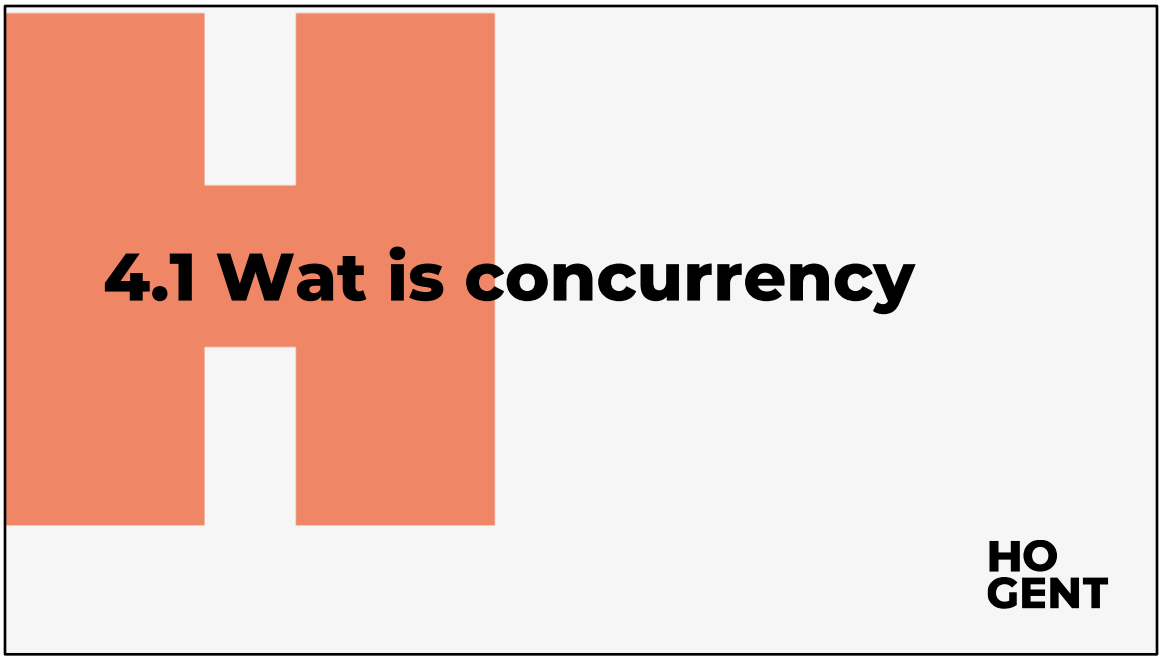
H4 - Concurrency

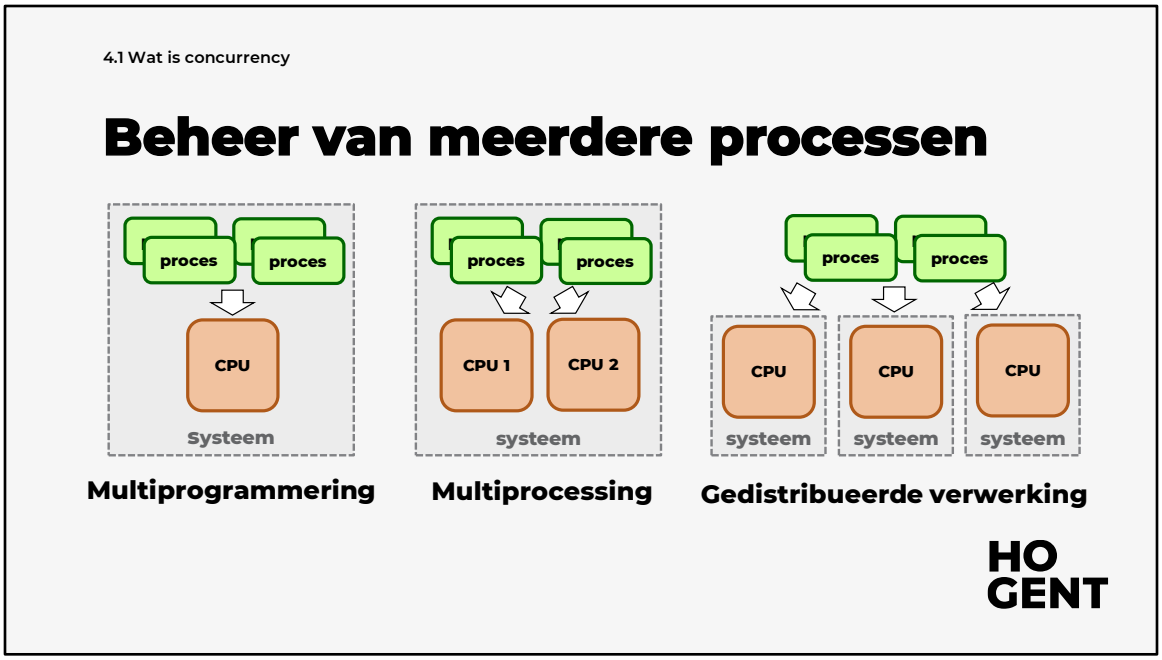
HO GENT

4. Concurrency

4.1 Wat is concurrency
4.2 Wederzijdse uitsluiting (mutual exclusion)
4.3 Synchronisatie
4.4 Deadlock
4.5 Threads

HO GENT

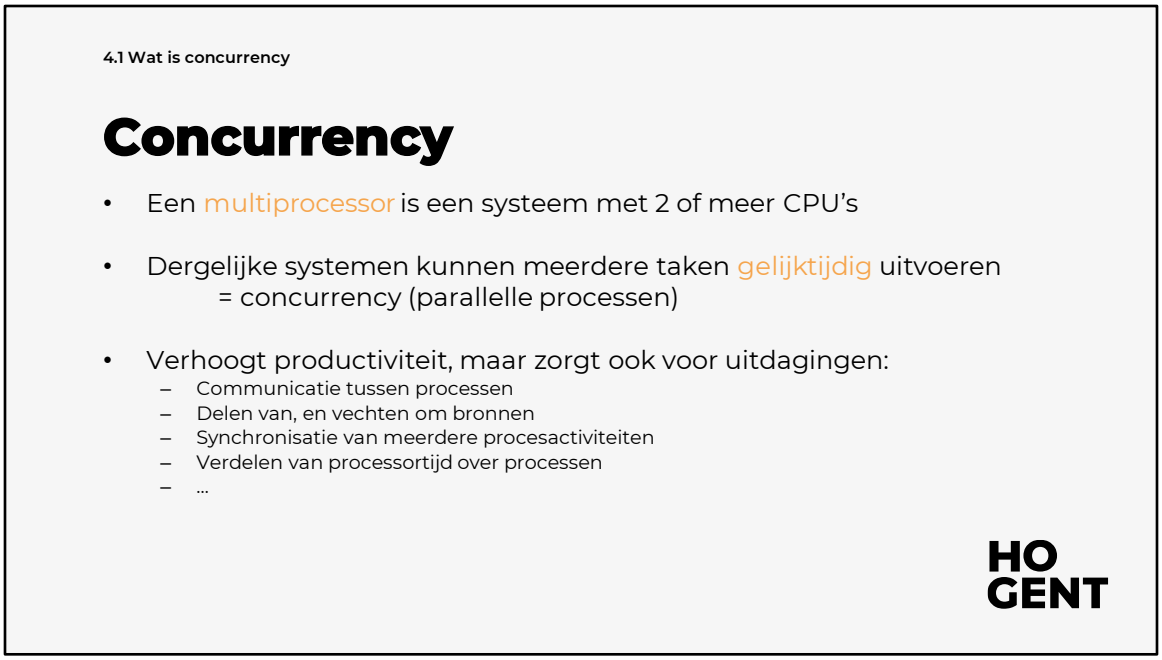

4.1 Wat is concurrency
HO
GENT



Beheer van meerdere processen

In het vorig hoofdstuk hebben we gezien hoe op de meeste moderne computersystemen meerdere processen gelijktijdig actief kunnen zijn. Deze processen kunnen al dan niet gelijktijdig uitgevoerd worden. Concreet kunnen we een onderscheid maken tussen:

