1 INLEIDING OPERATING SYSTEMS

Wanneer een computersysteem wordt aangezet, zal het voor gebruikers niet toegankelijk zijn als het systeem niet met een programma in de juiste banen wordt geleid. Zo'n programma dat het computersysteem laat werken, noemt men een besturingssysteem of een operating system. In eerste instantie komt de CPU na een power-up of een reset in een programma terecht dat in ROM is opgeslagen. Dit programma is een hulpprogramma om het eigenlijke besturingsprogramma of operating system binnen te halen. De code in ROM noemt men bootstrapprogramma. Vaak haalt de ROM-bootstrap een wat meer uitgebreid bootstrapprogramma van disk. Deze tweede of secondary bootstrap haalt dan het operating system binnen. Het besturingsprogramma, dat op deze wijze met het bootstrapprogramma van de disk wordt gehaald, zal ervoor zorgen dat de computer toegankelijk wordt voor de gebruiker(s) en zich op een voorgeschreven manier zal gaan gedragen.

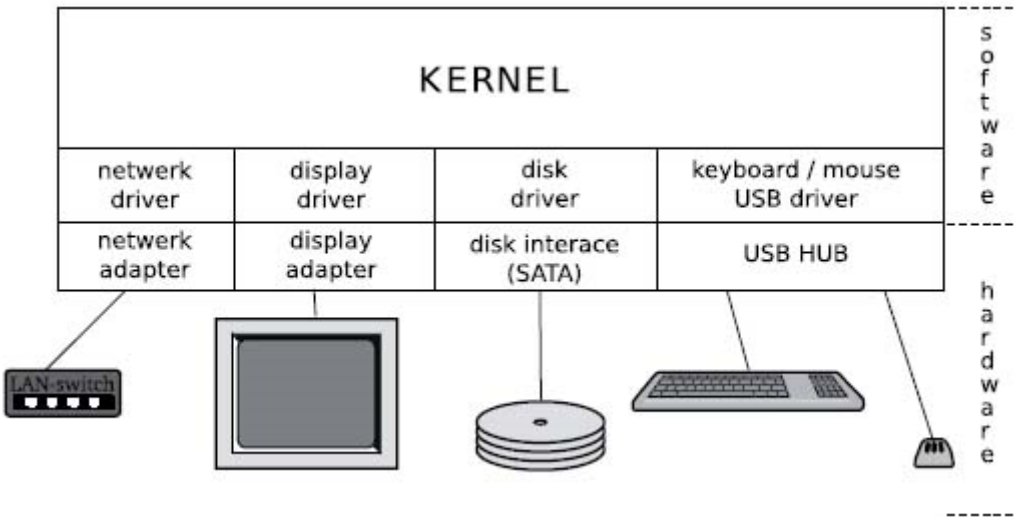
1.1 Taken van het operating system

Een operating system of OS heeft de volgende taken bij het functioneren van een computer:

- Het regelt de communicatie met de randapparaten (disk, netwerk, user-I/O).
- Het verdeelt, regelt en organiseert de systeemresources (CPU-tijd, geheugen, diskruimte).
- Het biedt de gebruiker de mogelijkheid op een hardwareonafhankelijke manier van het systeem gebruik te maken.

1.2 Kernel, device drivers

Het operating system bestaat uit een machineonafhankelijk deel en een machineafhankelijk deel. Het hardwareonafhankelijke deel noemt men de kernel. Omdat het operating system met randapparatuur moet communiceren, worden aan deze kernel stukken programma toegevoegd die op een voorgeschreven manier de communicatie met deze randapparaten of devices regelen. Deze programma's noemt men device drivers. In onderstaande figuur is een operating system van een pc of werkstation schematisch weergegeven. De device drivers vormen de koppeling met de hardware waar de randapparatuur weer op aangesloten is.



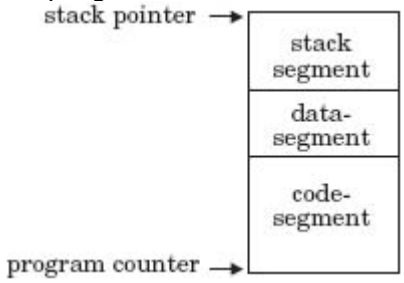
1.3 Process

Een gebruiker zal op een gegeven ogenblik zijn eigen applicatieprogramma op de computer gaan uitvoeren. Zo'n programma in uitvoering noemt men in OS-jargon een process. Een process ontstaat bij het starten van een programma en verdwijnt als het programma klaar is.

> *Wanneer men een teksteditor start, komt er een proces dat de gebruiker de mogelijkheid biedt een tekstbestand met bijvoorbeeld de sourcecode van een C-programma aan te maken. Als men de editor verlaat komt aan dit proces een einde.*
> *Een opgestarte compiler is een proces dat sourcecode (broncode) inleest van een bestand en machinecode (objectcode) produceert. Als de sourcecode vertaald is, zal dit proces automatisch stoppen. De door de compiler gemaakte code biedt de gebruiker de mogelijkheid een proces te starten dat handelingen verricht zoals hij die zelf in een hogere programmeertaal heeft gespecificeerd.*

Een proces is dus in vrijwel alle gevallen ooit als een programma in een hogere taal begonnen. In het computergeheugen bestaat zo'n proces dan uit machine-instructies (codesegment), een gedeelte waar de gegevens staan waarmee het proces werkt (datasegment) en een stuk geheugen dat gereserveerd is voor de stack, dus waar de stack pointer naar zit te wijzen (stacksegment). Het codegedeelte bestaat enkel en alleen uit machineinstructies en wordt door de CPU alleen gelezen. De nachtmerrie van elke informaticus is code die zichzelf verandert (self-modifying code). Het datasegment wordt gelezen en naar alle waarschijnlijkheid ook geschreven. Het stacksegment wordt gelezen en geschreven.

In onderstaande figuur is de beginpositie van de program counter en de stack pointer aangegeven. De program counter zal tijdens het runnen van dit proces adressen van instructies in het codesegment bevatten. Via de stack pointer worden allerlei tijdelijke gegevens, zoals returnadressen van subroutines, van en naar de stack getransporteerd. De programmavariabelen bevinden zich in het datasegment.

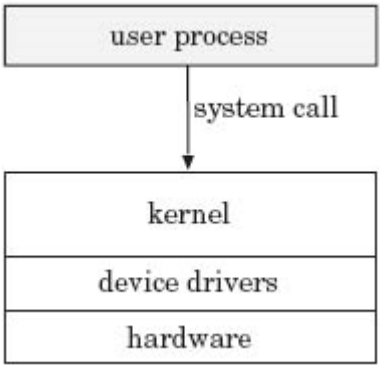


1.4 System calls

Een proces zal op een gegeven moment dingen willen doen die het zelf niet kan (of mag). Het lezen van een bestand van een disk is een taak die door het operating system moet worden uitgevoerd. Het operating system biedt processen de mogelijkheid dit soort zaken te doen. Processen doen een beroep op het operating system via zogenoemde system calls.

Een system call is de aanroep van een speciale routine uit het operating system om een bepaalde taak gedaan te krijgen. De wijze waarop dit in de praktijk meestal gaat, is gebruik te maken van speciale machine-instructies van de CPU, zogenoemde software interrupts. Een bepaalde set system calls voor de gebruiker karakteriseert een bepaald operating system. Een gebruikersproces of user process hoeft geen 'kennis' te hebben van de specifieke hardware. De system call levert een pasklare oplossing voor bijvoorbeeld het communiceren met een bepaald randapparaat. De hardwareverschillen

van twee computersystemen, die hetzelfde operating system draaien, zijn door het operating system afgeschermd. Door aan een kernel geschikte device drivers toe te voegen, is een operating system voor een computersysteem te realiseren.



Het OS met z'n system calls vormt een virtuele machine waar de user-processen gebruik van kunnen maken. De system calls vormen een 'uitbreiding' op de instructieset van de CPU waarmee processen de gelegenheid krijgen specifiek hardwareafhankelijke taken te verrichten.

