De store wordt alleen uitgevoerd indien register r3=1 (d.w.z. de waarde ‘true’ bevat).

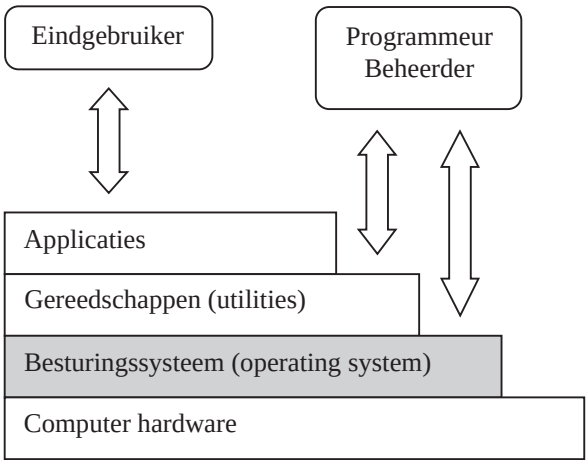
1.7 Operating systems

Algemeen

De computerhardware en de aangesloten randapparatuur worden beheerd door een softwarelaag die het *besturingssysteem* (*operating system*) genoemd wordt. Het besturingssysteem is verantwoordelijk voor efficiënt en eerlijk gebruik van de mogelijkheden van de hardware: in de eerste plaats voor de gebruiker(s) van applicatieprogramma’s die door de computer worden uitgevoerd, in de tweede plaats voor het ondersteunen van de programmeur(s) bij het maken, in gebruik nemen en onderhouden van applicaties, en ten slotte voor het algemene beheer van het computersysteem (zie figuur 1.27).

Een besturingssysteem is doorgaans gelaagd en modulair opgezet. De onderste laag zorgt voor abstractie van de computerhardware en randapparatuur d.m.v. bijvoorbeeld *device drivers*: kleine modules die de details van specifieke hardware en randapparatuur afschermen voor de programmeur achter een gestandaardiseerde interface. Hogere lagen van het besturingssysteem implementeren allerlei services zoals procesbeheer, geheugenbeheer, bestandsbeheer, communicatie en beveiliging. De gelaagdheid zit ’m erin dat de services gebruik maken van de functies van de onderliggende lagen. Bepaalde functies van een besturingssysteem worden aangeboden aan programmeurs en kunnen in applicatieprogramma’s toegepast worden. Deze functies maken onderdeel uit van een standaard interface ook wel *API* genoemd, *Application Programming Interface*. Al deze functies zijn beschikbaar via programmabibliotheken (library’s) (zie ook subparagraaf 1.6). Om het beheer, administratie, programmaontwikkeling e.d. goed te kunnen faciliteren bevat het besturingssysteem ook een grote hoeveelheid aan gereedschappen (utilities).

Besturingssystemen maken veelal onderscheid tussen de kernel en bovenliggende subsystemen. De *kernel* is een afgeschermd deel dat onder alle omstandigheden moet blijven functioneren, alleen via interrupts en specifiek omschreven systemcalls toegankelijk is en volledige controle over de hardware heeft. De subsystemen bieden een standaard systeemomgeving voor het uitvoeren van applicaties op basis van de functionaliteit die de kernel verschaft. Het gebruik daarvan is niet afgeschermd t.o.v. de applicatie zelf.



Figuur 1.28 Positionering van het besturingssysteem

Geschiedenis

Begrippen vanuit historisch oogpunt en ontwikkeling

De eerste computers hadden geen besturingssysteem. Programma's, ook wel *jobs* genoemd, werden direct ingelezen in het geheugen bijvoorbeeld d.m.v. een ponskaartlezer, met diverse lampjes en schakelaars kon men de programma's starten en in de gaten houden, en de uitvoer kwam direct op een printer. Dit wordt *serial processing* genoemd.

Veel dure computertijd werd verkwist met het handmatig wisselen tussen jobs. Daarom werd er een speciaal programma ontwikkeld, de monitor, die meerdere jobs automatisch achterelkaar in één batch kon executeren. Zo'n *monitorprogramma* dat permanent in het interne geheugen bleef (en daarom ook als resident monitor aangeduid werd) kan als voorloper van de huidige besturingssystemen beschouwd worden: het bevatte rudimentaire vormen van device drivers en andere voor algemeen gebruik beschikbare functies en beveiliging (bijvoorbeeld jobs mogen het geheugenbereik van de monitor niet veranderen). De werkwijze waarbij een reeks jobs zonder tussenkomst van gebruikers worden uitgevoerd wordt *batch processing* genoemd.