- 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

Concurrency (ofwel parallelle processen) is bij computerprocessen een belangrijke plaats gaan innemen. Doordat het mogelijk werd enorme rekencapaciteit in een kleine chip te stoppen, zijn multiprocessors gemeengoed geworden. Een multiprocessor is een computersysteem met twee of meer processoren. Dergelijke systemen kunnen meerdere taken gelijktijdig uitvoeren. Dit verhoogt de productiviteit, maar zorgt ook voor enkele uitdagingen:

- Hoe kunnen verschillende processen met elkaar communiceren?
- Wat moet het OS doen wanneer processen computerbronnen delen, of vechten om bepaalde bronnen?
- Hoe kunnen activiteiten binnen verschillende processen gesynchroniseerd worden?
- Hoe moet de beschikbare processortijd verdeeld worden over de verschillende processen?

Algemeen kunnen we stellen dat concurrency verwijst naar processen of activiteiten die gelijktijdig uitgevoerd worden. Wanneer deze moeten samenwerken, bijvoorbeeld wanneer ze informatie moeten uitwisselen of bronnen delen, is het een uitdaging om

dit probleemloos te laten verlopen. Het tegengestelde van concurrency zijn sequentiele processen, waarbij alle stappen strikt na elkaar worden uitgevoerd, en er dus van gelijktijdigheid geen sprake is.

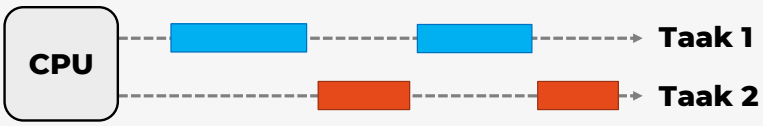
Verschillende situaties

Concurrency beperkt zich niet tot computersystemen met meerdere processoren. Ook bij systemen met maar 1 CPU kan concurrency optreden. Concurrency treedt op in verschillende situaties:

- Meerdere toepassingen: bij multiprogrammering kunnen er verschillende processen gelijktijdig actief zijn, en multiprogrammering werd uitgevonden om de verwerkingstijd (processortijd) dynamisch te kunnen verdelen over een aantal actieve toepassingen
- Gestructureerde toepassingen: als uitbreiding op de beginselen van modulair ontwerpen en gestructureerd programmeren kunnen sommige toepassingen effectief worden geprogrammeerd als een verzameling van gelijktijdige processen
- Structuur van besturingssysteem: dezelfde voordelen van het structureren gelden ook voor de systeemprogrammeur, en besturingssystemen zelf worden vaak geïmplementeerd als een verzameling processen

4.1 Wat is concurrency

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

HO GENT

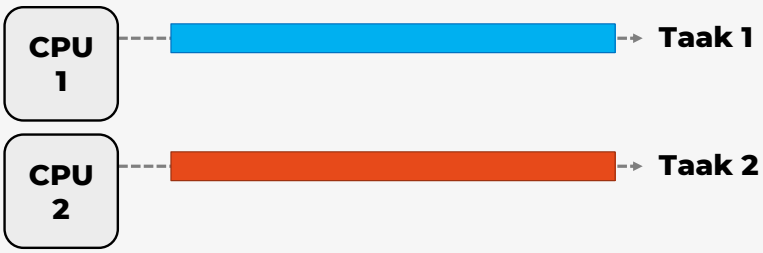
Concurrency bij systemen met 1 processor

Concurrency betekent dat een toepassing vooruitgang boekt op meer dan één taak - tegelijkertijd of op zijn minst schijnbaar tegelijkertijd (gelijktijdig).