Enkele system calls uit het Unix-OS zijn de volgende:

- creat maakt een nieuwe file (vreemd genoeg heet deze system call niet create);
- open opent een file;
- close sluit een file;
- pipe vormt een FIFO (first-in, first-out) communicatiekanaal tussen twee processen;
- fork maakt een nieuw proces;
- kill stuurt een signaal naar een proces.

Keren we nog even terug naar ons procesmodel, dan zal het codesegment machinecode bevatten, aangevuld met de software interrupt-instructies (die natuurlijk welbeschouwd ook machinecodes zijn) om het proces met het operating system te laten samenwerken. De compiler die de code voor dit proces gegenereerd heeft, zal dus kennis moeten hebben van de mogelijke system calls. De maker van de compiler heeft hierin meestal via zogeheten bibliotheekroutines (libraries) voorzien.

Samengevat is een operating system met onderliggende hardware als een virtuele machine op te vatten. Een executeerbaar bestand zal machinecodes bevatten, aangevuld met diensten van het operating system. Deze diensten worden meestal met een software interrupt aangeroepen. De basisset van machine-instructies van de CPU is als het ware verder uitgebreid met functies van een virtuele machine die door het OS gevormd wordt.

1.5 Command interpreter

Een speciaal programma dat vaak samen met het operating system geleverd wordt en er vaak mee verward wordt, is de command interpreter. Dit programma zal als proces rechtstreeks met de gebruiker communiceren en is een 'laag' tussen de gebruiker en het operating system; de benaming shell die men in Unix gekozen heeft geeft dit duidelijk aan. De command interpreter zal commando's uitvoeren die de gebruiker via het toetsenbord ingeeft. Een command interpreter geeft de gebruiker de mogelijkheid op eenvoudige wijze zijn eigen programma's te starten. Zodra dit gebeurd is, zal de command interpreter zich pas weer melden als het opgestarte programma afgelopen is (een uitzondering hierop vormen de zogenoemde achtergrondprocessen; deze processen draaien tegelijk met de command interpreter).

Command interpreters vormen de schakel tussen de gebruiker achter het toetsenbord en het operating system. Let op: om tekst van het toetsenbord te kunnen lezen, maakt de command interpreter als proces gebruik van system calls. In moderne operating systems is de taak van de command interpreter meer en meer vervangen door een grafische schil die met een pointing device (muis) te besturen is. Binnen deze grafische schil is echter vaak weer een window te starten met een tekstgeoriënteerde command interpreter.

1.6 Typen operating systems

Een operating system dat één proces kan uitvoeren, heet een single-tasking operating system. Een operating system dat meer processen schijnbaar tegelijk kan uitvoeren, heet een multi-tasking operating system. Een multi-tasking operating system kan op zijn beurt weer single-user of multi-user zijn, afhankelijk van het feit of het systeem verscheidene gebruikers kan onderscheiden.

Naast deze indeling kan een operating system wel of niet realtime zijn. Realtime houdt in dat een operating system binnen zekere tijd op een bepaalde gebeurtenis van buiten het systeem zal reageren. Realtime is vaak noodzakelijk als er een industrieel proces bestuurd moet worden, of wanneer de computer in een meet- en regelsysteem is opgenomen.

1.7 Procesmanagement

We gaan eerst bestuderen hoe een operating system met processen omgaat. Hierbij komen begrippen als multi-tasking, scheduling en memory management (geheugenbeheer) om de hoek kijken.

Achtereenvolgens behandelen we:

- multitasking;
- memory management;
- user en kernel mode;
- procesmanagement bij beperkt geheugen;
- virtueel geheugen;
- threads;
- shared code;
- dynamic link libraries, shared libraries;

.

1.7.1 Multi-tasking

Bij een multi-tasking operating system zijn meer processen tegelijk actief. In werkelijkheid is er maar één proces echt actief (bij systemen met één CPU), maar doordat de CPU vaak van proces wisselt lijkt het alsof de processen parallel draaien. Het computergeheugen bevat meer processen die in verschillende toestanden kunnen zijn. We onderscheiden:

- running: het proces wordt daadwerkelijk uitgevoerd. In de praktijk is dat er vaak maar één (in een single-CPU-systeem);
- ready to run: het proces kan gaan draaien, maar is toevallig niet aan de beurt;
- waiting: het proces kan niet verder omdat het op een gebeurtenis (event) wacht. Zo'n event kan zijn het indrukken door de gebruiker van een toets op het keyboard of het binnenkomen van een diskblok.

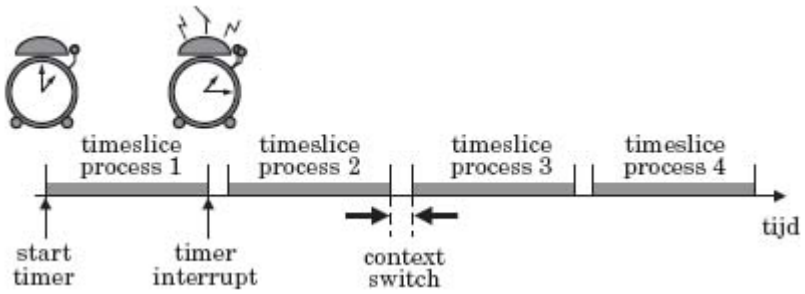
De kernel houdt procesgegevens in een tabel bij. Deze tabel noemen we de process table. Het gedeelte van de kernel dat een nieuw proces uitkiest voor executie (dus dit proces runable maakt) heet de scheduler. De scheduler kiest een volgend ready to run proces uit de process table.

We onderscheiden pre-emptive en non-pre-emptive schedulers:

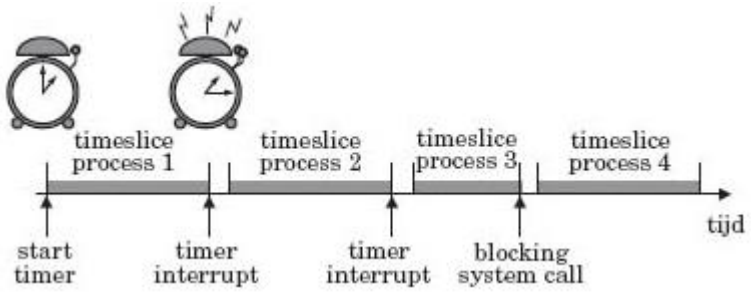
- Een pre-emptive scheduler zal bij een nieuwe run van een proces een hardwaretimer starten die na zekere tijd (een zogenoemde timeslice) een hardware interrupt genereert. Interrupt-afhandeling is het domein van de kernel en deze interrupt zal verder door de scheduler afgehandeld worden, die een nieuw proces kiest en deze een timeslice geeft. Een proces kan dus nooit het hele systeem voor zichzelf claimen, maar is afhankelijk van de door de scheduler toegekende timeslices.
- Een non-pre-emptive scheduler laat het aan het proces zelf over wanneer een volgend proces aan de beurt is. De macht is hier dus aan het proces met het gevaar dat een 'asociaal' proces vrolijk een ingewikkelde rekenpartij start en de CPU niet meer loslaat. Een tweede probleem dat zich hier kan voordoen, is een proces dat door een programmeerfout in een eindeloze lus raakt. De andere processen komen niet meer aan bod en het systeem is soms alleen nog maar met een reset tot de orde te roepen.

In beide gevallen geldt dat het uitvoeren van een system call een proces in de waitingtoestand kan brengen en de scheduler een ander proces kan kiezen.

