stichting
mathematisch
centrum

MC

AFDELING INFORMATICA
(DEPARTMENT OF COMPUTER SCIENCE)

IN 20/82 OKTOBER

J. HEERING

STANDAARD OPERATING SYSTEMS ALS OPLOSSING EN ALS PROBLEEM

kruislaan 413 1098 SJ amsterdam

Standaard operating systems als oplossing en als probleem†

door

Jan Heering

SAMENVATTING

De laatste jaren is duidelijk geworden, dat enkele operating systems zich aan het ontwikkelen zijn tot *de facto* standaards. Deze ontwikkeling komt geen dag te vroeg, want het machine-onafhankelijke operating system kan de hoognodige stabiele grondslag leveren voor software-ontwikkeling op lange termijn en wel op een moment, dat een steeds grotere verscheidenheid aan (micro)processors daarvoor steeds minder vaste grond biedt. Tevens begint het echter duidelijk te worden, dat de gebruiker (programmeur) gebaat is met een consistente, in hoge mate met de door hem/haar gebruikte programmeertaal geïntegreerde, ontwikkelomgeving. De standaardsystemen, zoals ze op dit moment naar voren komen, bieden zo'n omgeving echter nog niet.

†Voordracht gehouden op 30 september 1982 in het kader van de ASI-leergang 'De Software Crisis'.

1. WAT IS EEN OPERATING SYSTEM?

De chaos op het gebied van operating systems (maar helaas niet alleen daar), waarmee de computerkundige bij zijn dagelijks werk wordt geconfronteerd en waarvoor hij in meerdere of mindere mate mede verantwoordelijk is, is zó groot, dat velen vluchten in een toestand van bewustzijnsvernauwing. Zij bekennen zich tot het geloof, dat het door hen gebruikte BOSS/VS8 operating system het enig goede is, en als een aanhanger van VIPS/TS het waagt dit te betwijfelen ontspint zich een heftige discussie, die echter wegens het ontbreken van een gemeenschappelijke terminologie tot mislukken gedoemd is. Weliswaar heeft dit het voordeel, dat de superioriteit van het eigen systeem onbeperkt is vol te houden, maar tot vermeerdering van inzicht leidt het niet.

Misschien is het daarom beter BOSS/VS8 en VIPS/TS voorlopig de rug toe te keren en te trachten een dieper inzicht in de oorzaken van de problematiek te krijgen. De in de aanhef gestelde vraag

Wat is een operating system?

is dan de eerste, die zich ter beantwoording aandient. Ik hoop, dat de lezer het niet als een vorm van pedanterie beschouwt, dat deze elementaire vraag hier zo nadrukkelijk gesteld wordt. Het zou niet nodig zijn geweest hem te stellen, als het antwoord evident geweest was.

In de literatuur wordt uiteengezet, dat een computer sec niet geschikt is om programma's op te ontwikkelen of om door meerdere gebruikers tegelijkertijd gebruikt te worden. Het is de taak van een operating system een omgeving te leveren waarin dit wel kan. Vervolgens worden als belangrijkste door een operating system te verrichten concrete taken file- en procesbeheer genoemd. De functies, die het systeem kan verrichten, zijn op twee manieren te activeren. Ten eerste via de zg. commandotaal of job control language (JCL), die het karakter van de gebruikersinterface bepaalt, en ten tweede via een verzameling systeemprocedures, die het karakter van de programma-interface bepaalt.

Sommige auteurs wekken, gewild of ongewild, de indruk dat de commandotaal slechts een bijkomstig aspect van een operating system zou zijn. Zoals een kaas een korst moet hebben, zo moet een operating system een commandotaal hebben. De korst heeft wel een functie, maar het gaat uiteindelijk om de kaas. Dit standpunt is onhoudbaar, omdat het karakter van de commandotaal grotendeels bepalend is voor de wijze waarop de gebruiker het systeem 'ziet'.