Als de computer slechts één CPU heeft, kan de toepassing op exact hetzelfde moment geen vooruitgang boeken, maar is er meer dan één taak aan de gang op een moment binnen de toepassing. Om tegelijkertijd vooruitgang te boeken met meer dan één taak schakelt de CPU tussen de verschillende taken tijdens de uitvoering.

4.1 Wat is concurrency

Parallel Execution



Parallelle uitvoering
- Systemen met meer dan één CPU of CPU-kern
- Meer dan één taak van een toepassing worden gelijktijdig (parallel) uitgevoerd
- ≠ parallelisme (zie verder)

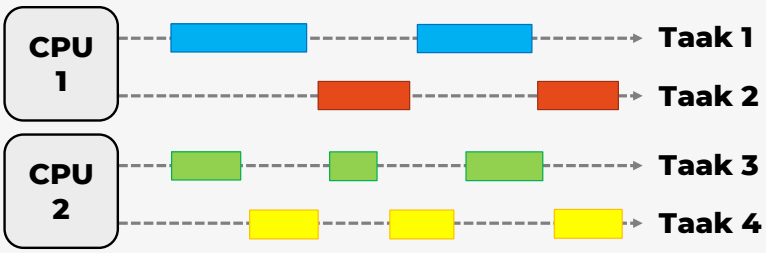
HO GENT

Parallelle uitvoering

Parallelle 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. Parallelle uitvoering verwijst echter niet naar hetzelfde fenomeen als parallellisme. Parallelle uitvoering wordt geïllustreerd in het diagram in deze slide.

4.1 Wat is concurrency

Parallel Concurrent Execution



Parallelle 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

HO GENT

Parallelle gelijktijdige uitvoering

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.

4.1 Wat is concurrency

Parallelism

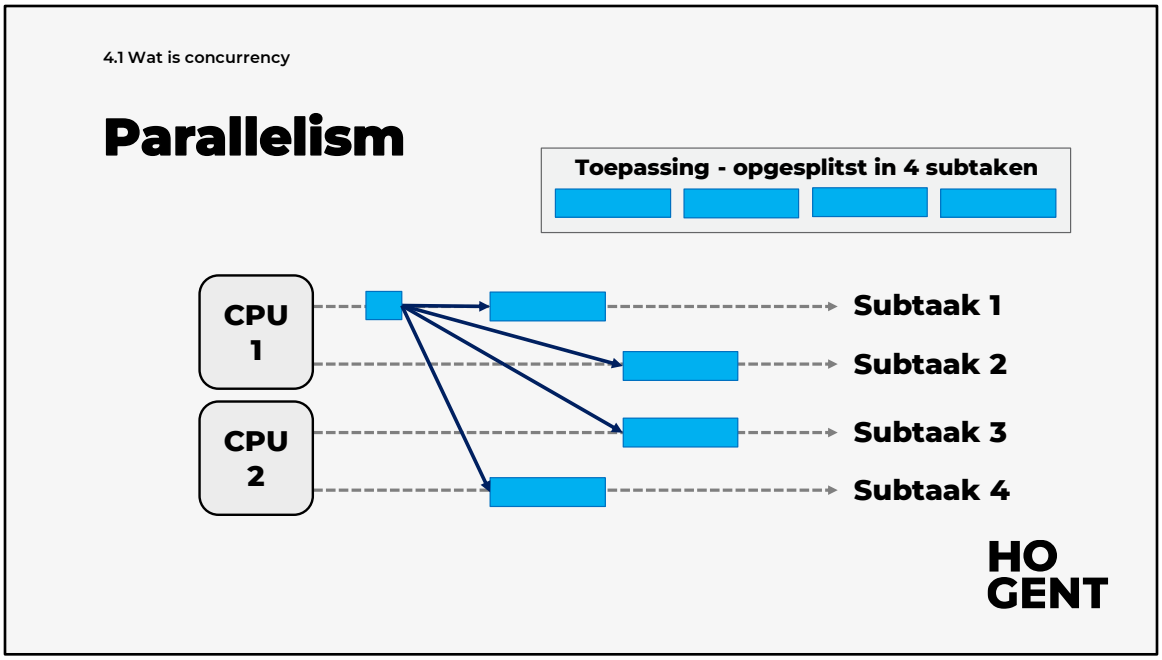
- Toepassing splitst zijn werk op in subtaken die parallel kunnen worden verwerkt
- Verwijst niet naar hetzelfde uitvoeringsmodel als parallelle gelijktijdige uitvoering
- Hoe parallellisme bereiken:
  - Meer dan 1 subtaak
  - Elke subtaak draait parallel op afzonderlijke CPU's of CPU-cores