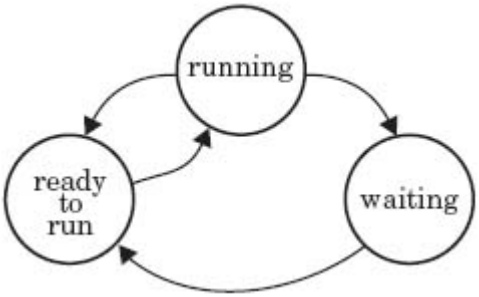
In onderstaande figuur is de werking van een operating system met pre-emptive scheduler in beeld gebracht. Bij het toekennen van een timeslice wordt de timer op scherp gezet. De timer genereert een interrupt en er treedt een context switch op. In de figuur is ook te zien dat een context switch even duurt. Het gevolg is dat men een timeslice niet te kort moet kiezen, omdat anders de overhead van de context switch in verhouding te groot wordt. Een timeslice mag ook weer niet te lang duren, want dan zal de gebruiker van een interactief proces (dit is een proces dat direct via toetsenbord en beeldscherm met de gebruiker communiceert) hiervan hinder ondervinden. Bij een tekstverwerker bijvoorbeeld lijkt het proces dan soms even 'dood' te zijn.



Onderstaand figuur laat ook een pre-emptive scheduler zien, maar nu geeft proces 3 zijn timeslice voortijdig op door een blocking system call uit te voeren. Zo'n blocking system call, zoals het wachten op een input character, brengt een proces in de wait-toestand. Het is dan niet zinvol nog meer tijd aan dit proces te verspillen. Pas als de input er is, wordt het proces weer 'ready to run' gemaakt.



Als we de toestanden waarin een proces kan verkeren in een toestandsdiagram zetten en ook de mogelijke overgangen tussen toestanden aangeven, krijgen we het plaatje van onderstaande figuur.



Een waiting-proces wacht op een gebeurtenis, ook wel 'event' genoemd. Het binnenkomen van een character via het toetsenbord, het binnenkomen van data via een netwerkwerk of vanaf een disk zijn allemaal events. Deze events gaan gepaard met het optreden van een interrupt. In de interrupt-afhandelingsroutine wordt, naast het afhandelen van de interrumperende hardware, ook gekeken welk proces op deze interrupt zat te wachten. Deze informatie is ook weer in de procesadministratie van het operating system te vinden. Op deze wijze wordt dan duidelijk welk proces weer 'ready to run' wordt.

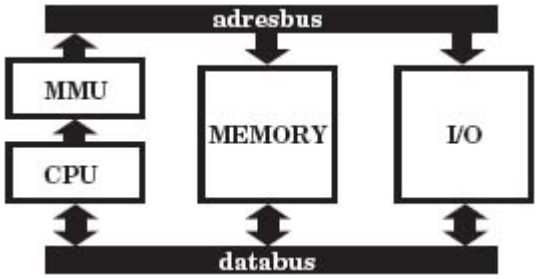
Om de vergelijking compleet te maken is in onderstaande figuur de werking van een non-preemptive scheduler weergegeven. Een proces doet op een geschikt moment een 'next-process' system call om een ander proces gelegenheid te geven de CPU-kracht te benutten. Zoals uit de figuur blijkt, laat het ene proces dat eerder toe dan het andere proces.

Het opslaan van de machinecode van de processen in het geheugen van de computer zal in eerste instantie bij kleine processen geen problemen geven. Wat wel een probleem is, is dat elk proces pas goed kan draaien als het zich op het juiste geheugenadres bevindt. Een compiler voor bijvoorbeeld C (een programmeertaal) maakt van C-code (source-code) machinecode (objectcode). In deze objectcode zitten meestal verwijzingen naar bepaalde geheugenadressen; bijvoorbeeld de aanroep van een subroutine in machinetaal is meestal een bepaalde machine-instructie gevolgd door het adres van de geheugenplek waar de subroutine begint. De meeste compilers vertalen de sourcecode naar machinecode die op een vast adres moet beginnen. Om echt multi-tasking te zijn, zou een operating system het actieve proces eerst naar de goede plaats kunnen kopiëren en het dan pas gaan executeren. Dit is echter een tijdrovende zaak en er zijn in de hardware beter oplossingen mogelijk.

Alvorens deze oplossing te bespreken, noemen we eerst nog de soms toegepaste softwareoplossing waarbij men zogenoemde relocatable code gebruikt. Een aantal CPU's biedt met hun adresseermodes de mogelijkheid om positieonafhankelijke machinecode te schrijven (dit 'schrijven' gebeurt natuurlijk meestal door een compiler die een hogere programmeertaal omzet in machinecodes). Dergelijke programma's hebben dus machinecode, waarbij het niet uitmaakt op welk geheugenadres zo'n programma wordt geladen. Een kenmerk van deze relocatable code is dat alle sprongen en subroutineaanroepen relatief ten opzichte van de huidige waarde van de program counter zijn. De code is in uitvoering vaak iets trager dan de variant die wel plaatsgebonden is.

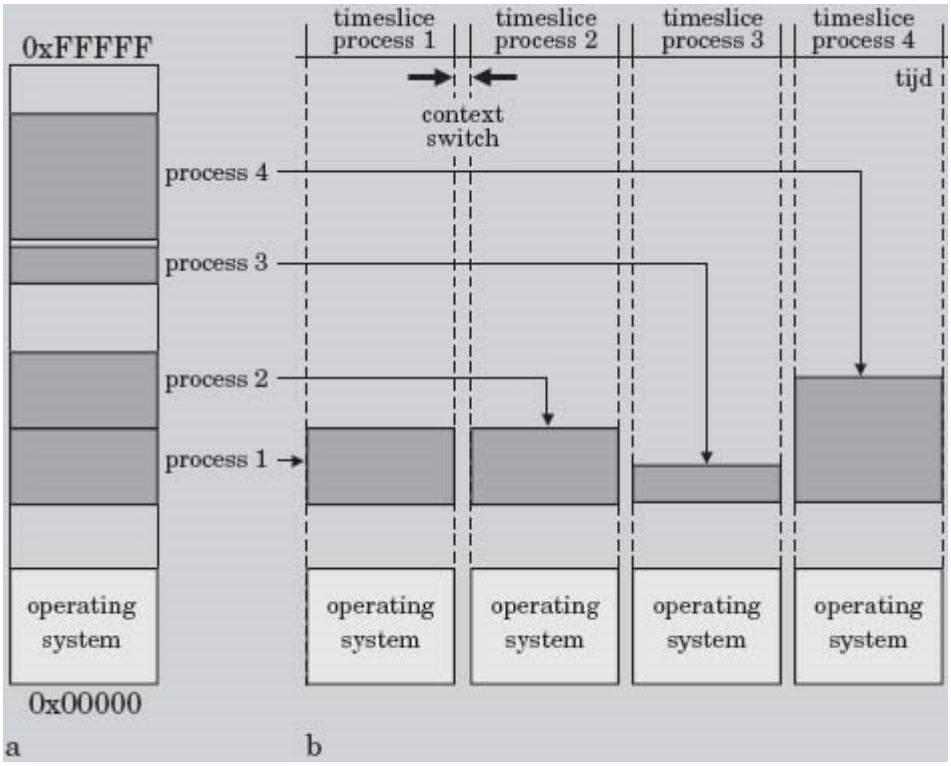
1.7.2 Memory management

De al eerder aangekondigde hardwareoplossing bestaat uit het aanbrengen van een memory management unit (MMU) tussen de adreslijnen van de CPU en de adreslijnen van het werkgeheugen (waar de objectcode van de processen zich bevindt). Zo'n MMU vertaalt de adressen van de CPU naar het adres waar het actieve proces zich bevindt. In het meest eenvoudige geval zal de MMU voor een bepaald proces een vast getal bij het CPU-adres optellen of aftrekken. Onderstaande figuur laat zien waar de MMU-hardware in een systeem gepositioneerd is. Veel CPU's hebben tegenwoordig de MMU-hardware aan boord.



