Informatica Deel III

Hoofdstuk 3

Besturingssystemen

Een *besturingssysteem* (in het Engels *operating system* of afgekort *OS*) is een programma (meestal een geheel van samenwerkende programma's) dat na het opstarten van een computer in het geheugen geladen wordt en alle mogelijkheden van de computer aan de gebruiker aanbiedt. Het OS biedt ook de functionaliteiten aan om andere programma's - applicaties genoemd - uit te voeren.

Zo'n applicatie maakt gebruik van het besturingssysteem door middel van een application programming interface (API). Deze API abstraheert de toegang tot de verschillende randapparatuur, zoals harde schijf, printer en beeldscherm. Het OS biedt zo een generieke interface aan voor allerhande software die op het systeem kan draaien, zodat deze software zich niet hoeft te bekommeren hoe de videokaart, harde schijf en andere hardware moet worden aangestuurd. Zonder OS moet elk programma hier zelf voor instaan. Dit 'verbergen' van de hardware voor de gebruiker en applicaties wordt getoond in Figuur 3.1.

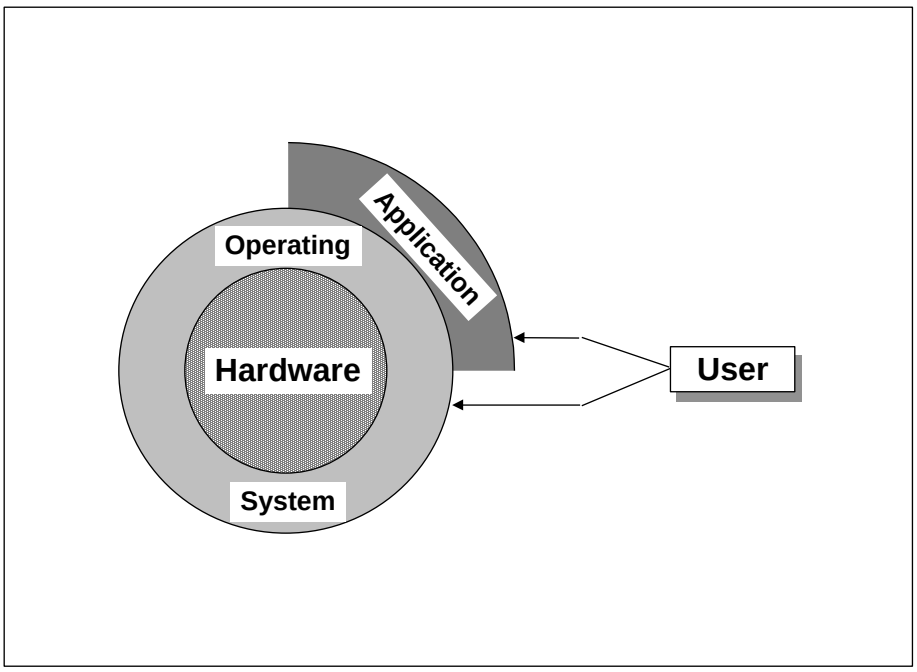

Fig.3.1. Operating system model

Er bestaan een heel aantal besturingssystemen. Het bekendste besturingssysteem is ontworpen door Microsoft: Windows. Apple computers hebben hun eigen besturingssysteem: MAC-OS. Dan is er nog een besturingssysteem dat gratis is: Linux. Linux is afgeleid van Unix, een veel gebruikt systeem in bedrijven. Google heeft onlangs zijn eigen besturingssysteem op de markt gebracht: Android. Maar

beide, Android en MAC-OS, zijn gebaseerd op Unix/Linux. Dit zijn de belangrijkste voor alledaagse computers.

3.1 Taken van een OS

3.1.1 Het opstarten van het systeem

In de BIOS van de computer staat beschreven welk programma gelaad moet worden bij het opstarten. Dit is meestal het besturingssysteem, omdat het OS voor het beheer van de computer zal zorgen. Het besturingssysteem wordt meestal van de harde schijf gelezen, maar het kan ook wel vanuit ROM-geheugen of vanaf een verwisselbaar medium zoals een cd-rom, dvd, of een flashgeheugen (USB-stick). Het opstarten van het systeem wordt *booten* genoemd.

3.1.2 Geheugenbeheer

De computer onthoudt waar elk bestand van de perifere of secundaire geheugens (zoals harde schijf) is opgeslagen. Dit noemen we 'het beheren van bestanden'en het filesysteem. Files zijn georganiseerd in een boomstructuur en worden, indien nodig, geladen in het werkgeheugen.