HO GENT

Parallellisme

De term parallellisme *(EN: parallelism)* betekent dat een toepassing zijn werk opsplitst in kleinere subtaken die parallel kunnen worden verwerkt, bijvoorbeeld op meerdere CPU's op exact hetzelfde moment. Parallellisme verwijst dus niet naar hetzelfde uitvoeringsmodel als parallelle gelijktijdige uitvoering - zelfs als ze er op het oppervlak vergelijkbaar uitzien.

Om echte parallellisme te bereiken moet een toepassing meer dan subtaak hebben die wordt uitgevoerd - en elke subtaak moet op afzonderlijke CPU's / CPU-cores / GPU-cores draaien.

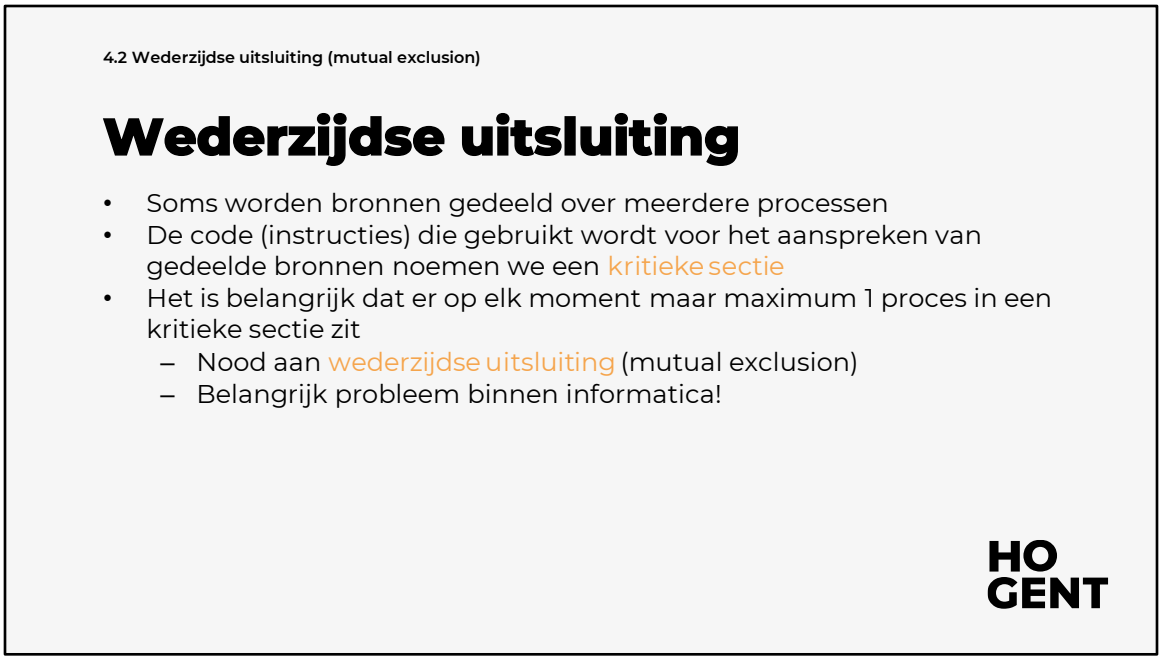


Parallellisme

Het diagram in deze slide illustreert een grotere taak die wordt opgesplitst in 4 subtaken. Deze 4 subtaken worden uitgevoerd op 2 verschillende CPU's. Dit betekent dat delen van deze toepassing gelijktijdig worden uitgevoerd (op dezelfde CPU) en delen parallel worden uitgevoerd (op verschillende CPU's).