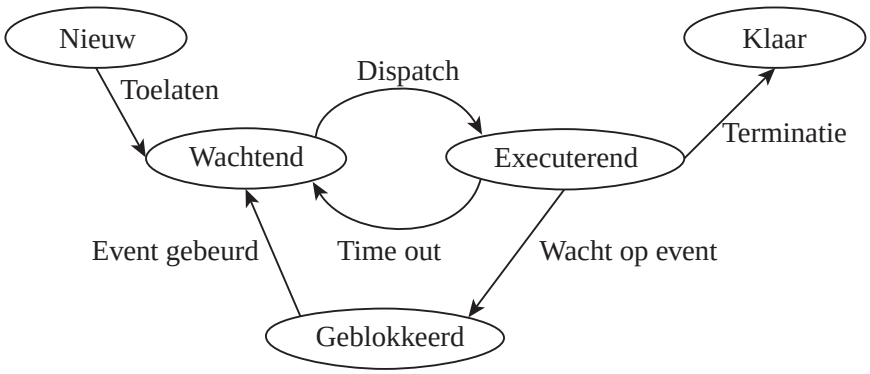
Met batchverwerking werd nog steeds dure computertijd verspild, omdat randapparatuur vergeleken met de CPU langzaam is. De CPU moet in principe wachten tot een I/O opdracht van een programma afgehandeld is. Het idee achter multiprogramming is dat gedurende de tijd dat de ene job moet wachten op I/O, een andere job kan executeren. Meerdere jobs zijn dan in het geheugen opgeslagen en verkeren tegelijkertijd in staat van executie.

Multiprogrammering bedoeld om de CPU optimaal te benutten gaat vloeiend over in het tegenwoordig meer gangbare begrip *multitasking*: het als het ware tegelijkertijd uitvoeren van meerdere applicaties. Door het toenemende aantal gebruikers van een computer, werd het ook steeds belangrijker dat de computertijd eerlijk verdeeld werd. Daarom is het multitasking van huidige besturingssystemen

ook op *timesharing* gebaseerd, d.w.z. alle jobs krijgen per toerbeurt kort de gelegenheid om een stukje verder te executeren. Welke job wanneer aan beurt is, wordt ook wel *scheduling* genoemd. Met de intrede van multitasking, deden ook andere services hun intrede, zoals geheugenbeheer, interactieve programmaverwerking en spooling. Immers, als meerdere jobs tegelijk executeren, dan moeten ze ook het geheugen delen en moet worden voorkomen dat de ene job het geheugenbereik van de andere verandert. *Interactieve programmaverwerking* houdt in dat meerdere gebruikers via eigen 'terminals' onafhankelijk van elkaar taken kunnen laten uitvoeren. Tijdens executie is zelfs interactie met de taak mogelijk d.m.v. terminal i/o. Het gelijktijdig kunnen bedienen van meerdere gebruikers dankzij timesharing is hierbij cruciaal, een reden waarom interactieve multiuser systemen vaak als timesharing systemen werden aangeduid. Spooling is het in goede banen leiden van gedeeld gebruik van randapparatuur. Zonder spooling zou bijvoorbeeld de uitvoer van twee jobs door elkaar uit de printer kunnen komen.

Naast deze ontwikkelingen rond programma-executie, hebben ook andere ontwikkelingen invloed gehad op het ontwerp van de huidige moderne besturingssystemen. Met het goedkoper worden van computers, veranderde ook de organisatie van een centrale krachtige 'mainframe' computer met terminals, naar clusters van meerdere personal computers of workstations verbonden in een netwerk. Die ontwikkeling creëerde de behoefte aan ondersteuning bij het uitvoeren van taken of services op andere computers, het kopiëren van bestanden over het netwerk, etc. Programma's voor dergelijke onmisbare services worden tegenwoordig als deel van het besturingssysteem gezien. Tenslotte heeft de overgang van tekstuele, commando-geörienteerde bediening van computers naar grafische schillen of GUI's (graphical user interface) ook zijn weerslag gehad op huidige besturingssystemen.