Alvorens van het begrip 'operating system' een omschrijving te geven, waarin de commandotaal voorop staat, is het nuttig eerst eens te onderzoeken hoe het met commandotalen op dit moment gesteld is. Dan valt op, dat de verst ontwikkelde vertegenwoordigers van de soort zo langzamerhand beginnen te lijken op rudimentaire hoog-niveau programmeertalen, met procedures, conditionele statements en operaties op files en processen in plaats van op getallen en records. Deze talen hebben een uiterst grillig karakter en dragen de sporen van een moeizame evolutie. Het lijkt wel alsof het nooit de bedoeling geweest is, dat ze op programmeertalen zouden gaan lijken, en zeker is het zo, dat de ontwikkeling van commando- en programmeertalen langs geheel gescheiden lijnen verlopen is. Maar dat commandotalen inderdaad gebruikt kunnen worden om in te programmeren moge blijken uit het feit, dat op vele installaties de helft van alle procedures in de commandotaal geschreven wordt.

Het feit, dat commandotalen zich in alle stilte hebben ontwikkeld tot geduchte concurrenten van programmeertalen, is des te ernstiger, omdat ook de verst ontwikkelde commandotalen geen behoorlijke semantiek bezitten. Bovendien zijn in commandotalen geschreven procedures volstrekt niet overdraagbaar. Hoe heeft deze schrikbarende ontwikkeling kunnen plaatsvinden? Er zijn verschillende oorzaken aan te geven, maar de belangrijkste lijkt wel te zijn, dat de meeste programmeertalen *onvolledig* zijn in die zin, dat ze een krachtige omgeving (lees: een krachtig operating system) vooronderstellen. Zelf bezitten zij onvoldoende uitdrukkingsmacht om het beheer van permanente gegevens (files) of het creëren van nieuwe programma's te beschrijven. Zo zijn bijvoorbeeld functies, die

een file opruimen of een nieuw file directory aanmaken, in de meeste programmeertalen eenvoudig niet gedefinieerd. Als ze wel beschikbaar zijn, dan alleen via de aanroep van een systeemprocedure, waarvan de body niet in de programmeertaal zelf geschreven kan worden en die dus systeemafhankelijk is. Commandotalen hebben de vereiste uitdrukkingsmacht wel en zijn in die zin *volledig*.

De ontwerpers van programmeertalen hebben, door hun geesteskinderen afhankelijk te maken van een krachtige, maar onvolledig gespecificeerde omgeving, zelf de ecologische niche gecreëerd, waarin de commandotalen tot ontwikkeling konden komen. Het onthutsende karakter van de zo ontstane nieuwe soort is daarmee echter niet verklaard. Het is misschien in dit verband niet overbodig op te merken, dat volledigheid niet hoeft te leiden tot onoverzichtelijkheid of tot een overmaat aan 'features'. Dit wordt geïllustreerd door LISP [1], één van de weinige programmeertalen die volledig is. Het is geen toeval, dat er in vele LISP-systemen geen aparte commandotaal is. In dergelijke systemen vallen operating system en taalimplementatie samen.

Terugkerend naar de oorspronkelijke vraag, stel ik voor een operating system primair te beschouwen als *de implementatie van een commandotaal*. Er is geen scherp onderscheid tussen commandotalen en programmeertalen, maar commandotalen zijn in elk geval *volledig* in de boven bedoelde betekenis. De implementatie heeft verder de eigenschap, dat andere talen (bijvoorbeeld programmeertalen) er in *ingebed* kunnen worden. De implementaties van dergelijke ingebedde talen kunnen onderdelen van de implementatie van de commandotaal in de vorm van systeemprocedures 'lenen'.

Deze omschrijving beoogt de emancipatie van de commandotaal en is meer op de toekomst gericht dan op het verleden. Hij moedigt er toe aan de overeenkomsten tussen commandotalen en programmeertalen nauwkeuriger dan tot nu toe gebeurd is te onderzoeken. Dat kan de consistentie van commandotalen en operating systems alleen maar ten goede komen en wellicht leert het ons ook nog iets over programmeertalen. Daarop kom ik in het vervolg nog terug.

2. HET MACHINE-ONAFHANKELIJKE OPERATING SYSTEM

Een tweede belangrijke vraag, die zich aandient, is:

Waarom zijn operating systems niet machine-onafhankelijk?

Met andere woorden, waarom zijn operating systems niet overdraagbaar naar machines met een andere architectuur dan die, waarvoor ze oorspronkelijk zijn ontworpen?