De computer kan verschillende dingen met bestanden doen:

• afdrukken maken op een printer;

• laten zien op het beeldscherm;

• beveiligen (zodat de inhoud van een bestand niet per ongeluk veranderd of gewist kan worden);

• een andere naam geven;

• kopiëren;

• verplaatsen;

• verwijderen

Zoals we verder zullen zien beheert het OS ook het werkgeheugen (RAM) die enkel aktief als de computer aan staat (dus samen met het aktief zijn van het OS).

3.1.3 Communicatie met randapparatuur: printers, USB, …

De OS regelt alle handelingen met randapparaten. Randapparaten zijn bijvoorbeeld de printer, het beeldscherm, toetsenbord en muis. Bij bijna alle randapparaten kun je eigenschappen instellen. Zo kun je via het besturingssysteem de scherpte van het beeldscherm instellen. Ook allerlei eigenschappen van de printer kun je met behulp van het besturingssysteem instellen. Dat maakt het voor een beginner vaak lastig. Gelukkig hoef je dat ook niet allemaal te doen. Als je niets doet, kiest de computer automatisch bepaalde instellingen. Zo'n instelling noemen we de *default instelling*.

Het communiceren van het OS met het randapparaat gebeurt via een *driver*. Deze driver wordt meestal automatisch geïnstalleerd als je het apparaat voor de eerste maal aansluit (windows geeft dit aan). Soms moet je het manueel doen of op het internet de laatste versie van de driver zoeken.

3.1.4 Communicatie met buitenwereld: netwerk & internet

Het OS regelt ook de communicatie met de buitenwereld. Dit gebeurt meestal automatisch door services of achtergrondprocessen.

3.1.5 Aanbieden gegevens (files) en applicaties van de computer aan gebruiker

Het OS biedt meestal een Graphical User Interface (GUI) waarmee je door de inhoud en mogelijkheden van je computer kan 'browsen'. Daarnaast biedt het ook de mogelijkheid aan applicaties om een gebruikersinterface weer te geven. In java kan dit via de Swing klasses. Daarmee maak je zelf een applicatie met een GUI.

Merk op dat je op elk OS de applicaties ook kan uitvoeren via commando's. In windows start je een commandowindow door 'cmd' in de 'Search program and files' te tikken. Je krijgt dan een zwart scherm waarin je DOS-commando's kan tikken. DOS was het OS van Microsoft in het pre-Windowstijdperk. In Unix/linux-gebaseerde systemen kan dit ook, daar wordt het een *shell* genoemd.

3.1.6 Beheer van aktieve applicaties en achtergrondprocessen

Het OS zorgt voor het uitvoeren van programma's. Het uit te voeren programma wordt naar het werkgeheugen gekopieerd. De processor voert vervolgens de instructies van het programma één voor één uit. Dit bespreken we in detail in 3.2.
Het OS werkt zelf met een groot aantal programma's. De meeste programma's zijn verborgen, we noemen dat *achtergrondprocessen*. Daar merk je dus niets van. Andere programma's zijn niet verborgen. Zo is er een programma waarmee je je files en folders kunt beheren (de explorer in windows). Deze 'utilitaire' programma's breiden in feite de hardware uit met allerlei praktisch-bruikbare functies.

3.1.7 Varia

Verder zijn er nog een aantal kleine taken:

- Toegangscontrole: Het besturingssysteem bevat meestal een login procedure waarbij gebruikers zich kenbaar moeten maken aan het computersysteem
- Energiebeheer bij laptops en computers die op batterijen werken.
- Uitwisseling van gegevens tussen programma's via het "Clipboard" (commando's *cut* - *copy* - *paste*).

3.2 Procesbeheer

Op een modern OS kan je meerdere programma's tegelijk draaien. Een achtergrondproces is in feite ook een gewoon programma, maar ze is niet zichtbaar voor de gebruiker. De achtergrondprocessen zorgen voor het beheer van het systeem of bieden *services* aan (zoals het sharen van je (muziek-)files en het checken van je mailbox).
Het uitvoeren van een programma resulteert in een proces. Als je het programma opnieuw start krijg je een nieuw proces. Een *proces* is dus een programma dat uitgevoerd wordt. Al die processen samen moeten beheerd worden door het OS.
Daarenboven kan één programma bestaan uit meerdere *threads*. Elke thread voert een sequentie van instructies uit. Een sequentie kan je zien als een 'draad', vandaar de benaming. Vele moderne toepassingen zijn zelf opgebouwd uit een groot aantal onafhankelijke threads die simultaan uitgevoerd worden. Het tekstverwerkingssysteem Word bv. gebruikt verschillende threads om teksten en figuren op het scherm te tonen, zodanig dat wanneer men snel door een tekst wenst te lopen men niet hoeft te wachten op het tekenen van alle figuren. Een andere thread zal je taalfouten opsporen (de spelling checker).