Taken besturingssysteem
Hier worden de meest belangrijke taken van het besturingssysteem behandeld. Voor functionaliteit gerelateerd aan netwerken, verwijzen we naar par. 3 *Computernetwerken*.



Figuur 1.29. Procestoestandsmodel

Procesbeheer
De taak van het procesbeheer is om het executeren van meerdere programma's op een computer op een ordentelijke en efficiënte manier te laten verlopen. Elk programma dat draait wordt een *proces* genoemd. Afhankelijk van het aantal CPU's kan één of een beperkt aantal processen tegelijkertijd verwerkt worden, de overige processen wachten in een queue. Het besturingssysteem kiest er steeds één uit om voor een bepaalde maximale duur op een CPU te laten draaien. Dit heet *dispatching*. Komt het proces in een toestand waarbij het moet wachten op een bepaald event, zoals de afhandeling van een I/O operatie, dan wordt het proces tot het optreden van het event geblokkeerd. Als de toegekende verwerkingstijd wordt overschreden dat treedt een time-out op en wordt van proces gewisseld. Als het proces klaar is, d.w.z. het uitgevoerde programma eindigt, dan wordt het proces getermineerd. Een proces kan zich daarom grofweg in een vijftal toestanden bevinden (zie figuur 1.29). Bestaande besturingssystemen hebben vaak meer toestanden, maar zijn in principe op deze vijf toestanden gebaseerd. Om tussen de verschillende processen te kunnen wisselen, moet van een onderbroken proces een grote hoeveelheid informatie bijgehouden worden: de inhoud van de CPU-registers, administratieve informatie van het proces, en process control information, zoals de toestand van het proces, gebruik van resources, privileges, prioriteit, etc. Tijdens executie kan een proces een nieuw proces starten dat een gegeven programma gaat uitvoeren. Dit wordt *spawning* genoemd. Het nieuwe proces heet het kind (child) en het oorspronkelijke de ouder (parent). Beide processen zijn in principe gereed om verder te executeren.
Moderne besturingssystemen kennen ook vaak meerdere modes waarin processen executeren, onder meer user mode en *kernel mode* (ook wel system of control mode genoemd). De laatste geeft volledige toegang tot alle instructies van de processor, al het geheugen, etc. Het is speciaal bedoeld voor processen van het besturingssysteem zelf en van device drivers. Op deze manier wordt de stabiliteit van het besturingssysteem gewaarborgd tegen mogelijk desastreuze fouten en acties van applicatiesoftware die in user mode executeert.

Zoals hier beschreven vormen processen gescheiden eenheden wat betreft resourcetoewijzing van geheugen en CPU-tijd. Het blijkt dat het wisselen tussen processen een redelijk dure operatie is waar zelf veel tijd in gaat zitten. *Multithreading* maakt het mogelijk om per proces meerdere lichtgewicht subprocessen te hebben, *threads* genaamd, die ogenschijnlijk parallel executeren maar de resources van het proces delen (o.a. de programmacode en globale data). Het creëren, termineren en wisselen van threads kost minder tijd en communicatie tussen threads is makkelijker, omdat ze bijvoorbeeld in eenzelfde stukje geheugen kunnen lezen en schrijven. In applicaties met een GUI wordt schermafhandeling bijvoorbeeld vaak door een individuele thread afgehandeld.

Bij multithreading, maar ook in het algemeen bij parallelle executie van programmacode op een of meerdere computers, kunnen er ongewenste effecten optreden als threads of processen niet onderling gesynchroniseerd worden ten aanzien van bewerkingen op gemeenschappelijk geheugen of gedeelde resources. Race condities tussen processen bij het uitvoeren van onderling interfererende operaties kunnen leiden tot inconsistente resultaten. Om dergelijke situaties te voorkomen, is het belangrijk in programmacode kritieke secties (critical sections) te onderkennen en ervoor te zorgen dat nooit twee threads of processen zich in een kritieke sectie bevinden die dezelfde gedeelde resources benaderen (*mutual exclusion*). Besturingssystemen voorzien in mechanismen zoals semaforen en monitors om threads en processen onder bepaalde condities tijdelijk te blokkeren. Het toepassen van synchronisatiemechanismen moet zorgvuldig gebeuren om niet weer andere problemen te veroorzaken zoals deadlock en starvation. *Deadlock* ontstaat wanneer meerdere threads of processen niet verder kunnen, omdat ze cyclisch op elkaar wachten, bijvoorbeeld omdat ze alle geblokkeerd zijn vanwege toegang op een gedeelde resource. *Starvation* is de situatie waarbij een thread of proces nooit voor dispatch gekozen wordt en dus nooit de toestand executerend bereikt.