4.2 Wederzijdse uitsluiting (mutual exclusion)

HO GENT



Wederzijdse uitsluiting

Soms willen meerdere taken of processen gelijktijdig gebruikmaken van dezelfde gedeelde computerbronnen, bijvoorbeeld wanneer ze hetzelfde deel van het gemeenschappelijk RAM geheugen willen aanspreken. Dit kan echter voor conflicten en inconsistenties zorgen. Zoals we reeds gezien hebben in het vorig hoofdstuk, bestaat een proces uit meerdere instructies die uitgevoerd worden op een processor. De instructies (code) voor het aanspreken van die gedeelde bronnen (bijvoorbeeld lezen of schrijven naar gemeenschappelijke data) noemen we een kritieke sectie. Het is belangrijk dat er op elk moment maar 1 proces in een kritieke sectie zit.

We willen processen gelijktijdig laten uitvoeren, en tegelijkertijd toch voorkomen dat bepaalde delen van die processen, de kritieke secties, parallel worden verwerkt. Wanneer parallelle processen zich toegang verschaffen tot het gemeenschappelijke geheugen, bevatten hun kritieke secties de opdrachten die deze resources aanspreken. De kritieke sectie van een proces is hier dus de code die naar gemeenschappelijke data verwijst. Als de uitvoering van een proces in de kritieke sectie is aangeland, moeten wij er voor zorgen dat elk ander proces zijn eigen kritieke sectie niet betreedt. Omgekeerd moeten wij ook opletten dat een proces zijn kritieke

sectie niet binnenkomt op het moment dat een ander proces in zijn kritieke sectie zit.

Wederzijdse uitsluiting is een term uit de informatica waarmee de eis bedoeld wordt dat wanneer een proces zich in een kritieke sectie bevindt en er gebruikgemaakt wordt van gedeelde bronnen, er geen andere processen zijn die zich ook in een kritieke sectie bevinden waarbij dezelfde gedeelde bronnen worden gebruikt. Het regelen van de toegang tot gedeelde bronnen is een belangrijk probleem in de computerwetenschappen. Dit komt doordat taken op elk willekeurig moment kunnen starten of stoppen.

4.2 Wederzijdse uitsluiting (mutual exclusion)

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

HO GENT

Wederzijdse uitsluiting: voorbeeld

We kunnen het belang van wederzijdse uitsluiting illustreren aan de hand van een eenvoudig voorbeeld. Stel dat je in het geheugen een globale variabele hebt, **getal**, die een geheel getal voorstelt. Een proces kan deze waarde aanpassen via volgende 3 instructies:

1. Lees de huidige waarde van de variabele **getal** in vanuit het geheugen
2. Verhoog deze waarde met 1
3. Schrijf de nieuwe waarde voor **getal** weg naar het geheugen

Stel nu dat er 2 processen zijn, proces A en proces B, die parallel uitgevoerd worden, en beide processen willen bovenstaande instructies uitvoeren. We kunnen dit voorstellen als A1, A2, A3 (instructies 1, 2 en 3 voor proces A) en B1, B2, B3 (voor proces B).