In de *task manager* (te starten in Windows met ctrl+alt+delete) zie je alle aktieve applicaties, alle processen en het processor- en geheugengebruik. Je ziet er wel niet de verschillende threads van een proces.

3.2.1 Process scheduling

Het simultaan uitvoeren van processen en threads stelt uiteraard verschillende technische problemen:

- hoe kiest men het programma waaraan de centrale verwerkingseenheid (Central Processing Unit of CPU) werkt?
- hoe bepaalt men welke programma's samen in werkgeheugen geplaatst worden?
- welk deel van het werkgeheugen mag gebruikt worden door elk programma? Elk programma krijgt slechts een deel van het werkgeheugen ter beschikking.
- hoe kan men informatie transfereren van een programma naar een ander ?
- hoe kan men de in- en uitvoer van verschillende programma's waaraan beurtelings gewerkt wordt gescheiden houden?

De *process-scheduler* is een deel van het besturingssysteem dat op elk ogenblik bepaalt aan welk van alle programma's die zich in werkgeheugen bevinden de centrale verwerkingseenheid zal werken. Het hart van de computer, de processor of CPU, kan immers maar 1 programma tegelijk uitvoeren. De processor is immers opgebouwd volgens de Von Neumann-architectuur. Karakteristiek voor deze architectuur is dat hij één programma stap-voor-stap uitvoert.
Moderne computers hebben echter meerdere processorcores (dual core, quadcore). Dat zijn dan in feite 2, respectievelijk 4 onafhankelijke processoren die elk één programma kunnen uitvoeren.
De *process-scheduler* zal de processortijd verdelen over de lopende processen: elk proces krijgt een periode (‘time slice’) toegekend.

De lopende processen, kunnen zich, op elk ogenblik in drie toestanden bevinden (fig.3.2):

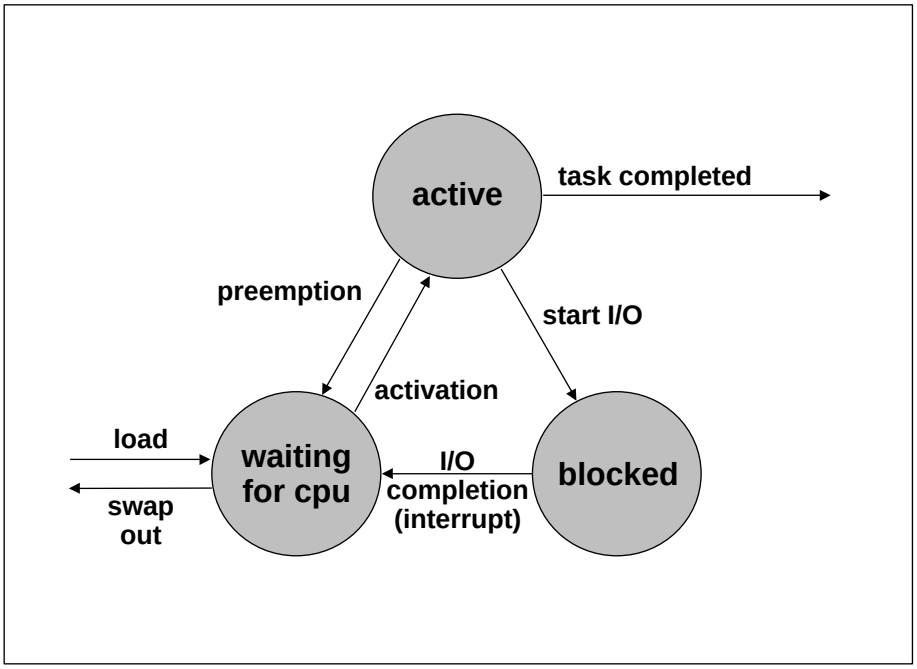

Fig.3.2. Procestoestanden

1) *actief*: de centrale verwerkingseenheid (processor) is aan het proces aan het werken; dwz. dat instructies van dat programma opgehaald worden door de stuureenheid,
2) *geblokkeerd*: een in- of uitvoeroperatie (Input/Output of I/O) is aan de gang en het proces moet wachten tot het einde ervan;

3) *wachtend*: het proces zou kunnen uitgevoerd worden, maar de centrale verwerkingseenheid is niet beschikbaar, ze is instructies van een ander proces aan het uitvoeren. Het proces moet wachten tot de proces-scheduler processorcycles ter beschikking stelt.