Geheugenbeheer
Waar procesbeheer het gebruik van de processor op een ordentelijke, eerlijke en efficiënte manier verdeelt over de processen, is het doel van geheugenbeheer om hetzelfde te bewerkstelligen voor het gebruik van het interne geheugen van de computer. Als een programma een hoeveelheid geheugen nodig heeft voor berekeningen en tijdelijke opslag van gegevens, dan dient het programma een geheugenverzoek in bij het besturingssysteem. Die kiest een ongebruikt deel van het geheugen uit en kent die aan het programma toe (*allocatie*). Programmacode die geëxecuteerd moet worden, moet zelf ook in het geheugen staan. Daarvoor wordt op een soortgelijke manier geheugen toegekend. Het besturingssysteem zorgt ook voor bescherming van de toegekende stukken geheugen om bijvoorbeeld te voorkomen dat ze overschreven kunnen worden door andere programma's. Het is overigens ook mogelijk dat verschillende processen op verzoek een stuk geheugen delen.

Het geheugen van een computer is uiteindelijk een lange rij geheugencellen waarbij elke cel een uniek adres heeft. Het toekennen van een stuk geheugen bestaat dus uit het toekennen van een bepaald adresbereik. Programma's worden daarom zo gecompileerd dat ze nooit afhankelijk zijn van een absoluut geheugenadres, maar dat programma en data in een willekeurig toegekende plek in het geheugen geplaatst kunnen worden.
Het is zelfs zo dat als een programma tijdens executie zich in toestand 'wachtend' of 'geblokkeerd' bevindt (zie figuur 1.29), programma en bijbehorende gegevens tijdelijk naar de harde schijf geschreven kunnen worden, zodat een ander programma tussendoor

even dat deel van het geheugen kan gebruiken. Dit wordt *swapping* genoemd.
Een meer geavanceerde techniek voor het wisselend gebruik van intern geheugen met een harde schijf als additioneel achtergrondgeheugen is *virtual memory*. Hierbij vormt de harde schijf een 'overloopgebied' van het interne geheugen en kan het besturingssysteem adresbereiken toekennen voor geheugen dat er eigenlijk niet is. (Processoren met een 32-bit architectuur hebben typisch een virtueel adresbereik van 232 bytes = 4 Gb.) De essentie van virtueel geheugen is dat alleen actief gebruikte delen van dit virtuele adresbereik in het interne geheugen geladen worden. Als een programma een 'niet-actief? geheugenstuk benadert, dan wordt ongemerkt ergens geheugen vrijgemaakt en het betreffende stuk geheugen wordt van de harde schijf ingelezen. Dit gaat meestal in blokken van enkele kilobytes (pagina's) en wordt ondersteund door speciale adresafbeeldingshardware in de processor (memory management unit). Er bestaan efficiënte methoden voor het vrijmaken van niet meer actief gebruikt intern geheugen zoals *Least Recently Used* (de pagina die de langste tijd niet gebruikt is, maakt plaats voor de nieuwe pagina) en het tijdig ophalen van actieve componenten d.m.v. *prefetching* (als de kans groot is dat een bepaalde pagina benaderd gaat worden, dan wordt het inlezen alvast in gang gezet).

Bestandsbeheer
Waar geheugenbeheer het interne geheugen beheert, gaat het bij bestandsbeheer om het secundaire geheugen zoals floppy's, CD's, DVD's en harde schijven. Bestanden (files) zijn bedoeld voor opslag van gegevens voor de lange termijn en verdwijnen niet als de gebruiker uitlogt of de computer uitgezet wordt. Om het beheer en terugvinden van bestanden te vergemakkelijken zijn bestanden georganiseerd in *directory's* (ook wel mappen of folders genoemd). Een directory kan zowel bestanden als ook andere directory's bevatten (subdirectory's).
Het besturingssysteem heeft een soortgelijk allocatiemechanisme om de beschikbare ruimte aan de bestanden toe te kennen als bij geheugenbeheer. Bij bestandsbeheer gaat dit echter meestal in blokken. Er bestaan diverse *file systems*, standaarden voor administratie en organisatie van bestanden. Enkele veelgebruikte zijn:
– FAT16/32: File system van Microsoft uit de MSDOS-tijd, dat nog steeds voor floppy's en (kleinere) harde schijven gebruikt wordt.
– NTFS: Opvolger van FAT32 met verbeterde recoverability (herstellen van bestanden na een computer- of diskfout), beveiliging en efficiënt beheer van grote schijven.
– Ext3: Een van de meestgebruikte file systems voor Linux.