Als beide processen na elkaar de instructies uitvoeren is er geen probleem, en zal de waarde van **getal** met 2 verhoogd zijn. De volgorde van uitvoering is dan bijvoorbeeld: A1 – A2 – A3 – B1 – B2 – B3

In het vorig hoofdstuk hebben we echter gezien dat een proces onderbroken kan worden tijdens de uitvoering. Stel bijvoorbeeld dat proces A onderbroken wordt na uitvoeren van de eerste instructie, en proces B van aan bod komt. De volgorde van uitvoeren is dan bijvoorbeeld: A1 – B1 – B2 – B3 – A2 – A3

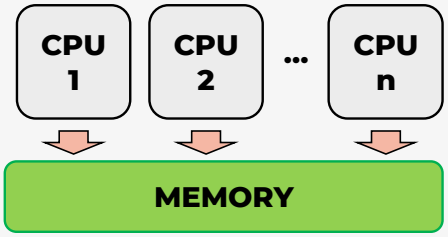
Merk op dat de eerste instructie hier niet opnieuw uitgevoerd wordt, waardoor proces A nog de 'oude' waarde kent van de variabele **getal**. Na uitvoeren van alle stappen zal **getal** hier dus niet verhoogd zijn met 2, maar met 1.

Om dit te voorkomen, zullen we de 3 instructies dus moeten groeperen in een kritieke sectie, en afdwingen dat proces B deze instructies niet mag uitvoeren wanneer proces A hier reeds mee begonnen is.

4.2 Wederzijdse uitsluiting (mutual exclusion)

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



HO GENT

Wederzijds uitsluiting bij multiprocessing

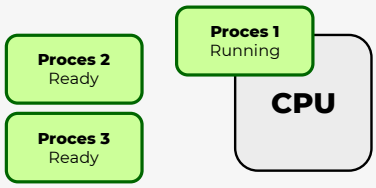
Zoals we reeds gezien hebben bestaan er verschillende niveaus van concurrency. Niet alleen processen, maar ook activiteiten binnen één proces kunnen parallel worden uitgevoerd. In dit hoofdstuk zullen we hoofdzakelijk processen bespreken, maar de principes gelden eveneens voor activiteiten binnen één proces.

Als parallelle processen niets gemeenschappelijk gebruiken, is er geen probleem. De moeilijkheden ontstaan wanneer de processen het gemeenschappelijke geheugen aanspreken.

4.2 Wederzijdse uitsluiting (mutual exclusion)

Wederzijdse uitsluiting bij multiprogrammering

- 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



HO GENT

Wederzijdse uitsluiting bij multiprogrammering

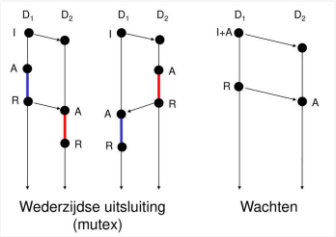
Ook wanneer een computersysteem maar één processor heeft, zijn gelijklopende processen mogelijk. De processen kunnen in dergelijk geval uiteraard niet parallel (gelijktijdig) uitgevoerd worden, maar ze kunnen wel op hetzelfde moment proberen de besturing van de CPU te krijgen.

Wanneer twee van zulke processen het gemeenschappelijk geheugen willen aanspreken, kunnen er nog steeds problemen ontstaan, en kan wederzijds uitsluiting nodig zijn.

4.2 Wederzijdse uitsluiting (mutual exclusion)

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



HO GENT

Wederzijdse uitsluiting afdwingen

Het is in de praktijk is het niet zo eenvoudig om wederzijdse uitsluiting af te dwingen. Een eenvoudige oplossing zou kunnen zijn om met 1 globale variabele (bijvoorbeeld een boolean) bij te houden of er een proces in een kritieke sectie zit, maar dan is het probleem verschoven naar de toegang tot deze globale variabele.

Er bestaan algoritmes die een oplossing bieden voor het probleem van wederzijdse uitsluiting. Het algoritme van Dekker bijvoorbeeld is de eerste bekende juiste oplossing voor dit probleem, maar is beperkt tot wederzijdse uitsluiting voor 2 parallelle processen. In 1981 werd er een oplossing geformuleerd, Peterson's algoritme, die ook bruikbaar is voor wederzijdse uitsluiting bij meer dan 2 processen. De details van deze algoritmes vallen echter buiten de scope van deze cursus.