Gezien het feit, dat het in het begin van de zeventiger jaren door Bell Laboratories ontwikkelde UNIX operating system [2] wel machine-onafhankelijk is, is de vraag eigenlijk, waarom alle andere systemen het dan niet zijn. Ik heb op deze vraag geen bevredigend antwoord weten te bedenken. De introductie van overdraagbare systemen zou de eerder gesignaleerde chaos stellig verminderen. Dat is, zeker op langere termijn, in ieders belang. Ook lijkt er geen inherent probleem te zijn. De door commandotalen verschafte functies voor file- en procesbeheer zijn in het algemeen van een zodanig hoog niveau, dat daarin weinig machine-afhankelijks is terug te vinden. Die gedeelten van het systeem, die zich op laag niveau bezighouden met geheugenbeheer en met interrupts en input/output zijn sterk machine-afhankelijk. Maar dàt is alles bij elkaar betrekkelijk weinig. Door die gedeelten goed van het machine-onafhankelijke gedeelte te scheiden en het gehele systeem in een hogere taal te programmeren, moet overdraagbaarheid toch binnen bereik komen.

Dat dit laatste inderdaad zo is, is door UNIX bewezen. Was het systeem tot 1977 voornamelijk beschikbaar op de diverse modellen van de PDP11-serie minicomputers van Digital Equipment, thans is het in gebruik op machines van Amdahl, Honeywell, IBM, Interdata, Perkin-Elmer, Philips, en Siemens. Met de opkomst van de 16-bit microprocessors (Intel 8086, Zilog Z8000 en Motorola MC68000 - de laatste in kracht vergelijkbaar met de grootste huidige minicomputers) zijn de merites van het systeem eerst recht onderkend. UNIX is op alle drie types beschikbaar. Ook de sprong naar

de binnenkort te verwachten (en op kleine schaal reeds beschikbare) 32-bit microprocessors zal geen probleem vormen. Er is al geruime tijd een versie voor de 32-bits VAX-11 van Digital Equipment, die daarvoor als uitgangspunt kan dienen.

UNIX is ook geschikt om als basis te dienen voor een gedistribueerd operating system voor lokale netwerken. De machine-onafhankelijkheid van het systeem is daarbij een belangrijke factor. Het is immers niet noodzakelijk, dat alle computers in een dergelijk netwerk van hetzelfde type zijn, te meer niet omdat netwerken kunnen groeien en omdat bestaande netwerken nooit in hun geheel vervangen worden. Zeker met de huidige hausse in microprocessors zullen netwerken de neiging hebben heterogener te worden naarmate ze langer bestaan. Omdat dus de hardware geen stabiele basis biedt, is het noodzakelijk een constante factor op een hoger niveau te introduceren. De meest voor de hand liggende manier om dat te doen is om het gebruikersniveau van individuele computers in het netwerk te baseren op een niet merkgebonden, machine-onafhankelijk, operating system en UNIX is daarvoor op dit moment de aangewezen kandidaat. Er wordt dan ook op verscheidene plaatsen aan dergelijke, op UNIX gebaseerde, gedistribueerde systemen gewerkt. Zie bijvoorbeeld [3].

UNIX is niet alleen een machine-onafhankelijk, maar ook een krachtig systeem, dat de gebruiker vele faciliteiten biedt. Deze twee factoren hebben er samen toe geleid, dat het systeem zich tot *de facto* standaard heeft ontwikkeld in de academische wereld (daarbij inbegrepen de universitaire informatica-afdelingen). Als gevolg daarvan komt een steeds toenemend aantal softwarepakketten en programmeertalen onder UNIX beschikbaar. Dit leidt weer tot toenemende acceptatie van het systeem, ook buiten de academische wereld. Alles wijst erop, dat UNIX in een dergelijke fase van zichzelf versterkende populariteitsgroei verkeert.

Ik zal hier verder niet ingaan op de specifieke eigenschappen van UNIX. Wel wil ik in het volgende hoofdstukje nog enkele opmerkingen maken over de commandotaal van het systeem. Deze zullen aansluiten bij mijn eerdere beschouwingen over commandotalen.