Invoer en uitvoer
De randapparatuur van een computer communiceert ofwel per karakter (zoals een toetsenbord, muis of modem) of per blok (zoals

een harde schijf). Beide worden op aparte wijze behandeld. Als er bijvoorbeeld een toets wordt ingedrukt, dan genereert dit een zogenaamde interrupt in de processor. Het draaiende programma wordt even onderbroken om de bijbehorende actie te kunnen ondernemen.
Bij blokgeoriënteerde randapparatuur is doorvoersnelheid van gegevens vaak van groot belang. Speciale hardware zorgt ervoor dat beschikbaar komende blokken direct in een vooraf toegekend geheugenbereik geschreven worden zonder de processor hiervoor te hoeven onderbreken. Dit wordt *Direct Memory Access* (*DMA*) genoemd.
Bij harde schijven kan een andere techniek genaamd *Redundant Array of Independent Disks* (*RAID*) voor nog meer snelheid en betrouwbaarheid zorgen als meerdere harde schijven tegelijk worden ingezet. RAID is een combinatie van meerdere deels onafhankelijke technieken onderverdeeld in zogenaamde RAID niveau's. De techniek van RAID niveau 0 is striping: de blokken van een bestand worden verdeeld over de harde schijven, zodat het lezen van meerdere blokken achterelkaar door de individuele harde schijven tegelijkertijd plaatsvindt en zo voor nog meer doorvoersnelheid zorgt. De techniek van RAID 1 is mirroring: de blokken van een bestand worden meerdere keren op verschillende harde schijven opgeslagen, zodat vooral meer betrouwbaarheid verkregen wordt. Een harde schijf kan dan immers zonder nare consequenties uitvallen.

Beveiliging
Zoals hierboven al meerdere keren naar voren is gekomen, is het besturingssysteem ook voor diverse vormen van beveiliging verantwoordelijk. Het moet vertrouwelijkheid, integriteit, beschikbaarheid en authenticiteit van informatie, computer en gebruikers waarborgen. Het besturingssyteem kent daarom de notie van gebruiker (user) die zich moet identificeren m.b.v. een wachtwoord (password). Bestanden, processen, programma's zijn allemaal 'eigendom' van een gebruiker. Moderne besturingssystemen kennen uitgebreide faciliteiten (access control lists, capability lists) om gebruikers en hun processen rechten of privileges te geven of ontnemen, als ook voor gebruikers om andere gebruikers bepaalde rechten te geven voor hun bestanden en programma's. Op deze manier wordt een gebruiker beschermd voor eventuele opzettelijke en niet opzettelijke acties van andere gebruikers of hun programma's.
De risico's en gevaren zijn daarmee echter niet geweken. Vooral bescherming voor opzettelijke acties zijn moeilijk te beschermen. Mogelijke soorten aanvallen zijn: zich proberen voor te doen als een andere gebruiker om zo diens rechten te kunnen benutten, afluisteren van netwerken, netwerkberichten modificeren of opnieuw verzenden, zich onrechtmatig toegang verschaffen tot bestanden, beschikbaarheid aantasten (denial of service attack) door bijvoorbeeld te pogen een computer te overbelasten, etc. Er is een soort

oorlog gaande tussen slimme aanvallers die zwakheden in besturingssysteemfuncties benutten en verbeteringen aan het besturingssysteem als verdediging daartegen.