Daarnaast bestaat er ook een alternatieve oplossing, die gebruik maakt van semaforen. Een semafoor is een soort van integer variabele, bedacht door Dijkstra, die slecht door enkele primitieve operaties kan gewijzigd worden. Een primitieve operatie is een operatie die niet onderbroken kan worden: ofwel wordt de operatie volledig uitgevoerd, ofwel wordt deze volledig ongedaan gemaakt. Een ander

alternatief is om monitoren te gebruiken. Een monitor is een constructie in een programmeertaal die een functionaliteit biedt die vergelijkbaar is met die van semaforen, maar gemakkelijker te besturen is.

Extra informatie (ter info):

- https://en.wikipedia.org/wiki/Dekker%27s_algorithm
- https://nl.wikipedia.org/wiki/Wederzijds_uitsluitingsalgoritme_van_Peterson
- https://nl.wikipedia.org/wiki/Semafoor_(computer)
- https://nl.wikipedia.org/wiki/Monitor_(gedistribueerd_programmeren)

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

HO GENT

Meer dan toegang tot gedeeld geheugen

In de vorige slides hebben we wederzijdse uitsluiting besproken in de context van het regelen van de toegang van verschillende processen die uitgevoerd worden op de CPU naar gemeenschappelijk geheugen. Er bestaan echter ook andere vormen van wederzijdse uitsluiting:

- Wederzijdse uitsluiting kan ook gebruikt worden om de toegang naar bestanden te regelen, en bijvoorbeeld te voorkomen dat 2 processen gelijktijdig naar hetzelfde bestand willen schrijven (en zo elkaars wijzigingen ondermijnen).
- Daarnaast kan wederzijdse uitsluiting ook nuttig zijn om de toegang tot bepaalde hardware bronnen te regelen. Stel bijvoorbeeld dat 2 processen op exact hetzelfde moment een taak willen versturen naar een printer... Gelukkig gaat het besturingssysteem dit niet toelaten, en de printertaken bijhouden in een wachtrij zodat de documenten één voor één (sequentieel) afgeprint worden.

Algemeen kunnen we stellen dat het de taak is van het besturingssysteem om (waar nodig) wederzijdse uitsluiting te garanderen tot gedeelde computerbronnen.

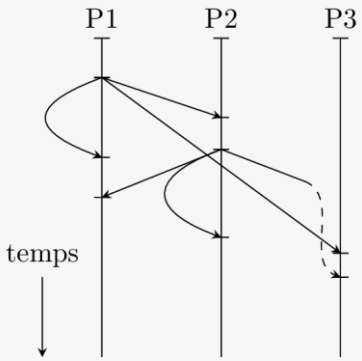
4.3 Synchronisatie

HO
GENT

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



HO GENT

Wat is synchronisatie?

Synchronisatie is het proces of het resultaat van iets gelijktijdig maken. Het is afgeleid van het Griekse συν (sýn) 'samen' en χρόνος (chrónos) 'tijd'. Pas in de negentiende eeuw is synchronisatie in de geschiedenis van de mens een rol van betekenis gaan spelen. De treinen gingen zo snel rijden dat een verschil in lokale tijd op ging vallen. Het gelijk zetten van de klokken langs de spoorlijn werd noodzakelijk. Synchronisatie van de klokken was ook een vereiste voor het veilig gebruik van enkelspoor. De treinen konden zo volgens het spoorboekje blijven rijden, zodat vermeden werd dat twee treinen tegelijkertijd op hetzelfde spoorvak op elkaar aanstormden. Sindsdien is het belang van synchronisatie alleen maar groter geworden (bron: https://nl.wikipedia.org/wiki/Synchronisatie).

Binnen de context van concurrency kunnen we synchronisatie ook definiëren als het opleggen van een dwingende volgorde aan events die door concurrente, asynchrone processen worden uitgevoerd. Bij concurrente processen weten we niet welk proces wanneer aan bod komt, maar toch willen we garanderen dat de uitvoering van (delen van) bepaalde processen in een bepaalde volgorde verlopen.