3. COMMANDOTALEN *VS.* PROGRAMMEERTALEN - DE STRIJD VAN NABIJ

Wie de eerder beschreven strijd tussen commandotalen en programmeertalen van nabij wil observeren, kan bij UNIX terecht. De commandotaal van UNIX, de zg. *shell* [2], is op dit moment één van de krachtigste in zijn soort, maar, alhoewel reeds duidelijk door programmeertalen beïnvloed, nog steeds een bizar mengsel van macro's en concurrente processen zonder goed gedefinieerde semantiek. Omdat de shell echter in een aantal opzichten krachtiger is dan de meeste programmeertalen, gebruikt iedereen hem toch om in te programmeren. Dat hij geïnterpreteerd wordt (zoals vrijwel alle commandotalen) en dus traag is en geen statische controle kent, blijkt evenmin een bezwaar.

Het is ontegenzeggelijk een vooruitgang, dat een machine-onafhankelijk systeem als UNIX algemeen geaccepteerd raakt. Niettemin geeft de gebruikers-interface, die UNIX in de vorm van de shell met zich meevoert, reden tot zorg. Heeft de grote verspreiding die het systeem krijgt als consequentie, dat de shell een soort standaard commandotaal wordt? Gezien de wijd verbreide kritiekloze bewondering voor het systeem is dat een gevaar, dat helaas maar al te serieus genomen moet worden. Dat is de keerzijde van de medaille.

De conclusie, die, naar ik meen, uit de concurrentie tussen commando- en programmeertalen getrokken moet worden, is, dat een veel grotere integratie tussen beide soorten talen noodzakelijk is. Meer in het algemeen dient de werkomgeving van de programmeur veel homogener en daardoor conceptueel eenvoudiger te worden. Tot die werkomgeving behoren niet alleen de commandotaal en de op dat moment gebruikte programmeertaal, maar ook de taal van de symbolische tracer en de taal van de editor.

Waar in de huidige systemen vele concepten en constructies, die qua abstracte semantiek nauw verwant of zelfs identiek zijn, in een zodanig verschillende vorm worden aangeboden, dat het de gebruiker vrijwel onmogelijk wordt gemaakt de overeenkomsten te zien, zal een homogene omgeving deze overeenkomsten juist benadrukken. Zijn bijvoorbeeld de verschillen tussen variabelen en files echt zo diepgaand, dat er in programma's allerlei ingewikkelde invoer/uitvoerconstructies en -conversies nodig zijn om ze te overbruggen? En zijn de verschillen tussen programma's en procedures werkelijk zo groot als de meeste systemen de gebruiker willen doen geloven? Het zijn slechts enkele voorbeelden van punten, waar grote vereenvoudiging is te bereiken. Er zouden vele andere te geven zijn [4,5,6].

Het streven naar grotere homogeniteit is niet nieuw - het voorbeeld van LISP toont het aan - , maar het heeft de laatste jaren een groter moment gekregen. Het bij Xerox ontwikkelde SMALLTALK systeem [7] is te beschouwen als voorloper van wat op dit gebied komen gaat. Het zou een illusie zijn te denken, dat programmeren in dergelijke homogene omgevingen plotseling veel eenvoudiger wordt. Maar als het aangenamer wordt en als enkele van de vele obstakels, die zich in het verleden hebben opgehoopt, uit de weg worden geruimd is al veel bereikt.

—0—

REFERENTIES

[1] McCarthy, J., "LISP - Notes on its past and future", *Proceedings of the 1980 LISP Conference*, The LISP Co., PO Box 487, Redwood Estates, CA 95044, 1981.

[2] Nummer geheel gewijd aan het UNIX operating system, *The Bell System Technical Journal*, **57**(1978), 6, part 2.

[3] Rowe, L.A., & Birman, K.P., "A local network based on the UNIX operating system", *IEEE Transactions on Software Engineering*, **SE-8**(1982), 2, pp. 137-146.

[4] Jones, A.K., "The narrowing gap between language systems and operating systems", *IFIP 77*, 1977, pp. 869-873.

[5] Boute, R.T., "Building a uniform programming environment based on data abstraction", *ACM International Computing Symposium*, 1981.

[6] Heering, J., & Klint, P., "Towards monolingual programming environments", Mathematisch Centrum, Rapport IW 185/81, 1981.

[7] Nummer geheel gewijd aan het SMALLTALK systeem, *BYTE*, augustus 1980.