Voorbeelden besturingssystemen

Windows

Windows staat voor een reeks van besturingssystemen die in de loop der tijd door Microsoft zijn uitgebracht als opvolgers van het eens zo populaire MS-DOS. Bekende versies in deze reeks zijn: Windows 3.1, Windows 95/98, Windows NT, Windows 2000 en Windows XP. Met het bieden van een grote mate van gebruiksgemak ('look-and-feel' en 'plug-and-play') en compatibiliteit tussen opvolgende versies heeft Microsoft zijn dominante positie op het gebied van desktop-besturingsystemen voor PC's steeds weten te behouden. De overweldigende hoeveelheid beschikbare software voor het Windows-platform versterkt deze positie.

Tot en met Windows 98 waren de verschillende Windows-versies in feite afgeleid van het single-user besturingssystem MS-DOS. Met Windows NT is een nieuwe weg ingeslagen (NT staat voor New Technology): een geheel nieuwe versie werd ontwikkeld op basis van de gangbare technieken voor multi-user / multi-tasking besturingssystemen. Dit heeft het betrouwbaar functioneren van Windows en de bruikbaarheid van Windows als server-besturingssysteem aanzienlijk verbeterd. Daarnaast is veel aandacht besteed aan portabiliteit en uitbreidbaarheid. Windows XP heeft een gelaagde, client-server architectuur die meerdere API's en typen subsystemen ondersteunt (virtual DOS machine, Posix, Win32). De kernel is object-georienteerd opgezet: kernel-data ligt opgeslagen in kernel-objects, terwijl daarop uitgevoerde methods de kernel-activiteiten vormen. In de kernel worden objecten bijgehouden voor bijvoorbeeld processen, threads, events, semaforen, files, enz. Deze worden beheerd door objectmanagers. Als een thread een file wil benaderen zal de open method van de file objectmanager aangeroepen worden om een referentie naar het file-object te krijgen. Windows bevat objectmanagers voor het beheer van allerlei systeemcomponenten, zoals een virtual memory manager, I/O managers en plugmanagers.

Linux

Linux is een besturingssysteem in de familie van UNIX-like operating systems. De oorspronkelijke UNIX-systemen zijn ontwikkeld bij Bell-labs voor minicomputers vanaf de zestiger-jaren in de vorige eeuw. De open benadering en vrije beschikbaarheid van de broncode hebben gezorgd dat UNIX-systemen zich steeds goed aan nieuwe hardware-ontwikkelingen hebben aangepast terwijl de originele en efficiënte basisconcepten van UNIX behouden bleven. Linux is geheel opnieuw ontworpen (door de Finse student Linus Torvalds) met het oogmerk om zo veel als mogelijk standaard UNIX-applicaties te kunnen draaien. Opnieuw, dankzij de vrije beschikbaarheid op internet vanaf de eerste versie in 1991, zijn de huidige Linux-systemen het resultaat van collectieve bijdragen van

een grote internationale Linux-gemeenschap. Er bestaan veel verschillende distributies (o.a. Red Hat, Debian, SuSE, Fedora) met de Linux-kernel als gemeenschappelijk basis.
Opvallend in Linux is de implementatie van processen en threads. Naast de beschikbaarheid van een 'user-level thread package' worden threads ook volwaardig door de kernel ondersteund op dezelfde wijze als (single-threaded) processen. Het enige verschil is dat threads de gemeenschappelijke context delen van het (ouder)-proces waaruit ze door 'spawning' zijn ontstaan. Linux kent een system call 'clone' als generalisatie van de system call 'fork'. Bij een *fork* ontstaat een onafhankelijk (kind)proces, bij een *clone* kan precies worden aangegeven welke componenten van de procescontext met het ouderproces gedeeld worden.
Linux heeft zich bewezen als een stabiele en betrouwbare UNIX-variant. Dankzij de 'UNIX-eigenschappen', 'open software benadering' en gratis beschikbaarheid heeft Linux een aanzienlijke positie verworven als besturingssysteem voor server-systemen. Als desktop-besturingssysteem wordt Linux vanwege zijn technische kwaliteiten vooral door specialisten gebruikt, maar is door zijn afwijkende eigenschappen ten opzichte van de 'Windows standaard' (nog) niet breed ingeburgerd.