Een proces wordt uit de aktieve toestand gehaald in 3mogelijke gevallen: (a) als het einde van het programma bereikt is, (b) hij moet wachten op I/O of (c) hij de hem toegewezen periode heeft opgebruikt (zie schema). Het wisselen van aktief proces wordt een *context switch* genoemd. De processor 'switcht' van 1 programma naar een ander.

3.2.2 Interrupts

Met een Interrupt wordt een lopend programma onderbroken. Deze is voorzien in de hardware van de processor! Je kan immers niet verwachten dat een lopend programma zelf zal checken of hij verder kan gaan. Ook kan het OS er niet van uitgaan dat elk programma in een redelijke tijd stopt. Het OS moet op elk moment een programma kunnen onderbreken. Anders zou een 'oneindige lus' volledig beslag leggen op je processor zonder dat je er iets aan kan doen, behalve dan de computer herstarten.
Let op: het OS is ook niets meer dan een programma. Er is geen 'big brother' in je computer die toekijkt wat de processor doet. Controle over het aktieve proces wordt verkregen door *interrupts* ('onderbreking')**.**
Om toe te laten op gelijk welk ogenblik sprongen tussen toepassingsprogramma en besturingssysteem te maken bevatten processoren een hardwareschakeling waardoor een programma kan onderbroken worden door middel van een interrupt. Vervolgens bepaalt de *interrupt handler* welk programma uitgevoerd moet worden. Aan de hand van de oorsprong van de interrupt veroorzaakt deze een sprong naar het gewenste programma. De *interrupt handler* speelt dus een centrale rol en vormt dan ook het hoofdbestanddeel van de kern van moderne besturingssystemen.

De volgende interrupts kunnen gebeuren:

a) de toegewezen periode is opgebruikt. Dit gebeurt door een hardware-timer ('wekker') in de processor die afloopt. De scheduler wordt geaktiveerd en kiest welk wachtend proces geaktiveerd wordt.
b) het proces crasht door een onuitvoerbare instructie (deling door 0 bvb). Het OS wordt geaktiveerd met de mededeling van de crash.
c) het proces moet wachten op I/O. Het heeft dan geen zin om deze aktief te houden. De scheduler wordt geaktiveerd en houdt het proces geblokkeerd tot de I/O klaar is (input aangekomen is of output verzonden is).
d) Er is input toegekomen. Wanneer de muis bewogen wordt of wanneer een bepaalde toets of toetsen-combinatie ingedrukt wordt, willen we (soms) dat de computer onmiddellijk reageert. Met een interrupt kan het aktieve proces onderbroken worden en de input afgehandeld worden.

3.2.2 Scheduling

Het besturingssysteem beslist welk programma op elk ogenblik toegang krijgt tot de CPU('s) en voor hoelang wanneer verschillende programma's gelijktijdig in uitvoering zijn multitasking. Een veel voorkomende geval is dat de tijd van de CPU in kleine stukjes (fracties van een seconde) wordt verdeeld, en dat deze time slices om beurten aan de verschillende programma's worden toegewezen. Dit systeem wordt timesharing genoemd. Wanneer veel programma's interactief zijn (de gebruiker interageert met de computer van achter een terminal), zullen de verschillende programma's hierdoor nauwelijks zichtbaar vertragen. De reden hiervoor is dat de reactiesnelheid van de gebruiker naar computernormen erg traag is.
De keuze van het programma dat actief moet worden uit al die die wachten is een delikaat probleem dat de *process scheduler* moet oplossen. Die keuze wordt best gemaakt telkens een programma onderbroken werd. De process scheduler is dan ook logischerwijze een deel van het besturingssysteem dat nauw aansluit bij de interrupt handler. Verschillende algoritmen worden in process schedulers gebruikt. De criteria waaraan ze algemeen moeten voldoen zijn:
- eenvoudig zijn zodat de processor niet te veel tijd zou verliezen met de keuze van zijn volgende taak.
- programma's die veel in- en uitvoer operaties verrichten een hogere prioriteit toekennen dan diegene die alleen maar rekenen, om niet alleen de processor maar ook de randapparatuur optimaal te gebruiken.