De MMU kan zich zowel in de CPU zelf bevinden (zoals bij de Intel-CPU's voor PC-gebruik), maar ook als een afzonderlijke chip of als een speciaal ontworpen hardwareschakeling uitgevoerd zijn. De moderne CPU's hebben meestal een interne MMU. De programmering van de MMU gebeurt door de CPU en alleen binnen het operating system. Als een nieuw proces opgestart moet worden, zal de CPU een vrij stuk geheugen opzoeken (ook dit wordt bijgehouden in de 'boekhouding' van het operating system). De machinecode van het proces zal daar geladen worden. Als het proces uitgevoerd gaat worden, zal de CPU de MMU vertellen waar het proces zich echt bevindt en ook waar de CPU het eigenlijk zou willen hebben. De MMU zorgt dan voor de vertaling van de CPU-adressen voor dit proces. Zo zal er voor elk proces een nieuwe vertaling door de MMU zijn. Dit programmeren van de MMU gaat aanzienlijk sneller dan het verplaatsen van een heel proces.

We bekijken ondertaande CPU. In het geheugen bevindt zich een aantal processen samen met het operating system. We gaan ervan uit dat een proces één stuk geheugen beslaat. Door de vertaalslag van de MMU lijkt het alsof alle processen (als ze actief zijn) op hetzelfde adres beginnen. Op een bepaald tijdstip is er echter maar één proces actief zodat dit geen probleem geeft. Elk proces heeft zijn eigen vertaalslag van de MMU en de MMU doet maar één vertaalslag tegelijk. Stel proces 2 is actief. De vertaalslag van de MMU is nu zodanig dat proces 2 op de 'goede plaats' zit. De werkelijke positie van de andere processen doet er op dat moment niet toe omdat die toch niet actief zijn. Zie onderstaande figuur.



a Memory map
b Memory zoals gezien door de CPU via de MMU op verschillende tijdstippen

We zullen verderop zien dat processen niet als één aaneengesloten blok in het geheugen hoeven te staan. Ook kan een MMU de mogelijkheid geven het geheugengebied van een proces te beschermen, met als gevolg dat dit proces ook weer niet buiten dit gebied kan komen.

1.7.3 User en kernel mode

Om ervoor te zorgen dat gebruikers van een systeem niet zomaar zelf de MMU gaan programmeren of willekeurige blokken op de disk gaan overschrijven, kennen de meer geavanceerde moderne CPU's een zogenoemde user mode en kernel mode. Het operating system draait (vaak gedeeltelijk) in kernel mode, de (user) processen altijd in user mode. In user mode mogen bepaalde instructies niet uitgevoerd worden. Een user process kan dat alleen doen door gebruik te maken van de system calls die het operating system biedt. Zo'n system call wordt in de kernel uitgevoerd in kernel mode van de CPU, maar het user process heeft niet meer direct invloed op de uitvoering. Doordat het echte werk van de system call door de kernel gedaan wordt, kan ook door de kernel gekeken worden of het user process niet buiten z'n boekje gaat (mag dit user process deze file wel openen eenzovoort). Op deze manier is te voorkomen dat een gebruiker met zijn user process gegevens kan lezen van een andere gebruiker, of dat een kwaadwillende of onhandige gebruiker het hele operating system in de war stuurt.

Een CPU kan vanuit kernel mode zonder meer overschakelen naar user mode. Het omgekeerde is natuurlijk niet zomaar mogelijk (de hele protectie zou hierdoor op losse schroeven komen te staan). Een interrupt, waaronder ook de software interrupt, brengt de CPU echter weer in kernel mode, maar de machinecode waar de interrupt de CPU naartoe heeft gebracht, hoort ook weer bij het operating system. Zo bestaat er voor een proces in user mode geen mogelijkheid stiekem kernel mode in te schakelen, zonder tussenkomst van het operating system. In plaats van user mode wordt ook wel de term protected mode gebruikt. Kernel mode wordt ook als supervisory mode aangeduid.

1.7.4 Procesmanagement bij beperkt geheugen

Als in een multi-tasking operating system op een gegeven moment veel processen ontstaan (doordat bijvoorbeeld in een multi-user systeem steeds meer gebruikers inloggen), dan kan het werkgeheugen (RAM) van de computer wel eens vol raken. In het algemeen zijn er drie oplossingen voor dit probleem (afgezien van het kopen van meer geheugenchips):

1. Het operating system laat geen nieuwe processen toe. Pas als er genoeg processen klaar en verdwenen zijn, komt er weer ruimte voor nieuwe processen. Dit is een simpele maar onelegante oplossing.
2. Het operating system zoekt een proces dat al een hele tijd niets doet, omdat het zit te wachten op I/O (waiting) en schrijft de objectcode van dit proces tijdelijk naar de disk. De plaats die vrijkomt, wordt gebruikt door het nieuwe proces. We noemen dit swapping. Dit systeem is niet erg ingewikkeld; het vereist een simpele boekhouding in het operating system. Een nadeel is dat het de performance (prestatie) van de computer nogal aantast. Een systeem dat tot op zeker moment nog een goede respons voor de interactieve gebruikers oplevert, zal van het ene op het andere moment aanzienlijk trager worden.
3. De derde oplossing is gebruik te maken van een meer ingewikkelde MMU en een CPU die demand paging ondersteunt.

De laatste oplossing wordt in moderne operating systems vaak toegepast. We zullen deze methode dan ook nader toelichten. Wat er gebeurt, is het volgende. Processen worden opgedeeld in stukjes met een vaste grootte, zogenoemde pages. Een page kan even groot zijn als een diskblock (512 bytes), maar dat hoeft niet (4096 bytes is een gebruikelijke waarde). Als het werkgeheugen vol is, wordt niet een heel proces maar slechts een page naar de disk geschreven. In deze vrije page wordt de eerste page van het nieuwe proces geplaatst en dit wordt opgestart. Als de CPU uit deze page loopt, wordt dit door de MMU gesignaleerd en deze geeft een signaal (page-trap) naar de CPU. Door dit signaal komt de CPU automatisch in kernel mode en kan een nieuw blok van het proces ophalen en in het werkgeheugen plaatsen (eventueel weer ten koste van een page van een ander proces). Vervolgens wordt het proces weer doorgestart enzovoort.

Dit systeem heeft vier voordelen:

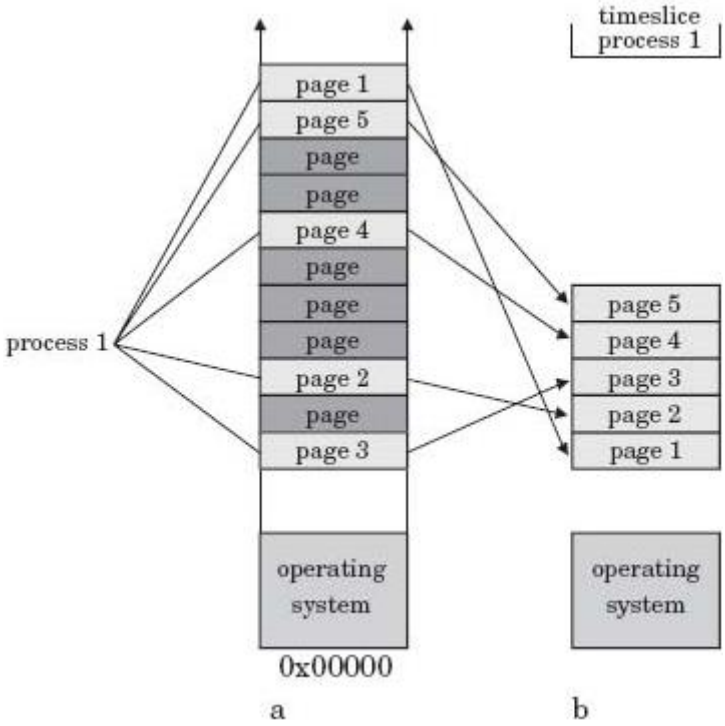
1. Op deze manier hoeven niet hele processen weggeschreven te worden zoals in een swappingsysteem, maar slechts pages. Dit is dus sneller.
2. Pages van een proces die alleen code bevatten die in uitzonderlijke situaties nodig is (error handlers), zullen meestal niet in het werkgeheugen geladen worden. Dit betekent dus weer snelheidswinst.
3. Het is mogelijk processen te verwerken die niet in hun geheel in het werkgeheugen passen.
4. Het probleem van het zoeken naar een passend aaneensluitend stuk geheugen voor een nieuw proces, zoals dat bij het swappingsysteem nodig was, is hier niet aanwezig.