Andere besturingsystemen
Windows en Linux zijn favoriete besturingssystemen voor general purpose toepassingen op Intel-PC's. Daarnaast bestaan er vele andere meer of minder verwante besturingssystemen, bijvoorbeeld voor computer systemen met 'eigen' hardware-architectuur (bijvoorbeeld Mac-OS voor Apple computers), maar ook voor specifieke toepassingen zoals realtime systemen (RTOS, VXworks). Veel leveranciers van servers en werkstations (SUN, HP, IBM) bieden een eigen UNIX-variant aan als standaard besturingsysteem (Solaris, HP-UX, AIX). Ook 'klassieke' besturingssystemen (MVS, VMS, MS-DOS) doen nog steeds dienst op oudere typen hardware platforms omdat het 'porten' van de toepassingssoftware niet rendabel is.
Een belangrijk nieuw toepassingsgebied voor besturingsystemen vormen handheld devices zoals mobiele telefoons, PDA's, spelcomputers, en meer in het algemeen embedded systems. Besturingssystemen die specifiek voor dergelijke apparaten zijn ontwikkeld (PALM-OS, Symbian) concurreren met aangepaste versies van Windows (Windows-CE) en Linux.

Ontwikkelingen
Besturingsystemen moeten voortdurend aangepast worden aan nieuwe hardware, randapparaten, protocollen, standaarden en toepassingen. Tegelijkertijd moeten bestaande conventies, interfaces en toepassingssoftware over langere tijd ondersteund blijven.
Omvang en complexiteit van een besturingssysteem neemt daardoor voortdurend toe. Veel ontwikkelingen zijn dan ook gericht op het efficiënt en beheersbaar houden van besturingssystemen.

Een van de doelen daarbij is de basis van een besturingssysteem klein en universeel te houden en alle configuratie- of toepassingsafhankelijke componenten zoveel mogelijk als losse modulen toe te voegen. Dit staat bekend als de *microkernel benadering*. Besturingssystemen detecteren wijzigingen in de systeemconfiguratie en passen zich automatisch aan. 'Administrators' hebben in het algemeen de mogelijkheid om drivers voor specifieke hardware apparaten dynamisch in de kernel te laden of weer te verwijderen.
Om de afhankelijkheden tussen de hardware, het besturingssysteem en toepassingen te verminderen wordt gebruik gemaakt van universele interfaces tussen systeemlagen die specifieke systeemeigenschappen verhullen. Zo'n interface creëert als het ware een *virtual machine* met algemenere functionaliteit. Het virtual machine concept is een terugkerend verschijnsel in de ontwikkeling van besturingssystemen teneinde verschillen te overbruggen. In de tijd van de IBM mainframes werd het ooit toegepast om timesharing en batchverwerking op één op platform te verenigen. Nu wordt dit concept weer gehanteerd om Windows- en Linux-applicaties gelijktijdig op een systeem te draaien (VM-ware). Op vergelijkbare wijze ondersteunen besturingsystemen de Java Virtual Machine (JVM) interface als wereldwijd machine-onafhankelijk platform voor webapplicaties.

1.8 HPC

HPC staat voor *High Performance Computing*. High-Performance Computing houdt zich bezig met zeer grootschalige, rekenintensieve berekeningen, gewoonlijk in de technisch-wetenschappelijke sfeer waarbij conventionele computers tekort schieten in rekenkracht. Om toch de vereiste rekenkracht te mobiliseren moet de bouw (ofwel architectuur) van de computersystemen worden aangepast om die extra hoge prestaties te kunnen leveren. We bespreken hier de mogelijkheden om de snelheid van computersystemen op te voeren tot boven die van conventionele systemen en we zullen de beperkingen aangeven die daarbij optreden. Daarvoor moeten we ingaan op de architectuur van HPC systemen en laten we de meest voorkomende typen de revue passeren. Een kernbegrip voor HPC systemen is de *theoretische piekprestatie* (Engels: theoretical peak performance). De *theoretical peak performance* is het aantal drijvende kommabewerkingen die per seconde maximaal door het systeem kunnen worden uitgevoerd. Ook de werkelijke snelheid, die meestal veel lager ligt dan de theoretische piekprestatie, wordt gegeven door het aantal drijvende kommabewerkingen per seconde. De eenheid daarvoor is flop/s (floating-point operation/s). De huidige processoren zijn zo snel dat in plaats van flop/s, Mflop/s, Gflop/s of Tflop/s als snelheidsmaten gebruikt worden, waarbij het voorvoegsel M staat voor Mega (miljoen), G voor Giga (miljard) en T voor Tera (biljoen).