maken (zoals databank inladen en tegelijk surfen op internet)**



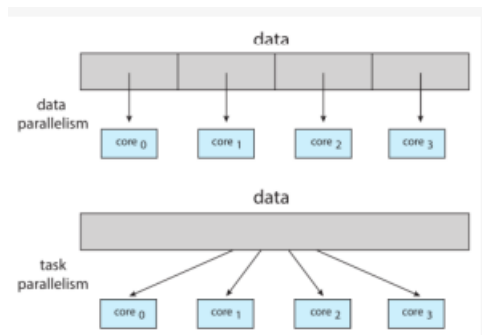
Uitdagingen bij multi-threading:

- Indelen van het rekenwerk in mogelijke threads
- Opsplitsen van data over de threads
- Afhankelijkheid van data tussen threads
- Testen en debuggen

6.4 Parallelisme

Soorten parallelisme:

- **Data** parallelisme
- **Task** parallelisme
- Vaak een **combinatie** van beide
- **Ontwerpen applicatie zodat het werk splitsbaar is**
- **Splits taken in verschillende processen**



Dataparalelisme:

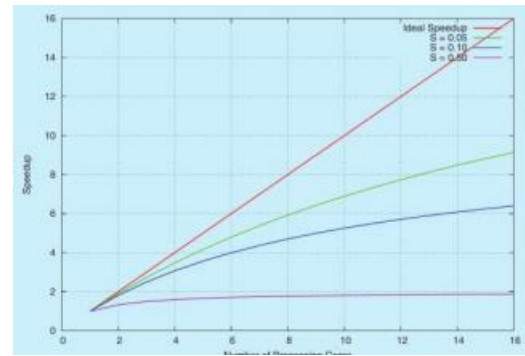
- Je splitst je data op in **3 stukken**, maar voert dezelfde functie uit met verschillende variabelen. `berekenScore()` voor speler1, `berekenScore()` voor speler 2, voor speler 3 ...
- Dit kan ook bv. met het **uploaden van een video**. De video wordt **gesplitst** in een aantal delen van 5 minuten (25 min video), elke 5 minuten worden dan apart, maar uiteindelijk tezamen gedenderd.

Amdahl's Law (formule krijg je op het examen):

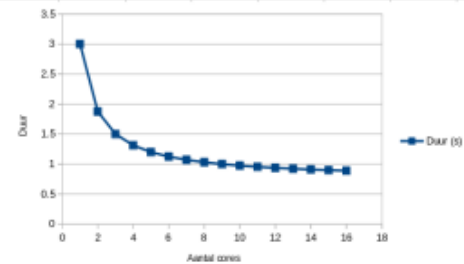
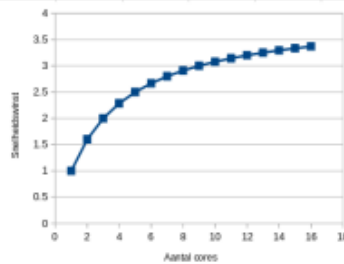
- Geeft de theoretische snelheidswinst bij het toevoegen van CPU cores
 - **S**: % van het programma dat **niet versneld** kan worden met meer CPU cores
 - **N**: aantal CPU cores
- Snelheidswinst = $\frac{1}{S + \frac{1-S}{N}}$ | $\frac{1}{0.25 + \frac{1-0.25}{2}} = 1.6x$ sneller | $\frac{1}{0.25 + \frac{1-0.25}{4}} = 2.29x$
- Bv: Stel dat een programma voor **25 %** bestaat uit een gedeelte dat **niet versneld kan worden** en voor **75 % uit een gedeelte dat wel versneld** kan worden met meerdere CPU cores:
 - Voor N = 2 CPU cores: het programma zal 1.6x sneller draaien met 2 CPU cores dan met 1 core
 - Voor N = 4 CPU cores: het programma zal 2.29x sneller draaien met 4 CPU cores dan met 1 core
- **Conclusie, hoe meer cores uiteindelijk, hoe minder snel je applicatie wordt.**
- **Hij wordt wel sneller, maar de curve gaat veel minder snel omhoog.**

Amdahl's Law:

- Wat als $N = \infty$ (oneindig)?
- Stel nu dat we een machine hebben met 1000000 cores. Als we $N = 1000000$ plaatsen in de formule van het vorig voorbeeld, dan komen we aan een versnelling van ongeveer 4.
- Deze winst zal niet meer verhogen, ook al voegen we bijvoorbeeld een miljard cores toe. Dit komt omdat $S = 0.25$ (25 %) van het programma geen snelheidswinst heeft bij het toevoegen van meerdere cores. Zelfs als we de resterende 75 % over zoveel cores verdelen dat dat gedeelte van het programma bijna geen tijd inneemt, gaat er toch nog steeds 25 % tijd naar het S-gedeelte.
- Het programma kan dus nooit meer versnellen dan dat. Ondanks dat threads voor een serieuze snelheidswinst kan zorgen, wordt deze snelheidswinst nog steeds beperkt door delen van het programma die niet versneld kunnen worden door multi-threading en multi-processing. Dit is bijvoorbeeld vaak het geval bij games. Veel zaken kunnen uitgespreid worden over meerdere threads, maar er is meestal één thread die de andere threads aanstuurt en beheert. Vaak is het deze thread die de snelheid bij multi-threaded games beperkt.



Uitvoertijd bij 1 core (s)		3
S		0.25
Aantal cores	Snelheidswinst	Duur (s)
1	1.000	3.000
2	1.600	1.875
3	2.000	1.500
4	2.296	1.313
5	2.500	1.200
6	2.667	1.125
7	2.800	1.071
8	2.909	1.031
9	3.000	1.000
10	3.077	0.975
11	3.143	0.955
12	3.200	0.938
13	3.250	0.923
14	3.294	0.911
15	3.333	0.900
16	3.368	0.891



6.5 Voorbeelden

Voorbeeld: game

- Main thread: beheert alle andere threads
- Render thread: genereert de visuals
- Audio thread: speelt de audio af
- I/O thread: uitwisseling data met RAM, HDD/SDD, netwerk, ...
- Extra threads, bijvoorbeeld:
 - Genereren wereld
 - Gedrag NPC's (AI)
 - Berekenen van paden (bv. AoE2) ...



Voorbeeld: word processor:

- Thread voor UI
- Thread voor spellchecker
- Thread voor opslaan van document
- Thread voor detecteren veranderingen aan document door andere programma's
- Thread voor controleren updates
- **Soort thread: instructie**



Voorbeeld: 3D modelling:

- Thread voor UI
- Werk voor rendering wordt verdeeld over meerdere threads voor parallelisme
 - Kan op CPU of op GPU



Voorbeeld: video conversies:

- Werk voor het converteren wordt verdeeld over meerdere threads voor parallelisme
- Hoe meer cores, hoe meer (muziek)bestanden hij kan converteren