Natuurlijk zijn er ook nadelen, namelijk:

1. De MMU van een pagingsysteem moet aan nogal wat eisen voldoen. Moderne CPU's met MMU hebben hier geen probleem mee.
2. Men kan deze techniek alleen toepassen als de CPU er hardwarematig op ingericht is. Ook hier geldt dat moderne CPU's meestal al op paging ingericht zijn.
3. De snelheid waarmee een proces wordt uitgevoerd, gaat van allerlei externe factoren afhangen (hoeveel andere processen zijn er, hoeveel geheugen is er beschikbaar). Hierdoor is deze snelheid onvoorspelbaar. Dit geldt overigens ook voor swapping.

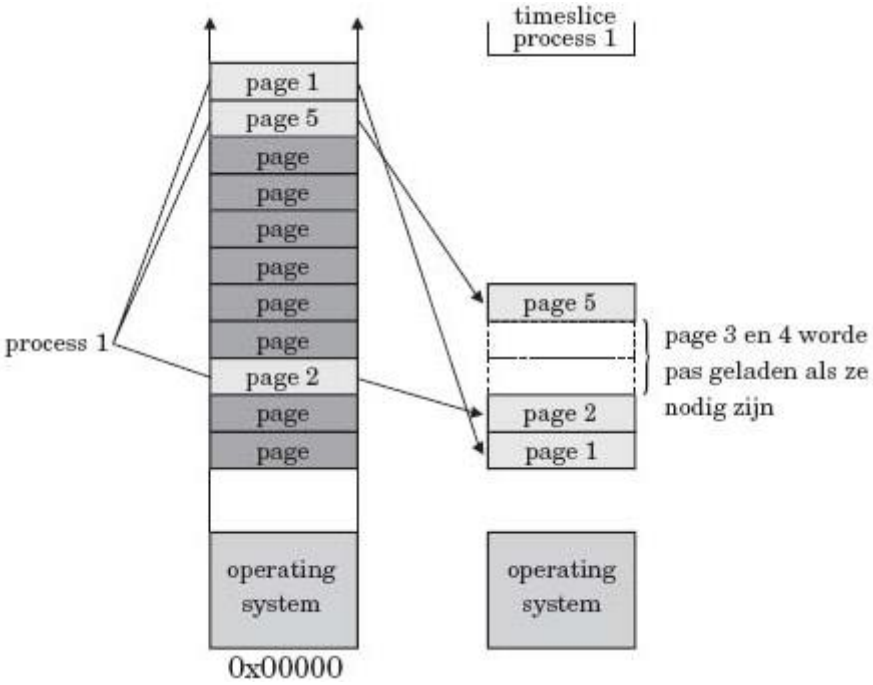
1.7.5 Virtueel geheugen

Een pagingsysteem wordt ook wel een virtueel geheugensysteem genoemd, omdat de computer schijnbaar een heel groot werkgeheugen heeft (groter dan er echt fysiek aanwezig is). Onderstaande figuur laat zien hoe we ons een pagingsysteem kunnen voorstellen.



a Memory map met pages
b Gezien door de CPU via de MMU

De pages van proces 1 staan kriskras door het geheugen. De MMU zet op het moment van executie (als proces 1 z'n timeslice krijgt) alle pages van proces 1 netjes op de goede plek. De MMU heeft nu dus voor elke page van een proces een eigen vertaalslag van de adressen van de CPU. In de 'administratie' van het operating system wordt nu per proces een tabel met informatie voor de MMU bijgehouden. Bij een nieuwe timeslice voor een ander proces volgt weer een nieuwe mapping door de MMU, zodat de pages voor dit nieuwe proces goed geadresseerd kunnen worden. Dat bij een zuiver pagingsysteem niet alle pages aanwezig hoeven te zijn, laat onderstaande figuur zien. Pages 3 en 4 ontbreken en worden pas geladen als ze nodig zijn.



Sommige operating systems hebben een gedeelte van de schijf ingericht voor swapping (van zowel hele processen als van pages). Zo'n gereserveerd stuk harddisk heet swap space of swap device. Andere operating systems kiezen voor een gewone file in het normaal toegankelijke filesysteem; we spreken dan van de swap file.

1.7.6 Shared code

MMU-hardware maakt naast het beheer van afzonderlijke processen ook andere dingen mogelijk. Bekijken we nog eens het geheugenmodel van een proces, dan blijkt dat het codegedeelte van zo'n proces (het deel dus waar de machine-instructies staan) altijd alleen gelezen wordt tijdens de uitvoer van een proces (geen self-modifying code). Gegeven dit feit is het mogelijk twee gelijke processen het codegedeelte gemeenschappelijk te laten hebben. We spreken dan van shared code. Dit verschijnsel van gelijke processen hoeft in een multi-user/multi-tasking-omgeving geen zeldzaamheid te zijn. Vrijwel elke gebruiker heeft één of soms zelfs meer command interpreters voor zich aan het werk.

1.7.7 Dynamic link libraries, shared libraries

Processen zijn programma's in uitvoering. Een programma is meestal geschreven in een hoge programmeertaal. Een compiler vertaalt deze programmacode naar machinecode. Vrijwel elke hoge programmeertaal levert voor veelvoorkomende zaken een bibliotheek aan voorgebakken oplossingen. Zo'n bibliotheek bestaat uit routines die door het programma gebruikt kunnen worden. Voorbeelden van zulke routines zijn de 'writeln' en 'printf' uit de talen Pascal en C. Het is mogelijk de machinecode van deze routines vast in te bouwen in de code van een programma. Dit inbouwen noemen we linken. Op het moment dat het programma een proces wordt, wordt de hele brok code in het geheugen geladen. We noemen deze aanpak static linking. Het is ook mogelijk dit linken uit te stellen en het pas te doen als een proces erom vraagt. We spreken van dynamic linking. Bibliotheken met routines die dynamisch gelinkt kunnen worden, heten dynamic link libraries.

Vrijwel dezelfde techniek wordt ook wel aangeduid met de naam shared library. Bibliotheekroutines kunnen dynamisch met een proces gelinkt worden. Deze routines kunnen door een heleboel processen gedeeld worden, vandaar de naam shared library.

1.8 Filemanagement

Operating systems geven de gebruiker op gecontroleerde wijze toegang tot bestanden. Bestanden kunnen gegevens, teksten, programma's en dergelijke bevatten. De bestanden bevinden zich meestal op een disk. De wijze waarop de disk georganiseerd is en hoe we bestanden kunnen bereiken, zijn de kenmerken van een filesysteem of file system.

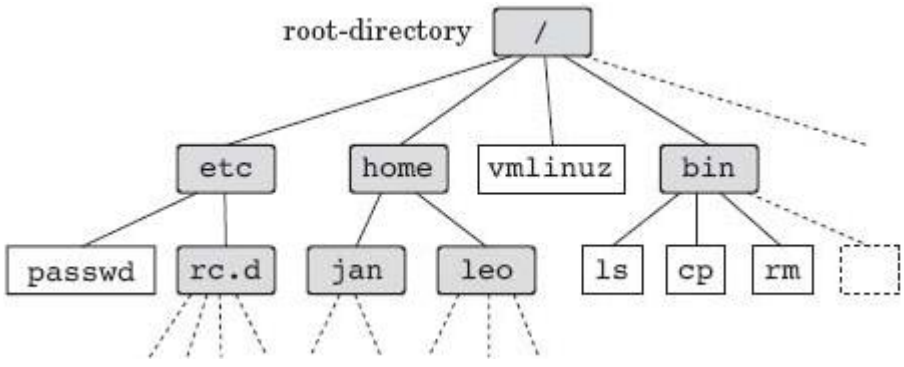
Eenvoudige bestandsmanipulaties zijn bewerkingen als het aanmaken van een bestand, het wijzigen, het opvragen van de inhoud en het verwijderen van een bestand. We besteden achtereenvolgens aandacht aan:

- directories;
- filenamen, extensies en permissies;
- filesystemen;
- meer filesystemen voor een operating system;
- disk met meer filesystemen;
- toegang tot 'vreemde' filesystemen;
- netwerkfilesystemen.