Volgend algoritme is een goed voorbeeld van wat men in grote computers ontmoet:
De verschillende wachtende programma's hebben elk een prioriteit. De *process scheduler*, telkens hij een programma kan activeren, kiest het meest prioritaire. Bij het begin van de uitvoering kan deze prioriteit voor alle programma's dezelfde zijn, of kan deze afhangen van externe factoren, eigen aan de toepassing. Zo zal bv. in Word het programma dat tekeningen toont een lagere prioriteit hebben dan het programma dat teksten toont. Wachtende programma’s zien hun prioriteit regelmatig verhogen, zodanig dat oudere programma’s prioriteit zouden krijgen op gelijkaardige jongere. Tijdens de uitvoering zal de prioriteit bij elke in- of uitvoeroperatie verhoogd worden, terwijl, wanneer een overgang van de actieve naar de wachtende toestand opgelegd wordt door de scheduler, de prioriteit verlaagd wordt. Deze vorm van prioriteitsbeheer zorgt voor een optimaal gebruik van gans de computer door programma’s die in- en uitvoer doen te bevoordelen ten opzichte van deze die alleen maar rekenwerk verrichten.

3.3 Interactiviteit

Men kan twee verschillende soorten programma's onderscheiden qua interactie tussen het programma en de omgeving: *batch* en *interactieve* programma's. Deze laatste delen we in commandogebaseerde interactiviteit en GUIs.

3.3.1 Batch

Batch programma's hebben helemaal geen interactie met hun omgeving tijdens de uitvoering: alle gegevens worden op voorhand klaargemaakt en samen met het programma aan de computer aangeboden, terwijl de resultaten als een geheel, na het beëindigen van de uitvoering aan de gebruiker overgemaakt worden. Het precieze ogenblik waarop dergelijke programma's uitgevoerd worden, noch de uitvoeringssnelheid belangen de gebruiker echt aan.

Een programma dat elke maand, aan de hand van gegevens die door een ander programma verzameld werden, de overschrijvingen voor de wedden van bedienden maakt, zou als een typisch voorbeeld kunnen geciteerd worden.

De situatie is gans anders bij *interactieve programma's*: zij zijn gekenmerkt door een permanente dialoog tussen het programma en zijn omgeving. Gegevens die ingevoerd worden zijn dikwijls zelf functie van resultaten die enkele ogenblikken eerder door het programma zelf bepaald werden. Tengevolge deze dialoog hebben ogenblik en snelheid van uitvoering wel veel belang.

3.3.2 Command line

Ter illustratie toont fig.3.3 een opeenvolging van commando's voor een interactieve command interpreter (*cmd.exe*, de command interpreter van MS/DOS, de voorloper van Windows). Het voorbeeld bestaat uit het overschrijven van 5 hoofdstukken van een cursus, die elk op een afzonderlijk bestand staan, op een nieuw bestand dat daarna afgedrukt wordt.

```
12:21:19.26 C> copy chap1.txt list.txt
      1 File(s) copied

12:22:45.77 C>copy list.txt +chap2.txt
LIST.TXT
CHAP2.TXT
      1 File(s) copied

12:23:05.76 C> copy list.txt +chap3.txt
LIST.TXT
CHAP3.TXT
      1 File(s) copied

12:23:20.81 C>  copy list.txt +chap4.txt
LIST.TXT
CHAP4.TXT
      1 File(s) copied

12:23:27.68 C> copy list.txt +chap5.txt
LIST.TXT
CHAP5.TXT
      1 File(s) copied

12:23:34.05 C>  print list.txt

   C:\LIST.TXT is currently being printed

12:24:20.13 C>
```

Fig. 3.3 Example of an interactive MS/DOS session.

In het voorbeeld meldt de *command interpreter* aan de gebruiker dat hij een commando verwacht door **het uur** gevolgd door **C>** op het scherm te doen verschijnen.
De gebruiker geeft het bevel (bevelen zijn hier voor duidelijkheid in *schuinschrift* weergegeven) het bestand *chap1.txt* over te schrijven op het bestand *list.txt* door middel van het programma *copy*. Dit programma meldt aan de gebruiker dat de opdracht uitgevoerd is en stopt. De *command interpreter* meldt dan dat hij een nieuwe opdracht kan aanvaarden.
De gebruiker beveelt dat hetzelfde programma *copy* het bestand *chap2.txt* zou toevoegen aan het reeds bestaande bestand *list.txt* (het + teken in het bevel betekent dat de inhoud van *chap2.txt* moet toegevoegd worden aan de bestaande inhoud van *list.txt*).
Deze opeenvolging van bevelen en uitvoeringen wordt verder gezet tot de 5 bestanden achter elkander overgeschreven zijn en het programma *print* begonnen is met het afdrukken van het resulterende bestand (het programma *print* kan als het ware in de achtergrond verder werken terwijl de gebruiker iets anders doet).

Een batchprogramma is niets anders dan een lijst van zulke commando's.

3.3.3 GUI

De GUI is alombekend bij de moderne computergebruiker. Anderzijds geven smart phones en tablets nieuwe mogelijkheden qua interactiviteit die niet mogelijk zijn met een PC of laptop.