1.8.1 Directories

De meeste moderne operating systems organiseren de bestanden op een hiërarchische wijze. Een directory bevat bestanden en eventueel subdirectories die op hun beurt weer bestanden en nieuwe subdirectories kunnen bevatten. Op deze wijze ontstaat een zogenoemde directoryboom. De directory van het eerste niveau (dus waar alles begint) heet de root directory.

In onderstaande figuur is een gedeelte van een directoryboom weergegeven. Het betreft hier een Unix (Linux)-systeem. De files zijn als rechthoeken getekend; de directories hebben afgeronde hoeken en zijn lichtgrijs. De root directory (aangeduid met / in tegenstelling tot de bij MS-Windows gebruikelijke\) bevat een file 'vmlinuz' en enkele directories. Een directory binnen een directory noemt men ook wel een subdirectory. De directory 'home' die zelf een subdirectory is van de root directory / bevat weer twee subdirectories, namelijk 'jan' en 'leo'. Een subdirectory kan zowel files als nieuwe subsubdirectories bevatten. Zo bevat 'etc' een subdirectory 'rc.d' en een file 'passwd'.



1.8.2 Filenamen, extensies en permissies

Files worden aangeduid met een filenaam. Het aantal tekens en welke tekens er mogen worden toegepast, wordt voorgeschreven door het filesysteem. Zo maken sommige systemen geen onderscheid tussen hoofd- en kleine letters, terwijl andere systemen dat wel doen. Onder extensie verstaan we een toegevoegde (meestal korte) naam.

Bekend zijn de MS-DOS- en MS-Windows-extensies '.COM' en '.EXE' waarmee namen van uitvoerbare programma's worden uitgebreid (COMMAND.COM, FORMAT.COM enzovoort). Het VAX-VMS-systeem houdt van bestanden ook een versie bij. Het versienummer staat achter de filenaam en wordt gescheiden door een puntkomma (;-teken). Wordt een file aangepast, dan blijft de oude versie beschikbaar en wordt het versienummer van het nieuwe bestand met één verhoogd. Natuurlijk is er een commando om alle oude versies in één keer op te ruimen, ongeacht hoeveel het er zijn.

Aan bestanden zitten meestal (file) attributes gekoppeld. De meest in het oog springende zijn de permissies, dat wil zeggen: wie mag wat met het bestand doen? Voorbeeld uit het Unix-systeem zijn de 'rwx'-permissies. Hier staat r voor read, w voor write en x voor execute. Een Unix-file heeft in feite drie groepjes van deze permissies. De eerste groep geldt voor de eigenaar van de file, de tweede groep voor alle gebruikers die in dezelfde groep zijn ingedeeld als de eigenaar en de derde groep geldt voor alle overige gebruikers. Een voorbeeld zal dit duidelijk maken.

```
-rwxr- x--- a.out
```

Toelichting: de file a.out mag door de eigenaar gelezen, geschreven en als programma uitgevoerd (geëxecuteerd) worden. Het is dus een programma, zoals Unix-kenners al uit de naam konden afleiden. Gebruikers van hetzelfde systeem die door systeembeheer in dezelfde groep zijn ingedeeld, kunnen het bestand lezen en uitvoeren. Andere gebruikers kunnen er niets mee. De permissies staan uit (allemaal mintekens).

```
-rw-rw-rw- lees_mij_eerst
```

Deze file kan door iedere gebruiker gelezen en geschreven worden. Het is geen programma (waarschijnlijk een tekstbestand).

```
drwx------ lettertypen
```

Dit is een directory, hetgeen af te leiden is uit de letter 'd' helemaal vooraan. De eigenaar kan hier alles mee doen (de 'x' voor directories wil zeggen dat je erin mag).

Bij MS-DOS en het daarbij gebruikte FAT-filesysteem, dat bij embedded toepassingen ook toegepast wordt, spelen andere attributes een rol. Een file kan één of meer van de volgende attributes hebben:

- Hidden.
  Een hidden file is (zoals de naam al zegt) wel aanwezig maar niet op de gewone manier zichtbaar
- System.
  Deze attribute zegt dat het om een system file gaat.
- Archive.
  Een attribute dat bij het maken van back-ups van dienst kan zijn.
- Subdirectory.
  Een attribute dat aangeeft dat het een directory betreft.
- Read-only.
  Dit attribute maakt een file alleen leesbaar.

1.8.3 Filesystemen

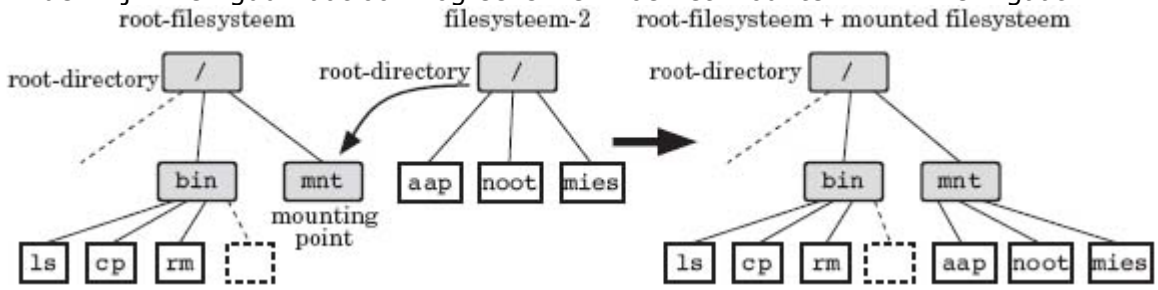
De wijze waarop de interne organisatie van files op de disk geregeld is, wordt het filesysteem genoemd. Het operating system MS-DOS en de eerste versies van MS-Windows maken gebruik van het FAT-filesysteem. Andere filesystemen zijn het VFAT-systeem (Windows 95, 98), NTFS (Windows NT, XP e.v.), i-node-systeem (Unix/Linux),

ISO 9660 voor de organisatie van files op een cd-rom en UDF (universal disk format) voor dvd.

1.8.4 Meer filesystemen voor een operating system

Wanneer er voor een operating system meer filesystemen beschikbaar zijn, zijn er ook meer manieren om die beschikbaar te maken voor de gebruiker. De filesystemen kunnen naast elkaar gerangschikt zijn en de gebruiker kan in een commando aangeven welk filesysteem hij bedoelt. De verschillende filesystemen zijn met een letter gevolgd door een dubbele punt aan te duiden (A:, B:, C: enzovoort). Deze benadering wordt door Windows en Windows NT gevolgd. We spreken van een multi-rooted file system.

Een andere manier die door Unix gevolgd wordt, is het opnemen van een filesysteem binnen een ander filesysteem. We spreken van een basis- of root-filesysteem; vervolgens worden hieraan op daarvoor aangemaakte (sub)directories andere filesystemen gekoppeld of 'gemount'. Onderstaande figuur geeft dit weer. Het root-filesysteem heeft op de subdirectory 'mnt' een nieuw filesysteem aangekoppeld gekregen. Het resultaat is een enkele boomstructuur waar alle bestanden van de diverse filesystemen in terug te vinden zijn. De figuur laat ook nog eens zien hoe het mounten in z'n werk gaat.



Het root-filesysteem heeft eerst een lege directory met de naam 'mnt'. Hieraan wordt het filesysteem-2 gekoppeld (gemount). Het resultaat is een grotere boom. Het mounten kan op een willekeurige (sub)directory plaatsvinden.

1.8.5 Toegang tot 'vreemde' filesystemen

Naast het eigen filesysteem bieden diverse operating systems de mogelijkheid andere filesystemen te gebruiken. Een en ander gaat wel ten koste van de efficiëntie of mogelijkheden, maar sommige systemen gaan hier heel ver in en bieden een vrijwel transparante toegang.

Zo kunnen de meeste operating systems bestanden op een FAT-USB-stick wel lezen en eventueel ook schrijven, ondanks het feit dat de organisatie van dit filesysteem volledig afwijkt van het door dat operating system gebruikte filesysteem.

1.8.6 Netwerkfilesystemen

Filesystemen kunnen ook via een computernetwerk beschikbaar zijn. De files hoeven dan dus niet lokaal op de harde schijf te staan, maar een andere computer stelt bestanden via een netwerk (datacommunicatie) als fileserver beschikbaar. Ook hier kan de toegang transparant zijn. Dan lijkt het net of de bestanden op de lokale disk aanwezig zijn. Voorbeelden hiervan zijn NFS (network file system) en CIFS (SMB).

1.9 Device drivers

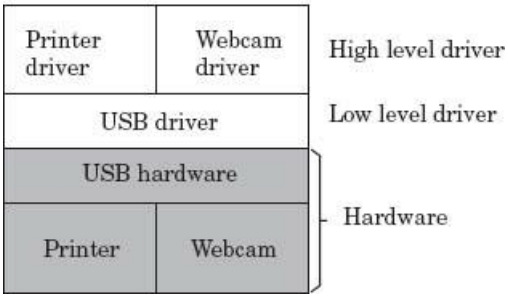
Device drivers zijn softwarecomponenten waarmee we de I/O aansturen. Meestal vormen device drivers een onderdeel van een operating system. De meeste moderne operating systems bieden de mogelijkheid device drivers aan het operating system toe te voegen.

Sommige systemen moeten dan opnieuw gestart worden, de wat beter doordachte systemen laten het toevoegen en eventueel verwijderen van device drivers zonder herstart (reboot) toe. Hardwarefabrikanten leveren via internet of cd-rom device drivers voor populaire operating systems met de hardware mee.

We gaan hierna eerst in op low- en high-level drivers. Daarna komen interrupts aan bod. Tot slot bespreken we de taken van device drivers.

1.9.1 Low- en high-level drivers

De eigenlijke aansturing van de hardware wordt ook wel de low-level device driver genoemd. Bij bepaalde I/O-poorten is het met deze device driver mogelijk om met de aangesloten hardware te communiceren. Dit is dan niet het hele verhaal, want de hardware kan wel eens heel divers zijn. Kijk bijvoorbeeld naar USB of SCSI, dan is de low-level of generieke driver nog niet genoeg om van alle mogelijk aansluitbare hardware alle eigenschappen te benutten. Een tweede stuk van een driver, die we high-level driver noemen, kan dan in een applicatie of als extra toevoeging aan het operating system zorg dragen voor het aansturen van de specifieke hardware. Onderstaande figuur laat zien hoe dat er voor USB uitziet, als we daar een webcam en een printer op aangesloten hebben.



De high-level driver maakt gebruik van de low-level driver. De low-level driver bevindt zich vrijwel altijd in kernel space. De high-level driver kan in user space zitten; dit is het deel waar ook de applicaties draaien.

1.9.2 Interrupts

Interrupts vormen het mechanisme om relatief trage I/O of I/O-events die op een willekeurig moment optreden met een systeem te combineren, waar de CPU zo efficiënt mogelijk wordt ingezet. Hiermee bedoelen we dat de CPU pas op de I/O reageert als dat ook werkelijk nodig is. We laten de CPU geen tijd verspillen met polling. Bij polling vraagt de CPU met enige regelmaat aan de I/O-devices of er nog wat te doen is.

1.9.3 Taken van device drivers

De software waar een low-level device driver uit bestaat, kent een viertal specifieke taken:

1. Initialisatie.
   De meeste I/O-hardware zal bij het inschakelen van de voedingsspanning (power-up) of na een reset in een inactieve toestand verkeren en wachten op een actie van de CPU, waarbij interne registers met de juiste configuratiecodes gevuld worden. Als dit proces, dat we initialisatie noemen, heeft plaatsgevonden, zal het device gaan functioneren.
2. Herconfiguratie.
   Na initialisatie is de I/O-hardware klaar voor gebruik. Het kan echter voorkomen

dat de oorspronkelijke initialisatie moet worden herzien. Neem bijvoorbeeld de initialisatie van een seriële poort, waarbij het aantal bits in een frame bij initialisatie is vastgelegd. Een applicatie zoals een modemaanstuurprogramma, kan een ander frameformaat eisen en dan moeten we de initialisatie aanpassen. We noemen dit herconfiguratie.

3. Aansturing.
   Dit is de hoofdtaak van de device driver. Via system calls maakt een applicatie gebruik van de I/O. Het operating system zal de data doorsturen naar het I/O-device of eigenlijk de device driver. Het operating system dient er zorg voor te dragen dat de dataoverdracht van of naar het device op de juiste wijze in zijn werk gaat. Als er bijvoorbeeld twee processen iets willen printen, moet de uitvoer niet door elkaar lopen. Het device moet voor een proces tijdelijk geblokkeerd zijn zolang het andere proces daarmee bezig is. Dit is een zaak van het operating system en niet van de device driver. De driver zorgt voor de dataoverdracht.
4. Interrupt-afhandeling.
   Een device driver levert ook het gedeelte van de interruptafhandelingsroutine voor het betreffende I/O-device.
   Hierbij gebeurt het volgende:
   - datatransfer afronden;
   - interrupt-signaal uitzetten;
   - juiste proces zoeken en wakker schudden.

1.10 Realtime scheduling

In het begin van dit hoofdstuk hebben we gezien hoe de CPU-tijd over verschillende taken wordt verdeeld. Dit gebeurde door de scheduler. We gaan nu kijken welke prominente rol deze scheduler speelt in wat we realtime systemen noemen.

We gaan in op scheduling in standaard operating systems. Daarna bespreken we de clock-driven en de priority-driven scheduling. Aan het einde van deze paragraaf kijken we naar de haalbaarheid. We beginnen met enkele algemene begrippen.

- Scheduling

Onder scheduling verstaan we het inroosteren of plannen van de taken die door een systeem moeten worden uitgevoerd. Kijken we naar een standaardcomputer, dan wordt het gedeelte van het operating system dat hiervoor verantwoordelijk is, de scheduler genoemd. De scheduler moet op een zeker moment beslissen welke taak uitgevoerd gaat worden of aan de beurt is. Hiervoor bestaan nogal wat mogelijkheden.

Bij de conventionele operating systems kent men tegenwoordig meestal een zogenoemde pre-emptive scheduler gebaseerd op timeslices. Bij dit type scheduler krijgt een taak een beperkte tijd de CPU toegewezen. Na deze tijd breekt een hardware interrupt afkomstig van een timer de taak af en de scheduler kan een nieuwe keuze maken. De afgebroken taak wordt in een later stadium weer opgepakt en zal eventueel in één keer afgerond worden. Welke taak voor een timeslice wordt gekozen, behoort ook tot de taak van de scheduler.

De meest eenvoudige vorm is de round robin scheduler. Alle taken staan in een rij en wanneer een taak aan de beurt is geweest sluit die weer achteraan.

Een tweede mogelijkheid maakt gebruik van prioriteiten. Taken met een hoge prioriteit gaan voor. Het is ook mogelijk om deze twee vormen van scheduling te combineren. Als twee taken dezelfde prioriteit hebben of zoals men dat zegt hetzelfde priority level hebben, dan kunnen ze binnen dat level weer op basis van round robin gekozen worden.

Bij realtime scheduling willen we voor één of meer taken een tijdsgarantie. Mochten we de vorm van scheduling met timeslices willen toepassen, dan ligt de keuze voor een op prioriteiten gebaseerde scheduler voor de hand. Realtime schedulers kennen vaak veel priority levels om te voorkomen dat meer dan één taak hetzelfde level gebruikt.

- Task en job

Onder een job verstaan we een klus die geklaard moet worden. Vaak is een job een onderdeel van een groter geheel. Dat groter geheel noemen we een task. Zo kan een job in een communicatiesysteem het verzenden van een datapakket zijn. De task kan dan het onderhouden van een communicatie tussen twee nodes zijn.

Onder een sporadic job verstaan we een job die naast de standaard jobs zo af en toe de kop opsteekt. Denk hierbij aan de reactie van een besturingssysteem op een ingreep van een operator. Onder een periodic job verstaat men een job die om de zoveel tijd steeds herhaald moet worden. Deze klasse van jobs komt nogal eens voor in realtime systemen.

- Release time, deadline, hard realtime en soft realtime

Jobs hebben een starttijd of release time. Daarmee bedoelen we dat op een zeker moment aan een job begonnen wordt. De release time zal niet voor alle jobs hetzelfde zijn. Vaak kan een job pas beginnen als andere jobs hun werk al gedaan hebben. In een realtime systeem zijn er jobs met een deadline. Dit is een tijdstip waarop de job zeker klaar moet zijn. Hoe strikt de deadline is, is bepalend voor het karakter van het realtime systeem. Systemen met een deadline die beslist gehaald moet worden, noemt men hard realtime systemen. Bij soft realtime mag een deadline binnen een zekere marge overschreden worden. Een volgende keer kan dat wellicht gecompenseerd worden.

Hard realtime treft men aan bij besturingen van bijvoorbeeld industriële processen of medische apparatuur. Ook een automatische piloot kent hard realtime aspecten. Een voorbeeld van soft realtime is een communicatiesysteem dat een Quality of Service (QoS) biedt. Dat wil zeggen: er wordt een garantie gegeven dat gegevens met een zekere snelheid verstuurd kunnen worden. Aangezien we hier over een tijdsgemiddelde spreken, mag het nu en dan gebeuren dat een deadline overschreden wordt.

- Static versus dynamic scheduling

Bij static scheduling ligt de scheduling volledig vast. Dit is natuurlijk niet echt flexibel, maar het kan in bepaalde situaties gewoon praktisch zijn om de jobs in een vaste volgorde uit te voeren. De scheduler is dan ook relatief eenvoudig van opzet. Bij dynamic scheduling wordt de scheduling van jobs bepaald door de scheduler. Hierbij spelen prioriteiten en deadlines een rol bij de keuze van de job die de resource krijgt. Een systeem dat op deze wijze werkt is meer flexibel en zal in meer uiteenlopende situaties ingezet kunnen worden. Ook als er sprake is van de eerdergenoemde sporadic jobs, is dynamic scheduling handig.

- Online versus offline

Bij static scheduling is het in complexe systemen mogelijk om (eventueel met zeer rekenintensieve optimalisatiealgoritmen) de scheduling van tevoren, dus vóór het systeem gestart is, uit te rekenen. Hierbij kan men vaak een systeem krijgen waarbij de resources optimaal benut zijn. Voorwaarde is wel dat de jobs met release time en deadline bekend zijn. Een voorbeeld van een pre-computed scheduling is het invoeren van bewegingspatronen in een industriële robot. Hiermee wordt dus de werking van tevoren vastgelegd.

- Pre-emption, non-pre-emption, slack

Zoals al opgemerkt kennen we schedulers die de resource toekennen aan een job en wachten tot de job zelf de resource weer vrijgeeft (non-pre-emption), en schedulers die een job na een zekere tijd afbreken en aan een nieuwe job de resource toekennen (preemption). De reden van het afbreken kan zijn het beschikbaar komen van een job met hogere prioriteit, of het aflopen van de toegekende timeslice. Non-pre-emption is qua opzet eenvoudiger, maar in de meeste gevallen minder gewenst. Als een job afgebroken wordt, noemt men de tijd die een job nog moet krijgen om afgerond te worden, de slack. Een job met een kleine slack zal dus weinig tijd van de resource meer nodig hebben om afgerond te worden.

- Haalbaarheid, feasibility

Bij een realtime scheduling is het natuurlijk van belang of de deadlines onder alle omstandigheden gehaald kunnen worden. We noemen dit de feasibility van het scheduling system. Veel onderzoek op het gebied van realtime spitst zich toe op het begrijpen van deze feasibility.

1.10.1 Scheduling in standaard operating systems

In standaard operating systems (zoals Windows-varianten en Unix/Linux) wordt een dynamic scheduler gebruikt die op basis van pre-emption werkt. De jobs zijn hier (threads of) processen en worden veelal ingedeeld op basis van priority-klassen. Binnen zo'n priority-klasse wordt de scheduling op round robin basis gedaan. Bij deze systemen krijgen de jobs timeslices toegekend. Een aantal mechanismen binnen zo'n operating system doet afbreuk aan de realtime aspecten. We zullen ze eens op een rijtje zetten:

- virtual memory;
- blockbuffercache;
- kernel tasks gaan meestal voor;
- interrupts.

Het grappige is dat deze eigenschappen eigenlijk de performance van het systeem als geheel verbeteren. De keerzijde is dat de voorspelbaarheid van de snelheid waarmee een bepaalde job afgerond wordt, soms in een worstcasescenario terecht kan komen (de job is niet in memory maar uitgeswapt, de datablokken staan niet in de buffercache en deze moet eerst geflushed worden waarbij alle blokken toevallig eerst weggeschreven moeten worden).

1.10.2 Clock-driven scheduling en priority-driven scheduling

Bij clock-driven scheduling staat een systeemclock centraal. Op vaste tijdstippen wordt de job gekozen die volgens een van tevoren bepaalde scheduling aan de beurt is. We hebben hier dus te maken met een offline scheduling. De overhead van de scheduler is hier minimaal.

Bij priority-driven scheduling bepaalt de priority van een job of hij aan de beurt komt. De priority wordt meestal door de scheduler berekend op basis van een aantal regels. Deze vorm van scheduling kan al dan niet pre-emption toepassen. Het gebruik van pre-emption is echter meestal gunstig.

1.10.3 Haalbaarheidsanalyse

Voor veel (vaak vereenvoudigde) scheduling systems zijn technieken beschikbaar die de haalbaarheid van een scheduling kunnen vaststellen. Wat nu te doen als een schedule niet haalbaar blijkt? We kunnen dan één of meer van de volgende stappen nemen:

- Kies een snellere CPU; de executietijden worden korter en de haalbaarheid van de deadlines komt wellicht binnen bereik.
- Bekijk de code van de jobs en optimaliseer naar snelle uitvoerbaarheid. Met andere woorden: kijk waar de bottlenecks zitten en probeer hier een oplossing voor te vinden.
- Bekijk de deadlines en kijk of er geen mogelijkheid tot verschuiving bestaat. Het kan zijn dat de deadlines in eerste instantie te scherp gesteld zijn.
- Kies voor een multiprocessorsysteem of meer parallel werkende systemen.

1.11 Conclusie

De multi-tasking operating systems zijn mogelijk en wenselijk dankzij de toegenomen rekenkracht van moderne CPU's. De gebruikersinterface is vaak grafisch georiënteerd. De taal C speelt voor systeemontwikkeling (operating systems) en technische programmatuur (device drivers) een belangrijke rol. Daarnaast is er ook bij operating systems een trend om van object-oriented technieken gebruik te maken. Moderne operating systems zijn hardwareonafhankelijk ontworpen en naar verschillende platforms over te zetten. Operating systems maken veel gebruik van de mogelijkheden die netwerkaansluitingen bieden.

Stabiliteit en veiligheid zijn belangrijke eigenschappen van een operating system. Een systeemreset of -herstart is een zwaktebod. Realtime systemen danken hun realtime eigenschappen voornamelijk aan de toegepaste scheduler.